Salt Documentation

Release 2014.1.4

SaltStack, Inc.

July 17, 2014

1 Frequently Asked Questions 1
1.1 Is Salt open-core? . . . 1
1.2 What ports should I open on my firewall? . . . 1
1.3 I'm seeing weird behavior (including but not limited to packages not installing their users properly) . 2
1.4 My script runs every time I run a *state.highstate*. Why? . . . 2
1.5 When I run *test.ping*, why don't the Minions that aren't responding return anything? Returning `False` would be helpful. . . . 2
1.6 How does Salt determine the Minion's id? . . . 2
1.7 I'm trying to manage packages/services but I get an error saying that the state is not available. Why? 2
1.8 I'm using gitfs and my custom modules/states/etc are not syncing. Why? . . . 3
1.9 Why aren't my custom modules/states/etc. available on my Minions? . . . 3
1.10 Module `X` isn't available, even though the shell command it uses is installed. Why? . . . 3
1.11 Can I run different versions of Salt on my Master and Minion? . . . 3
1.12 Does Salt support backing up managed files? . . . 3

2 Introduction to Salt 5
2.1 The 30 second summary . . . 5
2.2 Simplicity . . . 5
2.3 Parallel execution . . . 5
2.4 Building on proven technology . . . 6
2.5 Python client interface . . . 6
2.6 Fast, flexible, scalable . . . 6
2.7 Open . . . 6

3 Installation 7
3.1 Quick Install . . . 7
3.2 Platform-specific installation instructions . . . 7
3.3 Dependencies . . . 23
3.4 Optional Dependencies . . . 23
3.5 Upgrading Salt . . . 23

4 Configuring Salt 25
4.1 Master Configuration . . . 25
4.2 Minion Configuration . . . 25
4.3 Running Salt . . . 26
4.4 Key Management . . . 26
4.5 Sending Commands . . . 27
4.6 What's Next? . . . 27

5 Developing Salt 29
5.1 Sending a GitHub pull request . . . 29
5.2 Keeping Salt Forks in Sync . . . 30
5.3 Posting patches to the mailing list . . . 30
5.4 Installing Salt for development . . . 30
5.5 Installing Salt from the Python Package Index . . . 33
5.6 Editing and previewing the documentation . . . 33
5.7 Running unit and integration tests . . . 34

6 Targeting 35
6.1 Matching the `minion id` . . . 35
6.2 Grains . . . 36
6.3 Node groups . . . 39
6.4 Compound matchers . . . 40
6.5 Batch Size . . . 40

7 Salt tutorials 43
7.1 Introduction . . . 43
7.2 Basics . . . 44
7.3 States . . . 55
7.4 Advanced Topics . . . 82
7.5 Salt Cloud . . . 109
7.6 Halite . . . 113

8 Access Control System 117

9 External Authentication System 119
9.1 Tokens . . . 119

10 Pillar of Salt 121
10.1 Declaring the Master Pillar . . . 121
10.2 Pillar namespace flattened . . . 122
10.3 Including Other Pillars . . . 122
10.4 Viewing Minion Pillar . . . 123
10.5 Pillar “get” Function . . . 123
10.6 Refreshing Pillar Data . . . 124
10.7 Targeting with Pillar . . . 124
10.8 Master Config In Pillar . . . 124

11 Master Tops System 125

12 Job Management 127
12.1 The Minion proc System . . . 127
12.2 Functions in the saltutil Module . . . 127
12.3 The jobs Runner . . . 127

13 Salt Scheduling 129
13.1 States . . . 129
13.2 Highstates . . . 129
13.3 Runners . . . 130
13.4 Scheduler With Returner . . . 130

14 Running the Salt Master/Minion as an Unprivileged User 131

15 Troubleshooting 133
15.1 Troubleshooting the Salt Master . . . 133

15.2 Troubleshooting the Salt Minion . . . 133
15.3 Running in the Foreground . . . 133
15.4 What Ports do the Master and Minion Need Open? . . . 133
15.5 Using salt-call . . . 134
15.6 Too many open files . . . 134
15.7 Salt Master Stops Responding . . . 134
15.8 Salt and SELinux . . . 135
15.9 Red Hat Enterprise Linux 5 . . . 135
15.10 Common YAML Gotchas . . . 135
15.11 Live Python Debug Output . . . 136
15.12 Salt 0.16.x minions cannot communicate with a 0.17.x master . . . 136

16 Troubleshooting the Salt Master **137**
16.1 Running in the Foreground . . . 137
16.2 What Ports does the Master Need Open? . . . 137
16.3 Too many open files . . . 137
16.4 Salt Master Stops Responding . . . 138
16.5 Live Python Debug Output . . . 139
16.6 Commands Time Out or Do Not Return Output . . . 139
16.7 Passing the -c Option to Salt Returns a Permissions Error . . . 139

17 Troubleshooting the Salt Minion **141**
17.1 Running in the Foreground . . . 141
17.2 What Ports does the Minion Need Open? . . . 141
17.3 Using salt-call . . . 142
17.4 Live Python Debug Output . . . 142

18 YAML Idiosyncrasies **143**
18.1 Spaces vs Tabs . . . 143
18.2 Indentation . . . 143
18.3 True/False, Yes/No, On/Off . . . 144
18.4 Integers are Parsed as Integers . . . 144
18.5 YAML does not like “Double Short Decs” . . . 145
18.6 YAML support only plain ASCII . . . 146
18.7 Underscores stripped in Integer Definitions . . . 146
18.8 Automatic `datetime` conversion . . . 146

19 Community **149**
19.1 Mailing List . . . 149
19.2 IRC . . . 149
19.3 Follow on Github . . . 149
19.4 Blogs . . . 149
19.5 Example Salt States . . . 150
19.6 Follow on ohloh . . . 150
19.7 Other community links . . . 150
19.8 Hack the Source . . . 150

20 Salt Based Projects **151**
20.1 Salt Sandbox . . . 151

21 Salt Event System **153**
21.1 Listening for Events . . . 153
21.2 Firing Events . . . 154
21.3 Firing Events From Code . . . 154

22 The Salt Mine **155**
22.1 Mine Functions 155
22.2 Mine Interval 155

23 Salt Cloud Documentation **157**
23.1 Cloud Actions 157
23.2 Getting Started With AWS EC2 157
23.3 Getting Started With Azure 167
23.4 Writing Cloud Provider Modules 170
23.5 Core Configuration 176
23.6 OS Support for Cloud VMs 186
23.7 Getting Started With Digital Ocean 188
23.8 Feature Matrix 190
23.9 Cloud Functions 192
23.10 Getting Started With Google Compute Engine 192
23.11 Getting Started With GoGrid 196
23.12 Getting Started With Joyent 197
23.13 Getting Started With Linode 199
23.14 Cloud Map File 201
23.15 Miscellaneous Salt Cloud Options 202
23.16 Getting Started With OpenStack 205
23.17 Getting Started With Parallels 206
23.18 VM Profiles 208
23.19 Getting Started With Rackspace 210
23.20 Using the Salt Modules for Cloud 212
23.21 Getting Started With SoftLayer 216
23.22 Troubleshooting Salt Cloud 225
23.23 Spinning up Windows Minions 228
23.24 Install Salt Cloud 228

24 Getting Started **231**

25 Core Configuration **233**

26 Windows Configuration **235**

27 Using Salt Cloud **237**

28 Miscellaneous Options **239**

29 Troubleshooting Steps **241**

30 Extending Salt Cloud **243**

31 Using Salt Cloud from Salt **245**

32 Feature Comparison **247**

33 Legacy Releases **249**

34 Reference **251**

35 Virtual Machine Disk Profiles **253**
35.1 Define More Profiles 253

36 Salt Virt - The SaltStack Cloud Controller **255**
36.1 Salt Virt Tutorial . . . 255
36.2 The Salt Virt Runner . . . 255
36.3 Based on Live State Data . . . 256

37 Virtual Machine Network Profiles **257**
37.1 Define More Profiles . . . 257

38 Salt SSH **259**
38.1 Salt SSH Roster . . . 259
38.2 Calling Salt SSH . . . 260
38.3 States Via Salt SSH . . . 260
38.4 Targeting with Salt SSH . . . 260
38.5 Configuring Salt SSH . . . 260
38.6 Running Salt SSH as non-root user . . . 260

39 Salt Rosters **261**
39.1 How Rosters Work . . . 261

40 Running The Tests **263**
40.1 Running The Tests In A Docker Container . . . 263

41 Writing Tests **265**
41.1 Integration Tests . . . 265
41.2 Unit Tests . . . 265

42 Integration Tests **267**
42.1 Integration Classes . . . 267
42.2 Examples . . . 268

43 Writing Unit Tests **271**
43.1 Introduction . . . 271
43.2 Preparing to Write a Unit Test . . . 271
43.3 A Simple Example . . . 271
43.4 Evaluating Truth . . . 272
43.5 Tests Using Mock Objects . . . 272
43.6 Modifying `__salt__` In Place . . . 274

44 Reactor System **275**
44.1 Event System . . . 275
44.2 Mapping Events to Reactor SLS Files . . . 275
44.3 Fire an event . . . 276
44.4 Knowing what event is being fired . . . 276
44.5 Understanding the Structure of Reactor Formulas . . . 277
44.6 A complete example . . . 279

45 Salt Formulas **281**
45.1 Installation . . . 281
45.2 Usage . . . 282
45.3 Writing Formulas . . . 283

46 Salt Conventions **289**
46.1 Salt Formulas . . . 289
46.2 SaltStack Packaging Guide . . . 296
46.3 Salt Release Process . . . 300
46.4 Salt Coding Style . . . 301

47 SaltStack Packaging Guide **307**
47.1 Source Files . . . 307
47.2 Single Package . . . 307
47.3 Split Package . . . 307

48 Salt Release Process **313**
48.1 Feature Release Process . . . 313
48.2 Maintenance and Bugfix Releases . . . 313

49 Salt Coding Style **315**
49.1 Linting . . . 315
49.2 Strings . . . 315
49.3 Imports . . . 317
49.4 Vertical is Better . . . 317
49.5 Indenting . . . 318
49.6 Code Churn . . . 319

50 SaltStack Git Policy **321**
50.1 New Code Entry . . . 321
50.2 Release Branching . . . 321

51 Salt Development Guidelines **323**
51.1 Deprecating Code . . . 323
51.2 Dunder Dictionaries . . . 324
51.3 External Pillars . . . 325
51.4 Logging Internals . . . 328
51.5 Modular Systems . . . 328
51.6 Package Providers . . . 330

52 Translating Documentation **335**
52.1 Building A Localized Version of the Documentation . . . 335

53 Community Projects That Use Salt **337**

54 Salt Proxy Minion Documentation **339**
54.1 Getting Started . . . 339
54.2 The __proxyenabled__ directive . . . 343

55 Logging **345**
55.1 Available Configuration Settings . . . 345

56 External Logging Handlers **347**
56.1 Logstash Logging Handler . . . 347
56.2 Sentry Logging Handler . . . 348

57 Logstash Logging Handler **351**
57.1 UDP Logging Handler . . . 351
57.2 ZeroMQ Logging Handler . . . 351

58 Sentry Logging Handler **353**
58.1 Threaded Transports . . . 353

59 Introduction to Extending Salt **355**
59.1 Client API . . . 355
59.2 Adding Loadable Plugins . . . 355

60 Execution Modules 357
60.1 Modules Are Easy to Write! . . . 357
60.2 Cross Calling Modules . . . 357
60.3 Preloaded Modules Data . . . 358
60.4 Printout Configuration . . . 358
60.5 Virtual Modules . . . 359
60.6 Documentation . . . 359
60.7 How Functions are Read . . . 360
60.8 Useful Decorators for Modules . . . 360
60.9 Examples of Salt Modules . . . 361

61 Full list of builtin execution modules 363
61.1 salt.modules.pkg . . . 363
61.2 salt.modules.aliases . . . 367
61.3 salt.modules.alternatives . . . 368
61.4 salt.modules.apache . . . 369
61.5 salt.modules.aptpkg . . . 371
61.6 salt.modules.archive . . . 376
61.7 salt.modules.at . . . 378
61.8 salt.modules.augeas_cfg . . . 379
61.9 salt.modules.aws_sqs . . . 380
61.10 salt.modules.bluez . . . 381
61.11 salt.modules.brew . . . 383
61.12 salt.modules.bridge . . . 385
61.13 salt.modules.bsd_shadow . . . 386
61.14 salt.modules.cassandra . . . 387
61.15 salt.modules.chocolatey . . . 388
61.16 salt.modules.cloud . . . 391
61.17 salt.modules.cmdmod . . . 392
61.18 salt.modules.composer . . . 395
61.19 salt.modules.config . . . 396
61.20 salt.modules.cp . . . 397
61.21 salt.modules.cron . . . 400
61.22 salt.modules.daemontools . . . 401
61.23 salt.modules.darwin_sysctl . . . 403
61.24 salt.modules.data . . . 403
61.25 salt.modules.ddns . . . 404
61.26 salt.modules.debconfmod . . . 405
61.27 salt.modules.debian_ip . . . 406
61.28 salt.modules.debian_service . . . 407
61.29 salt.modules.dig . . . 409
61.30 salt.modules.disk . . . 410
61.31 salt.modules.djangomod . . . 410
61.32 salt.modules.dnsmasq . . . 411
61.33 salt.modules.dnsutil . . . 412
61.34 salt.modules.dockerio . . . 413
61.35 salt.modules.dpkg . . . 425
61.36 salt.modules.ebuild . . . 426
61.37 salt.modules.eix . . . 431
61.38 salt.modules.eselect . . . 431
61.39 salt.modules.event . . . 432
61.40 salt.modules.extfs . . . 432
61.41 salt.modules.file . . . 433
61.42 salt.modules.freebsd_sysctl . . . 448

61.43 salt.modules.freebsdjail 449
61.44 salt.modules.freebsdkmod 450
61.45 salt.modules.freebsdpkg 450
61.46 salt.modules.freebsdports 454
61.47 salt.modules.freebsdservice 455
61.48 salt.modules.gem 457
61.49 salt.modules.gentoo_service 459
61.50 salt.modules.gentoolkitmod 460
61.51 salt.modules.git 462
61.52 salt.modules.glance 468
61.53 salt.modules.gnomedesktop 469
61.54 salt.modules.grains 470
61.55 salt.modules.groupadd 473
61.56 salt.modules.grub_legacy 474
61.57 salt.modules.guestfs 474
61.58 salt.modules.hg 475
61.59 salt.modules.hosts 476
61.60 salt.modules.htpasswd 477
61.61 salt.modules.img 477
61.62 salt.modules.iptables 478
61.63 salt.modules.key 482
61.64 salt.modules.keyboard 482
61.65 salt.modules.keystone 482
61.66 salt.modules.kmod 488
61.67 salt.modules.launchctl 489
61.68 salt.modules.layman 490
61.69 salt.modules.ldapmod 490
61.70 salt.modules.linux_acl 491
61.71 salt.modules.linux_lvm 492
61.72 salt.modules.linux_sysctl 493
61.73 salt.modules.localemod 494
61.74 salt.modules.locate 494
61.75 salt.modules.logrotate 495
61.76 salt.modules.lvs 496
61.77 salt.modules.lxc 498
61.78 salt.modules.mac_group 500
61.79 salt.modules.mac_user 501
61.80 salt.modules.makeconf 502
61.81 salt.modules.match 509
61.82 salt.modules.mdadm 510
61.83 salt.modules.memcached 511
61.84 salt.modules.mine 513
61.85 salt.modules.modjk 513
61.86 salt.modules.mongodb 517
61.87 salt.modules.monit 518
61.88 salt.modules.moosefs 518
61.89 salt.modules.mount 519
61.90 salt.modules.munin 520
61.91 salt.modules.mysql 521
61.92 salt.modules.netbsd_sysctl 528
61.93 salt.modules.netbsdservice 528
61.94 salt.modules.network 530
61.95 salt.modules.nfs3 532
61.96 salt.modules.nginx 532

61.97 salt.modules.nova 533
61.98 salt.modules.npm 536
61.99 salt.modules.omapi 537
61.100 salt.modules.openbsdpkg 538
61.101 salt.modules.openbsdservice 539
61.102 salt.modules.openstack_config 540
61.103 salt.modules.osxdesktop 540
61.104 salt.modules.pacman 541
61.105 salt.modules.pagerduty 544
61.106 salt.modules.pam 544
61.107 salt.modules.parted 545
61.108 salt.modules.pecl 548
61.109 salt.modules.pillar 549
61.110 salt.modules.pip 550
61.111 salt.modules.pkg_resource 553
61.112 salt.modules.pkgin 554
61.113 salt.modules.pkgng 556
61.114 salt.modules.pkgutil 567
61.115 salt.modules.portage_config 569
61.116 salt.modules.postgres 570
61.117 salt.modules.poudriere 575
61.118 salt.modules.powerpath 576
61.119 salt.modules.ps 576
61.120 salt.modules.publish 580
61.121 salt.modules.puppet 581
61.122 salt.modules.pw_group 582
61.123 salt.modules.pw_user 582
61.124 salt.modules.qemu_img 584
61.125 salt.modules.qemu_nbd 584
61.126 salt.modules.quota 585
61.127 salt.modules.rabbitmq 586
61.128 salt.modules.rbenv 590
61.129 salt.modules.rdp 591
61.130 salt.modules.reg 591
61.131 salt.modules.ret 592
61.132 salt.modules.rh_ip 593
61.133 salt.modules.rh_service 594
61.134 salt.modules.riak 596
61.135 salt.modules.rpm 597
61.136 salt.modules.rvm 597
61.137 salt.modules.s3 600
61.138 salt.modules.saltcloudmod 602
61.139 salt.modules.saltutil 602
61.140 salt.modules.seed 605
61.141 salt.modules.selinux 605
61.142 salt.modules.service 606
61.143 salt.modules.shadow 607
61.144 salt.modules.smartos_imgadm 608
61.145 salt.modules.smartos_vmadm 609
61.146 salt.modules.smf 611
61.147 salt.modules.solaris_group 613
61.148 salt.modules.solaris_shadow 613
61.149 salt.modules.solaris_user 614
61.150 salt.modules.solarispkg 616

61.151 salt.modules.solr . . . 619
61.152 salt.modules.sqlite3 . . . 625
61.153 salt.modules.ssh . . . 626
61.154 salt.modules.state . . . 627
61.155 salt.modules.status . . . 630
61.156 salt.modules.supervisord . . . 632
61.157 salt.modules.svn . . . 634
61.158 salt.modules.sysbench . . . 637
61.159 salt.modules.sysmod . . . 638
61.160 salt.modules.system . . . 639
61.161 salt.modules.systemd . . . 640
61.162 salt.modules.test . . . 642
61.163 salt.modules.timezone . . . 645
61.164 salt.modules.tls . . . 645
61.165 salt.modules.tomcat . . . 648
61.166 salt.modules.upstart . . . 652
61.167 salt.modules.useradd . . . 654
61.168 salt.modules.virt . . . 656
61.169 salt.modules.virtualenv . . . 663
61.170 salt.modules.win_autoruns . . . 664
61.171 salt.modules.win_disk . . . 664
61.172 salt.modules.win_dns_client . . . 664
61.173 salt.modules.win_file . . . 665
61.174 salt.modules.win_firewall . . . 667
61.175 salt.modules.win_groupadd . . . 667
61.176 salt.modules.win_ip . . . 668
61.177 salt.modules.win_network . . . 670
61.178 salt.modules.win_ntp . . . 672
61.179 salt.modules.win_path . . . 672
61.180 salt.modules.win_pkg . . . 673
61.181 salt.modules.win_repo . . . 675
61.182 salt.modules.win_servermanager . . . 675
61.183 salt.modules.win_service . . . 676
61.184 salt.modules.win_shadow . . . 678
61.185 salt.modules.win_status . . . 678
61.186 salt.modules.win_system . . . 679
61.187 salt.modules.win_timezone . . . 681
61.188 salt.modules.win_useradd . . . 682
61.189 salt.modules.xapi . . . 683
61.190 salt.modules.xmpp . . . 688
61.191 salt.modules.yumpkg . . . 688
61.192 salt.modules.zcbuildout . . . 694
61.193 salt.modules.zfs . . . 696
61.194 salt.modules.zpool . . . 697
61.195 salt.modules.zypper . . . 698

62 Returners **701**
62.1 Using Returners . . . 701
62.2 Writing a Returner . . . 701

63 Full list of builtin returner modules **703**
63.1 salt.returners.carbon_return . . . 703
63.2 salt.returners.cassandra_return . . . 703
63.3 salt.returners.couchdb_return . . . 704

63.4 salt.returners.local . . . 704
63.5 salt.returners.memcache_return . . . 704
63.6 salt.returners.mongo_future_return . . . 705
63.7 salt.returners.mongo_return . . . 706
63.8 salt.returners.mysql . . . 706
63.9 salt.returners.postgres . . . 707
63.10 salt.returners.redis_return . . . 709
63.11 salt.returners.sentry_return . . . 709
63.12 salt.returners.smtp_return . . . 710
63.13 salt.returners.sqlite3 . . . 710
63.14 salt.returners.syslog_return . . . 711

64 File State Backups **713**
64.1 Backed-up Files . . . 713
64.2 Interacting with Backups . . . 713

65 Extending External SLS Data **717**
65.1 The Extend Declaration . . . 717
65.2 Extend is a Top Level Declaration . . . 717
65.3 The Requisite "in" Statement . . . 718
65.4 Rules to Extend By . . . 718

66 Failhard Global Option **719**
66.1 State Level Failhard . . . 719
66.2 Global Failhard . . . 719

67 Highstate data structure definitions **721**
67.1 The Salt State Tree . . . 721
67.2 Large example . . . 725

68 Include and Exclude **727**
68.1 Include . . . 727
68.2 Relative Include . . . 727
68.3 Exclude . . . 727

69 State Enforcement **729**
69.1 State management . . . 729
69.2 Understanding the Salt State System Components . . . 729

70 State System Layers **735**
70.1 Function Call . . . 735
70.2 Low Chunk . . . 735
70.3 Low State . . . 735
70.4 High Data . . . 736
70.5 SLS . . . 736
70.6 HighState . . . 736
70.7 OverState . . . 736

71 The Orchestrate Runner **737**

72 Ordering States **739**
72.1 State Auto Ordering . . . 739
72.2 Requisite Statements . . . 739
72.3 The Order Option . . . 743

73 OverState System **745**

74 State Providers 747
74.1 Setting a Provider in the Minion Config File . . . 747
74.2 Arbitrary Module Redirects . . . 749

75 Requisites 751
75.1 Requisite and Requisite in types . . . 752

76 Startup States 757
76.1 Examples: . . . 757

77 State Testing 759
77.1 Default Test . . . 759

78 The Top File 761
78.1 Environments . . . 761
78.2 Other Ways of Targeting Minions . . . 763
78.3 How Top Files Are Compiled . . . 764

79 SLS Template Variable Reference 767
79.1 Salt . . . 767
79.2 Opts . . . 767
79.3 Pillar . . . 767
79.4 Grains . . . 768
79.5 env . . . 768
79.6 sls . . . 768

80 State Modules 769
80.1 States are Easy to Write! . . . 769
80.2 Using Custom State Modules . . . 769
80.3 Cross Calling Modules . . . 770
80.4 Return Data . . . 770
80.5 Test State . . . 770
80.6 Watcher Function . . . 770
80.7 Mod_init Interface . . . 771

81 Full list of builtin state modules 773
81.1 salt.states.alias . . . 775
81.2 salt.states.alternatives . . . 775
81.3 salt.states.apt . . . 776
81.4 salt.states.archive . . . 776
81.5 salt.states.augeas . . . 777
81.6 salt.states.aws_sqs . . . 778
81.7 salt.states.cloud . . . 778
81.8 salt.states.cmd . . . 779
81.9 salt.states.composer . . . 785
81.10 salt.states.cron . . . 786
81.11 salt.states.ddns . . . 789
81.12 salt.states.debconfmod . . . 789
81.13 salt.states.disk . . . 791
81.14 salt.states.dockerio . . . 791
81.15 salt.states.eselect . . . 794
81.16 salt.states.file . . . 795
81.17 salt.states.gem . . . 809
81.18 salt.states.git . . . 810
81.19 salt.states.gnomedesktop . . . 811

81.20 salt.states.grains . . . 813
81.21 salt.states.group . . . 814
81.22 salt.states.hg . . . 814
81.23 salt.states.host . . . 815
81.24 salt.states.iptables . . . 816
81.25 salt.states.keyboard . . . 818
81.26 salt.states.keystone . . . 819
81.27 salt.states.kmod . . . 821
81.28 salt.states.layman . . . 821
81.29 salt.states.libvirt . . . 822
81.30 salt.states.locale . . . 822
81.31 salt.states.lvm . . . 822
81.32 salt.states.lvs_server . . . 823
81.33 salt.states.lvs_service . . . 824
81.34 salt.states.makeconf . . . 824
81.35 salt.states.mdadm . . . 825
81.36 salt.states.memcached . . . 826
81.37 salt.states.modjk_worker . . . 826
81.38 salt.states.module . . . 827
81.39 salt.states.mongodb_database . . . 828
81.40 salt.states.mongodb_user . . . 829
81.41 salt.states.mount . . . 829
81.42 salt.states.mysql_database . . . 830
81.43 salt.states.mysql_grants . . . 831
81.44 salt.states.mysql_user . . . 832
81.45 salt.states.network . . . 833
81.46 salt.states.npm . . . 836
81.47 salt.states.ntp . . . 837
81.48 salt.states.openstack_config . . . 838
81.49 salt.states.pagerduty . . . 838
81.50 salt.states.pecl . . . 839
81.51 salt.states.pip_state . . . 839
81.52 salt.states.pkg . . . 841
81.53 salt.states.pkgng . . . 844
81.54 salt.states.pkgrepo . . . 845
81.55 salt.states.portage_config . . . 847
81.56 salt.states.ports . . . 847
81.57 salt.states.postgres_database . . . 848
81.58 salt.states.postgres_group . . . 849
81.59 salt.states.postgres_user . . . 850
81.60 salt.states.powerpath . . . 852
81.61 salt.states.process . . . 852
81.62 salt.states.quota . . . 852
81.63 salt.states.rabbitmq_cluster . . . 853
81.64 salt.states.rabbitmq_plugin . . . 853
81.65 salt.states.rabbitmq_policy . . . 854
81.66 salt.states.rabbitmq_user . . . 854
81.67 salt.states.rabbitmq_vhost . . . 855
81.68 salt.states.rbenv . . . 856
81.69 salt.states.rdp . . . 857
81.70 salt.states.reg . . . 857
81.71 salt.states.rvm . . . 857
81.72 salt.states.saltmod . . . 859
81.73 salt.states.selinux . . . 861

81.74 salt.states.service 861
81.75 salt.states.ssh_auth 862
81.76 salt.states.ssh_known_hosts 864
81.77 salt.states.stateconf 865
81.78 salt.states.status 865
81.79 salt.states.supervisord 865
81.80 salt.states.svn 866
81.81 salt.states.sysctl 867
81.82 salt.states.timezone 867
81.83 salt.states.tomcat 868
81.84 salt.states.user 869
81.85 salt.states.virtualenv 871
81.86 salt.states.win_dns_client 872
81.87 salt.states.win_firewall 872
81.88 salt.states.win_network 872
81.89 salt.states.win_path 873
81.90 salt.states.win_servermanager 874
81.91 salt.states.win_system 874
81.92 salt.states.xmpp 875
81.93 salt.states.zcbuildout 875

82 Renderers 877
82.1 Multiple Renderers 877
82.2 Composing Renderers 877
82.3 Writing Renderers 878
82.4 Examples 878

83 Full list of builtin renderer modules 881
83.1 salt.renderers.jinja 881
83.2 salt.renderers.json 885
83.3 salt.renderers.mako 885
83.4 salt.renderers.py 886
83.5 salt.renderers.pydsl 886
83.6 salt.renderers.stateconf 891
83.7 salt.renderers.wempy 895
83.8 salt.renderers.yaml 895

84 Pillars 897

85 Full list of builtin pillar modules 899
85.1 salt.pillar.cmd_json 899
85.2 salt.pillar.cmd_yaml 899
85.3 salt.pillar.cobbler 899
85.4 salt.pillar.django_orm 900
85.5 salt.pillar.git_pillar 902
85.6 salt.pillar.hiera 902
85.7 salt.pillar.libvirt 903
85.8 salt.pillar.mongo 903
85.9 salt.pillar.pillar_ldap 904
85.10 salt.pillar.puppet 904
85.11 salt.pillar.reclass_adapter 904
85.12 salt.pillar.virtkey 905

86 Master Tops 907

87 Full list of builtin master tops modules 909
87.1 salt.tops.cobbler . . . 909
87.2 salt.tops.ext_nodes . . . 909
87.3 salt.tops.mongo . . . 910
87.4 salt.tops.reclass_adapter . . . 911

88 Salt Runners 913
88.1 Writing Salt Runners . . . 913
88.2 Examples . . . 913

89 Full list of runner modules 915
89.1 salt.runners.cache . . . 915
89.2 salt.runners.doc . . . 916
89.3 salt.runners.fileserver . . . 916
89.4 salt.runners.jobs . . . 917
89.5 salt.runners.launchd . . . 917
89.6 salt.runners.lxc . . . 917
89.7 salt.runners.manage . . . 919
89.8 salt.runners.network . . . 920
89.9 salt.runners.search . . . 921
89.10 salt.runners.state . . . 921
89.11 salt.runners.thin . . . 922
89.12 salt.runners.virt . . . 922
89.13 salt.runners.winrepo . . . 923

90 Full list of builtin wheel modules 925
90.1 salt.wheel.config . . . 925
90.2 salt.wheel.file_roots . . . 925
90.3 salt.wheel.key . . . 926
90.4 salt.wheel.pillar_roots . . . 926

91 Full list of builtin auth modules 927
91.1 salt.auth.auto . . . 927
91.2 salt.auth.keystone . . . 927
91.3 salt.auth.ldap . . . 927
91.4 salt.auth.pam . . . 928
91.5 salt.auth.stormpath_mod . . . 929

92 Full list of builtin output modules 931
92.1 salt.output.grains . . . 931
92.2 salt.output.highstate . . . 931
92.3 salt.output.json_out . . . 932
92.4 salt.output.key . . . 932
92.5 salt.output.nested . . . 932
92.6 salt.output.no_out . . . 932
92.7 salt.output.no_return . . . 932
92.8 salt.output.overstatestage . . . 933
92.9 salt.output.pprint_out . . . 933
92.10 salt.output.raw . . . 933
92.11 salt.output.txt . . . 933
92.12 salt.output.virt_query . . . 933
92.13 salt.output.yaml_out . . . 934

93 Full list of Salt Cloud modules 935
93.1 salt.cloud.clouds.botocore_aws . . . 935

93.2 salt.cloud.clouds.cloudstack 936
93.3 salt.cloud.clouds.digital_ocean 937
93.4 salt.cloud.clouds.ec2 938
93.5 salt.cloud.clouds.gce 942
93.6 salt.cloud.clouds.gogrid 944
93.7 salt.cloud.clouds.ibmsce 945
93.8 salt.cloud.clouds.joyent 946
93.9 salt.cloud.clouds.libcloud_aws 949
93.10 salt.cloud.clouds.linode 951
93.11 salt.cloud.clouds.msazure 952
93.12 salt.cloud.clouds.nova 953
93.13 salt.cloud.clouds.openstack 955
93.14 salt.cloud.clouds.parallels 957
93.15 salt.cloud.clouds.rackspace 959
93.16 salt.cloud.clouds.saltify 960
93.17 salt.cloud.clouds.softlayer 960

94 Python client API **963**
94.1 Calling Salt from shell scripts 963
94.2 Calling Salt via a REST API 963
94.3 Calling Salt from a Python application 963
94.4 Salt's Python interface 964

95 Peer Communication **971**
95.1 Peer Configuration 971
95.2 Peer Runner Communication 972
95.3 Using Peer Communication 972

96 Client ACL system **973**
96.1 Permission Issues 973

97 Salt Syndic **975**
97.1 Configuring the Syndic 975
97.2 Running the Syndic 975

98 File Server Backends **977**
98.1 Environments 977

99 Dynamic Module Distribution **979**
99.1 Sync Via States 979
99.2 Sync Via the saltutil Module 979

100File Server Configuration **981**
100.1 Environments 981
100.2 Directory Overlay 981
100.3 Local File Server 982

101Salt File Server **983**
101.1 The cp Module 983
101.2 File Server Client API 984

102Full list of builtin fileserver modules **987**
102.1 salt.fileserver.gitfs 987
102.2 salt.fileserver.hgfs 988
102.3 salt.fileserver.roots 988

102.4 salt.fileserver.s3fs 989
102.5 salt.fileserver.minionfs 990

103 Configuration file examples 993
103.1 Example master configuration file 993
103.2 Example minion configuration file 1003

104 Configuring the Salt Master 1015
104.1 Primary Master Configuration 1015
104.2 Master Security Settings 1018
104.3 Master Module Management 1020
104.4 Master State System Settings 1020
104.5 Master File Server Settings 1022
104.6 Pillar Configuration 1026
104.7 Syndic Server Settings 1026
104.8 Peer Publish Settings 1027
104.9 Node Groups 1028
104.10 Master Logging Settings 1029
104.11 Include Configuration 1030

105 Configuring the Salt Minion 1031
105.1 Minion Primary Configuration 1031
105.2 Minion Module Management 1035
105.3 State Management Settings 1036
105.4 File Directory Settings 1037
105.5 Security Settings 1038
105.6 Thread Settings 1039
105.7 Minion Logging Settings 1039
105.8 Include Configuration 1040
105.9 Frozen Build Update Settings 1041

106 Salt code and internals 1043
106.1 Contents 1043

107 Network Topology 1047
107.1 Servers 1047
107.2 PUB/SUB 1047
107.3 Return 1047

108 Windows Software Repository 1049
108.1 Operation 1049
108.2 Usage 1050
108.3 Generate Repo Cache File 1051
108.4 Install Windows Software 1052
108.5 Uninstall Windows Software 1052
108.6 Standalone Minion Salt Windows Repo Module 1052
108.7 Git Hosted Repo 1052
108.8 Troubleshooting 1053

109 Command Line Reference 1055
109.1 Using the Salt Command 1055

110 `salt` 1059
110.1 Synopsis 1059
110.2 Description 1059

110.3 Options 1059
110.4 See also 1062

111 `salt-master` 1063
111.1 Synopsis 1063
111.2 Description 1063
111.3 Options 1063
111.4 See also 1064

112 `salt-minion` 1065
112.1 Synopsis 1065
112.2 Description 1065
112.3 Options 1065
112.4 See also 1066

113 `salt-key` 1067
113.1 Synopsis 1067
113.2 Description 1067
113.3 Options 1067
113.4 See also 1069

114 `salt-cp` 1071
114.1 Synopsis 1071
114.2 Description 1071
114.3 Options 1071
114.4 See also 1072

115 `salt-call` 1073
115.1 Synopsis 1073
115.2 Description 1073
115.3 Options 1073
115.4 See also 1074

116 `salt-run` 1075
116.1 Synopsis 1075
116.2 Description 1075
116.3 Options 1075
116.4 See also 1076

117 `salt-ssh` 1077
117.1 Synopsis 1077
117.2 Description 1077
117.3 Options 1077
117.4 See also 1079

118 `salt-syndic` 1081
118.1 Synopsis 1081
118.2 Description 1081
118.3 Options 1081
118.4 See also 1082

119 `salt-cloud` 1083
119.1 Synopsis 1083
119.2 Description 1083
119.3 Options 1083

119.4 Examples 1084
119.5 See also 1085

120 Release notes and upgrade instructions 1087

120.1 Salt 0.10.0 Release Notes 1087
120.2 Salt 0.10.1 Release Notes 1088
120.3 Salt 0.10.2 Release Notes 1088
120.4 Salt 0.10.3 Release Notes 1091
120.5 Salt 0.10.4 Release Notes 1094
120.6 Salt 0.10.5 Release Notes 1096
120.7 Salt 0.11.0 Release Notes 1098
120.8 Salt 0.11.1 Release Notes 1100
120.9 Salt 0.12.0 Release Notes 1100
120.10 Salt 0.12.1 Release Notes 1102
120.11 Salt 0.13.0 Release Notes 1102
120.12 Salt 0.13.1 Release Notes 1103
120.13 Salt 0.13.2 Release Notes 1103
120.14 Salt 0.13.3 Release Notes 1103
120.15 Salt 0.14.0 Release Notes 1103
120.16 Salt 0.14.1 Release Notes 1105
120.17 Salt 0.15.0 Release Notes 1105
120.18 Salt 0.15.1 Release Notes 1107
120.19 Salt 0.15.2 Release Notes 1108
120.20 Salt 0.15.3 Release Notes 1108
120.21 Salt 0.16.0 Release Notes 1108
120.22 Salt 0.16.1 Release Notes 1110
120.23 Salt 0.16.2 Release Notes 1110
120.24 Salt 0.16.3 Release Notes 1113
120.25 Salt 0.16.4 Release Notes 1114
120.26 Salt 0.17.0 Release Notes 1114
120.27 Salt 0.17.1 Release Notes 1117
120.28 Salt 0.17.2 Release Notes 1122
120.29 Salt 0.17.3 Release Notes 1124
120.30 Salt 0.17.4 Release Notes 1125
120.31 Salt 0.6.0 release notes 1126
120.32 Salt 0.7.0 release notes 1127
120.33 Salt 0.8.0 release notes 1128
120.34 Salt 0.8.7 release notes 1130
120.35 Salt 0.8.8 release notes 1131
120.36 Salt 0.8.9 Release Notes 1132
120.37 Salt 0.9.0 Release Notes 1134
120.38 Salt 0.9.1 Release Notes 1136
120.39 Salt 0.9.2 Release Notes 1137
120.40 Salt 0.9.3 Release Notes 1138
120.41 Salt 0.9.4 Release Notes 1142
120.42 Salt 0.9.5 Release Notes 1144
120.43 Salt 0.9.6 Release Notes 1151
120.44 Salt 0.9.7 Release Notes 1152
120.45 Salt 0.9.8 Release Notes 1154
120.46 Salt 0.9.9 Release Notes 1159
120.47 Salt 2014.1.0 Release Notes - Codename Hydrogen 1163
120.48 Salt 2014.1.1 Release Notes 1167
120.49 Salt 2014.1.2 Release Notes 1168
120.50 Salt 2014.1.2 Release Notes 1170

120.51Salt 2014.1.4 Release Notes 1171

Salt Module Index **1173**

Frequently Asked Questions

FAQ

- Frequently Asked Questions
  - Is Salt open-core?
  - What ports should I open on my firewall?
  - I'm seeing weird behavior (including but not limited to packages not installing their users properly)
  - My script runs every time I run a *state.highstate*. Why?
  - When I run *test.ping*, why don't the Minions that aren't responding return anything? Returning `False` would be helpful.
  - How does Salt determine the Minion's id?
  - I'm trying to manage packages/services but I get an error saying that the state is not available. Why?
  - I'm using gitfs and my custom modules/states/etc are not syncing. Why?
  - Why aren't my custom modules/states/etc. available on my Minions?
  - Module `X` isn't available, even though the shell command it uses is installed. Why?
  - Can I run different versions of Salt on my Master and Minion?
  - Does Salt support backing up managed files?

1.1 Is Salt open-core?

No. Salt is 100% committed to being open-source, including all of our APIs and the new 'Halite' web interface which was introduced in version 0.17.0. It is developed under the Apache 2.0 license, allowing it to be used in both open and proprietary projects.

1.2 What ports should I open on my firewall?

Minions need to be able to connect to the Master on TCP ports 4505 and 4506. Minions do not need any inbound ports open. More detailed information on firewall settings can be found *here*.

1.3 I'm seeing weird behavior (including but not limited to packages not installing their users properly)

This is often caused by SELinux. Try disabling SELinux or putting it in permissive mode and see if the weird behavior goes away.

1.4 My script runs every time I run a *state.highstate*. Why?

You are probably using `cmd.run` rather than `cmd.wait`. A `cmd.wait` state will only run when there has been a change in a state that it is watching.

A `cmd.run` state will run the corresponding command *every time* (unless it is prevented from running by the `unless` or `onlyif` arguments).

More details can be found in the docmentation for the `cmd` states.

1.5 When I run *test.ping*, why don't the Minions that aren't responding return anything? Returning `False` would be helpful.

When you run *test.ping* the Master tells Minions to run commands/functions, and listens for the return data, printing it to the screen when it is received. If it doesn't receive anything back, it doesn't have anything to display for that Minion.

There are a couple options for getting information on Minions that are not responding. One is to use the verbose (`-v`) option when you run salt commands, as it will display "Minion did not return" for any Minions which time out.

```
salt -v '*' pkg.install zsh
```

Another option is to use the `manage.down` runner:

```
salt-run manage.down
```

1.6 How does Salt determine the Minion's id?

If the Minion id is not configured explicitly (using the `id` parameter), Salt will determine the id based on the hostname. Exactly how this is determined varies a little between operating systems and is described in detail *here*.

1.7 I'm trying to manage packages/services but I get an error saying that the state is not available. Why?

Salt detects the Minion's operating system and assigns the correct package or service management module based on what is detected. However, for certain custom spins and OS derivatives this detection fails. In cases like this, an issue should be opened on our tracker, with the following information:

1. The output of the following command:

   ```
   salt <minion_id> grains.items | grep os
   ```

2. The contents of `/etc/lsb-release`, if present on the Minion.

1.8 I'm using gitfs and my custom modules/states/etc are not syncing. Why?

In versions of Salt 0.16.3 or older, there is a bug in *gitfs* which can affect the syncing of custom types. Upgrading to 0.16.4 or newer will fix this.

1.9 Why aren't my custom modules/states/etc. available on my Minions?

Custom modules are only synced to Minions when `state.highstate`, `saltutil.sync_modules`, or `saltutil.sync_all` is run. Similarly, custom states are only synced to Minions when `state.highstate`, `saltutil.sync_states`, or `saltutil.sync_all` is run.

Other custom types (renderers, outputters, etc.) have similar behavior, see the documentation for the `saltutil` module for more information.

1.10 Module x isn't available, even though the shell command it uses is installed. Why?

This is most likely a PATH issue. Did you custom-compile the software which the module requires? RHEL/CentOS/etc. in particular override the root user's path in `/etc/init.d/functions`, setting it to `/sbin:/usr/sbin:/bin:/usr/bin`, making software installed into `/usr/local/bin` unavailable to Salt when the Minion is started using the initscript. In version 2014.1.0, Salt will have a better solution for these sort of PATH-related issues, but recompiling the software to install it into a location within the PATH should resolve the issue in the meantime. Alternatively, you can create a symbolic link within the PATH using a `file.symlink` state.

```
/usr/bin/foo:
  file.symlink:
    - target: /usr/local/bin/foo
```

1.11 Can I run different versions of Salt on my Master and Minion?

As of release 0.17.1 backwards compatibility was broken (specifically for 0.17.1 trying to interface with older releases) due to a protocol change for security purposes. The Salt team continues to emphasize backwards compatiblity as an important feature and plans to support it to the best of our ability to do so.

1.12 Does Salt support backing up managed files?

Yes. Salt provides an easy to use addition to your file.managed states that allow you to back up files via *backup_mode*, backup_mode can be configured on a per state basis, or in the minion config (note that if set in the minion config this would simply be the default method to use, you still need to specify that the file should be backed up!).

CHAPTER 2

Introduction to Salt

We're not just talking about NaCl.

2.1 The 30 second summary

Salt is:

- a configuration management system, capable of maintaining remote nodes in defined states (for example, ensuring that specific packages are installed and specific services are running)
- a distributed remote execution system used to execute commands and query data on remote nodes, either individually or by arbitrary selection criteria

It was developed in order to bring the best solutions found in the world of remote execution together and make them better, faster, and more malleable. Salt accomplishes this through its ability to handle large loads of information, and not just dozens but hundreds and even thousands of individual servers quickly through a simple and manageable interface.

2.2 Simplicity

Providing versatility between massive scale deployments and smaller systems may seem daunting, but Salt is very simple to set up and maintain, regardless of the size of the project. The architecture of Salt is designed to work with any number of servers, from a handful of local network systems to international deployments across different datacenters. The topology is a simple server/client model with the needed functionality built into a single set of daemons. While the default configuration will work with little to no modification, Salt can be fine tuned to meet specific needs.

2.3 Parallel execution

The core functions of Salt:

- enable commands to remote systems to be called in parallel rather than serially
- use a secure and encrypted protocol
- use the smallest and fastest network payloads possible
- provide a simple programming interface

Salt also introduces more granular controls to the realm of remote execution, allowing systems to be targeted not just by hostname, but also by system properties.

2.4 Building on proven technology

Salt takes advantage of a number of technologies and techniques. The networking layer is built with the excellent ZeroMQ networking library, so the Salt daemon includes a viable and transparent AMQ broker. Salt uses public keys for authentication with the master daemon, then uses faster AES encryption for payload communication; authentication and encryption are integral to Salt. Salt takes advantage of communication via msgpack, enabling fast and light network traffic.

2.5 Python client interface

In order to allow for simple expansion, Salt execution routines can be written as plain Python modules. The data collected from Salt executions can be sent back to the master server, or to any arbitrary program. Salt can be called from a simple Python API, or from the command line, so that Salt can be used to execute one-off commands as well as operate as an integral part of a larger application.

2.6 Fast, flexible, scalable

The result is a system that can execute commands at high speed on target server groups ranging from one to very many servers. Salt is very fast, easy to set up, amazingly malleable and provides a single remote execution architecture that can manage the diverse requirements of any number of servers. The Salt infrastructure brings together the best of the remote execution world, amplifies its capabilities and expands its range, resulting in a system that is as versatile as it is practical, suitable for any network.

2.7 Open

Salt is developed under the Apache 2.0 license, and can be used for open and proprietary projects. Please submit your expansions back to the Salt project so that we can all benefit together as Salt grows. Please feel free to sprinkle Salt around your systems and let the deliciousness come forth.

CHAPTER 3

Installation

The Salt system setup is amazingly simple, as this is one of the central design goals of Salt.

See also:

Installing Salt for development and contributing to the project.

3.1 Quick Install

Many popular distributions will be able to install the salt minion by executing the bootstrap script:

```
wget -O - http://bootstrap.saltstack.org | sudo sh
```

Run the following script to install just the Salt Master:

```
curl -L http://bootstrap.saltstack.org | sudo sh -s -- -M -N
```

The script should also make it simple to install a salt master, if desired.

Currently the install script has been tested to work on:

- Ubuntu 10.x/11.x/12.x
- Debian 6.x
- CentOS 6.3
- Fedora
- Arch
- FreeBSD 9.0

See Salt Bootstrap for more information.

3.2 Platform-specific installation instructions

These guides go into detail how to install salt on a given platform.

3.2.1 Arch Linux

Installation

Salt is currently available via the Arch User Repository (AUR). There are currently stable and -git packages available.

Stable Release

Install Salt stable releases from the Arch Linux AUR as follows:

```
wget https://aur.archlinux.org/packages/sa/salt/salt.tar.gz
tar xf salt.tar.gz
cd salt/
makepkg -is
```

A few of Salt's dependencies are currently only found within the AUR, so it is necessary to download and run `makepkg -is` on these as well. As a reference, Salt currently relies on the following packages which are only available via the AUR:

- https://aur.archlinux.org/packages/py/python2-msgpack/python2-msgpack.tar.gz
- https://aur.archlinux.org/packages/py/python2-psutil/python2-psutil.tar.gz

---

Note: yaourt

If a tool such as Yaourt is used, the dependencies will be gathered and built automatically.

The command to install salt using the yaourt tool is:

```
yaourt salt
```

Tracking develop

To install the bleeding edge version of Salt (**may include bugs!**), use the -git package. Installing the -git package as follows:

```
wget https://aur.archlinux.org/packages/sa/salt-git/salt-git.tar.gz
tar xf salt-git.tar.gz
cd salt-git/
makepkg -is
```

See the note above about Salt's dependencies.

Post-installation tasks

systemd

Activate the Salt Master and/or Minion via `systemctl` as follows:

```
systemctl enable salt-master.service
systemctl enable salt-minion.service
```

Start the Master

Once you've completed all of these steps you're ready to start your Salt Master. You should be able to start your Salt Master now using the command seen here:

```
systemctl start salt-master
```

Now go to the *Configuring Salt* page.

3.2.2 Debian Installation

Currently the latest packages for Debian Old Stable, Stable and Unstable (Squeeze, Wheezy and Sid) are published in our (saltstack.com) debian repository.

Configure Apt

Squeeze (Old Stable)

For squeeze, you will need to enable the debian backports repository as well as the debian.saltstack.com repository. To do so, add the following to `/etc/apt/sources.list` or a file in `/etc/apt/sources.list.d`:

```
deb http://debian.saltstack.com/debian squeeze-saltstack main
deb http://backports.debian.org/debian-backports squeeze-backports main contrib non-free
```

Wheezy (Stable)

For wheezy, the following line is needed in either `/etc/apt/sources.list` or a file in `/etc/apt/sources.list.d`:

```
deb http://debian.saltstack.com/debian wheezy-saltstack main
```

Sid (Unstable)

For sid, the following line is needed in either `/etc/apt/sources.list` or a file in `/etc/apt/sources.list.d`:

```
deb http://debian.saltstack.com/debian unstable main
```

Import the repository key.

You will need to import the key used for signing.

```
wget -q -O- "http://debian.saltstack.com/debian-salt-team-joehealy.gpg.key" | apt-key add -
```

Note: You can optionally verify the key integrity with `sha512sum` using the public key signature shown here. E.g:

```
echo "b702969447140d5553e31e9701be13ca11cc0a7ed5fe2b30acb8491567560ee62f834772b5095d735dfcecb2384a5c
```

Update the package database

```
apt-get update
```

Install packages

Install the Salt master, minion, or syndic from the repository with the apt-get command. These examples each install one daemon, but more than one package name may be given at a time:

```
apt-get install salt-master
```

```
apt-get install salt-minion
```

```
apt-get install salt-syndic
```

Post-installation tasks

Now, go to the *Configuring Salt* page.

Notes

1. These packages will be backported from the packages intended to be uploaded into debian unstable. This means that the packages will be built for unstable first and then backported over the next day or so.

2. These packages will be tracking the released versions of salt rather than maintaining a stable fixed feature set. If a fixed version is what you desire, then either pinning or manual installation may be more appropriate for you.

3. The version numbering and backporting process should provide clean upgrade paths between debian versions.

If you have any questions regarding these, please email the mailing list or look for joehh on irc.

3.2.3 Fedora

Beginning with version 0.9.4, Salt has been available in the primary Fedora repositories and EPEL. It is installable using yum. Fedora will have more up to date versions of Salt than other members of the Red Hat family, which makes it a great place to help improve Salt!

Installation

Salt can be installed using `yum` and is available in the standard Fedora repositories.

Stable Release

Salt is packaged separately for the minion and the master. It is necessary only to install the appropriate package for the role the machine will play. Typically, there will be one master and multiple minions.

```
yum install salt-master
yum install salt-minion
```

Post-installation tasks

Master

To have the Master start automatically at boot time:

```
systemctl enable salt-master.service
```

To start the Master:

```
systemctl start salt-master.service
```

Minion

To have the Minion start automatically at boot time:

```
systemctl enable salt-minion.service
```

To start the Minion:

```
systemctl start salt-minion.service
```

Now go to the *Configuring Salt* page.

3.2.4 FreeBSD

Salt was added to the FreeBSD ports tree Dec 26th, 2011 by Christer Edwards <christer.edwards@gmail.com>. It has been tested on FreeBSD 7.4, 8.2, 9.0 and 9.1 releases.

Salt is dependent on the following additional ports. These will be installed as dependencies of the `sysutils/py-salt` port.

```
/devel/py-yaml
/devel/py-pyzmq
/devel/py-Jinja2
/devel/py-msgpack
/security/py-pycrypto
/security/py-m2crypto
```

Installation

To install Salt from the FreeBSD ports tree, use the command:

```
make -C /usr/ports/sysutils/py-salt install clean
```

Post-installation tasks

Master

Copy the sample configuration file:

```
cp /usr/local/etc/salt/master.sample /usr/local/etc/salt/master
```

rc.conf

Activate the Salt Master in `/etc/rc.conf` or `/etc/rc.conf.local` and add:

```
+ salt_master_enable="YES"
```

Start the Master

Start the Salt Master as follows:

```
service salt_master start
```

Minion

Copy the sample configuration file:

```
cp /usr/local/etc/salt/minion.sample /usr/local/etc/salt/minion
```

rc.conf

Activate the Salt Minion in `/etc/rc.conf` or `/etc/rc.conf.local` and add:

```
+ salt_minion_enable="YES"
+ salt_minion_paths="/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin"
```

Start the Minion

Start the Salt Minion as follows:

```
service salt_minion start
```

Now go to the *Configuring Salt* page.

3.2.5 Gentoo

Salt can be easily installed on Gentoo via Portage:

```
emerge app-admin/salt
```

Post-installation tasks

Now go to the *Configuring Salt* page.

3.2.6 OS X

Dependency Installation

When installing via Homebrew, dependency resolution is handled for you.

```
brew install saltstack
```

When using macports, zmq, swig, and pip may need to be installed this way:

```
sudo port install py-zmq
sudo port install py27-m2crypto
sudo port install py27-crypto
sudo port install py27-msgpack
sudo port install swig-python
sudo port install py-pip
```

For installs using the OS X system python, pip install needs to use 'sudo':

```
sudo pip install salt
```

Salt-Master Customizations

To run salt-master on OS X, the root user maxfiles limit must be increased:

```
sudo launchctl limit maxfiles 4096 8192
```

And sudo add this configuration option to the /etc/salt/master file:

```
max_open_files: 8192
```

Now the salt-master should run without errors:

```
sudo /usr/local/share/python/salt-master --log-level=all
```

Post-installation tasks

Now go to the *Configuring Salt* page.

3.2.7 RHEL / CentOS / Scientific Linux / Amazon Linux / Oracle Linux

Beginning with version 0.9.4, Salt has been available in EPEL. It is installable using yum. Salt should work properly with all mainstream derivatives of RHEL, including CentOS, Scientific Linux, Oracle Linux and Amazon Linux. Report any bugs or issues to the salt GitHub project.

Installation

Salt and all dependencies have been accepted into the yum repositories for EPEL5 and EPEL6. The latest salt version can be found in epel-testing, while an older but more tested version can be found in regular epel.

Example showing how to install salt from epel-testing:

```
yum --enablerepo=epel-testing install salt-minion
```

On RHEL6, the proper Jinja package ‘python-jinja2’ was moved from EPEL to the “RHEL Server Optional Channel”. Verify this repository is enabled before installing salt on RHEL6.

Salt can be installed using `yum` and is available in the standard Fedora repositories.

Enabling EPEL on RHEL

If EPEL is not enabled on your system, you can use the following commands to enable it.

For RHEL 5:

```
rpm -Uvh http://mirror.pnl.gov/epel/5/i386/epel-release-5-4.noarch.rpm
```

For RHEL 6:

```
rpm -Uvh http://ftp.linux.ncsu.edu/pub/epel/6/i386/epel-release-6-8.noarch.rpm
```

Stable Release

Salt is packaged separately for the minion and the master. It is necessary only to install the appropriate package for the role the machine will play. Typically, there will be one master and multiple minions.

On the salt-master, run this:

```
yum install salt-master
```

On each salt-minion, run this:

```
yum install salt-minion
```

Post-installation tasks

Master

To have the Master start automatically at boot time:

```
chkconfig salt-master on
```

To start the Master:

```
service salt-master start
```

Minion

To have the Minion start automatically at boot time:

```
chkconfig salt-minion on
```

To start the Minion:

```
service salt-minion start
```

Now go to the *Configuring Salt* page.

3.2.8 Solaris

Salt was added to the OpenCSW package repository in September of 2012 by Romeo Theriault <romeot@hawaii.edu> at version 0.10.2 of Salt. It has mainly been tested on Solaris 10 (sparc), though it is built for and has been tested minimally on Solaris 10 (x86), Solaris 9 (sparc/x86) and 11 (sparc/x86). (Please let me know if you're using it on these platforms!) Most of the testing has also just focused on the minion, though it has verified that the master starts up successfully on Solaris 10.

Comments and patches for better support on these platforms is very welcome.

As of version 0.10.4, Solaris is well supported under salt, with all of the following working well:

1. remote execution
2. grain detection
3. service control with SMF
4. 'pkg' states with 'pkgadd' and 'pkgutil' modules
5. cron modules/states
6. user and group modules/states

7. shadow password management modules/states

Salt is dependent on the following additional packages. These will automatically be installed as dependencies of the `py_salt` package.:

```
py_yaml
py_pyzmq
py_jinja2
py_msgpack_python
py_m2crypto
py_crypto
python
```

Installation

To install Salt from the OpenCSW package repository you first need to install pkgutil assuming you don't already have it installed:

On Solaris 10:

```
pkgadd -d http://get.opencsw.org/now
```

On Solaris 9:

```
wget http://mirror.opencsw.org/opencsw/pkgutil.pkg
pkgadd -d pkgutil.pkg all
```

Once pkgutil is installed you'll need to edit it's config file `/etc/opt/csw/pkgutil.conf` to point it at the unstable catalog:

```
- #mirror=http://mirror.opencsw.org/opencsw/testing
+ mirror=http://mirror.opencsw.org/opencsw/unstable
```

OK, time to install salt.

```
# Update the catalog
root> /opt/csw/bin/pkgutil -U
# Install salt
root> /opt/csw/bin/pkgutil -i -y py_salt
```

Minion Configuration

Now that salt is installed you can find it's configuration files in `/etc/opt/csw/salt/`.

You'll want to edit the minion config file to set the name of your salt master server:

```
- #master: salt
+ master: your-salt-server
```

If you would like to use pkgutil as the default package provider for your Solaris minions, you can do so using the `providers` option in the minion config file.

You can now start the salt minion like so:

On Solaris 10:

```
svcadm enable salt-minion
```

On Solaris 9:

```
/etc/init.d/salt-minion start
```

You should now be able to log onto the salt master and check to see if the salt-minion key is awaiting acceptance:

```
salt-key -l un
```

Accept the key:

```
salt-key -a <your-salt-minion>
```

Run a simple test against the minion:

```
salt '<your-salt-minion>' test.ping
```

Troubleshooting

Logs are in `/var/log/salt`

3.2.9 Ubuntu Installation

Add repository

The latest packages for Ubuntu are published in the saltstack PPA. If you have the `add-apt-repository` utility, you can add the repository and import the key in one step:

```
sudo add-apt-repository ppa:saltstack/salt
```

---

add-apt-repository: command not found?

The `add-apt-repository` command is not always present on Ubuntu systems. This can be fixed by installing *python-software-properties*:

```
sudo apt-get install python-software-properties
```

Note that since Ubuntu 12.10 (Raring Ringtail), `add-apt-repository` is found in the *software-properties-common* package, and is part of the base install. Thus, `add-apt-repository` should be able to be used out-of-the-box to add the PPA.

Alternately, manually add the repository and import the PPA key with these commands:

```
echo deb http://ppa.launchpad.net/saltstack/salt/ubuntu `lsb_release -sc` main | sudo tee /etc/apt/sc
wget -q -O- "http://keyserver.ubuntu.com:11371/pks/lookup?op=get&search=0x4759FA960E27C0A6" | sudo ap
```

After adding the repository, update the package management database:

```
sudo apt-get update
```

Install packages

Install the Salt master, minion, or syndic from the repository with the apt-get command. These examples each install one daemon, but more than one package name may be given at a time:

```
sudo apt-get install salt-master
sudo apt-get install salt-minion
```

```
sudo apt-get install salt-syndic
```

Post-installation tasks

Now go to the *Configuring Salt* page.

3.2.10 Windows

Salt has full support for running the Salt Minion on Windows.

There are no plans for the foreseeable future to develop a Salt Master on Windows. For now you must run your Salt Master on a supported operating system to control your Salt Minions on Windows.

Many of the standard Salt modules have been ported to work on Windows and many of the Salt States currently work on Windows, as well.

Windows Installer

A Salt Minion Windows installer can be found here:

Download here

- 0.17.4
- http://docs.saltstack.com/downloads/Salt-Minion-0.17.4-win32-Setup.exe
- http://docs.saltstack.com/downloads/Salt-Minion-0.17.4-AMD64-Setup.exe
- 0.17.2
- http://docs.saltstack.com/downloads/Salt-Minion-0.17.2-win32-Setup.exe
- http://docs.saltstack.com/downloads/Salt-Minion-0.17.2-AMD64-Setup.exe
- 0.17.1.1 - Windows Installer bugfix release
- http://docs.saltstack.com/downloads/Salt-Minion-0.17.1.1-win32-Setup.exe
- http://docs.saltstack.com/downloads/Salt-Minion-0.17.1.1-AMD64-Setup.exe
- 0.17.1
- http://docs.saltstack.com/downloads/Salt-Minion-0.17.1-win32-Setup.exe
- http://docs.saltstack.com/downloads/Salt-Minion-0.17.1-AMD64-Setup.exe
- 0.17.0
- http://docs.saltstack.com/downloads/Salt-Minion-0.17.0-win32-Setup.exe
- http://docs.saltstack.com/downloads/Salt-Minion-0.17.0-AMD64-Setup.exe
- 0.16.3
- http://docs.saltstack.com/downloads/Salt-Minion-0.16.3-win32-Setup.exe
- http://docs.saltstack.com/downloads/Salt-Minion-0.16.3-AMD64-Setup.exe
- 0.16.2

- http://docs.saltstack.com/downloads/Salt-Minion-0.16.2-win32-Setup.exe
- http://docs.saltstack.com/downloads/Salt-Minion-0.16.2-AMD64-Setup.exe
- 0.16.0
- http://docs.saltstack.com/downloads/Salt-Minion-0.16.0-win32-Setup.exe
- http://docs.saltstack.com/downloads/Salt-Minion-0.16.0-AMD64-Setup.exe
- 0.15.3
- http://docs.saltstack.com/downloads/Salt-Minion-0.15.3-win32-Setup.exe
- http://docs.saltstack.com/downloads/Salt-Minion-0.15.3-AMD64-Setup.exe
- 0.14.1
- http://docs.saltstack.com/downloads/Salt-Minion-0.14.1-win32-Setup.exe
- http://docs.saltstack.com/downloads/Salt-Minion-0.14.1-AMD64-Setup.exe
- 0.14.0
- http://docs.saltstack.com/downloads/Salt-Minion-0.14.0-win32-Setup.exe
- http://docs.saltstack.com/downloads/Salt-Minion-0.14.0-AMD64-Setup.exe

---

Note: The executables above will install dependencies that the Salt minion requires.

The 64bit installer has been tested on Windows 7 64bit and Windows Server 2008R2 64bit. The 32bit installer has been tested on Windows 2003 Server 32bit. Please file a bug report on our GitHub repo if issues for other platforms are found.

The installer asks for 2 bits of information; the master hostname and the minion name. The installer will update the minion config with these options and then start the minion.

The *salt-minion* service will appear in the Windows Service Manager and can be started and stopped there or with the command line program *sc* like any other Windows service.

If the minion won't start, try installing the Microsoft Visual C++ 2008 x64 SP1 redistributable. Allow all Windows updates to run salt-minion smoothly.

Silent Installer option

The installer can be run silently by providing the */S* option at the command line. The options */master* and */minion-name* allow for configuring the master hostname and minion name, respectively. Here's an example of using the silent installer:

```
Salt-Minion-0.17.0-Setup-amd64.exe /S /master=yoursaltmaster /minion-name=yourminionname
```

Setting up a Windows build environment

1. Install the Microsoft Visual C++ 2008 SP1 Redistributable, vcredist_x86 or vcredist_x64.
2. Install msysgit
3. Clone the Salt git repository from GitHub

```
git clone git://github.com/saltstack/salt.git
```

4. Install the latest point release of Python 2.7 for the architecture you wish to target
5. Add C:\Python27 and C:\Python27\Scripts to your system path
6. Download and run the Setuptools bootstrap - ez_setup.py

```
python ez_setup.py
```

7. Install Pip

```
easy_install pip
```

8. Install the latest point release of OpenSSL for Windows
    (a) During setup, choose first option to install in Windows system directory
9. Install the latest point release of M2Crypto
    (a) In general, be sure to download installers targeted at py2.7 for your chosen architecture
10. Install the latest point release of pycrypto
11. Install the latest point release of pywin32
12. Install the latest point release of Cython
13. Install the latest point release of jinja2
14. Install the latest point release of msgpack
15. Install psutil

```
easy_install psutil
```

16. Install pyzmq

```
easy_install pyzmq
```

17. Install PyYAML

```
easy_install pyyaml
```

18. Install bbfreeze

```
easy_install bbfreeze
```

19. Install wmi

```
pip install wmi
```

20. Install esky

```
pip install esky
```

21. Install Salt

```
cd salt
python setup.py install
```

22. Build a frozen binary distribution of Salt

```
python setup.py bdist_esky
```

A zip file has been created in the `dist/` folder, containing a frozen copy of Python and the dependency libraries, along with Windows executables for each of the Salt scripts.

Building the installer

The Salt Windows installer is built with the open-source NSIS compiler. The source for the installer is found in the pkg directory of the Salt repo here: https://github.com/saltstack/salt/blob/develop/pkg/windows/installer/Salt-Minion-Setup.nsi. To create the installer, extract the frozen archive from `dist/` into `pkg/windows/buildenv/` and run NSIS.

The NSIS installer can be found here: http://nsis.sourceforge.net/Main_Page

Testing the Salt minion

1. Create the directory C:\salt (if it doesn't exist already)
2. Copy the example `conf` and `var` directories from `pkg/windows/buildenv/` into C:\salt
3. Edit C:\salt\conf\minion

```
master: ipaddress or hostname of your salt-master
```

4. Start the salt-minion

```
cd C:\Python27\Scripts
python salt-minion
```

5. On the salt-master accept the new minion's key

```
sudo salt-key -A
```

```
(This accepts all unaccepted keys. If you're concerned about security just accept the key for this sp
```

6. Test that your minion is responding
   (a) On the salt-master run:

```
sudo salt '*' test.ping
```

You should get the following response: {'your minion hostname': True}

Single command bootstrap script

On a 64 bit Windows host the following script makes an unattended install of salt, including all dependencies:

Not up to date.
This script is not up to date. Please use the installer found above

---

```
"PowerShell (New-Object System.Net.WebClient).DownloadFile('http://csa-net.dk/salt/bootstrap64.bat','
```

```
(All in one line.)
```

You can execute the above command remotely from a Linux host using winexe:

```
winexe -U "administrator" //fqdn "PowerShell (New-Object ......);"
```

For more info check http://csa-net.dk/salt

Packages management under Windows 2003

On windows Server 2003, you need to install optional component "wmi windows installer provider" to have full list of installed packages. If you don't have this, salt-minion can't report some installed softwares.

3.2.11 SUSE Installation

With openSUSE 13.1, Salt 0.16.4 has been available in the primary repositories. The devel:language:python repo will have more up to date versions of salt, all package development will be done there.

Installation

Salt can be installed using `zypper` and is available in the standard openSUSE 13.1 repositories.

Stable Release

Salt is packaged separately for the minion and the master. It is necessary only to install the appropriate package for the role the machine will play. Typically, there will be one master and multiple minions.

```
zypper install salt-master
zypper install salt-minion
```

Post-installation tasks openSUSE

Master

To have the Master start automatically at boot time:

```
systemctl enable salt-master.service
```

To start the Master:

```
systemctl start salt-master.service
```

Minion

To have the Minion start automatically at boot time:

```
systemctl enable salt-minion.service
```

To start the Minion:

```
systemctl start salt-minion.service
```

Post-installation tasks SLES

Master

To have the Master start automatically at boot time:

```
chkconfig salt-master on
```

To start the Master:

```
rcsalt-master start
```

Minion

To have the Minion start automatically at boot time:

```
chkconfig salt-minion on
```

To start the Minion:

```
rcsalt-minion start
```

Unstable Release

openSUSE

For openSUSE Factory run the following as root:

```
zypper addrepo http://download.opensuse.org/repositories/devel:languages:python/openSUSE_Factory/deve
zypper refresh
zypper install salt salt-minion salt-master
```

For openSUSE 13.1 run the following as root:

```
zypper addrepo http://download.opensuse.org/repositories/devel:languages:python/openSUSE_13.1/devel:
zypper refresh
zypper install salt salt-minion salt-master
```

For openSUSE 12.3 run the following as root:

```
zypper addrepo http://download.opensuse.org/repositories/devel:languages:python/openSUSE_12.3/devel:
zypper refresh
zypper install salt salt-minion salt-master
```

For openSUSE 12.2 run the following as root:

```
zypper addrepo http://download.opensuse.org/repositories/devel:languages:python/openSUSE_12.2/devel:
zypper refresh
zypper install salt salt-minion salt-master
```

For openSUSE 12.1 run the following as root:

```
zypper addrepo http://download.opensuse.org/repositories/devel:languages:python/openSUSE_12.1/devel:
zypper refresh
zypper install salt salt-minion salt-master
```

For bleeding edge python Factory run the following as root:

```
zypper addrepo http://download.opensuse.org/repositories/devel:languages:python/bleeding_edge_python_
zypper refresh
zypper install salt salt-minion salt-master
```

Suse Linux Enterprise

For SLE 11 SP3 run the following as root:

```
zypper addrepo http://download.opensuse.org/repositories/devel:languages:python/SLE_11_SP3/devel:lan
zypper refresh
zypper install salt salt-minion salt-master
```

For SLE 11 SP2 run the following as root:

```
zypper addrepo http://download.opensuse.org/repositories/devel:languages:python/SLE_11_SP2/devel:lan
zypper refresh
zypper install salt salt-minion salt-master
```

Now go to the *Configuring Salt* page.

3.3 Dependencies

Salt should run on any Unix-like platform so long as the dependencies are met.

- Python 2.6 >= 2.6 <3.0
- ZeroMQ >= 3.2.0
- pyzmq >= 2.2.0 - ZeroMQ Python bindings
- PyCrypto - The Python cryptography toolkit
- msgpack-python - High-performance message interchange format
- YAML - Python YAML bindings
- Jinja2 - parsing Salt States (configurable in the master settings)

3.4 Optional Dependencies

- mako - an optional parser for Salt States (configurable in the master settings)
- gcc - dynamic Cython module compiling

3.5 Upgrading Salt

When upgrading Salt, the master(s) should always be upgraded first. Backwards compatibility for minions running newer versions of salt than their masters is not guaranteed.

Whenever possible, backwards compatibility between new masters and old minions will be preserved. Generally, the only exception to this policy is in case of a security vulnerability.

CHAPTER 4

Configuring Salt

Salt configuration is very simple. The default configuration for the *master* will work for most installations and the only requirement for setting up a *minion* is to set the location of the master in the minion configuration file.

master The Salt master is the central server that all minions connect to. Commands are run on the minions through the master, and minions send data back to the master (unless otherwise redirected with a *returner*). It is started with the **salt-master** program.

minion Salt minions are the potentially hundreds or thousands of servers that may be queried and controlled from the master.

The configuration files will be installed to `/etc/salt` and are named after the respective components, `/etc/salt/master` and `/etc/salt/minion`.

4.1 Master Configuration

By default the Salt master listens on ports 4505 and 4506 on all interfaces (0.0.0.0). To bind Salt to a specific IP, redefine the “interface” directive in the master configuration file, typically `/etc/salt/master`, as follows:

```
- #interface: 0.0.0.0
+ interface: 10.0.0.1
```

After updating the configuration file, restart the Salt master. See the *master configuration reference* for more details about other configurable options.

4.2 Minion Configuration

Although there are many Salt Minion configuration options, configuring a Salt Minion is very simple. By default a Salt Minion will try to connect to the DNS name “salt”; if the Minion is able to resolve that name correctly, no configuration is needed.

If the DNS name “salt” does not resolve to point to the correct location of the Master, redefine the “master” directive in the minion configuration file, typically `/etc/salt/minion`, as follows:

```
- #master: salt
+ master: 10.0.0.1
```

After updating the configuration file, restart the Salt minion. See the *minion configuration reference* for more details about other configurable options.

4.3 Running Salt

1. Start the master in the foreground (to daemonize the process, pass the *-d flag*):

   ```
   salt-master
   ```

2. Start the minion in the foreground (to daemonize the process, pass the *-d flag*):

   ```
   salt-minion
   ```

Having trouble?

The simplest way to troubleshoot Salt is to run the master and minion in the foreground with *log level* set to `debug`:

```
salt-master --log-level=debug
```

For information on salt's logging system please see the *logging document*.

Run as an unprivileged (non-root) user

To run Salt as another user, set the `user` parameter in the master config file.

Additionally, ownership and permissions need to be set such that the desired user can read from and write to the following directories (and their subdirectories, where applicable):

- /etc/salt
- /var/cache/salt
- /var/log/salt

More information about running salt as a non-privileged user can be found *here*.

There is also a full *troubleshooting guide* available.

4.4 Key Management

Salt uses AES encryption for all communication between the Master and the Minion. This ensures that the commands sent to the Minions cannot be tampered with, and that communication between Master and Minion is authenticated through trusted, accepted keys.

Before commands can be sent to a Minion, its key must be accepted on the Master. Run the `salt-key` command to list the keys known to the Salt Master:

```
[root@master ~]# salt-key -L
Unaccepted Keys:
alpha
bravo
charlie
delta
Accepted Keys:
```

This example shows that the Salt Master is aware of four Minions, but none of the keys has been accepted. To accept the keys and allow the Minions to be controlled by the Master, again use the `salt-key` command:

```
[root@master ~]# salt-key -A
[root@master ~]# salt-key -L
Unaccepted Keys:
Accepted Keys:
alpha
bravo
charlie
delta
```

The `salt-key` command allows for signing keys individually or in bulk. The example above, using `-A` bulk-accepts all pending keys. To accept keys individually use the lowercase of the same option, `-a keyname`.

See also:

salt-key manpage

4.5 Sending Commands

Communication between the Master and a Minion may be verified by running the `test.ping` command:

```
[root@master ~]# salt alpha test.ping
alpha:
    True
```

Communication between the Master and all Minions may be tested in a similar way:

```
[root@master ~]# salt '*' test.ping
alpha:
    True
bravo:
    True
charlie:
    True
delta:
    True
```

Each of the Minions should send a `True` response as shown above.

4.6 What's Next?

Understanding *targeting* is important. From there, depending on the way you wish to use Salt, you should also proceed to learn about *States* and *Execution Modules*.

CHAPTER 5

Developing Salt

There is a great need for contributions to Salt and patches are welcome! The goal here is to make contributions clear, make sure there is a trail for where the code has come from, and most importantly, to give credit where credit is due!

There are a number of ways to contribute to salt development.

5.1 Sending a GitHub pull request

This is the preferred method for contributions. Simply create a GitHub fork, commit changes to the fork, and then open up a pull request.

The following is an example (from Open Comparison Contributing Docs) of an efficient workflow for forking, cloning, branching, committing, and sending a pull request for a GitHub repository.

First, make a local clone of your GitHub fork of the salt GitHub repo and make edits and changes locally.

Then, create a new branch on your clone by entering the following commands:

```
git checkout -b fixed-broken-thing
```

```
Switched to a new branch 'fixed-broken-thing'
```

Choose a name for your branch that describes its purpose.

Now commit your changes to this new branch with the following command:

```
git commit -am 'description of my fixes for the broken thing'
```

Note: Using `git commit -am`, followed by a quoted string, both stages and commits all modified files in a single command. Depending on the nature of your changes, you may wish to stage and commit them separately. Also, note that if you wish to add newly-tracked files as part of your commit, they will not be caught using `git commit -am` and will need to be added using `git add` before committing.

Push your locally-committed changes back up to GitHub:

```
git push --set-upstream origin fixed-broken-thing
```

Now go look at your fork of the salt repo on the GitHub website. The new branch will now be listed under the "Source" tab where it says "Switch Branches". Select the new branch from this list, and then click the "Pull request" button.

Put in a descriptive comment, and include links to any project issues related to the pull request.

The repo managers will be notified of your pull request and it will be reviewed. If a reviewer asks for changes, just make the changes locally in the same local feature branch, push them to GitHub, then add a comment to the discussion section of the pull request.

Note: Travis-CI

To make reviewing pull requests easier for the maintainers, please enable Travis-CI on your fork. Salt is already configured, so simply follow the first 2 steps on the Travis-CI Getting Started Doc.

5.2 Keeping Salt Forks in Sync

Salt is advancing quickly. It is therefore critical to pull upstream changes from master into forks on a regular basis. Nothing is worse than putting in a days of hard work into a pull request only to have it rejected because it has diverged too far from master.

To pull in upstream changes:

```
# For ssh github
git remote add upstream git@github.com:saltstack/salt.git
git fetch upstream

# For https github
git remote add upstream https://github.com/saltstack/salt.git
git fetch upstream
```

To check the log to be sure that you actually want the changes, run the following before merging:

```
git log upstream/develop
```

Then to accept the changes and merge into the current branch:

```
git merge upstream/develop
```

For more info, see GitHub Fork a Repo Guide or Open Comparison Contributing Docs

5.3 Posting patches to the mailing list

Patches will also be accepted by email. Format patches using git format-patch and send them to the Salt users mailing list. The contributor will then get credit for the patch, and the Salt community will have an archive of the patch and a place for discussion.

5.4 Installing Salt for development

Clone the repository using:

```
git clone https://github.com/saltstack/salt
```

Note: tags

Just cloning the repository is enough to work with Salt and make contributions. However, fetching additional tags from git is required to have Salt report the correct version for itself. To do this, first add the git repository as an upstream source:

```
git remote add upstream http://github.com/saltstack/salt
```

Fetching tags is done with the git 'fetch' utility:

```
git fetch --tags upstream
```

Create a new virtualenv:

```
virtualenv /path/to/your/virtualenv
```

On Arch Linux, where Python 3 is the default installation of Python, use the `virtualenv2` command instead of `virtualenv`.

Note: Using system Python modules in the virtualenv

To use already-installed python modules in virtualenv (instead of having pip download and compile new ones), run `virtualenv --system-site-packages` Using this method eliminates the requirement to install the salt dependencies again, although it does assume that the listed modules are all installed in the system PYTHONPATH at the time of virtualenv creation.

Activate the virtualenv:

```
source /path/to/your/virtualenv/bin/activate
```

Install Salt (and dependencies) into the virtualenv:

```
pip install M2Crypto    # Don't install on Debian/Ubuntu (see below)
pip install pyzmq PyYAML pycrypto msgpack-python jinja2 psutil
pip install -e ./salt   # the path to the salt git clone from above
```

Note: Installing M2Crypto

`swig` and `libssl-dev` are required to build M2Crypto. To fix the error `command 'swig' failed with exit status 1` while installing M2Crypto, try installing it with the following command:

```
env SWIG_FEATURES="-cpperraswarn -includeall -D__`uname -m`__ -I/usr/include/openssl" pip install M2C
```

Debian and Ubuntu systems have modified openssl libraries and mandate that a patched version of M2Crypto be installed. This means that M2Crypto needs to be installed via apt:

```
apt-get install python-m2crypto
```

This also means that pulling in the M2Crypto installed using apt requires using `--system-site-packages` when creating the virtualenv.

If you're using a platform other than Debian or Ubuntu, and you are installing M2Crypto via pip instead of a system package, then you will also need the `gcc` compiler.

Note: Installing psutil

Python header files are required to build this module, otherwise the pip install will fail. If your distribution separates binaries and headers into separate packages, make sure that you have the headers installed. In most Linux distributions which split the headers into their own package, this can be done by installing the `python-dev` or `python-devel` package. For other platforms, the package will likely be similarly named.

Note: Installing dependencies on OS X.

You can install needed dependencies on OS X using homebrew or macports. See *OS X Installation*

5.4.1 Running a self-contained development version

During development it is easiest to be able to run the Salt master and minion that are installed in the virtualenv you created above, and also to have all the configuration, log, and cache files contained in the virtualenv as well.

Copy the master and minion config files into your virtualenv:

```
mkdir -p /path/to/your/virtualenv/etc/salt
cp ./salt/conf/master /path/to/your/virtualenv/etc/salt/master
cp ./salt/conf/minion /path/to/your/virtualenv/etc/salt/minion
```

Edit the master config file:

1. Uncomment and change the `user: root` value to your own user.
2. Uncomment and change the `root_dir: /` value to point to `/path/to/your/virtualenv`.
3. If you are running version 0.11.1 or older, uncomment and change the `pidfile: /var/run/salt-master.pid` value to point to `/path/to/your/virtualenv/salt-master.pid`.
4. If you are also running a non-development version of Salt you will have to change the `publish_port` and `ret_port` values as well.

Edit the minion config file:

1. Repeat the edits you made in the master config for the `user` and `root_dir` values as well as any port changes.
2. If you are running version 0.11.1 or older, uncomment and change the `pidfile: /var/run/salt-minion.pid` value to point to `/path/to/your/virtualenv/salt-minion.pid`.
3. Uncomment and change the `master: salt` value to point at `localhost`.
4. Uncomment and change the `id:` value to something descriptive like "saltdev". This isn't strictly necessary but it will serve as a reminder of which Salt installation you are working with.
5. If you changed the `ret_port` value in the master config because you are also running a non-development version of Salt, then you will have to change the `master_port` value in the minion config to match.

Note: Using *salt-call* with a *Standalone Minion*

If you plan to run *salt-call* with this self-contained development environment in a masterless setup, you should invoke *salt-call* with `-c /path/to/your/virtualenv/etc/salt` so that salt can find the minion config file. Without the `-c` option, Salt finds its config files in */etc/salt*.

Start the master and minion, accept the minion's key, and verify your local Salt installation is working:

```
cd /path/to/your/virtualenv
salt-master -c ./etc/salt -d
salt-minion -c ./etc/salt -d
salt-key -c ./etc/salt -L
salt-key -c ./etc/salt -A
salt -c ./etc/salt '*' test.ping
```

Running the master and minion in debug mode can be helpful when developing. To do this, add `-l debug` to the calls to `salt-master` and `salt-minion`. If you would like to log to the console instead of to the log file, remove the `-d`.

Once the minion starts, you may see an error like the following:

```
zmq.core.error.ZMQError: ipc path "/path/to/your/virtualenv/var/run/salt/minion/minion_event_7824dcb
```

This means the the path to the socket the minion is using is too long. This is a system limitation, so the only workaround is to reduce the length of this path. This can be done in a couple different ways:

1. Create your virtualenv in a path that is short enough.
2. Edit the `sock_dir` minion config variable and reduce its length. Remember that this path is relative to the value you set in `root_dir`.

`NOTE`: The socket path is limited to 107 characters on Solaris and Linux, and 103 characters on BSD-based systems.

Note: File descriptor limits

Ensure that the system open file limit is raised to at least 2047:

```
# check your current limit
ulimit -n

# raise the limit. persists only until reboot
# use 'limit descriptors 2047' for c-shell
ulimit -n 2047
```

To set file descriptors on OSX, refer to the *OS X Installation* instructions.

5.5 Installing Salt from the Python Package Index

If you are installing using `easy_install`, you will need to define a **USE_SETUPTOOLS** environment variable, otherwise dependencies will not be installed:

```
USE_SETUPTOOLS=1 easy_install salt
```

5.6 Editing and previewing the documentation

You need `sphinx-build` command to build the docs. In Debian/Ubuntu this is provided in the `python-sphinx` package. Sphinx can also be installed to a virtualenv using pip:

```
pip install Sphinx
```

Change to salt documentation directory, then:

```
cd doc; make html
```

- This will build the HTML docs. Run `make` without any arguments to see the available make targets, which include **html**, **man**, and **text**.
- The docs then are built within the **docs/_build/** folder. To update the docs after making changes, run `make` again.
- The docs use reStructuredText for markup. See a live demo at http://rst.ninjs.org/.
- The help information on each module or state is culled from the python code that runs for that piece. Find them in `salt/modules/` or `salt/states/`.

- To build the docs on Arch Linux, the **python2-sphinx** package is required. Additionally, it is necessary to tell **make** where to find the proper **sphinx-build** binary, like so:

```
make SPHINXBUILD=sphinx-build2 html
```

- To build the docs on RHEL/CentOS 6, the **python-sphinx10** package must be installed from EPEL, and the following make command must be used:

```
make SPHINXBUILD=sphinx-1.0-build html
```

Once you've updated the documentation, you can run the following command to launch a simple Python HTTP server to see your changes:

```
cd _build/html; python -m SimpleHTTPServer
```

5.7 Running unit and integration tests

Run the test suite with following command:

```
./setup.py test
```

See *here* for more information regarding the test suite.

CHAPTER 6

Targeting

Targeting Specifying which minions should run a command or execute a state by matching against hostnames, or system information, or defined groups, or even combinations thereof.

For example the command `salt web1 apache.signal restart` to restart the Apache httpd server specifies the machine `web1` as the target and the command will only be run on that one minion.

Similarly when using States, the following *top file* specifies that only the `web1` minion should execute the contents of `webserver.sls`:

```
base:
  'web1':
    - webserver
```

There are many ways to target individual minions or groups of minions in Salt:

6.1 Matching the `minion id`

minion id A unique identifier for a given minion. By default the minion id is the FQDN of that host but this can be overridden.

Each minion needs a unique identifier. By default when a minion starts for the first time it chooses its FQDN (fully qualified domain name) as that identifier. The minion id can be overridden via the minion's `id` configuration setting.

Tip: minion id and minion keys

The *minion id* is used to generate the minion's public/private keys and if it ever changes the master must then accept the new key as though the minion was a new host.

6.1.1 Globbing

The default matching that Salt utilizes is `shell-style globbing` around the *minion id*. This also works for states in the *top file*.

Note: You must wrap **salt** calls that use globbing in single-quotes to prevent the shell from expanding the globs before Salt is invoked.

Match all minions:

```
salt '*' test.ping
```

Match all minions in the example.net domain or any of the example domains:

```
salt '*.example.net' test.ping
salt '*.example.*' test.ping
```

Match all the `webN` minions in the example.net domain (`web1.example.net`, `web2.example.net` ... `webN.example.net`):

```
salt 'web?.example.net' test.ping
```

Match the `web1` through `web5` minions:

```
salt 'web[1-5]' test.ping
```

Match the `web1` and `web3` minions:

```
salt 'web[1,3]' test.ping
```

Match the `web-x`, `web-y`, and `web-z` minions:

```
salt 'web-[x-z]' test.ping
```

Note: For additional targeting methods please review the *compound matchers* documentation.

6.1.2 Regular Expressions

Minions can be matched using Perl-compatible `regular expressions` (which is globbing on steroids and a ton of caffeine).

Match both `web1-prod` and `web1-devel` minions:

```
salt -E 'web1-(prod|devel)' test.ping
```

When using regular expressions in a State's *top file*, you must specify the matcher as the first option. The following example executes the contents of `webserver.sls` on the above-mentioned minions.

```
base:
  'web1-(prod|devel)':
  - match: pcre
  - webserver
```

6.1.3 Lists

At the most basic level, you can specify a flat list of minion IDs:

```
salt -L 'web1,web2,web3' test.ping
```

6.2 Grains

Salt comes with an interface to derive information about the underlying system. This is called the grains interface, because it presents salt with grains of information.

Grains Static bits of information that a minion collects about the system when the minion first starts.

The grains interface is made available to Salt modules and components so that the right salt minion commands are automatically available on the right systems.

It is important to remember that grains are bits of information loaded when the salt minion starts, so this information is static. This means that the information in grains is unchanging, therefore the nature of the data is static. So grains information are things like the running kernel, or the operating system.

Match all CentOS minions:

```
salt -G 'os:CentOS' test.ping
```

Match all minions with 64-bit CPUs, and return number of CPU cores for each matching minion:

```
salt -G 'cpuarch:x86_64' grains.item num_cpus
```

Additionally, globs can be used in grain matches, and grains that are nested in a *dictionary* can be matched by adding a colon for each level that is traversed. For example, the following will match hosts that have a grain called `ec2_tags`, which itself is a *dict* with a key named `environment`, which has a value that contains the word `production`:

```
salt -G 'ec2_tags:environment:*production*'
```

6.2.1 Listing Grains

Available grains can be listed by using the 'grains.ls' module:

```
salt '*' grains.ls
```

Grains data can be listed by using the 'grains.items' module:

```
salt '*' grains.items
```

6.2.2 Grains in the Minion Config

Grains can also be statically assigned within the minion configuration file. Just add the option `grains` and pass options to it:

```
grains:
  roles:
    - webserver
    - memcache
  deployment: datacenter4
  cabinet: 13
  cab_u: 14-15
```

Then status data specific to your servers can be retrieved via Salt, or used inside of the State system for matching. It also makes targeting, in the case of the example above, simply based on specific data about your deployment.

6.2.3 Grains in /etc/salt/grains

If you do not want to place your custom static grains in the minion config file, you can also put them in `/etc/salt/grains` on the minion. They are configured in the same way as in the above example, only without a top-level `grains:` key:

```
roles:
  - webserver
  - memcache
deployment: datacenter4
cabinet: 13
cab_u: 14-15
```

6.2.4 Matching Grains in the Top File

With correctly setup grains on the Minion, the Top file used in Pillar or during Highstate can be made really efficient. Like for example, you could do:

```
'node_type:web':
    - match: grain
    - webserver

'node_type:postgres':
    - match: grain
    - database

'node_type:redis':
    - match: grain
    - redis

'node_type:lb':
    - match: grain
    - lb
```

For this example to work, you would need the grain `node_type` and the correct value to match on. This simple example is nice, but too much of the code is similar. To go one step further, we can place some Jinja template code into the Top file.

```
{% set self = grains['node_type'] %}

    'node_type:{{ self }}':
        - match: grain
        - {{ self }}
```

The Jinja code simplified the Top file, and allowed SaltStack to work its magic.

6.2.5 Writing Grains

Grains are easy to write. The grains interface is derived by executing all of the "public" functions found in the modules located in the grains package or the custom grains directory. The functions in the modules of the grains must return a Python *dict*, where the keys in the *dict* are the names of the grains and the values are the values.

Custom grains should be placed in a `_grains` directory located under the `file_roots` specified by the master config file. They will be distributed to the minions when `state.highstate` is run, or by executing the `saltutil.sync_grains` or `saltutil.sync_all` functions.

Before adding a grain to Salt, consider what the grain is and remember that grains need to be static data. If the data is something that is likely to change, consider using *Pillar* instead.

6.2.6 Precedence

Core grains can be overriden by custom grains. As there are several ways of defining custom grains, there is an order of precedence which should be kept in mind when defining them. The order of evaluation is as follows:

1. Core grains.
2. Custom grains in `/etc/salt/grains`.
3. Custom grains in `/etc/salt/minion`.
4. Custom grain modules in `_grains` directory, synced to minions.

Each successive evaluation overrides the previous ones, so any grains defined in `/etc/salt/grains` that have the same name as a core grain will override that core grain. Similarly, `/etc/salt/minion` overrides both core grains and grains set in `/etc/salt/grains`, and custom grain modules will override *any* grains of the same name.

6.2.7 Examples of Grains

The core module in the grains package is where the main grains are loaded by the Salt minion and provides the principal example of how to write grains:

https://github.com/saltstack/salt/blob/develop/salt/grains/core.py

6.2.8 Syncing Grains

Syncing grains can be done a number of ways, they are automatically synced when `state.highstate` is called, or (as noted above) the grains can be manually synced and reloaded by calling the `saltutil.sync_grains` or `saltutil.sync_all` functions.

6.3 Node groups

Node group A predefined group of minions declared in the master configuration file `nodegroups` setting as a compound target.

Nodegroups are declared using a compound target specification. The compound target documentation can be found *here*.

The `nodegroups` master config file parameter is used to define nodegroups. Here's an example nodegroup configuration within `/etc/salt/master`:

```
nodegroups:
  group1: 'L@foo.domain.com,bar.domain.com,baz.domain.com or bl*.domain.com'
  group2: 'G@os:Debian and foo.domain.com'
```

Note: The 'L' within group1 is matching a list of minions, while the 'G' in group2 is matching specific grains. See the *compound matchers* documentation for more details.

To match a nodegroup on the CLI, use the `-N` command-line option:

```
salt -N group1 test.ping
```

To match in your *top file*, make sure to put `- match:  nodegroup` on the line directly following the nodegroup name.

```
base:
  group1:
    - match: nodegroup
    - webserver
```

6.4 Compound matchers

Compound matcher A combination of many target definitions that can be combined with boolean operators.

Compound matchers allow very granular minion targeting using any of Salt's matchers. The default matcher is a `glob` match, just as with CLI and *top file* matching. To match using anything other than a glob, prefix the match string with the appropriate letter from the table below, followed by an `@` sign.

Letter	Match Type	Example
G	Grains glob	`G@os:Ubuntu`
E	PCRE Minion ID	`E@web\d+\.(dev|qa|prod)\.loc`
P	Grains PCRE	`P@os:(RedHat|Fedora|CentOS)`
L	List of minions	`L@minion1.example.com,minion3.domain.com or bl*.domain.com`
I	Pillar glob	`I@pdata:foobar`
S	Subnet/IP address	`S@192.168.1.0/24` or `S@192.168.1.100`
R	Range cluster	`R@%foo.bar`

Matchers can be joined using boolean `and`, `or`, and `not` operators.

For example, the following string matches all Debian minions with a hostname that begins with `webserv`, as well as any minions that have a hostname which matches the `regular expression` `web-dc1-srv.*`:

```
salt -C 'webserv* and G@os:Debian or E@web-dc1-srv.*' test.ping
```

That same example expressed in a *top file* looks like the following:

```
base:
  'webserv* and G@os:Debian or E@web-dc1-srv.*':
    - match: compound
    - webserver
```

Note that a leading `not` is not supported in compound matches. Instead, something like the following must be done:

```
salt -C '* and not G@kernel:Darwin' test.ping
```

Excluding a minion based on its ID is also possible:

```
salt -C '* and not web-dc1-srv' test.ping
```

6.5 Batch Size

The `-b` (or `--batch-size`) option allows commands to be executed on only a specified number of minions at a time. Both percentages and finite numbers are supported.

```
salt '*' -b 10 test.ping
```

```
salt -G 'os:RedHat' --batch-size 25% apache.signal restart
```

This will only run test.ping on 10 of the targeted minions at a time and then restart apache on 25% of the minions matching `os:RedHat` at a time and work through them all until the task is complete. This makes jobs like rolling web server restarts behind a load balancer or doing maintenance on BSD firewalls using carp much easier with salt.

The batch system maintains a window of running minions, so, if there are a total of 150 minions targeted and the batch size is 10, then the command is sent to 10 minions, when one minion returns then the command is sent to one additional minion, so that the job is constantly running on 10 minions.

CHAPTER 7

Salt tutorials

7.1 Introduction

7.1.1 Salt Masterless Quickstart

Running a masterless salt-minion lets you use salt's configuration management for a single machine. It is also useful for testing out state trees before deploying to a production setup.

The only real difference in using a standalone minion is that instead of issuing commands with `salt`, we use the `salt-call` command, like this:

```
salt-call --local state.highstate
```

Bootstrap Salt Minion

First we need to install the salt minion. The salt-bootstrap script makes this incredibly easy for any OS with a Bourne shell. You can use it like this:

```
wget -O - http://bootstrap.saltstack.org | sudo sh
```

Or see the salt-bootstrap documentation for other one liners. Additionally, if you are using Vagrant to test out salt, the salty-vagrant tool will provision the VM for you.

Create State Tree

Now we build an example state tree. This is where the configuration is defined. For more in depth directions, see the tutorial.

1. Create the top.sls file:

/srv/salt/top.sls:

```
base:
  '*':
    - webserver
```

2. Create our webserver state tree:

/srv/salt/webserver.sls:

```
apache:                 # ID declaration
  pkg:                  # state declaration
    - installed         # function declaration
```

The only thing left is to provision our minion using the highstate command. Salt-call also gives us an easy way to give us verbose output:

```
salt-call --local state.highstate -l debug
```

The `--local` flag tells the salt-minion to look for the state tree in the local file system. Normally the minion copies the state tree from the master and executes it from there.

That's it, good luck!

7.2 Basics

7.2.1 Salt Bootstrap

The Salt Bootstrap script allows for a user to install the Salt Minion or Master on a variety of system distributions and versions. This shell script known as `bootstrap-salt.sh` runs through a series of checks to determine the operating system type and version. It then installs the Salt binaries using the appropriate methods. The Salt Bootstrap script installs the minimum number of packages required to run Salt. This means that in the event you run the bootstrap to install via package, Git will not be installed. Installing the minimum number of packages helps ensure the script stays as lightweight as possible, assuming the user will install any other required packages after the Salt binaries are present on the system. The script source is available on GitHub: https://github.com/saltstack/salt-bootstrap

Supported Operating Systems

- Amazon Linux 2012.09
- Arch
- CentOS 5/6
- Debian 6.x/7.x
- Fedora 17/18
- FreeBSD 9.1
- Gentoo
- Linaro
- Linux Mint 13/14
- OpenSUSE 12.x
- Red Hat 5/6
- Red Hat Enterprise 5/6
- SmartOS
- SuSE 11 SP1/11 SP2
- Ubuntu 10.x/11.x/12.x/13.04

Note: In the event you do not see your distribution or version available please review the develop branch on Github as it main contain updates that are not present in the stable release: https://github.com/saltstack/salt-bootstrap/tree/develop

Example Usage

The Salt Bootstrap script has a wide variety of options that can be passed as well as several ways of obtaining the bootstrap script itself.

For example, using `curl` to install latest git:

```
curl -L http://bootstrap.saltstack.org | sudo sh -s -- git develop
```

If you have certificate issues using `curl`, try the following:

```
curl --insecure -L http://bootstrap.saltstack.org | sudo sh -s -- git develop
```

Using `wget` to install your distribution's stable packages:

```
wget -O - http://bootstrap.saltstack.org | sudo sh
```

If you have certificate issues using `wget` try the following:

```
wget --no-check-certificate -O - http://bootstrap.saltstack.org | sudo sh
```

Install a specific version from git using `wget`:

```
wget -O - http://bootstrap.saltstack.org | sh -s -- -P git v0.16.4
```

If you already have python installed, `python 2.6`, then it's as easy as:

```
python -m urllib "http://bootstrap.saltstack.org" | sudo sh -s -- git develop
```

All python versions should support the following one liner:

```
python -c 'import urllib; print urllib.urlopen("http://bootstrap.saltstack.org").read()' | \
sudo  sh -s -- git develop
```

On a FreeBSD base system you usually don't have either of the above binaries available. You **do** have `fetch` available though:

```
fetch -o - http://bootstrap.saltstack.org | sudo sh
```

If all you want is to install a `salt-master` using latest git:

```
curl -L http://bootstrap.saltstack.org | sudo sh -s -- -M -N git develop
```

If you want to install a specific release version (based on the git tags):

```
curl -L http://bootstrap.saltstack.org | sudo sh -s -- git v0.16.4
```

Downloading the develop branch (from here standard command line options may be passed):

```
wget https://raw.github.com/saltstack/salt-bootstrap/develop/bootstrap-salt.sh
```

Command Line Options

-h Display the help message and command line options.

-v Display script version.

-n No colours.

-D Show debug output.

-c Temporary configuration directory.

-k Temporary directory holding the minion keys which will pre-seed the master.

-M Also install salt-master.

-S Also install salt-syndic.

-N Do not install salt-minion.

-X Do not start daemons after installation.

-C Only run the configuration function. This option automatically bypasses any installation.

-P Allow pip based installations. On some distributions the required salt packages or its dependencies are not available as a package for that distribution. Using this flag allows the script to use pip as a last resort method.

Note: This works for functions which actually implement pip based installations.

-F Allow copied files to overwrite existing(config, init.d, etc).

-U If set, fully upgrade the system prior to bootstrapping salt.

-K If set, keep the temporary files in the temporary directories specified with -c and -k.

7.2.2 Standalone Minion

Since the Salt minion contains such extensive functionality it can be useful to run it standalone. A standalone minion can be used to do a number of things:

- Stand up a master server via States (Salting a Salt Master)
- Use salt-call commands on a system without connectivity to a master
- Masterless States, run states entirely from files local to the minion

Telling Salt Call to Run Masterless

The salt-call command is used to run module functions locally on a minion instead of executing them from the master. Normally the salt-call command checks into the master to retrieve file server and pillar data, but when running standalone salt-call needs to be instructed to not check the master for this data. To instruct the minion to not look for a master when running salt-call the `file_client` configuration option needs to be set. By default the `file_client` is set to `remote` so that the minion knows that file server and pillar data are to be gathered from the master. When setting the `file_client` option to `local` the minion is configured to not gather this data from the master.

```
file_client: local
```

Now the salt-call command will not look for a master and will assume that the local system has all of the file ad pillar resources.

Running States Masterless

The state system can be easily run without a Salt master, with all needed files local to the minion. To do this the minion configuration file needs to be set up to know how to return file_roots information like the master. The file_roots setting defaults to /srv/salt for the base environment just like on the master:

```
file_roots:
  base:
    - /srv/salt
```

Now set up the Salt State Tree, top file, and SLS modules in the same way that they would be set up on a master. Now, with the `file_client` option set to `local` and an available state tree then calls to functions in the state module will use the information in the file_roots on the minion instead of checking in with the master.

Remember that when creating a state tree on a minion there are no syntax or path changes needed, SLS modules written to be used from a master do not need to be modified in any way to work with a minion.

This makes it easy to "script" deployments with Salt states without having to set up a master, and allows for these SLS modules to be easily moved into a Salt master as the deployment grows.

Now the declared state can now be executed with:

```
salt-call state.highstate
```

Or the salt-call command can be executed with the `--local` flag, this makes it unnecessary to change the configuration file:

```
salt-call state.highstate --local
```

7.2.3 Opening the Firewall up for Salt

The Salt master communicates with the minions using an AES-encrypted ZeroMQ connection. These communications are done over TCP ports 4505 and 4506, which need to be accessible on the master only. This document outlines suggested firewall rules for allowing these incoming connections to the master.

Note: No firewall configuration needs to be done on Salt minions. These changes refer to the master only.

RHEL 6 / CentOS 6

The `lokkit` command packaged with some Linux distributions makes opening iptables firewall ports very simple via the command line. Just be careful to not lock out access to the server by neglecting to open the ssh port.

lokkit example:

```
lokkit -p 22:tcp -p 4505:tcp -p 4506:tcp
```

The `system-config-firewall-tui` command provides a text-based interface to modifying the firewall.

system-config-firewall-tui:

```
system-config-firewall-tui
```

openSUSE

Salt installs firewall rules in /etc/sysconfig/SuSEfirewall2.d/services/salt. Enable with:

```
SuSEfirewall2 open
SuSEfirewall2 start
```

If you have an older package of Salt where the above configuration file is not included, the `SuSEfirewall2` command makes opening iptables firewall ports very simple via the command line.

SuSEfirewall example:

```
SuSEfirewall2 open EXT TCP 4505
SuSEfirewall2 open EXT TCP 4506
```

The firewall module in YaST2 provides a text-based interface to modifying the firewall.

YaST2:

```
yast2 firewall
```

iptables

Different Linux distributions store their *iptables* (also known as netfilter) rules in different places, which makes it difficult to standardize firewall documentation. Included are some of the more common locations, but your mileage may vary.

Fedora / RHEL / CentOS:

```
/etc/sysconfig/iptables
```

Arch Linux:

```
/etc/iptables/iptables.rules
```

Debian

Follow these instructions: http://wiki.debian.org/iptables

Once you've found your firewall rules, you'll need to add the two lines below to allow traffic on `tcp/4505` and `tcp/4506`:

```
-A INPUT -m state --state new -m tcp -p tcp --dport 4505 -j ACCEPT
-A INPUT -m state --state new -m tcp -p tcp --dport 4506 -j ACCEPT
```

Ubuntu

Salt installs firewall rules in /etc/ufw/applications.d/salt.ufw. Enable with:

```
ufw allow salt
```

pf.conf

The BSD-family of operating systems uses packet filter (pf). The following example describes the additions to `pf.conf` needed to access the Salt master.

```
pass in on $int_if proto tcp from any to $int_if port 4505
pass in on $int_if proto tcp from any to $int_if port 4506
```

Once these additions have been made to the `pf.conf` the rules will need to be reloaded. This can be done using the `pfctl` command.

```
pfctl -vf /etc/pf.conf
```

7.2.4 Whitelist communication to Master

There are situations where you want to selectively allow Minon traffic from specific hosts or networks into your Salt Master. The first scenario which comes to mind is to prevent unwanted traffic to your Master out of security concerns, but another scenario is to handle Minion upgrades when there are backwards incompatible changes between the installed Salt versions in your environment.

Here is an example *Linux iptables* ruleset to be set on the Master:

```
# Allow Minions from these networks
-I INPUT -s 10.1.2.0/24 -p tcp -m multiport --dports 4505,4506 -j ACCEPT
-I INPUT -s 10.1.3.0/24 -p tcp -m multiport --dports 4505,4506 -j ACCEPT
# Allow Salt to communicate with Master on the loopback interface
-A INPUT -i lo -p tcp -m multiport --dports 4505,4506 -j ACCEPT
# Reject everything else
-A INPUT -p tcp -m multiport --dports 4505,4506 -j REJECT
```

Note: The important thing to note here is that the `salt` command needs to communicate with the listening network socket of `salt-master` on the *loopback* interface. Without this you will see no outgoing Salt traffic from the master, even for a simple `salt '*' test.ping`, because the `salt` client never reached the `salt-master` to tell it to carry out the execution.

7.2.5 Using cron with Salt

The Salt Minion can initiate its own highstate using the `salt-call` command.

```
$ salt-call state.highstate
```

This will cause the minion to check in with the master and ensure it is in the correct 'state'.

Use cron to initiate a highstate

If you would like the Salt Minion to regularly check in with the master you can use the venerable cron to run the `salt-call` command.

```
# PATH=/bin:/sbin:/usr/bin:/usr/sbin

00 00 * * * salt-call state.highstate
```

The above cron entry will run a highstate every day at midnight.

Note: Be aware that you may need to ensure the PATH for cron includes any scripts or commands that need to be executed.

7.2.6 Remote execution tutorial

Before continuing make sure you have a working Salt installation by following the *installation* and the *configuration* instructions.

Stuck?

There are many ways to *get help from the Salt community* including our mailing list and our IRC channel #salt.

Order your minions around

Now that you have a *master* and at least one *minion* communicating with each other you can perform commands on the minion via the **salt** command. Salt calls are comprised of three main components:

```
salt '<target>' <function> [arguments]
```

See also:

salt manpage

target

The target component allows you to filter which minions should run the following function. The default filter is a glob on the minion id. For example:

```
salt '*' test.ping
salt '*.example.org' test.ping
```

Targets can be based on minion system information using the Grains system:

```
salt -G 'os:Ubuntu' test.ping
```

See also:

Grains system

Targets can be filtered by regular expression:

```
salt -E 'virtmach[0-9]' test.ping
```

Targets can be explicitly specified in a list:

```
salt -L 'foo,bar,baz,quo' test.ping
```

Or Multiple target types can be combined in one command:

```
salt -C 'G@os:Ubuntu and webser* or E@database.*' test.ping
```

function

A function is some functionality provided by a module. Salt ships with a large collection of available functions. List all available functions on your minions:

```
salt '*' sys.doc
```

Here are some examples:

Show all currently available minions:

```
salt '*' test.ping
```

Run an arbitrary shell command:

```
salt '*' cmd.run 'uname -a'
```

See also:

the full list of modules

arguments

Space-delimited arguments to the function:

```
salt '*' cmd.exec_code python 'import sys; print sys.version'
```

Optional, keyword arguments are also supported:

```
salt '*' pip.install salt timeout=5 upgrade=True
```

They are always in the form of `kwarg=argument`.

7.2.7 Pillar Walkthrough

Note: This walkthrough assumes that the reader has already completed the initial Salt *walkthrough*.

The pillar interface inside of Salt is one of the most important components of a Salt deployment. Pillar is the interface used to generate arbitrary data for specific minions. The data generated in pillar is made available to almost every component of Salt and is used for a number of purposes:

Highly Sensitive Data: Information transferred via pillar is guaranteed to only be presented to the minions that are targeted, this makes pillar the engine to use in Salt for managing security information, such as cryptographic keys and passwords.

Minion Configuration: Minion modules such as the execution modules, states, and returners can often be configured via data stored in pillar.

Variables: Variables which need to be assigned to specific minions or groups of minions can be defined in pillar and then accessed inside sls formulas and template files.

Arbitrary Data: Pillar can contain any basic data structure, so a list of values, or a key/value store can be defined making it easy to iterate over a group of values in sls formulas

Pillar is therefore one of the most important systems when using Salt, this walkthrough is designed to get a simple pillar up and running in a few minutes and then to dive into the capabilities of pillar and where the data is available.

Setting Up Pillar

The pillar is already running in Salt by default. The data in the minion's pillars can be seen via the following command:

```
salt '*' pillar.items
```

Note: Prior to version 0.16.2, this function is named `pillar.data`. This function name is still supported for backwards compatibility.

By default the contents of the master configuration file are loaded into pillar for all minions, this is to enable the master configuration file to be used for global configuration of minions.

The pillar is built in a similar fashion as the state tree, it is comprised of sls files and has a top file, just like the state tree. The pillar is stored in a different location on the Salt master than the state tree. The default location for the pillar is in /srv/pillar.

Note: The pillar location can be configured via the *pillar_roots* option inside the master configuration file.

To start setting up the pillar, the /srv/pillar directory needs to be present:

```
mkdir /srv/pillar
```

Now a simple top file, following the same format as the top file used for states needs to be created:

`/srv/pillar/top.sls`:

```
base:
  '*':
    - data
```

This top file associates the data.sls file to all minions. Now the `/srv/pillar/data.sls` file needs to be populated:

`/srv/pillar/data.sls`:

```
info: some data
```

Now that the file has been saved the minions' pillars will be updated:

```
salt '*' pillar.items
```

The key `info` should now appear in the returned pillar data.

More Complex Data

Pillar files are sls files, just like states, but unlike states they do not need to define **formulas**, the data can be arbitrary, this example for instance sets up user data with a UID:

`/srv/pillar/users/init.sls`:

```
users:
  thatch: 1000
  shouse: 1001
  utahdave: 1002
  redbeard: 1003
```

Note: The same directory lookups that exist in states exist in pillar, so the file `users/init.sls` can be referenced with `users` in the *top file*.

The top file will need to be updated to include this sls file:

`/srv/pillar/top.sls`:

```
base:
  '*':
    - data
    - users
```

Now the data will be available to the minions. To use the pillar data in a state just access the pillar via Jinja:

`/srv/salt/users/init.sls`

```
{% for user, uid in pillar.get('users', {}).items() %}
{{user}}:
  user.present:
    - uid: {{uid}}
{% endfor %}
```

This approach allows for users to be safely defined in a pillar and then the user data is applied in an sls file.

Paramaterizing States With Pillar

One of the most powerful abstractions in pillar is the ability to parameterize states. Instead of defining macros or functions within the state context the entire state tree can be freely parameterized relative to the minion's pillar.

This approach allows for Salt to be very flexible while staying very straightforward. It also means that simple sls formulas used in the state tree can be directly parameterized without needing to refactor the state tree.

A simple example is to set up a mapping of package names in pillar for separate Linux distributions:

`/srv/pillar/pkg/init.sls`:

```
pkgs:
  {% if grains['os_family'] == 'RedHat' %}
  apache: httpd
  vim: vim-enhanced
  {% elif grains['os_family'] == 'Debian' %}
  apache: apache2
  vim: vim
  {% elif grains['os'] == 'Arch' %}
  apache: apache
  vim: vim
  {% endif %}
```

The new `pkg` sls needs to be added to the top file:

`/srv/pillar/top.sls`:

```
base:
  '*':
    - data
    - users
    - pkg
```

Now the minions will auto map values based on respective operating systems inside of the pillar, so sls files can be safely parameterized:

`/srv/salt/apache/init.sls`:

```
apache:
  pkg.installed:
    - name: {{ pillar['pkgs']['apache'] }}
```

Or, if no pillar is available a default can be set as well:

Note: The function `pillar.get` used in this example was added to Salt in version 0.14.0

`/srv/salt/apache/init.sls`:

```
apache:
  pkg.installed:
    - name: {{ salt['pillar.get']('pkgs:apache', 'httpd') }}
```

In the above example, if the pillar value `pillar['pkgs']['apache']` is not set in the minion's pillar, then the default of `httpd` will be used.

Note: Under the hood, pillar is just a python dict, so python dict methods such as *get* and *items* can be used.

Pillar Makes Simple States Grow Easily

One of the design goals of pillar is to make simple sls formulas easily grow into more flexible formulas without refactoring or complicating the states.

A simple formula:

`/srv/salt/edit/vim.sls`:

```
vim:
  pkg:
    - installed

/etc/vimrc:
  file.managed:
    - source: salt://edit/vimrc
    - mode: 644
    - user: root
    - group: root
    - require:
      - pkg: vim
```

Can be easily transformed into a powerful, parameterized formula:

`/srv/salt/edit/vim.sls`:

```
vim:
  pkg:
    - installed
    - name: {{ pillar['pkgs']['vim'] }}

/etc/vimrc:
  file.managed:
    - source: {{ pillar['vimrc'] }}
    - mode: 644
    - user: root
    - group: root
    - require:
      - pkg: vim
```

Where the vimrc source location can now be changed via pillar:

`/srv/pillar/edit/vim.sls`:

```
{% if grains['id'].startswith('dev') %}
vimrc: salt://edit/dev_vimrc
{% elif grains['id'].startswith('qa') %}
vimrc: salt://edit/qa_vimrc
{% else %}
vimrc: salt://edit/vimrc
{% endif %}
```

Ensuring that the right vimrc is sent out to the correct minions.

More On Pillar

The pillar data is generated on the Salt master and securely distributed to minions. Salt is not restricted to the pillar sls files when defining the pillar but can retrieve data from external sources. This can be useful when information about an infrastructure is stored in a separate location.

Reference information on pillar and the external pillar interface can be found in the Salt documentation:

Pillar

7.3 States

7.3.1 How Do I Use Salt States?

Simplicity, Simplicity, Simplicity

Many of the most powerful and useful engineering solutions are founded on simple principles. Salt States strive to do just that: K.I.S.S. (Keep It Stupidly Simple)

The core of the Salt State system is the SLS, or **S**a**L**t **S**tate file. The SLS is a representation of the state in which a system should be in, and is set up to contain this data in a simple format. This is often called configuration management.

Note: This is just the beginning of using states, make sure to read up on pillar *Pillar* next.

It is All Just Data

Before delving into the particulars, it will help to understand that the SLS file is just a data structure under the hood. While understanding that the SLS is just a data structure isn't critical for understanding and making use of Salt States, it should help bolster knowledge of where the real power is.

SLS files are therefore, in reality, just *dictionaries*, *lists*, *strings*, and *numbers*. By using this approach Salt can be much more flexible. As one writes more state files, it becomes clearer exactly what is being written. The result is a system that is easy to understand, yet grows with the needs of the admin or developer.

The Top File

The example SLS files in the below sections can be assigned to hosts using a file called **top.sls**. This file is described in-depth *here*.

Default Data - YAML

By default Salt represents the SLS data in what is one of the simplest serialization formats available - YAML.

A typical SLS file will often look like this in YAML:

Note: These demos use some generic service and package names, different distributions often use different names for packages and services. For instance *apache* should be replaced with *httpd* on a Red Hat system. Salt uses the name of the init script, systemd name, upstart name etc. based on what the underlying service management for the platform. To get a list of the available service names on a platform execute the service.get_all salt function.

Information on how to make states work with multiple distributions is later in the tutorial.

```
apache:
  pkg:
    - installed
  service:
    - running
    - require:
      - pkg: apache
```

This SLS data will ensure that the package named apache is installed, and that the apache service is running. The components can be explained in a simple way.

The first line is the ID for a set of data, and it is called the ID Declaration. This ID sets the name of the thing that needs to be manipulated.

The second and fourth lines are the start of the State Declarations, so they are using the pkg and service states respectively. The pkg state manages a software package to be installed via the system's native package manager, and the service state manages a system daemon.

The third and fifth lines are the function to run. This function defines what state the named package and service should be in. Here, the package is to be installed, and the service should be running.

Finally, on line six, is the word `require`. This is called a Requisite Statement, and it makes sure that the Apache service is only started after a successful installation of the apache package.

Adding Configs and Users

When setting up a service like an Apache web server, many more components may need to be added. The Apache configuration file will most likely be managed, and a user and group may need to be set up.

```
apache:
  pkg:
    - installed
  service:
    - running
    - watch:
      - pkg: apache
      - file: /etc/httpd/conf/httpd.conf
      - user: apache
  user.present:
    - uid: 87
    - gid: 87
    - home: /var/www/html
    - shell: /bin/nologin
    - require:
      - group: apache
  group.present:
    - gid: 87
    - require:
      - pkg: apache

/etc/httpd/conf/httpd.conf:
  file.managed:
    - source: salt://apache/httpd.conf
    - user: root
    - group: root
    - mode: 644
```

This SLS data greatly extends the first example, and includes a config file, a user, a group and new requisite statement: `watch`.

Adding more states is easy, since the new user and group states are under the Apache ID, the user and group will be the Apache user and group. The `require` statements will make sure that the user will only be made after the group, and that the group will be made only after the Apache package is installed.

Next,the `require` statement under service was changed to watch, and is now watching 3 states instead of just one. The watch statement does the same thing as require, making sure that the other states run before running the state with a watch, but it adds an extra component. The `watch` statement will run the state's watcher function for any changes to the watched states. So if the package was updated, the config file changed, or the user uid modified, then the service state's watcher will be run. The service state's watcher just restarts the service, so in this case, a change in the config file will also trigger a restart of the respective service.

Moving Beyond a Single SLS

When setting up Salt States in a scalable manner, more than one SLS will need to be used. The above examples were in a single SLS file, but two or more SLS files can be combined to build out a State Tree. The above example also references a file with a strange source - `salt://apache/httpd.conf`. That file will need to be available as well.

The SLS files are laid out in a directory structure on the Salt master; an SLS is just a file and files to download are just files.

The Apache example would be laid out in the root of the Salt file server like this:

```
apache/init.sls
apache/httpd.conf
```

So the httpd.conf is just a file in the apache directory, and is referenced directly.

But when using more than one single SLS file, more components can be added to the toolkit. Consider this SSH example:

`ssh/init.sls:`

```
openssh-client:
  pkg.installed

/etc/ssh/ssh_config:
  file.managed:
    - user: root
    - group: root
    - mode: 644
    - source: salt://ssh/ssh_config
    - require:
      - pkg: openssh-client
```

`ssh/server.sls:`

```
include:
  - ssh

openssh-server:
  pkg.installed

sshd:
  service.running:
    - require:
      - pkg: openssh-client
      - pkg: openssh-server
      - file: /etc/ssh/banner
      - file: /etc/ssh/sshd_config

/etc/ssh/sshd_config:
  file.managed:
    - user: root
    - group: root
    - mode: 644
    - source: salt://ssh/sshd_config
    - require:
      - pkg: openssh-server

/etc/ssh/banner:
  file:
    - managed
    - user: root
    - group: root
    - mode: 644
    - source: salt://ssh/banner
    - require:
      - pkg: openssh-server
```

Note: Notice that we use two similar ways of denoting that a file is managed by Salt. In the */etc/ssh/sshd_config* state section above, we use the *file.managed* state declaration whereas with the */etc/ssh/banner* state section, we use the *file* state declaration and add a *managed* attribute to that state declaration. Both ways produce an identical result; the first way – using *file.managed* – is merely a shortcut.

Now our State Tree looks like this:

```
apache/init.sls
apache/httpd.conf
ssh/init.sls
ssh/server.sls
ssh/banner
ssh/ssh_config
ssh/sshd_config
```

This example now introduces the `include` statement. The include statement includes another SLS file so that components found in it can be required, watched or as will soon be demonstrated - extended.

The include statement allows for states to be cross linked. When an SLS has an include statement it is literally extended to include the contents of the included SLS files.

Note that some of the SLS files are called init.sls, while others are not. More info on what this means can be found in the *States Tutorial*.

Extending Included SLS Data

Sometimes SLS data needs to be extended. Perhaps the apache service needs to watch additional resources, or under certain circumstances a different file needs to be placed.

In these examples, the first will add a custom banner to ssh and the second will add more watchers to apache to include mod_python.

`ssh/custom-server.sls:`

```
include:
  - ssh.server

extend:
  /etc/ssh/banner:
    file:
      - source: salt://ssh/custom-banner
```

`python/mod_python.sls`:

```
include:
  - apache

extend:
  apache:
    service:
      - watch:
        - pkg: mod_python

mod_python:
  pkg.installed
```

The `custom-server.sls` file uses the extend statement to overwrite where the banner is being downloaded from, and therefore changing what file is being used to configure the banner.

In the new mod_python SLS the mod_python package is added, but more importantly the apache service was extended to also watch the mod_python package.

Using extend with require or watch

The `extend` statement works differently for `require` or `watch`. It appends to, rather than replacing the requisite component.

Understanding the Render System

Since SLS data is simply that (data), it does not need to be represented with YAML. Salt defaults to YAML because it is very straightforward and easy to learn and use. But the SLS files can be rendered from almost any imaginable medium, so long as a renderer module is provided.

The default rendering system is the `yaml_jinja` renderer. The `yaml_jinja` renderer will first pass the template through the Jinja2 templating system, and then through the YAML parser. The benefit here is that full programming constructs are available when creating SLS files.

Other renderers available are `yaml_mako` and `yaml_wempy` which each use the Mako or Wempy templating system respectively rather than the jinja templating system, and more notably, the pure Python or `py` and `pydsl` renderers. The `py` renderer allows for SLS files to be written in pure Python, allowing for the utmost level of flexibility and power when preparing SLS data; while the *pydsl* renderer provides a flexible, domain-specific language for authoring SLS data in Python.

Note: The templating engines described above aren't just available in SLS files. They can also be used in `file.managed` states, making file management much more dynamic and flexible. Some examples for using templates in managed files can be found in the documentation for the *file states*, as well as the *MooseFS example* below.

Getting to Know the Default - yaml_jinja

The default renderer - `yaml_jinja`, allows for use of the jinja templating system. A guide to the Jinja templating system can be found here: http://jinja.pocoo.org/docs

When working with renderers a few very useful bits of data are passed in. In the case of templating engine based renderers, three critical components are available, `salt`, `grains`, and `pillar`. The `salt` object allows for any Salt function to be called from within the template, and `grains` allows for the Grains to be accessed from within the template. A few examples:

`apache/init.sls:`

```
apache:
  pkg.installed:
    {% if grains['os'] == 'RedHat'%}
    - name: httpd
    {% endif %}
  service.running:
    {% if grains['os'] == 'RedHat'%}
    - name: httpd
    {% endif %}
    - watch:
      - pkg: apache
      - file: /etc/httpd/conf/httpd.conf
      - user: apache
  user.present:
    - uid: 87
    - gid: 87
    - home: /var/www/html
    - shell: /bin/nologin
    - require:
      - group: apache
  group.present:
    - gid: 87
    - require:
      - pkg: apache

/etc/httpd/conf/httpd.conf:
  file.managed:
    - source: salt://apache/httpd.conf
    - user: root
    - group: root
    - mode: 644
```

This example is simple. If the `os` grain states that the operating system is Red Hat, then the name of the Apache package and service needs to be httpd. A more aggressive way to use Jinja can be found here, in a module to set up a MooseFS distributed filesystem chunkserver:

`moosefs/chunk.sls:`

```
include:
  - moosefs

{% for mnt in salt['cmd.run']('ls /dev/data/moose*').split() %}
/mnt/moose{{ mnt[-1] }}:
  mount.mounted:
    - device: {{ mnt }}
    - fstype: xfs
    - mkmnt: True
  file.directory:
    - user: mfs
    - group: mfs
    - require:
      - user: mfs
      - group: mfs
{% endfor %}

/etc/mfshdd.cfg:
  file.managed:
    - source: salt://moosefs/mfshdd.cfg
    - user: root
    - group: root
    - mode: 644
    - template: jinja
    - require:
      - pkg: mfs-chunkserver

/etc/mfschunkserver.cfg:
  file.managed:
    - source: salt://moosefs/mfschunkserver.cfg
    - user: root
    - group: root
    - mode: 644
    - template: jinja
    - require:
      - pkg: mfs-chunkserver

mfs-chunkserver:
  pkg:
    - installed
mfschunkserver:
  service:
    - running
    - require:
{% for mnt in salt['cmd.run']('ls /dev/data/moose*') %}
      - mount: /mnt/moose{{ mnt[-1] }}
      - file: /mnt/moose{{ mnt[-1] }}
{% endfor %}
      - file: /etc/mfschunkserver.cfg
      - file: /etc/mfshdd.cfg
      - file: /var/lib/mfs
```

This example shows much more of the available power of Jinja. Multiple for loops are used to dynamically detect available hard drives and set them up to be mounted, and the `salt` object is used multiple times to call shell commands to gather data.

Introducing the Python and the PyDSL Renderers

Sometimes the chosen default renderer might not have enough logical power to accomplish the needed task. When this happens, the Python renderer can be used. Normally a YAML renderer should be used for the majority of SLS files, but an SLS file set to use another renderer can be easily added to the tree.

This example shows a very basic Python SLS file:

`python/django.sls:`

```
#!py

def run():
    '''
    Install the django package
    '''
    return {'include': ['python'],
            'django': {'pkg': ['installed']}}
```

This is a very simple example; the first line has an SLS shebang that tells Salt to not use the default renderer, but to use the `py` renderer. Then the run function is defined, the return value from the run function must be a Salt friendly data structure, or better known as a Salt *HighState data structure*.

Alternatively, using the *pydsl* renderer, the above example can be written more succinctly as:

`python/django.sls`:

```
#!pydsl

include('python', delayed=True)
state('django').pkg.installed()
```

This Python examples would look like this if they were written in YAML:

```
include:
  - python

django:
  pkg.installed
```

This example clearly illustrates that; one, using the YAML renderer by default is a wise decision and two, unbridled power can be obtained where needed by using a pure Python SLS.

Running and debugging salt states.

Once the rules in an SLS are ready, they should be tested to ensure they work properly. To invoke these rules, simply execute `salt '*' state.highstate` on the command line. If you get back only hostnames with a `:` after, but no return, chances are there is a problem with one or more of the sls files. On the minion, use the `salt-call` command: `salt-call state.highstate -l debug` to examine the output for errors. This should help troubleshoot the issue. The minions can also be started in the foreground in debug mode: `salt-minion -l debug`.

Next Reading

With an understanding of states, the next recommendation is to become familiar with Salt's pillar interface:

> *Pillar Walkthrough*

7.3.2 States tutorial, part 1 - Basic Usage

The purpose of this tutorial is to demonstrate how quickly you can configure a system to be managed by Salt States. For detailed information about the state system please refer to the full *states reference*.

This tutorial will walk you through using Salt to configure a minion to run the Apache HTTP server and to ensure the server is running.

Before continuing make sure you have a working Salt installation by following the *installation* and the *configuration* instructions.

Stuck?

There are many ways to *get help from the Salt community* including our mailing list and our IRC channel #salt.

Setting up the Salt State Tree

States are stored in text files on the master and transferred to the minions on demand via the master's File Server. The collection of state files make up the *State Tree*.

To start using a central state system in Salt, the Salt File Server must first be set up. Edit the master config file (`file_roots`) and uncomment the following lines:

```
file_roots:
  base:
    - /srv/salt
```

Note: If you are deploying on FreeBSD via ports, the `file_roots` path defaults to `/usr/local/etc/salt/states`.

Restart the Salt master in order to pick up this change:

```
pkill salt-master
salt-master -d
```

Preparing the Top File

On the master, in the directory uncommented in the previous step, (`/srv/salt` by default), create a new file called `top.sls` and add the following:

```
base:
  '*':
    - webserver
```

The *top file* is separated into environments (discussed later). The default environment is `base`. Under the `base` environment a collection of minion matches is defined; for now simply specify all hosts (`*`).

Targeting minions

The expressions can use any of the targeting mechanisms used by Salt — minions can be matched by glob, PCRE regular expression, or by *grains*. For example:

```
base:
  'os:Fedora':
    - match: grain
    - webserver
```

Create an `sls` file

In the same directory as the *top file*, create a file named `webserver.sls`, containing the following:

```
apache:                 # ID declaration
  pkg:                  # state declaration
    - installed         # function declaration
```

The first line, called the *ID declaration*, is an arbitrary identifier. In this case it defines the name of the package to be installed.

Note: The package name for the Apache httpd web server may differ depending on OS or distro — for example, on Fedora it is `httpd` but on Debian/Ubuntu it is `apache2`.

The second line, called the *state declaration*, defines which of the Salt States we are using. In this example, we are using the `pkg state` to ensure that a given package is installed.

The third line, called the *function declaration*, defines which function in the `pkg state` module to call.

Renderers

States *sls* files can be written in many formats. Salt requires only a simple data structure and is not concerned with how that data structure is built. Templating languages and DSLs are a dime-a-dozen and everyone has a favorite.

Building the expected data structure is the job of Salt *renderers* and they are dead-simple to write.

In this tutorial we will be using YAML in Jinja2 templates, which is the default format. The default can be changed by editing `renderer` in the master configuration file.

Install the package

Next, let's run the state we created. Open a terminal on the master and run:

```
% salt '*' state.highstate
```

Our master is instructing all targeted minions to run `state.highstate`. When a minion executes a highstate call it will download the *top file* and attempt to match the expressions. When it does match an expression the modules listed for it will be downloaded, compiled, and executed.

Once completed, the minion will report back with a summary of all actions taken and all changes made.

SLS File Namespace

Note that in the *example* above, the SLS file `webserver.sls` was referred to simply as `webserver`. The namespace for SLS files follows a few simple rules:

1. The `.sls` is discarded (i.e. `webserver.sls` becomes `webserver`).
2. **Subdirectories can be used for better organization.**
    (a) Each subdirectory is represented by a dot.
    (b) `webserver/dev.sls` is referred to as `webserver.dev`.
3. A file called `init.sls` in a subdirectory is referred to by the path of the directory. So, `webserver/init.sls` is referred to as `webserver`.
4. If both `webserver.sls` and `webserver/init.sls` happen to exist, `webserver/init.sls` will be ignored and `webserver.sls` will be the file referred to as `webserver`.

Troubleshooting Salt

If the expected output isn't seen, the following tips can help to narrow down the problem.

Turn up logging Salt can be quite chatty when you change the logging setting to `debug`:

```
salt-minion -l debug
```

Run the minion in the foreground By not starting the minion in daemon mode (*-d*) one can view any output from the minion as it works:

```
salt-minion &
```

Increase the default timeout value when running **salt**. For example, to change the default timeout to 60 seconds:

```
salt -t 60
```

For best results, combine all three:

```
salt-minion -l debug &          # On the minion
salt '*' state.highstate -t 60  # On the master
```

Next steps

This tutorial focused on getting a simple Salt States configuration working. *Part 2* will build on this example to cover more advanced *sls* syntax and will explore more of the states that ship with Salt.

7.3.3 States tutorial, part 2 - More Complex States, Requisites

Note: This tutorial builds on topics covered in *part 1*. It is recommended that you begin there.

In the *last part* of the Salt States tutorial we covered the basics of installing a package. We will now modify our `webserver.sls` file to have requirements, and use even more Salt States.

Call multiple States

You can specify multiple *state declarations* under an *ID declaration*. For example, a quick modification to our `webserver.sls` to also start Apache if it is not running:

```
apache:
  pkg:
    - installed
  service:
    - running
    - require:
      - pkg: apache
```

Try stopping Apache before running `state.highstate` once again and observe the output.

Expand the SLS module

As you have seen, SLS modules are appended with the file extension `.sls` and are referenced by name starting at the root of the state tree. An SLS module can be also defined as a directory. Demonstrate that now by creating a directory named `webserver` and moving and renaming `webserver.sls` to `webserver/init.sls`. Your state directory should now look like this:

```
|- top.sls
`- webserver/
   `- init.sls
```

Organizing SLS modules

You can place additional `.sls` files in a state file directory. This affords much cleaner organization of your state tree on the filesystem. For example, if we created a `webserver/django.sls` file that module would be referenced as `webserver.django`.

In addition, States provide powerful includes and extending functionality which we will cover in *Part 3*.

Require other states

We now have a working installation of Apache so let's add an HTML file to customize our website. It isn't exactly useful to have a website without a webserver so we don't want Salt to install our HTML file until Apache is installed and running. Include the following at the bottom of your `webserver/init.sls` file:

```
1  apache:
2    pkg:
3      - installed
4    service:
5      - running
6      - require:
7        - pkg: apache
8
9  /var/www/index.html:                        # ID declaration
10   file:                                     # state declaration
11     - managed                               # function
12     - source: salt://webserver/index.html   # function arg
13     - require:                              # requisite declaration
14       - pkg: apache                         # requisite reference
```

line 9 is the *ID declaration*. In this example it is the location we want to install our custom HTML file. (**Note:** the default location that Apache serves may differ from the above on your OS or distro. `/srv/www` could also be a likely place to look.)

Line 10 the *state declaration*. This example uses the Salt `file state`.

Line 11 is the *function declaration*. The `managed function` will download a file from the master and install it in the location specified.

Line 12 is a *function arg declaration* which, in this example, passes the `source` argument to the `managed function`.

Line 13 is a *requisite declaration*.

Line 14 is a *requisite reference* which refers to a state and an ID. In this example, it is referring to the `ID declaration` from our example in *part 1*. This declaration tells Salt not to install the HTML file until Apache is installed.

Next, create the `index.html` file and save it in the `webserver` directory:

```
<html>
    <head><title>Salt rocks</title></head>
    <body>
        <h1>This file brought to you by Salt</h1>
    </body>
</html>
```

Last, call `state.highstate` again and the minion will fetch and execute the highstate as well as our HTML file from the master using Salt's File Server:

```
salt '*' state.highstate
```

Verify that Apache is now serving your custom HTML.

`require` vs. `watch`

There are two *requisite declarations*, "require" and "watch". Not every state supports "watch". The `service state` does support "watch" and will restart a service based on the watch condition.

For example, if you use Salt to install an Apache virtual host configuration file and want to restart Apache whenever that file is changed you could modify our Apache example from earlier as follows:

```
/etc/httpd/extra/httpd-vhosts.conf:
  file:
    - managed
    - source: salt://webserver/httpd-vhosts.conf

apache:
  pkg:
    - installed
  service:
    - running
    - watch:
      - file: /etc/httpd/extra/httpd-vhosts.conf
    - require:
      - pkg: apache
```

If the pkg and service names differ on your OS or distro of choice you can specify each one separately using a *name declaration* which explained in *Part 3*.

Next steps

In *part 3* we will discuss how to use includes, extends and templating to make a more complete State Tree configuration.

7.3.4 States tutorial, part 3 - Templating, Includes, Extends

Note: This tutorial builds on topics covered in *part 1* and *part 2*. It is recommended that you begin there.

This part of the tutorial will cover more advanced templating and configuration techniques for `sls` files.

Templating SLS modules

SLS modules may require programming logic or inline execution. This is accomplished with module templating. The default module templating system used is Jinja2 and may be configured by changing the `renderer` value in the master config.

All states are passed through a templating system when they are initially read. To make use of the templating system, simply add some templating markup. An example of an sls module with templating markup may look like this:

```
{% for usr in ['moe','larry','curly'] %}
{{ usr }}:
  user.present
{% endfor %}
```

This templated sls file once generated will look like this:

```
moe:
  user.present
larry:
  user.present
curly:
  user.present
```

Here's a more complex example:

```
{% for usr in 'moe','larry','curly' %}
{{ usr }}:
  group:
    - present
  user:
    - present
    - gid_from_name: True
    - require:
      - group: {{ usr }}
{% endfor %}
```

Using Grains in SLS modules

Often times a state will need to behave differently on different systems. *Salt grains* objects are made available in the template context. The *grains* can be used from within sls modules:

```
apache:
  pkg.installed:
    {% if grains['os'] == 'RedHat' %}
    - name: httpd
    {% elif grains['os'] == 'Ubuntu' %}
    - name: apache2
    {% endif %}
```

Calling Salt modules from templates

All of the Salt modules loaded by the minion are available within the templating system. This allows data to be gathered in real time on the target system. It also allows for shell commands to be run easily from within the sls modules.

The Salt module functions are also made available in the template context as `salt`:

```
moe:
  user:
    - present
    - gid: {{ salt['file.group_to_gid']('some_group_that_exists') }}
```

Note that for the above example to work, `some_group_that_exists` must exist before the state file is processed by the templating engine.

Below is an example that uses the `network.hw_addr` function to retrieve the MAC address for eth0:

```
salt['network.hw_addr']('eth0')
```

Advanced SLS module syntax

Lastly, we will cover some incredibly useful techniques for more complex State trees.

Include declaration

A previous example showed how to spread a Salt tree across several files. Similarly, *requisites* span multiple files by using an *include declaration*. For example:

`python/python-libs.sls:`

```
python-dateutil:
  pkg.installed
```

`python/django.sls:`

```
include:
  - python.python-libs

django:
  pkg.installed:
    - require:
      - pkg: python-dateutil
```

Extend declaration

You can modify previous declarations by using an *extend declaration*. For example the following modifies the Apache tree to also restart Apache when the vhosts file is changed:

`apache/apache.sls:`

```
apache:
  pkg.installed
```

`apache/mywebsite.sls:`

```
include:
  - apache.apache

extend:
  apache:
    service:
      - running
      - watch:
        - file: /etc/httpd/extra/httpd-vhosts.conf

/etc/httpd/extra/httpd-vhosts.conf:
  file.managed:
    - source: salt://apache/httpd-vhosts.conf
```

Using extend with require or watch

The `extend` statement works differently for `require` or `watch`. It appends to, rather than replacing the requisite component.

Name declaration

You can override the *ID declaration* by using a *name declaration*. For example, the previous example is a bit more maintainable if rewritten as follows:

`apache/mywebsite.sls`:

```
include:
  - apache.apache

extend:
  apache:
    service:
      - running
      - watch:
        - file: mywebsite

mywebsite:
  file.managed:
    - name: /etc/httpd/extra/httpd-vhosts.conf
    - source: salt://apache/httpd-vhosts.conf
```

Names declaration

Even more powerful is using a *names declaration* to override the *ID declaration* for multiple states at once. This often can remove the need for looping in a template. For example, the first example in this tutorial can be rewritten without the loop:

```
stooges:
  user.present:
    - names:
      - moe
      - larry
      - curly
```

Next steps

In *part 4* we will discuss how to use salt's `file_roots` to set up a workflow in which states can be "promoted" from dev, to QA, to production.

7.3.5 States tutorial, part 4

Note: This tutorial builds on topics covered in *part 1*, *part 2* and *part 3*. It is recommended that you begin there.

This part of the tutorial will show how to use salt's `file_roots` to set up a workflow in which states can be "promoted" from dev, to QA, to production.

Salt fileserver path inheritance

Salt's fileserver allows for more than one root directory per environment, like in the below example, which uses both a local directory and a secondary location shared to the salt master via NFS:

```
# In the master config file (/etc/salt/master)
file_roots:
  base:
    - /srv/salt
    - /mnt/salt-nfs/base
```

Salt's fileserver collapses the list of root directories into a single virtual environment containing all files from each root. If the same file exists at the same relative path in more than one root, then the top-most match "wins". For example, if `/srv/salt/foo.txt` and `/mnt/salt-nfs/base/foo.txt` both exist, then `salt://foo.txt` will point to `/srv/salt/foo.txt`.

Environment configuration

Configure a multiple-environment setup like so:

```
file_roots:
  base:
    - /srv/salt/prod
  qa:
    - /srv/salt/qa
    - /srv/salt/prod
  dev:
    - /srv/salt/dev
    - /srv/salt/qa
    - /srv/salt/prod
```

Given the path inheritance described above, files within `/srv/salt/prod` would be available in all environments. Files within `/srv/salt/qa` would be available in both `qa`, and `dev`. Finally, the files within `/srv/salt/dev` would only be available within the `dev` environment.

Based on the order in which the roots are defined, new files/states can be placed within `/srv/salt/dev`, and pushed out to the dev hosts for testing.

Those files/states can then be moved to the same relative path within `/srv/salt/qa`, and they are now available only in the `dev` and `qa` environments, allowing them to be pushed to QA hosts and tested.

Finally, if moved to the same relative path within `/srv/salt/prod`, the files are now available in all three environments.

Practical Example

As an example, consider a simple website, installed to `/var/www/foobarcom`. Below is a top.sls that can be used to deploy the website:

`/srv/salt/prod/top.sls:`

```
base:
  'web*prod*':
    - webserver.foobarcom
qa:
  'web*qa*':
    - webserver.foobarcom
dev:
  'web*dev*':
    - webserver.foobarcom
```

Using pillar, roles can be assigned to the hosts:

`/srv/pillar/top.sls:`

```
base:
  'web*prod*':
    - webserver.prod
  'web*qa*':
    - webserver.qa
  'web*dev*':
    - webserver.dev
```

`/srv/pillar/webserver/prod.sls:`

```
webserver_role: prod
```

`/srv/pillar/webserver/qa.sls:`

```
webserver_role: qa
```

`/srv/pillar/webserver/dev.sls:`

```
webserver_role: dev
```

And finally, the SLS to deploy the website:

`/srv/salt/prod/webserver/foobarcom.sls:`

```
{% if pillar.get('webserver_role', '') %}
/var/www/foobarcom:
  file.recurse:
    - source: salt://webserver/src/foobarcom
    - env: {{ pillar['webserver_role'] }}
    - user: www
    - group: www
    - dir_mode: 755
    - file_mode: 644
{% endif %}
```

Given the above SLS, the source for the website should initially be placed in `/srv/salt/dev/webserver/src/foobarcom`.

First, let's deploy to dev. Given the configuration in the top file, this can be done using `state.highstate`:

```
salt --pillar 'webserver_role:dev' state.highstate
```

However, in the event that it is not desirable to apply all states configured in the top file (which could be likely in more complex setups), it is possible to apply just the states for the `foobarcom` website, using `state.sls`:

```
salt --pillar 'webserver_role:dev' state.sls webserver.foobarcom
```

Once the site has been tested in dev, then the files can be moved from `/srv/salt/dev/webserver/src/foobarcom` to `/srv/salt/qa/webserver/src/foobarcom`, and deployed using the following:

```
salt --pillar 'webserver_role:qa' state.sls webserver.foobarcom
```

Finally, once the site has been tested in qa, then the files can be moved from `/srv/salt/qa/webserver/src/foobarcom` to `/srv/salt/prod/webserver/src/foobarcom`, and deployed using the following:

```
salt --pillar 'webserver_role:prod' state.sls webserver.foobarcom
```

Thanks to Salt's fileserver inheritance, even though the files have been moved to within `/srv/salt/prod`, they are still available from the same `salt://` URI in both the qa and dev environments.

Continue Learning

The best way to continue learning about Salt States is to read through the *reference documentation* and to look through examples of existing *state trees*. Many pre-configured state trees can be found on Github in the saltstack-formulas collection of repositories.

If you have any questions, suggestions, or just want to chat with other people who are using Salt, we have a very *active community* and we'd love to hear from you.

In addition, by continuing to *part 5*, you can learn about the powerful orchestration of which Salt is capable.

7.3.6 States Tutorial, Part 5 - Orchestration with Salt

Note: This tutorial builds on some of the topics covered in the earlier *States Walkthrough* pages. It is recommended to start with *Part 1* if you are not familiar with how to use states.

Orchestration can be accomplished in two distinct ways:

1. The *OverState System*. Added in version 0.11.0, this Salt *Runner* allows for SLS files to be organized into stages, and to require that one or more stages successfully execute before another stage will run.
2. The *Orchestrate Runner*. Added in version 0.17.0, this Salt *Runner* can use the full suite of *requisites* available in states, and can also execute states/functions using salt-ssh. This runner was designed with the eventual goal of replacing the *OverState*.

The OverState System

Often servers need to be set up and configured in a specific order, and systems should only be set up if systems earlier in the sequence has been set up without any issues.

The OverState system can be used to orchestrate deployment in a smooth and reliable way across multiple systems in small to large environments.

The OverState SLS

The OverState system is managed by an SLS file named `overstate.sls`, located in the root of a Salt fileserver environment.

The overstate.sls configures an unordered list of stages, each stage defines the minions on which to execute the state, and can define what sls files to run, execute a `state.highstate`, or execute a function. Here's a sample overstate.sls:

```
mysql:
  match: 'db*'
  sls:
    - mysql.server
    - drbd
webservers:
  match: 'web*'
  require:
    - mysql
all:
  match: '*'
  require:
    - mysql
    - webservers
```

Given the above setup, the OverState will be carried out as follows:

1. The `mysql` stage will be executed first because it is required by the `webservers` and `all` stages. It will execute `state.sls` once for each of the two listed SLS targets (`mysql.server` and `drbd`). These states will be executed on all minions whose minion ID starts with "db".
2. The `webservers` stage will then be executed, but only if the `mysql` stage executes without any failures. The `webservers` stage will execute a `state.highstate` on all minions whose minion IDs start with "web".
3. Finally, the `all` stage will execute, running `state.highstate` on all systems, if and only if the `mysql` and `webservers` stages completed without any failures.

Any failure in the above steps would cause the requires to fail, preventing the dependent stages from executing.

Using Functions with OverState

In the above example, you'll notice that the stages lacking an `sls` entry run a `state.highstate`. As mentioned earlier, it is also possible to execute other functions in a stage. This functionality was added in version 0.15.0.

Running a function is easy:

```
http:
  function:
    pkg.install:
      - httpd
```

The list of function arguments are defined after the declared function. So, the above stage would run `pkg.install http`. Requisites only function properly if the given function supports returning a custom return code.

Executing an OverState

Since the OverState is a *Runner*, it is executed using the `salt-run` command. The runner function for the OverState is `state.over`.

```
salt-run state.over
```

The function will by default look in the root of the `base` environment (as defined in `file_roots`) for a file called `overstate.sls`, and then execute the stages defined within that file.

Different environments and paths can be used as well, by adding them as positional arguments:

```
salt-run state.over dev /root/other-overstate.sls
```

The above would run an OverState using the `dev` fileserver environment, with the stages defined in `/root/other-overstate.sls`.

> **Warning:** Since these are positional arguments, when defining the path to the overstate file the environment must also be specified, even if it is the `base` environment.

Note: Remember, salt-run is always executed on the master.

The Orchestrate Runner

New in version 0.17.0.

As noted above in the introduction, the Orchestrate Runner (originally called the state.sls runner) offers all the functionality of the OverState, but with a couple advantages:

- All *requisites* available in states can be used.
- The states/functions can be executed using salt-ssh.

The Orchestrate Runner was added with the intent to eventually deprecate the OverState system, however the OverState will still be maintained for the foreseeable future.

Configuration Syntax

The configuration differs slightly from that of the OverState, and more closely resembles the configuration schema used for states.

To execute a state, use `salt.state`:

```
install_nginx:
  salt.state:
    - tgt: 'web*'
    - sls:
      - nginx
```

To execute a function, use `salt.function`:

```
cmd.run:
  salt.function:
    - tgt: '*'
    - arg:
      - rm -rf /tmp/foo
```

Triggering a Highstate

Wheras with the OverState, a Highstate is run by simply omitting an `sls` or `function` argument, with the Orchestrate Runner the Highstate must explicitly be requested by using `highstate:  True`:

```
webserver_setup:
  salt.state:
    - tgt: 'web*'
    - highstate: True
```

Executing the Orchestrate Runner

The Orchestrate Runner can be executed using the `state.orchestrate` runner function. `state.orch` also works, for those that would like to type less.

Assuming that your `base` environment is located at `/srv/salt`, and you have placed a configuration file in `/srv/salt/orchestration/webserver.sls`, then the following could both be used:

```
salt-run state.orchestrate orchestration.webserver
salt-run state.orch orchestration.webserver
```

Changed in version 2014.1.1: The runner function was renamed to `state.orchestrate`. In versions 0.17.0 through 2014.1.0, `state.sls` must be used. This was renamed to avoid confusion with the `state.sls` execution function.

```
salt-run state.sls orchestration.webserver
```

More Complex Orchestration

Many states/functions can be configured in a single file, which when combined with the full suite of *requisites*, can be used to easily configure complex orchestration tasks. Additionally, the states/functions will be executed in the order in which they are defined, unless prevented from doing so by any *requisites*, as is the default in SLS files since 0.17.0.

```
cmd.run:
  salt.function:
    - tgt: 10.0.0.0/24
    - tgt_type: ipcidr
    - arg:
      - bootstrap

storage_setup:
  salt.state:
    - tgt: 'role:storage'
    - tgt_type: grain
    - sls: ceph
    - require:
      - salt: webserver_setup

webserver_setup:
  salt.state:
    - tgt: 'web*'
    - highstate: True
```

Given the above setup, the orchestration will be carried out as follows:

1. The shell command `bootstrap` will be executed on all minions in the 10.0.0.0/24 subnet.
2. A Highstate will be run on all minions whose ID starts with "web", since the `storage_setup` state requires it.
3. Finally, the `ceph` SLS target will be executed on all minions which have a grain called `role` with a value of `storage`.

7.3.7 Understanding State Ordering

Note: This tutorial is an intermediate level tutorial. Some basic understanding of the state system and writing Salt Formulas is assumed.

Salt's state system is built to deliver all of the power of configuration management systems without sacrificing simplicity. This tutorial is made to help users understand in detail just how the order is defined for state executions in Salt.

This tutorial is written to represent the behavior of Salt as of version 0.17.0.

Compiler Basics

To understand ordering in depth some very basic knowledge about the state compiler is very helpful. No need to worry though, this is very high level!

High Data and Low Data

When defining Salt Formulas in YAML the data that is being represented is referred to by the compiler as High Data. When the data is initially loaded into the compiler it is a single large python dictionary, this dictionary can be viewed raw by running:

```
salt '*' state.show_highstate
```

This "High Data" structure is then compiled down to "Low Data". The Low Data is what is matched up to create individual executions in Salt's configuration management system. The low data is an ordered list of single state calls to execute. Once the low data is compiled the evaluation order can be seen.

The low data can be viewed by running:

```
salt '*' state.show_lowstate
```

Note: The state execution module contains MANY functions for evaluating the state system and is well worth a read! These routines can be very useful when debugging states or to help deepen one's understanding of Salt's state system.

As an example, a state written thusly:

```
apache:
  pkg:
    - installed
  service:
    - running
    - watch:
      - file: /etc/httpd/conf.d/httpd.conf
      - pkg: apache
/etc/httpd/conf.d/httpd.conf:
  file:
    - managed
    - source: salt://apache/httpd.conf
```

Will have High Data which looks like this represented in json:

```
{
    "apache": {
        "pkg": [
            "installed",
            {
                "order": 10000
            }
        ],
        "service": [
            "running",
            {
                "watch": [
                    {
                        "file": "/etc/httpd/conf.d/httpd.conf"
                    },
                    {
                        "pkg": "apache"
                    }
                ]
            },
            {
                "order": 10001
            }
        ],
        "__sls__": "apache",
        "__env__": "base"
    },
    "/etc/httpd/conf.d/httpd.conf": {
        "file": [
            "managed",
            {
                "source": "salt://apache/httpd.conf"
            },
            {
                "order": 10002
            }
        ],
        "__sls__": "apache",
        "__env__": "base"
    }
}
```

The subsequent Low Data will look like this:

```
[
    {
        "name": "apache",
        "state": "pkg",
        "__id__": "apache",
        "fun": "installed",
        "__env__": "base",
        "__sls__": "apache",
        "order": 10000
    },
    {
        "name": "apache",
        "watch": [
            {
                "file": "/etc/httpd/conf.d/httpd.conf"
            },
            {
                "pkg": "apache"
            }
        ],
        "state": "service",
        "__id__": "apache",
        "fun": "running",
        "__env__": "base",
        "__sls__": "apache",
        "order": 10001
    },
    {
        "name": "/etc/httpd/conf.d/httpd.conf",
        "source": "salt://apache/httpd.conf",
        "state": "file",
        "__id__": "/etc/httpd/conf.d/httpd.conf",
        "fun": "managed",
        "__env__": "base",
        "__sls__": "apache",
        "order": 10002
    }
]
```

This tutorial discusses the Low Data evaluation and the state runtime.

Ordering Layers

Salt defines 2 order interfaces which are evaluated in the state runtime and defines these orders in a number of passes.

Definition Order

Note: The Definition Order system can be disabled by turning the option *state_auto_order* to *False* in the master configuration file.

The top level of ordering is the *Definition Order*. The *Definition Order* is the order in which states are defined in salt formulas. This is very straightforward on basic states which do not contain `include` statements or a `top` file, as the states are just ordered from the top of the file, but the include system starts to bring in some simple rules for how the *Definitiion Order* is defined.

Looking back at the "Low Data" and "High Data" shown above, the order key has been transparently added to the data to enable the *Definition Order*.

The Include Statement Basically, if there is an include statement in a formula, then the formulas which are included will be run BEFORE the contents of the formula which is including them. Also, the include statement is a list, so they will be loaded in the order in which they are included.

In the following case:

`foo.sls`

```
include:
  - bar
  - baz
```

`bar.sls`

```
include:
  - quo
```

`baz.sls`

```
include:
  - qux
```

In the above case if *state.sls foo* were called then the formulas will be loaded in the following order:

1. quo
2. bar
3. qux
4. baz
5. foo

The *order* Flag

The *Definition Order* happens transparently in the background, but the ordering can be explicitly overridden using the *order* flag in states:

```
apache:
  pkg:
    - installed
    - order: 1
```

This order flag will over ride the definition order, this makes it very simple to create states that are always executed first, last or in specific stages, a great example is defining a number of package repositories that need to be set up before anything else, or final checks that need to be run at the end of a state run by using *order: last* or *order: -1*.

When the order flag is explicitly set the *Definition Order* system will omit setting an order for that state and directly use the order flag defined.

Lexicographical Fall-back

Salt states were written to ALWAYS execute in the same order. Before the introduction of *Definition Order* in version 0.17.0 everything was ordered lexicographically according to the name of the state, then function then id.

This is the way Salt has always ensured that states always run in the same order regardless of where they are deployed, the addition of the *Definition Order* method mealy makes this finite ordering easier to follow.

The lexicographical ordering is still applied but it only has any effect when two order statements collide. This means that if multiple states are assigned the same order number that they will fall back to lexicographical ordering to ensure that every execution still happens in a finite order.

Note: If running with *state_auto_order: False* the *order* key is not set automatically, since the Lexicographical order can be derived from other keys.

Requisite Ordering

Salt states are fully declarative, in that they are written to declare the state in which a system should be. This means that components can require that other components have been set up successfully. Unlike the other ordering systems, the *Requisite* system in Salt is evaluated at runtime.

The requisite system is also built to ensure that the ordering of execution never changes, but is always the same for a given set of states. This is accomplished by using a runtime that processes states in a completely predictable order instead of using an event loop based system like other declarative configuration management systems.

Runtime Requisite Evaluation

The requisite system is evaluated as the components are found, and the requisites are always evaluated in the same order. This explanation will be followed by an example, as the raw explanation may be a little dizzying at first as it creates a linear dependency evaluation sequence.

The "Low Data" is an ordered list or dictionaries, the state runtime evaluates each dictionary in the order in which they are arranged in the list. When evaluating a single dictionary it is checked for requisites, requisites are evaluated in order, *require* then *watch* then *prereq*.

Note: If using requisite in statements like require_in and watch_in these will be compiled down to require and watch statements before runtime evaluation.

Each requisite contains an ordered list of requisites, these requisites are looked up in the list of dictionaries and then executed. Once all requisites have been evaluated and executed then the requiring state can safely be run (or not run if requisites have not been met).

This means that the requisites are always evaluated in the same order, again ensuring one of the core design principals of Salt's State system to ensure that execution is always finite is intact.

Simple Runtime Evaluation Example

Given the above "Low Data" the states will be evaluated in the following order:

1. The pkg.installed is executed ensuring that the apache package is installed, it contains no requisites and is therefore the first defined state to execute.
2. The service.running state is evaluated but NOT executed, a watch requisite is found, therefore they are read in order, the runtime first checks for the file, sees that it has not been executed and calls for the file state to be evaluated.
3. The file state is evaluated AND executed, since it, like the pkg state does not contain any requisites.
4. The evaluation of the service state continues, it next checks the pkg requisite and sees that it is met, with all requisites met the service state is now executed.

Best Practice

The best practice in Salt is to choose a method and stick with it, official states are written using requisites for all associations since requisites create clean, traceable dependency trails and make for the most portable formulas. To accomplish something similar to how classical imperative systems function all requisites can be omitted and the `failhard` option then set to *True* in the master configuration, this will stop all state runs at the first instance of a failure.

In the end, using requisites creates very tight and fine grained states, not using requisites makes full sequence runs and while slightly easier to write, and gives much less control over the executions.

7.4 Advanced Topics

7.4.1 SaltStack Walkthrough

Welcome!

Welcome to SaltStack! I am excited that you are interested in Salt and starting down the path to better infrastructure management. I developed (and am continuing to develop) Salt with the goal of making the best software available to manage computers of almost any kind. I hope you enjoy working with Salt and that the software can solve your real world needs!

- Thomas S Hatch
- Salt creator and chief developer
- CTO of SaltStack, Inc.

Note: This is the first of a series of walkthroughs and serves as the best entry point for people new to Salt, after this be sure to read up on pillar and more on states:

Starting States

Pillar Walkthrough

Getting Started

What is Salt?

Salt is a different approach to infrastructure management, it is founded on the idea that high speed communication with large numbers of systems can open up new capabilities. This approach makes Salt a powerful multitasking system that can solve many specific problems in an infrastructure. The backbone of Salt is the remote execution engine, which creates a high speed, secure and bi-directional communication net for groups of systems. On top of this communication system Salt provides an extremely fast, flexible and easy to use configuration management system called `Salt States`.

This unique approach to management makes for a transparent control system that is not only amazingly easy to set up and use, but also capable of solving very complex problems in infrastructures; as will be explored in this walk through.

Salt is being used today by some of the largest infrastructures in the world and has a proven ability to scale to astounding proportions without modification. With the proven ability to scale out well beyond many tens of thousands of servers, Salt has also proven to be an excellent choice for small deployments as well, lowering compute and management overhead for infrastructures as small as just a few systems.

Installing Salt

SaltStack has been made to be very easy to install and get started. Setting up Salt should be as easy as installing Salt via distribution packages on Linux or via the Windows installer. The *installation documents* cover specific platform installation in depth.

Starting Salt

Salt functions on a master/minion topology. A master server acts as a central control bus for the clients (called minions), and the minions connect back to the master.

Setting Up the Salt Master Turning on the Salt Master is easy, just turn it on! The default configuration is suitable for the vast majority of installations. The Salt master can be controlled by the local Linux/Unix service manager:

On Systemd based platforms (OpenSuse, Fedora):

```
systemctl start salt-master
```

On Upstart based systems (Ubuntu, older Fedora/RHEL):

```
service salt-master start
```

On SysV Init systems (Debian, Gentoo etc.):

```
/etc/init.d/salt-master start
```

Or the master can be started directly on the command line:

```
salt-master -d
```

The Salt Master can also be started in the foreground in debug mode, thus greatly increasing the command output:

```
salt-master -l debug
```

The Salt Master needs to bind to 2 TCP network ports on the system, these ports are 4505 and 4506. For more in depth information on firewalling these ports, the firewall tutorial is available *here*.

Setting up a Salt Minion

Note: The Salt Minion can operate with or without a Salt Master. This walkthrough assumes that the minion will be connected to the master, for information on how to run a master-less minion please see the masterless quickstart guide:

Masterless Minion Quickstart

The Salt Minion only needs to be aware of one piece of information to run, the network location of the master. By default the minion will look for the DNS name `salt` for the master, making the easiest approach to set internal DNS to resolve the name `salt` back to the Salt Master IP. Otherwise the minion configuration file will need to be edited, edit the configuration option `master` to point to the DNS name or the IP of the Salt Master:

Note: The default location of the configuration files is `/etc/salt`. Most platforms adhere to this convention, but platforms such as FreeBSD and Microsoft Windows place this file in different locations.

`/etc/salt/minion:`

```
master: saltmaster.example.com
```

Now that the master can be found, start the minion in the same way as the master; with the platform init system, or via the command line directly:

As a daemon:

```
salt-minion -d
```

In the foreground in debug mode:

```
salt-minion -l debug
```

Now that the minion is started it will generate cryptographic keys and attempt to connect to the master. The next step is to venture back to the master server and accept the new minion's public key. When the minion is started, it will generate an `id` value, unless it has been generated on a previous run and cached in the configuration directory (`/etc/salt` by default). This is the name by which the minion will attempt to authenticate to the master. The following steps are attempted, in order to try to find a value that is not `localhost`:

1. The Python function `socket.getfqdn()` is run
2. `/etc/hostname` is checked (non-Windows only)
3. `/etc/hosts` (`%WINDIR%\system32\drivers\etc\hosts` on Windows hosts) is checked for hostnames that map to anything within **127.0.0.0/8**.

If none of the above are able to produce an id which is not `localhost`, then a sorted list of IP addresses on the minion (excluding any within **127.0.0.0/8**) is inspected. The first publicly-routable IP address is used, if there is one. Otherwise, the first privately-routable IP address is used.

If all else fails, then `localhost` is used as a fallback.

Note: Overriding the `id`

The minion id can be manually specified using the `id` parameter in the minion config file. If this configuration value is specified, it will override all other sources for the `id`.

Using salt-key Salt authenticates minions using public key encryption and authentication. For a minion to start accepting commands from the master the minion keys need to be accepted. The `salt-key` command is used to manage all of the keys on the master. To list the keys that are on the master run a salt-key list command:

```
salt-key -L
```

The keys that have been rejected, accepted and pending acceptance are listed. The easiest way to accept the minion key is to accept all pending keys:

```
salt-key -A
```

Note: Keys should be verified! The secure thing to do before accepting a key is to run `salt-key -p minion-id` to print the public key for the minion. This can then be compared against the minion's public key file, which is located (on the minion, of course) at `/etc/salt/pki/minion/minion.pub`.

On the master:

```
# salt-key -p foo.domain.com
Accepted Keys:
foo.domain.com:  -----BEGIN PUBLIC KEY-----
MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEA0JcA0IEp/yqghK5V2VLM
jbG7FWV6qtw/ubTDBnpDGQgrvSNOtd0QcJsAzAtDcHwrudQgyxTZGVJqPY7gLc7P
5b4EFWt5E1w3+KZ+XXy4YtW5oOzVN5BvsJ85g7c0TUnmjL7p3MUUXE4049Ue/zgX
jtbFJ0aa1HB8bnlQdWWOeflYRNEQL8482ZCmXXATFP1l5uJA9Pr6/ltdWtQTsXUA
bEseUGEpmq83vAkwtZIyJRG2cJh8ZRlJ6whSMg6wr7lFvStHQQzKHt9pRPml3lLK
ba2X07myAEJq/lpJNXJm5bkKV0+o8hqYQZ1ndh9HblHb2EoDBNbuIlhYft1uv8Tp
8beaEbq8ZST082sS/NjeL7W1T9JS6w2rw4GlUFuQlbqW8FSl1VDo+Alxu0VAr4GZ
gZpl2DgVoL59YDEVrlB464goly2c+eY4XkNT+JdwQ9LwMr83/yAAG6EGNpjT3pZg
Wey7WRnNTIF7H7ISwEzvik1GrhyBkn6K1RX3uAf760ZsQdhxwHmop+krgVcC0S93
xFjbBFF3+53mNv7BNPPgl0iwgA9/WuPE3aoE0A8Cm+Q6asZjf8P/h7KS67rIBEKV
zrQtgf3aZBbW38CT4fTzyWAP138yrU7VSGhPMm5KfTLywNsmXeaR5DnZl6GGNdL1
fZDM+J9FIGb/50Ee77saAlUCAwEAAQ==
-----END PUBLIC KEY-----
```

On the minion:

```
# cat /etc/salt/pki/minion/minion.pub
-----BEGIN PUBLIC KEY-----
MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEA0JcA0IEp/yqghK5V2VLM
jbG7FWV6qtw/ubTDBnpDGQgrvSNOtd0QcJsAzAtDcHwrudQgyxTZGVJqPY7gLc7P
5b4EFWt5E1w3+KZ+XXy4YtW5oOzVN5BvsJ85g7c0TUnmjL7p3MUUXE4049Ue/zgX
```

```
jtbFJ0aa1HB8bnlQdWWOeflYRNEQL8482ZCmXXATFP1l5uJA9Pr6/ltdWtQTsXUA
bEseUGEpmq83vAkwtZIyJRG2cJh8ZRlJ6whSMg6wr7lFvStHQQzKHt9pRPml3lLK
ba2X07myAEJq/lpJNXJm5bkKV0+o8hqYQZ1ndh9HblHb2EoDBNbuIlhYft1uv8Tp
8beaEbq8ZST082sS/NjeL7W1T9JS6w2rw4GlUFuQlbqW8FSl1VDo+Alxu0VAr4GZ
gZpl2DgVoL59YDEVrlB464goly2c+eY4XkNT+JdwQ9LwMr83/yAAG6EGNpjT3pZg
Wey7WRnNTIF7H7ISwEzvik1GrhyBkn6K1RX3uAf760ZsQdhxwHmop+krgVcC0S93
xFjbBFF3+53mNv7BNPPgl0iwgA9/WuPE3aoE0A8Cm+Q6asZjf8P/h7KS67rIBEKV
zrQtgf3aZBbW38CT4fTzyWAP138yrU7VSGhPMm5KfTLywNsmXeaR5DnZl6GGNdL1
fZDM+J9FIGb/50Ee77saAlUCAwEAAQ==
-----END PUBLIC KEY-----
```

Sending the First Commands Now that the minion is connected to the master and authenticated, the master can start to command the minion. Salt commands allow for a vast set of functions to be executed and for specific minions and groups of minions to be targeted for execution. This makes the `salt` command very powerful, but the command is also very usable, and easy to understand.

The `salt` command is comprised of command options, target specification, the function to execute, and arguments to the function. A simple command to start with looks like this:

```
salt '*' test.ping
```

The `*` is the target, which specifies all minions, and `test.ping` tells the minion to run the `test.ping` function. The result of running this command will be the master instructing all of the minions to execute `test.ping` in parallel and return the result. This is not an actual ICMP ping, but rather a simple function which returns `True`. Using `test.ping` is a good way of confirming that a minion is connected.

Note: Each minion registers itself with a unique minion id. This id defaults to the minion's hostname, but can be explicitly defined in the minion config as well by using the `id` parameter.

Getting to Know the Functions Salt comes with a vast library of functions available for execution, and Salt functions are self documenting. To see what functions are available on the minions execute the `sys.doc` function:

```
salt '*' sys.doc
```

This will display a very large list of available functions and documentation on them, this documentation is also available *here*.

These functions cover everything from shelling out to package management to manipulating database servers. They comprise a powerful system management API which is the backbone to Salt configuration management and many other aspects of Salt.

Note: Salt comes with many plugin systems. The functions that are available via the `salt` command are called *Execution Modules*.

Helpful Functions to Know The *cmd* module contains functions to shell out on minions, such as `cmd.run` and `cmd.run_all`:

```
salt '*' cmd.run 'ls -l /etc'
```

The `pkg` functions automatically map local system package managers to the same salt functions. This means that `pkg.install` will install packages via yum on Red Hat based systems, apt on Debian systems, etc.:

```
salt '*' pkg.install vim
```

Note: Some custom Linux spins and derivatives of other distros are not properly detected by Salt. If the above command returns an error message saying that `pkg.install` is not available, then you may need to override the pkg provider. This process is explained *here*.

The `network.interfaces` function will list all interfaces on a minion, along with their IP addresses, netmasks, MAC addresses, etc:

```
salt '*' network.interfaces
```

`salt-call` The examples so far have described running commands from the Master using the `salt` command, but when troubleshooting it can be more beneficial to login to the minion directly and use `salt-call`. Doing so allows you to see the minion log messages specific to the command you are running (which are *not* part of the return data you see when running the command from the Master using `salt`), making it unnecessary to tail the minion log. More information on `salt-call` and how to use it can be found *here*.

Grains Salt uses a system called *Grains* to build up static data about minions. This data includes information about the operating system that is running, CPU architecture and much more. The grains system is used throughout Salt to deliver platform data to many components and to users.

Grains can also be statically set, this makes it easy to assign values to minions for grouping and managing. A common practice is to assign grains to minions to specify what the role or roles a minion might be. These static grains can be set in the minion configuration file or via the `grains.setval` function.

Targeting Salt allows for minions to be targeted based on a wide range of criteria. The default targeting system uses globular expressions to match minions, hence if there are minions named `larry1`, `larry2`, `curly1` and `curly2`, a glob of `larry*` will match `larry1` and `larry2`, and a glob of `*1` will match `larry1` and `curly1`.

Many other targeting systems can be used other than globs, these systems include:

Regular Expressions Target using PCRE compliant regular expressions

Grains Target based on grains data: *Targeting with Grains*

Pillar Target based on pillar data: *Targeting with Pillar*

IP Target based on IP addr/subnet/range

Compound Create logic to target based on multiple targets: *Targeting with Compound*

Nodegroup Target with nodegroups: *Targeting with Nodegroup*

The concepts of targets are used on the command line with salt, but also function in many other areas as well, including the state system and the systems used for ACLs and user permissions.

Passing in Arguments Many of the functions available accept arguments, these arguments can be passed in on the command line:

```
salt '*' pkg.install vim
```

This example passes the argument `vim` to the pkg.install function, since many functions can accept more complex input then just a string the arguments are parsed through YAML, allowing for more complex data to be sent on the command line:

```
salt '*' test.echo 'foo: bar'
```

In this case Salt translates the string 'foo: bar' into the dictionary "{'foo': 'bar'}"

Note: Any line that contains a newline will not be parsed by yaml.

Salt States

Now that the basics are covered the time has come to evaluate `States`. Salt `States`, or the `State System` is the component of Salt made for configuration management. The State system is a fully functional configuration management system which has been designed to be exceptionally powerful while still being simple to use, fast, lightweight, deterministic and with salty levels of flexibility.

The state system is already available with a basic salt setup, no additional configuration is required, states can be set up immediately.

Note: Before diving into the state system, a brief overview of how states are constructed will make many of the concepts clearer. Salt states are based on data modeling, and build on a low level data structure that is used to execute each state function. Then more logical layers are built on top of each other. The high layers of the state system which this tutorial will cover consists of everything that needs to be known to use states, the two high layers covered here are the *sls* layer and the highest layer *highstate*.

Again, knowing that there are many layers of data management, will help with understanding states, but they never need to be used. Just as understanding how a compiler functions when learning a programming language, understanding what is going on under the hood of a configuration management system will also prove to be a valuable asset.

The First SLS Formula

The state system is built on sls formulas, these formulas are built out in files on Salt's file server. To make a very basic sls formula open up a file under /srv/salt named vim.sls and get vim installed:

`/srv/salt/vim.sls:`

```
vim:
  pkg.installed
```

Now install vim on the minions by calling the sls directly:

```
salt '*' state.sls vim
```

This command will invoke the state system and run the named sls which was just created, `vim`.

Now, to beef up the vim sls formula, a vimrc can be added:

`/srv/salt/vim.sls:`

```
vim:
  pkg.installed

/etc/vimrc:
  file.managed:
    - source: salt://vimrc
    - mode: 644
    - user: root
    - group: root
```

Now the desired vimrc needs to be copied into the Salt file server to /srv/salt/vimrc, in Salt everything is a file, so no path redirection needs to be accounted for. The vimrc file is placed right next to the vim.sls file. The same command as above can be executed to all the vim sls formulas and now include managing the file.

Note: Salt does not need to be restarted/reloaded or have the master manipulated in any way when changing sls formulas, they are instantly available.

Adding Some Depth

Obviously maintaining sls formulas right in the root of the file server will not scale out to reasonably sized deployments. This is why more depth is required. Start by making an nginx formula a better way, make an nginx subdirectory and add an init.sls file:

`/srv/salt/nginx/init.sls:`

```
nginx:
  pkg:
    - installed
  service:
    - running
    - require:
      - pkg: nginx
```

A few things are introduced in this sls formula, first is the service statement which ensures that the nginx service is running, but the nginx service can't be started unless the package is installed, hence the `require`. The `require` statement makes sure that the required component is executed before and that it results in success.

Note: The *require* option belongs to a family of options called *requisites*. Requisites are a powerful component of Salt States, for more information on how requisites work and what is available see: *Requisites* Also evaluation ordering is available in Salt as well: *Ordering States*

Now this new sls formula has a special name, `init.sls`, when an sls formula is named `init.sls` it inherits the name of the directory path that contains it, so this formula can be referenced via the following command:

```
salt '*' state.sls nginx
```

Now that subdirectories can be used the vim.sls formula can be cleaned up, but to make things more flexible (and to illustrate another point of course), move the vim.sls and vimrc into a new subdirectory called `edit` and change the vim.sls file to reflect the change:

`/srv/salt/edit/vim.sls:`

```
vim:
  pkg.installed

/etc/vimrc:
  file.managed:
    - source: salt://edit/vimrc
    - mode: 644
    - user: root
    - group: root
```

The only change in the file is fixing the source path for the vimrc file. Now the formula is referenced as `edit.vim` because it resides in the edit subdirectory. Now the edit subdirectory can contain formulas for emacs, nano, joe or any other editor that may need to be deployed.

Next Reading

Two walkthroughs are specifically recommended at this point. First, a deeper run through States, followed by an explanation of Pillar.

1. *Starting States*
2. *Pillar Walkthrough*

An understanding of Pillar is extremely helpful in using States.

Getting Deeper Into States

Two more in-depth States tutorials exist, which delve much more deeply into States functionality.

1. Thomas' original states tutorial, *How Do I Use Salt States?*, covers much more to get off the ground with States.
2. The *States Tutorial* also provides a fantastic introduction.

These tutorials include much more in depth information including templating sls formulas etc.

So Much More!

This concludes the initial Salt walkthrough, but there are many more things to learn still! These documents will cover important core aspects of Salt:

- *Pillar*
- *Job Management*

A few more tutorials are also available:

- *Remote Execution Tutorial*
- *Standalone Minion*

This still is only scratching the surface, many components such as the reactor and event systems, extending Salt, modular components and more are not covered here. For an overview of all Salt features and documentation, look at the *Table of Contents*.

7.4.2 MinionFS Backend Walkthrough

New in version 2014.1.0: (Hydrogen)

Sometimes, you might need to propagate files that are generated on a minion. Salt already has a feature to send files from a minion to the master:

```
salt 'minion-id' cp.push /path/to/the/file
```

This command will store the file, including its full path, under `cachedir` `/master/minions/minion-id/files`. With the default `cachedir` the example file above would be stored as */var/cache/salt/master/minions/minion-id/files/path/to/the/file*.

Note: This walkthrough assumes basic knowledge of Salt and `cp.push`. To get up to speed, check out the *walkthrough*.

Since it is not a good idea to expose the whole `cachedir`, MinionFS should be used to send these files to other minions.

Simple Configuration

To use the minionfs backend only two configuration changes are required on the master. The `fileserver_backend` option needs to contain a value of `minion` and `file_recv` needs to be set to true:

```
fileserver_backend:
  - roots
  - minion

file_recv: True
```

These changes require a restart of the master, then new requests for the `salt://minion-id/` protocol will send files that are pushed by `cp.push` from `minion-id` to the master.

Note: All of the files that are pushed to the master are going to be available to all of the minions. If this is not what you want, please remove `minion` from `fileserver_backend` in the master config file.

Note: Having directories with the same name as your minions in the root that can be accessed like `salt://minion-id/` might cause confusion.

Commandline Example

Lets assume that we are going to generate SSH keys on a minion called `minion-source` and put the public part in `~/.ssh/authorized_keys` of root user of a minion called `minion-destination`.

First, lets make sure that `/root/.ssh` exists and has the right permissions:

```
[root@salt-master file]# salt '*' file.mkdir dir_path=/root/.ssh user=root group=root mode=700
minion-source:
    None
minion-destination:
    None
```

We create an RSA key pair without a passphrase [1]:

```
[root@salt-master file]# salt 'minion-source' cmd.run 'ssh-keygen -N "" -f /root/.ssh/id_rsa'
minion-source:
    Generating public/private rsa key pair.
    Your identification has been saved in /root/.ssh/id_rsa.
    Your public key has been saved in /root/.ssh/id_rsa.pub.
    The key fingerprint is:
    9b:cd:1c:b9:c2:93:8e:ad:a3:52:a0:8b:0a:cc:d4:9b root@minion-source
    The key's randomart image is:
    +--[ RSA 2048]----+
    |                 |
    |                 |
    |                 |
    |  o        .     |
    | o o    S o      |
    |=   +  . B o     |
    |o+ E    B =      |
    |+ .   .+ o       |
    |o  ...ooo        |
    +-----------------+
```

---

[1] Yes, that was the actual key on my server, but the server is already destroyed.

and we send the public part to the master to be available to all minions:

```
[root@salt-master file]# salt 'minion-source' cp.push /root/.ssh/id_rsa.pub
minion-source:
    True
```

now it can be seen by everyone:

```
[root@salt-master file]# salt 'minion-destination' cp.list_master_dirs
minion-destination:
    - .
    - etc
    - minion-source/root
    - minion-source/root/.ssh
```

Lets copy that as the only authorized key to `minion-destination`:

```
[root@salt-master file]# salt 'minion-destination' cp.get_file salt://minion-source/root/.ssh/id_rsa
minion-destination:
    /root/.ssh/authorized_keys
```

Or we can use a more elegant and salty way to add an SSH key:

```
[root@salt-master file]# salt 'minion-destination' ssh.set_auth_key_from_file user=root source=salt:/
minion-destination:
    new
```

7.4.3 Automatic Updates / Frozen Deployments

New in version 0.10.3.d.

Salt has support for the Esky application freezing and update tool. This tool allows one to build a complete zipfile out of the salt scripts and all their dependencies - including shared objects / DLLs.

Getting Started

To build frozen applications, you'll need a suitable build environment for each of your platforms. You should probably set up a virtualenv in order to limit the scope of Q/A.

This process does work on Windows. Follow the directions at https://github.com/saltstack/salt-windows-install for details on installing Salt in Windows. Only the 32-bit Python and dependencies have been tested, but they have been tested on 64-bit Windows.

You will need to install `esky` and `bbfreeze` from Pypi in order to enable the `bdist_esky` command in `setup.py`.

Building and Freezing

Once you have your tools installed and the environment configured, you can then `python setup.py bdist` to get the eggs prepared. After that is done, run `python setup.py bdist_esky` to have Esky traverse the module tree and pack all the scripts up into a redistributable. There will be an appropriately versioned `salt-VERSION.zip` in `dist/` if everything went smoothly.

Windows

You will need to add `C:\Python27\lib\site-packages\zmq` to your PATH variable. This helps bbfreeze find the zmq dll so it can pack it up.

Using the Frozen Build

Unpack the zip file in your desired install location. Scripts like `salt-minion` and `salt-call` will be in the root of the zip file. The associated libraries and bootstrapping will be in the directories at the same level. (Check the Esky documentation for more information)

To support updating your minions in the wild, put your builds on a web server that your minions can reach. `salt.modules.saltutil.update()` will trigger an update and (optionally) a restart of the minion service under the new version.

Gotchas

My Windows minion isn't responding

The process dispatch on Windows is slower than it is on *nix. You may need to add '-t 15' to your salt calls to give them plenty of time to return.

Windows and the Visual Studio Redist

You will need to install the Visual C++ 2008 32-bit redistributable on all Windows minions. Esky has an option to pack the library into the zipfile, but OpenSSL does not seem to acknowledge the new location. If you get a `no OPENSSL_Applink` error on the console when trying to start your frozen minion, you have forgotten to install the redistributable.

Mixed Linux environments and Yum

The Yum Python module doesn't appear to be available on any of the standard Python package mirrors. If you need to support RHEL/CentOS systems, you should build on that platform to support all your Linux nodes. Also remember to build your virtualenv with `--system-site-packages` so that the `yum` module is included.

Automatic (Python) module discovery

Automatic (Python) module discovery does not work with the late-loaded scheme that Salt uses for (Salt) modules. You will need to explicitly add any misbehaving modules to the `freezer_includes` in Salt's `setup.py`. Always check the zipped application to make sure that the necessary modules were included.

7.4.4 Multi Master Tutorial

As of Salt 0.16.0, the ability to connect minions to multiple masters has been made available. The multi-master system allows for redundancy of Salt masters and facilitates multiple points of communication out to minions. When using a multi-master setup, all masters are running hot, and any active master can be used to send commands out to the minions.

In 0.16.0, the masters do not share any information, keys need to be accepted on both masters, and shared files need to be shared manually or use tools like the git fileserver backend to ensure that the `file_roots` are kept consistent.

Summary of Steps

1. Create a redundant master server
2. Copy primary master key to redundant master
3. Start redundant master
4. Configure minions to connect to redundant master
5. Restart minions
6. Accept keys on redundant master

Prepping a Redundant Master

The first task is to prepare the redundant master. There is only one requirement when preparing a redundant master, which is that masters share the same private key. When the first master was created, the master's identifying key was generated and placed in the master's `pki_dir`. The default location of the key is `/etc/salt/pki/master/master.pem`. Take this key and copy it to the same location on the redundant master. Assuming that no minions have yet been connected to the new redundant master, it is safe to delete any existing key in this location and replace it.

Note: There is no logical limit to the number of redundant masters that can be used.

Once the new key is in place, the redundant master can be safely started.

Configure Minions

Since minions need to be master-aware, the new master needs to be added to the minion configurations. Simply update the minion configurations to list all connected masters:

```
master:
  - saltmaster1.example.com
  - saltmaster2.example.com
```

Now the minion can be safely restarted.

Now the minions will check into the original master and also check into the new redundant master. Both masters are first-class and have rights to the minions.

Sharing Files Between Masters

Salt does not automatically share files between multiple masters. A number of files should be shared or sharing of these files should be strongly considered.

Minion Keys

Minion keys can be accepted the normal way using **salt-key** on both masters. Keys accepted, deleted, or rejected on one master will NOT be automatically managed on redundant masters; this needs to be taken care of by running salt-key on both masters or sharing the `/etc/salt/pki/master/{minions,minions_pre,minions_rejected}` directories between masters.

Note: While sharing the **/etc/salt/pki/master** directory will work, it is strongly discouraged, since allowing access to the **master.pem** key outside of Salt creates a *SERIOUS* security risk.

File_Roots

The `file_roots` contents should be kept consistent between masters. Otherwise state runs will not always be consistent on minions since instructions managed by one master will not agree with other masters.

The recommended way to sync these is to use a fileserver backend like gitfs or to keep these files on shared storage.

Pillar_Roots

Pillar roots should be given the same considerations as `file_roots`.

Master Configurations

While reasons may exist to maintain separate master configurations, it is wise to remember that each master maintains independent control over minions. Therefore, access controls should be in sync between masters unless a valid reason otherwise exists to keep them inconsistent.

These access control options include but are not limited to:

- external_auth
- client_acl
- peer
- peer_run

7.4.5 Preseed Minion with Accepted Key

In some situations, it is not convenient to wait for a minion to start before accepting its key on the master. For instance, you may want the minion to bootstrap itself as soon as it comes online. You may also want to to let your developers provision new development machines on the fly.

There is a general four step process to do this:

1. Generate the keys on the master:

```
root@saltmaster# salt-key --gen-keys=[key_name]
```

Pick a name for the key, such as the minion's id.

2. Add the public key to the accepted minion folder:

```
root@saltmaster# cp key_name.pub /etc/salt/pki/master/minions/[minion_id]
```

It is necessary that the public key file has the same name as your minion id. This is how Salt matches minions with their keys. Also note that the pki folder could be in a different location, depending on your OS or if specified in the master config file.

3. Distribute the minion keys.

There is no single method to get the keypair to your minion. The difficulty is finding a distribution method which is secure.

Security Warning

Since the minion key is already accepted on the master, distributing the private key poses a potential security risk. A malicious party will have access to your entire state tree and other sensitive data if they gain access to a preseeded minion key.

---

4. Preseed the Minion with the keys

You will want to place the minion keys before starting the salt-minion daemon:

```
/etc/salt/pki/minion/minion.pem
/etc/salt/pki/minion/minion.pub
```

Once in place, you should be able to start salt-minion and run `salt-call state.highstate` or any other salt commands that require master authentication.

7.4.6 Bootstrapping Salt on Linux EC2 with Cloud-Init

Salt is a great tool for remote execution and configuration management, however you will still need to bootstrap the daemon when spinning up a new node. One option is to create and save a custom AMI, but this creates another resource to maintain and document.

A better method for Linux machines uses Canonical's CloudInit to run a bootstrap script during an EC2 Instance initialization. Cloud-init takes the `user_data` string passed into a new AWS instance and runs it in a manner similar to rc.local. The bootstrap script needs to:

1. Install Salt with dependencies
2. Point the minion to the master

Here is a sample script:

```
#!/bin/bash

# Install saltstack
add-apt-repository ppa:saltstack/salt -y
apt-get update -y
apt-get install salt-minion -y
apt-get install salt-master -y
apt-get upgrade -y

# Set salt master location and start minion
sed -i 's/#master: salt/master: [salt_master_fqdn]/' /etc/salt/minion
salt-minion -d
```

First the script adds the saltstack ppa and installs the package. Then we copy over the minion config template and tell it where to find the master. You will have to replace `[salt_master_fqdn]` with something that resolves to your Salt master.

Used With Boto

Boto will accept a string for user data which can be used to pass our bootstrap script. If the script is saved to a file, you can read it into a string:

```
import boto

user_data = open('salt_bootstrap.sh')

conn = boto.connect_ec2(<AWS_ACCESS_ID>, <AWS_SECRET_KEY>)
```

```
reservation = conn.run_instances(image_id=<ami_id>,
                                 key_name=<key_name>,
                                 user_data=user_data.read())
```

Additional Notes

Sometime in the future the ppa will include and install an upstart file. In the meantime, you can use the bootstrap to build one.

It may also be useful to set the node's role during this phase. One option would be saving the node's role to a file and then using a custom Grain to select it.

7.4.7 Salt Bootstrap

The Salt Bootstrap script allows for a user to install the Salt Minion or Master on a variety of system distributions and versions. This shell script known as `bootstrap-salt.sh` runs through a series of checks to determine the operating system type and version. It then installs the Salt binaries using the appropriate methods. The Salt Bootstrap script installs the minimum number of packages required to run Salt. This means that in the event you run the bootstrap to install via package, Git will not be installed. Installing the minimum number of packages helps ensure the script stays as lightweight as possible, assuming the user will install any other required packages after the Salt binaries are present on the system. The script source is available on GitHub: https://github.com/saltstack/salt-bootstrap

Supported Operating Systems

- Amazon Linux 2012.09
- Arch
- CentOS 5/6
- Debian 6.x/7.x
- Fedora 17/18
- FreeBSD 9.1
- Gentoo
- Linaro
- Linux Mint 13/14
- OpenSUSE 12.x
- Red Hat 5/6
- Red Hat Enterprise 5/6
- SmartOS
- SuSE 11 SP1/11 SP2
- Ubuntu 10.x/11.x/12.x/13.04

Note: In the event you do not see your distribution or version available please review the develop branch on Github as it main contain updates that are not present in the stable release: https://github.com/saltstack/salt-bootstrap/tree/develop

Example Usage

The Salt Bootstrap script has a wide variety of options that can be passed as well as several ways of obtaining the bootstrap script itself.

For example, using `curl` to install latest git:

```
curl -L http://bootstrap.saltstack.org | sudo sh -s -- git develop
```

If you have certificate issues using `curl`, try the following:

```
curl --insecure -L http://bootstrap.saltstack.org | sudo sh -s -- git develop
```

Using `wget` to install your distribution's stable packages:

```
wget -O - http://bootstrap.saltstack.org | sudo sh
```

If you have certificate issues using `wget` try the following:

```
wget --no-check-certificate -O - http://bootstrap.saltstack.org | sudo sh
```

Install a specific version from git using `wget`:

```
wget -O - http://bootstrap.saltstack.org | sh -s -- -P git v0.16.4
```

If you already have python installed, `python 2.6`, then it's as easy as:

```
python -m urllib "http://bootstrap.saltstack.org" | sudo sh -s -- git develop
```

All python versions should support the following one liner:

```
python -c 'import urllib; print urllib.urlopen("http://bootstrap.saltstack.org").read()' | \
sudo  sh -s -- git develop
```

On a FreeBSD base system you usually don't have either of the above binaries available. You **do** have `fetch` available though:

```
fetch -o - http://bootstrap.saltstack.org | sudo sh
```

If all you want is to install a `salt-master` using latest git:

```
curl -L http://bootstrap.saltstack.org | sudo sh -s -- -M -N git develop
```

If you want to install a specific release version (based on the git tags):

```
curl -L http://bootstrap.saltstack.org | sudo sh -s -- git v0.16.4
```

Downloading the develop branch (from here standard command line options may be passed):

```
wget https://raw.github.com/saltstack/salt-bootstrap/develop/bootstrap-salt.sh
```

Command Line Options

-h Display the help message and command line options.

-v Display script version.

-n No colours.

-D Show debug output.

-c Temporary configuration directory.

-k Temporary directory holding the minion keys which will pre-seed the master.

-M Also install salt-master.

-S Also install salt-syndic.

-N Do not install salt-minion.

-X Do not start daemons after installation.

-C Only run the configuration function. This option automatically bypasses any installation.

-P Allow pip based installations. On some distributions the required salt packages or its dependencies are not available as a package for that distribution. Using this flag allows the script to use pip as a last resort method.

Note: This works for functions which actually implement pip based installations.

-F Allow copied files to overwrite existing(config, init.d, etc).

-U If set, fully upgrade the system prior to bootstrapping salt.

-K If set, keep the temporary files in the temporary directories specified with -c and -k.

7.4.8 GitFS Backend Walkthrough

While the default location of the salt state tree is on the Salt master, in /srv/salt, the master can create a bridge to external resources for files. One of these resources is the ability for the master to directly pull files from a git repository and serve them to minions.

Note: This walkthrough assumes basic knowledge of Salt. To get up to speed, check out the *walkthrough*.

The gitfs backend hooks into any number of remote git repositories and caches the data from the repository on the master. This makes distributing a state tree to multiple masters seamless and automated.

Salt's file server also has a concept of environments, when using the gitfs backend, Salt translates git branches and tags into environments, making environment management very simple. Just merging a QA or staging branch up to a production branch can be all that is required to make those file changes available to Salt.

Simple Configuration

To use the gitfs backend only two configuration changes are required on the master. The `fileserver_backend` option needs to be set with a value of `git`:

```
fileserver_backend:
  - git
```

To configure what fileserver backends will be searched for requested files.

Now the gitfs system needs to be configured with a remote:

```
gitfs_remotes:
  - git://github.com/saltstack/salt-states.git
```

Note: The salt-states repo is not currently updated with the latest versions of the available states. Please review https://github.com/saltstack-formulas for the latest versions.

These changes require a restart of the master, then the git repo will be cached on the master and new requests for the `salt://` protocol will send files found in the remote git repository via the master.

Note: The master caches the files from the git server and serves them out, minions do not connect directly to the git server meaning that only requested files are delivered to minions.

Multiple Remotes

The `gitfs_remotes` option can accept a list of git remotes, the remotes are then searched in order for the requested file. A simple scenario can illustrate this behavior.

Assuming that the `gitfs_remotes` option specifies three remotes:

```
gitfs_remotes:
  - git://github.com/example/first.git
  - git://github.com/example/second.git
  - file:///root/third
```

Note: This example is purposefully contrived to illustrate the behavior of the gitfs backend. This example should not be read as a recommended way to lay out files and git repos.

Note: The **file://** prefix denotes a git repository in a local directory. However, it will still use the given **file://** URL as a remote, rather than copying the git repo to the salt cache. This means that any refs you want accessible must exist as *local* refs in the specified repo.

Assume that each repository contains some files:

```
first.git:
    top.sls
    edit/vim.sls
    edit/vimrc
    nginx/init.sls

second.git:
    edit/dev_vimrc
    haproxy/init.sls

third:
    haproxy/haproxy.conf
    edit/dev_vimrc
```

The repositories will be searched for files by the master in the order in which they are defined in the configuration, Therefore the remote **git://github.com/example/first.git** will be searched first, if the requested file is found then it is served and no further searching is executed. This means that if the file **salt://haproxy/init.sls** is requested then it will be pulled from the **git://github.com/example/second.git** git repo. If **salt://haproxy/haproxy.conf** is requested then it will be pulled from the third repo.

Serving from a Subdirectory

The `gitfs_root` option gives the ability to serve files from a subdirectory within the repository. The path is defined relative to the root of the repository.

With this repository structure:

```
repository.git:
    somefolder
        otherfolder
            top.sls
            edit/vim.sls
            edit/vimrc
            nginx/init.sls
```

Configuration and files can be accessed normally with:

```
gitfs_root: somefolder/otherfolder
```

Multiple Backends

Sometimes it may make sense to use multiple backends. For instance, if sls files are stored in git, but larger files need to be stored directly on the master.

The logic used for multiple remotes is also used for multiple backends. If the `fileserver_backend` option contains multiple backends:

```
fileserver_backend:
  - roots
  - git
```

Then the `roots` backend (the default backend of files in `/srv/salt`) will be searched first for the requested file, then if it is not found on the master the git remotes will be searched.

Branches, environments and top.sls files

As stated above, when using the `gitfs` backend, branches will be mapped to environments using the branch name as identifier. There is an exception to this rule thought: the `master` branch is implicitly mapped to the `base` environment. Therefore, for a typical `base`, `qa`, `dev` setup, you'll have to create the following branches:

```
master
qa
dev
```

Also, `top.sls` files from different branches will be merged into one big file at runtime. Since this could lead to hardly manageable configurations, the recommended setup is to have the `top.sls` file only in your master branch, and use environment-specific branches for states definitions.

GitFS Remotes over SSH

In order to configure a `gitfs_remotes` repository over SSH transport the `git+ssh` URL form must be used.

```
gitfs_remotes:
  - git+ssh://git@github.com/example/salt-states.git
```

The private key used to connect to the repository must be located in `~/.ssh/id_rsa` for the user running the salt-master.

Note: GitFS requires the Python module `GitPython`, version 0.3.0 or newer. If your Master runs Ubuntu 12.04 LTS, you will likely need to install GitPython using pip.

```
# pip install GitPython
```

Using Git as an External Pillar Source

Git repositories can also be used to provide *Pillar* data, using the *External Pillar* system. To define a git external pillar, you can add a section like the following to your master config file:

```
ext_pillar:
  - git: <branch> <repo>
```

The `<branch>` param is the branch containing the pillar SLS tree, and the `<repo>` param is the URI for the repository. The below example would add the `master` branch of the specified repo as an external pillar source.

```
ext_pillar:
  - git: master https://domain.com/pillar.git
```

More information on the git external pillar can be found `here`.

Why aren't my custom modules/states/etc. syncing to my Minions?

In versions 0.16.3 and older, when using the *git fileserver backend*, certain versions of GitPython may generate errors when fetching, which Salt fails to catch. While not fatal to the fetch process, these interrupt the fileserver update that takes place before custom types are synced, and thus interrupt the sync itself. Try disabling the git fileserver backend in the master config, restarting the master, and attempting the sync again.

This issue is worked around in Salt 0.16.4 and newer.

7.4.9 The MacOS X (Maverick) Developer Step By Step Guide To Salt Installation

This document provides a step by step guide to installing a salt cluster (one master, and one minion running on a local VM) on Mac OS X to start playing with salt. It is aimed at developers, so it may be a little too slow for experienced admins. At the end of it, you will have installed a bare Nginx server on a minion using the master. The official (Linux) walkthrough can be found here.

The 5 Cent Salt Intro

Since you're here you've probably already heard about Salt, so you already know Salt lets you configure and run commands on hordes of servers easily. Here's a brief overview of a Salt cluster:

- Salt works by having a "master" server sending commands to one or multiple "minion" servers [2]. The master server is the "command center". It is going to be the place where you store your configuration files, aka: "which server is the db, which is the web server, and what libraries and software they should have installed". The minions receive orders from the master. Minions are the servers actually performing work for your business.
- Salt has two types of configuration files:

  1. the "salt communication channels" or "meta" or "config" configuration files (not official names): one for the master (usually is /etc/salt/master , **on the master server**), and one for minions (default is /etc/salt/minion or /etc/salt/minion.conf, **on the minion servers**). Those files are used to determine things like the Salt Master IP, port, Salt folder locations, etc.. If these are configured incorrectly, your minions will probably be unable to receive orders from the master, or the master will not know which software a given minion should install.

---

[2] Salt also works with "masterless" configuration where a minion is autonomous (in which case salt can be seen as a local configuration tool), or in "multiple master" configuration. See the documentation for more on that.

2. the "business" or "service" configuration files (once again, not an official name): these are configuration files, ending with ".sls" extension, that describe which software should run on which server, along with particular configuration properties for the software that is being installed. These files should be created in the /srv/salt folder by default, but their location can be changed using ... /etc/salt/master configuration file!

Note: This tutorial contains a third important configuration file, not to be confused with the previous two: the virtual machine provisioning configuration file. This in itself is not specifically tied to Salt, but it also contains some Salt configuration. More on that in step 3. Also note that all configuration files are YAML files. So indentation matters.

Before Digging In, The Architecture Of The Salt Cluster

Salt Master

The "Salt master" server is going to be the Mac OS machine, directly. Commands will be run from a terminal app, so Salt will need to be installed on the Mac. This is going to be more convenient for toying around with configuration files.

Salt Minion

We'll only have one "Salt minion" server. It is going to be running on a Virtual Machine running on the Mac, using VirtualBox. It will run an Ubuntu distribution.

7.4.10 Step 1 - Configuring The Salt Master On Your Mac

official documentation

Because Salt has a lot of dependencies that are not built in Mac OS X, we will use Homebrew to install Salt. Homebrew is a package manager for Mac, it's great, use it (for this tutorial at least!). Some people spend a lot of time installing libs by hand to better understand dependencies, and then realize how useful a package manager is once they're configuring a brand new machine and have to do it all over again. It also lets you *uninstall* things easily.

Note: Brew is a Ruby program (Ruby is installed by default with your Mac). Brew downloads, compile and links software. The linking phase is when compiled software is deployed on your machine. It may conflict with manually installed software, especially in the /usr/local directory. It's ok, remove the manually installed version then refresh the link by typing `brew link 'packageName'`. Brew has a `brew doctor` command that can help you troubleshoot. It's a great command, use it often. Brew requires xcode command line tools. When you run brew the first time it asks you to install them if they're not already on your system. Brew installs software in /usr/local/bin (system bins are in /usr/bin). In order to use those bins you need your $PATH to search there first. Brew tells you if your $PATH needs to be fixed.

Tip: Use the keyboard shortcut `cmd + shift + period` in the "open" Mac OS X dialog box to display hidden files and folders, such as .profile.

Install Homebrew

Install Homebrew here http://brew.sh/ Or just type

```
ruby -e "$(curl -fsSL https://raw.github.com/mxcl/homebrew/go)"
```

Now type the following commands in your terminal (you may want to type brew doctor after each to make sure everything's fine):

```
brew install python
brew install swig
brew install zmq
```

Note: zmq is ZeroMQ. It's a fantastic library used for server to server network communication and is at the core of Salt efficiency.

Install Salt

you should now have everything ready to launch this command:

```
pip install salt
```

Note: There should be no need for `sudo pip install salt`. Brew installed Python for your user, so you should have all the access. In case you would like to check, type `which python` to ensure that it's /usr/local/bin/python, and `which pip` which should be /usr/local/bin/pip.

Now type `python` in a terminal then, `import salt`. There should be no errors. Now exit the Python terminal using `exit()`.

Create The Master Configuration

If the default /etc/salt/master configuration file was not created, copy-paste it from here: http://docs.saltstack.com/ref/configuration/examples.html#configuration-examples-master

---

Note: `/etc/salt/master` is a file, not a folder.

Salt Master configuration changes. The Salt master needs a few customization to be able to run on Mac OS X:

```
sudo launchctl limit maxfiles 4096 8192
```

In the /etc/salt/master file, change max_open_files to 8192 (or just add the line: `max_open_files: 8192` (no quote) if it doesn't already exists).

You should now be able to launch the Salt master:

```
sudo salt-master --log-level=all
```

There should be no errors when running the above command.

Note: This command is supposed to be a daemon, but for toying around, we'll keep it running on a terminal to monitor the activity.

Now that the master is set, let's configure a minion on a VM.

7.4.11 Step 2 - Configuring The Minion VM

The Salt minion is going to run on a Virtual Machine. There are a lot of software options that let you run virtual machines on a mac, But for this tutorial we're going to use VirtualBox. In addition to virtualBox, we will use Vagrant, which allows you to create the base VM configuration.

Vagrant lets you build ready to use VM images, starting from an OS image and customizing it using "provisioners". In our case, we'll use it to:

- Download the base Ubuntu image
- Install salt on that Ubuntu image (Salt is going to be the "provisioner" for the VM).
- Launch the VM
- SSH into the VM to debug
- Stop the VM once you're done.

Install VirtualBox

Go get it here: https://www.virtualBox.org/wiki/Downloads (click on VirtualBox for OS X hosts => x86/amd64)

Install Vagrant

Go get it here: http://downloads.vagrantup.com/ and choose the latest version (1.3.5 at time of writing), then the .dmg file. double-click to install it. Make sure the `vagrant` command is found when run in the terminal. Type `vagrant`. It should display a list of commands.

Create The Minion VM Folder

Create a folder in which you will store your minion's VM. In this tutorial, it's going to be a minion folder in the $home directory.

```
cd $home
mkdir minion
```

Initialize Vagrant

From the minion folder, type

```
vagrant init
```

This command creates a default Vagrantfile configuration file. This configuration file will be used to pass configuration parameters to the Salt provisioner in Step 3.

Import Precise64 Ubuntu Box

```
vagrant box add precise64 http://files.vagrantup.com/precise64.box
```

Note: This box is added at the global Vagrant level. You only need to do it once as each VM will use this same file.

Modify the Vagrantfile

Modify ./minion/Vagrantfile to use th precise64 box. Change the `config.vm.box` line to:

```
config.vm.box = "precise64"
```

Uncomment the line creating a host-only IP. This is the ip of your minion (you can change it to something else if that IP is already in use):

```
config.vm.network :private_network, ip: "192.168.33.10"
```

At this point you should have a VM that can run, although there won't be much in it. Let's check that.

Checking The VM

From the $home/minion folder type:

```
vagrant up
```

A log showing the VM booting should be present. Once it's done you'll be back to the terminal:

```
ping 192.168.33.10
```

The VM should respond to your ping request.

Now log inside the VM in ssh using Vagrant again:

```
vagrant ssh
```

You should see the shell prompt changing to something similar to `vagrant@precise64:~$` meaning you're inside the VM. From there enter the following:

```
ping 10.0.2.2
```

Note: That ip is the ip of your VM host (the Mac OS X OS). The number is a VirtualBox default and is displayed in the log after the Vagrant ssh command. We'll use that IP to tell the minion where the Salt master is. Once you're done, end the ssh session by typing `exit`.

It's now time to connect the VM to the salt master

7.4.12 Step 3 - Connecting Master and Minion

Creating The Minion Configuration File

Create the `/etc/salt/minion` file. In that file, put the following lines, giving the ID for this minion, and the IP of the master:

```
master: 10.0.2.2
id: 'minion1'
file_client: remote
```

Minions authenticate with the master using keys. Keys are generated automatically if you don't provide one, and you can accept them later on. But this requires you to accept the minion key every time you destroy and recreate a minion (which could be quite often). A better way is to create those keys in advance, feed them to the minion, and authorize them once.

Preseed minion keys

From the minion folder on your Mac run:

```
sudo salt-key --gen-keys=minion1
```

This should create two files: minion1.pem, and minion1.pub. Since those files have been created by sudo, but will be used by vagrant, you need to change ownership:

```
sudo chown youruser:yourgroup minion1.pem
sudo chown youruser:yourgroup minion1.pub
```

Then copy the .pub file into the list of accepted minions:

```
sudo cp minion1.pub /etc/salt/pki/master/minions/minion1
```

Modify Vagrantfile to Use Salt Provisioner

Let's now modify the Vagrantfile used to provision the Salt VM. Add the following section in the Vagrantfile (note: it should be at the same indentation level as the other properties):

```
# salt-vagrant config
config.vm.provision :salt do |salt|
    salt.run_highstate = true
    salt.minion_config = "./minion.conf"
    salt.minion_key = "./minion1.pem"
    salt.minion_pub = "./minion1.pub"
end
```

Now destroy the vm and recreate it from the /minion folder:

```
vagrant destroy
vagrant up
```

If everything is fine you should see the following message:

```
"Bootstrapping Salt... (this may take a while)
Salt successfully configured and installed!"
```

Checking Master-Minion Communication

To make sure the master and minion are talking to each other, enter the following:

```
sudo salt '*' test.ping
```

You should see your minion answering the ping. It's now time to do some configuration.

7.4.13 Step 4 - Configure Services to Install On the Minion

In this step we'll use the Salt master to instruct our minion to install Nginx.

Checking the system's original state

First, make sure that an HTTP server is not installed on our minion. When opening a browser directed at `http://192.168.33.10/` You should get an error saying the site cannot be reached.

Initialize the top.sls file

System configuration is done in the /srv/salt/top.sls file (and subfiles/folders), and then applied by running the `state.highstate` command to have the Salt master give orders so minions will update their instructions and run the associated commands.

First Create an empty file on your Salt master (Mac OS X machine):

```
touch /srv/salt/top.sls
```

When the file is empty, or if no configuration is found for our minion an error is reported:

```
sudo salt 'minion1' state.highstate
```

Should return an error stating: "No Top file or external nodes data matches found".

Create The Nginx Configuration

Now is finally the time to enter the real meat of our server's configuration. For this tutorial our minion will be treated as a web server that needs to have Nginx installed.

Insert the following lines into the `/srv/salt/top.sls` file (which should current be empty).

```
base:
  'minion1':
    - bin.nginx
```

Now create a `/srv/salt/bin/nginx.sls` file containing the following:

```
nginx:
  pkg.installed:
    - name: nginx
  service.running:
    - enable: True
    - reload: True
```

Check Minion State

Finally run the state.highstate command again:

```
sudo salt 'minion1' state.highstate
```

You should see a log showing that the Nginx package has been installed and the service configured. To prove it, open your browser and navigate to http://192.168.33.10/, you should see the standard Nginx welcome page.

Congratulations!

Where To Go From Here

A full description of configuration management within Salt (sls files among other things) is available here: http://docs.saltstack.com/index.html#configuration-management

7.4.14 Writing Salt Tests

Note: THIS TUTORIAL IS A WORK IN PROGRESS

Salt comes with a powerful integration and unit test suite. The test suite allows for the fully automated run of integration and/or unit tests from a single interface. The integration tests are surprisingly easy to write and can be written to be either destructive or non-destructive.

Gettign Set Up For Tests

To walk through adding an integration test, start by getting the latest development code and the test system from github:

Note: The develop branch often has failing tests and should always be considered a staging area. For a checkout that tests should be running perfectly on please checkout from a specific release tag.

```
git clone git@github.com:saltstack/salt.git
pip install git+https://github.com/saltstack/salt-testing.git#egg=SaltTesting
```

Now that a fresh checkout is available run the test suite

Destructive vs Non-destructive

Since Salt is used to change the settings and behavior of systems often the best approach to run tests is to make actual changes to an underlying system. This is where the concept of destructive integration tests comes into play. Tests can be written to alter the system they are running on. This capability is what fills in the gap needed to properly test aspects of system management like package installation.

To write a destructive test import and use the *destructiveTest* decorator for the test method:

```
import integration
from salttesting.helpers import destructiveTest

class PkgTest(integration.ModuleCase):
    @destructiveTest
    def test_pkg_install(self):
        ret = self.run_function('pkg.install', name='finch')
        self.assertSaltTrueReturn(ret)
        ret = self.run_function('pkg.purge', name='finch')
        self.assertSaltTrueReturn(ret)
```

Automated Test Runs

SaltStack maintains a Jenkins server which can be viewed at http://jenkins.saltstack.com. The tests executed from this Jenkins server create fresh virtual machines for each test run, then execute the destructive tests on the new clean virtual machine. This allows for the execution of tests across supported platforms.

7.5 Salt Cloud

7.5.1 Salt as a Cloud Controller

In Salt 0.14.0 advanced cloud control systems were introduced, allowing for private cloud vms to be managed directly with Salt. This system is generally referred to as **Salt Virt**.

The Salt Virt system already exists and is installed within Salt itself, this means that beyond setting up Salt no additional salt code needs to be deployed.

The main goal of Salt Virt is the facilitate a very fast and simple cloud. A cloud that can scale and a fully featured cloud. Salt Virt comes with the ability to set up and manage complex virtual machine networking, powerful image and disk management, as well as virtual machine migration, migration with and without shared storage.

This means that Salt Virt can be used to create a cloud from a blade center and a SAN, but can also create a cloud out of a swarm of Linux Desktops without a single shared storage system. Salt Virt can make clouds from truly commodity hardware, but can also stand up the power of specialized hardware as well.

Setting up Hypervisors

The first step to set up the hypervisors involves getting the correct software installed and setting up the hypervisor network interfaces.

Installing Hypervisor Software

Salt Virt is made to be hypervisor agnostic, but currently the only fully implemented hypervisor is KVM via libvirt.

The required software for a hypervisor is libvirt and kvm. For advanced features install libguestfs or qemu-nbd.

Note: Libguestfs and qemu-nbd allow for virtual machine images to be mounted before startup and get pre-seeded with configurations and a salt minion

This sls will set up the needed software for a hypervisor, and run the routines to set up the libvirt pki keys.

Note: Package names and setup used is Red Hat specific, different package names will be required for different platforms

```
libvirt:
  pkg:
    - installed
  file:
    - managed
    - name: /etc/sysconfig/libvirtd
    - contents: 'LIBVIRTD_ARGS="--listen"'
    - require:
      - pkg: libvirt
  libvirt:
    - keys
    - require:
      - pkg: libvirt
  service:
    - running
    - name: libvirtd
    - require:
      - pkg: libvirt
      - network: br0
      - libvirt: libvirt
    - watch:
      - file: libvirt

libvirt-python:
  pkg:
    - installed

libguestfs:
  pkg:
    - installed
    - pkgs:
      - libguestfs
      - libguestfs-tools
```

Hypervisor Network Setup

The hypervisors will need to be running a network bridge to serve up network devices for virtual machines, this formula will set up a standard bridge on a hypervisor connecting the bridge to eth0:

```
eth0:
  network.managed:
    - enabled: True
    - type: eth
    - bridge: br0

br0:
  network.managed:
    - enabled: True
    - type: bridge
    - proto: dhcp
    - require:
      - network: eth0
```

Virtual Machine Network Setup

Salt Virt comes with a system to model the network interfaces used by the deployed virtual machines, by default a single interface is created for the deployed virtual machine and is bridged to `br0`. To get going with the default networking setup ensure that the bridge interface named `br0` exists on the hypervisor and is bridged to an active network device.

Note: To use more advanced networking in Salt Virt read the *Salt Virt Networking* document:

Salt Virt Networking

Libvirt State

One of the challenges of deploying a libvirt based cloud is the distribution of libvirt certificates. These certificates allow for virtual machine migration. Salt comes with a system used to auto deploy these certificates. Salt manages the signing authority key and generates keys for libvirt clients on the master, signs them with the certificate authority and uses pillar to distribute them. This is managed via the `libvirt` state. Simply execute this formula on the minion to ensure that the certificate is in place and up to date:

Note: The above formula includes the calls needed to set up libvirt keys.

```
libvirt_keys:
  libvirt.keys
```

Getting Virtual Machine Images Ready

Salt Virt, requires that virtual machine images be provided as these are not generated on the fly. Generating these virtual machine images differs greatly based on the underlying platform.

Virtual machine images can be manually created using KVM and running through the installer, but this process is not recommended since it is very manual and prone to errors.

Virtual Machine generation applications are available for many platforms:

vm-builder: http://wiki.debian.org/VMBuilder

Once virtual machines images are available the easiest way to make them available to Salt Virt is to place them in the Salt file server. Just copy an image into `/srv/salt` and it can now be used by Salt Virt.

For purposes of this demo, the file name `centos.img` will be used.

Existing Virtual Machine Images

Many existing Linux distributions distribute virtual machine images which can be used with Salt Virt. Please be advised that NONE OF THESE IMAGES ARE SUPPORTED BY SALTSTACK.

CentOS These images have been prepared for OpenNebula but should work without issue with Salt Virt, only the raw qcow image file is needed: http://wiki.centos.org/Cloud/OpenNebula

Fedora Linux Images for Fedora Linux can be found here: http://cloud.fedoraproject.org

Ubuntu Linux Images for Ubuntu Linux can be found here: http://cloud-images.ubuntu.com/

Using Salt Virt

With hypervisors set up and virtual machine images ready, Salt can start issuing cloud commands.

Start by running a Salt Virt hypervisor info command:

```
salt-run virt.hyper_info
```

This will query what the running hypervisor stats are and display information for all configured hypervisors. This command will also validate that the hypervisors are properly configured.

Now that hypervisors are available a virtual machine can be provisioned. The `virt.init` routine will create a new virtual machine:

```
salt-run virt.init centos1 2 512 salt://centos.img
```

This command assumes that the CentOS virtual machine image is sitting in the root of the Salt fileserver. Salt Virt will now select a hypervisor to deploy the new virtual machine on and copy the virtual machine image down to the hypervisor.

Once the VM image has been copied down the new virtual machine will be seeded. Seeding the VMs involves setting pre-authenticated Salt keys on the new VM and if needed, will install the Salt Minion on the new VM before it is started.

Note: The biggest bottleneck in starting VMs is when the Salt Minion needs to be installed. Making sure that the source VM images already have Salt installed will GREATLY speed up virtual machine deployment.

Now that the new VM has been prepared, it can be seen via the `virt.query` command:

```
salt-run virt.query
```

This command will return data about all of the hypervisors and respective virtual machines.

Now that the new VM is booted it should have contacted the Salt Master, a `test.ping` will reveal if the new VM is running.

Migrating Virtual Machines

Salt Virt comes with full support for virtual machine migration, and using the libvirt state in the above formula makes migration possible.

A few things need to be available to support migration. Many operating systems turn on firewalls when originally set up, the firewall needs to be opened up to allow for libvirt and kvm to cross communicate and execution migration routines. On Red Hat based hypervisors in particular port 16514 needs to be opened on hypervisors:

```
iptables -A INPUT -m state --state NEW -m tcp -p tcp --dport 16514 -j ACCEPT
```

Note: More in-depth information regarding distribution specific firewall settings can read in:

Opening the Firewall up for Salt

Salt also needs an additional flag to be turned on as well. The `virt.tunnel` option needs to be turned on. This flag tells Salt to run migrations securely via the libvirt TLS tunnel and to use port 16514. Without `virt.tunnel` libvirt tries to bind to random ports when running migrations. To turn on `virt.tunnel` simple apply it to the master config file:

```
virt.tunnel: True
```

Once the master config has been updated, restart the master and send out a call to the minions to refresh the pillar to pick up on the change:

```
salt \* saltutil.refresh_modules
```

Now, migration routines can be run! To migrate a VM, simply run the Salt Virt migrate routine:

```
salt-run virt.migrate centos <new hypervisor>
```

VNC Consoles

Salt Virt also sets up VNC consoles by default, allowing for remote visual consoles to be oped up. The information from a `virt.query` routine will display the vnc console port for the specific vms:

```
centos
  CPU: 2
  Memory: 524288
  State: running
  Graphics: vnc - hyper6:5900
  Disk - vda:
    Size: 2.0G
    File: /srv/salt-images/ubuntu2/system.qcow2
    File Format: qcow2
  Nic - ac:de:48:98:08:77:
    Source: br0
    Type: bridge
```

The line *Graphics: vnc - hyper6:5900* holds the key. First the port named, in this case 5900, will need to be available in the hypervisor's firewall. Once the port is open, then the console can be easily opened via vncviewer:

```
vncviewer hyper6:5900
```

By default there is no VNC security set up on these ports, which suggests that keeping them firewalled and mandating that SSH tunnels be used to access these VNC interfaces. Keep in mind that activity on a VNC interface that is accessed can be viewed by any other user that accesses that same VNC interface, and any other user logging in can also operate with the logged in user on the virtual machine.

Conclusion

Now with Salt Virt running, new hypervisors can be seamlessly added just by running the above states on new bare metal machines, and these machines will be instantly available to Salt Virt.

7.6 Halite

7.6.1 Installing and Configuring Halite

In this tutorial, we'll walk through installing and setting up Halite. As of 2013-10-12, a packaged version of Halite is not available. In addition, the current version of Halite is considered pre-alpha and is supported only in Salt 0.17 or greater. Additional information is available on GitHub: https://github.com/saltstack/halite

Before beginning this tutorial, ensure that the salt-master is installed. To install the salt-master, please review the installation documentation: http://docs.saltstack.com/topics/installation/index.html

Note: Halite only works with Salt versions greater than 0.17.

Installing Halite Via Package

On CentOS, RHEL, or Fedora:

```
$ yum install python-halite
```

Note: By default python-halite only installs CherryPy. If you would like to use a different webserver please review the instructions below to install pip and your server of choice. The package does not modify the master configuration with `/etc/salt/master`.

Installing Halite Using pip

To begin the installation of Halite from PyPi, you'll need to install pip. The Salt package, as well as the bootstrap, do not install pip by default.

On CentOS, RHEL, or Fedora:

```
$ yum install python-pip
```

On Debian:

```
$ apt-get install python-pip
```

Once you have pip installed, use it to install halite:

```
$ pip install -U halite
```

Depending on the webserver you want to run halite through, you'll need to install that piece as well. On RHEL based distros, use one of the following:

```
$ pip install cherrypy
```

```
$ pip install paste
```

```
$ yum install python-devel
$ yum install gcc
$ pip install gevent
```

On Debian based distributions:

```
$ pip install CherryPy
```

```
$ pip install paste
```

```
$ apt-get install gcc
$ apt-get install python-dev
$ apt-get install libevent-dev
$ pip install gevent
```

Configuring Halite Permissions

Configuring Halite access permissions is easy. By default, you only need to ensure that the @runner group is configured. In the `/etc/salt/master file`, uncomment and modify the following lines:

```
external_auth:
  pam:
    testuser:
      - .*
      - '@runner'
```

Note: You cannot use the root user for pam login; it will fail to authenticate.

Halite uses the runner manage.status to get the status of minions, so runner permissions are required. As you can see in this example, the root user has been configured. If you aren't running Halite as the root user, you'll need to modify this value. For example:

```
external_auth:
  pam:
    mytestuser:
      - .*
      - '@runner'
      - '@wheel'
```

Currently Halite allows, but does not require, any wheel modules.

Configuring Halite Settings

Once you've configured the permissions for Halite, you'll need to set up the Halite settings in the /etc/salt/master file. Halite supports CherryPy, Paste and Gevent out of the box.

To configure cherrypy, add the following to the bottom of your /etc/salt/master file:

```
halite:
  level: 'debug'
  server: 'cherrypy'
  host: '0.0.0.0'
  port: '8080'
  cors: False
  tls: True
  certpath: '/etc/pki/tls/certs/localhost.crt'
  keypath: '/etc/pki/tls/certs/localhost.key'
  pempath: '/etc/pki/tls/certs/localhost.pem'
```

If you wish to use paste:

```
halite:
  level: 'debug'
  server: 'paste'
  host: '0.0.0.0'
  port: '8080'
  cors: False
  tls: True
  certpath: '/etc/pki/tls/certs/localhost.crt'
  keypath: '/etc/pki/tls/certs/localhost.key'
  pempath: '/etc/pki/tls/certs/localhost.pem'
```

To use gevent:

```
halite:
  level: 'debug'
  server: 'gevent'
  host: '0.0.0.0'
  port: '8080'
  cors: False
  tls: True
  certpath: '/etc/pki/tls/certs/localhost.crt'
  keypath: '/etc/pki/tls/certs/localhost.key'
  pempath: '/etc/pki/tls/certs/localhost.pem'
```

The "cherrypy" and "gevent" servers require the certpath and keypath files to run tls/ssl. The .crt file holds the public cert and the .key file holds the private key. Whereas the "paste" server requires a single .pem file that contains both the cert and key. This can be created simply by concatenating the .crt and .key files.

If you want to use a self-signed cert, you can create one using the Salt.tls module:

Note: You might wish to target only a specific minion. The example below targets all connected minions.

```
salt '*' tls.create_self_signed_cert test
```

You can also use `salt-call` to create a self-signed cert. .. code-block:: bash

> salt-call tls.create_self_signed_cert tls

When using self-signed certs, browsers will need approval before accepting the cert. If the web application page has been cached with a non-HTTPS version of the app, then the browser cache will have to be cleared before it will recognize and prompt to accept the self-signed certificate.

Starting Halite

Once you've configured the halite section of your /etc/salt/master, you can restart the salt-master service, and your halite instance will be available. Depending on your configuration, the instance will be available either at http://localhost:8080/app, http://domain:8080/app, or http://123.456.789.012:8080/app .

Note: halite requires an HTML 5 compliant browser.

All logs relating to halite are logged to the default /var/log/salt/master file.

Running Your Halite Instance Through Nginx

Running Your Halite Instance Through Apache

CHAPTER 8

Access Control System

New in version 0.10.4.

Salt maintains a standard system used to open granular control to non administrative users to execute Salt commands. The access control system has been applied to all systems used to configure access to non administrative control interfaces in Salt.These interfaces include, the `peer` system, the `external auth` system and the `client acl` system.

The access control system mandated a standard configuration syntax used in all of the three aforementioned systems. While this adds functionality to the configuration in 0.10.4, it does not negate the old configuration.

Now specific functions can be opened up to specific minions from specific users in the case of external auth and client ACLs, and for specific minions in the case of the peer system.

The access controls are manifested using matchers in these configurations:

```
client_acl:
  fred:
    - web\*:
      - pkg.list_pkgs
      - test.*
      - apache.*
```

In the above example, fred is able to send commands only to minions which match the specified glob target. This can be expanded to include other functions for other minions based on standard targets.

```
external_auth:
  pam:
    dave:
      - test.ping
      - mongo\*:
        - network.*
      - log\*:
        - network.*
        - pkg.*
      - 'G@os:RedHat':
        - kmod.*
    steve:
      - .*
```

The above allows for all minions to be hit by test.ping by dave, and adds a few functions that dave can execute on other minions. It also allows steve unrestricted access to salt commands.

CHAPTER 9

External Authentication System

Salt 0.10.4 comes with a fantastic new way to open up running Salt commands to users. This system allows for Salt itself to pass through authentication to any authentication system (The Unix PAM system was the first) to determine if a user has permission to execute a Salt command.

The external authentication system allows for specific users to be granted access to execute specific functions on specific minions. Access is configured in the master configuration file, and uses the new *access control system*:

```
external_auth:
  pam:
    thatch:
      - 'web*':
        - test.*
        - network.*
    steve:
      - .*
```

So, the above allows the user thatch to execute functions in the test and network modules on the minions that match the web* target. User steve is given unrestricted access to minion commands.

Note: The PAM module does not allow authenticating as `root`.

To allow access to *wheel modules* or *runner modules* the following `@` syntax must be used:

```
external_auth:
  pam:
    thatch:
      - '@wheel'
      - '@runner'
```

The external authentication system can then be used from the command line by any user on the same system as the master with the *-a* option:

```
$ salt -a pam web\* test.ping
```

The system will ask the user for the credentials required by the authentication system and then publish the command.

9.1 Tokens

With external authentication alone the authentication credentials will be required with every call to Salt. This can be alleviated with Salt tokens.

The tokens are short term authorizations and can be easily created by just adding a `-T` option when authenticating:

```
$ salt -T -a pam web\* test.ping
```

Now a token will be created that has a expiration of, by default, 12 hours. This token is stored in a file named `.salt_token` in the active user's home directory. Once the token is created, it is sent with all subsequent communications. The user authentication does not need to be entered again until the token expires. The token expiration time can be set in the Salt master config file.

CHAPTER 10

Pillar of Salt

Pillar is an interface for Salt designed to offer global values that can be distributed to all minions. Pillar data is managed in a similar way as the Salt State Tree.

Pillar was added to Salt in version 0.9.8

Note: Storing sensitive data

Unlike state tree, pillar data is only available for the targeted minion specified by the matcher type. This makes it useful for storing sensitive data specific to a particular minion.

10.1 Declaring the Master Pillar

The Salt Master server maintains a pillar_roots setup that matches the structure of the file_roots used in the Salt file server. Like the Salt file server the `pillar_roots` option in the master config is based on environments mapping to directories. The pillar data is then mapped to minions based on matchers in a top file which is laid out in the same way as the state top file. Salt pillars can use the same matcher types as the standard top file.

The configuration for the `pillar_roots` in the master config file is identical in behavior and function as `file_roots`:

```
pillar_roots:
  base:
    - /srv/pillar
```

This example configuration declares that the base environment will be located in the `/srv/pillar` directory. The top file used matches the name of the top file used for States, and has the same structure:

`/srv/pillar/top.sls`

```
base:
  '*':
    - packages
```

This further example shows how to use other standard top matching types (grain matching is used in this example) to deliver specific salt pillar data to minions with different `os` grains:

```
dev:
  'os:Debian':
    - match: grain
    - servers
```

/srv/pillar/packages.sls

```
{% if grains['os'] == 'RedHat' %}
apache: httpd
git: git
{% elif grains['os'] == 'Debian' %}
apache: apache2
git: git-core
{% endif %}
```

Now this data can be used from within modules, renderers, State SLS files, and more via the shared pillar *dict*:

```
apache:
  pkg:
    - installed
    - name: {{ pillar['apache'] }}

git:
  pkg:
    - installed
    - name: {{ pillar['git'] }}
```

Note that you cannot just list key/value-information in `top.sls`.

10.2 Pillar namespace flattened

The separate pillar files all share the same namespace. Given a `top.sls` of:

```
base:
  '*':
    - packages
    - services
```

a `packages.sls` file of:

```
bind: bind9
```

and a `services.sls` file of:

```
bind: named
```

Then a request for the `bind` pillar will only return 'named'; the 'bind9' value is not available. It is better to structure your pillar files with more hierarchy. For example your `package.sls` file could look like:

```
packages:
  bind: bind9
```

10.3 Including Other Pillars

New in version 0.16.0.

Pillar SLS files may include other pillar files, similar to State files. Two syntaxes are available for this purpose. The simple form simply includes the additional pillar as if it were part of the same file:

```
include:
  - users
```

The full include form allows two additional options – passing default values to the templating engine for the included pillar file as well as an optional key under which to nest the results of the included pillar:

```
include:
  - users:
      defaults:
          sudo: ['bob', 'paul']
      key: users
```

With this form, the included file (users.sls) will be nested within the 'users' key of the compiled pillar. Additionally, the 'sudo' value will be available as a template variable to users.sls.

10.4 Viewing Minion Pillar

Once the pillar is set up the data can be viewed on the minion via the `pillar` module, the pillar module comes with two functions, `pillar.items` and and `pillar.raw`. `pillar.items` will return a freshly reloaded pillar and `pillar.raw` will return the current pillar without a refresh:

```
salt '*' pillar.items
```

Note: Prior to version 0.16.2, this function is named `pillar.data`. This function name is still supported for backwards compatibility.

10.5 Pillar "get" Function

New in version 0.14.0.

The `pillar.get` function works much in the same way as the `get` method in a python dict, but with an enhancement: nested dict components can be extracted using a *:* delimiter.

If a structure like this is in pillar:

```
foo:
  bar:
    baz: qux
```

Extracting it from the raw pillar in an sls formula or file template is done this way:

```
{{ pillar['foo']['bar']['baz'] }}
```

Now, with the new `pillar.get` function the data can be safely gathered and a default can be set, allowing the template to fall back if the value is not available:

```
{{ salt['pillar.get']('foo:bar:baz', 'qux') }}
```

This makes handling nested structures much easier.

Note: `pillar.get()` vs `salt['pillar.get']()`

It should be noted that within templating, the `pillar` variable is just a dictionary. This means that calling `pillar.get()` inside of a template will just use the default dictionary `.get()` function which does not include the extra `:` delimeter functionality. It must be called using the above syntax (`salt['pillar.get']('foo:bar:baz', 'qux')`) to get the salt function, instead of the default dictionary behavior.

10.6 Refreshing Pillar Data

When pillar data is changed on the master the minions need to refresh the data locally. This is done with the `saltutil.refresh_pillar` function.

```
salt '*' saltutil.refresh_pillar
```

This function triggers the minion to asynchronously refresh the pillar and will always return `None`.

10.7 Targeting with Pillar

Pillar data can be used when targeting minions. This allows for ultimate control and flexibility when targeting minions.

```
salt -I 'somekey:specialvalue' test.ping
```

Like with *Grains*, it is possible to use globbing as well as match nested values in Pillar, by adding colons for each level that is being traversed. The below example would match minions with a pillar named `foo`, which is a dict containing a key `bar`, with a value beginning with `baz`:

```
salt -I 'foo:bar:baz*' test.ping
```

10.8 Master Config In Pillar

For convenience the data stored in the master configuration file is made available in all minion's pillars. This makes global configuration of services and systems very easy but may not be desired if sensitive data is stored in the master configuration.

To disable the master config from being added to the pillar set `pillar_opts` to `False`:

```
pillar_opts: False
```

Master Tops System

In 0.10.4 the *external_nodes* system was upgraded to allow for modular subsystems to be used to generate the top file data for a highstate run on the master.

The old *external_nodes* option still works, but will be removed in the future in favor of the new *master_tops* option which uses the modular system instead. The master tops system contains a number of subsystems that are loaded via the Salt loader interfaces like modules, states, returners, runners, etc.

Using the new *master_tops* option is simple:

```
master_tops:
  ext_nodes: cobbler-external-nodes
```

for *Cobbler* or:

```
master_tops:
  reclass:
    inventory_base_uri: /etc/reclass
    classes_uri: roles
```

for *Reclass*.

CHAPTER 12

Job Management

New in version 0.9.7.

Since Salt executes jobs running on many systems, Salt needs to be able to manage jobs running on many systems. As of Salt 0.9.7, the capability was added for more advanced job management.

12.1 The Minion proc System

The Salt Minions now maintain a *proc* directory in the Salt cachedir, the *proc* directory maintains files named after the executed job ID. These files contain the information about the current running jobs on the minion and allow for jobs to be looked up. This is located in the *proc* directory under the cachedir, with a default configuration it is under */var/cache/salt/proc*.

12.2 Functions in the saltutil Module

Salt 0.9.7 introduced a few new functions to the *saltutil* module for managing jobs. These functions are:

1. `running` Returns the data of all running jobs that are found in the *proc* directory.
2. `find_job` Returns specific data about a certain job based on job id.
3. `signal_job` Allows for a given jid to be sent a signal.
4. `term_job` Sends a termination signal (SIGTERM, 15) to the process controlling the specified job.
5. `kill_job` Sends a kill signal (SIGKILL, 9) to the process controlling the specified job.

These functions make up the core of the back end used to manage jobs at the minion level.

12.3 The jobs Runner

A convenience runner front end and reporting system has been added as well. The jobs runner contains functions to make viewing data easier and cleaner.

The jobs runner contains a number of functions...

12.3.1 active

The active function runs saltutil.running on all minions and formats the return data about all running jobs in a much more usable and compact format. The active function will also compare jobs that have returned and jobs that are still running, making it easier to see what systems have completed a job and what systems are still being waited on.

```
# salt-run jobs.active
```

12.3.2 lookup_jid

When jobs are executed the return data is sent back to the master and cached. By default is is cached for 24 hours, but this can be configured via the `keep_jobs` option in the master configuration. Using the lookup_jid runner will display the same return data that the initial job invocation with the salt command would display.

```
# salt-run jobs.lookup_jid <job id number>
```

12.3.3 list_jobs

Before finding a historic job, it may be required to find the job id. list_jobs will parse the cached execution data and display all of the job data for jobs that have already, or partially returned.

```
# salt-run jobs.list_jobs
```

CHAPTER 13

Salt Scheduling

In Salt versions greater than 0.12.0, the scheduling system allows incremental executions on minions or the master. The schedule system exposes the execution of any execution function on minions or any runner on the master.

Scheduling is enabled via the `schedule` option on either the master or minion config files, or via a minion's pillar data.

Note: The scheduler executes different functions on the master and minions. When running on the master the functions reference runner functions, when running on the minion the functions specify execution functions.

Specify `maxrunning` to ensure that there are no more than N copies of a particular routine running. Use this for jobs that may be long-running and could step on each other or otherwise double execute. The default for `maxrunning` is 1.

States are executed on the minion, as all states are. You can pass positional arguments are provide a yaml dict of named arguments.

13.1 States

```
schedule:
  log-loadavg:
    function: cmd.run
    seconds: 3660
    args:
      - 'logger -t salt < /proc/loadavg'
    kwargs:
      stateful: False
      shell: True
```

13.2 Highstates

To set up a highstate to run on a minion every 60 minutes set this in the minion config or pillar:

```
schedule:
  highstate:
    function: state.highstate
    minutes: 60
```

Time intervals can be specified as seconds, minutes, hours, or days.

13.3 Runners

Runner executions can also be specified on the master within the master configuration file:

```
schedule:
  overstate:
    function: state.over
    seconds: 35
    minutes: 30
    hours: 3
```

The above configuration will execute the state.over runner every 3 hours, 30 minutes and 35 seconds, or every 12,635 seconds.

13.4 Scheduler With Returner

The scheduler is also useful for tasks like gathering monitoring data about a minion, this schedule option will gather status data and send it to a mysql returner database:

```
schedule:
  uptime:
    function: status.uptime
    seconds: 60
    returner: mysql
  meminfo:
    function: status.meminfo
    minutes: 5
    returner: mysql
```

Since specifying the returner repeatedly can be tiresome, the `schedule_returner` option is available to specify one or a list of global returners to be used by the minions when scheduling.

CHAPTER 14

Running the Salt Master/Minion as an Unprivileged User

While the default setup runs the master and minion as the root user, it is generally wise to run the master as an unprivileged user.

As of Salt 0.9.10 it is possible to run Salt as as a non-root user. This can be done by setting the `user` parameter in the master configuration file. and restarting the `salt-master` service.

The minion has it's own `user` parameter as well, but running the minion as an unprivileged user will keep it from making changes to things like users, installed packages, etc. unless access controls (sudo, etc.) are setup on the minion to permit the non-root user to make the needed changes.

In order to allow Salt to successfully run as a non-root user, ownership and permissions need to be set such that the desired user can read from and write to the following directories (and their subdirectories, where applicable):

- /etc/salt
- /var/cache/salt
- /var/log/salt

Ownership can be easily changed with `chown`, like so:

```
# chown -R user /etc/salt /var/cache/salt /var/log/salt
```

Warning: Running either the master or minion with the `root_dir` parameter specified will affect these paths, as will setting options like `pki_dir`, `cachedir`, `log_file`, and other options that normally live in the above directories.

CHAPTER 15

Troubleshooting

The intent of the troubleshooting section is to introduce solutions to a number of common issues encountered by users and the tools that are available to aid in developing States and Salt code.

15.1 Troubleshooting the Salt Master

If your Salt master is having issues such as minions not returning data, slow execution times, or a variety of other issues the *Salt master troubleshooting page* contains details on troubleshooting the most common issues encountered.

15.2 Troubleshooting the Salt Minion

In the event that your Salt minion is having issues a variety of solutions and suggestions are available at the *Salt minion troubleshooting page*.

15.3 Running in the Foreground

A great deal of information is available via the debug logging system, if you are having issues with minions connecting or not starting run the minion and/or master in the foreground:

```
salt-master -l debug
salt-minion -l debug
```

Anyone wanting to run Salt daemons via a process supervisor such as monit, runit, or supervisord, should omit the `-d` argument to the daemons and run them in the foreground.

15.4 What Ports do the Master and Minion Need Open?

No ports need to be opened up on each minion. For the master, TCP ports 4505 and 4506 need to be open. If you've put both your Salt master and minion in debug mode and don't see an acknowledgment that your minion has connected, it could very well be a firewall.

You can check port connectivity from the minion with the nc command:

```
nc -v -z salt.master.ip 4505
nc -v -z salt.master.ip 4506
```

There is also a *firewall configuration* document that might help as well.

If you've enabled the right TCP ports on your operating system or Linux distribution's firewall and still aren't seeing connections, check that no additional access control system such as SELinux or AppArmor is blocking Salt.

15.5 Using salt-call

The `salt-call` command was originally developed for aiding in the development of new Salt modules. Since then, many applications have been developed for running any Salt module locally on a minion. These range from the original intent of salt-call, development assistance, to gathering more verbose output from calls like `state.highstate`.

When creating your state tree, it is generally recommended to invoke `state.highstate` with `salt-call`. This displays far more information about the highstate execution than calling it remotely. For even more verbosity, increase the loglevel with the same argument as `salt-minion`:

```
salt-call -l debug state.highstate
```

The main difference between using `salt` and using `salt-call` is that `salt-call` is run from the minion, and it only runs the selected function on that minion. By contrast, `salt` is run from the master, and requires you to specify the minions on which to run the command using salt's *targeting system*.

15.6 Too many open files

The salt-master needs at least 2 sockets per host that connects to it, one for the Publisher and one for response port. Thus, large installations may, upon scaling up the number of minions accessing a given master, encounter:

```
12:45:29,289 [salt.master    ][INFO    ] Starting Salt worker process 38
Too many open files
sock != -1 (tcp_listener.cpp:335)
```

The solution to this would be to check the number of files allowed to be opened by the user running salt-master (root by default):

```
[root@salt-master ~]# ulimit -n
1024
```

And modify that value to be at least equal to the number of minions x 2. This setting can be changed in limits.conf as the nofile value(s), and activated upon new a login of the specified user.

So, an environment with 1800 minions, would need 1800 x 2 = 3600 as a minimum.

15.7 Salt Master Stops Responding

There are known bugs with ZeroMQ versions less than 2.1.11 which can cause the Salt master to not respond properly. If you're running a ZeroMQ version greater than or equal to 2.1.9, you can work around the bug by setting the sysctls `net.core.rmem_max` and `net.core.wmem_max` to 16777216. Next, set the third field in `net.ipv4.tcp_rmem` and `net.ipv4.tcp_wmem` to at least 16777216.

You can do it manually with something like:

```
# echo 16777216 > /proc/sys/net/core/rmem_max
# echo 16777216 > /proc/sys/net/core/wmem_max
# echo "4096 87380 16777216" > /proc/sys/net/ipv4/tcp_rmem
# echo "4096 87380 16777216" > /proc/sys/net/ipv4/tcp_wmem
```

Or with the following Salt state:

```
net.core.rmem_max:
  sysctl:
    - present
    - value: 16777216

net.core.wmem_max:
  sysctl:
    - present
    - value: 16777216

net.ipv4.tcp_rmem:
  sysctl:
    - present
    - value: 4096 87380 16777216

net.ipv4.tcp_wmem:
  sysctl:
    - present
    - value: 4096 87380 16777216
```

15.8 Salt and SELinux

Currently there are no SELinux policies for Salt. For the most part Salt runs without issue when SELinux is running in Enforcing mode. This is because when the minion executes as a daemon the type context is changed to `initrc_t`. The problem with SELinux arises when using salt-call or running the minion in the foreground, since the type context stays `unconfined_t`.

This problem is generally manifest in the rpm install scripts when using the pkg module. Until a full SELinux Policy is available for Salt the solution to this issue is to set the execution context of `salt-call` and `salt-minion` to rpm_exec_t:

```
# CentOS 5 and RHEL 5:
chcon -t system_u:system_r:rpm_exec_t:s0 /usr/bin/salt-minion
chcon -t system_u:system_r:rpm_exec_t:s0 /usr/bin/salt-call

# CentOS 6 and RHEL 6:
chcon system_u:object_r:rpm_exec_t:s0 /usr/bin/salt-minion
chcon system_u:object_r:rpm_exec_t:s0 /usr/bin/salt-call
```

This works well, because the `rpm_exec_t` context has very broad control over other types.

15.9 Red Hat Enterprise Linux 5

Salt requires Python 2.6 or 2.7. Red Hat Enterprise Linux 5 and its variants come with Python 2.4 installed by default. When installing on RHEL 5 from the EPEL repository this is handled for you. But, if you run Salt from git, be advised that its dependencies need to be installed from EPEL and that Salt needs to be run with the `python26` executable.

15.10 Common YAML Gotchas

An extensive list of *YAML idiosyncrasies* has been compiled.

15.11 Live Python Debug Output

If the minion or master seems to be unresponsive, a SIGUSR1 can be passed to the processes to display where in the code they are running. If encountering a situation like this, this debug information can be invaluable. First make sure the master of minion are running in the foreground:

```
salt-master -l debug
salt-minion -l debug
```

Then pass the signal to the master or minion when it seems to be unresponsive:

```
killall -SIGUSR1 salt-master
killall -SIGUSR1 salt-minion
```

Also under BSD and Mac OS X in addition to SIGUSR1 signal, debug subroutine set up for SIGINFO which has an advantage of being sent by Ctrl+T shortcut.

When filing an issue or sending questions to the mailing list for a problem with an unresponsive daemon this information can be invaluable.

15.12 Salt 0.16.x minions cannot communicate with a 0.17.x master

As of release 0.17.1 you can no longer run different versions of Salt on your Master and Minion servers. This is due to a protocol change for security purposes. The Salt team will continue to attempt to ensure versions are as backwards compatible as possible.

CHAPTER 16

Troubleshooting the Salt Master

16.1 Running in the Foreground

A great deal of information is available via the debug logging system, if you are having issues with minions connecting or not starting run the master in the foreground:

```
# salt-master -l debug
```

Anyone wanting to run Salt daemons via a process supervisor such as monit, runit, or supervisord, should omit the `-d` argument to the daemons and run them in the foreground.

16.2 What Ports does the Master Need Open?

For the master, TCP ports 4505 and 4506 need to be open. If you've put both your Salt master and minion in debug mode and don't see an acknowledgment that your minion has connected, it could very well be a firewall interfering with the connection. See our *firewall configuration* page for help opening the firewall on various platforms.

If you've opened the correct TCP ports and still aren't seeing connections, check that no additional access control system such as SELinux or AppArmor is blocking Salt.

16.3 Too many open files

The salt-master needs at least 2 sockets per host that connects to it, one for the Publisher and one for response port. Thus, large installations may, upon scaling up the number of minions accessing a given master, encounter:

```
12:45:29,289 [salt.master    ][INFO    ] Starting Salt worker process 38
Too many open files
sock != -1 (tcp_listener.cpp:335)
```

The solution to this would be to check the number of files allowed to be opened by the user running salt-master (root by default):

```
[root@salt-master ~]# ulimit -n
1024
```

If this value is not equal to at least twice the number of minions, then it will need to be raised. For example, in an environment with 1800 minions, the `nofile` limit should be set to no less than 3600. This can be done by creating the file `/etc/security/limits.d/99-salt.conf`, with the following contents:

```
root        hard    nofile        4096
root        soft    nofile        4096
```

Replace `root` with the user under which the master runs, if different.

If your master does not have an `/etc/security/limits.d` directory, the lines can simply be appended to `/etc/security/limits.conf`.

As with any change to resource limits, it is best to stay logged into your current shell and open another shell to run `ulimit -n` again and verify that the changes were applied correctly. Additionally, if your master is running upstart, it may be necessary to specify the `nofile` limit in `/etc/default/salt-master` if upstart isn't respecting your resource limits:

```
limit nofile 4096 4096
```

Note: The above is simply an example of how to set these values, and you may wish to increase them even further if your Salt master is doing more than just running Salt.

16.4 Salt Master Stops Responding

There are known bugs with ZeroMQ versions less than 2.1.11 which can cause the Salt master to not respond properly. If you're running a ZeroMQ version greater than or equal to 2.1.9, you can work around the bug by setting the sysctls `net.core.rmem_max` and `net.core.wmem_max` to 16777216. Next, set the third field in `net.ipv4.tcp_rmem` and `net.ipv4.tcp_wmem` to at least 16777216.

You can do it manually with something like:

```
# echo 16777216 > /proc/sys/net/core/rmem_max
# echo 16777216 > /proc/sys/net/core/wmem_max
# echo "4096 87380 16777216" > /proc/sys/net/ipv4/tcp_rmem
# echo "4096 87380 16777216" > /proc/sys/net/ipv4/tcp_wmem
```

Or with the following Salt state:

```
net.core.rmem_max:
  sysctl:
    - present
    - value: 16777216

net.core.wmem_max:
  sysctl:
    - present
    - value: 16777216

net.ipv4.tcp_rmem:
  sysctl:
    - present
    - value: 4096 87380 16777216

net.ipv4.tcp_wmem:
  sysctl:
    - present
    - value: 4096 87380 16777216
```

16.5 Live Python Debug Output

If the master seems to be unresponsive, a SIGUSR1 can be passed to the salt-master threads to display what piece of code is executing. This debug information can be invaluable in tracking down bugs.

To pass a SIGUSR1 to the master, first make sure the minion is running in the foreground. Stop the service if it is running as a daemon, and start it in the foreground like so:

```
# salt-master -l debug
```

Then pass the signal to the master when it seems to be unresponsive:

```
# killall -SIGUSR1 salt-master
```

When filing an issue or sending questions to the mailing list for a problem with an unresponsive daemon, be sure to include this information if possible.

16.6 Commands Time Out or Do Not Return Output

Depending on your OS (this is most common on Ubuntu due to apt-get) you may sometimes encounter times where your highstate, or other long running commands do not return output. This is most commonly due to the timeout being reached. By default the timeout is set to 5 seconds. The timeout value can easily be increased by modifying the `timeout` line within your `/etc/salt/master` configuration file.

16.7 Passing the -c Option to Salt Returns a Permissions Error

Using the `-c` option with the Salt command modifies the configuration directory. When the configuratio file is read it will still base data off of the `root_dir` setting. This can result in unintended behavior if you are expecting files such as `/etc/salt/pki` to be pulled from the location specified with `-c`. Modify the `root_dir` setting to address this behavior.

CHAPTER 17

Troubleshooting the Salt Minion

17.1 Running in the Foreground

A great deal of information is available via the debug logging system, if you are having issues with minions connecting or not starting run the minion in the foreground:

```
# salt-minion -l debug
```

Anyone wanting to run Salt daemons via a process supervisor such as monit, runit, or supervisord, should omit the `-d` argument to the daemons and run them in the foreground.

17.2 What Ports does the Minion Need Open?

No ports need to be opened on the minion, as it makes outbound connections to the master. If you've put both your Salt master and minion in debug mode and don't see an acknowledgment that your minion has connected, it could very well be a firewall interfering with the connection. See our *firewall configuration* page for help opening the firewall on various platforms.

If you have netcat installed, you can check port connectivity from the minion with the `nc` command:

```
$ nc -v -z salt.master.ip.addr 4505
Connection to salt.master.ip.addr 4505 port [tcp/unknown] succeeded!
$ nc -v -z salt.master.ip.addr 4506
Connection to salt.master.ip.addr 4506 port [tcp/unknown] succeeded!
```

The Nmap utility can also be used to check if these ports are open:

```
# nmap -sS -q -p 4505-4506 salt.master.ip.addr

Starting Nmap 6.40 ( http://nmap.org ) at 2013-12-29 19:44 CST
Nmap scan report for salt.master.ip.addr (10.0.0.10)
Host is up (0.0026s latency).
PORT     STATE  SERVICE
4505/tcp open   unknown
4506/tcp open   unknown
MAC Address: 00:11:22:AA:BB:CC (Intel)

Nmap done: 1 IP address (1 host up) scanned in 1.64 seconds
```

If you've opened the correct TCP ports and still aren't seeing connections, check that no additional access control system such as SELinux or AppArmor is blocking Salt.

17.3 Using salt-call

The `salt-call` command was originally developed for aiding in the development of new Salt modules. Since then, many applications have been developed for running any Salt module locally on a minion. These range from the original intent of salt-call, development assistance, to gathering more verbose output from calls like `state.highstate`.

When initially creating your state tree, it is generally recommended to invoke `state.highstate` from the minion with `salt-call`. This displays far more information about the highstate execution than calling it remotely. For even more verbosity, increase the loglevel with the same argument as `salt-minion`:

```
# salt-call -l debug state.highstate
```

The main difference between using `salt` and using `salt-call` is that `salt-call` is run from the minion, and it only runs the selected function on that minion. By contrast, `salt` is run from the master, and requires you to specify the minions on which to run the command using salt's *targeting system*.

17.4 Live Python Debug Output

If the minion seems to be unresponsive, a SIGUSR1 can be passed to the process to display what piece of code is executing. This debug information can be invaluable in tracking down bugs.

To pass a SIGUSR1 to the minion, first make sure the minion is running in the foreground. Stop the service if it is running as a daemon, and start it in the foreground like so:

```
# salt-minion -l debug
```

Then pass the signal to the minion when it seems to be unresponsive:

```
# killall -SIGUSR1 salt-minion
```

When filing an issue or sending questions to the mailing list for a problem with an unresponsive daemon, be sure to include this information if possible.

CHAPTER 18

YAML Idiosyncrasies

One of Salt's strengths, the use of existing serialization systems for representing SLS data, can also backfire. YAML is a general purpose system and there are a number of things that would seem to make sense in an sls file that cause YAML issues. It is wise to be aware of these issues. While reports or running into them are generally rare they can still crop up at unexpected times.

18.1 Spaces vs Tabs

YAML uses spaces, period. Do not use tabs in your SLS files! If strange errors are coming up in rendering SLS files, make sure to check that no tabs have crept in! In Vim, after enabling search highlighting with: `:set hlsearch`, you can check with the following key sequence in normal mode(you can hit *ESC* twice to be sure): `/`, *Ctrl-v*, *Tab*, then hit *Enter*. Also, you can convert tabs to 2 spaces by these commands in Vim: `:set tabstop=2 expandtab` and then `:retab`.

18.2 Indentation

The suggested syntax for YAML files is to use 2 spaces for indentation, but YAML will follow whatever indentation system that the individual file uses. Indentation of two spaces works very well for SLS files given the fact that the data is uniform and not deeply nested.

18.2.1 Nested Dicts (key=value)

When *dicts* are more deeply nested, they no longer follow the same indentation logic. This is rarely something that comes up in Salt, since deeply nested options like these are discouraged when making State modules, but some do exist. A good example of this can be found in the `context` and `default` options from the *file.managed* state:

```
/etc/http/conf/http.conf:
  file:
    - managed
    - source: salt://apache/http.conf
    - user: root
    - group: root
    - mode: 644
    - template: jinja
    - context:
        custom_var: "override"
    - defaults:
        custom_var: "default value"
        other_var: 123
```

Notice that while the indentation is two spaces per level, for the values under the `context` and `defaults` options there is a four-space indent. If only two spaces are used to indent, then the information will not be loaded correctly. If using a double indent is not desirable, then a deeply-nested dict can be declared with curly braces:

```
/etc/http/conf/http.conf:
  file:
    - managed
    - source: salt://apache/http.conf
    - user: root
    - group: root
    - mode: 644
    - template: jinja
    - context: {
      custom_var: "override" }
    - defaults: {
      custom_var: "default value",
      other_var: 123 }
```

18.3 True/False, Yes/No, On/Off

PyYAML will load these values as boolean `True` or `False`. Un-capitalized versions will also be loaded as booleans (`true`, `false`, `yes`, `no`, `on`, and `off`). This can be especially problematic when constructing Pillar data. Make sure that your Pillars which need to use the string versions of these values are enclosed in quotes.

18.4 Integers are Parsed as Integers

NOTE: This has been fixed in salt 0.10.0, as of this release passing an integer that is preceded by a 0 will be correctly parsed

When passing `integers` into an SLS file, they are passed as integers. This means that if a state accepts a string value and an integer is passed, that an integer will be sent. The solution here is to send the integer as a string.

This is best explained when setting the mode for a file:

```
/etc/vimrc:
  file:
    - managed
    - source: salt://edit/vimrc
    - user: root
    - group: root
    - mode: 644
```

Salt manages this well, since the mode is passed as 644, but if the mode is zero padded as 0644, then it is read by YAML as an integer and evaluated as an octal value, 0644 becomes 420. Therefore, if the file mode is preceded by a 0 then it needs to be passed as a string:

```
/etc/vimrc:
  file:
    - managed
    - source: salt://edit/vimrc
    - user: root
    - group: root
    - mode: '0644'
```

18.5 YAML does not like "Double Short Decs"

If I can find a way to make YAML accept "Double Short Decs" then I will, since I think that double short decs would be awesome. So what is a "Double Short Dec"? It is when you declare a multiple short decs in one ID. Here is a standard short dec, it works great:

```
vim:
  pkg.installed
```

The short dec means that there are no arguments to pass, so it is not required to add any arguments, and it can save space.

YAML though, gets upset when declaring multiple short decs, for the record...

THIS DOES NOT WORK:

```
vim:
  pkg.installed
  user.present
```

Similarly declaring a short dec in the same ID dec as a standard dec does not work either...

ALSO DOES NOT WORK:

```
fred:
  user.present
  ssh_auth.present:
    - name: AAAAB3NzaC...
    - user: fred
    - enc: ssh-dss
    - require:
      - user: fred
```

The correct way is to define them like this:

```
vim:
  pkg.installed: []
  user.present: []

fred:
  user.present: []
  ssh_auth.present:
    - name: AAAAB3NzaC...
    - user: fred
    - enc: ssh-dss
    - require:
      - user: fred
```

Alternatively, they can be defined the "old way", or with multiple "full decs":

```
vim:
  pkg:
    - installed
  user:
    - present
```

```
fred:
  user:
    - present
  ssh_auth:
    - present
    - name: AAAAB3NzaC...
    - user: fred
    - enc: ssh-dss
    - require:
      - user: fred
```

18.6 YAML support only plain ASCII

According to YAML specification, only ASCII characters can be used.

Within double-quotes, special characters may be represented with C-style escape sequences starting with a backslash (\ ).

Examples:

```
- micro: "\u00b5"
- copyright: "\u00A9"
- A: "\x41"
- alpha: "\u0251"
- Alef: "\u05d0"
```

List of usable Unicode characters will help you to identify correct numbers.

Python can also be used to discover the Unicode number for a character:

```
repr(u"Text with wrong characters i need to figure out")
```

This shell command can find wrong characters in your SLS files:

```
find . -name '*.sls'  -exec  grep --color='auto' -P -n '[^\x00-\x7F]' \{} \;
```

18.7 Underscores stripped in Integer Definitions

If a definition only includes numbers and underscores, it is parsed by YAML as an integer and all underscores are stripped. To ensure the object becomes a string, it should be surrounded by quotes. More information here.

Here's an example:

```
>>> import yaml
>>> yaml.safe_load('2013_05_10')
20130510
>>> yaml.safe_load('"2013_05_10"')
'2013_05_10'
```

18.8 Automatic `datetime` conversion

If there is a value in a YAML file formatted `2014-01-20 14:23:23` or similar, YAML will automatically convert this to a Python `datetime` object. These objects are not msgpack serializable, and so may break core salt functionality. If values such as these are needed in a salt YAML file (specifically a configuration file), they should be formatted with surrounding strings to force YAML to serialize them as strings:

```
>>> import yaml
>>> yaml.safe_load('2014-01-20 14:23:23')
datetime.datetime(2014, 1, 20, 14, 23, 23)
>>> yaml.safe_load('"2014-01-20 14:23:23"')
'2014-01-20 14:23:23'
```

CHAPTER 19

Community

Join the Salt!

There are many ways to participate in and communicate with the Salt community.

Salt has an active IRC channel and a mailing list.

19.1 Mailing List

Join the salt-users mailing list. It is the best place to ask questions about Salt and see whats going on with Salt development! The Salt mailing list is hosted by Google Groups. It is open to new members.

https://groups.google.com/forum/#!forum/salt-users

19.2 IRC

The `#salt` IRC channel is hosted on the popular Freenode network. You can use the Freenode webchat client right from your browser.

Logs of the IRC channel activity are being collected courtesy of Moritz Lenz.

If you wish to discuss the development of Salt itself join us in `#salt-devel`.

19.3 Follow on Github

The Salt code is developed via Github. Follow Salt for constant updates on what is happening in Salt development:

https://github.com/saltstack/salt

19.4 Blogs

SaltStack Inc. keeps a blog with recent news and advancements:

http://www.saltstack.com/blog/

Thomas Hatch also shares news and thoughts on Salt and related projects in his personal blog The Red45:

http://red45.wordpress.com/

19.5 Example Salt States

The official `salt-states` repository is: https://github.com/saltstack/salt-states

A few examples of salt states from the community:

- https://github.com/blast-hardcheese/blast-salt-states
- https://github.com/kevingranade/kevingranade-salt-state
- https://github.com/uggedal/states
- https://github.com/mattmcclean/salt-openstack/tree/master/salt
- https://github.com/rentalita/ubuntu-setup/
- https://github.com/brutasse/states
- https://github.com/bclermont/states
- https://github.com/pcrews/salt-data

19.6 Follow on ohloh

https://www.ohloh.net/p/salt

19.7 Other community links

- Salt Stack Inc.
- Subreddit
- Google+
- YouTube
- Facebook
- Twitter
- Wikipedia page

19.8 Hack the Source

If you want to get involved with the development of source code or the documentation efforts, please review the *hacking section*!

CHAPTER 20

Salt Based Projects

A number of unofficial open source projects, based on Salt, or written to enhance Salt have been created.

20.1 Salt Sandbox

Created by Aaron Bull Schaefer, aka "elasticdog".

https://github.com/elasticdog/salt-sandbox

Salt Sandbox is a multi-VM Vagrant-based Salt development environment used for creating and testing new Salt state modules outside of your production environment. It's also a great way to learn firsthand about Salt and its remote execution capabilities.

Salt Sandbox will set up three separate virtual machines:

- salt.example.com - the Salt master server
- minion1.example.com - the first Salt minion machine
- minion2.example.com - the second Salt minion machine

These VMs can be used in conjunction to segregate and test your modules based on node groups, top file environments, grain values, etc. You can even test modules on different Linux distributions or release versions to better match your production infrastructure.

CHAPTER 21

Salt Event System

Salt 0.9.10 introduced the Salt Event System. This system is used to fire off events enabling third party applications or external processes to react to behavior within Salt.

The event system is comprised of a few components, the event sockets which publish events, and the event library which can listen to events and send events into the salt system.

21.1 Listening for Events

The event system is accessed via the event library and can only be accessed by the same system user that Salt is running as. To listen to events a SaltEvent object needs to be created and then the get_event function needs to be run. The SaltEvent object needs to know the location that the Salt Unix sockets are kept. In the configuration this is the `sock_dir` option. The `sock_dir` option defaults to "/var/run/salt/master" on most systems.

The following code will check for a single event:

```
import salt.utils.event

event = salt.utils.event.MasterEvent('/var/run/salt/master')

data = event.get_event()
```

Events will also use a "tag". A "tag" allows for events to be filtered. By default all events will be returned, but if only authentication events are desired, then pass the tag "auth". Also, the get_event method has a default poll time assigned of 5 seconds, to change this time set the "wait" option. This example will only listen for auth events and will wait for 10 seconds instead of the default 5.

```
import salt.utils.event

event = salt.utils.event.MasterEvent('/var/run/salt/master')

data = event.get_event(wait=10, tag='auth')
```

Instead of looking for a single event, the iter_events method can be used to make a generator which will continually yield salt events. The iter_events method also accepts a tag, but not a wait time:

```
import salt.utils.event

event = salt.utils.event.MasterEvent('/var/run/salt/master')

for data in event.iter_events(tag='auth'):
    print(data)
```

21.2 Firing Events

It is possible to fire events on either the minion's local bus, or to fire events intended for the master. To fire a local event from the minion, on the command line:

```
salt-call event.fire '{"data": "message to be sent in the event"}' 'tag'
```

To fire an event to be sent to the master, from the minion:

```
salt-call event.fire_master '{"data": "message for the master"}' 'tag'
```

If a process is listening on the minion, it may be useful for a user on the master to fire an event to it:

```
salt minionname event.fire '{"data": "message for the minion"}' 'tag'
```

21.3 Firing Events From Code

Events can be very useful when writing execution modules, in order to inform various processes on the master when a certain task has taken place. In Salt versions previous to 0.17.0, the basic code looks like:

```
# Import the proper library
import salt.utils.event
# Fire deploy action
sock_dir = '/var/run/salt/minion'
event = salt.utils.event.SaltEvent('master', sock_dir)
event.fire_event('Message to be sent', 'tag')
```

In Salt version 0.17.0, the ability to send a payload with a more complex data structure than a string was added. When using this interface, a Python dictionary should be sent instead.

```
# Import the proper library
import salt.utils.event
# Fire deploy action
sock_dir = '/var/run/salt/minion'
payload = {'sample-msg': 'this is a test',
           'example': 'this is the same test'}
event = salt.utils.event.SaltEvent('master', sock_dir)
event.fire_event(payload, 'tag')
```

It should be noted that this code can be used in 3rd party applications as well. So long as the salt-minion process is running, the minion socket can be used:

```
sock_dir = '/var/run/salt/minion'
```

So long as the salt-master process is running, the master socket can be used:

```
sock_dir = '/var/run/salt/master'
```

This allows 3rd party applications to harness the power of the Salt event bus programmatically, without having to make other calls to Salt. A 3rd party process can listen to the event bus on the master, and another 3rd party process can fire events to the process on the master, which Salt will happily pass along.

CHAPTER 22

The Salt Mine

Granted, it took a while for this name to be used in Salt, but version 0.15.0 introduces a new system to Salt called the Salt Mine.

The Salt Mine is used to bridge the gap between setting static variables and gathering live data. The Salt mine is used to collect arbitrary data from minions and store it on the master. This data is then made available to all minions via the `mine` module.

The data is gathered on the minion and sent back to the master where only the most recent data is maintained (if long term data is required use returners or the external job cache).

22.1 Mine Functions

To enable the Salt Mine the *mine_functions* option needs to be applied to a minion. This option can be applied via the minion's configuration file, or the minion's pillar. The *mine_functions* option dictates what functions are being executed and allows for arguments to be passed in:

```
mine_functions:
  network.interfaces: []
  test.ping: []
```

22.2 Mine Interval

The Salt Mine functions are executed when the minion starts and at a given interval by the scheduler. The default interval is every 60 minutes and can be adjusted for the minion via the *mine_interval* option:

```
mine_interval: 60
```

CHAPTER 23

Salt Cloud Documentation

23.1 Cloud Actions

Once a VM has been created, there are a number of actions that can be performed on it. The "reboot" action can be used across all providers, but all other actions are specific to the cloud provider. In order to perform an action, you may specify it from the command line, including the name(s) of the VM to perform the action on:

```
$ salt-cloud -a reboot vm_name
$ salt-cloud -a reboot vm1 vm2 vm2
```

Or you may specify a map which includes all VMs to perform the action on:

```
$ salt-cloud -a reboot -m /path/to/mapfile
```

The following is a list of actions currently supported by salt-cloud:

```
all providers:
    - reboot
ec2:
    - start
    - stop
joyent:
    - stop
```

23.2 Getting Started With AWS EC2

Amazon EC2 is a very widely used public cloud platform and one of the core platforms Salt Cloud has been built to support.

Previously, the suggested provider for AWS EC2 was the *aws* provider. This has been deprecated in favor of the *ec2* provider. Configuration using the old *aws* provider will still function, but that driver is no longer in active development.

```
# Note: This example is for /etc/salt/cloud.providers or any file in the
# /etc/salt/cloud.providers.d/ directory.

my-ec2-southeast-public-ips:
  # Set up the location of the salt master
  #
  minion:
    master: saltmaster.example.com
  # Set up grains information, which will be common for all nodes
  # using this provider
  grains:
    node_type: broker
    release: 1.0.1

  # Specify whether to use public or private IP for deploy script.
  #
  # Valid options are:
  #     private_ips - The salt-master is also hosted with EC2
  #     public_ips - The salt-master is hosted outside of EC2
  #
  ssh_interface: public_ips

  # Set the EC2 access credentials (see below)
  #
  id: HJGRYCILJLKJYG
  key: 'kdjgfsgm;woormgl/aserigjksjdhasdfgn'

  # Make sure this key is owned by root with permissions 0400.
  #
  private_key: /etc/salt/my_test_key.pem
  keyname: my_test_key
  securitygroup: default

  # Optionally configure default region
  #
  location: ap-southeast-1
  availability_zone: ap-southeast-1b

  # Configure which user to use to run the deploy script. This setting is
  # dependent upon the AMI that is used to deploy. It is usually safer to
  # configure this individually in a profile, than globally. Typical users
  # are:
  #
  # Amazon Linux -> ec2-user
  # RHEL         -> ec2-user
  # CentOS       -> ec2-user
  # Ubuntu       -> ubuntu
  #
  ssh_username: ec2-user

  # Optionally add an IAM profile
  iam_profile: 'arn:aws:iam::123456789012:instance-profile/ExampleInstanceProfile'

  provider: ec2


my-ec2-southeast-private-ips:
  # Set up the location of the salt master
  #
  minion:
    master: saltmaster.example.com

  # Specify whether to use public or private IP for deploy script.
  #
  # Valid options are:
  #     private_ips - The salt-master is also hosted with EC2
#     public_ips - The salt-master is hosted outside of EC2
#
ssh_interface: private_ips

# Set the EC2 access credentials (see below)
#
id: HJGRYCILJLKJYG
key: 'kdjgfsgm;woormgl/aserigjksjdhasdfgn'

# Make sure this key is owned by root with permissions 0400.
#
private_key: /etc/salt/my_test_key.pem
keyname: my_test_key
securitygroup: default

# Optionally configure default region
#
location: ap-southeast-1
availability_zone: ap-southeast-1b

# Configure which user to use to run the deploy script. This setting is
# dependent upon the AMI that is used to deploy. It is usually safer to
# configure this individually in a profile, than globally. Typical users
# are:
#
# Amazon Linux -> ec2-user
# RHEL         -> ec2-user
# CentOS       -> ec2-user
# Ubuntu       -> ubuntu
#
ssh_username: ec2-user

# Optionally add an IAM profile
iam_profile: 'my other profile name'

provider: ec2
```

23.2.1 Access Credentials

The `id` and `key` settings may be found in the Security Credentials area of the AWS Account page:

https://portal.aws.amazon.com/gp/aws/securityCredentials

Both are located in the Access Credentials area of the page, under the Access Keys tab. The `id` setting is labeled Access Key ID, and the `key` setting is labeled Secret Access Key.

23.2.2 Key Pairs

In order to create an instance with Salt installed and configured, a key pair will need to be created. This can be done in the EC2 Management Console, in the Key Pairs area. These key pairs are unique to a specific region. Keys in the us-east-1 region can be configured at:

https://console.aws.amazon.com/ec2/home?region=us-east-1#s=KeyPairs

Keys in the us-west-1 region can be configured at

https://console.aws.amazon.com/ec2/home?region=us-west-1#s=KeyPairs

...and so on. When creating a key pair, the browser will prompt to download a pem file. This file must be placed in a directory accessable by Salt Cloud, with permissions set to either 0400 or 0600.

23.2.3 Security Groups

An instance on EC2 needs to belong to a security group. Like key pairs, these are unique to a specific region. These are also configured in the EC2 Management Console. Security groups for the us-east-1 region can be configured at:

https://console.aws.amazon.com/ec2/home?region=us-east-1#s=SecurityGroups

...and so on.

A security group defines firewall rules which an instance will adhere to. If the salt-master is configured outside of EC2, the security group must open the SSH port (usually port 22) in order for Salt Cloud to install Salt.

23.2.4 IAM Profile

Amazon EC2 instances support the concept of an instance profile, which is a logical container for the IAM role. At the time that you launch an EC2 instance, you can associate the instance with an instance profile, which in turn corresponds to the IAM role. Any software that runs on the EC2 instance is able to access AWS using the permissions associated with the IAM role.

Scaffolding the profile is a 2 steps configurations:

1. Configure an IAM Role from the IAM Management Console.
2. Attach this role to a new profile. It can be done with the AWS CLI:

```
> aws iam create-instance-profile --instance-profile-name PROFILE_NAME
> aws iam add-role-to-instance-profile --instance-profile-name PROFILE_NAME --role-name ROLE_NAM
```

Once the profile is created, you can use the **PROFILE_NAME** to configure your cloud profiles.

23.2.5 Cloud Profiles

Set up an initial profile at `/etc/salt/cloud.profiles`:

```
base_ec2_private:
  provider: my-ec2-southeast-private-ips
  image: ami-e565ba8c
  size: Micro Instance
  ssh_username: ec2-user

base_ec2_public:
  provider: my-ec2-southeast-public-ips
  image: ami-e565ba8c
  size: Micro Instance
  ssh_username: ec2-user

base_ec2_db:
  provider: my-ec2-southeast-public-ips
  image: ami-e565ba8c
  size: m1.xlarge
  ssh_username: ec2-user
  volumes:
    - { size: 10, device: /dev/sdf }
    - { size: 10, device: /dev/sdg, type: io1, iops: 1000 }
    - { size: 10, device: /dev/sdh, type: io1, iops: 1000 }
```

The profile can be realized now with a salt command:

```
# salt-cloud -p base_ec2 ami.example.com
# salt-cloud -p base_ec2_public ami.example.com
# salt-cloud -p base_ec2_private ami.example.com
```

This will create an instance named `ami.example.com` in EC2. The minion that is installed on this instance will have an `id` of `ami.example.com`. If the command was executed on the salt-master, its Salt key will automatically be signed on the master.

Once the instance has been created with salt-minion installed, connectivity to it can be verified with Salt:

```
# salt 'ami.example.com' test.ping
```

23.2.6 Required Settings

The following settings are always required for EC2:

```
# Set the EC2 login data
my-ec2-config:
  id: HJGRYCILJLKJYG
  key: 'kdjgfsgm;woormgl/aserigjksjdhasdfgn'
  keyname: test
  securitygroup: quick-start
  private_key: /root/test.pem
  provider: ec2
```

23.2.7 Optional Settings

EC2 allows a location to be set for servers to be deployed in. Availability zones exist inside regions, and may be added to increase specificity.

```
my-ec2-config:
  # Optionally configure default region
  location: ap-southeast-1
  availability_zone: ap-southeast-1b
```

EC2 instances can have a public or private IP, or both. When an instance is deployed, Salt Cloud needs to log into it via SSH to run the deploy script. By default, the public IP will be used for this. If the salt-cloud command is run from another EC2 instance, the private IP should be used.

```
my-ec2-config:
  # Specify whether to use public or private IP for deploy script
  # private_ips or public_ips
  ssh_interface: public_ips
```

Many EC2 instances do not allow remote access to the root user by default. Instead, another user must be used to run the deploy script using sudo. Some common usernames include ec2-user (for Amazon Linux), ubuntu (for Ubuntu instances), admin (official Debian) and bitnami (for images provided by Bitnami).

```
my-ec2-config:
  # Configure which user to use to run the deploy script
  ssh_username: ec2-user
```

Multiple usernames can be provided, in which case Salt Cloud will attempt to guess the correct username. This is mostly useful in the main configuration file:

```
my-ec2-config:
  ssh_username:
    - ec2-user
    - ubuntu
    - admin
    - bitnami
```

Multiple security groups can also be specified in the same fashion:

```
my-ec2-config:
  securitygroup:
    - default
    - extra
```

Block device mappings enable you to specify additional EBS volumes or instance store volumes when the instance is launched. This setting is also available on each cloud profile. Note that the number of instance stores varies by instance type. If more mappings are provided than are supported by the instance type, mappings will be created in the order provided and additional mappings will be ignored. Consult the AWS documentation for a listing of the available instance stores, device names, and mount points.

```
my-ec2-config:
  block_device_mappings:
    - DeviceName: /dev/sdb
      VirtualName: ephemeral0
    - DeviceName: /dev/sdc
      VirtualName: ephemeral1
```

You can also use block device mappings to change the size of the root device at the provisioing time. For example, assuming the root device is '/dev/sda', you can set its size to 100G by using the following configuration.

```
my-ec2-config:
  block_device_mappings:
    - DeviceName: /dev/sda
      Ebs.VolumeSize: 100
```

Tags can be set once an instance has been launched.

```
my-ec2-config:
    tag:
        tag0: value
        tag2: value
```

23.2.8 Modify EC2 Tags

One of the features of EC2 is the ability to tag resources. In fact, under the hood, the names given to EC2 instances by salt-cloud are actually just stored as a tag called Name. Salt Cloud has the ability to manage these tags:

```
salt-cloud -a get_tags mymachine
salt-cloud -a set_tags mymachine tag1=somestuff tag2='Other stuff'
salt-cloud -a del_tags mymachine tag1,tag2,tag3
```

23.2.9 Rename EC2 Instances

As mentioned above, EC2 instances are named via a tag. However, renaming an instance by renaming its tag will cause the salt keys to mismatch. A rename function exists which renames both the instance, and the salt keys.

```
salt-cloud -a rename mymachine newname=yourmachine
```

23.2.10 EC2 Termination Protection

EC2 allows the user to enable and disable termination protection on a specific instance. An instance with this protection enabled cannot be destroyed.

```
salt-cloud -a enable_term_protect mymachine
salt-cloud -a disable_term_protect mymachine
```

23.2.11 Rename on Destroy

When instances on EC2 are destroyed, there will be a lag between the time that the action is sent, and the time that Amazon cleans up the instance. During this time, the instance still retails a Name tag, which will cause a collision if the creation of an instance with the same name is attempted before the cleanup occurs. In order to avoid such collisions, Salt Cloud can be configured to rename instances when they are destroyed. The new name will look something like:

```
myinstance-DEL20f5b8ad4eb64ed88f2c428df80a1a0c
```

In order to enable this, add rename_on_destroy line to the main configuration file:

```
my-ec2-config:
  rename_on_destroy: True
```

23.2.12 Listing Images

Normally, images can be queried on a cloud provider by passing the `--list-images` argument to Salt Cloud. This still holds true for EC2:

```
salt-cloud --list-images my-ec2-config
```

However, the full list of images on EC2 is extremely large, and querying all of the available images may cause Salt Cloud to behave as if frozen. Therefore, the default behavior of this option may be modified, by adding an `owner` argument to the provider configuration:

```
owner: aws-marketplace
```

The possible values for this setting are `amazon`, `aws-marketplace`, `self`, `<AWS account ID>` or `all`. The default setting is `amazon`. Take note that `all` and `aws-marketplace` may cause Salt Cloud to appear as if it is freezing, as it tries to handle the large amount of data.

It is also possible to perform this query using different settings without modifying the configuration files. To do this, call the `avail_images` function directly:

```
salt-cloud -f avail_images my-ec2-config owner=aws-marketplace
```

23.2.13 EC2 Images

The following are lists of available AMI images, generally sorted by OS. These lists are on 3rd-party websites, are not managed by Salt Stack in any way. They are provided here as a reference for those who are interested, and contain no warranty (express or implied) from anyone affiliated with Salt Stack. Most of them have never been used, much less tested, by the Salt Stack team.

- Arch Linux
- FreeBSD
- Fedora
- CentOS
- Ubuntu
- Debian
- Gentoo
- OmniOS
- All Images on Amazon

23.2.14 show_image

This is a function that describes an AMI on EC2. This will give insight as to the defaults that will be applied to an instance using a particular AMI.

```
$ salt-cloud -f show_image ec2 image=ami-fd20ad94
```

23.2.15 show_instance

This action is a thin wrapper around –full-query, which displays details on a single instance only. In an environment with several machines, this will save a user from having to sort through all instance data, just to examine a single instance.

```
$ salt-cloud -a show_instance myinstance
```

23.2.16 ebs_optimized

This argument enables switching of the EbsOptimized setting which default to 'false'. Indicates whether the instance is optimized for EBS I/O. This optimization provides dedicated throughput to Amazon EBS and an optimized configuration stack to provide optimal Amazon EBS I/O performance. This optimization isn't available with all instance types. Additional usage charges apply when using an EBS-optimized instance.

This setting can be added to the profile or map file for an instance.

If set to True, this setting will enable an instance to be EbsOptimized

```
ebs_optimized: True
```

This can also be set as a cloud provider setting in the EC2 cloud configuration:

```
my-ec2-config:
  ebs_optimized: True
```

23.2.17 del_root_vol_on_destroy

This argument overrides the default DeleteOnTermination setting in the AMI for the EBS root volumes for an instance. Many AMIs contain 'false' as a default, resulting in orphaned volumes in the EC2 account, which may unknowingly be charged to the account. This setting can be added to the profile or map file for an instance.

If set, this setting will apply to the root EBS volume

```
del_root_vol_on_destroy: True
```

This can also be set as a cloud provider setting in the EC2 cloud configuration:

```
my-ec2-config:
  del_root_vol_on_destroy: True
```

23.2.18 del_all_vols_on_destroy

This argument overrides the default DeleteOnTermination setting in the AMI for the not-root EBS volumes for an instance. Many AMIs contain 'false' as a default, resulting in orphaned volumes in the EC2 account, which may unknowingly be charged to the account. This setting can be added to the profile or map file for an instance.

If set, this setting will apply to any (non-root) volumes that were created by salt-cloud using the 'volumes' setting.

The volumes will not be deleted under the following conditions * If a volume is detached before terminating the instance * If a volume is created without this setting and attached to the instance

```
del_all_vols_on_destroy: True
```

This can also be set as a cloud provider setting in the EC2 cloud configuration:

```
my-ec2-config:
  del_all_vols_on_destroy: True
```

The setting for this may be changed on all volumes of an existing instance using one of the following commands:

```
salt-cloud -a delvol_on_destroy myinstance
salt-cloud -a keepvol_on_destroy myinstance
salt-cloud -a show_delvol_on_destroy myinstance
```

The setting for this may be changed on a volume on an existing instance using one of the following commands:

```
salt-cloud -a delvol_on_destroy myinstance device=/dev/sda1
salt-cloud -a delvol_on_destroy myinstance volume_id=vol-1a2b3c4d
salt-cloud -a keepvol_on_destroy myinstance device=/dev/sda1
salt-cloud -a keepvol_on_destroy myinstance volume_id=vol-1a2b3c4d
salt-cloud -a show_delvol_on_destroy myinstance device=/dev/sda1
salt-cloud -a show_delvol_on_destroy myinstance volume_id=vol-1a2b3c4d
```

23.2.19 EC2 Termination Protection

EC2 allows the user to enable and disable termination protection on a specific instance. An instance with this protection enabled cannot be destroyed. The EC2 driver adds a show_term_protect action to the regular EC2 functionality.

```
salt-cloud -a show_term_protect mymachine
salt-cloud -a enable_term_protect mymachine
salt-cloud -a disable_term_protect mymachine
```

23.2.20 Alternate Endpoint

Normally, EC2 endpoints are build using the region and the service_url. The resulting endpoint would follow this pattern:

```
ec2.<region>.<service_url>
```

This results in an endpoint that looks like:

```
ec2.us-east-1.amazonaws.com
```

There are other projects that support an EC2 compatibility layer, which this scheme does not account for. This can be overridden by specifying the endpoint directly in the main cloud configuration file:

```
my-ec2-config:
  endpoint: myendpoint.example.com:1138/services/Cloud
```

23.2.21 Volume Management

The EC2 driver has several functions and actions for management of EBS volumes.

Creating Volumes

A volume may be created, independent of an instance. A zone must be specified. A size or a snapshot may be specified (in GiB). If neither is given, a default size of 10 GiB will be used. If a snapshot is given, the size of the snapshot will be used.

```
salt-cloud -f create_volume ec2 zone=us-east-1b
salt-cloud -f create_volume ec2 zone=us-east-1b size=10
salt-cloud -f create_volume ec2 zone=us-east-1b snapshot=snap12345678
salt-cloud -f create_volume ec2 size=10 type=standard
salt-cloud -f create_volume ec2 size=10 type=io1 iops=1000
```

Attaching Volumes

Unattached volumes may be attached to an instance. The following values are required; name or instance_id, volume_id and device.

```
salt-cloud -a attach_volume myinstance volume_id=vol-12345 device=/dev/sdb1
```

Show a Volume

The details about an existing volume may be retrieved.

```
salt-cloud -a show_volume myinstance volume_id=vol-12345
salt-cloud -f show_volume ec2 volume_id=vol-12345
```

Detaching Volumes

An existing volume may be detached from an instance.

```
salt-cloud -a detach_volume myinstance volume_id=vol-12345
```

Deleting Volumes

A volume that is not attached to an instance may be deleted.

```
salt-cloud -f delete_volume ec2 volume_id=vol-12345
```

23.2.22 Managing Key Pairs

The EC2 driver has the ability to manage key pairs.

Creating a Key Pair

A key pair is required in order to create an instance. When creating a key pair with this function, the return data will contain a copy of the private key. This private key is not stored by Amazon, and will not be obtainable past this point, and should be stored immediately.

```
salt-cloud -f create_keypair ec2 keyname=mykeypair
```

Show a Key Pair

This function will show the details related to a key pair, not including the private key itself (which is not stored by Amazon).

```
salt-cloud -f show_keypair ec2 keyname=mykeypair
```

Delete a Key Pair

This function removes the key pair from Amazon.

```
salt-cloud -f delete_keypair ec2 keyname=mykeypair
```

23.3 Getting Started With Azure

New in version 2014.1.0.

Azure is a cloud service by Microsoft providing virtual machines, SQL services, media services, and more. This document describes how to use Salt Cloud to create a virtual machine on Azure, with Salt installed.

You can find more information about Azure at http://www.windowsazure.com/.

23.3.1 Dependencies

- The Azure Python SDK.
- A Microsoft Azure account
- OpenSSL (to generate the certificates)
- Salt

23.3.2 Configuration

Set up the provider config at `/etc/salt/cloud.providers.d/azure.conf`:

```
# Note: This example is for /etc/salt/cloud.providers.d/azure.conf

my-azure-config:
  provider: azure
  subscription_id: 3287abc8-f98a-c678-3bde-326766fd3617
  certificate_path: /etc/salt/azure.pem

  # Set up the location of the salt master
  #
  minion:
    master: saltmaster.example.com

  provider: gce
```

The certificate used must be generated by the user. OpenSSL can be used to create the management certificates. Two certificates are needed: a .cer file, which is uploaded to Azure, and a .pem file, which is stored locally.

To create the .pem file, execute the following command:

```
openssl req -x509 -nodes -days 365 -newkey rsa:1024 -keyout /etc/salt/azure.pem -out /etc/salt/azure
```

To create the .cer file, execute the following command:

```
openssl x509 -inform pem -in /etc/salt/azure.pem -outform der -out /etc/salt/azure.cer
```

After you creating these files, the .cer file will need to be uploaded to Azure via the “Upload” action of the “Settings” tab of the management portal.

23.3.3 Cloud Profiles

Set up an initial profile at `/etc/salt/cloud.profiles`:

```
azure-ubuntu:
  provider: my-azure-config
  image: 'b39f27a8b8c64d52b05eac6a62ebad85__Ubuntu-12_04_3-LTS-amd64-server-20131003-en-us-30GB'
  size: Small
  location: 'East US'
  ssh_username: azureuser
  ssh_password: verybadpass
  slot: production
  media_link: 'http://portalvhdabcdefghijklmn.blob.core.windows.net/vhds'
```

These options are described in more detail below. Once configured, the profile can be realized with a salt command:

```
salt-cloud -p azure-ubuntu newinstance
```

This will create an salt minion instance named `newinstance` in Azure. If the command was executed on the salt-master, its Salt key will automatically be signed on the master.

Once the instance has been created with salt-minion installed, connectivity to it can be verified with Salt:

```
salt newminion test.ping
```

23.3.4 Profile Options

The following options are currently available for Azure.

provider

The name of the provider as configured in */etc/salt/cloud.providers.d/azure.conf*.

image

The name of the image to use to create a VM. Available images can be viewed using the following command:

```
salt-cloud --list-images my-azure-config
```

size

The name of the size to use to create a VM. Available sizes can be viewed using the following command:

```
salt-cloud --list-sizes my-azure-config
```

location

The name of the location to create a VM in. Available locations can be viewed using the following command:

```
salt-cloud --list-locations my-azure-config
```

ssh_username

The user to use to log into the newly-created VM to install Salt.

ssh_password

The password to use to log into the newly-created VM to install Salt.

slot

The environment to which the hosted service is deployed. Valid values are *staging* or *production*. When set to *production*, the resulting URL of the new VM will be *<vm_name>.cloudapp.net*. When set to *staging*, the resulting URL will contain a generated hash instead.

media_link

This is the URL of the container that will store the disk that this VM uses. Currently, this container must already exist. If a VM has previously been created in the associated account, a container should already exist. In the web interface, go into the Storage area and click one of the available storage selections. Click the Containers link, and then copy the URL from the container that will be used. It generally looks like:

```
http://portalvhdabcdefghijklmn.blob.core.windows.net/vhds
```

23.3.5 Show Instance

This action is a thin wrapper around `--full-query`, which displays details on a single instance only. In an environment with several machines, this will save a user from having to sort through all instance data, just to examine a single instance.

```
salt-cloud -a show_instance myinstance
```

23.4 Writing Cloud Provider Modules

Salt Cloud runs on a module system similar to the main Salt project. The modules inside saltcloud exist in the `salt/cloud/clouds` directory of the salt source.

There are two basic types of cloud modules. If a cloud provider is supported by libcloud, then using it is the fastest route to getting a module written. The Apache Libcloud project is located at:

http://libcloud.apache.org/

Not every cloud provider is supported by libcloud. Additionally, not every feature in a supported cloud provider is necessary supported by libcloud. In either of these cases, a module can be created which does not rely on libcloud.

23.4.1 All Modules

The following functions are required by all modules, whether or not they are based on libcloud.

The __virtual__() Function

This function determines whether or not to make this cloud module available upon execution. Most often, it uses `get_configured_provider()` to determine if the necessary configuration has been set up. It may also check for necessary imports, to decide whether to load the module. In most cases, it will return a `True` or `False` value. If the name of the driver used does not match the filename, then that name should be returned instead of `True`. An example of this may be seen in the Azure module:

https://github.com/saltstack/salt/blob/develop/salt/cloud/clouds/msazure.py

The get_configured_provider() Function

This function uses `config.is_provider_configured()` to determine wither all required information for this driver has been configured. The last value in the list of required settings should be followed by a comma.

23.4.2 Libcloud Based Modules

Writing a cloud module based on libcloud has two major advantages. First of all, much of the work has already been done by the libcloud project. Second, most of the functions necessary to Salt have already been added to the Salt Cloud project.

The create() Function

The most important function that does need to be manually written is the `create()` function. This is what is used to request a virtual machine to be created by the cloud provider, wait for it to become available, and then (optionally) log in and install Salt on it.

A good example to follow for writing a cloud provider module based on libcloud is the module provided for Linode:

https://github.com/saltstack/salt/blob/develop/salt/cloud/clouds/linode.py

The basic flow of a `create()` function is as follows:

- Send a request to the cloud provider to create a virtual machine.
- Wait for the virtual machine to become available.
- Generate kwargs to be used to deploy Salt.
- Log into the virtual machine and deploy Salt.
- Return a data structure that describes the newly-created virtual machine.

At various points throughout this function, events may be fired on the Salt event bus. Four of these events, which are described below, are required. Other events may be added by the user, where appropriate.

When the `create()` function is called, it is passed a data structure called `vm_`. This dict contains a composite of information describing the virtual machine to be created. A dict called `__opts__` is also provided by Salt, which contains the options used to run Salt Cloud, as well as a set of configuration and environment variables.

The first thing the `create()` function must do is fire an event stating that it has started the create process. This event is tagged `salt/cloud/<vm name>/creating`. The payload contains the names of the VM, profile and provider.

A set of kwargs is then usually created, to describe the parameters required by the cloud provider to request the virtual machine.

An event is then fired to state that a virtual machine is about to be requested. It is tagged as `salt/cloud/<vm name>/requesting`. The payload contains most or all of the parameters that will be sent to the cloud provider. Any private information (such as passwords) should not be sent in the event.

After a request is made, a set of deploy kwargs will be generated. These will be used to install Salt on the target machine. Windows options are supported at this point, and should be generated, even if the cloud provider does not currently support Windows. This will save time in the future if the provider does eventually decide to support Windows.

An event is then fired to state that the deploy process is about to begin. This event is tagged `salt/cloud/<vm name>/deploying`. The payload for the event will contain a set of deploy kwargs, useful for debugging purposed. Any private data, including passwords and keys (including public keys) should be stripped from the deploy kwargs before the event is fired.

If any Windows options have been passed in, the `salt.utils.cloud.deploy_windows()` function will be called. Otherwise, it will be assumed that the target is a Linux or Unix machine, and the `salt.utils.cloud.deploy_script()` will be called.

Both of these functions will wait for the target machine to become available, then the necessary port to log in, then a successful login that can be used to install Salt. Minion configuration and keys will then be uploaded to a temporary directory on the target by the appropriate function. On a Windows target, the Windows Minion Installer will be run in silent mode. On a Linux/Unix target, a deploy script (bootstrap-salt.sh, by default) will be run, which will auto-detect the operating system, and install Salt using its native package manager. These do not need to be handled by the developer in the cloud module.

After the appropriate deploy function completes, a final event is fired which describes the virtual machine that has just been created. This event is tagged `salt/cloud/<vm name>/created`. The payload contains the names of the VM, profile and provider.

Finally, a dict (queried from the provider) which describes the new virtual machine is returned to the user. Because this data is not fired on the event bus it can, and should, return any passwords that were returned by the cloud provider. In some cases (for example, Rackspace), this is the only time that the password can be queried by the user; post-creation queries may not contain password information (depending upon the provider).

The libcloudfuncs Functions

A number of other functions are required for all cloud providers. However, with libcloud-based modules, these are all provided for free by the libcloudfuncs library. The following two lines set up the imports:

```
from salt.cloud.libcloudfuncs import *   # pylint: disable=W0614,W0401
from salt.utils import namespaced_function
```

And then a series of declarations will make the necessary functions available within the cloud module.

```
get_size = namespaced_function(get_size, globals())
get_image = namespaced_function(get_image, globals())
avail_locations = namespaced_function(avail_locations, globals())
avail_images = namespaced_function(avail_images, globals())
avail_sizes = namespaced_function(avail_sizes, globals())
script = namespaced_function(script, globals())
destroy = namespaced_function(destroy, globals())
list_nodes = namespaced_function(list_nodes, globals())
list_nodes_full = namespaced_function(list_nodes_full, globals())
list_nodes_select = namespaced_function(list_nodes_select, globals())
show_instance = namespaced_function(show_instance, globals())
```

If necessary, these functions may be replaced by removing the appropriate declaration line, and then adding the function as normal.

These functions are required for all cloud modules, and are described in detail in the next section.

23.4.3 Non-Libcloud Based Modules

In some cases, using libcloud is not an option. This may be because libcloud has not yet included the necessary driver itself, or it may be that the driver that is included with libcloud does not contain all of the necessary features required by the developer. When this is the case, some or all of the functions in `libcloudfuncs` may be replaced. If they are all replaced, the libcloud imports should be absent from the Salt Cloud module.

A good example of a non-libcloud provider is the Digital Ocean module:

https://github.com/saltstack/salt/blob/develop/salt/cloud/clouds/digital_ocean.py

The `create()` Function

The `create()` function must be created as described in the libcloud-based module documentation.

The get_size() Function

This function is only necessary for libcloud-based modules, and does not need to exist otherwise.

The get_image() Function

This function is only necessary for libcloud-based modules, and does not need to exist otherwise.

The avail_locations() Function

This function returns a list of locations available, if the cloud provider uses multiple data centers. It is not necessary if the cloud provider only uses one data center. It is normally called using the `--list-locations` option.

```
salt-cloud --list-locations my-cloud-provider
```

The avail_images() Function

This function returns a list of images available for this cloud provider. There are not currently any known cloud providers that do not provide this functionality, though they may refer to images by a different name (for example, "templates"). It is normally called using the `--list-images` option.

```
salt-cloud --list-images my-cloud-provider
```

The avail_sizes() Function

This function returns a list of sizes available for this cloud provider. Generally, this refers to a combination of RAM, CPU and/or disk space. This functionality may not be present on some cloud providers. For example, the Parallels module breaks down RAM, CPU and disk space into separate options, whereas in other providers, these options are baked into the image. It is normally called using the `--list-sizes` option.

```
salt-cloud --list-sizes my-cloud-provider
```

The script() Function

This function builds the deploy script to be used on the remote machine. It is likely to be moved into the `salt.utils.cloud` library in the near future, as it is very generic and can usually be copied wholesale from another module. An excellent example is in the Azure driver.

The destroy() Function

This function irreversably destroys a virtual machine on the cloud provider. Before doing so, it should fire an event on the Salt event bus. The tag for this event is `salt/cloud/<vm name>/destroying`. Once the virtual machine has been destroyed, another event is fired. The tag for that event is `salt/cloud/<vm name>/destroyed`.

This function is normally called with the `-d` options:

```
salt-cloud -d myinstance
```

The list_nodes() Function

This function returns a list of nodes available on this cloud provider, using the following fields:

- id (str)
- image (str)

- size (str)
- state (str)
- private_ips (list)
- public_ips (list)

No other fields should be returned in this function, and all of these fields should be returned, even if empty. The private_ips and public_ips fields should always be of a list type, even if empty, and the other fields should always be of a str type. This function is normally called with the `-Q` option:

```
salt-cloud -Q
```

The list_nodes_full() Function

All information available about all nodes should be returned in this function. The fields in the list_nodes() function should also be returned, even if they would not normally be provided by the cloud provider. This is because some functions both within Salt and 3rd party will break if an expected field is not present. This function is normally called with the `-F` option:

```
salt-cloud -F
```

The list_nodes_select() Function

This function returns only the fields specified in the `query.selection` option in `/etc/salt/cloud`. Because this function is so generic, all of the heavy lifting has been moved into the `salt.utils.cloud` library.

A function to call `list_nodes_select()` still needs to be present. In general, the following code can be used as-is:

```
def list_nodes_select(call=None):
    '''
    Return a list of the VMs that are on the provider, with select fields
    '''
    return salt.utils.cloud.list_nodes_select(
        list_nodes_full('function'), __opts__['query.selection'], call,
    )
```

However, depending on the cloud provider, additional variables may be required. For instance, some modules use a `conn` object, or may need to pass other options into `list_nodes_full()`. In this case, be sure to update the function appropriately:

```
def list_nodes_select(conn=None, call=None):
    '''
    Return a list of the VMs that are on the provider, with select fields
    '''
    if not conn:
        conn = get_conn()   # pylint: disable=E0602

    return salt.utils.cloud.list_nodes_select(
        list_nodes_full(conn, 'function'),
        __opts__['query.selection'],
        call,
    )
```

This function is normally called with the `-S` option:

```
salt-cloud -S
```

The show_instance() Function

This function is used to display all of the information about a single node that is available from the cloud provider. The simplest way to provide this is usually to call `list_nodes_full()`, and return just the data for the requested node. It is normally called as an action:

```
salt-cloud -a show_instance myinstance
```

23.4.4 Actions and Functions

Extra functionality may be added to a cloud provider in the form of an `--action` or a `--function`. Actions are performed against a cloud instance/virtual machine, and functions are performed against a cloud provider.

Actions

Actions are calls that are performed against a specific instance or virtual machine. The `show_instance` action should be available in all cloud modules. Actions are normally called with the `-a` option:

```
salt-cloud -a show_instance myinstance
```

Actions must accept a `name` as a first argument, may optionally support any number of kwargs as appropriate, and must accept an argument of `call`, with a default of `None`.

Before performing any other work, an action should normally verify that it has been called correctly. It may then perform the desired feature, and return useful information to the user. A basic action looks like:

```
def show_instance(name, call=None):
'''
Show the details from EC2 concerning an AMI
'''
if call != 'action':
    raise SaltCloudSystemExit(
        'The show_instance action must be called with -a or --action.'
    )

return _get_node(name)
```

Please note that generic kwargs, if used, are passed through to actions as `kwargs` and not `**kwargs`. An example of this is seen in the Functions section.

Functions

Functions are called that are performed against a specific cloud provider. An optional function that is often useful is `show_image`, which describes an image in detail. Functions are normally called with the `-f` option:

```
salt-cloud -f show_image my-cloud-provider image='Ubuntu 13.10 64-bit'
```

A function may accept any number of kwargs as appropriate, and must accept an argument of `call` with a default of `None`.

Before performing any other work, a function should normally verify that it has been called correctly. It may then perform the desired feature, and return useful information to the user. A basic function looks like:

```
def show_image(kwargs, call=None):
    '''
    Show the details from EC2 concerning an AMI
    '''
    if call != 'function':
        raise SaltCloudSystemExit(
            'The show_image action must be called with -f or --function.'
        )

    params = {'ImageId.1': kwargs['image'],
              'Action': 'DescribeImages'}
    result = query(params)
    log.info(result)

    return result
```

Take note that generic kwargs are passed through to functions as `kwargs` and not `**kwargs`.

23.5 Core Configuration

A number of core configuration options and some options that are global to the VM profiles can be set in the cloud configuration file. By default this file is located at `/etc/salt/cloud`.

23.5.1 Minion Configuration

The default minion configuration is set up in this file. This is where the minions that are created derive their configuration.

```
minion:
    master: saltmaster.example.com
```

This is the location in particular to specify the location of the salt master.

23.5.2 New Cloud Configuration Syntax

The data specific to interacting with public clouds is set up here.

ATTENTION: Since version 0.8.7 a new cloud provider configuration syntax was implemented. It will allow for multiple configurations of the same cloud provider where only minor details can change, for example, the region for an EC2 instance. While the old format is still supported and automatically migrated every time salt-cloud configuration is parsed, a choice was made to warn the user or even exit with an error if both formats are mixed.

Migrating Configurations

If you wish to migrate, there are several alternatives. Since the old syntax was mainly done on the main cloud configuration file, see the next before and after migration example.

- Before migration in `/etc/salt/cloud`:

```
AWS.id: HJGRYCILJLKJYG
AWS.key: 'kdjgfsgm;woormgl/aserigjksjdhasdfgn'
AWS.keyname: test
AWS.securitygroup: quick-start
AWS.private_key: /root/test.pem
```

- After migration in `/etc/salt/cloud`:

```
providers:
  my-aws-migrated-config:
    id: HJGRYCILJLKJYG
    key: 'kdjgfsgm;woormgl/aserigjksjdhasdfgn'
    keyname: test
    securitygroup: quick-start
    private_key: /root/test.pem
    provider: aws
```

Notice that it's not longer required to name a cloud provider's configuration after it's provider, it can be an alias, though, an additional configuration key is added, `provider`. This allows for multiple configuration for the same cloud provider to coexist.

While moving towards an improved and extensible configuration handling regarding the cloud providers, `--providers-config`, which defaults to `/etc/salt/cloud.providers`, was added to the cli parser. It allows for the cloud providers configuration to be provided in a different file, and/or even any matching file on a sub-directory, `cloud.providers.d/*.conf` which is relative to the providers configuration file(with the above configuration file as an example, `/etc/salt/cloud.providers.d/*.conf`).

So, using the example configuration above, after migration in `/etc/salt/cloud.providers` or `/etc/salt/cloud.providers.d/aws-migrated.conf`:

```
my-aws-migrated-config:
  id: HJGRYCILJLKJYG
  key: 'kdjgfsgm;woormgl/aserigjksjdhasdfgn'
  keyname: test
  securitygroup: quick-start
  private_key: /root/test.pem
  provider: aws
```

Notice that on this last migrated example, it **no longer** includes the `providers` starting key.

While migrating the cloud providers configuration, if the provider alias(from the above example `my-aws-migrated-config`) changes from what you had(from the above example `aws`), you will also need to change the `provider` configuration key in the defined profiles.

- From:

```
rhel_aws:
  provider: aws
  image: ami-e565ba8c
  size: Micro Instance
```

- To:

```
rhel_aws:
  provider: my-aws-migrated-config
  image: ami-e565ba8c
  size: Micro Instance
```

This new configuration syntax even allows you to have multiple cloud configurations under the same alias, for example:

```
production-config:
  - id: HJGRYCILJLKJYG
    key: 'kdjgfsgm;woormgl/aserigjksjdhasdfgn'
    keyname: test
    securitygroup: quick-start
    private_key: /root/test.pem

  - user: example_user
    apikey: 123984bjjas87034
    provider: rackspace
```

Notice the dash and indentation on the above example.

Having multiple entries for a configuration alias also makes the `provider` key on any defined profile to change, see the example:

```
rhel_aws_dev:
  provider: production-config:aws
  image: ami-e565ba8c
  size: Micro Instance

rhel_aws_prod:
  provider: production-config:aws
  image: ami-e565ba8c
  size: High-CPU Extra Large Instance


database_prod:
  provider: production-config:rackspace
  image: Ubuntu 12.04 LTS
  size: 256 server
```

Notice that because of the multiple entries, one has to be explicit about the provider alias and name, from the above example, `production-config:aws`.

This new syntax also changes the interaction with the `salt-cloud` binary. `--list-location`, `--list-images` and `--list-sizes` which needs a cloud provider as an argument. Since 0.8.7 the argument used should be the configured cloud provider alias. If the provider alias only as a single entry, use `<provider-alias>`. If it has multiple entries, `<provider-alias>:<provider-name>` should be used.

23.5.3 Cloud Configurations

Rackspace

Rackspace cloud requires two configuration options:

- Using the old format:

```
RACKSPACE.user: example_user
RACKSPACE.apikey: 123984bjjas87034
```

- Using the new configuration format:

```
my-rackspace-config:
  user: example_user
  apikey: 123984bjjas87034
  provider: rackspace
```

NOTE: With the new providers configuration syntax you would have `provider:  rackspace-config` instead of `provider:  rackspace` on a profile configuration.

Amazon AWS

A number of configuration options are required for Amazon AWS:

- Using the old format:

```
AWS.id: HJGRYCILJLKJYG
AWS.key: 'kdjgfsgm;woormgl/aserigjksjdhasdfgn'
AWS.keyname: test
AWS.securitygroup: quick-start
AWS.private_key: /root/test.pem
```

- Using the new configuration format:

```
my-aws-quick-start:
  id: HJGRYCILJLKJYG
  key: 'kdjgfsgm;woormgl/aserigjksjdhasdfgn'
  keyname: test
  securitygroup: quick-start
  private_key: /root/test.pem
  provider: aws

my-aws-default:
  id: HJGRYCILJLKJYG
  key: 'kdjgfsgm;woormgl/aserigjksjdhasdfgn'
  keyname: test
  securitygroup: default
  private_key: /root/test.pem
  provider: aws
```

NOTE: With the new providers configuration syntax you would have `provider:  my-aws-quick-start` or `provider:  my-aws-default` instead of `provider:  aws` on a profile configuration.

Linode

Linode requires a single API key, but the default root password also needs to be set:

- Using the old format:

```
LINODE.apikey: asldkgfakl;sdfjsjaslfjaklsdjf;askldjfaaklsjdfhasldsadfghdkf
LINODE.password: F00barbaz
```

- Using the new configuration format:

```
my-linode-config:
  apikey: asldkgfakl;sdfjsjaslfjaklsdjf;askldjfaaklsjdfhasldsadfghdkf
  password: F00barbaz
  provider: linode
```

NOTE: With the new providers configuration syntax you would have `provider:  my-linode-config` instead of `provider:  linode` on a profile configuration.

The password needs to be 8 characters and contain lowercase, uppercase and numbers.

Joyent Cloud

The Joyent cloud requires three configuration parameters. The user name and password that are used to log into the Joyent system, and the location of the private ssh key associated with the Joyent account. The ssh key is needed to send the provisioning commands up to the freshly created virtual machine,

- Using the old format:

```
JOYENT.user: fred
JOYENT.password: saltybacon
JOYENT.private_key: /root/joyent.pem
```

- Using the new configuration format:

```
my-joyent-config:
    user: fred
    password: saltybacon
    private_key: /root/joyent.pem
    provider: joyent
```

NOTE: With the new providers configuration syntax you would have `provider: my-joyent-config` instead of `provider: joyent` on a profile configuration.

GoGrid

To use Salt Cloud with GoGrid log into the GoGrid web interface and create an API key. Do this by clicking on "My Account" and then going to the API Keys tab.

The GOGRID.apikey and the GOGRID.sharedsecret configuration parameters need to be set in the configuration file to enable interfacing with GoGrid:

- Using the old format:

```
GOGRID.apikey: asdff7896asdh789
GOGRID.sharedsecret: saltybacon
```

- Using the new configuration format:

```
my-gogrid-config:
  apikey: asdff7896asdh789
  sharedsecret: saltybacon
  provider: gogrid
```

NOTE: With the new providers configuration syntax you would have `provider: my-gogrid-config` instead of `provider: gogrid` on a profile configuration.

OpenStack

OpenStack configuration differs between providers, and at the moment several options need to be specified. This module has been officially tested against the HP and the Rackspace implementations, and some examples are provided for both.

- Using the old format:

```
# For HP
OPENSTACK.identity_url: 'https://region-a.geo-1.identity.hpcloudsvc.com:35357/v2.0/'
OPENSTACK.compute_name: Compute
OPENSTACK.compute_region: 'az-1.region-a.geo-1'
OPENSTACK.tenant: myuser-tenant1
OPENSTACK.user: myuser
OPENSTACK.ssh_key_name: mykey
OPENSTACK.ssh_key_file: '/etc/salt/hpcloud/mykey.pem'
OPENSTACK.password: mypass
```

```
# For Rackspace
OPENSTACK.identity_url: 'https://identity.api.rackspacecloud.com/v2.0/tokens'
OPENSTACK.compute_name: cloudServersOpenStack
OPENSTACK.protocol: ipv4
OPENSTACK.compute_region: DFW
OPENSTACK.protocol: ipv4
OPENSTACK.user: myuser
OPENSTACK.tenant: 5555555
OPENSTACK.password: mypass
```

If you have an API key for your provider, it may be specified instead of a password:

```
OPENSTACK.apikey: 901d3f579h23c8v73q9
```

- Using the new configuration format:

```
# For HP
my-openstack-hp-config:
  identity_url:
  'https://region-a.geo-1.identity.hpcloudsvc.com:35357/v2.0/'
  compute_name: Compute
  compute_region: 'az-1.region-a.geo-1'
  tenant: myuser-tenant1
  user: myuser
  ssh_key_name: mykey
  ssh_key_file: '/etc/salt/hpcloud/mykey.pem'
  password: mypass
  provider: openstack

# For Rackspace
my-openstack-rackspace-config:
  identity_url: 'https://identity.api.rackspacecloud.com/v2.0/tokens'
  compute_name: cloudServersOpenStack
  protocol: ipv4
  compute_region: DFW
  protocol: ipv4
  user: myuser
  tenant: 5555555
  password: mypass
  provider: openstack
```

If you have an API key for your provider, it may be specified instead of a password:

```
my-openstack-hp-config:
  apikey: 901d3f579h23c8v73q9

my-openstack-rackspace-config:
  apikey: 901d3f579h23c8v73q9
```

NOTE: With the new providers configuration syntax you would have `provider: my-openstack-hp-config` or `provider: my-openstack-rackspace-config` instead of `provider: openstack` on a profile configuration.

You will certainly need to configure the `user`, `tenant` and either `password` or `apikey`.

If your OpenStack instances only have private IP addresses and a CIDR range of private addresses are not reachable from the salt-master, you may set your preference to have Salt ignore it. Using the old could configurations syntax:

```
OPENSTACK.ignore_cidr: 192.168.0.0/16
```

Using the new syntax:

```
my-openstack-config:
  ignore_cidr: 192.168.0.0/16
```

For in-house Openstack Essex installation, libcloud needs the service_type :

```
my-openstack-config:
  identity_url: 'http://control.openstack.example.org:5000/v2.0/'
  compute_name : Compute Service
  service_type : compute
```

Digital Ocean

Using Salt for Digital Ocean requires a client_key and an api_key. These can be found in the Digital Ocean web interface, in the "My Settings" section, under the API Access tab.

- Using the old format:

```
DIGITAL_OCEAN.client_key: wFGEwgregeqw3435gDger
DIGITAL_OCEAN.api_key: GDE43t43REGTrkilg43934t34qT43t4dgegerGEgg
```

- Using the new configuration format:

```
my-digitalocean-config:
  provider: digital_ocean
  client_key: wFGEwgregeqw3435gDger
  api_key: GDE43t43REGTrkilg43934t34qT43t4dgegerGEgg
  location: New York 1
```

NOTE: With the new providers configuration syntax you would have `provider: my-digitalocean-config` instead of `provider: digital_ocean` on a profile configuration.

Parallels

Using Salt with Parallels requires a user, password and url. These can be obtained from your cloud provider.

- Using the old format:

```
PARALLELS.user: myuser
PARALLELS.password: xyzzy
PARALLELS.url: https://api.cloud.xmission.com:4465/paci/v1.0/
```

- Using the new configuration format:

```
my-parallels-config:
  user: myuser
  password: xyzzy
  url: https://api.cloud.xmission.com:4465/paci/v1.0/
  provider: parallels
```

NOTE: With the new providers configuration syntax you would have `provider: my-parallels-config` instead of `provider: parallels` on a profile configuration.

IBM SmartCloud Enterprise

In addition to a username and password, the IBM SCE module requires an SSH key, which is currently configured inside IBM's web interface. A location is also required to create instances, but not to query their cloud. This is important, because you need to use salt-cloud –list-locations (with the other options already set) in order to find the name of the location that you want to use.

- Using the old format:

```
IBMSCE.user: myuser@mycorp.com
IBMSCE.password: mypass
IBMSCE.ssh_key_name: mykey
IBMSCE.ssh_key_file: '/etc/salt/ibm/mykey.pem'
IBMSCE.location: Raleigh
```

- Using the new configuration format:

```
my-ibmsce-config:
  user: myuser@mycorp.com
  password: mypass
  ssh_key_name: mykey
  ssh_key_file: '/etc/salt/ibm/mykey.pem'
  location: Raleigh
  provider: ibmsce
```

NOTE: With the new providers configuration syntax you would have `provider: my-imbsce-config` instead of `provider: ibmsce` on a profile configuration.

Saltify

The Saltify driver is a new, experimental driver for installing Salt on existing machines (virtual or bare metal). Because it does not use an actual cloud provider, it needs no configuration in the main cloud config file. However, it does still require a profile to be set up, and is most useful when used inside a map file. The key parameters to be set are `ssh_host`, `ssh_username` and either `ssh_keyfile` or `ssh_password`. These may all be set in either the profile or the map. An example configuration might use the following in cloud.profiles:

```
make_salty:
  provider: saltify
```

And in the map file:

```
make_salty:
  - myinstance:
    ssh_host: 54.262.11.38
    ssh_username: ubuntu
    ssh_keyfile: '/etc/salt/mysshkey.pem'
    sudo: True
```

23.5.4 Extending Profiles and Cloud Providers Configuration

As of 0.8.7, the option to extend both the profiles and cloud providers configuration and avoid duplication was added. The extends feature works on the current profiles configuration, but, regarding the cloud providers configuration, **only** works in the new syntax and respective configuration files, ie, `/etc/salt/salt/cloud.providers` or `/etc/salt/cloud.providers.d/*.conf`.

Extending Profiles

Some example usage on how to use `extends` with profiles. Consider `/etc/salt/salt/cloud.profiles` containing:

```
development-instances:
  provider: my-ec2-config
  size: Micro Instance
  ssh_username: ec2_user
  securitygroup:
    - default
  deploy: False

Amazon-Linux-AMI-2012.09-64bit:
  image: ami-54cf5c3d
  extends: development-instances

Fedora-17:
  image: ami-08d97e61
  extends: development-instances

CentOS-5:
  provider: my-aws-config
  image: ami-09b61d60
  extends: development-instances
```

The above configuration, once parsed would generate the following profiles data:

```
[{'deploy': False,
  'image': 'ami-08d97e61',
  'profile': 'Fedora-17',
  'provider': 'my-ec2-config',
  'securitygroup': ['default'],
  'size': 'Micro Instance',
  'ssh_username': 'ec2_user'},
 {'deploy': False,
  'image': 'ami-09b61d60',
  'profile': 'CentOS-5',
  'provider': 'my-aws-config',
  'securitygroup': ['default'],
  'size': 'Micro Instance',
  'ssh_username': 'ec2_user'},
 {'deploy': False,
  'image': 'ami-54cf5c3d',
  'profile': 'Amazon-Linux-AMI-2012.09-64bit',
  'provider': 'my-ec2-config',
  'securitygroup': ['default'],
  'size': 'Micro Instance',
  'ssh_username': 'ec2_user'},
 {'deploy': False,
  'profile': 'development-instances',
  'provider': 'my-ec2-config',
  'securitygroup': ['default'],
  'size': 'Micro Instance',
  'ssh_username': 'ec2_user'}]
```

Pretty cool right?

Extending Providers

Some example usage on how to use `extends` within the cloud providers configuration. Consider `/etc/salt/salt/cloud.providers` containing:

```
my-develop-envs:
  - id: HJGRYCILJLKJYG
    key: 'kdjgfsgm;woormgl/aserigjksjdhasdfgn'
    keyname: test
    securitygroup: quick-start
    private_key: /root/test.pem
    location: ap-southeast-1
    availability_zone: ap-southeast-1b
    provider: aws

  - user: myuser@mycorp.com
    password: mypass
    ssh_key_name: mykey
    ssh_key_file: '/etc/salt/ibm/mykey.pem'
    location: Raleigh
    provider: ibmsce


my-productions-envs:
  - extends: my-develop-envs:ibmsce
    user: my-production-user@mycorp.com
    location: us-east-1
    availability_zone: us-east-1
```

The above configuration, once parsed would generate the following providers data:

```
'providers': {
    'my-develop-envs': [
        {'availability_zone': 'ap-southeast-1b',
         'id': 'HJGRYCILJLKJYG',
         'key': 'kdjgfsgm;woormgl/aserigjksjdhasdfgn',
         'keyname': 'test',
         'location': 'ap-southeast-1',
         'private_key': '/root/test.pem',
         'provider': 'aws',
         'securitygroup': 'quick-start'
        },
        {'location': 'Raleigh',
         'password': 'mypass',
         'provider': 'ibmsce',
         'ssh_key_file': '/etc/salt/ibm/mykey.pem',
         'ssh_key_name': 'mykey',
         'user': 'myuser@mycorp.com'
        }
    ],
    'my-productions-envs': [
        {'availability_zone': 'us-east-1',
         'location': 'us-east-1',
         'password': 'mypass',
         'provider': 'ibmsce',
         'ssh_key_file': '/etc/salt/ibm/mykey.pem',
         'ssh_key_name': 'mykey',
         'user': 'my-production-user@mycorp.com'
        }
    ]
}
```

23.6 OS Support for Cloud VMs

Salt Cloud works primarily by executing a script on the virtual machines as soon as they become available. The script that is executed is referenced in the cloud profile as the `script`. In older versions, this was the `os` argument. This was changed in 0.8.2.

A number of legacy scripts exist in the deploy directory in the saltcloud source tree. The preferred method is currently to use the salt-bootstrap script. A stable version is included with each release tarball starting with 0.8.4. The most updated version can be found at:

https://github.com/saltstack/salt-bootstrap

If you do not specify a script argument, this script will be used at the default.

If the Salt Bootstrap script does not meet your needs, you may write your own. The script should be written in bash and is a Jinja template. Deploy scripts need to execute a number of functions to do a complete salt setup. These functions include:

1. Install the salt minion. If this can be done via system packages this method is HIGHLY preferred.
2. Add the salt minion keys before the minion is started for the first time. The minion keys are available as strings that can be copied into place in the Jinja template under the dict named "vm".
3. Start the salt-minion daemon and enable it at startup time.
4. Set up the minion configuration file from the "minion" data available in the Jinja template.

A good, well commented, example of this process is the Fedora deployment script:

https://github.com/saltstack/salt-cloud/blob/master/saltcloud/deploy/Fedora.sh

A number of legacy deploy scripts are included with the release tarball. None of them are as functional or complete as Salt Bootstrap, and are still included for academic purposes.

23.6.1 Other Generic Deploy Scripts

If you want to be assured of always using the latest Salt Bootstrap script, there are a few generic templates available in the deploy directory of your saltcloud source tree:

```
curl-bootstrap
curl-bootstrap-git
python-bootstrap
wget-bootstrap
wget-bootstrap-git
```

These are example scripts which were designed to be customized, adapted, and refit to meet your needs. One important use of them is to pass options to the salt-bootstrap script, such as updating to specific git tags.

23.6.2 Post-Deploy Commands

Once a minion has been deployed, it has the option to run a salt command. Normally, this would be the state.highstate command, which would finish provisioning the VM. Another common option is state.sls, or for just testing, test.ping. This is configured in the main cloud config file:

```
start_action: state.highstate
```

This is currently considered to be experimental functionality, and may not work well with all providers. If you experience problems with Salt Cloud hanging after Salt is deployed, consider using Startup States instead:

http://docs.saltstack.org/en/latest/ref/states/startup.html

23.6.3 Skipping the Deploy Script

For whatever reason, you may want to skip the deploy script altogether. This results in a VM being spun up much faster, with absolutely no configuration. This can be set from the command line:

```
salt-cloud --no-deploy -p micro_aws my_instance
```

Or it can be set from the main cloud config file:

```
deploy: False
```

Or it can be set from the provider's configuration:

```
RACKSPACE.user: example_user
RACKSPACE.apikey: 123984bjjas87034
RACKSPACE.deploy: False
```

Or even on the VM's profile settings:

```
ubuntu_aws:
  provider: aws
  image: ami-7e2da54e
  size: Micro Instance
  deploy: False
```

The default for deploy is True.

In the profile, you may also set the script option to `None`:

```
script: None
```

This is the slowest option, since it still uploads the None deploy script and executes it.

23.6.4 Updating Salt Bootstrap

Salt Bootstrap can be updated automatically with salt-cloud:

```
salt-cloud -u
salt-cloud --update-bootstrap
```

Bear in mind that this updates to the latest (unstable) version, so use with caution.

23.6.5 Keeping /tmp/ Files

When Salt Cloud deploys an instance, it uploads temporary files to /tmp/ for salt-bootstrap to put in place. After the script has run, they are deleted. To keep these files around (mostly for debugging purposes), the –keep-tmp option can be added:

```
salt-cloud -p myprofile mymachine --keep-tmp
```

For those wondering why /tmp/ was used instead of /root/, this had to be done for images which require the use of sudo, and therefore do not allow remote root logins, even for file transfers (which makes /root/ unavailable).

23.6.6 Deploy Script Arguments

Custom deploy scripts are unlikely to need custom arguments to be passed to them, but salt-bootstrap has been extended quite a bit, and this may be necessary. script_args can be specified in either the profile or the map file, to pass arguments to the deploy script:

```
aws-amazon:
    provider: aws
    image: ami-1624987f
    size: Micro Instance
    ssh_username: ec2-user
    script: bootstrap-salt
    script_args: -c /tmp/
```

This has also been tested to work with pipes, if needed:

```
script_args: | head
```

23.7 Getting Started With Digital Ocean

Digital Ocean is a public cloud provider that specializes in Linux instances.

23.7.1 Dependencies

The Digital Ocean driver requires no special dependencies outside of Salt.

23.7.2 Configuration

Using Salt for Digital Ocean requires a client_key and an api_key. These can be found in the Digital Ocean web interface, in the "My Settings" section, under the API Access tab.

```
# Note: This example is for /etc/salt/cloud.providers or any file in the
# /etc/salt/cloud.providers.d/ directory.

my-digitalocean-config:
  provider: digital_ocean
  client_key: wFGEwgregeqw3435gDger
  api_key: GDE43t43REGTrkilg43934t34qT43t4dgegerGEgg
  location: New York 1
```

23.7.3 Profiles

Cloud Profiles

Set up an initial profile at `/etc/salt/cloud.profiles` or in the `/etc/salt/cloud.profiles.d/` directory:

```
digitalocean-ubuntu:
    provider: my-digitalocean-config
    image: Ubuntu 12.10 x64
    size: 512MB
    location: New York 1
```

Sizes can be obtained using the `--list-sizes` option for the `salt-cloud` command:

```
# salt-cloud --list-sizes my-digitalocean-config
my-digitalocean-config:
    ----------
    digital_ocean:
        ----------
        512MB:
            ----------
            cost_per_hour:
                0.00744
            cost_per_month:
                5.0
            cpu:
                1
            disk:
                20
            id:
                66
            memory:
                512
            name:
                512MB
            slug:
                None
...SNIP...
```

Images can be obtained using the `--list-images` option for the `salt-cloud` command:

```
# salt-cloud --list-images my-digitalocean-config
my-digitalocean-config:
    ----------
    digital_ocean:
        ----------
        Arch Linux 2013.05 x64:
            ----------
            distribution:
                Arch Linux
            id:
                350424
            name:
                Arch Linux 2013.05 x64
            public:
                True
            slug:
                None
...SNIP...
```

Note: DigitalOcean's concept of `Applications` is nothing more than a pre-configured instance (same as a normal Droplet). You will find examples such `Docker 0.7 Ubuntu 13.04 x64` and `Wordpress on Ubuntu 12.10` when using the `--list-images` option. These names can be used just like the rest of the standard instances when specifying an image in the cloud profile configuration.

Note: Additional documentation is available from Digital Ocean.

23.8 Feature Matrix

A number of features are available in most cloud providers, but not all are available everywhere. This may be because the feature isn't supported by the cloud provider itself, or it may only be that the feature has not yet been added to Salt Cloud. In a handful of cases, it is because the feature does not make sense for a particular cloud provider (Saltify, for instance).

This matrix shows which features are available in which cloud providers, as far as Salt Cloud is concerned. This is not a comprehensive list of all features available in all cloud providers, and shoult not be used to make business decisions concerning choosing a cloud provider. In most cases, adding support for a feature to Salt Cloud requires only a little effort.

23.8.1 Legacy Drivers

Both AWS and Rackspace are listed as "Legacy". This is because those drivers have been replaced by other drivers, which are generally the prerferred method for working with those providers.

The EC2 driver should be used instead of the AWS driver, when possible. The OpenStack driver should be used instead of the Rackspace driver, unless the user is dealing with instances in "the old cloud" in Rackspace.

23.8.2 Note for Developers

When adding new features to a particular cloud provider, please make sure to add the feature to this table. Additionally, if you notice a feature that is not properly listed here, pull requests to fix them is appreciated.

23.8.3 Standard Features

These are features that are available for almost every provider.

	AWS (Legacy)	CloudStack	Digital Ocean	EC2	GoGrid	IBM SCE	Joyent	Linode	OpenStack	Parallels	Rackspace (Legacy)	Saltify	Softlayer	Softlayer Hardware
Query	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes		Yes	Yes
Full Query	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes		Yes	Yes
Selective Query	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes		Yes	Yes
List Sizes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes		Yes	Yes
List Images	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes		Yes	Yes
List Locations	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes		Yes	Yes
create	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
destroy	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes		Yes	Yes

23.8.4 Actions

These are features that are performed on a specific instance, and require an instance name to be passed in. For example:

```
# salt-cloud -a attach_volume ami.example.com
```

Actions	AWS (Legacy)	CloudStack	Digital Ocean	EC2	GoGrid	IBM SCE	JoyEnt	Linode	OpenStack	Parallels	Rackspace (Legacy)	Saltify	Softlayer	Softlayer Hardware
attach_volume				Yes										
create_attach_volumes	Yes			Yes										
del_tags	Yes			Yes										
delvol_on_destroy				Yes										
detach_volume				Yes										
disable_term_protect	Yes			Yes										
enable_term_protect	Yes			Yes										
get_tags	Yes			Yes										
keepvol_on_destroy				Yes										
list_keypairs			Yes											
rename	Yes			Yes										
set_tags	Yes			Yes										
show_delvol_on_destroy				Yes										
show_instance			Yes	Yes						Yes			Yes	Yes
show_term_protect				Yes										
start	Yes			Yes			Yes			Yes				
stop	Yes			Yes			Yes			Yes				
take_action							Yes							

23.8.5 Functions

These are features that are performed against a specific cloud provider, and require the name of the provider to be passed in. For example:

```
# salt-cloud -f list_images my_digitalocean
```

Functions	AWS (Legacy)	CloudStack	Digital Ocean	EC2	GoGrid	IBM SCE	JoyEnt	Linoc
block_device_mappings	Yes							
create_keypair				Yes				
create_volume				Yes				
delete_key							Yes	
delete_keypair				Yes				
delete_volume				Yes				
get_image			Yes				Yes	
get_ip		Yes						
get_key		Yes						
get_keyid			Yes					

Table 23.1 – continued from previ

Functions	AWS (Legacy)	CloudStack	Digital Ocean	EC2	GoGrid	IBM SCE	JoyEnt	Lino
get_keypair		Yes						
get_networkid		Yes						
get_node							Yes	
get_password		Yes						
get_size			Yes				Yes	
get_spot_config				Yes				
get_subnetid				Yes				
iam_profile	Yes			Yes				
import_key							Yes	
key_list							Yes	
keyname	Yes			Yes				
list_availability_zones				Yes				
list_custom_images								
list_keys							Yes	
list_vlans								
rackconnect								
reboot				Yes			Yes	
reformat_node							Yes	
securitygroup	Yes			Yes				
securitygroupid				Yes				
show_image				Yes				
show_key							Yes	
show_keypair			Yes	Yes				
show_volume				Yes				

23.9 Cloud Functions

Cloud functions work much the same way as cloud actions, except that they don't perform an operation on a specific instance, and so do not need a machine name to be specified. However, since they perform an operation on a specific cloud provider, that provider must be specified.

```
$ salt-cloud -f show_image ec2 image=ami-fd20ad94
```

23.10 Getting Started With Google Compute Engine

Google Compute Engine (GCE) is Google-infrastructure as a service that lets you run your large-scale computing workloads on virtual machines. This document covers how to use Salt Cloud to provision and manage your virtual machines hosted within Google's infrastructure.

You can find out more about GCE and other Google Cloud Platform services at https://cloud.google.com.

23.10.1 Dependencies

- Source install of Libcloud (or greater than 0.14.0-beta3 when available)
- PyCrypto >= 2.1.
- A Google Cloud Platform account with Compute Engine enabled
- A registered Service Account for authorization

- Oh, and obviously you'll need salt

23.10.2 Google Compute Engine Setup

1. Sign up for Google Cloud Platform Go to https://cloud.google.com and use your Google account to sign up for Google Cloud Platform and complete the guided instructions.
2. Create a Project Next, go to the console at https://cloud.google.com/console and create a new Project. Make sure to select your new Project if you are not automatically directed to the Project.

   Projects are a way of grouping together related users, services, and billing. You may opt to create multiple Projects and the remaining instructions will need to be completed for each Project if you wish to use GCE and Salt Cloud to manage your virtual machines.
3. Enable the Google Compute Engine service In your Project, go to the *APIs & auth* section and *APIs* link and enable the Google Compute Engine service.
4. Create a Service Account To set up authorization, navigate to *APIs & auth* section and then the *Registered apps* link and click the *REGISTER APP* button. Give it a meaningful name like and select *Web Application*. After clicking the *Register* button, select *Certificate* in the next screen. Click the *Generate Certificate* button, record the generated email address for use in the `service_account_email_address` of your `/etc/salt/cloud` file. Also download and save the generated private key.
5. You will need to convert the private key to a format compatible with libcloud. The original Google-generated private key was encrypted using *notasecret* as a passphrase. Use the following command and record the location of the converted private key and record the location for use in the `service_account_private_key` of your `/etc/salt/cloud` file:

   ```
   openssl pkcs12 -in ORIG.pkey -passin pass:notasecret \
   -nodes -nocerts | openssl rsa -out NEW.pem
   ```

23.10.3 Configuration

Set up the cloud config at `/etc/salt/cloud`:

```
# Note: This example is for /etc/salt/cloud

providers:
  gce-config:
    # Set up the Project name and Service Account authorization
    #
    project: "your_project_name"
    service_account_email_address: "123-a5gt@developer.gserviceaccount.com"
    service_account_private_key: "/path/to/your/NEW.pem"

    # Set up the location of the salt master
    #
    minion:
      master: saltmaster.example.com

    # Set up grains information, which will be common for all nodes
    # using this provider
    grains:
      node_type: broker
      release: 1.0.1

    provider: gce
```

23.10.4 Cloud Profiles

Set up an initial profile at `/etc/salt/cloud.profiles`:

```
all_settings:
  image: centos-6
  size: n1-standard-1
  location: europe-west1-b
  network: default
  tags: '["one", "two", "three"]'
  metadata: '{"one": "1", "2": "two"}'
  use_persistent_disk: True
  delete_boot_pd: False
  deploy: True
  make_master: False
  provider: gce-config
```

The profile can be realized now with a salt command:

```
salt-cloud -p all_settings gce-instance
```

This will create an salt minion instance named `gce-instance` in GCE. If the command was executed on the salt-master, its Salt key will automatically be signed on the master.

Once the instance has been created with salt-minion installed, connectivity to it can be verified with Salt:

```
salt 'ami.example.com' test.ping
```

23.10.5 GCE Specific Settings

Consult the sample profile below for more information about GCE specific settings. Some of them are mandatory and are properly labeled below but typically also include a hard-coded default.

```
all_settings:

  # Image is used to define what Operating System image should be used
  # to for the instance.  Examples are Debian 7 (wheezy) and CentOS 6.
  #
  # MANDATORY
  #
  image: centos-6

  # A 'size', in GCE terms, refers to the instance's 'machine type'.  See
  # the on-line documentation for a complete list of GCE machine types.
  #
  # MANDATORY
  #
  size: n1-standard-1

  # A 'location', in GCE terms, refers to the instance's 'zone'.  GCE
  # has the notion of both Regions (e.g. us-central1, europe-west1, etc)
  # and Zones (e.g. us-central1-a, us-central1-b, etc).
  #
  # MANDATORY
  #
  location: europe-west1-b

  # Use this setting to define the network resource for the instance.
  # All GCE projects contain a network named 'default' but it's possible
  # to use this setting to create instances belonging to a different
  # network resource.
  #
  network: default

  # GCE supports instance/network tags and this setting allows you to
  # set custom tags.  It should be a list of strings and must be
  # parse-able by the python ast.literal_eval() function to convert it
  # to a python list.
  #
  tags: '["one", "two", "three"]'

  # GCE supports instance metadata and this setting allows you to
  # set custom metadata.  It should be a hash of key/value strings and
  # parse-able by the python ast.literal_eval() function to convert it
  # to a python dictionary.
  #
  metadata: '{"one": "1", "2": "two"}'

  # Use this setting to ensure that when new instances are created,
  # they will use a persistent disk to preserve data between instance
  # terminations and re-creations.
  #
  use_persistent_disk: True

  # In the event that you wish the boot persistent disk to be permanently
  # deleted when you destroy an instance, set delete_boot_pd to True.
  #
  delete_boot_pd: False
```

GCE instances do not allow remote access to the root user by default. Instead, another user must be used to run the deploy script using sudo.

```
my-gce-config:
  # Configure which user to use to run the deploy script
  ssh_username: user
  ssh_keyfile: /home/user/.ssh/google_compute_engine
```

23.10.6 Single instance details

This action is a thin wrapper around `--full-query`, which displays details on a single instance only. In an environment with several machines, this will save a user from having to sort through all instance data, just to examine a single instance.

```
salt-cloud -a show_instance myinstance
```

23.10.7 Destroy, persistent disks, and metadata

As noted in the provider configuration, it's possible to force the boot persistent disk to be deleted when you destroy the instance. The way that this has been implemented is to use the instance metadata to record the cloud profile used when creating the instance. When `destroy` is called, if the instance contains a `salt-cloud-profile` key, it's value is used to reference the matching profile to determine if `delete_boot_pd` is set to `True`.

Be aware that any GCE instances created with salt cloud will contain this custom `salt-cloud-profile` metadata entry.

23.10.8 List various resources

It’s also possible to list several GCE resources similar to what can be done with other providers. The following commands can be used to list GCE zones (locations), machine types (sizes), and images.

```
salt-cloud --list-locations gce
salt-cloud --list-sizes gce
salt-cloud --list-images gce
```

23.11 Getting Started With GoGrid

GoGrid is a public cloud provider supporting Linux and Windows.

23.11.1 Dependencies

The GoGrid driver for Salt Cloud requires Libcloud 0.13.2 or higher to be installed.

23.11.2 Configuration

To use Salt Cloud with GoGrid log into the GoGrid web interface and create an API key. Do this by clicking on “My Account” and then going to the API Keys tab.

The `apikey` and the `sharedsecret` configuration parameters need to be set in the configuration file to enable interfacing with GoGrid:

```
# Note: This example is for /etc/salt/cloud.providers or any file in the
# /etc/salt/cloud.providers.d/ directory.

my-gogrid-config:
  provider: gogrid
  apikey: asdff7896asdh789
  sharedsecret: saltybacon
```

23.11.3 Profiles

Cloud Profiles

Set up an initial profile at `/etc/salt/cloud.profiles` or in the `/etc/salt/cloud.profiles.d/` directory:

```
gogrid_512:
  provider: my-gogrid-config
  size: 512MB
  image: CentOS 6.2 (64-bit) w/ None
```

Sizes can be obtained using the `--list-sizes` option for the `salt-cloud` command:

```
# salt-cloud --list-sizes my-gogrid-config
my-gogrid-config:
    ----------
    gogrid:
        ----------
        512MB:
            ----------
            bandwidth:
                None
            disk:
                30
            driver:
            get_uuid:
            id:
                512MB
            name:
                512MB
            price:
                0.095
            ram:
                512
            uuid:
                bde1e4d7c3a643536e42a35142c7caac34b060e9
...SNIP...
```

Images can be obtained using the `--list-images` option for the `salt-cloud` command:

```
# salt-cloud --list-images my-gogrid-config
my-gogrid-config:
    ----------
    gogrid:
        ----------
        CentOS 6.4 (64-bit) w/ None:
            ----------
            driver:
            extra:
                ----------
            get_uuid:
            id:
                18094
            name:
                CentOS 6.4 (64-bit) w/ None
            uuid:
                bfd4055389919e01aa6261828a96cf54c8dcc2c4
...SNIP...
```

23.12 Getting Started With Joyent

Joyent is a public cloud provider supporting SmartOS, Linux, FreeBSD and Windows.

23.12.1 Dependencies

The Joyent driver for Salt Cloud requires Libcloud 0.13.2 or higher to be installed.

23.12.2 Configuration

The Joyent cloud requires three configuration parameters. The user name and password that are used to log into the Joyent system, and the location of the private ssh key associated with the Joyent account. The ssh key is needed to send the provisioning commands up to the freshly created virtual machine.

```
# Note: This example is for /etc/salt/cloud.providers or any file in the
# /etc/salt/cloud.providers.d/ directory.

my-joyent-config:
    provider: joyent
    user: fred
    password: saltybacon
    private_key: /root/joyent.pem
```

23.12.3 Profiles

Cloud Profiles

Set up an initial profile at `/etc/salt/cloud.profiles` or in the `/etc/salt/cloud.profiles.d/` directory:

```
joyent_512
  provider: my-joyent-config
  size: Extra Small 512 MB
  image: Arch Linux 2013.06
```

Sizes can be obtained using the `--list-sizes` option for the `salt-cloud` command:

```
# salt-cloud --list-sizes my-joyent-config
my-joyent-config:
    ----------
    joyent:
        ----------
        Extra Small 512 MB:
            ----------
            default:
                false
            disk:
                15360
            id:
                Extra Small 512 MB
            memory:
                512
            name:
                Extra Small 512 MB
            swap:
                1024
            vcpus:
                1
...SNIP...
```

Images can be obtained using the `--list-images` option for the `salt-cloud` command:

```
# salt-cloud --list-images my-joyent-config
my-joyent-config:
    ----------
    joyent:
        ----------
        base:
            ----------
            description:
                A 32-bit SmartOS image with just essential packages
                installed. Ideal for users who are comfortable with setting
                up their own environment and tools.
            disabled:
                False
            files:
                ----------
                - compression:
                    bzip2
                - sha1:
                    40cdc6457c237cf6306103c74b5f45f5bf2d9bbe
                - size:
                    82492182
            name:
                base
            os:
                smartos
            owner:
                352971aa-31ba-496c-9ade-a379feaecd52
            public:
                True
...SNIP...
```

23.13 Getting Started With Linode

Linode is a public cloud provider with a focus on Linux instances.

23.13.1 Dependencies

The Linode driver for Salt Cloud requires Libcloud 0.13.2 or higher to be installed.

23.13.2 Configuration

Linode requires a single API key, but the default root password for new instances also needs to be set:

```
# Note: This example is for /etc/salt/cloud.providers or any file in the
# /etc/salt/cloud.providers.d/ directory.

my-linode-config:
  apikey: asldkgfakl;sdfjsjaslfjaklsdjf;askldjfaaklsjdfhasldsadfghdkf
  password: F00barbaz
  provider: linode
```

The password needs to be 8 characters and contain lowercase, uppercase and numbers.

23.13.3 Profiles

Cloud Profiles

Set up an initial profile at `/etc/salt/cloud.profiles` or in the `/etc/salt/cloud.profiles.d/` directory:

```
linode_1024:
  provider: my-linode-config
  size: Linode 1024
  image: Arch Linux 2013.06
```

Sizes can be obtained using the `--list-sizes` option for the `salt-cloud` command:

```
# salt-cloud --list-sizes my-linode-config
my-linode-config:
    ----------
    linode:
        ----------
        Linode 1024:
            ----------
            bandwidth:
                2000
            disk:
                49152
            driver:
            get_uuid:
            id:
                1
            name:
                Linode 1024
            price:
                20.0
            ram:
                1024
            uuid:
                03e18728ce4629e2ac07c9cbb48afffb8cb499c4
...SNIP...
```

Images can be obtained using the `--list-images` option for the `salt-cloud` command:

```
# salt-cloud --list-images my-linode-config
my-linode-config:
    ----------
    linode:
        ----------
        Arch Linux 2013.06:
            ----------
            driver:
            extra:
                ----------
                64bit:
                    1
                pvops:
                    1
            get_uuid:
            id:
                112
            name:
                Arch Linux 2013.06
            uuid:
                8457f92eaffc92b7666b6734a96ad7abe1a8a6dd
...SNIP...
```

23.14 Cloud Map File

A number of options exist when creating virtual machines. They can be managed directly from profiles and the command line execution, or a more complex map file can be created. The map file allows for a number of virtual machines to be created and associated with specific profiles.

Map files have a simple format, specify a profile and then a list of virtual machines to make from said profile:

```
fedora_small:
  - web1
  - web2
  - web3
fedora_high:
  - redis1
  - redis2
  - redis3
cent_high:
  - riak1
  - riak2
  - riak3
```

This map file can then be called to roll out all of these virtual machines. Map files are called from the salt-cloud command with the -m option:

```
$ salt-cloud -m /path/to/mapfile
```

Remember, that as with direct profile provisioning the -P option can be passed to create the virtual machines in parallel:

```
$ salt-cloud -m /path/to/mapfile -P
```

A map file can also be enforced to represent the total state of a cloud deployment by using the `--hard` option. When using the hard option any vms that exist but are not specified in the map file will be destroyed:

```
$ salt-cloud -m /path/to/mapfile -P -H
```

Be careful with this argument, it is very dangerous! In fact, it is so dangerous that in order to use it, you must explicitly enable it in the main configuration file.

```
enable_hard_maps: True
```

A map file can include grains and minion configuration options:

```
fedora_small:
  - web1:
      minion:
        log_level: debug
      grains:
        cheese: tasty
        omelet: du fromage
  - web2:
      minion:
        log_level: warn
      grains:
        cheese: more tasty
        omelet: with peppers
```

A map file may also be used with the various query options:

```
$ salt-cloud -m /path/to/mapfile -Q
{'ec2': {'web1': {'id': 'i-e6aqfegb',
                     'image': None,
                     'private_ips': [],
                     'public_ips': [],
                     'size': None,
                     'state': 0}},
         'web2': {'Absent'}}
```

...or with the delete option:

```
$ salt-cloud -m /path/to/mapfile -d
The following virtual machines are set to be destroyed:
  web1
  web2

Proceed? [N/y]
```

23.15 Miscellaneous Salt Cloud Options

This page describes various miscellaneous options available in Salt Cloud

23.15.1 Deploy Script Arguments

Custom deploy scripts are unlikely to need custom arguments to be passed to them, but salt-bootstrap has been extended quite a bit, and this may be necessary. script_args can be specified in either the profile or the map file, to pass arguments to the deploy script:

```
ec2-amazon:
    provider: ec2
    image: ami-1624987f
    size: Micro Instance
    ssh_username: ec2-user
    script: bootstrap-salt
    script_args: -c /tmp/
```

This has also been tested to work with pipes, if needed:

```
script_args: | head
```

23.15.2 Sync After Install

Salt allows users to create custom modules, grains and states which can be synchronised to minions to extend Salt with further functionality.

This option will inform Salt Cloud to synchronise your custom modules, grains, states or all these to the minion just after it has been created. For this to happen, the following line needs to be added to the main cloud configuration file:

```
sync_after_install: all
```

The available options for this setting are:

```
modules
grains
states
all
```

23.15.3 Setting up New Salt Masters

It has become increasingly common for users to set up multi-hierarchal infrastructures using Salt Cloud. This sometimes involves setting up an instance to be a master in addition to a minion. With that in mind, you can now lay down master configuration on a machine by specifying master options in the profile or map file.

```
make_master: True
```

This will cause Salt Cloud to generate master keys for the instance, and tell salt-bootstrap to install the salt-master package, in addition to the salt-minion package.

The default master configuration is usually appropriate for most users, and will not be changed unless specific master configuration has been added to the profile or map:

```
master:
    user: root
    interface: 0.0.0.0
```

23.15.4 Delete SSH Keys

When Salt Cloud deploys an instance, the SSH pub key for the instance is added to the known_hosts file for the user that ran the salt-cloud command. When an instance is deployed, a cloud provider generally recycles the IP address for the instance. When Salt Cloud attempts to deploy an instance using a recycled IP address that has previously been accessed from the same machine, the old key in the known_hosts file will cause a conflict.

In order to mitigate this issue, Salt Cloud can be configured to remove old keys from the known_hosts file when destroying the node. In order to do this, the following line needs to be added to the main cloud configuration file:

```
delete_sshkeys: True
```

23.15.5 Keeping /tmp/ Files

When Salt Cloud deploys an instance, it uploads temporary files to /tmp/ for salt-bootstrap to put in place. After the script has run, they are deleted. To keep these files around (mostly for debugging purposes), the –keep-tmp option can be added:

```
salt-cloud -p myprofile mymachine --keep-tmp
```

For those wondering why /tmp/ was used instead of /root/, this had to be done for images which require the use of sudo, and therefore do not allow remote root logins, even for file transfers (which makes /root/ unavailable).

23.15.6 Hide Output From Minion Install

By default Salt Cloud will stream the output from the minion deploy script directly to STDOUT. Although this can been very useful, in certain cases you may wish to switch this off. The following config option is there to enable or disable this output:

```
display_ssh_output: False
```

23.15.7 Connection Timeout

There are several stages when deploying Salt where Salt Cloud needs to wait for something to happen. The VM getting it's IP address, the VM's SSH port is available, etc.

If you find that the Salt Cloud defaults are not enough and your deployment fails because Salt Cloud did not wait log enough, there are some settings you can tweak.

Note

All values should be provided in seconds

You can tweak these settings globally, per cloud provider, or event per profile definition.

wait_for_ip_timeout

The amount of time Salt Cloud should wait for a VM to start and get an IP back from the cloud provider. Default: 5 minutes.

wait_for_ip_interval

The amount of time Salt Cloud should sleep while querying for the VM's IP. Default: 5 seconds.

ssh_connect_timeout

The amount of time Salt Cloud should wait for a successful SSH connection to the VM. Default: 5 minutes.

wait_for_passwd_timeout

The amount of time until an ssh connection can be established via password or ssh key. Default 15 seconds.

wait_for_fun_timeout

Some cloud drivers check for an available IP or a successful SSH connection using a function, namely, SoftLayer and SoftLayer-HW. So, the amount of time Salt Cloud should retry such functions before failing. Default: 5 minutes.

wait_for_spot_timeout

The amount of time Salt Cloud should wait before an EC2 Spot instance is available. This setting is only available for the EC2 cloud driver.

23.16 Getting Started With OpenStack

OpenStack is one the most popular cloud projects. It's an open source project to build public and/or private clouds. You can use Salt Cloud to launch OpenStack instances.

- Using the new format, set up the cloud configuration at `/etc/salt/cloud.providers` or `/etc/salt/cloud.providers.d/openstack.conf`:

```
my-openstack-config:
  # Set the location of the salt-master
  #
  minion:
    master: saltmaster.example.com

  # Configure the OpenStack driver
  #
  identity_url: http://identity.youopenstack.com/v2.0/
  compute_name: nova
  protocol: ipv4

  compute_region: RegionOne

  # Configure Openstack authentication credentials
  #
  user: myname
  password: 123456
  # tenant is the project name
  tenant: myproject

  provider: openstack
```

23.16.1 Using nova client to get information from OpenStack

One of the best ways to get information about OpenStack is using the novaclient python package (available in pypi as python-novaclient). The client configuration is a set of environment variables that you can get from the Dashboard. Log in and then go to Project -> Access & security -> API Access and download the "OpenStack RC file". Then:

```
source /path/to/your/rcfile
nova credentials
nova endpoints
```

In the `nova endpoints` output you can see the information about `compute_region` and `compute_name`.

23.16.2 Compute Region

It depends on the OpenStack cluster that you are using. Please, have a look at the previous sections.

23.16.3 Authentication

The `user` and `password` is the same user as is used to log into the OpenStack Dashboard.

23.16.4 Profiles

Here is an example of a profile:

```
openstack_512:
    provider: my-openstack-config
    size: m1.tiny
    image: cirros-0.3.1-x86_64-uec
    ssh_key_file: /tmp/test.pem
```

`size` can be one of the options listed in the output of `nova flavor-list`.

`image` can be one of the options listed in the output of `nova image-list`.

23.17 Getting Started With Parallels

Parallels Cloud Server is a product by Parallels that delivers a cloud hosting solution. The PARALLELS module for Salt Cloud enables you to manage instances hosted by a provider using PCS. Further information can be found at:

http://www.parallels.com/products/pcs/

- Using the old format, set up the cloud configuration at `/etc/salt/cloud`:

```
# Set up the location of the salt master
#
minion:
    master: saltmaster.example.com

# Set the PARALLELS access credentials (see below)
#
PARALLELS.user: myuser
PARALLELS.password: badpass

# Set the access URL for your PARALLELS provider
#
PARALLELS.url: https://api.cloud.xmission.com:4465/paci/v1.0/
```

- Using the new format, set up the cloud configuration at `/etc/salt/cloud.providers` or `/etc/salt/cloud.providers.d/parallels.conf`:

```
my-parallels-config:
  # Set up the location of the salt master
  #
  minion:
    master: saltmaster.example.com

  # Set the PARALLELS access credentials (see below)
  #
  user: myuser
  password: badpass

  # Set the access URL for your PARALLELS provider
  #
  url: https://api.cloud.xmission.com:4465/paci/v1.0/
  provider: parallels
```

23.17.1 Access Credentials

The `user`, `password` and `url` will be provided to you by your cloud provider. These are all required in order for the PARALLELS driver to work.

23.17.2 Cloud Profiles

Set up an initial profile at `/etc/salt/cloud.profiles` or `/etc/salt/cloud.profiles.d/parallels.conf`:

- Using the old cloud configuration format:

```
parallels-ubuntu:
    provider: parallels
    image: ubuntu-12.04-x86_64
```

- Using the new cloud configuration format and the cloud configuration example from above:

```
parallels-ubuntu:
    provider: my-parallels-config
    image: ubuntu-12.04-x86_64
```

The profile can be realized now with a salt command:

```
# salt-cloud -p parallels-ubuntu myubuntu
```

This will create an instance named `myubuntu` on the cloud provider. The minion that is installed on this instance will have an `id` of `myubuntu`. If the command was executed on the salt-master, its Salt key will automatically be signed on the master.

Once the instance has been created with salt-minion installed, connectivity to it can be verified with Salt:

```
# salt myubuntu test.ping
```

23.17.3 Required Settings

The following settings are always required for PARALLELS:

- Using the old cloud configuration format:

```
PARALLELS.user: myuser
PARALLELS.password: badpass
PARALLELS.url: https://api.cloud.xmission.com:4465/paci/v1.0/
```

- Using the new cloud configuration format:

```
my-parallels-config:
  user: myuser
  password: badpass
  url: https://api.cloud.xmission.com:4465/paci/v1.0/
  provider: parallels
```

23.17.4 Optional Settings

Unlike other cloud providers in Salt Cloud, Parallels does not utilize a `size` setting. This is because Parallels allows the end-user to specify a more detailed configuration for their instances, than is allowed by many other cloud providers. The following options are available to be used in a profile, with their default settings listed.

```
# Description of the instance. Defaults to the instance name.
desc: <instance_name>

# How many CPU cores, and how fast they are (in MHz)
cpu_number: 1
cpu_power: 1000

# How many megabytes of RAM
ram: 256

# Bandwidth available, in kbps
bandwidth: 100

# How many public IPs will be assigned to this instance
ip_num: 1

# Size of the instance disk (in GiB)
disk_size: 10

# Username and password
ssh_username: root
password: <value from PARALLELS.password>

# The name of the image, from ''salt-cloud --list-images parallels''
image: ubuntu-12.04-x86_64
```

23.18 VM Profiles

Salt cloud designates virtual machines inside the profile configuration file. The profile configuration file defaults to `/etc/salt/cloud.profiles` and is a yaml configuration. The syntax for declaring profiles is simple:

```
fedora_rackspace:
    provider: rackspace
    image: Fedora 17
    size: 256 server
    script: Fedora
```

A few key pieces of information need to be declared and can change based on the public cloud provider. A number of additional parameters can also be inserted:

```
centos_rackspace:
    provider: rackspace
    image: CentOS 6.2
    size: 1024 server
    script: RHEL6
    minion:
        master: salt.example.com
        append_domain: webs.example.com
        grains:
            role: webserver
```

The image must be selected from available images. Similarly, sizes must be selected from the list of sizes. To get a list of available images and sizes use the following command:

```
salt-cloud --list-images openstack
salt-cloud --list-sizes openstack
```

Some parameters can be specified in the main Salt cloud configuration file and then are applied to all cloud profiles. For instance if only a single cloud provider is being used then the provider option can be declared in the Salt cloud configuration file.

23.18.1 Multiple Configuration Files

In addition to `/etc/salt/cloud.profiles`, profiles can also be specified in any file matching `cloud.profiles.d/*conf` which is a sub-directory relative to the profiles configuration file(with the above configuration file as an example, `/etc/salt/cloud.profiles.d/*.conf`). This allows for more extensible configuration, and plays nicely with various configuration management tools as well as version control systems.

23.18.2 Larger Example

```
rhel_ec2:
    provider: ec2
    image: ami-e565ba8c
    size: Micro Instance
    script: RHEL6
    minion:
        cheese: edam

ubuntu_ec2:
    provider: ec2
    image: ami-7e2da54e
    size: Micro Instance
    script: Ubuntu
    minion:
        cheese: edam

ubuntu_rackspace:
    provider: rackspace
    image: Ubuntu 12.04 LTS
    size: 256 server
    script: Ubuntu
    minion:
        cheese: edam

fedora_rackspace:
    provider: rackspace
    image: Fedora 17
    size: 256 server
    script: Fedora
    minion:
        cheese: edam

cent_linode:
    provider: linode
    image: CentOS 6.2 64bit
    size: Linode 512
    script: RHEL6

cent_gogrid:
    provider: gogrid
    image: 12834
    size: 512MB
    script: RHEL6

cent_joyent:
    provider: joyent
    image: centos-6
    script: RHEL6
    size: Small 1GB
```

23.19 Getting Started With Rackspace

Rackspace is a major public cloud platform which may be configured using either the *rackspace* or the *openstack* driver, depending on your needs.

Please note that the *rackspace* driver is only intended for 1st gen instances, aka, "the old cloud" at Rackspace. It is required for 1st gen instances, but will *not* work with OpenStack-based instances. Unless you explicitly have a reason to use it, it is highly recommended that you use the *openstack* driver instead.

To use the *openstack* driver (recommended), set up the cloud configuration at `/etc/salt/cloud.providers` or `/etc/salt/cloud.providers.d/rackspace.conf`:

```
my-rackspace-config:
  # Set the location of the salt-master
  #
  minion:
    master: saltmaster.example.com

  # Configure Rackspace using the OpenStack plugin
  #
  identity_url: 'https://identity.api.rackspacecloud.com/v2.0/tokens'
  compute_name: cloudServersOpenStack
  protocol: ipv4

  # Set the compute region:
  #
  compute_region: DFW

  # Configure Rackspace authentication credentials
  #
  user: myname
  tenant: 123456
  apikey: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

  provider: openstack
```

To use the *rackspace* driver, set up the cloud configuration at `/etc/salt/cloud.providers` or `/etc/salt/cloud.providers.d/rackspace.conf`:

```
my-rackspace-config:
  provider: rackspace
  # The Rackspace login user
  user: fred
  # The Rackspace user's apikey
  apikey: 901d3f579h23c8v73q9
```

The settings that follow are for using Rackspace with the *openstack* driver, and will not work with the *rackspace* driver.

23.19.1 Compute Region

Rackspace currently has six compute regions which may be used:

```
DFW -> Dallas/Forth Worth
ORD -> Chicago
SYD -> Sydney
LON -> London
IAD -> Northern Virginia
HKG -> Hong Kong
```

Note: Currently the LON region is only avaiable with a UK account, and UK accounts cannot access other regions

23.19.2 Authentication

The `user` is the same user as is used to log into the Rackspace Control Panel. The `tenant` and `apikey` can be found in the API Keys area of the Control Panel. The `apikey` will be labeled as API Key (and may need to be generated), and `tenant` will be labeled as Cloud Account Number.

An initial profile can be configured in `/etc/salt/cloud.profiles` or `/etc/salt/cloud.profiles.d/rackspace.conf`:

```
openstack_512:
    provider: my-rackspace-config
    size: 512 MB Standard
    image: Ubuntu 12.04 LTS (Precise Pangolin)
```

To instantiate a machine based on this profile:

```
# salt-cloud -p openstack_512 myinstance
```

This will create a virtual machine at Rackspace with the name `myinstance`. This operation may take several minutes to complete, depending on the current load at the Rackspace data center.

Once the instance has been created with salt-minion installed, connectivity to it can be verified with Salt:

```
# salt myinstance test.ping
```

RackConnect Environments

Rackspace offers a hybrid hosting configuration option called RackConnect that allows you to use a physical firewall appliance with your cloud servers. When this service is in use the public_ip assigned by nova will be replaced by a NAT ip on the firewall. For salt-cloud to work properly it must use the newly assigned “access ip” instead of the Nova assigned public ip. You can enable that capability by adding this to your profiles:

```
openstack_512:
    provider: my-openstack-config
    size: 512 MB Standard
    image: Ubuntu 12.04 LTS (Precise Pangolin)
    rackconnect: True
```

Managed Cloud Environments

Rackspace offers a managed service level of hosting. As part of the managed service level you have the ability to choose from base of lamp installations on cloud server images. The post build process for both the base and the lamp installations used Chef to install things such as the cloud monitoring agent and the cloud backup agent. It also takes care of installing the lamp stack if selected. In order to prevent the post installation process from stomping over the bootstrapping you can add the below to your profiles.

```
openstack_512:
    provider: my-rackspace-config
    size: 512 MB Standard
    image: Ubuntu 12.04 LTS (Precise Pangolin)
    managedcloud: True
```

First and Next Generation Images

Rackspace provides two sets of virtual machine images, *first* and *next* generation. As of `0.8.9` salt-cloud will default to using the *next* generation images. To force the use of first generation images, on the profile configuration please add:

```
FreeBSD-9.0-512:
  provider: my-rackspace-config
  size: 512 MB Standard
  image: FreeBSD 9.0
  force_first_gen: True
```

23.20 Using the Salt Modules for Cloud

In addition to the `salt-cloud` command, Salt Cloud can be called from Salt, in a variety of different ways. Most users will be interested in either the execution module or the state module, but it is also possible to call Salt Cloud as a runner.

Because the actual work will be performed on a remote minion, the normal Salt Cloud configuration must exist on any target minion that needs to execute a Salt Cloud command. Because Salt Cloud now supports breaking out configuration into individual files, the configuration is easily managed using Salt's own `file.managed` state function. For example, the following directories allow this configuration to be managed easily:

```
/etc/salt/cloud.providers.d/
/etc/salt/cloud.profiles.d/
```

23.20.1 Minion Keys

Keep in mind that when creating minions, Salt Cloud will create public and private minion keys, upload them to the minion, and place the public key on the machine that created the minion. It will *not* attempt to place any public minion keys on the master, unless the minion which was used to create the instance is also the Salt Master. This is because granting arbitrary minions access to modify keys on the master is a serious security risk, and must be avoided.

23.20.2 Execution Module

The `cloud` module is available to use from the command line. At the moment, almost every standard Salt Cloud feature is available to use. The following commands are available:

list_images

This command is designed to show images that are available to be used to create an instance using Salt Cloud. In general they are used in the creation of profiles, but may also be used to create an instance directly (see below). Listing images requires a provider to be configured, and specified:

```
salt myminion cloud.list_images my-cloud-provider
```

list_sizes

This command is designed to show sizes that are available to be used to create an instance using Salt Cloud. In general they are used in the creation of profiles, but may also be used to create an instance directly (see below). This command is not available for all cloud providers; see the provider-specific documentation for details. Listing sizes requires a provider to be configured, and specified:

```
salt myminion cloud.list_sizes my-cloud-provider
```

list_locations

This command is designed to show locations that are available to be used to create an instance using Salt Cloud. In general they are used in the creation of profiles, but may also be used to create an instance directly (see below). This command is not available for all cloud providers; see the provider-specific documentation for details. Listing locations requires a provider to be configured, and specified:

```
salt myminion cloud.list_locations my-cloud-provider
```

query

This command is used to query all configured cloud providers, and display all instances associated with those accounts. By default, it will run a standard query, returning the following fields:

`id` The name or ID of the instance, as used by the cloud provider.

`image` The disk image that was used to create this instance.

`private_ips` Any public IP addresses currently assigned to this instance.

`public_ips` Any private IP addresses currently assigned to this instance.

`size` The size of the instance; can refer to RAM, CPU(s), disk space, etc., depending on the cloud provider.

`state` The running state of the instance; for example, `running`, `stopped`, `pending`, etc. This state is dependent upon the provider.

This command may also be used to perform a full query or a select query, as described below. The following usages are available:

```
salt myminion cloud.query
salt myminion cloud.query list_nodes
salt myminion cloud.query list_nodes_full
```

full_query

This command behaves like the `query` command, but lists all information concerning each instance as provided by the cloud provider, in addition to the fields returned by the `query` command.

```
salt myminion cloud.full_query
```

select_query

This command behaves like the `query` command, but only returned select fields as defined in the `/etc/salt/cloud` configuration file. A sample configuration for this section of the file might look like:

```
query.selection:
  - id
  - key_name
```

This configuration would only return the `id` and `key_name` fields, for those cloud providers that support those two fields. This would be called using the following command:

```
salt myminion cloud.select_query
```

profile

This command is used to create an instance using a profile that is configured on the target minion. Please note that the profile must be configured before this command can be used with it.

```
salt myminion cloud.profile ec2-centos64-x64 my-new-instance
```

Please note that the execution module does *not* run in parallel mode. Using multiple minions to create instances can effectively perform parallel instance creation.

create

This command is similar to the `profile` command, in that it is used to create a new instance. However, it does not require a profile to be pre-configured. Instead, all of the options that are normally configured in a profile are passed directly to Salt Cloud to create the instance:

```
salt myminion cloud.create my-ec2-config my-new-instance \
    image=ami-1624987f size='Micro Instance' ssh_username=ec2-user \
    securitygroup=default delvol_on_destroy=True
```

Please note that the execution module does *not* run in parallel mode. Using multiple minions to create instances can effectively perform parallel instance creation.

destroy

This command is used to destroy an instance or instances. This command will search all configured providers and remove any instance(s) which matches the name(s) passed in here. The results of this command are *non-reversable* and should be used with caution.

```
salt myminion cloud.destroy myinstance
salt myminion cloud.destroy myinstance1,myinstance2
```

action

This command implements both the `action` and the `function` commands used in the standard `salt-cloud` command. If one of the standard `action` commands is used, an instance name must be provided. If one of the standard `function` commands is used, a provider configuration must be named.

```
salt myminion cloud.action start instance=myinstance
salt myminion cloud.action show_image provider=my-ec2-config \
    image=ami-1624987f
```

The actions available are largely dependant upon the module for the specific cloud provider. The following actions are available for all cloud providers:

`list_nodes` This is a direct call to the `query` function as described above, but is only performed against a single cloud provider. A provider configuration must be included.

`list_nodes_select` This is a direct call to the `full_query` function as described above, but is only performed against a single cloud provider. A provider configuration must be included.

`list_nodes_select` This is a direct call to the `select_query` function as described above, but is only performed against a single cloud provider. A provider configuration must be included.

`show_instance` This is a thin wrapper around `list_nodes`, which returns the full information about a single instance. An instance name must be provided.

23.20.3 State Module

A subset of the execution module is available through the `cloud` state module. Not all functions are currently included, because there is currently insufficient code for them to perform statefully. For example, a command to create an istance may be issued with a series of options, but those options cannot currently be statefully managed. Additional states to manage these options will be released at a later time.

cloud.present

This state will ensure that an instance is present inside a particular cloud provider. Any option that is normally specified in the `cloud.create` execution module and function may be declared here, but only the actual presence of the instance will be managed statefully.

```
my-instance-name:
  cloud.present:
    - provider: my-ec2-config
    - image: ami-1624987f
    - size: 'Micro Instance'
    - ssh_username: ec2-user
    - securitygroup: default
    - delvol_on_destroy: True
```

cloud.profile

This state will ensure that an instance is present inside a particular cloud provider. This function calls the `cloud.profile` execution module and function, but as with `cloud.present`, only the actual presence of the instance will be managed statefully.

```
my-instance-name:
  cloud.present:
    - profile ec2-centos64-x64
```

cloud.absent

This state will ensure that an instance (identified by name) does not exist in any of the cloud providers configured on the target minion. Please note that this state is *non-reversable* and may be considered especially destructive when issued as a cloud state.

```
my-instance-name:
  cloud.absent
```

23.20.4 Runner Module

The `cloud` runner module is executed on the master, and performs actions using the configuration and Salt modules on the master itself. This means that any public minion keys will also be properly accepted by the master.

Using the functions in the runner module is no different than using those in the execution module, outside of the behavior described in the above paragraph. The following functions are available inside the runner:

- list_images
- list_sizes
- list_locations
- query
- full_query
- select_query
- profile
- destroy
- action

Outside of the standard usage of `salt-run` itself, commands are executed as usual:

```
salt-run cloud.profile ec2-centos64-x86_64 my-instance-name
```

23.20.5 CloudClient

The execution, state and runner modules ultimately all use the CloudClient library that ships with Salt. To use the CloudClient library locally (either on the master or a minion), create a client object and issue a command against it:

```
import salt.cloud
import pprint
client = salt.cloud.CloudClient('/etc/salt/cloud')
nodes = client.query()
pprint.pprint(nodes)
```

23.21 Getting Started With SoftLayer

SoftLayer is a public cloud provider, and baremetal hardware hosting provider.

23.21.1 Dependencies

The SoftLayer driver for Salt Cloud requires the softlayer package, which is available at PyPi:

https://pypi.python.org/pypi/SoftLayer

This package can be installed using *pip* or *easy_install*:

```
# pip install softlayer
# easy_install softlayer
```

23.21.2 Configuration

Set up the cloud config at `/etc/salt/cloud.providers`:

```
# Note: These examples are for /etc/salt/cloud.providers

  my-softlayer:
    # Set up the location of the salt master
    minion:
      master: saltmaster.example.com

    # Set the SoftLayer access credentials (see below)
    user: MYUSER1138
    apikey: 'e3b68aa711e6deadc62d5b76355674beef7cc3116062ddbacafe5f7e465bfdc9'

    provider: softlayer


  my-softlayer-hw:
    # Set up the location of the salt master
    minion:
      master: saltmaster.example.com

    # Set the SoftLayer access credentials (see below)
    user: MYUSER1138
    apikey: 'e3b68aa711e6deadc62d5b76355674beef7cc3116062ddbacafe5f7e465bfdc9'

    provider: softlayer_hw
```

23.21.3 Access Credentials

The *user* setting is the same user as is used to log into the SoftLayer Administration area. The *apikey* setting is found inside the Admin area after logging in:

- Hover over the *Administrative* menu item.
- Click the *API Access* link.
- The *apikey* is located next to the *user* setting.

23.21.4 Profiles

Cloud Profiles

Set up an initial profile at `/etc/salt/cloud.profiles`:

```
base_softlayer_ubuntu:
  provider: my-softlayer
  image: UBUNTU_LATEST
  cpu_number: 1
  ram: 1024
  disk_size: 100
  local_disk: True
  hourly_billing: True
  domain: example.com
  location: sjc01
  # Optional
  max_net_speed: 1000
  private_vlan: 396
  private_network: True
  private_ssh: True
  # May be used _instead_of_ image
  global_identifier: 320d8be5-46c0-dead-cafe-13e3c51
```

Most of the above items are required; optional items are specified below.

image

Images to build an instance can be found using the *–list-images* option:

```
# salt-cloud --list-images my-softlayer
```

The setting used will be labeled as *template*.

cpu_number

This is the number of CPU cores that will be used for this instance. This number may be dependent upon the image that is used. For instance:

```
Red Hat Enterprise Linux 6 - Minimal Install (64 bit) (1 - 4 Core):
    ----------
    name:
        Red Hat Enterprise Linux 6 - Minimal Install (64 bit) (1 - 4 Core)
    template:
        REDHAT_6_64
Red Hat Enterprise Linux 6 - Minimal Install (64 bit) (5 - 100 Core):
    ----------
    name:
        Red Hat Enterprise Linux 6 - Minimal Install (64 bit) (5 - 100 Core)
    template:
        REDHAT_6_64
```

Note that the template (meaning, the *image* option) for both of these is the same, but the names suggests how many CPU cores are supported.

ram

This is the amount of memory, in megabytes, that will be allocated to this instance.

disk_size

The amount of disk space that will be allocated to this image, in megabytes.

local_disk

When true the disks for the computing instance will be provisioned on the host which it runs, otherwise SAN disks will be provisioned.

hourly_billing

When true the computing instance will be billed on hourly usage, otherwise it will be billed on a monthly basis.

domain

The domain name that will be used in the FQDN (Fully Qualified Domain Name) for this instance. The *domain* setting will be used in conjunction with the instance name to form the FQDN.

location

Images to build an instance can be found using the *–list-locations* option:

```
# salt-cloud --list-location my-softlayer
```

max_net_speed

Specifies the connection speed for the instance's network components. This setting is optional. By default, this is set to 10.

public_vlan

If it is necessary for an instance to be created within a specific frontend VLAN, the ID for that VLAN can be specified in either the provider or profile configuration.

This ID can be queried using the *list_vlans* function, as described below. This setting is optional.

private_vlan

If it is necessary for an instance to be created within a specific backend VLAN, the ID for that VLAN can be specified in either the provider or profile configuration.

This ID can be queried using the *list_vlans* function, as described below. This setting is optional.

private_network

If a server is to only be used internally, meaning it does not have a public VLAN associated with it, this value would be set to True. This setting is optional. The default is False.

private_ssh

Whether to run the deploy script on the server using the public IP address or the private IP address. If set to True, Salt Cloud will attempt to SSH into the new server using the private IP address. The default is False. This settiong is optional.

global_identifier

When creating an instance using a custom template, this option is set to the corresponding value obtained using the *list_custom_images* function. This option will not be used if an *image* is set, and if an *image* is not set, it is required.

The profile can be realized now with a salt command:

```
# salt-cloud -p base_softlayer_ubuntu myserver
```

Using the above configuration, this will create *myserver.example.com*.

Once the instance has been created with salt-minion installed, connectivity to it can be verified with Salt:

```
# salt 'myserver.example.com' test.ping
```

Cloud Profiles

Set up an initial profile at `/etc/salt/cloud.profiles`:

```
base_softlayer_hw_centos:
  provider: my-softlayer-hw
  # CentOS 6.0 - Minimal Install (64 bit)
  image: 13963
  # 2 x 2.0 GHz Core Bare Metal Instance - 2 GB Ram
  size: 1921
  # 250GB SATA II
  hdd: 19
  # San Jose 01
  location: 168642
  domain: example.com
  # Optional
  vlan: 396
  port_speed: 273
  banwidth: 248
```

Most of the above items are required; optional items are specified below.

image

Images to build an instance can be found using the *–list-images* option:

```
# salt-cloud --list-images my-softlayer-hw
```

A list of *id‘s and names will be provided. The ‘name* will describe the operating system and architecture. The *id* will be the setting to be used in the profile.

size

Sizes to build an instance can be found using the *–list-sizes* option:

```
# salt-cloud --list-sizes my-softlayer-hw
```

A list of *id‘s and names will be provided. The ‘name* will describe the speed and quantity of CPU cores, and the amount of memory that the hardware will contain. The *id* will be the setting to be used in the profile.

hdd

There are currently two sizes of hard disk drive (HDD) that are available for hardware instances on SoftLayer:

```
19: 250GB SATA II
1267: 500GB SATA II
```

The *hdd* setting in the profile will be either 19 or 1267. Other sizes may be added in the future.

location

Locations to build an instance can be found using the *–list-images* option:

```
# salt-cloud --list-locations my-softlayer-hw
```

A list of IDs and names will be provided. The *location* will describe the location in human terms. The *id* will be the setting to be used in the profile.

domain

The domain name that will be used in the FQDN (Fully Qualified Domain Name) for this instance. The *domain* setting will be used in conjunction with the instance name to form the FQDN.

vlan

If it is necessary for an instance to be created within a specific VLAN, the ID for that VLAN can be specified in either the provider or profile configuration.

This ID can be queried using the *list_vlans* function, as described below.

port_speed

Specifies the speed for the instance's network port. This setting refers to an ID within the SoftLayer API, which sets the port speed. This setting is optional. The default is 273, or, 100 Mbps Public & Private Networks. The following settings are available:

- 273: 100 Mbps Public & Private Networks
- 274: 1 Gbps Public & Private Networks
- 21509: 10 Mbps Dual Public & Private Networks (up to 20 Mbps)
- 21513: 100 Mbps Dual Public & Private Networks (up to 200 Mbps)
- 2314: 1 Gbps Dual Public & Private Networks (up to 2 Gbps)

- 272: 10 Mbps Public & Private Networks

bandwidth

Specifies the network bandwidth available for the instance. This setting refers to an ID within the SoftLayer API, which sets the bandwidth. This setting is optional. The default is 248, or, 5000 GB Bandwidth. The following settings are available:

- 248: 5000 GB Bandwidth
- 129: 6000 GB Bandwidth
- 130: 8000 GB Bandwidth
- 131: 10000 GB Bandwidth
- 36: Unlimited Bandwidth (10 Mbps Uplink)
- 125: Unlimited Bandwidth (100 Mbps Uplink)

23.21.5 Actions

The following actions are currently supported by the SoftLayer Salt Cloud driver.

show_instance

This action is a thin wrapper around *–full-query*, which displays details on a single instance only. In an environment with several machines, this will save a user from having to sort through all instance data, just to examine a single instance.

```
$ salt-cloud -a show_instance myinstance
```

23.21.6 Functions

The following functions are currently supported by the SoftLayer Salt Cloud driver.

list_vlans

This function lists all VLANs associated with the account, and all known data from the SoftLayer API concerning those VLANs.

```
$ salt-cloud -f list_vlans my-softlayer
$ salt-cloud -f list_vlans my-softlayer-hw
```

The *id* returned in this list is necessary for the *vlan* option when creating an instance.

list_custom_images

This function lists any custom templates associated with the account, that can be used to create a new instance.

```
$ salt-cloud -f list_custom_images my-softlayer
```

The *globalIdentifier* returned in this list is necessary for the *global_identifier* option when creating an image using a custom template.

23.21.7 Optional Products for SoftLayer HW

The softlayer_hw provider supports the ability to add optional products, which are supported by SoftLayer's API. These products each have an ID associated with them, that can be passed into Salt Cloud with the *optional_products* option:

```
softlayer_hw_test:
  provider: my-softlayer-hw
  # CentOS 6.0 - Minimal Install (64 bit)
  image: 13963
  # 2 x 2.0 GHz Core Bare Metal Instance - 2 GB Ram
  size: 1921
  # 250GB SATA II
  hdd: 19
  # San Jose 01
  location: 168642
  domain: example.com
  optional_products:
    # MySQL for Linux
    - id: 28
    # Business Continuance Insurance
    - id: 104
```

These values can be manually obtained by looking at the source of an order page on the SoftLayer web interface. For convenience, many of these values are listed here:

Public Secondary IP Addresses

- 22: 4 Public IP Addresses
- 23: 8 Public IP Addresses

Primary IPv6 Addresses

- 17129: 1 IPv6 Address

Public Static IPv6 Addresses

- 1481: /64 Block Static Public IPv6 Addresses

OS-Specific Addon

- 17139: XenServer Advanced for XenServer 6.x
- 17141: XenServer Enterprise for XenServer 6.x
- 2334: XenServer Advanced for XenServer 5.6
- 2335: XenServer Enterprise for XenServer 5.6
- 13915: Microsoft WebMatrix
- 21276: VMware vCenter 5.1 Standard

Control Panel Software

- 121: cPanel/WHM with Fantastico and RVskin
- 20778: Parallels Plesk Panel 11 (Linux) 100 Domain w/ Power Pack
- 20786: Parallels Plesk Panel 11 (Windows) 100 Domain w/ Power Pack
- 20787: Parallels Plesk Panel 11 (Linux) Unlimited Domain w/ Power Pack
- 20792: Parallels Plesk Panel 11 (Windows) Unlimited Domain w/ Power Pack
- 2340: Parallels Plesk Panel 10 (Linux) 100 Domain w/ Power Pack
- 2339: Parallels Plesk Panel 10 (Linux) Unlimited Domain w/ Power Pack
- 13704: Parallels Plesk Panel 10 (Windows) Unlimited Domain w/ Power Pack

Database Software

- 29: MySQL 5.0 for Windows
- 28: MySQL for Linux
- 21501: Riak 1.x
- 20893: MongoDB
- 30: Microsoft SQL Server 2005 Express
- 92: Microsoft SQL Server 2005 Workgroup
- 90: Microsoft SQL Server 2005 Standard
- 94: Microsoft SQL Server 2005 Enterprise
- 1330: Microsoft SQL Server 2008 Express
- 1340: Microsoft SQL Server 2008 Web
- 1337: Microsoft SQL Server 2008 Workgroup
- 1334: Microsoft SQL Server 2008 Standard
- 1331: Microsoft SQL Server 2008 Enterprise
- 2179: Microsoft SQL Server 2008 Express R2
- 2173: Microsoft SQL Server 2008 Web R2
- 2183: Microsoft SQL Server 2008 Workgroup R2
- 2180: Microsoft SQL Server 2008 Standard R2
- 2176: Microsoft SQL Server 2008 Enterprise R2

Anti-Virus & Spyware Protection

- 594: McAfee VirusScan Anti-Virus - Windows
- 414: McAfee Total Protection - Windows

Insurance

- 104: Business Continuance Insurance

Monitoring

- 55: Host Ping
- 56: Host Ping and TCP Service Monitoring

Notification

- 57: Email and Ticket

Advanced Monitoring

- 2302: Monitoring Package - Basic
- 2303: Monitoring Package - Advanced
- 2304: Monitoring Package - Premium Application

Response

- 58: Automated Notification
- 59: Automated Reboot from Monitoring
- 60: 24x7x365 NOC Monitoring, Notification, and Response

Intrusion Detection & Protection

- 413: McAfee Host Intrusion Protection w/Reporting

Hardware & Software Firewalls

- 411: APF Software Firewall for Linux
- 894: Microsoft Windows Firewall
- 410: 10Mbps Hardware Firewall
- 409: 100Mbps Hardware Firewall
- 408: 1000Mbps Hardware Firewall

23.22 Troubleshooting Salt Cloud

This page describes various steps for troubleshooting problems that may arise while using Salt Cloud.

23.22.1 Generic Troubleshooting Steps

This section describes a set of instructions that are useful to a large number of situations, and are likely to solve most issues that arise.

Version Compatibility

One of the most common issues that Salt Cloud users run into is import errors. These are often caused by version compatibility issues with Salt.

Salt 0.16.x works with Salt Cloud 0.8.9 or greater.

Salt 0.17.x requires Salt Cloud 0.8.11.

Releases after 0.17.x (0.18 or greater) should not encounter issues as Salt Cloud has been merged into Salt itself.

Debug Mode

Frequently, running Salt Cloud in debug mode will reveal information about a deployment which would otherwise not be obvious:

```
salt-cloud -p myprofile myinstance -l debug
```

Keep in mind that a number of messages will appear that look at first like errors, but are in fact intended to give developers factual information to assist in debugging. A number of messages that appear will be for cloud providers that you do not have configured; in these cases, the message usually is intended to confirm that they are not configured.

Salt Bootstrap

By default, Salt Cloud uses the Salt Bootstrap script to provision instances:

This script is packaged with Salt Cloud, but may be updated without updating the Salt package:

```
salt-cloud -u
```

The Bootstrap Log

If the default deploy script was used, there should be a file in the `/tmp/` directory called `bootstrap-salt.log`. This file contains the full output from the deployment, including any errors that may have occurred.

Keeping Temp Files

Salt Cloud uploads minion-specific files to instances once they are available via SSH, and then executes a deploy script to put them into the correct place and install Salt. The `--keep-tmp` option will instruct Salt Cloud not to remove those files when finished with them, so that the user may inspect them for problems:

```
salt-cloud -p myprofile myinstance --keep-tmp
```

By default, Salt Cloud will create a directory on the target instance called `/tmp/.saltcloud/`. This directory should be owned by the user that is to execute the deploy script, and should have permissions of `0700`.

Most cloud providers are configured to use `root` as the default initial user for deployment, and as such, this directory and all files in it should be owned by the `root` user.

The `/tmp/.saltcloud/` directory should the following files:

- A `deploy.sh` script. This script should have permissions of `0755`.
- A `.pem` and `.pub` key named after the minion. The `.pem` file should have permissions of `0600`. Ensure that the `.pem` and `.pub` files have been properly copied to the `/etc/salt/pki/minion/` directory.
- A file called `minion`. This file should have been copied to the `/etc/salt/` directory.
- Optionally, a file called `grains`. This file, if present, should have been copied to the `/etc/salt/` directory.

Unprivileged Primary Users

Some providers, most notably EC2, are configured with a different primary user. Some common examples are `ec2-user`, `ubuntu`, `fedora` and `bitnami`. In these cases, the `/tmp/.saltcloud/` directory and all files in it should be owned by this user.

Some providers, such as EC2, are configured to not require these users to provide a password when using the `sudo` command. Because it is more secure to require `sudo` users to provide a password, other providers are configured that way.

If this instance is required to provide a password, it needs to be configured in Salt Cloud. A password for sudo to use may be added to either the provider configuration or the profile configuration:

```
sudo_password: mypassword
```

`/tmp/` is Mounted as `noexec`

It is more secure to mount the `/tmp/` directory with a `noexec` option. This is uncommon on most cloud providers, but very common in private environments. To see if the `/tmp/` directory is mounted this way, run the following command:

```
mount | grep tmp
```

The if the output of this command includes a line that looks like this, then the `/tmp/` directory is mounted as `noexec`:

```
tmpfs on /tmp type tmpfs (rw,noexec)
```

If this is the case, then the `deploy_command` will need to be changed in order to run the deploy script through the `sh` command, rather than trying to execute it directly. This may be specified in either the provider or the profile config:

```
deploy_command: sh /tmp/.saltcloud/deploy.sh
```

Please note that by default, Salt Cloud will place its files in a directory called `/tmp/.saltcloud/`. This may be also be changed in the provider or profile configuration:

```
tmp_dir: /tmp/.saltcloud/
```

If this directory is changed, then the `deploy_command` need to be changed in order to reflect the `tmp_dir` configuration.

Executing the Deploy Script Manually

If all of the files needed for deployment were successfully uploaded to the correct locations, and contain the correct permissions and ownerships, the deploy script may be executed manually in order to check for other issues:

```
cd /tmp/.saltcloud/
./deploy.sh
```

23.23 Spinning up Windows Minions

It is possible to use Salt Cloud to spin up Windows instances, and then install Salt on them. This functionality is available on all cloud providers that are supported by Salt Cloud. However, it may not necessarily be available on all Windows images.

23.23.1 Requirements

Salt Cloud makes use of *smbclient* and *winexe* to set up the Windows Salt Minion installer. *smbclient* may be part of either the *samba* package, or its own *smbclient* package, depending on the distribution. *winexe* is less commonly available in distribution-specific repositories. However, it is currently being built for various distributions in 3rd party channels:

- RPMs at pbone.net
- OpenSuse Build Service

Additionally, a copy of the Salt Minion Windows installer must be present on the system on which Salt Cloud is running. This installer may be downloaded from saltstack.com:

- SaltStack Download Area

23.23.2 Firewall Settings

Because Salt Cloud makes use of *smbclient* and *winexe*, port 445 must be open on the target image. This port is not generally open by default on a standard Windows distribution, and care must be taken to use an image in which this port is open, or the Windows firewall is disabled.

23.23.3 Configuration

Configuration is set as usual, with some extra configuration settings. The location of the Windows installer on the machine that Salt Cloud is running on must be specified. This may be done in any of the regular configuration files (main, providers, profiles, maps). For example:

Setting the installer in `/etc/salt/cloud.providers`:

```
my-softlayer:
  provider: softlayer
  user: MYUSER1138
  apikey: 'e3b68aa711e6deadc62d5b76355674beef7cc3116062ddbacafe5f7e465bfdc9'
  minion:
    master: saltmaster.example.com
  win_installer: /root/Salt-Minion-0.17.0-AMD64-Setup.exe
  win_username: Administrator
  win_password: letmein
```

The default Windows user is *Administrator*, and the default Windows password is blank.

23.24 Install Salt Cloud

Salt Cloud is now part of Salt proper. It was merged in as of *Salt version 2014.1.0*.

Salt Cloud depends on `apache-libcloud`. Libcloud can be installed via pip with `pip install apache-libcloud`.

23.24.1 Installing Salt Cloud for development

Installing Salt for development enables Salt Cloud development as well, just make sure `apache-libcloud` is installed as per above paragraph.

See these instructions: `Installing Salt for development`.

Salt cloud is a public cloud provisioning tool. Salt cloud is made to integrate Salt into cloud providers in a clean way so that minions on public cloud systems can be quickly and easily modeled and provisioned.

Salt cloud allows for cloud based minions to be managed via virtual machine maps and profiles. This means that individual cloud VMs can be created, or large groups of cloud VMs can be created at once or managed.

Virtual machines created with Salt cloud install salt on the target virtual machine and assign it to the specified master. This means that virtual machines can be provisioned and then potentially never logged into.

While Salt Cloud has been made to work with Salt, it is also a generic cloud management platform and can be used to manage non Salt centric clouds.

CHAPTER 24

Getting Started

- *Installing salt cloud*

Some quick guides covering getting started with each of the various cloud providers.

- *Getting Started With Azure*
- *Getting Started With Digital Ocean*
- *Getting Started With EC2*
- *Getting Started With GoGrid*
- *Getting Started With Google Compute Engine*
- *Getting Started With Joyent*
- *Getting Started With Linode*
- *Getting Started With OpenStack*
- *Getting Started With Parallels*
- *Getting Started With Rackspace*
- *Getting Started With SoftLayer*

CHAPTER 25

Core Configuration

The core configuration of Salt cloud is handled in the cloud configuration file. This file is comprised of global configurations for interfacing with cloud providers.

- *Core Configuration*

CHAPTER 26

Windows Configuration

Salt Cloud may be used to spin up a Windows minion, and then install the Salt Minion client on that instance. At this time, Salt Cloud itself still needs to be run from a Linux or Unix machine.

- *Windows Configuration*

CHAPTER 27

Using Salt Cloud

Salt cloud works via profiles and maps. Simple profiles for cloud VMs are defined and can be used directly, or a map can be defined specifying a large group of virtual machines to create.

- *Profiles*
- *Maps*

Once a VM has been deployed, a number of actions may be available to perform on it, depending on the specific cloud provider.

- *Actions*

Depending on your cloud provider, a number of functions may also be available which do not require a VM to be specified.

- *Functions*

CHAPTER 28

Miscellaneous Options

- *Miscellaneous*

CHAPTER 29

Troubleshooting Steps

- *Troubleshooting*

CHAPTER 30

Extending Salt Cloud

Salt Cloud extensions work in a way similar to Salt modules. Therefore extending Salt cloud to manage more public cloud providers and operating systems is easy.

- *Adding Cloud Providers*
- *Adding OS Support*

CHAPTER 31

Using Salt Cloud from Salt

Several Salt Cloud modules exist within Salt itself in order to manage cloud instances using Salt's own powerful feature set.

- *Using Salt Cloud from Salt*

CHAPTER 32

Feature Comparison

A table is available which compares various features available across all supported cloud providers.

- *Features*

CHAPTER 33

Legacy Releases

Changed in version 2014.1.0: (Hydrogen) Release notes will be part of Salt's main release notes starting with Salt's 2014.1.0 (Hydrogen) release.

- `Legacy Release Notes`

CHAPTER 34

Reference

- *Command-line interface*
- *Full table of contents*

CHAPTER 35

Virtual Machine Disk Profiles

Salt Virt allows for the disks created for deployed virtual machines to be finely configured. The configuration is a simple data structure which is read from the `config.option` function, meaning that the configuration can be stored in the minion config file, the master config file, or the minion's pillar.

This configuration option is called `virt.disk`. The default `virt.disk` data structure looks like this:

```
virt.disk:
  default:
    - system:
      size: 8192
      format: qcow2
      model: virtio
```

Note: The format and model does not need to be defined, Salt will default to the optimal format used by the underlying hypervisor, in the case of kvm this it is **qcow2** and **virtio**.

This configuration sets up a disk profile called default. The default profile creates a single system disk on the virtual machine.

35.1 Define More Profiles

Many environments will require more complex disk profiles and may require more than one profile, this can be easily accomplished:

```
virt.disk:
  default:
    - system:
        size: 8192
  database:
    - system:
        size: 8192
    - data:
        size: 30720
  web:
    - system:
        size: 1024
    - logs:
        size: 5120
```

This configuration allows for one of three profiles to be selected, allowing virtual machines to be created with different storage needs of the deployed vm.

Salt Virt - The SaltStack Cloud Controller

The Salt Virt cloud controller capability was initial added to Salt in version 0.14.0 as an alpha technology.

The initial Salt Virt system supports core cloud operations:

- Virtual machine deployment
- Inspection of deployed VMs
- Virtual machine migration
- Network profiling
- Automatic VM integration with all aspects of Salt
- Image Pre-seeding

Many features are currently under development to enhance the capabilities of the Salt Virt systems.

Note: It is noteworthy that Salt was originally developed with the intent of using the Salt communication system as the backbone to a cloud controller. This means that the Salt Virt system is not an afterthought, simply a system that took the back seat to other development. The original attempt to develop the cloud control aspects of Salt was a project called butter. This project never took off, but was functional and proves the early viability of Salt to be a cloud controller.

36.1 Salt Virt Tutorial

A tutorial about how to get Salt Virt up and running has been added to the tutorial section:

Cloud Controller Tutorial

36.2 The Salt Virt Runner

The point of interaction with the cloud controller is the **virt** runner. The **virt** runner comes with routines to execute specific virtual machine routines.

Reference documentation for the virt runner is available with the runner module documentation:

`Virt Runner Reference`

36.3 Based on Live State Data

The Salt Virt system is based on using Salt to query live data about hypervisors and then using the data gathered to make decisions about cloud operations. This means that no external resources are required to run Salt Virt, and that the information gathered about the cloud is live and accurate.

CHAPTER 37

Virtual Machine Network Profiles

Salt Virt allows for the network devices created for deployed virtual machines to be finely configured. The configuration is a simple data structure which is read from the `config.option` function, meaning that the configuration can be stored in the minion config file, the master config file, or the minion's pillar.

This configuration option is called `virt.nic`. By default the `virt.nic` option is empty but defaults to a data structure which looks like this:

```
virt.nic:
  default:
    eth0:
      bridge: br0
      model: virtio
```

Note: The model does not need to be defined, Salt will default to the optimal model used by the underlying hypervisor, in the case of kvm this model is **virtio**

This configuration sets up a network profile called default. The default profile creates a single Ethernet device on the virtual machine that is bridged to the hypervisor's **br0** interface. This default setup does not require setting up the `virt.nic` configuration, and is the reason why a default install only requires setting up the **br0** bridge device on the hypervisor.

37.1 Define More Profiles

Many environments will require more complex network profiles and may require more than one profile, this can be easily accomplished:

```
virt.nic:
  dual:
    eth0:
      bridge: service_br
    eth1:
      bridge: storage_br
  single:
    eth0:
      bridge: service_br
  triple:
    eth0:
      bridge: service_br
    eth1:
      bridge: storage_br
    eth2:
      bridge: dmz_br
  all:
    eth0:
      bridge: service_br
    eth1:
      bridge: storage_br
    eth2:
      bridge: dmz_br
    eth3:
      bridge: database_br
  dmz:
    eth0:
      bridge: service_br
    eth1:
      bridge: dmz_br
  database:
    eth0:
      bridge: service_br
    eth1:
      bridge: database_br
```

This configuration allows for one of six profiles to be selected, allowing virtual machines to be created which attach to different network depending on the needs of the deployed vm.

Salt SSH

Note: SALT-SSH IS ALPHA SOFTWARE AND MAY NOT BE READY FOR PRODUCTION USE

Note: On many systems, `salt-ssh` will be in its own package, usually named `salt-ssh`.

In version 0.17.0 of Salt a new transport system was introduced, the ability to use SSH for Salt communication. This addition allows for Salt routines to be executed on remote systems entirely through ssh, bypassing the need for a Salt Minion to be running on the remote systems and the need for a Salt Master.

Note: The Salt SSH system does not supercede the standard Salt communication systems, it simply offers an SSH based alternative that does not require ZeroMQ and a remote agent. Be aware that since all communication with Salt SSH is executed via SSH it is substantially slower than standard Salt with ZeroMQ.

Salt SSH is very easy to use, simply set up a basic *roster* file of the systems to connect to and run `salt-ssh` commands in a similar way as standard `salt` commands.

38.1 Salt SSH Roster

The roster system in Salt allows for remote minions to be easily defined.

Note: See the *Roster documentation* for more details.

Simply create the roster file, the default location is */etc/salt/roster*:

```
web1: 192.168.42.1
```

This is a very basic roster file where a Salt ID is being assigned to an IP address. A more elaborate roster can be created:

```
web1:
  host: 192.168.42.1 # The IP addr or DNS hostname
  user: fred         # Remote executions will be executed as user fred
  passwd: foobarbaz  # The password to use for login, if omitted, keys are used
  sudo: True         # Whether to sudo to root, not enabled by default
web2:
  host: 192.168.42.2
```

38.2 Calling Salt SSH

The `salt-ssh` command can be easily executed in the same was as a salt command:

```
salt-ssh '*' test.ping
```

Commands with `salt-ssh` follow the same syntax as the `salt` command.

The standard salt functions are available! The output is the same as `salt` and many of the same flags are available. Please see http://docs.saltstack.com/ref/cli/salt-ssh.html for all of the available options.

38.2.1 Raw Shell Calls

By default `salt-ssh` runs Salt execution modules on the remote system, but `salt-ssh` can also execute raw shell commands:

```
salt-ssh '*' -r 'ifconfig'
```

38.3 States Via Salt SSH

The Salt State system can also be used with `salt-ssh`. The state system abstracts the same interface to the user in `salt-ssh` as it does when using standard `salt`. The intent is that Salt Formulas defined for standard `salt` will work seamlessly with `salt-ssh` and vice-versa.

The standard Salt States walkthroughs function by simply replacing `salt` commands with `salt-ssh`.

38.4 Targeting with Salt SSH

Due to the fact that the targeting approach differs in salt-ssh, only glob and regex targets are supported as of this writing, the remaining target systems still need to be implemented.

38.5 Configuring Salt SSH

Salt SSH takes its configuration from a master configuration file. Normally, this file is in `/etc/salt/master`. If one wishes to use a customized configuration file, the `-c` option to Salt SSH facilitates passing in a directory to look inside for a configuration file named `master`.

38.6 Running Salt SSH as non-root user

By default, Salt read all the configuration from /etc/salt/. If you are running Salt SSH with a regular user you have to modify some paths or you will get "Permission denied" messages. You have to modify two parameters: `pki_dir` and `cachedir`. Those should point to a full path writable for the user.

It's recommed not to modify /etc/salt for this purpose. Create a private copy of /etc/salt for the user and run the command with `-c /new/config/path`.

CHAPTER 39

Salt Rosters

Salt rosters are plugable systems added in Salt 0.17.0 to facilitate the `salt-ssh` system. The roster system was created because `salt-ssh` needs a means to identify which systems need to be targeted for execution.

Note: The Roster System is not needed or used in standard Salt because the master does not need to be initially aware of target systems, since the Salt Minion checks itself into the master.

Since the roster system is pluggable, it can be easily augmented to attach to any existing systems to gather information about what servers are presently available and should be attached to by `salt-ssh`. By default the roster file is located at /etc/salt/roster.

39.1 How Rosters Work

The roster system compiles a data structure internally refered to as *targets*. The *targets* is a list of target systems and attributes about how to connect to said systems. The only requirement for a roster module in Salt is to return the *targets* data structure.

39.1.1 Targets Data

The information which can be stored in a roster *target* is the following:

```
<Salt ID>:   # The id to reference the target system with
    host:     # The IP address or DNS name of the remote host
    user:     # The user to log in as
    passwd:   # The password to log in with

    # Optional parameters
    port:     # The target system's ssh port number
    sudo:     # Boolean to run command via sudo
    priv:     # File path to ssh private key, defaults to salt-ssh.rsa
    timeout:  # Number of seconds to wait for response
```

CHAPTER 40

Running The Tests

There are requirements, in addition to Salt's requirements, which needs to be installed in order to run the test suite. Install one of the lines below, depending on the relevant Python version:

```
pip install -r dev_requirements_python26.txt
pip install -r dev_requirements_python27.txt
```

Note: In Salt 0.17, testing libraries were migrated into their own repo. To install them:

```
pip install git+https://github.com/saltstack/salt-testing.git#egg=SaltTesting
```

Failure to install SaltTesting will result in import errors similar to the following:

```
ImportError: No module named salttesting
```

Once all require requirements are set, use `tests/runtests.py` to run the tests, see `--help` for more info.

And alternative way of invoking the tests is available in setup.py, run the tests with the following command:

```
./setup.py test
```

Examples:

- Run unit tests only: `sudo ./tests/runtests.py --unit-tests`
- Run a specific set of integration tests only: `sudo ./tests/runtests.py -n integration.modules.virt -vv`
- Run a specific set of unit tests only: `./tests/runtests.py -n unit.modules.virt_test -vv`

40.1 Running The Tests In A Docker Container

If the `runtests.py` binary supports the `--docked` option flag, you can choose to execute the tests suite under the provided docker container. You need to have your docker properly configured on your system and the containers need to have access to the internet.

Here's a simple usage example:

```
tests/runtests.py --docked=ubuntu-12.04 -v
```

You can also provide the full *docker* container repository:

```
tests/runtests.py --docked=salttest/ubuntu-12.04 -v
```

The SaltStack team is creating some containers which will have the necessary dependencies pre-installed allowing you, for example, to run the destructive tests without making a single destructive change to your system, or, to run the tests suite under a different distribution than the one you're currently using.

You can see the current list of test suite images on our docker repository.

If you wish to provide your own docker container, you can submit pull requests against our docker salt test containers repository.

Writing Tests

Salt uses a test platform to verify functionality of components in a simple way. Two testing systems exist to enable testing salt functions in somewhat real environments. The two subsystems available are integration tests and unit tests.

Salt uses the python standard library unittest2 system for testing.

41.1 Integration Tests

The integration tests start up a number of salt daemons to test functionality in a live environment. These daemons include 2 salt masters, 1 syndic and 2 minions. This allows for the syndic interface to be tested and master/minion communication to be verified. All of the integration tests are executed as live salt commands sent through the started daemons.

- *Writing integration tests*

Integration tests are particularly good at testing modules, states and shell commands.

41.2 Unit Tests

Direct unit tests are also available, these tests are good for internal functions.

- *Writing unit tests*

CHAPTER 42

Integration Tests

The Salt integration tests come with a number of classes and methods which allow for components to be easily tested. These classes are generally inherited from and provide specific methods for hooking into the running integration test environment created by the integration tests.

It is noteworthy that since integration tests validate against a running environment that they are generally the preferred means to write tests.

The integration system is all located under tests/integration in the Salt source tree.

42.1 Integration Classes

The integration classes are located in tests/integration/__init__.py and can be extended therein. There are three classes available to extend:

42.1.1 ModuleCase

Used to define executions run via the master to minions and to call single modules and states.

The available methods are as follows:

run_function: Run a single salt function and condition the return down to match the behavior of the raw function call. This will run the command and only return the results from a single minion to verify.

state_result: Return the result data from a single state return

run_state: Run the state.single command and return the state return structure

42.1.2 SyndicCase

Used to execute remote commands via a syndic, only used to verify the capabilities of the Syndic.

The available methods are as follows:

run_function: Run a single salt function and condition the return down to match the behavior of the raw function call. This will run the command and only return the results from a single minion to verify.

42.1.3 ShellCase

Shell out to the scripts which ship with Salt.

The available methods are as follows:

run_script: Execute a salt script with the given argument string

run_salt: Execute the salt command, pass in the argument string as it would be passed on the command line.

run_run: Execute the salt-run command, pass in the argument string as it would be passed on the command line.

run_run_plus: Execute Salt run and the salt run function and return the data from each in a dict

run_key: Execute the salt-key command, pass in the argument string as it would be passed on the command line.

run_cp: Execute salt-cp, pass in the argument string as it would be passed on the command line.

run_call: Execute salt-call, pass in the argument string as it would be passed on the command line.

42.2 Examples

42.2.1 Module Example via ModuleCase Class

Import the integration module, this module is already added to the python path by the test execution. Inherit from the `integration.ModuleCase` class. The tests that execute against salt modules should be placed in the *tests/integration/modules* directory so that they will be detected by the test system.

Now the workhorse method `run_function` can be used to test a module:

```
import os
import integration


class TestModuleTest(integration.ModuleCase):
    '''
    Validate the test module
    '''
    def test_ping(self):
        '''
        test.ping
        '''
        self.assertTrue(self.run_function('test.ping'))

    def test_echo(self):
        '''
        test.echo
        '''
        self.assertEqual(self.run_function('test.echo', ['text']), 'text')
```

ModuleCase can also be used to test states, when testing states place the test module in the *tests/integration/states* directory. The `state_result` and the `run_state` methods are the workhorse here:

```
import os
import shutil
import integration

HFILE = os.path.join(integration.TMP, 'hosts')
```

```
class HostTest(integration.ModuleCase):
    '''
    Validate the host state
    '''

    def setUp(self):
        shutil.copyfile(os.path.join(integration.FILES, 'hosts'), HFILE)
        super(HostTest, self).setUp()

    def tearDown(self):
        if os.path.exists(HFILE):
            os.remove(HFILE)
        super(HostTest, self).tearDown()

    def test_present(self):
        '''
        host.present
        '''
        name = 'spam.bacon'
        ip = '10.10.10.10'
        ret = self.run_state('host.present', name=name, ip=ip)
        result = self.state_result(ret)
        self.assertTrue(result)
        with open(HFILE) as fp_:
            output = fp_.read()
            self.assertIn('{0}\t\t{1}'.format(ip, name), output)
```

The above example also demonstrates using the integration files and the integration state tree. The variable *integration.FILES* will point to the directory used to store files that can be used or added to to help enable tests that require files. The location *integration.TMP* can also be used to store temporary files that the test system will clean up when the execution finishes.

The integration state tree can be found at *tests/integration/files/file/base*. This is where the referenced *host.present* sls file resides.

42.2.2 Shell Example via ShellCase

Validating the shell commands can be done via shell tests. Here are some examples:

```
import sys
import shutil
import tempfile

import integration

class KeyTest(integration.ShellCase):
    '''
    Test salt-key script
    '''

    _call_binary_ = 'salt-key'

    def test_list(self):
        '''
        test salt-key -L
        '''
        data = self.run_key('-L')
        expect = [
                'Unaccepted Keys:',
                'Accepted Keys:',
                'minion',
                'sub_minion',
                'Rejected:', '']
        self.assertEqual(data, expect)
```

This example verifies that the `salt-key` command executes and returns as expected by making use of the `run_key` method.

All shell tests should be placed in the *tests/integraion/shell* directory.

CHAPTER 43

Writing Unit Tests

43.1 Introduction

Like many software projects, Salt has two broad-based testing approaches – integration testing and unit testing. While integration testing focuses on the interaction between components in a sandboxed environment, unit testing focuses on the singular implementation of individual functions.

43.2 Preparing to Write a Unit Test

Unit tests live in: tests/unit/.

Most commonly, the following imports are necessary to create a unit test:

```
# Import Salt Testing libs
from salttesting import skipIf, TestCase
from salttesting.helpers import ensure_in_syspath
```

If you need mock support to your tests, please also import:

```
from salttesting.mock import NO_MOCK, NO_MOCK_REASON, MagicMock, patch, call
```

43.3 A Simple Example

Let's assume that we're testing a very basic function in an imaginary Salt execution module. Given a module called `fib.py` that has a function called 'calculate(num_of_results)', which given a 'num_of_results', produces a list of sequential Fibonacci numbers of that length.

A unit test to test this function might be commonly placed in a file called `tests/unit/modules/fib_test.py`. The convention is to place unit tests for Salt execution modules in `test/unit/modules/` and to name the tests module suffixed with `_test.py`.

Tests are grouped around test cases, which are logically grouped sets of tests against a piece of functionality in the tested software. Test cases are created as Python classes in the unit test module. To return to our example, here's how we might write the skeleton for testing `fib.py`:

```
# Import Salt Testing libs
from salttesting import TestCase

# Import Salt execution module to test
```

```
from salt.modules import fib

# Create test case class and inherit from Salt’s customized TestCase
class FibTestCase(TestCase):

    ’’’
    If we want to set up variables common to all unit tests, we can do so
    by defining a setUp method, which will be run automatically before
    tests begin.
    ’’’
    def setUp(self):
        # Declare a simple set of five Fibonacci numbers starting at zero that we know are correct.
        self.fib_five = [0, 1, 1, 2, 3]

    def test_fib(self):
        ’’’
        To create a unit test, we should prefix the name with ‘test_’ so that it’s recognized by the
        ’’’
        self.assertEqual(fib.calculate(5), self.fib_five)
```

At this point, the test can now be run, either individually or as a part of a full run of the test runner. To ease development, a single test can be executed:

```
tests/runtests.py -n unit.modules.fib_test
```

This will produce output indicating the success or failure of the tests in given test case. For more detailed results, one can also include a flag to increase verbosity:

```
tests/runtests.py -n unit.modules.fib_test -v
```

To review the results of a particular run, take a note of the log location given in the output for each test:

> **Logging tests on /var/folders/nl/d809xbq577l3qrbj3ymtpbq80000gn/T/salt-runtests.log**

43.4 Evaluating Truth

A longer discussion on the types of assertions one can make can be found by reading Python’s documentation on unit testing.

43.5 Tests Using Mock Objects

In many cases, the very purpose of a Salt module is to interact with some external system, whether it be to control a database, manipulate files on a filesystem or many other examples. In these varied cases, it’s necessary to design a unit test which can test the function whilst replacing functions which might actually call out to external systems. One might think of this as “blocking the exits” for code under tests and redirecting the calls to external systems with our own code which produces known results during the duration of the test.

To achieve this behavior, Salt makes heavy use of the MagicMock package.

To understand how one might integrate Mock into writing a unit test for Salt, let’s imagine a scenario in which we’re testing an execution module that’s designed to operate on a database. Furthermore, let’s imagine two separate methods, here presented in pseduo-code in an imaginary execution module called ‘db.py.

```
def create_user(username):
    qry = ’CREATE USER {0}’.format(username)
    execute_query(qry)

def execute_query(qry):
    # Connect to a database and actually do the query...
```

Here, let’s imagine that we want to create a unit test for the *create_user* function. In doing so, we want to avoid any calls out to an external system and so while we are running our unit tests, we want to replace the actual interaction with a database with a function that can capture the parameters sent to it and return pre-defined values. Therefore, our task is clear – to write a unit test which tests the functionality of *create_user* while also replacing ‘execute_query’ with a mocked function.

To begin, we set up the skeleton of our class much like we did before, but with additional imports for MagicMock:

```
# Import Salt Testing libs
from salttesting import TestCase

# Import Salt execution module to test
from salt.modules import db

# NEW! -- Import Mock libraries
from salttesting.mock import NO_MOCK, NO_MOCK_REASON, MagicMock, patch, call

# Create test case class and inherit from Salt’s customized TestCase

@skipIf(NO_MOCK, NO_MOCK_REASON) #  Skip this test case if we don’t have access to mock!
class DbTestCase(TestCase):
    def test_create_user(self):
        # First, we replace ’execute_query’ with our own mock function
        db.execute_query = MagicMock()

        # Now that the exits are blocked, we can run the function under test.

        db.create_user(’testuser’)

        # We could now query our mock object to see which calls were made to it.
        ## print db.execute_query.mock_calls

        ’’’
        We want to test to ensure that the correct query was formed.
        This is a contrived example, just designed to illustrate the concepts at hand.

        We’re going to first contruct a call() object that represents the way we expect
        our mocked execute_query() function to have been called.
        Then, we’ll examine the list of calls that were actually made to to execute_function().

        By comparing our expected call to execute_query() with create_user()’s call to
        execute_query(), we can determine the success or failure of our unit test.
        ’’’

        expected_call = call(’CREATE USER testuser’)

        # Do the comparison! Will assert False if execute_query() was not called with the given call

        db.execute_query.assert_has_calls(expected_call)
```

43.6 Modifying `__salt__` In Place

At times, it becomes necessary to make modifications to a module's view of functions in its own `__salt__` dictionary. Luckily, this process is quite easy.

Below is an example that uses MagicMock's `patch` functionality to insert a function into `__salt__` that's actually a MagicMock instance.

```
def show_patch(self):
    with patch.dict(my_module.__salt__, {'function.to_replace': MagicMock()}:
        # From this scope, carry on with testing, with a modified __salt__!
```

CHAPTER 44

Reactor System

Salt version 0.11.0 introduced the reactor system. The premise behind the reactor system is that with Salt's events and the ability to execute commands, a logic engine could be put in place to allow events to trigger actions, or more accurately, reactions.

This system binds sls files to event tags on the master. These sls files then define reactions. This means that the reactor system has two parts. First, the reactor option needs to be set in the master configuration file. The reactor option allows for event tags to be associated with sls reaction files. Second, these reaction files use highdata (like the state system) to define reactions to be executed.

44.1 Event System

A basic understanding of the event system is required to understand reactors. The event system is a local ZeroMQ PUB interface which fires salt events. This event bus is an open system used for sending information notifying Salt and other systems about operations.

The event system fires events with a very specific criteria. Every event has a **tag**. Event tags allow for fast top level filtering of events. In addition to the tag, each event has a data structure. This data structure is a dict, which contains information about the event.

44.2 Mapping Events to Reactor SLS Files

Reactor SLS files and event tags are associated in the master config file. By default this is /etc/salt/master, or /etc/salt/master.d/reactor.conf.

In the master config section 'reactor:' is a list of event tags to be matched and each event tag has a list of reactor SLS files to be run.

```
reactor:                           # Master config section "reactor"

  - 'salt/minion/*/start':                # Match tag "salt/minion/*/start"
    - /srv/reactor/start.sls       # Things to do when a minion starts
    - /srv/reactor/monitor.sls     # Other things to do

  - 'salt/cloud/*/destroyed':     # Globs can be used to matching tags
    - /srv/reactor/decommision.sls # Things to do when a server is removed
```

Reactor sls files are similar to state and pillar sls files. They are by default yaml + Jinja templates and are passed familar context variables.

They differ because of the addtion of the `tag` and `data` variables.

- The `tag` variable is just the tag in the fired event.
- The `data` variable is the event's data dict.

Here is a simple reactor sls:

```
{% if data['id'] == 'mysql1' %}
highstate_run:
  cmd.state.highstate:
    - tgt: mysql1
{% endif %}
```

This simple reactor file uses Jinja to further refine the reaction to be made. If the `id` in the event data is `mysql1` (in other words, if the name of the minion is `mysql1`) then the following reaction is defined. The same data structure and compiler used for the state system is used for the reactor system. The only difference is that the data is matched up to the salt command API and the runner system. In this example, a command is published to the `mysql1` minion with a function of `state.highstate`. Similarly, a runner can be called:

```
{% if data['data']['overstate'] == 'refresh' %}
overstate_run:
  runner.state.over
{% endif %}
```

This example will execute the state.overstate runner and initiate an overstate execution.

44.3 Fire an event

To fire an event from a minion call `event.fire_master`

```
salt-call event.fire_master '{"overstate": "refresh"}' 'foo'
```

After this is called, any reactor sls files matching event tag `foo` will execute with `{{ data['data']['overstate'] }}` equal to `'refresh'`.

See `salt.modules.event` for more information.

44.4 Knowing what event is being fired

Knowing exactly which event is being fired and what data is has for use in the sls files can be challenging. The easiest way to see exactly what's going on is to use the **eventlisten.py** script. This script is not part of packages but is part of the source.

If the master process is using the default socket, no additional options will be required. Otherwise, you will need to specify the socket location.

Example usage:

```
wget https://raw.github.com/saltstack/salt/develop/tests/eventlisten.py
python eventlisten.py

# OR
python eventlisten.py --sock-dir /path/to/var/run/salt
```

Example output:

```
Event fired at Fri Dec 20 10:43:00 2013
*************************
Tag: salt/auth
Data:
{'_stamp': '2013-12-20_10:47:54.584699',
 'act': 'accept',
 'id': 'fuzzer.domain.tld',
 'pub': '-----BEGIN PUBLIC KEY-----\nMIICIDANBgk+TRIMMED+EMZ8CAQE=\n-----END PUBLIC KEY-----\n',
 'result': True}

Event fired at Fri Dec 20 10:43:01 2013
*************************
Tag: salt/minion/fuzzer.domain.tld/start
Data:
{'_stamp': '2013-12-20_10:43:01.638387',
 'cmd': '_minion_event',
 'data': 'Minion fuzzer.domain.tld started at Fri Dec 20 10:43:01 2013',
 'id': 'fuzzer.domain.tld',
 'pretag': None,
 'tag': 'salt/minion/fuzzer.domain.tld/start'}
```

44.5 Understanding the Structure of Reactor Formulas

While the reactor system uses the same data structure as the state system, this data does not translate the same way to operations. In state files formula information is mapped to the state functions, but in the reactor system information is mapped to a number of available subsystems on the master. These systems are the **LocalClient** and the **Runners**. The **state declaration** field takes a reference to the function to call in each interface. So to trigger a salt-run call the **state declaration** field will start with **runner**, followed by the runner function to call. This means that a call to what would be on the command line **salt-run manage.up** will be **runner.manage.up**. An example of this in a reactor formula would look like this:

```
manage_up:
  runner.manage.up
```

If the runner takes arguments then they can be specified as well:

```
overstate_dev_env:
  runner.state.over:
    - env: dev
```

Executing remote commands maps to the **LocalClient** interface which is used by the **salt** command. This interface more specifically maps to the **cmd_async** method inside of the **LocalClient** class. This means that the arguments passed are being passed to the **cmd_async** method, not the remote method. A field starts with **cmd** to use the **LocalClient** subsystem. The result is, to execute a remote command, a reactor fomular would look like this:

```
clean_tmp:
  cmd.cmd.run:
    - tgt: '*'
    - arg:
      - rm -rf /tmp/*
```

The `arg` option takes a list of arguments as they would be presented on the command line, so the above declaration is the same as running this salt command:

```
salt '*' cmd.run 'rm -rf /tmp/*'
```

Use the `expr_form` argument to specify a matcher:

```
clean_tmp:
  cmd.cmd.run:
    - tgt: 'os:Ubuntu'
    - expr_form: grain
    - arg:
      - rm -rf /tmp/*
```

```
clean_tmp:
  cmd.cmd.run:
    - tgt: 'G@roles:hbase_master'
    - expr_form: compound
    - arg:
      - rm -rf /tmp/*
```

An interesting trick to pass data from the Reactor script to `state.highstate` or `state.sls` is to pass it as inline Pillar data since both functions take a keyword argument named `pillar`.

The following example uses Salt's Reactor to listen for the event that is fired when the key for a new minion is accepted on the master using `salt-key`.

`/etc/salt/master.d/reactor.conf`:

```
reactor:
  - 'salt/key':
    - /srv/salt/haproxy/react_new_minion.sls
```

The Reactor then fires a `state.sls` command targeted to the HAProxy servers and passes the ID of the new minion from the event to the state file via inline Pillar.

Note, the Pillar data will need to be passed as a string since that is how it is passed at the CLI. That string will be parsed as YAML on the minion (same as how it works at the CLI).

`/srv/salt/haproxy/react_new_minion.sls`:

```
{% if data['act'] == 'accept' and data['id'].startswith('web') %}
add_new_minion_to_pool:
  cmd.state.sls:
    - tgt: 'haproxy*'
    - arg:
      - haproxy.refresh_pool
      - 'pillar={new_minion: {{ data['id'] }}}'
{% endif %}
```

The above command is equivalent to the following command at the CLI:

```
salt 'haproxy*' state.sls haproxy.refresh_pool 'pillar={new_minion: minionid}'
```

Finally, that data is available in the state file using the normal Pillar lookup syntax. The following example is grabbing web server names and IP addresses from *Salt Mine*. If this state is invoked from the Reactor then the custom Pillar value from above will be available and the new minion will be added to the pool but with the `disabled` flag so that HAProxy won't yet direct traffic to it.

`/srv/salt/haproxy/refresh_pool.sls`:

```
{% set new_minion = salt['pillar.get']('new_minion') %}

listen web *:80
    balance source
```

```
{% for server,ip in salt[’mine.get’](’web*’, ’network.interfaces’, [’eth0’]).items() %}
{% if server == new_minion %}
server {{ server }} {{ ip }}:80 disabled
{% else %}
server {{ server }} {{ ip }}:80 check
{% endif %}
{% endfor %}
```

44.6 A complete example

In this example, we’re going to assume that we have a group of servers that will come online at random and need to have keys automatically accepted. We’ll also add that we don’t want all servers being automatically accepted. For this example, we’ll assume that all hosts that have an id that starts with ‘ink’ will be automatically accepted and have state.highstate executed. On top of thise, we’re going to add that a host coming up that was replaced (meaning a new key) will also be accepted.

Our master configuration will be rather simple. All minions that attempte to authenticate will match the **tag** of **salt/auth**. When it comes to the minion key being accepted, we get a more refined **tag** that includes the minion id, which we can use for matching.

`/etc/salt/master.d/reactor.conf`:

```
reactor:
  - ’salt/auth’:
    - /srv/reactor/auth-pending.sls
  - ’salt/minion/ink*/start’:
    - /srv/reactor/auth-complete.sls
```

In this sls file, we say that if the key was rejected we will delete the key on the master and then also tell the master to ssh in to the minion and tell it to restart the minion, since a minion process will die if the key is rejected.

We also say that if the key is pending and the id starts with ink we will accept the key. A minion that is waiting on a pending key will retry authentication authentication every ten second by default.

`/srv/reactor/auth-pending.sls`:

```
{# Ink server faild to authenticate -- remove accepted key #}
{% if not data[’result’] and data[’id’].startswith(’ink’) %}
minion_remove:
  wheel.key.delete:
    - match: {{ data[’id’] }}
minion_rejoin:
  cmd.cmd.run:
    - tgt: salt-master.domain.tld
    - arg:
      - ssh -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no "{{ data[’id’] }}" ’sleep 10
{% endif %}

{# Ink server is sending new key -- accept this key #}
{% if ’act’ in data and data[’act’] == ’pend’ and data[’id’].startswith(’ink’) %}
minion_add:
  wheel.key.accept:
    - match: {{ data[’id’] }}
{% endif %}
```

No if statements are needed here because we already limited this action to just Ink servers in the master configuration.

`/srv/reactor/auth-complete.sls`:

```
{# When an Ink server connects, run state.highstate. #}
highstate_run:
  cmd.state.highstate:
    - tgt: {{ data['id'] }}
```

CHAPTER 45

Salt Formulas

Formulas are pre-written Salt States. They are as open-ended as Salt States themselves and can be used for tasks such as installing a package, configuring and starting a service, setting up users or permissions, and many other common tasks.

Note: Formulas require Salt 0.17 or later.

More accurately, Formulas are not tested on earlier versions of Salt so your mileage may vary.

All Formulas require the grains execution module that shipped with Salt 0.16.4. Earlier Salt versions may copy https://github.com/saltstack/salt/blob/develop/salt/modules/grains.py into the `/srv/salt/_modules` directory and it will be automatically distributed to all minions.

Some Formula utilize features added in Salt 0.17 and will not work on earlier Salt versions.

All official Salt Formulas are found as separate Git repositories in the "saltstack-formulas" organization on GitHub:

https://github.com/saltstack-formulas

As an example, quickly install and configure the popular memcached server using sane defaults simply by including the `memcached-formula` repository into an existing Salt States tree.

45.1 Installation

Each Salt Formula is an individual Git repository designed as a drop-in addition to an existing Salt State tree. Formulas can be installed in the following ways.

45.1.1 Adding a Formula as a GitFS remote

One design goal of Salt's GitFS fileserver backend was to facilitate reusable States so this is a quick and natural way to use Formulas.

See also:

Setting up GitFS

1. Add one or more Formula repository URLs as remotes in the `gitfs_remotes` list in the Salt Master configuration file.
2. Restart the Salt master.

45.1.2 Adding a Formula directory manually

Since Formulas are simply directories they can be copied onto the local file system by using Git to clone the repository or by downloading and expanding a tarball or zip file of the directory.

- Clone the repository manually and add a new entry to `file_roots` pointing to the clone's directory.
- Clone the repository manually and then copy or link the Formula directory into `file_roots`.

45.2 Usage

Each Formula is intended to be immediately usable with sane defaults without any additional configuration. Many formulas are also configurable by including data in Pillar; see the `pillar.example` file in each Formula repository for available options.

45.2.1 Including a Formula in an existing State tree

Formula may be included in an existing `sls` file. This is often useful when a state you are writing needs to `require` or `extend` a state defined in the formula.

Here is an example of a state that uses the `epel-formula` in a `require` declaration which directs Salt to not install the `python26` package until after the EPEL repository has also been installed:

```
include:
  - epel

python26:
  pkg:
    - installed
    - require:
      - pkg: epel
```

45.2.2 Including a Formula from a Top File

Some Formula perform completely standalone installations that are not referenced from other state files. It is usually cleanest to include these Formula directly from a Top File.

For example the easiest way to set up an OpenStack deployment on a single machine is to include the `openstack-standalone-formula` directly from a `top.sls` file:

```
base:
  'myopenstackmaster':
    - openstack
```

Quickly deploying OpenStack across several dedicated machines could also be done directly from a Top File and may look something like this:

```
base:
  'controller':
    - openstack.horizon
    - openstack.keystone
  'hyper-*':
    - openstack.nova
    - openstack.glance
  'storage-*':
    - openstack.swift
```

45.2.3 Configuring Formula using Pillar

Salt Formulas are designed to work out of the box with no additional configuration. However, many Formula support additional configuration and customization through *Pillar*. Examples of available options can be found in a file named `pillar.example` in the root directory of each Formula repository.

45.2.4 Modifying default Formula behavior

Remember that Formula are regular Salt States and can be used with all Salt's normal mechanisms for determining execution order. Formula can be required from other States with `require` declarations, they can be modified using `extend`, they can made to watch other states with `watch_in`, they can be used as templates for other States with `use`. Don't be shy to read through the source for each Formula!

45.2.5 Reporting problems & making additions

Each Formula is a separate repository on GitHub. If you encounter a bug with a Formula please file an issue in the respective repository! Send fixes and additions as a pull request. Add tips and tricks to the repository wiki.

45.3 Writing Formulas

Each Formula is a separate repository in the saltstack-formulas organization on GitHub.

Note: Get involved creating new Formulas

The best way to create new Formula repositories for now is to create a repository in your own account on GitHub and notify a SaltStack employee when it is ready. We will add you as a collaborator on the saltstack-formulas organization and help you transfer the repository over. Ping a SaltStack employee on IRC (`#salt` on Freenode) or send an email to the Salt mailing list.

45.3.1 Repository structure

A basic Formula repository should have the following layout:

```
foo-formula
|-- foo/
|   |-- map.jinja
|   |-- init.sls
|   `-- bar.sls
|-- CHANGELOG.rst
|-- LICENSE
|-- pillar.example
|-- README.rst
`-- VERSION
```

See also:

`template-formula`

The `template-formula` repository has a pre-built layout that serves as the basic structure for a new formula repository. Just copy the files from there and edit them.

45.3.2 `README.rst`

The README should detail each available `.sls` file by explaining what it does, whether it has any dependencies on other formulas, whether it has a target platform, and any other installation or usage instructions or tips.

A sample skeleton for the `README.rst` file:

```
foo
===

Install and configure the FOO service.

.. note::

    See the full `Salt Formulas installation and usage instructions
    <http://docs.saltstack.com/topics/conventions/formulas.html>`_.

Available states
================

``foo``
    Install the ``foo`` package and enable the service.
``foo.bar``
    Install the ``bar`` package.
```

45.3.3 `CHANGELOG.rst`

The `CHANGELOG.rst` file should detail the individual versions, their release date and a set of bullet points for each version highlighting the overall changes in a given version of the formula.

A sample skeleton for the *CHANGELOG.rst* file:

`CHANGELOG.rst`:

```
foo formula
===========

0.0.2 (2013-01-01)

- Re-organized formula file layout
- Fixed filename used for upstart logger template
- Allow for pillar message to have default if none specified
```

45.3.4 `map.jinja`

It is useful to have a single source for platform-specific or other parameterized information that can be reused throughout a Formula. See "*Configuration and parameterization*" below for more information. Such a file should be named `map.jinja` and live alongside the state files.

The following is an example from the MySQL Formula that has been slightly modified to be more readable and less terse.

In essence, it is a simple dictionary that serves as a lookup table. The `grains.filter_by` function then does a lookup on that table using the `os_family` grain (by default) and sets the result to a variable that can be used throughout the formula.

See also:

`grains.filter_by`

`map.jinja`:

```
{% set mysql_lookup_table = {
    'Debian': {
        'server': 'mysql-server',
        'client': 'mysql-client',
        'service': 'mysql',
        'config': '/etc/mysql/my.cnf',
    },
    'RedHat': {
        'server': 'mysql-server',
        'client': 'mysql',
        'service': 'mysqld',
        'config': '/etc/my.cnf',
    },
    'Gentoo': {
        'server': 'dev-db/mysql',
        'mysql-client': 'dev-db/mysql',
        'service': 'mysql',
        'config': '/etc/mysql/my.cnf',
    },
} %}

{% set mysql = salt['grains.filter_by'](mysql_lookup_table,
    merge=salt['pillar.get']('mysql:lookup'))
```

The above example is used to help explain how the mapping works. In most map files you will see the following structure:

```
{% set mysql = salt['grains.filter_by']({
    'Debian': {
        'server': 'mysql-server',
        'client': 'mysql-client',
        'service': 'mysql',
        'config': '/etc/mysql/my.cnf',
        'python': 'python-mysqldb',
    },
    'RedHat': {
        'server': 'mysql-server',
        'client': 'mysql',
        'service': 'mysqld',
        'config': '/etc/my.cnf',
        'python': 'MySQL-python',
    },
    'Gentoo': {
        'server': 'dev-db/mysql',
        'mysql-client': 'dev-db/mysql',
        'service': 'mysql',
        'config': '/etc/mysql/my.cnf',
        'python': 'dev-python/mysql-python',
    },
}, merge=salt['pillar.get']('mysql:lookup')) %}
```

The `merge` keyword specifies the location of a dictionary in Pillar that can be used to override values returned from the lookup table. If the value exists in Pillar it will take precedence, otherwise `merge` will be ignored. This is useful when software or configuration files is installed to non-standard locations. For example, the following Pillar would replace the `config` value from the call above.

```
mysql:
  lookup:
    config: /usr/local/etc/mysql/my.cnf
```

Any of the values defined above can be fetched for the current platform in any state file using the following syntax:

```
{% from "mysql/map.jinja" import mysql with context %}

mysql-server:
  pkg:
    - installed
    - name: {{ mysql.server }}
  service:
    - running
    - name: {{ mysql.service }}
    - require:
      - pkg: mysql-server

mysql-config:
  file:
    - managed
    - name: {{ mysql.config }}
    - source: salt://mysql/conf/my.cnf
    - watch:
      - service: mysql-server
```

45.3.5 SLS files

Each state in a Formula should use sane defaults (as much as is possible) and use Pillar to allow for customization.

The root state, in particular, and most states in general, should strive to do no more than the basic expected thing and advanced configuration should be put in child states build on top of the basic states.

For example, the root Apache should only install the Apache httpd server and make sure the httpd service is running. It can then be used by more advanced states:

```
# apache/init.sls
httpd:
  pkg:
    - installed
  service:
    - running

# apache/mod_wsgi.sls
include:
  - apache

mod_wsgi:
  pkg:
    - installed
    - require:
      - pkg: apache
```

```
# apache/debian/vhost_setup.sls
{% if grains['os_family'] == 'Debian' %}
a2dissite 000-default:
  cmd.run:
    - onlyif: test -L /etc/apache2/sites-enabled/000-default
    - require:
      - pkg: apache
{% endif %}
```

Platform agnostic

Each Salt Formula must be able to be run without error on any platform. If the formula is not applicable to a platform it should do nothing. See the `epel-formula` for an example.

Any platform-specific states must be wrapped in conditional statements:

```
{% if grains['os_family'] == 'Debian' %}
...
{% endif %}
```

A handy method for using platform-specific values is to create a lookup table using the `filter_by()` function:

```
{% set apache = salt['grains.filter_by']({
    'Debian': {'conf': '/etc/apache2/conf.d'},
    'RedHat': {'conf': '/etc/httpd/conf.d'},
}) %}

myconf:
  file:
    - managed
    - name: {{ apache.conf }}/myconf.conf
```

45.3.6 Configuration and parameterization

Each Formula should strive for sane defaults that can then be customized using Pillar. Pillar lookups must use the safe `get()` and must provide a default value:

```
{% if salt['pillar.get']('horizon:use_ssl', False) %}
ssl_crt: {{ salt['pillar.get']('horizon:ssl_crt', '/etc/ssl/certs/horizon.crt') }}
ssl_key: {{ salt['pillar.get']('horizon:ssl_key', '/etc/ssl/certs/horizon.key') }}
{% endif %}
```

Any default values used in the Formula must also be documented in the `pillar.example` file in the root of the repository. Comments should be used liberally to explain the intent of each configuration value. In addition, users should be able copy-and-paste the contents of this file into their own Pillar to make any desired changes.

45.3.7 Scripting

Remember that both State files and Pillar files can easily call out to Salt *execution modules* and have access to all the system grains as well.

```
{% if '/storage' in salt['mount.active']() %}
/usr/local/etc/myfile.conf:
  file:
    - symlink
    - target: /storage/myfile.conf
{% endif %}
```

Jinja macros are generally discouraged in favor of adding functions to existing Salt modules or adding new modules. An example of this is the `filter_by()` function.

45.3.8 Versioning

Formula are versioned according to Semantic Versioning, http://semver.org/.

> Given a version number MAJOR.MINOR.PATCH, increment the:
>
> 1. MAJOR version when you make incompatible API changes,
> 2. MINOR version when you add functionality in a backwards-compatible manner, and
> 3. PATCH version when you make backwards-compatible bug fixes.
>
> Additional labels for pre-release and build metadata are available as extensions to the MAJOR.MINOR.PATCH format.

Formula versions are tracked using Git tags as well as the `VERSION` file in the formula repository. The `VERSION` file should contain the currently released version of the particular formula.

45.3.9 Testing Formulas

Salt Formulas are tested by running each `.sls` file via `state.sls` and checking the output for success or failure. This is done for each supported platform.

CHAPTER 46

Salt Conventions

46.1 Salt Formulas

Formulas are pre-written Salt States. They are as open-ended as Salt States themselves and can be used for tasks such as installing a package, configuring and starting a service, setting up users or permissions, and many other common tasks.

Note: Formulas require Salt 0.17 or later.

More accurately, Formulas are not tested on earlier versions of Salt so your mileage may vary.

All Formulas require the grains execution module that shipped with Salt 0.16.4. Earlier Salt versions may copy https://github.com/saltstack/salt/blob/develop/salt/modules/grains.py into the `/srv/salt/_modules` directory and it will be automatically distributed to all minions.

Some Formula utilize features added in Salt 0.17 and will not work on earlier Salt versions.

All official Salt Formulas are found as separate Git repositories in the "saltstack-formulas" organization on GitHub:

https://github.com/saltstack-formulas

As an example, quickly install and configure the popular memcached server using sane defaults simply by including the `memcached-formula` repository into an existing Salt States tree.

46.1.1 Installation

Each Salt Formula is an individual Git repository designed as a drop-in addition to an existing Salt State tree. Formulas can be installed in the following ways.

Adding a Formula as a GitFS remote

One design goal of Salt's GitFS fileserver backend was to facilitate reusable States so this is a quick and natural way to use Formulas.

See also:

Setting up GitFS

1. Add one or more Formula repository URLs as remotes in the `gitfs_remotes` list in the Salt Master configuration file.
2. Restart the Salt master.

Adding a Formula directory manually

Since Formulas are simply directories they can be copied onto the local file system by using Git to clone the repository or by downloading and expanding a tarball or zip file of the directory.

- Clone the repository manually and add a new entry to `file_roots` pointing to the clone's directory.
- Clone the repository manually and then copy or link the Formula directory into `file_roots`.

46.1.2 Usage

Each Formula is intended to be immediately usable with sane defaults without any additional configuration. Many formulas are also configurable by including data in Pillar; see the `pillar.example` file in each Formula repository for available options.

Including a Formula in an existing State tree

Formula may be included in an existing `sls` file. This is often useful when a state you are writing needs to `require` or `extend` a state defined in the formula.

Here is an example of a state that uses the `epel-formula` in a `require` declaration which directs Salt to not install the `python26` package until after the EPEL repository has also been installed:

```
include:
  - epel

python26:
  pkg:
    - installed
    - require:
      - pkg: epel
```

Including a Formula from a Top File

Some Formula perform completely standalone installations that are not referenced from other state files. It is usually cleanest to include these Formula directly from a Top File.

For example the easiest way to set up an OpenStack deployment on a single machine is to include the `openstack-standalone-formula` directly from a `top.sls` file:

```
base:
  'myopenstackmaster':
    - openstack
```

Quickly deploying OpenStack across several dedicated machines could also be done directly from a Top File and may look something like this:

```
base:
  'controller':
    - openstack.horizon
    - openstack.keystone
  'hyper-*':
    - openstack.nova
    - openstack.glance
  'storage-*':
    - openstack.swift
```

Configuring Formula using Pillar

Salt Formulas are designed to work out of the box with no additional configuration. However, many Formula support additional configuration and customization through *Pillar*. Examples of available options can be found in a file named `pillar.example` in the root directory of each Formula repository.

Modifying default Formula behavior

Remember that Formula are regular Salt States and can be used with all Salt's normal mechanisms for determining execution order. Formula can be required from other States with `require` declarations, they can be modified using `extend`, they can made to watch other states with `watch_in`, they can be used as templates for other States with `use`. Don't be shy to read through the source for each Formula!

Reporting problems & making additions

Each Formula is a separate repository on GitHub. If you encounter a bug with a Formula please file an issue in the respective repository! Send fixes and additions as a pull request. Add tips and tricks to the repository wiki.

46.1.3 Writing Formulas

Each Formula is a separate repository in the saltstack-formulas organization on GitHub.

Note: Get involved creating new Formulas

The best way to create new Formula repositories for now is to create a repository in your own account on GitHub and notify a SaltStack employee when it is ready. We will add you as a collaborator on the saltstack-formulas organization and help you transfer the repository over. Ping a SaltStack employee on IRC (`#salt` on Freenode) or send an email to the Salt mailing list.

Repository structure

A basic Formula repository should have the following layout:

```
foo-formula
|-- foo/
|   |-- map.jinja
|   |-- init.sls
|   `-- bar.sls
|-- CHANGELOG.rst
|-- LICENSE
|-- pillar.example
|-- README.rst
`-- VERSION
```

See also:

`template-formula`

The `template-formula` repository has a pre-built layout that serves as the basic structure for a new formula repository. Just copy the files from there and edit them.

README.rst

The README should detail each available `.sls` file by explaining what it does, whether it has any dependencies on other formulas, whether it has a target platform, and any other installation or usage instructions or tips.

A sample skeleton for the `README.rst` file:

```
foo
===

Install and configure the FOO service.

.. note::

    See the full `Salt Formulas installation and usage instructions
    <http://docs.saltstack.com/topics/conventions/formulas.html>`_.

Available states
----------------

``foo``
    Install the ``foo`` package and enable the service.
``foo.bar``
    Install the ``bar`` package.
```

CHANGELOG.rst

The `CHANGELOG.rst` file should detail the individual versions, their release date and a set of bullet points for each version highlighting the overall changes in a given version of the formula.

A sample skeleton for the *CHANGELOG.rst* file:

`CHANGELOG.rst`:

```
foo formula
===========

0.0.2 (2013-01-01)

- Re-organized formula file layout
- Fixed filename used for upstart logger template
- Allow for pillar message to have default if none specified
```

map.jinja

It is useful to have a single source for platform-specific or other parameterized information that can be reused throughout a Formula. See "*Configuration and parameterization*" below for more information. Such a file should be named `map.jinja` and live alongside the state files.

The following is an example from the MySQL Formula that has been slightly modified to be more readable and less terse.

In essence, it is a simple dictionary that serves as a lookup table. The `grains.filter_by` function then does a lookup on that table using the `os_family` grain (by default) and sets the result to a variable that can be used throughout the formula.

See also:

`grains.filter_by`

`map.jinja`:

```
{% set mysql_lookup_table = {
    'Debian': {
        'server': 'mysql-server',
        'client': 'mysql-client',
        'service': 'mysql',
        'config': '/etc/mysql/my.cnf',
    },
    'RedHat': {
        'server': 'mysql-server',
        'client': 'mysql',
        'service': 'mysqld',
        'config': '/etc/my.cnf',
    },
    'Gentoo': {
        'server': 'dev-db/mysql',
        'mysql-client': 'dev-db/mysql',
        'service': 'mysql',
        'config': '/etc/mysql/my.cnf',
    },
} %}

{% set mysql = salt['grains.filter_by'](mysql_lookup_table,
    merge=salt['pillar.get']('mysql:lookup'))
```

The above example is used to help explain how the mapping works. In most map files you will see the following structure:

```
{% set mysql = salt['grains.filter_by']({
    'Debian': {
        'server': 'mysql-server',
        'client': 'mysql-client',
        'service': 'mysql',
        'config': '/etc/mysql/my.cnf',
        'python': 'python-mysqldb',
    },
    'RedHat': {
        'server': 'mysql-server',
        'client': 'mysql',
        'service': 'mysqld',
        'config': '/etc/my.cnf',
        'python': 'MySQL-python',
    },
    'Gentoo': {
        'server': 'dev-db/mysql',
        'mysql-client': 'dev-db/mysql',
        'service': 'mysql',
        'config': '/etc/mysql/my.cnf',
        'python': 'dev-python/mysql-python',
    },
}, merge=salt['pillar.get']('mysql:lookup')) %}
```

The `merge` keyword specifies the location of a dictionary in Pillar that can be used to override values returned from the lookup table. If the value exists in Pillar it will take precedence, otherwise `merge` will be ignored. This is useful when software or configuration files is installed to non-standard locations. For example, the following Pillar would replace the `config` value from the call above.

```
mysql:
  lookup:
    config: /usr/local/etc/mysql/my.cnf
```

Any of the values defined above can be fetched for the current platform in any state file using the following syntax:

```
{% from "mysql/map.jinja" import mysql with context %}

mysql-server:
  pkg:
    - installed
    - name: {{ mysql.server }}
  service:
    - running
    - name: {{ mysql.service }}
    - require:
      - pkg: mysql-server

mysql-config:
  file:
    - managed
    - name: {{ mysql.config }}
    - source: salt://mysql/conf/my.cnf
    - watch:
      - service: mysql-server
```

SLS files

Each state in a Formula should use sane defaults (as much as is possible) and use Pillar to allow for customization.

The root state, in particular, and most states in general, should strive to do no more than the basic expected thing and advanced configuration should be put in child states build on top of the basic states.

For example, the root Apache should only install the Apache httpd server and make sure the httpd service is running. It can then be used by more advanced states:

```
# apache/init.sls
httpd:
  pkg:
    - installed
  service:
    - running

# apache/mod_wsgi.sls
include:
  - apache

mod_wsgi:
  pkg:
    - installed
    - require:
      - pkg: apache

# apache/debian/vhost_setup.sls
{% if grains['os_family'] == 'Debian' %}
a2dissite 000-default:
  cmd.run:
    - onlyif: test -L /etc/apache2/sites-enabled/000-default
    - require:
      - pkg: apache
{% endif %}
```

Platform agnostic

Each Salt Formula must be able to be run without error on any platform. If the formula is not applicable to a platform it should do nothing. See the `epel-formula` for an example.

Any platform-specific states must be wrapped in conditional statements:

```
{% if grains['os_family'] == 'Debian' %}
...
{% endif %}
```

A handy method for using platform-specific values is to create a lookup table using the `filter_by()` function:

```
{% set apache = salt['grains.filter_by']({
    'Debian': {'conf': '/etc/apache2/conf.d'},
    'RedHat': {'conf': '/etc/httpd/conf.d'},
}) %}

myconf:
  file:
    - managed
    - name: {{ apache.conf }}/myconf.conf
```

Configuration and parameterization

Each Formula should strive for sane defaults that can then be customized using Pillar. Pillar lookups must use the safe `get()` and must provide a default value:

```
{% if salt['pillar.get']('horizon:use_ssl', False) %}
ssl_crt: {{ salt['pillar.get']('horizon:ssl_crt', '/etc/ssl/certs/horizon.crt') }}
ssl_key: {{ salt['pillar.get']('horizon:ssl_key', '/etc/ssl/certs/horizon.key') }}
{% endif %}
```

Any default values used in the Formula must also be documented in the `pillar.example` file in the root of the repository. Comments should be used liberally to explain the intent of each configuration value. In addition, users should be able copy-and-paste the contents of this file into their own Pillar to make any desired changes.

Scripting

Remember that both State files and Pillar files can easily call out to Salt *execution modules* and have access to all the system grains as well.

```
{% if '/storage' in salt['mount.active']() %}
/usr/local/etc/myfile.conf:
  file:
    - symlink
    - target: /storage/myfile.conf
{% endif %}
```

Jinja macros are generally discouraged in favor of adding functions to existing Salt modules or adding new modules. An example of this is the `filter_by()` function.

Versioning

Formula are versioned according to Semantic Versioning, http://semver.org/.

> Given a version number MAJOR.MINOR.PATCH, increment the:
>
> 1. MAJOR version when you make incompatible API changes,
> 2. MINOR version when you add functionality in a backwards-compatible manner, and
> 3. PATCH version when you make backwards-compatible bug fixes.
>
> Additional labels for pre-release and build metadata are available as extensions to the MAJOR.MINOR.PATCH format.

Formula versions are tracked using Git tags as well as the `VERSION` file in the formula repository. The `VERSION` file should contain the currently released version of the particular formula.

Testing Formulas

Salt Formulas are tested by running each `.sls` file via `state.sls` and checking the output for success or failure. This is done for each supported platform.

46.2 SaltStack Packaging Guide

Since Salt provides a powerful toolkit for system management and automation, the package can be spit into a number of sub-tools. While packaging Salt as a single package containing all components is perfectly acceptable, the split packages should follow this convention.

46.2.1 Source Files

Release packages should always be built from the source tarball distributed via pypi. Release packages should *NEVER* use a git checkout as the source for distribution.

46.2.2 Single Package

Shipping Salt as a single package, where the minion, master and all tools are together is perfectly acceptable and practiced by distributions such as FreeBSD.

46.2.3 Split Package

Salt Should always be split in a standard way, with standard dependencies, this lowers cross distribution confusion about what components are going to be shipped with specific packages. These packages can be defined from the Salt Source as of Salt 0.17.0:

Salt Common

The *salt-common* or *salt* package should contain the files provided by the salt python package, or all files distributed from the `salt/` directory in the source distribution packages. The documentation contained under the `doc/` directory can be a part of this package but splitting out a doc package is preferred. Since salt-call is the entry point to utilize the libs and is useful for all salt packages it is included in the salt-common package.

Name

- *salt* OR *salt-common*

Files

- *salt/**
- *man/salt.7*
- *scripts/salt-call*
- *tests/**
- *man/salt-call.1*

Depends

- *Python 2.6-2.7*
- *PyYAML*
- *Jinja2*

Salt Master

The *salt-master* package contains the applicable scripts, related man pages and init information for the given platform.

Name

- *salt-master*

Files

- *scripts/salt-master*
- *scripts/salt*
- *scripts/salt-run*
- *scripts/salt-key*
- *scripts/salt-cp*
- *pkg/<master init data>*
- *man/salt.1*
- *man/salt-master.1*
- *man/salt-run.1*
- *man/salt-key.1*
- *man/salt-cp.1*
- *conf/master*

Depends

- *Salt Common*
- *ZeroMQ* >= 3.2
- *PyZMQ* >= 2.10
- *PyCrypto*
- *M2Crypto*
- *Python MessagePack* (Messagepack C lib, or msgpack-pure)

Salt Syndic

The Salt Syndic package can be rolled completely into the Salt Master package. Platforms which start services as part of the package deployment need to maintain a separate *salt-syndic* package (primarily Debian based platforms).

The Syndic may optionally not depend on the anything more than the Salt Master since the master will bring in all needed dependencies, but fall back to the platform specific packaging guidelines.

Name

- *salt-syndic*

Files

- *scripts/salt-syndic*
- *pkg/<syndic init data>*
- *man/salt-syndic.1*

Depends

- *Salt Common*
- *Salt Master*
- *ZeroMQ* >= 3.2
- *PyZMQ* >= 2.10
- *PyCrypto*
- *M2Crypto*
- *Python MessagePack* (Messagepack C lib, or msgpack-pure)

Salt Minion

The Minion is a standalone package and should not be split beyond the *salt-minion* and *salt-common* packages.

Name

- *salt-minion*

Files

- *scripts/salt-minion*
- *pkg/<minion init data>*
- *man/salt-minion.1*
- *conf/minion*

Depends

- *Salt Common*
- *ZeroMQ* >= 3.2
- *PyZMQ* >= 2.10
- *PyCrypto*
- *M2Crypto*
- *Python MessagePack* (Messagepack C lib, or msgpack-pure)

Salt SSH

Since Salt SSH does not require the same dependencies as the minion and master, it should be split out.

Name

- *salt-ssh*

Files

- *scripts/salt-ssh*
- *man/salt-ssh.1*

Depends

- *Salt Common*
- *sshpass*
- *Python MessagePack* (Messagepack C lib, or msgpack-pure)

Salt Doc

The documentation package is very distribution optional. A completely split package will split out the documentation, but some platform conventions do not prefer this. If the documentation is not split out, it should be included with the *Salt Common* package.

Name

- *salt-doc*

Files

- *doc/**

Optional Depends

- *Salt Common*
- *Python Sphinx*
- *Make*

46.3 Salt Release Process

The goal for Salt projects is to cut a new feature release every four to six weeks. This document outlines the process for these releases, and the subsequent bug fix releases which follow.

46.3.1 Feature Release Process

When a new release is ready to be cut, the person responsible for cutting the release will follow the following steps (written using the 0.16 release as an example):

1. All open issues on the release milestone should be moved to the next release milestone. (e.g. from the `0.16` milestone to the `0.17` milestone)
2. Release notes should be created documenting the major new features and bugfixes in the release.
3. Create an annotated tag with only the major and minor version numbers, preceded by the letter `v`. (e.g. `v0.16`) This tag will reside on the `develop` branch.
4. Create a branch for the new release, using only the major and minor version numbers. (e.g. `0.16`)
5. On this new branch, create an annotated tag for the first revision release, which is generally a release candidate. It should be preceded by the letter `v`. (e.g. `v0.16.0RC`)
6. The release should be packaged from this annotated tag and uploaded to PyPI.
7. The packagers should be notified on the `salt-packagers` mailing list so they can create packages for all the major operating systems. (note that release candidates should go in the testing repositories)
8. After the packagers have been given a few days to compile the packages, the release is announced on the `salt-users` mailing list.
9. Log into RTD and add the new release there. (Have to do it manually)

46.3.2 Maintenance and Bugfix Releases

Once a release has been cut, regular cherry-picking sessions should begin to cherry-pick any bugfixes from the `develop` branch to the release branch (e.g. `0.16`). Once major bugs have been fixes and cherry-picked, a bugfix release can be cut:

1. On the release branch (i.e. `0.16`), create an annotated tag for the revision release. It should be preceded by the letter `v`. (e.g. `v0.16.2`) Release candidates are unnecessary for bugfix releases.
2. The release should be packaged from this annotated tag and uploaded to PyPI.
3. The packagers should be notified on the `salt-packagers` mailing list so they can create packages for all the major operating systems.
4. After the packagers have been given a few days to compile the packages, the release is announced on the `salt-users` mailing list.

46.4 Salt Coding Style

Salt is developed with a certain coding style, while the style is dominantly PEP 8 it is not completely PEP 8. It is also noteworthy that a few development techniques are also employed which should be adhered to. In the end, the code is made to be "Salty".

Most importantly though, we will accept code that violates the coding style and KINDLY ask the contributor to fix it, or go ahead and fix the code on behalf of the contributor. Coding style is NEVER grounds to reject code contributions, and is never grounds to talk down to another member of the community (There are no grounds to treat others without respect, especially people working to improve Salt)!!

46.4.1 Linting

Most Salt style conventions are codified in Salt's *.pylintrc* file. This file is found in the root of the Salt project and can be passed as an argument to the pylint program as follows:

pylint –rcfile=/path/to/salt/.pylintrc salt/dir/to/lint

46.4.2 Strings

Salt follows a few rules when formatting strings:

Single Quotes

In Salt, all strings use single quotes unless there is a good reason not to. This means that docstrings use single quotes, standard strings use single quotes etc.:

```
def foo():
    '''
    A function that does things
    '''
    name = 'A name'
    return name
```

Formatting Strings

All strings which require formatting should use the *.format* string method:

```
data = 'some text'
more = '{0} and then some'.format(data)
```

Make sure to use indices or identifiers in the format brackets, since empty brackets are not supported by python 2.6.

Please do NOT use printf formatting.

Docstring Conventions

Docstrings should always add a newline, docutils takes care of the new line and it makes the code cleaner and more vertical:

GOOD:

```
def bar():
    '''
    Here lies a docstring with a newline after the quotes and is the salty
    way to handle it! Vertical code is the way to go!
    '''
    return
```

BAD:

```
def baz():
    '''This is not ok!'''
    return
```

When adding a new function or state, where possible try to use a `versionadded` directive to denote when the function or state was added.

```
def new_func(msg=''):
    '''
    .. versionadded:: 0.16.0

    Prints what was passed to the function.

    msg : None
        The string to be printed.
    '''
    print msg
```

If you are uncertain what version should be used, either *consult a core developer in IRC* or bring this up when opening your *pull request* and a core developer will add the proper version once your pull request has been merged. Bugfixes will be available in a bugfix release (i.e. 0.17.1, the first bugfix release for 0.17.0), while new features are held for feature releases, and this will affect what version number should be used in the `versionadded` directive.

Similar to the above, when an existing function or state is modified (for example, when an argument is added), then under the explanation of that new argument a `versionadded` directive should be used to note the version in which the new argument was added. If an argument's function changes significantly, the `versionchanged` directive can be used to clarify this:

```
def new_func(msg='', signature=''):
    '''
    .. versionadded:: 0.16.0
    Prints what was passed to the function.

    msg : None
        The string to be printed. Will be prepended with 'Greetings! '.

    .. versionchanged:: 0.17.1

    signature : None
        An optional signature.

    .. versionadded 0.17.0
    '''
    print 'Greetings! {0}\n\n{1}'.format(msg, signature)
```

46.4.3 Imports

Salt code prefers importing modules and not explicit functions. This is both a style and functional preference. The functional preference originates around the fact that the module import system used by pluggable modules will include callable objects (functions) that exist in the direct module namespace. This is not only messy, but may unintentionally expose code python libs to the Salt interface and pose a security problem.

To say this more directly with an example, this is *GOOD*:

```
import os

def minion_path():
    path = os.path.join(self.opts['cachedir'], 'minions')
    return path
```

This on the other hand is *DISCOURAGED*:

```
from os.path import join

def minion_path():
    path = join(self.opts['cachedir'], 'minions')
    return path
```

The time when this is changed is for importing exceptions, generally directly importing exceptions is preferred:

This is a good way to import exceptions:

```
from salt.exceptions import CommandExecutionError
```

Absolute Imports

Although absolute imports seems like an awesome idea, please do not use it. Extra care would be necessary all over salt's code in order for absolute imports to work as supposed. Believe it, it has been tried before and, as a tried example, by renaming `salt.modules.sysmod` to `salt.modules.sys`, all other salt modules which needed to import `sys` would have to also import `absolute_import`, which should be avoided.

46.4.4 Vertical is Better

When writing Salt code, vertical code is generally preferred. This is not a hard rule but more of a guideline. As PEP 8 specifies, Salt code should not exceed 79 characters on a line, but it is preferred to separate code out into more newlines in some cases for better readability:

```
import os

os.chmod(
        os.path.join(self.opts['sock_dir'],
            'minion_event_pub.ipc'),
        448
        )
```

Where there are more line breaks, this is also apparent when constructing a function with many arguments, something very common in state functions for instance:

```
def managed(name,
        source=None,
        source_hash='',
        user=None,
        group=None,
        mode=None,
        template=None,
        makedirs=False,
        context=None,
        replace=True,
        defaults=None,
        env=None,
        backup='',
        **kwargs):
```

Note: Making function and class definitions vertical is only required if the arguments are longer then 80 characters. Otherwise, the formatting is optional and both are acceptable.

46.4.5 Indenting

Some confusion exists in the python world about indenting things like function calls, the above examples use 8 spaces when indenting comma-delimited constructs.

The confusion arises because the pep8 program INCORRECTLY flags this as wrong, where PEP 8, the document, cites only using 4 spaces here as wrong, as it doesn't differentiate from a new indent level.

Right:

```
def managed(name,
        source=None,
        source_hash='',
        user=None)
```

WRONG:

```
def managed(name,
    source=None,
    source_hash='',
    user=None)
```

Lining up the indent is also correct:

```
def managed(name,
            source=None,
            source_hash='',
            user=None)
```

This also applies to function calls and other hanging indents.

pep8 and Flake8 (and, by extension, the vim plugin Syntastic) will complain about the double indent for hanging indents. This is a known conflict between pep8 (the script) and the actual PEP 8 standard. It is recommended that this particular warning be ignored with the following lines in `~/.config/flake8`:

```
[flake8]
ignore = E226,E241,E242,E126
```

Make sure your Flake8/pep8 are up to date. The first three errors are ignored by default and are present here to keep the behavior the same. This will also work for pep8 without the Flake8 wrapper – just replace all instances of 'flake8' with 'pep8', including the filename.

46.4.6 Code Churn

Many pull requests have been submitted that only churn code in the name of PEP 8. Code churn is a leading source of bugs and is strongly discouraged. While style fixes are encouraged they should be isolated to a single file per commit, and the changes should be legitimate, if there are any questions about whether a style change is legitimate please reference this document and the official PEP 8 (http://www.python.org/dev/peps/pep-0008/) document before changing code. Many claims that a change is PEP 8 have been invalid, please double check before committing fixes.

CHAPTER 47

SaltStack Packaging Guide

Since Salt provides a powerful toolkit for system management and automation, the package can be spit into a number of sub-tools. While packaging Salt as a single package containing all components is perfectly acceptable, the split packages should follow this convention.

47.1 Source Files

Release packages should always be built from the source tarball distributed via pypi. Release packages should *NEVER* use a git checkout as the source for distribution.

47.2 Single Package

Shipping Salt as a single package, where the minion, master and all tools are together is perfectly acceptable and practiced by distributions such as FreeBSD.

47.3 Split Package

Salt Should always be split in a standard way, with standard dependencies, this lowers cross distribution confusion about what components are going to be shipped with specific packages. These packages can be defined from the Salt Source as of Salt 0.17.0:

47.3.1 Salt Common

The *salt-common* or *salt* package should contain the files provided by the salt python package, or all files distributed from the `salt/` directory in the source distribution packages. The documentation contained under the `doc/` directory can be a part of this package but splitting out a doc package is preferred. Since salt-call is the entry point to utilize the libs and is useful for all salt packages it is included in the salt-common package.

Name

- *salt* OR *salt-common*

Files

- *salt/**
- *man/salt.7*
- *scripts/salt-call*
- *tests/**
- *man/salt-call.1*

Depends

- *Python 2.6-2.7*
- *PyYAML*
- *Jinja2*

47.3.2 Salt Master

The *salt-master* package contains the applicable scripts, related man pages and init information for the given platform.

Name

- *salt-master*

Files

- *scripts/salt-master*
- *scripts/salt*
- *scripts/salt-run*
- *scripts/salt-key*
- *scripts/salt-cp*
- *pkg/<master init data>*
- *man/salt.1*
- *man/salt-master.1*
- *man/salt-run.1*
- *man/salt-key.1*
- *man/salt-cp.1*
- *conf/master*

Depends

- *Salt Common*
- *ZeroMQ* >= 3.2
- *PyZMQ* >= 2.10
- *PyCrypto*
- *M2Crypto*
- *Python MessagePack* (Messagepack C lib, or msgpack-pure)

47.3.3 Salt Syndic

The Salt Syndic package can be rolled completely into the Salt Master package. Platforms which start services as part of the package deployment need to maintain a separate *salt-syndic* package (primarily Debian based platforms).

The Syndic may optionally not depend on the anything more than the Salt Master since the master will bring in all needed dependencies, but fall back to the platform specific packaging guidelines.

Name

- *salt-syndic*

Files

- *scripts/salt-syndic*
- *pkg/<syndic init data>*
- *man/salt-syndic.1*

Depends

- *Salt Common*
- *Salt Master*
- *ZeroMQ* >= 3.2
- *PyZMQ* >= 2.10
- *PyCrypto*
- *M2Crypto*
- *Python MessagePack* (Messagepack C lib, or msgpack-pure)

47.3.4 Salt Minion

The Minion is a standalone package and should not be split beyond the *salt-minion* and *salt-common* packages.

Name

- *salt-minion*

Files

- *scripts/salt-minion*
- *pkg/<minion init data>*
- *man/salt-minion.1*
- *conf/minion*

Depends

- *Salt Common*
- *ZeroMQ* >= 3.2
- *PyZMQ* >= 2.10
- *PyCrypto*
- *M2Crypto*
- *Python MessagePack* (Messagepack C lib, or msgpack-pure)

47.3.5 Salt SSH

Since Salt SSH does not require the same dependencies as the minion and master, it should be split out.

Name

- *salt-ssh*

Files

- *scripts/salt-ssh*
- *man/salt-ssh.1*

Depends

- *Salt Common*
- *sshpass*
- *Python MessagePack* (Messagepack C lib, or msgpack-pure)

47.3.6 Salt Doc

The documentation package is very distribution optional. A completely split package will split out the documentation, but some platform conventions do not prefer this. If the documentation is not split out, it should be included with the *Salt Common* package.

47.3.7 Name

- *salt-doc*

Files

- *doc/**

Optional Depends

- *Salt Common*
- *Python Sphinx*
- *Make*

CHAPTER 48

Salt Release Process

The goal for Salt projects is to cut a new feature release every four to six weeks. This document outlines the process for these releases, and the subsequent bug fix releases which follow.

48.1 Feature Release Process

When a new release is ready to be cut, the person responsible for cutting the release will follow the following steps (written using the 0.16 release as an example):

1. All open issues on the release milestone should be moved to the next release milestone. (e.g. from the `0.16` milestone to the `0.17` milestone)
2. Release notes should be created documenting the major new features and bugfixes in the release.
3. Create an annotated tag with only the major and minor version numbers, preceded by the letter `v`. (e.g. `v0.16`) This tag will reside on the `develop` branch.
4. Create a branch for the new release, using only the major and minor version numbers. (e.g. `0.16`)
5. On this new branch, create an annotated tag for the first revision release, which is generally a release candidate. It should be preceded by the letter `v`. (e.g. `v0.16.0RC`)
6. The release should be packaged from this annotated tag and uploaded to PyPI.
7. The packagers should be notified on the `salt-packagers` mailing list so they can create packages for all the major operating systems. (note that release candidates should go in the testing repositories)
8. After the packagers have been given a few days to compile the packages, the release is announced on the `salt-users` mailing list.
9. Log into RTD and add the new release there. (Have to do it manually)

48.2 Maintenance and Bugfix Releases

Once a release has been cut, regular cherry-picking sessions should begin to cherry-pick any bugfixes from the `develop` branch to the release branch (e.g. `0.16`). Once major bugs have been fixes and cherry-picked, a bugfix release can be cut:

1. On the release branch (i.e. `0.16`), create an annotated tag for the revision release. It should be preceded by the letter `v`. (e.g. `v0.16.2`) Release candidates are unnecessary for bugfix releases.
2. The release should be packaged from this annotated tag and uploaded to PyPI.

3. The packagers should be notified on the `salt-packagers` mailing list so they can create packages for all the major operating systems.
4. After the packagers have been given a few days to compile the packages, the release is announced on the `salt-users` mailing list.

CHAPTER 49

Salt Coding Style

Salt is developed with a certain coding style, while the style is dominantly PEP 8 it is not completely PEP 8. It is also noteworthy that a few development techniques are also employed which should be adhered to. In the end, the code is made to be "Salty".

Most importantly though, we will accept code that violates the coding style and KINDLY ask the contributor to fix it, or go ahead and fix the code on behalf of the contributor. Coding style is NEVER grounds to reject code contributions, and is never grounds to talk down to another member of the community (There are no grounds to treat others without respect, especially people working to improve Salt)!!

49.1 Linting

Most Salt style conventions are codified in Salt's *.pylintrc* file. This file is found in the root of the Salt project and can be passed as an argument to the pylint program as follows:

pylint –rcfile=/path/to/salt/.pylintrc salt/dir/to/lint

49.2 Strings

Salt follows a few rules when formatting strings:

49.2.1 Single Quotes

In Salt, all strings use single quotes unless there is a good reason not to. This means that docstrings use single quotes, standard strings use single quotes etc.:

```
def foo():
    '''
    A function that does things
    '''
    name = 'A name'
    return name
```

49.2.2 Formatting Strings

All strings which require formatting should use the *.format* string method:

```
data = 'some text'
more = '{0} and then some'.format(data)
```

Make sure to use indices or identifiers in the format brackets, since empty brackets are not supported by python 2.6.

Please do NOT use printf formatting.

49.2.3 Docstring Conventions

Docstrings should always add a newline, docutils takes care of the new line and it makes the code cleaner and more vertical:

GOOD:

```
def bar():
    '''
    Here lies a docstring with a newline after the quotes and is the salty
    way to handle it! Vertical code is the way to go!
    '''
    return
```

BAD:

```
def baz():
    '''This is not ok!'''
    return
```

When adding a new function or state, where possible try to use a `versionadded` directive to denote when the function or state was added.

```
def new_func(msg=''):
    '''
    .. versionadded:: 0.16.0

    Prints what was passed to the function.

    msg : None
        The string to be printed.
    '''
    print msg
```

If you are uncertain what version should be used, either *consult a core developer in IRC* or bring this up when opening your *pull request* and a core developer will add the proper version once your pull request has been merged. Bugfixes will be available in a bugfix release (i.e. 0.17.1, the first bugfix release for 0.17.0), while new features are held for feature releases, and this will affect what version number should be used in the `versionadded` directive.

Similar to the above, when an existing function or state is modified (for example, when an argument is added), then under the explanation of that new argument a `versionadded` directive should be used to note the version in which the new argument was added. If an argument's function changes significantly, the `versionchanged` directive can be used to clarify this:

```
def new_func(msg='', signature=''):
    '''
    .. versionadded:: 0.16.0

    Prints what was passed to the function.

    msg : None
        The string to be printed. Will be prepended with 'Greetings! '.
    .. versionchanged:: 0.17.1

    signature : None
        An optional signature.

    .. versionadded 0.17.0
    '''
    print 'Greetings! {0}\n\n{1}'.format(msg, signature)
```

49.3 Imports

Salt code prefers importing modules and not explicit functions. This is both a style and functional preference. The functional preference originates around the fact that the module import system used by pluggable modules will include callable objects (functions) that exist in the direct module namespace. This is not only messy, but may unintentionally expose code python libs to the Salt interface and pose a security problem.

To say this more directly with an example, this is *GOOD*:

```
import os

def minion_path():
    path = os.path.join(self.opts['cachedir'], 'minions')
    return path
```

This on the other hand is *DISCOURAGED*:

```
from os.path import join

def minion_path():
    path = join(self.opts['cachedir'], 'minions')
    return path
```

The time when this is changed is for importing exceptions, generally directly importing exceptions is preferred:

This is a good way to import exceptions:

```
from salt.exceptions import CommandExecutionError
```

49.3.1 Absolute Imports

Although absolute imports seems like an awesome idea, please do not use it. Extra care would be necessary all over salt's code in order for absolute imports to work as supposed. Believe it, it has been tried before and, as a tried example, by renaming `salt.modules.sysmod` to `salt.modules.sys`, all other salt modules which needed to import `sys` would have to also import `absolute_import`, which should be avoided.

49.4 Vertical is Better

When writing Salt code, vertical code is generally preferred. This is not a hard rule but more of a guideline. As PEP 8 specifies, Salt code should not exceed 79 characters on a line, but it is preferred to separate code out into more newlines in some cases for better readability:

```
import os

os.chmod(
        os.path.join(self.opts['sock_dir'],
            'minion_event_pub.ipc'),
        448
        )
```

Where there are more line breaks, this is also apparent when constructing a function with many arguments, something very common in state functions for instance:

```
def managed(name,
        source=None,
        source_hash='',
        user=None,
        group=None,
        mode=None,
        template=None,
        makedirs=False,
        context=None,
        replace=True,
        defaults=None,
        env=None,
        backup='',
        **kwargs):
```

Note: Making function and class definitions vertical is only required if the arguments are longer then 80 characters. Otherwise, the formatting is optional and both are acceptable.

49.5 Indenting

Some confusion exists in the python world about indenting things like function calls, the above examples use 8 spaces when indenting comma-delimited constructs.

The confusion arises because the pep8 program INCORRECTLY flags this as wrong, where PEP 8, the document, cites only using 4 spaces here as wrong, as it doesn't differentiate from a new indent level.

Right:

```
def managed(name,
        source=None,
        source_hash='',
        user=None)
```

WRONG:

```
def managed(name,
    source=None,
    source_hash='',
    user=None)
```

Lining up the indent is also correct:

```
def managed(name,
            source=None,
            source_hash='',
            user=None)
```

This also applies to function calls and other hanging indents.

pep8 and Flake8 (and, by extension, the vim plugin Syntastic) will complain about the double indent for hanging indents. This is a known conflict between pep8 (the script) and the actual PEP 8 standard. It is recommended that this particular warning be ignored with the following lines in `~/.config/flake8`:

```
[flake8]
ignore = E226,E241,E242,E126
```

Make sure your Flake8/pep8 are up to date. The first three errors are ignored by default and are present here to keep the behavior the same. This will also work for pep8 without the Flake8 wrapper – just replace all instances of 'flake8' with 'pep8', including the filename.

49.6 Code Churn

Many pull requests have been submitted that only churn code in the name of PEP 8. Code churn is a leading source of bugs and is strongly discouraged. While style fixes are encouraged they should be isolated to a single file per commit, and the changes should be legitimate, if there are any questions about whether a style change is legitimate please reference this document and the official PEP 8 (http://www.python.org/dev/peps/pep-0008/) document before changing code. Many claims that a change is PEP 8 have been invalid, please double check before committing fixes.

CHAPTER 50

SaltStack Git Policy

The SaltStack team follows a git policy to maintain stability and consistency with the repository. The git policy has been developed to encourage contributions and make contributing to Salt as easy as possible. Code contributors to SaltStack projects DO NOT NEED TO READ THIS DOCUMENT, because all contributions come into SaltStack via a single gateway to make it as easy as possible for contributors to give us code.

The primary rule of git management in SaltStack is to make life easy on contributors and developers to send in code. Simplicity is always a goal!

50.1 New Code Entry

All new SaltStack code is posted to the *develop* branch, this is the single point of entry. The only exception here is when a bugfix to develop cannot be cleanly merged into a release branch and the bugfix needs to be rewritten for the release branch.

50.2 Release Branching

SaltStack maintains two types of releases, *Feature Releases* and *Point Releases*. A feature release is managed by incrementing the first or second release point number, so 0.10.5 -> 0.11.0 signifies a feature release and 0.11.0 -> 0.11.1 signifies a point release, also a hypothetical 0.42.7 -> 1.0.0 would also signify a feature release.

50.2.1 Feature Release Branching

Each feature release is maintained in a dedicated git branch derived from the last applicable release commit on develop. All file changes relevant to the feature release will be completed in the develop branch prior to the creation of the feature release branch. The feature release branch will be named after the relevant numbers to the feature release, which constitute the first two numbers. This means that the release branch for the 0.11.0 series is named 0.11.

A feature release branch is created with the following command:

```
# git checkout -b 0.11 # From the develop branch
# git push origin 0.11
```

50.2.2 Point Releases

Each point release is derived from its parent release branch. Constructing point releases is a critical aspect of Salt development and is managed by members of the core development team. Point releases comprise bug and security fixes which are cherry picked from develop onto the aforementioned release branch. At the time when a core developer accepts a pull request a determination needs to be made if the commits in the pull request need to be backported to the release branch. Some simple criteria are used to make this determination:

- Is this commit fixing a bug? Backport
- Does this commit change or add new features in any way? Don't backport
- Is this a PEP8 or code cleanup commit? Don't backport
- Does this commit fix a security issue? Backport

Determining when a point release is going to be made is up to the project leader (Thomas Hatch). Generally point releases are made every 1-2 weeks or if there is a security fix they can be made sooner.

The point release is only designated by tagging the commit on the release branch with release number using the existing convention (version 0.11.1 is tagged with v0.11.1). From the tag point a new source tarball is generated and published to PyPI, and a release announcement is made.

CHAPTER 51

Salt Development Guidelines

51.1 Deprecating Code

Salt should remain backwards compatible, though sometimes, this backwards compatibility needs to be broken because a specific feature and/or solution is no longer necessary or required. At first one might think, let me change this code, it seems that it's not used anywhere else so it should be safe to remove. Then, once there's a new release, users complain about functionality which was removed and they where using it, etc. This should, at all costs, be avoided, and, in these cases, *that* specific code should be deprecated.

Depending on the complexity and usage of a specific piece of code, the deprecation time frame should be properly evaluated. As an example, a deprecation warning which is shown for 2 major releases, for example *0.17.0* and *2014.1.0*, gives users enough time to stop using the deprecated code and adapt to the new one.

For example, if you're deprecating the usage of a keyword argument to a function, that specific keyword argument should remain in place for the full deprecation time frame and if that keyword argument is used, a deprecation warning should be shown to the user.

To help in this deprecation task, salt provides `salt.utils.warn_until`. The idea behind this helper function is to show the deprecation warning until salt reaches the provided version. Once that provided version is equaled `salt.utils.warn_until` will raise a `RuntimeError` making salt stop its execution. This stoppage is unpleasant and will remind the developer that the deprecation limit has been reached and that the code can then be safely removed.

Consider the following example:

```
def some_function(bar=False, foo=None):
    if foo is not None:
        salt.utils.warn_until(
            (0, 18),
            'The \'foo\' argument has been deprecated and its '
            'functionality removed, as such, its usage is no longer '
            'required.'
        )
```

Consider that the current salt release is `0.16.0`. Whenever `foo` is passed a value different from `None` that warning will be shown to the user. This will happen in versions `0.16.2` to `2014.1.0`, after which a `RuntimeError` will be raised making us aware that the deprecated code should now be removed.

51.2 Dunder Dictionaries

Salt provides several special "dunder" dictionaries as a convenience for Salt development. These include `__opts__`, `__context__`, `__salt__`, and others. This document will describe each dictionary and detail where they exist and what information and/or functionality they provide.

51.2.1 __opts__

Available in

- All loader modules

The `__opts__` dictionary contains all of the options passed in the configuration file for the master or minion.

Note: In many places in salt, instead of pulling raw data from the __opts__ dict, configuration data should be pulled from the salt *get* functions such as config.get, aka - __salt__['config.get']('foo:bar') The *get* functions also allow for dict traversal via the *:* delimiter. Consider using get functions whenever using __opts__ or __pillar__ and __grains__ (when using grains for configuration data)

The configuration file data made available in the `__opts__` dictionary is the configuration data relative to the running daemon. If the modules are loaded and executed by the master, then the master configuration data is available, if the modules are executed by the minion, then the minion configuration is available. Any additional information passed into the respective configuration files is made available

51.2.2 __salt__

Available in

- Execution Modules
- State Modules
- Returners

`__salt__` contains the execution module functions. This allows for all functions to be called as they have been set up by the salt loader.

```
__salt__['cmd.run']('fdisk -l')
__salt__['network.ip_addrs']()
```

51.2.3 __grains__

Available in

- Execution Modules
- State Modules
- Returners
- External Pillar

The `__grains__` dictionary contains the grains data generated by the minion that is currently being worked with. In execution modules, state modules and returners this is the grains of the minion running the calls, when generating the external pillar the `__grains__` is the grains data from the minion that the pillar is being generated for.

51.2.4 __pillar__

Available in

- Execution Modules
- State Modules
- Returners

The `__pillar__` dictionary contains the pillar for the respective minion.

51.2.5 __context__

`__context__` exists in state modules and execution modules.

During a state run the `__context__` dictionary persists across all states that are run and then is destroyed when the state ends.

When running an execution module `__context__` persists across all module executions until the modules are refreshed; such as when `saltutils.sync_all` or `state.highstate` are executed.

A great place to see how to use `__context__` is in the cp.py module in salt/modules/cp.py. The fileclient authenticates with the master when it is instantiated and then is used to copy files to the minion. Rather than create a new fileclient for each file that is to be copied down, one instance of the fileclient is instantiated in the `__context__` dictionary and is reused for each file. Here is an example from salt/modules/cp.py:

```
if not 'cp.fileclient' in __context__:
    __context__['cp.fileclient'] = salt.fileclient.get_file_client(__opts__)
```

Note: Because __context__ may or may not have been destroyed, always be sure to check for the existence of the key in __context__ and generate the key before using it.

51.3 External Pillars

Salt provides a mechanism for generating pillar data by calling external pillar interfaces. This document will describe an outline of an ext_pillar module.

51.3.1 Location

Salt expects to find your `ext_pillar` module in the same location where it looks for other python modules. If the `extension_modules` option in your Salt master configuration is set, Salt will look for a `pillar` directory under there and load all the modules it finds. Otherwise, it will look in your Python site-packages `salt/pillar` directory.

51.3.2 Configuration

The external pillars that are called when a minion refreshes its pillars is controlled by the `ext_pillar` option in the Salt master configuration. You can pass a single argument, a list of arguments or a dictionary of arguments to your pillar:

```
ext_pillar:
  - example_a: some argument
  - example_b:
    - argumentA
    - argumentB
  - example_c:
      keyA: valueA
      keyB: valueB
```

51.3.3 The Module

51.3.4 Imports and Logging

Import modules your external pillar module needs. You should first include generic modules that come with stock Python:

```
import logging
```

And then start logging. This is an idiomatic way of setting up logging in Salt:

```
log = logging.getLogger(__name__)
```

Finally, load modules that are specific to what you are doing. You should catch import errors and set a flag that the the `__virtual__` function can use later.

```
try:
    import weird_thing
    example_a_loaded = True
except ImportError:
    example_a_loaded = False
```

51.3.5 Options

If you define an `__opts__` dictionary, it will be merged into the `__opts__` dictionary handed to the `ext_pillar` function later. This is a good place to put default configuration items. The convention is to name things `modulename.option`.

```
__opts__ = { 'example_a.someconfig': 137 }
```

51.3.6 Initialization

If you define an `__init__` function, it will be called with the following signature:

```
def __init__( __opts__ ):
    # Do init work here
```

Note: The `__init__` function is ran every time a particular minion causes the external pillar to be called, so don't put heavy initialization code here. The `__init__` functionality is a side-effect of the Salt loader, so it may not be as useful in pillars as it is in other Salt items.

51.3.7 __virtual__

If you define a `__virtual__` function, you can control whether or not this module is visible. If it returns `False` then Salt ignores this module. If it returns a string, then that string will be how Salt identifies this external pillar in its `ext_pillar` configuration. If this function does not exist, then the name Salt's `ext_pillar` will use to identify this module is its conventional name in Python.

This is useful to write modules that can be installed on all Salt masters, but will only be visible if a particular piece of software your module requires is installed.

```
# This external pillar will be known as 'example_a'
def __virtual__():
    if example_a_loaded:
        return 'example_a'
    else:
        return False

# This external pillar will be known as 'something_else'
def __virtual__():
    if example_a_loaded:
        return 'something_else'
    else:
        return False
```

51.3.8 ext_pillar

This is where the real work of an external pillar is done. If this module is active and has a function called `ext_pillar`, whenever a minion updates its pillar this function is called.

How it is called depends on how it is configured in the Salt master configuration. The first argument is always the current pillar dictionary, this contains pillar items that have already been added, starting with the data from `pillar_roots`, and then from any already-ran external pillars.

Using our example above:

```
ext_pillar( id, pillar, 'some argument' )                   # example_a
ext_pillar( id, pillar, 'argumentA', 'argumentB' )          # example_b
ext_pillar( id, pillar, keyA='valueA', keyB='valueB' } )    # example_c
```

In the `example_a` case, `pillar` will contain the items from the `pillar_roots`, in `example_b` `pillar` will contain that plus the items added by `example_a`, and in `example_c` `pillar` will contain that plus the items added by `example_b`. In all three cases, `id` will contain the ID of the minion making the pillar request.

This function should return a dictionary, the contents of which are merged in with all of the other pillars and returned to the minion. **Note**: this function is called once for each minion that fetches its pillar data.

```
def ext_pillar( minion_id, pillar, *args, **kwargs ):

    my_pillar = {}

    # Do stuff

    return my_pillar
```

You shouldn't just add items to `pillar` and return that, since that will cause Salt to merge data that already exists. Rather, just return the items you are adding or changing. You could, however, use `pillar` in your module to make some decision based on pillar data that already exists.

This function has access to some useful globals:

__opts__ A dictionary of mostly Salt configuration options. If you had an `__opts__` dictionary defined in your module, those values will be included.

__salt__ A dictionary of Salt module functions, useful so you don't have to duplicate functions that already exist. E.g. `__salt__['cmd.run']( 'ls -l' )` **Note**, runs on the *master*

__grains__ A dictionary of the grains of the minion making this pillar call.

51.3.9 Example configuration

As an example, if you wanted to add external pillar via the `cmd_json` external pillar, add something like this to your master config:

```
ext_pillar:
  - cmd_json: "echo {'arg':'value'}"
```

51.4 Logging Internals

TODO

51.5 Modular Systems

When first working with Salt, it is not always clear where all of the modular components are and what they do. Salt comes loaded with more modular systems than many users are aware of, making Salt very easy to extend in many places.

The most commonly used modular systems are execution modules and states. But the modular systems extend well beyond the more easily exposed components and are often added to Salt to make the complete system more flexible.

51.5.1 Execution Modules

Execution modules make up the core of the functionality used by Salt to interact with client systems. The execution modules create the core system management library used by all Salt systems, including states, which interact with minion systems.

Execution modules are completely open ended in their execution. They can be used to do anything required on a minion, from installing packages to detecting information about the system. The only restraint in execution modules is that the defined functions always return a JSON serializable object.

For a list of all built in execution modules, click *here*

For information on writing execution modules, see *this page*.

51.5.2 State Modules

State modules are used to define the state interfaces used by Salt States. These modules are restrictive in that they must follow a number of rules to function properly.

Note: State modules define the available routines in sls files. If calling an execution module directly is desired, take a look at the *module* state.

51.5.3 Auth

The auth module system allows for external authentication routines to be easily added into Salt. The *auth* function needs to be implemented to satisfy the requirements of an auth module. Use the `pam` module as an example.

51.5.4 Fileserver

The fileserver module system is used to create fileserver backends used by the Salt Master. These modules need to implement the functions used in the fileserver subsystem. Use the `gitfs` module as an example.

51.5.5 Grains

Grain modules define extra routines to populate grains data. All defined public functions will be executed and MUST return a Python dict object. The dict keys will be added to the grains made available to the minion.

51.5.6 Output

The output modules supply the outputter system with routines to display data in the terminal. These modules are very simple and only require the *output* function to execute. The default system outputter is the `nested` module.

51.5.7 Pillar

Used to define optional external pillar systems. The pillar generated via the filesystem pillar is passed into external pillars. This is commonly used as a bridge to database data for pillar, but is also the backend to the libvirt state used to generate and sign libvirt certificates on the fly.

51.5.8 Renderers

Renderers are the system used to render sls files into salt highdata for the state compiler. They can be as simple as the `py` renderer and as complex as `stateconf` and `pydsl`.

51.5.9 Returners

Returners are used to send data from minions to external sources, commonly databases. A full returner will implement all routines to be supported as an external job cache. Use the `redis` returner as an example.

51.5.10 Runners

Runners are purely master-side execution sequences. These range from simple reporting to orchestration engines like the overstate.

51.5.11 Tops

Tops modules are used to convert external data sources into top file data for the state system.

51.5.12 Wheel

The wheel system is used to manage master side management routines. These routines are primarily intended for the API to enable master configuration.

51.6 Package Providers

This page contains guidelines for writing package providers.

51.6.1 Package Functions

One of the most important features of Salt is package management. There is no shortage of package managers, so in the interest of providing a consistent experience in `pkg` states, there are certain functions that should be present in a package provider. Note that these are subject to change as new features are added or existing features are enhanced.

list_pkgs

This function should declare an empty dict, and then add packages to it by calling `pkg_resource.add_pkg`, like so:

```
__salt__['pkg_resource.add_pkg'](ret, name, version)
```

The last thing that should be done before returning is to execute `pkg_resource.sort_pkglist`. This function does not presently do anything to the return dict, but will be used in future versions of Salt.

```
__salt__['pkg_resource.sort_pkglist'](ret)
```

`list_pkgs` returns a dictionary of installed packages, with the keys being the package names and the values being the version installed. Example return data:

```
{'foo': '1.2.3-4',
 'bar': '5.6.7-8'}
```

latest_version

Accepts an arbitrary number of arguments. Each argument is a package name. The return value for a package will be an empty string if the package is not found or if the package is up-to-date. The only case in which a non-empty string is returned is if the package is available for new installation (i.e. not already installed) or if there is an upgrade available.

If only one argument was passed, this function return a string, otherwise a dict of name/version pairs is returned.

This function must also accept `**kwargs`, in order to receive the `fromrepo` and `repo` keyword arguments from pkg states. Where supported, these arguments should be used to find the install/upgrade candidate in the specified repository. The `fromrepo` kwarg takes precedence over `repo`, so if both of those kwargs are present, the repository specified in `fromrepo` should be used. However, if `repo` is used instead of `fromrepo`, it should still work, to preserve backwards compatibility with older versions of Salt.

version

Like `latest_version`, accepts an arbitrary number of arguments and returns a string if a single package name was passed, or a dict of name/value pairs if more than one was passed. The only difference is that the return values are the currently-installed versions of whatever packages are passed. If the package is not installed, an empty string is returned for that package.

upgrade_available

Deprecated and destined to be removed. For now, should just do the following:

```
return __salt__['pkg.latest_version'](name) != ''
```

install

The following arguments are required and should default to `None`:

1. name (for single-package pkg states)
2. pkgs (for multiple-package pkg states)
3. sources (for binary package file installation)

The first thing that this function should do is call `pkg_resource.parse_targets` (see below). This function will convert the SLS input into a more easily parsed data structure. `pkg_resource.parse_targets` may need to be modified to support your new package provider, as it does things like parsing package metadata which cannot be done for every package management system.

```
pkg_params, pkg_type = __salt__['pkg_resource.parse_targets'](name,
                                                              pkgs,
                                                              sources)
```

Two values will be returned to the **install** function. The first of them will be a dictionary. The keys of this dictionary will be package names, though the values will differ depending on what kind of installation is being done:

- If **name** was provided (and **pkgs** was not), then there will be a single key in the dictionary, and its value will be `None`. Once the data has been returned, if the **version** keyword argument was provided, then it should replace the `None` value in the dictionary.
- If **pkgs** was provided, then **name** is ignored, and the dictionary will contain one entry for each package in the **pkgs** list. The values in the dictionary will be `None` if a version was not specified for the package, and the desired version if specified. See the **Multiple Package Installation Options** section of the `pkg.installed` state for more info.
- If **sources** was provided, then **name** is ignored, and the dictionary values will be the path/URI for the package.

The second return value will be a string with two possible values: `repository` or `file`. The **install** function can use this value (if necessary) to build the proper command to install the targeted package(s).

Both before and after the installing the target(s), you should run **list_pkgs** to obtain a list of the installed packages. You should then return the output of `salt.utils.compare_dicts()`

```
return salt.utils.compare_dicts(old, new)
```

remove

Removes the passed package and return a list of the packages removed.

51.6.2 Package Repo Functions

There are some functions provided by `pkg` which are specific to package repositories, and not to packages themselves. When writing modules for new package managers, these functions should be made available as stated below, in order to provide compatibility with the `pkgrepo` state.

All repo functions should accept a basedir option, which defines which directory repository configuration should be found in. The default for this is dictated by the repo manager that is being used, and rarely needs to be changed.

```
basedir = '/etc/yum.repos.d'
__salt__['pkg.list_repos'](basedir)
```

list_repos

Lists the repositories that are currently configured on this system.

```
__salt__['pkg.list_repos']()
```

Returns a dictionary, in the following format:

```
{'reponame': 'config_key_1': 'config value 1',
             'config_key_2': 'config value 2',
             'config_key_3': ['list item 1 (when appropriate)',
                              'list item 2 (when appropriate)]}
```

get_repo

Displays all local configuration for a specific repository.

```
__salt__['pkg.get_repo'](repo='myrepo')
```

The information is formatted in much the same way as list_repos, but is specific to only one repo.

```
{'config_key_1': 'config value 1',
 'config_key_2': 'config value 2',
 'config_key_3': ['list item 1 (when appropriate)',
                  'list item 2 (when appropriate)]}
```

del_repo

Removes the local configuration for a specific repository. Requires a *repo* argument, which must match the locally configured name. This function returns a string, which informs the user as to whether or not the operation was a success.

```
__salt__['pkg.del_repo'](repo='myrepo')
```

mod_repo

Modify the local configuration for one or more option for a configured repo. This is also the way to create new repository configuration on the local system; if a repo is specified which does not yet exist, it will be created.

The options specified for this function are specific to the system; please refer to the documentation for your specific repo manager for specifics.

```
__salt__['pkg.mod_repo'](repo='myrepo', url='http://myurl.com/repo')
```

51.6.3 Low-Package Functions

In general, the standard package functions as describes above will meet your needs. These functions use the system's native repo manager (for instance, yum or the apt tools). In most cases, the repo manager is actually separate from the package manager. For instance, yum is usually a front-end for rpm, and apt is usually a front-end for dpkg. When possible, the package functions that use those package managers directly should do so through the low package functions.

It is normal and sane for `pkg` to make calls to `lowpkgs`, but `lowpkg` must never make calls to `pkg`. This is affects functions which are required by both `pkg` and `lowpkg`, but the technique in `pkg` is more performant than what is available to `lowpkg`. When this is the case, the `lowpkg` function that requires that technique must still use the `lowpkg` version.

list_pkgs

Returns a dict of packages installed, including the package name and version. Can accept a list of packages; if none are specified, then all installed packages will be listed.

```
installed = __salt__['lowpkg.list_pkgs']('foo', 'bar')
```

Example output:

```
{'foo': '1.2.3-4',
 'bar': '5.6.7-8'}
```

verify

Many (but not all) package management systems provide a way to verify that the files installed by the package manager have or have not changed. This function accepts a list of packages; if none are specified, all packages will be included.

```
installed = __salt__['lowpkg.verify']('httpd')
```

Example output:

```
{'/etc/httpd/conf/httpd.conf': {'mismatch': ['size', 'md5sum', 'mtime'],
                                'type': 'config'}}
```

file_list

Lists all of the files installed by all packages specified. If not packages are specified, then all files for all known packages are returned.

```
installed = __salt__['lowpkg.file_list']('httpd', 'apache')
```

This function does not return which files belong to which packages; all files are returned as one giant list (hence the *file_list* function name. However, This information is still returned inside of a dict, so that it can provide any errors to the user in a sane manner.

```
{'errors': ['package apache is not installed'],
  'files': ['/etc/httpd',
            '/etc/httpd/conf',
            '/etc/httpd/conf.d',
            '...SNIP...']}
```

file_dict

Lists all of the files installed by all packages specified. If not packages are specified, then all files for all known packages are returned.

```
installed = __salt__['lowpkg.file_dict']('httpd', 'apache', 'kernel')
```

Unlike *file_list*, this function will break down which files belong to which packages. It will also return errors in the same manner as *file_list*.

```
{'errors': ['package apache is not installed'],
 'packages': {'httpd': ['/etc/httpd',
                        '/etc/httpd/conf',
                        '...SNIP...'],
              'kernel': ['/boot/.vmlinuz-2.6.32-279.el6.x86_64.hmac',
                         '/boot/System.map-2.6.32-279.el6.x86_64',
                         '...SNIP...']}}
```

CHAPTER 52

Translating Documentation

If you wish to help translate the Salt documentation to your language, please head over to the Transifex website and signup for an account.

Once registered, head over to the Salt Translation Project, and either click on **Request Language** if you can't find yours, or, select the language for which you wish to contribute and click **Join Team**.

Transifex provides some useful reading resources on their support domain, namely, some useful articles directed to translators.

52.1 Building A Localized Version of the Documentation

While you're working on your translation on Transifex, you might want to have a look at how it's rendering.

52.1.1 Install The Transifex Client

To interact with the Transifex web service you will need to install the transifex-client:

```
pip install transifex-client
```

52.1.2 Configure The Transifex Client

Once installed, you will need to set it up on your computer. We created a script to help you with that:

```
.scripts/setup-transifex-config
```

52.1.3 Download Remote Translations

There's a little script which simplifies the download process of the translations(which isn't that complicated in the first place). So, let's assume you're translating `pt_PT`, Portuguese(Portugal). To download the translations, execute from the `doc/` directory of your Salt checkout:

```
make download-translations SPHINXLANG=pt_PT
```

To download `pt_PT`, Portuguese(Portugal) and `nl`, Dutch, you can use the helper script directly:

```
.scripts/download-translation-catalog pt_PT nl
```

52.1.4 Build Localized Documentation

After the download process finishes, which might take a while, the next step is to build a localized version of the documentation. Following the `pt_PT` example above:

```
make html SPHINXLANG=pt_PT
```

52.1.5 View Localized Documentation

Open your browser, point it to the local documentation path and check the localized output you've just build.

CHAPTER 53

Community Projects That Use Salt

Below is a list of repositories that show real world Salt applications that you can use to get started. Please note that these projects do not adhere to any standards and express a wide variety of ideas and opinions on how an action can be completed with Salt.

https://github.com/terminalmage/djangocon2013-sls

https://github.com/jesusaurus/hpcs-salt-state

https://github.com/gravyboat/hungryadmin-sls

Salt Proxy Minion Documentation

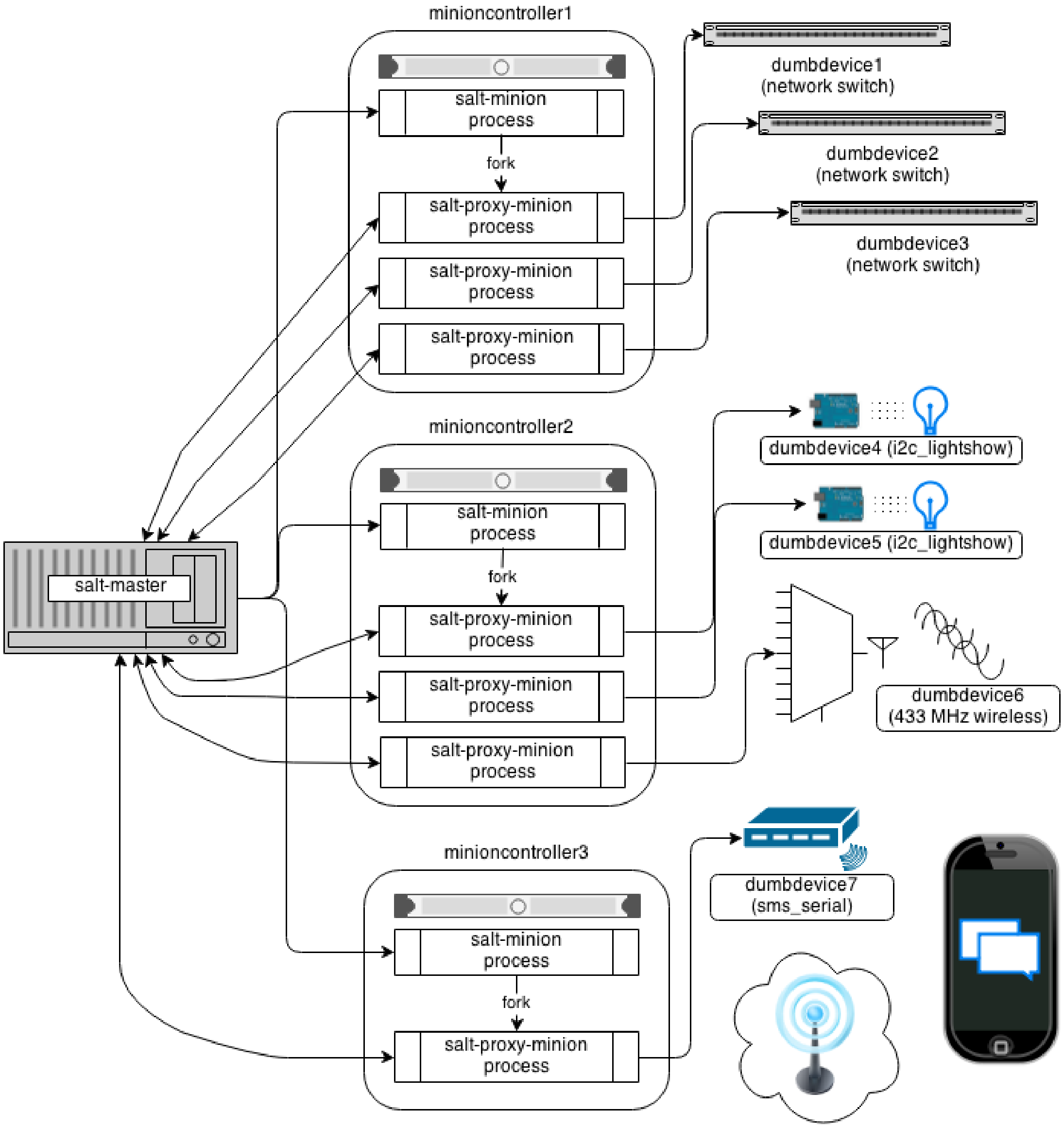
Proxy minions are a developing Salt feature that enables controlling devices that, for whatever reason, cannot run a standard salt-minion. Examples include network gear that has an API but runs a proprietary OS, devices with limited CPU or memory, or devices that could run a minion, but for security reasons, will not.

Proxy minions are not an "out of the box" feature. Because there are an infinite number of controllable devices, you will most likely have to write the interface yourself. Fortunately, this is only as difficult as the actual interface to the proxied device. Devices that have an existing Python module (PyUSB for example) would be relatively simple to interface. Code to control a device that has an HTML REST-based interface should be easy. Code to control your typical housecat would be excellent source material for a PhD thesis.

Salt proxy-minions provide the 'plumbing' that allows device enumeration and discovery, control, status, remote execution, and state management.

54.1 Getting Started

The following diagram may be helpful in understanding the structure of a Salt installation that includes proxy-minions:


minioncontroller1
salt-minion process
fork
salt-proxy-minion process
salt-proxy-minion process
salt-proxy-minion process
dumbdevice1 (network switch)
dumbdevice2 (network switch)
dumbdevice3 (network switch)
minioncontroller2
salt-minion process
fork
salt-proxy-minion process
salt-proxy-minion process
salt-proxy-minion process
dumbdevice4 (i2c_lightshow)
dumbdevice5 (i2c_lightshow)
dumbdevice6 (433 MHz wireless)
salt-master
minioncontroller3
salt-minion process
fork
salt-proxy-minion process
dumbdevice7 (sms_serial)

The key thing to remember is the left-most section of the diagram. Salt's nature is to have a minion connect to a master, then the master may control the minion. However, for proxy minions, the target device cannot run a minion, and thus must rely on a separate minion to fire up the proxy-minion and make the initial and persistent connection.

After the proxy minion is started and initiates its connection to the 'dumb' device, it connects back to the salt-master and ceases to be affiliated in any way with the minion that started it.

To create support for a proxied device one needs to create four things:

1. The proxytype connection class (located in salt/proxy).
2. The grains support code (located in salt/grains).
3. **'Salt modules'**_ specific to the controlled device.
4. Salt states specific to the controlled device.

54.1.1 Configuration parameters on the master

Proxy minions require no configuration parameters in /etc/salt/master.

Salt's Pillar system is ideally suited for configuring proxy-minions. Proxies can either be designated via a pillar file in pillar_roots, or through an external pillar. External pillars afford the opportunity for interfacing with a configuration management system, database, or other knowledgeable system that that may already contain all the details of proxy targets. To use static files in pillar_roots, pattern your files after the following examples, which are based on the diagram above:

`/srv/salt/pillar/top.sls`

```
base:
  minioncontroller1:
    - networkswitches
  minioncontroller2:
    - reallydumbdevices
  minioncontroller3:
    - smsgateway
```

`/srv/salt/pillar/networkswitches.sls`

```
proxy:
  dumbdevice1:
    proxytype: networkswitch
    host: 172.23.23.5
    username: root
    passwd: letmein
  dumbdevice2:
    proxytype: networkswitch
    host: 172.23.23.6
    username: root
    passwd: letmein
  dumbdevice3:
    proxytype: networkswitch
    host: 172.23.23.7
    username: root
    passwd: letmein
```

`/srv/salt/pillar/reallydumbdevices.sls`

```
proxy:
  dumbdevice4:
    proxytype: i2c_lightshow
    i2c_address: 1
  dumbdevice5:
    proxytype: i2c_lightshow
    i2c_address: 2
  dumbdevice6:
    proxytype: 433mhz_wireless
```

`/srv/salt/pillar/smsgateway.sls`

```
proxy:
  minioncontroller3:
    dumbdevice7:
      proxytype: sms_serial
      deventry: /dev/tty04
```

Note the contents of each minioncontroller key may differ widely based on the type of device that the proxy-minion is managing.

In the above example

- dumbdevices 1, 2, and 3 are network switches that have a management interface available at a particular IP address.
- dumbdevices 4 and 5 are very low-level devices controlled over an i2c bus. In this case the devices are physically connected to machine 'minioncontroller2', and are addressable on the i2c bus at their respective i2c addresses.
- dumbdevice6 is a 433 MHz wireless transmitter, also physically connected to minioncontroller2
- dumbdevice7 is an SMS gateway connected to machine minioncontroller3 via a serial port.

Because of the way pillar works, each of the salt-minions that fork off the proxy minions will only see the keys specific to the proxies it will be handling. In other words, from the above example, only minioncontroller1 will see the connection information for dumbdevices 1, 2, and 3. Minioncontroller2 will see configuration data for dumbdevices 4, 5, and 6, and minioncontroller3 will be privy to dumbdevice7.

Also, in general, proxy-minions are lightweight, so the machines that run them could conceivably control a large number of devices. The example above is just to illustrate that it is possible for the proxy services to be spread across many machines if necessary, or intentionally run on machines that need to control devices because of some physical interface (e.g. i2c and serial above). Another reason to divide proxy services might be security. In more secure environments only certain machines may have a network path to certain devices.

Now our salt-minions know if they are supposed to spawn a proxy-minion process to control a particular device. That proxy-minion process will initiate a connection back to the master to enable control.

54.1.2 Proxytypes

A proxytype is a Python class called 'Proxyconn' that encapsulates all the code necessary to interface with a device. Proxytypes are located inside the salt.proxy module. At a minimum a proxytype object must implement the following methods:

`proxytype(self)`: Returns a string with the name of the proxy type.

`proxyconn(self, **kwargs)`: Provides the primary way to connect and communicate with the device. Some proxyconns instantiate a particular object that opens a network connection to a device and leaves the connection open for communication. Others simply abstract a serial connection or even implement endpoints to communicate via REST over HTTP.

`id(self, opts)`: Returns a unique, unchanging id for the controlled device. This is the "name" of the device, and is used by the salt-master for targeting and key authentication.

Optionally, the class may define a `shutdown(self, opts)` method if the controlled device should be informed when the minion goes away cleanly.

It is highly recommended that the `test.ping` execution module also be defined for a proxytype. The code for `ping` should contact the controlled device and make sure it is really available.

Here is an example proxytype used to interface to Juniper Networks devices that run the Junos operating system. Note the additional library requirements–most of the "hard part" of talking to these devices is handled by the jnpr.junos, jnpr.junos.utils and jnpr.junos.cfg modules.

```python
# Import python libs
import logging
import os

import jnpr.junos
import jnpr.junos.utils
import jnpr.junos.cfg
HAS_JUNOS = True
```

```
class Proxyconn(object):


    def __init__(self, details):
        self.conn = jnpr.junos.Device(user=details['username'], host=details['host'], password=detai
        self.conn.open()
        self.conn.bind(cu=jnpr.junos.cfg.Resource)


    def proxytype(self):
        return 'junos'


    def id(self, opts):
        return self.conn.facts['hostname']


    def ping(self):
        return self.conn.connected


    def shutdown(self, opts):

        print('Proxy module {} shutting down!!'.format(opts['id']))
        try:
            self.conn.close()
        except Exception:
            pass
```

Grains are data about minions. Most proxied devices will have a paltry amount of data as compared to a typical Linux server. Because proxy-minions are started by a regular minion, they inherit a sizeable number of grain settings which can be useful, especially when targeting (PYTHONPATH, for example).

All proxy minions set a grain called 'proxy'. If it is present, you know the minion is controlling another device. To add more grains to your proxy minion for a particular device, create a file in salt/grains named [proxytype].py and place inside it the different functions that need to be run to collect the data you are interested in. Here's an example:

54.2 The __proxyenabled__ directive

Salt states and execution modules, by and large, cannot "automatically" work with proxied devices. Execution modules like `pkg` or `sqlite3` have no meaning on a network switch or a housecat. For a state/execution module to be available to a proxy-minion, the `__proxyenabled__` variable must be defined in the module as an array containing the names of all the proxytypes that this module can support. The array can contain the special value `*` to indicate that the module supports all proxies.

If no `__proxyenabled__` variable is defined, then by default, the state/execution module is unavailable to any proxy.

Here is an excerpt from a module that was modified to support proxy-minions:

```
def ping():

    if 'proxyobject' in __opts__:
        if 'ping' in __opts__['proxyobject'].__attr__():
            return __opts['proxyobject'].ping()
        else:
            return False
    else:
        return True
```

And then in salt.proxy.junos we find

```
def ping(self):

   return self.connected
```

The Junos API layer lacks the ability to do a traditional ‘ping’, so the example simply checks the connection object field that indicates if the ssh connection was successfully made to the device.

CHAPTER 55

Logging

The salt project tries to get the logging to work for you and help us solve any issues you might find along the way.

If you want to get some more information on the nitty-gritty of salt's logging system, please head over to the *logging development document*, if all you're after is salt's logging configurations, please continue reading.

55.1 Available Configuration Settings

55.1.1 `log_file`

The log records can be sent to a regular file, local path name, or network location. Remote logging works best when configured to use rsyslogd(8) (e.g.: `file:///dev/log`), with rsyslogd(8) configured for network logging. The format for remote addresses is: `<file|udp|tcp>://<host|socketpath>:<port-if-required>/<log-facility>`.

Default: Dependent of the binary being executed, for example, for `salt-master`, `/var/log/salt/master`.

Examples:

```
log_file: /var/log/salt/master
```

```
log_file: /var/log/salt/minion
```

```
log_file: file:///dev/log
```

```
log_file: udp://loghost:10514
```

55.1.2 `log_level`

Default: `warning`

The level of log record messages to send to the console. One of `all`, `garbage`, `trace`, `debug`, `info`, `warning`, `error`, `critical`, `quiet`.

```
log_level: warning
```

55.1.3 `log_level_logfile`

Default: `warning`

The level of messages to send to the log file. One of `all`, `garbage`, `trace`, `debug`, `info`, `warning`, `error`, `critical`, `quiet`.

```
log_level_logfile: warning
```

55.1.4 `log_datefmt`

Default: `%H:%M:%S`

The date and time format used in console log messages. Allowed date/time formatting can be seen on `time.strftime`.

```
log_datefmt: '%H:%M:%S'
```

55.1.5 `log_datefmt_logfile`

Default: `%Y-%m-%d %H:%M:%S`

The date and time format used in log file messages. Allowed date/time formatting can be seen on `time.strftime`.

```
log_datefmt_logfile: '%Y-%m-%d %H:%M:%S'
```

55.1.6 `log_fmt_console`

Default: `[%(levelname)-8s] %(message)s`

The format of the console logging messages. Allowed formatting options can be seen on the *LogRecord attributes*.

```
log_fmt_console: '[%(levelname)-8s] %(message)s'
```

55.1.7 `log_fmt_logfile`

Default: `%(asctime)s,%(msecs)03.0f [%(name)-17s][%(levelname)-8s] %(message)s`

The format of the log file logging messages. Allowed formatting options can be seen on the *LogRecord attributes*.

```
log_fmt_logfile: '%(asctime)s,%(msecs)03.0f [%(name)-17s][%(levelname)-8s] %(message)s'
```

55.1.8 `log_granular_levels`

Default: `{}`

This can be used to control logging levels more specifically. The example sets the main salt library at the 'warning' level, but sets `salt.modules` to log at the `debug` level:

```
log_granular_levels:
  'salt': 'warning',
  'salt.modules': 'debug'
```

55.1.9 External Logging Handlers

Besides the internal logging handlers used by salt, there are some external which can be used, see the *external logging handlers* document.

CHAPTER 56

External Logging Handlers

`logstash_mod`	Logstash Logging Handler
`sentry_mod`	Sentry Logging Handler

56.1 Logstash Logging Handler

New in version 0.17.0.

This module provides some Logstash logging handlers.

56.1.1 UDP Logging Handler

In order to setup the datagram handler for Logstash, please define on the salt configuration file:

```
logstash_udp_handler:
  host: 127.0.0.1
  port: 9999
```

On the Logstash configuration file you need something like:

```
input {
  udp {
    type => "udp-type"
    format => "json_event"
  }
}
```

Please read the UDP input configuration page for additional information.

56.1.2 ZeroMQ Logging Handler

In order to setup the ZMQ handler for Logstash, please define on the salt configuration file:

```
logstash_zmq_handler:
  address: tcp://127.0.0.1:2021
```

On the Logstash configuration file you need something like:

```
input {
  zeromq {
    type => "zeromq-type"
    mode => "server"
    topology => "pubsub"
    address => "tcp://0.0.0.0:2021"
    charset => "UTF-8"
    format => "json_event"
  }
}
```

Please read the ZeroMQ input configuration page for additional information.

Important Logstash Setting

One of the most important settings that you should not forget on your Logstash configuration file regarding these logging handlers is `format`. Both the *UDP* and *ZeroMQ* inputs need to have `format` as `json_event` which is what we send over the wire.

Log Level

Both the `logstash_udp_handler` and the `logstash_zmq_handler` configuration sections accept an additional setting `log_level`. If not set, the logging level used will be the one defined for `log_level` in the global configuration file section.

HWM

The high water mark for the ZMQ socket setting. Only applicable for the `logstash_zmq_handler`.

Inspiration

This work was inspired in pylogstash, python-logstash, canary and the PyZMQ logging handler.

56.2 Sentry Logging Handler

New in version 0.17.0.

This module provides a Sentry logging handler.

Note

The Raven library needs to be installed on the system for this logging handler to be available.

Configuring the python Sentry client, Raven, should be done under the `sentry_handler` configuration key. At the bare minimum, you need to define the DSN. As an example:

```
sentry_handler:
  dsn: https://pub-key:secret-key@app.getsentry.com/app-id
```

More complex configurations can be achieved, for example:

```
sentry_handler:
  servers:
    - https://sentry.example.com
    - http://192.168.1.1
  project: app-id
  public_key: deadbeefdeadbeefdeadbeefdeadbeef
  secret_key: beefdeadbeefdeadbeefdeadbeefdead
```

All the client configuration keys are supported, please see the Raven client documentation.

The default logging level for the sentry handler is `ERROR`. If you wish to define a different one, define `log_level` under the `sentry_handler` configuration key:

```
sentry_handler:
  dsn: https://pub-key:secret-key@app.getsentry.com/app-id
  log_level: warning
```

The available log levels are those also available for the salt `cli` tools and configuration; `salt --help` should give you the required information.

56.2.1 Threaded Transports

Raven's documents rightly suggest using its threaded transport for critical applications. However, don't forget that if you start having troubles with Salt after enabling the threaded transport, please try switching to a non-threaded transport to see if that fixes your problem.

CHAPTER 57

Logstash Logging Handler

New in version 0.17.0.

This module provides some Logstash logging handlers.

57.1 UDP Logging Handler

In order to setup the datagram handler for Logstash, please define on the salt configuration file:

```
logstash_udp_handler:
  host: 127.0.0.1
  port: 9999
```

On the Logstash configuration file you need something like:

```
input {
  udp {
    type => "udp-type"
    format => "json_event"
  }
}
```

Please read the UDP input configuration page for additional information.

57.2 ZeroMQ Logging Handler

In order to setup the ZMQ handler for Logstash, please define on the salt configuration file:

```
logstash_zmq_handler:
  address: tcp://127.0.0.1:2021
```

On the Logstash configuration file you need something like:

```
input {
  zeromq {
    type => "zeromq-type"
    mode => "server"
    topology => "pubsub"
    address => "tcp://0.0.0.0:2021"
    charset => "UTF-8"
    format => "json_event"
  }
}
```

Please read the ZeroMQ input configuration page for additional information.

Important Logstash Setting

One of the most important settings that you should not forget on your Logstash configuration file regarding these logging handlers is `format`. Both the *UDP* and *ZeroMQ* inputs need to have `format` as `json_event` which is what we send over the wire.

57.2.1 Log Level

Both the `logstash_udp_handler` and the `logstash_zmq_handler` configuration sections accept an additional setting `log_level`. If not set, the logging level used will be the one defined for `log_level` in the global configuration file section.

57.2.2 HWM

The high water mark for the ZMQ socket setting. Only applicable for the `logstash_zmq_handler`.

Inspiration

This work was inspired in pylogstash, python-logstash, canary and the PyZMQ logging handler.

CHAPTER 58

Sentry Logging Handler

New in version 0.17.0.

This module provides a Sentry logging handler.

Note

The Raven library needs to be installed on the system for this logging handler to be available.

Configuring the python Sentry client, Raven, should be done under the `sentry_handler` configuration key. At the bare minimum, you need to define the DSN. As an example:

```
sentry_handler:
  dsn: https://pub-key:secret-key@app.getsentry.com/app-id
```

More complex configurations can be achieved, for example:

```
sentry_handler:
  servers:
    - https://sentry.example.com
    - http://192.168.1.1
  project: app-id
  public_key: deadbeefdeadbeefdeadbeefdeadbeef
  secret_key: beefdeadbeefdeadbeefdeadbeefdead
```

All the client configuration keys are supported, please see the Raven client documentation.

The default logging level for the sentry handler is `ERROR`. If you wish to define a different one, define `log_level` under the `sentry_handler` configuration key:

```
sentry_handler:
  dsn: https://pub-key:secret-key@app.getsentry.com/app-id
  log_level: warning
```

The available log levels are those also available for the salt `cli` tools and configuration; `salt --help` should give you the required information.

58.1 Threaded Transports

Raven's documents rightly suggest using its threaded transport for critical applications. However, don't forget that if you start having troubles with Salt after enabling the threaded transport, please try switching to a non-threaded transport to see if that fixes your problem.

CHAPTER 59

Introduction to Extending Salt

Salt is made to be used, and made to be extended. The primary goal of Salt is to provide a foundation which can be used to solve problems without assuming what those problems might be.

One of the greatest benefit of developing Salt has been the vast array of ways in which people have wanted to use it, while the original intention was as a communication layer for a cloud controller Salt has been extended to facilitate so much more.

59.1 Client API

The primary interface used to extend Salt, is to simply use it. Salt executions can be called via the Salt client API, making programming master side solutions with Salt is easy.

See also:

Python client API

59.2 Adding Loadable Plugins

Salt is comprised of a core platform that loads many types of easy to write plugins. The idea is to enable all of the breaking points in the Salt processes to have a point of pluggable interaction. This means that all of the main features of Salt can be extended, modified or used.

The breaking points and helping interfaces span from convenience master side executions to manipulating the flow of how data is handled by Salt.

59.2.1 Minion Execution Modules

The minion execution modules or just `modules` are the core to what Salt is and does. These modules are found in:

https://github.com/saltstack/salt/blob/develop/salt/modules

These modules are what is called by the Salt command line and the salt client API. Adding modules is done by simply adding additional Python modules to the *modules* directory and restarting the minion.

59.2.2 Grains

Salt grains, or "grains of truth" are bits of static information that are generated when the minion starts. This information is useful when determining what package manager to default to, or where certain configuration files are stored on the minion.

The Salt grains are the interface used for auto detection and dynamic assignment of execution modules and types to specific Salt minions.

The code used to generate the Salt grains can be found here:

https://github.com/saltstack/salt/blob/develop/salt/grains

59.2.3 States

Salt supports state enforcement, this makes Salt a high speed and very efficient solution for system configuration management.

States can be easily added to Salt by dropping a new state module in:

https://github.com/saltstack/salt/blob/develop/salt/states

59.2.4 Renderers

Salt states are controlled by simple data structures, these structures can be abstracted in a number of ways. While the default is to be in a YAML file wrapped in a jinja template, any abstraction can be used. This means that any format that can be dreamed is possible, so long as a renderer is written for it.

The existing renderers can be found here:

https://github.com/saltstack/salt/blob/develop/salt/renderers

59.2.5 Returners

The Salt commands all produce a return value, that return value is sent to the Salt master by default, but it can be sent anywhere. The returner interface makes it programmatically possible for the information to be sent to anything from an SQL or NoSQL database, to a custom application made to use Salt.

The existing returners can be found here:

https://github.com/saltstack/salt/blob/develop/salt/returners

59.2.6 Runners

Sometimes a certain application can be made to execute and run from the existing Salt command line. This is where the Salt runners come into play. The Salt Runners what is called by the Salt-run command and are meant to act as a generic interface for encapsulating master side executions.

Existing Salt runners are located here:

https://github.com/saltstack/salt/blob/develop/salt/runners

CHAPTER 60

Execution Modules

Salt execution modules are the functions called by the **salt** command.

Note: Salt execution modules are different from state modules and cannot be called directly within state files. You must use the *module* state module to execute execution modules within state runs.

See also:

Full list of builtin modules

Salt ships with many modules that cover a wide variety of tasks.

60.1 Modules Are Easy to Write!

Salt modules are amazingly simple to write. Just write a regular Python module or a regular Cython module and place it a directory called `_modules/` within the `file_roots` specified by the master config file, and they will be synced to the minions when `state.highstate` is run, or by executing the `saltutil.sync_modules` or `saltutil.sync_all` functions.

Any custom modules which have been synced to a minion, that are named the same as one of Salt's default set of modules, will take the place of the default module with the same name. Note that a module's default name is its filename (i.e. `foo.py` becomes module `foo`), but that its name can be overridden by using a *__virtual__ function*.

Since Salt modules are just Python/Cython modules, there are no restraints on what you can put inside of a Salt module. If a Salt module has errors and cannot be imported, the Salt minion will continue to load without issue and the module with errors will simply be omitted.

If adding a Cython module the file must be named `<modulename>.pyx` so that the loader knows that the module needs to be imported as a Cython module. The compilation of the Cython module is automatic and happens when the minion starts, so only the `*.pyx` file is required.

60.2 Cross Calling Modules

All of the Salt modules are available to each other, and can be "cross called". This means that, when creating a module, functions in modules that already exist can be called.

The variable `__salt__` is packed into the modules after they are loaded into the Salt minion. This variable is a *Python dictionary* of all of the Salt functions, laid out in the same way that they are made available to the Salt command.

Salt modules can be cross called by accessing the value in the `__salt__` dict:

```
def foo(bar):
    return __salt__['cmd.run'](bar)
```

This code will call the Salt cmd module's `run` function and pass the argument `bar`.

60.3 Preloaded Modules Data

When interacting with modules often it is nice to be able to read information dynamically about the minion, or load in configuration parameters for a module. Salt allows for different types of data to be loaded into the modules by the minion, as of this writing Salt loads information gathered from the Salt Grains system and from the minion configuration file.

60.3.1 Grains Data

The Salt minion detects information about the system when started. This allows for modules to be written dynamically with respect to the underlying hardware and operating system. This information is referred to as Salt Grains, or "grains of salt". The Grains system was introduced to replace Facter, since relying on a Ruby application from a Python application was both slow and inefficient. Grains support replaces Facter in all Salt releases after 0.8

The values detected by the Salt Grains on the minion are available in a *dict* named `__grains__` and can be accessed from within callable objects in the Python modules.

To see the contents of the grains dict for a given system in your deployment run the `grains.items()` function:

```
salt 'hostname' grains.items
```

To use the `__grains__` dict simply call it as a Python dict from within your code, an excellent example is available in the Grains module: `salt.modules.grains`.

60.3.2 Module Configuration

Since parameters for configuring a module may be desired, Salt allows for configuration information stored in the main minion config file to be passed to the modules.

Since the minion configuration file is a YAML document, arbitrary configuration data can be passed in the minion config that is read by the modules. It is **strongly** recommended that the values passed in the configuration file match the module. This means that a value intended for the `test` module should be named `test.<value>`.

Configuration also requires that default configuration parameters need to be loaded as well. This can be done simply by adding the `__opts__` dict to the top level of the module.

The test module contains usage of the module configuration, and the default configuration file for the minion contains the information and format used to pass data to the modules. `salt.modules.test`, `conf/minion`.

60.4 Printout Configuration

Since module functions can return different data, and the way the data is printed can greatly change the presentation, Salt has a printout configuration.

When writing a module the `__outputter__` dict can be declared in the module. The `__outputter__` dict contains a mapping of function name to Salt Outputter.

```
__outputter__ = {
                'run': 'txt'
                }
```

This will ensure that the text outputter is used.

60.5 Virtual Modules

Sometimes a module should be presented in a generic way. A good example of this can be found in the package manager modules. The package manager changes from one operating system to another, but the Salt module that interfaces with the package manager can be presented in a generic way.

The Salt modules for package managers all contain a `__virtual__` function which is called to define what systems the module should be loaded on.

The `__virtual__` function is used to return either a *string* or `False`. If False is returned then the module is not loaded, if a string is returned then the module is loaded with the name of the string.

This means that the package manager modules can be presented as the `pkg` module regardless of what the actual module is named.

The package manager modules are the best example of using the `__virtual__` function. Some examples:

- pacman.py
- yumpkg.py
- apt.py

60.6 Documentation

Salt modules are self documenting, the `sys.doc()` function will return the documentation for all available modules:

```
salt '*' sys.doc
```

This function simply prints out the docstrings found in the modules; when writing Salt modules, please follow the formatting conventions for docstrings as they appear in the other modules.

60.6.1 Adding Documentation to Salt Modules

Since life is much better with documentation, it is strongly suggested that all Salt modules have documentation added. Any Salt modules submitted for inclusion in the main distribution of Salt will be required to have documentation.

Documenting Salt modules is easy! Just add a Python docstring to the function.

```
def spam(eggs):
    '''
    A function to make some spam with eggs!

    CLI Example::

        salt '*' test.spam eggs
    '''
    return eggs
```

Now when the sys.doc call is executed the docstring will be cleanly returned to the calling terminal.

60.6.2 Add Module metadata

Add information about the module using the following field lists:

```
:maintainer:    Thomas Hatch <thatch@saltstack.com, Seth House <shouse@saltstack.com>
:maturity:      new
:depends:       python-mysqldb
:platform:      all
```

The maintainer field is a comma-delimited list of developers who help maintain this module.

The maturity field indicates the level of quality and testing for this module. Standard labels will be determined.

The depends field is a comma-delimited list of modules that this module depends on.

The platform field is a comma-delimited list of platforms that this module is known to run on.

60.7 How Functions are Read

In Salt, Python callable objects contained within a module are made available to the Salt minion for use. The only exception to this rule is a callable object with a name starting with an underscore _.

60.7.1 Objects Loaded Into the Salt Minion

```
def foo(bar):
    return bar

class baz:
    def __init__(self, quo):
        pass
```

60.7.2 Objects NOT Loaded into the Salt Minion

```
def _foobar(baz): # Preceded with an _
    return baz

cheese = {} # Not a callable Python object
```

Note: Some callable names also end with an underscore _, to avoid keyword clashes with Python keywords. When using Salt modules, or state modules, with these in them the trailing underscore should be omitted.

60.8 Useful Decorators for Modules

Sometimes when writing modules for large scale deployments you run into some small things that end up severely complicating the code. To alleviate some of this pain Salt has some useful decorators for use within modules!

60.8.1 Depends Decorator

When writing custom modules there are many times where some of the module will work on all hosts, but some functions require (for example) a service to be installed. Instead of trying to wrap much of the code in large try/except blocks you can use a simple decorator to do this. If the dependencies passed to the decorator don't exist, then the salt minion will remove those functions from the module on that host. If a "fallback_function" is defined, it will replace the function instead of removing it

```
from salt.utils.decorators import depends
try:
    import dependency_that_sometimes_exists
except ImportError:
    pass

@depends('dependency_that_sometimes_exists')
def foo():
    '''
    Function with a dependency on the "dependency_that_sometimes_exists" module,
    if the "dependency_that_sometimes_exists" is missing this function will not exist
    '''
    return True

def _fallback():
    '''
    Fallback function for the depends decorator to replace a function with
    '''
    return '"dependency_that_sometimes_exists" needs to be installed for this function to exist'

@depends('dependency_that_sometimes_exists', fallback_function=_fallback)
def foo():
    '''
    Function with a dependency on the "dependency_that_sometimes_exists" module.
    If the "dependency_that_sometimes_exists" is missing this function will be
    replaced with "_fallback"
    '''
    return True
```

60.9 Examples of Salt Modules

The existing Salt modules should be fairly easy to read and understand, the goal of the main distribution's Salt modules is not only to build a set of functions for Salt, but to stand as examples for building out more Salt modules.

The existing modules can be found here: https://github.com/saltstack/salt/blob/develop/salt/modules

The most simple module is the test module, it contains the simplest Salt function, `test.ping`:

```
def ping():
    '''
    Just used to make sure the minion is up and responding
    Return True

    CLI Example::

        salt '*' test.ping
    '''
    return True
```

Full list of builtin execution modules

Virtual modules

61.1 salt.modules.pkg

`pkg` is a virtual module that is fulfilled by one of the following modules:

- `salt.modules.apt`
- `salt.modules.brew`
- `salt.modules.ebuild`
- `salt.modules.freebsdpkg`
- `salt.modules.openbsdpkg`
- `salt.modules.pacman`
- `salt.modules.pkgin`
- `salt.modules.pkgng`
- `salt.modules.pkgutil`
- `salt.modules.solarispkg`
- `salt.modules.win_pkg`
- `salt.modules.yumpkg`
- `salt.modules.yumpkg5`
- `salt.modules.zypper`

`aliases`	Manage the information in the aliases file
`alternatives`	Support for Alternatives system
`apache`	Support for Apache
`aptpkg`	Support for APT (Advanced Packaging Tool)
`archive`	A module to wrap archive calls
`at`	Wrapper module for at(1)
`augeas_cfg`	Manages configuration files via augeas
`aws_sqs`	Support for the Amazon Simple Queue Service.
`bluez`	Support for Bluetooth (using BlueZ in Linux).

Table 61.1 – continued from previous page

`brew`	Homebrew for Mac OS X
`bridge`	Module for gathering and managing bridging information
`bsd_shadow`	Manage the password database on BSD systems
`cassandra`	Cassandra NoSQL Database Module
`chocolatey`	A dead simple module wrapping calls to the Chocolatey package manager
`cloud`	Salt-specific interface for calling Salt Cloud directly
`cmdmod`	A module for shelling out
`composer`	Use composer to install PHP dependencies for a directory
`config`	Return config information
`cp`	Minion side functions for salt-cp
`cron`	Work with cron
`daemontools`	daemontools service module. This module will create daemontools type
`darwin_sysctl`	Module for viewing and modifying sysctl parameters
`data`	Manage a local persistent data structure that can hold any arbitrary data
`ddns`	Support for RFC 2136 dynamic DNS updates.
`debconfmod`	Support for Debconf
`debian_ip`	The networking module for Debian based distros
`debian_service`	Service support for Debian systems (uses update-rc.d and /sbin/service)
`dig`	Compendium of generic DNS utilities
`disk`	Module for gathering disk information
`djangomod`	Manage Django sites
`dnsmasq`	Module for managing dnqmasq
`dnsutil`	Compendium of generic DNS utilities
`dockerio`	Management of dockers
`dpkg`	Support for DEB packages
`ebuild`	Support for Portage
`eix`	Support for Eix
`eselect`	Support for eselect, Gentoo's configuration and management tool.
`event`	Use the *Salt Event System* to fire events from the master to the minion and vice-versa.
`extfs`	Module for managing ext2/3/4 file systems
`file`	Manage information about regular files, directories,
`freebsd_sysctl`	Module for viewing and modifying sysctl parameters
`freebsdjail`	The jail module for FreeBSD
`freebsdkmod`	Module to manage FreeBSD kernel modules
`freebsdpkg`	Remote package support using `pkg_add(1)`
`freebsdports`	Install software from the FreeBSD `ports(7)` system
`freebsdservice`	The service module for FreeBSD
`gem`	Manage ruby gems.
`gentoo_service`	Top level package command wrapper, used to translate the os detected by grains
`gentoolkitmod`	Support for Gentoolkit
`git`	Support for the Git SCM
`glance`	Module for handling openstack glance calls.
`gnomedesktop`	GNOME implementations
`grains`	Return/control aspects of the grains data
`groupadd`	Manage groups on Linux and OpenBSD
`grub_legacy`	Support for GRUB Legacy
`guestfs`	Interact with virtual machine images via libguestfs
`hg`	Support for the Mercurial SCM
`hosts`	Manage the information in the hosts file
`htpasswd`	Support for htpasswd command
`img`	Virtual machine image management tools
`iptables`	Support for iptables

Table 61.1 – continued from previous page

`key`	Functions to view the minion's public key information
`keyboard`	Module for managing keyboards on POSIX-like systems.
`keystone`	Module for handling openstack keystone calls.
`kmod`	Module to manage Linux kernel modules
`launchctl`	Module for the management of MacOS systems that use launchd/launchctl
`layman`	Support for Layman
`ldapmod`	Salt interface to LDAP commands
`linux_acl`	Support for Linux File Access Control Lists
`linux_lvm`	Support for Linux LVM2
`linux_sysctl`	Module for viewing and modifying sysctl parameters
`localemod`	Module for managing locales on POSIX-like systems.
`locate`	Module for using the locate utilities
`logrotate`	Module for managing logrotate.
`lvs`	Support for LVS (Linux Virtual Server)
`lxc`	Control Linux Containers via Salt
`mac_group`	Manage groups on Mac OS 10.7+
`mac_user`	Manage users on Mac OS 10.7+
`makeconf`	Support for modifying make.conf under Gentoo
`match`	The match module allows for match routines to be run and determine target specs
`mdadm`	Salt module to manage RAID arrays with mdadm
`memcached`	Module for Management of Memcached Keys
`mine`	The function cache system allows for data to be stored on the master so it can be easily read by other mini
`modjk`	Control Modjk via the Apache Tomcat "Status" worker
`mongodb`	Module to provide MongoDB functionality to Salt
`monit`	Monit service module.
`moosefs`	Module for gathering and managing information about MooseFS
`mount`	Salt module to manage unix mounts and the fstab file
`munin`	Run munin plugins/checks from salt and format the output as data.
`mysql`	Module to provide MySQL compatibility to salt.
`netbsd_sysctl`	Module for viewing and modifying sysctl parameters
`netbsdservice`	The service module for NetBSD
`network`	Module for gathering and managing network information
`nfs3`	Module for managing NFS version 3.
`nginx`	Support for nginx
`nova`	Module for handling OpenStack Nova calls.
`npm`	Manage and query NPM packages.
`omapi`	This module interacts with an ISC DHCP Server via OMAPI.
`openbsdpkg`	Package support for OpenBSD
`openbsdservice`	The service module for OpenBSD
`openstack_config`	Modify, retrieve, or delete values from OpenStack configuration files.
`osxdesktop`	Mac OS X implementations of various commands in the "desktop" interface
`pacman`	A module to wrap pacman calls, since Arch is the best
`pagerduty`	Module for Firing Events via PagerDuty
`pam`	Support for pam
`parted`	Module for managing partitions on POSIX-like systems.
`pecl`	Manage PHP pecl extensions.
`pillar`	Extract the pillar data for this minion
`pip`	Install Python packages with pip to either the system or a virtualenv
`pkg_resource`	Resources needed by pkg providers
`pkgin`	Package support for pkgin based systems, inspired from freebsdpkg module
`pkgng`	Support for `pkgng`, the new package manager for FreeBSD
`pkgutil`	Pkgutil support for Solaris

Table 61.1 – continued from previous page

`portage_config`	Configure `portage(5)`
`postgres`	Module to provide Postgres compatibility to salt.
`poudriere`	Support for poudriere
`powerpath`	powerpath support.
`ps`	A salt interface to psutil, a system and process library.
`publish`	Publish a command from a minion to a target
`puppet`	Execute puppet routines
`pw_group`	Manage groups on FreeBSD
`pw_user`	Manage users with the useradd command
`qemu_img`	Qemu-img Command Wrapper
`qemu_nbd`	Qemu Command Wrapper
`quota`	Module for managing quotas on POSIX-like systems.
`rabbitmq`	Module to provide RabbitMQ compatibility to Salt.
`rbenv`	Manage ruby installations with rbenv.
`rdp`	Manage RDP Service on Windows servers
`reg`	Manage the registry on Windows
`ret`	Module to integrate with the returner system and retrieve data sent to a salt returner
`rh_ip`	The networking module for RHEL/Fedora based distros
`rh_service`	Service support for RHEL-based systems, including support for both upstart and sysvinit
`riak`	Riak Salt Module
`rpm`	Support for rpm
`rvm`	Manage ruby installations and gemsets with RVM, the Ruby Version Manager.
`s3`	Connection module for Amazon S3
`saltcloudmod`	Control a salt cloud system
`saltutil`	The Saltutil module is used to manage the state of the salt minion itself.
`seed`	Virtual machine image management tools
`selinux`	Execute calls on selinux
`service`	The default service module, if not otherwise specified salt will fall back
`shadow`	Manage the shadow file
`smartos_imgadm`	Module for running imgadm command on SmartOS
`smartos_vmadm`	Module for managing VMs on SmartOS
`smf`	Service support for Solaris 10 and 11, should work with other systems that use SMF also.
`solaris_group`	Manage groups on Solaris
`solaris_shadow`	Manage the password database on Solaris systems
`solaris_user`	Manage users with the useradd command
`solarispkg`	Package support for Solaris
`solr`	Apache Solr Salt Module
`sqlite3`	Support for SQLite3
`ssh`	Manage client ssh components
`state`	Control the state system on the minion
`status`	Module for returning various status data about a minion.
`supervisord`	Provide the service module for system supervisord or supervisord in a
`svn`	Subversion SCM
`sysbench`	The ‘sysbench’ module is used to analyse the performance of the minions, right from the master! It measu
`sysmod`	The sys module provides information about the available functions on the minion
`system`	Support for reboot, shutdown, etc
`systemd`	Provide the service module for systemd
`test`	Module for running arbitrary tests
`timezone`	Module for managing timezone on POSIX-like systems.
`tls`	A salt module for SSL/TLS.
`tomcat`	Support for Tomcat
`upstart`	Module for the management of upstart systems.

Table 61.1 – continued from previous page

`useradd`	Manage users with the useradd command
`virt`	Work with virtual machines managed by libvirt
`virtualenv_mod`	Create virtualenv environments
`win_autoruns`	Module for listing programs that automatically run on startup
`win_disk`	Module for gathering disk information on Windows
`win_dns_client`	Module for configuring DNS Client on Windows systems
`win_file`	Manage information about files on the minion, set/read user, group
`win_firewall`	Module for configuring Windows Firewall
`win_groupadd`	Manage groups on Windows
`win_ip`	The networking module for Windows based systems
`win_network`	Module for gathering and managing network information
`win_ntp`	Management of NTP servers on Windows
`win_path`	Manage the Windows System PATH
`win_pkg`	A module to manage software on Windows
`win_repo`	Module to manage Windows software repo on a Standalone Minion
`win_servermanager`	Manage Windows features via the ServerManager powershell module
`win_service`	Windows Service module.
`win_shadow`	Manage the shadow file
`win_status`	Module for returning various status data about a minion.
`win_system`	Support for reboot, shutdown, etc
`win_timezone`	Module for managing timezone on Windows systems.
`win_useradd`	Manage Windows users with the net user command
`xapi`	This module (mostly) uses the XenAPI to manage Xen virtual machines.
`xmpp`	Module for Sending Messages via XMPP (a.k.a.
`yumpkg`	Support for YUM
`zcbuildout`	Management of zc.buildout
`zfs`	Module for running ZFS command
`zpool`	Module for running ZFS zpool command
`zypper`	Package support for openSUSE via the zypper package manager

61.2 salt.modules.aliases

Manage the information in the aliases file

`salt.modules.aliases.`**`get_target`**`(`*alias*`)`

Return the target associated with an alias

CLI Example:

```
salt '*' aliases.get_target alias
```

`salt.modules.aliases.`**`has_target`**`(`*alias*, *target*`)`

Return true if the alias/target is set

CLI Example:

```
salt '*' aliases.has_target alias target
```

`salt.modules.aliases.`**`list_aliases`**`()`

Return the aliases found in the aliases file in this format:

```
{'alias': 'target'}
```

CLI Example:

```
salt '*' aliases.list_aliases
```

`salt.modules.aliases.`**`rm_alias`**`(`*alias*`)`

Remove an entry from the aliases file

CLI Example:

```
salt '*' aliases.rm_alias alias
```

`salt.modules.aliases.`**`set_target`**`(`*alias*, *target*`)`

Set the entry in the aliases file for the given alias, this will overwrite any previous entry for the given alias or create a new one if it does not exist.

CLI Example:

```
salt '*' aliases.set_target alias target
```

61.3 salt.modules.alternatives

Support for Alternatives system

codeauthor Radek Rada <radek.rada@gmail.com>

copyright © 2012 by the SaltStack Team, see AUTHORS for more details.

license Apache 2.0, see LICENSE for more details.

`salt.modules.alternatives.`**`auto`**`(`*name*`)`

Trigger alternatives to set the path for <name> as specified by priority.

CLI Example:

```
salt '*' alternatives.auto name
```

`salt.modules.alternatives.`**`check_installed`**`(`*name*, *path*`)`

Check if the current highest-priority match for a given alternatives link is set to the desired path

CLI Example:

```
salt '*' alternatives.check_installed name path
```

`salt.modules.alternatives.`**`display`**`(`*name*`)`

Display alternatives settings for defined command name

CLI Example:

```
salt '*' alternatives.display editor
```

`salt.modules.alternatives.`**`install`**`(`*name*, *link*, *path*, *priority*`)`

Install symbolic links determining default commands

CLI Example:

```
salt '*' alternatives.install editor /usr/bin/editor /usr/bin/emacs23 50
```

`salt.modules.alternatives.`**`remove`**`(`*name*, *path*`)`

Remove symbolic links determining the default commands.

CLI Example:

```
salt '*' alternatives.remove name path
```

`salt.modules.alternatives.`**`set_`**(*name*, *path*)

Manually set the alternative <path> for <name>.

CLI Example:

```
salt '*' alternatives.set name path
```

`salt.modules.alternatives.`**`show_current`**(*name*)

Display the current highest-priority alternative for a given alternatives link

CLI Example:

```
salt '*' alternatives.show_current editor
```

61.4 salt.modules.apache

Support for Apache

`salt.modules.apache.`**`a2dissite`**(*site*)

Runs a2dissite for the given site.

This will only be functional on Debian-based operating systems (Ubuntu, Mint, etc).

CLI Examples:

```
salt '*' apache.a2dissite example.com
```

`salt.modules.apache.`**`a2ensite`**(*site*)

Runs a2ensite for the given site.

This will only be functional on Debian-based operating systems (Ubuntu, Mint, etc).

CLI Examples:

```
salt '*' apache.a2ensite example.com
```

`salt.modules.apache.`**`check_site_enabled`**(*site*)

Checks to see if the specific Site symlink is in /etc/apache2/sites-enabled.

This will only be functional on Debian-based operating systems (Ubuntu, Mint, etc).

CLI Examples:

```
salt '*' apache.check_site_enabled example.com
```

`salt.modules.apache.`**`directives`**()

Return list of directives together with expected arguments and places where the directive is valid (`apachectl -L`)

CLI Example:

```
salt '*' apache.directives
```

`salt.modules.apache.`**`fullversion`**()

Return server version from apachectl -V

CLI Example:

```
salt '*' apache.fullversion
```

salt.modules.apache.**modules**()

Return list of static and shared modules from apachectl -M

CLI Example:

```
salt '*' apache.modules
```

salt.modules.apache.**server_status**(*profile='default'*)

Get Information from the Apache server-status handler

NOTE: the server-status handler is disabled by default. in order for this function to work it needs to be enabled. http://httpd.apache.org/docs/2.2/mod/mod_status.html

The following configuration needs to exists in pillar/grains each entry nested in apache.server-status is a profile of a vhost/server this would give support for multiple apache servers/vhosts

apache.server-status:

> **'default':** 'url': http://localhost/server-status 'user': someuser 'pass': password 'realm': 'authentication realm for digest passwords' 'timeout': 5

CLI Examples:

```
salt '*' apache.server_status
salt '*' apache.server_status other-profile
```

salt.modules.apache.**servermods**()

Return list of modules compiled into the server (apachectl -l)

CLI Example:

```
salt '*' apache.servermods
```

salt.modules.apache.**signal**(*signal=None*)

Signals httpd to start, restart, or stop.

CLI Example:

```
salt '*' apache.signal restart
```

salt.modules.apache.**useradd**(*pwfile*, *user*, *password*, *opts=''*)

Add an HTTP user using the htpasswd command. If the htpasswd file does not exist, it will be created. Valid options that can be passed are:

> n Don't update file; display results on stdout. m Force MD5 encryption of the password (default). d Force CRYPT encryption of the password. p Do not encrypt the password (plaintext). s Force SHA encryption of the password.

CLI Examples:

```
salt '*' apache.useradd /etc/httpd/htpasswd larry badpassword
salt '*' apache.useradd /etc/httpd/htpasswd larry badpass opts=ns
```

salt.modules.apache.**userdel**(*pwfile*, *user*)

Delete an HTTP user from the specified htpasswd file.

CLI Examples:

```
salt '*' apache.userdel /etc/httpd/htpasswd larry
```

`salt.modules.apache.`**`version`**`()`

Return server version from apachectl -v

CLI Example:

```
salt '*' apache.version
```

`salt.modules.apache.`**`vhosts`**`()`

Show the settings as parsed from the config file (currently only shows the virtualhost settings). (`apachectl -S`) Because each additional virtual host adds to the execution time, this command may require a long timeout be specified.

CLI Example:

```
salt -t 10 '*' apache.vhosts
```

61.5 salt.modules.aptpkg

Support for APT (Advanced Packaging Tool)

`salt.modules.aptpkg.`**`del_repo`**`(`*repo*, ***kwargs*`)`

Delete a repo from the sources.list / sources.list.d

If the .list file is in the sources.list.d directory and the file that the repo exists in does not contain any other repo configuration, the file itself will be deleted.

The repo passed in must be a fully formed repository definition string.

CLI Examples:

```
salt '*' pkg.del_repo "myrepo definition"
```

`salt.modules.aptpkg.`**`expand_repo_def`**`(`*repokwargs*`)`

Take a repository definition and expand it to the full pkg repository dict that can be used for comparison. This is a helper function to make the Debian/Ubuntu apt sources sane for comparison in the pkgrepo states.

There is no use to calling this function via the CLI.

`salt.modules.aptpkg.`**`file_dict`**`(`**packages*`)`

List the files that belong to a package, grouped by package. Not specifying any packages will return a list of _every_ file on the system's package database (not generally recommended).

CLI Examples:

```
salt '*' pkg.file_list httpd
salt '*' pkg.file_list httpd postfix
salt '*' pkg.file_list
```

`salt.modules.aptpkg.`**`file_list`**`(`**packages*`)`

List the files that belong to a package. Not specifying any packages will return a list of _every_ file on the system's package database (not generally recommended).

CLI Examples:

```
salt '*' pkg.file_list httpd
salt '*' pkg.file_list httpd postfix
salt '*' pkg.file_list
```

`salt.modules.aptpkg.`**`get_repo`**`(`*repo*, ***kwargs*`)`

Display a repo from the sources.list / sources.list.d

The repo passed in needs to be a complete repo entry.

CLI Examples:

```
salt '*' pkg.get_repo "myrepo definition"
```

`salt.modules.aptpkg.`**`get_selections`**(*pattern=None*, *state=None*)

View package state from the dpkg database.

Returns a dict of dicts containing the state, and package names:

```
{'<host>':
    {'<state>': ['pkg1',
                 ...
                ]
    },
    ...
}
```

CLI Example:

```
salt '*' pkg.get_selections
salt '*' pkg.get_selections 'python-*'
salt '*' pkg.get_selections state=hold
salt '*' pkg.get_selections 'openssh*' state=hold
```

`salt.modules.aptpkg.`**`install`**(*name=None*, *refresh=False*, *fromrepo=None*, *skip_verify=False*, *debconf=None*, *pkgs=None*, *sources=None*, ***kwargs*)

Install the passed package, add refresh=True to update the dpkg database.

name The name of the package to be installed. Note that this parameter is ignored if either "pkgs" or "sources" is passed. Additionally, please note that this option can only be used to install packages from a software repository. To install a package file manually, use the "sources" option.

32-bit packages can be installed on 64-bit systems by appending the architecture designation (`:i386`, etc.) to the end of the package name.

CLI Example:

```
salt '*' pkg.install <package name>
```

refresh Whether or not to refresh the package database before installing.

fromrepo Specify a package repository to install from (e.g., `apt-get -t unstable install somepackage`)

skip_verify Skip the GPG verification check (e.g., `--allow-unauthenticated`, or `--force-bad-verify` for install from package file).

debconf Provide the path to a debconf answers file, processed before installation.

version Install a specific version of the package, e.g. 1.2.3~0ubuntu0. Ignored if "pkgs" or "sources" is passed.

Multiple Package Installation Options:

pkgs A list of packages to install from a software repository. Must be passed as a python list.

CLI Example:

```
salt '*' pkg.install pkgs='["foo", "bar"]'
salt '*' pkg.install pkgs='["foo", {"bar": "1.2.3-0ubuntu0"}]'
```

sources A list of DEB packages to install. Must be passed as a list of dicts, with the keys being package names, and the values being the source URI or local path to the package. Dependencies are automatically resolved and marked as auto-installed.

32-bit packages can be installed on 64-bit systems by appending the architecture designation (`:i386`, etc.) to the end of the package name.

Changed in version Helium.

CLI Example:

```
salt '*' pkg.install sources='[{"foo": "salt://foo.deb"},{"bar": "salt://bar.deb"}]'
```

force_yes Passes `--force-yes` to the apt-get command. Don't use this unless you know what you're doing.

New in version 0.17.4.

Returns a dict containing the new package names and versions:

```
{'<package>': {'old': '<old-version>',
               'new': '<new-version>'}}
```

`salt.modules.aptpkg.`**`latest_version`**`(`**names*, ***kwargs*`)`

Return the latest version of the named package available for upgrade or installation. If more than one package name is specified, a dict of name/version pairs is returned.

If the latest version of a given package is already installed, an empty string will be returned for that package.

A specific repo can be requested using the `fromrepo` keyword argument.

CLI Example:

```
salt '*' pkg.latest_version <package name>
salt '*' pkg.latest_version <package name> fromrepo=unstable
salt '*' pkg.latest_version <package1> <package2> <package3> ...
```

`salt.modules.aptpkg.`**`list_pkgs`**`(`*versions_as_list=False*, *removed=False*, *purge_desired=False*, ***kwargs*`)`

List the packages currently installed in a dict:

```
{'<package_name>': '<version>'}
```

removed If `True`, then only packages which have been removed (but not purged) will be returned.

purge_desired If `True`, then only packages which have been marked to be purged, but can't be purged due to their status as dependencies for other installed packages, will be returned. Note that these packages will appear in installed

Changed in version 2014.1.1: Packages in this state now correctly show up in the output of this function.

External dependencies:

```
Virtual package resolution requires dctrl-tools. Without dctrl-tools
virtual packages will be reported as not installed.
```

CLI Example:

```
salt '*' pkg.list_pkgs
salt '*' pkg.list_pkgs versions_as_list=True
```

`salt.modules.aptpkg.`**`list_repos`**`()`

Lists all repos in the sources.list (and sources.lists.d) files

CLI Example:

```
salt '*' pkg.list_repos
salt '*' pkg.list_repos disabled=True
```

`salt.modules.aptpkg.`**`list_upgrades`**`(`*refresh=True*`)`

List all available package upgrades.

CLI Example:

```
salt '*' pkg.list_upgrades
```

`salt.modules.aptpkg.`**`mod_repo`**`(`*repo*, *saltenv='base'*, ***kwargs*`)`

Modify one or more values for a repo. If the repo does not exist, it will be created, so long as the definition is well formed. For Ubuntu the "ppa:<project>/repo" format is acceptable. "ppa:" format can only be used to create a new repository.

The following options are available to modify a repo definition:

```
comps (a comma separated list of components for the repo, e.g. "main")
file (a file name to be used)
keyserver (keyserver to get gpg key from)
keyid (key id to load with the keyserver argument)
key_url (URL to a gpg key to add to the apt gpg keyring)
consolidate (if true, will attempt to de-dup and consolidate sources)

* Note: Due to the way keys are stored for apt, there is a known issue
        where the key wont be updated unless another change is made
        at the same time.  Keys should be properly added on initial
        configuration.
```

CLI Examples:

```
salt '*' pkg.mod_repo 'myrepo definition' uri=http://new/uri
salt '*' pkg.mod_repo 'myrepo definition' comps=main,universe
```

`salt.modules.aptpkg.`**`purge`**`(`*name=None*, *pkgs=None*, ***kwargs*`)`

Remove packages via `apt-get purge` along with all configuration files and unused dependencies.

name The name of the package to be deleted.

Multiple Package Options:

pkgs A list of packages to delete. Must be passed as a python list. The `name` parameter will be ignored if this option is passed.

New in version 0.16.0.

Returns a dict containing the changes.

CLI Example:

```
salt '*' pkg.purge <package name>
salt '*' pkg.purge <package1>,<package2>,<package3>
salt '*' pkg.purge pkgs='["foo", "bar"]'
```

`salt.modules.aptpkg.`**`refresh_db`**`()`

Updates the APT database to latest packages based upon repositories

Returns a dict, with the keys being package databases and the values being the result of the update attempt. Values can be one of the following:

- `True`: Database updated successfully

- `False`: Problem updating database
- `None`: Database already up-to-date

CLI Example:

```
salt '*' pkg.refresh_db
```

`salt.modules.aptpkg.`**`remove`**(*name=None*, *pkgs=None*, ***kwargs*)

Remove packages using `apt-get remove`.

name The name of the package to be deleted.

Multiple Package Options:

pkgs A list of packages to delete. Must be passed as a python list. The `name` parameter will be ignored if this option is passed.

New in version 0.16.0.

Returns a dict containing the changes.

CLI Example:

```
salt '*' pkg.remove <package name>
salt '*' pkg.remove <package1>,<package2>,<package3>
salt '*' pkg.remove pkgs='["foo", "bar"]'
```

`salt.modules.aptpkg.`**`set_selections`**(*path=None*, *selection=None*, *clear=False*, *saltenv='base'*)

Change package state in the dpkg database.

The state can be any one of, documented in `dpkg(1)`:

- install
- hold
- deinstall
- purge

This command is commonly used to mark specific packages to be held from being upgraded, that is, to be kept at a certain version. When a state is changed to anything but being held, then it is typically followed by `apt-get -u dselect-upgrade`.

Note: Be careful with the `clear` argument, since it will start with setting all packages to deinstall state.

Returns a dict of dicts containing the package names, and the new and old versions:

```
{'<host>':
    {'<package>': {'new': '<new-state>',
                   'old': '<old-state>'}
    },
    ...
}
```

CLI Example:

```
salt '*' pkg.set_selections selection='{"install": ["netcat"]}'
salt '*' pkg.set_selections selection='{"hold": ["openssh-server", "openssh-client"]}'
salt '*' pkg.set_selections salt://path/to/file
salt '*' pkg.set_selections salt://path/to/file clear=True
```

`salt.modules.aptpkg.`**`upgrade`**`(`*refresh=True*, ***kwargs*`)`

Upgrades all packages via `apt-get dist-upgrade`

Returns a dict containing the changes.

{'<package>': {'old': '<old-version>', 'new': '<new-version>'}}

CLI Example:

```
salt '*' pkg.upgrade
```

`salt.modules.aptpkg.`**`upgrade_available`**`(`*name*`)`

Check whether or not an upgrade is available for a given package

CLI Example:

```
salt '*' pkg.upgrade_available <package name>
```

`salt.modules.aptpkg.`**`version`**`(`**names*, ***kwargs*`)`

Returns a string representing the package version or an empty string if not installed. If more than one package name is specified, a dict of name/version pairs is returned.

CLI Example:

```
salt '*' pkg.version <package name>
salt '*' pkg.version <package1> <package2> <package3> ...
```

`salt.modules.aptpkg.`**`version_cmp`**`(`*pkg1*, *pkg2*`)`

Do a cmp-style comparison on two packages. Return -1 if pkg1 < pkg2, 0 if pkg1 == pkg2, and 1 if pkg1 > pkg2. Return None if there was a problem making the comparison.

CLI Example:

```
salt '*' pkg.version_cmp '0.2.4-0ubuntu1' '0.2.4.1-0ubuntu1'
```

61.6 salt.modules.archive

A module to wrap archive calls

`salt.modules.archive.`**`gunzip`**`(`*gzipfile*, *template=None*`)`

Uses the gunzip command to unpack gzip files

CLI Example to create `/tmp/sourcefile.txt`:

```
salt '*' archive.gunzip /tmp/sourcefile.txt.gz
```

The template arg can be set to 'jinja' or another supported template engine to render the command arguments before execution.

CLI Example:

```
salt '*' archive.gunzip template=jinja /tmp/{{grains.id}}.txt.gz
```

`salt.modules.archive.`**`gzip`**`(`*sourcefile*, *template=None*`)`

Uses the gzip command to create gzip files

CLI Example to create `/tmp/sourcefile.txt.gz`:

```
salt '*' archive.gzip /tmp/sourcefile.txt
```

The template arg can be set to 'jinja' or another supported template engine to render the command arguments before execution.

CLI Example:

```
salt '*' archive.gzip template=jinja /tmp/{{grains.id}}.txt
```

`salt.modules.archive.`**`rar`**(*rarfile*, *sources*, *template=None*)

Uses the rar command to create rar files Uses rar for Linux from http://www.rarlab.com/

CLI Example:

```
salt '*' archive.rar /tmp/rarfile.rar /tmp/sourcefile1,/tmp/sourcefile2
```

The template arg can be set to 'jinja' or another supported template engine to render the command arguments before execution.

For example:

```
salt '*' archive.rar template=jinja /tmp/rarfile.rar /tmp/sourcefile1,/tmp/{{grains.id}}.txt
```

`salt.modules.archive.`**`tar`**(*options*, *tarfile*, *sources=None*, *dest=None*, *cwd=None*, *template=None*)

Note: This function has changed for version 0.17.0. In prior versions, the `cwd` and `template` arguments must be specified, with the source directories/files coming as a space-separated list at the end of the command. Beginning with 0.17.0, `sources` must be a comma-separated list, and the `cwd` and `template` arguments are optional.

Uses the tar command to pack, unpack, etc tar files

options: Options to pass to the `tar` binary.

tarfile: The tar filename to pack/unpack.

sources: Comma delimited list of files to **pack** into the tarfile.

dest: The destination directory to **unpack** the tarfile to.

cwd: The directory in which the tar command should be executed.

template: Template engine name to render the command arguments before execution.

CLI Example:

```
salt '*' archive.tar cjvf /tmp/tarfile.tar.bz2 /tmp/file_1,/tmp/file_2
```

The template arg can be set to `jinja` or another supported template engine to render the command arguments before execution. For example:

```
salt '*' archive.tar template=jinja cjvf /tmp/salt.tar.bz2 {{grains.saltpath}}
```

To unpack a tarfile, for example:

```
salt '*' archive.tar foo.tar xf dest=/target/directory
```

`salt.modules.archive.`**`unrar`**(*rarfile*, *dest*, *excludes=None*, *template=None*)

Uses the unrar command to unpack rar files Uses rar for Linux from http://www.rarlab.com/

CLI Example:

```
salt '*' archive.unrar /tmp/rarfile.rar /home/strongbad/ excludes=file_1,file_2
```

The template arg can be set to 'jinja' or another supported template engine to render the command arguments before execution.

For example:

```
salt '*' archive.unrar template=jinja /tmp/rarfile.rar /tmp/{{grains.id}}/ excludes=file_1,file_
```

`salt.modules.archive.`**`unzip`**(*zipfile*, *dest*, *excludes=None*, *template=None*, *options=None*)

Uses the unzip command to unpack zip files

options: Options to pass to the `unzip` binary.

CLI Example:

```
salt '*' archive.unzip /tmp/zipfile.zip /home/strongbad/ excludes=file_1,file_2
```

The template arg can be set to 'jinja' or another supported template engine to render the command arguments before execution.

For example:

```
salt '*' archive.unzip template=jinja /tmp/zipfile.zip /tmp/{{grains.id}}/ excludes=file_1,file_
```

`salt.modules.archive.`**`zip_`**(*zipfile*, *sources*, *template=None*)

Uses the zip command to create zip files

CLI Example:

```
salt '*' archive.zip /tmp/zipfile.zip /tmp/sourcefile1,/tmp/sourcefile2
```

The template arg can be set to 'jinja' or another supported template engine to render the command arguments before execution.

For example:

```
salt '*' archive.zip template=jinja /tmp/zipfile.zip /tmp/sourcefile1,/tmp/{{grains.id}}.txt
```

61.7 salt.modules.at

Wrapper module for at(1)

Also, a 'tag' feature has been added to more easily tag jobs.

`salt.modules.at.`**`at`**(**args*, ***kwargs*)

Add a job to the queue.

The 'timespec' follows the format documented in the at(1) manpage.

CLI Example:

```
salt '*' at.at <timespec> <cmd> [tag=<tag>] [runas=<user>]
salt '*' at.at 12:05am '/sbin/reboot' tag=reboot
salt '*' at.at '3:05am +3 days' 'bin/myscript' tag=nightly runas=jim
```

`salt.modules.at.`**`atc`**(*jobid*)

Print the at(1) script that will run for the passed job id. This is mostly for debugging so the output will just be text.

CLI Example:

```
salt '*' at.atc <jobid>
```

`salt.modules.at.`**`atq`**`(`*tag=None*`)`

List all queued and running jobs or only those with an optional 'tag'.

CLI Example:

```
salt '*' at.atq
salt '*' at.atq [tag]
salt '*' at.atq [job number]
```

`salt.modules.at.`**`atrm`**`(`**args*`)`

Remove jobs from the queue.

CLI Example:

```
salt '*' at.atrm <jobid> <jobid> .. <jobid>
salt '*' at.atrm all
salt '*' at.atrm all [tag]
```

`salt.modules.at.`**`jobcheck`**`(`***kwargs*`)`

Check the job from queue. The kwargs dict include 'hour minute day month year tag runas' Other parameters will be ignored.

CLI Example:

```
salt '*' at.jobcheck runas=jam day=13
salt '*' at.jobcheck day=13 month=12 year=13 tag=rose
```

61.8 salt.modules.augeas_cfg

Manages configuration files via augeas

This module requires the `augeas` Python module.

> **Warning:** Minimal installations of Debian and Ubuntu have been seen to have packaging bugs with python-augeas, causing the augeas module to fail to import. If the minion has the augeas module installed, but the functions in this execution module fail to run due to being unavailable, first restart the salt-minion service. If the problem persists past that, the following command can be run from the master to determine what is causing the import to fail:
>
> ```
> salt minion-id cmd.run 'python -c "from augeas import Augeas"'
> ```
>
> For affected Debian/Ubuntu hosts, installing `libpython2.7` has been known to resolve the issue.

`salt.modules.augeas_cfg.`**`get`**`(`*path*, *value=''*`)`

Get a value for a specific augeas path

CLI Example:

```
salt '*' augeas.get /files/etc/hosts/1/ ipaddr
```

`salt.modules.augeas_cfg.`**`ls`**`(`*path*`)`

List the direct children of a node

CLI Example:

```
salt '*' augeas.ls /files/etc/passwd
```

`salt.modules.augeas_cfg.`**`match`**`(`*path*, *value=''*`)`

Get matches for path expression

CLI Example:

```
salt '*' augeas.match /files/etc/services/service-name ssh
```

`salt.modules.augeas_cfg.`**`remove`**`(`*path*`)`

Get matches for path expression

CLI Example:

```
salt '*' augeas.remove /files/etc/sysctl.conf/net.ipv4.conf.all.log_martians
```

`salt.modules.augeas_cfg.`**`setvalue`**`(`**args*`)`

Set a value for a specific augeas path

CLI Example:

```
salt '*' augeas.setvalue /files/etc/hosts/1/canonical localhost
```

This will set the first entry in /etc/hosts to localhost

CLI Example:

```
salt '*' augeas.setvalue /files/etc/hosts/01/ipaddr 192.168.1.1 \
                         /files/etc/hosts/01/canonical test
```

Adds a new host to /etc/hosts the ip address 192.168.1.1 and hostname test

CLI Example:

```
salt '*' augeas.setvalue prefix=/files/etc/sudoers/ \
         "spec[user = '%wheel']/user" "%wheel" \
         "spec[user = '%wheel']/host_group/host" 'ALL' \
         "spec[user = '%wheel']/host_group/command[1]" 'ALL' \
         "spec[user = '%wheel']/host_group/command[1]/tag" 'PASSWD' \
         "spec[user = '%wheel']/host_group/command[2]" '/usr/bin/apt-get' \
         "spec[user = '%wheel']/host_group/command[2]/tag" NOPASSWD
```

Ensures that the following line is present in /etc/sudoers:

```
%wheel ALL = PASSWD : ALL , NOPASSWD : /usr/bin/apt-get , /usr/bin/aptitude
```

`salt.modules.augeas_cfg.`**`tree`**`(`*path*`)`

Returns recursively the complete tree of a node

CLI Example:

```
salt '*' augeas.tree /files/etc/
```

61.9 salt.modules.aws_sqs

Support for the Amazon Simple Queue Service.

`salt.modules.aws_sqs.`**`create_queue`**`(`*name*, *region*, *opts=None*, *user=None*`)`

Creates a queue with the correct name.

name Name of the SQS queue to create

region Region to create the SQS queue in

opts [None] Any additional options to add to the command line

user [None] Run hg as a user other than what the minion runs as

`salt.modules.aws_sqs.`**`delete_queue`**`(`*name*, *region*, *opts=None*, *user=None*`)`
Deletes a queue in the region.

name Name of the SQS queue to deletes

region Name of the region to delete the queue from

opts [None] Any additional options to add to the command line

user [None] Run hg as a user other than what the minion runs as

`salt.modules.aws_sqs.`**`list_queues`**`(`*region*, *opts=None*, *user=None*`)`
List the queues in the selected region.

region Region to list SQS queues for

opts [None] Any additional options to add to the command line

user [None] Run hg as a user other than what the minion runs as

`salt.modules.aws_sqs.`**`queue_exists`**`(`*name*, *region*, *opts=None*, *user=None*`)`
Returns True or False on whether the queue exists in the region

name Name of the SQS queue to search for

region Name of the region to search for the queue in

opts [None] Any additional options to add to the command line

user [None] Run hg as a user other than what the minion runs as

61.10 salt.modules.bluez

Support for Bluetooth (using BlueZ in Linux).

The following packages are required packages for this module:

bluez >= 5.7 bluez-libs >= 5.7 bluez-utils >= 5.7 pybluez >= 0.18

`salt.modules.bluez.`**`address_`**`()`
Get the many addresses of the Bluetooth adapter

CLI Example:

```
salt '*' bluetooth.address
```

`salt.modules.bluez.`**`block`**`(`*bdaddr*`)`
Block a specific bluetooth device by BD Address

CLI Example:

```
salt '*' bluetooth.block DE:AD:BE:EF:CA:FE
```

`salt.modules.bluez.`**`discoverable`**`(`*dev*`)`
Enable this bluetooth device to be discoverable.

CLI Example:

```
salt '*' bluetooth.discoverable hci0
```

`salt.modules.bluez.`**`noscan`**`(`*dev*`)`

Turn off scanning modes on this device.

CLI Example:

```
salt '*' bluetooth.noscan hci0
```

`salt.modules.bluez.`**`pair`**`(`*address*, *key*`)`

Pair the bluetooth adapter with a device

CLI Example:

```
salt '*' bluetooth.pair DE:AD:BE:EF:CA:FE 1234
```

Where DE:AD:BE:EF:CA:FE is the address of the device to pair with, and 1234 is the passphrase.

TODO: This function is currently broken, as the bluez-simple-agent program no longer ships with BlueZ >= 5.0. It needs to be refactored.

`salt.modules.bluez.`**`power`**`(`*dev*, *mode*`)`

Power a bluetooth device on or off

CLI Examples:

```
salt '*' bluetooth.power hci0 on
salt '*' bluetooth.power hci0 off
```

`salt.modules.bluez.`**`scan`**`()`

Scan for bluetooth devices in the area

CLI Example:

```
salt '*' bluetooth.scan
```

`salt.modules.bluez.`**`start`**`()`

Start the bluetooth service.

CLI Example:

```
salt '*' bluetooth.start
```

`salt.modules.bluez.`**`stop`**`()`

Stop the bluetooth service.

CLI Example:

```
salt '*' bluetooth.stop
```

`salt.modules.bluez.`**`unblock`**`(`*bdaddr*`)`

Unblock a specific bluetooth device by BD Address

CLI Example:

```
salt '*' bluetooth.unblock DE:AD:BE:EF:CA:FE
```

`salt.modules.bluez.`**`unpair`**`(`*address*`)`

Unpair the bluetooth adapter from a device

CLI Example:

```
salt '*' bluetooth.unpair DE:AD:BE:EF:CA:FE
```

Where DE:AD:BE:EF:CA:FE is the address of the device to unpair.

TODO: This function is currently broken, as the bluez-simple-agent program no longer ships with BlueZ >= 5.0. It needs to be refactored.

`salt.modules.bluez.`**`version`**`()`

Return Bluez version from bluetoothd -v

CLI Example:

```
salt '*' bluetoothd.version
```

61.11 salt.modules.brew

Homebrew for Mac OS X

`salt.modules.brew.`**`install`**`(`*name=None*, *pkgs=None*, *taps=None*, *options=None*, ***kwargs*`)`

Install the passed package(s) with `brew install`

name The name of the formula to be installed. Note that this parameter is ignored if "pkgs" is passed.

CLI Example:

```
salt '*' pkg.install <package name>
```

taps Unofficial Github repos to use when updating and installing formulas.

CLI Example:

```
salt '*' pkg.install <package name> tap='<tap>'
salt '*' pkg.install zlib taps='homebrew/dupes'
salt '*' pkg.install php54 taps='["josegonzalez/php", "homebrew/dupes"]'
```

options Options to pass to brew. Only applies to inital install. Due to how brew works, modifying chosen options requires a full uninstall followed by a fresh install. Note that if "pkgs" is used, all options will be passed to all packages. Unreconized options for a package will be silently ignored by brew.

CLI Example:

```
salt '*' pkg.install <package name> tap='<tap>'
salt '*' pkg.install php54 taps='["josegonzalez/php", "homebrew/dupes"]' options='["--with-f
```

Multiple Package Installation Options:

pkgs A list of formulas to install. Must be passed as a python list.

CLI Example:

```
salt '*' pkg.install pkgs='["foo","bar"]'
```

Returns a dict containing the new package names and versions:

```
{'<package>': {'old': '<old-version>',
               'new': '<new-version>'}}
```

CLI Example:

```
salt '*' pkg.install 'package package package'
```

`salt.modules.brew.`**`latest_version`**`(`**names*, ***kwargs*`)`

Return the latest version of the named package available for upgrade or installation

Note that this currently not fully implemented but needs to return something to avoid a traceback when calling pkg.latest.

CLI Example:

```
salt '*' pkg.latest_version <package name>
salt '*' pkg.latest_version <package1> <package2> <package3>
```

`salt.modules.brew.`**`list_pkgs`**`(`*versions_as_list=False*, ***kwargs*`)`

List the packages currently installed in a dict:

```
{'<package_name>': '<version>'}
```

CLI Example:

```
salt '*' pkg.list_pkgs
```

`salt.modules.brew.`**`list_upgrades`**`()`

Check whether or not an upgrade is available for all packages

CLI Example:

```
salt '*' pkg.list_upgrades
```

`salt.modules.brew.`**`refresh_db`**`()`

Update the homebrew package repository.

CLI Example:

```
salt '*' pkg.refresh_db
```

`salt.modules.brew.`**`remove`**`(`*name=None*, *pkgs=None*, ***kwargs*`)`

Removes packages with `brew uninstall`.

name The name of the package to be deleted.

Multiple Package Options:

pkgs A list of packages to delete. Must be passed as a python list. The `name` parameter will be ignored if this option is passed.

New in version 0.16.0.

Returns a dict containing the changes.

CLI Example:

```
salt '*' pkg.remove <package name>
salt '*' pkg.remove <package1>,<package2>,<package3>
salt '*' pkg.remove pkgs='["foo", "bar"]'
```

`salt.modules.brew.`**`upgrade_available`**`(`*pkg*`)`

Check whether or not an upgrade is available for a given package

CLI Example:

```
salt '*' pkg.upgrade_available <package name>
```

`salt.modules.brew.`**`version`**`(`**names*, ***kwargs*`)`

Returns a string representing the package version or an empty string if not installed. If more than one package name is specified, a dict of name/version pairs is returned.

CLI Example:

```
salt '*' pkg.version <package name>
salt '*' pkg.version <package1> <package2> <package3>
```

61.12 salt.modules.bridge

Module for gathering and managing bridging information

`salt.modules.bridge.`**`add`**`(`*br=None*`)`

Creates a bridge

CLI Example:

```
salt '*' bridge.add br0
```

`salt.modules.bridge.`**`addif`**`(`*br=None*, *iface=None*`)`

Adds an interface to a bridge

CLI Example:

```
salt '*' bridge.addif br0 eth0
```

`salt.modules.bridge.`**`delete`**`(`*br=None*`)`

Deletes a bridge

CLI Example:

```
salt '*' bridge.delete br0
```

`salt.modules.bridge.`**`delif`**`(`*br=None*, *iface=None*`)`

Removes an interface from a bridge

CLI Example:

```
salt '*' bridge.delif br0 eth0
```

`salt.modules.bridge.`**`find_interfaces`**`(`**args*`)`

Returns the bridge to which the interfaces are bond to

CLI Example:

```
salt '*' bridge.find_interfaces eth0 [eth1...]
```

`salt.modules.bridge.`**`interfaces`**`(`*br=None*`)`

Returns interfaces attached to a bridge

CLI Example:

```
salt '*' bridge.interfaces br0
```

`salt.modules.bridge.`**`list_`**`()`

Returns the machine's bridges list

CLI Example:

```
salt '*' bridge.list
```

`salt.modules.bridge.`**`show`**`(`*br=None*`)`

Returns bridges interfaces along with enslaved physical interfaces. If no interface is given, all bridges are shown, else only the specified bridge values are returned.

CLI Example:

```
salt '*' bridge.show
salt '*' bridge.show br0
```

`salt.modules.bridge.`**`stp`**(*br=None*, *state='disable'*, *iface=None*)

Sets Spanning Tree Protocol state for a bridge

CLI Example:

```
salt '*' bridge.stp br0 enable
salt '*' bridge.stp br0 disable
```

For the NetBSD operating system, it is required to add the interface on which to enable the STP.

CLI Example:

```
salt '*' bridge.stp bridge0 enable fxp0
salt '*' bridge.stp bridge0 disable fxp0
```

61.13 salt.modules.bsd_shadow

Manage the password database on BSD systems

`salt.modules.bsd_shadow.`**`default_hash`**()

Returns the default hash used for unset passwords

CLI Example:

```
salt '*' shadow.default_hash
```

`salt.modules.bsd_shadow.`**`info`**(*name*)

Return information for the specified user

CLI Example:

```
salt '*' shadow.info someuser
```

`salt.modules.bsd_shadow.`**`set_password`**(*name*, *password*)

Set the password for a named user. The password must be a properly defined hash. The password hash can be generated with this command:

`python -c "import crypt; print crypt.crypt('password', ciphersalt)"`

NOTE: When constructing the `ciphersalt` string, you must escape any dollar signs, to avoid them being interpolated by the shell.

`'password'` is, of course, the password for which you want to generate a hash.

`ciphersalt` is a combination of a cipher identifier, an optional number of rounds, and the cryptographic salt. The arrangement and format of these fields depends on the cipher and which flavor of BSD you are using. For more information on this, see the manpage for `crpyt(3)`. On NetBSD, additional information is available in `passwd.conf(5)`.

It is important to make sure that a supported cipher is used.

CLI Example:

```
salt '*' shadow.set_password someuser '$1$UYCIxa628.9qXjpQCjM4a..'
```

61.14 salt.modules.cassandra

Cassandra NoSQL Database Module

depends

- pycassa Cassandra Python adapter

configuration The location of the 'nodetool' command, host, and thrift port needs to be specified via pillar:

```
cassandra.nodetool: /usr/local/bin/nodetool
cassandra.host: localhost
cassandra.thrift_port: 9160
```

`salt.modules.cassandra.`**`column_families`**`(`*keyspace=None*`)`

Return existing column families for all keyspaces or just the provided one.

CLI Example:

```
salt '*' cassandra.column_families
salt '*' cassandra.column_families <keyspace>
```

`salt.modules.cassandra.`**`column_family_definition`**`(`*keyspace=None, column_family=None*`)`

Return a dictionary of column family definitions for the given keyspace/column_family

CLI Example:

```
salt '*' cassandra.column_family_definition <keyspace> <column_family>
```

`salt.modules.cassandra.`**`compactionstats`**`()`

Return compactionstats info

CLI Example:

```
salt '*' cassandra.compactionstats
```

`salt.modules.cassandra.`**`info`**`()`

Return cassandra node info

CLI Example:

```
salt '*' cassandra.info
```

`salt.modules.cassandra.`**`keyspaces`**`()`

Return existing keyspaces

CLI Example:

```
salt '*' cassandra.keyspaces
```

`salt.modules.cassandra.`**`netstats`**`()`

Return netstats info

CLI Example:

```
salt '*' cassandra.netstats
```

`salt.modules.cassandra.`**`ring`**`()`

Return cassandra ring info

CLI Example:

```
salt '*' cassandra.ring
```

`salt.modules.cassandra.`**`tpstats`**`()`

Return tpstats info

CLI Example:

```
salt '*' cassandra.tpstats
```

`salt.modules.cassandra.`**`version`**`()`

Return the cassandra version

CLI Example:

```
salt '*' cassandra.version
```

61.15 salt.modules.chocolatey

A dead simple module wrapping calls to the Chocolatey package manager (http://chocolatey.org)

New in version 2014.1.0: (Hydrogen)

`salt.modules.chocolatey.`**`bootstrap`**`(`*force=False*`)`

Download and install the latest version of the Chocolatey package manager via the official bootstrap.

Chocolatey requires Windows PowerShell and the .NET v4.0 runtime. Depending on the host's version of Windows, chocolatey.bootstrap will attempt to ensure these prerequisites are met by downloading and executing the appropriate installers from Microsoft.

Note that if PowerShell is installed, you may have to restart the host machine for Chocolatey to work.

force Run the bootstrap process even if Chocolatey is found in the path.

```
salt '*' chocolatey.bootstrap
salt '*' chocolatey.bootstrap force=True
```

`salt.modules.chocolatey.`**`install`**`(`*name*, *version=None*, *source=None*, *force=False*`)`

Instructs Chocolatey to install a package.

name The name of the package to be installed. Only accepts a single argument.

version Install a specific version of the package. Defaults to latest version.

source Chocolatey repository (directory, share or remote URL feed) the package comes from. Defaults to the official Chocolatey feed.

force Reinstall the current version of an existing package.

CLI Example:

```
salt '*' chocolatey.install <package name>
salt '*' chocolatey.install <package name> version=<package version>
```

`salt.modules.chocolatey.`**`install_cygwin`**`(`*name*`)`

Instructs Chocolatey to install a package via Cygwin.

name The name of the package to be installed. Only accepts a single argument.

CLI Example:

```
salt '*' chocolatey.install_cygwin <package name>
```

salt.modules.chocolatey.**install_gem**(*name*, *version=None*)

Instructs Chocolatey to install a package via Ruby's Gems.

name The name of the package to be installed. Only accepts a single argument.

version Install a specific version of the package. Defaults to latest version available.

CLI Example:

```
salt '*' chocolatey.install_gem <package name>
salt '*' chocolatey.install_gem <package name> version=<package version>
```

salt.modules.chocolatey.**install_missing**(*name*, *version=None*, *source=None*)

Instructs Chocolatey to install a package if it doesn't already exist.

name The name of the package to be installed. Only accepts a single argument.

version Install a specific version of the package. Defaults to latest version available.

source Chocolatey repository (directory, share or remote URL feed) the package comes from. Defaults to the official Chocolatey feed.

CLI Example:

```
salt '*' chocolatey.install_missing <package name>
salt '*' chocolatey.install_missing <package name> version=<package version>
```

salt.modules.chocolatey.**install_python**(*name*, *version=None*)

Instructs Chocolatey to install a package via Python's easy_install.

name The name of the package to be installed. Only accepts a single argument.

version Install a specific version of the package. Defaults to latest version available.

CLI Example:

```
salt '*' chocolatey.install_python <package name>
salt '*' chocolatey.install_python <package name> version=<package version>
```

salt.modules.chocolatey.**install_webpi**(*name*)

Instructs Chocolatey to install a package via the Microsoft Web PI service.

name The name of the package to be installed. Only accepts a single argument.

CLI Example:

```
salt '*' chocolatey.install_webpi <package name>
```

salt.modules.chocolatey.**install_windowsfeatures**(*name*)

Instructs Chocolatey to install a Windows Feature via the Deployment Image Servicing and Management tool.

name The name of the feature to be installed. Only accepts a single argument.

CLI Example:

```
salt '*' chocolatey.install_windowsfeatures <package name>
```

salt.modules.chocolatey.**list_**(*filter*, *all_versions=False*, *pre_versions=False*, *source=None*)

Instructs Chocolatey to pull a vague package list from the repository.

filter Term used to filter down results. Searches against name/description/tag.

all_versions Display all available package versions in results. Defaults to False.

pre_versions Display pre-release packages in results. Defaults to False.

source Chocolatey repository (directory, share or remote URL feed) the package comes from. Defaults to the official Chocolatey feed.

CLI Example:

```
salt '*' chocolatey.list <filter>
salt '*' chocolatey.list <filter> all_versions=True
```

`salt.modules.chocolatey.`**`list_webpi`**`()`

Instructs Chocolatey to pull a full package list from the Microsoft Web PI repository.

CLI Example:

```
salt '*' chocolatey.list_webpi
```

`salt.modules.chocolatey.`**`list_windowsfeatures`**`()`

Instructs Chocolatey to pull a full package list from the Windows Features list, via the Deployment Image Servicing and Management tool.

CLI Example:

```
salt '*' chocolatey.list_windowsfeatures
```

`salt.modules.chocolatey.`**`uninstall`**`(`*name*, *version=None*`)`

Instructs Chocolatey to uninstall a package.

name The name of the package to be uninstalled. Only accepts a single argument.

version Uninstalls a specific version of the package. Defaults to latest version installed.

CLI Example:

```
salt '*' chocolatey.uninstall <package name>
salt '*' chocolatey.uninstall <package name> version=<package version>
```

`salt.modules.chocolatey.`**`update`**`(`*name*, *source=None*, *pre_versions=False*`)`

Instructs Chocolatey to update packages on the system.

name The name of the package to update, or "all" to update everything installed on the system.

source Chocolatey repository (directory, share or remote URL feed) the package comes from. Defaults to the official Chocolatey feed.

pre_versions Include pre-release packages in comparison. Defaults to False.

CLI Example:

```
salt "*" chocolatey.update all
salt "*" chocolatey.update <package name> pre_versions=True
```

`salt.modules.chocolatey.`**`version`**`(`*name*, *check_remote=False*, *source=None*, *pre_versions=False*`)`

Instructs Chocolatey to check an installed package version, and optionally compare it to one available from a remote feed.

name The name of the package to check.

check_remote Get the version number of the latest package from the remote feed. Defaults to False.

source Chocolatey repository (directory, share or remote URL feed) the package comes from. Defaults to the official Chocolatey feed.

pre_versions Include pre-release packages in comparison. Defaults to False.

CLI Example:

```
salt "*" chocolatey.version <package name>
salt "*" chocolatey.version <package name> check_remote=True
```

61.16 salt.modules.cloud

Salt-specific interface for calling Salt Cloud directly

`salt.modules.cloud.`**`action`**(*fun=None*, *cloudmap=None*, *names=None*, *provider=None*, *instance=None*, ***kwargs*)

Execute a single action on the given provider/instance

CLI Example:

```
salt '*' cloud.action start instance=myinstance
salt '*' cloud.action stop instance=myinstance
salt '*' cloud.action show_image provider=my-ec2-config            image=ami-1624987f
```

`salt.modules.cloud.`**`create`**(*provider*, *names*, ***kwargs*)

Create an instance using Salt Cloud

CLI Example:

```
salt minionname cloud.create my-ec2-config myinstance            image=ami-1624987f size='Micr
```

`salt.modules.cloud.`**`destroy`**(*names*)

Destroy the named VM(s)

CLI Example:

```
salt '*' cloud.destroy myinstance
```

`salt.modules.cloud.`**`full_query`**(*query_type='list_nodes_full'*)

List all available cloud provider data

CLI Example:

```
salt '*' cloud.full_query
```

`salt.modules.cloud.`**`list_images`**(*provider='all'*)

List cloud provider images for the given providers

CLI Example:

```
salt '*' cloud.list_images my-gce-config
```

`salt.modules.cloud.`**`list_locations`**(*provider='all'*)

List cloud provider locations for the given providers

CLI Example:

```
salt '*' cloud.list_locations my-gce-config
```

`salt.modules.cloud.`**`list_sizes`**(*provider='all'*)

List cloud provider sizes for the given providers

CLI Example:

```
salt '*' cloud.list_sizes my-gce-config
```

salt.modules.cloud.**profile_**(*profile*, *names*, ***kwargs*)

Spin up an instance using Salt Cloud

CLI Example:

```
salt '*' cloud.profile my-gce-config myinstance
```

salt.modules.cloud.**query**(*query_type='list_nodes'*)

List cloud provider data for all providers

CLI Examples:

```
salt '*' cloud.query
salt '*' cloud.query list_nodes_full
salt '*' cloud.query list_nodes_select
```

salt.modules.cloud.**select_query**(*query_type='list_nodes_select'*)

List selected nodes

CLI Example:

```
salt '*' cloud.select_query
```

61.17 salt.modules.cmdmod

A module for shelling out

Keep in mind that this module is insecure, in that it can give whomever has access to the master root execution access to all salt minions

salt.modules.cmdmod.**exec_code**(*lang*, *code*, *cwd=None*)

Pass in two strings, the first naming the executable language, aka - python2, python3, ruby, perl, lua, etc. the second string containing the code you wish to execute. The stdout and stderr will be returned

CLI Example:

```
salt '*' cmd.exec_code ruby 'puts "cheese"'
```

salt.modules.cmdmod.**has_exec**(*cmd*)

Returns true if the executable is available on the minion, false otherwise

CLI Example:

```
salt '*' cmd.has_exec cat
```

salt.modules.cmdmod.**retcode**(*cmd*, *cwd=None*, *stdin=None*, *runas=None*, *shell='/bin/bash'*, *python_shell=True*, *env=None*, *clean_env=False*, *template=None*, *umask=None*, *output_loglevel='info'*, *quiet=False*, *timeout=None*, *reset_system_locale=True*, *ignore_retcode=False*, *saltenv='base'*, ***kwargs*)

Execute a shell command and return the command's return code.

Note that `env` represents the environment variables for the command, and should be formatted as a dict, or a YAML string which resolves to a dict.

CLI Example:

```
salt '*' cmd.retcode "file /bin/bash"
```

The template arg can be set to 'jinja' or another supported template engine to render the command arguments before execution. For example:

```
salt '*' cmd.retcode template=jinja "file {{grains.pythonpath[0]}}/python"
```

A string of standard input can be specified for the command to be run using the `stdin` parameter. This can be useful in cases where sensitive information must be read from standard input.:

```
salt '*' cmd.retcode "grep f" stdin='one\ntwo\nthree\nfour\nfive\n'
```

`salt.modules.cmdmod.`**`run`**(*cmd*, *cwd=None*, *stdin=None*, *runas=None*, *shell='/bin/bash'*, *python_shell=True*, *env=None*, *clean_env=False*, *template=None*, *rstrip=True*, *umask=None*, *output_loglevel='info'*, *quiet=False*, *timeout=None*, *reset_system_locale=True*, *ignore_retcode=False*, *saltenv='base'*, ***kwargs*)

Execute the passed command and return the output as a string

Note that `env` represents the environment variables for the command, and should be formatted as a dict, or a YAML string which resolves to a dict.

CLI Example:

```
salt '*' cmd.run "ls -l | awk '/foo/{print \$2}'"
```

The template arg can be set to 'jinja' or another supported template engine to render the command arguments before execution. For example:

```
salt '*' cmd.run template=jinja "ls -l /tmp/{{grains.id}} | awk '/foo/{print \$2}'"
```

Specify an alternate shell with the shell parameter:

```
salt '*' cmd.run "Get-ChildItem C:\ " shell='powershell'
```

A string of standard input can be specified for the command to be run using the `stdin` parameter. This can be useful in cases where sensitive information must be read from standard input.:

```
salt '*' cmd.run "grep f" stdin='one\ntwo\nthree\nfour\nfive\n'
```

`salt.modules.cmdmod.`**`run_all`**(*cmd*, *cwd=None*, *stdin=None*, *runas=None*, *shell='/bin/bash'*, *python_shell=True*, *env=None*, *clean_env=False*, *template=None*, *rstrip=True*, *umask=None*, *output_loglevel='info'*, *quiet=False*, *timeout=None*, *reset_system_locale=True*, *ignore_retcode=False*, *saltenv='base'*, ***kwargs*)

Execute the passed command and return a dict of return data

Note that `env` represents the environment variables for the command, and should be formatted as a dict, or a YAML string which resolves to a dict.

CLI Example:

```
salt '*' cmd.run_all "ls -l | awk '/foo/{print \$2}'"
```

The template arg can be set to 'jinja' or another supported template engine to render the command arguments before execution. For example:

```
salt '*' cmd.run_all template=jinja "ls -l /tmp/{{grains.id}} | awk '/foo/{print \$2}'"
```

A string of standard input can be specified for the command to be run using the `stdin` parameter. This can be useful in cases where sensitive information must be read from standard input.:

```
salt '*' cmd.run_all "grep f" stdin='one\ntwo\nthree\nfour\nfive\n'
```

`salt.modules.cmdmod.`**`run_stderr`**(*cmd*, *cwd=None*, *stdin=None*, *runas=None*, *shell='/bin/bash'*, *python_shell=True*, *env=None*, *clean_env=False*, *template=None*, *rstrip=True*, *umask=None*, *output_loglevel='info'*, *quiet=False*, *timeout=None*, *reset_system_locale=True*, *ignore_retcode=False*, *saltenv='base'*, ***kwargs*)

Execute a command and only return the standard error

Note that `env` represents the environment variables for the command, and should be formatted as a dict, or a YAML string which resolves to a dict.

CLI Example:

```
salt '*' cmd.run_stderr "ls -l | awk '/foo/{print \$2}'"
```

The template arg can be set to 'jinja' or another supported template engine to render the command arguments before execution. For example:

```
salt '*' cmd.run_stderr template=jinja "ls -l /tmp/{{grains.id}} | awk '/foo/{print \$2}'"
```

A string of standard input can be specified for the command to be run using the `stdin` parameter. This can be useful in cases where sensitive information must be read from standard input.:

```
salt '*' cmd.run_stderr "grep f" stdin='one\ntwo\nthree\nfour\nfive\n'
```

`salt.modules.cmdmod.`**`run_stdout`**(*cmd*, *cwd=None*, *stdin=None*, *runas=None*, *shell='/bin/bash'*, *python_shell=True*, *env=None*, *clean_env=False*, *template=None*, *rstrip=True*, *umask=None*, *output_loglevel='info'*, *quiet=False*, *timeout=None*, *reset_system_locale=True*, *ignore_retcode=False*, *saltenv='base'*, ***kwargs*)

Execute a command, and only return the standard out

Note that `env` represents the environment variables for the command, and should be formatted as a dict, or a YAML string which resolves to a dict.

CLI Example:

```
salt '*' cmd.run_stdout "ls -l | awk '/foo/{print \$2}'"
```

The template arg can be set to 'jinja' or another supported template engine to render the command arguments before execution. For example:

```
salt '*' cmd.run_stdout template=jinja "ls -l /tmp/{{grains.id}} | awk '/foo/{print \$2}'"
```

A string of standard input can be specified for the command to be run using the `stdin` parameter. This can be useful in cases where sensitive information must be read from standard input.:

```
salt '*' cmd.run_stdout "grep f" stdin='one\ntwo\nthree\nfour\nfive\n'
```

`salt.modules.cmdmod.`**`script`**(*source*, *args=None*, *cwd=None*, *stdin=None*, *runas=None*, *shell='/bin/bash'*, *python_shell=True*, *env=None*, *template='jinja'*, *umask=None*, *output_loglevel='info'*, *quiet=False*, *timeout=None*, *reset_system_locale=True*, *__env__=None*, *saltenv='base'*, ***kwargs*)

Download a script from a remote location and execute the script locally. The script can be located on the salt master file server or on an HTTP/FTP server.

The script will be executed directly, so it can be written in any available programming language.

The script can also be formated as a template, the default is jinja. Arguments for the script can be specified as well.

CLI Example:

```
salt '*' cmd.script salt://scripts/runme.sh
salt '*' cmd.script salt://scripts/runme.sh 'arg1 arg2 "arg 3"'
salt '*' cmd.script salt://scripts/windows_task.ps1 args=' -Input c:\tmp\infile.txt' shell='powe
```

A string of standard input can be specified for the command to be run using the `stdin` parameter. This can be useful in cases where sensitive information must be read from standard input.:

```
salt '*' cmd.script salt://scripts/runme.sh stdin='one\ntwo\nthree\nfour\nfive\n'
```

`salt.modules.cmdmod.`**`script_retcode`**(*source*, *cwd=None*, *stdin=None*, *runas=None*, *shell='/bin/bash'*, *python_shell=True*, *env=None*, *template='jinja'*, *umask=None*, *timeout=None*, *reset_system_locale=True*, *__env__=None*, *saltenv='base'*, ***kwargs*)

Download a script from a remote location and execute the script locally. The script can be located on the salt master file server or on an HTTP/FTP server.

The script will be executed directly, so it can be written in any available programming language.

The script can also be formated as a template, the default is jinja.

Only evaluate the script return code and do not block for terminal output

CLI Example:

```
salt '*' cmd.script_retcode salt://scripts/runme.sh
```

A string of standard input can be specified for the command to be run using the `stdin` parameter. This can be useful in cases where sensitive information must be read from standard input.:

```
salt '*' cmd.script_retcode salt://scripts/runme.sh stdin='one\ntwo\nthree\nfour\nfive\n'
```

`salt.modules.cmdmod.`**`which`**(*cmd*)

Returns the path of an executable available on the minion, None otherwise

CLI Example:

```
salt '*' cmd.which cat
```

`salt.modules.cmdmod.`**`which_bin`**(*cmds*)

Returns the first command found in a list of commands

CLI Example:

```
salt '*' cmd.which_bin '[pip2, pip, pip-python]'
```

61.18 salt.modules.composer

Use composer to install PHP dependencies for a directory

`salt.modules.composer.`**`install`**(*dir*, *composer=None*, *php=None*, *runas=None*, *prefer_source=None*, *prefer_dist=None*, *no_scripts=None*, *no_plugins=None*, *optimize=None*, *no_dev=None*, *quiet=False*, *composer_home='/root'*)

Install composer dependencies for a directory.

If composer has not been installed globally making it avaliable in the system PATH & making it executible, the `composer` and `php` parameters will need to be set to the location of the executables.

dir Directory location of the composer.json file.

composer Location of the composer.phar file. If not set composer will just execute "composer" as if it is installed globally. (i.e. /path/to/composer.phar)

php Location of the php executible to use with composer. (i.e. /usr/bin/php)

runas Which system user to run composer as.

prefer_source –prefer-source option of composer.

prefer_dist –prefer-dist option of composer.

no_scripts –no-scripts option of composer.

no_plugins –no-plugins option of composer.

optimize –optimize-autoloader option of composer. Recommended for production.

no_dev –no-dev option for composer. Recommended for production.

quiet –quiet option for composer. Whether or not to return output from composer.

composer_home $COMPOSER_HOME environment variable

CLI Example:

```
salt '*' composer.install /var/www/application

salt '*' composer.install /var/www/application             no_dev=True optimize=True
```

61.19 salt.modules.config

Return config information

`salt.modules.config.`**`backup_mode`**(*backup=''*)

Return the backup mode

CLI Example:

```
salt '*' config.backup_mode
```

`salt.modules.config.`**`dot_vals`**(*value*)

Pass in a configuration value that should be preceded by the module name and a dot, this will return a list of all read key/value pairs

CLI Example:

```
salt '*' config.dot_vals host
```

`salt.modules.config.`**`gather_bootstrap_script`**(*replace=False*)

Download the salt-bootstrap script, set replace to True to refresh the script if it has already been downloaded

CLI Example:

```
salt '*' config.gather_bootstrap_script True
```

`salt.modules.config.`**`get`**(*key*, *default=''*)

Attempt to retrieve the named value from opts, pillar, grains of the master config, if the named value is not available return the passed default. The default return is an empty string.

The value can also represent a value in a nested dict using a ":" delimiter for the dict. This means that if a dict looks like this:

```
{'pkg': {'apache': 'httpd'}}
```

To retrieve the value associated with the apache key in the pkg dict this key can be passed:

```
pkg:apache
```

This routine traverses these data stores in this order:

- Local minion config (opts)
- Minion's grains
- Minion's pillar
- Master config

CLI Example:

```
salt '*' config.get pkg:apache
```

salt.modules.config.**manage_mode**(*mode*)

Return a mode value, normalized to a string

CLI Example:

```
salt '*' config.manage_mode
```

salt.modules.config.**merge**(*value*, *default=''*, *omit_opts=False*, *omit_master=False*, *omit_pillar=False*)

Retrieves an option based on key, merging all matches.

Same as `option()` except that it merges all matches, rather than taking the first match.

CLI Example:

```
salt '*' config.merge schedule
```

salt.modules.config.**option**(*value*, *default=''*, *omit_opts=False*, *omit_master=False*, *omit_pillar=False*)

Pass in a generic option and receive the value that will be assigned

CLI Example:

```
salt '*' config.option redis.host
```

salt.modules.config.**valid_fileproto**(*uri*)

Returns a boolean value based on whether or not the URI passed has a valid remote file protocol designation

CLI Example:

```
salt '*' config.valid_fileproto salt://path/to/file
```

61.20 salt.modules.cp

Minion side functions for salt-cp

salt.modules.cp.**cache_dir**(*path*, *saltenv='base'*, *include_empty=False*, *env=None*)

Download and cache everything under a directory from the master

CLI Example:

```
salt '*' cp.cache_dir salt://path/to/dir
```

`salt.modules.cp.`**`cache_file`**(*path*, *saltenv='base'*, *env=None*)

Used to cache a single file in the local salt-master file cache.

CLI Example:

```
salt '*' cp.cache_file salt://path/to/file
```

`salt.modules.cp.`**`cache_files`**(*paths*, *saltenv='base'*, *env=None*)

Used to gather many files from the master, the gathered files will be saved in the minion cachedir reflective to the paths retrieved from the master.

CLI Example:

```
salt '*' cp.cache_files salt://pathto/file1,salt://pathto/file1
```

`salt.modules.cp.`**`cache_local_file`**(*path*)

Cache a local file on the minion in the localfiles cache

CLI Example:

```
salt '*' cp.cache_local_file /etc/hosts
```

`salt.modules.cp.`**`cache_master`**(*saltenv='base'*, *env=None*)

Retrieve all of the files on the master and cache them locally

CLI Example:

```
salt '*' cp.cache_master
```

`salt.modules.cp.`**`get_dir`**(*path*, *dest*, *saltenv='base'*, *template=None*, *gzip=None*, *env=None*)

Used to recursively copy a directory from the salt master

CLI Example:

```
salt '*' cp.get_dir salt://path/to/dir/ /minion/dest
```

get_dir supports the same template and gzip arguments as get_file.

`salt.modules.cp.`**`get_file`**(*path*, *dest*, *saltenv='base'*, *makedirs=False*, *template=None*, *gzip=None*, *env=None*)

Used to get a single file from the salt master

CLI Example:

```
salt '*' cp.get_file salt://path/to/file /minion/dest
```

Template rendering can be enabled on both the source and destination file names like so:

```
salt '*' cp.get_file "salt://{{grains.os}}/vimrc" /etc/vimrc template=jinja
```

This example would instruct all Salt minions to download the vimrc from a directory with the same name as their os grain and copy it to /etc/vimrc

For larger files, the cp.get_file module also supports gzip compression. Because gzip is CPU-intensive, this should only be used in scenarios where the compression ratio is very high (e.g. pretty-printed JSON or YAML files).

Use the *gzip* named argument to enable it. Valid values are 1..9, where 1 is the lightest compression and 9 the heaviest. 1 uses the least CPU on the master (and minion), 9 uses the most.

`salt.modules.cp.`**`get_file_str`**(*path*, *saltenv='base'*, *env=None*)

Return the contents of a file from a URL

CLI Example:

```
salt '*' cp.get_file_str salt://my/file
```

`salt.modules.cp.`**`get_template`**(*path*, *dest*, *template='jinja'*, *saltenv='base'*, *env=None*, ***kwargs*)

Render a file as a template before setting it down

CLI Example:

```
salt '*' cp.get_template salt://path/to/template /minion/dest
```

`salt.modules.cp.`**`get_url`**(*path*, *dest*, *saltenv='base'*, *env=None*)

Used to get a single file from a URL.

CLI Example:

```
salt '*' cp.get_url salt://my/file /tmp/mine
salt '*' cp.get_url http://www.slashdot.org /tmp/index.html
```

`salt.modules.cp.`**`hash_file`**(*path*, *saltenv='base'*, *env=None*)

Return the hash of a file, to get the hash of a file on the salt master file server prepend the path with salt://<file on server> otherwise, prepend the file with / for a local file.

CLI Example:

```
salt '*' cp.hash_file salt://path/to/file
```

`salt.modules.cp.`**`is_cached`**(*path*, *saltenv='base'*, *env=None*)

Return a boolean if the given path on the master has been cached on the minion

CLI Example:

```
salt '*' cp.is_cached salt://path/to/file
```

`salt.modules.cp.`**`list_master`**(*saltenv='base'*, *prefix='‘*, *env=None*)

List all of the files stored on the master

CLI Example:

```
salt '*' cp.list_master
```

`salt.modules.cp.`**`list_master_dirs`**(*saltenv='base'*, *prefix='‘*, *env=None*)

List all of the directories stored on the master

CLI Example:

```
salt '*' cp.list_master_dirs
```

`salt.modules.cp.`**`list_master_symlinks`**(*saltenv='base'*, *prefix='‘*, *env=None*)

List all of the symlinks stored on the master

CLI Example:

```
salt '*' cp.list_master_symlinks
```

`salt.modules.cp.`**`list_minion`**(*saltenv='base'*, *env=None*)

List all of the files cached on the minion

CLI Example:

```
salt '*' cp.list_minion
```

`salt.modules.cp.`**`list_states`**(*saltenv='base'*, *env=None*)

List all of the available state modules in an environment

CLI Example:

```
salt '*' cp.list_states
```

`salt.modules.cp.`**`push`**(*path*)

Push a file from the minion up to the master, the file will be saved to the salt master in the master's minion files cachedir (defaults to `/var/cache/salt/master/minions/minion-id/files`)

Since this feature allows a minion to push a file up to the master server it is disabled by default for security purposes. To enable, set `file_recv` to `True` in the master configuration file, and restart the master.

CLI Example:

```
salt '*' cp.push /etc/fstab
```

`salt.modules.cp.`**`recv`**(*files*, *dest*)

Used with salt-cp, pass the files dict, and the destination.

This function receives small fast copy files from the master via salt-cp. It does not work via the CLI.

61.21 salt.modules.cron

Work with cron

`salt.modules.cron.`**`list_tab`**(*user*)

Return the contents of the specified user's crontab

CLI Example:

```
salt '*' cron.list_tab root
```

`salt.modules.cron.`**`ls`**(*user*)

Return the contents of the specified user's crontab

CLI Example:

```
salt '*' cron.list_tab root
```

`salt.modules.cron.`**`raw_cron`**(*user*)

Return the contents of the user's crontab

CLI Example:

```
salt '*' cron.raw_cron root
```

`salt.modules.cron.`**`rm`**(*user*, *cmd*, *minute=None*, *hour=None*, *daymonth=None*, *month=None*, *dayweek=None*, *identifier=None*)

Remove a cron job for a specified user. If any of the day/time params are specified, the job will only be removed if the specified params match.

CLI Example:

```
salt '*' cron.rm_job root /usr/local/weekly
salt '*' cron.rm_job root /usr/bin/foo dayweek=1
```

`salt.modules.cron.`**`rm_env`**(*user*, *name*)

Remove cron environment variable for a specified user.

CLI Example:

```
salt '*' cron.rm_env root MAILTO
```

`salt.modules.cron.`**`rm_job`**(*user*, *cmd*, *minute=None*, *hour=None*, *daymonth=None*, *month=None*, *dayweek=None*, *identifier=None*)

Remove a cron job for a specified user. If any of the day/time params are specified, the job will only be removed if the specified params match.

CLI Example:

```
salt '*' cron.rm_job root /usr/local/weekly
salt '*' cron.rm_job root /usr/bin/foo dayweek=1
```

`salt.modules.cron.`**`set_env`**(*user*, *name*, *value=None*)

Set up an environment variable in the crontab.

CLI Example:

```
salt '*' cron.set_env root MAILTO user@example.com
```

`salt.modules.cron.`**`set_job`**(*user*, *minute*, *hour*, *daymonth*, *month*, *dayweek*, *cmd*, *comment*, *identifier=None*)

Sets a cron job up for a specified user.

CLI Example:

```
salt '*' cron.set_job root '*' '*' '*' '*' 1 /usr/local/weekly
```

`salt.modules.cron.`**`set_special`**(*user*, *special*, *cmd*)

Set up a special command in the crontab.

CLI Example:

```
salt '*' cron.set_special root @hourly 'echo foobar'
```

`salt.modules.cron.`**`write_cron_file`**(*user*, *path*)

Writes the contents of a file to a user's crontab

CLI Example:

```
salt '*' cron.write_cron_file root /tmp/new_cron
```

`salt.modules.cron.`**`write_cron_file_verbose`**(*user*, *path*)

Writes the contents of a file to a user's crontab and return error message on error

CLI Example:

```
salt '*' cron.write_cron_file_verbose root /tmp/new_cron
```

61.22 salt.modules.daemontools

daemontools service module. This module will create daemontools type service watcher.

This module is compatible with the `service` states, so it can be used to maintain services using the `provider` argument:

```
myservice:
  service:
    - running
    - provider: daemontools
```

`salt.modules.daemontools.`**`available`**`(`*name*`)`

Returns `True` if the specified service is available, otherwise returns `False`.

CLI Example:

```
salt '*' daemontools.available foo
```

`salt.modules.daemontools.`**`full_restart`**`(`*name*`)`

Calls daemontools.restart() function

CLI Example:

```
salt '*' daemontools.full_restart <service name>
```

`salt.modules.daemontools.`**`get_all`**`()`

Return a list of all available services

CLI Example:

```
salt '*' daemontools.get_all
```

`salt.modules.daemontools.`**`missing`**`(`*name*`)`

The inverse of daemontools.available. Returns `True` if the specified service is not available, otherwise returns `False`.

CLI Example:

```
salt '*' daemontools.missing foo
```

`salt.modules.daemontools.`**`reload_`**`(`*name*`)`

Wrapper for term()

CLI Example:

```
salt '*' daemontools.reload <service name>
```

`salt.modules.daemontools.`**`restart`**`(`*name*`)`

Restart service via daemontools. This will stop/start service

CLI Example:

```
salt '*' daemontools.restart <service name>
```

`salt.modules.daemontools.`**`start`**`(`*name*`)`

Starts service via daemontools

CLI Example:

```
salt '*' daemontools.start <service name>
```

`salt.modules.daemontools.`**`status`**`(`*name*, *sig=None*`)`

Return the status for a service via daemontools, return pid if running

CLI Example:

```
salt '*' daemontools.status <service name>
```

`salt.modules.daemontools.`**`stop`**`(`*name*`)`
Stops service via daemontools

CLI Example:

```
salt '*' daemontools.stop <service name>
```

`salt.modules.daemontools.`**`term`**`(`*name*`)`
Send a TERM to service via daemontools

CLI Example:

```
salt '*' daemontools.term <service name>
```

61.23 salt.modules.darwin_sysctl

Module for viewing and modifying sysctl parameters

`salt.modules.darwin_sysctl.`**`assign`**`(`*name*, *value*`)`
Assign a single sysctl parameter for this minion

CLI Example:

```
salt '*' sysctl.assign net.inet.icmp.icmplim 50
```

`salt.modules.darwin_sysctl.`**`get`**`(`*name*`)`
Return a single sysctl parameter for this minion

CLI Example:

```
salt '*' sysctl.get hw.physmem
```

`salt.modules.darwin_sysctl.`**`persist`**`(`*name*, *value*, *config='/etc/sysctl.conf'*`)`
Assign and persist a simple sysctl parameter for this minion

CLI Example:

```
salt '*' sysctl.persist net.inet.icmp.icmplim 50
salt '*' sysctl.persist coretemp_load NO config=/etc/sysctl.conf
```

`salt.modules.darwin_sysctl.`**`show`**`()`
Return a list of sysctl parameters for this minion

CLI Example:

```
salt '*' sysctl.show
```

61.24 salt.modules.data

Manage a local persistent data structure that can hold any arbitrary data specific to the minion

`salt.modules.data.`**`cas`**`(`*key*, *value*, *old_value*`)`
Check and set a value in the minion datastore

CLI Example:

```
salt '*' data.cas <key> <value> <old_value>
```

`salt.modules.data.`**`clear`**`()`

Clear out all of the data in the minion datastore, this function is destructive!

CLI Example:

```
salt '*' data.clear
```

`salt.modules.data.`**`dump`**`(`*new_data*`)`

Replace the entire datastore with a passed data structure

CLI Example:

```
salt '*' data.dump '{'eggs': 'spam'}'
```

`salt.modules.data.`**`getval`**`(`*key*`)`

Get a value from the minion datastore

CLI Example:

```
salt '*' data.getval <key>
```

`salt.modules.data.`**`getvals`**`(`**keys*`)`

Get values from the minion datastore

CLI Example:

```
salt '*' data.getvals <key> [<key> ...]
```

`salt.modules.data.`**`load`**`()`

Return all of the data in the minion datastore

CLI Example:

```
salt '*' data.load
```

`salt.modules.data.`**`update`**`(`*key*, *value*`)`

Update a key with a value in the minion datastore

CLI Example:

```
salt '*' data.update <key> <value>
```

61.25 salt.modules.ddns

Support for RFC 2136 dynamic DNS updates.

depends

- dnspython Python module

configuration If you want to use TSIG authentication for the server, there are a couple of optional configuration parameters made available to support this (the keyname is only needed if the keyring contains more than one key):

```
ddns.keyring: keyring file (default=None)
ddns.keyname: key name in file (default=None)
```

The keyring file needs to be in json format and the key name needs to end with an extra period in the file, similar to this:

```
{'keyname.': 'keycontent'}
```

`salt.modules.ddns.`**`add_host`**(*zone*, *name*, *ttl*, *ip*, *nameserver='127.0.0.1'*, *replace=True*, ***kwargs*)

Add, replace, or update the A and PTR (reverse) records for a host.

CLI Example:

```
salt ns1 ddns.add_host example.com host1 60 10.1.1.1
```

`salt.modules.ddns.`**`delete`**(*zone*, *name*, *rdtype=None*, *data=None*, *nameserver='127.0.0.1'*, ***kwargs*)

Delete a DNS record.

CLI Example:

```
salt ns1 ddns.delete example.com host1 A
```

`salt.modules.ddns.`**`delete_host`**(*zone*, *name*, *nameserver='127.0.0.1'*, ***kwargs*)

Delete the forward and reverse records for a host.

Returns true if any records are deleted.

CLI Example:

```
salt ns1 ddns.delete_host example.com host1
```

`salt.modules.ddns.`**`update`**(*zone*, *name*, *ttl*, *rdtype*, *data*, *nameserver='127.0.0.1'*, *replace=False*, ***kwargs*)

Add, replace, or update a DNS record. nameserver must be an IP address and the minion running this module must have update privileges on that server. If replace is true, first deletes all records for this name and type.

CLI Example:

```
salt ns1 ddns.update example.com host1 60 A 10.0.0.1
```

61.26 salt.modules.debconfmod

Support for Debconf

`salt.modules.debconfmod.`**`get_selections`**(*fetchempty=True*)

Answers to debconf questions for all packages in the following format:

```
{'package': [['question', 'type', 'value'], ...]}
```

CLI Example:

```
salt '*' debconf.get_selections
```

`salt.modules.debconfmod.`**`set_`**(*package*, *question*, *type*, *value*, **extra*)

Set answers to debconf questions for a package.

CLI Example:

```
salt '*' debconf.set <package> <question> <type> <value> [<value> ...]
```

`salt.modules.debconfmod.`**`set_file`**(*path*, *saltenv='base'*, ***kwargs*)

Set answers to debconf questions from a file.

CLI Example:

```
salt '*' debconf.set_file salt://pathto/pkg.selections
```

`salt.modules.debconfmod.`**`show`**`(`*name*`)`

Answers to debconf questions for a package in the following format:

```
[['question', 'type', 'value'], ...]
```

If debconf doesn't know about a package, we return None.

CLI Example:

```
salt '*' debconf.show <package name>
```

61.27 salt.modules.debian_ip

The networking module for Debian based distros

`salt.modules.debian_ip.`**`apply_network_settings`**`(`***settings*`)`

Apply global network configuration.

CLI Example:

```
salt '*' ip.apply_network_settings
```

`salt.modules.debian_ip.`**`build_bond`**`(`*iface*, ***settings*`)`

Create a bond script in /etc/modprobe.d with the passed settings and load the bonding kernel module.

CLI Example:

```
salt '*' ip.build_bond bond0 mode=balance-alb
```

`salt.modules.debian_ip.`**`build_interface`**`(`*iface*, *iface_type*, *enabled*, ***settings*`)`

Build an interface script for a network interface.

CLI Example:

```
salt '*' ip.build_interface eth0 eth <settings>
```

`salt.modules.debian_ip.`**`build_network_settings`**`(`***settings*`)`

Build the global network script.

CLI Example:

```
salt '*' ip.build_network_settings <settings>
```

`salt.modules.debian_ip.`**`build_routes`**`(`*iface*, ***settings*`)`

Add route scripts for a network interface using up commands.

CLI Example:

```
salt '*' ip.build_routes eth0 <settings>
```

`salt.modules.debian_ip.`**`down`**`(`*iface*, *iface_type*`)`

Shutdown a network interface

CLI Example:

```
salt '*' ip.down eth0
```

`salt.modules.debian_ip.`**`get_bond`**`(`*iface*`)`
Return the content of a bond script

CLI Example:

```
salt '*' ip.get_bond bond0
```

`salt.modules.debian_ip.`**`get_interface`**`(`*iface*`)`
Return the contents of an interface script

CLI Example:

```
salt '*' ip.get_interface eth0
```

`salt.modules.debian_ip.`**`get_network_settings`**`()`
Return the contents of the global network script.

CLI Example:

```
salt '*' ip.get_network_settings
```

`salt.modules.debian_ip.`**`get_routes`**`(`*iface*`)`
Return the routes for the interface

CLI Example:

```
salt '*' ip.get_interface eth0
```

`salt.modules.debian_ip.`**`up`**`(`*iface*, *iface_type*`)`
Start up a network interface

CLI Example:

```
salt '*' ip.up eth0
```

61.28 salt.modules.debian_service

Service support for Debian systems (uses update-rc.d and /sbin/service)

`salt.modules.debian_service.`**`available`**`(`*name*`)`
Returns `True` if the specified service is available, otherwise returns `False`.

CLI Example:

```
salt '*' service.available sshd
```

`salt.modules.debian_service.`**`disable`**`(`*name*, ***kwargs*`)`
Disable the named service to start at boot

CLI Example:

```
salt '*' service.disable <service name>
```

`salt.modules.debian_service.`**`disabled`**`(`*name*`)`
Return True if the named service is enabled, false otherwise

CLI Example:

```
salt '*' service.disabled <service name>
```

`salt.modules.debian_service.`**`enable`**`(`*name*, ***kwargs*`)`
Enable the named service to start at boot

CLI Example:

```
salt '*' service.enable <service name>
```

`salt.modules.debian_service.`**`enabled`**`(`*name*`)`
Return True if the named service is enabled, false otherwise

CLI Example:

```
salt '*' service.enabled <service name>
```

`salt.modules.debian_service.`**`force_reload`**`(`*name*`)`
Force-reload the named service

CLI Example:

```
salt '*' service.force_reload <service name>
```

`salt.modules.debian_service.`**`get_all`**`()`
Return all available boot services

CLI Example:

```
salt '*' service.get_all
```

`salt.modules.debian_service.`**`get_disabled`**`()`
Return a set of services that are installed but disabled

CLI Example:

```
salt '*' service.get_disabled
```

`salt.modules.debian_service.`**`get_enabled`**`()`
Return a list of service that are enabled on boot

CLI Example:

```
salt '*' service.get_enabled
```

`salt.modules.debian_service.`**`missing`**`(`*name*`)`
The inverse of service.available. Returns `True` if the specified service is not available, otherwise returns `False`.

CLI Example:

```
salt '*' service.missing sshd
```

`salt.modules.debian_service.`**`reload_`**`(`*name*`)`
Reload the named service

CLI Example:

```
salt '*' service.reload <service name>
```

`salt.modules.debian_service.`**`restart`**`(`*name*`)`
Restart the named service

CLI Example:

```
salt '*' service.restart <service name>
```

`salt.modules.debian_service.`**`start`**(*name*)

Start the specified service

CLI Example:

```
salt '*' service.start <service name>
```

`salt.modules.debian_service.`**`status`**(*name*, *sig=None*)

Return the status for a service, pass a signature to use to find the service via ps

CLI Example:

```
salt '*' service.status <service name>
```

`salt.modules.debian_service.`**`stop`**(*name*)

Stop the specified service

CLI Example:

```
salt '*' service.stop <service name>
```

61.29 salt.modules.dig

Compendium of generic DNS utilities

`salt.modules.dig.`**`A`**(*host*, *nameserver=None*)

Return the A record for `host`.

Always returns a list.

CLI Example:

```
salt ns1 dig.A www.google.com
```

`salt.modules.dig.`**`AAAA`**(*host*, *nameserver=None*)

Return the AAAA record for `host`.

Always returns a list.

CLI Example:

```
salt ns1 dig.AAAA www.google.com
```

`salt.modules.dig.`**`MX`**(*domain*, *resolve=False*, *nameserver=None*)

Return a list of lists for the MX of `domain`.

If the `resolve` argument is True, resolve IPs for the servers.

It's limited to one IP, because although in practice it's very rarely a round robin, it is an acceptable configuration and pulling just one IP lets the data be similar to the non-resolved version. If you think an MX has multiple IPs, don't use the resolver here, resolve them in a separate step.

CLI Example:

```
salt ns1 dig.MX google.com
```

`salt.modules.dig.`**`NS`**(*domain*, *resolve=True*, *nameserver=None*)

Return a list of IPs of the nameservers for `domain`

If `resolve` is False, don't resolve names.

CLI Example:

```
salt ns1 dig.NS google.com
```

`salt.modules.dig.`**`SPF`**(*domain*, *record='SPF'*, *nameserver=None*)

Return the allowed IPv4 ranges in the SPF record for `domain`.

If record is `SPF` and the SPF record is empty, the TXT record will be searched automatically. If you know the domain uses TXT and not SPF, specifying that will save a lookup.

CLI Example:

```
salt ns1 dig.SPF google.com
```

`salt.modules.dig.`**`check_ip`**(*addr*)

Check if address is a valid IP. returns True if valid, otherwise False.

CLI Example:

```
salt ns1 dig.check_ip 127.0.0.1
salt ns1 dig.check_ip 1111:2222:3333:4444:5555:6666:7777:8888
```

61.30 salt.modules.disk

Module for gathering disk information

`salt.modules.disk.`**`inodeusage`**(*args=None*)

Return inode usage information for volumes mounted on this minion

CLI Example:

```
salt '*' disk.inodeusage
```

`salt.modules.disk.`**`percent`**(*args=None*)

Return partion information for volumes mounted on this minion

CLI Example:

```
salt '*' disk.percent /var
```

`salt.modules.disk.`**`usage`**(*args=None*)

Return usage information for volumes mounted on this minion

CLI Example:

```
salt '*' disk.usage
```

61.31 salt.modules.djangomod

Manage Django sites

`salt.modules.djangomod.`**`collectstatic`**(*settings_module*, *bin_env=None*, *no_post_process=False*, *ignore=None*, *dry_run=False*, *clear=False*, *link=False*, *no_default_ignore=False*, *pythonpath=None*, *env=None*)

Collect static files from each of your applications into a single location that can easily be served in production.

CLI Example:

```
salt '*' django.collectstatic <settings_module>
```

`salt.modules.djangomod.`**`command`**`(`*settings_module*, *command*, *bin_env=None*, *pythonpath=None*, *env=None*, **args*, ***kwargs*`)`

Run arbitrary django management command

CLI Example:

```
salt '*' django.command <settings_module> <command>
```

`salt.modules.djangomod.`**`createsuperuser`**`(`*settings_module*, *username*, *email*, *bin_env=None*, *database=None*, *pythonpath=None*, *env=None*`)`

Create a super user for the database. This function defaults to use the `--noinput` flag which prevents the creation of a password for the superuser.

CLI Example:

```
salt '*' django.createsuperuser <settings_module> user user@example.com
```

`salt.modules.djangomod.`**`loaddata`**`(`*settings_module*, *fixtures*, *bin_env=None*, *database=None*, *pythonpath=None*, *env=None*`)`

Load fixture data

Fixtures: comma separated list of fixtures to load

CLI Example:

```
salt '*' django.loaddata <settings_module> <comma delimited list of fixtures>
```

`salt.modules.djangomod.`**`syncdb`**`(`*settings_module*, *bin_env=None*, *migrate=False*, *database=None*, *pythonpath=None*, *env=None*, *noinput=True*`)`

Run syncdb

Execute the Django-Admin syncdb command, if South is available on the minion the `migrate` option can be passed as `True` calling the migrations to run after the syncdb completes

CLI Example:

```
salt '*' django.syncdb <settings_module>
```

61.32 salt.modules.dnsmasq

Module for managing dnqmasq

`salt.modules.dnsmasq.`**`fullversion`**`()`

Shows installed version of dnsmasq, and compile options

CLI Example:

```
salt '*' dnsmasq.version
```

`salt.modules.dnsmasq.`**`get_config`**`(`*config_file='/etc/dnsmasq.conf'*`)`

Dumps all options from the config file

CLI Examples:

```
salt '*' dnsmasq.get_config
salt '*' dnsmasq.get_config file=/etc/dnsmasq.conf
```

`salt.modules.dnsmasq.`**`set_config`**(*config_file='/etc/dnsmasq.conf'*, *follow=True*, ***kwargs*)

Sets a value or a set of values in the specified file. By default, if conf-dir is configured in this file, salt will attempt to set the option in any file inside the conf-dir where it has already been enabled. If it does not find it inside any files, it will append it to the main config file. Setting follow to False will turn off this behavior.

If a config option currently appears multiple times (such as dhcp-host, which is specified at least once per host), the new option will be added to the end of the main config file (and not to any includes). If you need an option added to a specific include file, specify it as the config_file.

CLI Examples:

```
salt '*' dnsmasq.set_config domain=mydomain.com
salt '*' dnsmasq.set_config follow=False domain=mydomain.com
salt '*' dnsmasq.set_config file=/etc/dnsmasq.conf domain=mydomain.com
```

`salt.modules.dnsmasq.`**`version`**()

Shows installed version of dnsmasq

CLI Example:

```
salt '*' dnsmasq.version
```

61.33 salt.modules.dnsutil

Compendium of generic DNS utilities

`salt.modules.dnsutil.`**`A`**(*host*, *nameserver=None*)

Return the A record for 'host'.

Always returns a list.

CLI Example:

```
salt ns1 dig.A www.google.com
```

`salt.modules.dnsutil.`**`MX`**(*domain*, *resolve=False*, *nameserver=None*)

Return a list of lists for the MX of `domain`.

If the 'resolve' argument is True, resolve IPs for the servers.

It's limited to one IP, because although in practice it's very rarely a round robin, it is an acceptable configuration and pulling just one IP lets the data be similar to the non-resolved version. If you think an MX has multiple IPs, don't use the resolver here, resolve them in a separate step.

CLI Example:

```
salt ns1 dig.MX google.com
```

`salt.modules.dnsutil.`**`NS`**(*domain*, *resolve=True*, *nameserver=None*)

Return a list of IPs of the nameservers for `domain`

If 'resolve' is False, don't resolve names.

CLI Example:

```
salt ns1 dig.NS google.com
```

`salt.modules.dnsutil.`**`SPF`**(*domain*, *record='SPF'*, *nameserver=None*)

Return the allowed IPv4 ranges in the SPF record for `domain`.

If record is `SPF` and the SPF record is empty, the TXT record will be searched automatically. If you know the domain uses TXT and not SPF, specifying that will save a lookup.

CLI Example:

```
salt ns1 dig.SPF google.com
```

`salt.modules.dnsutil.`**`check_ip`**(*ip_addr*)

Check that string ip_addr is a valid IP

CLI Example:

```
salt ns1 dig.check_ip 127.0.0.1
```

`salt.modules.dnsutil.`**`hosts_append`**(*hostsfile='/etc/hosts'*, *ip_addr=None*, *entries=None*)

Append a single line to the /etc/hosts file.

CLI Example:

```
salt '*' dnsutil.hosts_append /etc/hosts 127.0.0.1 ad1.yuk.co,ad2.yuk.co
```

`salt.modules.dnsutil.`**`hosts_remove`**(*hostsfile='/etc/hosts'*, *entries=None*)

Remove a host from the /etc/hosts file. If doing so will leave a line containing only an IP address, then the line will be deleted. This function will leave comments and blank lines intact.

CLI Examples:

```
salt '*' dnsutil.hosts_remove /etc/hosts ad1.yuk.co
salt '*' dnsutil.hosts_remove /etc/hosts ad2.yuk.co,ad1.yuk.co
```

`salt.modules.dnsutil.`**`parse_hosts`**(*hostsfile='/etc/hosts'*, *hosts=None*)

Parse /etc/hosts file.

CLI Example:

```
salt '*' dnsutil.parse_hosts
```

`salt.modules.dnsutil.`**`parse_zone`**(*zonefile=None*, *zone=None*)

Parses a zone file. Can be passed raw zone data on the API level.

CLI Example:

```
salt ns1 dnsutil.parse_zone /var/lib/named/example.com.zone
```

61.34 salt.modules.dockerio

61.34.1 Management of dockers

New in version 2014.1.0: (Hydrogen)

Note: The DockerIO integration is still in beta; the API is subject to change

---

General notes

- As we use states, we don't want to pop continuously dockers, we will map each container id (or image) with a grain whenever it is relevant.

- As a corollary, we will resolve for a container id either directly this container id or try to find a container id matching something stocked in grain

Installation prerequisites

- You will need the 'docker-py' python package in your python installation running salt. The version of docker-py should support version 1.6 of docker remote API..
- For now, you need docker-py from sources:

  https://github.com/dotcloud/docker-py

Prerequisite pillar configuration for authentication

- To push or pull you will need to be authenticated as the docker-py bindings require it
- For this to happen, you will need to configure a mapping in the pillar representing your per URL authentication bits:

  ```
  docker-registries:
      registry_url:
          email: foo@foo.com
          password: s3cr3t
          username: foo
  ```

- You need at least an entry to the default docker index:

  ```
  docker-registries:
      https://index.docker.io/v1:
          email: foo@foo.com
          password: s3cr3t
          username: foo
  ```

you can define multiple registries blocks for them to be aggregated, their id just must finish with -docker-registries:

```
ac-docker-registries:
     https://index.bar.io/v1:
         email: foo@foo.com
         password: s3cr3t
         username: foo

ab-docker-registries:
     https://index.foo.io/v1:
         email: foo@foo.com
         password: s3cr3t
         username: foo
```

Would be the equivalent to:

```
docker-registries:
     https://index.bar.io/v1:
         email: foo@foo.com
         password: s3cr3t
         username: foo
     https://index.foo.io/v1:
         email: foo@foo.com
         password: s3cr3t
         username: foo
```

Registry dialog methods

- login
- push
- pull

Docker management

- version
- info

Image management

You have those methods:

- search
- inspect_image
- get_images
- remove_image
- import_image
- build
- tag

Container management

You have those methods:

- start
- stop
- restart
- kill
- wait
- get_containers
- inspect_container
- remove_container
- is_running
- top
- ports
- logs
- diff
- commit

- create_container
- export
- get_container_root

Runtime execution within a specific already existing and running container

- Idea is to use lxc-attach to execute inside the container context.
- We do not use a "docker run command" but want to execute something inside a running container.

You have those methods:

- retcode
- run
- run_all
- run_stderr
- run_stdout
- script
- script_retcode

`salt.modules.dockerio.`**`build`**(*path=None*, *tag=None*, *quiet=False*, *fileobj=None*, *nocache=False*, *rm=True*, *timeout=None*, **args*, ***kwargs*)

Build a docker image from a dockerfile or an URL

You can either:

- give the url/branch/docker_dir
- give a path on the file system

path URL or path in the filesystem to the dockerfile

tag Tag of the image

quiet quiet mode

nocache do not use docker image cache

rm remove intermediate commits

timeout timeout is seconds before aborting

CLI Example:

```
salt '*' docker.build
```

`salt.modules.dockerio.`**`commit`**(*container*, *repository=None*, *tag=None*, *message=None*, *author=None*, *conf=None*, **args*, ***kwargs*)

Commit a container (promotes it to an image)

container container id

repository repository/imageName to commit to

tag optional tag

message optional commit message

author optional author

conf optional conf

CLI Example:

```
salt '*' docker.commit <container id>
```

`salt.modules.dockerio.`**`create_container`**(*image*, *command=None*, *hostname=None*, *user=None*, *detach=True*, *stdin_open=False*, *tty=False*, *mem_limit=0*, *ports=None*, *environment=None*, *dns=None*, *volumes=None*, *volumes_from=None*, *name=None*, **args*, ***kwargs*)

Create a new container

image image to create the container from

command command to execute while starting

hostname hostname of the container

user user to run docker as

detach daemon mode

environment environment variable mapping ({'foo':'BAR'})

dns list of DNS servers

ports ports redirections ({'222': {}})

volumes list of volumes mapping:

```
(['/mountpoint/in/container:/guest/foo',
  '/same/path/mounted/point'])
```

tty attach ttys

stdin_open let stdin open

volumes_from container to get volumes definition from

name name given to container

EG:

> salt-call docker.create_container o/ubuntu volumes="['/s','/m:/f']"

CLI Example:

```
salt '*' docker.create_container <image>
```

`salt.modules.dockerio.`**`diff`**(*container*, **args*, ***kwargs*)

Get container diffs

container container id

CLI Example:

```
salt '*' docker.diff <container id>
```

`salt.modules.dockerio.`**`exists`**(*container*, **args*, ***kwargs*)

Check if a given container exists

Parameters **container** (*string*) – Container id

Return type boolean:

CLI Example:

```
salt '*' docker.exists <container>
```

`salt.modules.dockerio.`**`export`**(*container*, *path*, **args*, ***kwargs*)
Export a container to a file

container container id

path path to the export

CLI Example:

```
salt '*' docker.export <container id>
```

`salt.modules.dockerio.`**`get_container_root`**(*container*)
Get the container rootfs path

container container id or grain

CLI Example:

```
salt '*' docker.get_container_root <container id>
```

`salt.modules.dockerio.`**`get_containers`**(*all=True*, *trunc=False*, *since=None*, *before=None*, *limit=-1*, **args*, ***kwargs*)
Get a list of mappings representing all containers

all Return all containers

trunc Set it to True to have the short ID

Returns a mapping of something which looks like container

CLI Example:

```
salt '*' docker.get_containers
```

`salt.modules.dockerio.`**`get_images`**(*name=None*, *quiet=False*, *all=True*, **args*, ***kwargs*)
List docker images

Parameters

- **name** (*string*) – A repository name to filter on
- **quiet** (*boolean*) – Only show image ids
- **all** (*boolean*) – Show all images

Return type dict

Returns A status message with the command output

CLI Example:

```
salt '*' docker.get_images [name] [quiet=True|False] [all=True|False]
```

`salt.modules.dockerio.`**`import_image`**(*src*, *repo*, *tag=None*, **args*, ***kwargs*)
Import content from a local tarball or a URL to a docker image

Parameters

- **src** (*string*) – The content to import (URL, absolute path to a tarball)
- **repo** (*string*) – The repository to import to
- **tag** (*string*) – An optional tag to set

CLI Example:

```
salt '*' docker.import_image <src> <repo> [tag]
```

`salt.modules.dockerio.`**`info`**(**args*, ***kwargs*)

Get the version information about docker

Return type dict

Returns A status message with the command output

CLI Example:

```
salt '*' docker.info
```

`salt.modules.dockerio.`**`inspect_container`**(*container*, **args*, ***kwargs*)

Get container information. This is similar to the docker inspect command.

Parameters **container** (*string*) – The id of the container to inspect

Return type dict

Returns A status message with the command output

CLI Example:

```
salt '*' docker.inspect_container <container>
```

`salt.modules.dockerio.`**`inspect_image`**(*image*, **args*, ***kwargs*)

Inspect the status of an image and return relative data

CLI Example:

```
salt '*' docker.inspect_image <image>
```

`salt.modules.dockerio.`**`invalid`**(*m*, *id=<object object at 0x7f223274a430>*, *comment='We did not get any expectable answer from docker'*, *out=None*)

Return invalid status

`salt.modules.dockerio.`**`is_running`**(*container*, **args*, ***kwargs*)

Is this container running

container Container id

Return boolean

CLI Example:

```
salt '*' docker.is_running <container id>
```

`salt.modules.dockerio.`**`kill`**(*container*, **args*, ***kwargs*)

Kill a running container

Parameters **container** (*string*) – The container id to kill

Return type dict

Returns

A status message with the command output ex:

```
 {'id': 'abcdef123456789',
'status': True}
```

CLI Example:

```
salt '*' docker.kill <container id>
```

`salt.modules.dockerio.`**`login`**(*url=None*, *username=None*, *password=None*, *email=None*, **args*, ***kwargs*)

Wrapper to the docker.py login method, does not do much yet

CLI Example:

```
salt '*' docker.login <container id>
```

`salt.modules.dockerio.`**`logs`**(*container*, **args*, ***kwargs*)

Return logs for a specified container

container container id

CLI Example:

```
salt '*' docker.logs <container id>
```

`salt.modules.dockerio.`**`port`**(*container*, *private_port*, **args*, ***kwargs*)

Private/Public for a specific port mapping allocation information This method is broken on docker-py side Just use the result of inspect to mangle port allocation

container container id

private_port private port on the container to query for

CLI Example:

```
salt '*' docker.port <container id>
```

`salt.modules.dockerio.`**`pull`**(*repo*, *tag=None*, **args*, ***kwargs*)

Pulls an image from any registry. See above documentation for how to configure authenticated access.

Parameters

- **repo** (*string*) – The repository to pull. [registryurl://]REPOSITORY_NAME_image eg:

  ```
  index.docker.io:MyRepo/image
  superaddress.cdn:MyRepo/image
  MyRepo/image
  ```

- **tag** (*string*) – The specific tag to pull

Return type dict

Returns

A status message with the command output Example:

```
----------
comment:
    Image NAME was pulled (ID
id:
    None
out:
    ----------
    - id:
        2c80228370c9
    - status:
        Download complete
    ----------
    - id:
```

```
                2c80228370c9
            - progress:
                [=========================>                              ]
            - status:
                Downloading
            ----------
            - id:
                2c80228370c9
            - status
                Pulling image (latest) from foo/ubuntubox
            ----------
            - status:
                Pulling repository foo/ubuntubox
        status:
            True
```

CLI Example:

```
salt '*' docker.pull <repository> [tag]
```

`salt.modules.dockerio.`**`push`**(*repo*, **args*, ***kwargs*)

Pushes an image from any registry See this top level documentation to know how to configure authenticated access

repo [registryurl://]REPOSITORY_NAME_image eg:

```
index.docker.io:MyRepo/image
superaddress.cdn:MyRepo/image
MyRepo/image
```

CLI Example:

```
salt '*' docker.push <repo>
```

`salt.modules.dockerio.`**`remove_container`**(*container=None*, *force=False*, *v=False*, **args*, ***kwargs*)

Removes a container from a docker installation

container Container id to remove

force By default, do not remove a running container, set this to remove it unconditionally

v verbose mode

Return True or False in the status mapping and also any information about docker in status['out']

CLI Example:

```
salt '*' docker.remove_container <container id>
```

`salt.modules.dockerio.`**`remove_image`**(*image*, **args*, ***kwargs*)

Remove an image from a system.

Parameters **image** (*string*) – The image to remove

Return type string

Returns A status message.

CLI Example:

```
salt '*' docker.remove_image <image>
```

`salt.modules.dockerio.`**`restart`**(*container*, *timeout=10*, **args*, ***kwargs*)

Restart a running container

Parameters

- **container** (*string*) – The container id to restart
- **timeout** – Wait for a timeout to let the container exit gracefully before killing it

Return type dict

Returns

A status message with the command output ex:

```
 {'id': 'abcdef123456789',
'status': True}
```

CLI Example:

```
salt '*' docker.restart <container id>
```

`salt.modules.dockerio.`**`retcode`**(*container*, *cmd*, **args*, ***kwargs*)

Wrapper for cmdmod.retcode inside a container context

container container id (or grain)

Other params: See cmdmod documentation

The return is a bit different as we use the docker struct, The output of the command is in 'out' The result is false if command failed

WARNING: Be advised that this function allows for raw shell access to the named container! If allowing users to execute this directly it may allow more rights than intended!

CLI Example:

```
salt '*' docker.retcode <container id> 'ls -l /etc'
```

`salt.modules.dockerio.`**`run`**(*container*, *cmd*, **args*, ***kwargs*)

Wrapper for cmdmod.run inside a container context

container container id (or grain)

Other params: See cmdmod documentation

The return is a bit different as we use the docker struct, The output of the command is in 'out' The result is always True

WARNING: Be advised that this function allows for raw shell access to the named container! If allowing users to execute this directly it may allow more rights than intended!

CLI Example:

```
salt '*' docker.run <container id> 'ls -l /etc'
```

`salt.modules.dockerio.`**`run_all`**(*container*, *cmd*, **args*, ***kwargs*)

Wrapper for cmdmod.run_all inside a container context

container container id (or grain)

Other params: See cmdmod documentation

The return is a bit different as we use the docker struct, The output of the command is in 'out' The result if false if command failed

WARNING: Be advised that this function allows for raw shell access to the named container! If allowing users to execute this directly it may allow more rights than intended!

CLI Example:

```
salt '*' docker.run_all <container id> 'ls -l /etc'
```

`salt.modules.dockerio.`**`run_stderr`**(*container*, *cmd*, **args*, ***kwargs*)

Wrapper for cmdmod.run_stderr inside a container context

container container id (or grain)

Other params: See cmdmod documentation

The return is a bit different as we use the docker struct, The output of the command is in 'out' The result is always True

WARNING: Be advised that this function allows for raw shell access to the named container! If allowing users to execute this directly it may allow more rights than intended!

CLI Example:

```
salt '*' docker.run_stderr <container id> 'ls -l /etc'
```

`salt.modules.dockerio.`**`run_stdout`**(*container*, *cmd*, **args*, ***kwargs*)

Wrapper for cmdmod.run_stdout inside a container context

container container id (or grain)

Other params: See cmdmod documentation

The return is a bit different as we use the docker struct, The output of the command is in 'out' The result is always True

WARNING: Be advised that this function allows for raw shell access to the named container! If allowing users to execute this directly it may allow more rights than intended!

CLI Example:

```
salt '*' docker.run_stdout <container id> 'ls -l /etc'
```

`salt.modules.dockerio.`**`script`**(*container*, *source*, *args=None*, *cwd=None*, *stdin=None*, *runas=None*, *shell='/bin/bash'*, *env=None*, *template='jinja'*, *umask=None*, *timeout=None*, *reset_system_locale=True*, *no_clean=False*, *saltenv='base'*, **nargs*, ***kwargs*)

Same usage as cmd.script but running inside a container context

container container id or grain

others params and documentation See cmd.retcode

WARNING: Be advised that this function allows for raw shell access to the named container! If allowing users to execute this directly it may allow more rights than intended!

CLI Example:

```
salt '*' docker.script <container id> salt://docker_script.py
```

`salt.modules.dockerio.`**`script_retcode`**(*container*, *source*, *cwd=None*, *stdin=None*, *runas=None*, *shell='/bin/bash'*, *env=None*, *template='jinja'*, *umask=None*, *timeout=None*, *reset_system_locale=True*, *no_clean=False*, *saltenv='base'*, **args*, ***kwargs*)

Same usage as cmd.script_retcode but running inside a container context

container container id or grain

others params and documentation See cmd.retcode

WARNING: Be advised that this function allows for raw shell access to the named container! If allowing users to execute this directly it may allow more rights than intended!

CLI Example:

```
salt '*' docker.script_retcode <container id> salt://docker_script.py
```

`salt.modules.dockerio.`**`search`**(*term*, **args*, ***kwargs*)

Search for an image on the registry

Parameters term (*string*) – The search keyword to query

CLI Example:

```
salt '*' docker.search <term>
```

`salt.modules.dockerio.`**`start`**(*container*, *binds=None*, *ports=None*, *port_bindings=None*, *lxc_conf=None*, *publish_all_ports=None*, *links=None*, *privileged=False*, **args*, ***kwargs*)

restart the specified container

container Container id

Returns the status mapping as usual

{'id': id of the container, 'status': True if started }

CLI Example:

```
salt '*' docker.start <container id>
```

`salt.modules.dockerio.`**`stop`**(*container*, *timeout=10*, **args*, ***kwargs*)

Stop a running container

Parameters

- **container** (*string*) – The container id to stop
- **timeout** (*int*) – Wait for a timeout to let the container exit gracefully before killing it

Return type dict

Returns

A status message with the command output ex:

```
{'id': 'abcdef123456789',
 'status': True}
```

CLI Example:

```
salt '*' docker.stop <container id>
```

`salt.modules.dockerio.`**`tag`**(*image*, *repository*, *tag=None*, *force=False*, **args*, ***kwargs*)

Tag an image into a repository

Parameters

- **image** (*string*) – The image to tag
- **repository** (*string*) – The repository to tag the image

- **tag** (*string*) – The tag to apply
- **force** (*boolean*) – Forces application of the tag

CLI Example:

```
salt '*' docker.tag <image> <repository> [tag] [force=(True|False)]
```

`salt.modules.dockerio.`**`top`**(*container*, **args*, ***kwargs*)

Run the docker top command on a specific container

container Container id

Returns in the 'out' status mapping a mapping for those running processes:

```
{
     'Titles': top titles list,
     'processes': list of ordered by
                  titles processes information,
     'mprocesses': list of mappings processes information
     constructed above the upon information
}
```

CLI Example:

```
salt '*' docker.top <container id>
```

`salt.modules.dockerio.`**`valid`**(*m*, *id=<object object at 0x7f223274a430>*, *comment=''*, *out=None*)

Return valid status

`salt.modules.dockerio.`**`version`**(**args*, ***kwargs*)

Get docker version

CLI Example:

```
salt '*' docker.version
```

`salt.modules.dockerio.`**`wait`**(*container*, **args*, ***kwargs*)

Blocking wait for a container exit gracefully without timeout killing it

container Container id

Return container id if successful

{'id': id of the container, 'status': True if stopped }

CLI Example:

```
salt '*' docker.wait <container id>
```

61.35 salt.modules.dpkg

Support for DEB packages

`salt.modules.dpkg.`**`file_dict`**(**packages*)

List the files that belong to a package, grouped by package. Not specifying any packages will return a list of _every_ file on the system's package database (not generally recommended).

CLI Examples:

```
salt '*' lowpkg.file_list httpd
salt '*' lowpkg.file_list httpd postfix
salt '*' lowpkg.file_list
```

`salt.modules.dpkg.`**`file_list`**`(`**packages*`)`

List the files that belong to a package. Not specifying any packages will return a list of _every_ file on the system's package database (not generally recommended).

CLI Examples:

```
salt '*' lowpkg.file_list httpd
salt '*' lowpkg.file_list httpd postfix
salt '*' lowpkg.file_list
```

`salt.modules.dpkg.`**`list_pkgs`**`(`**packages*`)`

List the packages currently installed in a dict:

```
{'<package_name>': '<version>'}
```

External dependencies:

```
Virtual package resolution requires aptitude. Because this function
uses dpkg, virtual packages will be reported as not installed.
```

CLI Example:

```
salt '*' lowpkg.list_pkgs
salt '*' lowpkg.list_pkgs httpd
```

`salt.modules.dpkg.`**`unpurge`**`(`**packages*`)`

Change package selection for each package specified to 'install'

CLI Example:

```
salt '*' lowpkg.unpurge curl
```

61.36 salt.modules.ebuild

Support for Portage

optdepends

- portage Python adapter

For now all package names *MUST* include the package category, i.e. `'vim'` will not work, `'app-editors/vim'` will.

`salt.modules.ebuild.`**`check_db`**`(`**names*, ***kwargs*`)`

New in version 0.17.0.

Returns a dict containing the following information for each specified package:

1. A key `found`, which will be a boolean value denoting if a match was found in the package database.
2. If `found` is `False`, then a second key called `suggestions` will be present, which will contain a list of possible matches. This list will be empty if the package name was specified in `category/pkgname` format, since the suggestions are only intended to disambiguate ambiguous package names (ones submitted without a category).

CLI Examples:

```
salt '*' pkg.check_db <package1> <package2> <package3>
```

`salt.modules.ebuild.`**`check_extra_requirements`**(*pkgname*, *pkgver*)

Check if the installed package already has the given requirements.

CLI Example:

```
salt '*' pkg.check_extra_requirements 'sys-devel/gcc' '~>4.1.2:4.1::gentoo[nls,fortran]'
```

`salt.modules.ebuild.`**`depclean`**(*name=None*, *slot=None*, *fromrepo=None*, *pkgs=None*)

Portage has a function to remove unused dependencies. If a package is provided, it will only removed the package if no other package depends on it.

name The name of the package to be cleaned.

slot Restrict the remove to a specific slot. Ignored if `name` is None.

fromrepo Restrict the remove to a specific slot. Ignored if `name` is None.

pkgs Clean multiple packages. `slot` and `fromrepo` arguments are ignored if this argument is present. Must be passed as a python list.

Return a list containing the removed packages:

CLI Example:

```
salt '*' pkg.depclean <package name>
```

`salt.modules.ebuild.`**`ex_mod_init`**(*low*)

If the config option `ebuild.enforce_nice_config` is set to True, this module will enforce a nice tree structure for /etc/portage/package.* configuration files.

New in version 0.17.0: Initial automatic enforcement added when pkg is used on a Gentoo system.

Changed in version 2014.1.0-Hydrogen: Configure option added to make this behaviour optional, defaulting to off.

See also:

`ebuild.ex_mod_init` is called automatically when a state invokes a pkg state on a Gentoo system. `salt.states.pkg.mod_init()`

`ebuild.ex_mod_init` uses `portage_config.enforce_nice_config` to do the lifting. `salt.modules.portage_config.enforce_nice_config()`

CLI Example:

```
salt '*' pkg.ex_mod_init
```

`salt.modules.ebuild.`**`install`**(*name=None*, *refresh=False*, *pkgs=None*, *sources=None*, *slot=None*, *fromrepo=None*, *uses=None*, ***kwargs*)

Install the passed package(s), add refresh=True to sync the portage tree before package is installed.

name The name of the package to be installed. Note that this parameter is ignored if either "pkgs" or "sources" is passed. Additionally, please note that this option can only be used to emerge a package from the portage tree. To install a tbz2 package manually, use the "sources" option described below.

CLI Example:

```
salt '*' pkg.install <package name>
```

refresh Whether or not to sync the portage tree before installing.

version Install a specific version of the package, e.g. 1.0.9-r1. Ignored if "pkgs" or "sources" is passed.

slot Similar to version, but specifies a valid slot to be installed. It will install the latest available version in the specified slot. Ignored if "pkgs" or "sources" or "version" is passed.

CLI Example:

```
salt '*' pkg.install sys-devel/gcc slot='4.4'
```

fromrepo Similar to slot, but specifies the repository from the package will be installed. It will install the latest available version in the specified repository. Ignored if "pkgs" or "sources" or "version" is passed.

CLI Example:

```
salt '*' pkg.install salt fromrepo='gentoo'
```

uses Similar to slot, but specifies a list of use flag. Ignored if "pkgs" or "sources" or "version" is passed.

CLI Example:

```
salt '*' pkg.install sys-devel/gcc uses='["nptl","-nossp"]'
```

Multiple Package Installation Options:

pkgs A list of packages to install from the portage tree. Must be passed as a python list.

CLI Example:

```
salt '*' pkg.install pkgs='["foo","bar","~category/package:slot::repository[use]"]'
```

sources A list of tbz2 packages to install. Must be passed as a list of dicts, with the keys being package names, and the values being the source URI or local path to the package.

CLI Example:

```
salt '*' pkg.install sources='[{"foo": "salt://foo.tbz2"},{"bar": "salt://bar.tbz2"}]'
```

Returns a dict containing the new package names and versions:

```
{'<package>': {'old': '<old-version>',
               'new': '<new-version>'}}
```

`salt.modules.ebuild.`**`latest_version`**`(`**names*, ***kwargs*`)`

Return the latest version of the named package available for upgrade or installation. If more than one package name is specified, a dict of name/version pairs is returned.

If the latest version of a given package is already installed, an empty string will be returned for that package.

CLI Example:

```
salt '*' pkg.latest_version <package name>
salt '*' pkg.latest_version <package1> <package2> <package3> ...
```

`salt.modules.ebuild.`**`list_pkgs`**`(`*versions_as_list=False*, ***kwargs*`)`

List the packages currently installed in a dict:

```
{'<package_name>': '<version>'}
```

CLI Example:

```
salt '*' pkg.list_pkgs
```

`salt.modules.ebuild.`**`list_upgrades`**`(`*refresh=True*`)`

List all available package upgrades.

CLI Example:

```
salt '*' pkg.list_upgrades
```

salt.modules.ebuild.**porttree_matches**(*name*)

Returns a list containing the matches for a given package name from the portage tree. Note that the specific version of the package will not be provided for packages that have several versions in the portage tree, but rather the name of the package (i.e. "dev-python/paramiko").

salt.modules.ebuild.**purge**(*name=None*, *slot=None*, *fromrepo=None*, *pkgs=None*, ***kwargs*)

Portage does not have a purge, this function calls remove followed by depclean to emulate a purge process

name The name of the package to be deleted.

slot Restrict the remove to a specific slot. Ignored if name is None.

fromrepo Restrict the remove to a specific slot. Ignored if `name` is None.

Multiple Package Options:

pkgs Uninstall multiple packages. `slot` and `fromrepo` arguments are ignored if this argument is present. Must be passed as a python list.

New in version 0.16.0.

Returns a dict containing the changes.

CLI Example:

```
salt '*' pkg.purge <package name>
salt '*' pkg.purge <package name> slot=4.4
salt '*' pkg.purge <package1>,<package2>,<package3>
salt '*' pkg.purge pkgs='["foo", "bar"]'
```

salt.modules.ebuild.**refresh_db**()

Updates the portage tree (emerge –sync). Uses eix-sync if available.

CLI Example:

```
salt '*' pkg.refresh_db
```

salt.modules.ebuild.**remove**(*name=None*, *slot=None*, *fromrepo=None*, *pkgs=None*, ***kwargs*)

Remove packages via emerge –unmerge.

name The name of the package to be deleted.

slot Restrict the remove to a specific slot. Ignored if `name` is None.

fromrepo Restrict the remove to a specific slot. Ignored if `name` is None.

Multiple Package Options:

pkgs Uninstall multiple packages. `slot` and `fromrepo` arguments are ignored if this argument is present. Must be passed as a python list.

New in version 0.16.0.

Returns a dict containing the changes.

CLI Example:

```
salt '*' pkg.remove <package name>
salt '*' pkg.remove <package name> slot=4.4 fromrepo=gentoo
salt '*' pkg.remove <package1>,<package2>,<package3>
salt '*' pkg.remove pkgs='["foo", "bar"]'
```

`salt.modules.ebuild.`**`update`**`(`*pkg*, *slot=None*, *fromrepo=None*, *refresh=False*`)`

Updates the passed package (emerge –update package)

slot Restrict the update to a particular slot. It will update to the latest version within the slot.

fromrepo Restrict the update to a particular repository. It will update to the latest version within the repository.

Return a dict containing the new package names and versions:

```
{'<package>': {'old': '<old-version>',
               'new': '<new-version>'}}
```

CLI Example:

```
salt '*' pkg.update <package name>
```

`salt.modules.ebuild.`**`upgrade`**`(`*refresh=True*`)`

Run a full system upgrade (emerge –update world)

Return a dict containing the new package names and versions:

```
{'<package>': {'old': '<old-version>',
               'new': '<new-version>'}}
```

CLI Example:

```
salt '*' pkg.upgrade
```

`salt.modules.ebuild.`**`upgrade_available`**`(`*name*`)`

Check whether or not an upgrade is available for a given package

CLI Example:

```
salt '*' pkg.upgrade_available <package name>
```

`salt.modules.ebuild.`**`version`**`(`**names*, ***kwargs*`)`

Returns a string representing the package version or an empty string if not installed. If more than one package name is specified, a dict of name/version pairs is returned.

CLI Example:

```
salt '*' pkg.version <package name>
salt '*' pkg.version <package1> <package2> <package3> ...
```

`salt.modules.ebuild.`**`version_clean`**`(`*version*`)`

Clean the version string removing extra data.

CLI Example:

```
salt '*' pkg.version_clean <version_string>
```

`salt.modules.ebuild.`**`version_cmp`**`(`*pkg1*, *pkg2*`)`

Do a cmp-style comparison on two packages. Return -1 if pkg1 < pkg2, 0 if pkg1 == pkg2, and 1 if pkg1 > pkg2. Return None if there was a problem making the comparison.

CLI Example:

```
salt '*' pkg.version_cmp '0.2.4-0' '0.2.4.1-0'
```

61.37 salt.modules.eix

Support for Eix

`salt.modules.eix.`**`sync`**`()`

Sync portage/overlay trees and update the eix database

CLI Example:

```
salt '*' eix.sync
```

`salt.modules.eix.`**`update`**`()`

Update the eix database

CLI Example:

```
salt '*' eix.update
```

61.38 salt.modules.eselect

Support for eselect, Gentoo's configuration and management tool.

`salt.modules.eselect.`**`exec_action`**`(`*module*, *action*, *parameter=''*, *state_only=False*`)`

Execute an arbitrary action on a module.

CLI Example:

```
salt '*' eselect.exec_action <module name> <action> [parameter]
```

`salt.modules.eselect.`**`get_current_target`**`(`*module*`)`

Get the currently selected target for the given module.

CLI Example:

```
salt '*' eselect.get_current_target <module name>
```

`salt.modules.eselect.`**`get_modules`**`()`

Get available modules list.

CLI Example:

```
salt '*' eselect.get_modules
```

`salt.modules.eselect.`**`get_target_list`**`(`*module*`)`

Get available target for the given module.

CLI Example:

```
salt '*' eselect.get_target_list <module name>
```

`salt.modules.eselect.`**`set_target`**`(`*module*, *target*`)`

Set the target for the given module. Target can be specified by index or name.

CLI Example:

```
salt '*' eselect.set_target <module name> <target>
```

61.39 salt.modules.event

Use the *Salt Event System* to fire events from the master to the minion and vice-versa.

`salt.modules.event.`**`fire`**`(`*data*, *tag*`)`

Fire an event on the local minion event bus. Data must be formed as a dict.

CLI Example:

```
salt '*' event.fire '{"data":"my event data"}' 'tag'
```

`salt.modules.event.`**`fire_master`**`(`*data*, *tag*, *preload=None*`)`

Fire an event off up to the master server

CLI Example:

```
salt '*' event.fire_master '{"data":"my event data"}' 'tag'
```

61.40 salt.modules.extfs

Module for managing ext2/3/4 file systems

`salt.modules.extfs.`**`attributes`**`(`*device*, *args=None*`)`

Return attributes from dumpe2fs for a specified device

CLI Example:

```
salt '*' extfs.attributes /dev/sda1
```

`salt.modules.extfs.`**`blocks`**`(`*device*, *args=None*`)`

Return block and inode info from dumpe2fs for a specified device

CLI Example:

```
salt '*' extfs.blocks /dev/sda1
```

`salt.modules.extfs.`**`dump`**`(`*device*, *args=None*`)`

Return all contents of dumpe2fs for a specified device

CLI Example:

```
salt '*' extfs.dump /dev/sda1
```

`salt.modules.extfs.`**`mkfs`**`(`*device*, *fs_type*, ***kwargs*`)`

Create a file system on the specified device

CLI Example:

```
salt '*' extfs.mkfs /dev/sda1 fs_type=ext4 opts='acl,noexec'
```

Valid options are:

```
block_size: 1024, 2048 or 4096
check: check for bad blocks
direct: use direct IO
ext_opts: extended file system options (comma-separated)
fragment_size: size of fragments
force: setting force to True will cause mke2fs to specify the -F option
       twice (it is already set once); this is truly dangerous
blocks_per_group: number of blocks in a block group
```

```
number_of_groups: ext4 option for a virtual block group
bytes_per_inode: set the bytes/inode ratio
inode_size: size of the inode
journal: set to True to create a journal (default on ext3/4)
journal_opts: options for the fs journal (comma separated)
blocks_file: read bad blocks from file
label: label to apply to the file system
reserved: percentage of blocks reserved for super-user
last_dir: last mounted directory
test: set to True to not actually create the file system (mke2fs -n)
number_of_inodes: override default number of inodes
creator_os: override "creator operating system" field
opts: mount options (comma separated)
revision: set the filesystem revision (default 1)
super: write superblock and group descriptors only
fs_type: set the filesystem type (REQUIRED)
usage_type: how the filesystem is going to be used
uuid: set the UUID for the file system
```

See the `mke2fs(8)` manpage for a more complete description of these options.

`salt.modules.extfs.`**`tune`**(*device*, ***kwargs*)

Set attributes for the specified device (using tune2fs)

CLI Example:

```
salt '*' extfs.tune /dev/sda1 force=True label=wildstallyns opts='acl,noexec'
```

Valid options are:

```
max: max mount count
count: mount count
error: error behavior
extended_opts: extended options (comma separated)
force: force, even if there are errors (set to True)
group: group name or gid that can use the reserved blocks
interval: interval between checks
journal: set to True to create a journal (default on ext3/4)
journal_opts: options for the fs journal (comma separated)
label: label to apply to the file system
reserved: percentage of blocks reserved for super-user
last_dir: last mounted directory
opts: mount options (comma separated)
feature: set or clear a feature (comma separated)
mmp_check: mmp check interval
reserved: reserved blocks count
quota_opts: quota options (comma separated)
time: time last checked
user: user or uid who can use the reserved blocks
uuid: set the UUID for the file system
```

See the `mke2fs(8)` manpage for a more complete description of these options.

61.41 salt.modules.file

Manage information about regular files, directories, and special files on the minion, set/read user, group, mode, and data `salt.modules.file.`**`access`**(*path*, *mode*)

Test whether the Salt process has the specified access to the file. One of the following modes must be specified:

> f: Test the existence of the path r: Test the readability of the path w: Test the writability of the path x: Test whether the path can be executed

New in version 2014.1.0: (Hydrogen)

CLI Example:

```
salt '*' file.access /path/to/file f
salt '*' file.access /path/to/file x
```

`salt.modules.file.`**`append`**(*path*, **args*)

New in version 0.9.5.

Append text to the end of a file

CLI Example:

```
salt '*' file.append /etc/motd \
        "With all thine offerings thou shalt offer salt." \
        "Salt is what makes things taste bad when it isn't in them."
```

`salt.modules.file.`**`blockreplace`**(*path*, *marker_start='#– start managed zone –'*, *marker_end='#– end managed zone –'*, *content=''*, *append_if_not_found=False*, *prepend_if_not_found=False*, *backup='.bak'*, *dry_run=False*, *show_changes=True*)

Replace content of a text block in a file, delimited by line markers

New in version 2014.1.0: (Hydrogen)

A block of content delimited by comments can help you manage several lines entries without worrying about old entries removal.

Note: This function will store two copies of the file in-memory (the original version and the edited version) in order to detect changes and only edit the targeted file if necessary.

---

path Filesystem path to the file to be edited

marker_start The line content identifying a line as the start of the content block. Note that the whole line containing this marker will be considered, so whitespaces or extra content before or after the marker is included in final output

marker_end The line content identifying a line as the end of the content block. Note that the whole line containing this marker will be considered, so whitespaces or extra content before or after the marker is included in final output

content The content to be used between the two lines identified by marker_start and marker_stop.

append_if_not_found [False] If markers are not found and set to `True` then, the markers and content will be appended to the file.

prepend_if_not_found [False] If markers are not found and set to `True` then, the markers and content will be prepended to the file.

backup The file extension to use for a backup of the file if any edit is made. Set to `False` to skip making a backup.

dry_run Don't make any edits to the file.

show_changes Output a unified diff of the old file and the new file. If `False`, return a boolean if any changes were made.

CLI Example:

```
salt '*' file.blockreplace /etc/hosts '#-- start managed zone foobar : DO NOT EDIT --' \
'#-- end managed zone foobar --' $'10.0.1.1 foo.foobar\n10.0.1.2 bar.foobar' True
```

`salt.modules.file.`**`check_file_meta`**(*name*, *sfn*, *source*, *source_sum*, *user*, *group*, *mode*, *saltenv*, *template=None*, *contents=None*)

Check for the changes in the file metadata.

CLI Example:

```
salt '*' file.check_file_meta /etc/httpd/conf.d/httpd.conf salt://http/httpd.conf '{hash_type: '
```

`salt.modules.file.`**`check_hash`**(*path*, *file_hash*)

Check if a file matches the given hash string

Returns true if the hash matched, otherwise false. Raises ValueError if the hash was not formatted correctly.

path A file path

hash A string in the form <hash_type>=<hash_value>. For example: `md5=e138491e9d5b97023cea823fe17bac22`

CLI Example:

```
salt '*' file.check_hash /etc/fstab md5=<md5sum>
```

`salt.modules.file.`**`check_managed`**(*name*, *source*, *source_hash*, *user*, *group*, *mode*, *template*, *context*, *defaults*, *saltenv*, *contents=None*, ***kwargs*)

Check to see what changes need to be made for a file

CLI Example:

```
salt '*' file.check_managed /etc/httpd/conf.d/httpd.conf salt://http/httpd.conf '{hash_type: 'm
```

`salt.modules.file.`**`check_perms`**(*name*, *ret*, *user*, *group*, *mode*, *follow_symlinks=False*)

Check the permissions on files and chown if needed

CLI Example:

```
salt '*' file.check_perms /etc/sudoers '{}' root root 400
```

Changed in version 2014.1.3: `follow_symlinks` option added

`salt.modules.file.`**`chgrp`**(*path*, *group*)

Change the group of a file

CLI Example:

```
salt '*' file.chgrp /etc/passwd root
```

`salt.modules.file.`**`chown`**(*path*, *user*, *group*)

Chown a file, pass the file the desired user and group

CLI Example:

```
salt '*' file.chown /etc/passwd root root
```

`salt.modules.file.`**`comment`**(*path*, *regex*, *char='#'*, *backup='.bak'*)

Deprecated since version 0.17.0: Use `replace()` instead.

Comment out specified lines in a file

path The full path to the file to be edited

regex A regular expression used to find the lines that are to be commented; this pattern will be wrapped in parenthesis and will move any preceding/trailing `^` or `$` characters outside the parenthesis (e.g., the pattern `^foo$` will be rewritten as `^(foo)$`)

char [#] The character to be inserted at the beginning of a line in order to comment it out

backup [`.bak`] The file will be backed up before edit with this file extension

> **Warning:** This backup will be overwritten each time `sed` / `comment` / `uncomment` is called. Meaning the backup will only be useful after the first invocation.

CLI Example:

```
salt '*' file.comment /etc/modules pcspkr
```

`salt.modules.file.`**`contains`**(*path*, *text*)

Deprecated since version 0.17.0: Use `search()` instead.

Return `True` if the file at `path` contains `text`

CLI Example:

```
salt '*' file.contains /etc/crontab 'mymaintenance.sh'
```

`salt.modules.file.`**`contains_glob`**(*path*, *glob_expr*)

Deprecated since version 0.17.0: Use `search()` instead.

Return `True` if the given glob matches a string in the named file

CLI Example:

```
salt '*' file.contains_glob /etc/foobar '*cheese*'
```

`salt.modules.file.`**`contains_regex`**(*path*, *regex*, *lchar=''*)

Deprecated since version 0.17.0: Use `search()` instead.

Return True if the given regular expression matches on any line in the text of a given file.

If the lchar argument (leading char) is specified, it will strip *lchar* from the left side of each line before trying to match

CLI Example:

```
salt '*' file.contains_regex /etc/crontab
```

`salt.modules.file.`**`contains_regex_multiline`**(*path*, *regex*)

Deprecated since version 0.17.0: Use `search()` instead.

Return True if the given regular expression matches anything in the text of a given file

Traverses multiple lines at a time, via the salt BufferedReader (reads in chunks)

CLI Example:

```
salt '*' file.contains_regex_multiline /etc/crontab '^maint'
```

`salt.modules.file.`**`copy`**(*src*, *dst*)

Copy a file or directory

CLI Example:

```
salt '*' file.copy /path/to/src /path/to/dst
```

`salt.modules.file.`**`delete_backup`**(*path*, *backup_id*)

Restore a previous version of a file that was backed up using Salt's *file state backup* system.

New in version 0.17.0.

path The path on the minion to check for backups

backup_id The numeric id for the backup you wish to delete, as found using `file.list_backups`

CLI Example:

```
salt '*' file.restore_backup /foo/bar/baz.txt 0
```

`salt.modules.file.`**`directory_exists`**(*path*)

Tests to see if path is a valid directory. Returns True/False.

CLI Example:

```
salt '*' file.directory_exists /etc
```

`salt.modules.file.`**`extract_hash`**(*hash_fn*, *hash_type='md5'*, *file_name='`*)

This routine is called from the `file.managed` state to pull a hash from a remote file. Regular expressions are used line by line on the `source_hash` file, to find a potential candidate of the indicated hash type. This avoids many problems of arbitrary file lay out rules. It specifically permits pulling hash codes from debian `*.dsc` files.

For example:

```
openerp_7.0-latest-1.tar.gz:
  file.managed:
    - name: /tmp/openerp_7.0-20121227-075624-1_all.deb
    - source: http://nightly.openerp.com/7.0/nightly/deb/openerp_7.0-20121227-075624-1.tar.gz
    - source_hash: http://nightly.openerp.com/7.0/nightly/deb/openerp_7.0-20121227-075624-1.dsc
```

CLI Example:

```
salt '*' file.extract_hash /etc/foo sha512 /path/to/hash/file
```

`salt.modules.file.`**`file_exists`**(*path*)

Tests to see if path is a valid file. Returns True/False.

CLI Example:

```
salt '*' file.file_exists /etc/passwd
```

`salt.modules.file.`**`find`**(*path*, ***kwargs*)

Approximate the Unix `find(1)` command and return a list of paths that meet the specified criteria.

The options include match criteria:

```
name    = path-glob                 # case sensitive
iname   = path-glob                 # case insensitive
regex   = path-regex                # case sensitive
iregex  = path-regex                # case insensitive
type    = file-types                # match any listed type
user    = users                     # match any listed user
group   = groups                    # match any listed group
size    = [+-]number[size-unit]     # default unit = byte
mtime   = interval                  # modified since date
grep    = regex                     # search file contents
```

and/or actions:

```
delete [= file-types]               # default type = 'f'
exec    = command [arg ...]         # where {} is replaced by pathname
print  [= print-opts]
```

The default action is 'print=path'.

file-glob:

```
*                 = match zero or more chars
?                 = match any char
[abc]             = match a, b, or c
[!abc] or [^abc] = match anything except a, b, and c
[x-y]             = match chars x through y
[!x-y] or [^x-y] = match anything except chars x through y
{a,b,c}           = match a or b or c
```

path-regex: a Python re (regular expression) pattern to match pathnames

file-types: a string of one or more of the following:

```
a: all file types
b: block device
c: character device
d: directory
p: FIFO (named pipe)
f: plain file
l: symlink
s: socket
```

users: a space and/or comma separated list of user names and/or uids

groups: a space and/or comma separated list of group names and/or gids

size-unit:

```
b: bytes
k: kilobytes
m: megabytes
g: gigabytes
t: terabytes
```

interval:

```
[<num>w] [<num>d] [<num>h] [<num>m] [<num>s]

where:
    w: week
    d: day
    h: hour
    m: minute
    s: second
```

print-opts: a comma and/or space separated list of one or more of the following:

```
group: group name
md5:   MD5 digest of file contents
mode:  file permissions (as integer)
mtime: last modification time (as time_t)
name:  file basename
path:  file absolute path
```

```
size:  file size in bytes
type:  file type
user:  user name
```

CLI Examples:

```
salt '*' file.find / type=f name=\*.bak size=+10m
salt '*' file.find /var mtime=+30d size=+10m print=path,size,mtime
salt '*' file.find /var/log name=\*.[0-9] mtime=+30d size=+10m delete
```

`salt.modules.file.`**`get_devmm`**(*name*)

Get major/minor info from a device

CLI Example:

```
salt '*' file.get_devmm /dev/chr
```

`salt.modules.file.`**`get_diff`**(*minionfile*, *masterfile*, *env=None*, *saltenv='base'*)

Return unified diff of file compared to file on master

CLI Example:

```
salt '*' file.get_diff /home/fred/.vimrc salt://users/fred/.vimrc
```

`salt.modules.file.`**`get_gid`**(*path*, *follow_symlinks=True*)

Return the id of the group that owns a given file

CLI Example:

```
salt '*' file.get_gid /etc/passwd
```

Changed in version 0.16.4: `follow_symlinks` option added

`salt.modules.file.`**`get_group`**(*path*, *follow_symlinks=True*)

Return the group that owns a given file

CLI Example:

```
salt '*' file.get_group /etc/passwd
```

Changed in version 0.16.4: `follow_symlinks` option added

`salt.modules.file.`**`get_hash`**(*path*, *form='md5'*, *chunk_size=4096*)

Get the hash sum of a file

This is better than `get_sum` for the following reasons:

- It does not read the entire file into memory.
- **It does not return a string on error. The returned value of** `get_sum` cannot really be trusted since it is vulnerable to collisions: `get_sum(..., 'xyz') == 'Hash xyz not supported'`

CLI Example:

```
salt '*' file.get_hash /etc/shadow
```

`salt.modules.file.`**`get_managed`**(*name*, *template*, *source*, *source_hash*, *user*, *group*, *mode*, *saltenv*, *context*, *defaults*, ***kwargs*)

Return the managed file data for file.managed

CLI Example:

```
salt '*' file.get_managed /etc/httpd/conf.d/httpd.conf jinja salt://http/httpd.conf '{hash_type:
```

salt.modules.file.**get_mode**(*path*, *follow_symlinks=True*)
Return the mode of a file

CLI Example:

```
salt '*' file.get_mode /etc/passwd
```

Changed in version 2014.1.0: `follow_symlinks` option added

salt.modules.file.**get_selinux_context**(*path*)
Get an SELinux context from a given path

CLI Example:

```
salt '*' file.get_selinux_context /etc/hosts
```

salt.modules.file.**get_sum**(*path*, *form='md5'*)
Return the sum for the given file, default is md5, sha1, sha224, sha256, sha384, sha512 are supported

CLI Example:

```
salt '*' file.get_sum /etc/passwd sha512
```

salt.modules.file.**get_uid**(*path*, *follow_symlinks=True*)
Return the id of the user that owns a given file

CLI Example:

```
salt '*' file.get_uid /etc/passwd
```

Changed in version 0.16.4: `follow_symlinks` option added

salt.modules.file.**get_user**(*path*, *follow_symlinks=True*)
Return the user that owns a given file

CLI Example:

```
salt '*' file.get_user /etc/passwd
```

Changed in version 0.16.4: `follow_symlinks` option added

salt.modules.file.**gid_to_group**(*gid*)
Convert the group id to the group name on this system

CLI Example:

```
salt '*' file.gid_to_group 0
```

salt.modules.file.**grep**(*path*, *pattern*, **args*)
Grep for a string in the specified file

Note: This function's return value is slated for refinement in future versions of Salt

path A file path

pattern A string. For example: `test` `a[0-5]`

args grep options. For example: `" -v"` `" -i -B2"`

CLI Example:

```
salt '*' file.grep /etc/passwd nobody
salt '*' file.grep /etc/sysconfig/network-scripts/ifcfg-eth0 ipaddr " -i"
salt '*' file.grep /etc/sysconfig/network-scripts/ifcfg-eth0 ipaddr " -i -B2"
salt '*' file.grep "/etc/sysconfig/network-scripts/*" ipaddr " -i -l"
```

`salt.modules.file.`**`group_to_gid`**(*group*)

Convert the group to the gid on this system

CLI Example:

```
salt '*' file.group_to_gid root
```

`salt.modules.file.`**`is_blkdev`**(*name*)

Check if a file exists and is a block device.

CLI Example:

```
salt '*' file.is_blkdev /dev/blk
```

`salt.modules.file.`**`is_chrdev`**(*name*)

Check if a file exists and is a character device.

CLI Example:

```
salt '*' file.is_chrdev /dev/chr
```

`salt.modules.file.`**`is_fifo`**(*name*)

Check if a file exists and is a FIFO.

CLI Example:

```
salt '*' file.is_fifo /dev/fifo
```

`salt.modules.file.`**`lchown`**(*path*, *user*, *group*)

Chown a file, pass the file the desired user and group without following symlinks.

CLI Example:

```
salt '*' file.chown /etc/passwd root root
```

`salt.modules.file.`**`link`**(*src*, *path*)

Create a hard link to a file

New in version 2014.1.0: (Hydrogen)

CLI Example:

```
salt '*' file.link /path/to/file /path/to/link
```

`salt.modules.file.`**`list_backups`**(*path*, *limit=None*)

Lists the previous versions of a file backed up using Salt's *file state backup* system.

New in version 0.17.0.

path The path on the minion to check for backups

limit Limit the number of results to the most recent N backups

CLI Example:

```
salt '*' file.list_backups /foo/bar/baz.txt
```

salt.modules.file.**lstat**(*path*)

Returns the lstat attributes for the given file or dir. Does not support symbolic links.

CLI Example:

New in version 2014.1.0: (Hydrogen)

```
salt '*' file.lstat /path/to/file
```

salt.modules.file.**makedirs_**(*path*, *user=None*, *group=None*, *mode=None*)

Ensure that the directory containing this path is available.

CLI Example:

```
salt '*' file.makedirs /opt/code
```

salt.modules.file.**makedirs_perms**(*name*, *user=None*, *group=None*, *mode='0755'*)

Taken and modified from os.makedirs to set user, group and mode for each directory created.

CLI Example:

```
salt '*' file.makedirs_perms /opt/code
```

salt.modules.file.**manage_file**(*name*, *sfn*, *ret*, *source*, *source_sum*, *user*, *group*, *mode*, *saltenv*, *backup*, *makedirs=False*, *template=None*, *show_diff=True*, *contents=None*, *dir_mode=None*)

Checks the destination against what was retrieved with get_managed and makes the appropriate modifications (if necessary).

CLI Example:

```
salt '*' file.manage_file /etc/httpd/conf.d/httpd.conf '{}' salt://http/httpd.conf '{hash_type:
```

salt.modules.file.**mkdir**(*dir_path*, *user=None*, *group=None*, *mode=None*)

Ensure that a directory is available.

CLI Example:

```
salt '*' file.mkdir /opt/jetty/context
```

salt.modules.file.**mknod**(*name*, *ntype*, *major=0*, *minor=0*, *user=None*, *group=None*, *mode='0600'*)

Create a block device, character device, or fifo pipe. Identical to the gnu mknod.

CLI Examples:

```
salt '*' file.mknod /dev/chr c 180 31
salt '*' file.mknod /dev/blk b 8 999
salt '*' file.nknod /dev/fifo p
```

salt.modules.file.**mknod_blkdev**(*name*, *major*, *minor*, *user=None*, *group=None*, *mode='0660'*)

Create a block device.

CLI Example:

```
salt '*' file.mknod_blkdev /dev/blk 8 999
```

salt.modules.file.**mknod_chrdev**(*name*, *major*, *minor*, *user=None*, *group=None*, *mode='0660'*)

Create a character device.

CLI Example:

```
salt '*' file.mknod_chrdev /dev/chr 180 31
```

`salt.modules.file.`**`mknod_fifo`**(*name*, *user=None*, *group=None*, *mode='0660'*)

Create a FIFO pipe.

CLI Example:

```
salt '*' file.mknod_fifo /dev/fifo
```

`salt.modules.file.`**`patch`**(*originalfile*, *patchfile*, *options=''*, *dry_run=False*)

New in version 0.10.4.

Apply a patch to a file

Equivalent to:

```
patch <options> <originalfile> <patchfile>
```

originalfile The full path to the file or directory to be patched

patchfile A patch file to apply to `originalfile`

options Options to pass to patch.

CLI Example:

```
salt '*' file.patch /opt/file.txt /tmp/file.txt.patch
```

`salt.modules.file.`**`psed`**(*path*, *before*, *after*, *limit=''*, *backup='.bak'*, *flags='gMS'*, *escape_all=False*, *multi=False*)

Deprecated since version 0.17.0: Use `replace()` instead.

Make a simple edit to a file (pure Python version)

Equivalent to:

```
sed <backup> <options> "/<limit>/ s/<before>/<after>/<flags> <file>"
```

path The full path to the file to be edited

before A pattern to find in order to replace with `after`

after Text that will replace `before`

limit [`''`] An initial pattern to search for before searching for `before`

backup [`.bak`] The file will be backed up before edit with this file extension; **WARNING:** each time `sed`/`comment`/`uncomment` is called will overwrite this backup

flags [`gMS`]

Flags to modify the search. Valid values are:

- `g`: Replace all occurrences of the pattern, not just the first.
- `I`: Ignore case.
- `L`: Make `\w`, `\W`, `\b`, `\B`, `\s` and `\S` dependent on the locale.
- `M`: Treat multiple lines as a single line.
- `S`: Make `.` match all characters, including newlines.
- `U`: Make `\w`, `\W`, `\b`, `\B`, `\d`, `\D`, `\s` and `\S` dependent on Unicode.
- `X`: Verbose (whitespace is ignored).

multi: `False` If True, treat the entire file as a single line

Forward slashes and single quotes will be escaped automatically in the `before` and `after` patterns.

CLI Example:

```
salt '*' file.sed /etc/httpd/httpd.conf 'LogLevel warn' 'LogLevel info'
```

`salt.modules.file.`**`readdir`**(*path*)

Return a list containing the contents of a directory

New in version 2014.1.0: (Hydrogen)

CLI Example:

```
salt '*' file.readdir /path/to/dir/
```

`salt.modules.file.`**`readlink`**(*path*)

Return the path that a symlink points to

New in version 2014.1.0: (Hydrogen)

CLI Example:

```
salt '*' file.readlink /path/to/link
```

`salt.modules.file.`**`remove`**(*path*)

Remove the named file

CLI Example:

```
salt '*' file.remove /tmp/foo
```

`salt.modules.file.`**`rename`**(*src*, *dst*)

Rename a file or directory

CLI Example:

```
salt '*' file.rename /path/to/src /path/to/dst
```

`salt.modules.file.`**`replace`**(*path*, *pattern*, *repl*, *count=0*, *flags=0*, *bufsize=1*, *backup='.bak'*, *dry_run=False*, *search_only=False*, *show_changes=True*)

Replace occurances of a pattern in a file

New in version 0.17.0.

This is a pure Python implementation that wraps Python's `sub()`.

Parameters

- **path** – Filesystem path to the file to be edited
- **pattern** – The PCRE search
- **repl** – The replacement text
- **count** – Maximum number of pattern occurrences to be replaced
- **flags** (*list or int*) – A list of flags defined in the *re module documentation*. Each list item should be a string that will correlate to the human-friendly flag name. E.g., `['IGNORECASE', 'MULTILINE']`. Note: multiline searches must specify `file` as the `bufsize` argument below.
- **bufsize** (*int or str*) – How much of the file to buffer into memory at once. The default value `1` processes one line at a time. The special value `file` may be specified which will read the entire file into memory before processing. Note: multiline searches must specify `file` buffering.

- **backup** – The file extension to use for a backup of the file before editing. Set to `False` to skip making a backup.
- **dry_run** – Don't make any edits to the file
- **search_only** – Just search for the pattern; ignore the replacement; stop on the first match
- **show_changes** – Output a unified diff of the old file and the new file. If `False` return a boolean if any changes were made. Note: using this option will store two copies of the file in-memory (the original version and the edited version) in order to generate the diff.

Return type bool or str

CLI Example:

```
salt '*' file.replace /etc/httpd/httpd.conf 'LogLevel warn' 'LogLevel info'
salt '*' file.replace /some/file 'before' 'after' flags='[MULTILINE, IGNORECASE]'
```

`salt.modules.file.`**`restore_backup`**(*path*, *backup_id*)

Restore a previous version of a file that was backed up using Salt's *file state backup* system.

New in version 0.17.0.

path The path on the minion to check for backups

backup_id The numeric id for the backup you wish to restore, as found using `file.list_backups`

CLI Example:

```
salt '*' file.restore_backup /foo/bar/baz.txt 0
```

`salt.modules.file.`**`restorecon`**(*path*, *recursive=False*)

Reset the SELinux context on a given path

CLI Example:

```
salt '*' file.restorecon /home/user/.ssh/authorized_keys
```

`salt.modules.file.`**`rmdir`**(*path*)

Remove the specified directory. Fails if a directory is not empty.

New in version 2014.1.0: (Hydrogen)

CLI Example:

```
salt '*' file.rmdir /tmp/foo/
```

`salt.modules.file.`**`search`**(*path*, *pattern*, *flags=0*, *bufsize=1*)

Search for occurances of a pattern in a file

New in version 0.17.0.

Params are identical to `replace()`.

CLI Example:

```
salt '*' file.search /etc/crontab 'mymaintenance.sh'
```

`salt.modules.file.`**`sed`**(*path*, *before*, *after*, *limit='‘*, *backup='.bak'*, *options='-r -e'*, *flags='g'*, *escape_all=False*, *negate_match=False*)

Deprecated since version 0.17.0: Use `replace()` instead.

Make a simple edit to a file

Equivalent to:

```
sed <backup> <options> "/<limit>/ s/<before>/<after>/<flags> <file>"
```

path The full path to the file to be edited

before A pattern to find in order to replace with `after`

after Text that will replace `before`

limit [`''`] An initial pattern to search for before searching for `before`

backup [`.bak`] The file will be backed up before edit with this file extension; **WARNING:** each time `sed`/`comment`/`uncomment` is called will overwrite this backup

options [`-r -e`] Options to pass to sed

flags [`g`] Flags to modify the sed search; e.g., `i` for case-insensitve pattern matching

negate_match [False] Negate the search command (`!`)

New in version 0.17.0.

Forward slashes and single quotes will be escaped automatically in the `before` and `after` patterns.

CLI Example:

```
salt '*' file.sed /etc/httpd/httpd.conf 'LogLevel warn' 'LogLevel info'
```

`salt.modules.file.`**`sed_contains`**(*path*, *text*, *limit=''*, *flags='g'*)

Deprecated since version 0.17.0: Use `search()` instead.

Return True if the file at `path` contains `text`. Utilizes sed to perform the search (line-wise search).

Note: the `p` flag will be added to any flags you pass in.

CLI Example:

```
salt '*' file.contains /etc/crontab 'mymaintenance.sh'
```

`salt.modules.file.`**`seek_read`**(*path*, *size*, *offset*)

Seek to a position on a file and write to it

New in version 2014.1.0: (Hydrogen)

CLI Example:

```
salt '*' file.seek_read /path/to/file 4096 0
```

`salt.modules.file.`**`seek_write`**(*path*, *data*, *offset*)

Seek to a position on a file and write to it

New in version 2014.1.0: (Hydrogen)

CLI Example:

```
salt '*' file.seek_write /path/to/file 'some data' 4096
```

`salt.modules.file.`**`set_mode`**(*path*, *mode*)

Set the mode of a file

CLI Example:

```
salt '*' file.set_mode /etc/passwd 0644
```

`salt.modules.file.`**`set_selinux_context`**`(`*path*, *user=None*, *role=None*, *type=None*, *range=None*`)`

Set a specific SELinux label on a given path

CLI Example:

```
salt '*' file.set_selinux_context path <role> <type> <range>
```

`salt.modules.file.`**`source_list`**`(`*source*, *source_hash*, *saltenv*`)`

Check the source list and return the source to use

CLI Example:

```
salt '*' file.source_list salt://http/httpd.conf '{hash_type: 'md5', 'hsum': <md5sum>}' base
```

`salt.modules.file.`**`stats`**`(`*path*, *hash_type='md5'*, *follow_symlinks=True*`)`

Return a dict containing the stats for a given file

CLI Example:

```
salt '*' file.stats /etc/passwd
```

`salt.modules.file.`**`statvfs`**`(`*path*`)`

Perform a statvfs call against the filesystem that the file resides on

New in version 2014.1.0: (Hydrogen)

CLI Example:

```
salt '*' file.statvfs /path/to/file
```

`salt.modules.file.`**`symlink`**`(`*src*, *path*`)`

Create a symbolic link to a file

CLI Example:

```
salt '*' file.symlink /path/to/file /path/to/link
```

`salt.modules.file.`**`touch`**`(`*name*, *atime=None*, *mtime=None*`)`

New in version 0.9.5.

Just like the `touch` command, create a file if it doesn't exist or simply update the atime and mtime if it already does.

atime: Access time in Unix epoch time

mtime: Last modification in Unix epoch time

CLI Example:

```
salt '*' file.touch /var/log/emptyfile
```

`salt.modules.file.`**`truncate`**`(`*path*, *length*`)`

Seek to a position on a file and write to it

New in version 2014.1.0: (Hydrogen)

CLI Example:

```
salt '*' file.truncate /path/to/file 512
```

`salt.modules.file.`**`uid_to_user`**`(`*uid*`)`

Convert a uid to a user name

CLI Example:

```
salt '*' file.uid_to_user 0
```

`salt.modules.file.`**`uncomment`**`(`*path*, *regex*, *char='#'*, *backup='.bak'*`)`

Deprecated since version 0.17.0: Use `replace()` instead.

Uncomment specified commented lines in a file

path The full path to the file to be edited

regex A regular expression used to find the lines that are to be uncommented. This regex should not include the comment character. A leading ^ character will be stripped for convenience (for easily switching between comment() and uncomment()).

char [#] The character to remove in order to uncomment a line

backup [`.bak`] The file will be backed up before edit with this file extension; **WARNING:** each time `sed`/`comment`/`uncomment` is called will overwrite this backup

CLI Example:

```
salt '*' file.uncomment /etc/hosts.deny 'ALL: PARANOID'
```

`salt.modules.file.`**`user_to_uid`**`(`*user*`)`

Convert user name to a uid

CLI Example:

```
salt '*' file.user_to_uid root
```

61.42 salt.modules.freebsd_sysctl

Module for viewing and modifying sysctl parameters

`salt.modules.freebsd_sysctl.`**`assign`**`(`*name*, *value*`)`

Assign a single sysctl parameter for this minion

CLI Example:

```
salt '*' sysctl.assign net.inet.icmp.icmplim 50
```

`salt.modules.freebsd_sysctl.`**`get`**`(`*name*`)`

Return a single sysctl parameter for this minion

CLI Example:

```
salt '*' sysctl.get hw.physmem
```

`salt.modules.freebsd_sysctl.`**`persist`**`(`*name*, *value*, *config='/etc/sysctl.conf'*`)`

Assign and persist a simple sysctl parameter for this minion

CLI Example:

```
salt '*' sysctl.persist net.inet.icmp.icmplim 50
salt '*' sysctl.persist coretemp_load NO config=/boot/loader.conf
```

`salt.modules.freebsd_sysctl.`**`show`**`()`

Return a list of sysctl parameters for this minion

CLI Example:

```
salt '*' sysctl.show
```

61.43 salt.modules.freebsdjail

The jail module for FreeBSD

`salt.modules.freebsdjail.`**`fstab`**(*jail*)
Display contents of a fstab(5) file defined in specified jail's configuration. If no file is defined, return False.

CLI Example:

```
salt '*' jail.fstab <jail name>
```

`salt.modules.freebsdjail.`**`get_enabled`**()
Return which jails are set to be run

CLI Example:

```
salt '*' jail.get_enabled
```

`salt.modules.freebsdjail.`**`is_enabled`**()
See if jail service is actually enabled on boot

CLI Example:

```
salt '*' jail.is_enabled <jail name>
```

`salt.modules.freebsdjail.`**`restart`**(*jail=''*)
Restart the specified jail or all, if none specified

CLI Example:

```
salt '*' jail.restart [<jail name>]
```

`salt.modules.freebsdjail.`**`show_config`**(*jail*)
Display specified jail's configuration

CLI Example:

```
salt '*' jail.show_config <jail name>
```

`salt.modules.freebsdjail.`**`start`**(*jail=''*)
Start the specified jail or all, if none specified

CLI Example:

```
salt '*' jail.start [<jail name>]
```

`salt.modules.freebsdjail.`**`status`**(*jail*)
See if specified jail is currently running

CLI Example:

```
salt '*' jail.status <jail name>
```

`salt.modules.freebsdjail.`**`stop`**(*jail=''*)
Stop the specified jail or all, if none specified

CLI Example:

```
salt '*' jail.stop [<jail name>]
```

`salt.modules.freebsdjail.`**`sysctl`**`()`

Dump all jail related kernel states (sysctl)

CLI Example:

```
salt '*' jail.sysctl
```

61.44 salt.modules.freebsdkmod

Module to manage FreeBSD kernel modules

`salt.modules.freebsdkmod.`**`available`**`()`

Return a list of all available kernel modules

CLI Example:

```
salt '*' kmod.available
```

`salt.modules.freebsdkmod.`**`check_available`**`(`*mod*`)`

Check to see if the specified kernel module is available

CLI Example:

```
salt '*' kmod.check_available kvm
```

`salt.modules.freebsdkmod.`**`load`**`(`*mod*`)`

Load the specified kernel module

CLI Example:

```
salt '*' kmod.load kvm
```

`salt.modules.freebsdkmod.`**`lsmod`**`()`

Return a dict containing information about currently loaded modules

CLI Example:

```
salt '*' kmod.lsmod
```

`salt.modules.freebsdkmod.`**`remove`**`(`*mod*`)`

Remove the specified kernel module

CLI Example:

```
salt '*' kmod.remove kvm
```

61.45 salt.modules.freebsdpkg

Remote package support using `pkg_add(1)`

> **Warning:** This module has been completely rewritten. Up to and including version 0.17.0, it supported `pkg_add(1)`, but checked for the existence of a pkgng local database and, if found, would provide some of pkgng's functionality. The rewrite of this module has removed all pkgng support, and moved it to the `pkgng` execution module. For verisions <= 0.17.0, the documentation here should not be considered accurate. If your Minion is running one of these versions, then the documentation for this module can be viewed using the `sys.doc` function:
>
> ```
> salt bsdminion sys.doc pkg
> ```

This module acts as the default package provider for FreeBSD 9 and older. If you need to use pkgng on a FreeBSD 9 system, you will need to override the `pkg` provider by setting the `providers` parameter in your Minion config file, in order to use pkgng.

```
providers:
  pkg: pkgng
```

More information on pkgng support can be found in the documentation for the `pkgng` module.

This module will respect the `PACKAGEROOT` and `PACKAGESITE` environment variables, if set, but these values can also be overridden in several ways:

1. **Salt configuration parameters.** The configuration parameters `freebsdpkg.PACKAGEROOT` and `freebsdpkg.PACKAGESITE` are recognized. These config parameters are looked up using `config.get` and can thus be specified in the Master config file, Grains, Pillar, or in the Minion config file. Example:

   ```
   freebsdpkg.PACKAGEROOT: ftp://ftp.freebsd.org/
   freebsdpkg.PACKAGESITE: ftp://ftp.freebsd.org/pub/FreeBSD/ports/ia64/packages-9-stable/Latest/
   ```

2. **CLI arguments.** Both the `packageroot` (used interchangeably with `fromrepo` for API compatibility) and `packagesite` CLI arguments are recognized, and override their config counterparts from section 1 above.

   ```
       salt -G 'os:FreeBSD' pkg.install zsh fromrepo=ftp://ftp2.freebsd.org/
       salt -G 'os:FreeBSD' pkg.install zsh packageroot=ftp://ftp2.freebsd.org/
       salt -G 'os:FreeBSD' pkg.install zsh packagesite=ftp://ftp2.freebsd.org/pub/FreeBSD/ports/ia

   .. note::

       These arguments can also be passed through in states:

       .. code-block:: yaml

           zsh:
             pkg.installed:
               - fromrepo: ftp://ftp2.freebsd.org/
   ```

`salt.modules.freebsdpkg.`**`file_dict`**`(`**packages*`)`

List the files that belong to a package, grouped by package. Not specifying any packages will return a list of _every_ file on the system's package database (not generally recommended).

CLI Examples:

```
salt '*' pkg.file_list httpd
salt '*' pkg.file_list httpd postfix
salt '*' pkg.file_list
```

`salt.modules.freebsdpkg.`**`file_list`**`(`**packages*`)`

List the files that belong to a package. Not specifying any packages will return a list of _every_ file on the system's package database (not generally recommended).

CLI Examples:

```
salt '*' pkg.file_list httpd
salt '*' pkg.file_list httpd postfix
salt '*' pkg.file_list
```

`salt.modules.freebsdpkg.`**`install`**`(`*name=None, refresh=False, fromrepo=None, pkgs=None, sources=None, **kwargs*`)`

Install package(s) using `pkg_add(1)`

name The name of the package to be installed.

refresh Whether or not to refresh the package database before installing.

fromrepo or packageroot Specify a package repository from which to install. Overrides the system default, as well as the PACKAGEROOT environment variable.

packagesite Specify the exact directory from which to install the remote package. Overrides the PACKAGESITE environment variable, if present.

Multiple Package Installation Options:

pkgs A list of packages to install from a software repository. Must be passed as a python list.

CLI Example:

```
salt '*' pkg.install pkgs='["foo", "bar"]'
```

sources A list of packages to install. Must be passed as a list of dicts, with the keys being package names, and the values being the source URI or local path to the package.

CLI Example:

```
salt '*' pkg.install sources='[{"foo": "salt://foo.deb"}, {"bar": "salt://bar.deb"}]'
```

Return a dict containing the new package names and versions:

```
{'<package>': {'old': '<old-version>',
               'new': '<new-version>'}}
```

CLI Example:

```
salt '*' pkg.install <package name>
```

`salt.modules.freebsdpkg.`**`latest_version`**`(`**names, **kwargs*`)`

`pkg_add(1)` is not capable of querying for remote packages, so this function will always return results as if there is no package available for install or upgrade.

CLI Example:

```
salt '*' pkg.latest_version <package name>
salt '*' pkg.latest_version <package1> <package2> <package3> ...
```

`salt.modules.freebsdpkg.`**`list_pkgs`**`(`*versions_as_list=False, with_origin=False, **kwargs*`)`

List the packages currently installed as a dict:

```
{'<package_name>': '<version>'}
```

with_origin [False] Return a nested dictionary containing both the origin name and version for each installed package.

New in version 2014.1.0: (Hydrogen)

CLI Example:

```
salt '*' pkg.list_pkgs
```

`salt.modules.freebsdpkg.`**`refresh_db`**`()`

`pkg_add(1)` does not use a local database of available packages, so this function simply returns `True`. it exists merely for API compatibility.

CLI Example:

```
salt '*' pkg.refresh_db
```

`salt.modules.freebsdpkg.`**`rehash`**`()`

Recomputes internal hash table for the PATH variable. Use whenever a new command is created during the current session.

CLI Example:

```
salt '*' pkg.rehash
```

`salt.modules.freebsdpkg.`**`remove`**`(`*name=None*, *pkgs=None*, ***kwargs*`)`

Remove packages using `pkg_delete(1)`

name The name of the package to be deleted.

Multiple Package Options:

pkgs A list of packages to delete. Must be passed as a python list. The `name` parameter will be ignored if this option is passed.

New in version 0.16.0.

Returns a dict containing the changes.

CLI Example:

```
salt '*' pkg.remove <package name>
salt '*' pkg.remove <package1>,<package2>,<package3>
salt '*' pkg.remove pkgs='["foo", "bar"]'
```

`salt.modules.freebsdpkg.`**`upgrade`**`()`

Upgrades are not supported with `pkg_add(1)`. This function is included for API compatibility only and always returns an empty dict.

CLI Example:

```
salt '*' pkg.upgrade
```

`salt.modules.freebsdpkg.`**`version`**`(`**names*, ***kwargs*`)`

Returns a string representing the package version or an empty string if not installed. If more than one package name is specified, a dict of name/version pairs is returned.

with_origin [False] Return a nested dictionary containing both the origin name and version for each specified package.

New in version 2014.1.0: (Hydrogen)

CLI Example:

```
salt '*' pkg.version <package name>
salt '*' pkg.version <package1> <package2> <package3> ...
```

61.46 salt.modules.freebsdports

Install software from the FreeBSD `ports(7)` system

New in version 2014.1.0: (Hydrogen)

This module allows you to install ports using `BATCH=yes` to bypass configuration prompts. It is recommended to use the the `ports state` to install ports, but it it also possible to use this module exclusively from the command line.

```
salt minion-id ports.config security/nmap IPV6=off
salt minion-id ports.install security/nmap
```

`salt.modules.freebsdports.`**`config`**`(`*name*, *reset=False*, ***kwargs*`)`

Modify configuration options for a given port. Multiple options can be specified. To see the available options for a port, use `ports.showconfig`.

name The port name, in `category/name` format

reset [False] If `True`, runs a `make rmconfig` for the port, clearing its configuration before setting the desired options

CLI Examples:

```
salt '*' ports.config security/nmap IPV6=off
```

`salt.modules.freebsdports.`**`deinstall`**`(`*name*`)`

De-install a port.

CLI Example:

```
salt '*' ports.deinstall security/nmap
```

`salt.modules.freebsdports.`**`install`**`(`*name*, *clean=True*`)`

Install a port from the ports tree. Installs using `BATCH=yes` for non-interactive building. To set config options for a given port, use `ports.config`.

clean [True] If `True`, cleans after installation. Equivalent to running `make install clean BATCH=yes`.

Note: It may be helpful to run this function using the `-t` option to set a higher timeout, since compiling a port may cause the Salt command to exceed the default timeout.

CLI Example:

```
salt -t 1200 '*' ports.install security/nmap
```

`salt.modules.freebsdports.`**`list_all`**`()`

Lists all ports available.

CLI Example:

```
salt '*' ports.list_all
```

Warning: Takes a while to run, and returns a **LOT** of output

`salt.modules.freebsdports.`**`rmconfig`**`(`*name*`)`

Clear the cached options for the specified port; run a `make rmconfig`

name The name of the port to clear

CLI Example:

```
salt '*' ports.rmconfig security/nmap
```

`salt.modules.freebsdports.`**`search`**`(`*name*`)`

Search for matches in the ports tree. Globs are supported, and the category is optional

CLI Examples:

```
salt '*' ports.search 'security/*'
salt '*' ports.search 'security/n*'
salt '*' ports.search nmap
```

Warning: Takes a while to run

`salt.modules.freebsdports.`**`showconfig`**`(`*name*, *default=False*, *dict_return=False*`)`

Show the configuration options for a given port.

default [False] Show the default options for a port (not necessarily the same as the current configuration)

dict_return [False] Instead of returning the output of `make showconfig`, return the data in an dictionary

CLI Example:

```
salt '*' ports.showconfig security/nmap
salt '*' ports.showconfig security/nmap default=True
```

`salt.modules.freebsdports.`**`update`**`(`*extract=False*`)`

Update the ports tree

extract [False] If `True`, runs a `portsnap extract` after fetching, should be used for first-time installation of the ports tree.

CLI Example:

```
salt '*' ports.update
```

61.47 salt.modules.freebsdservice

The service module for FreeBSD

`salt.modules.freebsdservice.`**`available`**`(`*name*`)`

Check that the given service is available.

CLI Example:

```
salt '*' service.available sshd
```

`salt.modules.freebsdservice.`**`disable`**`(`*name*, ***kwargs*`)`

Disable the named service to start at boot

Arguments the same as for enable()

CLI Example:

```
salt '*' service.disable <service name>
```

`salt.modules.freebsdservice.`**`disabled`**`(`*name*`)`

Return True if the named service is enabled, false otherwise

CLI Example:

```
salt '*' service.disabled <service name>
```

`salt.modules.freebsdservice.`**`enable`**`(`*name*, ***kwargs*`)`

Enable the named service to start at boot

name service name

config [/etc/rc.conf] Config file for managing service. If config value is empty string, then /etc/rc.conf.d/<service> used. See man rc.conf(5) for details.

Also service.config variable can be used to change default.

CLI Example:

```
salt '*' service.enable <service name>
```

`salt.modules.freebsdservice.`**`enabled`**`(`*name*`)`

Return True if the named service is enabled, false otherwise

name Service name

CLI Example:

```
salt '*' service.enabled <service name>
```

`salt.modules.freebsdservice.`**`get_all`**`()`

Return a list of all available services

CLI Example:

```
salt '*' service.get_all
```

`salt.modules.freebsdservice.`**`get_disabled`**`()`

Return what services are available but not enabled to start at boot

CLI Example:

```
salt '*' service.get_disabled
```

`salt.modules.freebsdservice.`**`get_enabled`**`()`

Return what services are set to run on boot

CLI Example:

```
salt '*' service.get_enabled
```

`salt.modules.freebsdservice.`**`missing`**`(`*name*`)`

The inverse of service.available. Returns `True` if the specified service is not available, otherwise returns `False`.

CLI Example:

```
salt '*' service.missing sshd
```

`salt.modules.freebsdservice.`**`reload_`**`(`*name*`)`

Restart the named service

CLI Example:

```
salt '*' service.reload <service name>
```

`salt.modules.freebsdservice.`**`restart`**`(`*name*`)`

Restart the named service

CLI Example:

```
salt '*' service.restart <service name>
```

`salt.modules.freebsdservice.`**`start`**(*name*)

Start the specified service

CLI Example:

```
salt '*' service.start <service name>
```

`salt.modules.freebsdservice.`**`status`**(*name*, *sig=None*)

Return the status for a service (True or False).

name Name of service

CLI Example:

```
salt '*' service.status <service name>
```

`salt.modules.freebsdservice.`**`stop`**(*name*)

Stop the specified service

CLI Example:

```
salt '*' service.stop <service name>
```

61.48 salt.modules.gem

Manage ruby gems.

`salt.modules.gem.`**`install`**(*gems*, *ruby=None*, *runas=None*, *version=None*, *rdoc=False*, *ri=False*)

Installs one or several gems.

gems The gems to install

ruby [None] If RVM is installed, the ruby version and gemset to use.

runas [None] The user to run gem as.

version [None] Specify the version to install for the gem. Doesn't play nice with multiple gems at once

rdoc [False] Generate RDoc documentation for the gem(s).

ri [False] Generate RI documentation for the gem(s).

CLI Example:

```
salt '*' gem.install vagrant
```

`salt.modules.gem.`**`list_`**(*prefix=''*, *ruby=None*, *runas=None*)

List locally installed gems.

prefix : Only list gems when the name matches this prefix.

ruby [None] If RVM is installed, the ruby version and gemset to use.

runas [None] The user to run gem as.

CLI Example:

```
salt '*' gem.list
```

`salt.modules.gem.`**`sources_add`**(*source_uri*, *ruby=None*, *runas=None*)

Add a gem source.

source_uri The source URI to add.

ruby [None] If RVM is installed, the ruby version and gemset to use.

runas [None] The user to run gem as.

CLI Example:

```
salt '*' gem.sources_add http://rubygems.org/
```

`salt.modules.gem.`**`sources_list`**(*ruby=None*, *runas=None*)

List the configured gem sources.

ruby [None] If RVM is installed, the ruby version and gemset to use.

runas [None] The user to run gem as.

CLI Example:

```
salt '*' gem.sources_list
```

`salt.modules.gem.`**`sources_remove`**(*source_uri*, *ruby=None*, *runas=None*)

Remove a gem source.

source_uri The source URI to remove.

ruby [None] If RVM is installed, the ruby version and gemset to use.

runas [None] The user to run gem as.

CLI Example:

```
salt '*' gem.sources_remove http://rubygems.org/
```

`salt.modules.gem.`**`uninstall`**(*gems*, *ruby=None*, *runas=None*)

Uninstall one or several gems.

gems The gems to uninstall.

ruby [None] If RVM is installed, the ruby version and gemset to use.

runas [None] The user to run gem as.

CLI Example:

```
salt '*' gem.uninstall vagrant
```

`salt.modules.gem.`**`update`**(*gems*, *ruby=None*, *runas=None*)

Update one or several gems.

gems The gems to update.

ruby [None] If RVM is installed, the ruby version and gemset to use.

runas [None] The user to run gem as.

CLI Example:

```
salt '*' gem.update vagrant
```

`salt.modules.gem.`**`update_system`**(*version=''*, *ruby=None*, *runas=None*)

Update rubygems.

version [(newest)] The version of rubygems to install.

ruby [None] If RVM is installed, the ruby version and gemset to use.

runas [None] The user to run gem as.

CLI Example:

```
salt '*' gem.update_system
```

61.49 salt.modules.gentoo_service

Top level package command wrapper, used to translate the os detected by grains to the correct service manager

`salt.modules.gentoo_service.`**`available`**`(`*name*`)`
Returns `True` if the specified service is available, otherwise returns `False`.

CLI Example:

```
salt '*' service.available sshd
```

`salt.modules.gentoo_service.`**`disable`**`(`*name*, ***kwargs*`)`
Disable the named service to start at boot

CLI Example:

```
salt '*' service.disable <service name>
```

`salt.modules.gentoo_service.`**`disabled`**`(`*name*`)`
Return True if the named service is enabled, false otherwise

CLI Example:

```
salt '*' service.disabled <service name>
```

`salt.modules.gentoo_service.`**`enable`**`(`*name*, ***kwargs*`)`
Enable the named service to start at boot

CLI Example:

```
salt '*' service.enable <service name>
```

`salt.modules.gentoo_service.`**`enabled`**`(`*name*`)`
Return True if the named service is enabled, false otherwise

CLI Example:

```
salt '*' service.enabled <service name>
```

`salt.modules.gentoo_service.`**`get_all`**`()`
Return all available boot services

CLI Example:

```
salt '*' service.get_all
```

`salt.modules.gentoo_service.`**`get_disabled`**`()`
Return a set of services that are installed but disabled

CLI Example:

```
salt '*' service.get_disabled
```

`salt.modules.gentoo_service.`**`get_enabled`**`()`

Return a list of service that are enabled on boot

CLI Example:

```
salt '*' service.get_enabled
```

`salt.modules.gentoo_service.`**`missing`**`(`*name*`)`

The inverse of service.available. Returns `True` if the specified service is not available, otherwise returns `False`.

CLI Example:

```
salt '*' service.missing sshd
```

`salt.modules.gentoo_service.`**`restart`**`(`*name*`)`

Restart the named service

CLI Example:

```
salt '*' service.restart <service name>
```

`salt.modules.gentoo_service.`**`start`**`(`*name*`)`

Start the specified service

CLI Example:

```
salt '*' service.start <service name>
```

`salt.modules.gentoo_service.`**`status`**`(`*name*, *sig=None*`)`

Return the status for a service, returns the PID or an empty string if the service is running or not, pass a signature to use to find the service via ps

CLI Example:

```
salt '*' service.status <service name> [service signature]
```

`salt.modules.gentoo_service.`**`stop`**`(`*name*`)`

Stop the specified service

CLI Example:

```
salt '*' service.stop <service name>
```

61.50 salt.modules.gentoolkitmod

Support for Gentoolkit

`salt.modules.gentoolkitmod.`**`eclean_dist`**`(`*destructive=False*, *package_names=False*, *size_limit=0*, *time_limit=0*, *fetch_restricted=False*, *exclude_file='/etc/eclean/distfiles.exclude'*`)`

Clean obsolete portage sources

destructive Only keep minimum for reinstallation

package_names Protect all versions of installed packages. Only meaningful if used with destructive=True

size_limit <size> Don't delete distfiles bigger than <size>. <size> is a size specification: “10M” is “ten megabytes”, “200K” is “two hundreds kilobytes”, etc. Units are: G, M, K and B.

time_limit <time> Don't delete distfiles files modified since <time> <time> is an amount of time: "1y" is "one year", "2w" is "two weeks", etc. Units are: y (years), m (months), w (weeks), d (days) and h (hours).

fetch_restricted Protect fetch-restricted files. Only meaningful if used with destructive=True

exclude_file Path to exclusion file. Default is /etc/eclean/distfiles.exclude This is the same default eclean-dist uses. Use None if this file exists and you want to ignore.

Returns a dict containing the cleaned, saved, and deprecated dists:

```
{'cleaned': {<dist file>: <size>},
 'deprecated': {<package>: <dist file>},
 'saved': {<package>: <dist file>},
 'total_cleaned': <size>}
```

CLI Example:

```
salt '*' gentoolkit.eclean_dist destructive=True
```

`salt.modules.gentoolkitmod.`**`eclean_pkg`**(*destructive=False, package_names=False, time_limit=0, exclude_file='/etc/eclean/packages.exclude'*)

Clean obsolete binary packages

destructive Only keep minimum for reinstallation

package_names Protect all versions of installed packages. Only meaningful if used with destructive=True

time_limit <time> Don't delete distfiles files modified since <time> <time> is an amount of time: "1y" is "one year", "2w" is "two weeks", etc. Units are: y (years), m (months), w (weeks), d (days) and h (hours).

exclude_file Path to exclusion file. Default is /etc/eclean/packages.exclude This is the same default eclean-pkg uses. Use None if this file exists and you want to ignore.

Returns a dict containing the cleaned binary packages:

```
{'cleaned': {<dist file>: <size>},
 'total_cleaned': <size>}
```

CLI Example:

```
salt '*' gentoolkit.eclean_pkg destructive=True
```

`salt.modules.gentoolkitmod.`**`glsa_check_list`**(*glsa_list*)

List the status of Gentoo Linux Security Advisories

glsa_list can contain an arbitrary number of GLSA ids, filenames containing GLSAs or the special identifiers 'all' and 'affected'

Returns a dict containing glsa ids with a description, status, and CVEs:

```
{<glsa_id>: {'description': <glsa_description>,
 'status': <glsa status>,
 'CVEs': [<list of CVEs>]}}
```

CLI Example:

```
salt '*' gentoolkit.glsa_check_list 'affected'
```

`salt.modules.gentoolkitmod.`**`revdep_rebuild`**(*lib=None*)

Fix up broken reverse dependencies

lib Search for reverse dependencies for a particular library rather than every library on the system. It can be a full path to a library or basic regular expression.

CLI Example:

```
salt '*' gentoolkit.revdep_rebuild
```

61.51 salt.modules.git

Support for the Git SCM

`salt.modules.git.`**`add`**(*cwd*, *file_name*, *user=None*, *opts=None*)

add a file to git

cwd The path to the Git repository

file_name Path to the file in the cwd

opts [None] Any additional options to add to the command line

user [None] Run git as a user other than what the minion runs as

CLI Example:

```
salt '*' git.add /path/to/git/repo /path/to/file
```

`salt.modules.git.`**`archive`**(*cwd*, *output*, *rev='HEAD'*, *fmt=None*, *prefix=None*, *user=None*)

Export a tarball from the repository

cwd The path to the Git repository

output The path to the archive tarball

rev: HEAD The revision to create an archive from

fmt: None Format of the resulting archive, zip and tar are commonly used

prefix [None] Prepend <prefix>/ to every filename in the archive

user [None] Run git as a user other than what the minion runs as

If `prefix` is not specified it defaults to the basename of the repo directory.

CLI Example:

```
salt '*' git.archive /path/to/repo /path/to/archive.tar.gz
```

`salt.modules.git.`**`checkout`**(*cwd*, *rev*, *force=False*, *opts=None*, *user=None*)

Checkout a given revision

cwd The path to the Git repository

rev The remote branch or revision to checkout

force [False] Force a checkout even if there might be overwritten changes

opts [None] Any additional options to add to the command line

user [None] Run git as a user other than what the minion runs as

CLI Examples:

```
salt '*' git.checkout /path/to/repo somebranch user=jeff
```

```
salt '*' git.checkout /path/to/repo opts='testbranch -- conf/file1 file2'
```

```
salt '*' git.checkout /path/to/repo rev=origin/mybranch opts=--track
```

`salt.modules.git.`**`clone`**(*cwd*, *repository*, *opts=None*, *user=None*, *identity=None*)

Clone a new repository

cwd The path to the Git repository

repository The git URI of the repository

opts [None] Any additional options to add to the command line

user [None] Run git as a user other than what the minion runs as

identity [None] A path to a private key to use over SSH

CLI Example:

```
salt '*' git.clone /path/to/repo git://github.com/saltstack/salt.git

salt '*' git.clone /path/to/repo.git\
        git://github.com/saltstack/salt.git '--bare --origin github'
```

`salt.modules.git.`**`commit`**(*cwd*, *message*, *user=None*, *opts=None*)

create a commit

cwd The path to the Git repository

message The commit message

opts [None] Any additional options to add to the command line

user [None] Run git as a user other than what the minion runs as

CLI Example:

```
salt '*' git.commit /path/to/git/repo 'The commit message'
```

`salt.modules.git.`**`config_get`**(*cwd*, *setting_name*, *user=None*)

Get a key from the git configuration file (.git/config) of the repository.

cwd The path to the Git repository

setting_name The name of the configuration key to get

user [None] Run git as a user other than what the minion runs as

CLI Example:

```
salt '*' git.config_get /path/to/repo user.email
```

`salt.modules.git.`**`config_set`**(*cwd*, *setting_name*, *setting_value*, *user=None*, *is_global=False*)

Set a key in the git configuration file (.git/config) of the repository or globally.

cwd The path to the Git repository

setting_name The name of the configuration key to set

setting_value The (new) value to set

user [None] Run git as a user other than what the minion runs as

is_global [False] Set to True to use the '–global' flag with 'git config'

CLI Example:

```
salt '*' git.config_set /path/to/repo user.email me@example.com
```

`salt.modules.git.`**`current_branch`**`(`*cwd*, *user=None*`)`

Returns the current branch name, if on a branch.

CLI Example:

```
salt '*' git.current_branch /path/to/repo
```

`salt.modules.git.`**`describe`**`(`*cwd*, *rev='HEAD'*, *user=None*`)`

Returns the git describe string (or the SHA hash if there are no tags) for the given revision

cwd The path to the Git repository

rev: HEAD The revision to describe

user [None] Run git as a user other than what the minion runs as

CLI Examples:

```
salt '*' git.describe /path/to/repo
```

```
salt '*' git.describe /path/to/repo develop
```

`salt.modules.git.`**`fetch`**`(`*cwd*, *opts=None*, *user=None*, *identity=None*`)`

Perform a fetch on the given repository

cwd The path to the Git repository

opts [None] Any additional options to add to the command line

user [None] Run git as a user other than what the minion runs as

identity [None] A path to a private key to use over SSH

CLI Example:

```
salt '*' git.fetch /path/to/repo '--all'
```

```
salt '*' git.fetch cwd=/path/to/repo opts='--all' user=johnny
```

`salt.modules.git.`**`init`**`(`*cwd*, *opts=None*, *user=None*`)`

Initialize a new git repository

cwd The path to the Git repository

opts [None] Any additional options to add to the command line

user [None] Run git as a user other than what the minion runs as

CLI Example:

```
salt '*' git.init /path/to/repo.git opts='--bare'
```

`salt.modules.git.`**`ls_remote`**`(`*cwd*, *repository='origin'*, *branch='master'*, *user=None*, *identity=None*`)`

Returns the upstream hash for any given URL and branch.

cwd The path to the Git repository

repository: origin The name of the repository to get the revision from. Can be the name of a remote, an URL, etc.

branch: master The name of the branch to get the revision from.

user [none] run git as a user other than what the minion runs as

identity [none] a path to a private key to use over ssh

CLI Example:

```
salt '*' git.ls_remote /pat/to/repo origin master
```

`salt.modules.git.`**`merge`**(*cwd*, *branch='@{upstream}'*, *opts=None*, *user=None*)

Merge a given branch

cwd The path to the Git repository

branch [@{upstream}] The remote branch or revision to merge into the current branch

opts [None] Any additional options to add to the command line

user [None] Run git as a user other than what the minion runs as

CLI Example:

```
salt '*' git.fetch /path/to/repo
salt '*' git.merge /path/to/repo @{upstream}
```

`salt.modules.git.`**`pull`**(*cwd*, *opts=None*, *user=None*, *identity=None*)

Perform a pull on the given repository

cwd The path to the Git repository

opts [None] Any additional options to add to the command line

user [None] Run git as a user other than what the minion runs as

identity [None] A path to a private key to use over SSH

CLI Example:

```
salt '*' git.pull /path/to/repo opts='--rebase origin master'
```

`salt.modules.git.`**`push`**(*cwd*, *remote_name*, *branch='master'*, *user=None*, *opts=None*, *identity=None*)

Push to remote

cwd The path to the Git repository

remote_name Name of the remote to push to

branch [master] Name of the branch to push

opts [None] Any additional options to add to the command line

user [None] Run git as a user other than what the minion runs as

identity [None] A path to a private key to use over SSH

CLI Example:

```
salt '*' git.push /path/to/git/repo remote-name
```

`salt.modules.git.`**`rebase`**(*cwd*, *rev='master'*, *opts=None*, *user=None*)

Rebase the current branch

cwd The path to the Git repository

rev [master] The revision to rebase onto the current branch

opts [None] Any additional options to add to the command line

user [None] Run git as a user other than what the minion runs as

CLI Example:

```
salt '*' git.rebase /path/to/repo master
salt '*' git.rebase /path/to/repo 'origin master'
```

That is the same as:

```
git rebase master
git rebase origin master
```

`salt.modules.git.`**`remote_get`**(*cwd*, *remote='origin'*, *user=None*)

get the fetch and push URL for a specified remote name

remote [origin] the remote name used to define the fetch and push URL

user [None] Run git as a user other than what the minion runs as

CLI Example:

```
salt '*' git.remote_get /path/to/repo
salt '*' git.remote_get /path/to/repo upstream
```

`salt.modules.git.`**`remote_set`**(*cwd*, *name='origin'*, *url=None*, *user=None*)

sets a remote with name and URL like git remote add <remote_name> <remote_url>

remote_name [origin] defines the remote name

remote_url [None] defines the remote URL; should not be None!

user [None] Run git as a user other than what the minion runs as

CLI Example:

```
salt '*' git.remote_set /path/to/repo remote_url=git@github.com:saltstack/salt.git
salt '*' git.remote_set /path/to/repo origin git@github.com:saltstack/salt.git
```

`salt.modules.git.`**`remotes`**(*cwd*, *user=None*)

Get remotes like git remote -v

cwd The path to the Git repository

user [None] Run git as a user other than what the minion runs as

CLI Example:

```
salt '*' git.remotes /path/to/repo
```

`salt.modules.git.`**`reset`**(*cwd*, *opts=None*, *user=None*)

Reset the repository checkout

cwd The path to the Git repository

opts [None] Any additional options to add to the command line

user [None] Run git as a user other than what the minion runs as

CLI Example:

```
salt '*' git.reset /path/to/repo master
```

`salt.modules.git.`**`revision`**(*cwd*, *rev='HEAD'*, *short=False*, *user=None*)

Returns the long hash of a given identifier (hash, branch, tag, HEAD, etc)

cwd The path to the Git repository

rev: HEAD The revision

short: False Return an abbreviated SHA1 git hash

user [None] Run git as a user other than what the minion runs as

CLI Example:

```
salt '*' git.revision /path/to/repo mybranch
```

`salt.modules.git.`**`rm`**(*cwd*, *file_name*, *user=None*, *opts=None*)

Remove a file from git

cwd The path to the Git repository

file_name Path to the file in the cwd

opts [None] Any additional options to add to the command line

user [None] Run git as a user other than what the minion runs as

CLI Example:

```
salt '*' git.rm /path/to/git/repo /path/to/file
```

`salt.modules.git.`**`stash`**(*cwd*, *opts=None*, *user=None*)

Stash changes in the repository checkout

cwd The path to the Git repository

opts [None] Any additional options to add to the command line

user [None] Run git as a user other than what the minion runs as

CLI Example:

```
salt '*' git.stash /path/to/repo master
```

`salt.modules.git.`**`status`**(*cwd*, *user=None*)

Return the status of the repository. The returned format uses the status codes of gits 'porcelain' output mode

cwd The path to the Git repository

user [None] Run git as a user other than what the minion runs as

CLI Example:

```
salt '*' git.status /path/to/git/repo
```

`salt.modules.git.`**`submodule`**(*cwd*, *init=True*, *opts=None*, *user=None*, *identity=None*)

Initialize git submodules

cwd The path to the Git repository

init [True] Ensure that new submodules are initialized

opts [None] Any additional options to add to the command line

user [None] Run git as a user other than what the minion runs as

identity [None] A path to a private key to use over SSH

CLI Example:

```
salt '*' git.submodule /path/to/repo.git/sub/repo
```

61.52 salt.modules.glance

Module for handling openstack glance calls.

optdepends

- glanceclient Python adapter

configuration This module is not usable until the following are specified either in a pillar or in the minion's config file:

```
keystone.user: admin
keystone.password: verybadpass
keystone.tenant: admin
keystone.tenant_id: f80919baedab48ec8931f200c65a50df
keystone.insecure: False   #(optional)
keystone.auth_url: 'http://127.0.0.1:5000/v2.0/'
```

If configuration for multiple openstack accounts is required, they can be set up as different configuration profiles: For example:

```
openstack1:
  keystone.user: admin
  keystone.password: verybadpass
  keystone.tenant: admin
  keystone.tenant_id: f80919baedab48ec8931f200c65a50df
  keystone.auth_url: 'http://127.0.0.1:5000/v2.0/'

openstack2:
  keystone.user: admin
  keystone.password: verybadpass
  keystone.tenant: admin
  keystone.tenant_id: f80919baedab48ec8931f200c65a50df
  keystone.auth_url: 'http://127.0.0.2:5000/v2.0/'
```

With this configuration in place, any of the keystone functions can make use of a configuration profile by declaring it explicitly. For example:

```
salt '*' glance.image_list profile=openstack1
```

`salt.modules.glance.`**`image_create`**(*profile=None*, ***kwargs*)

Create an image (glance image-create)

CLI Example:

```
salt '*' glance.image_create name=f16-jeos is_public=true \
         disk_format=qcow2 container_format=ovf \
         copy_from=http://berrange.fedorapeople.org/images/2012-02-29/f16-x86_64-openstack-sda.q
```

For all possible values, run `glance help image-create` on the minion.

`salt.modules.glance.`**`image_delete`**(*id=None*, *name=None*, *profile=None*)

Delete an image (glance image-delete)

CLI Examples:

```
salt '*' glance.image_delete c2eb2eb0-53e1-4a80-b990-8ec887eae7df
salt '*' glance.image_delete id=c2eb2eb0-53e1-4a80-b990-8ec887eae7df
salt '*' glance.image_delete name=f16-jeos
```

salt.modules.glance.**image_list**(*id=None*, *profile=None*)
Return a list of available images (glance image-list)

CLI Example:

```
salt '*' glance.image_list
```

salt.modules.glance.**image_show**(*id=None*, *name=None*, *profile=None*)
Return details about a specific image (glance image-show)

CLI Example:

```
salt '*' glance.image_get
```

61.53 salt.modules.gnomedesktop

GNOME implementations

salt.modules.gnomedesktop.**get**(*schema=None*, *key=None*, *user=None*, ***kwargs*)
Get key in a particular GNOME schema

CLI Example:

```
salt '*' gnome.get user=<username> schema=org.gnome.desktop.screensaver key=idle-activation-enab
```

salt.modules.gnomedesktop.**getClockFormat**(***kwargs*)
Return the current clock format, either 12h or 24h format.

CLI Example:

```
salt '*' gnome.getClockFormat user=<username>
```

salt.modules.gnomedesktop.**getClockShowDate**(***kwargs*)
Return the current setting, if the date is shown in the clock

CLI Example:

```
salt '*' gnome.getClockShowDate user=<username>
```

salt.modules.gnomedesktop.**getIdleActivation**(***kwargs*)
Get whether the idle activation is enabled

CLI Example:

```
salt '*' gnome.getIdleActivation user=<username>
```

salt.modules.gnomedesktop.**getIdleDelay**(***kwargs*)
Return the current idle delay setting in seconds

CLI Example:

```
salt '*' gnome.getIdleDelay user=<username>
```

salt.modules.gnomedesktop.**ping**(***kwargs*)
A test to ensure the GNOME module is loaded

CLI Example:

```
salt '*' gnome.ping user=<username>
```

salt.modules.gnomedesktop.**setClockFormat**(clockFormat, **kwargs)

Set the clock format, either 12h or 24h format.

CLI Example:

```
salt '*' gnome.setClockFormat <12h|24h> user=<username>
```

salt.modules.gnomedesktop.**setClockShowDate**(kvalue, **kwargs)

Set whether the date is visable in the clock

CLI Example:

```
salt '*' gnome.setClockShowDate <True|False> user=<username>
```

salt.modules.gnomedesktop.**setIdleActivation**(kvalue, **kwargs)

Set whether the idle activation is enabled

CLI Example:

```
salt '*' gnome.setIdleActivation <True|False> user=<username>
```

salt.modules.gnomedesktop.**setIdleDelay**(delaySeconds, **kwargs)

Set the current idle delay setting in seconds

CLI Example:

```
salt '*' gnome.setIdleDelay <seconds> user=<username>
```

salt.modules.gnomedesktop.**set_**(schema=None, key=None, user=None, value=None, **kwargs)

Set key in a particular GNOME schema

CLI Example:

```
salt '*' gnome.set user=<username> schema=org.gnome.desktop.screensaver key=idle-activation-enab
```

61.54 salt.modules.grains

Return/control aspects of the grains data

salt.modules.grains.**append**(key, val)

New in version 0.17.0.

Append a value to a list in the grains config file

CLI Example:

```
salt '*' grains.append key val
```

salt.modules.grains.**delval**(key, destructive=False)

New in version 0.17.0.

Delete a grain from the grains config file

Parameters **Destructive** – Delete the key, too. Defaults to False.

CLI Example:

```
salt '*' grains.delval key
```

`salt.modules.grains.`**`filter_by`**(*lookup_dict*, *grain='os_family'*, *merge=None*, *default='default'*)

New in version 0.17.0.

Look up the given grain in a given dictionary for the current OS and return the result

Although this may occasionally be useful at the CLI, the primary intent of this function is for use in Jinja to make short work of creating lookup tables for OS-specific data. For example:

```
{% set apache = salt['grains.filter_by']({
    'Debian': {'pkg': 'apache2', 'srv': 'apache2'},
    'RedHat': {'pkg': 'httpd', 'srv': 'httpd'},
    'default': 'Debian',
}) %}

myapache:
  pkg:
    - installed
    - name: {{ apache.pkg }}
  service:
    - running
    - name: {{ apache.srv }}
```

Values in the lookup table may be overridden by values in Pillar. An example Pillar to override values in the example above could be as follows:

```
apache:
  lookup:
    pkg: apache_13
    srv: apache
```

The call to `filter_by()` would be modified as follows to reference those Pillar values:

```
{% set apache = salt['grains.filter_by']({
    ...
}, merge=salt['pillar.get']('apache:lookup')) %}
```

Parameters

- **lookup_dict** – A dictionary, keyed by a grain, containing a value or values relevant to systems matching that grain. For example, a key could be the grain for an OS and the value could the name of a package on that particular OS.
- **grain** – The name of a grain to match with the current system's grains. For example, the value of the "os_family" grain for the current system could be used to pull values from the `lookup_dict` dictionary.
- **merge** – A dictionary to merge with the `lookup_dict` before doing the lookup. This allows Pillar to override the values in the `lookup_dict`. This could be useful, for example, to override the values for non-standard package names such as when using a different Python version from the default Python version provided by the OS (e.g., `python26-mysql` instead of `python-mysql`).
- **default** – default lookup_dict's key used if the grain does not exists or if the grain value has no match on lookup_dict.

  New in version 2014.1.

CLI Example:

```
salt '*' grains.filter_by '{Debian: Debheads rule, RedHat: I love my hat}'
# this one will render {D: {E: I, G: H}, J: K}
salt '*' grains.filter_by '{A: B, C: {D: {E: F,G: H}}}' 'xxx' '{D: {E: I},J: K}' 'C'
```

salt.modules.grains.**get**(*key*, *default=’‘*)

Attempt to retrieve the named value from grains, if the named value is not available return the passed default. The default return is an empty string.

The value can also represent a value in a nested dict using a ”:” delimiter for the dict. This means that if a dict in grains looks like this:

```
{'pkg': {'apache': 'httpd'}}
```

To retrieve the value associated with the apache key in the pkg dict this key can be passed:

```
pkg:apache
```

CLI Example:

```
salt '*' grains.get pkg:apache
```

salt.modules.grains.**get_or_set_hash**(*name*, *length=8*, *chars=’abcdefghijklmnopqrstuvwxyz0123456789!@#$%^&*(-_=+)’*)

Perform a one-time generation of a hash and write it to the local grains. If that grain has already been set return the value instead.

This is useful for generating passwords or keys that are specific to a single minion that don’t need to be stored somewhere centrally.

State Example:

```
some_mysql_user:
  mysql_user:
    - present
    - host: localhost
    - password: {{ grains.get_or_set_hash('mysql:some_mysql_user') }}
```

CLI Example:

```
salt '*' grains.get_or_set_hash 'django:SECRET_KEY' 50
```

salt.modules.grains.**has_value**(*key*)

Determine whether a named value exists in the grains dictionary.

Given a grains dictionary that contains the following structure:

```
{'pkg': {'apache': 'httpd'}}
```

One would determine if the apache key in the pkg dict exists by:

```
pkg:apache
```

CLI Example:

```
salt '*' grains.has_value pkg:apache
```

salt.modules.grains.**item**(**args*, ***kwargs*)

Return one or more grains

CLI Example:

```
salt '*' grains.item os
salt '*' grains.item os osrelease oscodename
```

Sanitized CLI Example:

```
salt '*' grains.item host sanitize=True
```

`salt.modules.grains.`**`items`**`(`*sanitize=False*`)`

Return all of the minion's grains

CLI Example:

```
salt '*' grains.items
```

Sanitized CLI Example:

```
salt '*' grains.items sanitize=True
```

`salt.modules.grains.`**`ls`**`()`

Return a list of all available grains

CLI Example:

```
salt '*' grains.ls
```

`salt.modules.grains.`**`remove`**`(`*key*, *val*`)`

New in version 0.17.0.

Remove a value from a list in the grains config file

CLI Example:

```
salt '*' grains.remove key val
```

`salt.modules.grains.`**`setval`**`(`*key*, *val*, *destructive=False*`)`

Set a grains value in the grains config file

Parameters **Destructive** – If an operation results in a key being removed, delete the key, too. Defaults to False.

CLI Example:

```
salt '*' grains.setval key val
salt '*' grains.setval key "{'sub-key': 'val', 'sub-key2': 'val2'}"
```

61.55 salt.modules.groupadd

Manage groups on Linux and OpenBSD

`salt.modules.groupadd.`**`add`**`(`*name*, *gid=None*, *system=False*`)`

Add the specified group

CLI Example:

```
salt '*' group.add foo 3456
```

`salt.modules.groupadd.`**`chgid`**`(`*name*, *gid*`)`

Change the gid for a named group

CLI Example:

```
salt '*' group.chgid foo 4376
```

`salt.modules.groupadd.`**`delete`**`(`*name*`)`

Remove the named group

CLI Example:

```
salt '*' group.delete foo
```

`salt.modules.groupadd.`**`getent`**`(`*refresh=False*`)`

Return info on all groups

CLI Example:

```
salt '*' group.getent
```

`salt.modules.groupadd.`**`info`**`(`*name*`)`

Return information about a group

CLI Example:

```
salt '*' group.info foo
```

61.56 salt.modules.grub_legacy

Support for GRUB Legacy

`salt.modules.grub_legacy.`**`conf`**`()`

Parse GRUB conf file

CLI Example:

```
salt '*' grub.conf
```

`salt.modules.grub_legacy.`**`version`**`()`

Return server version from grub –version

CLI Example:

```
salt '*' grub.version
```

61.57 salt.modules.guestfs

Interact with virtual machine images via libguestfs

depends

- libguestfs

`salt.modules.guestfs.`**`mount`**`(`*location, access='rw'*`)`

Mount an image

CLI Example:

```
salt '*' guest.mount /srv/images/fedora.qcow
```

61.58 salt.modules.hg

Support for the Mercurial SCM

`salt.modules.hg.`**`archive`**(*cwd*, *output*, *rev='tip'*, *fmt=None*, *prefix=None*, *user=None*)

Export a tarball from the repository

cwd The path to the Mercurial repository

output The path to the archive tarball

rev: tip The revision to create an archive from

fmt: None Format of the resulting archive. Mercurial supports: tar, tbz2, tgz, zip, uzip, and files formats.

prefix [None] Prepend <prefix>/ to every filename in the archive

user [None] Run hg as a user other than what the minion runs as

If `prefix` is not specified it defaults to the basename of the repo directory.

CLI Example:

```
salt '*' hg.archive /path/to/repo output=/tmp/archive.tgz fmt=tgz
```

`salt.modules.hg.`**`clone`**(*cwd*, *repository*, *opts=None*, *user=None*)

Clone a new repository

cwd The path to the Mercurial repository

repository The hg URI of the repository

opts [None] Any additional options to add to the command line

user [None] Run hg as a user other than what the minion runs as

CLI Example:

```
salt '*' hg.clone /path/to/repo https://bitbucket.org/birkenfeld/sphinx
```

`salt.modules.hg.`**`describe`**(*cwd*, *rev='tip'*, *user=None*)

Mimick git describe and return an identifier for the given revision

cwd The path to the Mercurial repository

rev: tip The path to the archive tarball

user [None] Run hg as a user other than what the minion runs as

CLI Example:

```
salt '*' hg.describe /path/to/repo
```

`salt.modules.hg.`**`pull`**(*cwd*, *opts=None*, *user=None*)

Perform a pull on the given repository

cwd The path to the Mercurial repository

opts [None] Any additional options to add to the command line

user [None] Run hg as a user other than what the minion runs as

CLI Example:

```
salt '*' hg.pull /path/to/repo '-u'
```

`salt.modules.hg.`**`revision`**(*cwd*, *rev='tip'*, *short=False*, *user=None*)

Returns the long hash of a given identifier (hash, branch, tag, HEAD, etc)

cwd The path to the Mercurial repository

rev: tip The revision

short: False Return an abbreviated commit hash

user [None] Run hg as a user other than what the minion runs as

CLI Example:

```
salt '*' hg.revision /path/to/repo mybranch
```

`salt.modules.hg.`**`update`**(*cwd*, *rev*, *force=False*, *user=None*)

Update to a given revision

cwd The path to the Mercurial repository

rev The revision to update to

force [False] Force an update

user [None] Run hg as a user other than what the minion runs as

CLI Example:

```
salt devserver1 hg.update /path/to/repo somebranch
```

61.59 salt.modules.hosts

Manage the information in the hosts file

`salt.modules.hosts.`**`add_host`**(*ip*, *alias*)

Add a host to an existing entry, if the entry is not in place then create it with the given host

CLI Example:

```
salt '*' hosts.add_host <ip> <alias>
```

`salt.modules.hosts.`**`get_alias`**(*ip*)

Return the list of aliases associated with an ip

CLI Example:

```
salt '*' hosts.get_alias <ip addr>
```

`salt.modules.hosts.`**`get_ip`**(*host*)

Return the ip associated with the named host

CLI Example:

```
salt '*' hosts.get_ip <hostname>
```

`salt.modules.hosts.`**`has_pair`**(*ip*, *alias*)

Return true if the alias is set

CLI Example:

```
salt '*' hosts.has_pair <ip> <alias>
```

`salt.modules.hosts.`**`list_hosts`**`()`

Return the hosts found in the hosts file in this format:

```
{'<ip addr>': ['alias1', 'alias2', ...]}
```

CLI Example:

```
salt '*' hosts.list_hosts
```

`salt.modules.hosts.`**`rm_host`**`(`*ip*, *alias*`)`

Remove a host entry from the hosts file

CLI Example:

```
salt '*' hosts.rm_host <ip> <alias>
```

`salt.modules.hosts.`**`set_host`**`(`*ip*, *alias*`)`

Set the host entry in the hosts file for the given ip, this will overwrite any previous entry for the given ip

CLI Example:

```
salt '*' hosts.set_host <ip> <alias>
```

61.60 salt.modules.htpasswd

Support for htpasswd command

The functions here will load inside the webutil module. This allows other functions that don't use htpasswd to use the webutil module name.

`salt.modules.htpasswd.`**`useradd`**`(`*pwfile*, *user*, *password*, *opts=''*`)`

Add an HTTP user using the htpasswd command. If the htpasswd file does not exist, it will be created. Valid options that can be passed are:

> n Don't update file; display results on stdout. m Force MD5 encryption of the password (default). d Force CRYPT encryption of the password. p Do not encrypt the password (plaintext). s Force SHA encryption of the password.

CLI Examples:

```
salt '*' webutil.useradd /etc/httpd/htpasswd larry badpassword
salt '*' webutil.useradd /etc/httpd/htpasswd larry badpass opts=ns
```

`salt.modules.htpasswd.`**`userdel`**`(`*pwfile*, *user*`)`

Delete an HTTP user from the specified htpasswd file.

CLI Examples:

```
salt '*' webutil.userdel /etc/httpd/htpasswd larry
```

61.61 salt.modules.img

Virtual machine image management tools

`salt.modules.img.`**`bootstrap`**`(`*location*, *size*, *fmt*`)`

HIGHLY EXPERIMENTAL Bootstrap a virtual machine image

location: The location to create the image

size: The size of the image to create in megabytes

fmt: The image format, raw or qcow2

CLI Example:

```
salt '*' qemu_nbd.bootstrap /srv/salt-images/host.qcow 4096 qcow2
```

`salt.modules.img.`**`mnt_image`**`(`*location*`)`

Mount the named image and return the mount point

CLI Example:

```
salt '*' img.mount_image /tmp/foo
```

`salt.modules.img.`**`mount_image`**`(`*location*`)`

Mount the named image and return the mount point

CLI Example:

```
salt '*' img.mount_image /tmp/foo
```

`salt.modules.img.`**`umount_image`**`(`*mnt*`)`

Unmount an image mountpoint

CLI Example:

```
salt '*' img.umount_image /mnt/foo
```

61.62 salt.modules.iptables

Support for iptables

`salt.modules.iptables.`**`append`**`(`*table='filter', chain=None, rule=None, family='ipv4'*`)`

Append a rule to the specified table/chain.

This function accepts a rule in a standard iptables command format, starting with the chain. Trying to force users to adapt to a new method of creating rules would be irritating at best, and we already have a parser that can handle it.

CLI Example:

```
salt '*' iptables.append filter INPUT \
    rule='-m state --state RELATED,ESTABLISHED -j ACCEPT'

IPv6:
salt '*' iptables.append filter INPUT \
    rule='-m state --state RELATED,ESTABLISHED -j ACCEPT' \
    family=ipv6
```

`salt.modules.iptables.`**`build_rule`**`(`*table=None, chain=None, command=None, position='', full=None, family='ipv4', **kwargs*`)`

Build a well-formatted iptables rule based on kwargs. Long options must be used (*–jump* instead of *-j*) because they will have the – added to them. A *table* and *chain* are not required, unless *full* is True.

If *full* is *True*, then *table*, *chain* and *command* are required. *command* may be specified as either a short option ('I') or a long option (*–insert*). This will return the iptables command, exactly as it would be used from the command line.

If a position is required (as with *-I* or *-D*), it may be specified as *position*. This will only be useful if *full* is True.

If *connstate* is passed in, it will automatically be changed to *state*.

CLI Examples:

```
salt '*' iptables.build_rule match=state \
    connstate=RELATED,ESTABLISHED jump=ACCEPT
salt '*' iptables.build_rule filter INPUT command=I position=3 \
    full=True match=state state=RELATED,ESTABLISHED jump=ACCEPT

IPv6:
salt '*' iptables.build_rule match=state \
    connstate=RELATED,ESTABLISHED jump=ACCEPT \
    family=ipv6
salt '*' iptables.build_rule filter INPUT command=I position=3 \
    full=True match=state state=RELATED,ESTABLISHED jump=ACCEPT \
    family=ipv6
```

`salt.modules.iptables.`**`check`**(*table='filter'*, *chain=None*, *rule=None*, *family='ipv4'*)

Check for the existance of a rule in the table and chain

This function accepts a rule in a standard iptables command format, starting with the chain. Trying to force users to adapt to a new method of creating rules would be irritating at best, and we already have a parser that can handle it.

CLI Example:

```
salt '*' iptables.check filter INPUT \
    rule='-m state --state RELATED,ESTABLISHED -j ACCEPT'

IPv6:
salt '*' iptables.check filter INPUT \
    rule='-m state --state RELATED,ESTABLISHED -j ACCEPT' \
    family=ipv6
```

`salt.modules.iptables.`**`check_chain`**(*table='filter'*, *chain=None*, *family='ipv4'*)

New in version 2014.1.0: (Hydrogen)

Check for the existance of a chain in the table

CLI Example:

```
salt '*' iptables.check_chain filter INPUT

IPv6:
salt '*' iptables.check_chain filter INPUT family=ipv6
```

`salt.modules.iptables.`**`delete`**(*table*, *chain=None*, *position=None*, *rule=None*, *family='ipv4'*)

Delete a rule from the specified table/chain, specifying either the rule in its entirety, or the rule's position in the chain.

This function accepts a rule in a standard iptables command format, starting with the chain. Trying to force users to adapt to a new method of creating rules would be irritating at best, and we already have a parser that can handle it.

CLI Examples:

```
salt '*' iptables.delete filter INPUT position=3
salt '*' iptables.delete filter INPUT \
    rule='-m state --state RELATED,ESTABLISHED -j ACCEPT'

IPv6:
```

```
salt '*' iptables.delete filter INPUT position=3 family=ipv6
salt '*' iptables.delete filter INPUT \
    rule='-m state --state RELATED,ESTABLISHED -j ACCEPT' \
    family=ipv6
```

`salt.modules.iptables.`**`delete_chain`**(*table='filter'*, *chain=None*, *family='ipv4'*)

New in version 2014.1.0: (Hydrogen)

Delete custom chain to the specified table.

CLI Example:

```
salt '*' iptables.delete_chain filter CUSTOM_CHAIN

IPv6:
salt '*' iptables.delete_chain filter CUSTOM_CHAIN family=ipv6
```

`salt.modules.iptables.`**`flush`**(*table='filter'*, *chain=''*, *family='ipv4'*)

Flush the chain in the specified table, flush all chains in the specified table if not specified chain.

CLI Example:

```
salt '*' iptables.flush filter INPUT

IPv6:
salt '*' iptables.flush filter INPUT family=ipv6
```

`salt.modules.iptables.`**`get_policy`**(*table='filter'*, *chain=None*, *family='ipv4'*)

Return the current policy for the specified table/chain

CLI Example:

```
salt '*' iptables.get_policy filter INPUT

IPv6:
salt '*' iptables.get_policy filter INPUT family=ipv6
```

`salt.modules.iptables.`**`get_rules`**(*family='ipv4'*)

Return a data structure of the current, in-memory rules

CLI Example:

```
salt '*' iptables.get_rules

IPv6:
salt '*' iptables.get_rules family=ipv6
```

`salt.modules.iptables.`**`get_saved_policy`**(*table='filter'*, *chain=None*, *conf_file=None*, *family='ipv4'*)

Return the current policy for the specified table/chain

CLI Examples:

```
salt '*' iptables.get_saved_policy filter INPUT
salt '*' iptables.get_saved_policy filter INPUT \
    conf_file=/etc/iptables.saved

IPv6:
salt '*' iptables.get_saved_policy filter INPUT family=ipv6
salt '*' iptables.get_saved_policy filter INPUT \
    conf_file=/etc/iptables.saved family=ipv6
```

`salt.modules.iptables.`**`get_saved_rules`**(*conf_file=None*, *family='ipv4'*)

Return a data structure of the rules in the conf file

CLI Example:

```
salt '*' iptables.get_saved_rules

IPv6:
salt '*' iptables.get_saved_rules family=ipv6
```

`salt.modules.iptables.`**`insert`**(*table='filter'*, *chain=None*, *position=None*, *rule=None*, *family='ipv4'*)

Insert a rule into the specified table/chain, at the specified position.

This function accepts a rule in a standard iptables command format, starting with the chain. Trying to force users to adapt to a new method of creating rules would be irritating at best, and we already have a parser that can handle it.

CLI Examples:

```
salt '*' iptables.insert filter INPUT position=3 \
    rule='-m state --state RELATED,ESTABLISHED -j ACCEPT'

IPv6:
salt '*' iptables.insert filter INPUT position=3 \
    rule='-m state --state RELATED,ESTABLISHED -j ACCEPT' \
    family=ipv6
```

`salt.modules.iptables.`**`new_chain`**(*table='filter'*, *chain=None*, *family='ipv4'*)

New in version 2014.1.0: (Hydrogen)

Create new custom chain to the specified table.

CLI Example:

```
salt '*' iptables.new_chain filter CUSTOM_CHAIN

IPv6:
salt '*' iptables.new_chain filter CUSTOM_CHAIN family=ipv6
```

`salt.modules.iptables.`**`save`**(*filename=None*, *family='ipv4'*)

Save the current in-memory rules to disk

CLI Example:

```
salt '*' iptables.save /etc/sysconfig/iptables

IPv6:
salt '*' iptables.save /etc/sysconfig/iptables family=ipv6
```

`salt.modules.iptables.`**`set_policy`**(*table='filter'*, *chain=None*, *policy=None*, *family='ipv4'*)

Set the current policy for the specified table/chain

CLI Example:

```
salt '*' iptables.set_policy filter INPUT ACCEPT

IPv6:
salt '*' iptables.set_policy filter INPUT ACCEPT family=ipv6
```

`salt.modules.iptables.`**`version`**(*family='ipv4'*)

Return version from iptables –version

CLI Example:

```
salt '*' iptables.version

IPv6:
salt '*' iptables.version family=ipv6
```

61.63 salt.modules.key

Functions to view the minion's public key information

`salt.modules.key.`**`finger`**`()`

Return the minion's public key fingerprint

CLI Example:

```
salt '*' key.finger
```

61.64 salt.modules.keyboard

Module for managing keyboards on POSIX-like systems.

`salt.modules.keyboard.`**`get_sys`**`()`

Get current system keyboard setting

CLI Example:

```
salt '*' keyboard.get_sys
```

`salt.modules.keyboard.`**`get_x`**`()`

Get current X keyboard setting

CLI Example:

```
salt '*' keyboard.get_x
```

`salt.modules.keyboard.`**`set_sys`**`(`*layout*`)`

Set current system keyboard setting

CLI Example:

```
salt '*' keyboard.set_sys dvorak
```

`salt.modules.keyboard.`**`set_x`**`(`*layout*`)`

Set current X keyboard setting

CLI Example:

```
salt '*' keyboard.set_x dvorak
```

61.65 salt.modules.keystone

Module for handling openstack keystone calls.

optdepends

- keystoneclient Python adapter

configuration This module is not usable until the following are specified either in a pillar or in the minion's config file:

```
keystone.user: admin
keystone.password: verybadpass
keystone.tenant: admin
keystone.tenant_id: f80919baedab48ec8931f200c65a50df
keystone.auth_url: 'http://127.0.0.1:5000/v2.0/'

OR (for token based authentication)

keystone.token: 'ADMIN'
keystone.endpoint: 'http://127.0.0.1:35357/v2.0'
```

If configuration for multiple openstack accounts is required, they can be set up as different configuration profiles: For example:

```
openstack1:
  keystone.user: admin
  keystone.password: verybadpass
  keystone.tenant: admin
  keystone.tenant_id: f80919baedab48ec8931f200c65a50df
  keystone.auth_url: 'http://127.0.0.1:5000/v2.0/'

openstack2:
  keystone.user: admin
  keystone.password: verybadpass
  keystone.tenant: admin
  keystone.tenant_id: f80919baedab48ec8931f200c65a50df
  keystone.auth_url: 'http://127.0.0.2:5000/v2.0/'
```

With this configuration in place, any of the keystone functions can make use of a configuration profile by declaring it explicitly. For example:

```
salt '*' keystone.tenant_list profile=openstack1
```

`salt.modules.keystone.`**`auth`**(*profile=None*, ***connection_args*)

Set up keystone credentials

Only intended to be used within Keystone-enabled modules

`salt.modules.keystone.`**`ec2_credentials_create`**(*user_id=None*, *name=None*, *tenant_id=None*, *tenant=None*, *profile=None*, ***connection_args*)

Create EC2-compatibile credentials for user per tenant

CLI Examples:

```
salt '*' keystone.ec2_credentials_create name=admin tenant=admin
salt '*' keystone.ec2_credentials_create             user_id=c965f79c4f864eaaa9c3b41904e67082
```

`salt.modules.keystone.`**`ec2_credentials_delete`**(*user_id=None*, *name=None*, *access_key=None*, *profile=None*, ***connection_args*)

Delete EC2-compatibile credentials

CLI Examples:

```
salt '*' keystone.ec2_credentials_delete          860f8c2c38ca4fab989f9bc56a061a64
    access_key=5f66d2f24f604b8bb9cd28886106f442
salt '*' keystone.ec2_credentials_delete name=admin           access_key=5f66d2f24f604b8bb9cd2
```

`salt.modules.keystone.`**`ec2_credentials_get`**(*user_id=None*, *name=None*, *access=None*, *profile=None*, ***connection_args*)

Return ec2_credentials for a user (keystone ec2-credentials-get)

CLI Examples:

```
salt '*' keystone.ec2_credentials_get c965f79c4f864eaaa9c3b41904e67082 access=722787eb5408491586
salt '*' keystone.ec2_credentials_get user_id=c965f79c4f864eaaa9c3b41904e67082 access=722787eb54
salt '*' keystone.ec2_credentials_get name=nova access=722787eb540849158668370dc627ec5f
```

`salt.modules.keystone.`**`ec2_credentials_list`**(*user_id=None*, *name=None*, *profile=None*, ***connection_args*)

Return a list of ec2_credentials for a specific user (keystone ec2-credentials-list)

CLI Examples:

```
salt '*' keystone.ec2_credentials_list 298ce377245c4ec9b70e1c639c89e654
salt '*' keystone.ec2_credentials_list user_id=298ce377245c4ec9b70e1c639c89e654
salt '*' keystone.ec2_credentials_list name=jack
```

`salt.modules.keystone.`**`endpoint_get`**(*service*, *profile=None*, ***connection_args*)

Return a specific endpoint (keystone endpoint-get)

CLI Example:

```
salt '*' keystone.endpoint_get ec2
```

`salt.modules.keystone.`**`endpoint_list`**(*profile=None*, ***connection_args*)

Return a list of available endpoints (keystone endpoints-list)

CLI Example:

```
salt '*' keystone.endpoint_list
```

`salt.modules.keystone.`**`role_create`**(*name*, *profile=None*, ***connection_args*)

Create named role

```
salt '*' keystone.role_create admin
```

`salt.modules.keystone.`**`role_delete`**(*role_id=None*, *name=None*, *profile=None*, ***connection_args*)

Delete a role (keystone role-delete)

CLI Examples:

```
salt '*' keystone.role_delete c965f79c4f864eaaa9c3b41904e67082
salt '*' keystone.role_delete role_id=c965f79c4f864eaaa9c3b41904e67082
salt '*' keystone.role_delete name=admin
```

`salt.modules.keystone.`**`role_get`**(*role_id=None*, *name=None*, *profile=None*, ***connection_args*)

Return a specific roles (keystone role-get)

CLI Examples:

```
salt '*' keystone.role_get c965f79c4f864eaaa9c3b41904e67082
salt '*' keystone.role_get role_id=c965f79c4f864eaaa9c3b41904e67082
salt '*' keystone.role_get name=nova
```

`salt.modules.keystone.`**`role_list`**(*profile=None*, ***connection_args*)

Return a list of available roles (keystone role-list)

CLI Example:

```
salt '*' keystone.role_list
```

`salt.modules.keystone.`**`service_create`**(*name*, *service_type*, *description=None*, *profile=None*, ***connection_args*)

Add service to Keystone service catalog

CLI Examples:

```
salt '*' keystone.service_create nova compute                'OpenStack Compute Service'
```

`salt.modules.keystone.`**`service_delete`**(*service_id=None*, *name=None*, *profile=None*, ***connection_args*)

Delete a service from Keystone service catalog

CLI Examples:

```
salt '*' keystone.service_delete c965f79c4f864eaaa9c3b41904e67082
salt '*' keystone.service_delete name=nova
```

`salt.modules.keystone.`**`service_get`**(*service_id=None*, *name=None*, *profile=None*, ***connection_args*)

Return a specific services (keystone service-get)

CLI Examples:

```
salt '*' keystone.service_get c965f79c4f864eaaa9c3b41904e67082
salt '*' keystone.service_get service_id=c965f79c4f864eaaa9c3b41904e67082
salt '*' keystone.service_get name=nova
```

`salt.modules.keystone.`**`service_list`**(*profile=None*, ***connection_args*)

Return a list of available services (keystone services-list)

CLI Example:

```
salt '*' keystone.service_list
```

`salt.modules.keystone.`**`tenant_create`**(*name*, *description=None*, *enabled=True*, *profile=None*, ***connection_args*)

Create a keystone tenant

CLI Examples:

```
salt '*' keystone.tenant_create nova description='nova tenant'
salt '*' keystone.tenant_create test enabled=False
```

`salt.modules.keystone.`**`tenant_delete`**(*tenant_id=None*, *name=None*, *profile=None*, ***connection_args*)

Delete a tenant (keystone tenant-delete)

CLI Examples:

```
salt '*' keystone.tenant_delete c965f79c4f864eaaa9c3b41904e67082
salt '*' keystone.tenant_delete tenant_id=c965f79c4f864eaaa9c3b41904e67082
salt '*' keystone.tenant_delete name=demo
```

`salt.modules.keystone.`**`tenant_get`**(*tenant_id=None*, *name=None*, *profile=None*, ***connection_args*)

Return a specific tenants (keystone tenant-get)

CLI Examples:

```
salt '*' keystone.tenant_get c965f79c4f864eaaa9c3b41904e67082
salt '*' keystone.tenant_get tenant_id=c965f79c4f864eaaa9c3b41904e67082
salt '*' keystone.tenant_get name=nova
```

`salt.modules.keystone.`**`tenant_list`**(profile=None, **connection_args)

Return a list of available tenants (keystone tenants-list)

CLI Example:

```
salt '*' keystone.tenant_list
```

`salt.modules.keystone.`**`tenant_update`**(tenant_id=None, name=None, email=None, enabled=None, profile=None, **connection_args)

Update a tenant's information (keystone tenant-update) The following fields may be updated: name, email, enabled. Can only update name if targeting by ID

CLI Examples:

```
salt '*' keystone.tenant_update name=admin enabled=True
salt '*' keystone.tenant_update c965f79c4f864eaaa9c3b41904e67082 name=admin email=admin@domain.c
```

`salt.modules.keystone.`**`token_get`**(profile=None, **connection_args)

Return the configured tokens (keystone token-get)

CLI Example:

```
salt '*' keystone.token_get c965f79c4f864eaaa9c3b41904e67082
```

`salt.modules.keystone.`**`user_create`**(name, password, email, tenant_id=None, enabled=True, profile=None, **connection_args)

Create a user (keystone user-create)

CLI Examples:

```
salt '*' keystone.user_create name=jack password=zero email=jack@halloweentown.org tenant_id=a28
```

`salt.modules.keystone.`**`user_delete`**(user_id=None, name=None, profile=None, **connection_args)

Delete a user (keystone user-delete)

CLI Examples:

```
salt '*' keystone.user_delete c965f79c4f864eaaa9c3b41904e67082
salt '*' keystone.user_delete user_id=c965f79c4f864eaaa9c3b41904e67082
salt '*' keystone.user_delete name=nova
```

`salt.modules.keystone.`**`user_get`**(user_id=None, name=None, profile=None, **connection_args)

Return a specific users (keystone user-get)

CLI Examples:

```
salt '*' keystone.user_get c965f79c4f864eaaa9c3b41904e67082
salt '*' keystone.user_get user_id=c965f79c4f864eaaa9c3b41904e67082
salt '*' keystone.user_get name=nova
```

`salt.modules.keystone.`**`user_list`**(profile=None, **connection_args)

Return a list of available users (keystone user-list)

CLI Example:

```
salt '*' keystone.user_list
```

salt.modules.keystone.**user_password_update**(*user_id=None*, *name=None*, *password=None*, *profile=None*, ***connection_args*)

Update a user's password (keystone user-password-update)

CLI Examples:

```
salt '*' keystone.user_delete c965f79c4f864eaaa9c3b41904e67082 password=12345
salt '*' keystone.user_delete user_id=c965f79c4f864eaaa9c3b41904e67082 password=12345
salt '*' keystone.user_delete name=nova password=12345
```

salt.modules.keystone.**user_role_add**(*user_id=None*, *user=None*, *tenant_id=None*, *tenant=None*, *role_id=None*, *role=None*, *profile=None*, ***connection_args*)

Add role for user in tenant (keystone user-role-add)

CLI Examples:

```
salt '*' keystone.user_role_add             user_id=298ce377245c4ec9b70e1c639c89e654
salt '*' keystone.user_role_add user=admin tenant=admin role=admin
```

salt.modules.keystone.**user_role_list**(*user_id=None*, *tenant_id=None*, *user_name=None*, *tenant_name=None*, *profile=None*, ***connection_args*)

Return a list of available user_roles (keystone user-roles-list)

CLI Examples:

```
salt '*' keystone.user_role_list             user_id=298ce377245c4ec9b70e1c639c89e654
salt '*' keystone.user_role_list user_name=admin tenant_name=admin
```

salt.modules.keystone.**user_role_remove**(*user_id=None*, *user=None*, *tenant_id=None*, *tenant=None*, *role_id=None*, *role=None*, *profile=None*, ***connection_args*)

Remove role for user in tenant (keystone user-role-remove)

CLI Examples:

```
salt '*' keystone.user_role_remove             user_id=298ce377245c4ec9b70e1c639c89e654
salt '*' keystone.user_role_remove user=admin tenant=admin role=admin
```

salt.modules.keystone.**user_update**(*user_id=None*, *name=None*, *email=None*, *enabled=None*, *tenant=None*, *profile=None*, ***connection_args*)

Update a user's information (keystone user-update) The following fields may be updated: name, email, enabled, tenant. Because the name is one of the fields, a valid user id is required.

CLI Examples:

```
salt '*' keystone.user_update user_id=c965f79c4f864eaaa9c3b41904e67082 name=newname
salt '*' keystone.user_update c965f79c4f864eaaa9c3b41904e67082 name=newname email=newemail@domai
```

salt.modules.keystone.**user_verify_password**(*user_id=None*, *name=None*, *password=None*, *profile=None*, ***connection_args*)

Verify a user's password

CLI Examples:

```
salt '*' keystone.user_verify_password name=test password=foobar
salt '*' keystone.user_verify_password user_id=c965f79c4f864eaaa9c3b41904e67082 password=foobar
```

61.66 salt.modules.kmod

Module to manage Linux kernel modules

`salt.modules.kmod.`**`available`**`()`

Return a list of all available kernel modules

CLI Example:

```
salt '*' kmod.available
```

`salt.modules.kmod.`**`check_available`**`(`*mod*`)`

Check to see if the specified kernel module is available

CLI Example:

```
salt '*' kmod.check_available kvm
```

`salt.modules.kmod.`**`is_loaded`**`(`*mod*`)`

Check to see if the specified kernel module is loaded

CLI Example:

```
salt '*' kmod.is_loaded kvm
```

`salt.modules.kmod.`**`load`**`(`*mod*, *persist=False*`)`

Load the specified kernel module

mod Name of module to add

persist Write module to /etc/modules to make it load on system reboot

CLI Example:

```
salt '*' kmod.load kvm
```

`salt.modules.kmod.`**`lsmod`**`()`

Return a dict containing information about currently loaded modules

CLI Example:

```
salt '*' kmod.lsmod
```

`salt.modules.kmod.`**`mod_list`**`(`*only_persist=False*`)`

Return a list of the loaded module names

CLI Example:

```
salt '*' kmod.mod_list
```

`salt.modules.kmod.`**`remove`**`(`*mod*, *persist=False*, *comment=True*`)`

Remove the specified kernel module

mod Name of module to remove

persist Also remove module from /etc/modules

comment If persist is set don’t remove line from /etc/modules but only comment it

CLI Example:

```
salt '*' kmod.remove kvm
```

61.67 salt.modules.launchctl

Module for the management of MacOS systems that use launchd/launchctl

depends

- plistlib Python module

`salt.modules.launchctl.`**`available`**`(`*job_label*`)`

Check that the given service is available.

CLI Example:

```
salt '*' service.available com.openssh.sshd
```

`salt.modules.launchctl.`**`get_all`**`()`

Return all installed services

CLI Example:

```
salt '*' service.get_all
```

`salt.modules.launchctl.`**`missing`**`(`*job_label*`)`

The inverse of service.available Check that the given service is not available.

CLI Example:

```
salt '*' service.missing com.openssh.sshd
```

`salt.modules.launchctl.`**`restart`**`(`*job_label*, *runas=None*`)`

Restart the named service

CLI Example:

```
salt '*' service.restart <service label>
```

`salt.modules.launchctl.`**`start`**`(`*job_label*, *runas=None*`)`

Start the specified service

CLI Example:

```
salt '*' service.start <service label>
salt '*' service.start org.ntp.ntpd
salt '*' service.start /System/Library/LaunchDaemons/org.ntp.ntpd.plist
```

`salt.modules.launchctl.`**`status`**`(`*job_label*, *runas=None*`)`

Return the status for a service, returns a bool whether the service is running.

CLI Example:

```
salt '*' service.status <service label>
```

`salt.modules.launchctl.`**`stop`**`(`*job_label*, *runas=None*`)`

Stop the specified service

CLI Example:

```
salt '*' service.stop <service label>
salt '*' service.stop org.ntp.ntpd
salt '*' service.stop /System/Library/LaunchDaemons/org.ntp.ntpd.plist
```

61.68 salt.modules.layman

Support for Layman

`salt.modules.layman.`**`add`**`(`*overlay*`)`

Add the given overlay from the caced remote list to your locally installed overlays. Specify 'ALL' to add all overlays from the remote list.

Return a list of the new overlay(s) added:

CLI Example:

```
salt '*' layman.add <overlay name>
```

`salt.modules.layman.`**`delete`**`(`*overlay*`)`

Remove the given overlay from the your locally installed overlays. Specify 'ALL' to remove all overlays.

Return a list of the overlays(s) that were removed:

CLI Example:

```
salt '*' layman.delete <overlay name>
```

`salt.modules.layman.`**`list_local`**`()`

List the locally installed overlays.

Return a list of installed overlays:

CLI Example:

```
salt '*' layman.list_local
```

`salt.modules.layman.`**`sync`**`(`*overlay='ALL'*`)`

Update the specified overlay. Use 'ALL' to synchronize all overlays. This is the default if no overlay is specified.

overlay Name of the overlay to sync. (Defaults to 'ALL')

CLI Example:

```
salt '*' layman.sync
```

61.69 salt.modules.ldapmod

Salt interface to LDAP commands

depends

- ldap Python module

configuration In order to connect to LDAP, certain configuration is required in the minion config on the LDAP server. The minimum configuration items that must be set are:

```
ldap.basedn: dc=acme,dc=com (example values, adjust to suit)
```

If your LDAP server requires authentication then you must also set:

```
ldap.binddn: admin
ldap.bindpw: password
```

In addition, the following optional values may be set:

```
ldap.server: localhost (default=localhost, see warning below)
ldap.port: 389 (default=389, standard port)
ldap.tls: False (default=False, no TLS)
ldap.scope: 2 (default=2, ldap.SCOPE_SUBTREE)
ldap.attrs: [saltAttr] (default=None, return all attributes)
```

Warning: At the moment this module only recommends connection to LDAP services listening on `localhost`. This is deliberate to avoid the potentially dangerous situation of multiple minions sending identical update commands to the same LDAP server. It's easy enough to override this behaviour, but badness may ensue - you have been warned.

`salt.modules.ldapmod.`**`search`**(*filter*, *dn=None*, *scope=None*, *attrs=None*, ***kwargs*)

Run an arbitrary LDAP query and return the results.

CLI Example:

```
salt 'ldaphost' ldap.search "filter=cn=myhost"
```

Return data:

```
{'myhost': {'count': 1,
            'results': [['cn=myhost,ou=hosts,o=acme,c=gb',
                         {'saltKeyValue': ['ntpserver=ntp.acme.local',
                                           'foo=myfoo'],
                          'saltState': ['foo', 'bar']}]],
            'time': {'human': '1.2ms', 'raw': '0.00123'}}}
```

Search and connection options can be overridden by specifying the relevant option as key=value pairs, for example:

```
salt 'ldaphost' ldap.search filter=cn=myhost dn=ou=hosts,o=acme,c=gb
scope=1 attrs='' server='localhost' port='7393' tls=True bindpw='ssh'
```

61.70 salt.modules.linux_acl

Support for Linux File Access Control Lists

`salt.modules.linux_acl.`**`delfacl`**(*acl_type*, *acl_name*, **args*)

Remove specific FACL from the specified file(s)

CLI Examples:

```
salt '*' acl.delfacl user myuser /tmp/house/kitchen
salt '*' acl.delfacl default:group mygroup /tmp/house/kitchen
salt '*' acl.delfacl d:u myuser /tmp/house/kitchen
salt '*' acl.delfacl g myuser /tmp/house/kitchen /tmp/house/livingroom
```

`salt.modules.linux_acl.`**`getfacl`**(**args*)

Return (extremely verbose) map of FACLs on specified file(s)

CLI Examples:

```
salt '*' acl.getfacl /tmp/house/kitchen
salt '*' acl.getfacl /tmp/house/kitchen /tmp/house/livingroom
```

`salt.modules.linux_acl.`**`modfacl`**(*acl_type*, *acl_name*, *perms*, **args*)

Add or modify a FACL for the specified file(s)

CLI Examples:

```
salt '*' acl.addfacl user myuser rwx /tmp/house/kitchen
salt '*' acl.addfacl default:group mygroup rx /tmp/house/kitchen
salt '*' acl.addfacl d:u myuser 7 /tmp/house/kitchen
salt '*' acl.addfacl g mygroup 0 /tmp/house/kitchen /tmp/house/livingroom
```

`salt.modules.linux_acl.`**`version`**`()`

Return facl version from getfacl –version

CLI Example:

```
salt '*' acl.version
```

`salt.modules.linux_acl.`**`wipefacls`**`(`**args*`)`

Remove all FACLs from the specified file(s)

CLI Examples:

```
salt '*' acl.wipefacls /tmp/house/kitchen
salt '*' acl.wipefacls /tmp/house/kitchen /tmp/house/livingroom
```

61.71 salt.modules.linux_lvm

Support for Linux LVM2

`salt.modules.linux_lvm.`**`fullversion`**`()`

Return all version info from lvm version

CLI Example:

```
salt '*' lvm.fullversion
```

`salt.modules.linux_lvm.`**`lvcreate`**`(`*lvname*, *vgname*, *size=None*, *extents=None*, *snapshot=None*, *pv=''*`)`

Create a new logical volume, with option for which physical volume to be used

CLI Examples:

```
salt '*' lvm.lvcreate new_volume_name vg_name size=10G
salt '*' lvm.lvcreate new_volume_name vg_name extents=100 /dev/sdb
salt '*' lvm.lvcreate new_snapshot    vg_name snapshot=volume_name size=3G
```

`salt.modules.linux_lvm.`**`lvdisplay`**`(`*lvname=''*`)`

Return information about the logical volume(s)

CLI Examples:

```
salt '*' lvm.lvdisplay
salt '*' lvm.lvdisplay /dev/vg_myserver/root
```

`salt.modules.linux_lvm.`**`lvremove`**`(`*lvname*, *vgname*`)`

Remove a given existing logical volume from a named existing volume group

CLI Example:

```
salt '*' lvm.lvremove lvname vgname force=True
```

`salt.modules.linux_lvm.`**`pvcreate`**`(`*devices*, ***kwargs*`)`

Set a physical device to be used as an LVM physical volume

CLI Examples:

```
salt mymachine lvm.pvcreate /dev/sdb1,/dev/sdb2
salt mymachine lvm.pvcreate /dev/sdb1 dataalignmentoffset=7s
```

`salt.modules.linux_lvm.`**`pvdisplay`**(*pvname=''*)

Return information about the physical volume(s)

CLI Examples:

```
salt '*' lvm.pvdisplay
salt '*' lvm.pvdisplay /dev/md0
```

`salt.modules.linux_lvm.`**`version`**()

Return LVM version from lvm version

CLI Example:

```
salt '*' lvm.version
```

`salt.modules.linux_lvm.`**`vgcreate`**(*vgname*, *devices*, ***kwargs*)

Create an LVM volume group

CLI Examples:

```
salt mymachine lvm.vgcreate my_vg /dev/sdb1,/dev/sdb2
salt mymachine lvm.vgcreate my_vg /dev/sdb1 clustered=y
```

`salt.modules.linux_lvm.`**`vgdisplay`**(*vgname=''*)

Return information about the volume group(s)

CLI Examples:

```
salt '*' lvm.vgdisplay
salt '*' lvm.vgdisplay nova-volumes
```

`salt.modules.linux_lvm.`**`vgremove`**(*vgname*)

Remove an LVM volume group

CLI Examples:

```
salt mymachine lvm.vgremove vgname
salt mymachine lvm.vgremove vgname force=True
```

61.72 salt.modules.linux_sysctl

Module for viewing and modifying sysctl parameters

`salt.modules.linux_sysctl.`**`assign`**(*name*, *value*)

Assign a single sysctl parameter for this minion

CLI Example:

```
salt '*' sysctl.assign net.ipv4.ip_forward 1
```

`salt.modules.linux_sysctl.`**`default_config`**()

Linux hosts using systemd 207 or later ignore `/etc/sysctl.conf` and only load from `/etc/sysctl.d/*.conf`. This function will do the proper checks and return a default config file which will be valid for the Minion. Hosts running systemd >= 207 will use `/etc/sysctl.d/99-salt.conf`.

CLI Example:

```
salt -G 'kernel:Linux' sysctl.default_config
```

`salt.modules.linux_sysctl.`**`get`**`(`*name*`)`
Return a single sysctl parameter for this minion

CLI Example:

```
salt '*' sysctl.get net.ipv4.ip_forward
```

`salt.modules.linux_sysctl.`**`persist`**`(`*name*, *value*, *config=None*`)`
Assign and persist a simple sysctl parameter for this minion. If `config` is not specified, a sensible default will be chosen using `sysctl.default_config`.

CLI Example:

```
salt '*' sysctl.persist net.ipv4.ip_forward 1
```

`salt.modules.linux_sysctl.`**`show`**`()`
Return a list of sysctl parameters for this minion

CLI Example:

```
salt '*' sysctl.show
```

61.73 salt.modules.localemod

Module for managing locales on POSIX-like systems.

`salt.modules.localemod.`**`get_locale`**`()`
Get the current system locale

CLI Example:

```
salt '*' locale.get_locale
```

`salt.modules.localemod.`**`list_avail`**`()`
Lists available (compiled) locales

CLI Example:

```
salt '*' locale.list_avail
```

`salt.modules.localemod.`**`set_locale`**`(`*locale*`)`
Sets the current system locale

CLI Example:

```
salt '*' locale.set_locale 'en_US.UTF-8'
```

61.74 salt.modules.locate

Module for using the locate utilities

`salt.modules.locate.`**`locate`**`(`*pattern*, *database=''*, *limit=0*, ***kwargs*`)`
Performs a file lookup. Valid options (and their defaults) are:

```
basename=False
count=False
existing=False
follow=True
ignore=False
nofollow=False
wholename=True
regex=False
database=<locate's default database>
limit=<integer, not set by default>
```

See the manpage for `locate(1)` for further explanation of these options.

CLI Example:

```
salt '*' locate.locate
```

`salt.modules.locate.`**`stats`**`()`

Returns statistics about the locate database

CLI Example:

```
salt '*' locate.stats
```

`salt.modules.locate.`**`updatedb`**`()`

Updates the locate database

CLI Example:

```
salt '*' locate.updatedb
```

`salt.modules.locate.`**`version`**`()`

Returns the version of locate

CLI Example:

```
salt '*' locate.version
```

61.75 salt.modules.logrotate

Module for managing logrotate.

`salt.modules.logrotate.`**`set_`**`(`*key*, *value*, *setting=None*, *conf_file='/etc/logrotate.conf'*`)`

Set a new value for a specific configuration line

CLI Example:

```
salt '*' logrotate.set rotate 2
```

Can also be used to set a single value inside a multiline configuration block. For instance, to change rotate in the following block:

```
/var/log/wtmp {
    monthly
    create 0664 root root
    rotate 1
}
```

Use the following command:

```
salt '*' logrotate.set /var/log/wtmp rotate 2
```

This module also has the ability to scan files inside an include directory, and make changes in the appropriate file.

`salt.modules.logrotate.`**`show_conf`**(*conf_file='/etc/logrotate.conf'*)
Show parsed configuration

CLI Example:

```
salt '*' logrotate.show_conf
```

61.76 salt.modules.lvs

Support for LVS (Linux Virtual Server)

`salt.modules.lvs.`**`add_server`**(*protocol=None*, *service_address=None*, *server_address=None*, *packet_forward_method='dr'*, *weight=1*, ***kwargs*)
Add a real server to a virtual service.

protocol The service protocol(only support `tcp`, `udp` and `fwmark` service).

service_address The LVS service address.

server_address The real server address.

packet_forward_method The LVS packet forwarding method(`dr` for direct routing, `tunnel` for tunneling, `nat` for network access translation).

weight The capacity of a server relative to the others in the pool.

CLI Example:

```
salt '*' lvs.add_server tcp 1.1.1.1:80 192.168.0.11:8080 nat 1
```

`salt.modules.lvs.`**`add_service`**(*protocol=None*, *service_address=None*, *scheduler='wlc'*)
Add a virtual service.

protocol The service protocol(only support tcp, udp and fwmark service).

service_address The LVS service address.

scheduler Algorithm for allocating TCP connections and UDP datagrams to real servers.

CLI Example:

```
salt '*' lvs.add_service tcp 1.1.1.1:80 rr
```

`salt.modules.lvs.`**`check_server`**(*protocol=None*, *service_address=None*, *server_address=None*, ***kwargs*)
Check the real server exists in the specified service.

CLI Example:

```
salt '*' lvs.check_server tcp 1.1.1.1:80 192.168.0.11:8080
```

`salt.modules.lvs.`**`check_service`**(*protocol=None*, *service_address=None*, ***kwargs*)
Check the virtual service exists.

CLI Example:

```
salt '*' lvs.check_service tcp 1.1.1.1:80
```

`salt.modules.lvs.`**`clear`**`()`

Clear the virtual server table

CLI Example:

```
salt '*' lvs.clear
```

`salt.modules.lvs.`**`delete_server`**`(`*protocol=None*, *service_address=None*, *server_address=None*`)`

Delete the realserver from the virtual service.

protocol The service protocol(only support `tcp`, `udp` and `fwmark` service).

service_address The LVS service address.

server_address The real server address.

CLI Example:

```
salt '*' lvs.delete_server tcp 1.1.1.1:80 192.168.0.11:8080
```

`salt.modules.lvs.`**`delete_service`**`(`*protocol=None*, *service_address=None*`)`

Delete the virtual service.

protocol The service protocol(only support tcp, udp and fwmark service).

service_address The LVS service address.

CLI Example:

```
salt '*' lvs.delete_service tcp 1.1.1.1:80
```

`salt.modules.lvs.`**`edit_server`**`(`*protocol=None*, *service_address=None*, *server_address=None*, *packet_forward_method=None*, *weight=None*, ***kwargs*`)`

Edit a real server to a virtual service.

protocol The service protocol(only support `tcp`, `udp` and `fwmark` service).

service_address The LVS service address.

server_address The real server address.

packet_forward_method The LVS packet forwarding method(`dr` for direct routing, `tunnel` for tunneling, `nat` for network access translation).

weight The capacity of a server relative to the others in the pool.

CLI Example:

```
salt '*' lvs.edit_server tcp 1.1.1.1:80 192.168.0.11:8080 nat 1
```

`salt.modules.lvs.`**`edit_service`**`(`*protocol=None*, *service_address=None*, *scheduler=None*`)`

Edit the virtual service.

protocol The service protocol(only support tcp, udp and fwmark service).

service_address The LVS service address.

scheduler Algorithm for allocating TCP connections and UDP datagrams to real servers.

CLI Example:

```
salt '*' lvs.edit_service tcp 1.1.1.1:80 rr
```

`salt.modules.lvs.`**`get_rules`**`()`

Get the virtual server rules

CLI Example:

```
salt '*' lvs.get_rules
```

`salt.modules.lvs.`**`list_`**`(`*protocol=None*, *service_address=None*`)`

List the virtual server table if service_address is not specified. If a service_address is selected, list this service only.

CLI Example:

```
salt '*' lvs.list
```

`salt.modules.lvs.`**`zero`**`(`*protocol=None*, *service_address=None*`)`

Zero the packet, byte and rate counters in a service or all services.

CLI Example:

```
salt '*' lvs.zero
```

61.77 salt.modules.lxc

Control Linux Containers via Salt

depends lxc package for distribution

`salt.modules.lxc.`**`create`**`(`*name*, *config=None*, *profile=None*, *options=None*, ***kwargs*`)`

Create a new container.

```
salt 'minion' lxc.create name [config=config_file] \
        [profile=profile] [template=template_name] \
        [backing=backing_store] [ vgname=volume_group] \
        [size=filesystem_size] [options=template_options]
```

name Name of the container.

config The config file to use for the container. Defaults to system-wide config (usually in /etc/lxc/lxc.conf).

profile A LXC profile (defined in config or pillar).

template The template to use. E.g., 'ubuntu' or 'fedora'.

backing The type of storage to use. Set to 'lvm' to use an LVM group. Defaults to filesystem within /var/lib/lxc/.

vgname Name of the LVM volume group in which to create the volume for this container. Only applicable if backing=lvm. Defaults to 'lxc'.

size Size of the volume to create. Only applicable if backing=lvm. Defaults to 1G.

options Template specific options to pass to the lxc-create command.

`salt.modules.lxc.`**`destroy`**`(`*name*, *stop=True*`)`

Destroy the named container. WARNING: Destroys all data associated with the container.

```
salt '*' lxc.destroy name [stop=(true|false)]
```

`salt.modules.lxc.`**`exists`**`(`*name*`)`

Returns whether the named container exists.

```
salt '*' lxc.exists name
```

`salt.modules.lxc.`**`freeze`**`(`*name*`)`
Freeze the named container.

```
salt '*' lxc.freeze name
```

`salt.modules.lxc.`**`get_parameter`**`(`*name*, *parameter*`)`
Returns the value of a cgroup parameter for a container.

```
salt '*' lxc.get_parameter name parameter
```

`salt.modules.lxc.`**`info`**`(`*name*`)`
Returns information about a container.

```
salt '*' lxc.info name
```

`salt.modules.lxc.`**`init`**`(`*name*, *cpuset=None*, *cpushare=None*, *memory=None*, *nic='default'*, *profile=None*, *nic_opts=None*, ***kwargs*`)`
Initialize a new container.

```
salt 'minion' lxc.init name [cpuset=cgroups_cpuset] \
        [cpushare=cgroups_cpushare] [memory=cgroups_memory] \
        [nic=nic_profile] [profile=lxc_profile] \
        [nic_opts=nic_opts] [start=(true|false)] \
        [seed=(true|false)] [install=(true|false)] \
        [config=minion_config]
```

name Name of the container.

cpuset cgroups cpuset.

cpushare cgroups cpu shares.

memory cgroups memory limit, in MB.

nic Network interfaces profile (defined in config or pillar).

profile A LXC profile (defined in config or pillar).

nic_opts Extra options for network interfaces. E.g: {"eth0": {"mac": "aa:bb:cc:dd:ee:ff", "ipv4": "10.1.1.1", "ipv6": "2001:db8::ff00:42:8329"}}

start Start the newly created container.

seed Seed the container with the minion config. Default: true

install If salt-minion is not already installed, install it. Default: true

config Optional config paramers. By default, the id is set to the name of the container.

`salt.modules.lxc.`**`list_`**`()`
List defined containers (running, stopped, and frozen).

```
salt '*' lxc.list
```

`salt.modules.lxc.`**`set_parameter`**`(`*name*, *parameter*, *value*`)`
Set the value of a cgroup parameter for a container.

```
salt '*' lxc.set_parameter name parameter value
```

`salt.modules.lxc.`**`start`**`(`*name*`)`
Start the named container.

```
salt '*' lxc.start name
```

`salt.modules.lxc.`**`state`**`(`*name*`)`

Returns the state of a container.

```
salt '*' lxc.state name
```

`salt.modules.lxc.`**`stop`**`(`*name*`)`

Stop the named container.

```
salt '*' lxc.stop name
```

`salt.modules.lxc.`**`unfreeze`**`(`*name*`)`

Unfreeze the named container.

```
salt '*' lxc.unfreeze name
```

61.78 salt.modules.mac_group

Manage groups on Mac OS 10.7+

`salt.modules.mac_group.`**`add`**`(`*name*, *gid=None*, ***kwargs*`)`

Add the specified group

CLI Example:

```
salt '*' group.add foo 3456
```

`salt.modules.mac_group.`**`chgid`**`(`*name*, *gid*`)`

Change the gid for a named group

CLI Example:

```
salt '*' group.chgid foo 4376
```

`salt.modules.mac_group.`**`delete`**`(`*name*`)`

Remove the named group

CLI Example:

```
salt '*' group.delete foo
```

`salt.modules.mac_group.`**`getent`**`(`*refresh=False*`)`

Return info on all groups

CLI Example:

```
salt '*' group.getent
```

`salt.modules.mac_group.`**`info`**`(`*name*`)`

Return information about a group

CLI Example:

```
salt '*' group.info foo
```

61.79 salt.modules.mac_user

Manage users on Mac OS 10.7+

`salt.modules.mac_user.`**`add`**(*name*, *uid=None*, *gid=None*, *groups=None*, *home=None*, *shell=None*, *fullname=None*, *createhome=True*, ***kwargs*)

Add a user to the minion

CLI Example:

```
salt '*' user.add name <uid> <gid> <groups> <home> <shell>
```

`salt.modules.mac_user.`**`chfullname`**(*name*, *fullname*)

Change the user's Full Name

CLI Example:

```
salt '*' user.chfullname foo 'Foo Bar'
```

`salt.modules.mac_user.`**`chgid`**(*name*, *gid*)

Change the default group of the user

CLI Example:

```
salt '*' user.chgid foo 4376
```

`salt.modules.mac_user.`**`chgroups`**(*name*, *groups*, *append=False*)

Change the groups to which the user belongs. Note that the user's primary group does not have to be one of the groups passed, membership in the user's primary group is automatically assumed.

groups Groups to which the user should belong, can be passed either as a python list or a comma-separated string

append Instead of removing user from groups not included in the `groups` parameter, just add user to any groups for which they are not members

CLI Example:

```
salt '*' user.chgroups foo wheel,root
```

`salt.modules.mac_user.`**`chhome`**(*name*, *home*)

Change the home directory of the user

CLI Example:

```
salt '*' user.chhome foo /Users/foo
```

`salt.modules.mac_user.`**`chshell`**(*name*, *shell*)

Change the default shell of the user

CLI Example:

```
salt '*' user.chshell foo /bin/zsh
```

`salt.modules.mac_user.`**`chuid`**(*name*, *uid*)

Change the uid for a named user

CLI Example:

```
salt '*' user.chuid foo 4376
```

`salt.modules.mac_user.`**`delete`**(*name*, **args*)

Remove a user from the minion

CLI Example:

```
salt '*' user.delete foo
```

`salt.modules.mac_user.`**`getent`**`(`*refresh=False*`)`
Return the list of all info for all users

CLI Example:

```
salt '*' user.getent
```

`salt.modules.mac_user.`**`info`**`(`*name*`)`
Return user information

CLI Example:

```
salt '*' user.info root
```

`salt.modules.mac_user.`**`list_groups`**`(`*name*`)`
Return a list of groups the named user belongs to

CLI Example:

```
salt '*' user.list_groups foo
```

`salt.modules.mac_user.`**`list_users`**`()`
Return a list of all users

CLI Example:

```
salt '*' user.list_users
```

61.80 salt.modules.makeconf

Support for modifying make.conf under Gentoo

`salt.modules.makeconf.`**`append_cflags`**`(`*value*`)`
Add to or create a new CFLAGS in the make.conf

Return a dict containing the new value for variable:

```
{'<variable>': {'old': '<old-value>',
                'new': '<new-value>'}}
```

CLI Example:

```
salt '*' makeconf.append_cflags '-pipe'
```

`salt.modules.makeconf.`**`append_cxxflags`**`(`*value*`)`
Add to or create a new CXXFLAGS in the make.conf

Return a dict containing the new value for variable:

```
{'<variable>': {'old': '<old-value>',
                'new': '<new-value>'}}
```

CLI Example:

```
salt '*' makeconf.append_cxxflags '-pipe'
```

`salt.modules.makeconf.`**`append_emerge_default_opts`**`(`*value*`)`

Add to or create a new EMERGE_DEFAULT_OPTS in the make.conf

Return a dict containing the new value for variable:

```
{'<variable>': {'old': '<old-value>',
                'new': '<new-value>'}}
```

CLI Example:

```
salt '*' makeconf.append_emerge_default_opts '--jobs'
```

`salt.modules.makeconf.`**`append_features`**`(`*value*`)`

Add to or create a new FEATURES in the make.conf

Return a dict containing the new value for variable:

```
{'<variable>': {'old': '<old-value>',
                'new': '<new-value>'}}
```

CLI Example:

```
salt '*' makeconf.append_features 'webrsync-gpg'
```

`salt.modules.makeconf.`**`append_gentoo_mirrors`**`(`*value*`)`

Add to or create a new GENTOO_MIRRORS in the make.conf

Return a dict containing the new value for variable:

```
{'<variable>': {'old': '<old-value>',
                'new': '<new-value>'}}
```

CLI Example:

```
salt '*' makeconf.append_gentoo_mirrors 'http://distfiles.gentoo.org'
```

`salt.modules.makeconf.`**`append_makeopts`**`(`*value*`)`

Add to or create a new MAKEOPTS in the make.conf

Return a dict containing the new value for variable:

```
{'<variable>': {'old': '<old-value>',
                'new': '<new-value>'}}
```

CLI Example:

```
salt '*' makeconf.append_makeopts '-j3'
```

`salt.modules.makeconf.`**`append_var`**`(`*var*, *value*`)`

Add to or create a new variable in the make.conf

Return a dict containing the new value for variable:

```
{'<variable>': {'old': '<old-value>',
                'new': '<new-value>'}}
```

CLI Example:

```
salt '*' makeconf.append_var 'LINGUAS' 'en'
```

`salt.modules.makeconf.`**`cflags_contains`**`(`*value*`)`

Verify if CFLAGS variable contains a value in make.conf

Return True if value is set for var

CLI Example:

```
salt '*' makeconf.cflags_contains '-pipe'
```

`salt.modules.makeconf.`**`chost_contains`**(*value*)
Verify if CHOST variable contains a value in make.conf

Return True if value is set for var

CLI Example:

```
salt '*' makeconf.chost_contains 'x86_64-pc-linux-gnu'
```

`salt.modules.makeconf.`**`cxxflags_contains`**(*value*)
Verify if CXXFLAGS variable contains a value in make.conf

Return True if value is set for var

CLI Example:

```
salt '*' makeconf.cxxflags_contains '-pipe'
```

`salt.modules.makeconf.`**`emerge_default_opts_contains`**(*value*)
Verify if EMERGE_DEFAULT_OPTS variable contains a value in make.conf

Return True if value is set for var

CLI Example:

```
salt '*' makeconf.emerge_default_opts_contains '--jobs'
```

`salt.modules.makeconf.`**`features_contains`**(*value*)
Verify if FEATURES variable contains a value in make.conf

Return True if value is set for var

CLI Example:

```
salt '*' makeconf.features_contains 'webrsync-gpg'
```

`salt.modules.makeconf.`**`gentoo_mirrors_contains`**(*value*)
Verify if GENTOO_MIRRORS variable contains a value in make.conf

Return True if value is set for var

CLI Example:

```
salt '*' makeconf.gentoo_mirrors_contains 'http://distfiles.gentoo.org'
```

`salt.modules.makeconf.`**`get_cflags`**()
Get the value of CFLAGS variable in the make.conf

Return the value of the variable or None if the variable is not in the make.conf

CLI Example:

```
salt '*' makeconf.get_cflags
```

`salt.modules.makeconf.`**`get_chost`**()
Get the value of CHOST variable in the make.conf

Return the value of the variable or None if the variable is not in the make.conf

CLI Example:

```
salt '*' makeconf.get_chost
```

salt.modules.makeconf.**get_cxxflags**()

Get the value of CXXFLAGS variable in the make.conf

Return the value of the variable or None if the variable is not in the make.conf

CLI Example:

```
salt '*' makeconf.get_cxxflags
```

salt.modules.makeconf.**get_emerge_default_opts**()

Get the value of EMERGE_DEFAULT_OPTS variable in the make.conf

Return the value of the variable or None if the variable is not in the make.conf

CLI Example:

```
salt '*' makeconf.get_emerge_default_opts
```

salt.modules.makeconf.**get_features**()

Get the value of FEATURES variable in the make.conf

Return the value of the variable or None if the variable is not in the make.conf

CLI Example:

```
salt '*' makeconf.get_features
```

salt.modules.makeconf.**get_gentoo_mirrors**()

Get the value of GENTOO_MIRRORS variable in the make.conf

Return the value of the variable or None if the variable is not in the make.conf

CLI Example:

```
salt '*' makeconf.get_gentoo_mirrors
```

salt.modules.makeconf.**get_makeopts**()

Get the value of MAKEOPTS variable in the make.conf

Return the value of the variable or None if the variable is not in the make.conf

CLI Example:

```
salt '*' makeconf.get_makeopts
```

salt.modules.makeconf.**get_sync**()

Get the value of SYNC variable in the make.conf

Return the value of the variable or None if the variable is not in the make.conf

CLI Example:

```
salt '*' makeconf.get_sync
```

salt.modules.makeconf.**get_var**(*var*)

Get the value of a variable in make.conf

Return the value of the variable or None if the variable is not in make.conf

CLI Example:

```
salt '*' makeconf.get_var 'LINGUAS'
```

salt.modules.makeconf.**makeopts_contains**(*value*)

Verify if MAKEOPTS variable contains a value in make.conf

Return True if value is set for var

CLI Example:

```
salt '*' makeconf.makeopts_contains '-j3'
```

salt.modules.makeconf.**remove_var**(*var*)

Remove a variable from the make.conf

Return a dict containing the new value for the variable:

```
{'<variable>': {'old': '<old-value>',
                'new': '<new-value>'}}
```

CLI Example:

```
salt '*' makeconf.remove_var 'LINGUAS'
```

salt.modules.makeconf.**set_cflags**(*value*)

Set the CFLAGS variable

Return a dict containing the new value for variable:

```
{'<variable>': {'old': '<old-value>',
                'new': '<new-value>'}}
```

CLI Example:

```
salt '*' makeconf.set_cflags '-march=native -O2 -pipe'
```

salt.modules.makeconf.**set_chost**(*value*)

Set the CHOST variable

Return a dict containing the new value for variable:

```
{'<variable>': {'old': '<old-value>',
                'new': '<new-value>'}}
```

CLI Example:

```
salt '*' makeconf.set_chost 'x86_64-pc-linux-gnu'
```

salt.modules.makeconf.**set_cxxflags**(*value*)

Set the CXXFLAGS variable

Return a dict containing the new value for variable:

```
{'<variable>': {'old': '<old-value>',
                'new': '<new-value>'}}
```

CLI Example:

```
salt '*' makeconf.set_cxxflags '-march=native -O2 -pipe'
```

salt.modules.makeconf.**set_emerge_default_opts**(*value*)

Set the EMERGE_DEFAULT_OPTS variable

Return a dict containing the new value for variable:

```
{'<variable>': {'old': '<old-value>',
                'new': '<new-value>'}}
```

CLI Example:

```
salt '*' makeconf.set_emerge_default_opts '--jobs'
```

`salt.modules.makeconf.`**`set_gentoo_mirrors`**`(`*value*`)`

Set the GENTOO_MIRRORS variable

Return a dict containing the new value for variable:

```
{'<variable>': {'old': '<old-value>',
                'new': '<new-value>'}}
```

CLI Example:

```
salt '*' makeconf.set_gentoo_mirrors 'http://distfiles.gentoo.org'
```

`salt.modules.makeconf.`**`set_makeopts`**`(`*value*`)`

Set the MAKEOPTS variable

Return a dict containing the new value for variable:

```
{'<variable>': {'old': '<old-value>',
                'new': '<new-value>'}}
```

CLI Example:

```
salt '*' makeconf.set_makeopts '-j3'
```

`salt.modules.makeconf.`**`set_sync`**`(`*value*`)`

Set the SYNC variable

Return a dict containing the new value for variable:

```
{'<variable>': {'old': '<old-value>',
                'new': '<new-value>'}}
```

CLI Example:

```
salt '*' makeconf.set_sync 'rsync://rsync.namerica.gentoo.org/gentoo-portage'
```

`salt.modules.makeconf.`**`set_var`**`(`*var*, *value*`)`

Set a variable in the make.conf

Return a dict containing the new value for variable:

```
{'<variable>': {'old': '<old-value>',
                'new': '<new-value>'}}
```

CLI Example:

```
salt '*' makeconf.set_var 'LINGUAS' 'en'
```

`salt.modules.makeconf.`**`sync_contains`**`(`*value*`)`

Verify if SYNC variable contains a value in make.conf

Return True if value is set for var

CLI Example:

```
salt '*' makeconf.sync_contains 'rsync://rsync.namerica.gentoo.org/gentoo-portage'
```

salt.modules.makeconf.**trim_cflags**(*value*)

Remove a value from CFLAGS variable in the make.conf

Return a dict containing the new value for variable:

```
{'<variable>': {'old': '<old-value>',
                'new': '<new-value>'}}
```

CLI Example:

```
salt '*' makeconf.trim_cflags '-pipe'
```

salt.modules.makeconf.**trim_cxxflags**(*value*)

Remove a value from CXXFLAGS variable in the make.conf

Return a dict containing the new value for variable:

```
{'<variable>': {'old': '<old-value>',
                'new': '<new-value>'}}
```

CLI Example:

```
salt '*' makeconf.trim_cxxflags '-pipe'
```

salt.modules.makeconf.**trim_emerge_default_opts**(*value*)

Remove a value from EMERGE_DEFAULT_OPTS variable in the make.conf

Return a dict containing the new value for variable:

```
{'<variable>': {'old': '<old-value>',
                'new': '<new-value>'}}
```

CLI Example:

```
salt '*' makeconf.trim_emerge_default_opts '--jobs'
```

salt.modules.makeconf.**trim_features**(*value*)

Remove a value from FEATURES variable in the make.conf

Return a dict containing the new value for variable:

```
{'<variable>': {'old': '<old-value>',
                'new': '<new-value>'}}
```

CLI Example:

```
salt '*' makeconf.trim_features 'webrsync-gpg'
```

salt.modules.makeconf.**trim_gentoo_mirrors**(*value*)

Remove a value from GENTOO_MIRRORS variable in the make.conf

Return a dict containing the new value for variable:

```
{'<variable>': {'old': '<old-value>',
                'new': '<new-value>'}}
```

CLI Example:

```
salt '*' makeconf.trim_gentoo_mirrors 'http://distfiles.gentoo.org'
```

`salt.modules.makeconf.`**`trim_makeopts`**`(`*value*`)`

Remove a value from MAKEOPTS variable in the make.conf

Return a dict containing the new value for variable:

```
{'<variable>': {'old': '<old-value>',
                'new': '<new-value>'}}
```

CLI Example:

```
salt '*' makeconf.trim_makeopts '-j3'
```

`salt.modules.makeconf.`**`trim_var`**`(`*var*, *value*`)`

Remove a value from a variable in the make.conf

Return a dict containing the new value for variable:

```
{'<variable>': {'old': '<old-value>',
                'new': '<new-value>'}}
```

CLI Example:

```
salt '*' makeconf.trim_var 'LINGUAS' 'en'
```

`salt.modules.makeconf.`**`var_contains`**`(`*var*, *value*`)`

Verify if variable contains a value in make.conf

Return True if value is set for var

CLI Example:

```
salt '*' makeconf.var_contains 'LINGUAS' 'en'
```

61.81 salt.modules.match

The match module allows for match routines to be run and determine target specs

`salt.modules.match.`**`compound`**`(`*tgt*`)`

Return True if the minion matches the given compound target

CLI Example:

```
salt '*' match.compound 'L@cheese,foo and *'
```

`salt.modules.match.`**`data`**`(`*tgt*`)`

Return True if the minion matches the given data target

CLI Example:

```
salt '*' match.data 'spam:eggs'
```

`salt.modules.match.`**`glob`**`(`*tgt*`)`

Return True if the minion matches the given glob target

CLI Example:

```
salt '*' match.glob '*'
```

`salt.modules.match.`**`grain`**`(`*tgt*, *delim=':'*`)`

Return True if the minion matches the given grain target. The `delim` argument can be used to specify a different delimiter.

CLI Example:

```
salt '*' match.grain 'os:Ubuntu'
salt '*' match.grain_pcre 'ipv6|2001:db8::ff00:42:8329' delim='|'
```

Changed in version 0.16.4: `delim` argument added

`salt.modules.match.`**`grain_pcre`**(*tgt*, *delim=':'*)

Return True if the minion matches the given grain_pcre target. The `delim` argument can be used to specify a different delimiter.

CLI Example:

```
salt '*' match.grain_pcre 'os:Fedo.*'
salt '*' match.grain_pcre 'ipv6|2001:.*' delim='|'
```

Changed in version 0.16.4: `delim` argument added

`salt.modules.match.`**`ipcidr`**(*tgt*)

Return True if the minion matches the given ipcidr target

CLI Example:

```
salt '*' match.ipcidr '192.168.44.0/24'
```

`salt.modules.match.`**`list_`**(*tgt*)

Return True if the minion matches the given list target

CLI Example:

```
salt '*' match.list 'server1,server2'
```

`salt.modules.match.`**`pcre`**(*tgt*)

Return True if the minion matches the given pcre target

CLI Example:

```
salt '*' match.pcre '.*'
```

`salt.modules.match.`**`pillar`**(*tgt*, *delim=':'*)

Return True if the minion matches the given pillar target. The `delim` argument can be used to specify a different delimiter.

CLI Example:

```
salt '*' match.pillar 'cheese:foo'
salt '*' match.pillar 'clone_url|https://github.com/saltstack/salt.git' delim='|'
```

Changed in version 0.16.4: `delim` argument added

61.82 salt.modules.mdadm

Salt module to manage RAID arrays with mdadm

`salt.modules.mdadm.`**`create`**(**args*)

Create a RAID device.

Warning: Use with CAUTION, as this function can be very destructive if not used properly!

Use this module just as a regular mdadm command.

For more info, read the `mdadm(8)` manpage

NOTE: It takes time to create a RAID array. You can check the progress in "resync_status:" field of the results from the following command:

```
salt '*' raid.detail /dev/md0
```

CLI Examples:

```
salt '*' raid.create /dev/md0 level=1 chunk=256 raid-devices=2 /dev/xvdd /dev/xvde test_mode=Tru
```

Note: Test mode

Adding `test_mode=True` as an argument will print out the mdadm command that would have been run.

Parameters

- **args** – The arguments u pass to this function.
- **arguments** – arguments['new_array']: The name of the new RAID array that will be created. arguments['opt_val']: Option with Value. Example: raid-devices=2 arguments['opt_raw']: Option without Value. Example: force arguments['disks_to_array']: The disks that will be added to the new raid.

Returns

test_mode=True: Prints out the full command.

test_mode=False (Default): Executes command on remote the host(s) and Prints out the mdadm output.

`salt.modules.mdadm.`**`destroy`**`(`*device*`)`

Destroy a RAID device.

WARNING This will zero the superblock of all members of the RAID array..

CLI Example:

```
salt '*' raid.destroy /dev/md0
```

`salt.modules.mdadm.`**`detail`**`(`*device='/dev/md0'*`)`

Show detail for a specified RAID device

CLI Example:

```
salt '*' raid.detail '/dev/md0'
```

`salt.modules.mdadm.`**`list_`**`()`

List the RAID devices.

CLI Example:

```
salt '*' raid.list
```

61.83 salt.modules.memcached

61.83.1 Module for Management of Memcached Keys

New in version 2014.1.0: (Hydrogen)

`salt.modules.memcached.`**`add`**(*key*, *value*, *host='127.0.0.1'*, *port=11211*, *time=0*, *min_compress_len=0*)

Add a key to the memcached server, but only if it does not exist. Returns False if the key already exists.

CLI Example:

```
salt '*' memcached.add <key> <value>
```

`salt.modules.memcached.`**`decrement`**(*key*, *delta=1*, *host='127.0.0.1'*, *port=11211*)

Decrement the value of a key

CLI Example:

```
salt '*' memcached.decrement <key>
salt '*' memcached.decrement <key> 2
```

`salt.modules.memcached.`**`delete`**(*key*, *host='127.0.0.1'*, *port=11211*, *time=0*)

Delete a key from memcache server

CLI Example:

```
salt '*' memcached.delete <key>
```

`salt.modules.memcached.`**`get`**(*key*, *host='127.0.0.1'*, *port=11211*)

Retrieve value for a key

CLI Example:

```
salt '*' memcached.get <key>
```

`salt.modules.memcached.`**`increment`**(*key*, *delta=1*, *host='127.0.0.1'*, *port=11211*)

Increment the value of a key

CLI Example:

```
salt '*' memcached.increment <key>
salt '*' memcached.increment <key> 2
```

`salt.modules.memcached.`**`replace`**(*key*, *value*, *host='127.0.0.1'*, *port=11211*, *time=0*, *min_compress_len=0*)

Replace a key on the memcached server. This only succeeds if the key already exists. This is the opposite of `memcached.add`

CLI Example:

```
salt '*' memcached.replace <key> <value>
```

`salt.modules.memcached.`**`set_`**(*key*, *value*, *host='127.0.0.1'*, *port=11211*, *time=0*, *min_compress_len=0*)

Set a key on the memcached server, overwriting the value if it exists.

CLI Example:

```
salt '*' memcached.set <key> <value>
```

`salt.modules.memcached.`**`status`**(*host='127.0.0.1'*, *port=11211*)

Get memcached status

CLI Example:

```
salt '*' memcached.status
```

61.84 salt.modules.mine

The function cache system allows for data to be stored on the master so it can be easily read by other minions

`salt.modules.mine.`**`delete`**`(`*fun*`)`

Remove specific function contents of minion. Returns True on success.

CLI Example:

```
salt '*' mine.delete 'network.interfaces'
```

`salt.modules.mine.`**`flush`**`()`

Remove all mine contents of minion. Returns True on success.

CLI Example:

```
salt '*' mine.flush
```

`salt.modules.mine.`**`get`**`(`*tgt*, *fun*, *expr_form='glob'*`)`

Get data from the mine based on the target, function and expr_form

Targets can be matched based on any standard matching system that can be matched on the master via these keywords:

```
glob
pcre
grain
grain_pcre
```

CLI Example:

```
salt '*' mine.get '*' network.interfaces
salt '*' mine.get 'os:Fedora' network.interfaces grain
```

`salt.modules.mine.`**`send`**`(`*func*, **args*, ***kwargs*`)`

Send a specific function to the mine.

CLI Example:

```
salt '*' mine.send network.interfaces eth0
```

`salt.modules.mine.`**`update`**`(`*clear=False*`)`

Execute the configured functions and send the data back up to the master The functions to be executed are merged from the master config, pillar and minion config under the option “function_cache”:

```
mine_functions:
  network.ip_addrs:
    - eth0
  disk.usage: []
```

The function cache will be populated with information from executing these functions

CLI Example:

```
salt '*' mine.update
```

61.85 salt.modules.modjk

Control Modjk via the Apache Tomcat “Status” worker (http://tomcat.apache.org/connectors-doc/reference/status.html)

Below is an example of the configuration needed for this module. This configuration data can be placed either in *grains* or *pillar*.

If using grains, this can be accomplished *statically* or via a *grain module*.

If using pillar, the yaml configuration can be placed directly into a pillar SLS file, making this both the easier and more dynamic method of configuring this module.

```
modjk:
  default:
    url: http://localhost/jkstatus
    user: modjk
    pass: secret
    realm: authentication realm for digest passwords
    timeout: 5
  otherVhost:
    url: http://otherVhost/jkstatus
    user: modjk
    pass: secret2
    realm: authentication realm2 for digest passwords
    timeout: 600
```

`salt.modules.modjk.`**`bulk_activate`**(*workers*, *lbn*, *profile='default'*)

Activate all the given workers in the specific load balancer

CLI Examples:

```
salt '*' modjk.bulk_activate node1,node2,node3 loadbalancer1
salt '*' modjk.bulk_activate node1,node2,node3 loadbalancer1 other-profile

salt '*' modjk.bulk_activate ["node1","node2","node3"] loadbalancer1
salt '*' modjk.bulk_activate ["node1","node2","node3"] loadbalancer1 other-profile
```

`salt.modules.modjk.`**`bulk_disable`**(*workers*, *lbn*, *profile='default'*)

Disable all the given workers in the specific load balancer

CLI Examples:

```
salt '*' modjk.bulk_disable node1,node2,node3 loadbalancer1
salt '*' modjk.bulk_disable node1,node2,node3 loadbalancer1 other-profile

salt '*' modjk.bulk_disable ["node1","node2","node3"] loadbalancer1
salt '*' modjk.bulk_disable ["node1","node2","node3"] loadbalancer1 other-profile
```

`salt.modules.modjk.`**`bulk_recover`**(*workers*, *lbn*, *profile='default'*)

Recover all the given workers in the specific load balancer

CLI Examples:

```
salt '*' modjk.bulk_recover node1,node2,node3 loadbalancer1
salt '*' modjk.bulk_recover node1,node2,node3 loadbalancer1 other-profile

salt '*' modjk.bulk_recover ["node1","node2","node3"] loadbalancer1
salt '*' modjk.bulk_recover ["node1","node2","node3"] loadbalancer1 other-profile
```

`salt.modules.modjk.`**`bulk_stop`**(*workers*, *lbn*, *profile='default'*)

Stop all the given workers in the specific load balancer

CLI Examples:

```
salt '*' modjk.bulk_stop node1,node2,node3 loadbalancer1
salt '*' modjk.bulk_stop node1,node2,node3 loadbalancer1 other-profile
salt '*' modjk.bulk_stop ["node1","node2","node3"] loadbalancer1
salt '*' modjk.bulk_stop ["node1","node2","node3"] loadbalancer1 other-profile
```

`salt.modules.modjk.`**`dump_config`**(*profile='default'*)
Dump the original configuration that was loaded from disk

CLI Examples:

```
salt '*' modjk.dump_config
salt '*' modjk.dump_config other-profile
```

`salt.modules.modjk.`**`get_running`**(*profile='default'*)
Get the current running config (not from disk)

CLI Examples:

```
salt '*' modjk.get_running
salt '*' modjk.get_running other-profile
```

`salt.modules.modjk.`**`lb_edit`**(*lbn*, *settings*, *profile='default'*)
Edit the loadbalancer settings

Note: http://tomcat.apache.org/connectors-doc/reference/status.html Data Parameters for the standard Update Action

CLI Examples:

```
salt '*' modjk.lb_edit loadbalancer1 "{'vlr': 1, 'vlt': 60}"
salt '*' modjk.lb_edit loadbalancer1 "{'vlr': 1, 'vlt': 60}" other-profile
```

`salt.modules.modjk.`**`list_configured_members`**(*lbn*, *profile='default'*)
Return a list of member workers from the configuration files

CLI Examples:

```
salt '*' modjk.list_configured_members loadbalancer1
salt '*' modjk.list_configured_members loadbalancer1 other-profile
```

`salt.modules.modjk.`**`recover_all`**(*lbn*, *profile='default'*)
Set the all the workers in lbn to recover and activate them if they are not

CLI Examples:

```
salt '*' modjk.recover_all loadbalancer1
salt '*' modjk.recover_all loadbalancer1 other-profile
```

`salt.modules.modjk.`**`reset_stats`**(*lbn*, *profile='default'*)
Reset all runtime statistics for the load balancer

CLI Examples:

```
salt '*' modjk.reset_stats loadbalancer1
salt '*' modjk.reset_stats loadbalancer1 other-profile
```

`salt.modules.modjk.`**`version`**(*profile='default'*)
Return the modjk version

CLI Examples:

```
salt '*' modjk.version
salt '*' modjk.version other-profile
```

`salt.modules.modjk.`**`worker_activate`**(*worker*, *lbn*, *profile='default'*)

Set the worker to activate state in the lbn load balancer

CLI Examples:

```
salt '*' modjk.worker_activate node1 loadbalancer1
salt '*' modjk.worker_activate node1 loadbalancer1 other-profile
```

`salt.modules.modjk.`**`worker_disable`**(*worker*, *lbn*, *profile='default'*)

Set the worker to disable state in the lbn load balancer

CLI Examples:

```
salt '*' modjk.worker_disable node1 loadbalancer1
salt '*' modjk.worker_disable node1 loadbalancer1 other-profile
```

`salt.modules.modjk.`**`worker_edit`**(*worker*, *lbn*, *settings*, *profile='default'*)

Edit the worker settings

Note: http://tomcat.apache.org/connectors-doc/reference/status.html Data Parameters for the standard Update Action

CLI Examples:

```
salt '*' modjk.worker_edit node1 loadbalancer1 "{'vwf': 500, 'vwd': 60}"
salt '*' modjk.worker_edit node1 loadbalancer1 "{'vwf': 500, 'vwd': 60}" other-profile
```

`salt.modules.modjk.`**`worker_recover`**(*worker*, *lbn*, *profile='default'*)

Set the worker to recover this module will fail if it is in OK state

CLI Examples:

```
salt '*' modjk.worker_recover node1 loadbalancer1
salt '*' modjk.worker_recover node1 loadbalancer1 other-profile
```

`salt.modules.modjk.`**`worker_status`**(*worker*, *profile='default'*)

Return the state of the worker

CLI Examples:

```
salt '*' modjk.worker_status node1
salt '*' modjk.worker_status node1 other-profile
```

`salt.modules.modjk.`**`worker_stop`**(*worker*, *lbn*, *profile='default'*)

Set the worker to stopped state in the lbn load balancer

CLI Examples:

```
salt '*' modjk.worker_activate node1 loadbalancer1
salt '*' modjk.worker_activate node1 loadbalancer1 other-profile
```

`salt.modules.modjk.`**`workers`**(*profile='default'*)

Return a list of member workers and their status

CLI Examples:

```
salt '*' modjk.workers
salt '*' modjk.workers other-profile
```

61.86 salt.modules.mongodb

Module to provide MongoDB functionality to Salt

configuration This module uses PyMongo, and accepts configuration details as parameters as well as configuration settings:

```
mongodb.host: 'localhost'
mongodb.port: 27017
mongodb.user: ''
mongodb.password: ''
```

This data can also be passed into pillar. Options passed into opts will overwrite options passed into pillar.

`salt.modules.mongodb.`**`db_exists`**(*name*, *user=None*, *password=None*, *host=None*, *port=None*)

Checks if a database exists in Mongodb

CLI Example:

```
salt '*' mongodb.db_exists <name> <user> <password> <host> <port>
```

`salt.modules.mongodb.`**`db_list`**(*user=None*, *password=None*, *host=None*, *port=None*)

List all Mongodb databases

CLI Example:

```
salt '*' mongodb.db_list <user> <password> <host> <port>
```

`salt.modules.mongodb.`**`db_remove`**(*name*, *user=None*, *password=None*, *host=None*, *port=None*)

Remove a Mongodb database

CLI Example:

```
salt '*' mongodb.db_remove <name> <user> <password> <host> <port>
```

`salt.modules.mongodb.`**`user_create`**(*name*, *passwd*, *user=None*, *password=None*, *host=None*, *port=None*, *database='admin'*)

Create a Mongodb user

CLI Example:

```
salt '*' mongodb.user_create <name> <user> <password> <host> <port> <database>
```

`salt.modules.mongodb.`**`user_exists`**(*name*, *user=None*, *password=None*, *host=None*, *port=None*, *database='admin'*)

Checks if a user exists in Mongodb

CLI Example:

```
salt '*' mongodb.user_exists <name> <user> <password> <host> <port> <database>
```

`salt.modules.mongodb.`**`user_list`**(*user=None*, *password=None*, *host=None*, *port=None*, *database='admin'*)

List users of a Mongodb database

CLI Example:

```
salt '*' mongodb.user_list <name> <user> <password> <host> <port> <database>
```

`salt.modules.mongodb.`**`user_remove`**(*name*, *user=None*, *password=None*, *host=None*, *port=None*, *database='admin'*)

Remove a Mongodb user

CLI Example:

```
salt '*' mongodb.user_remove <name> <user> <password> <host> <port> <database>
```

61.87 salt.modules.monit

Monit service module. This module will create a monit type service watcher.

`salt.modules.monit.`**`monitor`**(*name*)

monitor service via monit

CLI Example:

```
salt '*' monit.monitor <service name>
```

`salt.modules.monit.`**`restart`**(*name*)

Restart service via monit

CLI Example:

```
salt '*' monit.restart <service name>
```

`salt.modules.monit.`**`start`**(*name*)

CLI Example:

```
salt '*' monit.start <service name>
```

`salt.modules.monit.`**`stop`**(*name*)

Stops service via monit

CLI Example:

```
salt '*' monit.stop <service name>
```

`salt.modules.monit.`**`summary`**(*svc_name=''*)

Display a summary from monit

CLI Example:

```
salt '*' monit.summary
salt '*' monit.summary <service name>
```

`salt.modules.monit.`**`unmonitor`**(*name*)

Unmonitor service via monit

CLI Example:

```
salt '*' monit.unmonitor <service name>
```

61.88 salt.modules.moosefs

Module for gathering and managing information about MooseFS

`salt.modules.moosefs.`**`dirinfo`**(*path*, *opts=None*)

Return information on a directory located on the Moose

CLI Example:

```
salt '*' moosefs.dirinfo /path/to/dir/ [-[n][h|H]]
```

salt.modules.moosefs.**fileinfo**(*path*)
Return information on a file located on the Moose

CLI Example:

```
salt '*' moosefs.fileinfo /path/to/dir/
```

salt.modules.moosefs.**getgoal**(*path*, *opts=None*)
Return goal(s) for a file or directory

CLI Example:

```
salt '*' moosefs.getgoal /path/to/file [-[n][h|H]]
salt '*' moosefs.getgoal /path/to/dir/ [-[n][h|H][r]]
```

salt.modules.moosefs.**mounts**()
Return a list of current MooseFS mounts

CLI Example:

```
salt '*' moosefs.mounts
```

61.89 salt.modules.mount

Salt module to manage unix mounts and the fstab file

salt.modules.mount.**active**()
List the active mounts.

CLI Example:

```
salt '*' mount.active
```

salt.modules.mount.**fstab**(*config='/etc/fstab'*)
List the contents of the fstab

CLI Example:

```
salt '*' mount.fstab
```

salt.modules.mount.**is_fuse_exec**(*cmd*)
Returns true if the command passed is a fuse mountable application.

CLI Example:

```
salt '*' mount.is_fuse_exec sshfs
```

salt.modules.mount.**mount**(*name*, *device*, *mkmnt=False*, *fstype=''*, *opts='defaults'*)
Mount a device

CLI Example:

```
salt '*' mount.mount /mnt/foo /dev/sdz1 True
```

salt.modules.mount.**remount**(*name*, *device*, *mkmnt=False*, *fstype=''*, *opts='defaults'*)
Attempt to remount a device, if the device is not already mounted, mount is called

CLI Example:

```
salt '*' mount.remount /mnt/foo /dev/sdz1 True
```

`salt.modules.mount.`**`rm_fstab`**`(`*name*, *config='/etc/fstab'*`)`

Remove the mount point from the fstab

CLI Example:

```
salt '*' mount.rm_fstab /mnt/foo
```

`salt.modules.mount.`**`set_fstab`**`(`*name*, *device*, *fstype*, *opts='defaults'*, *dump=0*, *pass_num=0*, *config='/etc/fstab'*, *test=False*, ***kwargs*`)`

Verify that this mount is represented in the fstab, change the mount to match the data passed, or add the mount if it is not present.

CLI Example:

```
salt '*' mount.set_fstab /mnt/foo /dev/sdz1 ext4
```

`salt.modules.mount.`**`swapoff`**`(`*name*`)`

Deactivate a named swap mount

CLI Example:

```
salt '*' mount.swapoff /root/swapfile
```

`salt.modules.mount.`**`swapon`**`(`*name*, *priority=None*`)`

Activate a swap disk

CLI Example:

```
salt '*' mount.swapon /root/swapfile
```

`salt.modules.mount.`**`swaps`**`()`

Return a dict containing information on active swap

CLI Example:

```
salt '*' mount.swaps
```

`salt.modules.mount.`**`umount`**`(`*name*`)`

Attempt to unmount a device by specifying the directory it is mounted on

CLI Example:

```
salt '*' mount.umount /mnt/foo
```

61.90 salt.modules.munin

Run munin plugins/checks from salt and format the output as data.

`salt.modules.munin.`**`list_plugins`**`()`

List all the munin plugins

CLI Example:

```
salt '*' munin.list_plugins
```

`salt.modules.munin.`**`run`**`(`*plugins*`)`

Run one or more named munin plugins

CLI Example:

```
salt '*' munin.run uptime
salt '*' munin.run uptime,cpu,load,memory
```

`salt.modules.munin.`**`run_all`**`()`

Run all the munin plugins

CLI Example:

```
salt '*' munin.run_all
```

61.91 salt.modules.mysql

Module to provide MySQL compatibility to salt.

depends

- MySQLdb Python module

Note: On CentOS 5 (and possibly RHEL 5) both MySQL-python and python26-mysqldb need to be installed.

configuration In order to connect to MySQL, certain configuration is required in /etc/salt/minion on the relevant minions. Some sample configs might look like:

```
mysql.host: 'localhost'
mysql.port: 3306
mysql.user: 'root'
mysql.pass: ''
mysql.db: 'mysql'
mysql.unix_socket: '/tmp/mysql.sock'
mysql.charset: 'utf8'
```

You can also use a defaults file:

```
mysql.default_file: '/etc/mysql/debian.cnf'
```

Changed in version 2014.1.0: (Hydrogen) charset connection argument added. This is a MySQL charset, not a python one

Changed in version 0.16.2: Connection arguments from the minion config file can be overridden on the CLI by using the arguments defined *here*. Additionally, it is now possible to setup a user with no password.

`salt.modules.mysql.`**`db_check`**`(`*name*, *table=None*, ***connection_args*`)`

Repairs the full database or just a given table

CLI Example:

```
salt '*' mysql.db_check dbname
salt '*' mysql.db_check dbname dbtable
```

`salt.modules.mysql.`**`db_create`**`(`*name*, *character_set=None*, *collate=None*, ***connection_args*`)`

Adds a databases to the MySQL server.

name The name of the database to manage

character_set The character set, if left empty the MySQL default will be used

collate The collation, if left empty the MySQL default will be used

CLI Example:

```
salt '*' mysql.db_create 'dbname'
salt '*' mysql.db_create 'dbname' 'utf8' 'utf8_general_ci'
```

`salt.modules.mysql.`**`db_exists`**`(`*name*, ***connection_args*`)`

Checks if a database exists on the MySQL server.

CLI Example:

```
salt '*' mysql.db_exists 'dbname'
```

`salt.modules.mysql.`**`db_list`**`(`***connection_args*`)`

Return a list of databases of a MySQL server using the output from the `SHOW DATABASES` query.

CLI Example:

```
salt '*' mysql.db_list
```

`salt.modules.mysql.`**`db_optimize`**`(`*name*, *table=None*, ***connection_args*`)`

Optimizes the full database or just a given table

CLI Example:

```
salt '*' mysql.db_optimize dbname
```

`salt.modules.mysql.`**`db_remove`**`(`*name*, ***connection_args*`)`

Removes a databases from the MySQL server.

CLI Example:

```
salt '*' mysql.db_remove 'dbname'
```

`salt.modules.mysql.`**`db_repair`**`(`*name*, *table=None*, ***connection_args*`)`

Repairs the full database or just a given table

CLI Example:

```
salt '*' mysql.db_repair dbname
```

`salt.modules.mysql.`**`db_tables`**`(`*name*, ***connection_args*`)`

Shows the tables in the given MySQL database (if exists)

CLI Example:

```
salt '*' mysql.db_tables 'database'
```

`salt.modules.mysql.`**`free_slave`**`(`***connection_args*`)`

Frees a slave from its master. This is a WIP, do not use.

CLI Example:

```
salt '*' mysql.free_slave
```

`salt.modules.mysql.`**`get_master_status`**`(`***connection_args*`)`

Retrieves the master status from the minion.

Returns:

{'host.domain.com': {'Binlog_Do_DB': '', 'Binlog_Ignore_DB': '', 'File': 'mysql-bin.000021', 'Position': 107}}

CLI Example:

```
salt '*' mysql.get_master_status
```

salt.modules.mysql.**get_slave_status**(***connection_args*)

Retrieves the slave status from the minion.

Returns:

```
{'host.domain.com': {'Connect_Retry': 60,
                     'Exec_Master_Log_Pos': 107,
                     'Last_Errno': 0,
                     'Last_Error': '',
                     'Last_IO_Errno': 0,
                     'Last_IO_Error': '',
                     'Last_SQL_Errno': 0,
                     'Last_SQL_Error': '',
                     'Master_Host': 'comet.scion-eng.com',
                     'Master_Log_File': 'mysql-bin.000021',
                     'Master_Port': 3306,
                     'Master_SSL_Allowed': 'No',
                     'Master_SSL_CA_File': '',
                     'Master_SSL_CA_Path': '',
                     'Master_SSL_Cert': '',
                     'Master_SSL_Cipher': '',
                     'Master_SSL_Key': '',
                     'Master_SSL_Verify_Server_Cert': 'No',
                     'Master_Server_Id': 1,
                     'Master_User': 'replu',
                     'Read_Master_Log_Pos': 107,
                     'Relay_Log_File': 'klo-relay-bin.000071',
                     'Relay_Log_Pos': 253,
                     'Relay_Log_Space': 553,
                     'Relay_Master_Log_File': 'mysql-bin.000021',
                     'Replicate_Do_DB': '',
                     'Replicate_Do_Table': '',
                     'Replicate_Ignore_DB': '',
                     'Replicate_Ignore_Server_Ids': '',
                     'Replicate_Ignore_Table': '',
                     'Replicate_Wild_Do_Table': '',
                     'Replicate_Wild_Ignore_Table': '',
                     'Seconds_Behind_Master': 0,
                     'Skip_Counter': 0,
                     'Slave_IO_Running': 'Yes',
                     'Slave_IO_State': 'Waiting for master to send event',
                     'Slave_SQL_Running': 'Yes',
                     'Until_Condition': 'None',
                     'Until_Log_File': '',
                     'Until_Log_Pos': 0}}
```

CLI Example:

```
salt '*' mysql.get_slave_status
```

salt.modules.mysql.**grant_add**(*grant*, *database*, *user*, *host='localhost'*, *grant_option=False*, *escape=True*, ***connection_args*)

Adds a grant to the MySQL server.

For database, make sure you specify database.table or database.*

CLI Example:

```
salt '*' mysql.grant_add             'SELECT,INSERT,UPDATE,...' 'database.*' 'frank' 'localhost'
```

salt.modules.mysql.**grant_exists**(*grant*, *database*, *user*, *host='localhost'*, *grant_option=False*, *escape=True*, **connection_args)

Checks to see if a grant exists in the database

CLI Example:

```
salt '*' mysql.grant_exists             'SELECT,INSERT,UPDATE,...' 'database.*' 'frank' 'localh
```

salt.modules.mysql.**grant_revoke**(*grant*, *database*, *user*, *host='localhost'*, *grant_option=False*, *escape=True*, **connection_args)

Removes a grant from the MySQL server.

CLI Example:

```
salt '*' mysql.grant_revoke            'SELECT,INSERT,UPDATE' 'database.*' 'frank' 'localhost'
```

salt.modules.mysql.**processlist**(**connection_args)

Retrieves the processlist from the MySQL server via "SHOW FULL PROCESSLIST".

Returns: a list of dicts, with each dict representing a process:

> **{'Command': 'Query',** 'Host': 'localhost', 'Id': 39, 'Info': 'SHOW FULL PROCESSLIST', 'Rows_examined': 0, 'Rows_read': 1, 'Rows_sent': 0, 'State': None, 'Time': 0, 'User': 'root', 'db': 'mysql'}

CLI Example:

```
salt '*' mysql.processlist
```

salt.modules.mysql.**query**(*database*, *query*, **connection_args)

Run an arbitrary SQL query and return the results or the number of affected rows.

CLI Example:

```
salt '*' mysql.query mydb "UPDATE mytable set myfield=1 limit 1"
```

Return data:

```
{'query time': {'human': '39.0ms', 'raw': '0.03899'}, 'rows affected': 1L}
```

CLI Example:

```
salt '*' mysql.query mydb "SELECT id,name,cash from users limit 3"
```

Return data:

```
{'columns': ('id', 'name', 'cash'),
    'query time': {'human': '1.0ms', 'raw': '0.001'},
    'results': ((1L, 'User 1', Decimal('110.000000')),
                (2L, 'User 2', Decimal('215.636756')),
                (3L, 'User 3', Decimal('0.040000'))),
    'rows returned': 3L}
```

CLI Example:

```
salt '*' mysql.query mydb 'INSERT into users values (null,"user 4", 5)'
```

Return data:

```
{'query time': {'human': '25.6ms', 'raw': '0.02563'}, 'rows affected': 1L}
```

CLI Example:

```
salt '*' mysql.query mydb 'DELETE from users where id = 4 limit 1'
```

Return data:

```
{'query time': {'human': '39.0ms', 'raw': '0.03899'}, 'rows affected': 1L}
```

Jinja Example: Run a query on `mydb` and use row 0, column 0's data.

```
{{ salt['mysql.query']('mydb', 'SELECT info from mytable limit 1')['results'][0][0] }}
```

`salt.modules.mysql.`**`quote_identifier`**`(`*identifier*, *for_grants=False*`)`

Return an identifier name (column, table, database, etc) escaped for MySQL

This means surrounded by "`" character and escaping this charater inside. It also means doubling the '%' character for MySQLdb internal usage.

Parameters

- **identifier** – the table, column or database identifier
- **for_grants** – is False by default, when using database names on grant queries you should set it to True to also escape "_" and "%" characters as requested by MySQL. Note that theses characters should only be escaped when requesting grants on the database level (*my_%db*.*) but not for table level grants (*my_%db*.`foo`)

CLI Example:

```
salt '*' mysql.quote_identifier 'foo`bar'
```

`salt.modules.mysql.`**`showglobal`**`(`***connection_args*`)`

Retrieves the show global variables from the minion.

Returns:: show global variables full dict

CLI Example:

```
salt '*' mysql.showglobal
```

`salt.modules.mysql.`**`showvariables`**`(`***connection_args*`)`

Retrieves the show variables from the minion.

Returns:: show variables full dict

CLI Example:

```
salt '*' mysql.showvariables
```

`salt.modules.mysql.`**`slave_lag`**`(`***connection_args*`)`

Return the number of seconds that a slave SQL server is lagging behind the master, if the host is not a slave it will return -1. If the server is configured to be a slave for replication but slave IO is not running then -2 will be returned. If there was an error connecting to the database or checking the slave status, -3 will be returned.

CLI Example:

```
salt '*' mysql.slave_lag
```

`salt.modules.mysql.`**`status`**`(`***connection_args*`)`

Return the status of a MySQL server using the output from the `SHOW STATUS` query.

CLI Example:

```
salt ’*’ mysql.status
```

salt.modules.mysql.**tokenize_grant**(*grant*)

External wrapper function :param grant: :return: dict

CLI Example:

```
salt ’*’ mysql.tokenize_grant              "GRANT SELECT, INSERT ON testdb.* TO ’testuser’@’local
```

salt.modules.mysql.**user_chpass**(*user*, *host=’localhost’*, *password=None*, *password_hash=None*, *allow_passwordless=False*, *unix_socket=None*, ***connection_args*)

Change password for a MySQL user

host Host for which this user/password combo applies

password The password to set for the new user. Will take precedence over the `password_hash` option if both are specified.

password_hash The password in hashed form. Be sure to quote the password because YAML doesn’t like the `*`. A password hash can be obtained from the mysql command-line client like so:

```
mysql> SELECT PASSWORD(’mypass’);
+-------------------------------------------+
| PASSWORD(’mypass’)                        |
+-------------------------------------------+
| *6C8989366EAF75BB670AD8EA7A7FC1176A95CEF4 |
+-------------------------------------------+
1 row in set (0.00 sec)
```

allow_passwordless If `True`, then `password` and `password_hash` can be omitted (or set to `None`) to permit a passwordless login.

New in version 0.16.2: The `allow_passwordless` option was added.

CLI Examples:

```
salt ’*’ mysql.user_chpass frank localhost newpassword
salt ’*’ mysql.user_chpass frank localhost password_hash=’hash’
salt ’*’ mysql.user_chpass frank localhost allow_passwordless=True
```

salt.modules.mysql.**user_create**(*user*, *host=’localhost’*, *password=None*, *password_hash=None*, *allow_passwordless=False*, *unix_socket=False*, ***connection_args*)

Creates a MySQL user

host Host for which this user/password combo applies

password The password to use for the new user. Will take precedence over the `password_hash` option if both are specified.

password_hash The password in hashed form. Be sure to quote the password because YAML doesn’t like the `*`. A password hash can be obtained from the mysql command-line client like so:

```
mysql> SELECT PASSWORD(’mypass’);
+-------------------------------------------+
| PASSWORD(’mypass’)                        |
+-------------------------------------------+
| *6C8989366EAF75BB670AD8EA7A7FC1176A95CEF4 |
+-------------------------------------------+
1 row in set (0.00 sec)
```

allow_passwordless If `True`, then `password` and `password_hash` can be omitted (or set to `None`) to permit a passwordless login.

unix_socket If `True` and allow_passwordless is `True` then will be used unix_socket auth plugin.

New in version 0.16.2: The `allow_passwordless` option was added.

CLI Examples:

```
salt '*' mysql.user_create 'username' 'hostname' 'password'
salt '*' mysql.user_create 'username' 'hostname' password_hash='hash'
salt '*' mysql.user_create 'username' 'hostname' allow_passwordless=True
```

`salt.modules.mysql.`**`user_exists`**(*user*, *host='localhost'*, *password=None*, *password_hash=None*, *passwordless=False*, *unix_socket=False*, ***connection_args*)

Checks if a user exists on the MySQL server. A login can be checked to see if passwordless login is permitted by omitting `password` and `password_hash`, and using `passwordless=True`.

New in version 0.16.2: The `passwordless` option was added.

CLI Example:

```
salt '*' mysql.user_exists 'username' 'hostname' 'password'
salt '*' mysql.user_exists 'username' 'hostname' password_hash='hash'
salt '*' mysql.user_exists 'username' passwordless=True
```

`salt.modules.mysql.`**`user_grants`**(*user*, *host='localhost'*, ***connection_args*)

Shows the grants for the given MySQL user (if it exists)

CLI Example:

```
salt '*' mysql.user_grants 'frank' 'localhost'
```

`salt.modules.mysql.`**`user_info`**(*user*, *host='localhost'*, ***connection_args*)

Get full info on a MySQL user

CLI Example:

```
salt '*' mysql.user_info root localhost
```

`salt.modules.mysql.`**`user_list`**(***connection_args*)

Return a list of users on a MySQL server

CLI Example:

```
salt '*' mysql.user_list
```

`salt.modules.mysql.`**`user_remove`**(*user*, *host='localhost'*, ***connection_args*)

Delete MySQL user

CLI Example:

```
salt '*' mysql.user_remove frank localhost
```

`salt.modules.mysql.`**`version`**(***connection_args*)

Return the version of a MySQL server using the output from the `SELECT VERSION()` query.

CLI Example:

```
salt '*' mysql.version
```

61.92 salt.modules.netbsd_sysctl

Module for viewing and modifying sysctl parameters

`salt.modules.netbsd_sysctl.`**`assign`**`(`*name*, *value*`)`

Assign a single sysctl parameter for this minion

CLI Example:

```
salt '*' sysctl.assign net.inet.icmp.icmplim 50
```

`salt.modules.netbsd_sysctl.`**`get`**`(`*name*`)`

Return a single sysctl parameter for this minion

CLI Example:

```
salt '*' sysctl.get hw.physmem
```

`salt.modules.netbsd_sysctl.`**`persist`**`(`*name*, *value*`)`

Assign and persist a simple sysctl parameter for this minion

CLI Example:

```
salt '*' sysctl.persist net.inet.icmp.icmplim 50
```

`salt.modules.netbsd_sysctl.`**`show`**`()`

Return a list of sysctl parameters for this minion

CLI Example:

```
salt '*' sysctl.show
```

61.93 salt.modules.netbsdservice

The service module for NetBSD

`salt.modules.netbsdservice.`**`available`**`(`*name*`)`

Returns `True` if the specified service is available, otherwise returns `False`.

CLI Example:

```
salt '*' service.available sshd
```

`salt.modules.netbsdservice.`**`disable`**`(`*name*, ***kwargs*`)`

Disable the named service to start at boot

CLI Example:

```
salt '*' service.disable <service name>
```

`salt.modules.netbsdservice.`**`disabled`**`(`*name*`)`

Return True if the named service is enabled, false otherwise

CLI Example:

```
salt '*' service.disabled <service name>
```

`salt.modules.netbsdservice.`**`enable`**`(`*name*, ***kwargs*`)`

Enable the named service to start at boot

CLI Example:

```
salt '*' service.enable <service name>
```

`salt.modules.netbsdservice.`**`enabled`**`(`*name*`)`

Return True if the named service is enabled, false otherwise

CLI Example:

```
salt '*' service.enabled <service name>
```

`salt.modules.netbsdservice.`**`force_reload`**`(`*name*`)`

Force-reload the named service

CLI Example:

```
salt '*' service.force_reload <service name>
```

`salt.modules.netbsdservice.`**`get_all`**`()`

Return all available boot services

CLI Example:

```
salt '*' service.get_all
```

`salt.modules.netbsdservice.`**`get_disabled`**`()`

Return a set of services that are installed but disabled

CLI Example:

```
salt '*' service.get_disabled
```

`salt.modules.netbsdservice.`**`get_enabled`**`()`

Return a list of service that are enabled on boot

CLI Example:

```
salt '*' service.get_enabled
```

`salt.modules.netbsdservice.`**`missing`**`(`*name*`)`

The inverse of service.available. Returns `True` if the specified service is not available, otherwise returns `False`.

CLI Example:

```
salt '*' service.missing sshd
```

`salt.modules.netbsdservice.`**`reload_`**`(`*name*`)`

Reload the named service

CLI Example:

```
salt '*' service.reload <service name>
```

`salt.modules.netbsdservice.`**`restart`**`(`*name*`)`

Restart the named service

CLI Example:

```
salt '*' service.restart <service name>
```

`salt.modules.netbsdservice.`**`start`**`(`*name*`)`

Start the specified service

CLI Example:

```
salt '*' service.start <service name>
```

`salt.modules.netbsdservice.`**`status`**`(`*name*, *sig=None*`)`
Return the status for a service, returns a bool whether the service is running.

CLI Example:

```
salt '*' service.status <service name>
```

`salt.modules.netbsdservice.`**`stop`**`(`*name*`)`
Stop the specified service

CLI Example:

```
salt '*' service.stop <service name>
```

61.94 salt.modules.network

Module for gathering and managing network information

`salt.modules.network.`**`active_tcp`**`()`
Return a dict containing information on all of the running TCP connections

CLI Example:

```
salt '*' network.active_tcp
```

`salt.modules.network.`**`arp`**`()`
Return the arp table from the minion

CLI Example:

```
salt '*' network.arp
```

`salt.modules.network.`**`dig`**`(`*host*`)`
Performs a DNS lookup with dig

CLI Example:

```
salt '*' network.dig archlinux.org
```

`salt.modules.network.`**`get_hostname`**`()`
Get hostname

CLI Example:

```
salt '*' network.get_hostname
```

`salt.modules.network.`**`hw_addr`**`(`*iface*`)`
Return the hardware address (a.k.a. MAC address) for a given interface

CLI Example:

```
salt '*' network.hw_addr eth0
```

`salt.modules.network.`**`hwaddr`**`(`*iface*`)`
Return the hardware address (a.k.a. MAC address) for a given interface

CLI Example:

```
salt '*' network.hw_addr eth0
```

salt.modules.network.**in_subnet**(*cidr*)

Returns True if host is within specified subnet, otherwise False.

CLI Example:

```
salt '*' network.in_subnet 10.0.0.0/16
```

salt.modules.network.**interfaces**()

Return a dictionary of information about all the interfaces on the minion

CLI Example:

```
salt '*' network.interfaces
```

salt.modules.network.**ip_addrs**(*interface=None*, *include_loopback=False*, *cidr=None*)

Returns a list of IPv4 addresses assigned to the host. 127.0.0.1 is ignored, unless 'include_loopback=True' is indicated. If 'interface' is provided, then only IP addresses from that interface will be returned. Providing a CIDR via 'cidr="10.0.0.0/8"' will return only the addresses which are within that subnet.

CLI Example:

```
salt '*' network.ip_addrs
```

salt.modules.network.**ip_addrs6**(*interface=None*, *include_loopback=False*)

Returns a list of IPv6 addresses assigned to the host. ::1 is ignored, unless 'include_loopback=True' is indicated. If 'interface' is provided, then only IP addresses from that interface will be returned.

CLI Example:

```
salt '*' network.ip_addrs6
```

salt.modules.network.**ipaddrs**(*interface=None*, *include_loopback=False*, *cidr=None*)

Returns a list of IPv4 addresses assigned to the host. 127.0.0.1 is ignored, unless 'include_loopback=True' is indicated. If 'interface' is provided, then only IP addresses from that interface will be returned. Providing a CIDR via 'cidr="10.0.0.0/8"' will return only the addresses which are within that subnet.

CLI Example:

```
salt '*' network.ip_addrs
```

salt.modules.network.**ipaddrs6**(*interface=None*, *include_loopback=False*)

Returns a list of IPv6 addresses assigned to the host. ::1 is ignored, unless 'include_loopback=True' is indicated. If 'interface' is provided, then only IP addresses from that interface will be returned.

CLI Example:

```
salt '*' network.ip_addrs6
```

salt.modules.network.**mod_hostname**(*hostname*)

Modify hostname

CLI Example:

```
salt '*' network.mod_hostname   master.saltstack.com
```

salt.modules.network.**netstat**()

Return information on open ports and states

CLI Example:

```
salt '*' network.netstat
```

`salt.modules.network.`**`ping`**`(`*host*`)`

Performs a ping to a host

CLI Example:

```
salt '*' network.ping archlinux.org
```

`salt.modules.network.`**`subnets`**`()`

Returns a list of subnets to which the host belongs

CLI Example:

```
salt '*' network.subnets
```

`salt.modules.network.`**`traceroute`**`(`*host*`)`

Performs a traceroute to a 3rd party host

CLI Example:

```
salt '*' network.traceroute archlinux.org
```

61.95 salt.modules.nfs3

Module for managing NFS version 3.

`salt.modules.nfs3.`**`del_export`**`(`*exports='/etc/exports', path=None*`)`

Remove an export

CLI Example:

```
salt '*' nfs.del_export /media/storage
```

`salt.modules.nfs3.`**`list_exports`**`(`*exports='/etc/exports'*`)`

List configured exports

CLI Example:

```
salt '*' nfs.list_exports
```

61.96 salt.modules.nginx

Support for nginx

`salt.modules.nginx.`**`configtest`**`()`

test configuration and exit

CLI Example:

```
salt '*' nginx.configtest
```

`salt.modules.nginx.`**`signal`**`(`*signal=None*`)`

Signals nginx to start, reload, reopen or stop.

CLI Example:

```
salt '*' nginx.signal reload
```

`salt.modules.nginx.`**`version`**`()`

Return server version from nginx -v

CLI Example:

```
salt '*' nginx.version
```

61.97 salt.modules.nova

Module for handling OpenStack Nova calls.

depends

- novaclient Python module

configuration This module is not usable until the user, password, tenant, and auth URL are specified either in a pillar or in the minion's config file. For example:

```
keystone.user: admin
keystone.password: verybadpass
keystone.tenant: admin
keystone.auth_url: 'http://127.0.0.1:5000/v2.0/'
# Optional
keystone.region_name: 'regionOne'
```

If configuration for multiple OpenStack accounts is required, they can be set up as different configuration profiles: For example:

```
openstack1:
  keystone.user: admin
  keystone.password: verybadpass
  keystone.tenant: admin
  keystone.auth_url: 'http://127.0.0.1:5000/v2.0/'

openstack2:
  keystone.user: admin
  keystone.password: verybadpass
  keystone.tenant: admin
  keystone.auth_url: 'http://127.0.0.2:5000/v2.0/'
```

With this configuration in place, any of the nova functions can make use of a configuration profile by declaring it explicitly. For example:

```
salt '*' nova.flavor_list profile=openstack1
```

`salt.modules.nova.`**`boot`**`(`*name*, *flavor_id=0*, *image_id=0*, *profile=None*, *timeout=300*`)`

Boot (create) a new instance

name Name of the new instance (must be first)

flavor_id Unique integer ID for the flavor

image_id Unique integer ID for the image

timeout How long to wait, after creating the instance, for the provider to return information about it (default 300 seconds).

New in version 2014.1.0: (Hydrogen)

CLI Example:

```
salt '*' nova.boot myinstance flavor_id=4596 image_id=2
```

The flavor_id and image_id are obtained from nova.flavor_list and nova.image_list

```
salt '*' nova.flavor_list
salt '*' nova.image_list
```

`salt.modules.nova.`**`delete`**(*instance_id*, *profile=None*)

Delete an instance

instance_id ID of the instance to be deleted

CLI Example:

```
salt '*' nova.delete 1138
```

`salt.modules.nova.`**`flavor_create`**(*name*, *id=0*, *ram=0*, *disk=0*, *vcpus=1*, *profile=None*)

Add a flavor to nova (nova flavor-create). The following parameters are required:

name Name of the new flavor (must be first)

id Unique integer ID for the new flavor

ram Memory size in MB

disk Disk size in GB

vcpus Number of vcpus

CLI Example:

```
salt '*' nova.flavor_create myflavor id=6 ram=4096 disk=10 vcpus=1
```

`salt.modules.nova.`**`flavor_delete`**(*id*, *profile=None*)

Delete a flavor from nova by id (nova flavor-delete)

CLI Example:

```
salt '*' nova.flavor_delete 7
```

`salt.modules.nova.`**`flavor_list`**(*profile=None*)

Return a list of available flavors (nova flavor-list)

CLI Example:

```
salt '*' nova.flavor_list
```

`salt.modules.nova.`**`image_list`**(*name=None*, *profile=None*)

Return a list of available images (nova images-list + nova image-show) If a name is provided, only that image will be displayed.

CLI Examples:

```
salt '*' nova.image_list
salt '*' nova.image_list myimage
```

`salt.modules.nova.`**`image_meta_delete`**(*id=None*, *name=None*, *keys=None*, *profile=None*)

Delete a key=value pair from the metadata for an image (nova image-meta set)

CLI Examples:

```
salt '*' nova.image_meta_delete id=6f52b2ff-0b31-4d84-8fd1-af45b84824f6 keys=cheese
salt '*' nova.image_meta_delete name=myimage keys=salad,beans
```

`salt.modules.nova.`**`image_meta_set`**(*id=None*, *name=None*, *profile=None*, ***kwargs*)

Sets a key=value pair in the metadata for an image (nova image-meta set)

CLI Examples:

```
salt '*' nova.image_meta_set id=6f52b2ff-0b31-4d84-8fd1-af45b84824f6 cheese=gruyere
salt '*' nova.image_meta_set name=myimage salad=pasta beans=baked
```

`salt.modules.nova.`**`keypair_add`**(*name*, *pubfile=None*, *pubkey=None*, *profile=None*)

Add a keypair to nova (nova keypair-add)

CLI Examples:

```
salt '*' nova.keypair_add mykey pubfile='/home/myuser/.ssh/id_rsa.pub'
salt '*' nova.keypair_add mykey pubkey='ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEAuGj4A7HcPLPl/etc== m
```

`salt.modules.nova.`**`keypair_delete`**(*name*, *profile=None*)

Add a keypair to nova (nova keypair-delete)

CLI Example:

```
salt '*' nova.keypair_delete mykey'
```

`salt.modules.nova.`**`keypair_list`**(*profile=None*)

Return a list of available keypairs (nova keypair-list)

CLI Example:

```
salt '*' nova.keypair_list
```

`salt.modules.nova.`**`list_`**(*profile=None*)

To maintain the feel of the nova command line, this function simply calls the server_list function.

`salt.modules.nova.`**`lock`**(*instance_id*, *profile=None*)

Lock an instance

instance_id ID of the instance to be locked

CLI Example:

```
salt '*' nova.lock 1138
```

`salt.modules.nova.`**`resume`**(*instance_id*, *profile=None*)

Resume an instance

instance_id ID of the instance to be resumed

CLI Example:

```
salt '*' nova.resume 1138
```

`salt.modules.nova.`**`secgroup_create`**(*name*, *description*, *profile=None*)

Add a secgroup to nova (nova secgroup-create)

CLI Example:

```
salt '*' nova.secgroup_create mygroup 'This is my security group'
```

`salt.modules.nova.`**`secgroup_delete`**(*name*, *profile=None*)

Delete a secgroup to nova (nova secgroup-delete)

CLI Example:

```
salt '*' nova.secgroup_delete mygroup
```

`salt.modules.nova.`**`secgroup_list`**(*profile=None*)
Return a list of available security groups (nova items-list)

CLI Example:

```
salt '*' nova.secgroup_list
```

`salt.modules.nova.`**`server_list`**(*profile=None*)
Return list of active servers

CLI Example:

```
salt '*' nova.show
```

`salt.modules.nova.`**`server_list_detailed`**(*profile=None*)
Return detailed list of active servers

CLI Example:

```
salt '*' nova.server_list_detailed
```

`salt.modules.nova.`**`server_show`**(*server_id*, *profile=None*)
Return detailed information for an active server

CLI Example:

```
salt '*' nova.server_show <server_id>
```

`salt.modules.nova.`**`show`**(*server_id*, *profile=None*)
To maintain the feel of the nova command line, this function simply calls the server_show function.

CLI Example:

```
salt '*' nova.show
```

`salt.modules.nova.`**`suspend`**(*instance_id*, *profile=None*)
Suspend an instance

instance_id ID of the instance to be suspended

CLI Example:

```
salt '*' nova.suspend 1138
```

61.98 salt.modules.npm

Manage and query NPM packages.

`salt.modules.npm.`**`install`**(*pkg=None*, *dir=None*, *runas=None*)
Install an NPM package.

If no directory is specified, the package will be installed globally. If no package is specified, the dependencies (from package.json) of the package in the given directory will be installed.

pkg A package name in any format accepted by NPM

dir The target directory in which to install the package, or None for global installation

runas The user to run NPM with

CLI Example:

```
salt '*' npm.install coffee-script
```

`salt.modules.npm.`**`list_`**(*pkg=None*, *dir=None*)

List installed NPM packages.

If no directory is specified, this will return the list of globally- installed packages.

pkg Limit package listing by name

dir The directory whose packages will be listed, or None for global installation

CLI Example:

```
salt '*' npm.list
```

`salt.modules.npm.`**`uninstall`**(*pkg*, *dir=None*, *runas=None*)

Uninstall an NPM package.

If no directory is specified, the package will be uninstalled globally.

pkg A package name in any format accepted by NPM

dir The target directory from which to uninstall the package, or None for global installation

runas The user to run NPM with

CLI Example:

```
salt '*' npm.uninstall coffee-script
```

61.99 salt.modules.omapi

This module interacts with an ISC DHCP Server via OMAPI. server_ip and server_port params may be set in the minion config or pillar:

```
omapi.server_ip: 127.0.0.1
omapi.server_port: 7991
```

depends pypureomapi Python module

`salt.modules.omapi.`**`add_host`**(*mac*, *name=None*, *ip=None*, *ddns=False*, *group=None*, *supersede_host=False*)

Add a host object for the given mac.

CLI Example:

```
salt dhcp-server omapi.add_host ab:ab:ab:ab:ab:ab name=host1
```

Add ddns-hostname and a fixed-ip statements:

```
salt dhcp-server omapi.add_host ab:ab:ab:ab:ab:ab name=host1 ip=10.1.1.1 ddns=true
```

`salt.modules.omapi.`**`delete_host`**(*mac=None*, *name=None*)

Delete the host with the given mac or name.

CLI Examples:

```
salt dhcp-server omapi.delete_host name=host1
salt dhcp-server omapi.delete_host mac=ab:ab:ab:ab:ab:ab
```

61.100 salt.modules.openbsdpkg

Package support for OpenBSD

`salt.modules.openbsdpkg.`**`install`**(*name=None*, *pkgs=None*, *sources=None*, ***kwargs*)

Install the passed package

Return a dict containing the new package names and versions:

```
{'<package>': {'old': '<old-version>',
               'new': '<new-version>'}}
```

CLI Example, Install one package:

```
salt '*' pkg.install <package name>
```

CLI Example, Install more than one package:

```
salt '*' pkg.install pkgs='["<package name>", "<package name>"]'
```

CLI Example, Install more than one package from a alternate source (e.g. salt file-server, HTTP, FTP, local filesystem):

```
salt '*' pkg.install sources='[{"<pkg name>": "salt://pkgs/<pkg filename>"}]'
```

`salt.modules.openbsdpkg.`**`latest_version`**(**names*, ***kwargs*)

The available version of the package in the repository

CLI Example:

```
salt '*' pkg.latest_version <package name>
```

`salt.modules.openbsdpkg.`**`list_pkgs`**(*versions_as_list=False*, ***kwargs*)

List the packages currently installed as a dict:

```
{'<package_name>': '<version>'}
```

CLI Example:

```
salt '*' pkg.list_pkgs
```

`salt.modules.openbsdpkg.`**`purge`**(*name=None*, *pkgs=None*, ***kwargs*)

Package purges are not supported, this function is identical to `remove()`.

name The name of the package to be deleted.

Multiple Package Options:

pkgs A list of packages to delete. Must be passed as a python list. The `name` parameter will be ignored if this option is passed.

New in version 0.16.0.

Returns a dict containing the changes.

CLI Example:

```
salt '*' pkg.purge <package name>
salt '*' pkg.purge <package1>,<package2>,<package3>
salt '*' pkg.purge pkgs='["foo", "bar"]'
```

`salt.modules.openbsdpkg.`**`remove`**`(`*name=None*, *pkgs=None*, ***kwargs*`)`

Remove a single package with pkg_delete

Multiple Package Options:

pkgs A list of packages to delete. Must be passed as a python list. The `name` parameter will be ignored if this option is passed.

New in version 0.16.0.

Returns a dict containing the changes.

CLI Example:

```
salt '*' pkg.remove <package name>
salt '*' pkg.remove <package1>,<package2>,<package3>
salt '*' pkg.remove pkgs='["foo", "bar"]'
```

`salt.modules.openbsdpkg.`**`version`**`(`**names*, ***kwargs*`)`

Returns a string representing the package version or an empty string if not installed. If more than one package name is specified, a dict of name/version pairs is returned.

CLI Example:

```
salt '*' pkg.version <package name>
salt '*' pkg.version <package1> <package2> <package3> ...
```

61.101 salt.modules.openbsdservice

The service module for OpenBSD

`salt.modules.openbsdservice.`**`restart`**`(`*name*`)`

Restart the named service

CLI Example:

```
salt '*' service.restart <service name>
```

`salt.modules.openbsdservice.`**`start`**`(`*name*`)`

Start the specified service

CLI Example:

```
salt '*' service.start <service name>
```

`salt.modules.openbsdservice.`**`status`**`(`*name*, *sig=None*`)`

Return the status for a service, returns a bool whether the service is running.

CLI Example:

```
salt '*' service.status <service name>
```

`salt.modules.openbsdservice.`**`stop`**`(`*name*`)`

Stop the specified service

CLI Example:

```
salt '*' service.stop <service name>
```

61.102 salt.modules.openstack_config

Modify, retrieve, or delete values from OpenStack configuration files.

maintainer Jeffrey C. Ollie <jeff@ocjtech.us>

maturity new

depends

platform linux

`salt.modules.openstack_config.`**`delete`**(*filename*, *section*, *parameter*)

Delete a value from an OpenStack configuration file.

filename The full path to the configuration file

section The section from which to delete the parameter

parameter The parameter to delete

CLI Example:

```
salt-call openstack_config.delete /etc/keystone/keystone.conf sql connection
```

`salt.modules.openstack_config.`**`get`**(*filename*, *section*, *parameter*)

Get a value from an OpenStack configuration file.

filename The full path to the configuration file

section The section from which to search for the parameter

parameter The parameter to return

CLI Example:

```
salt-call openstack_config.get /etc/keystone/keystone.conf sql connection
```

`salt.modules.openstack_config.`**`set_`**(*filename*, *section*, *parameter*, *value*)

Set a value in an OpenStack configuration file.

filename The full path to the configuration file

section The section in which the parameter will be set

parameter The parameter to change

value The value to set

CLI Example:

```
salt-call openstack_config.set /etc/keystone/keystone.conf sql connection foo
```

61.103 salt.modules.osxdesktop

Mac OS X implementations of various commands in the "desktop" interface

`salt.modules.osxdesktop.`**`get_output_volume`**()

Get the output volume (range 0 to 100)

CLI Example:

```
salt '*' desktop.get_output_volume
```

`salt.modules.osxdesktop.`**`lock`**`()`

Lock the desktop session

CLI Example:

```
salt '*' desktop.lock
```

`salt.modules.osxdesktop.`**`say`**`(`**words*`)`

Say some words.

CLI Example:

```
salt '*' desktop.say <word0> <word1> ... <wordN>
```

`salt.modules.osxdesktop.`**`screensaver`**`()`

Launch the screensaver

CLI Example:

```
salt '*' desktop.screensaver
```

`salt.modules.osxdesktop.`**`set_output_volume`**`(`*volume*`)`

Set the volume of sound (range 0 to 100)

CLI Example:

```
salt '*' desktop.set_output_volume <volume>
```

61.104 salt.modules.pacman

A module to wrap pacman calls, since Arch is the best (https://wiki.archlinux.org/index.php/Arch_is_the_best)

`salt.modules.pacman.`**`file_dict`**`(`**packages*`)`

List the files that belong to a package, grouped by package. Not specifying any packages will return a list of _every_ file on the system's package database (not generally recommended).

CLI Examples:

```
salt '*' pkg.file_list httpd
salt '*' pkg.file_list httpd postfix
salt '*' pkg.file_list
```

`salt.modules.pacman.`**`file_list`**`(`**packages*`)`

List the files that belong to a package. Not specifying any packages will return a list of _every_ file on the system's package database (not generally recommended).

CLI Examples:

```
salt '*' pkg.file_list httpd
salt '*' pkg.file_list httpd postfix
salt '*' pkg.file_list
```

`salt.modules.pacman.`**`install`**`(`*name=None*, *refresh=True*, *pkgs=None*, *sources=None*, ***kwargs*`)`

Install the passed package, add refresh=True to install with an -Sy.

name The name of the package to be installed. Note that this parameter is ignored if either "pkgs" or "sources" is passed. Additionally, please note that this option can only be used to install packages from a software repository. To install a package file manually, use the "sources" option.

CLI Example:

```
salt ’*’ pkg.install <package name>
```

refresh Whether or not to refresh the package database before installing.

Multiple Package Installation Options:

pkgs A list of packages to install from a software repository. Must be passed as a python list. A specific version number can be specified by using a single-element dict representing the package and its version. As with the `version` parameter above, comparison operators can be used to target a specific version of a package.

CLI Examples:

```
salt ’*’ pkg.install pkgs=’["foo", "bar"]’
salt ’*’ pkg.install pkgs=’["foo", {"bar": "1.2.3-4"}]’
salt ’*’ pkg.install pkgs=’["foo", {"bar": "<1.2.3-4"}]’
```

sources A list of packages to install. Must be passed as a list of dicts, with the keys being package names, and the values being the source URI or local path to the package.

CLI Example:

```
salt ’*’ pkg.install sources=’[{"foo": "salt://foo.pkg.tar.xz"},{"bar": "salt://bar.pkg.tar.
```

Returns a dict containing the new package names and versions:

```
{’<package>’: {’old’: ’<old-version>’,
               ’new’: ’<new-version>’}}
```

`salt.modules.pacman.`**`latest_version`**`(`**names*, ***kwargs*`)`

Return the latest version of the named package available for upgrade or installation. If more than one package name is specified, a dict of name/version pairs is returned.

If the latest version of a given package is already installed, an empty string will be returned for that package.

CLI Example:

```
salt ’*’ pkg.latest_version <package name>
salt ’*’ pkg.latest_version <package1> <package2> <package3> ...
```

`salt.modules.pacman.`**`list_pkgs`**`(`*versions_as_list=False*, ***kwargs*`)`

List the packages currently installed as a dict:

```
{’<package_name>’: ’<version>’}
```

CLI Example:

```
salt ’*’ pkg.list_pkgs
```

`salt.modules.pacman.`**`list_upgrades`**`()`

List all available package upgrades on this system

CLI Example:

```
salt ’*’ pkg.list_upgrades
```

`salt.modules.pacman.`**`purge`**`(`*name=None*, *pkgs=None*, ***kwargs*`)`

Recursively remove a package and all dependencies which were installed with it, this will call a `pacman -Rsc`

name The name of the package to be deleted.

Multiple Package Options:

pkgs A list of packages to delete. Must be passed as a python list. The `name` parameter will be ignored if this option is passed.

New in version 0.16.0.

Returns a dict containing the changes.

CLI Example:

```
salt '*' pkg.purge <package name>
salt '*' pkg.purge <package1>,<package2>,<package3>
salt '*' pkg.purge pkgs='["foo", "bar"]'
```

`salt.modules.pacman.`**`refresh_db`**`()`

Just run a `pacman -Sy`, return a dict:

```
{'<database name>': Bool}
```

CLI Example:

```
salt '*' pkg.refresh_db
```

`salt.modules.pacman.`**`remove`**`(`*name=None*, *pkgs=None*, ***kwargs*`)`

Remove packages with `pacman -R`.

name The name of the package to be deleted.

Multiple Package Options:

pkgs A list of packages to delete. Must be passed as a python list. The `name` parameter will be ignored if this option is passed.

New in version 0.16.0.

Returns a dict containing the changes.

CLI Example:

```
salt '*' pkg.remove <package name>
salt '*' pkg.remove <package1>,<package2>,<package3>
salt '*' pkg.remove pkgs='["foo", "bar"]'
```

`salt.modules.pacman.`**`upgrade`**`()`

Run a full system upgrade, a pacman -Syu

Return a dict containing the new package names and versions:

```
{'<package>': {'old': '<old-version>',
               'new': '<new-version>'}}
```

CLI Example:

```
salt '*' pkg.upgrade
```

`salt.modules.pacman.`**`upgrade_available`**`(`*name*`)`

Check whether or not an upgrade is available for a given package

CLI Example:

```
salt '*' pkg.upgrade_available <package name>
```

`salt.modules.pacman.`**`version`**`(`**names*, ***kwargs*`)`

Returns a string representing the package version or an empty string if not installed. If more than one package name is specified, a dict of name/version pairs is returned.

CLI Example:

```
salt '*' pkg.version <package name>
salt '*' pkg.version <package1> <package2> <package3> ...
```

61.105 salt.modules.pagerduty

Module for Firing Events via PagerDuty

New in version 2014.1.0: (Hydrogen)

depends

- pygerduty python module

configuration This module can be used by either passing a jid and password directly to send_message, or by specifying the name of a configuration profile in the minion config, minion pillar, or master config.

For example:

```
my-pagerduty-account:
    pagerduty.api_key: F3Rbyjbve43rfFWf2214
    pagerduty.subdomain: mysubdomain
```

`salt.modules.pagerduty.`**`create_event`**(*service_key*, *description*, *details*, *incident_key=None*, *profile=None*)

Create an event in PagerDuty. Designed for use in states.

CLI Example:

pagerduty.create_event <service_key> <description> <details> profile=my-pagerduty-account

The following parameters are required:

service_key This key can be found by using pagerduty.list_services.

description This is a short description of the event.

details This can be a more detailed description of the event.

profile This refers to the configuration profile to use to connect to the PagerDuty service.

`salt.modules.pagerduty.`**`list_incidents`**(*profile*)

List services belonging to this account

CLI Example:

pagerduty.list_incidents my-pagerduty-account

`salt.modules.pagerduty.`**`list_services`**(*profile*)

List services belonging to this account

CLI Example:

pagerduty.list_services my-pagerduty-account

61.106 salt.modules.pam

Support for pam

`salt.modules.pam.`**`read_file`**`(`*file_name*`)`

This is just a test function, to make sure parsing works

CLI Example:

```
salt '*' pam.read_file /etc/pam.d/login
```

61.107 salt.modules.parted

Module for managing partitions on POSIX-like systems.

Some functions may not be available, depending on your version of parted.

Check the manpage for `parted(8)` for more information, or the online docs at:

http://www.gnu.org/software/parted/manual/html_chapter/parted_2.html

In light of parted not directly supporting partition IDs, some of this module has been written to utilize sfdisk instead. For further information, please reference the man page for `sfdisk(8)`.

`salt.modules.parted.`**`align_check`**`(`*device*, *part_type*, *partition*`)`

partition.align_check device part_type partition

Check if partition satisfies the alignment constraint of part_type. Type must be "minimal" or "optimal".

CLI Example:

```
salt '*' partition.align_check /dev/sda minimal 1
```

`salt.modules.parted.`**`check`**`(`*device*, *minor*`)`

partition.check device minor

Checks if the file system on partition <minor> has any errors.

CLI Example:

```
salt '*' partition.check 1
```

`salt.modules.parted.`**`cp`**`(`*device*, *from_minor*, *to_minor*`)`

partition.check device from_minor to_minor

Copies the file system on the partition <from-minor> to partition <to-minor>, deleting the original contents of the destination partition.

CLI Example:

```
salt '*' partition.cp /dev/sda 2 3
```

`salt.modules.parted.`**`get_id`**`(`*device*, *minor*`)`

Prints the system ID for the partition. Some typical values are:

```
 b: FAT32 (vfat)
 7: HPFS/NTFS
82: Linux Swap
83: Linux
8e: Linux LVM
fd: Linux RAID Auto
```

CLI Example:

```
salt '*' partition.get_id /dev/sda 1
```

`salt.modules.parted.`**`list_`**`(`*device*, *unit=None*`)`
partition.list device unit

Prints partition information of given <device>

CLI Examples:

```
salt '*' partition.list /dev/sda
salt '*' partition.list /dev/sda unit=s
salt '*' partition.list /dev/sda unit=kB
```

`salt.modules.parted.`**`mkfs`**`(`*device*, *fs_type*`)`
partition.mkfs device fs_type

Makes a file system <fs_type> on partition <device>, destroying all data that resides on that partition. <fs_type> must be one of "ext2", "fat32", "fat16", "linux-swap" or "reiserfs" (if libreiserfs is installed)

CLI Example:

```
salt '*' partition.mkfs /dev/sda2 fat32
```

`salt.modules.parted.`**`mklabel`**`(`*device*, *label_type*`)`
partition.mklabel device label_type

Create a new disklabel (partition table) of label_type. Type should be one of "aix", "amiga", "bsd", "dvh", "gpt", "loop", "mac", "msdos", "pc98", or "sun".

CLI Example:

```
salt '*' partition.mklabel /dev/sda msdos
```

`salt.modules.parted.`**`mkpart`**`(`*device*, *part_type*, *fs_type*, *start*, *end*`)`
partition.mkpart device part_type fs_type start end

Make a part_type partition for filesystem fs_type, beginning at start and ending at end (by default in megabytes). part_type should be one of "primary", "logical", or "extended".

CLI Example:

```
salt '*' partition.mkpart /dev/sda primary fat32 0 639
```

`salt.modules.parted.`**`mkpartfs`**`(`*device*, *part_type*, *fs_type*, *start*, *end*`)`
partition.mkpartfs device part_type fs_type start end

Make a <part_type> partition with a new filesystem of <fs_type>, beginning at <start> and ending at <end> (by default in megabytes). <part_type> should be one of "primary", "logical", or "extended". <fs_type> must be one of "ext2", "fat32", "fat16", "linux-swap" or "reiserfs" (if libreiserfs is installed)

CLI Example:

```
salt '*' partition.mkpartfs /dev/sda logical ext2 440 670
```

`salt.modules.parted.`**`name`**`(`*device*, *partition*, *name*`)`
partition.name device partition name

Set the name of partition to name. This option works only on Mac, PC98, and GPT disklabels. The name can be placed in quotes, if necessary.

CLI Example:

```
salt '*' partition.name /dev/sda 1 'My Documents'
```

`salt.modules.parted.`**`part_list`**`(`*device*, *unit=None*`)`

Deprecated. Calls partition.list.

CLI Examples:

```
salt '*' partition.part_list /dev/sda
salt '*' partition.part_list /dev/sda unit=s
salt '*' partition.part_list /dev/sda unit=kB
```

`salt.modules.parted.`**`probe`**`(`*device=''*`)`

Ask the kernel to update its local partition data

CLI Examples:

```
salt '*' partition.probe
salt '*' partition.probe /dev/sda
```

`salt.modules.parted.`**`rescue`**`(`*device*, *start*, *end*`)`

partition.rescue device start end

Rescue a lost partition that was located somewhere between start and end. If a partition is found, parted will ask if you want to create an entry for it in the partition table.

CLI Example:

```
salt '*' partition.rescue /dev/sda 0 8056
```

`salt.modules.parted.`**`resize`**`(`*device*, *minor*, *start*, *end*`)`

partition.resize device minor, start, end

Resizes the partition with number <minor>. The partition will start <start> from the beginning of the disk, and end <end> from the beginning of the disk. resize never changes the minor number. Extended partitions can be resized, so long as the new extended partition completely contains all logical partitions.

CLI Example:

```
salt '*' partition.resize /dev/sda 3 200 850
```

`salt.modules.parted.`**`rm`**`(`*device*, *minor*`)`

partition.rm device minor

Removes the partition with number <minor>.

CLI Example:

```
salt '*' partition.rm /dev/sda 5
```

`salt.modules.parted.`**`set_`**`(`*device*, *minor*, *flag*, *state*`)`

partition.set device minor flag state

Changes a flag on the partition with number <minor>. A flag can be either "on" or "off". Some or all of these flags will be available, depending on what disk label you are using.

CLI Example:

```
salt '*' partition.set /dev/sda 1 boot on
```

`salt.modules.parted.`**`set_id`**`(`*device*, *minor*, *system_id*`)`

Sets the system ID for the partition. Some typical values are:

```
 b: FAT32 (vfat)
 7: HPFS/NTFS
82: Linux Swap
83: Linux
8e: Linux LVM
fd: Linux RAID Auto
```

CLI Example:

```
salt '*' partition.set_id /dev/sda 1 83
```

`salt.modules.parted.`**`system_types`**`()`

List the system types that are supported by the installed version of sfdisk

CLI Example:

```
salt '*' partition.system_types
```

`salt.modules.parted.`**`toggle`**`(`*device*, *partition*, *flag*`)`

partition.toggle device partition flag

Toggle the state of <flag> on <partition>

CLI Example:

```
salt '*' partition.name /dev/sda 1 boot
```

61.108 salt.modules.pecl

Manage PHP pecl extensions.

`salt.modules.pecl.`**`install`**`(`*pecls*, *defaults=False*, *force=False*, *preferred_state='stable'*`)`

Installs one or several pecl extensions.

pecls The pecl extensions to install.

defaults Use default answers for extensions such as pecl_http which ask questions before installation. Without this option, the pecl.installed state will hang indefinitely when trying to install these extensions.

force Whether to force the installed version or not

Note: The `defaults` option will be available in version 0.17.0.

CLI Example:

```
salt '*' pecl.install fuse
```

`salt.modules.pecl.`**`list_`**`()`

List installed pecl extensions.

CLI Example:

```
salt '*' pecl.list
```

`salt.modules.pecl.`**`uninstall`**`(`*pecls*`)`

Uninstall one or several pecl extensions.

pecls The pecl extensions to uninstall.

CLI Example:

```
salt '*' pecl.uninstall fuse
```

`salt.modules.pecl.`**`update`**`(`*pecls*`)`

Update one or several pecl extensions.

pecls The pecl extensions to update.

CLI Example:

```
salt '*' pecl.update fuse
```

61.109 salt.modules.pillar

Extract the pillar data for this minion

`salt.modules.pillar.`**`ext`**`(`*external*`)`

Generate the pillar and apply an explicit external pillar

CLI Example:

```
salt '*' pillar.ext 'libvirt: _'
```

`salt.modules.pillar.`**`get`**`(`*key*, *default=''*`)`

New in version 0.14.

Attempt to retrieve the named value from pillar, if the named value is not available return the passed default. The default return is an empty string.

The value can also represent a value in a nested dict using a ":" delimiter for the dict. This means that if a dict in pillar looks like this:

```
{'pkg': {'apache': 'httpd'}}
```

To retrieve the value associated with the apache key in the pkg dict this key can be passed:

```
pkg:apache
```

CLI Example:

```
salt '*' pillar.get pkg:apache
```

`salt.modules.pillar.`**`item`**`(`**args*`)`

New in version 0.16.2.

Return one ore more pillar entries

CLI Examples:

```
salt '*' pillar.item foo
salt '*' pillar.item foo bar baz
```

`salt.modules.pillar.`**`items`**`(`**args*`)`

This function calls the master for a fresh pillar and generates the pillar data on the fly, unlike pillar.raw which returns the pillar data which is currently loaded into the minion.

CLI Example:

```
salt '*' pillar.items
```

`salt.modules.pillar.`**`raw`**(*key=None*)

Return the raw pillar data that is available in the module. This will show the pillar as it is loaded as the __pillar__ dict.

CLI Example:

```
salt '*' pillar.raw
```

With the optional key argument, you can select a subtree of the pillar raw data.:

```
salt '*' pillar.raw key='roles'
```

61.110 salt.modules.pip

Install Python packages with pip to either the system or a virtualenv

`salt.modules.pip.`**`freeze`**(*bin_env=None*, *user=None*, *runas=None*, *cwd=None*)

Return a list of installed packages either globally or in the specified virtualenv

bin_env path to pip bin or path to virtualenv. If doing an uninstall from the system python and want to use a specific pip bin (pip-2.7, pip-2.6, etc..) just specify the pip bin you want. If uninstalling from a virtualenv, just use the path to the virtualenv (/home/code/path/to/virtualenv/)

user The user under which to run pip

Note: The `runas` argument is deprecated as of 0.16.2. `user` should be used instead.

cwd Current working directory to run pip from

CLI Example:

```
salt '*' pip.freeze /home/code/path/to/virtualenv/
```

`salt.modules.pip.`**`install`**(*pkgs=None*, *requirements=None*, *env=None*, *bin_env=None*, *use_wheel=False*, *log=None*, *proxy=None*, *timeout=None*, *editable=None*, *find_links=None*, *index_url=None*, *extra_index_url=None*, *no_index=False*, *mirrors=None*, *build=None*, *target=None*, *download=None*, *download_cache=None*, *source=None*, *upgrade=False*, *force_reinstall=False*, *ignore_installed=False*, *exists_action=None*, *no_deps=False*, *no_install=False*, *no_download=False*, *global_options=None*, *install_options=None*, *user=None*, *runas=None*, *no_chown=False*, *cwd=None*, *activate=False*, *pre_releases=False*, *__env__=None*, *saltenv='base'*)

Install packages with pip

Install packages individually or from a pip requirements file. Install packages globally or to a virtualenv.

pkgs comma separated list of packages to install

requirements path to requirements

bin_env path to pip bin or path to virtualenv. If doing a system install, and want to use a specific pip bin (pip-2.7, pip-2.6, etc..) just specify the pip bin you want. If installing into a virtualenv, just use the path to the virtualenv (/home/code/path/to/virtualenv/)

env deprecated, use bin_env now

use_wheel Prefer wheel archives (requires pip>=1.4)

log Log file where a complete (maximum verbosity) record will be kept

proxy Specify a proxy in the form user:passwd@proxy.server:port. Note that the user:password@ is optional and required only if you are behind an authenticated proxy. If you provide user@proxy.server:port then you will be prompted for a password.

timeout Set the socket timeout (default 15 seconds)

editable install something editable (i.e. git+https://github.com/worldcompany/djangoembed.git#egg=djangoembed)

find_links URL to look for packages at

index_url Base URL of Python Package Index

extra_index_url Extra URLs of package indexes to use in addition to `index_url`

no_index Ignore package index

mirrors Specific mirror URL(s) to query (automatically adds –use-mirrors)

build Unpack packages into `build` dir

target Install packages into `target` dir

download Download packages into `download` instead of installing them

download_cache Cache downloaded packages in `download_cache` dir

source Check out `editable` packages into `source` dir

upgrade Upgrade all packages to the newest available version

force_reinstall When upgrading, reinstall all packages even if they are already up-to-date.

ignore_installed Ignore the installed packages (reinstalling instead)

exists_action Default action when a path already exists: (s)witch, (i)gnore, (w)wipe, (b)ackup

no_deps Ignore package dependencies

no_install Download and unpack all packages, but don't actually install them

no_download Don't download any packages, just install the ones already downloaded (completes an install run with –no-install)

install_options Extra arguments to be supplied to the setup.py install command (use like –install-option="–install- scripts=/usr/local/bin"). Use multiple –install- option options to pass multiple options to setup.py install. If you are using an option with a directory path, be sure to use absolute path.

global_options Extra global options to be supplied to the setup.py call before the install command.

user The user under which to run pip

Note: The `runas` argument is deprecated as of 0.16.2. `user` should be used instead.

---

no_chown When user is given, do not attempt to copy and chown a requirements file

cwd Current working directory to run pip from

activate Activates the virtual environment, if given via bin_env, before running install.

pre_releases Include pre-releases in the available versions

CLI Example:

```
salt '*' pip.install <package name>,<package2 name>
salt '*' pip.install requirements=/path/to/requirements.txt
salt '*' pip.install <package name> bin_env=/path/to/virtualenv
salt '*' pip.install <package name> bin_env=/path/to/pip_bin
```

Complicated CLI example:

```
salt '*' pip.install markdown,django editable=git+https://github.com/worldcompany/djangoembed.gi
```

`salt.modules.pip.`**`list_`**(*prefix=None*, *bin_env=None*, *user=None*, *runas=None*, *cwd=None*)

Filter list of installed apps from `freeze` and check to see if `prefix` exists in the list of packages installed.

CLI Example:

```
salt '*' pip.list salt
```

`salt.modules.pip.`**`uninstall`**(*pkgs=None*, *requirements=None*, *bin_env=None*, *log=None*, *proxy=None*, *timeout=None*, *user=None*, *runas=None*, *no_chown=False*, *cwd=None*, *__env__=None*, *saltenv='base'*)

Uninstall packages with pip

Uninstall packages individually or from a pip requirements file. Uninstall packages globally or from a virtualenv.

pkgs comma separated list of packages to install

requirements path to requirements

bin_env path to pip bin or path to virtualenv. If doing an uninstall from the system python and want to use a specific pip bin (pip-2.7, pip-2.6, etc..) just specify the pip bin you want. If uninstalling from a virtualenv, just use the path to the virtualenv (/home/code/path/to/virtualenv/)

log Log file where a complete (maximum verbosity) record will be kept

proxy Specify a proxy in the form user:passwd@proxy.server:port. Note that the user:password@ is optional and required only if you are behind an authenticated proxy. If you provide user@proxy.server:port then you will be prompted for a password.

timeout Set the socket timeout (default 15 seconds)

user The user under which to run pip

Note: The `runas` argument is deprecated as of 0.16.2. `user` should be used instead.

no_chown When user is given, do not attempt to copy and chown a requirements file

cwd Current working directory to run pip from

CLI Example:

```
salt '*' pip.uninstall <package name>,<package2 name>
salt '*' pip.uninstall requirements=/path/to/requirements.txt
salt '*' pip.uninstall <package name> bin_env=/path/to/virtualenv
salt '*' pip.uninstall <package name> bin_env=/path/to/pip_bin
```

`salt.modules.pip.`**`version`**(*bin_env=None*)

New in version 0.17.0.

Returns the version of pip. Use `bin_env` to specify the path to a virtualenv and get the version of pip in that virtualenv.

If unable to detect the pip version, returns `None`.

CLI Example:

```
salt '*' pip.version
```

61.111 salt.modules.pkg_resource

Resources needed by pkg providers

`salt.modules.pkg_resource.`**`add_pkg`**(*pkgs*, *name*, *version*)

Add a package to a dict of installed packages.

CLI Example:

```
salt '*' pkg_resource.add_pkg '{}' bind 9
```

`salt.modules.pkg_resource.`**`check_extra_requirements`**(*pkgname*, *pkgver*)

Check if the installed package already has the given requirements. This function will simply try to call "pkg.check_extra_requirements".

CLI Example:

```
salt '*' pkg_resource.check_extra_requirements <pkgname> <extra_requirements>
```

`salt.modules.pkg_resource.`**`pack_sources`**(*sources*)

Accepts list of dicts (or a string representing a list of dicts) and packs the key/value pairs into a single dict.

`'[{"foo": "salt://foo.rpm"}, {"bar": "salt://bar.rpm"}]'` would become `{"foo": "salt://foo.rpm", "bar": "salt://bar.rpm"}`

CLI Example:

```
salt '*' pkg_resource.pack_sources '[{"foo": "salt://foo.rpm"}, {"bar": "salt://bar.rpm"}]'
```

`salt.modules.pkg_resource.`**`parse_targets`**(*name=None*, *pkgs=None*, *sources=None*, *saltenv='base'*, ***kwargs*)

Parses the input to pkg.install and returns back the package(s) to be installed. Returns a list of packages, as well as a string noting whether the packages are to come from a repository or a binary package.

CLI Example:

```
salt '*' pkg_resource.parse_targets
```

`salt.modules.pkg_resource.`**`sort_pkglist`**(*pkgs*)

Accepts a dict obtained from pkg.list_pkgs() and sorts in place the list of versions for any packages that have multiple versions installed, so that two package lists can be compared to one another.

CLI Example:

```
salt '*' pkg_resource.sort_pkglist '["3.45", "2.13"]'
```

`salt.modules.pkg_resource.`**`stringify`**(*pkgs*)

Takes a dict of package name/version information and joins each list of installed versions into a string.

CLI Example:

```
salt '*' pkg_resource.stringify 'vim: 7.127'
```

`salt.modules.pkg_resource.`**`version`**(**names*, ***kwargs*)

Common interface for obtaining the version of installed packages.

CLI Example:

```
salt '*' pkg_resource.version vim
salt '*' pkg_resource.version foo bar baz
salt '*' pkg_resource.version 'python*'
```

`salt.modules.pkg_resource.`**`version_clean`**`(`*version*`)`

Clean the version string removing extra data. This function will simply try to call `pkg.version_clean`.

CLI Example:

```
salt '*' pkg_resource.version_clean <version_string>
```

61.112 salt.modules.pkgin

Package support for pkgin based systems, inspired from freebsdpkg module

`salt.modules.pkgin.`**`available_version`**`(`**names*, ***kwargs*`)`

Return the latest version of the named package available for upgrade or installation.

If the latest version of a given package is already installed, an empty string will be returned for that package.

CLI Example:

```
salt '*' pkg.latest_version <package name>
salt '*' pkg.latest_version <package1> <package2> ...
```

`salt.modules.pkgin.`**`file_dict`**`(`*package*`)`

List the files that belong to a package.

CLI Examples:

```
salt '*' pkg.file_list nginx
```

`salt.modules.pkgin.`**`file_list`**`(`*package*`)`

List the files that belong to a package.

CLI Examples:

```
salt '*' pkg.file_list nginx
```

`salt.modules.pkgin.`**`install`**`(`*name=None*, *refresh=False*, *fromrepo=None*, *pkgs=None*, *sources=None*, ***kwargs*`)`

Install the passed package

name The name of the package to be installed.

refresh Whether or not to refresh the package database before installing.

fromrepo Specify a package repository to install from.

Multiple Package Installation Options:

pkgs A list of packages to install from a software repository. Must be passed as a python list.

CLI Example:

```
salt '*' pkg.install pkgs='["foo","bar"]'
```

sources A list of packages to install. Must be passed as a list of dicts, with the keys being package names, and the values being the source URI or local path to the package.

CLI Example:

```
salt '*' pkg.install sources='[{"foo": "salt://foo.deb"},{"bar": "salt://bar.deb"}]'
```

Return a dict containing the new package names and versions:

```
{'<package>': {'old': '<old-version>',
               'new': '<new-version>'}}
```

CLI Example:

```
salt '*' pkg.install <package name>
```

`salt.modules.pkgin.`**`latest_version`**`(`**names*, ***kwargs*`)`

Return the latest version of the named package available for upgrade or installation.

If the latest version of a given package is already installed, an empty string will be returned for that package.

CLI Example:

```
salt '*' pkg.latest_version <package name>
salt '*' pkg.latest_version <package1> <package2> ...
```

`salt.modules.pkgin.`**`list_pkgs`**`(`*versions_as_list=False*, ***kwargs*`)`

List the packages currently installed as a dict:

```
{'<package_name>': '<version>'}
```

CLI Example:

```
salt '*' pkg.list_pkgs
```

`salt.modules.pkgin.`**`purge`**`(`*name=None*, *pkgs=None*, ***kwargs*`)`

Package purges are not supported, this function is identical to `remove()`.

name The name of the package to be deleted.

Multiple Package Options:

pkgs A list of packages to delete. Must be passed as a python list. The `name` parameter will be ignored if this option is passed.

New in version 0.16.0.

Returns a dict containing the changes.

CLI Example:

```
salt '*' pkg.purge <package name>
salt '*' pkg.purge <package1>,<package2>,<package3>
salt '*' pkg.purge pkgs='["foo", "bar"]'
```

`salt.modules.pkgin.`**`refresh_db`**`()`

Use pkg update to get latest pkg_summary

CLI Example:

```
salt '*' pkg.refresh_db
```

`salt.modules.pkgin.`**`rehash`**`()`

Recomputes internal hash table for the PATH variable. Use whenever a new command is created during the current session.

CLI Example:

```
salt '*' pkg.rehash
```

`salt.modules.pkgin.`**`remove`**`(`*name=None*, *pkgs=None*, ***kwargs*`)`

name The name of the package to be deleted.

Multiple Package Options:

pkgs A list of packages to delete. Must be passed as a python list. The `name` parameter will be ignored if this option is passed.

New in version 0.16.0.

Returns a list containing the removed packages.

CLI Example:

```
salt '*' pkg.remove <package name>
salt '*' pkg.remove <package1>,<package2>,<package3>
salt '*' pkg.remove pkgs='["foo", "bar"]'
```

`salt.modules.pkgin.`**`search`**`(`*pkg_name*`)`

Searches for an exact match using pkgin ^package$

CLI Example:

```
salt '*' pkg.search 'mysql-server'
```

`salt.modules.pkgin.`**`upgrade`**`()`

Run pkg upgrade, if pkgin used. Otherwise do nothing

Return a dict containing the new package names and versions:

```
{'<package>': {'old': '<old-version>',
               'new': '<new-version>'}}
```

CLI Example:

```
salt '*' pkg.upgrade
```

`salt.modules.pkgin.`**`version`**`(`**names*, ***kwargs*`)`

Returns a string representing the package version or an empty string if not installed. If more than one package name is specified, a dict of name/version pairs is returned.

CLI Example:

```
salt '*' pkg.version <package name>
salt '*' pkg.version <package1> <package2> <package3> ...
```

61.113 salt.modules.pkgng

Support for `pkgng`, the new package manager for FreeBSD

Warning: This module has been completely rewritten. Up to and includng version 0.17.x, it was available as the `pkgng` module, (`pkgng.install`, `pkgng.delete`, etc.), but moving forward this module will no longer be available as `pkgng`, as it will behave like a normal Salt `pkg` provider. The documentation below should not be considered to apply to this module in versions <= 0.17.x. If your minion is running a 0.17.x release or older, then the documentation for this module can be viewed using the `sys.doc` function:

```
salt bsdminion sys.doc pkgng
```

This module provides an interface to `pkg(8)`. It acts as the default package provider for FreeBSD 10 and newer. For FreeBSD hosts which have been upgraded to use pkgng, you will need to override the `pkg` provider by setting the `providers` parameter in your Minion config file, in order to use this module to manage packages, like so:

```
providers:
  pkg: pkgng
```

`salt.modules.pkgng.`**`audit`**(*jail=None*, *chroot=None*)

Audits installed packages against known vulnerabilities

CLI Example:

```
salt '*' pkg.audit
```

jail Audit packages within the specified jail

CLI Example:

```
salt '*' pkg.audit jail=<jail name or id>
```

chroot Audit packages within the specified chroot (ignored if `jail` is specified)

CLI Example:

```
salt '*' pkg.audit chroot=/path/to/chroot
```

`salt.modules.pkgng.`**`autoremove`**(*jail=None*, *chroot=None*, *dryrun=False*)

Delete packages which were automatically installed as dependencies and are not required anymore.

dryrun Dry-run mode. The list of changes to packages is always printed, but no changes are actually made.

CLI Example:

```
salt '*' pkg.autoremove
salt '*' pkg.autoremove jail=<jail name or id>
salt '*' pkg.autoremove dryrun=True
salt '*' pkg.autoremove jail=<jail name or id> dryrun=True
```

`salt.modules.pkgng.`**`backup`**(*file_name*, *jail=None*, *chroot=None*)

Export installed packages into yaml+mtree file

CLI Example:

```
salt '*' pkg.backup /tmp/pkg
```

jail Backup packages from the specified jail. Note that this will run the command within the jail, and so the path to the backup file will be relative to the root of the jail

CLI Example:

```
salt '*' pkg.backup /tmp/pkg jail=<jail name or id>
```

chroot Backup packages from the specified chroot (ignored if `jail` is specified). Note that this will run the command within the chroot, and so the path to the backup file will be relative to the root of the chroot.

CLI Example:

```
salt '*' pkg.backup /tmp/pkg chroot=/path/to/chroot
```

`salt.modules.pkgng.`**`check`**(*jail=None*, *chroot=None*, *depends=False*, *recompute=False*, *checksum=False*)

Sanity checks installed packages

jail Perform the sanity check in the specified jail

CLI Example:

```
salt '*' pkg.check jail=<jail name or id>
```

chroot Perform the sanity check in the specified chroot (ignored if `jail` is specified)

CLI Example:

```
salt '*' pkg.check chroot=/path/to/chroot
```

Of the below, at least one must be set to `True`.

depends Check for and install missing dependencies.

CLI Example:

```
salt '*' pkg.check recompute=True
```

recompute Recompute sizes and checksums of installed packages.

CLI Example:

```
salt '*' pkg.check depends=True
```

checksum Find invalid checksums for installed packages.

CLI Example:

```
salt '*' pkg.check checksum=True
```

`salt.modules.pkgng.`**`clean`**(*jail=None*, *chroot=None*)

Cleans the local cache of fetched remote packages

CLI Example:

```
salt '*' pkg.clean
salt '*' pkg.clean jail=<jail name or id>
salt '*' pkg.clean chroot=/path/to/chroot
```

`salt.modules.pkgng.`**`fetch`**(*name*, *jail=None*, *chroot=None*, *fetch_all=False*, *quiet=False*, *fromrepo=None*, *glob=True*, *regex=False*, *pcre=False*, *local=False*, *depends=False*)

Fetches remote packages

CLI Example:

```
salt '*' pkg.fetch <package name>
```

jail Fetch package in the specified jail

CLI Example:

```
salt '*' pkg.fetch <package name> jail=<jail name or id>
```

chroot Fetch package in the specified chroot (ignored if `jail` is specified)

CLI Example:

```
salt '*' pkg.fetch <package name> chroot=/path/to/chroot
```

fetch_all Fetch all packages.

CLI Example:

```
salt '*' pkg.fetch <package name> fetch_all=True
```

quiet Quiet mode. Show less output.

CLI Example:

```
salt '*' pkg.fetch <package name> quiet=True
```

fromrepo Fetches packages from the given repo if multiple repo support is enabled. See `pkg.conf(5)`.

CLI Example:

```
salt '*' pkg.fetch <package name> fromrepo=repo
```

glob Treat pkg_name as a shell glob pattern.

CLI Example:

```
salt '*' pkg.fetch <package name> glob=True
```

regex Treat pkg_name as a regular expression.

CLI Example:

```
salt '*' pkg.fetch <regular expression> regex=True
```

pcre Treat pkg_name is an extended regular expression.

CLI Example:

```
salt '*' pkg.fetch <extended regular expression> pcre=True
```

local Skip updating the repository catalogues with pkg-update(8). Use the local cache only.

CLI Example:

```
salt '*' pkg.fetch <package name> local=True
```

depends Fetch the package and its dependencies as well.

CLI Example:

```
salt '*' pkg.fetch <package name> depends=True
```

`salt.modules.pkgng.`**`install`**(*name=None*, *fromrepo=None*, *pkgs=None*, *sources=None*, *jail=None*, *chroot=None*, *orphan=False*, *force=False*, *glob=False*, *local=False*, *dryrun=False*, *quiet=False*, *require=False*, *regex=False*, *pcre=False*, ***kwargs*)

Install package(s) from a repository

name The name of the package to install

CLI Example:

```
salt '*' pkg.install <package name>
```

jail Install the package into the specified jail

chroot Install the paackage into the specified chroot (ignored if `jail` is specified)

orphan Mark the installed package as orphan. Will be automatically removed if no other packages depend on them. For more information please refer to `pkg-autoremove(8)`.

CLI Example:

```
salt '*' pkg.install <package name> orphan=True
```

force Force the reinstallation of the package if already installed.

CLI Example:

```
salt '*' pkg.install <package name> force=True
```

glob Treat the package names as shell glob patterns.

CLI Example:

```
salt '*' pkg.install <package name> glob=True
```

local Do not update the repository catalogues with `pkg-update(8)`. A value of `True` here is equivalent to using the `-U` flag with `pkg install`.

CLI Example:

```
salt '*' pkg.install <package name> local=True
```

dryrun Dru-run mode. The list of changes to packages is always printed, but no changes are actually made.

CLI Example:

```
salt '*' pkg.install <package name> dryrun=True
```

quiet Force quiet output, except when dryrun is used, where pkg install will always show packages to be installed, upgraded or deleted.

CLI Example:

```
salt '*' pkg.install <package name> quiet=True
```

require When used with force, reinstalls any packages that require the given package.

CLI Example:

```
salt '*' pkg.install <package name> require=True force=True
```

fromrepo In multi-repo mode, override the pkg.conf ordering and only attempt to download packages from the named repository.

CLI Example:

```
salt '*' pkg.install <package name> fromrepo=repo
```

regex Treat the package names as a regular expression

CLI Example:

```
salt '*' pkg.install <regular expression> regex=True
```

pcre Treat the package names as extended regular expressions.

CLI Example:

```
salt '*' pkg.install <extended regular expression> pcre=True
```

`salt.modules.pkgng.`**`latest_version`**(**names*, ***kwargs*)

Return the latest version of the named package available for upgrade or installation. If more than one package name is specified, a dict of name/version pairs is returned.

If the latest version of a given package is already installed, an empty string will be returned for that package.

CLI Example:

```
salt '*' pkg.latest_version <package name>
salt '*' pkg.latest_version <package name> jail=<jail name or id>
salt '*' pkg.latest_version <package name> chroot=/path/to/chroot
```

`salt.modules.pkgng.`**`list_pkgs`**(*versions_as_list=False*, *jail=None*, *chroot=None*, *with_origin=False*, ***kwargs*)

List the packages currently installed as a dict:

```
{'<package_name>': '<version>'}
```

jail List the packages in the specified jail

chroot List the pacakges in the specified chroot (ignored if `jail` is specified)

with_origin [False] Return a nested dictionary containing both the origin name and version for each installed package.

New in version 2014.1.0: (Hydrogen)

CLI Example:

```
salt '*' pkg.list_pkgs
salt '*' pkg.list_pkgs jail=<jail name or id>
salt '*' pkg.list_pkgs chroot=/path/to/chroot
```

`salt.modules.pkgng.`**`parse_config`**(*file_name='/usr/local/etc/pkg.conf'*)

Return dict of uncommented global variables.

CLI Example:

```
salt '*' pkg.parse_config
```

`NOTE`**:** not working properly right now

`salt.modules.pkgng.`**`refresh_db`**(*jail=None*, *chroot=None*, *force=False*)

Refresh PACKAGESITE contents

Note: This function can accessed using `pkg.update` in addition to `pkg.refresh_db`, to more closely match the CLI usage of `pkg(8)`.

CLI Example:

```
salt '*' pkg.refresh_db
```

jail Refresh the pkg database within the specified jail

chroot Refresh the pkg database within the specified chroot (ignored if `jail` is specified)

force Force a full download of the repository catalogue without regard to the respective ages of the local and remote copies of the catalogue.

CLI Example:

```
salt '*' pkg.refresh_db force=True
```

`salt.modules.pkgng.`**`remove`**(*name=None*, *pkgs=None*, *jail=None*, *chroot=None*, *all_installed=False*, *force=False*, *glob=False*, *dryrun=False*, *recurse=False*, *regex=False*, *pcre=False*, ***kwargs*)

Remove a package from the database and system

Note: This function can accessed using `pkg.delete` in addition to `pkg.remove`, to more closely match the CLI usage of `pkg(8)`.

name The package to remove

CLI Example:

```
salt '*' pkg.remove <package name>
```

jail Delete the package from the specified jail

chroot Delete the paackage grom the specified chroot (ignored if `jail` is specified)

all_installed Deletes all installed packages from the system and empties the database. USE WITH CAUTION!

CLI Example:

```
salt '*' pkg.remove all all_installed=True force=True
```

force Forces packages to be removed despite leaving unresolved dependencies.

CLI Example:

```
salt '*' pkg.remove <package name> force=True
```

glob Treat the package names as shell glob patterns.

CLI Example:

```
salt '*' pkg.remove <package name> glob=True
```

dryrun Dry run mode. The list of packages to delete is always printed, but no packages are actually deleted.

CLI Example:

```
salt '*' pkg.remove <package name> dryrun=True
```

recurse Delete all packages that require the listed package as well.

CLI Example:

```
salt '*' pkg.remove <package name> recurse=True
```

regex Treat the package names as regular expressions.

CLI Example:

```
salt '*' pkg.remove <regular expression> regex=True
```

pcre Treat the package names as extended regular expressions.

CLI Example:

```
salt '*' pkg.remove <extended regular expression> pcre=True
```

`salt.modules.pkgng.`**`restore`**(*file_name*, *jail=None*, *chroot=None*)

Reads archive created by pkg backup -d and recreates the database.

CLI Example:

```
salt '*' pkg.restore /tmp/pkg
```

jail Restore database to the specified jail. Note that this will run the command within the jail, and so the path to the file from which the pkg database will be restored is relative to the root of the jail.

CLI Example:

```
salt '*' pkg.restore /tmp/pkg jail=<jail name or id>
```

chroot Restore database to the specified chroot (ignored if `jail` is specified). Note that this will run the command within the chroot, and so the path to the file from which the pkg database will be restored is relative to the root of the chroot.

CLI Example:

```
salt '*' pkg.restore /tmp/pkg chroot=/path/to/chroot
```

`salt.modules.pkgng.`**`search`**(*name*, *jail=None*, *chroot=None*, *exact=False*, *glob=False*, *regex=False*, *pcre=False*, *comment=False*, *desc=False*, *full=False*, *depends=False*, *size=False*, *quiet=False*, *origin=False*, *prefix=False*)

Searches in remote package repositories

CLI Example:

```
salt '*' pkg.search pattern
```

jail Perform the search using the `pkg.conf(5)` from the specified jail

CLI Example:

```
salt '*' pkg.search pattern jail=<jail name or id>
```

chroot Perform the search using the `pkg.conf(5)` from the specified chroot (ignored if `jail` is specified)

CLI Example:

```
salt '*' pkg.search pattern chroot=/path/to/chroot
```

exact Treat pattern as exact pattern.

CLI Example:

```
salt '*' pkg.search pattern exact=True
```

glob Treat pattern as a shell glob pattern.

CLI Example:

```
salt '*' pkg.search pattern glob=True
```

regex Treat pattern as a regular expression.

CLI Example:

```
salt '*' pkg.search pattern regex=True
```

pcre Treat pattern as an extended regular expression.

CLI Example:

```
salt '*' pkg.search pattern pcre=True
```

comment Search for pattern in the package comment one-line description.

CLI Example:

```
salt '*' pkg.search pattern comment=True
```

desc Search for pattern in the package description.

CLI Example:

```
salt '*' pkg.search pattern desc=True
```

full Displays full information about the matching packages.

CLI Example:

```
salt '*' pkg.search pattern full=True
```

depends Displays the dependencies of pattern.

CLI Example:

```
salt '*' pkg.search pattern depends=True
```

size Displays the size of the package

CLI Example:

```
salt '*' pkg.search pattern size=True
```

quiet Be quiet. Prints only the requested information without displaying many hints.

CLI Example:

```
salt '*' pkg.search pattern quiet=True
```

origin Displays pattern origin.

CLI Example:

```
salt '*' pkg.search pattern origin=True
```

prefix Displays the installation prefix for each package matching pattern.

CLI Example:

```
salt '*' pkg.search pattern prefix=True
```

`salt.modules.pkgng.`**`stats`**(*local=False*, *remote=False*, *jail=None*, *chroot=None*)

Return pkgng stats.

CLI Example:

```
salt '*' pkg.stats
```

local Display stats only for the local package database.

CLI Example:

```
salt '*' pkg.stats local=True
```

remote Display stats only for the remote package database(s).

CLI Example:

```
salt '*' pkg.stats remote=True
```

jail Retrieve stats from the specified jail.

CLI Example:

```
salt '*' pkg.stats jail=<jail name or id>
salt '*' pkg.stats jail=<jail name or id> local=True
salt '*' pkg.stats jail=<jail name or id> remote=True
```

chroot Retrieve stats from the specified chroot (ignored if `jail` is specified).

CLI Example:

```
salt '*' pkg.stats chroot=/path/to/chroot
salt '*' pkg.stats chroot=/path/to/chroot local=True
salt '*' pkg.stats chroot=/path/to/chroot remote=True
```

`salt.modules.pkgng.`**`update_package_site`**(*new_url*)

Updates remote package repo URL, PACKAGESITE var to be exact.

Must use `http://`, `ftp://`, or `https://` protocol

CLI Example:

```
salt '*' pkg.update_package_site http://127.0.0.1/
```

`salt.modules.pkgng.`**`updating`**(*name*, *jail=None*, *chroot=None*, *filedate=None*, *filename=None*)

‘ Displays UPDATING entries of software packages

CLI Example:

```
salt '*' pkg.updating foo
```

jail Perform the action in the specified jail

CLI Example:

```
salt '*' pkg.updating foo jail=<jail name or id>
```

chroot Perform the action in the specified chroot (ignored if `jail` is specified)

CLI Example:

```
salt '*' pkg.updating foo chroot=/path/to/chroot
```

filedate Only entries newer than date are shown. Use a YYYYMMDD date format.

CLI Example:

```
salt '*' pkg.updating foo filedate=20130101
```

filename Defines an alternative location of the UPDATING file.

CLI Example:

```
salt '*' pkg.updating foo filename=/tmp/UPDATING
```

`salt.modules.pkgng.`**`upgrade`**(*jail=None*, *chroot=None*, *force=False*, *local=False*, *dryrun=False*)

Upgrade all packages (run a `pkg upgrade`)

CLI Example:

```
salt '*' pkg.upgrade
```

jail Audit packages within the specified jail

CLI Example:

```
salt '*' pkg.upgrade jail=<jail name or id>
```

chroot Audit packages within the specified chroot (ignored if `jail` is specified)

CLI Example:

```
salt '*' pkg.upgrade chroot=/path/to/chroot
```

Any of the below options can also be used with `jail` or `chroot`.

force Force reinstalling/upgrading the whole set of packages.

CLI Example:

```
salt '*' pkg.upgrade force=True
```

local Do not update the repository catalogues with `pkg-update(8)`. A value of `True` here is equivalent to using the `-U` flag with `pkg upgrade`.

CLI Example:

```
salt '*' pkg.upgrade local=True
```

dryrun Dry-run mode: show what packages have updates available, but do not perform any upgrades. Repository catalogues will be updated as usual unless the local option is also given.

CLI Example:

```
salt '*' pkg.upgrade dryrun=True
```

`salt.modules.pkgng.`**`version`**(**names*, ***kwargs*)

Returns a string representing the package version or an empty string if not installed. If more than one package name is specified, a dict of name/version pairs is returned.

Note: This function can accessed using `pkg.info` in addition to `pkg.version`, to more closely match the CLI usage of `pkg(8)`.

jail Get package version information for the specified jail

chroot Get package version information for the specified chroot (ignored if `jail` is specified)

with_origin [False] Return a nested dictionary containing both the origin name and version for each specified package.

New in version 2014.1.0: (Hydrogen)

CLI Example:

```
salt '*' pkg.version <package name>
salt '*' pkg.version <package name> jail=<jail name or id>
salt '*' pkg.version <package1> <package2> <package3> ...
```

`salt.modules.pkgng.`**`which`**(*path*, *jail=None*, *chroot=None*, *origin=False*, *quiet=False*)

Displays which package installed a specific file

CLI Example:

```
salt '*' pkg.which <file name>
```

jail Perform the check in the specified jail

CLI Example:

```
salt '*' pkg.which <file name> jail=<jail name or id>
```

chroot Perform the check in the specified chroot (ignored if `jail` is specified)

CLI Example:

```
salt '*' pkg.which <file name> chroot=/path/to/chroot
```

origin Shows the origin of the package instead of name-version.

CLI Example:

```
salt '*' pkg.which <file name> origin=True
```

quiet Quiet output.

CLI Example:

```
salt '*' pkg.which <file name> quiet=True
```

61.114 salt.modules.pkgutil

Pkgutil support for Solaris

`salt.modules.pkgutil.`**`install`**(*name=None*, *refresh=False*, *version=None*, *pkgs=None*, ***kwargs*)

Install packages using the pkgutil tool.

CLI Example:

```
salt '*' pkg.install <package_name>
salt '*' pkg.install SMClgcc346
```

Multiple Package Installation Options:

pkgs A list of packages to install from OpenCSW. Must be passed as a python list.

CLI Example:

```
salt '*' pkg.install pkgs='["foo", "bar"]'
salt '*' pkg.install pkgs='["foo", {"bar": "1.2.3"}]'
```

Returns a dict containing the new package names and versions:

```
{'<package>': {'old': '<old-version>',
               'new': '<new-version>'}}
```

`salt.modules.pkgutil.`**`latest_version`**`(`**names*, ***kwargs*`)`

Return the latest version of the named package available for upgrade or installation. If more than one package name is specified, a dict of name/version pairs is returned.

If the latest version of a given package is already installed, an empty string will be returned for that package.

CLI Example:

```
salt '*' pkgutil.latest_version CSWpython
salt '*' pkgutil.latest_version <package1> <package2> <package3> ...
```

`salt.modules.pkgutil.`**`list_pkgs`**`(`*versions_as_list=False*, ***kwargs*`)`

List the packages currently installed as a dict:

```
{'<package_name>': '<version>'}
```

CLI Example:

```
salt '*' pkg.list_pkgs
salt '*' pkg.list_pkgs versions_as_list=True
```

`salt.modules.pkgutil.`**`list_upgrades`**`(`*refresh=True*`)`

List all available package upgrades on this system

CLI Example:

```
salt '*' pkgutil.list_upgrades
```

`salt.modules.pkgutil.`**`purge`**`(`*name=None*, *pkgs=None*, ***kwargs*`)`

Package purges are not supported, this function is identical to `remove()`.

name The name of the package to be deleted.

Multiple Package Options:

pkgs A list of packages to delete. Must be passed as a python list. The `name` parameter will be ignored if this option is passed.

New in version 0.16.0.

Returns a dict containing the changes.

CLI Example:

```
salt '*' pkg.purge <package name>
salt '*' pkg.purge <package1>,<package2>,<package3>
salt '*' pkg.purge pkgs='["foo", "bar"]'
```

`salt.modules.pkgutil.`**`refresh_db`**`()`

Updates the pkgutil repo database (pkgutil -U)

CLI Example:

```
salt '*' pkgutil.refresh_db
```

`salt.modules.pkgutil.`**`remove`**(*name=None*, *pkgs=None*, ***kwargs*)

Remove a package and all its dependencies which are not in use by other packages.

name The name of the package to be deleted.

Multiple Package Options:

pkgs A list of packages to delete. Must be passed as a python list. The `name` parameter will be ignored if this option is passed.

New in version 0.16.0.

Returns a dict containing the changes.

CLI Example:

```
salt '*' pkg.remove <package name>
salt '*' pkg.remove <package1>,<package2>,<package3>
salt '*' pkg.remove pkgs='["foo", "bar"]'
```

`salt.modules.pkgutil.`**`upgrade`**(*refresh=True*, ***kwargs*)

Upgrade all of the packages to the latest available version.

Returns a dict containing the changes:

```
{'<package>': {'old': '<old-version>',
               'new': '<new-version>'}}
```

CLI Example:

```
salt '*' pkgutil.upgrade
```

`salt.modules.pkgutil.`**`upgrade_available`**(*name*)

Check if there is an upgrade available for a certain package

CLI Example:

```
salt '*' pkgutil.upgrade_available CSWpython
```

`salt.modules.pkgutil.`**`version`**(**names*, ***kwargs*)

Returns a version if the package is installed, else returns an empty string

CLI Example:

```
salt '*' pkgutil.version CSWpython
```

61.115 salt.modules.portage_config

Configure `portage(5)`

`salt.modules.portage_config.`**`append_to_package_conf`**(*conf*, *atom=’‘*, *flags=None*, *string=’‘*, *overwrite=False*)

Append a string or a list of flags for a given package or DEPEND atom to a given configuration file.

CLI Example:

```
salt '*' portage_config.append_to_package_conf use string="app-admin/salt ldap -libvirt"
salt '*' portage_config.append_to_package_conf use atom="> = app-admin/salt-0.14.1" flags="['lda
```

`salt.modules.portage_config.`**`append_use_flags`**(*atom*, *uses=None*, *overwrite=False*)

Append a list of use flags for a given package or DEPEND atom

CLI Example:

```
salt '*' portage_config.append_use_flags "app-admin/salt[ldap, -libvirt]"
salt '*' portage_config.append_use_flags ">=app-admin/salt-0.14.1" "['ldap', '-libvirt']"
```

`salt.modules.portage_config.`**`enforce_nice_config`**`()`

Enforce a nice tree structure for /etc/portage/package.* configuration files.

See also:

`salt.modules.ebuild.ex_mod_init()` for information on automatically running this when pkg is used.

CLI Example:

```
salt '*' portage_config.enforce_nice_config
```

`salt.modules.portage_config.`**`get_flags_from_package_conf`**`(`*conf*, *atom*`)`

Get flags for a given package or DEPEND atom. Warning: This only works if the configuration files tree is in the correct format (the one enforced by enforce_nice_config)

CLI Example:

```
salt '*' portage_config.get_flags_from_package_conf license salt
```

`salt.modules.portage_config.`**`get_missing_flags`**`(`*conf*, *atom*, *flags*`)`

Find out which of the given flags are currently not set. CLI Example:

```
salt '*' portage_config.get_missing_flags use salt "['ldap', '-libvirt', 'openssl']"
```

`salt.modules.portage_config.`**`has_flag`**`(`*conf*, *atom*, *flag*`)`

Verify if the given package or DEPEND atom has the given flag. Warning: This only works if the configuration files tree is in the correct format (the one enforced by enforce_nice_config)

CLI Example:

```
salt '*' portage_config.has_flag license salt Apache-2.0
```

`salt.modules.portage_config.`**`has_use`**`(`*atom*, *use*`)`

Verify if the given package or DEPEND atom has the given use flag. Warning: This only works if the configuration files tree is in the correct format (the one enforced by enforce_nice_config)

CLI Example:

```
salt '*' portage_config.has_use salt libvirt
```

`salt.modules.portage_config.`**`is_present`**`(`*conf*, *atom*`)`

Tell if a given package or DEPEND atom is present in the configuration files tree. Warning: This only works if the configuration files tree is in the correct format (the one enforced by enforce_nice_config)

CLI Example:

```
salt '*' portage_config.is_present unmask salt
```

61.116 salt.modules.postgres

Module to provide Postgres compatibility to salt.

configuration In order to connect to Postgres, certain configuration is required in /etc/salt/minion on the relevant minions. Some sample configs might look like:

```
postgres.host: 'localhost'
postgres.port: '5432'
postgres.user: 'postgres' -> db user
postgres.pass: ''
postgres.maintenance_db: 'postgres'
```

The default for the maintenance_db is 'postgres' and in most cases it can be left at the default setting. This data can also be passed into pillar. Options passed into opts will overwrite options passed into pillar

`salt.modules.postgres.`**`available_extensions`**(*user=None*, *host=None*, *port=None*, *maintenance_db=None*, *password=None*, *runas=None*)

List available postgresql extensions

CLI Example:

```
salt '*' postgres.available_extensions
```

`salt.modules.postgres.`**`create_extension`**(*name*, *if_not_exists=None*, *schema=None*, *ext_version=None*, *from_version=None*, *user=None*, *host=None*, *port=None*, *maintenance_db=None*, *password=None*, *runas=None*)

Install a postgresql extension

CLI Example:

```
salt '*' postgres.create_extension 'adminpack'
```

`salt.modules.postgres.`**`create_metadata`**(*name*, *ext_version=None*, *schema=None*, *user=None*, *host=None*, *port=None*, *maintenance_db=None*, *password=None*, *runas=None*)

Get lifecycle informations about an extension

CLI Example:

```
salt '*' postgres.create_metadata adminpack
```

`salt.modules.postgres.`**`db_alter`**(*name*, *user=None*, *host=None*, *port=None*, *maintenance_db=None*, *password=None*, *tablespace=None*, *owner=None*, *runas=None*)

Change tablesbase or/and owner of databse.

CLI Example:

```
salt '*' postgres.db_alter dbname owner=otheruser
```

`salt.modules.postgres.`**`db_create`**(*name*, *user=None*, *host=None*, *port=None*, *maintenance_db=None*, *password=None*, *tablespace=None*, *encoding=None*, *lc_collate=None*, *lc_ctype=None*, *owner=None*, *template=None*, *runas=None*)

Adds a databases to the Postgres server.

CLI Example:

```
salt '*' postgres.db_create 'dbname'

salt '*' postgres.db_create 'dbname' template=template_postgis
```

`salt.modules.postgres.`**`db_exists`**`(`*name*, *user=None*, *host=None*, *port=None*, *maintenance_db=None*, *password=None*, *runas=None*`)`

Checks if a database exists on the Postgres server.

CLI Example:

```
salt '*' postgres.db_exists 'dbname'
```

`salt.modules.postgres.`**`db_list`**`(`*user=None*, *host=None*, *port=None*, *maintenance_db=None*, *password=None*, *runas=None*`)`

Return dictionary with information about databases of a Postgres server.

CLI Example:

```
salt '*' postgres.db_list
```

`salt.modules.postgres.`**`db_remove`**`(`*name*, *user=None*, *host=None*, *port=None*, *maintenance_db=None*, *password=None*, *runas=None*`)`

Removes a databases from the Postgres server.

CLI Example:

```
salt '*' postgres.db_remove 'dbname'
```

`salt.modules.postgres.`**`drop_extension`**`(`*name*, *if_exists=None*, *restrict=None*, *cascade=None*, *user=None*, *host=None*, *port=None*, *maintenance_db=None*, *password=None*, *runas=None*`)`

Drop an installed postgresql extension

CLI Example:

```
salt '*' postgres.drop_extension 'adminpack'
```

`salt.modules.postgres.`**`get_available_extension`**`(`*name*, *user=None*, *host=None*, *port=None*, *maintenance_db=None*, *password=None*, *runas=None*`)`

Get info about an available postgresql extension

CLI Example:

```
salt '*' postgres.get_available_extension plpgsql
```

`salt.modules.postgres.`**`get_installed_extension`**`(`*name*, *user=None*, *host=None*, *port=None*, *maintenance_db=None*, *password=None*, *runas=None*`)`

Get info about an installed postgresql extension

CLI Example:

```
salt '*' postgres.get_installed_extension plpgsql
```

`salt.modules.postgres.`**`group_create`**`(`*groupname*, *user=None*, *host=None*, *port=None*, *maintenance_db=None*, *password=None*, *createdb=None*, *createuser=None*, *createroles=None*, *encrypted=None*, *login=None*, *inherit=None*, *superuser=None*, *replication=None*, *rolepassword=None*, *groups=None*, *runas=None*`)`

Creates a Postgres group. A group is postgres is similar to a user, but cannot login.

CLI Example:

```
salt '*' postgres.group_create 'groupname' user='user' \
        host='hostname' port='port' password='password' \
        rolepassword='rolepassword'
```

`salt.modules.postgres.`**`group_remove`**(*groupname*, *user=None*, *host=None*, *port=None*, *maintenance_db=None*, *password=None*, *runas=None*)

Removes a group from the Postgres server.

CLI Example:

```
salt '*' postgres.group_remove 'groupname'
```

`salt.modules.postgres.`**`group_update`**(*groupname*, *user=None*, *host=None*, *port=None*, *maintenance_db=None*, *password=None*, *createdb=None*, *createroles=None*, *createuser=None*, *encrypted=None*, *inherit=None*, *login=None*, *superuser=None*, *replication=None*, *rolepassword=None*, *groups=None*, *runas=None*)

Updated a postgres group

CLI Examples:

```
salt '*' postgres.group_update 'username' user='user' \
        host='hostname' port='port' password='password' \
        rolepassword='rolepassword'
```

`salt.modules.postgres.`**`installed_extensions`**(*user=None*, *host=None*, *port=None*, *maintenance_db=None*, *password=None*, *runas=None*)

List installed postgresql extensions

CLI Example:

```
salt '*' postgres.installed_extensions
```

`salt.modules.postgres.`**`is_available_extension`**(*name*, *user=None*, *host=None*, *port=None*, *maintenance_db=None*, *password=None*, *runas=None*)

Test if a specific extension is installed

CLI Example:

```
salt '*' postgres.is_installed_extension
```

`salt.modules.postgres.`**`is_installed_extension`**(*name*, *user=None*, *host=None*, *port=None*, *maintenance_db=None*, *password=None*, *runas=None*)

Test if a specific extension is installed

CLI Example:

```
salt '*' postgres.is_installed_extension
```

`salt.modules.postgres.`**`owner_to`**(*dbname*, *ownername*, *user=None*, *host=None*, *port=None*, *password=None*, *runas=None*)

Set the owner of all schemas, functions, tables, views and sequences to the given username.

CLI Example:

```
salt '*' postgres.owner_to 'dbname' 'username'
```

`salt.modules.postgres.`**`psql_query`**(*query*, *user=None*, *host=None*, *port=None*, *maintenance_db=None*, *password=None*, *runas=None*)

Run an SQL-Query and return the results as a list. This command only supports SELECT statements.

CLI Example:

```
salt '*' postgres.psql_query 'select * from pg_stat_activity'
```

`salt.modules.postgres.`**`role_get`**(*name*, *user=None*, *host=None*, *port=None*, *maintenance_db=None*, *password=None*, *runas=None*, *return_password=False*)

Return a dict with information about users of a Postgres server.

Set return_password to True to get password hash in the result.

CLI Example:

```
salt '*' postgres.role_get postgres
```

`salt.modules.postgres.`**`user_create`**(*username*, *user=None*, *host=None*, *port=None*, *maintenance_db=None*, *password=None*, *createdb=None*, *createuser=None*, *createroles=None*, *inherit=None*, *login=None*, *encrypted=None*, *superuser=None*, *replication=None*, *rolepassword=None*, *groups=None*, *runas=None*)

Creates a Postgres user.

CLI Examples:

```
salt '*' postgres.user_create 'username' user='user' \
        host='hostname' port='port' password='password' \
        rolepassword='rolepassword'
```

`salt.modules.postgres.`**`user_exists`**(*name*, *user=None*, *host=None*, *port=None*, *maintenance_db=None*, *password=None*, *runas=None*)

Checks if a user exists on the Postgres server.

CLI Example:

```
salt '*' postgres.user_exists 'username'
```

`salt.modules.postgres.`**`user_list`**(*user=None*, *host=None*, *port=None*, *maintenance_db=None*, *password=None*, *runas=None*, *return_password=False*)

Return a dict with information about users of a Postgres server.

Set return_password to True to get password hash in the result.

CLI Example:

```
salt '*' postgres.user_list
```

`salt.modules.postgres.`**`user_remove`**(*username*, *user=None*, *host=None*, *port=None*, *maintenance_db=None*, *password=None*, *runas=None*)

Removes a user from the Postgres server.

CLI Example:

```
salt '*' postgres.user_remove 'username'
```

`salt.modules.postgres.`**`user_update`**(*username*, *user=None*, *host=None*, *port=None*, *maintenance_db=None*, *password=None*, *createdb=None*, *createuser=None*, *createroles=None*, *encrypted=None*, *superuser=None*, *inherit=None*, *login=None*, *replication=None*, *rolepassword=None*, *groups=None*, *runas=None*)

Creates a Postgres user.

CLI Examples:

```
salt '*' postgres.user_create 'username' user='user' \
        host='hostname' port='port' password='password' \
        rolepassword='rolepassword'
```

`salt.modules.postgres.`**`version`**(*user=None*, *host=None*, *port=None*, *maintenance_db=None*, *password=None*, *runas=None*)

Return the version of a Postgres server.

CLI Example:

```
salt '*' postgres.version
```

61.117 salt.modules.poudriere

Support for poudriere

`salt.modules.poudriere.`**`bulk_build`**(*jail*, *pkg_file*, *keep=False*)

Run bulk build on poudriere server.

Return number of pkg builds, failures, and errors, on error dump to CLI

CLI Example:

```
salt -N buildbox_group poudriere.bulk_build 90amd64 /root/pkg_list
```

`salt.modules.poudriere.`**`create_jail`**(*name*, *arch*, *version='9.0-RELEASE'*)

Creates a new poudriere jail if one does not exist

NOTE creating a new jail will take some time the master is not hanging

CLI Example:

```
salt '*' poudriere.create_jail 90amd64 amd64
```

`salt.modules.poudriere.`**`create_ports_tree`**()

Not working need to run portfetch non interactive

`salt.modules.poudriere.`**`delete_jail`**(*name*)

Deletes poudriere jail with *name*

CLI Example:

```
salt '*' poudriere.delete_jail 90amd64
```

`salt.modules.poudriere.`**`is_jail`**(*name*)

Return True if jail exists False if not

CLI Example:

```
salt '*' poudriere.is_jail <jail name>
```

`salt.modules.poudriere.`**`list_jails`**`()`

Return a list of current jails managed by poudriere

CLI Example:

```
salt '*' poudriere.list_jails
```

`salt.modules.poudriere.`**`list_ports`**`()`

Return a list of current port trees managed by poudriere

CLI Example:

```
salt '*' poudriere.list_ports
```

`salt.modules.poudriere.`**`make_pkgng_aware`**`(`*jname*`)`

Make jail `jname` pkgng aware

CLI Example:

```
salt '*' poudriere.make_pkgng_aware <jail name>
```

`salt.modules.poudriere.`**`parse_config`**`(`*config_file=None*`)`

Returns a dict of poudriere main configuration definitions

CLI Example:

```
salt '*' poudriere.parse_config
```

`salt.modules.poudriere.`**`version`**`()`

Return poudriere version

CLI Example:

```
salt '*' poudriere.version
```

61.118 salt.modules.powerpath

powerpath support.

Assumes RedHat

`salt.modules.powerpath.`**`add_license`**`(`*key*`)`

Add a license

`salt.modules.powerpath.`**`has_powerpath`**`()`

`salt.modules.powerpath.`**`list_licenses`**`()`

returns a list of applied powerpath license keys

`salt.modules.powerpath.`**`remove_license`**`(`*key*`)`

Remove a license

61.119 salt.modules.ps

A salt interface to psutil, a system and process library. See http://code.google.com/p/psutil.

depends

- psutil Python module, version 0.3.0 or later

- python-utmp package (optional)

`salt.modules.ps.`**`boot_time`**`(`*time_format=None*`)`

Return the boot time in number of seconds since the epoch began.

CLI Example:

time_format Optionally specify a strftime format string. Use `time_format='%c'` to get a nicely-formatted locale specific date and time (i.e. `Fri May 2 19:08:32 2014`).

New in version 2014.1.4.

```
salt '*' ps.boot_time
```

`salt.modules.ps.`**`cached_physical_memory`**`()`

Return the amount cached memory.

CLI Example:

```
salt '*' ps.cached_physical_memory
```

`salt.modules.ps.`**`cpu_percent`**`(`*interval=0.1*, *per_cpu=False*`)`

Return the percent of time the CPU is busy.

interval the number of seconds to sample CPU usage over

per_cpu if True return an array of CPU percent busy for each CPU, otherwise aggregate all percents into one number

CLI Example:

```
salt '*' ps.cpu_percent
```

`salt.modules.ps.`**`cpu_times`**`(`*per_cpu=False*`)`

Return the percent of time the CPU spends in each state, e.g. user, system, idle, nice, iowait, irq, softirq.

per_cpu if True return an array of percents for each CPU, otherwise aggregate all percents into one number

CLI Example:

```
salt '*' ps.cpu_times
```

`salt.modules.ps.`**`disk_io_counters`**`()`

Return disk I/O statisitics.

CLI Example:

```
salt '*' ps.disk_io_counters
```

`salt.modules.ps.`**`disk_partition_usage`**`(`*all=False*`)`

Return a list of disk partitions plus the mount point, filesystem and usage statistics.

CLI Example:

```
salt '*' ps.disk_partition_usage
```

`salt.modules.ps.`**`disk_partitions`**`(`*all=False*`)`

Return a list of disk partitions and their device, mount point, and filesystem type.

all if set to False, only return local, physical partitions (hard disk, USB, CD/DVD partitions). If True, return all filesystems.

CLI Example:

```
salt '*' ps.disk_partitions
```

`salt.modules.ps.`**`disk_usage`**(*path*)

Given a path, return a dict listing the total available space as well as the free space, and used space.

CLI Example:

```
salt '*' ps.disk_usage /home
```

`salt.modules.ps.`**`get_pid_list`**()

Return a list of process ids (PIDs) for all running processes.

CLI Example:

```
salt '*' ps.get_pid_list
```

`salt.modules.ps.`**`get_users`**()

Return logged-in users.

CLI Example:

```
salt '*' ps.get_users
```

`salt.modules.ps.`**`kill_pid`**(*pid*, *signal=15*)

Kill a proccess by PID.

```
salt 'minion' ps.kill_pid pid [signal=signal_number]
```

pid PID of process to kill.

signal Signal to send to the process. See manpage entry for kill for possible values. Default: 15 (SIGTERM).

Example:

Send SIGKILL to process with PID 2000:

```
salt 'minion' ps.kill_pid 2000 signal=9
```

`salt.modules.ps.`**`network_io_counters`**()

Return network I/O statisitics.

CLI Example:

```
salt '*' ps.network_io_counters
```

`salt.modules.ps.`**`num_cpus`**()

Return the number of CPUs.

CLI Example:

```
salt '*' ps.num_cpus
```

`salt.modules.ps.`**`pgrep`**(*pattern*, *user=None*, *full=False*)

Return the pids for processes matching a pattern.

If full is true, the full command line is searched for a match, otherwise only the name of the command is searched.

```
salt '*' ps.pgrep pattern [user=username] [full=(true|false)]
```

pattern Pattern to search for in the process list.

user Limit matches to the given username. Default: All users.

full A boolean value indicating whether only the name of the command or the full command line should be matched against the pattern.

Examples:

Find all httpd processes on all 'www' minions:

```
salt 'www.*' ps.pgrep httpd
```

Find all bash processes owned by user 'tom':

```
salt '*' ps.pgrep bash user=tom
```

`salt.modules.ps.`**`physical_memory_buffers`**`()`

Return the amount of physical memory buffers.

CLI Example:

```
salt '*' ps.physical_memory_buffers
```

`salt.modules.ps.`**`physical_memory_usage`**`()`

Return a dict that describes free and available physical memory.

CLI Examples:

```
salt '*' ps.physical_memory_usage
```

`salt.modules.ps.`**`pkill`**(*pattern*, *user=None*, *signal=15*, *full=False*)

Kill processes matching a pattern.

```
salt '*' ps.pkill pattern [user=username] [signal=signal_number] \
        [full=(true|false)]
```

pattern Pattern to search for in the process list.

user Limit matches to the given username. Default: All users.

signal Signal to send to the process(es). See manpage entry for kill for possible values. Default: 15 (SIGTERM).

full A boolean value indicating whether only the name of the command or the full command line should be matched against the pattern.

Examples:

Send SIGHUP to all httpd processes on all 'www' minions:

```
salt 'www.*' ps.pkill httpd signal=1
```

Send SIGKILL to all bash processes owned by user 'tom':

```
salt '*' ps.pkill bash signal=9 user=tom
```

`salt.modules.ps.`**`top`**(*num_processes=5*, *interval=3*)

Return a list of top CPU consuming processes during the interval. num_processes = return the top N CPU consuming processes interval = the number of seconds to sample CPU usage over

CLI Examples:

```
salt '*' ps.top
```

```
salt '*' ps.top 5 10
```

`salt.modules.ps.`**`total_physical_memory`**`()`

Return the total number of bytes of physical memory.

CLI Example:

```
salt '*' ps.total_physical_memory
```

`salt.modules.ps.`**`virtual_memory_usage`**`()`

Return a dict that describes free and available memory, both physical and virtual.

CLI Example:

```
salt '*' ps.virtual_memory_usage
```

61.120 salt.modules.publish

Publish a command from a minion to a target

`salt.modules.publish.`**`full_data`**`(`*tgt*, *fun*, *arg=None*, *expr_form='glob'*, *returner='`*, *timeout=5*`)`

Return the full data about the publication, this is invoked in the same way as the publish function

CLI Example:

```
salt system.example.com publish.full_data '*' cmd.run 'ls -la /tmp'
```

Attention

If you need to pass a value to a function argument and that value contains an equal sign, you **must** include the argument name. For example:

```
salt '*' publish.full_data test.kwarg arg='cheese=spam'
```

`salt.modules.publish.`**`publish`**`(`*tgt*, *fun*, *arg=None*, *expr_form='glob'*, *returner='`*, *timeout=5*`)`

Publish a command from the minion out to other minions.

Publications need to be enabled on the Salt master and the minion needs to have permission to publish the command. The Salt master will also prevent a recursive publication loop, this means that a minion cannot command another minion to command another minion as that would create an infinite command loop.

The expr_form argument is used to pass a target other than a glob into the execution, the available options are:

- glob
- pcre
- grain
- grain_pcre
- pillar
- ipcidr
- range
- compound

The arguments sent to the minion publish function are separated with commas. This means that for a minion executing a command with multiple args it will look like this:

```
salt system.example.com publish.publish '*' user.add 'foo,1020,1020'
salt system.example.com publish.publish 'os:Fedora' network.interfaces '' grain
```

CLI Example:

```
salt system.example.com publish.publish '*' cmd.run 'ls -la /tmp'
```

Attention

If you need to pass a value to a function argument and that value contains an equal sign, you **must** include the argument name. For example:

```
salt '*' publish.publish test.kwarg arg='cheese=spam'
```

`salt.modules.publish.`**`runner`**`(`*fun*, *arg=None*`)`

Execute a runner on the master and return the data from the runner function

CLI Example:

```
salt publish.runner manage.down
```

61.121 salt.modules.puppet

Execute puppet routines

`salt.modules.puppet.`**`fact`**`(`*name*, *puppet=False*`)`

Run facter for a specific fact

CLI Example:

```
salt '*' puppet.fact kernel
```

`salt.modules.puppet.`**`facts`**`(`*puppet=False*`)`

Run facter and return the results

CLI Example:

```
salt '*' puppet.facts
```

`salt.modules.puppet.`**`noop`**`(`**args*, ***kwargs*`)`

Execute a puppet noop run and return a dict with the stderr, stdout, return code, etc. Usage is the same as for puppet.run.

CLI Example:

```
salt '*' puppet.noop
salt '*' puppet.noop tags=basefiles::edit,apache::server
salt '*' puppet.noop debug
salt '*' puppet.noop apply /a/b/manifest.pp modulepath=/a/b/modules tags=basefiles::edit,apache:
```

`salt.modules.puppet.`**`run`**`(`**args*, ***kwargs*`)`

Execute a puppet run and return a dict with the stderr, stdout, return code, etc. The first positional argument given is checked as a subcommand. Following positional arguments should be ordered with arguments required by the subcommand first, followed by non-keyvalue pair options. Tags are specified by a tag keyword and comma separated list of values. – http://projects.puppetlabs.com/projects/1/wiki/Using_Tags

CLI Examples:

```
salt '*' puppet.run
salt '*' puppet.run tags=basefiles::edit,apache::server
salt '*' puppet.run agent onetime no-daemonize no-usecacheonfailure no-splay ignorecache
salt '*' puppet.run debug
salt '*' puppet.run apply /a/b/manifest.pp modulepath=/a/b/modules tags=basefiles::edit,apache::
```

61.122 salt.modules.pw_group

Manage groups on FreeBSD

`salt.modules.pw_group.`**`add`**(*name*, *gid=None*, ***kwargs*)

Add the specified group

CLI Example:

```
salt '*' group.add foo 3456
```

`salt.modules.pw_group.`**`chgid`**(*name*, *gid*)

Change the gid for a named group

CLI Example:

```
salt '*' group.chgid foo 4376
```

`salt.modules.pw_group.`**`delete`**(*name*)

Remove the named group

CLI Example:

```
salt '*' group.delete foo
```

`salt.modules.pw_group.`**`getent`**(*refresh=False*)

Return info on all groups

CLI Example:

```
salt '*' group.getent
```

`salt.modules.pw_group.`**`info`**(*name*)

Return information about a group

CLI Example:

```
salt '*' group.info foo
```

61.123 salt.modules.pw_user

Manage users with the useradd command

`salt.modules.pw_user.`**`add`**(*name*, *uid=None*, *gid=None*, *groups=None*, *home=None*, *shell=None*, *unique=True*, *fullname=’‘*, *roomnumber=’‘*, *workphone=’‘*, *homephone=’‘*, *createhome=True*, ***kwargs*)

Add a user to the minion

CLI Example:

```
salt '*' user.add name <uid> <gid> <groups> <home> <shell>
```

salt.modules.pw_user.**chfullname**(*name*, *fullname*)
Change the user's Full Name

CLI Example:

```
salt '*' user.chfullname foo "Foo Bar"
```

salt.modules.pw_user.**chgid**(*name*, *gid*)
Change the default group of the user

CLI Example:

```
salt '*' user.chgid foo 4376
```

salt.modules.pw_user.**chgroups**(*name*, *groups*, *append=False*)
Change the groups this user belongs to, add append to append the specified groups

CLI Example:

```
salt '*' user.chgroups foo wheel,root True
```

salt.modules.pw_user.**chhome**(*name*, *home*, *persist=False*)
Change the home directory of the user, pass true for persist to copy files to the new home dir

CLI Example:

```
salt '*' user.chhome foo /home/users/foo True
```

salt.modules.pw_user.**chhomephone**(*name*, *homephone*)
Change the user's Home Phone

CLI Example:

```
salt '*' user.chhomephone foo "7735551234"
```

salt.modules.pw_user.**chroomnumber**(*name*, *roomnumber*)
Change the user's Room Number

CLI Example:

```
salt '*' user.chroomnumber foo 123
```

salt.modules.pw_user.**chshell**(*name*, *shell*)
Change the default shell of the user

CLI Example:

```
salt '*' user.chshell foo /bin/zsh
```

salt.modules.pw_user.**chuid**(*name*, *uid*)
Change the uid for a named user

CLI Example:

```
salt '*' user.chuid foo 4376
```

salt.modules.pw_user.**chworkphone**(*name*, *workphone*)
Change the user's Work Phone

CLI Example:

```
salt '*' user.chworkphone foo "7735550123"
```

`salt.modules.pw_user.`**`delete`**(*name*, *remove=False*, *force=False*)

Remove a user from the minion

CLI Example:

```
salt '*' user.delete name remove=True force=True
```

`salt.modules.pw_user.`**`getent`**(*refresh=False*)

Return the list of all info for all users

CLI Example:

```
salt '*' user.getent
```

`salt.modules.pw_user.`**`info`**(*name*)

Return user information

CLI Example:

```
salt '*' user.info root
```

`salt.modules.pw_user.`**`list_groups`**(*name*)

Return a list of groups the named user belongs to

CLI Example:

```
salt '*' user.list_groups foo
```

61.124 salt.modules.qemu_img

61.124.1 Qemu-img Command Wrapper

The qemu img command is wrapped for specific functions

depends qemu-img

`salt.modules.qemu_img.`**`make_image`**(*location*, *size*, *fmt*)

Create a blank virtual machine image file of the specified size in megabytes. The image can be created in any format supported by qemu

CLI Example:

```
salt '*' qemu_img.make_image /tmp/image.qcow 2048 qcow2
salt '*' qemu_img.make_image /tmp/image.raw 10240 raw
```

61.125 salt.modules.qemu_nbd

61.125.1 Qemu Command Wrapper

The qemu system comes with powerful tools, such as qemu-img and qemu-nbd which are used here to build up kvm images.

`salt.modules.qemu_nbd.`**`clear`**`(`*mnt*`)`

Pass in the mnt dict returned from nbd_mount to unmount and disconnect the image from nbd. If all of the partitions are unmounted return an empty dict, otherwise return a dict containing the still mounted partitions

CLI Example:

```
salt '*' qemu_nbd.clear '{"/mnt/foo": "/dev/nbd0p1"}'
```

`salt.modules.qemu_nbd.`**`connect`**`(`*image*`)`

Activate nbd for an image file.

CLI Example:

```
salt '*' qemu_nbd.connect /tmp/image.raw
```

`salt.modules.qemu_nbd.`**`init`**`(`*image*`)`

Mount the named image via qemu-nbd and return the mounted roots

CLI Example:

```
salt '*' qemu_nbd.init /srv/image.qcow2
```

`salt.modules.qemu_nbd.`**`mount`**`(`*nbd*`)`

Pass in the nbd connection device location, mount all partitions and return a dict of mount points

CLI Example:

```
salt '*' qemu_nbd.mount /dev/nbd0
```

61.126 salt.modules.quota

Module for managing quotas on POSIX-like systems.

`salt.modules.quota.`**`get_mode`**`(`*device*`)`

Report whether the quota system for this device is on or off

CLI Example:

```
salt '*' quota.get_mode
```

`salt.modules.quota.`**`off`**`(`*device*`)`

Turns off the quota system

CLI Example:

```
salt '*' quota.off
```

`salt.modules.quota.`**`on`**`(`*device*`)`

Turns on the quota system

CLI Example:

```
salt '*' quota.on
```

`salt.modules.quota.`**`report`**`(`*mount*`)`

Report on quotas for a specific volume

CLI Example:

```
salt '*' quota.report /media/data
```

`salt.modules.quota.`**`set_`**(*device*, ***kwargs*)

Calls out to setquota, for a specific user or group

CLI Example:

```
salt '*' quota.set /media/data user=larry block-soft-limit=1048576
salt '*' quota.set /media/data group=painters file-hard-limit=1000
```

`salt.modules.quota.`**`stats`**()

Runs the quotastats command, and returns the parsed output

CLI Example:

```
salt '*' quota.stats
```

`salt.modules.quota.`**`warn`**()

Runs the warnquota command, to send warning emails to users who are over their quota limit.

CLI Example:

```
salt '*' quota.warn
```

61.127 salt.modules.rabbitmq

Module to provide RabbitMQ compatibility to Salt. Todo: A lot, need to add cluster support, logging, and minion configuration data.

`salt.modules.rabbitmq.`**`add_user`**(*name*, *password=None*, *runas=None*)

Add a rabbitMQ user via rabbitmqctl user_add <user> <password>

CLI Example:

```
salt '*' rabbitmq.add_user rabbit_user password
```

`salt.modules.rabbitmq.`**`add_vhost`**(*vhost*, *runas=None*)

Adds a vhost via rabbitmqctl add_vhost.

CLI Example:

```
salt '*' rabbitmq add_vhost '<vhost_name>'
```

`salt.modules.rabbitmq.`**`change_password`**(*name*, *password*, *runas=None*)

Changes a user's password.

CLI Example:

```
salt '*' rabbitmq.change_password rabbit_user password
```

`salt.modules.rabbitmq.`**`clear_password`**(*name*, *runas=None*)

Removes a user's password.

CLI Example:

```
salt '*' rabbitmq.clear_password rabbit_user
```

`salt.modules.rabbitmq.`**`cluster_status`**(*user=None*)

return rabbitmq cluster_status

CLI Example:

```
salt '*' rabbitmq.cluster_status
```

`salt.modules.rabbitmq.`**`delete_policy`**(*vhost*, *name*, *runas=None*)

Delete a policy based on rabbitmqctl clear_policy.

Reference: http://www.rabbitmq.com/ha.html

CLI Example:

```
salt '*' rabbitmq.delete_policy / HA'
```

`salt.modules.rabbitmq.`**`delete_user`**(*name*, *runas=None*)

Deletes a user via rabbitmqctl delete_user.

CLI Example:

```
salt '*' rabbitmq.delete_user rabbit_user
```

`salt.modules.rabbitmq.`**`delete_vhost`**(*vhost*, *runas=None*)

Deletes a vhost rabbitmqctl delete_vhost.

CLI Example:

```
salt '*' rabbitmq.delete_vhost '<vhost_name>'
```

`salt.modules.rabbitmq.`**`disable_plugin`**(*name*, *runas=None*)

Disable a RabbitMQ plugin via the rabbitmq-plugins command.

CLI Example:

```
salt '*' rabbitmq.disable_plugin foo
```

`salt.modules.rabbitmq.`**`enable_plugin`**(*name*, *runas=None*)

Enable a RabbitMQ plugin via the rabbitmq-plugins command.

CLI Example:

```
salt '*' rabbitmq.enable_plugin foo
```

`salt.modules.rabbitmq.`**`force_reset`**(*runas=None*)

Forcefully Return a RabbitMQ node to its virgin state

CLI Example:

```
salt '*' rabbitmq.force_reset
```

`salt.modules.rabbitmq.`**`join_cluster`**(*host*, *user='rabbit'*, *runas=None*)

Join a rabbit cluster

CLI Example:

```
salt '*' rabbitmq.join_cluster 'rabbit' 'rabbit.example.com'
```

`salt.modules.rabbitmq.`**`list_permissions`**(*vhost*, *runas=None*)

Lists permissions for vhost via rabbitmqctl list_permissions

CLI Example:

```
salt '*' rabbitmq.list_permissions '/myvhost'
```

`salt.modules.rabbitmq.`**`list_policies`**(*runas=None*)

Return a dictionary of policies nested by vhost and name based on the data returned from rabbitmqctl list_policies.

Reference: http://www.rabbitmq.com/ha.html

CLI Example:

```
salt '*' rabbitmq.list_policies'
```

salt.modules.rabbitmq.**list_queues**(**kwargs*)

Returns queue details of the / virtual host

CLI Example:

```
salt '*' rabbitmq.list_queues messages consumers
```

salt.modules.rabbitmq.**list_queues_vhost**(*vhost*, **kwargs*)

Returns queue details of specified virtual host. This command will consider first parameter as the vhost name and rest will be treated as queueinfoitem. For getting details on vhost `/`, use `list_queues` instead).

CLI Example:

```
salt '*' rabbitmq.list_queues messages consumers
```

salt.modules.rabbitmq.**list_user_permissions**(*name*, *user=None*)

List permissions for a user via rabbitmqctl list_user_permissions

CLI Example:

```
salt '*' rabbitmq.list_user_permissions 'user'.
```

salt.modules.rabbitmq.**list_users**(*runas=None*)

Return a list of users based off of rabbitmqctl user_list.

CLI Example:

```
salt '*' rabbitmq.list_users
```

salt.modules.rabbitmq.**list_vhosts**(*runas=None*)

Return a list of vhost based on rabbitmqctl list_vhosts.

CLI Example:

```
salt '*' rabbitmq.list_vhosts
```

salt.modules.rabbitmq.**plugin_is_enabled**(*name*, *runas=None*)

Return whether the plugin is enabled.

CLI Example:

```
salt '*' rabbitmq.plugin_is_enabled foo
```

salt.modules.rabbitmq.**policy_exists**(*vhost*, *name*, *runas=None*)

Return whether the policy exists based on rabbitmqctl list_policies.

Reference: http://www.rabbitmq.com/ha.html

CLI Example:

```
salt '*' rabbitmq.policy_exists / HA
```

salt.modules.rabbitmq.**reset**(*runas=None*)

Return a RabbitMQ node to its virgin state

CLI Example:

```
salt '*' rabbitmq.reset
```

salt.modules.rabbitmq.**set_permissions**(*vhost*, *user*, *conf='.*'*, *write='.*'*, *read='.*'*, *runas=None*)

Sets permissions for vhost via rabbitmqctl set_permissions

CLI Example:

```
salt '*' rabbitmq.set_permissions 'myvhost' 'myuser'
```

salt.modules.rabbitmq.**set_policy**(*vhost*, *name*, *pattern*, *definition*, *priority=None*, *runas=None*)

Set a policy based on rabbitmqctl set_policy.

Reference: http://www.rabbitmq.com/ha.html

CLI Example:

```
salt '*' rabbitmq.set_policy / HA '.*' '{"ha-mode": "all"}'
```

salt.modules.rabbitmq.**set_user_tags**(*name*, *tags*, *runas=None*)

Add user tags via rabbitctl set_user_tags

CLI Example:

```
salt '*' rabbitmq.set_user_tags 'myadmin' 'administrator'
```

salt.modules.rabbitmq.**start_app**(*runas=None*)

Start the RabbitMQ application.

CLI Example:

```
salt '*' rabbitmq.start_app
```

salt.modules.rabbitmq.**status**(*user=None*)

return rabbitmq status

CLI Example:

```
salt '*' rabbitmq.status
```

salt.modules.rabbitmq.**stop_app**(*runas=None*)

Stops the RabbitMQ application, leaving the Erlang node running.

CLI Example:

```
salt '*' rabbitmq.stop_app
```

salt.modules.rabbitmq.**user_exists**(*name*, *runas=None*)

Return whether the user exists based on rabbitmqctl list_users.

CLI Example:

```
salt '*' rabbitmq.user_exists rabbit_user
```

salt.modules.rabbitmq.**vhost_exists**(*name*, *runas=None*)

Return whether the vhost exists based on rabbitmqctl list_vhosts.

CLI Example:

```
salt '*' rabbitmq.vhost_exists rabbit_host
```

61.128 salt.modules.rbenv

Manage ruby installations with rbenv.

New in version 0.16.0.

`salt.modules.rbenv.`**`default`**`(`*ruby=None*, *runas=None*`)`
Returns or sets the currently defined default ruby.

ruby=None The version to set as the default. Should match one of the versions listed by `rbenv.versions`. Leave blank to return the current default.

CLI Example:

```
salt '*' rbenv.default
salt '*' rbenv.default 2.0.0-p0
```

`salt.modules.rbenv.`**`do`**`(`*cmdline=None*, *runas=None*`)`
Execute a ruby command with rbenv's shims from the user or the system.

CLI Example:

```
salt '*' rbenv.do 'gem list bundler'
salt '*' rbenv.do 'gem list bundler' deploy
```

`salt.modules.rbenv.`**`do_with_ruby`**`(`*ruby*, *cmdline*, *runas=None*`)`
Execute a ruby command with rbenv's shims using a specific ruby version.

CLI Example:

```
salt '*' rbenv.do_with_ruby 2.0.0-p0 'gem list bundler'
salt '*' rbenv.do_with_ruby 2.0.0-p0 'gem list bundler' deploy
```

`salt.modules.rbenv.`**`install`**`(`*runas=None*, *path=None*`)`
Install Rbenv systemwide

CLI Example:

```
salt '*' rbenv.install
```

`salt.modules.rbenv.`**`install_ruby`**`(`*ruby*, *runas=None*`)`
Install a ruby implementation.

ruby The version of Ruby to install, should match one of the versions listed by rbenv.list

CLI Example:

```
salt '*' rbenv.install_ruby 2.0.0-p0
```

`salt.modules.rbenv.`**`is_installed`**`(`*runas=None*`)`
Check if Rbenv is installed.

CLI Example:

```
salt '*' rbenv.is_installed
```

`salt.modules.rbenv.`**`list_`**`(`*runas=None*`)`
List the installable versions of ruby.

CLI Example:

```
salt '*' rbenv.list
```

`salt.modules.rbenv.`**`rehash`**`(`*runas=None*`)`

Run rbenv rehash to update the installed shims.

CLI Example:

```
salt '*' rbenv.rehash
```

`salt.modules.rbenv.`**`uninstall_ruby`**`(`*ruby*, *runas=None*`)`

Uninstall a ruby implementation.

ruby The version of ruby to uninstall. Should match one of the versions listed by `rbenv.versions`

CLI Example:

```
salt '*' rbenv.uninstall_ruby 2.0.0-p0
```

`salt.modules.rbenv.`**`update`**`(`*runas=None*, *path=None*`)`

Updates the current versions of Rbenv and Ruby-Build

CLI Example:

```
salt '*' rbenv.update
```

`salt.modules.rbenv.`**`versions`**`(`*runas=None*`)`

List the installed versions of ruby.

CLI Example:

```
salt '*' rbenv.versions
```

61.129 salt.modules.rdp

Manage RDP Service on Windows servers

`salt.modules.rdp.`**`disable`**`()`

Disable RDP the service on the server

CLI Example:

```
salt '*' rdp.disable
```

`salt.modules.rdp.`**`enable`**`()`

Enable RDP the service on the server

CLI Example:

```
salt '*' rdp.enable
```

`salt.modules.rdp.`**`status`**`()`

Show if rdp is enabled on the server

CLI Example:

```
salt '*' rdp.status
```

61.130 salt.modules.reg

Manage the registry on Windows

depends

- winreg Python module

class `salt.modules.reg.`**`Registry`**

Delay '_winreg' usage until this module is used

`salt.modules.reg.`**`create_key`**(*hkey*, *path*, *key*, *value=None*)

Create a registry key

CLI Example:

```
salt '*' reg.create_key HKEY_CURRENT_USER 'SOFTWARE\Salt' 'version' '0.97'
```

`salt.modules.reg.`**`delete_key`**(*hkey*, *path*, *key*)

Delete a registry key

Note: This cannot delete a key with subkeys

CLI Example:

```
salt '*' reg.delete_key HKEY_CURRENT_USER 'SOFTWARE\Salt' 'version'
```

`salt.modules.reg.`**`read_key`**(*hkey*, *path*, *key*)

Read registry key value

CLI Example:

```
salt '*' reg.read_key HKEY_LOCAL_MACHINE 'SOFTWARE\Salt' 'version'
```

`salt.modules.reg.`**`set_key`**(*hkey*, *path*, *key*, *value*, *vtype='REG_DWORD'*)

Set a registry key vtype: http://docs.python.org/2/library/_winreg.html#value-types

CLI Example:

```
salt '*' reg.set_key HKEY_CURRENT_USER 'SOFTWARE\Salt' 'version' '0.97' REG_DWORD
```

61.131 salt.modules.ret

Module to integrate with the returner system and retrieve data sent to a salt returner

`salt.modules.ret.`**`get_fun`**(*returner*, *fun*)

Return info about last time fun was called on each minion

CLI Example:

```
salt '*' ret.get_fun mysql network.interfaces
```

`salt.modules.ret.`**`get_jid`**(*returner*, *jid*)

Return the information for a specified job id

CLI Example:

```
salt '*' ret.get_jid redis 20421104181954700505
```

`salt.modules.ret.`**`get_jids`**(*returner*)

Return a list of all job ids

CLI Example:

```
salt '*' ret.get_jids mysql
```

`salt.modules.ret.`**`get_minions`**(*returner*)

Return a list of all minions

CLI Example:

```
salt '*' ret.get_minions mysql
```

61.132 salt.modules.rh_ip

The networking module for RHEL/Fedora based distros

`salt.modules.rh_ip.`**`apply_network_settings`**(***settings*)

Apply global network configuration.

CLI Example:

```
salt '*' ip.apply_network_settings
```

`salt.modules.rh_ip.`**`build_bond`**(*iface*, ***settings*)

Create a bond script in /etc/modprobe.d with the passed settings and load the bonding kernel module.

CLI Example:

```
salt '*' ip.build_bond bond0 mode=balance-alb
```

`salt.modules.rh_ip.`**`build_interface`**(*iface*, *iface_type*, *enabled*, ***settings*)

Build an interface script for a network interface.

CLI Example:

```
salt '*' ip.build_interface eth0 eth <settings>
```

`salt.modules.rh_ip.`**`build_network_settings`**(***settings*)

Build the global network script.

CLI Example:

```
salt '*' ip.build_network_settings <settings>
```

`salt.modules.rh_ip.`**`build_routes`**(*iface*, ***settings*)

Build a route script for a network interface.

CLI Example:

```
salt '*' ip.build_routes eth0 <settings>
```

`salt.modules.rh_ip.`**`down`**(*iface*, *iface_type*)

Shutdown a network interface

CLI Example:

```
salt '*' ip.down eth0
```

`salt.modules.rh_ip.`**`get_bond`**(*iface*)

Return the content of a bond script

CLI Example:

```
salt '*' ip.get_bond bond0
```

`salt.modules.rh_ip.`**`get_interface`**`(`*iface*`)`
Return the contents of an interface script

CLI Example:

```
salt '*' ip.get_interface eth0
```

`salt.modules.rh_ip.`**`get_network_settings`**`()`
Return the contents of the global network script.

CLI Example:

```
salt '*' ip.get_network_settings
```

`salt.modules.rh_ip.`**`get_routes`**`(`*iface*`)`
Return the contents of the interface routes script.

CLI Example:

```
salt '*' ip.get_routes eth0
```

`salt.modules.rh_ip.`**`up`**`(`*iface*, *iface_type*`)`
Start up a network interface

CLI Example:

```
salt '*' ip.up eth0
```

61.133 salt.modules.rh_service

Service support for RHEL-based systems, including support for both upstart and sysvinit

`salt.modules.rh_service.`**`available`**`(`*name*, *limit=''*`)`
Return True if the named service is available. Use the `limit` param to restrict results to services of that type.

CLI Examples:

```
salt '*' service.available sshd
salt '*' service.available sshd limit=upstart
salt '*' service.available sshd limit=sysvinit
```

`salt.modules.rh_service.`**`disable`**`(`*name*, ***kwargs*`)`
Disable the named service to start at boot

CLI Example:

```
salt '*' service.disable <service name>
```

`salt.modules.rh_service.`**`disabled`**`(`*name*`)`
Check to see if the named service is disabled to start on boot

CLI Example:

```
salt '*' service.disabled <service name>
```

`salt.modules.rh_service.`**`enable`**`(`*name*, ***kwargs*`)`
Enable the named service to start at boot

CLI Example:

```
salt '*' service.enable <service name>
```

`salt.modules.rh_service.`**`enabled`**`(`*name*`)`

Check to see if the named service is enabled to start on boot

CLI Example:

```
salt '*' service.enabled <service name>
```

`salt.modules.rh_service.`**`get_all`**`(`*limit=''*`)`

Return all installed services. Use the `limit` param to restrict results to services of that type.

CLI Example:

```
salt '*' service.get_all
salt '*' service.get_all limit=upstart
salt '*' service.get_all limit=sysvinit
```

`salt.modules.rh_service.`**`get_disabled`**`(`*limit=''*`)`

Return the disabled services. Use the `limit` param to restrict results to services of that type.

CLI Example:

```
salt '*' service.get_disabled
salt '*' service.get_disabled limit=upstart
salt '*' service.get_disabled limit=sysvinit
```

`salt.modules.rh_service.`**`get_enabled`**`(`*limit=''*`)`

Return the enabled services. Use the `limit` param to restrict results to services of that type.

CLI Examples:

```
salt '*' service.get_enabled
salt '*' service.get_enabled limit=upstart
salt '*' service.get_enabled limit=sysvinit
```

`salt.modules.rh_service.`**`missing`**`(`*name*, *limit=''*`)`

The inverse of service.available. Return True if the named service is not available. Use the `limit` param to restrict results to services of that type.

CLI Examples:

```
salt '*' service.missing sshd
salt '*' service.missing sshd limit=upstart
salt '*' service.missing sshd limit=sysvinit
```

`salt.modules.rh_service.`**`reload_`**`(`*name*`)`

Reload the named service

CLI Example:

```
salt '*' service.reload <service name>
```

`salt.modules.rh_service.`**`restart`**`(`*name*`)`

Restart the named service

CLI Example:

```
salt '*' service.restart <service name>
```

`salt.modules.rh_service.`**`start`**`(`*name*`)`

Start the specified service

CLI Example:

```
salt '*' service.start <service name>
```

`salt.modules.rh_service.`**`status`**`(`*name*, *sig=None*`)`

Return the status for a service, returns a bool whether the service is running.

CLI Example:

```
salt '*' service.status <service name>
```

`salt.modules.rh_service.`**`stop`**`(`*name*`)`

Stop the specified service

CLI Example:

```
salt '*' service.stop <service name>
```

61.134 salt.modules.riak

61.134.1 Riak Salt Module

Author: David Boucha <boucha@gmail.com>

`salt.modules.riak.`**`cluster_commit`**`()`

Commit Cluster Changes

CLI Example:

```
salt '*' riak.cluster_commit
```

`salt.modules.riak.`**`cluster_join`**`(`*riak_user=None*, *riak_host=None*`)`

Join a Riak cluster

CLI Example:

```
salt '*' riak.cluster_join <user> <host>
```

`salt.modules.riak.`**`cluster_plan`**`()`

Review Cluster Plan

CLI Example:

```
salt '*' riak.cluster_plan
```

`salt.modules.riak.`**`member_status`**`()`

Get cluster member status

CLI Example:

```
salt '*' riak.member_status
```

`salt.modules.riak.`**`start`**`()`

Start Riak

CLI Example:

```
salt '*' riak.start
```

`salt.modules.riak.`**`stop`**`()`

Stop Riak

CLI Example:

```
salt '*' riak.stop
```

61.135 salt.modules.rpm

Support for rpm

`salt.modules.rpm.`**`file_dict`**`(`**packages*`)`

List the files that belong to a package, sorted by group. Not specifying any packages will return a list of _every_ file on the system's rpm database (not generally recommended).

CLI Examples:

```
salt '*' lowpkg.file_dict httpd
salt '*' lowpkg.file_dict httpd postfix
salt '*' lowpkg.file_dict
```

`salt.modules.rpm.`**`file_list`**`(`**packages*`)`

List the files that belong to a package. Not specifying any packages will return a list of _every_ file on the system's rpm database (not generally recommended).

CLI Examples:

```
salt '*' lowpkg.file_list httpd
salt '*' lowpkg.file_list httpd postfix
salt '*' lowpkg.file_list
```

`salt.modules.rpm.`**`list_pkgs`**`(`**packages*`)`

List the packages currently installed in a dict:

```
{'<package_name>': '<version>'}
```

CLI Example:

```
salt '*' lowpkg.list_pkgs
```

`salt.modules.rpm.`**`verify`**`(`**package*`)`

Runs an rpm -Va on a system, and returns the results in a dict

CLI Example:

```
salt '*' lowpkg.verify
```

61.136 salt.modules.rvm

Manage ruby installations and gemsets with RVM, the Ruby Version Manager.

`salt.modules.rvm.`**`do`**`(`*ruby*, *command*, *runas=None*`)`

Execute a command in an RVM controlled environment.

ruby: The ruby to use.

command: The command to execute.

runas [None] The user to run rvm as.

CLI Example:

```
salt '*' rvm.do 2.0.0 <command>
```

`salt.modules.rvm.`**`gemset_copy`**(*source*, *destination*, *runas=None*)

Copy all gems from one gemset to another.

source The name of the gemset to copy, complete with ruby version.

destination The destination gemset.

runas [None] The user to run rvm as.

CLI Example:

```
salt '*' rvm.gemset_copy foobar bazquo
```

`salt.modules.rvm.`**`gemset_create`**(*ruby*, *gemset*, *runas=None*)

Creates a gemset.

ruby The ruby version to create the gemset for.

gemset The name of the gemset to create.

runas [None] The user to run rvm as.

CLI Example:

```
salt '*' rvm.gemset_create 2.0.0 foobar
```

`salt.modules.rvm.`**`gemset_delete`**(*ruby*, *gemset*, *runas=None*)

Deletes a gemset.

ruby The ruby version the gemset belongs to.

gemset The gemset to delete.

runas [None] The user to run rvm as.

CLI Example:

```
salt '*' rvm.gemset_delete 2.0.0 foobar
```

`salt.modules.rvm.`**`gemset_empty`**(*ruby*, *gemset*, *runas=None*)

Remove all gems from a gemset.

ruby The ruby version the gemset belongs to.

gemset The gemset to empty.

runas [None] The user to run rvm as.

CLI Example:

```
salt '*' rvm.gemset_empty 2.0.0 foobar
```

`salt.modules.rvm.`**`gemset_list`**(*ruby='default'*, *runas=None*)

List all gemsets for the given ruby.

ruby [default] The ruby version to list the gemsets for

runas [None] The user to run rvm as.

CLI Example:

```
salt '*' rvm.gemset_list
```

salt.modules.rvm.**gemset_list_all**(*runas=None*)

List all gemsets for all installed rubies.

Note that you must have set a default ruby before this can work.

runas [None] The user to run rvm as.

CLI Example:

```
salt '*' rvm.gemset_list_all
```

salt.modules.rvm.**get**(*version='stable'*, *runas=None*)

Update RVM.

version [stable] Which version of RVM to install, e.g. stable or head.

ruby The version of ruby to reinstall.

CLI Example:

```
salt '*' rvm.get
```

salt.modules.rvm.**install**(*runas=None*)

Install RVM system wide.

CLI Example:

```
salt '*' rvm.install
```

salt.modules.rvm.**install_ruby**(*ruby*, *runas=None*)

Install a ruby implementation.

ruby The version of ruby to install.

runas [None] The user to run rvm as.

CLI Example:

```
salt '*' rvm.install_ruby 1.9.3-p385
```

salt.modules.rvm.**is_installed**(*runas=None*)

Check if RVM is installed.

CLI Example:

```
salt '*' rvm.is_installed
```

salt.modules.rvm.**list_**(*runas=None*)

List all rvm installed rubies.

runas [None] The user to run rvm as.

CLI Example:

```
salt '*' rvm.list
```

salt.modules.rvm.**reinstall_ruby**(*ruby*, *runas=None*)

Reinstall a ruby implementation.

ruby The version of ruby to reinstall.

runas [None] The user to run rvm as.

CLI Example:

```
salt '*' rvm.reinstall_ruby 1.9.3-p385
```

`salt.modules.rvm.`**`rubygems`**`(`*ruby*, *version*, *runas=None*`)`

Installs a specific rubygems version in the given ruby.

ruby The ruby to install rubygems for.

version The version of rubygems to install or 'remove' to use the version that ships with 1.9

runas [None] The user to run rvm as.

CLI Example:

```
salt '*' rvm.rubygems 2.0.0 1.8.24
```

`salt.modules.rvm.`**`set_default`**`(`*ruby*, *runas=None*`)`

Set the default ruby.

ruby The version of ruby to make the default.

runas [None] The user to run rvm as.

CLI Example:

```
salt '*' rvm.set_default 2.0.0
```

`salt.modules.rvm.`**`wrapper`**`(`*ruby_string*, *wrapper_prefix*, *runas=None*, **binaries*`)`

Install RVM wrapper scripts.

ruby_string Ruby/gemset to install wrappers for.

wrapper_prefix What to prepend to the name of the generated wrapper binaries.

runas [None] The user to run rvm as.

binaries [None] The names of the binaries to create wrappers for. When nothing is given, wrappers for ruby, gem, rake, irb, rdoc, ri and testrb are generated.

CLI Example:

```
salt '*' rvm.wrapper <ruby_string> <wrapper_prefix>
```

61.137 salt.modules.s3

Connection module for Amazon S3

configuration This module is not usable until the following are specified either in a pillar or in the minion's config file:

```
s3.keyid: GKTADJGHEIQSXMKKRBJ08H
s3.key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
```

A service_url may also be specified in the configuration:

```
s3.service_url: s3.amazonaws.com
```

If a service_url is not specified, the default is s3.amazonaws.com. This may appear in various documentation as an "endpoint". A comprehensive list for Amazon S3 may be found at:

```
http://docs.aws.amazon.com/general/latest/gr/rande.html#s3_region
```

The service_url will form the basis for the final endpoint that is used to query the service.

This module should be usable to query other S3-like services, such as Eucalyptus.

`salt.modules.s3.`**`delete`**(*bucket*, *path=None*, *action=None*, *key=None*, *keyid=None*, *service_url=None*)

Delete a bucket, or delete an object from a bucket.

CLI Example to delete a bucket:

```
salt myminion s3.delete mybucket
```

CLI Example to delete an object from a bucket:

```
salt myminion s3.delete mybucket remoteobject
```

`salt.modules.s3.`**`get`**(*bucket=None*, *path=None*, *return_bin=False*, *action=None*, *local_file=None*, *key=None*, *keyid=None*, *service_url=None*)

List the contents of a bucket, or return an object from a bucket. Set return_bin to True in order to retrieve an object wholesale. Otherwise, Salt will attempt to parse an XML response.

CLI Example to list buckets:

```
salt myminion s3.get
```

CLI Example to list the contents of a bucket:

```
salt myminion s3.get mybucket
```

CLI Example to return the binary contents of an object:

```
salt myminion s3.get mybucket myfile.png return_bin=True
```

CLI Example to save the binary contents of an object to a local file:

```
salt myminion s3.get mybucket myfile.png local_file=/tmp/myfile.png
```

It is also possible to perform an action on a bucket. Currently, S3 supports the following actions:

```
acl
cors
lifecycle
policy
location
logging
notification
tagging
versions
requestPayment
versioning
website
```

To perform an action on a bucket:

```
salt myminion s3.get mybucket myfile.png action=acl
```

`salt.modules.s3.`**`head`**(*bucket*, *path=None*, *key=None*, *keyid=None*, *service_url=None*)

Return the metadata for a bucket, or an object in a bucket.

CLI Examples:

```
salt myminion s3.head mybucket
salt myminion s3.head mybucket myfile.png
```

`salt.modules.s3.`**`put`**(*bucket*, *path=None*, *return_bin=False*, *action=None*, *local_file=None*, *key=None*, *keyid=None*, *service_url=None*)

Create a new bucket, or upload an object to a bucket.

CLI Example to create a bucket:

```
salt myminion s3.put mybucket
```

CLI Example to upload an object to a bucket:

```
salt myminion s3.put mybucket remotepath local_path=/path/to/file
```

61.138 salt.modules.saltcloudmod

Control a salt cloud system

`salt.modules.saltcloudmod.`**`create`**(*name*, *profile*)

Create the named vm

CLI Example:

```
salt <minion-id> saltcloud.create webserver rackspace_centos_512
```

61.139 salt.modules.saltutil

The Saltutil module is used to manage the state of the salt minion itself. It is used to manage minion modules as well as automate updates to the salt minion.

depends

- esky Python module for update functionality

`salt.modules.saltutil.`**`cmd`**(*tgt*, *fun*, *arg=()*, *timeout=None*, *expr_form='glob'*, *ret=''*, *kwarg=None*, *ssh=False*, ***kwargs*)

Assuming this minion is a master, execute a salt command

CLI Example:

```
salt '*' saltutil.cmd
```

`salt.modules.saltutil.`**`cmd_iter`**(*tgt*, *fun*, *arg=()*, *timeout=None*, *expr_form='glob'*, *ret=''*, *kwarg=None*, *ssh=False*, ***kwargs*)

Assuming this minion is a master, execute a salt command

CLI Example:

```
salt '*' saltutil.cmd
```

`salt.modules.saltutil.`**`find_job`**(*jid*)

Return the data for a specific job id

CLI Example:

```
salt '*' saltutil.find_job <job id>
```

`salt.modules.saltutil.`**`is_running`**(*fun*)

If the named function is running return the data associated with it/them. The argument can be a glob

CLI Example:

```
salt '*' saltutil.is_running state.highstate
```

`salt.modules.saltutil.`**`kill_job`**(*jid*)

Sends a kill signal (SIGKILL 9) to the named salt job's process

CLI Example:

```
salt '*' saltutil.kill_job <job id>
```

`salt.modules.saltutil.`**`refresh_modules`**()

Signal the minion to refresh the module and grain data

CLI Example:

```
salt '*' saltutil.refresh_modules
```

`salt.modules.saltutil.`**`refresh_pillar`**()

Signal the minion to refresh the pillar data.

CLI Example:

```
salt '*' saltutil.refresh_pillar
```

`salt.modules.saltutil.`**`regen_keys`**()

Used to regenerate the minion keys.

CLI Example:

```
salt '*' saltutil.regen_keys
```

`salt.modules.saltutil.`**`revoke_auth`**()

The minion sends a request to the master to revoke its own key. Note that the minion session will be revoked and the minion may not be able to return the result of this command back to the master.

CLI Example:

```
salt '*' saltutil.revoke_auth
```

`salt.modules.saltutil.`**`running`**()

Return the data on all running salt processes on the minion

CLI Example:

```
salt '*' saltutil.running
```

`salt.modules.saltutil.`**`signal_job`**(*jid*, *sig*)

Sends a signal to the named salt job's process

CLI Example:

```
salt '*' saltutil.signal_job <job id> 15
```

`salt.modules.saltutil.`**`sync_all`**(*saltenv=None*, *refresh=True*)

Sync down all of the dynamic modules from the file server for a specific environment

CLI Example:

```
salt '*' saltutil.sync_all
```

salt.modules.saltutil.**sync_grains**(*saltenv=None*, *refresh=True*)

Sync the grains from the _grains directory on the salt master file server. This function is environment aware, pass the desired environment to grab the contents of the _grains directory, base is the default environment.

CLI Example:

```
salt '*' saltutil.sync_grains
```

salt.modules.saltutil.**sync_modules**(*saltenv=None*, *refresh=True*)

Sync the modules from the _modules directory on the salt master file server. This function is environment aware, pass the desired environment to grab the contents of the _modules directory, base is the default environment.

CLI Example:

```
salt '*' saltutil.sync_modules
```

salt.modules.saltutil.**sync_outputters**(*saltenv=None*, *refresh=True*)

Sync the outputters from the _outputters directory on the salt master file server. This function is environment aware, pass the desired environment to grab the contents of the _outputters directory, base is the default environment.

CLI Example:

```
salt '*' saltutil.sync_outputters
```

salt.modules.saltutil.**sync_renderers**(*saltenv=None*, *refresh=True*)

Sync the renderers from the _renderers directory on the salt master file server. This function is environment aware, pass the desired environment to grab the contents of the _renderers directory, base is the default environment.

CLI Example:

```
salt '*' saltutil.sync_renderers
```

salt.modules.saltutil.**sync_returners**(*saltenv=None*, *refresh=True*)

Sync the returners from the _returners directory on the salt master file server. This function is environment aware, pass the desired environment to grab the contents of the _returners directory, base is the default environment.

CLI Example:

```
salt '*' saltutil.sync_returners
```

salt.modules.saltutil.**sync_states**(*saltenv=None*, *refresh=True*)

Sync the states from the _states directory on the salt master file server. This function is environment aware, pass the desired environment to grab the contents of the _states directory, base is the default environment.

CLI Example:

```
salt '*' saltutil.sync_states
```

salt.modules.saltutil.**term_job**(*jid*)

Sends a termination signal (SIGTERM 15) to the named salt job's process

CLI Example:

```
salt '*' saltutil.term_job <job id>
```

`salt.modules.saltutil.`**`update`**`(`*version=None*`)`

Update the salt minion from the URL defined in opts['update_url']

This feature requires the minion to be running a bdist_esky build.

The version number is optional and will default to the most recent version available at opts['update_url'].

Returns details about the transaction upon completion.

CLI Example:

```
salt '*' saltutil.update 0.10.3
```

61.140 salt.modules.seed

Virtual machine image management tools

`salt.modules.seed.`**`apply_`**`(`*path*, *id_=None*, *config=None*, *approve_key=True*, *install=True*`)`

Seed a location (disk image, directory, or block device) with the minion config, approve the minion's key, and/or install salt-minion.

CLI Example:

```
salt 'minion' seed.whatever path id [config=config_data] \
        [gen_key=(true|false)] [approve_key=(true|false)] \
        [install=(true|false)]
```

path Full path to the directory, device, or disk image on the target minion's file system.

id Minion id with which to seed the path.

config Minion configuration options. By default, the 'master' option is set to the target host's 'master'.

approve_key Request a pre-approval of the generated minion key. Requires that the salt-master be configured to either auto-accept all keys or expect a signing request from the target host. Default: true.

install Install salt-minion, if absent. Default: true.

61.141 salt.modules.selinux

Execute calls on selinux

Note: This module requires the `semanage` and `setsebool` commands to be available on the minion. On RHEL-based distros, this means that the `policycoreutils` and `policycoreutils-python` packages must be installed. If not on a RHEL-based distribution, consult the selinux documentation for your distro to ensure that the proper packages are installed.

`salt.modules.selinux.`**`getenforce`**`()`

Return the mode selinux is running in

CLI Example:

```
salt '*' selinux.getenforce
```

`salt.modules.selinux.`**`getsebool`**`(`*boolean*`)`

Return the information on a specific selinux boolean

CLI Example:

```
salt '*' selinux.getsebool virt_use_usb
```

`salt.modules.selinux.`**`list_sebool`**`()`

Return a structure listing all of the selinux booleans on the system and what state they are in

CLI Example:

```
salt '*' selinux.list_sebool
```

`salt.modules.selinux.`**`selinux_fs_path`**`(`**args*`)`

Return the location of the SELinux VFS directory

CLI Example:

```
salt '*' selinux.selinux_fs_path
```

`salt.modules.selinux.`**`setenforce`**`(`*mode*`)`

Set the SELinux enforcing mode

CLI Example:

```
salt '*' selinux.setenforce enforcing
```

`salt.modules.selinux.`**`setsebool`**`(`*boolean*, *value*, *persist=False*`)`

Set the value for a boolean

CLI Example:

```
salt '*' selinux.setsebool virt_use_usb off
```

`salt.modules.selinux.`**`setsebools`**`(`*pairs*, *persist=False*`)`

Set the value of multiple booleans

CLI Example:

```
salt '*' selinux.setsebools '{virt_use_usb: on, squid_use_tproxy: off}'
```

61.142 salt.modules.service

The default service module, if not otherwise specified salt will fall back to this basic module

`salt.modules.service.`**`available`**`(`*name*`)`

Returns `True` if the specified service is available, otherwise returns `False`.

CLI Example:

```
salt '*' service.available sshd
```

`salt.modules.service.`**`get_all`**`()`

Return a list of all available services

CLI Example:

```
salt '*' service.get_all
```

`salt.modules.service.`**`missing`**`(`*name*`)`

The inverse of service.available. Returns `True` if the specified service is not available, otherwise returns `False`.

CLI Example:

```
salt '*' service.missing sshd
```

`salt.modules.service.`**`reload_`**`(`*name*`)`

Restart the specified service

CLI Example:

```
salt '*' service.reload <service name>
```

`salt.modules.service.`**`restart`**`(`*name*`)`

Restart the specified service

CLI Example:

```
salt '*' service.restart <service name>
```

`salt.modules.service.`**`start`**`(`*name*`)`

Start the specified service

CLI Example:

```
salt '*' service.start <service name>
```

`salt.modules.service.`**`status`**`(`*name*, *sig=None*`)`

Return the status for a service, returns the PID or an empty string if the service is running or not, pass a signature to use to find the service via ps

CLI Example:

```
salt '*' service.status <service name> [service signature]
```

`salt.modules.service.`**`stop`**`(`*name*`)`

Stop the specified service

CLI Example:

```
salt '*' service.stop <service name>
```

61.143 salt.modules.shadow

Manage the shadow file

`salt.modules.shadow.`**`default_hash`**`()`

Returns the default hash used for unset passwords

CLI Example:

```
salt '*' shadow.default_hash
```

`salt.modules.shadow.`**`info`**`(`*name*`)`

Return information for the specified user

CLI Example:

```
salt '*' shadow.info root
```

`salt.modules.shadow.`**`set_date`**`(`*name*, *date*`)`

sets the value for the date the password was last changed to the epoch (January 1, 1970). See man chage.

CLI Example:

```
salt '*' shadow.set_date username 0
```

`salt.modules.shadow.`**`set_inactdays`**`(`*name*, *inactdays*`)`

Set the number of days of inactivity after a password has expired before the account is locked. See man chage.

CLI Example:

```
salt '*' shadow.set_inactdays username 7
```

`salt.modules.shadow.`**`set_maxdays`**`(`*name*, *maxdays*`)`

Set the maximum number of days during which a password is valid. See man chage.

CLI Example:

```
salt '*' shadow.set_maxdays username 90
```

`salt.modules.shadow.`**`set_mindays`**`(`*name*, *mindays*`)`

Set the minimum number of days between password changes. See man chage.

CLI Example:

```
salt '*' shadow.set_mindays username 7
```

`salt.modules.shadow.`**`set_password`**`(`*name*, *password*, *use_usermod=False*`)`

Set the password for a named user. The password must be a properly defined hash. The password hash can be generated with this command:

`python -c "import crypt; print crypt.crypt('password', '\$6\$SALTsalt')"`

`SALTsalt` is the 8-character crpytographic salt. Valid characters in the salt are `.`, `/`, and any alphanumeric character.

Keep in mind that the $6 represents a sha512 hash, if your OS is using a different hashing algorithm this needs to be changed accordingly

CLI Example:

```
salt '*' shadow.set_password root '$1$UYCIxa628.9qXjpQCjM4a..'
```

`salt.modules.shadow.`**`set_warndays`**`(`*name*, *warndays*`)`

Set the number of days of warning before a password change is required. See man chage.

CLI Example:

```
salt '*' shadow.set_warndays username 7
```

61.144 salt.modules.smartos_imgadm

Module for running imgadm command on SmartOS

`salt.modules.smartos_imgadm.`**`avail`**`(`*search=None*`)`

Return a list of available images

CLI Example:

```
salt '*' imgadm.avail [percona]
```

`salt.modules.smartos_imgadm.`**`delete`**`(`*uuid=None*`)`

Remove an installed image

CLI Example:

```
salt '*' imgadm.delete e42f8c84-bbea-11e2-b920-078fab2aab1f
```

`salt.modules.smartos_imgadm.`**`get`**`(`*uuid=None*`)`

Return info on an installed image

CLI Example:

```
salt '*' imgadm.get e42f8c84-bbea-11e2-b920-078fab2aab1f
```

`salt.modules.smartos_imgadm.`**`import_image`**`(`*uuid=None*`)`

Import an image from the repository

CLI Example:

```
salt '*' imgadm.import_image e42f8c84-bbea-11e2-b920-078fab2aab1f
```

`salt.modules.smartos_imgadm.`**`list_installed`**`()`

Return a list of installed images

CLI Example:

```
salt '*' imgadm.list_installed
```

`salt.modules.smartos_imgadm.`**`show`**`(`*uuid=None*`)`

Show manifest of a given image

CLI Example:

```
salt '*' imgadm.show e42f8c84-bbea-11e2-b920-078fab2aab1f
```

`salt.modules.smartos_imgadm.`**`update_installed`**`()`

Gather info on unknown images (locally installed)

CLI Example:

```
salt '*' imgadm.update_installed()
```

`salt.modules.smartos_imgadm.`**`version`**`()`

Return imgadm version

CLI Example:

```
salt '*' imgadm.version
```

61.145 salt.modules.smartos_vmadm

Module for managing VMs on SmartOS

`salt.modules.smartos_vmadm.`**`destroy`**`(`*uuid=None*`)`

Hard power down the virtual machine, this is equivalent to pulling the power

CLI Example:

```
salt '*' virt.destroy <uuid>
```

`salt.modules.smartos_vmadm.`**`get_macs`**`(`*uuid=None*`)`

Return a list off MAC addresses from the named VM

CLI Example:

```
salt '*' virt.get_macs <uuid>
```

`salt.modules.smartos_vmadm.`**`init`**`(`*`**kwargs`*`)`

Initialize a new VM

CLI Example:

```
salt '*' virt.init image_uuid='...' alias='...' [...]
```

`salt.modules.smartos_vmadm.`**`list_active_vms`**`()`

Return a list of uuids for active virtual machine on the minion

CLI Example:

```
salt '*' virt.list_active_vms
```

`salt.modules.smartos_vmadm.`**`list_inactive_vms`**`()`

Return a list of uuids for inactive virtual machine on the minion

CLI Example:

```
salt '*' virt.list_inactive_vms
```

`salt.modules.smartos_vmadm.`**`list_vms`**`()`

Return a list of virtual machine names on the minion

CLI Example:

```
salt '*' virt.list_vms
```

`salt.modules.smartos_vmadm.`**`reboot`**`(`*`uuid=None`*`)`

Reboot a domain via ACPI request

CLI Example:

```
salt '*' virt.reboot <uuid>
```

`salt.modules.smartos_vmadm.`**`setmem`**`(`*`uuid, memory`*`)`

Change the amount of memory allocated to VM. <memory> is to be specified in MB.

Note for KVM : this would require a restart of the VM.

CLI Example:

```
salt '*' virt.setmem <uuid> 512
```

`salt.modules.smartos_vmadm.`**`shutdown`**`(`*`uuid=None`*`)`

Send a soft shutdown signal to the named vm

CLI Example:

```
salt '*' virt.shutdown <uuid>
```

`salt.modules.smartos_vmadm.`**`start`**`(`*`uuid=None`*`)`

Start a defined domain

CLI Example:

```
salt '*' virt.start <uuid>
```

`salt.modules.smartos_vmadm.`**`vm_info`**`(`*`uuid=None`*`)`

Return a dict with information about the specified VM on this CN

CLI Example:

```
salt '*' virt.vm_info <uuid>
```

`salt.modules.smartos_vmadm.`**`vm_virt_type`**(*uuid=None*)

Return VM virtualization type : OS or KVM

CLI Example:

```
salt '*' virt.vm_virt_type <uuid>
```

61.146 salt.modules.smf

Service support for Solaris 10 and 11, should work with other systems that use SMF also. (e.g. SmartOS)

`salt.modules.smf.`**`available`**(*name*)

Returns `True` if the specified service is available, otherwise returns `False`.

CLI Example:

```
salt '*' service.available net-snmp
```

`salt.modules.smf.`**`disable`**(*name*, ***kwargs*)

Disable the named service to start at boot

CLI Example:

```
salt '*' service.disable <service name>
```

`salt.modules.smf.`**`disabled`**(*name*)

Check to see if the named service is disabled to start on boot

CLI Example:

```
salt '*' service.disabled <service name>
```

`salt.modules.smf.`**`enable`**(*name*, ***kwargs*)

Enable the named service to start at boot

CLI Example:

```
salt '*' service.enable <service name>
```

`salt.modules.smf.`**`enabled`**(*name*)

Check to see if the named service is enabled to start on boot

CLI Example:

```
salt '*' service.enabled <service name>
```

`salt.modules.smf.`**`get_all`**()

Return all installed services

CLI Example:

```
salt '*' service.get_all
```

`salt.modules.smf.`**`get_disabled`**()

Return the disabled services

CLI Example:

```
salt '*' service.get_disabled
```

`salt.modules.smf.`**`get_enabled`**`()`

Return the enabled services

CLI Example:

```
salt '*' service.get_enabled
```

`salt.modules.smf.`**`get_running`**`()`

Return the running services

CLI Example:

```
salt '*' service.get_running
```

`salt.modules.smf.`**`get_stopped`**`()`

Return the stopped services

CLI Example:

```
salt '*' service.get_stopped
```

`salt.modules.smf.`**`missing`**`(`*name*`)`

The inverse of service.available. Returns `True` if the specified service is not available, otherwise returns `False`.

CLI Example:

```
salt '*' service.missing net-snmp
```

`salt.modules.smf.`**`reload_`**`(`*name*`)`

Reload the named service

CLI Example:

```
salt '*' service.reload <service name>
```

`salt.modules.smf.`**`restart`**`(`*name*`)`

Restart the named service

CLI Example:

```
salt '*' service.restart <service name>
```

`salt.modules.smf.`**`start`**`(`*name*`)`

Start the specified service

CLI Example:

```
salt '*' service.start <service name>
```

`salt.modules.smf.`**`status`**`(`*name*, *sig=None*`)`

Return the status for a service, returns a bool whether the service is running.

CLI Example:

```
salt '*' service.status <service name>
```

`salt.modules.smf.`**`stop`**`(`*name*`)`

Stop the specified service

CLI Example:

```
salt '*' service.stop <service name>
```

61.147 salt.modules.solaris_group

Manage groups on Solaris

`salt.modules.solaris_group.`**`add`**`(`*name*, *gid=None*, ***kwargs*`)`

Add the specified group

CLI Example:

```
salt '*' group.add foo 3456
```

`salt.modules.solaris_group.`**`chgid`**`(`*name*, *gid*`)`

Change the gid for a named group

CLI Example:

```
salt '*' group.chgid foo 4376
```

`salt.modules.solaris_group.`**`delete`**`(`*name*`)`

Remove the named group

CLI Example:

```
salt '*' group.delete foo
```

`salt.modules.solaris_group.`**`getent`**`(`*refresh=False*`)`

Return info on all groups

CLI Example:

```
salt '*' group.getent
```

`salt.modules.solaris_group.`**`info`**`(`*name*`)`

Return information about a group

CLI Example:

```
salt '*' group.info foo
```

61.148 salt.modules.solaris_shadow

Manage the password database on Solaris systems

`salt.modules.solaris_shadow.`**`default_hash`**`()`

Returns the default hash used for unset passwords

CLI Example:

```
salt '*' shadow.default_hash
```

`salt.modules.solaris_shadow.`**`info`**`(`*name*`)`

Return information for the specified user

CLI Example:

```
salt '*' shadow.info root
```

`salt.modules.solaris_shadow.`**`set_maxdays`**(*name*, *maxdays*)

Set the maximum number of days during which a password is valid. See man passwd.

CLI Example:

```
salt '*' shadow.set_maxdays username 90
```

`salt.modules.solaris_shadow.`**`set_mindays`**(*name*, *mindays*)

Set the minimum number of days between password changes. See man passwd.

CLI Example:

```
salt '*' shadow.set_mindays username 7
```

`salt.modules.solaris_shadow.`**`set_password`**(*name*, *password*)

Set the password for a named user. The password must be a properly defined hash, the password hash can be generated with this command: `openssl passwd -1 <plaintext password>`

CLI Example:

```
salt '*' shadow.set_password root $1$UYCIxa628.9qXjpQCjM4a..
```

`salt.modules.solaris_shadow.`**`set_warndays`**(*name*, *warndays*)

Set the number of days of warning before a password change is required. See man passwd.

CLI Example:

```
salt '*' shadow.set_warndays username 7
```

61.149 salt.modules.solaris_user

Manage users with the useradd command

`salt.modules.solaris_user.`**`add`**(*name*, *uid=None*, *gid=None*, *groups=None*, *home=None*, *shell=None*, *unique=True*, *fullname=''*, *roomnumber=''*, *workphone=''*, *homephone=''*, *createhome=True*, ***kwargs*)

Add a user to the minion

CLI Example:

```
salt '*' user.add name <uid> <gid> <groups> <home> <shell>
```

`salt.modules.solaris_user.`**`chfullname`**(*name*, *fullname*)

Change the user's Full Name

CLI Example:

```
salt '*' user.chfullname foo "Foo Bar"
```

`salt.modules.solaris_user.`**`chgid`**(*name*, *gid*)

Change the default group of the user

CLI Example:

```
salt '*' user.chgid foo 4376
```

`salt.modules.solaris_user.`**`chgroups`**(*name*, *groups*, *append=False*)

Change the groups this user belongs to, add append to append the specified groups

CLI Example:

```
salt '*' user.chgroups foo wheel,root True
```

`salt.modules.solaris_user.`**`chhome`**(*name*, *home*, *persist=False*)

Change the home directory of the user, pass true for persist to copy files to the new home dir

CLI Example:

```
salt '*' user.chhome foo /home/users/foo True
```

`salt.modules.solaris_user.`**`chhomephone`**(*name*, *homephone*)

Change the user's Home Phone

CLI Example:

```
salt '*' user.chhomephone foo "7735551234"
```

`salt.modules.solaris_user.`**`chroomnumber`**(*name*, *roomnumber*)

Change the user's Room Number

CLI Example:

```
salt '*' user.chroomnumber foo 123
```

`salt.modules.solaris_user.`**`chshell`**(*name*, *shell*)

Change the default shell of the user

CLI Example:

```
salt '*' user.chshell foo /bin/zsh
```

`salt.modules.solaris_user.`**`chuid`**(*name*, *uid*)

Change the uid for a named user

CLI Example:

```
salt '*' user.chuid foo 4376
```

`salt.modules.solaris_user.`**`chworkphone`**(*name*, *workphone*)

Change the user's Work Phone

CLI Example:

```
salt '*' user.chworkphone foo "7735550123"
```

`salt.modules.solaris_user.`**`delete`**(*name*, *remove=False*, *force=False*)

Remove a user from the minion

CLI Example:

```
salt '*' user.delete name remove=True force=True
```

`salt.modules.solaris_user.`**`getent`**(*refresh=False*)

Return the list of all info for all users

CLI Example:

```
salt '*' user.getent
```

`salt.modules.solaris_user.`**`info`**(*name*)

Return user information

CLI Example:

```
salt '*' user.info root
```

salt.modules.solaris_user.**list_groups**(*name*)

Return a list of groups the named user belongs to

CLI Example:

```
salt '*' user.list_groups foo
```

61.150 salt.modules.solarispkg

Package support for Solaris

salt.modules.solarispkg.**install**(*name=None*, *sources=None*, *saltenv='base'*, ***kwargs*)

Install the passed package. Can install packages from the following sources:

```
* Locally (package already exists on the minion
* HTTP/HTTPS server
* FTP server
* Salt master
```

Returns a dict containing the new package names and versions:

```
{'<package>': {'old': '<old-version>',
               'new': '<new-version>'}}
```

CLI Example, installing a datastream pkg that already exists on the minion:

```
salt '*' pkg.install sources='[{"<pkg name>": "/dir/on/minion/<pkg filename>"}]'
salt '*' pkg.install sources='[{"SMClgcc346": "/var/spool/pkg/gcc-3.4.6-sol10-sparc-local.pkg"}]
```

CLI Example, installing a datastream pkg that exists on the salt master:

```
salt '*' pkg.install sources='[{"<pkg name>": "salt://pkgs/<pkg filename>"}]'
salt '*' pkg.install sources='[{"SMClgcc346": "salt://pkgs/gcc-3.4.6-sol10-sparc-local.pkg"}]'
```

CLI Example, installing a datastream pkg that exists on a HTTP server:

```
salt '*' pkg.install sources='[{"<pkg name>": "http://packages.server.com/<pkg filename>"}]'
salt '*' pkg.install sources='[{"SMClgcc346": "http://packages.server.com/gcc-3.4.6-sol10-sparc-
```

If working with solaris zones and you want to install a package only in the global zone you can pass 'current_zone_only=True' to salt to have the package only installed in the global zone. (Behind the scenes this is passing '-G' to the pkgadd command.) Solaris default when installing a package in the global zone is to install it in all zones. This overrides that and installs the package only in the global.

CLI Example, installing a datastream package only in the global zone:

```
salt 'global_zone' pkg.install sources='[{"SMClgcc346": "/var/spool/pkg/gcc-3.4.6-sol10-sparc-l
```

By default salt automatically provides an adminfile, to automate package installation, with these options set:

```
email=
instance=quit
partial=nocheck
runlevel=nocheck
idepend=nocheck
rdepend=nocheck
space=nocheck
```

```
setuid=nocheck
conflict=nocheck
action=nocheck
basedir=default
```

You can override any of these options in two ways. First you can optionally pass any of the options as a kwarg to the module/state to override the default value or you can optionally pass the 'admin_source' option providing your own adminfile to the minions.

Note: You can find all of the possible options to provide to the adminfile by reading the admin man page:

```
man -s 4 admin
```

CLI Example - Overriding the 'instance' adminfile option when calling the module directly:

```
salt '*' pkg.install sources='[{"<pkg name>": "salt://pkgs/<pkg filename>"}]' instance="overwrit
```

CLI Example - Overriding the 'instance' adminfile option when used in a state:

```
SMClgcc346:
  pkg.installed:
    - sources:
      - SMClgcc346: salt://srv/salt/pkgs/gcc-3.4.6-sol10-sparc-local.pkg
    - instance: overwrite
```

Note: the ID declaration is ignored, as the package name is read from the "sources" parameter.

CLI Example - Providing your own adminfile when calling the module directly:

```
salt '*' pkg.install sources='[{"<pkg name>": "salt://pkgs/<pkg filename>"}]' admin_source='salt
```

CLI Example - Providing your own adminfile when using states:

```
<pkg name>:
  pkg.installed:
    - sources:
      - <pkg name>: salt://pkgs/<pkg filename>
    - admin_source: salt://pkgs/<adminfile filename>
```

Note: the ID declaration is ignored, as the package name is read from the "sources" parameter.

`salt.modules.solarispkg.`**`latest_version`**`(*names, **kwargs)`

Return the latest version of the named package available for upgrade or installation. If more than one package name is specified, a dict of name/version pairs is returned.

If the latest version of a given package is already installed, an empty string will be returned for that package.

CLI Example:

```
salt '*' pkg.latest_version <package name>
salt '*' pkg.latest_version <package1> <package2> <package3> ...
```

NOTE: As package repositories are not presently supported for Solaris pkgadd, this function will always return an empty string for a given package.

`salt.modules.solarispkg.`**`list_pkgs`**`(versions_as_list=False, **kwargs)`

List the packages currently installed as a dict:

```
{'<package_name>': '<version>'}
```

CLI Example:

```
salt '*' pkg.list_pkgs
```

`salt.modules.solarispkg.`**`purge`**`(`*name=None*, *pkgs=None*, ***kwargs*`)`

Package purges are not supported, this function is identical to `remove()`.

name The name of the package to be deleted

Multiple Package Options:

pkgs A list of packages to delete. Must be passed as a python list. The `name` parameter will be ignored if this option is passed.

New in version 0.16.0.

Returns a dict containing the changes.

CLI Example:

```
salt '*' pkg.purge <package name>
salt '*' pkg.purge <package1>,<package2>,<package3>
salt '*' pkg.purge pkgs='["foo", "bar"]'
```

`salt.modules.solarispkg.`**`remove`**`(`*name=None*, *pkgs=None*, *saltenv='base'*, ***kwargs*`)`

Remove packages with pkgrm

name The name of the package to be deleted

By default salt automatically provides an adminfile, to automate package removal, with these options set:

```
email=
instance=quit
partial=nocheck
runlevel=nocheck
idepend=nocheck
rdepend=nocheck
space=nocheck
setuid=nocheck
conflict=nocheck
action=nocheck
basedir=default
```

You can override any of these options in two ways. First you can optionally pass any of the options as a kwarg to the module/state to override the default value or you can optionally pass the 'admin_source' option providing your own adminfile to the minions.

Note: You can find all of the possible options to provide to the adminfile by reading the admin man page:

```
man -s 4 admin
```

Multiple Package Options:

pkgs A list of packages to delete. Must be passed as a python list. The `name` parameter will be ignored if this option is passed.

New in version 0.16.0.

Returns a dict containing the changes.

CLI Example:

```
salt '*' pkg.remove <package name>
salt '*' pkg.remove SUNWgit
salt '*' pkg.remove <package1>,<package2>,<package3>
salt '*' pkg.remove pkgs='["foo", "bar"]'
```

`salt.modules.solarispkg.`**`upgrade_available`**`(`*name*`)`

Check whether or not an upgrade is available for a given package

CLI Example:

```
salt '*' pkg.upgrade_available <package name>
```

`salt.modules.solarispkg.`**`version`**`(`**names*, ***kwargs*`)`

Returns a string representing the package version or an empty string if not installed. If more than one package name is specified, a dict of name/version pairs is returned.

CLI Example:

```
salt '*' pkg.version <package name>
salt '*' pkg.version <package1> <package2> <package3> ...
```

61.151 salt.modules.solr

61.151.1 Apache Solr Salt Module

Author: Jed Glazner Version: 0.2.1 Modified: 12/09/2011

This module uses HTTP requests to talk to the apache solr request handlers to gather information and report errors. Because of this the minion doesn't necessarily need to reside on the actual slave. However if you want to use the signal function the minion must reside on the physical solr host.

This module supports multi-core and standard setups. Certain methods are master/slave specific. Make sure you set the solr.type. If you have questions or want a feature request please ask.

Coming Features in 0.3

1. Add command for checking for replication failures on slaves
2. Improve match_index_versions since it's pointless on busy solr masters
3. Add additional local fs checks for backups to make sure they succeeded

Override these in the minion config

solr.cores A list of core names eg ['core1','core2']. An empty list indicates non-multicore setup.

solr.baseurl The root level URL to access solr via HTTP

solr.request_timeout The number of seconds before timing out an HTTP/HTTPS/FTP request. If nothing is specified then the python global timeout setting is used.

solr.type Possible values are 'master' or 'slave'

solr.backup_path The path to store your backups. If you are using cores and you can specify to append the core name to the path in the backup method.

solr.num_backups For versions of solr >= 3.5. Indicates the number of backups to keep. This option is ignored if your version is less.

solr.init_script The full path to your init script with start/stop options

solr.dih.options A list of options to pass to the DIH.

Required Options for DIH

clean [False] Clear the index before importing

commit [True] Commit the documents to the index upon completion

optimize [True] Optimize the index after commit is complete

verbose [True] Get verbose output

`salt.modules.solr.`**`abort_import`**(*handler*, *host=None*, *core_name=None*, *verbose=False*)

MASTER ONLY Aborts an existing import command to the specified handler. This command can only be run if the minion is configured with solr.type=master

handler [str] The name of the data import handler.

host [str (None)] The solr host to query. __opts__['host'] is default.

core [str (None)] The core the handler belongs to.

verbose [boolean (False)] Run the command with verbose output.

Return : dict<str,obj>:

```
{'success':boolean, 'data':dict, 'errors':list, 'warnings':list}
```

CLI Example:

```
salt '*' solr.abort_import dataimport None music {'clean':True}
```

`salt.modules.solr.`**`backup`**(*host=None*, *core_name=None*, *append_core_to_path=False*)

Tell solr make a backup. This method can be mis-leading since it uses the backup API. If an error happens during the backup you are not notified. The status: 'OK' in the response simply means that solr received the request successfully.

host [str (None)] The solr host to query. __opts__['host'] is default.

core_name [str (None)] The name of the solr core if using cores. Leave this blank if you are not using cores or if you want to check all cores.

append_core_to_path [boolean (False)] If True add the name of the core to the backup path. Assumes that minion backup path is not None.

Return : dict<str,obj>:

```
{'success':boolean, 'data':dict, 'errors':list, 'warnings':list}
```

CLI Example:

```
salt '*' solr.backup music
```

`salt.modules.solr.`**`core_status`**(*host=None*, *core_name=None*)

MULTI-CORE HOSTS ONLY Get the status for a given core or all cores if no core is specified

host [str (None)] The solr host to query. __opts__['host'] is default.

core_name [str] The name of the core to reload

Return : dict<str,obj>:

```
{'success':boolean, 'data':dict, 'errors':list, 'warnings':list}
```

CLI Example:

```
salt '*' solr.core_status None music
```

salt.modules.solr.**delta_import**(*handler*, *host=None*, *core_name=None*, *options=None*, *extra=None*)

Submits an import command to the specified handler using specified options. This command can only be run if the minion is configured with solr.type=master

handler [str] The name of the data import handler.

host [str (None)] The solr host to query. __opts__['host'] is default.

core [str (None)] The core the handler belongs to.

options [dict (__opts__)] A list of options such as clean, optimize commit, verbose, and pause_replication. leave blank to use __opts__ defaults. options will be merged with __opts__

extra [dict ([])] Extra name value pairs to pass to the handler. eg ["name=value"]

Return : dict<str,obj>:

```
{'success':boolean, 'data':dict, 'errors':list, 'warnings':list}
```

CLI Example:

```
salt '*' solr.delta_import dataimport None music {'clean':True}
```

salt.modules.solr.**full_import**(*handler*, *host=None*, *core_name=None*, *options=None*, *extra=None*)

MASTER ONLY Submits an import command to the specified handler using specified options. This command can only be run if the minion is configured with solr.type=master

handler [str] The name of the data import handler.

host [str (None)] The solr host to query. __opts__['host'] is default.

core [str (None)] The core the handler belongs to.

options [dict (__opts__)] A list of options such as clean, optimize commit, verbose, and pause_replication. leave blank to use __opts__ defaults. options will be merged with __opts__

extra [dict ([])] Extra name value pairs to pass to the handler. e.g. ["name=value"]

Return : dict<str,obj>:

```
{'success':boolean, 'data':dict, 'errors':list, 'warnings':list}
```

CLI Example:

```
salt '*' solr.full_import dataimport None music {'clean':True}
```

salt.modules.solr.**import_status**(*handler*, *host=None*, *core_name=None*, *verbose=False*)

Submits an import command to the specified handler using specified options. This command can only be run if the minion is configured with solr.type: 'master'

handler [str] The name of the data import handler.

host [str (None)] The solr host to query. __opts__['host'] is default.

core [str (None)] The core the handler belongs to.

verbose [boolean (False)] Specifies verbose output

Return : dict<str,obj>:

```
{'success':boolean, 'data':dict, 'errors':list, 'warnings':list}
```

CLI Example:

```
salt '*' solr.import_status dataimport None music False
```

`salt.modules.solr.`**`is_replication_enabled`**(*host=None*, *core_name=None*)

SLAVE CALL Check for errors, and determine if a slave is replicating or not.

host [str (None)] The solr host to query. __opts__['host'] is default.

core_name [str (None)] The name of the solr core if using cores. Leave this blank if you are not using cores or if you want to check all cores.

Return : dict<str,obj>:

```
{'success':boolean, 'data':dict, 'errors':list, 'warnings':list}
```

CLI Example:

```
salt '*' solr.is_replication_enabled music
```

`salt.modules.solr.`**`lucene_version`**(*core_name=None*)

Gets the lucene version that solr is using. If you are running a multi-core setup you should specify a core name since all the cores run under the same servlet container, they will all have the same version.

core_name [str (None)] The name of the solr core if using cores. Leave this blank if you are not using cores or if you want to check all cores.

Return: dict<str,obj>:

```
{'success':boolean, 'data':dict, 'errors':list, 'warnings':list}
```

CLI Example:

```
salt '*' solr.lucene_version
```

`salt.modules.solr.`**`match_index_versions`**(*host=None*, *core_name=None*)

SLAVE CALL Verifies that the master and the slave versions are in sync by comparing the index version. If you are constantly pushing updates the index the master and slave versions will seldom match. A solution to this is pause indexing every so often to allow the slave to replicate and then call this method before allowing indexing to resume.

host [str (None)] The solr host to query. __opts__['host'] is default.

core_name [str (None)] The name of the solr core if using cores. Leave this blank if you are not using cores or if you want to check all cores.

Return : dict<str,obj>:

```
{'success':boolean, 'data':dict, 'errors':list, 'warnings':list}
```

CLI Example:

```
salt '*' solr.match_index_versions music
```

`salt.modules.solr.`**`optimize`**(*host=None*, *core_name=None*)

Search queries fast, but it is a very expensive operation. The ideal process is to run this with a master/slave configuration. Then you can optimize the master, and push the optimized index to the slaves. If you are running a single solr instance, or if you are going to run this on a slave be aware than search performance will be horrible while this command is being run. Additionally it can take a LONG time to run and your HTTP request may timeout. If that happens adjust your timeout settings.

host [str (None)] The solr host to query. __opts__['host'] is default.

core_name [str (None)] The name of the solr core if using cores. Leave this blank if you are not using cores or if you want to check all cores.

Return : dict<str,obj>:

```
{'success':boolean, 'data':dict, 'errors':list, 'warnings':list}
```

CLI Example:

```
salt '*' solr.optimize music
```

`salt.modules.solr.`**`ping`**(*host=None*, *core_name=None*)

Does a health check on solr, makes sure solr can talk to the indexes.

host [str (None)] The solr host to query. __opts__['host'] is default.

core_name [str (None)] The name of the solr core if using cores. Leave this blank if you are not using cores or if you want to check all cores.

Return : dict<str,obj>:

```
{'success':boolean, 'data':dict, 'errors':list, 'warnings':list}
```

CLI Example:

```
salt '*' solr.ping music
```

`salt.modules.solr.`**`reload_core`**(*host=None*, *core_name=None*)

MULTI-CORE HOSTS ONLY Load a new core from the same configuration as an existing registered core. While the "new" core is initializing, the "old" one will continue to accept requests. Once it has finished, all new request will go to the "new" core, and the "old" core will be unloaded.

host [str (None)] The solr host to query. __opts__['host'] is default.

core_name [str] The name of the core to reload

Return : dict<str,obj>:

```
{'success':boolean, 'data':dict, 'errors':list, 'warnings':list}
```

CLI Example:

```
salt '*' solr.reload_core None music
```

Return data is in the following format:

```
{'success':bool, 'data':dict, 'errors':list, 'warnings':list}
```

`salt.modules.solr.`**`reload_import_config`**(*handler*, *host=None*, *core_name=None*, *verbose=False*)

MASTER ONLY re-loads the handler config XML file. This command can only be run if the minion is a 'master' type

handler [str] The name of the data import handler.

host [str (None)] The solr host to query. __opts__['host'] is default.

core [str (None)] The core the handler belongs to.

verbose [boolean (False)] Run the command with verbose output.

Return : dict<str,obj>:

```
{'success':boolean, 'data':dict, 'errors':list, 'warnings':list}
```

CLI Example:

```
salt '*' solr.reload_import_config dataimport None music {'clean':True}
```

`salt.modules.solr.`**`replication_details`**(*host=None*, *core_name=None*)

Get the full replication details.

host [str (None)] The solr host to query. __opts__['host'] is default.

core_name [str (None)] The name of the solr core if using cores. Leave this blank if you are not using cores or if you want to check all cores.

Return : dict<str,obj>:

```
{'success':boolean, 'data':dict, 'errors':list, 'warnings':list}
```

CLI Example:

```
salt '*' solr.replication_details music
```

`salt.modules.solr.`**`set_is_polling`**(*polling*, *host=None*, *core_name=None*)

SLAVE CALL Prevent the slaves from polling the master for updates.

polling [boolean] True will enable polling. False will disable it.

host [str (None)] The solr host to query. __opts__['host'] is default.

core_name [str (None)] The name of the solr core if using cores. Leave this blank if you are not using cores or if you want to check all cores.

Return : dict<str,obj>:

```
{'success':boolean, 'data':dict, 'errors':list, 'warnings':list}
```

CLI Example:

```
salt '*' solr.set_is_polling False
```

`salt.modules.solr.`**`set_replication_enabled`**(*status*, *host=None*, *core_name=None*)

MASTER ONLY Sets the master to ignore poll requests from the slaves. Useful when you don't want the slaves replicating during indexing or when clearing the index.

status [boolean] Sets the replication status to the specified state.

host [str (None)] The solr host to query. __opts__['host'] is default.

core_name [str (None)] The name of the solr core if using cores. Leave this blank if you are not using cores or if you want to set the status on all cores.

Return : dict<str,obj>:

```
{'success':boolean, 'data':dict, 'errors':list, 'warnings':list}
```

CLI Example:

```
salt '*' solr.set_replication_enabled false, None, music
```

`salt.modules.solr.`**`signal`**(*signal=None*)

Signals Apache Solr to start, stop, or restart. Obviously this is only going to work if the minion resides on the solr host. Additionally Solr doesn't ship with an init script so one must be created.

signal [str (None)] The command to pass to the apache solr init valid values are 'start', 'stop', and 'restart'

CLI Example:

```
salt '*' solr.signal restart
```

`salt.modules.solr.`**`version`**`(`*core_name=None*`)`

Gets the solr version for the core specified. You should specify a core here as all the cores will run under the same servlet container and so will all have the same version.

core_name [str (None)] The name of the solr core if using cores. Leave this blank if you are not using cores or if you want to check all cores.

Return : dict<str,obj>:

```
{'success':boolean, 'data':dict, 'errors':list, 'warnings':list}
```

CLI Example:

```
salt '*' solr.version
```

61.152 salt.modules.sqlite3

Support for SQLite3

`salt.modules.sqlite3.`**`fetch`**`(`*db=None*, *sql=None*`)`

Retrieve data from an sqlite3 db (returns all rows, be careful!)

CLI Example:

```
salt '*' sqlite3.fetch /root/test.db 'SELECT * FROM test;'
```

`salt.modules.sqlite3.`**`indexes`**`(`*db=None*`)`

Show all indices in the database, for people with poor spelling skills

CLI Example:

```
salt '*' sqlite3.indexes /root/test.db
```

`salt.modules.sqlite3.`**`indices`**`(`*db=None*`)`

Show all indices in the database

CLI Example:

```
salt '*' sqlite3.indices /root/test.db
```

`salt.modules.sqlite3.`**`modify`**`(`*db=None*, *sql=None*`)`

Issue an SQL query to sqlite3 (with no return data), usually used to modify the database in some way (insert, delete, create, etc)

CLI Example:

```
salt '*' sqlite3.modify /root/test.db 'CREATE TABLE test(id INT, testdata TEXT);'
```

`salt.modules.sqlite3.`**`sqlite_version`**`()`

Return version of sqlite

CLI Example:

```
salt '*' sqlite3.sqlite_version
```

`salt.modules.sqlite3.`**`tables`**`(`*db=None*`)`

Show all tables in the database

CLI Example:

```
salt '*' sqlite3.tables /root/test.db
```

`salt.modules.sqlite3.`**`version`**`()`

Return version of pysqlite

CLI Example:

```
salt '*' sqlite3.version
```

61.153 salt.modules.ssh

Manage client ssh components

`salt.modules.ssh.`**`auth_keys`**`(`*user*, *config='.ssh/authorized_keys'*`)`

Return the authorized keys for the specified user

CLI Example:

```
salt '*' ssh.auth_keys root
```

`salt.modules.ssh.`**`check_key`**`(`*user*, *key*, *enc*, *comment*, *options*, *config='.ssh/authorized_keys'*`)`

Check to see if a key needs updating, returns "update", "add" or "exists"

CLI Example:

```
salt '*' ssh.check_key <user> <key> <enc> <comment> <options>
```

`salt.modules.ssh.`**`check_key_file`**`(`*user*, *source*, *config='.ssh/authorized_keys'*, *saltenv='base'*, *env=None*`)`

Check a keyfile from a source destination against the local keys and return the keys to change

CLI Example:

```
salt '*' root salt://ssh/keyfile
```

`salt.modules.ssh.`**`check_known_host`**`(`*user*, *hostname*, *key=None*, *fingerprint=None*, *config='.ssh/known_hosts'*`)`

Check the record in known_hosts file, either by its value or by fingerprint (it's enough to set up either key or fingerprint, you don't need to set up both).

If provided key or fingerprint doesn't match with stored value, return "update", if no value is found for a given host, return "add", otherwise return "exists".

If neither key, nor fingerprint is defined, then additional validation is not performed.

CLI Example:

```
salt '*' ssh.check_known_host <user> <hostname> key='AAAA...FAaQ=='
```

`salt.modules.ssh.`**`get_known_host`**`(`*user*, *hostname*, *config='.ssh/known_hosts'*`)`

Return information about known host from the configfile, if any. If there is no such key, return None.

CLI Example:

```
salt '*' ssh.get_known_host <user> <hostname>
```

salt.modules.ssh.**host_keys**(*keydir=None*)

Return the minion’s host keys

CLI Example:

```
salt '*' ssh.host_keys
```

salt.modules.ssh.**recv_known_host**(*hostname*, *enc=None*, *port=None*, *hash_hostname=False*)

Retrieve information about host public key from remote server

CLI Example:

```
salt '*' ssh.recv_known_host <hostname> enc=<enc> port=<port>
```

salt.modules.ssh.**rm_auth_key**(*user*, *key*, *config='.ssh/authorized_keys'*)

Remove an authorized key from the specified user’s authorized key file

CLI Example:

```
salt '*' ssh.rm_auth_key <user> <key>
```

salt.modules.ssh.**rm_known_host**(*user*, *hostname*, *config='.ssh/known_hosts'*)

Remove all keys belonging to hostname from a known_hosts file.

CLI Example:

```
salt '*' ssh.rm_known_host <user> <hostname>
```

salt.modules.ssh.**set_auth_key**(*user*, *key*, *enc='ssh-rsa'*, *comment=’‘*, *options=None*, *config='.ssh/authorized_keys'*)

Add a key to the authorized_keys file. The “key” parameter must only be the string of text that is the encoded key. If the key begins with “ssh-rsa” or ends with user@host, remove those from the key before passing it to this function.

CLI Example:

```
salt '*' ssh.set_auth_key <user> '<key>' enc='dsa'
```

salt.modules.ssh.**set_auth_key_from_file**(*user*, *source*, *config='.ssh/authorized_keys'*, *saltenv='base'*, *env=None*)

Add a key to the authorized_keys file, using a file as the source.

CLI Example:

```
salt '*' ssh.set_auth_key_from_file <user>                salt://ssh_keys/<user>.id_rsa.pub
```

salt.modules.ssh.**set_known_host**(*user*, *hostname*, *fingerprint=None*, *port=None*, *enc=None*, *hash_hostname=True*, *config='.ssh/known_hosts'*)

Download SSH public key from remote host “hostname”, optionally validate its fingerprint against “fingerprint” variable and save the record in the known_hosts file.

If such a record does already exists in there, do nothing.

CLI Example:

```
salt '*' ssh.set_known_host <user> fingerprint='xx:xx:..:xx'                enc='ssh-rsa' conf
```

61.154 salt.modules.state

Control the state system on the minion

`salt.modules.state.`**`clear_cache`**`()`

Clear out cached state files, forcing even cache runs to refresh the cache on the next state execution.

Remember that the state cache is completely disabled by default, this execution only applies if cache=True is used in states

CLI Example:

```
salt '*' state.clear_cache
```

`salt.modules.state.`**`high`**`(`*data*, *queue=False*, ***kwargs*`)`

Execute the compound calls stored in a single set of high data This function is mostly intended for testing the state system

CLI Example:

```
salt '*' state.high '{"vim": {"pkg": ["installed"]}}'
```

`salt.modules.state.`**`highstate`**`(`*test=None*, *queue=False*, ***kwargs*`)`

Retrieve the state data from the salt master for this minion and execute it

CLI Example:

```
salt '*' state.highstate

salt '*' state.highstate exclude=sls_to_exclude
salt '*' state.highstate exclude="[{'id': 'id_to_exclude'}, {'sls': 'sls_to_exclude'}]"
```

`salt.modules.state.`**`low`**`(`*data*, *queue=False*, ***kwargs*`)`

Execute a single low data call This function is mostly intended for testing the state system

CLI Example:

```
salt '*' state.low '{"state": "pkg", "fun": "installed", "name": "vi"}'
```

`salt.modules.state.`**`pkg`**`(`*pkg_path*, *pkg_sum*, *hash_type*, *test=False*, ***kwargs*`)`

Execute a packaged state run, the packaged state run will exist in a tarball available locally. This packaged state can be generated using salt-ssh.

CLI Example:

```
salt '*' state.pkg /tmp/state_pkg.tgz
```

`salt.modules.state.`**`running`**`()`

Return a dict of state return data if a state function is already running. This function is used to prevent multiple state calls from being run at the same time.

CLI Example:

```
salt '*' state.running
```

`salt.modules.state.`**`show_highstate`**`(`*queue=False*, ***kwargs*`)`

Retrieve the highstate data from the salt master and display it

CLI Example:

```
salt '*' state.show_highstate
```

`salt.modules.state.`**`show_low_sls`**`(`*mods*, *saltenv='base'*, *test=None*, *queue=False*, *env=None*, ***kwargs*`)`

Display the low data from a specific sls. The default environment is `base`, use `saltenv` (`env` in Salt 0.17.x and older) to specify a different environment.

CLI Example:

```
salt '*' state.show_low_sls foo
```

`salt.modules.state.`**`show_lowstate`**(*queue=False*, ***kwargs*)

List out the low data that will be applied to this minion

CLI Example:

```
salt '*' state.show_lowstate
```

`salt.modules.state.`**`show_sls`**(*mods*, *saltenv='base'*, *test=None*, *queue=False*, *env=None*, ***kwargs*)

Display the state data from a specific sls or list of sls files on the master. The default environment is `base`, use `saltenv` (`env` in Salt 0.17.x and older) to specify a different environment.

This function does not support topfiles. For `top.sls` please use `show_top` instead.

CLI Example:

```
salt '*' state.show_sls core,edit.vim dev
```

`salt.modules.state.`**`show_top`**(*queue=False*, ***kwargs*)

Return the top data that the minion will use for a highstate

CLI Example:

```
salt '*' state.show_top
```

`salt.modules.state.`**`single`**(*fun*, *name*, *test=None*, *queue=False*, ***kwargs*)

Execute a single state function with the named kwargs, returns False if insufficient data is sent to the command

By default, the values of the kwargs will be parsed as YAML. So, you can specify lists values, or lists of single entry key-value maps, as you would in a YAML salt file. Alternatively, JSON format of keyword values is also supported.

CLI Example:

```
salt '*' state.single pkg.installed name=vim
```

`salt.modules.state.`**`sls`**(*mods*, *saltenv='base'*, *test=None*, *exclude=None*, *queue=False*, *env=None*, ***kwargs*)

Execute a set list of state modules from an environment. The default environment is `base`, use `saltenv` (`env` in Salt 0.17.x and older) to specify a different environment

CLI Example:

```
salt '*' state.sls core,edit.vim dev
salt '*' state.sls core exclude="[{'id': 'id_to_exclude'}, {'sls': 'sls_to_exclude'}]"
```

`salt.modules.state.`**`template`**(*tem*, *queue=False*, ***kwargs*)

Execute the information stored in a template file on the minion

CLI Example:

```
salt '*' state.template '<Path to template on the minion>'
```

`salt.modules.state.`**`template_str`**(*tem*, *queue=False*, ***kwargs*)

Execute the information stored in a string from an sls template

CLI Example:

```
salt '*' state.template_str '<Template String>'
```

`salt.modules.state.`**`top`**`(`*topfn*, *test=None*, *queue=False*, ***kwargs*`)`

Execute a specific top file instead of the default

CLI Example:

```
salt '*' state.top reverse_top.sls
salt '*' state.top reverse_top.sls exclude=sls_to_exclude
salt '*' state.top reverse_top.sls exclude="[{'id': 'id_to_exclude'}, {'sls': 'sls_to_exclude'}]
```

61.155 salt.modules.status

Module for returning various status data about a minion. These data can be useful for compiling into stats later.

`salt.modules.status.`**`all_status`**`()`

Return a composite of all status data and info for this minion. Warning: There is a LOT here!

CLI Example:

```
salt '*' status.all_status
```

`salt.modules.status.`**`cpuinfo`**`()`

Return the CPU info for this minion

CLI Example:

```
salt '*' status.cpuinfo
```

`salt.modules.status.`**`cpustats`**`()`

Return the CPU stats for this minion

CLI Example:

```
salt '*' status.cpustats
```

`salt.modules.status.`**`custom`**`()`

Return a custom composite of status data and info for this minion, based on the minion config file. An example config like might be:

```
status.cpustats.custom: [ 'cpu', 'ctxt', 'btime', 'processes' ]
```

Where status refers to status.py, cpustats is the function where we get our data, and custom is this function It is followed by a list of keys that we want returned.

This function is meant to replace all_status(), which returns anything and everything, which we probably don't want.

By default, nothing is returned. Warning: Depending on what you include, there can be a LOT here!

CLI Example:

```
salt '*' status.custom
```

`salt.modules.status.`**`diskstats`**`()`

Return the disk stats for this minion

CLI Example:

```
salt '*' status.diskstats
```

`salt.modules.status.`**`diskusage`**`(`**args*`)`

Return the disk usage for this minion

Usage:

```
salt '*' status.diskusage [paths and/or filesystem types]
```

CLI Example:

```
salt '*' status.diskusage         # usage for all filesystems
salt '*' status.diskusage / /tmp  # usage for / and /tmp
salt '*' status.diskusage ext?    # usage for ext[234] filesystems
salt '*' status.diskusage / ext?  # usage for / and all ext filesystems
```

`salt.modules.status.`**`loadavg`**`()`

Return the load averages for this minion

CLI Example:

```
salt '*' status.loadavg
```

`salt.modules.status.`**`meminfo`**`()`

Return the CPU stats for this minion

CLI Example:

```
salt '*' status.meminfo
```

`salt.modules.status.`**`netdev`**`()`

Return the network device stats for this minion

CLI Example:

```
salt '*' status.netdev
```

`salt.modules.status.`**`netstats`**`()`

Return the network stats for this minion

CLI Example:

```
salt '*' status.netstats
```

`salt.modules.status.`**`pid`**`(`*sig*`)`

Return the PID or an empty string if the process is running or not. Pass a signature to use to find the process via ps.

CLI Example:

```
salt '*' status.pid <sig>
```

`salt.modules.status.`**`procs`**`()`

Return the process data

CLI Example:

```
salt '*' status.procs
```

`salt.modules.status.`**`uptime`**`()`

Return the uptime for this minion

CLI Example:

```
salt '*' status.uptime
```

`salt.modules.status.`**`version`**`()`

Return the system version for this minion

CLI Example:

```
salt '*' status.version
```

`salt.modules.status.`**`vmstats`**`()`

Return the virtual memory stats for this minion

CLI Example:

```
salt '*' status.vmstats
```

`salt.modules.status.`**`w`**`()`

Return a list of logged in users for this minion, using the w command

CLI Example:

```
salt '*' status.w
```

61.156 salt.modules.supervisord

Provide the service module for system supervisord or supervisord in a virtualenv

`salt.modules.supervisord.`**`add`**`(`*name*, *user=None*, *conf_file=None*, *bin_env=None*`)`

Activates any updates in config for process/group.

user user to run supervisorctl as

conf_file path to supervisord config file

bin_env path to supervisorctl bin or path to virtualenv with supervisor installed

CLI Example:

```
salt '*' supervisord.add <name>
```

`salt.modules.supervisord.`**`custom`**`(`*command*, *user=None*, *conf_file=None*, *bin_env=None*`)`

Run any custom supervisord command

user user to run supervisorctl as

conf_file path to supervisord config file

bin_env path to supervisorctl bin or path to virtualenv with supervisor installed

CLI Example:

```
salt '*' supervisord.custom "mstop '*gunicorn*'"
```

`salt.modules.supervisord.`**`options`**`(`*name*, *conf_file=None*`)`

New in version 2014.1.0: (Hydrogen)

Read the config file and return the config options for a given process

name Name of the configured process

conf_file path to supervisord config file

CLI Example:

```
salt '*' supervisord.options foo
```

`salt.modules.supervisord.`**`remove`**(*name*, *user=None*, *conf_file=None*, *bin_env=None*)

Removes process/group from active config

user user to run supervisorctl as

conf_file path to supervisord config file

bin_env path to supervisorctl bin or path to virtualenv with supervisor installed

CLI Example:

```
salt '*' supervisord.remove <name>
```

`salt.modules.supervisord.`**`reread`**(*user=None*, *conf_file=None*, *bin_env=None*)

Reload the daemon's configuration files

user user to run supervisorctl as

conf_file path to supervisord config file

bin_env path to supervisorctl bin or path to virtualenv with supervisor installed

CLI Example:

```
salt '*' supervisord.reread
```

`salt.modules.supervisord.`**`restart`**(*name='all'*, *user=None*, *conf_file=None*, *bin_env=None*)

Restart the named service. Process group names should not include a trailing asterisk.

user user to run supervisorctl as

conf_file path to supervisord config file

bin_env path to supervisorctl bin or path to virtualenv with supervisor installed

CLI Example:

```
salt '*' supervisord.restart <service>
salt '*' supervisord.restart <group>:
```

`salt.modules.supervisord.`**`start`**(*name='all'*, *user=None*, *conf_file=None*, *bin_env=None*)

Start the named service. Process group names should not include a trailing asterisk.

user user to run supervisorctl as

conf_file path to supervisord config file

bin_env path to supervisorctl bin or path to virtualenv with supervisor installed

CLI Example:

```
salt '*' supervisord.start <service>
salt '*' supervisord.start <group>:
```

`salt.modules.supervisord.`**`status`**(*name=None*, *user=None*, *conf_file=None*, *bin_env=None*)

List programs and its state

user user to run supervisorctl as

conf_file path to supervisord config file

bin_env path to supervisorctl bin or path to virtualenv with supervisor installed

CLI Example:

```
salt '*' supervisord.status
```

`salt.modules.supervisord.`**`status_raw`**(*name=None*, *user=None*, *conf_file=None*, *bin_env=None*)

Display the raw output of status

user user to run supervisorctl as

conf_file path to supervisord config file

bin_env path to supervisorctl bin or path to virtualenv with supervisor installed

CLI Example:

```
salt '*' supervisord.status_raw
```

`salt.modules.supervisord.`**`stop`**(*name='all'*, *user=None*, *conf_file=None*, *bin_env=None*)

Stop the named service. Process group names should not include a trailing asterisk.

user user to run supervisorctl as

conf_file path to supervisord config file

bin_env path to supervisorctl bin or path to virtualenv with supervisor installed

CLI Example:

```
salt '*' supervisord.stop <service>
salt '*' supervisord.stop <group>:
```

`salt.modules.supervisord.`**`update`**(*user=None*, *conf_file=None*, *bin_env=None*)

Reload config and add/remove as necessary

user user to run supervisorctl as

conf_file path to supervisord config file

bin_env path to supervisorctl bin or path to virtualenv with supervisor installed

CLI Example:

```
salt '*' supervisord.update
```

61.157 salt.modules.svn

Subversion SCM

`salt.modules.svn.`**`add`**(*cwd*, *targets*, *user=None*, *username=None*, *password=None*, **opts*)

Add files to be tracked by the Subversion working-copy checkout

cwd The path to the Subversion repository

targets [None] files and directories to pass to the command as arguments

user [None] Run svn as a user other than what the minion runs as

username [None] Connect to the Subversion server as another user

password [None] Connect to the Subversion server with this password

New in version 0.17.0.

CLI Example:

```
salt '*' svn.add /path/to/repo /path/to/new/file
```

`salt.modules.svn.`**`checkout`**(*cwd*, *remote*, *target=None*, *user=None*, *username=None*, *password=None*, **opts*)

Download a working copy of the remote Subversion repository directory or file

cwd The path to the Subversion repository

remote [None] URL to checkout

target [None] The name to give the file or directory working copy Default: svn uses the remote basename

user [None] Run svn as a user other than what the minion runs as

username [None] Connect to the Subversion server as another user

password [None] Connect to the Subversion server with this password

New in version 0.17.0.

CLI Example:

```
salt '*' svn.checkout /path/to/repo svn://remote/repo
```

`salt.modules.svn.`**`commit`**(*cwd*, *targets=None*, *msg=None*, *user=None*, *username=None*, *password=None*, **opts*)

Commit the current directory, files, or directories to the remote Subversion repository

cwd The path to the Subversion repository

targets [None] files and directories to pass to the command as arguments Default: svn uses '.'

msg [None] Message to attach to the commit log

user [None] Run svn as a user other than what the minion runs as

username [None] Connect to the Subversion server as another user

password [None] Connect to the Subversion server with this password

New in version 0.17.0.

CLI Example:

```
salt '*' svn.commit /path/to/repo
```

`salt.modules.svn.`**`diff`**(*cwd*, *targets=None*, *user=None*, *username=None*, *password=None*, **opts*)

Return the diff of the current directory, files, or directories from the remote Subversion repository

cwd The path to the Subversion repository

targets [None] files and directories to pass to the command as arguments Default: svn uses '.'

user [None] Run svn as a user other than what the minion runs as

username [None] Connect to the Subversion server as another user

password [None] Connect to the Subversion server with this password

New in version 0.17.0.

CLI Example:

```
salt '*' svn.diff /path/to/repo
```

`salt.modules.svn.`**`export`**(*cwd*, *remote*, *target=None*, *user=None*, *username=None*, *password=None*, **opts*)

Create an unversioned copy of a tree.

cwd The path to the Subversion repository

remote [None] URL and path to file or directory checkout

target [None] The name to give the file or directory working copy Default: svn uses the remote basename

user [None] Run svn as a user other than what the minion runs as

username [None] Connect to the Subversion server as another user

password [None] Connect to the Subversion server with this password

New in version 0.17.0.

CLI Example:

```
salt '*' svn.export /path/to/repo svn://remote/repo
```

`salt.modules.svn.`**`info`**(*cwd*, *targets=None*, *user=None*, *username=None*, *password=None*, *fmt='str'*)

Display the Subversion information from the checkout.

cwd The path to the Subversion repository

targets [None] files, directories, and URLs to pass to the command as arguments svn uses '.' by default

user [None] Run svn as a user other than what the minion runs as

username [None] Connect to the Subversion server as another user

password [None] Connect to the Subversion server with this password

New in version 0.17.0.

fmt [str] How to fmt the output from info. (str, xml, list, dict)

CLI Example:

```
salt '*' svn.info /path/to/svn/repo
```

`salt.modules.svn.`**`remove`**(*cwd*, *targets*, *msg=None*, *user=None*, *username=None*, *password=None*, **opts*)

Remove files and directories from the Subversion repository

cwd The path to the Subversion repository

targets [None] files, directories, and URLs to pass to the command as arguments

msg [None] Message to attach to the commit log

user [None] Run svn as a user other than what the minion runs as

username [None] Connect to the Subversion server as another user

password [None] Connect to the Subversion server with this password

New in version 0.17.0.

CLI Example:

```
salt '*' svn.remove /path/to/repo /path/to/repo/remove
```

`salt.modules.svn.`**`status`**(*cwd*, *targets=None*, *user=None*, *username=None*, *password=None*, **opts*)

Display the status of the current directory, files, or directories in the Subversion repository

cwd The path to the Subversion repository

targets [None] files, directories, and URLs to pass to the command as arguments Default: svn uses '.'

user [None] Run svn as a user other than what the minion runs as

username [None] Connect to the Subversion server as another user

password [None] Connect to the Subversion server with this password

New in version 0.17.0.

CLI Example:

```
salt '*' svn.status /path/to/repo
```

`salt.modules.svn.`**`switch`**`(`*cwd*, *remote*, *target=None*, *user=None*, *username=None*, *password=None*, **opts*`)`

New in version 2014.1.0: (Hydrogen)

Switch a working copy of a remote Subversion repository directory

cwd The path to the Subversion repository

remote [None] URL to switch

target [None] The name to give the file or directory working copy Default: svn uses the remote basename

user [None] Run svn as a user other than what the minion runs as

username [None] Connect to the Subversion server as another user

password [None] Connect to the Subversion server with this password

CLI Example:

```
salt '*' svn.switch /path/to/repo svn://remote/repo
```

`salt.modules.svn.`**`update`**`(`*cwd*, *targets=None*, *user=None*, *username=None*, *password=None*, **opts*`)`

Update the current directory, files, or directories from the remote Subversion repository

cwd The path to the Subversion repository

targets [None] files and directories to pass to the command as arguments Default: svn uses '.'

user [None] Run svn as a user other than what the minion runs as

password [None] Connect to the Subversion server with this password

New in version 0.17.0.

username [None] Connect to the Subversion server as another user

CLI Example:

```
salt '*' svn.update /path/to/repo
```

61.158 salt.modules.sysbench

The 'sysbench' module is used to analyse the performance of the minions, right from the master! It measures various system parameters such as CPU, Memory, FileI/O, Threads and Mutex.

`salt.modules.sysbench.`**`cpu`**`()`

Tests for the CPU performance of minions.

CLI Examples:

```
salt '*' sysbench.cpu
```

`salt.modules.sysbench.`**`fileio`**`()`

This tests for the file read and write operations Various modes of operations are

- sequential write
- sequential rewrite
- sequential read
- random read
- random write
- random read and write

The test works with 32 files with each file being 1Gb in size The test consumes a lot of time. Be patient!

CLI Examples:

```
salt '*' sysbench.fileio
```

`salt.modules.sysbench.`**`memory`**`()`

This tests the memory for read and write operations.

CLI Examples:

```
salt '*' sysbench.memory
```

`salt.modules.sysbench.`**`mutex`**`()`

Tests the implementation of mutex

CLI Examples:

```
salt '*' sysbench.mutex
```

`salt.modules.sysbench.`**`ping`**`()`

`salt.modules.sysbench.`**`threads`**`()`

This tests the performance of the processor's scheduler

CLI Example:

```
salt '*' sysbench.threads
```

61.159 salt.modules.sysmod

The sys module provides information about the available functions on the minion

`salt.modules.sysmod.`**`argspec`**`(`*module=''*`)`

Return the argument specification of functions in Salt execution modules.

CLI Example:

```
salt '*' sys.argspec pkg.install
salt '*' sys.argspec sys
salt '*' sys.argspec
```

`salt.modules.sysmod.`**`doc`**`(`**args*`)`

Return the docstrings for all modules. Optionally, specify a module or a function to narrow the selection.

The strings are aggregated into a single document on the master for easy reading.

Multiple modules/functions can be specified.

CLI Example:

```
salt '*' sys.doc
salt '*' sys.doc sys
salt '*' sys.doc sys.doc
salt '*' sys.doc network.traceroute user.info
```

`salt.modules.sysmod.`**`list_functions`**`(`**args*, ***kwargs*`)`

List the functions for all modules. Optionally, specify a module or modules from which to list.

CLI Example:

```
salt '*' sys.list_functions
salt '*' sys.list_functions sys
salt '*' sys.list_functions sys user
```

`salt.modules.sysmod.`**`list_modules`**`()`

List the modules loaded on the minion

CLI Example:

```
salt '*' sys.list_modules
```

`salt.modules.sysmod.`**`reload_modules`**`()`

Tell the minion to reload the execution modules

CLI Example:

```
salt '*' sys.reload_modules
```

61.160 salt.modules.system

Support for reboot, shutdown, etc

`salt.modules.system.`**`halt`**`()`

Halt a running system

CLI Example:

```
salt '*' system.halt
```

`salt.modules.system.`**`init`**`(`*runlevel*`)`

Change the system runlevel on sysV compatible systems

CLI Example:

```
salt '*' system.init 3
```

`salt.modules.system.`**`poweroff`**`()`

Poweroff a running system

CLI Example:

```
salt '*' system.poweroff
```

`salt.modules.system.`**`reboot`**`()`

Reboot the system using the 'reboot' command

CLI Example:

```
salt '*' system.reboot
```

`salt.modules.system.`**`shutdown`**`(`*at_time=None*`)`

Shutdown a running system

CLI Example:

```
salt '*' system.shutdown
```

61.161 salt.modules.systemd

Provide the service module for systemd

`salt.modules.systemd.`**`available`**`(`*name*`)`

Check that the given service is available taking into account template units.

CLI Example:

```
salt '*' service.available sshd
```

`salt.modules.systemd.`**`disable`**`(`*name*, ***kwargs*`)`

Disable the named service to not start when the system boots

CLI Example:

```
salt '*' service.disable <service name>
```

`salt.modules.systemd.`**`disabled`**`(`*name*`)`

Return if the named service is disabled to start on boot

CLI Example:

```
salt '*' service.disabled <service name>
```

`salt.modules.systemd.`**`enable`**`(`*name*, ***kwargs*`)`

Enable the named service to start when the system boots

CLI Example:

```
salt '*' service.enable <service name>
```

`salt.modules.systemd.`**`enabled`**`(`*name*`)`

Return if the named service is enabled to start on boot

CLI Example:

```
salt '*' service.enabled <service name>
```

`salt.modules.systemd.`**`force_reload`**`(`*name*`)`

Force-reload the specified service with systemd

CLI Example:

```
salt '*' service.force_reload <service name>
```

salt.modules.systemd.**get_all**()

Return a list of all available services

CLI Example:

```
salt '*' service.get_all
```

salt.modules.systemd.**get_disabled**()

Return a list of all disabled services

CLI Example:

```
salt '*' service.get_disabled
```

salt.modules.systemd.**get_enabled**()

Return a list of all enabled services

CLI Example:

```
salt '*' service.get_enabled
```

salt.modules.systemd.**missing**(*name*)

The inverse of service.available. Returns `True` if the specified service is not available, otherwise returns `False`.

CLI Example:

```
salt '*' service.missing sshd
```

salt.modules.systemd.**reload_**(*name*)

Reload the specified service with systemd

CLI Example:

```
salt '*' service.reload <service name>
```

salt.modules.systemd.**restart**(*name*)

Restart the specified service with systemd

CLI Example:

```
salt '*' service.restart <service name>
```

salt.modules.systemd.**start**(*name*)

Start the specified service with systemd

CLI Example:

```
salt '*' service.start <service name>
```

salt.modules.systemd.**status**(*name*, *sig=None*)

Return the status for a service via systemd, returns a bool whether the service is running.

CLI Example:

```
salt '*' service.status <service name>
```

salt.modules.systemd.**stop**(*name*)

Stop the specified service with systemd

CLI Example:

```
salt '*' service.stop <service name>
```

`salt.modules.systemd.`**`systemctl_reload`**`()`

Reloads systemctl, an action needed whenever unit files are updated.

CLI Example:

```
salt '*' service.systemctl_reload
```

61.162 salt.modules.test

Module for running arbitrary tests

`salt.modules.test.`**`arg`**`(`**args*, ***kwargs*`)`

Print out the data passed into the function `*args` and `` `kwargs ``, this is used to both test the publication data and cli argument passing, but also to display the information available within the publication data. Returns {"args": args, "kwargs": kwargs}.

CLI Example:

```
salt '*' test.arg 1 "two" 3.1 txt="hello" wow='{a: 1, b: "hello"}'
```

`salt.modules.test.`**`arg_repr`**`(`**args*, ***kwargs*`)`

Print out the data passed into the function `*args` and `` `kwargs ``, this is used to both test the publication data and cli argument passing, but also to display the information available within the publication data. Returns {"args": repr(args), "kwargs": repr(kwargs)}.

CLI Example:

```
salt '*' test.arg_repr 1 "two" 3.1 txt="hello" wow='{a: 1, b: "hello"}'
```

`salt.modules.test.`**`arg_type`**`(`**args*, ***kwargs*`)`

Print out the types of the args and kwargs. This is used to test the types of the args and kwargs passed down to the minion

CLI Example:

```
salt '*' test.arg_type 1 'int'
```

`salt.modules.test.`**`collatz`**`(`*start*`)`

Execute the collatz conjecture from the passed starting number, returns the sequence and the time it took to compute. Used for performance tests.

CLI Example:

```
salt '*' test.collatz 3
```

`salt.modules.test.`**`conf_test`**`()`

Return the value for test.foo in the minion configuration file, or return the default value

CLI Example:

```
salt '*' test.conf_test
```

`salt.modules.test.`**`cross_test`**`(`*func*, *args=None*`)`

Execute a minion function via the __salt__ object in the test module, used to verify that the minion functions can be called via the __salt__ module.

CLI Example:

```
salt '*' test.cross_test file.gid_to_group 0
```

`salt.modules.test.`**`echo`**`(`*text*`)`

Return a string - used for testing the connection

CLI Example:

```
salt '*' test.echo 'foo bar baz quo qux'
```

`salt.modules.test.`**`fib`**`(`*num*`)`

Return a Fibonacci sequence up to the passed number, and the timeit took to compute in seconds. Used for performance tests

CLI Example:

```
salt '*' test.fib 3
```

`salt.modules.test.`**`get_opts`**`()`

Return the configuration options passed to this minion

CLI Example:

```
salt '*' test.get_opts
```

`salt.modules.test.`**`kwarg`**`(`***kwargs*`)`

Print out the data passed into the function `**kwargs`, this is used to both test the publication data and cli kwarg passing, but also to display the information available within the publication data.

CLI Example:

```
salt '*' test.kwarg num=1 txt="two" env='{a: 1, b: "hello"}'
```

`salt.modules.test.`**`not_loaded`**`()`

List the modules that were not loaded by the salt loader system

CLI Example:

```
salt '*' test.not_loaded
```

`salt.modules.test.`**`opts_pkg`**`()`

Return an opts package with the grains and opts for this minion. This is primarily used to create the options used for master side state compiling routines

CLI Example:

```
salt '*' test.opts_pkg
```

`salt.modules.test.`**`outputter`**`(`*data*`)`

Test the outputter, pass in data to return

CLI Example:

```
salt '*' test.outputter foobar
```

`salt.modules.test.`**`ping`**`()`

Just used to make sure the minion is up and responding Return True

CLI Example:

```
salt '*' test.ping
```

`salt.modules.test.`**`provider`**`(`*module*`)`

Pass in a function name to discover what provider is being used

CLI Example:

```
salt '*' test.provider service
```

`salt.modules.test.`**`providers`**`()`

Return a dict of the provider names and the files that provided them

CLI Example:

```
salt '*' test.providers
```

`salt.modules.test.`**`rand_sleep`**`(`*max=60*`)`

Sleep for a random number of seconds, used to test long-running commands and minions returning at differing intervals

CLI Example:

```
salt '*' test.rand_sleep 60
```

`salt.modules.test.`**`retcode`**`(`*code=42*`)`

Test that the returncode system is functioning correctly

CLI Example:

```
salt '*' test.retcode 42
```

`salt.modules.test.`**`sleep`**`(`*length*`)`

Instruct the minion to initiate a process that will sleep for a given period of time.

CLI Example:

```
salt '*' test.sleep 20
```

`salt.modules.test.`**`tty`**`(`*device*, *echo=None*`)`

Echo a string to a specific tty

CLI Example:

```
salt '*' test.tty tty0 'This is a test'
salt '*' test.tty pts3 'This is a test'
```

`salt.modules.test.`**`version`**`()`

Return the version of salt on the minion

CLI Example:

```
salt '*' test.version
```

`salt.modules.test.`**`versions_information`**`()`

Returns versions of components used by salt as a dict

CLI Example:

```
salt '*' test.versions_information
```

`salt.modules.test.`**`versions_report`**`()`

Returns versions of components used by salt

CLI Example:

```
salt '*' test.versions_report
```

61.163 salt.modules.timezone

Module for managing timezone on POSIX-like systems.

`salt.modules.timezone.`**`get_hwclock`**`()`

Get current hardware clock setting (UTC or localtime)

CLI Example:

```
salt '*' timezone.get_hwclock
```

`salt.modules.timezone.`**`get_offset`**`()`

Get current numeric timezone offset from UCT (i.e. -0700)

CLI Example:

```
salt '*' timezone.get_offset
```

`salt.modules.timezone.`**`get_zone`**`()`

Get current timezone (i.e. America/Denver)

CLI Example:

```
salt '*' timezone.get_zone
```

`salt.modules.timezone.`**`get_zonecode`**`()`

Get current timezone (i.e. PST, MDT, etc)

CLI Example:

```
salt '*' timezone.get_zonecode
```

`salt.modules.timezone.`**`set_hwclock`**`(`*clock*`)`

Sets the hardware clock to be either UTC or localtime

CLI Example:

```
salt '*' timezone.set_hwclock UTC
```

`salt.modules.timezone.`**`set_zone`**`(`*timezone*`)`

Unlinks, then symlinks /etc/localtime to the set timezone.

The timezone is crucial to several system processes, each of which SHOULD be restarted (for instance, whatever you system uses as its cron and syslog daemons). This will not be magically done for you!

CLI Example:

```
salt '*' timezone.set_zone 'America/Denver'
```

`salt.modules.timezone.`**`zone_compare`**`(`*timezone*`)`

Checks the md5sum between the given timezone, and the one set in /etc/localtime. Returns True if they match, and False if not. Mostly useful for running state checks.

CLI Example:

```
salt '*' timezone.zone_compare 'America/Denver'
```

61.164 salt.modules.tls

A salt module for SSL/TLS. Can create a Certificate Authority (CA) or use Self-Signed certificates.

depends

- PyOpenSSL Python module

configuration Add the following values in /etc/salt/minion for the CA module to function properly:

```
ca.cert_base_path: '/etc/pki'
```

`salt.modules.tls.`**`create_ca`**(*ca_name*, *bits=2048*, *days=365*, *CN='localhost'*, *C='US'*, *ST='Utah'*, *L='Salt Lake City'*, *O='SaltStack'*, *OU=None*, *emailAddress='xyz@pdq.net'*)

Create a Certificate Authority (CA)

ca_name name of the CA

bits number of RSA key bits, default is 2048

days number of days the CA will be valid, default is 365

CN common name in the request, default is "localhost"

C country, default is "US"

ST state, default is "Utah"

L locality, default is "Centerville", the city where SaltStack originated

O organization, default is "SaltStack"

OU organizational unit, default is None

emailAddress email address for the CA owner, default is 'xyz@pdq.net'

Writes out a CA certificate based upon defined config values. If the file already exists, the function just returns assuming the CA certificate already exists.

If the following values were set:

```
ca.cert_base_path='/etc/pki'
ca_name='koji'
```

the resulting CA, and corresponding key, would be written in the following location:

```
/etc/pki/koji/koji_ca_cert.crt
/etc/pki/koji/koji_ca_cert.key
```

CLI Example:

```
salt '*' tls.create_ca test_ca
```

`salt.modules.tls.`**`create_ca_signed_cert`**(*ca_name*, *CN*, *days=365*)

Create a Certificate (CERT) signed by a named Certificate Authority (CA)

ca_name name of the CA

CN common name matching the certificate signing request

days number of days certificate is valid, default is 365 (1 year)

Writes out a Certificate (CERT) If the file already exists, the function just returns assuming the CERT already exists.

The CN *must* match an existing CSR generated by create_csr. If it does not, this method does nothing.

If the following values were set:

```
ca.cert_base_path='/etc/pki'
ca_name='koji'
CN='test.egavas.org'
```

the resulting signed certificate would be written in the following location:

```
/etc/pki/koji/certs/test.egavas.org.crt
```

CLI Example:

```
salt '*' tls.create_ca_signed_cert test localhost
```

`salt.modules.tls.`**`create_csr`**(*ca_name*, *bits=2048*, *CN='localhost'*, *C='US'*, *ST='Utah'*, *L='Salt Lake City'*, *O='SaltStack'*, *OU=None*, *emailAddress='xyz@pdq.net'*)

Create a Certificate Signing Request (CSR) for a particular Certificate Authority (CA)

ca_name name of the CA

bits number of RSA key bits, default is 2048

CN common name in the request, default is "localhost"

C country, default is "US"

ST state, default is "Utah"

L locality, default is "Centerville", the city where SaltStack originated

O organization, default is "SaltStack" NOTE: Must the same as CA certificate or an error will be raised

OU organizational unit, default is None

emailAddress email address for the request, default is 'xyz@pdq.net'

Writes out a Certificate Signing Request (CSR) If the file already exists, the function just returns assuming the CSR already exists.

If the following values were set:

```
ca.cert_base_path='/etc/pki'
ca_name='koji'
CN='test.egavas.org'
```

the resulting CSR, and corresponding key, would be written in the following location:

```
/etc/pki/koji/certs/test.egavas.org.csr
/etc/pki/koji/certs/test.egavas.org.key
```

CLI Example:

```
salt '*' tls.create_csr test
```

`salt.modules.tls.`**`create_pkcs12`**(*ca_name*, *CN*, *passphrase=''*)

Create a PKCS#12 browser certificate for a particular Certificate (CN)

ca_name name of the CA

CN common name matching the certificate signing request

passphrase used to unlock the PKCS#12 certificate when loaded into the browser

If the following values were set:

```
ca.cert_base_path='/etc/pki'
ca_name='koji'
CN='test.egavas.org'
```

the resulting signed certificate would be written in the following location:

```
/etc/pki/koji/certs/test.egavas.org.p12
```

CLI Example:

```
salt '*' tls.create_pkcs12 test localhost
```

`salt.modules.tls.`**`create_self_signed_cert`**(*tls_dir='tls', bits=2048, days=365, CN='localhost', C='US', ST='Utah', L='Salt Lake City', O='SaltStack', OU=None, emailAddress='xyz@pdq.net'*)

Create a Self-Signed Certificate (CERT)

tls_dir location appended to the ca.cert_base_path, default is 'tls'

bits number of RSA key bits, default is 2048

CN common name in the request, default is "localhost"

C country, default is "US"

ST state, default is "Utah"

L locality, default is "Centerville", the city where SaltStack originated

O organization, default is "SaltStack" NOTE: Must the same as CA certificate or an error will be raised

OU organizational unit, default is None

emailAddress email address for the request, default is 'xyz@pdq.net'

Writes out a Self-Signed Certificate (CERT). If the file already exists, the function just returns.

If the following values were set:

```
ca.cert_base_path='/etc/pki'
tls_dir='koji'
CN='test.egavas.org'
```

the resulting CERT, and corresponding key, would be written in the following location:

```
/etc/pki/koji/certs/test.egavas.org.crt
/etc/pki/koji/certs/test.egavas.org.key
```

CLI Example:

```
salt '*' tls.create_self_signed_cert
```

Passing options from the command line:

```
salt 'minion' tls.create_self_signed_cert CN='test.mysite.org'
```

61.165 salt.modules.tomcat

Support for Tomcat

This module uses the manager webapp to manage Apache tomcat webapps If the manager webapp is not configured some of the functions won't work

The following grains/pillar should be set:

```
tomcat-manager.user: admin user name
tomcat-manager.passwd: password
```

and also configure a user in the conf/tomcat-users.xml file:

```
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
    <role rolename="manager-script"/>
    <user username="tomcat" password="tomcat" roles="manager-script"/>
</tomcat-users>
```

Notes:

- More information about tomcat manager: http://tomcat.apache.org/tomcat-7.0-doc/manager-howto.html
- if you use only this module for deployments you've might want to strict access to the manager only from localhost for more info: http://tomcat.apache.org/tomcat-7.0-doc/manager-howto.html#Configuring_Manager_Application_Access
- Tested on:

  JVM Vendor: Sun Microsystems Inc.

  JVM Version: 1.6.0_43-b01

  OS Architecture: amd64

  OS Name: Linux

  OS Version: 2.6.32-358.el6.x86_64

  Tomcat Version: Apache Tomcat/7.0.37

`salt.modules.tomcat.`**`deploy_war`**(*war*, *context*, *force='no'*, *url='http://localhost:8080/manager'*, *saltenv='base'*, *timeout=180*, *env=None*)

Deploy a WAR file

war absolute path to WAR file (should be accessible by the user running tomcat) or a path supported by the salt.modules.cp.get_file function

context the context path to deploy

force [False] set True to deploy the webapp even one is deployed in the context

url [http://localhost:8080/manager] the URL of the server manager webapp

saltenv [base] the environment for WAR file in used by salt.modules.cp.get_url function

timeout [180] timeout for HTTP request

CLI Examples:

cp module

```
salt '*' tomcat.deploy_war salt://application.war /api
salt '*' tomcat.deploy_war salt://application.war /api no
salt '*' tomcat.deploy_war salt://application.war /api yes http://localhost:8080/manager
```

minion local file system

```
salt '*' tomcat.deploy_war /tmp/application.war /api
salt '*' tomcat.deploy_war /tmp/application.war /api no
salt '*' tomcat.deploy_war /tmp/application.war /api yes http://localhost:8080/manager
```

`salt.modules.tomcat.`**`fullversion`**`()`

Return all server information from catalina.sh version

CLI Example:

```
salt '*' tomcat.fullversion
```

`salt.modules.tomcat.`**`leaks`**`(`*url='http://localhost:8080/manager'*, *timeout=180*`)`

Find memory leaks in tomcat

url [http://localhost:8080/manager] the URL of the server manager webapp

timeout [180] timeout for HTTP request

CLI Examples:

```
salt '*' tomcat.leaks
```

`salt.modules.tomcat.`**`ls`**`(`*url='http://localhost:8080/manager'*, *timeout=180*`)`

list all the deployed webapps

url [http://localhost:8080/manager] the URL of the server manager webapp

timeout [180] timeout for HTTP request

CLI Examples:

```
salt '*' tomcat.ls
salt '*' tomcat.ls http://localhost:8080/manager
```

`salt.modules.tomcat.`**`passwd`**`(`*passwd*, *user='‘*, *alg='md5'*, *realm=None*`)`

This function replaces the $CATALINS_HOME/bin/digest.sh script convert a clear-text password to the $CATALINA_BASE/conf/tomcat-users.xml format

CLI Examples:

```
salt '*' tomcat.passwd secret
salt '*' tomcat.passwd secret tomcat sha1
salt '*' tomcat.passwd secret tomcat sha1 'Protected Realm'
```

`salt.modules.tomcat.`**`reload_`**`(`*app*, *url='http://localhost:8080/manager'*, *timeout=180*`)`

Reload the webapp

app the webapp context path

url [http://localhost:8080/manager] the URL of the server manager webapp

timeout [180] timeout for HTTP request

CLI Examples:

```
salt '*' tomcat.reload /jenkins
salt '*' tomcat.reload /jenkins http://localhost:8080/manager
```

`salt.modules.tomcat.`**`serverinfo`**`(`*url='http://localhost:8080/manager'*, *timeout=180*`)`

return detailes about the server

url [http://localhost:8080/manager] the URL of the server manager webapp

timeout [180] timeout for HTTP request

CLI Examples:

```
salt '*' tomcat.serverinfo
salt '*' tomcat.serverinfo http://localhost:8080/manager
```

`salt.modules.tomcat.`**`sessions`**(*app*, *url='http://localhost:8080/manager'*, *timeout=180*)

return the status of the webapp sessions

app the webapp context path

url [http://localhost:8080/manager] the URL of the server manager webapp

timeout [180] timeout for HTTP request

CLI Examples:

```
salt '*' tomcat.sessions /jenkins
salt '*' tomcat.sessions /jenkins http://localhost:8080/manager
```

`salt.modules.tomcat.`**`signal`**(*signal=None*)

Signals catalina to start, stop, securestart, forcestop.

CLI Example:

```
salt '*' tomcat.signal start
```

`salt.modules.tomcat.`**`start`**(*app*, *url='http://localhost:8080/manager'*, *timeout=180*)

Start the webapp

app the webapp context path

url [http://localhost:8080/manager] the URL of the server manager webapp

timeout timeout for HTTP request

CLI Examples:

```
salt '*' tomcat.start /jenkins
salt '*' tomcat.start /jenkins http://localhost:8080/manager
```

`salt.modules.tomcat.`**`status`**(*url='http://localhost:8080/manager'*, *timeout=180*)

Used to test if the tomcat manager is up

url [http://localhost:8080/manager] the URL of the server manager webapp

timeout [180] timeout for HTTP request

CLI Examples:

```
salt '*' tomcat.status
salt '*' tomcat.status http://localhost:8080/manager
```

`salt.modules.tomcat.`**`status_webapp`**(*app*, *url='http://localhost:8080/manager'*, *timeout=180*)

return the status of the webapp (stopped | running | missing)

app the webapp context path

url [http://localhost:8080/manager] the URL of the server manager webapp

timeout [180] timeout for HTTP request

CLI Examples:

```
salt '*' tomcat.status_webapp /jenkins
salt '*' tomcat.status_webapp /jenkins http://localhost:8080/manager
```

`salt.modules.tomcat.`**`stop`**(*app*, *url='http://localhost:8080/manager'*, *timeout=180*)

Stop the webapp

app the webapp context path

url [http://localhost:8080/manager] the URL of the server manager webapp

timeout [180] timeout for HTTP request

CLI Examples:

```
salt '*' tomcat.stop /jenkins
salt '*' tomcat.stop /jenkins http://localhost:8080/manager
```

`salt.modules.tomcat.`**`undeploy`**(*app*, *url='http://localhost:8080/manager'*, *timeout=180*)

Undeploy a webapp

app the webapp context path

url [http://localhost:8080/manager] the URL of the server manager webapp

timeout [180] timeout for HTTP request

CLI Examples:

```
salt '*' tomcat.undeploy /jenkins
salt '*' tomcat.undeploy /jenkins http://localhost:8080/manager
```

`salt.modules.tomcat.`**`version`**()

Return server version from catalina.sh version

CLI Example:

```
salt '*' tomcat.version
```

61.166 salt.modules.upstart

Module for the management of upstart systems. The Upstart system only supports service starting, stopping and restarting.

Currently (as of Ubuntu 12.04) there is no tool available to disable Upstart services (like update-rc.d). This[1] is the recommended way to disable an Upstart service. So we assume that all Upstart services that have not been disabled in this manner are enabled.

But this is broken because we do not check to see that the dependent services are enabled. Otherwise we would have to do something like parse the output of "initctl show-config" to determine if all service dependencies are enabled to start on boot. For example, see the "start on" condition for the lightdm service below[2]. And this would be too hard. So we wait until the upstart developers have solved this problem. :) This is to say that an Upstart service that is enabled may not really be enabled.

Also, when an Upstart service is enabled, should the dependent services be enabled too? Probably not. But there should be a notice about this, at least.

[1] http://upstart.ubuntu.com/cookbook/#disabling-a-job-from-automatically-starting

[2] example upstart configuration file:

```
lightdm
emits login-session-start
emits desktop-session-start
emits desktop-shutdown
```

```
start on ((((filesystem and runlevel [!06]) and started dbus) and (drm-device-added card0 PRIMARY_DEV
stop on runlevel [016]
```

> **Warning:** This module should not be used on Red Hat systems. For these, the `rh_service` module should be used, as it supports the hybrid upstart/sysvinit system used in RHEL/CentOS 6.

`salt.modules.upstart.`**`available`**(*name*)

Returns `True` if the specified service is available, otherwise returns `False`.

CLI Example:

```
salt '*' service.available sshd
```

`salt.modules.upstart.`**`disable`**(*name*, ***kwargs*)

Disable the named service from starting on boot

CLI Example:

```
salt '*' service.disable <service name>
```

`salt.modules.upstart.`**`disabled`**(*name*)

Check to see if the named service is disabled to start on boot

CLI Example:

```
salt '*' service.disabled <service name>
```

`salt.modules.upstart.`**`enable`**(*name*, ***kwargs*)

Enable the named service to start at boot

CLI Example:

```
salt '*' service.enable <service name>
```

`salt.modules.upstart.`**`enabled`**(*name*)

Check to see if the named service is enabled to start on boot

CLI Example:

```
salt '*' service.enabled <service name>
```

`salt.modules.upstart.`**`force_reload`**(*name*)

Force-reload the named service

CLI Example:

```
salt '*' service.force_reload <service name>
```

`salt.modules.upstart.`**`full_restart`**(*name*)

Do a full restart (stop/start) of the named service

CLI Example:

```
salt '*' service.full_restart <service name>
```

`salt.modules.upstart.`**`get_all`**()

Return all installed services

CLI Example:

```
salt '*' service.get_all
```

`salt.modules.upstart.`**`get_disabled`**`()`

Return the disabled services

CLI Example:

```
salt '*' service.get_disabled
```

`salt.modules.upstart.`**`get_enabled`**`()`

Return the enabled services

CLI Example:

```
salt '*' service.get_enabled
```

`salt.modules.upstart.`**`missing`**`(`*name*`)`

The inverse of service.available. Returns `True` if the specified service is not available, otherwise returns `False`.

CLI Example:

```
salt '*' service.missing sshd
```

`salt.modules.upstart.`**`reload_`**`(`*name*`)`

Reload the named service

CLI Example:

```
salt '*' service.reload <service name>
```

`salt.modules.upstart.`**`restart`**`(`*name*`)`

Restart the named service

CLI Example:

```
salt '*' service.restart <service name>
```

`salt.modules.upstart.`**`start`**`(`*name*`)`

Start the specified service

CLI Example:

```
salt '*' service.start <service name>
```

`salt.modules.upstart.`**`status`**`(`*name*, *sig=None*`)`

Return the status for a service, returns a bool whether the service is running.

CLI Example:

```
salt '*' service.status <service name>
```

`salt.modules.upstart.`**`stop`**`(`*name*`)`

Stop the specified service

CLI Example:

```
salt '*' service.stop <service name>
```

61.167 salt.modules.useradd

Manage users with the useradd command

`salt.modules.useradd.`**`add`**(*name*, *uid=None*, *gid=None*, *groups=None*, *home=None*, *shell=None*, *unique=True*, *system=False*, *fullname='‘*, *roomnumber='‘*, *workphone='‘*, *homephone='‘*, *createhome=True*)

Add a user to the minion

CLI Example:

```
salt '*' user.add name <uid> <gid> <groups> <home> <shell>
```

`salt.modules.useradd.`**`chfullname`**(*name*, *fullname*)

Change the user's Full Name

CLI Example:

```
salt '*' user.chfullname foo "Foo Bar"
```

`salt.modules.useradd.`**`chgid`**(*name*, *gid*)

Change the default group of the user

CLI Example:

```
salt '*' user.chgid foo 4376
```

`salt.modules.useradd.`**`chgroups`**(*name*, *groups*, *append=False*)

Change the groups this user belongs to, add append to append the specified groups

CLI Example:

```
salt '*' user.chgroups foo wheel,root True
```

`salt.modules.useradd.`**`chhome`**(*name*, *home*, *persist=False*)

Change the home directory of the user, pass true for persist to copy files to the new home dir

CLI Example:

```
salt '*' user.chhome foo /home/users/foo True
```

`salt.modules.useradd.`**`chhomephone`**(*name*, *homephone*)

Change the user's Home Phone

CLI Example:

```
salt '*' user.chhomephone foo "7735551234"
```

`salt.modules.useradd.`**`chroomnumber`**(*name*, *roomnumber*)

Change the user's Room Number

CLI Example:

```
salt '*' user.chroomnumber foo 123
```

`salt.modules.useradd.`**`chshell`**(*name*, *shell*)

Change the default shell of the user

CLI Example:

```
salt '*' user.chshell foo /bin/zsh
```

`salt.modules.useradd.`**`chuid`**(*name*, *uid*)

Change the uid for a named user

CLI Example:

```
salt '*' user.chuid foo 4376
```

`salt.modules.useradd.`**`chworkphone`**`(`*name*, *workphone*`)`
Change the user's Work Phone

CLI Example:

```
salt '*' user.chworkphone foo "7735550123"
```

`salt.modules.useradd.`**`delete`**`(`*name*, *remove=False*, *force=False*`)`
Remove a user from the minion

CLI Example:

```
salt '*' user.delete name remove=True force=True
```

`salt.modules.useradd.`**`getent`**`(`*refresh=False*`)`
Return the list of all info for all users

CLI Example:

```
salt '*' user.getent
```

`salt.modules.useradd.`**`info`**`(`*name*`)`
Return user information

CLI Example:

```
salt '*' user.info root
```

`salt.modules.useradd.`**`list_groups`**`(`*name*`)`
Return a list of groups the named user belongs to

CLI Example:

```
salt '*' user.list_groups foo
```

`salt.modules.useradd.`**`list_users`**`()`
Return a list of all users

CLI Example:

```
salt '*' user.list_users
```

61.168 salt.modules.virt

Work with virtual machines managed by libvirt

depends libvirt Python module

`salt.modules.virt.`**`create`**`(`*vm_*`)`
Start a defined domain

CLI Example:

```
salt '*' virt.create <vm name>
```

`salt.modules.virt.`**`create_xml_path`**`(`*path*`)`
Start a domain based on the XML-file path passed to the function

CLI Example:

```
salt '*' virt.create_xml_path <path to XML file on the node>
```

salt.modules.virt.**create_xml_str**(*xml*)
Start a domain based on the XML passed to the function

CLI Example:

```
salt '*' virt.create_xml_str <XML in string format>
```

salt.modules.virt.**ctrl_alt_del**(*vm_*)
Sends CTRL+ALT+DEL to a VM

CLI Example:

```
salt '*' virt.ctrl_alt_del <vm name>
```

salt.modules.virt.**define_vol_xml_path**(*path*)
Define a volume based on the XML-file path passed to the function

CLI Example:

```
salt '*' virt.define_vol_xml_path <path to XML file on the node>
```

salt.modules.virt.**define_vol_xml_str**(*xml*)
Define a volume based on the XML passed to the function

CLI Example:

```
salt '*' virt.define_vol_xml_str <XML in string format>
```

salt.modules.virt.**define_xml_path**(*path*)
Define a domain based on the XML-file path passed to the function

CLI Example:

```
salt '*' virt.define_xml_path <path to XML file on the node>
```

salt.modules.virt.**define_xml_str**(*xml*)
Define a domain based on the XML passed to the function

CLI Example:

```
salt '*' virt.define_xml_str <XML in string format>
```

salt.modules.virt.**destroy**(*vm_*)
Hard power down the virtual machine, this is equivalent to pulling the power

CLI Example:

```
salt '*' virt.destroy <vm name>
```

salt.modules.virt.**freecpu**()
Return an int representing the number of unallocated cpus on this hypervisor

CLI Example:

```
salt '*' virt.freecpu
```

salt.modules.virt.**freemem**()
Return an int representing the amount of memory that has not been given to virtual machines on this node

CLI Example:

```
salt '*' virt.freemem
```

salt.modules.virt.**full_info**()

Return the node_info, vm_info and freemem

CLI Example:

```
salt '*' virt.full_info
```

salt.modules.virt.**get_disks**(*vm_*)

Return the disks of a named vm

CLI Example:

```
salt '*' virt.get_disks <vm name>
```

salt.modules.virt.**get_graphics**(*vm_*)

Returns the information on vnc for a given vm

CLI Example:

```
salt '*' virt.get_graphics <vm name>
```

salt.modules.virt.**get_macs**(*vm_*)

Return a list off MAC addresses from the named vm

CLI Example:

```
salt '*' virt.get_macs <vm name>
```

salt.modules.virt.**get_nics**(*vm_*)

Return info about the network interfaces of a named vm

CLI Example:

```
salt '*' virt.get_nics <vm name>
```

salt.modules.virt.**get_profiles**(*hypervisor=None*)

Return the virt profiles for hypervisor.

Currently there are profiles for:

- nic
- disk

CLI Example:

```
salt '*' virt.get_profiles
salt '*' virt.get_profiles hypervisor=esxi
```

salt.modules.virt.**get_xml**(*vm_*)

Returns the XML for a given vm

CLI Example:

```
salt '*' virt.get_xml <vm name>
```

salt.modules.virt.**init**(*name*, *cpu*, *mem*, *image=None*, *nic='default'*, *hypervisor='kvm'*, *start=True*, *disk='default'*, *saltenv='base'*, ***kwargs*)

Initialize a new vm

CLI Example:

```
salt 'hypervisor' virt.init vm_name 4 512 salt://path/to/image.raw
salt 'hypervisor' virt.init vm_name 4 512 nic=profile disk=profile
```

`salt.modules.virt.`**`is_hyper`**`()`

Returns a bool whether or not this node is a hypervisor of any kind

CLI Example:

```
salt '*' virt.is_hyper
```

`salt.modules.virt.`**`is_kvm_hyper`**`()`

Returns a bool whether or not this node is a KVM hypervisor

CLI Example:

```
salt '*' virt.is_kvm_hyper
```

`salt.modules.virt.`**`is_xen_hyper`**`()`

Returns a bool whether or not this node is a XEN hypervisor

CLI Example:

```
salt '*' virt.is_xen_hyper
```

`salt.modules.virt.`**`list_active_vms`**`()`

Return a list of names for active virtual machine on the minion

CLI Example:

```
salt '*' virt.list_active_vms
```

`salt.modules.virt.`**`list_inactive_vms`**`()`

Return a list of names for inactive virtual machine on the minion

CLI Example:

```
salt '*' virt.list_inactive_vms
```

`salt.modules.virt.`**`list_vms`**`()`

Return a list of virtual machine names on the minion

CLI Example:

```
salt '*' virt.list_vms
```

`salt.modules.virt.`**`migrate`**`(`*vm_*, *target*, *ssh=False*`)`

Shared storage migration

CLI Example:

```
salt '*' virt.migrate <vm name> <target hypervisor>
```

`salt.modules.virt.`**`migrate_non_shared`**`(`*vm_*, *target*, *ssh=False*`)`

Attempt to execute non-shared storage "all" migration

CLI Example:

```
salt '*' virt.migrate_non_shared <vm name> <target hypervisor>
```

`salt.modules.virt.`**`migrate_non_shared_inc`**`(`*vm_*, *target*, *ssh=False*`)`

Attempt to execute non-shared storage "all" migration

CLI Example:

```
salt '*' virt.migrate_non_shared_inc <vm name> <target hypervisor>
```

salt.modules.virt.**node_info**()

Return a dict with information about this node

CLI Example:

```
salt '*' virt.node_info
```

salt.modules.virt.**pause**(*vm_*)

Pause the named vm

CLI Example:

```
salt '*' virt.pause <vm name>
```

salt.modules.virt.**purge**(*vm_*, *dirs=False*)

Recursively destroy and delete a virtual machine, pass True for dir's to also delete the directories containing the virtual machine disk images - USE WITH EXTREME CAUTION!

CLI Example:

```
salt '*' virt.purge <vm name>
```

salt.modules.virt.**reboot**(*vm_*)

Reboot a domain via ACPI request

CLI Example:

```
salt '*' virt.reboot <vm name>
```

salt.modules.virt.**reset**(*vm_*)

Reset a VM by emulating the reset button on a physical machine

CLI Example:

```
salt '*' virt.reset <vm name>
```

salt.modules.virt.**resume**(*vm_*)

Resume the named vm

CLI Example:

```
salt '*' virt.resume <vm name>
```

salt.modules.virt.**seed_non_shared_migrate**(*disks*, *force=False*)

Non shared migration requires that the disks be present on the migration destination, pass the disks information via this function, to the migration destination before executing the migration.

CLI Example:

```
salt '*' virt.seed_non_shared_migrate <disks>
```

salt.modules.virt.**set_autostart**(*vm_*, *state='on'*)

Set the autostart flag on a VM so that the VM will start with the host system on reboot.

CLI Example:

```
salt "*" virt.set_autostart <vm name> <on | off>
```

salt.modules.virt.**setmem**(*vm_*, *memory*, *config=False*)

Changes the amount of memory allocated to VM. The VM must be shutdown for this to work.

memory is to be specified in MB If config is True then we ask libvirt to modify the config as well

CLI Example:

```
salt '*' virt.setmem myvm 768
```

`salt.modules.virt.`**`setvcpus`**`(`*vm_*, *vcpus*, *config=False*`)`

Changes the amount of vcpus allocated to VM. The VM must be shutdown for this to work.

vcpus is an int representing the number to be assigned If config is True then we ask libvirt to modify the config as well

CLI Example:

```
salt '*' virt.setvcpus myvm 2
```

`salt.modules.virt.`**`shutdown`**`(`*vm_*`)`

Send a soft shutdown signal to the named vm

CLI Example:

```
salt '*' virt.shutdown <vm name>
```

`salt.modules.virt.`**`start`**`(`*vm_*`)`

Alias for the obscurely named 'create' function

CLI Example:

```
salt '*' virt.start <vm name>
```

`salt.modules.virt.`**`stop`**`(`*vm_*`)`

Alias for the obscurely named 'destroy' function

CLI Example:

```
salt '*' virt.stop <vm name>
```

`salt.modules.virt.`**`undefine`**`(`*vm_*`)`

Remove a defined vm, this does not purge the virtual machine image, and this only works if the vm is powered down

CLI Example:

```
salt '*' virt.undefine <vm name>
```

`salt.modules.virt.`**`virt_type`**`()`

Returns the virtual machine type as a string

CLI Example:

```
salt '*' virt.virt_type
```

`salt.modules.virt.`**`vm_cputime`**`(`*vm_=None*`)`

Return cputime used by the vms on this hyper in a list of dicts:

```
[
    'your-vm': {
        'cputime' <int>
        'cputime_percent' <int>
        },
    ...
    ]
```

If you pass a VM name in as an argument then it will return info for just the named VM, otherwise it will return all VMs.

CLI Example:

```
salt '*' virt.vm_cputime
```

`salt.modules.virt.`**`vm_diskstats`**`(`*vm_=None*`)`

Return disk usage counters used by the vms on this hyper in a list of dicts:

```
[
    'your-vm': {
        'rd_req'   : 0,
        'rd_bytes' : 0,
        'wr_req'   : 0,
        'wr_bytes' : 0,
        'errs'     : 0
        },
    ...
    ]
```

If you pass a VM name in as an argument then it will return info for just the named VM, otherwise it will return all VMs.

CLI Example:

```
salt '*' virt.vm_blockstats
```

`salt.modules.virt.`**`vm_info`**`(`*vm_=None*`)`

Return detailed information about the vms on this hyper in a list of dicts:

```
[
    'your-vm': {
        'cpu': <int>,
        'maxMem': <int>,
        'mem': <int>,
        'state': '<state>',
        'cputime' <int>
        },
    ...
    ]
```

If you pass a VM name in as an argument then it will return info for just the named VM, otherwise it will return all VMs.

CLI Example:

```
salt '*' virt.vm_info
```

`salt.modules.virt.`**`vm_netstats`**`(`*vm_=None*`)`

Return combined network counters used by the vms on this hyper in a list of dicts:

```
[
    'your-vm': {
        'rx_bytes'   : 0,
        'rx_packets' : 0,
        'rx_errs'    : 0,
        'rx_drop'    : 0,
        'tx_bytes'   : 0,
        'tx_packets' : 0,
        'tx_errs'    : 0,
        'tx_drop'     : 0
        },
    ...
    ]
```

If you pass a VM name in as an argument then it will return info for just the named VM, otherwise it will return all VMs.

CLI Example:

```
salt '*' virt.vm_netstats
```

`salt.modules.virt.`**`vm_state`**`(`*vm_=None*`)`

Return list of all the vms and their state.

If you pass a VM name in as an argument then it will return info for just the named VM, otherwise it will return all VMs.

CLI Example:

```
salt '*' virt.vm_state <vm name>
```

61.169 salt.modules.virtualenv

Create virtualenv environments

`salt.modules.virtualenv_mod.`**`create`**`(`*path, venv_bin=None, no_site_packages=None, system_site_packages=False, distribute=False, clear=False, python=None, extra_search_dir=None, never_download=None, prompt=None, pip=False, symlinks=None, upgrade=None, user=None, runas=None, saltenv='base'*`)`

Create a virtualenv

path The path to create the virtualenv

venv_bin [None (default 'virtualenv')] The name (and optionally path) of the virtualenv command. This can also be set globally in the minion config file as `virtualenv.venv_bin`.

no_site_packages [None] Passthrough argument given to virtualenv if True. Deprecated since `salt>=0.17.0`. Use `system_site_packages=False` instead.

system_site_packages [False] Passthrough argument given to virtualenv or pyvenv

distribute [False] Passthrough argument given to virtualenv

pip [False] Install pip after creating a virtual environment, implies distribute=True

clear [False] Passthrough argument given to virtualenv or pyvenv

python [None (default)] Passthrough argument given to virtualenv

extra_search_dir [None (default)] Passthrough argument given to virtualenv

never_download [None (default)] Passthrough argument given to virtualenv if True

prompt [None (default)] Passthrough argument given to virtualenv if not None

symlinks [None] Passthrough argument given to pyvenv if True

upgrade [None] Passthrough argument given to pyvenv if True

user [None] Set ownership for the virtualenv

runas [None] Set ownership for the virtualenv

Note: The `runas` argument is deprecated as of Hydrogen. `user` should be used instead.

CLI Example:

```
salt '*' virtualenv.create /path/to/new/virtualenv
```

`salt.modules.virtualenv_mod.`**`get_site_packages`**`(`*venv*`)`

Returns the path to the site-packages directory inside a virtualenv

CLI Example:

```
salt '*' virtualenv.get_site_packages /path/to/my/venv
```

61.170 salt.modules.win_autoruns

Module for listing programs that automatically run on startup (very alpha...not tested on anything but my Win 7x64)

`salt.modules.win_autoruns.`**`list_`**`()`

Get a list of automatically running programs

CLI Example:

```
salt '*' autoruns.list
```

61.171 salt.modules.win_disk

Module for gathering disk information on Windows

depends

- win32api Python module

`salt.modules.win_disk.`**`usage`**`()`

Return usage information for volumes mounted on this minion

CLI Example:

```
salt '*' disk.usage
```

61.172 salt.modules.win_dns_client

Module for configuring DNS Client on Windows systems

`salt.modules.win_dns_client.`**`add_dns`**`(`*ip*, *interface='Local Area Connection'*, *index=1*`)`

Add the DNS server to the network interface (index starts from 1)

Note: if the interface DNS is configured by DHCP, all the DNS servers will be removed from the interface and the requested DNS will be the only one

CLI Example:

```
salt '*' win_dns_client.add_dns <interface> <index>
```

`salt.modules.win_dns_client.`**`dns_dhcp`**(*interface='Local Area Connection'*)

Configure the interface to get its DNS servers from the DHCP server

CLI Example:

```
salt '*' win_dns_client.dns_dhcp <interface>
```

`salt.modules.win_dns_client.`**`get_dns_config`**(*interface='Local Area Connection'*)

Get the type of DNS configuration (dhcp / static)

CLI Example:

```
salt '*' win_dns_client.get_dns_config 'Local Area Connection'
```

`salt.modules.win_dns_client.`**`get_dns_servers`**(*interface='Local Area Connection'*)

Return a list of the configured DNS servers of the specified interface

CLI Example:

```
salt '*' win_dns_client.get_dns_servers 'Local Area Connection'
```

`salt.modules.win_dns_client.`**`rm_dns`**(*ip*, *interface='Local Area Connection'*)

Remove the DNS server to the network interface

CLI Example:

```
salt '*' win_dns_client.rm_dns <interface>
```

61.173 salt.modules.win_file

Manage information about files on the minion, set/read user, group data

depends

- win32api
- win32con
- win32file
- win32security
- ntsecuritycon

`salt.modules.win_file.`**`chgrp`**(*path*, *group*)

Change the group of a file

CLI Example:

```
salt '*' file.chgrp c:\temp\test.txt administrators
```

`salt.modules.win_file.`**`chown`**(*path*, *user*, *group*)

Chown a file, pass the file the desired user and group

CLI Example:

```
salt '*' file.chown c:\temp\test.txt myusername administrators
```

`salt.modules.win_file.`**`get_attributes`**(*path*)

Return a dictionary object with the Windows file attributes for a file.

CLI Example:

```
salt '*' file.get_attributes c:\temp\a.txt
```

`salt.modules.win_file.`**`get_gid`**(*path*)

Return the id of the group that owns a given file

CLI Example:

```
salt '*' file.get_gid c:\temp\test.txt
```

`salt.modules.win_file.`**`get_group`**(*path*)

Return the group that owns a given file

CLI Example:

```
salt '*' file.get_group c:\temp\test.txt
```

`salt.modules.win_file.`**`get_mode`**(*path*)

Return the mode of a file

Right now we're just returning None because Windows' doesn't have a mode like Linux

CLI Example:

```
salt '*' file.get_mode /etc/passwd
```

`salt.modules.win_file.`**`get_uid`**(*path*)

Return the id of the user that owns a given file

CLI Example:

```
salt '*' file.get_uid c:\temp\test.txt
```

`salt.modules.win_file.`**`get_user`**(*path*)

Return the user that owns a given file

CLI Example:

```
salt '*' file.get_user c:\temp\test.txt
```

`salt.modules.win_file.`**`gid_to_group`**(*gid*)

Convert the group id to the group name on this system

CLI Example:

```
salt '*' file.gid_to_group S-1-5-21-626487655-2533044672-482107328-1010
```

`salt.modules.win_file.`**`group_to_gid`**(*group*)

Convert the group to the gid on this system

CLI Example:

```
salt '*' file.group_to_gid administrators
```

`salt.modules.win_file.`**`set_attributes`**(*path*, *archive=None*, *hidden=None*, *normal=None*, *notIndexed=None*, *readonly=None*, *system=None*, *temporary=None*)

Set file attributes for a file. Note that the normal attribute means that all others are false. So setting it will clear all others.

CLI Example:

```
salt '*' file.set_attributes c:\temp\a.txt normal=True
salt '*' file.set_attributes c:\temp\a.txt readonly=True hidden=True
```

`salt.modules.win_file.`**`set_mode`**`(`*path*, *mode*`)`

Set the mode of a file

This just calls get_mode, which returns None because we don't use mode on Windows

CLI Example:

```
salt '*' file.set_mode /etc/passwd 0644
```

`salt.modules.win_file.`**`stats`**`(`*path*, *hash_type='md5'*, *follow_symlinks=False*`)`

Return a dict containing the stats for a given file

CLI Example:

```
salt '*' file.stats /etc/passwd
```

`salt.modules.win_file.`**`uid_to_user`**`(`*uid*`)`

Convert a uid to a user name

CLI Example:

```
salt '*' file.uid_to_user S-1-5-21-626487655-2533044672-482107328-1010
```

`salt.modules.win_file.`**`user_to_uid`**`(`*user*`)`

Convert user name to a uid

CLI Example:

```
salt '*' file.user_to_uid myusername
```

61.174 salt.modules.win_firewall

Module for configuring Windows Firewall

`salt.modules.win_firewall.`**`disable`**`()`

Disable all the firewall profiles

CLI Example:

```
salt '*' firewall.disable
```

`salt.modules.win_firewall.`**`get_config`**`()`

Get the status of all the firewall profiles

CLI Example:

```
salt '*' firewall.get_config
```

61.175 salt.modules.win_groupadd

Manage groups on Windows

`salt.modules.win_groupadd.`**`add`**`(`*name*, *gid=None*, *system=False*`)`

Add the specified group

CLI Example:

```
salt '*' group.add foo
```

`salt.modules.win_groupadd.`**`delete`**`(`*name*`)`

Remove the named group

CLI Example:

```
salt '*' group.delete foo
```

`salt.modules.win_groupadd.`**`getent`**`(`*refresh=False*`)`

Return info on all groups

CLI Example:

```
salt '*' group.getent
```

`salt.modules.win_groupadd.`**`info`**`(`*name*`)`

Return information about a group

CLI Example:

```
salt '*' group.info foo
```

61.176 salt.modules.win_ip

The networking module for Windows based systems

`salt.modules.win_ip.`**`disable`**`(`*iface*`)`

Disable an interface

CLI Example:

```
salt -G 'os_family:Windows' ip.disable 'Local Area Connection #2'
```

`salt.modules.win_ip.`**`enable`**`(`*iface*`)`

Enable an interface

CLI Example:

```
salt -G 'os_family:Windows' ip.enable 'Local Area Connection #2'
```

`salt.modules.win_ip.`**`get_all_interfaces`**`()`

Return configs for all interfaces

CLI Example:

```
salt -G 'os_family:Windows' ip.get_all_interfaces
```

`salt.modules.win_ip.`**`get_default_gateway`**`()`

Set DNS source to DHCP on Windows

CLI Example:

```
salt -G 'os_family:Windows' ip.get_default_gateway
```

salt.modules.win_ip.**get_interface**(*iface*)

Return the configuration of a network interface

CLI Example:

```
salt -G 'os_family:Windows' ip.get_interface 'Local Area Connection'
```

salt.modules.win_ip.**get_subnet_length**(*mask*)

Convenience function to convert the netmask to the CIDR subnet length

CLI Example:

```
salt -G 'os_family:Windows' ip.get_subnet_length 255.255.255.0
```

salt.modules.win_ip.**is_disabled**(*iface*)

Returns `True` if interface is disabled, otherwise `False`

CLI Example:

```
salt -G 'os_family:Windows' ip.is_disabled 'Local Area Connection #2'
```

salt.modules.win_ip.**is_enabled**(*iface*)

Returns `True` if interface is enabled, otherwise `False`

CLI Example:

```
salt -G 'os_family:Windows' ip.is_enabled 'Local Area Connection #2'
```

salt.modules.win_ip.**raw_interface_configs**()

Return raw configs for all interfaces

CLI Example:

```
salt -G 'os_family:Windows' ip.raw_interface_configs
```

salt.modules.win_ip.**set_dhcp_all**(*iface*)

Set both IP Address and DNS to DHCP

CLI Example:

```
salt -G 'os_family:Windows' ip.set_dhcp_all 'Local Area Connection'
```

salt.modules.win_ip.**set_dhcp_dns**(*iface*)

Set DNS source to DHCP on Windows

CLI Example:

```
salt -G 'os_family:Windows' ip.set_dhcp_dns 'Local Area Connection'
```

salt.modules.win_ip.**set_dhcp_ip**(*iface*)

Set Windows NIC to get IP from DHCP

CLI Example:

```
salt -G 'os_family:Windows' ip.set_dhcp_ip 'Local Area Connection'
```

salt.modules.win_ip.**set_static_dns**(*iface*, **addrs*)

Set static DNS configuration on a Windows NIC

CLI Example:

```
salt -G 'os_family:Windows' ip.set_static_dns 'Local Area Connection' '192.168.1.1'
salt -G 'os_family:Windows' ip.set_static_dns 'Local Area Connection' '192.168.1.252' '192.168.1
```

`salt.modules.win_ip.`**`set_static_ip`**(*iface*, *addr*, *gateway=None*, *append=False*)
Set static IP configuration on a Windows NIC

iface The name of the interface to manage

addr IP address with subnet length (ex. `10.1.2.3/24`). The `ip.get_subnet_length` function can be used to calculate the subnet length from a netmask.

gateway [None] If specified, the default gateway will be set to this value.

append [False] If `True`, this IP address will be added to the interface. Default is `False`, which overrides any existing configuration for the interface and sets `addr` as the only address on the interface.

CLI Example:

```
salt -G 'os_family:Windows' ip.set_static_ip 'Local Area Connection' 10.1.2.3/24 gateway=10.1.2.
salt -G 'os_family:Windows' ip.set_static_ip 'Local Area Connection' 10.1.2.4/24 append=True
```

61.177 salt.modules.win_network

Module for gathering and managing network information

`salt.modules.win_network.`**`dig`**(*host*)
Performs a DNS lookup with dig

Note: dig must be installed on the Windows minion

CLI Example:

```
salt '*' network.dig archlinux.org
```

`salt.modules.win_network.`**`hw_addr`**(*iface*)
Return the hardware address (a.k.a. MAC address) for a given interface

CLI Example:

```
salt '*' network.hw_addr 'Wireless Connection #1'
```

`salt.modules.win_network.`**`hwaddr`**(*iface*)
Return the hardware address (a.k.a. MAC address) for a given interface

CLI Example:

```
salt '*' network.hw_addr 'Wireless Connection #1'
```

`salt.modules.win_network.`**`in_subnet`**(*cidr*)
Returns True if host is within specified subnet, otherwise False

CLI Example:

```
salt '*' network.in_subnet 10.0.0.0/16
```

`salt.modules.win_network.`**`interfaces`**()
Return a dictionary of information about all the interfaces on the minion

CLI Example:

```
salt '*' network.interfaces
```

`salt.modules.win_network.`**`ip_addrs`**(*interface=None*, *include_loopback=False*)
Returns a list of IPv4 addresses assigned to the host. 127.0.0.1 is ignored, unless 'include_loopback=True' is indicated. If 'interface' is provided, then only IP addresses from that interface will be returned.

CLI Example:

```
salt '*' network.ip_addrs
```

`salt.modules.win_network.`**`ip_addrs6`**`(`*interface=None*, *include_loopback=False*`)`
Returns a list of IPv6 addresses assigned to the host. ::1 is ignored, unless 'include_loopback=True' is indicated. If 'interface' is provided, then only IP addresses from that interface will be returned.

CLI Example:

```
salt '*' network.ip_addrs6
```

`salt.modules.win_network.`**`ipaddrs`**`(`*interface=None*, *include_loopback=False*`)`
Returns a list of IPv4 addresses assigned to the host. 127.0.0.1 is ignored, unless 'include_loopback=True' is indicated. If 'interface' is provided, then only IP addresses from that interface will be returned.

CLI Example:

```
salt '*' network.ip_addrs
```

`salt.modules.win_network.`**`ipaddrs6`**`(`*interface=None*, *include_loopback=False*`)`
Returns a list of IPv6 addresses assigned to the host. ::1 is ignored, unless 'include_loopback=True' is indicated. If 'interface' is provided, then only IP addresses from that interface will be returned.

CLI Example:

```
salt '*' network.ip_addrs6
```

`salt.modules.win_network.`**`netstat`**`()`
Return information on open ports and states

CLI Example:

```
salt '*' network.netstat
```

`salt.modules.win_network.`**`nslookup`**`(`*host*`)`
Query DNS for information about a domain or ip address

CLI Example:

```
salt '*' network.nslookup archlinux.org
```

`salt.modules.win_network.`**`ping`**`(`*host*`)`
Performs a ping to a host

CLI Example:

```
salt '*' network.ping archlinux.org
```

`salt.modules.win_network.`**`subnets`**`()`
Returns a list of subnets to which the host belongs

CLI Example:

```
salt '*' network.subnets
```

`salt.modules.win_network.`**`traceroute`**`(`*host*`)`
Performs a traceroute to a 3rd party host

CLI Example:

```
salt '*' network.traceroute archlinux.org
```

61.178 salt.modules.win_ntp

Management of NTP servers on Windows

New in version 2014.1.0: (Hydrogen)

`salt.modules.win_ntp.`**`get_servers`**`()`

Get list of configured NTP servers

CLI Example:

```
salt '*' ntp.get_servers
```

`salt.modules.win_ntp.`**`set_servers`**`(`**servers*`)`

Set Windows to use a list of NTP servers

CLI Example:

```
salt '*' ntp.set_servers 'pool.ntp.org' 'us.pool.ntp.org'
```

61.179 salt.modules.win_path

Manage the Windows System PATH

Note that not all Windows applications will rehash the PATH environment variable, Only the ones that listen to the WM_SETTINGCHANGE message http://support.microsoft.com/kb/104011

`salt.modules.win_path.`**`add`**`(`*path*, *index=0*`)`

Add the directory to the SYSTEM path in the index location

CLI Example:

```
# Will add to the beginning of the path
salt '*' win_path.add 'c:\python27' 0

# Will add to the end of the path
salt '*' win_path.add 'c:\python27' index='-1'
```

`salt.modules.win_path.`**`exists`**`(`*path*`)`

Check if the directory is configured in the SYSTEM path Case-insensitive and ignores trailing backslash

CLI Example:

```
salt '*' win_path.exists 'c:\python27'
salt '*' win_path.exists 'c:\python27\'
salt '*' win_path.exists 'C:\pyThon27'
```

`salt.modules.win_path.`**`get_path`**`()`

Returns the system path

`salt.modules.win_path.`**`rehash`**`()`

Send a WM_SETTINGCHANGE Broadcast to Windows to rehash the Environment variables

`salt.modules.win_path.`**`remove`**`(`*path*`)`

Remove the directory from the SYSTEM path

61.180 salt.modules.win_pkg

A module to manage software on Windows

depends

- pythoncom
- win32com
- win32con
- win32api
- pywintypes

`salt.modules.win_pkg.`**`get_repo_data`**`(`*saltenv='base'*`)`

Returns the cached winrepo data

CLI Example:

```
salt '*' pkg.get_repo_data
```

`salt.modules.win_pkg.`**`install`**`(`*name=None*, *refresh=False*, *pkgs=None*, *saltenv='base'*, ***kwargs*`)`

Install the passed package

Return a dict containing the new package names and versions:

```
{'<package>': {'old': '<old-version>',
               'new': '<new-version>'}}
```

CLI Example:

```
salt '*' pkg.install <package name>
```

`salt.modules.win_pkg.`**`latest_version`**`(`**names*, ***kwargs*`)`

Return the latest version of the named package available for upgrade or installation. If more than one package name is specified, a dict of name/version pairs is returned.

If the latest version of a given package is already installed, an empty string will be returned for that package.

CLI Example:

```
salt '*' pkg.latest_version <package name>
salt '*' pkg.latest_version <package1> <package2> <package3> ...
```

`salt.modules.win_pkg.`**`list_available`**`(`**names*`)`

Return a list of available versions of the specified package.

CLI Example:

```
salt '*' pkg.list_available <package name>
salt '*' pkg.list_available <package name01> <package name02>
```

`salt.modules.win_pkg.`**`list_pkgs`**`(`*versions_as_list=False*, ***kwargs*`)`

List the packages currently installed in a dict:

```
{'<package_name>': '<version>'}
```

CLI Example:

```
salt '*' pkg.list_pkgs
salt '*' pkg.list_pkgs versions_as_list=True
```

`salt.modules.win_pkg.`**`list_upgrades`**(*refresh=True*)

List all available package upgrades on this system

CLI Example:

```
salt '*' pkg.list_upgrades
```

`salt.modules.win_pkg.`**`purge`**(*name=None*, *pkgs=None*, *version=None*, ***kwargs*)

Package purges are not supported, this function is identical to `remove()`.

name The name of the package to be deleted.

version The version of the package to be deleted. If this option is used in combination with the `pkgs` option below, then this version will be applied to all targeted packages.

Multiple Package Options:

pkgs A list of packages to delete. Must be passed as a python list. The `name` parameter will be ignored if this option is passed.

New in version 0.16.0.

Returns a dict containing the changes.

CLI Example:

```
salt '*' pkg.purge <package name>
salt '*' pkg.purge <package1>,<package2>,<package3>
salt '*' pkg.purge pkgs='["foo", "bar"]'
```

`salt.modules.win_pkg.`**`refresh_db`**(*saltenv='base'*)

Just recheck the repository and return a dict:

```
{'<database name>': Bool}
```

CLI Example:

```
salt '*' pkg.refresh_db
```

`salt.modules.win_pkg.`**`remove`**(*name=None*, *pkgs=None*, *version=None*, *extra_uninstall_flags=None*, ***kwargs*)

Remove packages.

name The name of the package to be deleted.

version The version of the package to be deleted. If this option is used in combination with the `pkgs` option below, then this version will be applied to all targeted packages.

Multiple Package Options:

pkgs A list of packages to delete. Must be passed as a python list. The `name` parameter will be ignored if this option is passed.

New in version 0.16.0.

Returns a dict containing the changes.

CLI Example:

```
salt '*' pkg.remove <package name>
salt '*' pkg.remove <package1>,<package2>,<package3>
salt '*' pkg.remove pkgs='["foo", "bar"]'
```

`salt.modules.win_pkg.`**`upgrade`**`(`*refresh=True*`)`

Run a full system upgrade

Return a dict containing the new package names and versions:

```
{'<package>': {'old': '<old-version>',
               'new': '<new-version>'}}
```

CLI Example:

```
salt '*' pkg.upgrade
```

`salt.modules.win_pkg.`**`upgrade_available`**`(`*name*`)`

Check whether or not an upgrade is available for a given package

CLI Example:

```
salt '*' pkg.upgrade_available <package name>
```

`salt.modules.win_pkg.`**`version`**`(`**names*, ***kwargs*`)`

Returns a version if the package is installed, else returns an empty string

CLI Example:

```
salt '*' pkg.version <package name>
```

61.181 salt.modules.win_repo

Module to manage Windows software repo on a Standalone Minion

The following options must be set in the Minion config: file_client: local win_repo_cachefile: c:saltfile_rootswinrepowinrepo.p win_repo: c:saltfile_rootswinrepo

Place all Windows package files in the 'win_repo' directory.

`salt.modules.win_repo.`**`genrepo`**`()`

Generate win_repo_cachefile based on sls files in the win_repo

CLI Example:

```
salt-call winrepo.genrepo -c c:\salt\conf
```

`salt.modules.win_repo.`**`update_git_repos`**`()`

Checkout git repos containing Windows Software Package Definitions

Note: This function will not work unless git is installed and the git module is further updated to work on Windows. In the meantime just place all Windows package files in the `win_repo` directory.

61.182 salt.modules.win_servermanager

Manage Windows features via the ServerManager powershell module

`salt.modules.win_servermanager.`**`install`**`(`*feature*, *recurse=False*`)`

Install a feature

Note: Some features requires reboot after un/installation, if so until the server is restarted Other features can not be installed !

Note: Some features takes a long time to complete un/installation, set -t with a long timeout

CLI Example:

```
salt '*' win_servermanager.install Telnet-Client
salt '*' win_servermanager.install SNMP-Services True
```

`salt.modules.win_servermanager.`**`list_available`**`()`

List available features to install

CLI Example:

```
salt '*' win_servermanager.list_available
```

`salt.modules.win_servermanager.`**`list_installed`**`()`

List installed features

CLI Example:

```
salt '*' win_servermanager.list_installed
```

`salt.modules.win_servermanager.`**`remove`**`(`*feature*`)`

Remove an installed feature

Note: Some features require a reboot after installation/uninstallation. If one of these features are modified, then other features cannot be installed until the server is restarted. Additionally, some features take a while to complete installation/uninstallation, so it is a good idea to use the `-t` option to set a longer timeout.

CLI Example:

```
salt -t 600 '*' win_servermanager.remove Telnet-Client
```

61.183 salt.modules.win_service

Windows Service module.

`salt.modules.win_service.`**`available`**`(`*name*`)`

Returns `True` if the specified service is available, otherwise returns `False`.

CLI Example:

```
salt '*' service.available <service name>
```

`salt.modules.win_service.`**`disable`**`(`*name*, ***kwargs*`)`

Disable the named service to start at boot

CLI Example:

```
salt '*' service.disable <service name>
```

`salt.modules.win_service.`**`disabled`**`(`*name*`)`

Check to see if the named service is disabled to start on boot

CLI Example:

```
salt '*' service.disabled <service name>
```

`salt.modules.win_service.`**`enable`**`(`*name*, ***kwargs*`)`

Enable the named service to start at boot

CLI Example:

```
salt '*' service.enable <service name>
```

`salt.modules.win_service.`**`enabled`**(*name*)

Check to see if the named service is enabled to start on boot

CLI Example:

```
salt '*' service.enabled <service name>
```

`salt.modules.win_service.`**`get_all`**()

Return all installed services

CLI Example:

```
salt '*' service.get_all
```

`salt.modules.win_service.`**`get_disabled`**()

Return the disabled services

CLI Example:

```
salt '*' service.get_disabled
```

`salt.modules.win_service.`**`get_enabled`**()

Return the enabled services

CLI Example:

```
salt '*' service.get_enabled
```

`salt.modules.win_service.`**`get_service_name`**(**args*)

The Display Name is what is displayed in Windows when services.msc is executed. Each Display Name has an associated Service Name which is the actual name of the service. This function allows you to discover the Service Name by returning a dictionary of Display Names and Service Names, or filter by adding arguments of Display Names.

If no args are passed, return a dict of all services where the keys are the service Display Names and the values are the Service Names.

If arguments are passed, create a dict of Display Names and Service Names

CLI Example:

```
salt '*' service.get_service_name
salt '*' service.get_service_name 'Google Update Service (gupdate)' 'DHCP Client'
```

`salt.modules.win_service.`**`getsid`**(*name*)

Return the sid for this windows service

CLI Example:

```
salt '*' service.getsid <service name>
```

`salt.modules.win_service.`**`has_powershell`**()

Confirm if Powershell is available

CLI Example:

```
salt '*' service.has_powershell
```

`salt.modules.win_service.`**`missing`**`(`*name*`)`

The inverse of service.available. Returns `True` if the specified service is not available, otherwise returns `False`.

CLI Example:

```
salt '*' service.missing <service name>
```

`salt.modules.win_service.`**`restart`**`(`*name*`)`

Restart the named service

CLI Example:

```
salt '*' service.restart <service name>
```

`salt.modules.win_service.`**`start`**`(`*name*`)`

Start the specified service

CLI Example:

```
salt '*' service.start <service name>
```

`salt.modules.win_service.`**`status`**`(`*name*, *sig=None*`)`

Return the status for a service, returns the PID or an empty string if the service is running or not, pass a signature to use to find the service via ps

CLI Example:

```
salt '*' service.status <service name> [service signature]
```

`salt.modules.win_service.`**`stop`**`(`*name*`)`

Stop the specified service

CLI Example:

```
salt '*' service.stop <service name>
```

61.184 salt.modules.win_shadow

Manage the shadow file

`salt.modules.win_shadow.`**`info`**`(`*name*`)`

Return information for the specified user This is just returns dummy data so that salt states can work.

CLI Example:

```
salt '*' shadow.info root
```

`salt.modules.win_shadow.`**`set_password`**`(`*name*, *password*`)`

Set the password for a named user.

CLI Example:

```
salt '*' shadow.set_password root mysecretpassword
```

61.185 salt.modules.win_status

Module for returning various status data about a minion. These data can be useful for compiling into stats later.

depends

- pythoncom
- wmi

`salt.modules.win_status.`**`procs`**`()`

Return the process data

CLI Example:

```
salt '*' status.procs
```

61.186 salt.modules.win_system

Support for reboot, shutdown, etc

`salt.modules.win_system.`**`get_computer_desc`**`()`

Get the Windows computer description

CLI Example:

```
salt 'minion-id' system.get_computer_desc
```

`salt.modules.win_system.`**`get_computer_name`**`()`

Get the Windows computer name

CLI Example:

```
salt 'minion-id' system.get_computer_name
```

`salt.modules.win_system.`**`get_pending_computer_name`**`()`

Get a pending computer name. If the computer name has been changed, and the change is pending a system reboot, this function will return the pending computer name. Otherwise, `None` will be returned. If there was an error retrieving the pending computer name, `False` will be returned, and an error message will be logged to the minion log.

CLI Example:

```
salt 'minion-id' system.get_pending_computer_name
```

`salt.modules.win_system.`**`get_system_date`**`()`

Get the Windows system date

CLI Example:

```
salt '*' system.get_system_date
```

`salt.modules.win_system.`**`get_system_time`**`()`

Get the Windows system time

CLI Example:

```
salt '*' system.get_system_time
```

`salt.modules.win_system.`**`halt`**`(`*timeout=5*`)`

Halt a running system

CLI Example:

```
salt '*' system.halt
```

`salt.modules.win_system.`**`init`**(*runlevel*)

Change the system runlevel on sysV compatible systems

CLI Example:

```
salt '*' system.init 3
```

`salt.modules.win_system.`**`join_domain`**(*domain*, *username*, *passwd*, *ou=None*, *acct_exists=False*)

Join a computer the an Active Directory domain

CLI Example:

```
salt 'minion-id' system.join_domain 'mydomain.local' 'myusername'              'mysecretpasswd'
```

`salt.modules.win_system.`**`poweroff`**(*timeout=5*)

Poweroff a running system

CLI Example:

```
salt '*' system.poweroff
```

`salt.modules.win_system.`**`reboot`**(*timeout=5*)

Reboot the system

CLI Example:

```
salt '*' system.reboot
```

`salt.modules.win_system.`**`set_computer_desc`**(*desc*)

Set the Windows computer description

CLI Example:

```
salt 'minion-id' system.set_computer_desc 'This computer belongs to Dave!'
```

`salt.modules.win_system.`**`set_computer_name`**(*name*)

Set the Windows computer name

CLI Example:

```
salt 'minion-id' system.set_computer_name 'DavesComputer'
```

`salt.modules.win_system.`**`set_system_date`**(*newdate*)

Set the Windows system date. Use <mm-dd-yy> format for the date.

CLI Example:

```
salt '*' system.set_system_date '03-28-13'
```

`salt.modules.win_system.`**`set_system_time`**(*newtime*)

Set the Windows system time

CLI Example:

```
salt '*' system.set_system_time '11:31:15 AM'
```

`salt.modules.win_system.`**`shutdown`**(*timeout=5*)

Shutdown a running system

CLI Example:

```
salt '*' system.shutdown
```

`salt.modules.win_system.`**`shutdown_hard`**`()`
Shutdown a running system with no timeout or warning

CLI Example:

```
salt '*' system.shutdown_hard
```

`salt.modules.win_system.`**`start_time_service`**`()`
Start the Windows time service

CLI Example:

```
salt '*' system.start_time_service
```

`salt.modules.win_system.`**`stop_time_service`**`()`
Stop the Windows time service

CLI Example:

```
salt '*' system.stop_time_service
```

61.187 salt.modules.win_timezone

Module for managing timezone on Windows systems.

`salt.modules.win_timezone.`**`get_hwclock`**`()`
Get current hardware clock setting (UTC or localtime)

CLI Example:

```
salt '*' timezone.get_hwclock
```

`salt.modules.win_timezone.`**`get_offset`**`()`
Get current numeric timezone offset from UCT (i.e. -0700)

CLI Example:

```
salt '*' timezone.get_offset
```

`salt.modules.win_timezone.`**`get_zone`**`()`
Get current timezone (i.e. America/Denver)

CLI Example:

```
salt '*' timezone.get_zone
```

`salt.modules.win_timezone.`**`get_zonecode`**`()`
Get current timezone (i.e. PST, MDT, etc)

CLI Example:

```
salt '*' timezone.get_zonecode
```

`salt.modules.win_timezone.`**`set_hwclock`**`(`*clock*`)`
Sets the hardware clock to be either UTC or localtime

CLI Example:

```
salt '*' timezone.set_hwclock UTC
```

`salt.modules.win_timezone.`**`set_zone`**(*timezone*)

Unlinks, then symlinks /etc/localtime to the set timezone.

The timezone is crucial to several system processes, each of which SHOULD be restarted (for instance, whatever you system uses as its cron and syslog daemons). This will not be magically done for you!

CLI Example:

```
salt '*' timezone.set_zone 'America/Denver'
```

`salt.modules.win_timezone.`**`zone_compare`**(*timezone*)

Checks the md5sum between the given timezone, and the one set in /etc/localtime. Returns True if they match, and False if not. Mostly useful for running state checks.

Example:

```
salt '*' timezone.zone_compare 'America/Denver'
```

61.188 salt.modules.win_useradd

Manage Windows users with the net user command

NOTE: This currently only works with local user accounts, not domain accounts

`salt.modules.win_useradd.`**`add`**(*name*, *uid=None*, *gid=None*, *groups=None*, *home=False*, *shell=None*, *unique=False*, *system=False*, *fullname=False*, *roomnumber=False*, *workphone=False*, *homephone=False*, *createhome=False*)

Add a user to the minion

CLI Example:

```
salt '*' user.add name password
```

`salt.modules.win_useradd.`**`addgroup`**(*name*, *group*)

Add user to a group

CLI Example:

```
salt '*' user.addgroup username groupname
```

`salt.modules.win_useradd.`**`chfullname`**(*name*, *fullname*)

Change the full name of the user

CLI Example:

```
salt '*' user.chfullname user 'First Last'
```

`salt.modules.win_useradd.`**`chgroups`**(*name*, *groups*, *append=False*)

Change the groups this user belongs to, add append to append the specified groups

CLI Example:

```
salt '*' user.chgroups foo wheel,root True
```

`salt.modules.win_useradd.`**`chhome`**(*name*, *home*)

Change the home directory of the user

CLI Example:

```
salt '*' user.chhome foo \\fileserver\home\foo
```

salt.modules.win_useradd.**chprofile**(*name*, *profile*)
Change the profile directory of the user

CLI Example:

```
salt '*' user.chprofile foo \\fileserver\profiles\foo
```

salt.modules.win_useradd.**delete**(*name*, *purge=False*, *force=False*)
Remove a user from the minion NOTE: purge and force have not been implemented on Windows yet

CLI Example:

```
salt '*' user.delete name
```

salt.modules.win_useradd.**getent**(*refresh=False*)
Return the list of all info for all users

CLI Example:

```
salt '*' user.getent
```

salt.modules.win_useradd.**info**(*name*)
Return user information

CLI Example:

```
salt '*' user.info root
```

salt.modules.win_useradd.**list_groups**(*name*)
Return a list of groups the named user belongs to

CLI Example:

```
salt '*' user.list_groups foo
```

salt.modules.win_useradd.**list_users**()
Return a list of users on Windows

salt.modules.win_useradd.**removegroup**(*name*, *group*)
Remove user from a group

CLI Example:

```
salt '*' user.removegroup username groupname
```

salt.modules.win_useradd.**setpassword**(*name*, *password*)
Set a user's password

CLI Example:

```
salt '*' user.setpassword name password
```

61.189 salt.modules.xapi

This module (mostly) uses the XenAPI to manage Xen virtual machines.

Big fat warning: the XenAPI used in this file is the one bundled with Xen Source, NOT XenServer nor Xen Cloud Platform. As a matter of fact it *will* fail under those platforms. From what I've read, little work is needed to adapt this code to XS/XCP, mostly playing with XenAPI version, but as XCP is not taking precedence on Xen Source on many platforms, please keep compatibility in mind.

Useful documentation:

. http://downloads.xen.org/Wiki/XenAPI/xenapi-1.0.6.pdf . http://docs.vmd.citrix.com/XenServer/6.0.0/1.0/en_gb/api/ . https://github.com/xen-org/xen-api/tree/master/scripts/examples/python . http://xenbits.xen.org/gitweb/?p=xen.git;a=tree;f=tools/pytho

salt.modules.xapi.**create**(*config_*)
Start a defined domain

CLI Example:

```
salt '*' virt.create <path to Xen cfg file>
```

salt.modules.xapi.**destroy**(*vm_*)
Hard power down the virtual machine, this is equivalent to pulling the power

CLI Example:

```
salt '*' virt.destroy <vm name>
```

salt.modules.xapi.**freecpu**()
Return an int representing the number of unallocated cpus on this hypervisor

CLI Example:

```
salt '*' virt.freecpu
```

salt.modules.xapi.**freemem**()
Return an int representing the amount of memory that has not been given to virtual machines on this node

CLI Example:

```
salt '*' virt.freemem
```

salt.modules.xapi.**full_info**()
Return the node_info, vm_info and freemem

CLI Example:

```
salt '*' virt.full_info
```

salt.modules.xapi.**get_disks**(*vm_*)
Return the disks of a named vm

CLI Example:

```
salt '*' virt.get_disks <vm name>
```

salt.modules.xapi.**get_macs**(*vm_*)
Return a list off MAC addresses from the named vm

CLI Example:

```
salt '*' virt.get_macs <vm name>
```

salt.modules.xapi.**get_nics**(*vm_*)
Return info about the network interfaces of a named vm

CLI Example:

```
salt '*' virt.get_nics <vm name>
```

`salt.modules.xapi.`**`is_hyper`**`()`

Returns a bool whether or not this node is a hypervisor of any kind

CLI Example:

```
salt '*' virt.is_hyper
```

`salt.modules.xapi.`**`list_vms`**`()`

Return a list of virtual machine names on the minion

CLI Example:

```
salt '*' virt.list_vms
```

`salt.modules.xapi.`**`migrate`**`(`*vm_*, *target*, *live=1*, *port=0*, *node=-1*, *ssl=None*, *change_home_server=0*`)`

Migrates the virtual machine to another hypervisor

CLI Example:

```
salt '*' virt.migrate <vm name> <target hypervisor> [live] [port] [node] [ssl] [change_home_serv
```

Optional values:

live Use live migration

port Use a specified port

node Use specified NUMA node on target

ssl use ssl connection for migration

change_home_server change home server for managed domains

`salt.modules.xapi.`**`node_info`**`()`

Return a dict with information about this node

CLI Example:

```
salt '*' virt.node_info
```

`salt.modules.xapi.`**`pause`**`(`*vm_*`)`

Pause the named vm

CLI Example:

```
salt '*' virt.pause <vm name>
```

`salt.modules.xapi.`**`reboot`**`(`*vm_*`)`

Reboot a domain via ACPI request

CLI Example:

```
salt '*' virt.reboot <vm name>
```

`salt.modules.xapi.`**`reset`**`(`*vm_*`)`

Reset a VM by emulating the reset button on a physical machine

CLI Example:

```
salt '*' virt.reset <vm name>
```

`salt.modules.xapi.`**`resume`**`(`*vm_*`)`

Resume the named vm

CLI Example:

```
salt '*' virt.resume <vm name>
```

`salt.modules.xapi.`**`setmem`**`(`*vm_*, *memory*`)`

Changes the amount of memory allocated to VM.

Memory is to be specified in MB

CLI Example:

```
salt '*' virt.setmem myvm 768
```

`salt.modules.xapi.`**`setvcpus`**`(`*vm_*, *vcpus*`)`

Changes the amount of vcpus allocated to VM.

vcpus is an int representing the number to be assigned

CLI Example:

```
salt '*' virt.setvcpus myvm 2
```

`salt.modules.xapi.`**`shutdown`**`(`*vm_*`)`

Send a soft shutdown signal to the named vm

CLI Example:

```
salt '*' virt.shutdown <vm name>
```

`salt.modules.xapi.`**`start`**`(`*config_*`)`

Alias for the obscurely named 'create' function

CLI Example:

```
salt '*' virt.start <path to Xen cfg file>
```

`salt.modules.xapi.`**`vcpu_pin`**`(`*vm_*, *vcpu*, *cpus*`)`

Set which CPUs a VCPU can use.

CLI Example:

```
salt 'foo' virt.vcpu_pin domU-id 2 1
salt 'foo' virt.vcpu_pin domU-id 2 2-6
```

`salt.modules.xapi.`**`vm_cputime`**`(`*vm_=None*`)`

Return cputime used by the vms on this hyper in a list of dicts:

```
[
    'your-vm': {
        'cputime' <int>
        'cputime_percent' <int>
        },
    ...
    ]
```

If you pass a VM name in as an argument then it will return info for just the named VM, otherwise it will return all VMs.

CLI Example:

```
salt '*' virt.vm_cputime
```

`salt.modules.xapi.`**`vm_diskstats`**`(`*vm_=None*`)`

Return disk usage counters used by the vms on this hyper in a list of dicts:

```
[
    'your-vm': {
        'io_read_kbs'    : 0,
        'io_write_kbs'   : 0
        },
    ...
    ]
```

If you pass a VM name in as an argument then it will return info for just the named VM, otherwise it will return all VMs.

CLI Example:

```
salt '*' virt.vm_diskstats
```

`salt.modules.xapi.`**`vm_info`**`(`*vm_=None*`)`

Return detailed information about the vms.

If you pass a VM name in as an argument then it will return info for just the named VM, otherwise it will return all VMs.

CLI Example:

```
salt '*' virt.vm_info
```

`salt.modules.xapi.`**`vm_netstats`**`(`*vm_=None*`)`

Return combined network counters used by the vms on this hyper in a list of dicts:

```
[
    'your-vm': {
        'io_read_kbs'            : 0,
        'io_total_read_kbs'      : 0,
        'io_total_write_kbs'     : 0,
        'io_write_kbs'           : 0
        },
    ...
    ]
```

If you pass a VM name in as an argument then it will return info for just the named VM, otherwise it will return all VMs.

CLI Example:

```
salt '*' virt.vm_netstats
```

`salt.modules.xapi.`**`vm_state`**`(`*vm_=None*`)`

Return list of all the vms and their state.

If you pass a VM name in as an argument then it will return info for just the named VM, otherwise it will return all VMs.

CLI Example:

```
salt '*' virt.vm_state <vm name>
```

61.190 salt.modules.xmpp

Module for Sending Messages via XMPP (a.k.a. Jabber)

New in version 2014.1.0: (Hydrogen)

depends

- sleekxmpp python module

configuration This module can be used by either passing a jid and password directly to send_message, or by specifying the name of a configuration profile in the minion config, minion pillar, or master config.

For example:

```
my-xmpp-login:
    xmpp.jid: myuser@jabber.example.org/resourcename
    xmpp.password: verybadpass
```

The resourcename refers to the resource that is using this account. It is user-definable, and optional. The following configurations are both valid:

```
my-xmpp-login:
    xmpp.jid: myuser@jabber.example.org/salt
    xmpp.password: verybadpass

my-xmpp-login:
    xmpp.jid: myuser@jabber.example.org
    xmpp.password: verybadpass
```

class `salt.modules.xmpp.`**`SendMsgBot`**(*jid*, *password*, *recipient*, *msg*)

`start`(*event*)

`salt.modules.xmpp.`**`send_msg`**(*recipient*, *message*, *jid=None*, *password=None*, *profile=None*)

Send a message to an XMPP recipient. Designed for use in states.

CLI Examples:

```
xmpp.send_msg 'admins@xmpp.example.com' 'This is a salt module test'             profile='my-xmp
xmpp.send_msg 'admins@xmpp.example.com' 'This is a salt module test'             jid='myuser@xmp
```

61.191 salt.modules.yumpkg

Support for YUM

`salt.modules.yumpkg.`**`check_db`**(**names*, ***kwargs*)

New in version 0.17.0.

Returns a dict containing the following information for each specified package:

1. A key `found`, which will be a boolean value denoting if a match was found in the package database.
2. If `found` is `False`, then a second key called `suggestions` will be present, which will contain a list of possible matches.

The `fromrepo`, `enablerepo`, and `disablerepo` arguments are supported, as used in pkg states.

CLI Examples:

```
salt '*' pkg.check_db <package1> <package2> <package3>
salt '*' pkg.check_db <package1> <package2> <package3> fromrepo=epel-testing
```

salt.modules.yumpkg.**clean_metadata**()

New in version 2014.1.0: (Hydrogen)

Cleans local yum metadata. Functionally identical to `refresh_db()`.

CLI Example:

```
salt '*' pkg.clean_metadata
```

salt.modules.yumpkg.**del_repo**(*repo*, *basedir='/etc/yum.repos.d'*, ***kwargs*)

Delete a repo from <basedir> (default basedir: /etc/yum.repos.d).

If the .repo file that the repo exists in does not contain any other repo configuration, the file itself will be deleted.

CLI Examples:

```
salt '*' pkg.del_repo myrepo
salt '*' pkg.del_repo myrepo basedir=/path/to/dir
```

salt.modules.yumpkg.**expand_repo_def**(*repokwargs*)

Take a repository definition and expand it to the full pkg repository dict that can be used for comparison. This is a helper function to make certain repo managers sane for comparison in the pkgrepo states.

There is no use to calling this function via the CLI.

salt.modules.yumpkg.**file_dict**(**packages*)

New in version 2014.1.0: (Hydrogen)

List the files that belong to a package, grouped by package. Not specifying any packages will return a list of _every_ file on the system's rpm database (not generally recommended).

CLI Examples:

```
salt '*' pkg.file_list httpd
salt '*' pkg.file_list httpd postfix
salt '*' pkg.file_list
```

salt.modules.yumpkg.**file_list**(**packages*)

New in version 2014.1.0: (Hydrogen)

List the files that belong to a package. Not specifying any packages will return a list of _every_ file on the system's rpm database (not generally recommended).

CLI Examples:

```
salt '*' pkg.file_list httpd
salt '*' pkg.file_list httpd postfix
salt '*' pkg.file_list
```

salt.modules.yumpkg.**get_repo**(*repo*, *basedir='/etc/yum.repos.d'*, ***kwargs*)

Display a repo from <basedir> (default basedir: /etc/yum.repos.d).

CLI Examples:

```
salt '*' pkg.get_repo myrepo
salt '*' pkg.get_repo myrepo basedir=/path/to/dir
```

salt.modules.yumpkg.**group_diff**(*name*)

New in version 2014.1.0: (Hydrogen)

Lists packages belonging to a certain group, and which are installed

CLI Example:

```
salt '*' pkg.group_diff 'Perl Support'
```

`salt.modules.yumpkg.`**`group_info`**`(`*name*`)`

New in version 2014.1.0: (Hydrogen)

Lists packages belonging to a certain group

CLI Example:

```
salt '*' pkg.group_info 'Perl Support'
```

`salt.modules.yumpkg.`**`group_install`**`(`*name*, *skip=()*, *include=()*, ***kwargs*`)`

New in version 2014.1.0: (Hydrogen)

Install the passed package group(s). This is basically a wrapper around pkg.install, which performs package group resolution for the user. This function is currently considered experimental, and should be expected to undergo changes.

name Package group to install. To install more than one group, either use a comma-separated list or pass the value as a python list.

CLI Examples:

```
salt '*' pkg.group_install 'Group 1'
salt '*' pkg.group_install 'Group 1,Group 2'
salt '*' pkg.group_install '["Group 1", "Group 2"]'
```

skip The name(s), in a list, of any packages that would normally be installed by the package group (“default” packages), which should not be installed. Can be passed either as a comma-separated list or a python list.

CLI Examples:

```
salt '*' pkg.group_install 'My Group' skip='foo,bar'
salt '*' pkg.group_install 'My Group' skip='["foo", "bar"]'
```

include The name(s), in a list, of any packages which are included in a group, which would not normally be installed (“optional” packages). Note that this will not enforce group membership; if you include packages which are not members of the specified groups, they will still be installed. Can be passed either as a comma-separated list or a python list.

CLI Examples:

```
salt '*' pkg.group_install 'My Group' include='foo,bar'
salt '*' pkg.group_install 'My Group' include='["foo", "bar"]'
```

Note: Because this is essentially a wrapper around pkg.install, any argument which can be passed to pkg.install may also be included here, and it will be passed along wholesale.

`salt.modules.yumpkg.`**`group_list`**`()`

New in version 2014.1.0: (Hydrogen)

Lists all groups known by yum on this system

CLI Example:

```
salt '*' pkg.group_list
```

`salt.modules.yumpkg.`**`install`**`(`*name=None*, *refresh=False*, *fromrepo=None*, *skip_verify=False*, *pkgs=None*, *sources=None*, ***kwargs*`)`

Install the passed package(s), add refresh=True to clean the yum database before package is installed.

name The name of the package to be installed. Note that this parameter is ignored if either "pkgs" or "sources" is passed. Additionally, please note that this option can only be used to install packages from a software repository. To install a package file manually, use the "sources" option.

32-bit packages can be installed on 64-bit systems by appending the architecture designation (`.i686`, `.i586`, etc.) to the end of the package name.

CLI Example:

```
salt '*' pkg.install <package name>
```

refresh Whether or not to update the yum database before executing.

skip_verify Skip the GPG verification check (e.g., `--nogpgcheck`)

version Install a specific version of the package, e.g. 1.2.3-4.el5. Ignored if "pkgs" or "sources" is passed.

Repository Options:

fromrepo Specify a package repository (or repositories) from which to install. (e.g., `yum --disablerepo='*' --enablerepo='somerepo'`)

enablerepo (ignored if `fromrepo` is specified) Specify a disabled package repository (or repositories) to enable. (e.g., `yum --enablerepo='somerepo'`)

disablerepo (ignored if `fromrepo` is specified) Specify an enabled package repository (or repositories) to disable. (e.g., `yum --disablerepo='somerepo'`)

Multiple Package Installation Options:

pkgs A list of packages to install from a software repository. Must be passed as a python list. A specific version number can be specified by using a single-element dict representing the package and its version.

CLI Examples:

```
salt '*' pkg.install pkgs='["foo", "bar"]'
salt '*' pkg.install pkgs='["foo", {"bar": "1.2.3-4.el5"}]'
```

sources A list of RPM packages to install. Must be passed as a list of dicts, with the keys being package names, and the values being the source URI or local path to the package.

CLI Example:

```
salt '*' pkg.install sources='[{"foo": "salt://foo.rpm"}, {"bar": "salt://bar.rpm"}]'
```

Returns a dict containing the new package names and versions:

```
{'<package>': {'old': '<old-version>',
               'new': '<new-version>'}}
```

`salt.modules.yumpkg.`**`latest_version`**`(`**names*, ***kwargs*`)`

Return the latest version of the named package available for upgrade or installation. If more than one package name is specified, a dict of name/version pairs is returned.

If the latest version of a given package is already installed, an empty string will be returned for that package.

A specific repo can be requested using the `fromrepo` keyword argument.

CLI Example:

```
salt '*' pkg.latest_version <package name>
salt '*' pkg.latest_version <package name> fromrepo=epel-testing
salt '*' pkg.latest_version <package1> <package2> <package3> ...
```

`salt.modules.yumpkg.`**`list_pkgs`**(*versions_as_list=False*, ***kwargs*)

List the packages currently installed in a dict:

```
{'<package_name>': '<version>'}
```

CLI Example:

```
salt '*' pkg.list_pkgs
```

`salt.modules.yumpkg.`**`list_repo_pkgs`**(**args*, ***kwargs*)

New in version 2014.1.0: (Hydrogen)

Returns all available packages. Optionally, package names can be passed and the results will be filtered to packages matching those names. This can be helpful in discovering the version or repo to specify in a pkg.installed state. The return data is a dictionary of repo names, with each repo having a list of dictionaries denoting the package name and version. An example of the return data would look like this:

```
{
    '<repo_name>': [
        {'<package1>': '<version1>'},
        {'<package2>': '<version2>'},
        {'<package3>': '<version3>'}
    ]
}
```

fromrepo [None] Only include results from the specified repo(s). Multiple repos can be specified, comma-separated.

CLI Example:

```
salt '*' pkg.list_repo_pkgs
salt '*' pkg.list_repo_pkgs foo bar baz
salt '*' pkg.list_repo_pkgs 'samba4*' fromrepo=base,updates
```

`salt.modules.yumpkg.`**`list_repos`**(*basedir='/etc/yum.repos.d'*)

Lists all repos in <basedir> (default: /etc/yum.repos.d/).

CLI Example:

```
salt '*' pkg.list_repos
```

`salt.modules.yumpkg.`**`list_upgrades`**(*refresh=True*, ***kwargs*)

Check whether or not an upgrade is available for all packages

CLI Example:

```
salt '*' pkg.list_upgrades
```

`salt.modules.yumpkg.`**`mod_repo`**(*repo*, *basedir=None*, ***kwargs*)

Modify one or more values for a repo. If the repo does not exist, it will be created, so long as the following values are specified:

repo name by which the yum refers to the repo

name a human-readable name for the repo

baseurl the URL for yum to reference

mirrorlist the URL for yum to reference

Key/Value pairs may also be removed from a repo's configuration by setting a key to a blank value. Bear in mind that a name cannot be deleted, and a baseurl can only be deleted if a mirrorlist is specified (or vice versa).

CLI Examples:

```
salt '*' pkg.mod_repo reponame enabled=1 gpgcheck=1
salt '*' pkg.mod_repo reponame basedir=/path/to/dir enabled=1
salt '*' pkg.mod_repo reponame baseurl= mirrorlist=http://host.com/
```

`salt.modules.yumpkg.`**`normalize_name`**`(`*name*`)`

Strips the architecture from the specified package name, if necessary. Circumstances where this would be done include:

- If the arch is 32 bit and the package name ends in a 32-bit arch.
- If the arch matches the OS arch, or is `noarch`.

CLI Example:

```
salt '*' pkg.normalize_name zsh.x86_64
```

`salt.modules.yumpkg.`**`purge`**`(`*name=None*, *pkgs=None*, ***kwargs*`)`

Package purges are not supported by yum, this function is identical to `pkg.remove`.

name The name of the package to be deleted.

Multiple Package Options:

pkgs A list of packages to delete. Must be passed as a python list. The `name` parameter will be ignored if this option is passed.

New in version 0.16.0.

Returns a dict containing the changes.

CLI Example:

```
salt '*' pkg.purge <package name>
salt '*' pkg.purge <package1>,<package2>,<package3>
salt '*' pkg.purge pkgs='["foo", "bar"]'
```

`salt.modules.yumpkg.`**`refresh_db`**`()`

Check the yum repos for updated packages

Returns:

- `True`: Updates are available
- `False`: An error occured
- `None`: No updates are available

CLI Example:

```
salt '*' pkg.refresh_db
```

`salt.modules.yumpkg.`**`remove`**`(`*name=None*, *pkgs=None*, ***kwargs*`)`

Remove packages with `yum -q -y remove`.

name The name of the package to be deleted.

Multiple Package Options:

pkgs A list of packages to delete. Must be passed as a python list. The `name` parameter will be ignored if this option is passed.

New in version 0.16.0.

Returns a dict containing the changes.

CLI Example:

```
salt '*' pkg.remove <package name>
salt '*' pkg.remove <package1>,<package2>,<package3>
salt '*' pkg.remove pkgs='["foo", "bar"]'
```

`salt.modules.yumpkg.`**`upgrade`**`(`*refresh=True*`)`

Run a full system upgrade, a yum upgrade

Return a dict containing the new package names and versions:

```
{'<package>': {'old': '<old-version>',
               'new': '<new-version>'}}
```

CLI Example:

```
salt '*' pkg.upgrade
```

`salt.modules.yumpkg.`**`upgrade_available`**`(`*name*`)`

Check whether or not an upgrade is available for a given package

CLI Example:

```
salt '*' pkg.upgrade_available <package name>
```

`salt.modules.yumpkg.`**`verify`**`(`**names*`)`

New in version 2014.1.0: (Hydrogen)

Runs an rpm -Va on a system, and returns the results in a dict

CLI Example:

```
salt '*' pkg.verify
```

`salt.modules.yumpkg.`**`version`**`(`**names*, ***kwargs*`)`

Returns a string representing the package version or an empty string if not installed. If more than one package name is specified, a dict of name/version pairs is returned.

CLI Example:

```
salt '*' pkg.version <package name>
salt '*' pkg.version <package1> <package2> <package3> ...
```

61.192 salt.modules.zcbuildout

61.192.1 Management of zc.buildout

This module is inspired from minitage's buildout maker

New in version Boron.

Note: The zc.buildout integration is still in beta; the API is subject to change

General notes

You have those following methods:

- upgrade_bootstrap
- bootstrap
- run_buildout
- buildout

`salt.modules.zcbuildout.`**`bootstrap`**`(`**a*, ***kw*`)`

Run the buildout bootstrap dance (python bootstrap.py).

directory directory to execute in

config alternative buildout configuration file to use

runas User used to run buildout as

env environment variables to set when running

buildout_ver force a specific buildout version (1 | 2)

test_release buildout accept test release

offline are we executing buildout in offline mode

distribute Forcing use of distribute

new_set Forcing use of setuptools >= 0.7

python path to a python executable to use in place of default (salt one)

onlyif Only execute cmd if statement on the host return 0

unless Do not execute cmd if statement on the host return 0

CLI Example:

```
salt '*' buildout.bootstrap /srv/mybuildout
```

`salt.modules.zcbuildout.`**`buildout`**`(`**a*, ***kw*`)`

Run buildout in a directory.

directory directory to execute in

config buildout config to use

parts specific buildout parts to run

runas user used to run buildout as

env environment variables to set when running

buildout_ver force a specific buildout version (1 | 2)

test_release buildout accept test release

new_set Forcing use of setuptools >= 0.7

distribute use distribute over setuptools if possible

offline does buildout run offline

python python to use

debug run buildout with -D debug flag

onlyif Only execute cmd if statement on the host return 0

unless Do not execute cmd if statement on the host return 0

newest run buildout in newest mode

verbose run buildout in verbose mode (-vvvvv)

CLI Example:

```
salt '*' buildout.buildout /srv/mybuildout
```

`salt.modules.zcbuildout.`**`run_buildout`**`(*a, **kw)`

Run a buildout in a directory.

directory directory to execute in

config alternative buildout configuration file to use

offline are we executing buildout in offline mode

runas user used to run buildout as

env environment variables to set when running

onlyif Only execute cmd if statement on the host return 0

unless Do not execute cmd if statement on the host return 0

newest run buildout in newest mode

force run buildout unconditionally

verbose run buildout in verbose mode (-vvvvv)

CLI Example:

```
salt '*' buildout.run_buildout /srv/mybuildout
```

`salt.modules.zcbuildout.`**`upgrade_bootstrap`**`(*a, **kw)`

Upgrade current bootstrap.py with the last released one.

Indeed, when we first run a buildout, a common source of problem is to have a locally stale bootstrap, we just try to grab a new copy

directory directory to execute in

offline are we executing buildout in offline mode

buildout_ver forcing to use a specific buildout version (1 | 2)

onlyif Only execute cmd if statement on the host return 0

unless Do not execute cmd if statement on the host return 0

CLI Example:

```
salt '*' buildout.upgrade_bootstrap /srv/mybuildout
```

61.193 salt.modules.zfs

Module for running ZFS command

61.194 salt.modules.zpool

Module for running ZFS zpool command

`salt.modules.zpool.`**`add`**(*pool_name*, *vdev*)

Add the specified vdev to the given pool

CLI Example:

```
salt '*' zpool.add myzpool /path/to/vdev
```

`salt.modules.zpool.`**`create`**(*pool_name*, **vdevs*)

Create a new storage pool

CLI Example:

```
salt '*' zpool.create myzpool /path/to/vdev1 [/path/to/vdev2] [...]
```

`salt.modules.zpool.`**`create_file_vdev`**(*size*, **vdevs*)

Creates file based `virtual devices` for a zpool

`*vdevs` is a list of full paths for mkfile to create

CLI Example:

```
salt '*' zpool.create_file_vdev 7g /path/to/vdev1 [/path/to/vdev2] [...]

Depending on file size this may take a while to return
```

`salt.modules.zpool.`**`destroy`**(*pool_name*)

Destroys a storage pool

CLI Example:

```
salt '*' zpool.destroy myzpool
```

`salt.modules.zpool.`**`exists`**(*pool_name*)

Check if a ZFS storage pool is active

CLI Example:

```
salt '*' zpool.exists myzpool
```

`salt.modules.zpool.`**`iostat`**(*name=''*)

Display I/O statistics for the given pools

CLI Example:

```
salt '*' zpool.iostat
```

`salt.modules.zpool.`**`replace`**(*pool_name*, *old*, *new*)

Replaces old device with new device.

CLI Example:

```
salt '*' zpool.replace myzpool /path/to/vdev1 /path/to/vdev2
```

`salt.modules.zpool.`**`scrub`**(*pool_name=None*)

Begin a scrub

CLI Example:

```
salt '*' zpool.scrub myzpool
```

salt.modules.zpool.**status**(*name=''*)
Return the status of the named zpool

CLI Example:

```
salt '*' zpool.status
```

salt.modules.zpool.**zpool_list**()
Return a list of all pools in the system with health status and space usage

CLI Example:

```
salt '*' zpool.zpool_list
```

61.195 salt.modules.zypper

Package support for openSUSE via the zypper package manager

salt.modules.zypper.**install**(*name=None*, *refresh=False*, *fromrepo=None*, *pkgs=None*, *sources=None*, ***kwargs*)
Install the passed package(s), add refresh=True to run 'zypper refresh' before package is installed.

name The name of the package to be installed. Note that this parameter is ignored if either "pkgs" or "sources" is passed. Additionally, please note that this option can only be used to install packages from a software repository. To install a package file manually, use the "sources" option.

CLI Example:

```
salt '*' pkg.install <package name>
```

refresh Whether or not to refresh the package database before installing.

fromrepo Specify a package repository to install from.

version Can be either a version number, or the combination of a comparison operator (<, >, <=, >=, =) and a version number (ex. '>1.2.3-4'). This parameter is ignored if "pkgs" or "sources" is passed.

Multiple Package Installation Options:

pkgs A list of packages to install from a software repository. Must be passed as a python list. A specific version number can be specified by using a single-element dict representing the package and its version. As with the `version` parameter above, comparison operators can be used to target a specific version of a package.

CLI Examples:

```
salt '*' pkg.install pkgs='["foo", "bar"]'
salt '*' pkg.install pkgs='["foo", {"bar": "1.2.3-4"}]'
salt '*' pkg.install pkgs='["foo", {"bar": "<1.2.3-4"}]'
```

sources A list of RPM packages to install. Must be passed as a list of dicts, with the keys being package names, and the values being the source URI or local path to the package.

CLI Example:

```
salt '*' pkg.install sources='[{"foo": "salt://foo.rpm"},{"bar": "salt://bar.rpm"}]'
```

Returns a dict containing the new package names and versions:

```
{'<package>': {'old': '<old-version>',
               'new': '<new-version>'}}
```

`salt.modules.zypper.`**`latest_version`**`(`**names*, ***kwargs*`)`

Return the latest version of the named package available for upgrade or installation. If more than one package name is specified, a dict of name/version pairs is returned.

If the latest version of a given package is already installed, an empty string will be returned for that package.

CLI Example:

```
salt '*' pkg.latest_version <package name>
salt '*' pkg.latest_version <package1> <package2> <package3> ...
```

`salt.modules.zypper.`**`list_pkgs`**`(`*versions_as_list=False*, ***kwargs*`)`

List the packages currently installed as a dict:

```
{'<package_name>': '<version>'}
```

CLI Example:

```
salt '*' pkg.list_pkgs
```

`salt.modules.zypper.`**`list_upgrades`**`(`*refresh=True*`)`

List all available package upgrades on this system

CLI Example:

```
salt '*' pkg.list_upgrades
```

`salt.modules.zypper.`**`purge`**`(`*name=None*, *pkgs=None*, ***kwargs*`)`

Recursively remove a package and all dependencies which were installed with it, this will call a `zypper -n remove -u`

name The name of the package to be deleted.

Multiple Package Options:

pkgs A list of packages to delete. Must be passed as a python list. The `name` parameter will be ignored if this option is passed.

New in version 0.16.0.

Returns a dict containing the changes.

CLI Example:

```
salt '*' pkg.purge <package name>
salt '*' pkg.purge <package1>,<package2>,<package3>
salt '*' pkg.purge pkgs='["foo", "bar"]'
```

`salt.modules.zypper.`**`refresh_db`**`()`

Just run a `zypper refresh`, return a dict:

```
{'<database name>': Bool}
```

CLI Example:

```
salt '*' pkg.refresh_db
```

`salt.modules.zypper.`**`remove`**`(`*name=None*, *pkgs=None*, ***kwargs*`)`

Remove packages with `zypper -n remove`

name The name of the package to be deleted.

Multiple Package Options:

pkgs A list of packages to delete. Must be passed as a python list. The `name` parameter will be ignored if this option is passed.

New in version 0.16.0.

Returns a dict containing the changes.

CLI Example:

```
salt '*' pkg.remove <package name>
salt '*' pkg.remove <package1>,<package2>,<package3>
salt '*' pkg.remove pkgs='["foo", "bar"]'
```

`salt.modules.zypper.`**`upgrade`**(*refresh=True*)

Run a full system upgrade, a zypper upgrade

Return a dict containing the new package names and versions:

```
{'<package>': {'old': '<old-version>',
               'new': '<new-version>'}}
```

CLI Example:

```
salt '*' pkg.upgrade
```

`salt.modules.zypper.`**`upgrade_available`**(*name*)

Check whether or not an upgrade is available for a given package

CLI Example:

```
salt '*' pkg.upgrade_available <package name>
```

`salt.modules.zypper.`**`version`**(**names*, ***kwargs*)

Returns a string representing the package version or an empty string if not installed. If more than one package name is specified, a dict of name/version pairs is returned.

CLI Example:

```
salt '*' pkg.version <package name>
salt '*' pkg.version <package1> <package2> <package3> ...
```

CHAPTER 62

Returners

By default the return values of the commands sent to the Salt minions are returned to the salt-master. But since the commands executed on the Salt minions are detached from the call on the Salt master, anything at all can be done with the results data.

This is where the returner interface comes in. Returners are modules called in addition to returning the data to the Salt master.

The returner interface allows the return data to be sent to any system that can receive data. This means that return data can be sent to a Redis server, a MongoDB server, a MySQL server, or any system!

See also:

Full list of builtin returners

62.1 Using Returners

All commands will return the command data back to the master. Adding more returners will ensure that the data is also sent to the specified returner interfaces.

Specifying what returners to use is done when the command is invoked:

```
salt '*' test.ping --return redis_return
```

This command will ensure that the redis_return returner is used.

It is also possible to specify multiple returners:

```
salt '*' test.ping --return mongo_return,redis_return,cassandra_return
```

In this scenario all three returners will be called and the data from the test.ping command will be sent out to the three named returners.

62.2 Writing a Returner

A returner is a module which contains a returner function, the returner function must accept a single argument. this argument is the return data from the called minion function. So if the minion function `test.ping` is called the value of the argument will be `True`.

A simple returner is implemented here:

```
import redis
import json

def returner(ret):
    '''
    Return information to a redis server
    '''
    # Get a redis connection
    serv = redis.Redis(
                host='redis-serv.example.com',
                port=6379,
                db='0')
    serv.sadd("%(id)s:jobs" % ret, ret['jid'])
    serv.set("%(jid)s:%(id)s" % ret, json.dumps(ret['return']))
    serv.sadd('jobs', ret['jid'])
    serv.sadd(ret['jid'], ret['id'])
```

This simple example of a returner set to send the data to a redis server serializes the data as json and sets it in redis.

You can place your custom returners in a `_returners` directory within the `file_roots` specified by the master config file. These custom returners are distributed when `state.highstate` is run, or by executing the `saltutil.sync_returners` or `saltutil.sync_all` functions.

Any custom returners which have been synced to a minion, that are named the same as one of Salt's default set of returners, will take the place of the default returner with the same name. Note that a returner's default name is its filename (i.e. `foo.py` becomes returner `foo`), but that its name can be overridden by using a *__virtual__ function*. A good example of this can be found in the redis returner, which is named `redis_return.py` but is loaded as simply `redis`:

```
try:
    import redis
    HAS_REDIS = True
except ImportError:
    HAS_REDIS = False

def __virtual__():
    if not HAS_REDIS:
        return False
    return 'redis'
```

62.2.1 Examples

The collection of built-in Salt returners can be found here: https://github.com/saltstack/salt/blob/develop/salt/returners

Full list of builtin returner modules

`carbon_return`	Take data from salt and "return" it into a carbon receiver
`cassandra_return`	Return data to a Cassandra ColumnFamily
`couchdb_return`	Simple returner for CouchDB.
`local`	The local returner is used to test the returner interface, it just prints the
`memcache_return`	Return data to a memcache server
`mongo_future_return`	Return data to a mongodb server
`mongo_return`	Return data to a mongodb server
`mysql`	Return data to a mysql server
`postgres`	Return data to a postgresql server
`redis_return`	Return data to a redis server
`sentry_return`	Salt returner that report execution results back to sentry.
`smtp_return`	Return salt data via email
`sqlite3_return`	Insert minion return data into a sqlite3 database
`syslog_return`	Return data to the host operating system's syslog facility

63.1 salt.returners.carbon_return

Take data from salt and "return" it into a carbon receiver

Add the following configuration to your minion configuration files:

```
carbon.host: <server ip address>
carbon.port: 2003
```

`salt.returners.carbon_return.`**`returner`**`(`*ret*`)`
Return data to a remote carbon server using the text metric protocol

63.2 salt.returners.cassandra_return

Return data to a Cassandra ColumnFamily

Here's an example Keyspace / ColumnFamily setup that works with this returner:

```
create keyspace salt;
use salt;
create column family returns
  with key_validation_class='UTF8Type'
```

```
and comparator='UTF8Type'
and default_validation_class='UTF8Type';
```

Required python modules: pycassa

`salt.returners.cassandra_return.`**`returner`**`(`*ret*`)`
Return data to a Cassandra ColumnFamily

63.3 salt.returners.couchdb_return

Simple returner for CouchDB. Optional configuration settings are listed below, along with sane defaults.

couchdb.db: 'salt' couchdb.url: 'http://salt:5984/'

`salt.returners.couchdb_return.`**`ensure_views`**`()`
This function makes sure that all the views that should exist in the design document do exist.

`salt.returners.couchdb_return.`**`get_fun`**`(`*fun*`)`
Return a dict with key being minion and value being the job details of the last run of function 'fun'.

`salt.returners.couchdb_return.`**`get_jid`**`(`*jid*`)`
Get the document with a given JID.

`salt.returners.couchdb_return.`**`get_jids`**`()`
List all the jobs that we have..

`salt.returners.couchdb_return.`**`get_minions`**`()`
Return a list of minion identifiers from a request of the view.

`salt.returners.couchdb_return.`**`get_valid_salt_views`**`()`
Returns a dict object of views that should be part of the salt design document.

`salt.returners.couchdb_return.`**`returner`**`(`*ret*`)`
Take in the return and shove it into the couchdb database.

`salt.returners.couchdb_return.`**`set_salt_view`**`()`
Helper function that sets the salt design document. Uses get_valid_salt_views and some hardcoded values.

63.4 salt.returners.local

The local returner is used to test the returner interface, it just prints the return data to the console to verify that it is being passed properly

`salt.returners.local.`**`returner`**`(`*ret*`)`
Print the return data to the terminal to verify functionality

63.5 salt.returners.memcache_return

Return data to a memcache server

To enable this returner the minion will need the python client for memcache installed and the following values configured in the minion or master config, these are the defaults:

> memcache.host: 'localhost' memcache.port: '11211'

python2-memcache uses 'localhost' and '11211' as syntax on connection.

`salt.returners.memcache_return.`**`get_fun`**(*fun*)
Return a dict of the last function called for all minions

`salt.returners.memcache_return.`**`get_jid`**(*jid*)
Return the information returned when the specified job id was executed

`salt.returners.memcache_return.`**`get_jids`**()
Return a list of all job ids

`salt.returners.memcache_return.`**`get_load`**(*jid*)
Return the load data that marks a specified jid

`salt.returners.memcache_return.`**`get_minions`**()
Return a list of minions

`salt.returners.memcache_return.`**`returner`**(*ret*)
Return data to a memcache data store

`salt.returners.memcache_return.`**`save_load`**(*jid*, *load*)
Save the load to the specified jid

63.6 salt.returners.mongo_future_return

Return data to a mongodb server

Required python modules: pymongo

This returner will send data from the minions to a MongoDB server. To configure the settings for your MongoDB server, add the following lines to the minion config files:

```
mongo.db: <database name>
mongo.host: <server ip address>
mongo.user: <MongoDB username>
mongo.password: <MongoDB user password>
mongo.port: 27017
```

This mongo returner is being developed to replace the default mongodb returner in the future and should not be considered API stable yet.

`salt.returners.mongo_future_return.`**`get_fun`**(*fun*)
Return the most recent jobs that have executed the named function

`salt.returners.mongo_future_return.`**`get_jid`**(*jid*)
Return the return information associated with a jid

`salt.returners.mongo_future_return.`**`get_jids`**()
Return a list of job ids

`salt.returners.mongo_future_return.`**`get_load`**(*jid*)
Return the load associated with a given job id

`salt.returners.mongo_future_return.`**`get_minions`**()
Return a list of minions

`salt.returners.mongo_future_return.`**`returner`**(*ret*)
Return data to a mongodb server

`salt.returners.mongo_future_return.`**`save_load`**(*jid*, *load*)
Save the load for a given job id

63.7 salt.returners.mongo_return

Return data to a mongodb server

Required python modules: pymongo

This returner will send data from the minions to a MongoDB server. To configure the settings for your MongoDB server, add the following lines to the minion config files:

```
mongo.db: <database name>
mongo.host: <server ip address>
mongo.user: <MongoDB username>
mongo.password: <MongoDB user password>
mongo.port: 27017
```

`salt.returners.mongo_return.`**`get_fun`**(*fun*)
Return the most recent jobs that have executed the named function

`salt.returners.mongo_return.`**`get_jid`**(*jid*)
Return the return information associated with a jid

`salt.returners.mongo_return.`**`returner`**(*ret*)
Return data to a mongodb server

63.8 salt.returners.mysql

Return data to a mysql server

maintainer Dave Boucha <dave@saltstack.com>, Seth House <shouse@saltstack.com>

maturity new

depends python-mysqldb

platform all

To enable this returner the minion will need the python client for mysql installed and the following values configured in the minion or master config, these are the defaults:

```
mysql.host: 'salt'
mysql.user: 'salt'
mysql.pass: 'salt'
mysql.db: 'salt'
mysql.port: 3306
```

Use the following mysql database schema:

```
CREATE DATABASE  `salt`
  DEFAULT CHARACTER SET utf8
  DEFAULT COLLATE utf8_general_ci;

USE `salt`;

--
-- Table structure for table `jids`
--

DROP TABLE IF EXISTS `jids`;
CREATE TABLE `jids` (
  `jid` varchar(255) NOT NULL,
  `load` mediumtext NOT NULL,
  UNIQUE KEY `jid` (`jid`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

--
-- Table structure for table `salt_returns`
--

DROP TABLE IF EXISTS `salt_returns`;
CREATE TABLE `salt_returns` (
  `fun` varchar(50) NOT NULL,
  `jid` varchar(255) NOT NULL,
  `return` mediumtext NOT NULL,
  `id` varchar(255) NOT NULL,
  `success` varchar(10) NOT NULL,
  `full_ret` mediumtext NOT NULL,
  `alter_time` TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  KEY `id` (`id`),
  KEY `jid` (`jid`),
  KEY `fun` (`fun`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Required python modules: MySQLdb

`salt.returners.mysql.`**`get_fun`**`(`*fun*`)`
Return a dict of the last function called for all minions

`salt.returners.mysql.`**`get_jid`**`(`*jid*`)`
Return the information returned when the specified job id was executed

`salt.returners.mysql.`**`get_jids`**`()`
Return a list of all job ids

`salt.returners.mysql.`**`get_load`**`(`*jid*`)`
Return the load data that marks a specified jid

`salt.returners.mysql.`**`get_minions`**`()`
Return a list of minions

`salt.returners.mysql.`**`returner`**`(`*ret*`)`
Return data to a mysql server

`salt.returners.mysql.`**`save_load`**`(`*jid*, *load*`)`
Save the load to the specified jid id

63.9 salt.returners.postgres

Return data to a postgresql server

maintainer None

maturity New

depends psycopg2

platform all

To enable this returner the minion will need the psycopg2 installed and the following values configured in the minion or master config:

```
returner.postgres.host: 'salt'
returner.postgres.user: 'salt'
returner.postgres.passwd: 'salt'
returner.postgres.db: 'salt'
returner.postgres.port: 5432
```

Running the following commands as the postgres user should create the database correctly:

```
psql << EOF
CREATE ROLE salt WITH PASSWORD 'salt';
CREATE DATABASE salt WITH OWNER salt;
EOF

psql -h localhost -U salt << EOF
--
-- Table structure for table 'jids'
--

DROP TABLE IF EXISTS jids;
CREATE TABLE jids (
  jid   bigint PRIMARY KEY,
  load  text NOT NULL
);

--
-- Table structure for table 'salt_returns'
--

DROP TABLE IF EXISTS salt_returns;
CREATE TABLE salt_returns (
  added     TIMESTAMP WITH TIME ZONE DEFAULT now(),
  fun       text NOT NULL,
  jid       varchar(20) NOT NULL,
  return    text NOT NULL,
  id        text NOT NULL,
  success   boolean
);
CREATE INDEX ON salt_returns (added);
CREATE INDEX ON salt_returns (id);
CREATE INDEX ON salt_returns (jid);
CREATE INDEX ON salt_returns (fun);
EOF
```

Required python modules: psycopg2

`salt.returners.postgres.`**`get_fun`**`(`*fun*`)`
Return a dict of the last function called for all minions

`salt.returners.postgres.`**`get_jid`**`(`*jid*`)`
Return the information returned when the specified job id was executed

`salt.returners.postgres.`**`get_jids`**`()`
Return a list of all job ids

`salt.returners.postgres.`**`get_load`**`(`*jid*`)`
Return the load data that marks a specified jid

`salt.returners.postgres.`**`get_minions`**`()`
Return a list of minions

`salt.returners.postgres.`**`returner`**`(`*ret*`)`
Return data to a postgres server

`salt.returners.postgres.`**`save_load`**`(`*jid*, *load*`)`
Save the load to the specified jid id

63.10 salt.returners.redis_return

Return data to a redis server

To enable this returner the minion will need the python client for redis installed and the following values configured in the minion or master config, these are the defaults:

> redis.db: '0' redis.host: 'salt' redis.port: 6379

`salt.returners.redis_return.`**`get_fun`**`(`*fun*`)`
Return a dict of the last function called for all minions

`salt.returners.redis_return.`**`get_jid`**`(`*jid*`)`
Return the information returned when the specified job id was executed

`salt.returners.redis_return.`**`get_jids`**`()`
Return a list of all job ids

`salt.returners.redis_return.`**`get_load`**`(`*jid*`)`
Return the load data that marks a specified jid

`salt.returners.redis_return.`**`get_minions`**`()`
Return a list of minions

`salt.returners.redis_return.`**`returner`**`(`*ret*`)`
Return data to a redis data store

`salt.returners.redis_return.`**`save_load`**`(`*jid*, *load*`)`
Save the load to the specified jid

63.11 salt.returners.sentry_return

Salt returner that report execution results back to sentry. The returner will inspect the payload to identify errors and flag them as such.

Pillar need something like:

```
raven:
  servers:
    - http://192.168.1.1
    - https://sentry.example.com
  public_key: deadbeefdeadbeefdeadbeefdeadbeef
  secret_key: beefdeadbeefdeadbeefdeadbeefdead
  project: 1
  tags:
    - os
    - master
    - saltversion
    - cpuarch
```

and http://pypi.python.org/pypi/raven installed

The tags list (optional) specifies grains items that will be used as sentry tags, allowing tagging of events in the sentry ui.

`salt.returners.sentry_return.`**`returner`**`(`*ret*`)`

Log outcome to sentry. The returner tries to identify errors and report them as such. All other messages will be reported at info level.

63.12 salt.returners.smtp_return

Return salt data via email

The following fields can be set in the minion conf file:

> smtp.from (required) smtp.to (required) smtp.host (required) smtp.port (optional, defaults to 25) smtp.username (optional) smtp.password (optional) smtp.tls (optional, defaults to False) smtp.subject (optional, but helpful) smtp.fields (optional)

There are a few things to keep in mind:

- If a username is used, a password is also required.
- You should at least declare a subject, but you don't have to.
- smtp.fields lets you include the value(s) of various fields in the subject line of the email. These are comma-delimited. For instance:

  > smtp.fields: id,fun

  ...will display the id of the minion and the name of the function in the subject line. You may also use 'jid' (the job id), but it is generally recommended not to use 'return', which contains the entire return data structure (which can be very large).

`salt.returners.smtp_return.`**`returner`**`(`*ret*`)`

Send an email with the data

63.13 salt.returners.sqlite3

Insert minion return data into a sqlite3 database

> **maintainer** Mickey Malone <mickey.malone@gmail.com>
>
> **maturity** New
>
> **depends** None
>
> **platform** All

Sqlite3 is a serverless database that lives in a single file. In order to use this returner the database file must exist, have the appropriate schema defined, and be accessible to the user whom the minion process is running as. This returner requires the following values configured in the master or minion config:

```
returner.sqlite3.database: /usr/lib/salt/salt.db
returner.sqlite3.timeout: 5.0
```

Use the commands to create the sqlite3 database and tables:

```
sqlite3 /usr/lib/salt/salt.db << EOF
--
-- Table structure for table 'jids'
--

CREATE TABLE jids (
  jid TEXT PRIMARY KEY,
  load TEXT NOT NULL
  );

--
-- Table structure for table 'salt_returns'
--

CREATE TABLE salt_returns (
  fun TEXT KEY,
  jid TEXT KEY,
  id TEXT KEY,
  fun_args TEXT,
  date TEXT NOT NULL,
  full_ret TEXT NOT NULL,
  success TEXT NOT NULL
  );
EOF
```

`salt.returners.sqlite3_return.`**`get_fun`**(*fun*)
Return a dict of the last function called for all minions

`salt.returners.sqlite3_return.`**`get_jid`**(*jid*)
Return the information returned from a specified jid

`salt.returners.sqlite3_return.`**`get_jids`**()
Return a list of all job ids

`salt.returners.sqlite3_return.`**`get_load`**(*jid*)
Return the load from a specified jid

`salt.returners.sqlite3_return.`**`get_minions`**()
Return a list of minions

`salt.returners.sqlite3_return.`**`returner`**(*ret*)
Insert minion return data into the sqlite3 database

`salt.returners.sqlite3_return.`**`save_load`**(*jid*, *load*)
Save the load to the specified jid

63.14 salt.returners.syslog_return

Return data to the host operating system's syslog facility

Required python modules: syslog, json

The syslog returner simply reuses the operating system's syslog facility to log return data

`salt.returners.syslog_return.`**`returner`**(*ret*)
Return data to the local syslog

CHAPTER 64

File State Backups

In 0.10.2 a new feature was added for backing up files that are replaced by the file.managed and file.recurse states. The new feature is called the backup mode. Setting the backup mode is easy, but it can be set in a number of places.

The backup_mode can be set in the minion config file:

```
backup_mode: minion
```

Or it can be set for each file:

```
/etc/ssh/sshd_config:
  file.managed:
    - source: salt://ssh/sshd_config
    - backup: minion
```

64.1 Backed-up Files

The files will be saved in the minion cachedir under the directory named `file_backup`. The files will be in the location relative to where they were under the root filesystem and be appended with a timestamp. This should make them easy to browse.

64.2 Interacting with Backups

Starting with version 0.17.0, it will be possible to list, restore, and delete previously-created backups.

64.2.1 Listing

The backups for a given file can be listed using `file.list_backups`:

```
# salt foo.bar.com file.list_backups /tmp/foo.txt
foo.bar.com:
    ----------
    0:
        ----------
        Backup Time:
            Sat Jul 27 2013 17:48:41.738027
        Location:
            /var/cache/salt/minion/file_backup/tmp/foo.txt_Sat_Jul_27_17:48:41_738027_2013
        Size:
            13
    1:
        ----------
        Backup Time:
            Sat Jul 27 2013 17:48:28.369804
        Location:
            /var/cache/salt/minion/file_backup/tmp/foo.txt_Sat_Jul_27_17:48:28_369804_2013
        Size:
            35
```

64.2.2 Restoring

Restoring is easy using `file.restore_backup`, just pass the path and the numeric id found with `file.list_backups`:

```
# salt foo.bar.com file.restore_backup /tmp/foo.txt 1
foo.bar.com:
    ----------
    comment:
        Successfully restored /var/cache/salt/minion/file_backup/tmp/foo.txt_Sat_Jul_27_17:48:28_3698
    result:
        True
```

The existing file will be backed up, just in case, as can be seen if `file.list_backups` is run again:

```
# salt foo.bar.com file.list_backups /tmp/foo.txt
foo.bar.com:
    ----------
    0:
        ----------
        Backup Time:
            Sat Jul 27 2013 18:00:19.822550
        Location:
            /var/cache/salt/minion/file_backup/tmp/foo.txt_Sat_Jul_27_18:00:19_822550_2013
        Size:
            53
    1:
        ----------
        Backup Time:
            Sat Jul 27 2013 17:48:41.738027
        Location:
            /var/cache/salt/minion/file_backup/tmp/foo.txt_Sat_Jul_27_17:48:41_738027_2013
        Size:
            13
    2:
        ----------
        Backup Time:
            Sat Jul 27 2013 17:48:28.369804
        Location:
            /var/cache/salt/minion/file_backup/tmp/foo.txt_Sat_Jul_27_17:48:28_369804_2013
        Size:
            35
```

Note: Since no state is being run, restoring a file will not trigger any watches for the file. So, if you are restoring a config file for a service, it will likely still be necessary to run a `service.restart`.

64.2.3 Deleting

Deleting backups can be done using mod:*file.delete_backup* <*salt.modules.file.delete_backup*>:

```
# salt foo.bar.com file.delete_backup /tmp/foo.txt 0
foo.bar.com:
    ----------
    comment:
        Successfully removed /var/cache/salt/minion/file_backup/tmp/foo.txt_Sat_Jul_27_18:00:19_82255
    result:
        True
```

CHAPTER 65

Extending External SLS Data

Sometimes a state defined in one SLS file will need to be modified from a separate SLS file. A good example of this is when an argument needs to be overwritten or when a service needs to watch an additional state.

65.1 The Extend Declaration

The standard way to extend is via the extend declaration. The extend declaration is a top level declaration like `include` and encapsulates ID declaration data included from other SLS files. A standard extend looks like this:

```
include:
  - http
  - ssh

extend:
  apache:
    file:
      - name: /etc/httpd/conf/httpd.conf
      - source: salt://http/httpd2.conf
  ssh-server:
    service:
      - watch:
        - file: /etc/ssh/banner

/etc/ssh/banner:
  file.managed:
    - source: salt://ssh/banner
```

A few critical things happened here, first off the SLS files that are going to be extended are included, then the extend dec is defined. Under the extend dec 2 IDs are extended, the apache ID's file state is overwritten with a new name and source. Than the ssh server is extended to watch the banner file in addition to anything it is already watching.

65.2 Extend is a Top Level Declaration

This means that `extend` can only be called once in an sls, if if is used twice then only one of the extend blocks will be read. So this is WRONG:

```
include:
  - http
  - ssh
```

```
extend:
  apache:
    file:
      - name: /etc/httpd/conf/httpd.conf
      - source: salt://http/httpd2.conf
# Second extend will overwrite the first!! Only make one
extend:
  ssh-server:
    service:
      - watch:
        - file: /etc/ssh/banner
```

65.3 The Requisite "in" Statement

Since one of the most common things to do when extending another SLS is to add states for a service to watch, or anything for a watcher to watch, the requisite in statement was added to 0.9.8 to make extending the watch and require lists easier. The ssh-server extend statement above could be more cleanly defined like so:

```
include:
  - ssh

/etc/ssh/banner:
  file.managed:
    - source: salt://ssh/banner
    - watch_in:
      - service: ssh-server
```

65.4 Rules to Extend By

There are a few rules to remember when extending states:

1. Always include the SLS being extended with an include declaration
2. Requisites (watch and require) are appended to, everything else is overwritten
3. extend is a top level declaration, like an ID declaration, cannot be declared twice in a single SLS
4. Many IDs can be extended under the extend declaration

CHAPTER 66

Failhard Global Option

Normally, when a state fails Salt continues to execute the remainder of the defined states and will only refuse to execute states that require the failed state.

But the situation may exist, where you would want all state execution to stop if a single state execution fails. The capability to do this is called `failing hard`.

66.1 State Level Failhard

A single state can have a failhard set, this means that if this individual state fails that all state execution will immediately stop. This is a great thing to do if there is a state that sets up a critical config file and setting a require for each state that reads the config would be cumbersome. A good example of this would be setting up a package manager early on:

```
/etc/yum.repos.d/company.repo:
  file.managed:
    - source: salt://company/yumrepo.conf
    - user: root
    - group: root
    - mode: 644
    - order: 1
    - failhard: True
```

In this situation, the yum repo is going to be configured before other states, and if it fails to lay down the config file, than no other states will be executed.

66.2 Global Failhard

It may be desired to have failhard be applied to every state that is executed, if this is the case, then failhard can be set in the master configuration file. Setting failhard in the master configuration file will result in failing hard when any minion gathering states from the master have a state fail.

This is NOT the default behavior, normally Salt will only fail states that require a failed state.

Using the global failhard is generally not recommended, since it can result in states not being executed or even checked. It can also be confusing to see states failhard if an admin is not actively aware that the failhard has been set.

To use the global failhard set failhard: True in the master configuration file.

Highstate data structure definitions

67.1 The Salt State Tree

Top file The main state file that instructs minions what environment and modules to use during state execution.

Configurable via `state_top`.

See also:

A detailed description of the top file

State tree A collection of `SLS` files that live under the directory specified in `file_roots`. A state tree can be organized into `SLS modules`.

67.1.1 Include declaration

Include declaration Defines a list of *module reference* strings to include in this *SLS*.

Occurs only in the top level of the highstate structure.

Example:

```
include:
  - edit.vim
  - http.server
```

67.1.2 Module reference

Module reference The name of a SLS module defined by a separate SLS file and residing on the Salt Master. A module named `edit.vim` is a reference to the SLS file `salt://edit/vim.sls`.

67.1.3 ID declaration

ID declaration Defines an individual highstate component. Always references a value of a dictionary containing keys referencing *state declarations* and *requisite declarations*. Can be overridden by a *name declaration* or a *names declaration*.

Occurs on the top level or under the *extend declaration*.

Must be unique across entire state tree. If the same ID declaration is used twice, only the first one matched will be used. All subsequent ID declarations with the same name will be ignored.

Note: Naming gotchas

Until 0.9.6, IDs could **not** contain a dot, otherwise highstate summary output was unpredictable. (It was fixed in versions 0.9.7 and above)

67.1.4 Extend declaration

Extend declaration Extends a *name declaration* from an included `SLS module`. The keys of the extend declaration always define existing *ID declarations* which have been defined in included `SLS modules`.

Occurs only in the top level and defines a dictionary.

Extend declarations are useful for adding-to or overriding parts of a *state declaration* that is defined in another `SLS` file. In the following contrived example, the shown `mywebsite.sls` file is `include` -ing and `extend` -ing the `apache.sls` module in order to add a `watch` declaration that will restart Apache whenever the Apache configuration file, `mywebsite` changes.

```
include:
  - apache

extend:
  apache:
    service:
      - watch:
        - file: mywebsite

mywebsite:
  file:
    - managed
```

See also:

watch_in and require_in

Sometimes it is more convenient to use the *watch_in* or *require_in* syntax instead of extending another `SLS` file.

State Requisites

67.1.5 State declaration

State declaration A list which contains one string defining the *function declaration* and any number of *function arg declaration* dictionaries.

Can, optionally, contain a number of additional components like the name override components — *name* and *names*. Can also contain *requisite declarations*.

Occurs under an *ID declaration*.

67.1.6 Requisite declaration

Requisite declaration A list containing *requisite references*.

Used to build the action dependency tree. While Salt states are made to execute in a deterministic order, this order is managed by requiring and watching other Salt states.

Occurs as a list component under a *state declaration* or as a key under an *ID declaration*.

67.1.7 Requisite reference

Requisite reference A single key dictionary. The key is the name of the referenced *state declaration* and the value is the ID of the referenced *ID declaration*.

Occurs as a single index in a *requisite declaration* list.

67.1.8 Function declaration

Function declaration The name of the function to call within the state. A state declaration can contain only a single function declaration.

For example, the following state declaration calls the `installed` function in the `pkg` state module:

```
httpd:
  pkg.installed
```

The function can be declared inline with the state as a shortcut, but the actual data structure is better referenced in this form:

```
httpd:
  pkg:
    - installed
```

Where the function is a string in the body of the state declaration. Technically when the function is declared in dot notation the compiler converts it to be a string in the state declaration list. Note that the use of the first example more than once in an ID declaration is invalid yaml.

INVALID:

```
httpd:
  pkg.installed
  service.running
```

When passing a function without arguments and another state declaration within a single ID declaration, then the long or “standard” format needs to be used since otherwise it does not represent a valid data structure.

VALID:

```
httpd:
  pkg:
    - installed
  service:
    - running
```

Occurs as the only index in the *state declaration* list.

67.1.9 Function arg declaration

Function arg declaration A single key dictionary referencing a Python type which is to be passed to the named *function declaration* as a parameter. The type must be the data type expected by the function.

Occurs under a *function declaration*.

For example in the following state declaration `user`, `group`, and `mode` are passed as arguments to the `managed` function in the `file` state module:

```
/etc/http/conf/http.conf:
  file.managed:
    - user: root
    - group: root
    - mode: 644
```

67.1.10 Name declaration

Name declaration Overrides the `name` argument of a *state declaration*. If `name` is not specified the *ID declaration* satisfies the `name` argument.

The name is always a single key dictionary referencing a string.

Overriding `name` is useful for a variety of scenarios.

For example, avoiding clashing ID declarations. The following two state declarations cannot both have `/etc/motd` as the ID declaration:

```
motd_perms:
  file.managed:
    - name: /etc/motd
    - mode: 644

motd_quote:
  file.append:
    - name: /etc/motd
    - text: "Of all smells, bread; of all tastes, salt."
```

Another common reason to override `name` is if the ID declaration is long and needs to be referenced in multiple places. In the example below it is much easier to specify `mywebsite` than to specify `/etc/apache2/sites-available/mywebsite.com` multiple times:

```
mywebsite:
  file.managed:
    - name: /etc/apache2/sites-available/mywebsite.com
    - source: salt://mywebsite.com

a2ensite mywebsite.com:
  cmd.wait:
    - unless: test -L /etc/apache2/sites-enabled/mywebsite.com
    - watch:
      - file: mywebsite

apache2:
  service:
    - running
    - watch:
      - file: mywebsite
```

67.1.11 Names declaration

Names declaration Expands the contents of the containing *state declaration* into multiple state declarations, each with its own name.

For example, given the following state declaration:

```
python-pkgs:
  pkg.installed:
    - names:
      - python-django
      - python-crypto
      - python-yaml
```

Once converted into the lowstate data structure the above state declaration will be expanded into the following three state declarations:

```
python-django:
  pkg.installed

python-crypto:
  pkg.installed

python-yaml:
  pkg.installed
```

67.2 Large example

Here is the layout in yaml using the names of the highdata structure components.

```
<Include Declaration>:
  - <Module Reference>
  - <Module Reference>

<Extend Declaration>:
  <ID Declaration>:
    [<overrides>]


# standard declaration

<ID Declaration>:
  <State Declaration>:
    - <Function>
    - <Function Arg>
    - <Function Arg>
    - <Function Arg>
    - <Name>: <name>
    - <Requisite Declaration>:
      - <Requisite Reference>
      - <Requisite Reference>


# inline function and names

<ID Declaration>:
  <State Declaration>.<Function>:
    - <Function Arg>
    - <Function Arg>
    - <Function Arg>
    - <Names>:
      - <name>
      - <name>
      - <name>
    - <Requisite Declaration>:
      - <Requisite Reference>
      - <Requisite Reference>


# multiple states for single id

<ID Declaration>:
  <State Declaration>:
    - <Function>
    - <Function Arg>
    - <Name>: <name>
    - <Requisite Declaration>:
      - <Requisite Reference>
  <State Declaration>:
    - <Function>
    - <Function Arg>
    - <Names>:
      - <name>
      - <name>
    - <Requisite Declaration>:
      - <Requisite Reference>
```

CHAPTER 68

Include and Exclude

Salt sls files can include other sls files and exclude sls files that have been otherwise included. This allows for an sls file to easily extend or manipulate other sls files.

68.1 Include

When other sls files are included, everything defined in the included sls file will be added to the state run. When including define a list of sls formulas to include:

```
include:
  - http
  - libvirt
```

The include statement will include sls formulas from the same environment that the including sls formula is in. But the environment can be explicitly defined in the configuration to override the running environment, therefore if an sls formula needs to be included from an external environment named "dev" the following syntax is used:

```
include:
  - dev: http
```

68.2 Relative Include

In Salt 0.16.0 the capability to include sls formulas which are relative to the running sls formula was added, simply precede the formula name with a .:

```
include:
  - .virt
  - .virt.hyper
```

68.3 Exclude

The exclude statement, added in Salt 0.10.3 allows an sls to hard exclude another sls file or a specific id. The component is excluded after the high data has been compiled, so nothing should be able to override an exclude.

Since the exclude can remove an id or an sls the type of component to exclude needs to be defined. an exclude statement that verifies that the running highstate does not contain the *http* sls and the */etc/vimrc* id would look like this:

```
exclude:
  - sls: http
  - id: /etc/vimrc
```

CHAPTER 69

State Enforcement

Salt offers an optional interface to manage the configuration or "state" of the Salt minions. This interface is a fully capable mechanism used to enforce the state of systems from a central manager.

The Salt state system is made to be accurate, simple, and fast. And like the rest of the Salt system, Salt states are highly modular.

69.1 State management

State management, also frequently called software configuration management (SCM), is a program that puts and keeps a system into a predetermined state. It installs software packages, starts or restarts services, or puts configuration files in place and watches them for changes.

Having a state management system in place allows you to easily and reliably configure and manage a few servers or a few thousand servers. It allows you to keep that configuration under version control.

Salt States is an extension of the Salt Modules that we discussed in the previous *remote execution* tutorial. Instead of calling one-off executions the state of a system can be easily defined and then enforced.

69.2 Understanding the Salt State System Components

The Salt state system is comprised of a number of components. As a user, an understanding of the SLS and renderer systems are needed. But as a developer, an understanding of Salt states and how to write the states is needed as well.

Note: States are compiled and executed only on minions that have been targeted. To execute things on masters, see runners.

69.2.1 Salt SLS System

SLS The primary system used by the Salt state system is the SLS system. SLS stands for **S**a**L**t **S**tate.

The Salt States are files which contain the information about how to configure Salt minions. The states are laid out in a directory tree and can be written in many different formats.

The contents of the files and they way they are laid out is intended to be as simple as possible while allowing for maximum flexibility. The files are laid out in states and contains information about how the minion needs to be configured.

SLS File Layout

SLS files are laid out in the Salt file server. A simple layout can look like this:

```
top.sls
ssh.sls
sshd_config
users/init.sls
users/admin.sls
salt/init.sls
salt/master.sls
```

This example shows the core concepts of file layout. The top file is a key component and is used with Salt matchers to match SLS states with minions. The `.sls` files are states. The rest of the files are seen by the Salt master as just files that can be downloaded.

The states are translated into dot notation, so the `ssh.sls` file is seen as the ssh state, the `users/admin.sls` file is seen as the users.admin states.

The init.sls files are translated to be the state name of the parent directory, so the `salt/init.sls` file translates to the Salt state.

The plain files are visible to the minions, as well as the state files. In Salt, everything is a file; there is no "magic translation" of files and file types. This means that a state file can be distributed to minions just like a plain text or binary file.

SLS Files

The Salt state files are simple sets of data. Since the SLS files are just data they can be represented in a number of different ways. The default format is yaml generated from a Jinja template. This allows for the states files to have all the language constructs of Python and the simplicity of yaml. State files can then be complicated Jinja templates that translate down to yaml, or just plain and simple yaml files!

The State files are constructed data structures in a simple format. The format allows for many real activates to be expressed in very little text, while maintaining the utmost in readability and usability.

Here is an example of a Salt State:

```
vim:
  pkg:
    - installed

salt:
  pkg:
    - latest
  service.running:
    - require:
      - file: /etc/salt/minion
      - pkg: salt
    - names:
      - salt-master
      - salt-minion
    - watch:
      - file: /etc/salt/minion

/etc/salt/minion:
  file.managed:
    - source: salt://salt/minion
    - user: root
    - group: root
    - mode: 644
    - require:
      - pkg: salt
```

This short stanza will ensure that vim is installed, Salt is installed and up to date, the salt-master and salt-minion daemons are running and the Salt minion configuration file is in place. It will also ensure everything is deployed in the right order and that the Salt services are restarted when the watched file updated.

The Top File

The top file is the mapping for the state system. The top file specifies which minions should have which modules applied and which environments they should draw the states from.

The top file works by specifying the environment, containing matchers with lists of Salt states sent to the matching minions:

```
base:
  '*':
    - salt
    - users
    - users.admin
  'saltmaster.*':
    - match: pcre
    - salt.master
```

This simple example uses the base environment, which is built into the default Salt setup, and then all minions will have the modules salt, users and users.admin since '*' will match all minions. Then the regular expression matcher will match all minions' with an id matching saltmaster.* and add the salt.master state.

69.2.2 Renderer System

The Renderer system is a key component to the state system. SLS files are representations of Salt "high data" structures. All Salt cares about when reading an SLS file is the data structure that is produced from the file.

This allows Salt states to be represented by multiple types of files. The Renderer system can be used to allow different formats to be used for SLS files.

The available renderers can be found in the renderers directory in the Salt source code:

https://github.com/saltstack/salt/blob/develop/salt/renderers

By default SLS files are rendered using Jinja as a templating engine, and yaml as the serialization format. Since the rendering system can be extended simply by adding a new renderer to the renderers directory, it is possible that any structured file could be used to represent the SLS files.

In the future XML will be added, as well as many other formats.

69.2.3 Reloading Modules

Some salt states require specific packages to be installed in order for the module to load, as an example the `pip` state module requires the pip package for proper name and version parsing. On most of the common cases, salt is clever enough to transparently reload the modules, for example, if you install a package, salt reloads modules because some other module or state might require just that package which was installed. On some edge-cases salt might need to be told to reload the modules. Consider the following state file which we'll call `pep8.sls`:

```
python-pip:
  cmd:
    - run
    - cwd: /
    - name: easy_install --script-dir=/usr/bin -U pip

pep8:
  pip.installed
  requires:
    - cmd: python-pip
```

The above example installs pip using `easy_install` from setuptools and installs pep8 using `pip`, which, as told earlier, requires pip to be installed system-wide. Let's execute this state:

```
salt-call state.sls pep8
```

The execution output would be something like:

```
----------
    State: - pip
    Name:      pep8
    Function:  installed
        Result:    False
        Comment:   State pip.installed found in sls pep8 is unavailable

        Changes:

Summary
------------
Succeeded: 1
Failed:    1
------------
Total:     2
```

If we executed the state again the output would be:

```
----------
    State: - pip
    Name:      pep8
    Function:  installed
        Result:    True
        Comment:   Package was successfully installed
        Changes:   pep8==1.4.6: Installed

Summary
------------
Succeeded: 2
Failed:    0
------------
Total:     2
```

Since we installed pip using `cmd`, salt has no way to know that a system-wide package was installed. On the second execution, since the required pip package was installed, the state executed perfectly.

To those thinking, could not salt reload modules on every state step since it already does for some cases? It could, but it should not since it would greatly slow down state execution.

So how do we solve this *edge-case*? `reload_modules`!

`reload_modules` is a boolean option recognized by salt on **all** available states which, does exactly what it tells use, forces salt to reload it's modules once that specific state finishes. The fixed state file would now be:

```
python-pip:
  cmd:
    - run
    - cwd: /
    - name: easy_install --script-dir=/usr/bin -U pip
    - reload_modules: true

pep8:
  pip.installed
  requires:
    - cmd: python-pip
```

Let's run it, once:

```
salt-call state.sls pep8
```

And it's output now is:

```
----------
    State: - pip
    Name:      pep8
    Function:  installed
        Result:    True
        Comment:   Package was successfully installed
        Changes:   pep8==1.4.6: Installed

Summary
------------
Succeeded: 2
Failed:    0
------------
Total:     2
```

CHAPTER 70

State System Layers

The Salt state system is comprised of multiple layers. While using Salt does not require an understanding of the state layers, a deeper understanding of how Salt compiles and manages states can be very beneficial.

70.1 Function Call

The lowest layer of functionality in the state system is the direct state function call. State executions are executions of single state functions at the core. These individual functions are defined in state modules and can be called directly via the `state.single` command.

```
salt '*' state.single pkg.installed name='vim'
```

70.2 Low Chunk

The low chunk is the bottom of the Salt state compiler. This is a data representation of a single function call. The low chunk is sent to the state caller and used to execute a single state function.

A single low chunk can be executed manually via the `state.low` command.

```
salt '*' state.low '{name: vim, state: pkg, fun: installed}'
```

The passed data reflects what the state execution system gets after compiling the data down from sls formulas.

70.3 Low State

The *Low State* layer is the list of low chunks "evaluated" in order. To see what the low state looks like for a highstate, run:

```
salt '*' state.show_lowstate
```

This will display the raw lowstate in the order which each low chunk will be evaluated. The order of evaluation is not necessarily the order of execution, since requisites are evaluated at runtime. Requisite execution and evaluation is finite; this means that the order of execution can be ascertained with 100% certainty based on the order of the low state.

70.4 High Data

High data is the data structure represented in YAML via SLS files. The High data structure is created by merging the data components rendered inside sls files (or other render systems). The High data can be easily viewed by executing the `state.show_highstate` or `state.show_sls` functions. Since this data is a somewhat complex data structure, it may be easier to read using the json, yaml, or pprint outputters:

```
salt '*' state.show_highstate --out yaml
salt '*' state.show_sls edit.vim --out pprint
```

70.5 SLS

Above "High Data", the logical layers are no longer technically required to be executed, or to be executed in a hierarchy. This means that how the High data is generated is optional and very flexible. The SLS layer allows for many mechanisms to be used to render sls data from files or to use the fileserver backend to generate sls and file data from external systems.

The SLS layer can be called directly to execute individual sls formulas.

Note: SLS Formulas have historically been called "SLS files". This is because a single SLS was only constituted in a single file. Now the term "SLS Formula" better expresses how a compartmentalized SLS can be expressed in a much more dynamic way by combining pillar and other sources, and the SLS can be dynamically generated.

To call a single SLS formula named `edit.vim`, execute `state.sls`:

```
salt '*' state.sls edit.vim
```

70.6 HighState

Calling SLS directly logically assigns what states should be executed from the context of the calling minion. The Highstate layer is used to allow for full contextual assignment of what is executed where to be tied to groups of, or individual, minions entirely from the master. This means that the environment of a minion, and all associated execution data pertinent to said minion, can be assigned from the master without needing to execute or configure anything on the target minion. This also means that the minion can independently retrieve information about its complete configuration from the master.

To execute the High State call `state.highstate`:

```
salt '*' state.highstate
```

70.7 OverState

The overstate layer expresses the highest functional layer of Salt's automated logic systems. The Overstate allows for stateful and functional orchestration of routines from the master. The overstate defines in data execution stages which minions should execute states, or functions, and in what order using requisite logic.

CHAPTER 71

The Orchestrate Runner

Note: This documentation has been moved *here*.

CHAPTER 72

Ordering States

The way in which configuration management systems are executed is a hotly debated topic in the configuration management world. Two major philosophies exist on the subject, to either execute in an imperative fashion where things are executed in the order in which they are defined, or in a declarative fashion where dependencies need to be mapped between objects.

Imperative ordering is finite and generally considered easier to write, but declarative ordering is much more powerful and flexible but generally considered more difficult to create.

Salt has been created to get the best of both worlds. States are evaluated in a finite order, which guarantees that states are always executed in the same order, and the states runtime is declarative, making Salt fully aware of dependencies via the *requisite* system.

72.1 State Auto Ordering

Salt always executes states in a finite manner, meaning that they will always execute in the same order regardless of the system that is executing them. But in Salt 0.17.0, the `state_auto_order` option was added. This option makes states get evaluated in the order in which they are defined in sls files.

The evaluation order makes it easy to know what order the states will be executed in, but it is important to note that the requisite system will override the ordering defined in the files, and the `order` option described below will also override the order in which states are defined in sls files.

If the classic ordering is preferred (lexicographic), then set `state_auto_order` to `False` in the master configuration file.

72.2 Requisite Statements

Note: This document represents behavior exhibited by Salt requisites as of version 0.9.7 of Salt.

Often when setting up states any single action will require or depend on another action. Salt allows for the building of relationships between states with requisite statements. A requisite statement ensures that the named state is evaluated before the state requiring it. There are three types of requisite statements in Salt, **require**, **watch** and **prereq**.

These requisite statements are applied to a specific state declaration:

```
httpd:
  pkg:
    - installed
  file.managed:
    - name: /etc/httpd/conf/httpd.conf
    - source: salt://httpd/httpd.conf
    - require:
      - pkg: httpd
```

In this example, the **require** requisite is used to declare that the file /etc/httpd/conf/httpd.conf should only be set up if the pkg state executes successfully.

The requisite system works by finding the states that are required and executing them before the state that requires them. Then the required states can be evaluated to see if they have executed correctly.

Require statements can refer to the following requisite types: pkg, file, sls

In addition to state declarations such as pkg, file, etc., **sls** type requisites are also recognized, and essentially allow 'chaining' of states. This provides a mechanism to ensure the proper sequence for complex state formulas, especially when the discrete states are split or groups into separate sls files:

```
include:
  - network

httpd:
  pkg:
    - installed
  service:
    - running
    - require:
      - pkg: httpd
      - sls: network
```

In this example, the httpd sevice running state will not be applied (i.e., the httpd service will not be started) unless both the https package is installed AND the network state is satisfied.

Note: Requisite matching

Requisites match on both the ID Declaration and the `name` parameter. Therefore, if using the `pkgs` or `sources` argument to install a list of packages in a pkg state, it's important to note that it is impossible to match an individual package in the list, since all packages are installed as a single state.

72.2.1 Multiple Requisites

The requisite statement is passed as a list, allowing for the easy addition of more requisites. Both requisite types can also be separately declared:

```
httpd:
  pkg:
    - installed
  service.running:
    - enable: True
    - watch:
      - file: /etc/httpd/conf/httpd.conf
    - require:
      - pkg: httpd
      - user: httpd
      - group: httpd
  file.managed:
    - name: /etc/httpd/conf/httpd.conf
    - source: salt://httpd/httpd.conf
    - require:
      - pkg: httpd
  user:
    - present
  group:
    - present
```

In this example, the httpd service is only going to be started if the package, user, group and file are executed successfully.

72.2.2 The Require Requisite

The foundation of the requisite system is the `require` requisite. The require requisite ensures that the required state(s) are executed before the requiring state. So, if a state is declared that sets down a vimrc, then it would be pertinent to make sure that the vimrc file would only be set down if the vim package has been installed:

```
vim:
  pkg:
    - installed
  file.managed:
    - source: salt://vim/vimrc
    - require:
      - pkg: vim
```

In this case, the vimrc file will only be applied by Salt if and after the vim package is installed.

72.2.3 The Watch Requisite

The `watch` requisite is more advanced than the `require` requisite. The watch requisite executes the same logic as require (therefore if something is watched it does not need to also be required) with the addition of executing logic if the required states have changed in some way.

The watch requisite checks to see if the watched states have returned any changes. If the watched state returns changes, and the watched states execute successfully, then the watching state will execute a function that reacts to the changes in the watched states.

Perhaps an example can better explain the behavior:

```
redis:
  pkg:
    - latest
  file.managed:
    - source: salt://redis/redis.conf
    - name: /etc/redis.conf
    - require:
      - pkg: redis
  service.running:
    - enable: True
    - watch:
      - file: /etc/redis.conf
      - pkg: redis
```

In this example, the redis service will only be started if the file /etc/redis.conf is applied, and the file is only applied if the package is installed. This is normal require behavior, but if the watched file changes, or the watched package is installed or upgraded, then the redis service is restarted.

Note: To reiterate: watch does not alter the original behavior of a function in any way. The original behavior stays, but additional behavior (defined by mod_watch as explored below) will be run if there are changes in the watched state. This is why, for example, we have to have a `cmd.wait` state for watching purposes. If you examine the source code, you'll see that `cmd.wait` is an empty function. However, you'll notice that `mod_watch` is actually just an alias of `cmd.run`. So if there are changes, we run the command, otherwise, we do nothing.

72.2.4 Watch and the mod_watch Function

The watch requisite is based on the `mod_watch` function. Python state modules can include a function called `mod_watch` which is then called if the watch call is invoked. When `mod_watch` is called depends on the execution of the watched state, which:

- If no changes then just run the watching state itself as usual. `mod_watch` is not called. This behavior is same as using a `require`.
- If changes then run the watching state *AND* if that changes nothing then react by calling `mod_watch`.

When reacting, in the case of the service module the underlying service is restarted. In the case of the cmd state the command is executed.

The `mod_watch` function for the service state looks like this:

```
def mod_watch(name, sig=None, reload=False, full_restart=False):
    '''
    The service watcher, called to invoke the watch command.

    name
        The name of the init or rc script used to manage the service

    sig
        The string to search for when looking for the service process with ps
    '''
    if __salt__['service.status'](name, sig):
        if 'service.reload' in __salt__ and reload:
            restart_func = __salt__['service.reload']
        elif 'service.full_restart' in __salt__ and full_restart:
            restart_func = __salt__['service.full_restart']
        else:
            restart_func = __salt__['service.restart']
    else:
        restart_func = __salt__['service.start']

    result = restart_func(name)
    return {'name': name,
            'changes': {name: result},
            'result': result,
            'comment': 'Service restarted' if result else \
                       'Failed to restart the service'
           }
```

The watch requisite only works if the state that is watching has a `mod_watch` function written. If watch is set on a state that does not have a `mod_watch` function (like pkg), then the listed states will behave only as if they were under a `require` statement.

Also notice that a `mod_watch` may accept additional keyword arguments, which, in the sls file, will be taken from the same set of arguments specified for the state that includes the `watch` requisite. This means, for the earlier `service.running` example above, the service can be set to `reload` instead of restart like this:

```
redis:

  # ... other state declarations omitted ...

    service.running:
      - enable: True
      - reload: True
      - watch:
        - file: /etc/redis.conf
        - pkg: redis
```

72.3 The Order Option

Before using the *order* option, remember that the majority of state ordering should be done with a *requisite declaration*, and that a requisite declaration will override an *order* option, so a state with order option should not require or required by other states.

The order option is used by adding an order number to a state declaration with the option *order*:

```
vim:
  pkg.installed:
    - order: 1
```

By adding the order option to *1* this ensures that the vim package will be installed in tandem with any other state declaration set to the order *1*.

Any state declared without an order option will be executed after all states with order options are executed.

But this construct can only handle ordering states from the beginning. Certain circumstances will present a situation where it is desirable to send a state to the end of the line. To do this, set the order to `last`:

```
vim:
  pkg.installed:
    - order: last
```

CHAPTER 73

OverState System

Note: This documentation has been moved *here*.

CHAPTER 74

State Providers

New in version 0.9.8.

Salt predetermines what modules should be mapped to what uses based on the properties of a system. These determinations are generally made for modules that provide things like package and service management.

Sometimes in states, it may be necessary to use an alternative module to provide the needed functionality. For instance, an older Arch Linux system may not be running systemd, so instead of using the systemd service module, you can revert to the default service module:

```
httpd:
  service.running:
    - enable: True
    - provider: service
```

In this instance, the basic `service` module (which manages **sysvinit**-based services) will replace the `systemd` module which is used by default on Arch Linux.

However, if it is necessary to make this override for most or every service, it is better to just override the provider in the minion config file, as described in the section below.

74.1 Setting a Provider in the Minion Config File

Sometimes, when running Salt on custom Linux spins, or distribution that are derived from other distributions, Salt does not successfully detect providers. The providers which are most likely to be affected by this are:

- pkg
- service
- user
- group

When something like this happens, rather than specifying the provider manually in each state, it easier to use the `providers` parameter in the minion config file to set the provider.

If you end up needing to override a provider because it was not detected, please let us know! File an issue on the issue tracker, and provide the output from the `grains.items` function, taking care to sanitize any sensitive information.

Below are tables that should help with deciding which provider to use if one needs to be overridden.

74.1.1 Provider: `pkg`

Execution Module	Used for
apt	Debian/Ubuntu-based distros which use `apt-get(8)` for package management
brew	Mac OS software management using Homebrew
ebuild	Gentoo-based systems (utilizes the `portage` python module as well as `emerge(1)`)
freebsdpkg	FreeBSD-based OSes using `pkg_add(1)`
openbsdpkg	OpenBSD-based OSes using `pkg_add(1)`
pacman	Arch Linux-based distros using `pacman(8)`
pkgin	NetBSD-based OSes using `pkgin(1)`
pkgng	FreeBSD-based OSes using `pkg(8)`
pkgutil	Solaris-based OSes using OpenCSW's `pkgutil(1)`
solarispkg	Solaris-based OSes using `pkgadd(1M)`
win_pkg	Windows
yumpkg	RedHat-based distros and derivatives (wraps `yum(8)`)
zypper	SUSE-based distros using `zypper(8)`

74.1.2 Provider: `service`

Execution Module	Used for
debian_service	Debian (non-systemd)
freebsdservice	FreeBSD-based OSes using `service(8)`
gentoo_service	Gentoo Linux using **sysvinit** and `rc-update(8)`
launchctl	Mac OS hosts using `launchctl(1)`
netbsdservice	NetBSD-based OSes
openbsdservice	OpenBSD-based OSes
rh_service	RedHat-based distros and derivatives using `service(8)` and `chkconfig(8)`. Supports both pure sysvinit and mixed sysvinit/upstart systems.
service	Fallback which simply wraps sysvinit scripts
smf	Solaris-based OSes which use SMF
systemd	Linux distros which use systemd
upstart	Ubuntu-based distros using upstart
win_service	Windows

74.1.3 Provider: `user`

Execution Module	Used for
useradd	Linux, NetBSD, and OpenBSD systems using `useradd(8)`, `userdel(8)`, and `usermod(8)`
pw_user	FreeBSD-based OSes using `pw(8)`
solaris_user	Solaris-based OSes using `useradd(1M)`, `userdel(1M)`, and `usermod(1M)`
win_useradd	Windows

74.1.4 Provider: `group`

Execution Module	Used for
groupadd	Linux, NetBSD, and OpenBSD systems using `groupadd(8)`, `groupdel(8)`, and `groupmod(8)`
pw_group	FreeBSD-based OSes using `pw(8)`
solaris_group	Solaris-based OSes using `groupadd(1M)`, `groupdel(1M)`, and `groupmod(1M)`
win_groupadd	Windows

74.2 Arbitrary Module Redirects

The provider statement can also be used for more powerful means, instead of overwriting or extending the module used for the named service an arbitrary module can be used to provide certain functionality.

```
emacs:
  pkg.installed:
    - provider:
      - cmd: customcmd
```

In this example, the state is being instructed to use a custom module to invoke commands.

Arbitrary module redirects can be used to dramatically change the behavior of a given state.

CHAPTER 75

Requisites

The Salt requisite system is used to create relationships between states. The core idea being that, when one state is dependent somehow on another, that inter-dependency can be easily defined.

Requisites come in two types. Direct requisites, and requisite_ins. The relationships are directional, so a requisite statement makes the requiring state declaration depend on the required state declaration:

```
vim:
  pkg.installed

/etc/vimrc:
  file.managed:
    - source: salt://edit/vimrc
    - require:
      - pkg: vim
```

So in this example, the file `/etc/vimrc` depends on the vim package.

Requisite_in statements are the opposite, instead of saying "I depend on something", requisite_ins say "Someone depends on me":

```
vim:
  pkg.installed:
    - require_in:
      - file: /etc/vimrc

/etc/vimrc:
  file.managed:
    - source: salt://edit/vimrc
```

So here, with a requisite_in, the same thing is accomplished, but just from the other way around. The vim package is saying "/etc/vimrc depends on me".

In the end, a single dependency map is created and everything is executed in a finite and predictable order.

Note: Requisite matching

Requisites match on both the ID Declaration and the `name` parameter. This means that, in the example above, the `require_in` requisite would also have been matched if the `/etc/vimrc` state was written as follows:

```
vimrc:
  file.managed:
    - name: /etc/vimrc
    - source: salt://edit/vimrc
```

75.1 Requisite and Requisite in types

There are three requisite statements that can be used in Salt. the `require`, `watch` and `use` requisites. Each requisite also has a corresponding requisite_in: `require_in`, `watch_in` and `use_in`. All of the requisites define specific relationships and always work with the dependency logic defined above.

75.1.1 Require

The most basic requisite statement is `require`. The behavior of require is simple. Make sure that the dependent state is executed before the depending state, and if the dependent state fails, don't run the depending state. So in the above examples the file `/etc/vimrc` will only be applied after the vim package is installed and only if the vim package is installed successfully.

75.1.2 Require an entire sls file

As of Salt 0.16.0, it is possible to require an entire sls file. Do this by first including the sls file and then setting a state to `require` the included sls file.

```
include:
  - foo

bar:
  pkg.installed:
    - require:
      - sls: foo
```

75.1.3 Watch

The watch statement does everything the require statement does, but with a little more. The watch statement looks into the state modules for a function called `mod_watch`. If this function is not available in the corresponding state module, then watch does the same thing as require. If the `mod_watch` function is in the state module, then the watched state is checked to see if it made any changes to the system, if it has, then `mod_watch` is called.

Perhaps the best example of using watch is with a `service.running` state. When a service watches a state, then the service is reloaded/restarted when the watched state changes:

```
ntpd:
  service.running:
    - watch:
      - file: /etc/ntp.conf
  file.managed:
    - name: /etc/ntp.conf
    - source: salt://ntp/files/ntp.conf
```

75.1.4 Prereq

The `prereq` requisite is a powerful requisite added in 0.16.0. This requisite allows for actions to be taken based on the expected results of a state that has not yet been executed. In more practical terms, a service can be shut down because the `prereq` knows that underlying code is going to be updated and the service should be off-line while the update occurs.

The motivation to add this requisite was to allow for routines to remove a system from a load balancer while code is being updated.

The `prereq` checks if the required state expects to have any changes by running the single state with `test=True`. If the pre-required state returns changes, then the state requiring it will execute.

```
graceful-down:
  cmd.run:
    - name: service apache graceful
    - prereq:
      - file: site-code

site-code:
  file.recurse:
    - name: /opt/site_code
    - source: salt://site/code
```

In this case the apache server will only be shutdown if the site-code state expects to deploy fresh code via the file.recurse call, and the site-code deployment will only be executed if the graceful-down run completes successfully.

75.1.5 Use

The `use` requisite is used to inherit the arguments passed in another id declaration. This is useful when many files need to have the same defaults.

```
/etc/foo.conf:
  file.managed:
    - source: salt://foo.conf
    - template: jinja
    - mkdirs: True
    - user: apache
    - group: apache
    - mode: 755

/etc/bar.conf
  file.managed:
    - source: salt://bar.conf
    - use:
      - file: /etc/foo.conf
```

The `use` statement was developed primarily for the networking states but can be used on any states in Salt. This made sense for the networking state because it can define a long list of options that need to be applied to multiple network interfaces.

The `use` statement does not inherit the requisites arguments of the targeted state. This means also a chain of `use` requisites would not inherit inherited options.

75.1.6 Require In

The `require_in` requisite is the literal reverse of `require`. If a state declaration needs to be required by another state declaration then require_in can accommodate it, so these two sls files would be the same in the end:

Using `require`

```
httpd:
  pkg:
    - installed
  service:
    - running
    - require:
      - pkg: httpd
```

Using `require_in`

```
httpd:
  pkg:
    - installed
    - require_in:
      - service: httpd
  service:
    - running
```

The `require_in` statement is particularly useful when assigning a require in a separate sls file. For instance it may be common for httpd to require components used to set up PHP or mod_python, but the HTTP state does not need to be aware of the additional components that require it when it is set up:

http.sls

```
httpd:
  pkg:
    - installed
  service:
    - running
    - require:
      - pkg: httpd
```

php.sls

```
include:
  - http

php:
  pkg:
    - installed
    - require_in:
      - service: httpd
```

mod_python.sls

```
include:
  - http

mod_python:
  pkg:
    - installed
    - require_in:
      - service: httpd
```

Now the httpd server will only start if php or mod_python are first verified to be installed. Thus allowing for a requisite to be defined "after the fact".

75.1.7 Watch In

Watch in functions the same as require in, but applies a watch statement rather than a require statement to the external state declaration.

75.1.8 Prereq In

The `prereq_in` requisite in follows the same assignment logic as the `require_in` requisite in. The `prereq_in` call simply assigns `prereq` to the state referenced. The above example for `prereq` can be modified to function in the same way using `prereq_in`:

```
graceful-down:
  cmd.run:
    - name: service apache graceful

site-code:
  file.recurse:
    - name: /opt/site_code
    - source: salt://site/code
    - prereq_in:
      - cmd: graceful-down
```

CHAPTER 76

Startup States

Sometimes it may be desired that the salt minion execute a state run when it is started. This alleviates the need for the master to initiate a state run on a new minion and can make provisioning much easier.

As of Salt 0.10.3 the minion config reads options that allow for states to be executed at startup. The options are *startup_states*, *sls_list* and *top_file*.

The *startup_states* option can be passed one of a number of arguments to define how to execute states. The available options are:

highstate Execute `state.highstate`

sls Read in the `sls_list` option and execute the named sls files

top Read in the `top_file` option and execute states based on that top file on the Salt Master

76.1 Examples:

Execute `state.highstate` when starting the minion:

```
startup_states: highstate
```

Execute the sls files *edit.vim* and *hyper*:

```
startup_states: sls

sls_list:
  - edit.vim
  - hyper
```

State Testing

Executing a Salt state run can potentially change many aspects of a system and it may be desirable to first see what a state run is going to change before applying the run.

Salt has a test interface to report on exactly what will be changed, this interface can be invoked on any of the major state run functions:

```
salt '*' state.highstate test=True
salt '*' state.sls test=True
salt '*' state.single test=True
```

The test run is mandated by adding the `test=True` option to the states. The return information will show states that will be applied in yellow and the result is reported as `None`.

77.1 Default Test

If the value `test` is set to `True` in the minion configuration file then states will default to being executed in test mode. If this value is set then states can still be run by calling test=False:

```
salt '*' state.highstate test=False
salt '*' state.sls test=False
salt '*' state.single test=False
```

The Top File

The top file is used to map what SLS modules get loaded onto what minions via the state system. The top file creates a few general abstractions. First it maps what nodes should pull from which environments, next it defines which matches systems should draw from.

78.1 Environments

Environment A configuration that allows conceptually organizing state tree directories. Environments can be made to be self-contained or state trees can be made to bleed through environments.

Note: Environments in Salt are very flexible, this section defines how the top file can be used to define what states from what environments are to be used for specific minions.

If the intent is to bind minions to specific environments, then the *environment* option can be set in the minion configuration file.

The environments in the top file corresponds with the environments defined in the `file_roots` variable. In a simple, single environment setup you only have the `base` environment, and therefore only one state tree. Here is a simple example of `file_roots` in the master configuration:

```
file_roots:
  base:
    - /srv/salt
```

This means that the top file will only have one environment to pull from, here is a simple, single environment top file:

```
base:
  '*':
    - core
    - edit
```

This also means that `/srv/salt` has a state tree. But if you want to use multiple environments, or partition the file server to serve more than just the state tree, then the `file_roots` option can be expanded:

```
file_roots:
  base:
    - /srv/salt/base
  dev:
    - /srv/salt/dev
  qa:
    - /srv/salt/qa
  prod:
    - /srv/salt/prod
```

Then our top file could reference the environments:

```
dev:
  'webserver*dev*':
    - webserver
  'db*dev*':
    - db
qa:
  'webserver*qa*':
    - webserver
  'db*qa*':
    - db
prod:
  'webserver*prod*':
    - webserver
  'db*prod*':
    - db
```

In this setup we have state trees in three of the four environments, and no state tree in the `base` environment. Notice that the targets for the minions specify environment data. In Salt the master determines who is in what environment, and many environments can be crossed together. For instance, a separate global state tree could be added to the `base` environment if it suits your deployment:

```
base:
  '*':
    - global
dev:
  'webserver*dev*':
    - webserver
  'db*dev*':
    - db
qa:
  'webserver*qa*':
    - webserver
  'db*qa*':
    - db
prod:
  'webserver*prod*':
    - webserver
  'db*prod*':
    - db
```

In this setup all systems will pull the global SLS from the base environment, as well as pull from their respective environments. If you assign only one SLS to a system, as in this example, a shorthand is also available:

```
base:
  '*': global
dev:
  'webserver*dev*': webserver
  'db*dev*':        db
qa:
  'webserver*qa*': webserver
  'db*qa*':        db
prod:
  'webserver*prod*': webserver
  'db*prod*':        db
```

Note: The top files from all defined environments will be compiled into a single top file for all states. Top files are environment agnostic.

Remember, that since everything is a file in Salt, the environments are primarily file server environments, this means that environments that have nothing to do with states can be defined and used to distribute other files. A clean and recommended setup for multiple environments would look like this:

```
# Master file_roots configuration:
file_roots:
  base:
    - /srv/salt/base
  dev:
    - /srv/salt/dev
  qa:
    - /srv/salt/qa
  prod:
    - /srv/salt/prod
```

Then only place state trees in the dev, qa and prod environments, leaving the base environment open for generic file transfers. Then the top.sls file would look something like this:

```
dev:
  'webserver*dev*':
    - webserver
  'db*dev*':
    - db
qa:
  'webserver*qa*':
    - webserver
  'db*qa*':
    - db
prod:
  'webserver*prod*':
    - webserver
  'db*prod*':
    - db
```

78.2 Other Ways of Targeting Minions

In addition to globs, minions can be specified in top files a few other ways. Some common ones are *compound matches* and *node groups*.

Here is a slightly more complex top file example, showing the different types of matches you can perform:

```
base:
    '*':
        - ldap-client
        - networking
        - salt.minion

    'salt-master*':
        - salt.master

    '^(memcache|web).(qa|prod).loc$':
        - match: pcre
        - nagios.mon.web
        - apache.server

    'os:Ubuntu':
        - match: grain
        - repos.ubuntu

    'os:(RedHat|CentOS)':
        - match: grain_pcre
        - repos.epel

    'foo,bar,baz':
        - match: list
        - database

    'somekey:abc':
        - match: pillar
        - xyz

    'nag1* or G@role:monitoring':
        - match: compound
        - nagios.server
```

In this example `top.sls`, all minions get the ldap-client, networking and salt.minion states. Any minion with an id matching the `salt-master*` glob will get the salt.master state. Any minion with ids matching the regular expression `^(memcache|web).(qa|prod).loc$` will get the nagios.mon.web and apache.server states. All Ubuntu minions will receive the repos.ubuntu state, while all RHEL and CentOS minions will receive the repos.epel state. The minions `foo`, `bar`, and `baz` will receive the database state. Any minion with a pillar named `somekey`, having a value of `abc` will receive the xyz state. Finally, minions with ids matching the nag1* glob or with a grain named `role` equal to `monitoring` will receive the nagios.server state.

78.3 How Top Files Are Compiled

As mentioned earlier, the top files in the different environments are compiled into a single set of data. The way in which this is done follows a few rules, which are important to understand when arranging top files in different environments. The examples below all assume that the `file_roots` are set as in the *above multi-environment example*.

1. The `base` environment's top file is processed first. Any environment which is defined in the `base` top.sls as well as another environment's top file, will use the instance of the environment configured in `base` and ignore all other instances. In other words, the `base` top file is authoritative when defining environments. Therefore, in the example below, the `dev` section in `/srv/salt/dev/top.sls` would be completely ignored.

`/srv/salt/base/top.sls:`

```
base:
  '*':
    - common
dev:
  'webserver*dev*':
    - webserver
  'db*dev*':
    - db
```

`/srv/salt/dev/top.sls:`

```
dev:
  '10.10.100.0/24':
    - match: ipcidr
    - deployments.dev.site1
  '10.10.101.0/24':
    - match: ipcidr
    - deployments.dev.site2
```

Note: The rules below assume that the environments being discussed were not defined in the `base` top file.

---

2. If, for some reason, the `base` environment is not configured in the `base` environment's top file, then the other environments will be checked in alphabetical order. The first top file found to contain a section for the `base` environment wins, and the other top files' `base` sections are ignored. So, provided there is no `base` section in the `base` top file, with the below two top files the `dev` environment would win out, and the `common.centos` SLS would not be applied to CentOS hosts.

`/srv/salt/dev/top.sls`:

```
base:
  'os:Ubuntu':
    - common.ubuntu
dev:
  'webserver*dev*':
    - webserver
  'db*dev*':
    - db
```

`/srv/salt/qa/top.sls`:

```
base:
  'os:Ubuntu':
    - common.ubuntu
  'os:CentOS':
    - common.centos
qa:
  'webserver*qa*':
    - webserver
  'db*qa*':
    - db
```

3. For environments other than `base`, the top file in a given environment will be checked for a section matching the environment's name. If one is found, then it is used. Otherwise, the remaining (non-`base`) environments will be checked in alphabetical order. In the below example, the `qa` section in `/srv/salt/dev/top.sls` will be ignored, but if `/srv/salt/qa/top.sls` were cleared or removed, then the states configured for the `qa` environment in `/srv/salt/dev/top.sls` will be applied.

`/srv/salt/dev/top.sls`:

```
dev:
  'webserver*dev*':
    - webserver
  'db*dev*':
    - db
qa:
  '10.10.200.0/24':
    - match: ipcidr
    - deployments.qa.site1
  '10.10.201.0/24':
    - match: ipcidr
    - deployments.qa.site2
```

`/srv/salt/qa/top.sls:`

```
qa:
  'webserver*qa*':
    - webserver
  'db*qa*':
    - db
```

Note: When in doubt, the simplest way to configure your states is with a single top.sls in the `base` environment.

SLS Template Variable Reference

The template engines available to sls files and file templates come loaded with a number of context variables. These variables contain information and functions to assist in the generation of templates. See each variable below for its availability – not all variables are available in all templating contexts.

79.1 Salt

The *salt* variable is available to abstract the salt library functions. This variable is a python dictionary containing all of the functions available to the running salt minion. It is available in all salt templates.

```
{% for file in salt['cmd.run']('ls -1 /opt/to_remove').splitlines() %}
/opt/to_remove/{{ file }}:
  file.absent
{% endfor %}
```

79.2 Opts

The *opts* variable abstracts the contents of the minion's configuration file directly to the template. The *opts* variable is a dictionary. It is available in all templates.

```
{{ opts['cachedir'] }}
```

The `config.get` function also searches for values in the *opts* dictionary.

79.3 Pillar

The *pillar* dictionary can be referenced directly, and is available in all templates:

```
{{ pillar['key'] }}
```

Using the `pillar.get` function via the *salt* variable is generally recommended since a default can be safely set in the event that the value is not available in pillar and dictionaries can be traversed directly:

```
{{ salt['pillar.get']('key', 'failover_value') }}
{{ salt['pillar.get']('stuff:more:deeper') }}
```

79.4 Grains

The *grains* dictionary makes the minion's grains directly available, and is available in all templates:

```
{{ grains['os'] }}
```

The `grains.get` function can be used to traverse deeper grains and set defaults:

```
{{ salt['grains.get']('os') }}
```

79.5 env

The *env* variable is available in only in sls files when gathering the sls from an environment.

```
{{ env }}
```

79.6 sls

The *sls* variable contains the sls reference value, and is only available in the actual SLS file (not in any files referenced in that SLS). The sls reference value is the value used to include the sls in top files or via the include option.

```
{{ sls }}
```

CHAPTER 80

State Modules

State Modules are the components that map to actual enforcement and management of Salt states.

80.1 States are Easy to Write!

State Modules should be easy to write and straightforward. The information passed to the SLS data structures will map directly to the states modules.

Mapping the information from the SLS data is simple, this example should illustrate:

```
/etc/salt/master: # maps to "name"
  file: # maps to State module filename e.g. https://github.com/saltstack/salt/blob/develop/salt/sta
    - managed # maps to the managed function in the file State module
    - user: root # one of many options passed to the manage function
    - group: root
    - mode: 644
    - source: salt://salt/master
```

Therefore this SLS data can be directly linked to a module, function and arguments passed to that function.

This does issue the burden, that function names, state names and function arguments should be very human readable inside state modules, since they directly define the user interface.

Keyword Arguments

Salt passes a number of keyword arguments to states when rendering them, including the environment, a unique identifier for the state, and more. Additionally, keep in mind that the requisites for a state are part of the keyword arguments. Therefore, if you need to iterate through the keyword arguments in a state, these must be considered and handled appropriately. One such example is in the `pkgrepo.managed` state, which needs to be able to handle arbitrary keyword arguments and pass them to module execution functions. An example of how these keyword arguments can be handled can be found here.

80.2 Using Custom State Modules

Place your custom state modules inside a `_states` directory within the `file_roots` specified by the master config file. These custom state modules can then be distributed in a number of ways. Custom state modules are distributed when `state.highstate` is run, or by executing the `saltutil.sync_states` or `saltutil.sync_all` functions.

Any custom states which have been synced to a minion, that are named the same as one of Salt's default set of states, will take the place of the default state with the same name. Note that a state's default name is its filename (i.e. `foo.py` becomes state `foo`), but that its name can be overridden by using a *__virtual__ function*.

80.3 Cross Calling Modules

As with Execution Modules, State Modules can also make use of the `__salt__` and `__grains__` data.

It is important to note that the real work of state management should not be done in the state module unless it is needed. A good example is the pkg state module. This module does not do any package management work, it just calls the pkg execution module. This makes the pkg state module completely generic, which is why there is only one pkg state module and many backend pkg execution modules.

On the other hand some modules will require that the logic be placed in the state module, a good example of this is the file module. But in the vast majority of cases this is not the best approach, and writing specific execution modules to do the backend work will be the optimal solution.

80.4 Return Data

A State Module must return a dict containing the following keys/values:

- **name:** The same value passed to the state as "name".
- **changes:** A dict describing the changes made. Each thing changed should be a key, with its value being another dict with keys called "old" and "new" containing the old/new values. For example, the pkg state's **changes** dict has one key for each package changed, with the "old" and "new" keys in its sub-dict containing the old and new versions of the package.
- **result:** A boolean value. *True* if the action was successful, otherwise *False*.
- **comment:** A string containing a summary of the result.

80.5 Test State

All states should check for and support `test` being passed in the options. This will return data about what changes would occur if the state were actually run. An example of such a check could look like this:

```
# Return comment of changes if test.
if __opts__['test']:
    ret['result'] = None
    ret['comment'] = 'State Foo will execute with param {0}'.format(bar)
    return ret
```

Make sure to test and return before performing any real actions on the minion.

80.6 Watcher Function

If the state being written should support the watch requisite then a watcher function needs to be declared. The watcher function is called whenever the watch requisite is invoked and should be generic to the behavior of the state itself.

The watcher function should accept all of the options that the normal state functions accept (as they will be passed into the watcher function).

A watcher function typically is used to execute state specific reactive behavior, for instance, the watcher for the service module restarts the named service and makes it useful for the watcher to make the service react to changes in the environment.

The watcher function also needs to return the same data that a normal state function returns.

80.7 Mod_init Interface

Some states need to execute something only once to ensure that an environment has been set up, or certain conditions global to the state behavior can be predefined. This is the realm of the mod_init interface.

A state module can have a function called **mod_init** which executes when the first state of this type is called. This interface was created primarily to improve the pkg state. When packages are installed the package metadata needs to be refreshed, but refreshing the package metadata every time a package is installed is wasteful. The mod_init function for the pkg state sets a flag down so that the first, and only the first, package installation attempt will refresh the package database (the package database can of course be manually called to refresh via the `refresh` option in the pkg state).

The mod_init function must accept the **Low State Data** for the given executing state as an argument. The low state data is a dict and can be seen by executing the state.show_lowstate function. Then the mod_init function must return a bool. If the return value is True, then the mod_init function will not be executed again, meaning that the needed behavior has been set up. Otherwise, if the mod_init function returns False, then the function will be called the next time.

A good example of the mod_init function is found in the pkg state module:

```
def mod_init(low):
    '''
    Refresh the package database here so that it only needs to happen once
    '''
    if low['fun'] == 'installed' or low['fun'] == 'latest':
        rtag = __gen_rtag()
        if not os.path.exists(rtag):
            open(rtag, 'w+').write('')
        return True
    else:
        return False
```

The mod_init function in the pkg state accepts the low state data as `low` and then checks to see if the function being called is going to install packages, if the function is not going to install packages then there is no need to refresh the package database. Therefore if the package database is prepared to refresh, then return True and the mod_init will not be called the next time a pkg state is evaluated, otherwise return False and the mod_init will be called next time a pkg state is evaluated.

CHAPTER 81

Full list of builtin state modules

`alias`	Configuration of email aliases
`alternatives`	Configuration of the alternatives system
`apt`	Package management operations specific to APT- and DEB-based systems
`archive`	Archive states.
`augeas`	Configuration management using Augeas
`aws_sqs`	Manage SQS Queues
`cloud`	Using states instead of maps to deploy clouds
`cmd`	Execution of arbitrary commands
`composer`	Installation of Composer Packages
`cron`	Management of cron, the Unix command scheduler
`ddns`	Dynamic DNS updates
`debconfmod`	Management of debconf selections
`disk`	Disk monitoring state
`dockerio`	Manage Docker containers
`eselect`	Management of Gentoo configuration using eselect
`file`	Operations on regular files, special files, directories, and symlinks
`gem`	Installation of Ruby modules packaged as gems
`git`	Interaction with Git repositories
`gnomedesktop`	Configuration of the GNOME desktop
`grains`	Manage grains on the minion
`group`	Management of user groups
`hg`	Interaction with Mercurial repositories
`host`	Management of addresses and names in hosts file
`iptables`	Management of iptables
`keyboard`	Management of keyboard layouts
`keystone`	Management of Keystone users
`kmod`	Loading and unloading of kernel modules
`layman`	Management of Gentoo Overlays using layman
`libvirt`	Manage libvirt certificates
`locale`	Management of languages/locales
`lvm`	Management of Linux logical volumes
`lvs_server`	Management of LVS(Linux Virtual Server) Real Server.
`lvs_service`	Management of LVS(Linux Virtual Server) Service.
`makeconf`	Management of Gentoo make.conf
`mdadm`	Managing software RAID with mdadm
`memcached`	States for Management of Memcached Keys
`modjk_worker`	Manage modjk workers
`module`	Execution of Salt modules from within states

Continued on next page

Table 81.1 – continued from previous page

mongodb_database	Management of Mongodb databases
mongodb_user	Management of Mongodb users
mount	Mounting of filesystems
mysql_database	Management of MySQL databases (schemas)
mysql_grants	Management of MySQL grants (user permissions)
mysql_user	Management of MySQL users
network	Configuration of network interfaces
npm	Installation of NPM Packages
ntp	Management of NTP servers
openstack_config	Manage OpenStack configuration file settings.
pagerduty	Create an Event in PagerDuty
pecl	Installation of PHP Extensions Using pecl
pip_state	Installation of Python Packages Using pip
pkg	Installation of packages using OS package managers such as yum or apt-get
pkgng	Manage package remote repo using FreeBSD pkgng
pkgrepo	Management of package repos
portage_config	Management of Portage package configuration on Gentoo
ports	Manage software from FreeBSD ports
postgres_database	Management of PostgreSQL databases
postgres_group	Management of PostgreSQL groups (roles)
postgres_user	Management of PostgreSQL users (roles)
powerpath	Powerpath configuration support
process	Process Management
quota	Management of POSIX Quotas
rabbitmq_cluster	Manage RabbitMQ Clusters
rabbitmq_plugin	Manage RabbitMQ Plugins
rabbitmq_policy	Manage RabbitMQ Policies
rabbitmq_user	Manage RabbitMQ Users
rabbitmq_vhost	Manage RabbitMQ Virtual Hosts
rbenv	Managing Ruby installations with rbenv
rdp	Manage RDP Service on Windows servers
reg	Manage the registry on Windows
rvm	Managing Ruby installations and gemsets with Ruby Version Manager (RVM)
saltmod	Control the Salt command interface
selinux	Management of SELinux rules
service	Starting or restarting of services and daemons
ssh_auth	Control of entries in SSH authorized_key files
ssh_known_hosts	Control of SSH known_hosts entries
stateconf	Stateconf System
status	Minion status monitoring
supervisord	Interaction with the Supervisor daemon
svn	Manage SVN repositories
sysctl	Configuration of the Linux kernel using sysctrl
timezone	Management of timezones
tomcat	This state uses the manager webapp to manage Apache tomcat webapps
user	Management of user accounts
virtualenv_mod	Setup of Python virtualenv sandboxes
win_dns_client	Module for configuring DNS Client on Windows systems
win_firewall	State for configuring Windows Firewall
win_network	Configuration of network interfaces on Windows hosts
win_path	Manage the Windows System PATH
win_servermanager	Manage Windows features via the ServerManager powershell module

Continued on next page

Table 81.1 – continued from previous page

`win_system`	Management of Windows system information
`xmpp`	Sending Messages over XMPP
`zcbuildout`	Management of zc.buildout

81.1 salt.states.alias

81.1.1 Configuration of email aliases

The mail aliases file can be managed to contain definitions for specific email aliases:

```
username:
  alias.present:
    - target: user@example.com
```

`salt.states.alias.`**`absent`**`(`*name*`)`

Ensure that the named alias is absent

name The alias to remove

`salt.states.alias.`**`present`**`(`*name*, *target*`)`

Ensures that the named alias is present with the given target

name The local user/address to assign an alias to

target The forwarding address

81.2 salt.states.alternatives

81.2.1 Configuration of the alternatives system

Control the alternatives system

```
{% set my_hadoop_conf = '/opt/hadoop/conf' %}

{{ my_hadoop_conf }}:
  file.directory

hadoop-0.20-conf:
  alternatives.install:
    - name: hadoop-0.20-conf
    - link: /etc/hadoop-0.20/conf
    - path: {{ my_hadoop_conf }}
    - priority: 30
    - require:
      - file: {{ my_hadoop_conf }}

hadoop-0.20-conf:
  alternatives.remove:
      - name: hadoop-0.20-conf
      - path: {{ my_hadoop_conf }}
```

`salt.states.alternatives.`**`auto`**`(`*name*`)`

New in version 0.17.0.

Instruct alternatives to use the highest priority path for <name>

name is the master name for this link group (e.g. pager)

`salt.states.alternatives.`**`install`**(*name*, *link*, *path*, *priority*)

Install new alternative for defined <name>

name is the master name for this link group (e.g. pager)

link is the symlink pointing to /etc/alternatives/<name>. (e.g. /usr/bin/pager)

path is the location of the new alternative target. NB: This file / directory must already exist. (e.g. /usr/bin/less)

priority is an integer; options with higher numbers have higher priority in automatic mode.

`salt.states.alternatives.`**`remove`**(*name*, *path*)

Removes installed alternative for defined <name> and <path> or fallback to default alternative, if some defined before.

name is the master name for this link group (e.g. pager)

path is the location of one of the alternative target files. (e.g. /usr/bin/less)

`salt.states.alternatives.`**`set_`**(*name*, *path*)

New in version 0.17.0.

Sets alternative for <name> to <path>, if <path> is defined as an alternative for <name>.

name is the master name for this link group (e.g. pager)

path is the location of one of the alternative target files. (e.g. /usr/bin/less)

81.3 salt.states.apt

81.3.1 Package management operations specific to APT- and DEB-based systems

`salt.states.apt.`**`held`**(*name*)

Set package in 'hold' state, meaning it will not be upgraded.

name The name of the package, e.g., 'tmux'

81.4 salt.states.archive

Archive states.

`salt.states.archive.`**`extracted`**(*name*, *source*, *archive_format*, *tar_options=None*, *source_hash=None*, *if_missing=None*)

New in version 2014.1.0: (Hydrogen)

State that make sure an archive is extracted in a directory. The downloaded archive is erased if succesfully extracted. The archive is downloaded only if necessary.

```
graylog2-server:
  archive:
    - extracted
    - name: /opt/
    - source: https://github.com/downloads/Graylog2/graylog2-server/graylog2-server-0.9.6p1.tar.
    - source_hash: md5=499ae16dcae71eeb7c3a30c75ea7a1a6
    - archive_format: tar
    - tar_options: z
    - if_missing: /opt/graylog2-server-0.9.6p1/
```

name Directory name where to extract the archive

source Archive source, same syntax as file.managed source argument.

archive_format tar, zip or rar

if_missing Some archive, such as tar, extract themself in a subfolder. This directive can be used to validate if the archive had been previously extracted.

tar_options Only used for tar format, it need to be the tar argument specific to this archive, such as 'j' for bzip2, 'z' for gzip, '' for uncompressed tar, 'J' for LZMA.

81.5 salt.states.augeas

81.5.1 Configuration management using Augeas

New in version 0.17.0.

This state requires the `augeas` Python module.

Augeas can be used to manage configuration files. Currently only the `set` command is supported via this state. The `augeas` module also has support for get, match, remove, etc.

> **Warning:** Minimal installations of Debian and Ubuntu have been seen to have packaging bugs with python-augeas, causing the augeas module to fail to import. If the minion has the augeas module installed, and the state fails with a comment saying that the state is unavailable, first restart the salt-minion service. If the problem persists past that, the following command can be run from the master to determine what is causing the import to fail:
>
> ```
> salt minion-id cmd.run 'python -c "from augeas import Augeas"'
> ```
>
> For affected Debian/Ubuntu hosts, installing `libpython2.7` has been known to resolve the issue.

Usage examples:

1. Set the first entry in `/etc/hosts` to `localhost`:

   ```
   hosts:
     augeas.setvalue:
       - changes:
         - /files/etc/hosts/1/canonical: localhost
   ```

2. Add a new host to `/etc/hosts` with the IP address `192.168.1.1` and hostname `test`:

   ```
   hosts:
     augeas.setvalue:
       - changes:
         - /files/etc/hosts/2/ipaddr: 192.168.1.1
         - /files/etc/hosts/2/canonical: foo.bar.com
         - /files/etc/hosts/2/alias[1]: foosite
         - /files/etc/hosts/2/alias[2]: foo
   ```

A prefix can also be set, to avoid redundancy:

```
nginx-conf:
  augeas.setvalue:
    - prefix: /files/etc/nginx/nginx.conf
    - changes:
      - user: www-data
```

```
    - worker_processes: 2
    - http/server_tokens: off
    - http/keepalive_timeout: 65
```

`salt.states.augeas.`**`setvalue`**(*name*, *prefix=None*, *changes=None*, ***kwargs*)
Set a value for a specific augeas path

81.6 salt.states.aws_sqs

81.6.1 Manage SQS Queues

Create and destroy SQS queues. Be aware that this interacts with Amazon's services, and so may incur charges.

This module uses the awscli tool provided by Amazon. This can be downloaded from pip. Also check the documentation for awscli for configuration information.

```
myqueue:
    aws_sqs.exists:
        - region: eu-west-1
```

`salt.states.aws_sqs.`**`absent`**(*name*, *region*, *user=None*, *opts=False*)
Remove the named SQS queue if it exists.

name Name of the SQS queue.

region Region to remove the queue from

user Name of the user performing the SQS operations

opts Include additional arguments and options to the aws command line

`salt.states.aws_sqs.`**`exists`**(*name*, *region*, *user=None*, *opts=False*)
Ensure the SQS queue exists.

name Name of the SQS queue.

region Region to create the queue

user Name of the user performing the SQS operations

opts Include additional arguments and options to the aws command line

81.7 salt.states.cloud

81.7.1 Using states instead of maps to deploy clouds

New in version 2014.1.0: (Hydrogen)

Use this minion to spin up a cloud instance:

```
my-ec2-instance:
  cloud.profile:
    my-ec2-config
```

`salt.states.cloud.`**`absent`**(*name*)
Ensure that no instances with the specified names exist.

CAUTION: This is a destructive state, which will search all configured cloud providers for the named instance, and destroy it.

name The name of the instance to destroy

`salt.states.cloud.`**`present`**(*name*, *provider*, ***kwargs*)

Spin up a single instance on a cloud provider, using salt-cloud. This state does not take a profile argument; rather, it takes the arguments that would normally be configured as part of the state.

Note that while this function does take any configuration argument that would normally be used to create an instance, it will not verify the state of any of those arguments on an existing instance. Stateful properties of an instance should be configured using their own individual state (i.e., cloud.tagged, cloud.untagged, etc).

name The name of the instance to create

provider The name of the cloud provider to use

`salt.states.cloud.`**`profile`**(*name*, *profile*)

Create a single instance on a cloud provider, using a salt-cloud profile.

Note that while profiles used this function do take any configuration argument that would normally be used to create an instance using a profile, this state will not verify the state of any of those arguments on an existing instance. Stateful properties of an instance should be configured using their own individual state (i.e., cloud.tagged, cloud.untagged, etc).

name The name of the instance to create

profile The name of the cloud profile to use

81.8 salt.states.cmd

81.8.1 Execution of arbitrary commands

The cmd state module manages the enforcement of executed commands, this state can tell a command to run under certain circumstances.

A simple example to execute a command:

```
date > /tmp/salt-run:
  cmd.run
```

Only run if another execution failed, in this case truncate syslog if there is no disk space:

```
> /var/log/messages:
  cmd.run:
    - unless: echo 'foo' > /tmp/.test
```

Note that when executing a command or script, the state (i.e., changed or not) of the command is unknown to Salt's state system. Therefore, by default, the `cmd` state assumes that any command execution results in a changed state.

This means that if a `cmd` state is watched by another state then the state that's watching will always be executed due to the *changed* state in the `cmd` state.

Many state functions in this module now also accept a `stateful` argument. If `stateful` is specified to be true then it is assumed that the command or script will determine its own state and communicate it back by following a simple protocol described below:

1. **If there's nothing in the stdout of the command, then assume no changes.** Otherwise, the stdout must be either in JSON or its *last* non-empty line must be a string of key=value pairs delimited by spaces (no spaces on either side of =).

2. **If it's JSON then it must be a JSON object (e.g., {}).** If it's key=value pairs then quoting may be used to include spaces. (Python's shlex module is used to parse the key=value string)

   Two special keys or attributes are recognized in the output:

   ```
   changed: bool (i.e., 'yes', 'no', 'true', 'false', case-insensitive)
   comment: str  (i.e., any string)
   ```

   So, only if `changed` is `True` then assume the command execution has changed the state, and any other key values or attributes in the output will be set as part of the changes.

3. **If there's a comment then it will be used as the comment of the state.**

   Here's an example of how one might write a shell script for use with a stateful command:

   ```
   #!/bin/bash
   #
   echo "Working hard..."

   # writing the state line
   echo  # an empty line here so the next line will be the last.
   echo "changed=yes comment='something has changed' whatever=123"
   ```

   And an example SLS file using this module:

   ```
   Run myscript:
     cmd.run:
       - name: /path/to/myscript
       - cwd: /
       - stateful: True

   Run only if myscript changed something:
     cmd.wait:
       - name: echo hello
       - cwd: /
       - watch:
           - cmd: Run myscript
   ```

   Note that if the `cmd.wait` state also specifies `stateful: True` it can then be watched by some other states as well.

`cmd.wait` is not restricted to watching only cmd states. For example it can also watch a git state for changes

```
# Watch for changes to a git repo and rebuild the project on updates
my-project:
  git.latest:
    - name: git@github.com/repo/foo
    - target: /opt/foo
    - rev: master
  cmd.wait:
    - name: make install
    - cwd: /opt/foo
    - watch:
      - git: my-project
```

Should I use `cmd.run` or `cmd.wait`?

These two states are often confused. The important thing to remember about them is that `cmd.run` states are run each time the SLS file that contains them is applied. If it is more desirable to have a command that only runs after some other state changes, then `cmd.wait` does just that. `cmd.wait` is designed to *watch* other states, and is executed when the state it is watching changes. Example:

```
/usr/local/bin/postinstall.sh:
  cmd:
    - wait
    - watch:
      - pkg: mycustompkg
  file:
    - managed
    - source: salt://utils/scripts/postinstall.sh

mycustompkg:
  pkg:
    - installed
    - require:
      - file: /usr/local/bin/postinstall.sh
```

How do I create a environment from a pillar map?

The map that comes from a pillar cannot be directly consumed by the env option. To use it one must convert it to a list. Example:

```
printenv:
  cmd.run:
    - env:
      {% for key, value in pillar['keys'].iteritems() %}
      - '{{ key }}': '{{ value }}'
      {% endfor %}
```

`salt.states.cmd.`**`call`**(*name*, *func*, *args=()*, *kws=None*, *onlyif=None*, *unless=None*, ***kwargs*)

Invoke a pre-defined Python function with arguments specified in the state declaration. This function is mainly used by the `salt.renderers.pydsl` renderer.

The interpretation of `onlyif` and `unless` arguments are identical to those of `cmd.run`, and all other arguments(`cwd`, `runas`, ...) allowed by `cmd.run` are allowed here, except that their effects apply only to the commands specified in *onlyif* and *unless* rather than to the function to be invoked.

In addition, the `stateful` argument has no effects here.

The return value of the invoked function will be interpreted as follows.

If it's a dictionary then it will be passed through to the state system, which expects it to have the usual structure returned by any salt state function.

Otherwise, the return value (denoted as `result` in the code below) is expected to be a JSON serializable object, and this dictionary is returned:

```
{
    'name': name
    'changes': {'retval': result},
    'result': True if result is None else bool(result),
    'comment': result if isinstance(result, basestring) else ''
}
```

`salt.states.cmd.`**`mod_watch`**(*name*, ***kwargs*)

Execute a cmd function based on a watch call `salt.states.cmd.`**`run`**(*name*, *onlyif=None*, *unless=None*, *cwd=None*, *user=None*, *group=None*, *shell=None*, *env=None*, *stateful=False*, *umask=None*, *output_loglevel='info'*, *quiet=False*, *timeout=None*, ***kwargs*)

Run a command if certain circumstances are met. Use `cmd.wait` if you want to use the `watch` requisite.

name The command to execute, remember that the command will execute with the path and permissions of the salt-minion.

onlyif A command to run as a check, run the named command only if the command passed to the `onlyif` option returns true

unless A command to run as a check, only run the named command if the command passed to the `unless` option returns false

cwd The current working directory to execute the command in, defaults to /root

user The user name to run the command as

group The group context to run the command as

shell The shell to use for execution, defaults to the shell grain

env A list of environment variables to be set prior to execution. Example:

```
salt://scripts/foo.sh:
  cmd.script:
    - env:
      - BATCH: 'yes'
```

> **Warning:** The above illustrates a common PyYAML pitfall, that **yes**, **no**, **on**, **off**, **true**, and **false** are all loaded as boolean `True` and `False` values, and must be enclosed in quotes to be used as strings. More info on this (and other) PyYAML idiosyncrasies can be found *here*.

stateful The command being executed is expected to return data about executing a state

umask The umask (in octal) to use when running the command.

output_loglevel Control the loglevel at which the output from the command is logged. Note that the command being run will still be logged at loglevel INFO regardless, unless `quiet` is used for this value.

quiet The command will be executed quietly, meaning no log entries of the actual command or its return data. This is deprecated as of the **2014.1.0 (Hydrogen)** release, and is being replaced with `output_loglevel: quiet`.

timeout If the command has not terminated after timeout seconds, send the subprocess sigterm, and if sigterm is ignored, follow up with sigkill

Note: cmd.run supports the usage of `reload_modules`. This functionality allows you to force Salt to reload all modules. You should only use `reload_modules` if your cmd.run does some sort of installation (such as `pip`), if you do not reload the modules future items in your state which rely on the software being installed will fail.

```
getpip:
  cmd.run:
    - name: /usr/bin/python /usr/local/sbin/get-pip.py
    - unless: which pip
    - require:
      - pkg: python
      - file: /usr/local/sbin/get-pip.py
    - reload_modules: True
```

`salt.states.cmd.`**`script`**(*name*, *source=None*, *template=None*, *onlyif=None*, *unless=None*, *cwd=None*, *user=None*, *group=None*, *shell=None*, *env=None*, *stateful=False*, *umask=None*, *timeout=None*, ***kwargs*)

Download a script from a remote source and execute it. The name can be the source or the source value can be defined.

source The source script being downloaded to the minion, this source script is hosted on the salt master server. If the file is located on the master in the directory named spam, and is called eggs, the source string is salt://spam/eggs

template If this setting is applied then the named templating engine will be used to render the downloaded file, currently jinja, mako, and wempy are supported

name The command to execute, remember that the command will execute with the path and permissions of the salt-minion.

onlyif A command to run as a check, run the named command only if the command passed to the `onlyif` option returns true

unless A command to run as a check, only run the named command if the command passed to the `unless` option returns false

cwd The current working directory to execute the command in, defaults to /root

user The user name to run the command as

group The group context to run the command as

shell The shell to use for execution, defaults to the shell grain

env A list of environment variables to be set prior to execution. Example:

```
salt://scripts/foo.sh:
  cmd.script:
    - env:
      - BATCH: 'yes'
```

> **Warning:** The above illustrates a common PyYAML pitfall, that **yes**, **no**, **on**, **off**, **true**, and **false** are all loaded as boolean `True` and `False` values, and must be enclosed in quotes to be used as strings. More info on this (and other) PyYAML idiosyncrasies can be found *here*.

umask The umask (in octal) to use when running the command.

stateful The command being executed is expected to return data about executing a state

timeout If the command has not terminated after timeout seconds, send the subprocess sigterm, and if sigterm is ignored, follow up with sigkill

args String of command line args to pass to the script. Only used if no args are specified as part of the *name* argument.

`salt.states.cmd.`**`wait`**(*name*, *onlyif=None*, *unless=None*, *cwd=None*, *user=None*, *group=None*, *shell=None*, *env=()*, *stateful=False*, *umask=None*, ***kwargs*)

Run the given command only if the watch statement calls it

name The command to execute, remember that the command will execute with the path and permissions of the salt-minion.

onlyif A command to run as a check, run the named command only if the command passed to the `onlyif` option returns true

unless A command to run as a check, only run the named command if the command passed to the `unless` option returns false

cwd The current working directory to execute the command in, defaults to /root

user The user name to run the command as

group The group context to run the command as

shell The shell to use for execution, defaults to /bin/sh

env A list of environment variables to be set prior to execution. Example:

```
salt://scripts/foo.sh:
  cmd.script:
    - env:
      - BATCH: 'yes'
```

> **Warning:** The above illustrates a common PyYAML pitfall, that **yes**, **no**, **on**, **off**, **true**, and **false** are all loaded as boolean `True` and `False` values, and must be enclosed in quotes to be used as strings. More info on this (and other) PyYAML idiosyncrasies can be found *here*.

umask The umask (in octal) to use when running the command.

stateful The command being executed is expected to return data about executing a state

`salt.states.cmd.`**`wait_call`**(*name*, *func*, *args=()*, *kws=None*, *onlyif=None*, *unless=None*, *stateful=False*, ***kwargs*)

`salt.states.cmd.`**`wait_script`**(*name*, *source=None*, *template=None*, *onlyif=None*, *unless=None*, *cwd=None*, *user=None*, *group=None*, *shell=None*, *env=None*, *stateful=False*, *umask=None*, ***kwargs*)

Download a script from a remote source and execute it only if a watch statement calls it.

source The source script being downloaded to the minion, this source script is hosted on the salt master server. If the file is located on the master in the directory named spam, and is called eggs, the source string is salt://spam/eggs

template If this setting is applied then the named templating engine will be used to render the downloaded file, currently jinja, mako, and wempy are supported

name The command to execute, remember that the command will execute with the path and permissions of the salt-minion.

onlyif A command to run as a check, run the named command only if the command passed to the `onlyif` option returns true

unless A command to run as a check, only run the named command if the command passed to the `unless` option returns false

cwd The current working directory to execute the command in, defaults to /root

user The user name to run the command as

group The group context to run the command as

shell The shell to use for execution, defaults to the shell grain

env A list of environment variables to be set prior to execution. Example:

```
salt://scripts/foo.sh:
  cmd.script:
    - env:
      - BATCH: 'yes'
```

> **Warning:** The above illustrates a common PyYAML pitfall, that **yes**, **no**, **on**, **off**, **true**, and **false** are all loaded as boolean `True` and `False` values, and must be enclosed in quotes to be used as strings. More info on this (and other) PyYAML idiosyncrasies can be found *here*.

umask The umask (in octal) to use when running the command.

stateful The command being executed is expected to return data about executing a state

81.9 salt.states.composer

81.9.1 Installation of Composer Packages

These states manage the installed packages for composer for PHP. Note that either composer is installed and accessible via a bin directory or you can pass the location of composer in the state.

```
get-composer:
  cmd.run:
    - name: 'CURL=`which curl`; $CURL -sS https://getcomposer.org/installer | php'
    - unless: test -f /usr/local/bin/composer
    - cwd: /root/

install-composer:
  cmd.wait:
    - name: mv /root/composer.phar /usr/local/bin/composer
    - cwd: /root/
    - watch:
      - cmd: get-composer

/path/to/project:
  composer.installed:
    - no_dev: true
    - require:
      - cmd: install-composer


# Without composer installed in your PATH

/path/to/composer:
  composer.installed:
    - composer: /path/to/composer.phar
    - php: /usr/local/bin/php
    - no_dev: true
```

`salt.states.composer.`**`installed`**(*name*, *composer=None*, *php=None*, *runas=None*, *user=None*, *prefer_source=None*, *prefer_dist=None*, *no_scripts=None*, *no_plugins=None*, *optimize=None*, *no_dev=None*, *composer_home='/root'*)

Verify that composer has installed the latest packages give a `composer.json` and `composer.lock` file in a directory.

dir Directory location of the composer.json file.

composer Location of the composer.phar file. If not set composer will just execute "composer" as if it is installed globally. (i.e. /path/to/composer.phar)

php Location of the php executible to use with composer. (i.e. /usr/bin/php)

runas Which system user to run composer as.

Deprecated since version 2014.1.4: (Hydrogen)

user Which system user to run composer as.

New in version 2014.1.4: (Hydrogen)

prefer_source –prefer-source option of composer.

prefer_dist –prefer-dist option of composer.

no_scripts –no-scripts option of composer.

no_plugins –no-plugins option of composer.

optimize –optimize-autoloader option of composer. Recommended for production.

no_dev –no-dev option for composer. Recommended for production.

quiet –quiet option for composer. Whether or not to return output from composer.

composer_home $COMPOSER_HOME environment variable

81.10 salt.states.cron

81.10.1 Management of cron, the Unix command scheduler

Cron declarations require a number of parameters. The following are the parameters used by Salt to define the various timing values for a cron job:

- `minute`
- `hour`
- `daymonth`
- `month`
- `dayweek` (0 to 6 are Sunday through Saturday, 7 can also be used for Sunday)

Warning: Any timing arguments not specified take a value of `*`. This means that setting `hour` to `5`, while not defining the `minute` param, will result in Salt adding a job that will execute every minute between 5 and 6 A.M.! Additionally, the default user for these states is `root`. Therefore, if the cron job is for another user, it is necessary to specify that user with the `user` parameter.

A long time ago (before 2014.2), when making changes to an existing cron job, the name declaration is the parameter used to uniquely identify the job, so if an existing cron that looks like this:

```
date > /tmp/crontest:
  cron.present:
    - user: root
    - minute: 5
```

Is changed to this:

```
date > /tmp/crontest:
  cron.present:
    - user: root
    - minute: 7
    - hour: 2
```

Then the existing cron will be updated, but if the cron command is changed, then a new cron job will be added to the user's crontab.

The current behavior is still relying on that mechanism, but you can also specify an identifier to identify your crontabs: .. versionadded:: 2014.2 .. code-block:: yaml

date > /tmp/crontest:

cron.present:

- identifier: SUPERCRON
- user: root
- minute: 7
- hour: 2

And, some months later, you modify it: .. versionadded:: 2014.2 .. code-block:: yaml

superscript > /tmp/crontest:

cron.present:

- identifier: SUPERCRON
- user: root
- minute: 3
- hour: 4

The old **date > /tmp/crontest** will be replaced by **superscript > /tmp/crontest**.

Additionally, Salt also supports running a cron every `x minutes` very similarly to the Unix convention of using `*/5` to have a job run every five minutes. In Salt, this looks like:

```
date > /tmp/crontest:
  cron.present:
    - user: root
    - minute: '*/5'
```

The job will now run every 5 minutes.

Additionally, the temporal parameters (minute, hour, etc.) can be randomized by using `random` instead of using a specific value. For example, by using the `random` keyword in the `minute` parameter of a cron state, the same cron job can be pushed to hundreds or thousands of hosts, and they would each use a randomly-generated minute. This can be helpful when the cron job accesses a network resource, and it is not desirable for all hosts to run the job concurrently.

```
/path/to/cron/script:
  cron.present:
    - user: root
    - minute: random
    - hour: 2
```

New in version 0.16.0.

Since Salt assumes a value of `*` for unspecified temporal parameters, adding a parameter to the state and setting it to `random` will change that value from `*` to a randomized numeric value. However, if that field in the cron entry on the minion already contains a numeric value, then using the `random` keyword will not modify it.

`salt.states.cron.`**`absent`**(*name*, *user='root'*, *identifier=None*, ***kwargs*)

Verifies that the specified cron job is absent for the specified user; only the name is matched when removing a cron job.

name The command that should be absent in the user crontab.

user The name of the user whose crontab needs to be modified, defaults to the root user

identifier Custom-defined identifier for tracking the cron line for future crontab edits. This defaults to the state id

`salt.states.cron.`**`env_absent`**`(`*name*, *user='root'*`)`

Verifies that the specified environment variable is absent from the crontab for the specified user

name The name of the environment variable to remove from the user crontab

user The name of the user whose crontab needs to be modified, defaults to the root user

`salt.states.cron.`**`env_present`**`(`*name*, *value=None*, *user='root'*`)`

Verifies that the specified environment variable is present in the crontab for the specified user.

name The name of the environment variable to set in the user crontab

user The name of the user whose crontab needs to be modified, defaults to the root user

value The value to set for the given environment variable

`salt.states.cron.`**`file`**`(`*name*, *source_hash=''*, *user='root'*, *template=None*, *context=None*, *replace=True*, *defaults=None*, *env=None*, *backup=''*, ***kwargs*`)`

Provides file.managed-like functionality (templating, etc.) for a pre-made crontab file, to be assigned to a given user.

name The source file to be used as the crontab. This source file can be hosted on either the salt master server, or on an HTTP or FTP server. For files hosted on the salt file server, if the file is located on the master in the directory named spam, and is called eggs, the source string is salt://spam/eggs.

If the file is hosted on a HTTP or FTP server then the source_hash argument is also required

source_hash This can be either a file which contains a source hash string for the source, or a source hash string. The source hash string is the hash algorithm followed by the hash of the file: md5=e138491e9d5b97023cea823fe17bac22

user The user to whom the crontab should be assigned. This defaults to root.

template If this setting is applied then the named templating engine will be used to render the downloaded file. Currently, jinja and mako are supported.

context Overrides default context variables passed to the template.

replace If the crontab should be replaced, if False then this command will be ignored if a crontab exists for the specified user. Default is True.

defaults Default context passed to the template.

backup Overrides the default backup mode for the user's crontab.

`salt.states.cron.`**`present`**`(`*name*, *user='root'*, *minute='*'*, *hour='*'*, *daymonth='*'*, *month='*'*, *dayweek='*'*, *comment=None*, *identifier=None*`)`

Verifies that the specified cron job is present for the specified user. For more advanced information about what exactly can be set in the cron timing parameters, check your cron system's documentation. Most Unix-like systems' cron documentation can be found via the crontab man page: `man 5 crontab`.

name The command that should be executed by the cron job.

user The name of the user whose crontab needs to be modified, defaults to the root user

minute The information to be set into the minute section, this can be any string supported by your cron system's the minute field. Default is `*`

hour The information to be set in the hour section. Default is `*`

daymonth The information to be set in the day of month section. Default is `*`

month The information to be set in the month section. Default is `*`

dayweek The information to be set in the day of week section. Default is `*`

comment User comment to be added on line previous the cron job

identifier Custom-defined identifier for tracking the cron line for future crontab edits. This defaults to the state id

81.11 salt.states.ddns

81.11.1 Dynamic DNS updates

Ensure a DNS record is present or absent utilizing RFC 2136 type dynamic updates. Requires dnspython module.

```
webserver:
  ddns.present:
    - zone: example.com
    - ttl: 60
```

`salt.states.ddns.`**`absent`**`(`*name*, *zone*, *data=None*, *rdtype=None*`)`

Ensures that the named DNS record is absent.

name The host portion of the DNS record, e.g., 'webserver'

zone The zone to check

data Data for the DNS record. E.g., the IP addres for an A record. If omitted, all records matching name (and rdtype, if provided) will be purged.

rdtype DNS resource type. If omitted, all types will be purged.

`salt.states.ddns.`**`present`**`(`*name*, *zone*, *ttl*, *data*, *rdtype='A'*`)`

Ensures that the named DNS record is present with the given ttl.

name The host portion of the DNS record, e.g., 'webserver'

zone The zone to check/update

ttl TTL for the record

data Data for the DNS record. E.g., the IP addres for an A record.

rdtype DNS resource type. Default 'A'.

81.12 salt.states.debconfmod

81.12.1 Management of debconf selections

The debconfmod state module manages the enforcement of debconf selections, this state can set those selections prior to package installation.

Available Functions

The debconfmod state has two functions, the `set` and `set_file` functions

set Set debconf selections from the state itself

set_file Set debconf selections from a file

```
nullmailer-debconf:
  debconf.set:
    - name: nullmailer
    - data:
        'shared/mailname': {'type': 'string', 'value': 'server.domain.tld'}
        'nullmailer/relayhost': {'type': 'string', 'value': 'mail.domain.tld'}
ferm-debconf:
  debconf.set:
    - name: ferm
    - data:
        'ferm/enable': {'type': 'boolean', 'value': True}
```

Note: Due to how PyYAML imports nested dicts (see *here*), the values in the `data` dict must be indented four spaces instead of two.

`salt.states.debconfmod.`**`set`**`(`*name*, *data*`)`

Set debconf selections

```
<state_id>:
  debconf.set:
    - name: <name>
    - data:
        <question>: {'type': <type>, 'value': <value>}
        <question>: {'type': <type>, 'value': <value>}

<state_id>:
  debconf.set:
    - name: <name>
    - data:
        <question>: {'type': <type>, 'value': <value>}
        <question>: {'type': <type>, 'value': <value>}
```

name: The package name to set answers for.

data: A set of questions/answers for debconf. Note that everything under this must be indented twice.

question: The question the is being pre-answered

type: The type of question that is being asked (string, boolean, select, etc.)

value: The answer to the question

`salt.states.debconfmod.`**`set_file`**`(`*name*, *source*, ***kwargs*`)`

Set debconf selections from a file

```
<state_id>:
  debconf.set_file:
    - source: salt://pathto/pkg.selections

<state_id>:
  debconf.set_file:
    - source: salt://pathto/pkg.selections?saltenv=myenvironment
```

source: The location of the file containing the package selections

81.13 salt.states.disk

Disk monitoring state

Monitor the state of disk resources

`salt.states.disk.`**`status`**(*name*, *maximum=None*, *minimum=None*)
Return the current disk usage stats for the named device

81.14 salt.states.dockerio

81.14.1 Manage Docker containers

Docker is a lightweight, portable, self-sufficient software container wrapper. The base supported wrapper type is LXC, cgroups, and the Linux Kernel.

Warning: This state module is beta. The API is subject to change. No promise as to performance or functionality is yet present.

Note: This state module requires docker-py which supports Docker Remote API version 1.6.

Available Functions

- built

```
corp/mysuperdocker_img:
    docker.built:
        - path: /path/to/dir/container/Dockerfile
```

- pulled

```
ubuntu:
  docker.pulled
```

- installed

```
mysuperdocker-container:
    docker.installed:
        - name: mysuperdocker
        - hostname: superdocker
        - image: corp/mysuperdocker_img
```

- running

```
my_service:
    docker.running:
        - container: mysuperdocker
        - port_bindings:
            "5000/tcp":
                HostIp: ""
                HostPort: "5000"
```

- absent

```
mys_old_uperdocker:
   docker.absent
```

- run

```
/finish-install.sh:
    docker.run:
        - container: mysuperdocker
        - unless: grep -q something /var/log/foo
        - docker_unless: grep -q done /install_log
```

Note: The docker modules are named *dockerio* because the name 'docker' would conflict with the underlying docker-py library.

We should add magic to all methods to also match containers by name now that the 'naming link' stuff has been merged in docker. This applies for exemple to:

- running
- absent
- run
- script

`salt.states.dockerio.`**`absent`**(*name*)

Ensure that the container is absent; if not, it will will be killed and destroyed. (*docker inspect*)

name: Either the container name or id

`salt.states.dockerio.`**`built`**(*name*, *path=None*, *quiet=False*, *nocache=False*, *rm=True*, *force=False*, *timeout=None*, **args*, ***kwargs*)

Build a docker image from a path or URL to a dockerfile. (*docker build*)

name Tag of the image

path URL (e.g. *url/branch/docker_dir/dockerfile*) or filesystem path to the dockerfile

`salt.states.dockerio.`**`installed`**(*name*, *image*, *command=None*, *hostname=None*, *user=None*, *detach=True*, *stdin_open=False*, *tty=False*, *mem_limit=0*, *ports=None*, *environment=None*, *dns=None*, *volumes=None*, *volumes_from=None*, **args*, ***kwargs*)

Ensure that a container with the given name exists; if not, build a new container from the specified image. (*docker run*)

name Name for the container

image Image from which to build this container

environment

Environment variables for the container, either

- a mapping of key, values
- a list of mappings of key values

ports

List of ports definitions, either:

- a port to map

   - a mapping of mapping portInHost : PortInContainer

volumes List of volumes

For other parameters, see absolutely first the salt.modules.dockerio execution module and the docker-py python bindings for docker documentation <https://github.com/dotcloud/docker-py#api>‘_ for *docker.create_container*.

Note: This command does not verify that the named container is running the specified image.

`salt.states.dockerio.`**`mod_watch`**(*name*, *sfun=None*, **args*, ***kw*)

`salt.states.dockerio.`**`present`**(*name*)

If a container with the given name is not present, this state will fail. (*docker inspect*)

name: container id

`salt.states.dockerio.`**`pulled`**(*name*, *force=False*, **args*, ***kwargs*)

Pull an image from a docker registry. (*docker pull*)

Note: See first the documentation for *docker login*, *docker pull*, *docker push*, and docker.import_image (docker import). NOTE that We added saltack a way to identify yourself via pillar, see in the salt.modules.dockerio execution module how to ident yourself via the pillar.

---

name Tag of the image

force Pull even if the image is already pulled

`salt.states.dockerio.`**`run`**(*name*, *cid=None*, *hostname=None*, *stateful=False*, *onlyif=None*, *unless=None*, *docked_onlyif=None*, *docked_unless=None*, **args*, ***kwargs*)

Run a command in a specific container

You can match by either name or hostname

name command to run in the container

cid Container id

state_id state_id

stateful stateful mode

onlyif Only execute cmd if statement on the host returns 0

unless Do not execute cmd if statement on the host returns 0

docked_onlyif Only execute cmd if statement in the container returns 0

docked_unless Do not execute cmd if statement in the container returns 0

`salt.states.dockerio.`**`running`**(*name*, *container=None*, *port_bindings=None*, *binds=None*, *publish_all_ports=False*, *links=None*, *lxc_conf=None*, *privileged=False*)

Ensure that a container is running. (*docker inspect*)

name name of the service

container name of the container to start

binds like -v of docker run command

```
- binds:
    - /var/log/service: /var/log/service
```

publish_all_ports

links Link several container together

```
- links:
    name_other_container: alias_for_other_container
```

port_bindings

List of ports to expose on host system

- a mapping port's guest, hostname's host and port's host.

```
- port_bindings:
    "5000/tcp":
        HostIp: ""
        HostPort: "5000"
```

`salt.states.dockerio.`**`script`**`(`*name*, *cid=None*, *hostname=None*, *state_id=None*, *stateful=False*, *onlyif=None*, *unless=None*, *docked_onlyif=None*, *docked_unless=None*, **args*, ***kwargs*`)`

Run a command in a specific container

XXX: TODO: IMPLEMENT

Matching can be done by either name or hostname

name command to run in the docker

cid Container id

state_id State Id

stateful stateful mode

onlyif Only execute cmd if statement on the host return 0

unless Do not execute cmd if statement on the host return 0

docked_onlyif Only execute cmd if statement in the container returns 0

docked_unless Do not execute cmd if statement in the container returns 0

81.15 salt.states.eselect

81.15.1 Management of Gentoo configuration using eselect

A state module to manage Gentoo configuration via eselect

```
profile:
    eselect.set:
        target: hardened/linux/amd64
```

`salt.states.eselect.`**`set_`**`(`*name*, *target*`)`

Verify that the given module is set to the given target

name The name of the module

81.16 salt.states.file

81.16.1 Operations on regular files, special files, directories, and symlinks

Salt States can aggressively manipulate files on a system. There are a number of ways in which files can be managed.

Regular files can be enforced with the `managed` function. This function downloads files from the salt master and places them on the target system. The downloaded files can be rendered as a jinja, mako, or wempy template, adding a dynamic component to file management. An example of `file.managed` which makes use of the jinja templating system would look like this:

```
/etc/http/conf/http.conf:
  file.managed:
    - source: salt://apache/http.conf
    - user: root
    - group: root
    - mode: 644
    - template: jinja
    - defaults:
        custom_var: "default value"
        other_var: 123
{% if grains['os'] == 'Ubuntu' %}
    - context:
        custom_var: "override"
{% endif %}
```

Note: When using both the `defaults` and `context` arguments, note the extra indentation (four spaces instead of the normal two). This is due to an idiosyncrasy of how PyYAML loads nested dictionaries, and is explained in greater detail *here*.

If using a template, any user-defined template variables in the file defined in `source` must be passed in using the `defaults` and/or `context` arguments. The general best practice is to place default values in `defaults`, with conditional overrides going into `context`, as seen above.

The `source` parameter can be specified as a list. If this is done, then the first file to be matched will be the one that is used. This allows you to have a default file on which to fall back if the desired file does not exist on the salt fileserver. Here's an example:

```
/etc/foo.conf:
  file.managed:
    - source:
      - salt://foo.conf.{{ grains['fqdn'] }}
      - salt://foo.conf.fallback
    - user: foo
    - group: users
    - mode: 644
    - backup: minion
```

Note: Salt supports backing up managed files via the backup option. For more details on this functionality please review the *backup_mode documentation*.

The `source` parameter can also specify a file in another Salt environment. In this example `foo.conf` in the `dev` environment will be used instead.

```
/etc/foo.conf:
  file.managed:
    - source:
      - salt://foo.conf?saltenv=dev
    - user: foo
    - group: users
    - mode: '0644'
```

Warning: When using a mode that includes a leading zero you must wrap the value in single quotes. If the value is not wrapped in quotes it will be read by YAML as an integer and evaluated as an octal.

Special files can be managed via the `mknod` function. This function will create and enforce the permissions on a special file. The function supports the creation of character devices, block devices, and fifo pipes. The function will create the directory structure up to the special file if it is needed on the minion. The function will not overwrite or operate on (change major/minor numbers) existing special files with the exception of user, group, and permissions. In most cases the creation of some special files require root permisisons on the minion. This would require that the minion to be run as the root user. Here is an example of a character device:

```
/var/named/chroot/dev/random:
  file.mknod:
    - ntype: c
    - major: 1
    - minor: 8
    - user: named
    - group: named
    - mode: 660
```

Here is an example of a block device:

```
/var/named/chroot/dev/loop0:
  file.mknod:
    - ntype: b
    - major: 7
    - minor: 0
    - user: named
    - group: named
    - mode: 660
```

Here is an example of a fifo pipe:

```
/var/named/chroot/var/log/logfifo:
  file.mknod:
    - ntype: p
    - user: named
    - group: named
    - mode: 660
```

Directories can be managed via the `directory` function. This function can create and enforce the permissions on a directory. A directory statement will look like this:

```
/srv/stuff/substuf:
  file.directory:
    - user: fred
    - group: users
    - mode: 755
    - makedirs: True
```

If you need to enforce user and/or group ownership or permissions recursively on the directory's contents, you can do so by adding a `recurse` directive:

```
/srv/stuff/substuf:
  file.directory:
    - user: fred
    - group: users
    - mode: 755
    - makedirs: True
    - recurse:
      - user
      - group
      - mode
```

As a default, `mode` will resolve to `dir_mode` and `file_mode`, to specify both directory and file permissions, use this form:

```
/srv/stuff/substuf:
  file.directory:
    - user: fred
    - group: users
    - file_mode: 744
    - dir_mode: 755
    - makedirs: True
    - recurse:
      - user
      - group
      - mode
```

Symlinks can be easily created; the symlink function is very simple and only takes a few arguments:

```
/etc/grub.conf:
  file.symlink:
    - target: /boot/grub/grub.conf
```

Recursive directory management can also be set via the `recurse` function. Recursive directory management allows for a directory on the salt master to be recursively copied down to the minion. This is a great tool for deploying large code and configuration systems. A state using `recurse` would look something like this:

```
/opt/code/flask:
  file.recurse:
    - source: salt://code/flask
    - include_empty: True
```

A more complex `recurse` example:

```
{% set site_user = 'testuser' %}
{% set site_name = 'test_site' %}
{% set project_name = 'test_proj' %}
{% set sites_dir = 'test_dir' %}

django-project:
  file.recurse:
    - name: {{ sites_dir }}/{{ site_name }}/{{ project_name }}
    - user: {{ site_user }}
    - dir_mode: 2775
    - file_mode: '0644'
    - template: jinja
    - source: salt://project/templates_dir
    - include_empty: True
```

`salt.states.file.`**`absent`**(*name*)

Verify that the named file or directory is absent, this will work to reverse any of the functions in the file state module.

name The path which should be deleted

`salt.states.file.`**`accumulated`**(*name*, *filename*, *text*, ***kwargs*)

Prepare accumulator which can be used in template in file.managed state. Accumulator dictionary becomes available in template. It can also be used in file.blockreplace.

name Accumulator name

filename Filename which would receive this accumulator (see file.managed state documentation about `name`)

text String or list for adding in accumulator

require_in / watch_in One of them required for sure we fill up accumulator before we manage the file. Probably the same as filename

Example:

Given the following:

```
animals_doing_things:
  file.accumulated:
    - filename: /tmp/animal_file.txt
    - text: ’ jumps over the lazy dog.’
    - require_in:
      - file: animal_file

animal_file:
  file.managed:
    - name: /tmp/animal_file.txt
    - source: salt://animal_file.txt
    - template: jinja
```

One might write a template for animal_file.txt like the following:

```
The quick brown fox{% for animal in accumulator[’animals_doing_things’] %}{{ animal }}{% endfor
```

Collectively, the above states and template file will produce:

```
The quick brown fox jumps over the lazy dog.
```

Multiple accumulators can be “chained” together.

Note: The ‘accumulator’ data structure is a Python dictionary. Do not expect any loop over the keys in a deterministic order!

`salt.states.file.`**`append`**(*name*, *text=None*, *makedirs=False*, *source=None*, *source_hash=None*, *template=’jinja’*, *sources=None*, *source_hashes=None*, *defaults=None*, *context=None*)

Ensure that some text appears at the end of a file

The text will not be appended again if it already exists in the file. You may specify a single line of text or a list of lines to append.

Multi-line example:

```
/etc/motd:
  file.append:
    - text: |
        Thou hadst better eat salt with the Philosophers of Greece,
        than sugar with the Courtiers of Italy.
        - Benjamin Franklin
```

Multiple lines of text:

```
/etc/motd:
  file.append:
    - text:
      - Trust no one unless you have eaten much salt with him.
      - "Salt is born of the purest of parents: the sun and the sea."
```

Gather text from multiple template files:

```
/etc/motd:
  file:
      - append
      - template: jinja
      - sources:
          - salt://motd/devops-messages.tmpl
          - salt://motd/hr-messages.tmpl
          - salt://motd/general-messages.tmpl
```

New in version 0.9.5.

`salt.states.file.`**`blockreplace`**(*name*, *marker_start=’#– start managed zone –‘*, *marker_end=’#– end managed zone –‘*, *content=’‘*, *append_if_not_found=False*, *prepend_if_not_found=False*, *backup=’.bak’*, *show_changes=True*)

Maintain an edit in a file in a zone delimited by two line markers

New in version 2014.1.0.

A block of content delimited by comments can help you manage several lines entries without worrying about old entries removal. This can help you maintaining an un-managed file containing manual edits. Note: this function will store two copies of the file in-memory (the original version and the edited version) in order to detect changes and only edit the targeted file if necessary.

Parameters

- **name** – Filesystem path to the file to be edited
- **marker_start** – The line content identifying a line as the start of the content block. Note that the whole line containing this marker will be considered, so whitespaces or extra content before or after the marker is included in final output
- **marker_end** – The line content identifying a line as the end of the content block. Note that the whole line containing this marker will be considered, so whitespaces or extra content before or after the marker is included in final output. Note: you can use file.accumulated and target this state. All accumulated data dictionaries content will be added as new lines in the content.
- **content** – The content to be used between the two lines identified by marker_start and marker_stop.
- **append_if_not_found** – False by default, if markers are not found and set to True then the markers and content will be appended to the file
- **prepend_if_not_found** – False by default, if markers are not found and set to True then the markers and content will be prepended to the file
- **backup** – The file extension to use for a backup of the file if any edit is made. Set to `False` to skip making a backup.

- **dry_run** – Don't make any edits to the file
- **show_changes** – Output a unified diff of the old file and the new file. If `False` return a boolean if any changes were made.

Return type

bool or str

Example of usage with an accumulator and with a variable:

```
{% set myvar = 42 %}
hosts-config-block-{{ myvar }}:
  file.blockreplace:
    - name: /etc/hosts
    - marker_start: "# START managed zone {{ myvar }} -DO-NOT-EDIT-"
    - marker_end: "# END managed zone {{ myvar }} --"
    - content: 'First line of content'
    - append_if_not_found: True
    - backup: '.bak'
    - show_changes: True

hosts-config-block-{{ myvar }}-accumulated1:
  file.accumulated:
    - filename: /etc/hosts
    - name: my-accumulator-{{ myvar }}
    - text: "text 2"
    - require_in:
      - file: hosts-config-block-{{ myvar }}

hosts-config-block-{{ myvar }}-accumulated2:
  file.accumulated:
    - filename: /etc/hosts
    - name: my-accumulator-{{ myvar }}
    - text: |
         text 3
         text 4
    - require_in:
      - file: hosts-config-block-{{ myvar }}
```

will generate and maintain a block of content in `/etc/hosts`:

```
# START managed zone 42 -DO-NOT-EDIT-
First line of content
text 2
text 3
text 4
# END managed zone 42 --
```

`salt.states.file.`**`comment`**(*name*, *regex*, *char='#'*, *backup='.bak'*)

Comment out specified lines in a file.

name The full path to the file to be edited

regex A regular expression used to find the lines that are to be commented; this pattern will be wrapped in parenthesis and will move any preceding/trailing `^` or `$` characters outside the parenthesis (e.g., the pattern `^foo$` will be rewritten as `^(foo)$`) Note that you _need_ the leading ^, otherwise each time you run highstate, another comment char will be inserted.

char [`#`] The character to be inserted at the beginning of a line in order to comment it out

backup [`.bak`] The file will be backed up before edit with this file extension

> **Warning:** This backup will be overwritten each time `sed` / `comment` / `uncomment` is called. Meaning the backup will only be useful after the first invocation.

Usage:

```
/etc/fstab:
  file.comment:
    - regex: ^bind 127.0.0.1
```

New in version 0.9.5.

`salt.states.file.`**`copy`**(*name*, *source*, *force=False*, *makedirs=False*)

If the source file exists on the system, copy it to the named file. The named file will not be overwritten if it already exists unless the force option is set to True.

name The location of the file to copy to

source The location of the file to copy to the location specified with name

force If the target location is present then the file will not be moved, specify "force: True" to overwrite the target file

makedirs If the target subdirectories don't exist create them

`salt.states.file.`**`directory`**(*name*, *user=None*, *group=None*, *recurse=None*, *dir_mode=None*, *file_mode=None*, *makedirs=False*, *clean=False*, *require=None*, *exclude_pat=None*, *follow_symlinks=False*, ***kwargs*)

Ensure that a named directory is present and has the right perms

name The location to create or manage a directory

user The user to own the directory; this defaults to the user salt is running as on the minion

group The group ownership set for the directory; this defaults to the group salt is running as on the minion

recurse Enforce user/group ownership and mode of directory recursively. Accepts a list of strings representing what you would like to recurse. Example:

```
/var/log/httpd:
    file.directory:
    - user: root
    - group: root
    - dir_mode: 755
    - file_mode: 644
    - recurse:
        - user
        - group
        - mode
```

dir_mode / mode The permissions mode to set any directories created.

file_mode The permissions mode to set any files created if 'mode' is ran in 'recurse'. This defaults to dir_mode.

makedirs If the directory is located in a path without a parent directory, then the state will fail. If makedirs is set to True, then the parent directories will be created to facilitate the creation of the named file.

clean Make sure that only files that are set up by salt and required by this function are kept. If this option is set then everything in this directory will be deleted unless it is required.

require Require other resources such as packages or files

exclude_pat When 'clean' is set to True, exclude this pattern from removal list and preserve in the destination.

follow_symlinks [False] If the desired path is a symlink (or `recurse` is defined and a symlink is encountered while recursing), follow it and check the permissions of the directory/file to which the symlink points.

New in version 2014.1.4.

`salt.states.file.`**`exists`**(*name*)

Verify that the named file or directory is present or exists. Ensures pre-requisites outside of Salt's purview (e.g., keytabs, private keys, etc.) have been previously satisfied before deployment.

name Absolute path which must exist

`salt.states.file.`**`managed`**(*name*, *source=None*, *source_hash=''*, *user=None*, *group=None*, *mode=None*, *template=None*, *makedirs=False*, *dir_mode=None*, *context=None*, *replace=True*, *defaults=None*, *env=None*, *backup=''*, *show_diff=True*, *create=True*, *contents=None*, *contents_pillar=None*, ***kwargs*)

Manage a given file, this function allows for a file to be downloaded from the salt master and potentially run through a templating system.

name The location of the file to manage

source The source file to download to the minion, this source file can be hosted on either the salt master server, or on an HTTP or FTP server. For files hosted on the salt file server, if the file is located on the master in the directory named spam, and is called eggs, the source string is salt://spam/eggs. If source is left blank or None (use ~ in YAML), the file will be created as an empty file and the content will not be managed

If the file is hosted on a HTTP or FTP server then the source_hash argument is also required

source_hash

This can be one of the following:

1. a source hash string
2. the URI of a file that contains source hash strings

The function accepts the first encountered long unbroken alphanumeric string of correct length as a valid hash, in order from most secure to least secure:

```
Type    Length
======  ======
sha512     128
sha384      96
sha256      64
sha224      56
sha1        40
md5         32
```

The file can contain several checksums for several files. Each line must contain both the file name and the hash. If no file name is matched, the first hash encountered will be used, otherwise the most secure hash with the correct source file name will be used.

Debian file type `*.dsc` is supported.

Examples:

```
/etc/rc.conf ef6e82e4006dee563d98ada2a2a80a27
sha254c8525aee419eb649f0233be91c151178b30f0dff8ebbdcc8de71b1d5c8bcc06a  /etc/resolv.conf
ead48423703509d37c4a90e6a0d53e143b6fc268
```

Known issues: If the remote server URL has the hash file as an apparent sub-directory of the source file, the module will discover that it has already cached a directory where a file should be cached. For example:

```
tomdroid-src-0.7.3.tar.gz:
  file.managed:
    - name: /tmp/tomdroid-src-0.7.3.tar.gz
    - source: https://launchpad.net/tomdroid/beta/0.7.3/+download/tomdroid-src-0.7.3.tar
    - source_hash: https://launchpad.net/tomdroid/beta/0.7.3/+download/tomdroid-src-0.7.
```

user The user to own the file, this defaults to the user salt is running as on the minion

group The group ownership set for the file, this defaults to the group salt is running as on the minion

mode The permissions to set on this file, aka 644, 0775, 4664

template If this setting is applied then the named templating engine will be used to render the downloaded file, currently jinja, mako, and wempy are supported

makedirs If the file is located in a path without a parent directory, then the state will fail. If makedirs is set to True, then the parent directories will be created to facilitate the creation of the named file.

dir_mode If directories are to be created, passing this option specifies the permissions for those directories. If this is not set, directories will be assigned permissions from the 'mode' argument.

replace If this file should be replaced. If false, this command will not overwrite file contents but will enforce permissions if the file exists already. Default is True.

context Overrides default context variables passed to the template.

defaults Default context passed to the template.

backup Overrides the default backup mode for this specific file.

show_diff If set to False, the diff will not be shown.

create Default is True, if create is set to False then the file will only be managed if the file already exists on the system.

contents Default is None. If specified, will use the given string as the contents of the file. Should not be used in conjunction with a source file of any kind. Ignores hashes and does not use a templating engine.

contents_pillar New in version 0.17.0.

Operates like `contents`, but draws from a value stored in pillar, using the pillar path syntax used in `pillar.get`. This is useful when the pillar value contains newlines, as referencing a pillar variable using a jinja/mako template can result in YAML formatting issues due to the newlines causing indentation mismatches.

For example, the following could be used to deploy an SSH private key:

```
/home/deployer/.ssh/id_rsa:
  file.managed:
    - user: deployer
    - group: deployer
    - mode: 600
    - contents_pillar: userdata:deployer:id_rsa
```

This would populate `/home/deployer/.ssh/id_rsa` with the contents of `pillar['userdata']['deployer']['id_rsa']`. An example of this pillar setup would be like so:

```
userdata:
  deployer:
    id_rsa: |
      -----BEGIN RSA PRIVATE KEY-----
```

```
MIIEowIBAAKCAQEAoQiwO3JhBquPAalQF9qP1lLZNXVjYMIswrMe2HcWUVBgh+vY
U7sCwx/dH6+VvNwmCoqmNnP+8gTPKGl1vgAObJAnMT623dMXjVKwnEagZPRJIxDy
B/HaAre9euNiY3LvIzBTWRSeMfT+rWvIKVBpvwlgGrfgz70m0pqxu+UyFbAGLin+
GpxzZAMaFpZw4sSbIlRuissXZj/sHpQb8p9M5IeO4Z3rjkCP1cxI
-----END RSA PRIVATE KEY-----
```

Note: The private key above is shortened to keep the example brief, but shows how to do multiline string in YAML. The key is followed by a pipe character, and the mutli-line string is indented two more spaces.

`salt.states.file.`**`missing`**(*name*)

Verify that the named file or directory is missing, this returns True only if the named file is missing but does not remove the file if it is present.

name Absolute path which must NOT exist

`salt.states.file.`**`mknod`**(*name*, *ntype*, *major=0*, *minor=0*, *user=None*, *group=None*, *mode='0600'*)

Create a special file similar to the 'nix mknod command. The supported device types are `p` (fifo pipe), `c` (character device), and `b` (block device). Provide the major and minor numbers when specifying a character device or block device. A fifo pipe does not require this information. The command will create the necessary dirs if needed. If a file of the same name not of the same type/major/minor exists, it will not be overwritten or unlinked (deleted). This is logically in place as a safety measure because you can really shoot yourself in the foot here and it is the behavior of 'nix `mknod`. It is also important to note that not just anyone can create special devices. Usually this is only done as root. If the state is executed as none other than root on a minion, you may receive a permission error.

name name of the file

ntype node type 'p' (fifo pipe), 'c' (character device), or 'b' (block device)

major major number of the device does not apply to a fifo pipe

minor minor number of the device does not apply to a fifo pipe

user owning user of the device/pipe

group owning group of the device/pipe

mode permissions on the device/pipe

Usage:

```
 /dev/chr:
   file.mknod:
     - ntype: c
     - major: 180
     - minor: 31
     - user: root
     - group: root
     - mode: 660

 /dev/blk:
   file.mknod:
     - ntype: b
     - major: 8
     - minor: 999
     - user: root
     - group: root
     - mode: 660

/dev/fifo:
  file.mknod:
    - ntype: p
    - user: root
    - group: root
    - mode: 660
```

New in version 0.17.0.

`salt.states.file.`**`patch`**(*name*, *source=None*, *hash=None*, *options=''*, *dry_run_first=True*, *env=None*, ***kwargs*)

Apply a patch to a file. Note: a suitable `patch` executable must be available on the minion when using this state function.

name The file to with the patch will be applied.

source The source patch to download to the minion, this source file must be hosted on the salt master server. If the file is located in the directory named spam, and is called eggs, the source string is salt://spam/eggs. A source is required.

hash Hash of the patched file. If the hash of the target file matches this value then the patch is assumed to have been applied. The hash string is the hash algorithm followed by the hash of the file: md5=e138491e9d5b97023cea823fe17bac22

options Extra options to pass to patch.

dry_run_first [`True`] Run patch with `--dry-run` first to check if it will apply cleanly.

env Specify the environment from which to retrieve the patch file indicated by the `source` parameter. If not provided, this defaults to the environment from which the state is being executed.

Usage:

```
# Equivalent to ``patch --forward /opt/file.txt file.patch``
/opt/file.txt:
  file.patch:
    - source: salt://file.patch
    - hash: md5=e138491e9d5b97023cea823fe17bac22
```

`salt.states.file.`**`recurse`**(*name*, *source*, *clean=False*, *require=None*, *user=None*, *group=None*, *dir_mode=None*, *file_mode=None*, *sym_mode=None*, *template=None*, *context=None*, *defaults=None*, *env=None*, *include_empty=False*, *backup=''*, *include_pat=None*, *exclude_pat=None*, *maxdepth=None*, *keep_symlinks=False*, *force_symlinks=False*, ***kwargs*)

Recurse through a subdirectory on the master and copy said subdirectory over to the specified path.

name The directory to set the recursion in

source The source directory, this directory is located on the salt master file server and is specified with the salt:// protocol. If the directory is located on the master in the directory named spam, and is called eggs, the source string is salt://spam/eggs

clean Make sure that only files that are set up by salt and required by this function are kept. If this option is set then everything in this directory will be deleted unless it is required.

require Require other resources such as packages or files

user The user to own the directory. This defaults to the user salt is running as on the minion

group The group ownership set for the directory. This defaults to the group salt is running as on the minion

dir_mode The permissions mode to set on any directories created

file_mode The permissions mode to set on any files created

sym_mode The permissions mode to set on any symlink created

template If this setting is applied then the named templating engine will be used to render the downloaded file. Supported templates are: *jinja*, *mako* and *wempy*.

Note: The template option is required when recursively applying templates.

---

context Overrides default context variables passed to the template.

defaults Default context passed to the template.

include_empty Set this to True if empty directories should also be created (default is False)

include_pat When copying, include only this pattern from the source. Default is glob match; if prefixed with 'E@', then regexp match. Example:

```
- include_pat: hello*       :: glob matches 'hello01', 'hello02'
                               ... but not 'otherhello'
- include_pat: E@hello      :: regexp matches 'otherhello',
                               'hello01' ...
```

exclude_pat Exclude this pattern from the source when copying. If both *include_pat* and *exclude_pat* are supplied, then it will apply conditions cumulatively. i.e. first select based on include_pat, and then within that result apply exclude_pat.

Also, when 'clean=True', exclude this pattern from the removal list and preserve in the destination. Example:

```
- exclude_pat: APPDATA*               :: glob matches APPDATA.01,
                                         APPDATA.02,.. for exclusion
- exclude_pat: E@(APPDATA)|(TEMPDATA) :: regexp matches APPDATA
                                         or TEMPDATA for exclusion
```

maxdepth When copying, only copy paths which are of depth *maxdepth* from the source path. Example:

```
- maxdepth: 0      :: Only include files located in the source
                      directory
- maxdepth: 1      :: Only include files located in the source
                      or immediate subdirectories
```

keep_symlinks Keep symlinks when copying from the source. This option will cause the copy operation to terminate at the symlink. If desire behavior similar to rsync, then set this to True.

force_symlinks Force symlink creation. This option will force the symlink creation. If a file or directory is obstructing symlink creation it will be recursively removed so that symlink creation can proceed. This option is usually not needed except in special circumstances.

`salt.states.file.`**`rename`**(*name*, *source*, *force=False*, *makedirs=False*)

If the source file exists on the system, rename it to the named file. The named file will not be overwritten if it already exists unless the force option is set to True.

name The location of the file to rename to

source The location of the file to move to the location specified with name

force If the target location is present then the file will not be moved, specify "force: True" to overwrite the target file

makedirs If the target subdirectories don't exist create them `salt.states.file.`**`replace`**`(`*name*, *pattern*, *repl*, *count=0*, *flags=0*, *bufsize=1*, *backup='.bak'*, *show_changes=True*`)`

Maintain an edit in a file

New in version 0.17.0.

Params are identical to `replace()`.

`salt.states.file.`**`sed`**`(`*name*, *before*, *after*, *limit='‘*, *backup='.bak'*, *options='-r -e'*, *flags='g'*, *negate_match=False*`)`

Deprecated since version 0.17.0: Use `replace()` instead.

Maintain a simple edit to a file

The file will be searched for the `before` pattern before making the edit. In general the `limit` pattern should be as specific as possible and `before` and `after` should contain the minimal text to be changed.

before A pattern that should exist in the file before the edit.

after A pattern that should exist in the file after the edit.

limit An optional second pattern that can limit the scope of the before pattern.

backup [‘.bak’] The extension for the backed-up version of the file before the edit. If no backups is desired, pass in the empty string: ‘’

options [`-r -e`] Any options to pass to the `sed` command. `-r` uses extended regular expression syntax and `-e` denotes that what follows is an expression that sed will execute.

flags [`g`] Any flags to append to the sed expression. `g` specifies the edit should be made globally (and not stop after the first replacement).

negate_match [False] Negate the search command (`!`)

New in version 0.17.0.

Usage:

```
# Disable the epel repo by default
/etc/yum.repos.d/epel.repo:
  file.sed:
    - before: 1
    - after: 0
    - limit: ^enabled=

# Remove ldap from nsswitch
/etc/nsswitch.conf:
  file.sed:
    - before: 'ldap'
    - after: ''
    - limit: '^passwd:'
```

New in version 0.9.5.

`salt.states.file.`**`serialize`**`(`*name*, *dataset*, *user=None*, *group=None*, *mode=None*, *env=None*, *backup='‘*, *makedirs=False*, *show_diff=True*, *create=True*, ***kwargs*`)`

Serializes dataset and store it into managed file. Useful for sharing simple configuration files.

name The location of the symlink to create

dataset the dataset that will be serialized

formatter Write the data as this format. Supported output formats:

- JSON

- YAML
- Python (via pprint.pformat)

user The user to own the directory, this defaults to the user salt is running as on the minion

group The group ownership set for the directory, this defaults to the group salt is running as on the minion

mode The permissions to set on this file, aka 644, 0775, 4664

backup Overrides the default backup mode for this specific file.

makedirs Create parent directories for destination file.

New in version 2014.1.3.

show_diff If set to False, the diff will not be shown.

create Default is True, if create is set to False then the file will only be managed if the file already exists on the system.

For example, this state:

```
/etc/dummy/package.json:
  file.serialize:
    - dataset:
        name: naive
        description: A package using naive versioning
        author: A confused individual <iam@confused.com>
        dependencies:
            express: >= 1.2.0
            optimist: >= 0.1.0
        engine: node 0.4.1
    - formatter: json
```

will manages the file `/etc/dummy/package.json`:

```
{
  "author": "A confused individual <iam@confused.com>",
  "dependencies": {
    "express": ">= 1.2.0",
    "optimist": ">= 0.1.0"
  },
  "description": "A package using naive versioning",
  "engine": "node 0.4.1"
  "name": "naive",
}
```

`salt.states.file.`**`symlink`**(*name*, *target*, *force=False*, *backupname=None*, *makedirs=False*, *user=None*, *group=None*, *mode=None*, ***kwargs*)

Create a symlink

If the file already exists and is a symlink pointing to any location other than the specified target, the symlink will be replaced. If the symlink is a regular file or directory then the state will return False. If the regular file or directory is desired to be replaced with a symlink pass force: True, if it is to be renamed, pass a backupname.

name The location of the symlink to create

target The location that the symlink points to

force If the name of the symlink exists and is not a symlink and force is set to False, the state will fail. If force is set to True, the file or directory in the way of the symlink file will be deleted to make room for the symlink, unless backupname is set, when it will be renamed

backupname If the name of the symlink exists and is not a symlink, it will be renamed to the backupname. If the backupname already exists and force is False, the state will fail. Otherwise, the backupname will be removed first.

makedirs If the location of the symlink does not already have a parent directory then the state will fail, setting makedirs to True will allow Salt to create the parent directory

`salt.states.file.`**`touch`**(*name*, *atime=None*, *mtime=None*, *makedirs=False*)

Replicate the 'nix "touch" command to create a new empty file or update the atime and mtime of an existing file.

Note that if you just want to create a file and don't care about atime or mtime, you should use `file.managed` instead, as it is more feature-complete. (Just leave out the `source`/`template`/`contents` arguments, and it will just create the file and/or check its permissions, without messing with contents)

name name of the file

atime atime of the file

mtime mtime of the file

makedirs whether we should create the parent directory/directories in order to touch the file

Usage:

```
/var/log/httpd/logrotate.empty:
  file.touch
```

New in version 0.9.5.

`salt.states.file.`**`uncomment`**(*name*, *regex*, *char='#'*, *backup='.bak'*)

Uncomment specified commented lines in a file

name The full path to the file to be edited

regex A regular expression used to find the lines that are to be uncommented. This regex should not include the comment character. A leading ^ character will be stripped for convenience (for easily switching between comment() and uncomment()). The regex will be searched for from the beginning of the line, ignoring leading spaces (we prepend '^[t]*')

char [#] The character to remove in order to uncomment a line

backup [`.bak`] The file will be backed up before edit with this file extension; **WARNING:** each time `sed`/`comment`/`uncomment` is called will overwrite this backup

Usage:

```
/etc/adduser.conf:
  file.uncomment:
    - regex: EXTRA_GROUPS
```

New in version 0.9.5.

81.17 salt.states.gem

81.17.1 Installation of Ruby modules packaged as gems

A state module to manage rubygems. Gems can be set up to be installed or removed. This module will use RVM if it is installed. In that case, you can specify what ruby version and gemset to target.

```
addressable:
  gem.installed:
    - user: rvm
    - ruby: jruby@jgemset
```

`salt.states.gem.`**`installed`**(*name*, *ruby=None*, *runas=None*, *user=None*, *version=None*, *rdoc=False*, *ri=False*)

Make sure that a gem is installed.

name The name of the gem to install

ruby: None For RVM installations: the ruby version and gemset to target.

runas: None The user under which to run the `gem` command

Deprecated since version 0.17.0.

user: None The user under which to run the `gem` command

New in version 0.17.0.

version [None] Specify the version to install for the gem. Doesn't play nice with multiple gems at once

rdoc [False] Generate RDoc documentation for the gem(s).

ri [False] Generate RI documentation for the gem(s).

`salt.states.gem.`**`removed`**(*name*, *ruby=None*, *runas=None*, *user=None*)

Make sure that a gem is not installed.

name The name of the gem to uninstall

ruby: None For RVM installations: the ruby version and gemset to target.

runas: None The user under which to run the `gem` command

Deprecated since version 0.17.0.

user: None The user under which to run the `gem` command

New in version 0.17.0.

81.18 salt.states.git

81.18.1 Interaction with Git repositories

Important: Before using git over ssh, make sure your remote host fingerprint exists in "~/.ssh/known_hosts" file. To avoid requiring password authentication, it is also possible to pass private keys to use explicitly.

```
https://github.com/saltstack/salt.git:
  git.latest:
    - rev: develop
    - target: /tmp/salt
```

`salt.states.git.`**`latest`**(*name*, *rev=None*, *target=None*, *runas=None*, *user=None*, *force=None*, *force_checkout=False*, *submodules=False*, *mirror=False*, *bare=False*, *remote_name='origin'*, *always_fetch=False*, *identity=None*, *onlyif=False*, *unless=False*)

Make sure the repository is cloned to the given directory and is up to date

name Address of the remote repository as passed to "git clone"

rev The remote branch, tag, or revision ID to checkout after clone / before update

target Name of the target directory where repository is about to be cloned

runas Name of the user performing repository management operations

Deprecated since version 0.17.0.

user Name of the user performing repository management operations

New in version 0.17.0.

force Force git to clone into pre-existing directories (deletes contents)

force_checkout Force a checkout even if there might be overwritten changes (Default: False)

submodules Update submodules on clone or branch change (Default: False)

mirror True if the repository is to be a mirror of the remote repository. This implies bare, and thus is incompatible with rev.

bare True if the repository is to be a bare clone of the remote repository. This is incompatible with rev, as nothing will be checked out.

remote_name defines a different remote name. For the first clone the given name is set to the default remote, else it is just a additional remote. (Default: 'origin')

always_fetch If a tag or branch name is used as the rev a fetch will not occur until the tag or branch name changes. Setting this to true will force a fetch to occur. Only applies when rev is set. (Default: False)

identity A path to a private key to use over SSH

onlyif A command to run as a check, run the named command only if the command passed to the `onlyif` option returns true

unless A command to run as a check, only run the named command if the command passed to the `unless` option returns false

`salt.states.git.`**`present`**(*name*, *bare=True*, *runas=None*, *user=None*, *force=False*)

Make sure the repository is present in the given directory

name Name of the directory where the repository is about to be created

bare Create a bare repository (Default: True)

runas Name of the user performing repository management operations

Deprecated since version 0.17.0.

user Name of the user performing repository management operations

New in version 0.17.0.

force Force-create a new repository into an pre-existing non-git directory (deletes contents)

81.19 salt.states.gnomedesktop

81.19.1 Configuration of the GNOME desktop

Control the GNOME settings

```
localdesktop_wm_prefs:
    gnomedesktop.wm_preferences:
        - user: username
        - audible_bell: false
        - action_double_click_titlebar: 'toggle-maximize'
        - visual_bell: true
        - num_workspaces: 6
localdesktop_lockdown:
    gnomedesktop.desktop_lockdown:
        - user: username
        - disable_user_switching: true
localdesktop_interface:
    gnomedesktop.desktop_interface:
        - user: username
        - clock_show_date: true
        - clock_show_format: 12h
```

`salt.states.gnomedesktop.`**`desktop_interface`**(*name*, *user=None*, *automatic_mnemonics=None*, *buttons_have_icons=None*, *can_change_accels=None*, *clock_format=None*, *clock_show_date=None*, *clock_show_seconds=None*, *cursor_blink=None*, *cursor_blink_time=None*, *cursor_blink_timeout=None*, *cursor_size=None*, *cursor_theme=None*, *document_font_name=None*, *enable_animations=None*, *font_name=None*, *gtk_color_palette=None*, *gtk_color_scheme=None*, *gtk_im_module=None*, *gtk_im_preedit_style=None*, *gtk_im_status_style=None*, *gtk_key_theme=None*, *gtk_theme=None*, *gtk_timeout_initial=None*, *gtk_timeout_repeat=None*, *icon_theme=None*, *menubar_accel=None*, *menubar_detachable=None*, *menus_have_icons=None*, *menus_have_tearoff=None*, *monospace_font_name=None*, *show_input_method_menu=None*, *show_unicode_menu=None*, *text_scaling_factor=None*, *toolbar_detachable=None*, *toolbar_icons_size=None*, *toolbar_style=None*, *toolkit_accessibility=None*, ***kwargs*)

desktop_interface: sets values in the org.gnome.desktop.interface schema `salt.states.gnomedesktop.`**`desktop_lockdown`**(*name*, *user=None*, *disable_application_handlers=None*, *disable_command_line=None*, *disable_lock_screen=None*, *disable_log_out=None*, *disable_print_setup=None*, *disable_printing=None*, *disable_save_to_disk=None*, *disable_user_switching=None*, *user_administration_disabled=None*, ***kwargs*)

desktop_lockdown: sets values in the org.gnome.desktop.lockdown schema

`salt.states.gnomedesktop.`**`wm_preferences`**(*name*, *user=None*, *action_double_click_titlebar=None*, *action_middle_click_titlebar=None*, *action_right_click_titlebar=None*, *application_based=None*, *audible_bell=None*, *auto_raise=None*, *auto_raise_delay=None*, *button_layout=None*, *disable_workarounds=None*, *focus_mode=None*, *focus_new_windows=None*, *mouse_button_modifier=None*, *num_workspaces=None*, *raise_on_click=None*, *resize_with_right_button=None*, *theme=None*, *titlebar_font=None*, *titlebar_uses_system_font=None*, *visual_bell=None*, *visual_bell_type=None*, *workspace_names=None*, ***kwargs*)

wm_preferences: sets values in the org.gnome.desktop.wm.preferences schema

81.20 salt.states.grains

81.20.1 Manage grains on the minion

This state allows for grains to be set. Grains set or altered this way are stored in the 'grains' file on the minions, by default at: /etc/salt/grains

Note: This does NOT override any grains set in the minion file.

`salt.states.grains.`**`list_absent`**(*name*, *value*)

Ensure the value is absent in the list type grain

name The grain name

value The value is absent in the list type grain

The grain should be list type .

```
roles:
  grains.list_absent:
    - value: db
```

`salt.states.grains.`**`list_present`**(*name*, *value*)

New in version 2014.1.0: (Hydrogen)

Ensure the value is present in the list type grain

name The grain name

value The value is present in the list type grain

The grain should be list type .

```
roles:
  grains.list_present:
    - value: web
```

`salt.states.grains.`**`present`**(*name*, *value*)

Ensure that a grain is set

name The grain name

value The value to set on the grain

If the grain with the given name exists, its value is updated to the new value. If the grain does not yet exist, a new grain is set to the given value.

```
cheese:
  grains.present:
    - value: edam
```

81.21 salt.states.group

81.21.1 Management of user groups

The group module is used to create and manage unix group settings, groups can be either present or absent:

```
cheese:
  group.present:
    - gid: 7648
    - system: True
```

`salt.states.group.`**`absent`**(*name*)

Ensure that the named group is absent

name The name of the group to remove

`salt.states.group.`**`present`**(*name*, *gid=None*, *system=False*)

Ensure that a group is present

name The name of the group to manage

gid The group id to assign to the named group; if left empty, then the next available group id will be assigned

system Whether or not the named group is a system group. This is essentially the '-r' option of 'groupadd'.

81.22 salt.states.hg

81.22.1 Interaction with Mercurial repositories

Before using hg over ssh, make sure the remote host fingerprint already exists in ~/.ssh/known_hosts, and the remote host has this host's public key.

```
https://bitbucket.org/example_user/example_repo:
    hg.latest:
      - rev: tip
      - target: /tmp/example_repo
```

`salt.states.hg.`**`latest`**(*name*, *rev=None*, *target=None*, *clean=False*, *runas=None*, *user=None*, *force=False*, *opts=False*)

Make sure the repository is cloned to the given directory and is up to date

name Address of the remote repository as passed to "hg clone"

rev The remote branch, tag, or revision hash to clone/pull

target Name of the target directory where repository is about to be cloned

clean Force a clean update with -C (Default: False)

runas Name of the user performing repository management operations

Deprecated since version 0.17.0.

user Name of the user performing repository management operations

force Force hg to clone into pre-existing directories (deletes contents)

opts Include additional arguments and options to the hg command line

81.23 salt.states.host

81.23.1 Management of addresses and names in hosts file

The /etc/hosts file can be managed to contain definitions for specific hosts:

```
salt-master:
  host.present:
    - ip: 192.168.0.42
```

Or using the "names:" directive, you can put several names for the same IP. (Do not try one name with space-separated values).

```
server1:
  host.present:
    - ip: 192.168.0.42
    - names:
      - server1
      - florida
```

NOTE: changing the name(s) in the present() function does not cause an update to remove the old entry.

`salt.states.host.`**`absent`**(*name*, *ip*)

Ensure that the named host is absent

name The host to remove

ip The ip addr of the host to remove

`salt.states.host.`**`present`**(*name*, *ip*)

Ensures that the named host is present with the given ip

name The host to assign an ip to

ip The ip addr to apply to the host

81.24 salt.states.iptables

81.24.1 Management of iptables

This is an iptables-specific module designed to manage Linux firewalls. It is expected that this state module, and other system-specific firewall states, may at some point be deprecated in favor of a more generic *firewall* state.

```
httpd:
  iptables.append:
    - table: filter
    - chain: INPUT
    - jump: ACCEPT
    - match: state
    - connstate: NEW
    - dport: 80
    - proto: tcp
    - sport: 1025:65535
    - save: True

httpd:
  iptables.append:
    - table: filter
    - family: ipv6
    - chain: INPUT
    - jump: ACCEPT
    - match: state
    - connstate: NEW
    - dport: 80
    - proto: tcp
    - sport: 1025:65535
    - save: True

httpd:
  iptables.insert:
    - position: 1
    - table: filter
    - chain: INPUT
    - jump: ACCEPT
    - match: state
    - connstate: NEW
    - dport: 80
    - proto: tcp
    - sport: 1025:65535
    - save: True

httpd:
  iptables.insert:
    - position: 1
    - table: filter
    - family: ipv6
    - chain: INPUT
    - jump: ACCEPT
    - match: state
    - connstate: NEW
    - dport: 80
    - proto: tcp
    - sport: 1025:65535
    - save: True

httpd:
  iptables.delete:
    - table: filter
    - chain: INPUT
    - jump: ACCEPT
    - match: state
    - connstate: NEW
    - dport: 80
    - proto: tcp
    - sport: 1025:65535
    - save: True

httpd:
  iptables.delete:
    - position: 1
    - table: filter
    - chain: INPUT
    - jump: ACCEPT
    - match: state
    - connstate: NEW
    - dport: 80
    - proto: tcp
    - sport: 1025:65535
    - save: True

httpd:
  iptables.delete:
    - table: filter
    - family: ipv6
    - chain: INPUT
    - jump: ACCEPT
    - match: state
    - connstate: NEW
    - dport: 80
    - proto: tcp
    - sport: 1025:65535
    - save: True
```

`salt.states.iptables.`**`append`**(*name*, *family='ipv4'*, ***kwargs*)

New in version 0.17.0.

Append a rule to a chain

name A user-defined name to call this rule by in another part of a state or formula. This should not be an actual rule.

family Network family, ipv4 or ipv6.

All other arguments are passed in with the same name as the long option that would normally be used for iptables, with one exception: *–state* is specified as *connstate* instead of *state* (not to be confused with *ctstate*).

`salt.states.iptables.`**`chain_absent`**(*name*, *table='filter'*, *family='ipv4'*)

New in version 2014.1.0: (Hydrogen)

Verify the chain is absent.

family Networking family, either ipv4 or ipv6 `salt.states.iptables.`**`chain_present`**(*name*, *table='filter'*, *family='ipv4'*)

New in version 2014.1.0: (Hydrogen)

Verify the chain is exist.

name A user-defined chain name.

table The table to own the chain.

family Networking family, either ipv4 or ipv6

`salt.states.iptables.`**`delete`**(*name*, *family='ipv4'*, ***kwargs*)

New in version 2014.1.0: (Hydrogen)

Delete a rule to a chain

name A user-defined name to call this rule by in another part of a state or formula. This should not be an actual rule.

family Networking family, either ipv4 or ipv6

All other arguments are passed in with the same name as the long option that would normally be used for iptables, with one exception: *–state* is specified as *connstate* instead of *state* (not to be confused with *ctstate*).

`salt.states.iptables.`**`flush`**(*name*, *family='ipv4'*, ***kwargs*)

New in version 2014.1.0: (Hydrogen)

Flush current iptables state

family Networking family, either ipv4 or ipv6

`salt.states.iptables.`**`insert`**(*name*, *family='ipv4'*, ***kwargs*)

New in version 2014.1.0: (Hydrogen)

Insert a rule into a chain

name A user-defined name to call this rule by in another part of a state or formula. This should not be an actual rule.

family Networking family, either ipv4 or ipv6

All other arguments are passed in with the same name as the long option that would normally be used for iptables, with one exception: *–state* is specified as *connstate* instead of *state* (not to be confused with *ctstate*).

`salt.states.iptables.`**`set_policy`**(*name*, *family='ipv4'*, ***kwargs*)

New in version 2014.1.0: (Hydrogen)

Sets the default policy for iptables firewall tables

family Networking family, either ipv4 or ipv6

81.25 salt.states.keyboard

81.25.1 Management of keyboard layouts

The keyboard layout can be managed for the system:

```
us:
  keyboard.system
```

Or it can be managed for XOrg:

```
us:
  keyboard.xorg
```

`salt.states.keyboard.`**`system`**`(`*name*`)`
Set the keyboard layout for the system

name The keyboard layout to use

`salt.states.keyboard.`**`xorg`**`(`*name*`)`
Set the keyboard layout for XOrg

layout The keyboard layout to use

81.26 salt.states.keystone

81.26.1 Management of Keystone users

depends

- keystoneclient Python module

configuration See `salt.modules.keystone` for setup instructions.

```
Keystone tenants:
  keystone.tenant_present:
    - names:
      - admin
      - demo
      - service

Keystone roles:
  keystone.role_present:
    - names:
      - admin
      - Member

admin:
  keystone.user_present:
    - password: R00T_4CC3SS
    - email: admin@domain.com
    - roles:
      - admin:   # tenants
        - admin  # roles
      - service:
        - admin
        - Member
    - require:
      - keystone: Keystone tenants
      - keystone: Keystone roles

nova:
  keystone.user_present:
    - password: '$up3rn0v4'
    - email: nova@domain.com
    - tenant: service
    - roles:
      - service:
        - admin
    - require:
      - keystone: Keystone tenants
      - keystone: Keystone roles

demo:
  keystone.user_present:
    - password: 'd3m0n$trati0n'
    - email: demo@domain.com
    - tenant: demo
    - roles:
      - demo:
        - Member
    - require:
      - keystone: Keystone tenants
      - keystone: Keystone roles

nova service:
  keystone.service_present:
    - name: nova
    - service_type: compute
    - description: OpenStack Compute Service
```

`salt.states.keystone.`**`role_absent`**(*name*, *profile=None*, ***connection_args*)

Ensure that the keystone role is absent.

name The name of the role that should not exist

`salt.states.keystone.`**`role_present`**(*name*, *profile=None*, ***connection_args*)

‘ Ensures that the keystone role exists

name The name of the role that should be present

`salt.states.keystone.`**`service_absent`**(*name*, *profile=None*, ***connection_args*)

Ensure that the service doesn’t exist in Keystone catalog

name The name of the service that should not exist

`salt.states.keystone.`**`service_present`**(*name*, *service_type*, *description=None*, *profile=None*, ***connection_args*)

Ensure service present in Keystone catalog

name The name of the service

service_type The type of Openstack Service

description (optional) Description of the service

`salt.states.keystone.`**`tenant_absent`**(*name*, *profile=None*, ***connection_args*)

Ensure that the keystone tenant is absent.

name The name of the tenant that should not exist

`salt.states.keystone.`**`tenant_present`**(*name*, *description=None*, *enabled=True*, *profile=None*, ***connection_args*)

‘ Ensures that the keystone tenant exists

name The name of the tenant to manage

description The description to use for this tenant

enabled Availability state for this tenant

`salt.states.keystone.`**`user_absent`**(*name*, *profile=None*, ***connection_args*)

Ensure that the keystone user is absent.

name The name of the user that should not exist

`salt.states.keystone.`**`user_present`**(*name*, *password*, *email*, *tenant=None*, *enabled=True*, *roles=None*, *profile=None*, ***connection_args*)

Ensure that the keystone user is present with the specified properties.

name The name of the user to manage

password The password to use for this user

email The email address for this user

tenant The tenant for this user

enabled Availability state for this user

roles The roles the user should have under tenants

81.27 salt.states.kmod

81.27.1 Loading and unloading of kernel modules

The Kernel modules on a system can be managed cleanly with the kmod state module:

```
kvm_amd:
  kmod.present
pcspkr:
  kmod.absent
```

`salt.states.kmod.`**`absent`**(*name*, *persist=False*, *comment=True*)

Verify that the named kernel module is not loaded

name The name of the kernel module to verify is not loaded

persist Delete module from /etc/modules

comment Don't remove module from /etc/modules, only comment it

`salt.states.kmod.`**`present`**(*name*, *persist=False*)

Ensure that the specified kernel module is loaded

name The name of the kernel module to verify is loaded

persist Also add module to /etc/modules

81.28 salt.states.layman

81.28.1 Management of Gentoo Overlays using layman

A state module to manage Gentoo package overlays via layman

```
sunrise:
    layman.present
```

`salt.states.layman.`**`absent`**(*name*)

Verify that the overlay is absent

name The name of the overlay to delete `salt.states.layman.`**`present`**`(`*name*`)`

Verify that the overlay is present

name The name of the overlay to add

81.29 salt.states.libvirt

81.29.1 Manage libvirt certificates

This state uses the external pillar in the master to call for the generation and signing of certificates for systems running libvirt:

```
libvirt_keys:
  libvirt.keys
```

`salt.states.libvirt.`**`keys`**`(`*name*, *basepath='/etc/pki'*`)`

Manage libvirt keys.

name The name variable used to track the execution

basepath Defaults to */etc/pki*, this is the root location used for libvirt keys on the hypervisor

81.30 salt.states.locale

Management of languages/locales ==============================+

The locale can be managed for the system:

```
en_US.UTF-8:
  locale.system
```

`salt.states.locale.`**`system`**`(`*name*`)`

Set the locale for the system

name The name of the locale to use

81.31 salt.states.lvm

81.31.1 Management of Linux logical volumes

A state module to manage LVMs

```
/dev/sda:
  lvm.pv_present

my_vg:
  lvm.vg_present:
    - devices: /dev/sda

lvroot:
  lvm.lv_present:
    - vgname: my_vg
    - size: 10G
```

`salt.states.lvm.`**`lv_absent`**(*name*, *vgname=None*)
Remove a given existing logical volume from a named existing volume group

name The logical volume to remove

vgname The volume group name

`salt.states.lvm.`**`lv_present`**(*name*, *vgname=None*, *size=None*, *extents=None*, *pv=''*)
Create a new logical volume

name The name of the logical volume

vgname The volume group name for this logical volume

size The initial size of the logical volume

extents The number of logical extents to allocate

pv The physical volume to use

`salt.states.lvm.`**`pv_present`**(*name*, ***kwargs*)
Set a physical device to be used as an LVM physical volume

name The device name to initialize.

kwargs Any supported options to pvcreate. See `linux_lvm` for more details.

`salt.states.lvm.`**`vg_absent`**(*name*)
Remove an LVM volume group

name The volume group to remove

`salt.states.lvm.`**`vg_present`**(*name*, *devices=None*, ***kwargs*)
Create an LVM volume group

name The volume group name to create

devices A list of devices that will be added to the volume group

kwargs Any supported options to vgcreate. See `linux_lvm` for more details.

81.32 salt.states.lvs_server

81.32.1 Management of LVS(Linux Virtual Server) Real Server.

This lvs_server module is used to add and manage LVS Real Server in the specified service. Server can be set as either absent or present.

`salt.states.lvs_server.`**`absent`**(*name*, *protocol=None*, *service_address=None*, *server_address=None*)
Ensure the LVS Real Server in specified service is absent.

name The name of the LVS server.

protocol The service protocol(only support `tcp`, `udp` and `fwmark` service).

service_address The LVS service adress.

server_address The LVS real server address.

`salt.states.lvs_server.`**`present`**(*name*, *protocol=None*, *service_address=None*, *server_address=None*, *packet_forward_method='dr'*, *weight=1*)
Ensure that the named service is present.

name The LVS server name

protocol The service protocol

service_address The LVS service address

server_address The real server address.

packet_forward_method The LVS packet forwarding method(`dr` for direct routing, `tunnel` for tunneling, `nat` for network access translation).

weight The capacity of a server relative to the others in the pool.

```
lvsrs:
  lvs_server.present:
    - protocol: tcp
    - service_address: 1.1.1.1:80
    - server_address: 192.168.0.11:8080
    - packet_forward_method: dr
    - weight: 10
```

81.33 salt.states.lvs_service

81.33.1 Management of LVS(Linux Virtual Server) Service.

This lvs_service module is used to create and manage LVS Service. Service can be set as either absent or present.

`salt.states.lvs_service.`**`absent`**(*name*, *protocol=None*, *service_address=None*)

Ensure the LVS service is absent.

name The name of the LVS service

protocol The service protocol

service_address The LVS service adress

`salt.states.lvs_service.`**`present`**(*name*, *protocol=None*, *service_address=None*, *scheduler='wlc'*)

Ensure that the named service is present.

name The LVS service name

protocol The service protocol

service_address The LVS service address

scheduler Algorithm for allocating TCP connections and UDP datagrams to real servers.

```
lvstest:
  lvs_service.present:
    - service_address: 1.1.1.1:80
    - protocol: tcp
    - scheduler: rr
```

81.34 salt.states.makeconf

81.34.1 Management of Gentoo make.conf

A state module to manage Gentoo's make.conf file

```
makeopts:
    makeconf.present:
        - value: '-j3'
```

`salt.states.makeconf.`**`absent`**(*name*)

Verify that the variable is not in the make.conf.

name The variable name. This will automatically be converted to all Upper Case since variables in make.conf are Upper Case

`salt.states.makeconf.`**`present`**(*name*, *value=None*, *contains=None*, *excludes=None*)

Verify that the variable is in the make.conf and has the provided settings. If value is set, contains and excludes will be ignored.

name The variable name. This will automatically be converted to all Upper Case since variables in make.conf are Upper Case

value Enforce that the value of the variable is set to the provided value

contains Enforce that the value of the variable contains the provided value

excludes Enforce that the value of the variable does not contain the provided value.

81.35 salt.states.mdadm

81.35.1 Managing software RAID with mdadm

A state module for creating or destroying software RAID devices.

```
/dev/md0:
  raid.present:
    - opts: level=1 chunk=256 raid-devices=2 /dev/xvdd /dev/xvde
```

`salt.states.mdadm.`**`absent`**(*name*)

Verify that the raid is absent

name The name of raid device to be destroyed

```
/dev/md0:
  raid:
    - absent
```

`salt.states.mdadm.`**`present`**(*name*, *opts=None*)

Verify that the raid is present

name The name of raid device to be created

opts The mdadm options to use to create the raid. See `mdadm` for more information. Opts can be expressed as a single string of options.

```
/dev/md0:
  raid.present:
    - opts: level=1 chunk=256 raid-devices=2 /dev/xvdd /dev/xvde
```

Or as a list of options.

```
/dev/md0:
  raid.present:
    - opts:
```

```
              - level=1
              - chunk=256
              - raid-devices=2
              - /dev/xvdd
              - /dev/xvde
```

81.36 salt.states.memcached

81.36.1 States for Management of Memcached Keys

New in version 2014.1.0: (Hydrogen)

`salt.states.memcached.`**`absent`**(*name*, *value=None*, *host='127.0.0.1'*, *port=11211*, *time=0*)

Ensure that a memcached key is not present.

name The key

value [None] If specified, only ensure that the key is absent if it matches the specified value.

host The memcached server IP address

port The memcached server port

```
foo:
  memcached.absent

bar:
  memcached.absent:
    - host: 10.0.0.1
```

`salt.states.memcached.`**`managed`**(*name*, *value=None*, *host='127.0.0.1'*, *port=11211*, *time=0*, *min_compress_len=0*)

Manage a memcached key.

name The key to manage

value The value to set for that key

host The memcached server IP address

port The memcached server port

```
foo:
  memcached.managed:
    - value: bar
```

81.37 salt.states.modjk_worker

81.37.1 Manage modjk workers

Send commands to a **modjk** load balancer via the peer system.

This module can be used with the *prereq* requisite to remove/add the worker from the load balancer before deploying/restarting service.

Mandatory Settings:

- The minion needs to have permission to publish the **modjk.*** functions (see *here* for information on configuring peer publishing permissions)
- The modjk load balancer must be configured as stated in the **modjk** execution module `documentation`

`salt.states.modjk_worker.`**`activate`**(*name*, *lbn*, *target*, *profile='default'*, *expr_form='glob'*)

Activate the named worker from the lbn load balancers at the targeted minions

Example:

```
disable-before-deploy:
  modjk_worker.activate:
    - name: {{ grains['id'] }}
    - lbn: application
    - target: 'roles:balancer'
    - expr_form: grain
```

`salt.states.modjk_worker.`**`disable`**(*name*, *lbn*, *target*, *profile='default'*, *expr_form='glob'*)

Disable the named worker from the lbn load balancers at the targeted minions. The worker will get traffic only for current sessions and won't get new ones.

Example:

```
disable-before-deploy:
  modjk_worker.disable:
    - name: {{ grains['id'] }}
    - lbn: application
    - target: 'roles:balancer'
    - expr_form: grain
```

`salt.states.modjk_worker.`**`stop`**(*name*, *lbn*, *target*, *profile='default'*, *expr_form='glob'*)

Stop the named worker from the lbn load balancers at the targeted minions The worker won't get any traffic from the lbn

Example:

```
disable-before-deploy:
  modjk_worker.stop:
    - name: {{ grains['id'] }}
    - lbn: application
    - target: 'roles:balancer'
    - expr_form: grain
```

81.38 salt.states.module

81.38.1 Execution of Salt modules from within states

These states allow individual execution module calls to be made via states. To call a single module function use a `module.run` state:

```
mine.send:
  module.run:
    - func: network.interfaces
```

Note that this example is probably unnecessary to use in practice, since the `mine_functions` and `mine_interval` config parameters can be used to schedule updates for the mine (see *here* for more info).

It is sometimes desirable to trigger a function call after a state is executed, for this the `module.wait` state can be used:

```
mine.send:
  module.wait:
    - func: network.interfaces
    - watch:
      - file: /etc/network/interfaces
```

All arguments are passed through to the module function being executed. However, due to how the state system works, if a module function accepts an argument called, `name`, then `m_name` must be used to specify that argument, to avoid a collision with the `name` argument. For example:

```
disable_nfs:
  module.run:
    - name: service.disable
    - m_name: nfs
```

`salt.states.module.`**`mod_watch`**`(`*name*, ***kwargs*`)`

Run a single module function

`name` The module function to execute

`returner` Specify the returner to send the return of the module execution to

`**kwargs` Pass any arguments needed to execute the function

`salt.states.module.`**`run`**`(`*name*, ***kwargs*`)`

Run a single module function

`name` The module function to execute

`returner` Specify the returner to send the return of the module execution to

`**kwargs` Pass any arguments needed to execute the function

`salt.states.module.`**`wait`**`(`*name*, ***kwargs*`)`

Run a single module function only if the watch statement calls it

`name` The module function to execute

`**kwargs` Pass any arguments needed to execute the function

Note: Like the `cmd.run` state, this state will return `True` but not actually execute, unless one of the following two things happens:

1. The state has a *watch requisite*, and the state which it is watching changes.
2. Another state has a *watch_in requisite* which references this state, and the state wth the `watch_in` changes.

81.39 salt.states.mongodb_database

Management of Mongodb databases

Only deletion is supported, creation doesn't make sense and can be done using mongodb_user.present

`salt.states.mongodb_database.`**`absent`**`(`*name*, *user=None*, *password=None*, *host=None*, *port=None*`)`

Ensure that the named database is absent

name The name of the database to remove

user The user to connect as (must be able to create the user)

password The password of the user

host The host to connect to

port The port to connect to

81.40 salt.states.mongodb_user

81.40.1 Management of Mongodb users

`salt.states.mongodb_user.`**`absent`**(*name*, *user=None*, *password=None*, *host=None*, *port=None*, *database='admin'*)

Ensure that the named user is absent

name The name of the user to remove

user The user to connect as (must be able to create the user)

password The password of the user

host The host to connect to

port The port to connect to

database The database to create the user in (if the db doesn't exist, it will be created)

`salt.states.mongodb_user.`**`present`**(*name*, *passwd*, *database='admin'*, *user=None*, *password=None*, *host=None*, *port=None*)

Ensure that the user is present with the specified properties

name The name of the user to manage

passwd The password of the user

user The user to connect as (must be able to create the user)

password The password of the user

host The host to connect to

port The port to connect to

database The database to create the user in (if the db doesn't exist, it will be created)

81.41 salt.states.mount

81.41.1 Mounting of filesystems

Mount any type of mountable filesystem with the mounted function:

```
/mnt/sdb:
  mount.mounted:
    - device: /dev/sdb1
    - fstype: ext4
    - mkmnt: True
    - opts:
      - defaults
```

`salt.states.mount.`**`mounted`**(*name*, *device*, *fstype*, *mkmnt=False*, *opts=None*, *dump=0*, *pass_num=0*, *config='/etc/fstab'*, *persist=True*)

Verify that a device is mounted

name The path to the location where the device is to be mounted

device The device name, typically the device node, such as /dev/sdb1

fstype The filesystem type, this will be xfs, ext2/3/4 in the case of classic filesystems, and fuse in the case of fuse mounts

mkmnt If the mount point is not present then the state will fail, set mkmnt to True to create the mount point if it is otherwise not present

opts A list object of options or a comma delimited list

dump The dump value to be passed into the fstab, default to 0

pass_num The pass value to be passed into the fstab, default to 0

config Set an alternative location for the fstab, default to /etc/fstab

remount Set if the file system can be remounted with the remount option, default to True

persist Set if the mount should be saved in the fstab, default to True

`salt.states.mount.`**`swap`**(*name*, *persist=True*, *config='/etc/fstab'*)

Activates a swap device

```
/root/swapfile:
  mount.swap
```

`salt.states.mount.`**`unmounted`**(*name*, *config='/etc/fstab'*, *persist=False*)

Note: This state will be available in version 0.17.0.

Verify that a device is not mounted

name The path to the location where the device is to be unmounted from

config Set an alternative location for the fstab, default to /etc/fstab

persist Set if the mount should be purged from the fstab, default to False

81.42 salt.states.mysql_database

81.42.1 Management of MySQL databases (schemas)

depends

- MySQLdb Python module

configuration See `salt.modules.mysql` for setup instructions.

The mysql_database module is used to create and manage MySQL databases. Databases can be set as either absent or present.

```
frank:
  mysql_database.present
```

`salt.states.mysql_database.`**`absent`**`(`*name*, ***connection_args*`)`

Ensure that the named database is absent

name The name of the database to remove

`salt.states.mysql_database.`**`present`**`(`*name*, ***connection_args*`)`

Ensure that the named database is present with the specified properties

name The name of the database to manage

81.43 salt.states.mysql_grants

81.43.1 Management of MySQL grants (user permissions)

depends

- MySQLdb Python module

configuration See `salt.modules.mysql` for setup instructions.

The mysql_grants module is used to grant and revoke MySQL permissions.

The `name` you pass in purely symbolic and does not have anything to do with the grant itself.

The `database` parameter needs to specify a 'priv_level' in the same specification as defined in the MySQL documentation:

- *
- *.*
- db_name.*
- db_name.tbl_name
- etc...

```
frank_exampledb:
   mysql_grants.present:
    - grant: select,insert,update
    - database: exampledb.*
    - user: frank
    - host: localhost

frank_otherdb:
  mysql_grants.present:
    - grant: all privileges
    - database: otherdb.*
    - user: frank

restricted_singletable:
  mysql_grants.present:
    - grant: select
    - database: somedb.sometable
    - user: joe
```

`salt.states.mysql_grants.`**`absent`**`(`*name*, *grant=None*, *database=None*, *user=None*, *host='localhost'*, *grant_option=False*, *escape=True*, ***connection_args*`)`

Ensure that the grant is absent

name The name (key) of the grant to add

grant The grant priv_type (i.e. select,insert,update OR all privileges)

database The database priv_level (i.e. db.tbl OR db.*)

user The user to apply the grant to

host The network/host that the grant should apply to

`salt.states.mysql_grants.`**`present`**(*name*, *grant=None*, *database=None*, *user=None*, *host='localhost'*, *grant_option=False*, *escape=True*, *revoke_first=False*, ***connection_args*)

Ensure that the grant is present with the specified properties

name The name (key) of the grant to add

grant The grant priv_type (i.e. select,insert,update OR all privileges)

database The database priv_level (ie. db.tbl OR db.*)

user The user to apply the grant to

host The network/host that the grant should apply to

grant_option Adds the WITH GRANT OPTION to the defined grant. default: False

escape Defines if the database value gets escaped or not. default: True

revoke_first By default, MySQL will not do anything if you issue a command to grant privileges that are more restrictive than what's already in place. This effectively means that you cannot downgrade permissions without first revoking permissions applied to a db.table/user pair first.

To have Salt forcibly revoke perms before applying a new grant, enable the 'revoke_first options.

WARNING: This will *remove* permissions for a database before attempting to apply new permissions. There is no guarantee that new permissions will be applied correctly which can leave your database security in an unknown and potentially dangerous state. Use with caution!

default: False

81.44 salt.states.mysql_user

81.44.1 Management of MySQL users

depends

- MySQLdb Python module

configuration See `salt.modules.mysql` for setup instructions.

```
frank:
  mysql_user.present:
    - host: localhost
    - password: bobcat
```

New in version 0.16.2: Authentication overrides have been added.

The MySQL authentication information specified in the minion config file can be overidden in states using the following arguments: `connection_host`, `connection_port`, `connection_user`, `connection_pass`, `connection_db`, `connection_unix_socket`, `connection_default_file` and `connection_charset`.

```
frank:
  mysql_user.present:
    - host: localhost
    - password: "bob@cat"
    - connection_user: someuser
    - connection_pass: somepass
    - connection_charset: utf8
    - saltenv:
      - LC_ALL: "en_US.utf8"
```

`salt.states.mysql_user.`**`absent`**(*name*, *host='localhost'*, ***connection_args*)

Ensure that the named user is absent

name The name of the user to remove

`salt.states.mysql_user.`**`present`**(*name*, *host='localhost'*, *password=None*, *password_hash=None*, *allow_passwordless=False*, *unix_socket=False*, ***connection_args*)

Ensure that the named user is present with the specified properties. A passwordless user can be configured by omitting `password` and `password_hash`, and setting `allow_passwordless` to `True`.

name The name of the user to manage

host Host for which this user/password combo applies

password The password to use for this user. Will take precedence over the `password_hash` option if both are specified.

password_hash The password in hashed form. Be sure to quote the password because YAML doesn't like the `*`. A password hash can be obtained from the mysql command-line client like so:

```
mysql> SELECT PASSWORD('mypass');
+-------------------------------------------+
| PASSWORD('mypass')                        |
+-------------------------------------------+
| *6C8989366EAF75BB670AD8EA7A7FC1176A95CEF4 |
+-------------------------------------------+
1 row in set (0.00 sec)
```

allow_passwordless If `True`, then `password` and `password_hash` can be omitted to permit a passwordless login.

unix_socket If `True` and allow_passwordless is `True` then will be used unix_socket auth plugin.

Note: The `allow_passwordless` option will be available in version 0.16.2.

81.45 salt.states.network

81.45.1 Configuration of network interfaces

The network module is used to create and manage network settings, interfaces can be set as either managed or ignored. By default all interfaces are ignored unless specified.

Please note that only Redhat and Debian style networking is currently supported. This module will therefore only work on RH/CentOS/Fedora and Debian/Ubuntu.

```
system:
    network.system:
      - enabled: True
      - hostname: server1.example.com
      - gateway: 192.168.0.1
      - gatewaydev: eth0
      - nozeroconf: True
      - nisdomain: example.com
      - require_reboot: True

eth0:
  network.managed:
    - enabled: True
    - type: eth
    - proto: none
    - ipaddr: 10.1.0.1
    - netmask: 255.255.255.0
    - dns:
      - 8.8.8.8
      - 8.8.4.4

routes:
  network.routes:
    - name: eth0
    - routes:
      - name: secure_network
        ipaddr: 10.2.0.0
        netmask: 255.255.255.0
        gateway: 10.1.0.3
      - name: HQ_network
        ipaddr: 10.100.0.0
        netmask: 255.255.0.0
        gateway: 10.1.0.10

eth2:
  network.managed:
    - type: slave
    - master: bond0

eth3:
  network.managed:
    - type: slave
    - master: bond0

eth4:
  network.managed:
    - enabled: True
    - type: eth
    - proto: dhcp
    - bridge: br0

bond0:
  network.managed:
    - type: bond
    - ipaddr: 10.1.0.1
    - netmask: 255.255.255.0
    - dns:
      - 8.8.8.8
      - 8.8.4.4
    - ipv6:
    - enabled: False
    - use_in:
      - network: eth2
      - network: eth3
    - require:
      - network: eth2
      - network: eth3
    - mode: 802.3ad
    - miimon: 100
    - arp_interval: 250
    - downdelay: 200
    - lacp_rate: fast
    - max_bonds: 1
    - updelay: 0
    - use_carrier: on
    - xmit_hash_policy: layer2
    - mtu: 9000
    - autoneg: on
    - speed: 1000
    - duplex: full
    - rx: on
    - tx: off
    - sg: on
    - tso: off
    - ufo: off
    - gso: off
    - gro: off
    - lro: off

bond0.2:
  network.managed:
    - type: vlan
    - ipaddr: 10.1.0.2
    - use:
      - network: bond0
    - require:
      - network: bond0

bond0.3:
  network.managed:
    - type: vlan
    - ipaddr: 10.1.0.3
    - use:
      - network: bond0
    - require:
      - network: bond0

bond0.10:
  network.managed:
    - type: vlan
    - ipaddr: 10.1.0.4
    - use:
      - network: bond0
    - require:
      - network: bond0
```

```
bond0.12:
  network.managed:
    - type: vlan
    - ipaddr: 10.1.0.5
    - use:
      - network: bond0
    - require:
      - network: bond0
br0:
  network.managed:
    - enabled: True
    - type: bridge
    - proto: dhcp
    - bridge: br0
    - delay: 0
    - bypassfirewall: True
    - use:
      - network: eth4
    - require:
      - network: eth4
```

`salt.states.network.`**`managed`**`(`*name*, *type*, *enabled=True*, ***kwargs*`)`

Ensure that the named interface is configured properly.

name The name of the interface to manage

type Type of interface and configuration.

enabled Designates the state of this interface.

kwargs The IP parameters for this interface.

`salt.states.network.`**`routes`**`(`*name*, ***kwargs*`)`

Manage network interface static routes.

name Interface name to apply the route to.

kwargs Named routes

`salt.states.network.`**`system`**`(`*name*, ***kwargs*`)`

Ensure that global network settings are configured properly.

name Custom name to represent this configuration change.

kwargs The global parameters for the system.

81.46 salt.states.npm

81.46.1 Installation of NPM Packages

These states manage the installed packages for node.js using the Node Package Manager (npm). Note that npm must be installed for these states to be available, so npm states should include a requisite to a pkg.installed state for the package which provides npm (simply `npm` in most cases). Example:

```
npm:
  pkg.installed

yaml:
  npm.installed:
    - require:
      - pkg: npm
```

`salt.states.npm.`**`bootstrap`**`(`*name*, *runas=None*, *user=None*`)`

Bootstraps a node.js application.

will execute npm install –json on the specified directory

runas The user to run NPM with

Deprecated since version 0.17.0.

user The user to run NPM with

New in version 0.17.0.

`salt.states.npm.`**`installed`**`(`*name*, *dir=None*, *runas=None*, *user=None*, *force_reinstall=False*, ***kwargs*`)`

Verify that the given package is installed and is at the correct version (if specified).

dir The target directory in which to install the package, or None for global installation

runas The user to run NPM with

Deprecated since version 0.17.0.

user The user to run NPM with

New in version 0.17.0.

force_reinstall Install the package even if it is already installed

`salt.states.npm.`**`removed`**`(`*name*, *dir=None*, *runas=None*, *user=None*, ***kwargs*`)`

Verify that the given package is not installed.

dir The target directory in which to install the package, or None for global installation

runas The user to run NPM with

Deprecated since version 0.17.0.

user The user to run NPM with

New in version 0.17.0.

81.47 salt.states.ntp

81.47.1 Management of NTP servers

New in version 2014.1.0: (Hydrogen)

This state is used to manage NTP servers. Currently only Windows is supported.

```
win_ntp:
  ntp.managed:
    - servers:
      - pool.ntp.org
      - us.pool.ntp.org
```

`salt.states.ntp.`**`managed`**`(`*name*, *servers=None*`)`

Manage NTP servers

servers A list of NTP servers

81.48 salt.states.openstack_config

Manage OpenStack configuration file settings.

maintainer Jeffrey C. Ollie <jeff@ocjtech.us>

maturity new

depends

platform linux

`salt.states.openstack_config.`**`absent`**(*name*, *filename*, *section*, *parameter=None*)

Ensure a value is not set in an OpenStack configuration file.

filename The full path to the configuration file

section The section in which the parameter will be set

parameter (optional) The parameter to change. If the parameter is not supplied, the name will be used as the parameter.

`salt.states.openstack_config.`**`present`**(*name*, *filename*, *section*, *value*, *parameter=None*)

Ensure a value is set in an OpenStack configuration file.

filename The full path to the configuration file

section The section in which the parameter will be set

parameter (optional) The parameter to change. If the parameter is not supplied, the name will be used as the parameter.

value The value to set

81.49 salt.states.pagerduty

81.49.1 Create an Event in PagerDuty

New in version 2014.1.0: (Hydrogen)

This state is useful for creating events on the PagerDuty service during state runs.

```
server-warning-message:
  pagerduty.create_event:
    - name: 'This is a server warning message'
    - details: 'This is a much more detailed message'
    - service_key: 9abcd123456789efabcde362783cdbaf
    - profile: my-pagerduty-account
```

`salt.states.pagerduty.`**`create_event`**(*name*, *details*, *service_key*, *profile*)

Create an event on the PagerDuty service

```
server-warning-message:
  pagerduty.create_event:
    - name: 'This is a server warning message'
    - details: 'This is a much more detailed message'
    - service_key: 9abcd123456789efabcde362783cdbaf
    - profile: my-pagerduty-account
```

The following parameters are required:

name This is a short description of the event.

details This can be a more detailed description of the event.

service_key This key can be found by using pagerduty.list_services.

profile This refers to the configuration profile to use to connect to the PagerDuty service.

81.50 salt.states.pecl

81.50.1 Installation of PHP Extensions Using pecl

These states manage the installed pecl extensions. Note that php-pear must be installed for these states to be available, so pecl states should include a requisite to a pkg.installed state for the package which provides pecl (`php-pear` in most cases). Example:

```
php-pear:
  pkg.installed

mongo:
  pecl.installed:
    - require:
      - pkg: php-pear
```

`salt.states.pecl.`**`installed`**(*name*, *version=None*, *defaults=False*, *force=False*, *preferred_state='stable'*)

Make sure that a pecl extension is installed.

name The pecl extension name to install

version The pecl extension version to install. This option may be ignored to install the latest stable version.

defaults Use default answers for extensions such as pecl_http which ask questions before installation. Without this option, the pecl.installed state will hang indefinitely when trying to install these extensions.

force Whether to force the installed version or not

preferred_state The pecl extension state to install

Note: The `defaults` option will be available in version 0.17.0.

`salt.states.pecl.`**`removed`**(*name*)

Make sure that a pecl extension is not installed.

name The pecl extension name to uninstall

81.51 salt.states.pip_state

81.51.1 Installation of Python Packages Using pip

These states manage system installed python packages. Note that pip must be installed for these states to be available, so pip states should include a requisite to a pkg.installed state for the package which provides pip (`python-pip` in most cases). Example:

```
python-pip:
  pkg.installed

virtualenvwrapper:
  pip.installed:
    - require:
      - pkg: python-pip
```

`salt.states.pip_state.`**`installed`**(*name*, *pip_bin=None*, *requirements=None*, *env=None*, *bin_env=None*, *use_wheel=False*, *log=None*, *proxy=None*, *timeout=None*, *repo=None*, *editable=None*, *find_links=None*, *index_url=None*, *extra_index_url=None*, *no_index=False*, *mirrors=None*, *build=None*, *target=None*, *download=None*, *download_cache=None*, *source=None*, *upgrade=False*, *force_reinstall=False*, *ignore_installed=False*, *exists_action=None*, *no_deps=False*, *no_install=False*, *no_download=False*, *install_options=None*, *global_options=None*, *user=None*, *runas=None*, *no_chown=False*, *cwd=None*, *activate=False*, *pre_releases=False*)

Make sure the package is installed

name The name of the python package to install. You can also specify version numbers here using the standard operators `==, >=, <=`. If `requirements` is given, this parameter will be ignored.

Example:

```
django:
  pip.installed:
    - name: django >= 1.6, <= 1.7
    - require:
      - pkg: python-pip
```

This will install the latest Django version greater than 1.6 but less than 1.7.

user The user under which to run pip

use_wheel [False] Prefer wheel archives (requires pip>=1.4)

bin_env [None] Absolute path to a virtual environment directory or absolute path to a pip executable. The example below assumes a virtual environment has been created at `/foo/.virtualenvs/bar`.

Example:

```
django:
  pip.installed:
    - name: django >= 1.6, <= 1.7
    - bin_env: /foo/.virtualenvs/bar
    - require:
      - pkg: python-pip
```

Or

Example:

```
django:
  pip.installed:
    - name: django >= 1.6, <= 1.7
    - bin_env: /foo/.virtualenvs/bar/bin/pip
    - require:
      - pkg: python-pip
```

Attention

The following arguments are deprecated, do not use.

---

pip_bin [None] Deprecated, use `bin_env`

env [None] Deprecated, use `bin_env`

Changed in version 0.17.0: `use_wheel` option added.

install_options

Extra arguments to be supplied to the setup.py install command. If you are using an option with a directory path, be sure to use absolute path.

Example:

```
django:
  pip.installed:
    - name: django
    - install_options:
      - --prefix=/blah
    - require:
      - pkg: python-pip
```

global_options Extra global options to be supplied to the setup.py call before the install command.

New in version 2014.1.3.

Attention

As of Salt 0.17.0 the pip state **needs** an importable pip module. This usually means having the system's pip package installed or running Salt from an active virtualenv.

The reason for this requirement is because `pip` already does a pretty good job parsing it's own requirements. It makes no sense for Salt to do `pip` requirements parsing and validation before passing them to the `pip` library. It's functionality duplication and it's more error prone.

`salt.states.pip_state.`**`removed`**`(`*name*, *requirements=None*, *bin_env=None*, *log=None*, *proxy=None*, *timeout=None*, *user=None*, *runas=None*, *cwd=None*`)`

Make sure that a package is not installed.

name The name of the package to uninstall

user The user under which to run pip

bin_env [None] the pip executable or virtualenenv to use

81.52 salt.states.pkg

81.52.1 Installation of packages using OS package managers such as yum or apt-get

Salt can manage software packages via the pkg state module, packages can be set up to be installed, latest, removed and purged. Package management declarations are typically rather simple:

```
vim:
  pkg.installed
```

A more involved example involves pulling from a custom repository. Note that the pkgrepo has a require_in clause. This is necessary and can not be replaced by a require clause in the pkg.

```
base:
  pkgrepo.managed:
    - humanname: Logstash PPA
    - name: deb http://ppa.launchpad.net/wolfnet/logstash/ubuntu precise main
    - dist: precise
    - file: /etc/apt/sources.list.d/logstash.list
    - keyid: 28B04E4A
    - keyserver: keyserver.ubuntu.com
    - require_in:
      - pkg: logstash

logstash:
  pkg.installed
```

`salt.states.pkg.`**`installed`**`(`*name*, *version=None*, *refresh=None*, *fromrepo=None*, *skip_verify=False*, *skip_suggestions=False*, *pkgs=None*, *sources=None*, ***kwargs*`)`

Verify that the package is installed, and that it is the correct version (if specified).

name The name of the package to be installed. This parameter is ignored if either "pkgs" or "sources" is used. Additionally, please note that this option can only be used to install packages from a software repository. To install a package file manually, use the "sources" option detailed below.

fromrepo Specify a repository from which to install

skip_verify Skip the GPG verification check for the package to be installed

skip_suggestions Force strict package naming. Disables lookup of package alternatives.

New in version 2014.1.1.

version Install a specific version of a package. This option is ignored if either "pkgs" or "sources" is used. Currently, this option is supported for the following pkg providers: `apt`, `ebuild`, `pacman`, `yumpkg`, and `zypper`.

refresh Update the repo database of available packages prior to installing the requested package.

Usage:

```
httpd:
  pkg.installed:
    - fromrepo: mycustomrepo
    - skip_verify: True
    - skip_suggestions: True
    - version: 2.0.6~ubuntu3
    - refresh: True
```

Multiple Package Installation Options: (not supported in Windows or pkgng)

pkgs A list of packages to install from a software repository.

Usage:

```
mypkgs:
  pkg.installed:
    - pkgs:
```

```
    - foo
    - bar
    - baz
```

`NOTE:` For `apt`, `ebuild`, `pacman`, `yumpkg`, and `zypper`, version numbers can be specified in the `pkgs` argument. Example:

```
mypkgs:
  pkg.installed:
    - pkgs:
      - foo
      - bar: 1.2.3-4
      - baz
```

Additionally, `ebuild`, `pacman` and `zypper` support the <, <=, >=, and > operators for more control over what versions will be installed. Example:

```
mypkgs:
  pkg.installed:
    - pkgs:
      - foo
      - bar: '>=1.2.3-4'
      - baz
```

`NOTE:` When using comparison operators, the expression must be enclosed in quotes to avoid a YAML render error.

With `ebuild` is also possible to specify a use flag list and/or if the given packages should be in package.accept_keywords file and/or the overlay from which you want the package to be installed. Example:

```
mypkgs:
    pkg.installed:
        - pkgs:
            - foo: '~'
            - bar: '~>=1.2:slot::overlay[use,-otheruse]'
            - baz
```

sources A list of packages to install, along with the source URI or local path from which to install each package. In the example below, `foo`, `bar`, `baz`, etc. refer to the name of the package, as it would appear in the output of the `pkg.version` or `pkg.list_pkgs` salt CLI commands.

Usage:

```
mypkgs:
  pkg.installed:
    - sources:
      - foo: salt://rpms/foo.rpm
      - bar: http://somesite.org/bar.rpm
      - baz: ftp://someothersite.org/baz.rpm
      - qux: /minion/path/to/qux.rpm
```

`salt.states.pkg.`**`latest`**(*name*, *refresh=None*, *fromrepo=None*, *skip_verify=False*, *pkgs=None*, ***kwargs*)

Verify that the named package is installed and the latest available package. If the package can be updated this state function will update the package. Generally it is better for the `installed` function to be used, as `latest` will update the package whenever a new package is available.

name The name of the package to maintain at the latest available version. This parameter is ignored if "pkgs" is used.

fromrepo Specify a repository from which to install

skip_verify Skip the GPG verification check for the package to be installed

Multiple Package Installation Options:

(Not yet supported for: Windows, FreeBSD, OpenBSD, MacOS, and Solaris pkgutil)

pkgs A list of packages to maintain at the latest available version.

Usage:

```
mypkgs:
  pkg.latest:
    - pkgs:
      - foo
      - bar
      - baz
```

`salt.states.pkg.`**`purged`**(*name*, *pkgs=None*, ***kwargs*)

Verify that a package is not installed, calling `pkg.purge` if necessary to purge the package.

name The name of the package to be purged.

Multiple Package Options:

pkgs A list of packages to purge. Must be passed as a python list. The `name` parameter will be ignored if this option is passed.

New in version 0.16.0.

`salt.states.pkg.`**`removed`**(*name*, *pkgs=None*, ***kwargs*)

Verify that a package is not installed, calling `pkg.remove` if necessary to remove the package.

name The name of the package to be removed.

Multiple Package Options:

pkgs A list of packages to remove. Must be passed as a python list. The `name` parameter will be ignored if this option is passed.

New in version 0.16.0.

81.53 salt.states.pkgng

81.53.1 Manage package remote repo using FreeBSD pkgng

Salt can manage the URL pkgng pulls packages from. ATM the state and module are small so use cases are typically rather simple:

```
pkgng_clients:
  pkgng:
    - update_packaging_site
    - name: "http://192.168.0.2"
```

`salt.states.pkgng.`**`update_packaging_site`**(*name*)

81.54 salt.states.pkgrepo

81.54.1 Management of package repos

Package repositories can be managed with the pkgrepo state:

```
base:
  pkgrepo.managed:
    - humanname: CentOS-$releasever - Base
    - mirrorlist: http://mirrorlist.centos.org/?release=$releasever&arch=$basearch&repo=os
    - comments:
        - '#http://mirror.centos.org/centos/$releasever/os/$basearch/'
    - gpgcheck: 1
    - gpgkey: file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-6

base:
  pkgrepo.managed:
    - humanname: Logstash PPA
    - name: deb http://ppa.launchpad.net/wolfnet/logstash/ubuntu precise main
    - dist: precise
    - file: /etc/apt/sources.list.d/logstash.list
    - keyid: 28B04E4A
    - keyserver: keyserver.ubuntu.com
    - require_in:
      - pkg: logstash

  pkg.latest:
    - name: logstash
    - refresh: True

base:
  pkgrepo.managed:
    - ppa: wolfnet/logstash
  pkg.latest:
    - name: logstash
    - refresh: True
```

Note: On Ubuntu systems, the `python-software-properties` package should be installed for better support of PPA repositories. To check if this package is installed, run `dpkg -l python-software-properties`.

Also, some Ubuntu releases have a bug in their `python-software-properties` package, a missing dependency on pycurl, so `python-pycurl` will need to be manually installed if it is not present once `python-software-properties` is installed.

`salt.states.pkgrepo.absent(name, **kwargs)`

This function deletes the specified repo on the system, if it exists. It is essentially a wrapper around pkg.del_repo.

name The name of the package repo, as it would be referred to when running the regular package manager commands.

ppa On Ubuntu, you can take advantage of Personal Package Archives on Launchpad simply by specifying the user and archive name.

```
logstash-ppa:
  pkgrepo.absent:
    - ppa: wolfnet/logstash
```

ppa_auth For Ubuntu PPAs there can be private PPAs that require authentication to access. For these PPAs the username/password can be specified. This is required for matching if the name format uses the "ppa:" specifier and is private (requires username/password to access, which is encoded in the URI).

```
logstash-ppa:
  pkgrepo.absent:
    - ppa: wolfnet/logstash
    - ppa_auth: username:password
```

`salt.states.pkgrepo.`**`managed`**`(`*name*, ***kwargs*`)`

This function manages the configuration on a system that points to the repositories for the system's package manager.

name The name of the package repo, as it would be referred to when running the regular package manager commands.

For yum-based systems, take note of the following configuration values:

humanname On yum-based systems, this is stored as the "name" value in the .repo file in /etc/yum.repos.d/. On yum-based systems, this is required.

baseurl On yum-based systems, baseurl refers to a direct URL to be used for this yum repo. One of baseurl or mirrorlist is required.

mirrorlist a URL which contains a collection of baseurls to choose from. On yum-based systems. One of baseurl or mirrorlist is required.

comments Sometimes you want to supply additional information, but not as enabled configuration. Anything supplied for this list will be saved in the repo configuration with a comment marker (#) in front.

Additional configuration values, such as gpgkey or gpgcheck, are used verbatim to update the options for the yum repo in question.

For apt-based systems, take note of the following configuration values:

ppa On Ubuntu, you can take advantage of Personal Package Archives on Launchpad simply by specifying the user and archive name. The keyid will be queried from launchpad and everything else is set automatically. You can override any of the below settings by simply setting them as you would normally. For example:

```
logstash-ppa:
  pkgrepo.managed:
    - ppa: wolfnet/logstash
```

ppa_auth For Ubuntu PPAs there can be private PPAs that require authentication to access. For these PPAs the username/password can be passed as an HTTP Basic style username/password combination.

```
logstash-ppa:
  pkgrepo.managed:
    - ppa: wolfnet/logstash
    - ppa_auth: username:password
```

name On apt-based systems this must be the complete entry as it would be seen in the sources.list file. This can have a limited subset of components (i.e. 'main') which can be added/modified with the "comps" option.

```
precise-repo:
  pkgrepo.managed:
    - name: deb http://us.archive.ubuntu.com/ubuntu precise main
```

disabled On apt-based systems, disabled toggles whether or not the repo is used for resolving dependencies and/or installing packages

comps On apt-based systems, comps dictate the types of packages to be installed from the repository (e.g. main, nonfree, ...). For purposes of this, comps should be a comma-separated list.

file The filename for the .list that the repository is configured in. It is important to include the full-path AND make sure it is in a directory that APT will look in when handling packages

dist This dictates the release of the distro the packages should be built for. (e.g. unstable)

keyid The KeyID of the GPG key to install. This option also requires the 'keyserver' option to be set.

keyserver This is the name of the keyserver to retrieve gpg keys from. The keyid option must also be set for this option to work.

key_url A web URL to retrieve the GPG key from.

consolidate If set to true, this will consolidate all sources definitions to the sources.list file, cleanup the now unused files, consolidate components (e.g. main) for the same URI, type, and architecture to a single line, and finally remove comments from the sources.list file. The consolidate will run every time the state is processed. The option only needs to be set on one repo managed by salt to take effect.

require_in Set this to a list of pkg.installed or pkg.latest to trigger the running of apt-get update prior to attempting to install these packages. Setting a require in the pkg will not work for this.

81.55 salt.states.portage_config

81.55.1 Management of Portage package configuration on Gentoo

A state module to manage Portage configuration on Gentoo

```
salt:
    portage_config.flags:
        - use:
            - openssl
```

`salt.states.portage_config.`**`flags`**(*name*, *use=None*, *accept_keywords=None*, *env=None*, *license=None*, *properties=None*, *unmask=False*, *mask=False*)

Enforce the given flags on the given package or DEPEND atom. Please be warned that, in most cases, you need to rebuild the affected packages in order to apply the changes.

name The name of the package or his DEPEND atom

use A list of use flags

accept_keywords A list of keywords to accept. "~ARCH" means current host arch, and will be translated in a line without keywords

env A list of environment files

license A list of accepted licenses

properties A list of additional properties

unmask A boolean to unmask the package

mask A boolean to mask the package

81.56 salt.states.ports

Manage software from FreeBSD ports

New in version 2014.1.0: (Hydrogen)

Note: It may be helpful to use a higher timeout when running a `ports.installed` state, since compiling the port may exceed Salt's timeout.

```
salt -t 1200 '*' state.highstate
```

`salt.states.ports.`**`installed`**`(`*name*, *options=None*`)`

Verify that the desired port is installed, and that it was compiled with the desired options.

options Make sure that the desired non-default options are set

> **Warning:** Any build options not passed here assume the default values for the port, and are not just differences from the existing cached options from a previous `make config`.

Example usage:

```
security/nmap:
  ports.installed:
    - options:
      - IPV6: off
```

81.57 salt.states.postgres_database

81.57.1 Management of PostgreSQL databases

The postgres_database module is used to create and manage Postgres databases. Databases can be set as either absent or present

```
frank:
  postgres_database.present
```

`salt.states.postgres_database.`**`absent`**`(`*name*, *runas=None*, *user=None*, *maintenance_db=None*, *db_password=None*, *db_host=None*, *db_port=None*, *db_user=None*`)`

Ensure that the named database is absent

name The name of the database to remove

db_user database username if different from config or defaul

db_password user password if any password for a specified user

db_host Database host if different from config or default

db_port Database port if different from config or default

runas System user all operations should be performed on behalf of

Deprecated since version 0.17.0.

user System user all operations should be performed on behalf of

New in version 0.17.0.

`salt.states.postgres_database.`**`present`**(*name*, *tablespace=None*, *encoding=None*, *lc_collate=None*, *lc_ctype=None*, *owner=None*, *template=None*, *runas=None*, *user=None*, *maintenance_db=None*, *db_password=None*, *db_host=None*, *db_port=None*, *db_user=None*)

Ensure that the named database is present with the specified properties. For more information about all of these options see man createdb(1)

name The name of the database to manage

tablespace Default tablespace for the database

encoding The character encoding scheme to be used in this database

lc_collate The LC_COLLATE setting to be used in this database

lc_ctype The LC_CTYPE setting to be used in this database

owner The username of the database owner

template The template database from which to build this database

runas System user all operations should be performed on behalf of

Deprecated since version 0.17.0.

user System user all operations should be performed on behalf of

db_user database username if different from config or defaul

db_password user password if any password for a specified user

db_host Database host if different from config or default

db_port Database port if different from config or default

New in version 0.17.0.

81.58 salt.states.postgres_group

81.58.1 Management of PostgreSQL groups (roles)

The postgres_group module is used to create and manage Postgres groups.

```
frank:
  postgres_group.present
```

`salt.states.postgres_group.`**`absent`**(*name*, *runas=None*, *user=None*, *maintenance_db=None*, *db_password=None*, *db_host=None*, *db_port=None*, *db_user=None*)

Ensure that the named group is absent

name The groupname of the group to remove

runas System user all operations should be performed on behalf of

Deprecated since version 0.17.0.

user System user all operations should be performed on behalf of

New in version 0.17.0.

db_user database username if different from config or defaul

db_password user password if any password for a specified user

db_host Database host if different from config or default

db_port Database port if different from config or default

`salt.states.postgres_group.`**`present`**(*name*, *createdb=None*, *createroles=None*, *createuser=None*, *encrypted=None*, *superuser=None*, *inherit=None*, *login=None*, *replication=None*, *password=None*, *groups=None*, *runas=None*, *user=None*, *maintenance_db=None*, *db_password=None*, *db_host=None*, *db_port=None*, *db_user=None*)

Ensure that the named group is present with the specified privileges Please note that the user/group notion in postgresql is just abstract, we have roles, where users can be seens as roles with the LOGIN privilege and groups the others.

name The name of the group to manage

createdb Is the group allowed to create databases?

createroles Is the group allowed to create other roles/users

createuser Alias to create roles, and history problem, in pgsql normally createuser == superuser

encrypted Should the password be encrypted in the system catalog?

login Should the group have login perm

inherit Should the group inherit permissions

superuser Should the new group be a "superuser"

replication Should the new group be allowed to initiate streaming replication

password The Group's password It can be either a plain string or a md5 postgresql hashed password:

```
'md5{MD5OF({password}{role}}'
```

If encrypted is None or True, the password will be automaticly encrypted to the previous format if it is not already done.

groups A string of comma separated groups the group should be in

runas System user all operations should be performed on behalf of

Deprecated since version 0.17.0.

user System user all operations should be performed on behalf of

New in version 0.17.0.

db_user database username if different from config or defaul

db_password user password if any password for a specified user

db_host Database host if different from config or default

db_port Database port if different from config or default

81.59 salt.states.postgres_user

81.59.1 Management of PostgreSQL users (roles)

The postgres_users module is used to create and manage Postgres users.

```
frank:
  postgres_user.present
```

`salt.states.postgres_user.`**`absent`**(*name*, *runas=None*, *user=None*, *maintenance_db=None*, *db_password=None*, *db_host=None*, *db_port=None*, *db_user=None*)

Ensure that the named user is absent

name The username of the user to remove

runas System user all operations should be performed on behalf of

Deprecated since version 0.17.0.

user System user all operations should be performed on behalf of

New in version 0.17.0.

db_user database username if different from config or default

db_password user password if any password for a specified user

db_host Database host if different from config or default

db_port Database port if different from config or default

`salt.states.postgres_user.`**`present`**(*name*, *createdb=None*, *createroles=None*, *createuser=None*, *encrypted=None*, *superuser=None*, *replication=None*, *inherit=None*, *login=None*, *password=None*, *groups=None*, *runas=None*, *user=None*, *maintenance_db=None*, *db_password=None*, *db_host=None*, *db_port=None*, *db_user=None*)

Ensure that the named user is present with the specified privileges Please note that the user/group notion in postgresql is just abstract, we have roles, where users can be seens as roles with the LOGIN privilege and groups the others.

name The name of the user to manage

createdb Is the user allowed to create databases?

createroles Is the user allowed to create other users?

createuser Alias to create roles

encrypted Should the password be encrypted in the system catalog?

login Should the group have login perm

inherit Should the group inherit permissions

superuser Should the new user be a "superuser"

replication Should the new user be allowed to initiate streaming replication

password The user's password It can be either a plain string or a md5 postgresql hashed password:

```
'md5{MD5OF({password}{role}}'
```

If encrypted is None or True, the password will be automaticly encrypted to the previous format if it is not already done.

groups A string of comma separated groups the user should be in

runas System user all operations should be performed on behalf of

Deprecated since version 0.17.0.

user System user all operations should be performed on behalf of

New in version 0.17.0.

db_user database username if different from config or default

db_password user password if any password for a specified user

db_host Database host if different from config or default

db_port Database port if different from config or default

81.60 salt.states.powerpath

81.60.1 Powerpath configuration support

Allows configuration of EMC Powerpath. Currently only addition/deletion of licenses is supported.

```
key:
  powerpath:
    - license_present
```

`salt.states.powerpath.`**`license_absent`**(*name*)

Ensures that the specified PowerPath license key is absent on the host.

name The licnese key to ensure is absent

`salt.states.powerpath.`**`license_present`**(*name*)

Ensures that the specified PowerPath license key is present on the host.

name The licnese key to ensure is present

81.61 salt.states.process

81.61.1 Process Management

Ensure a process matching a given pattern is absent.

```
httpd-absent:
  process.absent:
    - name: apache2
```

`salt.states.process.`**`absent`**(*name*)

Ensures that the named command is not running.

name The pattern to match.

81.62 salt.states.quota

81.62.1 Management of POSIX Quotas

The quota can be managed for the system:

```
/:
  quota.mode:
    mode: off
    quotatype: user
```

`salt.states.quota.`**`mode`**(*name*, *mode*, *quotatype*)

Set the quota for the system

name The filesystem to set the quota mode on

mode Whether the quota system is 'on' or 'off'

quotatype Need to be 'user' or 'group'

81.63 salt.states.rabbitmq_cluster

81.63.1 Manage RabbitMQ Clusters

Example:

```
rabbit@rabbit.example.com:
    rabbitmq_cluster.join:
      - user: rabbit
      - host: rabbit.example.com
```

`salt.states.rabbitmq_cluster.`**`join`**(*name*, *host*, *user='rabbit'*, *runas=None*)

Ensure the RabbitMQ plugin is enabled.

name Irrelevant, not used (recommended: user@host)

user The user to join the cluster as (default: rabbit)

host The cluster host to join to

runas The user to run the rabbitmq-plugin command as

81.64 salt.states.rabbitmq_plugin

81.64.1 Manage RabbitMQ Plugins

New in version 2014.1.0: (Hydrogen)

Example:

```
some_plugin:
    rabbitmq_plugin:
      - enabled
```

`salt.states.rabbitmq_plugin.`**`disabled`**(*name*, *runas=None*)

Ensure the RabbitMQ plugin is enabled.

name The name of the plugin

runas The user to run the rabbitmq-plugin command as

`salt.states.rabbitmq_plugin.`**`enabled`**(*name*, *runas=None*)

Ensure the RabbitMQ plugin is enabled.

name The name of the plugin

runas The user to run the rabbitmq-plugin command as

81.65 salt.states.rabbitmq_policy

81.65.1 Manage RabbitMQ Policies

maintainer Benn Eichhorn <benn@getlocalmeasure.com>

maturity new

platform all

Example:

```
rabbit_policy:
    rabbitmq_policy.present:
        - name: HA
        - pattern: '.*'
        - definition: '{"ha-mode": "all"}'
```

`salt.states.rabbitmq_policy.`**`absent`**(*name*, *vhost='/'*, *runas=None*)

Ensure the named policy is absent

Reference: http://www.rabbitmq.com/ha.html

name The name of the policy to remove

runas Name of the user to run the command as

`salt.states.rabbitmq_policy.`**`present`**(*name*, *pattern*, *definition*, *priority=0*, *vhost='/'*, *runas=None*)

Ensure the RabbitMQ policy exists.

Reference: http://www.rabbitmq.com/ha.html

name Policy name

pattern A regex of queues to apply the policy to

definition A json dict describing the policy

priority Priority (defaults to 0)

vhost Virtual host to apply to (defaults to '/')

runas Name of the user to run the command as

81.66 salt.states.rabbitmq_user

81.66.1 Manage RabbitMQ Users

Example:

```
rabbit_user:
    rabbitmq_user.present:
        - password: password
        - force: True
        - tags: administrator
        - perms:
          - '/':
            - '.*'
            - '.*'
            - '.*'
        - runas: rabbitmq
```

`salt.states.rabbitmq_user.`**`absent`**(*name*, *runas=None*)

Ensure the named user is absent

name The name of the user to remove

runas User to run the command

`salt.states.rabbitmq_user.`**`present`**(*name*, *password=None*, *force=False*, *tags=None*, *perms=()*, *runas=None*)

Ensure the RabbitMQ user exists.

name User name

password User's password, if one needs to be set

force If user exists, forcibly change the password

tags Optionally set user tags for user

perms A list of dicts with vhost keys and 3-tuple values

runas Name of the user to run the command

81.67 salt.states.rabbitmq_vhost

81.67.1 Manage RabbitMQ Virtual Hosts

Example:

```
virtual_host:
    rabbitmq_vhost.present:
        - user: rabbit_user
        - conf: .*
        - write: .*
        - read: .*
```

`salt.states.rabbitmq_vhost.`**`absent`**(*name*, *runas=None*)

Ensure the RabbitMQ Virtual Host is absent

name Name of the Virtual Host to remove

runas User to run the command

`salt.states.rabbitmq_vhost.`**`present`**(*name*, *user=None*, *owner=None*, *conf=None*, *write=None*, *read=None*, *runas=None*)

Ensure the RabbitMQ VHost exists.

name VHost name

user Initial user permission to set on the VHost, if present .. deprecated:: 0.17.0

owner Initial owner permission to set on the VHost, if present

conf Initial conf string to apply to the VHost and user. Defaults to .*

write Initial write permissions to apply to the VHost and user. Defaults to .*

read Initial read permissions to apply to the VHost and user. Defaults to .*

runas Name of the user to run the command

81.68 salt.states.rbenv

81.68.1 Managing Ruby installations with rbenv

This module is used to install and manage ruby installations with rbenv. Different versions of ruby can be installed, and uninstalled. Rbenv will be installed automatically the first time it is needed and can be updated later. This module will *not* automatically install packages which rbenv will need to compile the versions of ruby.

If rbenv is run as the root user then it will be installed to /usr/local/rbenv, otherwise it will be installed to the users ~/.rbenv directory. To make rbenv available in the shell you may need to add the rbenv/shims and rbenv/bin directories to the users PATH. If you are installing as root and want other users to be able to access rbenv then you will need to add RBENV_ROOT to their environment.

This is how a state configuration could look like:

```
rbenv-deps:
  pkg.installed:
    - pkgs:
      - bash
      - git
      - openssl
      - gmake
      - curl

ruby-1.9.3-p392:
  rbenv.absent:
    - require:
      - pkg: rbenv-deps

ruby-1.9.3-p429:
  rbenv.installed:
    - default: True
    - require:
      - pkg: rbenv-deps
```

`salt.states.rbenv.`**`absent`**(*name*, *runas=None*, *user=None*)

Verify that the specified ruby is not installed with rbenv. Rbenv is installed if necessary.

name The version of ruby to uninstall

runas: None The user to run rbenv as.

Deprecated since version 0.17.0.

user: None The user to run rbenv as.

New in version 0.17.0.

New in version 0.16.0.

`salt.states.rbenv.`**`installed`**(*name*, *default=False*, *runas=None*, *user=None*)

Verify that the specified ruby is installed with rbenv. Rbenv is installed if necessary.

name The version of ruby to install

default [False] Whether to make this ruby the default.

runas: None The user to run rbenv as.

Deprecated since version 0.17.0.

user: None The user to run rbenv as.

New in version 0.17.0.

New in version 0.16.0.

81.69 salt.states.rdp

Manage RDP Service on Windows servers

`salt.states.rdp.`**`disabled`**`(`*name*`)`
Disable RDP the service on the server

`salt.states.rdp.`**`enabled`**`(`*name*`)`
Enable RDP the service on the server

81.70 salt.states.reg

Manage the registry on Windows

`salt.states.reg.`**`present`**`(`*name*, *value*, *vtype='REG_DWORD'*`)`
Set a registry entry

Example:

```
'HKEY_CURRENT_USER\SOFTWARE\Salt\version':
  reg.present:
    - value: 0.15.3
    - vtype: REG_SZ
```

81.71 salt.states.rvm

81.71.1 Managing Ruby installations and gemsets with Ruby Version Manager (RVM)

This module is used to install and manage ruby installations and gemsets with RVM, the Ruby Version Manager. Different versions of ruby can be installed and gemsets created. RVM itself will be installed automatically if it's not present. This module will not automatically install packages that RVM depends on or ones that are needed to build ruby. If you want to run RVM as an unprivileged user (recommended) you will have to create this user yourself. This is how a state configuration could look like:

```
rvm:
  group:
    - present
  user.present:
    - gid: rvm
    - home: /home/rvm
    - require:
      - group: rvm

rvm-deps:
  pkg.installed:
    - names:
      - bash
      - coreutils
      - gzip
      - bzip2
      - gawk
      - sed
      - curl
      - git-core
      - subversion

mri-deps:
  pkg.installed:
    - names:
      - build-essential
      - openssl
      - libreadline6
      - libreadline6-dev
      - curl
      - git-core
      - zlib1g
      - zlib1g-dev
      - libssl-dev
      - libyaml-dev
      - libsqlite3-0
      - libsqlite3-dev
      - sqlite3
      - libxml2-dev
      - libxslt1-dev
      - autoconf
      - libc6-dev
      - libncurses5-dev
      - automake
      - libtool
      - bison
      - subversion
      - ruby

jruby-deps:
  pkg.installed:
    - names:
      - curl
      - g++
      - openjdk-6-jre-headless

ruby-1.9.2:
  rvm.installed:
    - default: True
    - user: rvm
    - require:
      - pkg: rvm-deps
      - pkg: mri-deps
      - user: rvm
```

```
jruby:
  rvm.installed:
    - user: rvm
    - require:
      - pkg: rvm-deps
      - pkg: jruby-deps
      - user: rvm

jgemset:
  rvm.gemset_present:
    - ruby: jruby
    - user: rvm
    - require:
      - rvm: jruby

mygemset:
  rvm.gemset_present:
    - ruby: ruby-1.9.2
    - user: rvm
    - require:
      - rvm: ruby-1.9.2
```

`salt.states.rvm.`**`gemset_present`**(*name*, *ruby='default'*, *runas=None*, *user=None*)

Verify that the gemset is present.

name The name of the gemset.

ruby: default The ruby version this gemset belongs to.

runas: None The user to run rvm as.

Deprecated since version 0.17.0.

user: None The user to run rvm as.

New in version 0.17.0.

`salt.states.rvm.`**`installed`**(*name*, *default=False*, *runas=None*, *user=None*)

Verify that the specified ruby is installed with RVM. RVM is installed when necessary.

name The version of ruby to install

default [False] Whether to make this ruby the default.

runas: None The user to run rvm as.

Deprecated since version 0.17.0.

user: None The user to run rvm as.

New in version 0.17.0.

81.72 salt.states.saltmod

81.72.1 Control the Salt command interface

The Salt state is used to control the salt command interface. This state is intended for use primarily from the state runner from the master.

The salt.state declaration can call out a highstate or a list of sls:

webservers:

salt.state:

- tgt: 'web*'
- sls: - apache - django - core
- saltenv: prod

databasees:

salt.state:

- tgt: role:database
- tgt_type: grain
- highstate: True

`salt.states.saltmod.`**`function`**(*name*, *tgt*, *ssh=False*, *tgt_type=None*, *ret=''*, *arg=()*, ***kwargs*)

Execute a single module function on a remote minion via salt or salt-ssh

name The name of the function to run, aka cmd.run or pkg.install

tgt The target specification, aka '*' for all minions

tgt_type | expr_form The target type, defaults to glob

arg The list of arguments to pass into the function

ret Optionally set a single or a list of returners to use

ssh Set to *True* to use the ssh client instaed of the standard salt client

`salt.states.saltmod.`**`state`**(*name*, *tgt*, *ssh=False*, *tgt_type=None*, *ret=''*, *highstate=None*, *sls=None*, *env=None*, *test=False*, *fail_minions=''*, *allow_fail=0*, ***kwargs*)

Invoke a state run on a given target

name An arbitrary name used to track the state execution

tgt The target specification for the state run.

tgt_type | expr_form The target type to resolve, defaults to glob

ret Optionally set a single or a list of returners to use

highstate Defaults to None, if set to True the target systems will ignore any sls references specified in the sls option and call state.highstate on the targeted minions

sls A group of sls files to execute. This can be defined as a single string containing a single sls file, or a list of sls files

saltenv The default salt environment to pull sls files from

ssh Set to *True* to use the ssh client instaed of the standard salt client

roster In the event of using salt-ssh, a roster system can be set

fail_minions An optional list of targeted minions where failure is an option

81.73 salt.states.selinux

81.73.1 Management of SELinux rules

If SELinux is available for the running system, the mode can be managed and booleans can be set.

```
enforcing:
    selinux.mode

samba_create_home_dirs:
    selinux.boolean:
      - value: True
      - persist: True
```

Note: Use of these states require that the `selinux` execution module is available.

`salt.states.selinux.`**`boolean`**(*name*, *value*, *persist=False*)

Set up an SELinux boolean

name The name of the boolean to set

value The value to set on the boolean

persist Defaults to False, set persist to true to make the boolean apply on a reboot

`salt.states.selinux.`**`mode`**(*name*)

Verifies the mode SELinux is running in, can be set to enforcing or permissive

name The mode to run SELinux in, permissive or enforcing

81.74 salt.states.service

81.74.1 Starting or restarting of services and daemons

Services are defined as system daemons typically started with system init or rc scripts, services can be defined as running or dead.

```
httpd:
  service:
    - running
```

The service can also be set to be started at runtime via the enable option:

```
openvpn:
  service:
    - running
    - enable: True
```

By default if a service is triggered to refresh due to a watch statement the service is by default restarted. If the desired behaviour is to reload the service, then set the reload value to True:

```
redis:
  service:
    - running
    - enable: True
    - reload: True
    - watch:
      - pkg: redis
```

`salt.states.service.`**`dead`**`(`*name*, *enable=None*, *sig=None*, ***kwargs*`)`
Ensure that the named service is dead by stopping the service if it is running

name The name of the init or rc script used to manage the service

enable Set the service to be enabled at boot time, `True` sets the service to be enabled, `False` sets the named service to be disabled. The default is `None`, which does not enable or disable anything.

sig The string to search for when looking for the service process with ps

`salt.states.service.`**`disabled`**`(`*name*, ***kwargs*`)`
Verify that the service is disabled on boot, only use this state if you don't want to manage the running process, remember that if you want to disable a service to use the enable: False option for the running or dead function.

name The name of the init or rc script used to manage the service

`salt.states.service.`**`enabled`**`(`*name*, ***kwargs*`)`
Verify that the service is enabled on boot, only use this state if you don't want to manage the running process, remember that if you want to enable a running service to use the enable: True option for the running or dead function.

name The name of the init or rc script used to manage the service

`salt.states.service.`**`mod_watch`**`(`*name*, *sfun=None*, *sig=None*, *reload=False*, *full_restart=False*`)`
The service watcher, called to invoke the watch command.

name The name of the init or rc script used to manage the service

sfun The original function which triggered the mod_watch call (*service.running*, for example). Currently not used, but must be supported for the future.

sig The string to search for when looking for the service process with ps

`salt.states.service.`**`running`**`(`*name*, *enable=None*, *sig=None*, ***kwargs*`)`
Verify that the service is running

name The name of the init or rc script used to manage the service

enable Set the service to be enabled at boot time, True sets the service to be enabled, False sets the named service to be disabled. The default is None, which does not enable or disable anything.

sig The string to search for when looking for the service process with ps

81.75 salt.states.ssh_auth

81.75.1 Control of entries in SSH authorized_key files

The information stored in a user's SSH authorized key file can be easily controlled via the ssh_auth state. Defaults can be set by the enc, options, and comment keys. These defaults can be overridden by including them in the name.

Since the YAML specification limits the length of simple keys to 1024 characters, and since SSH keys are often longer than that, you may have to use a YAML 'explicit key', as demonstrated in the second example below.

```
AAAAB3NzaC1kc3MAAACBAL0sQ9fJ5bYTEyY==:
  ssh_auth:
    - present
    - user: root
    - enc: ssh-dss

? AAAAB3NzaC1kc3MAAACBAL0sQ9fJ5bYTEyY==...
:
  ssh_auth:
    - present
    - user: root
    - enc: ssh-dss

thatch:
  ssh_auth:
    - present
    - user: root
    - source: salt://ssh_keys/thatch.id_rsa.pub

sshkeys:
  ssh_auth:
    - present
    - user: root
    - enc: ssh-rsa
    - options:
      - option1="value1"
      - option2="value2 flag2"
    - comment: myuser
    - names:
      - AAAAB3NzaC1kc3MAAACBAL0sQ9fJ5bYTEyY==
      - ssh-dss AAAAB3NzaCL0sQ9fJ5bYTEyY== user@domain
      - option3="value3" ssh-dss AAAAB3NzaC1kcQ9J5bYTEyY== other@testdomain
      - AAAAB3NzaC1kcQ9fJFF435bYTEyY== newcomment
```

`salt.states.ssh_auth.`**`absent`**(*name*, *user*, *enc='ssh-rsa'*, *comment=''*, *options=None*, *config='.ssh/authorized_keys'*)

Verifies that the specified SSH key is absent

name The SSH key to manage

user The user who owns the SSH authorized keys file to modify

enc Defines what type of key is being used; can be ecdsa, ssh-rsa or ssh-dss

comment The comment to be placed with the SSH public key

options The options passed to the key, pass a list object

config The location of the authorized keys file relative to the user's home directory, defaults to ".ssh/authorized_keys"

`salt.states.ssh_auth.`**`present`**(*name*, *user*, *enc='ssh-rsa'*, *comment=''*, *source=''*, *options=None*, *config='.ssh/authorized_keys'*, ***kwargs*)

Verifies that the specified SSH key is present for the specified user

name The SSH key to manage

user The user who owns the SSH authorized keys file to modify

enc Defines what type of key is being used; can be ecdsa, ssh-rsa or ssh-dss

comment The comment to be placed with the SSH public key

source The source file for the key(s). Can contain any number of public keys, in standard "authorized_keys" format. If this is set, comment, enc, and options will be ignored.

Note: The source file must contain keys in the format `<enc> <key> <comment>`. If you have generated a keypair using PuTTYgen, then you will need to do the following to retrieve an OpenSSH-compatible public key.

1. In PuTTYgen, click `Load`, and select the *private* key file (not the public key), and click `Open`.
2. Copy the public key from the box labeled `Public key for pasting into OpenSSH authorized_keys file`.
3. Paste it into a new file.

---

options The options passed to the key, pass a list object

config The location of the authorized keys file relative to the user's home directory, defaults to ".ssh/authorized_keys"

81.76 salt.states.ssh_known_hosts

81.76.1 Control of SSH known_hosts entries

Manage the information stored in the known_hosts files.

```
github.com:
  ssh_known_hosts:
    - present
    - user: root
    - fingerprint: 16:27:ac:a5:76:28:2d:36:63:1b:56:4d:eb:df:a6:48

example.com:
  ssh_known_hosts:
    - absent
    - user: root
```

`salt.states.ssh_known_hosts.`**`absent`**`(`*name*, *user*, *config='.ssh/known_hosts'*`)`

Verifies that the specified host is not known by the given user

name The host name

user The user who owns the ssh authorized keys file to modify

config The location of the authorized keys file relative to the user's home directory, defaults to ".ssh/known_hosts"

`salt.states.ssh_known_hosts.`**`present`**`(`*name*, *user*, *fingerprint=None*, *port=None*, *enc=None*, *config='.ssh/known_hosts'*, *hash_hostname=True*`)`

Verifies that the specified host is known by the specified user

On many systems, specifically those running with openssh 4 or older, the `enc` option must be set, only openssh 5 and above can detect the key type.

name The name of the remote host (e.g. "github.com")

user The user who owns the ssh authorized keys file to modify

enc Defines what type of key is being used, can be ecdsa ssh-rsa or ssh-dss

fingerprint The fingerprint of the key which must be presented in the known_hosts file

port optional parameter, denoting the port of the remote host, which will be used in case, if the public key will be requested from it. By default the port 22 is used.

config The location of the authorized keys file relative to the user's home directory, defaults to ".ssh/known_hosts"

hash_hostname [True] Hash all hostnames and addresses in the output.

81.77 salt.states.stateconf

81.77.1 Stateconf System

The stateconf system is intended for use only with the stateconf renderer. This State module presents the set function. This function does not execute any functionality, but is used to interact with the stateconf renderer.

`salt.states.stateconf.`**`context`**(*name*, ***kwargs*)
No-op state to support state config via the stateconf renderer.

`salt.states.stateconf.`**`set`**(*name*, ***kwargs*)
No-op state to support state config via the stateconf renderer.

81.78 salt.states.status

Minion status monitoring

Maps to the *status* execution module.

`salt.states.status.`**`loadavg`**(*name*, *maximum=None*, *minimum=None*)
Return the current load average for the specified minion. Available values for name are *1-min*, *5-min* and *15-min*. *minimum* and *maximum* values should be passed in as strings.

81.79 salt.states.supervisord

81.79.1 Interaction with the Supervisor daemon

```
wsgi_server:
  supervisord:
    - running
    - require:
      - pkg: supervisor
    - watch:
      - file: /etc/nginx/sites-enabled/wsgi_server.conf
```

`salt.states.supervisord.`**`dead`**(*name*, *user=None*, *runas=None*, *conf_file=None*, *bin_env=None*)
Ensure the named service is dead (not running).

name Service name as defined in the supervisor configuration file

runas Name of the user to run the supervisorctl command

Deprecated since version 0.17.0.

user Name of the user to run the supervisorctl command

New in version 0.17.0.

conf_file path to supervisorctl config file

bin_env path to supervisorctl bin or path to virtualenv with supervisor installed

`salt.states.supervisord.`**`mod_watch`**(*name*, *restart=True*, *update=False*, *user=None*, *runas=None*, *conf_file=None*, *bin_env=None*)

`salt.states.supervisord.`**`running`**(*name*, *restart=False*, *update=False*, *user=None*, *runas=None*, *conf_file=None*, *bin_env=None*)

Ensure the named service is running.

name Service name as defined in the supervisor configuration file

restart Whether to force a restart

update Whether to update the supervisor configuration.

runas Name of the user to run the supervisorctl command

Deprecated since version 0.17.0.

user Name of the user to run the supervisorctl command

New in version 0.17.0.

conf_file path to supervisorctl config file

bin_env path to supervisorctl bin or path to virtualenv with supervisor installed

81.80 salt.states.svn

81.80.1 Manage SVN repositories

Manage repositiry checkouts via the svn vcs system:

```
http://unladen-swallow.googlecode.com/svn/trunk/:
  svn.latest:
    - target: /tmp/swallow
```

`salt.states.svn.`**`dirty`**(*name*, *target*, *user=None*, *username=None*, *password=None*, *ignore_unversioned=False*)

Determine if the working directory has been changed.

`salt.states.svn.`**`export`**(*name*, *target=None*, *rev=None*, *user=None*, *username=None*, *password=None*, *force=False*, *externals=True*, *trust=False*)

Export a file or directory from an SVN repository

name Address and path to the file or directory to be exported.

target Name of the target directory where the checkout will put the working directory

rev [None] The name revision number to checkout. Enable "force" if the directory already exists.

user [None] Name of the user performing repository management operations

username [None] The user to access the name repository with. The svn default is the current user

password Connect to the Subversion server with this password

New in version 0.17.0.

force [False] Continue if conflicts are encountered

externals [True] Change to False to not checkout or update externals

trust [False] Automatically trust the remote server. SVN's –trust-server-cert `salt.states.svn.latest`(*name*, *target=None*, *rev=None*, *user=None*, *username=None*, *password=None*, *force=False*, *externals=True*, *trust=False*)

Checkout or update the working directory to the latest revision from the remote repository.

name Address of the name repository as passed to "svn checkout"

target Name of the target directory where the checkout will put the working directory

rev [None] The name revision number to checkout. Enable "force" if the directory already exists.

user [None] Name of the user performing repository management operations

username [None] The user to access the name repository with. The svn default is the current user

password Connect to the Subversion server with this password

New in version 0.17.0.

force [False] Continue if conflicts are encountered

externals [True] Change to False to not checkout or update externals

trust [False] Automatically trust the remote server. SVN's –trust-server-cert

81.81 salt.states.sysctl

81.81.1 Configuration of the Linux kernel using sysctrl

Control the kernel sysctl system.

```
vm.swappiness:
  sysctl.present:
    - value: 20
```

`salt.states.sysctl.present`(*name*, *value*, *config=None*)

Ensure that the named sysctl value is set in memory and persisted to the named configuration file. The default sysctl configuration file is /etc/sysctl.conf

name The name of the sysctl value to edit

value The sysctl value to apply

config The location of the sysctl configuration file. If not specified, the proper location will be detected based on platform.

81.82 salt.states.timezone

81.82.1 Management of timezones

The timezone can be managed for the system:

```
America/Denver:
  timezone.system
```

The system and the hardware clock are not necessarily set to the same time. By default, the hardware clock is set to localtime, meaning it is set to the same time as the system clock. If *utc* is set to True, then the hardware clock will be set to UTC, and the system clock will be an offset of that.

```
America/Denver:
  timezone.system:
    - utc: True
```

The Ubuntu community documentation contains an explanation of this setting, as it applies to systems that dual-boot with Windows. This is explained in greater detail here.

`salt.states.timezone.`**`system`**(*name*, *utc=''*)
Set the timezone for the system.

name The name of the timezone to use (e.g.: America/Denver)

utc Whether or not to set the hardware clock to UTC (default is True)

81.83 salt.states.tomcat

This state uses the manager webapp to manage Apache tomcat webapps This state requires the manager webapp to be enabled

The following grains/pillar should be set:

```
tomcat-manager.user: admin user name
tomcat-manager.passwd: password
```

and also configure a user in the conf/tomcat-users.xml file:

```
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
    <role rolename="manager-script"/>
    <user username="tomcat" password="tomcat" roles="manager-script"/>
</tomcat-users>
```

Notes:

- Not supported multiple version on the same context path
- **More information about tomcat manager:** http://tomcat.apache.org/tomcat-7.0-doc/manager-howto.html
- **if you use only this module for deployments you might want to restrict** access to the manager so its only accessible via localhost for more info: http://tomcat.apache.org/tomcat-7.0-doc/manager-howto.html#Configuring_Manager_Application_Access
- Tested on:

  JVM Vendor: Sun Microsystems Inc.

  JVM Version: 1.6.0_43-b01

  OS Architecture: amd64

  OS Name: Linux

  OS Version: 2.6.32-358.el6.x86_64

  Tomcat Version: Apache Tomcat/7.0.37

`salt.states.tomcat.`**`mod_watch`**(*name*, *url='http://localhost:8080/manager'*, *timeout=180*)
The tomcat watcher function. When called it will reload the webapp in question

`salt.states.tomcat.`**`wait`**(*name*, *url='http://localhost:8080/manager'*, *timeout=180*)
Wait for the tomcat manager to load

Notice that if tomcat is not running we won't wait for it start and the state will fail. This state can be required in the tomcat.war_deployed state to make sure tomcat is running and that the manager is running as well and ready for deployment

url [http://localhost:8080/manager] the URL of the server manager webapp

timeout [180] timeout for HTTP request to the tomcat manager

Example:

```
tomcat-service:
  service:
    - running
    - name: tomcat
    - enable: True

wait-for-tomcatmanager:
  tomcat:
    - wait
    - timeout: 300
    - require:
      - service: tomcat-service

jenkins:
  tomcat:
    - war_deployed
    - name: /ran
    - war: salt://jenkins-1.2.4.war
    - require:
      - tomcat: wait-for-tomcatmanager
```

`salt.states.tomcat.`**`war_deployed`**`(`*name*, *war*, *url='http://localhost:8080/manager'*, *timeout=180*`)`

Enforce that the WAR will be deployed and started in the context path it will make use of WAR versions

for more info: http://tomcat.apache.org/tomcat-7.0-doc/config/context.html#Naming

name the context path to deploy

war absolute path to WAR file (should be accessible by the user running tomcat) or a path supported by the salt.modules.cp.get_url function

url [http://localhost:8080/manager] the URL of the server manager webapp

timeout [180] timeout for HTTP request to the tomcat manager

Example:

```
jenkins:
  tomcat.war_deployed:
    - name: /ran
    - war: salt://jenkins-1.2.4.war
    - require:
      - service: application-service
```

81.84 salt.states.user

81.84.1 Management of user accounts

The user module is used to create and manage user settings, users can be set as either absent or present

```
fred:
  user.present:
    - fullname: Fred Jones
    - shell: /bin/zsh
    - home: /home/fred
    - uid: 4000
    - gid: 4000
    - groups:
      - wheel
      - storage
      - games

testuser:
  user.absent
```

`salt.states.user.`**`absent`**(*name*, *purge=False*, *force=False*)

Ensure that the named user is absent

name The name of the user to remove

purge Set purge to delete all of the user's files as well as the user

force If the user is logged in the absent state will fail, set the force option to True to remove the user even if they are logged in. Not supported in FreeBSD and Solaris.

`salt.states.user.`**`present`**(*name*, *uid=None*, *gid=None*, *gid_from_name=False*, *groups=None*, *optional_groups=None*, *remove_groups=True*, *home=None*, *createhome=True*, *password=None*, *enforce_password=True*, *shell=None*, *unique=True*, *system=False*, *fullname=None*, *roomnumber=None*, *workphone=None*, *homephone=None*)

Ensure that the named user is present with the specified properties

name The name of the user to manage

uid The user id to assign, if left empty then the next available user id will be assigned

gid The default group id

gid_from_name If True, the default group id will be set to the id of the group with the same name as the user.

groups A list of groups to assign the user to, pass a list object. If a group specified here does not exist on the minion, the state will fail. If set to the empty list, the user will be removed from all groups except the default group.

optional_groups A list of groups to assign the user to, pass a list object. If a group specified here does not exist on the minion, the state will silently ignore it.

NOTE: If the same group is specified in both "groups" and "optional_groups", then it will be assumed to be required and not optional.

remove_groups Remove groups that the user is a member of that weren't specified in the state, True by default

home The location of the home directory to manage

createhome If True, the home directory will be created if it doesn't exist. Please note that directories leading up to the home directory will NOT be created.

password A password hash to set for the user. This field is only supported on Linux, FreeBSD, NetBSD, OpenBSD, and Solaris.

Changed in version 0.16.0: BSD support added.

enforce_password Set to False to keep the password from being changed if it has already been set and the password hash differs from what is specified in the "password" field. This option will be ignored if "password" is not specified.

shell The login shell, defaults to the system default shell

unique Require a unique UID, True by default

system Choose UID in the range of FIRST_SYSTEM_UID and LAST_SYSTEM_UID.

User comment field (GECOS) support (currently Linux, FreeBSD, and MacOS only):

The below values should be specified as strings to avoid ambiguities when the values are loaded. (Especially the phone and room number fields which are likely to contain numeric data)

fullname The user's full name

roomnumber The user's room number (not supported in MacOS)

workphone The user's work phone number (not supported in MacOS)

homephone The user's home phone number (not supported in MacOS)

81.85 salt.states.virtualenv

81.85.1 Setup of Python virtualenv sandboxes

`salt.states.virtualenv_mod.`**`managed`**`(`*name, venv_bin=None, requirements=None, no_site_packages=None, system_site_packages=False, distribute=False, use_wheel=False, clear=False, python=None, extra_search_dir=None, never_download=None, prompt=None, user=None, runas=None, no_chown=False, cwd=None, index_url=None, extra_index_url=None, pre_releases=False, no_deps=False, proxy=None*`)`

Create a virtualenv and optionally manage it with pip

name Path to the virtualenv

requirements Path to a pip requirements file. If the path begins with `salt://` the file will be transferred from the master file server.

cwd Path to the working directory where "pip install" is executed.

use_wheel [False] Prefer wheel archives (requires pip>=1.4)

no_deps: False Pass *–no-deps* to *pip*.

proxy: None Proxy address which is passed to "pip install"

Also accepts any kwargs that the virtualenv module will.

```
/var/www/myvirtualenv.com:
  virtualenv.managed:
    - system_site_packages: False
    - requirements: salt://REQUIREMENTS.txt
```

81.86 salt.states.win_dns_client

Module for configuring DNS Client on Windows systems

`salt.states.win_dns_client.`**`dns_dhcp`**(*name*, *interface='Local Area Connection'*)

Configure the DNS server list from DHCP Server

`salt.states.win_dns_client.`**`dns_exists`**(*name*, *servers=None*, *interface='Local Area Connection'*)

Configure the DNS server list in the specified interface

Example:

```
config_dns_servers:
  win_dns_client.dns_exists:
    - servers:
      - 8.8.8.8
      - 8.8.8.9
```

81.87 salt.states.win_firewall

State for configuring Windows Firewall

`salt.states.win_firewall.`**`disabled`**(*name*)

Disable all the firewall profiles (Windows only)

81.88 salt.states.win_network

81.88.1 Configuration of network interfaces on Windows hosts

New in version 2014.1.0: (Hydrogen)

This module provides the `network` state(s) on Windows hosts. DNS servers, IP addresses and default gateways can currently be managed.

Below is an example of the configuration for an interface that uses DHCP for both DNS servers and IP addresses:

```
Local Area Connection #2:
  network.managed:
    - dns_proto: dhcp
    - ip_proto: dhcp
```

Note: Both the `dns_proto` and `ip_proto` arguments are required.

Static DNS and IP addresses can be configured like so:

```
Local Area Connection #2:
  network.managed:
    - dns_proto: static
    - dns_servers:
      - 8.8.8.8
      - 8.8.4.4
    - ip_proto: static
    - ip_addrs:
      - 10.2.3.4/24
```

Note: IP addresses are specified using the format `<ip-address>/<subnet-length>`. Salt provides a convenience function called `ip.get_subnet_length` to calculate the subnet length from a netmask.

Optionally, if you are setting a static IP address, you can also specify the default gateway using the `gateway` parameter:

```
Local Area Connection #2:
  network.managed:
    - dns_proto: static
    - dns_servers:
      - 8.8.8.8
      - 8.8.4.4
    - ip_proto: static
    - ip_addrs:
      - 10.2.3.4/24
    - gateway: 10.2.3.1
```

`salt.states.win_network.`**`managed`**`(`*name*, *dns_proto=None*, *dns_servers=None*, *ip_proto=None*, *ip_addrs=None*, *gateway=None*, *enabled=True*, ***kwargs*`)`

Ensure that the named interface is configured properly.

name The name of the interface to manage

dns_proto [None] Set to `static` and use the `dns_servers` parameter to provide a list of DNS nameservers. set to `dhcp` to use DHCP to get the DNS servers.

dns_servers [None] A list of static DNS servers.

ip_proto [None] Set to `static` and use the `ip_addrs` and (optionally) `gateway` parameters to provide a list of static IP addresses and the default gateway. Set to `dhcp` to use DHCP.

ip_addrs [None] A list of static IP addresses.

gateway [None] A list of static IP addresses.

enabled [True] Set to `False` to ensure that this interface is disabled.

81.89 salt.states.win_path

Manage the Windows System PATH

`salt.states.win_path.`**`absent`**`(`*name*`)`

Remove the directory from the SYSTEM path

index: where the directory should be placed in the PATH (default: 0)

Example:

```
'C:\sysinternals':
  win_path.absent
```

`salt.states.win_path.`**`exists`**`(`*name*, *index=None*`)`

Add the directory to the system PATH at index location

index: where the directory should be placed in the PATH (default: None) [Note: Providing no index will append directory to PATH and will not enforce its location within the PATH.]

Example:

```
'C:\python27':
  win_path.exists

'C:\sysinternals':
  win_path.exists:
    index: 0
```

81.90 salt.states.win_servermanager

Manage Windows features via the ServerManager powershell module

`salt.states.win_servermanager.`**`installed`**(*name*, *recurse=False*, *force=False*)

Install the windows feature

name: short name of the feature (the right column in win_servermanager.list_available)

recurse: install all sub-features as well

force: if the feature is installed but on of its sub-features are not installed set this to True to force the installation of the sub-features

Note: Some features requires reboot after un/installation, if so until the server is restarted Other features can not be installed !

81.91 salt.states.win_system

81.91.1 Management of Windows system information

New in version 2014.1.0: (Hydrogen)

This state is used to manage system information such as the computer name and description.

```
ERIK-WORKSTATION:
  system:
    - computer_name

This is Erik's computer, don't touch!:
  system:
    - computer_desc
```

`salt.states.win_system.`**`computer_desc`**(*name*)

Manage the computer's description field

name The desired computer description

`salt.states.win_system.`**`computer_name`**(*name*)

Manage the computer's name

name The desired computer name

81.92 salt.states.xmpp

81.92.1 Sending Messages over XMPP

New in version 2014.1.0: (Hydrogen)

This state is useful for firing messages during state runs, using the XMPP protocol

```
server-warning-message:
  xmpp.send_msg:
    - name: 'This is a server warning message'
    - profile: my-xmpp-account
    - recipient: admins@xmpp.example.com/salt
```

`salt.states.xmpp.`**`send_msg`**(*name*, *recipient*, *profile*)

Send a message to an XMPP user

```
server-warning-message:
  xmpp.send_msg:
    - name: 'This is a server warning message'
    - profile: my-xmpp-account
    - recipient: admins@xmpp.example.com/salt
```

name The message to send to the XMPP user

81.93 salt.states.zcbuildout

81.93.1 Management of zc.buildout

This module is inspired from minitage's buildout maker (https://github.com/minitage/minitage/blob/master/src/minitage/core/makers/bui

New in version Boron.

Note: This state module is beta; the API is subject to change and no promise as to performance or functionality is yet present

---

Available Functions

- built

```
installed1
  buildout.installed:
    - name: /path/to/buildout

installed2
  buildout.installed:
    - name: /path/to/buildout
    - parts:
      - a
      - b
    - python: /path/to/pythonpath/bin/python
    - unless: /bin/test_something_installed
    - onlyif: /bin/test_else_installed
```

`salt.states.zcbuildout.`**`installed`**(*name*, *config='buildout.cfg'*, *quiet=False*, *parts=None*, *runas=None*, *user=None*, *env=()*, *buildout_ver=None*, *test_release=False*, *distribute=None*, *new_st=None*, *offline=False*, *newest=False*, *python='/home/docs/bin/python2.7'*, *debug=False*, *verbose=False*, *unless=None*, *onlyif=None*)

Install buildout in a specific directory

It is a thin wrapper to modules.buildout.buildout

name directory to execute in

quiet

do not output console & logs

config buildout config to use (default: buildout.cfg)

parts specific buildout parts to run

runas user used to run buildout as

Deprecated since version 2014.1.4: (Hydrogen)

user user used to run buildout as

New in version 2014.1.4: (Hydrogen)

env environment variables to set when running

buildout_ver force a specific buildout version (1 | 2)

test_release buildout accept test release

new_st Forcing use of setuptools >= 0.7

distribute use distribute over setuptools if possible

offline does buildout run offline

python python to use

debug run buildout with -D debug flag

onlyif Only execute cmd if statement on the host return 0

unless Do not execute cmd if statement on the host return 0

newest run buildout in newest mode

verbose run buildout in verbose mode (-vvvvv)

CHAPTER 82

Renderers

The Salt state system operates by gathering information from simple data structures. The state system was designed in this way to make interacting with it generic and simple. This also means that state files (SLS files) can be one of many formats.

By default SLS files are rendered as Jinja templates and then parsed as YAML documents. But since the only thing the state system cares about is raw data, the SLS files can be any structured format that can be dreamed up.

Currently there is support for `Jinja + YAML`, `Mako + YAML`, `Wempy + YAML`, `Jinja + json` `Mako + json` and `Wempy + json`. But renderers can be written to support anything. This means that the Salt states could be managed by XML files, HTML files, puppet files, or any format that can be translated into the data structure used by the state system.

82.1 Multiple Renderers

When deploying a state tree a default renderer is selected in the master configuration file with the renderer option. But multiple renderers can be used inside the same state tree.

When rendering SLS files Salt checks for the presence of a Salt specific shebang line. The shebang line syntax was chosen because it is familiar to the target audience, the systems admin and systems engineer.

The shebang line directly calls the name of the renderer as it is specified within Salt. One of the most common reasons to use multiple renderers in to use the Python or `py` renderer:

```
#!py

def run():
    '''
    Install the python-mako package
    '''
    return {'include': ['python'],
            'python-mako': {'pkg': ['installed']}}
```

The first line is a shebang that references the `py` renderer.

82.2 Composing Renderers

A renderer can be composed from other renderers by connecting them in a series of pipes(`|`). In fact, the default `Jinja + YAML` renderer is implemented by combining a YAML renderer and a Jinja renderer. Such renderer configuration is specified as: `jinja | yaml`.

Other renderer combinations are possible, here's a few examples:

`yaml` i.e, just YAML, no templating.

`mako | yaml` pass the input to the `mako` renderer, whose output is then fed into the `yaml` renderer.

`jinja | mako | yaml` This one allows you to use both jinja and mako templating syntax in the input and then parse the final rendered output as YAML.

And here's a contrived example sls file using the `jinja | mako | yaml` renderer:

```
#!jinja|mako|yaml

An_Example:
  cmd.run:
    - name: |
        echo "Using Salt ${grains['saltversion']}" \
             "from path {{grains['saltpath']}}."
    - cwd: /

<%doc> ${...} is Mako's notation, and so is this comment. </%doc>
{#     Similarly, {{...}} is Jinja's notation, and so is this comment. #}
```

For backward compatibility, `jinja | yaml` can also be written as `yaml_jinja`, and similarly, the `yaml_mako`, `yaml_wempy`, `json_jinja`, `json_mako`, and `json_wempy` renderers are all supported as well.

Keep in mind that not all renderers can be used alone or with any other renderers. For example, the template renderers shouldn't be used alone as their outputs are just strings, which still need to be parsed by another renderer to turn them into highstate data structures. Also, for example, it doesn't make sense to specify `yaml | jinja` either, because the output of the yaml renderer is a highstate data structure(a dict in Python), which cannot be used as the input to a template renderer. Therefore, when combining renderers, you should know what each renderer accepts as input and what it returns as output.

82.3 Writing Renderers

Writing a renderer is easy, all that is required is that a Python module is placed in the rendered directory and that the module implements the `render` function. The `render` function will be passed the path of the SLS file. In the `render` function, parse the passed file and return the data structure derived from the file. You can place your custom renderers in a `_renderers` directory within the `file_roots` specified by the master config file. These custom renderers are distributed when `state.highstate` is run, or by executing the `saltutil.sync_renderers` or `saltutil.sync_all` functions.

Any custom renderers which have been synced to a minion, that are named the same as one of Salt's default set of renderers, will take the place of the default renderer with the same name.

82.4 Examples

The best place to find examples of renderers is in the Salt source code. The renderers included with Salt can be found here:

https://github.com/saltstack/salt/blob/develop/salt/renderers

Here is a simple YAML renderer example:

```
import yaml
def render(yaml_data, env='', sls='', **kws):
    if not isinstance(yaml_data, basestring):
        yaml_data = yaml_data.read()
    data = yaml.load(yaml_data)
    return data if data else {}
```

Full list of builtin renderer modules

`jinja`	
`json`	
`mako`	
`py`	Pure python state renderer
`pydsl`	A Python-based DSL
`stateconf`	A flexible renderer that takes a templating engine and a data format
`wempy`	
`yaml`	

83.1 salt.renderers.jinja

83.1.1 Jinja in States

The most basic usage of Jinja in state files is using control structures to wrap conditional or redundant state elements:

```
{% if grains['os'] != 'FreeBSD' %}
tcsh:
    pkg:
        - installed
{% endif %}

motd:
  file.managed:
    {% if grains['os'] == 'FreeBSD' %}
    - name: /etc/motd
    {% elif grains['os'] == 'Debian' %}
    - name: /etc/motd.tail
    {% endif %}
    - source: salt://motd
```

In this example, the first if block will only be evaluated on minions that aren't running FreeBSD, and the second block changes the file name based on the *os* grain.

Writing **if-else** blocks can lead to very redundant state files however. In this case, using *pillars*, or using a previously defined variable might be easier:

```
{% set motd = ['/etc/motd'] %}
{% if grains['os'] == 'Debian' %}
  {% set motd = ['/etc/motd.tail', '/var/run/motd'] %}
{% endif %}
{% for motdfile in motd %}
{{ motdfile }}:
  file.managed:
    - source: salt://motd
{% endfor %}
```

Using a variable set by the template, the for loop will iterate over the list of MOTD files to update, adding a state block for each file.

83.1.2 Passing Variables

It is also possible to pass additional variable context directly into a template, using the `defaults` and `context` mappings of the *file.managed* state:

```
/etc/motd:
  file.managed:
    - source: salt://motd
    - template: jinja
    - defaults:
        message: 'Foo'
    {% if grains['os'] == 'FreeBSD' %}
    - context:
        message: 'Bar'
    {% endif %}
```

The template will receive a variable `message`, which would be accessed in the template using `{{ message }}`. If the operating system is FreeBSD, the value of the variable `message` would be *Bar*, otherwise it is the default *Foo*

83.1.3 Include and Import

Includes and imports can be used to share common, reusable state configuration between state files and between files.

```
{% from 'lib.sls' import test %}
```

This would import the `test` template variable or macro, not the `test` state element, from the file `lib.sls`. In the case that the included file performs checks again grains, or something else that requires context, passing the context into the included file is required:

```
{% from 'lib.sls' import test with context %}
```

83.1.4 Variable and block Serializers

Salt allows one to serialize any variable into **json** or **yaml** or **python**. For example this variable:

```
data:
  foo: True
  bar: 42
  baz:
    - 1
    - 2
    - 3
  qux: 2.0
```

with this template:

```
yaml -> {{ data|yaml }}

json -> {{ data|json }}

python -> {{ data|python }}
```

will be rendered has:

```
yaml -> {bar: 42, baz: [1, 2, 3], foo: true, qux: 2.0}

json -> {"baz": [1, 2, 3], "foo": true, "bar": 42, "qux": 2.0}

python -> {u'data': {u'bar': 42, u'baz': [1, 2, 3], u'foo': True, u'qux': 2.0}}
```

Strings and variables can be deserialized with **load_yaml** and **load_json** tags and filters. It allows one to manipulate data directly in templates, easily:

```
{%- set json_var = '{"foo": "bar", "baz": "qux"}'|load_json %}
My json_var foo is {{ json_var.foo }}

{%- set yaml_var = "{bar: baz: qux}"|load_yaml %}
My yaml_var bar.baz is {{ yaml_var.bar.baz }}

{%- load_json as json_block %}
  {
    "qux": {{ yaml_var|json }},
  }
{% endload %}
My json_block qux.bar.baz is {{ json_block.qux.bar.baz }}

{%- load_yaml as yaml_block %}
  bar:
    baz:
      qux
{% endload %}
My yaml_block bar.baz is {{ yaml2.bar.baz }}
```

will be rendered has:

```
My json_var foo is bar

My yaml_var bar.baz is qux

My json_block foo is quz

My yaml_block bar.baz is qux
```

83.1.5 Template Serializers

Salt implements **import_yaml** and **import_json** tags. They work like the import tag, except that the document is also deserialized.

Imagine you have a generic state file in which you have the complete data of your infrastucture:

```
# everything.sls
users:
  foo:
    - john
  bar:
    - bob
  baz:
    - smith
```

But you don't want to expose everything to a minion. This state file:

```
# specialized.sls
{% import_yaml "everything.sls" as all %}
my_admins:
  my_foo: {{ all.users.foo|yaml }}
```

will be rendered has:

```
my_admins:
  my_foo: [john]
```

83.1.6 Macros

Macros are helpful for eliminating redundant code, however stripping whitespace from the template block, as well as contained blocks, may be necessary to emulate a variable return from the macro.

```
# init.sls
{% from 'lib.sls' import pythonpkg with context %}

python-virtualenv:
  pkg.installed:
    - name: {{ pythonpkg('virtualenv') }}

python-fabric:
  pkg.installed:
    - name: {{ pythonpkg('fabric') }}

# lib.sls
{% macro pythonpkg(pkg) -%}
  {%- if grains['os'] == 'FreeBSD' -%}
    py27-{{ pkg }}
  {%- elif grains['os'] == 'Debian' -%}
    python-{{ pkg }}
  {%- endif -%}
{%- endmacro %}
```

This would define a macro that would return a string of the full package name, depending on the packaging system's naming convention. The whitespace of the macro was eliminated, so that the macro would return a string without line breaks, using whitespace control.

83.1.7 Template Inheritance

Template inheritance works fine from state files and files. The search path starts at the root of the state tree or pillar.

83.1.8 Filters

Saltstack extends builtin filters with his custom filters:

strftime Converts any time related object into a time based string. It requires a valid *strftime directives*. An *exhaustive list* can be found in the official Python documentation. Fuzzy dates are parsed by timelib python module. Some examples are available on this pages.

```
{{ "2002/12/25"|strftime("%y") }}
{{ "1040814000"|strftime("%Y-%m-%d") }}
{{ datetime|strftime("%u") }}
{{ "now"|strftime }}
```

83.1.9 Jinja in Files

Jinja can be used in the same way in managed files:

```
# redis.sls
/etc/redis/redis.conf:
    file.managed:
        - source: salt://redis.conf
        - template: jinja
        - context:
            bind: 127.0.0.1

# lib.sls
{% set port = 6379 %}

# redis.conf
{% from 'lib.sls' import port with context %}
port {{ port }}
bind {{ bind }}
```

As an example, configuration was pulled from the file context and from an external template file.

Note: Macros and variables can be shared across templates. They should not be starting with one or more underscores, and should be managed by one of the following tags: *macro*, *set*, *load_yaml*, *load_json*, *import_yaml* and *import_json*.

`salt.renderers.jinja.`**`render`**(*template_file*, *saltenv='base'*, *sls=''*, *argline=''*, *context=None*, *tmplpath=None*, ***kws*)

Render the template_file, passing the functions and grains into the Jinja rendering system.

Return type string

83.2 salt.renderers.json

`salt.renderers.json.`**`render`**(*json_data*, *saltenv='base'*, *sls=''*, ***kws*)

Accepts JSON as a string or as a file object and runs it through the JSON parser.

Return type A Python data structure

83.3 salt.renderers.mako

`salt.renderers.mako.`**`render`**(*template_file*, *saltenv='base'*, *sls=''*, *context=None*, *tmplpath=None*, ***kws*)

Render the template_file, passing the functions and grains into the Mako rendering system.

Return type string

83.4 salt.renderers.py

Pure python state renderer

The sls file should contain a function called `run` which returns high state data

In this module, a few objects are defined for you, including the usual (with‘‘__‘‘ added) `__salt__` dictionary, `__grains__`, `__pillar__`, `__opts__`, `__env__`, and `__sls__`.

```
#!py

def run():
    config = {}

    if __grains__['os'] == 'Ubuntu':
        user = 'ubuntu'
        group = 'ubuntu'
        home = '/home/{0}'.format(user)
    else:
        user = 'root'
        group = 'root'
        home = '/root/'

    config['s3cmd'] = {
        'pkg': [
            'installed',
            {'name': 's3cmd'},
        ],
    }

    config[home + '/.s3cfg'] = {
        'file.managed': [
            {'source': 'salt://s3cfg/templates/s3cfg'},
            {'template': 'jinja'},
            {'user': user},
            {'group': group},
            {'mode': 600},
            {'context': {
                'aws_key': __pillar__['AWS_ACCESS_KEY_ID'],
                'aws_secret_key': __pillar__['AWS_SECRET_ACCESS_KEY'],
                },
            },
        ],
    }

    return config
```

`salt.renderers.py.`**`render`**(*template*, *saltenv='base'*, *sls=’‘*, *tmplpath=None*, ***kws*)

Render the python module’s components

Return type string

83.5 salt.renderers.pydsl

A Python-based DSL

maintainer Jack Kuan <kjkuan@gmail.com>

maturity new

platform all

The *pydsl* renderer allows one to author salt formulas(.sls files) in pure Python using a DSL that's easy to write and easy to read. Here's an example:

```
#!pydsl

apache = state('apache')
apache.pkg.installed()
apache.service.running()
state('/var/www/index.html') \
    .file('managed',
          source='salt://webserver/index.html') \
    .require(pkg='apache')
```

Notice that any Python code is allow in the file as it's really a Python module, so you have the full power of Python at your disposal. In this module, a few objects are defined for you, including the usual(with __ added) `__salt__` dictionary, `__grains__`, `__pillar__`, `__opts__`, `__env__`, and `__sls__`, plus a few more:

`__file__`

local file system path to the sls module.

`__pydsl__`

Salt PyDSL object, useful for configuring DSL behavior per sls rendering.

`include`

Salt PyDSL function for creating *include declaration*'s.

`extend`

Salt PyDSL function for creating *extend declaration*'s.

`state`

Salt PyDSL function for creating *ID declaration*'s.

A state *ID declaration* is created with a `state(id)` function call. Subsequent `state(id)` call with the same id returns the same object. This singleton access pattern applies to all declaration objects created with the DSL.

```
state('example')
assert state('example') is state('example')
assert state('example').cmd is state('example').cmd
assert state('example').cmd.running is state('example').cmd.running
```

The *id* argument is optional. If omitted, an UUID will be generated and used as the *id*.

`state(id)` returns an object under which you can create a *state declaration* object by accessing an attribute named after *any* state module available in Salt.

```
state('example').cmd
state('example').file
state('example').pkg
...
```

Then, a *function declaration* object can be created from a *state declaration* object by one of the following two ways:

1. by directly calling the attribute named for the *state declaration*, and supplying the state function name as the first argument.

```
state('example').file('managed', ...)
```

2. by calling a method named after the state function on the *state declaration* object.

```
state('example').file.managed(...)
```

With either way of creating a *function declaration* object, any *function arg declaration*'s can be passed as keyword arguments to the call. Subsequent calls of a *function declaration* will update the arg declarations.

```
state('example').file('managed', source='salt://webserver/index.html')
state('example').file.managed(source='salt://webserver/index.html')
```

As a shortcut, the special *name* argument can also be passed as the first(second if calling using the first way) positional argument.

```
state('example').cmd('run', 'ls -la', cwd='/')
state('example').cmd.run('ls -la', cwd='/')
```

Finally, a *requisite declaration* object with its *requisite reference*'s can be created by invoking one of the requisite methods(`require`, `watch`, `use`, `require_in`, `watch_in`, and `use_in`) on either a *function declaration* object or a *state declaration* object. The return value of a requisite call is also a *function declaration* object, so you can chain several requisite calls together.

Arguments to a requisite call can be a list of *state declaration* objects and/or a set of keyword arguments whose names are state modules and values are IDs of *ID declaration*'s or names of *name declaration*'s.

```
apache2 = state('apache2')
apache2.pkg.installed()
state('libapache2-mod-wsgi').pkg.installed()

# you can call requisites on function declaration
apache2.service.running() \
               .require(apache2.pkg,
                        pkg='libapache2-mod-wsgi') \
               .watch(file='/etc/apache2/httpd.conf')

# or you can call requisites on state declaration.
# this actually creates an anonymous function declaration object
# to add the requisites.
apache2.service.require(state('libapache2-mod-wsgi').pkg,
                        pkg='apache2') \
               .watch(file='/etc/apache2/httpd.conf')

# we still need to set the name of the function declaration.
apache2.service.running()
```

include declaration objects can be created with the `include` function, while *extend declaration* objects can be created with the `extend` function, whose arguments are just *function declaration* objects.

```
include('edit.vim', 'http.server')
extend(state('apache2').service.watch(file='/etc/httpd/httpd.conf')
```

The `include` function, by default, causes the included sls file to be rendered as soon as the `include` function is called. It returns a list of rendered module objects; sls files not rendered with the pydsl renderer return `None`'s. This behavior creates no *include declaration*'s in the resulting high state data structure.

```
import types

# including multiple sls returns a list.
_, mod = include('a-non-pydsl-sls', 'a-pydsl-sls')
```

```
assert _ is None
assert isinstance(slsmods[1], types.ModuleType)

# including a single sls returns a single object
mod = include('a-pydsl-sls')

# myfunc is a function that calls state(...) to create more states.
mod.myfunc(1, 2, "three")
```

Notice how you can define a reusable function in your pydsl sls module and then call it via the module returned by `include`.

It's still possible to do late includes by passing the `delayed=True` keyword argument to `include`.

```
include('edit.vim', 'http.server', delayed=True)
```

Above will just create a *include declaration* in the rendered result, and such call always returns `None`.

83.5.1 Special integration with the *cmd* state

Taking advantage of rendering a Python module, PyDSL allows you to declare a state that calls a pre-defined Python function when the state is executed.

```
greeting = "hello world"
def helper(something, *args, **kws):
    print greeting                # hello world
    print something, args, kws    # test123 ['a', 'b', 'c'] {'x': 1, 'y': 2}

state().cmd.call(helper, "test123", 'a', 'b', 'c', x=1, y=2)
```

The *cmd.call* state function takes care of calling our `helper` function with the arguments we specified in the states, and translates the return value of our function into a structure expected by the state system. See `salt.states.cmd.call()` for more information.

83.5.2 Implicit ordering of states

Salt states are explicitly ordered via *requisite declaration*'s. However, with *pydsl* it's possible to let the renderer track the order of creation for *function declaration* objects, and implicitly add `require` requisites for your states to enforce the ordering. This feature is enabled by setting the `ordered` option on `__pydsl__`.

Note: this feature is only available if your minions are using Python >= 2.7.

```
include('some.sls.file')

A = state('A').cmd.run(cwd='/var/tmp')
extend(A)

__pydsl__.set(ordered=True)

for i in range(10):
    i = str(i)
    state(i).cmd.run('echo '+i, cwd='/')
state('1').cmd.run('echo one')
state('2').cmd.run(name='echo two')
```

Notice that the `ordered` option needs to be set after any `extend` calls. This is to prevent *pydsl* from tracking the creation of a state function that's passed to an `extend` call.

Above example should create states from `0` to `9` that will output `0`, `one`, `two`, `3`, ... `9`, in that order.

It's important to know that *pydsl* tracks the *creations* of *function declaration* objects, and automatically adds a `require` requisite to a *function declaration* object that requires the last *function declaration* object created before it in the sls file.

This means later calls(perhaps to update the function's *function arg declaration*) to a previously created function declaration will not change the order.

83.5.3 Render time state execution

When Salt processes a salt formula file(*.sls*), the file is rendered to salt's high state data representation by a renderer before the states can be executed. In the case of the *pydsl* renderer, the .sls file is executed as a python module as it is being rendered which makes it easy to execute a state at render time. In *pydsl*, executing one or more states at render time can be done by calling a configured *ID declaration* object.

```
#!pydsl

s = state() # save for later invocation

# configure it
s.cmd.run('echo at render time', cwd='/')
s.file.managed('target.txt', source='salt://source.txt')

s() # execute the two states now
```

Once an *ID declaration* is called at render time it is detached from the sls module as if it was never defined.

Note: If *implicit ordering* is enabled(ie, via `__pydsl__.set(ordered=True)`) then the *first* invocation of a *ID declaration* object must be done before a new *function declaration* is created.

83.5.4 Integration with the stateconf renderer

The *salt.renderers.stateconf* renderer offers a few interesting features that can be leveraged by the *pydsl* renderer. In particular, when using with the *pydsl* renderer, we are interested in *stateconf*'s sls namespacing feature(via dot-prefixed id declarations), as well as, the automatic *start* and *goal* states generation.

Now you can use *pydsl* with *stateconf* like this:

```
#!pydsl|stateconf -ps

include('xxx', 'yyy')

# ensure that states in xxx run BEFORE states in this file.
extend(state('.start').stateconf.require(stateconf='xxx::goal'))

# ensure that states in yyy run AFTER states in this file.
extend(state('.goal').stateconf.require_in(stateconf='yyy::start'))

__pydsl__.set(ordered=True)

...
```

`-s` enables the generation of a stateconf *start* state, and `-p` lets us pipe high state data rendered by *pydsl* to *stateconf*. This example shows that by `require`-ing or `require_in`-ing the included sls' *start* or *goal* states, it's possible to ensure that the included sls files can be made to execute before or after a state in the including sls file.

`salt.renderers.pydsl.`**`render`**(*template*, *saltenv='base'*, *sls=''*, *tmplpath=None*, *rendered_sls=None*, ***kws*)

83.6 salt.renderers.stateconf

maintainer Jack Kuan <kjkuan@gmail.com>

maturity new

platform all

This module provides a custom renderer that processes a salt file with a specified templating engine (e.g., Jinja) and a chosen data renderer (e.g., YAML), extracts arguments for any `stateconf.set` state, and provides the extracted arguments (including Salt-specific args, such as `require`, etc) as template context. The goal is to make writing reusable/configurable/parameterized salt files easier and cleaner.

To use this renderer, either set it as the default renderer via the `renderer` option in master/minion's config, or use the shebang line in each individual sls file, like so: `#!stateconf`. Note, due to the way this renderer works, it must be specified as the first renderer in a render pipeline. That is, you cannot specify `#!mako|yaml|stateconf`, for example. Instead, you specify them as renderer arguments: `#!stateconf mako . yaml`.

Here's a list of features enabled by this renderer.

- Prefixes any state id (declaration or reference) that starts with a dot (`.`) to avoid duplicated state ids when the salt file is included by other salt files.

  For example, in the *salt://some/file.sls*, a state id such as `.sls_params` will be turned into `some.file::sls_params`. Example:

  ```
  #!stateconf yaml . jinja

  .vim:
    pkg.installed
  ```

  Above will be translated into:

  ```
  some.file::vim:
    pkg.installed:
      - name: vim
  ```

  Notice how that if a state under a dot-prefixed state id has no `name` argument then one will be added automatically by using the state id with the leading dot stripped off.

  The leading dot trick can be used with extending state ids as well, so you can include relatively and extend relatively. For example, when extending a state in *salt://some/other_file.sls*, e.g.,:

  ```
  #!stateconf yaml . jinja

  include:
    - .file

  extend:
    .file::sls_params:
      stateconf.set:
        - name1: something
  ```

Above will be pre-processed into:

```
include:
  - some.file

extend:
  some.file::sls_params:
    stateconf.set:
      - name1: something
```

- Adds a `sls_dir` context variable that expands to the directory containing the rendering salt file. So, you can write `salt://{{sls_dir}}/...` to reference templates files used by your salt file.
- Recognizes the special state function, `stateconf.set`, that configures a default list of named arguments usable within the template context of the salt file. Example:

```
#!stateconf yaml . jinja

.sls_params:
  stateconf.set:
    - name1: value1
    - name2: value2
    - name3:
      - value1
      - value2
      - value3
    - require_in:
      - cmd: output

# --- end of state config ---

.output:
  cmd.run:
    - name: |
        echo 'name1={{sls_params.name1}}
              name2={{sls_params.name2}}
              name3[1]={{sls_params.name3[1]}}
        '
```

This even works with `include` + `extend` so that you can override the default configured arguments by including the salt file and then `extend` the `stateconf.set` states that come from the included salt file. (*IMPORTANT: Both the included and the extending sls files must use the stateconf renderer for this ‘‘extend‘‘ to work!*)

Notice that the end of configuration marker (`# --- end of state config --`) is needed to separate the use of 'stateconf.set' form the rest of your salt file. The regex that matches such marker can be configured via the `stateconf_end_marker` option in your master or minion config file.

Sometimes, you'd like to set a default argument value that's based on earlier arguments in the same `stateconf.set`. For example, you may be tempted to do something like this:

```
#!stateconf yaml . jinja

.apache:
  stateconf.set:
    - host: localhost
    - port: 1234
    - url: 'http://{{host}}:{{port}}/'

# --- end of state config ---
```

```
.test:
  cmd.run:
    - name: echo '{{apache.url}}'
    - cwd: /
```

However, this won't work, but can be worked around like so:

```
#!stateconf yaml . jinja

.apache:
  stateconf.set:
    - host: localhost
    - port: 1234
{#  - url: 'http://{{host}}:{{port}}/' #}

# --- end of state config ---
# {{ apache.setdefault('url', "http://%(host)s:%(port)s/" % apache) }}

.test:
  cmd.run:
    - name: echo '{{apache.url}}'
    - cwd: /
```

- Adds support for relative include and exclude of .sls files. Example:

  ```
  #!stateconf yaml . jinja

  include:
    - .apache
    - .db.mysql

  exclude:
    - sls: .users
  ```

  If the above is written in a salt file at *salt://some/where.sls* then it will include *salt://some/apache.sls* and *salt://some/db/mysql.sls*, and exclude *salt://some/users.ssl*. Actually, it does that by rewriting the above `include` and `exclude` into:

  ```
  include:
    - some.apache
    - some.db.mysql

  exclude:
    - sls: some.users
  ```

- Optionally (enabled by default, *disable* via the *-G* renderer option, e.g., in the shebang line: `#!stateconf -G`), generates a `stateconf.set` goal state (state id named as `.goal` by default, configurable via the master/minion config option, `stateconf_goal_state`) that requires all other states in the salt file. Note, the `.goal` state id is subject to dot-prefix rename rule mentioned earlier.

  Such goal state is intended to be required by some state in an including salt file. For example, in your webapp salt file, if you include a sls file that is supposed to setup Tomcat, you might want to make sure that all states in the Tomcat sls file will be executed before some state in the webapp sls file.

- Optionally (enable via the *-o* renderer option, e.g., in the shebang line: `#!stateconf -o`), orders the states in a sls file by adding a `require` requisite to each state such that every state requires the state defined just before it. The order of the states here is the order they are defined in the sls file. (Note: this feature is only available if your minions are using Python >= 2.7. For Python2.6, it should also work if you install the *ordereddict* module from PyPI)

By enabling this feature, you are basically agreeing to author your sls files in a way that gives up the explicit (or implicit?) ordering imposed by the use of `require`, `watch`, `require_in` or `watch_in` requisites, and instead, you rely on the order of states you define in the sls files. This may or may not be a better way for you. However, if there are many states defined in a sls file, then it tends to be easier to see the order they will be executed with this feature.

You are still allowed to use all the requisites, with a few restrictions. You cannot `require` or `watch` a state defined *after* the current state. Similarly, in a state, you cannot `require_in` or `watch_in` a state defined *before* it. Breaking any of the two restrictions above will result in a state loop. The renderer will check for such incorrect uses if this feature is enabled.

Additionally, `names` declarations cannot be used with this feature because the way they are compiled into low states make it impossible to guarantee the order in which they will be executed. This is also checked by the renderer. As a workaround for not being able to use `names`, you can achieve the same effect, by generate your states with the template engine available within your sls file.

Finally, with the use of this feature, it becomes possible to easily make an included sls file execute all its states *after* some state (say, with id `X`) in the including sls file. All you have to do is to make state, `X`, `require_in` the first state defined in the included sls file.

When writing sls files with this renderer, you should avoid using what can be defined in a `name` argument of a state as the state's id. That is, avoid writing your states like this:

```
/path/to/some/file:
  file.managed:
    - source: salt://some/file

cp /path/to/some/file file2:
  cmd.run:
    - cwd: /
    - require:
      - file: /path/to/some/file
```

Instead, you should define the state id and the `name` argument separately for each state, and the id should be something meaningful and easy to reference within a requisite (which I think is a good habit anyway, and such extra indirection would also makes your sls file easier to modify later). Thus, the above states should be written like this:

```
add-some-file:
  file.managed:
    - name: /path/to/some/file
    - source: salt://some/file

copy-files:
  cmd.run:
    - name: cp /path/to/some/file file2
    - cwd: /
    - require:
      - file: add-some-file
```

Moreover, when referencing a state from a requisite, you should reference the state's id plus the state name rather than the state name plus its `name` argument. (Yes, in the above example, you can actually `require` the `file: /path/to/some/file`, instead of the `file: add-some-file`). The reason is that this renderer will re-write or rename state id's and their references for state id's prefixed with `.`. So, if you reference `name` then there's no way to reliably rewrite such reference.

83.7 salt.renderers.wempy

`salt.renderers.wempy.`**`render`**(*template_file*, *saltenv='base'*, *sls=''*, *argline=''*, *context=None*, ***kws*)

Render the data passing the functions and grains into the rendering system

Return type string

83.8 salt.renderers.yaml

`salt.renderers.yaml.`**`get_yaml_loader`**(*argline*)

Return the ordered dict yaml loader

`salt.renderers.yaml.`**`render`**(*yaml_data*, *saltenv='base'*, *sls=''*, *argline=''*, ***kws*)

Accepts YAML as a string or as a file object and runs it through the YAML parser.

Return type A Python data structure

CHAPTER 84

Pillars

Salt includes a number of built-in external pillars, listed at *Full list of builtin pillar modules*.

You may also wish to look at the standard pillar documentation, at *Pillar Configuration*

The source for the built-in Salt pillars can be found here: https://github.com/saltstack/salt/blob/develop/salt/pillar

CHAPTER 85

Full list of builtin pillar modules

`cmd_json`	Execute a command and read the output as JSON.
`cmd_yaml`	Execute a command and read the output as YAML.
`cobbler`	A module to pull data from Cobbler via its API into the Pillar dictionary
`django_orm`	Generate Pillar data from Django models through the Django ORM
`git_pillar`	Clone a remote git repository and use the filesystem as a Pillar source
`hiera`	Use hiera data as a Pillar source
`libvirt`	Load up the libvirt keys into Pillar for a given minion if said keys have been generated using the libvirt key r
`mongo`	Read Pillar data from a mongodb collection
`pillar_ldap`	Use LDAP data as a Pillar source
`puppet`	Execute an unmodified puppet_node_classifier and read the output as YAML.
`reclass_adapter`	Use the "reclass" database as a Pillar source
`virtkey`	Accept a key from a hypervisor if the virt runner has already submitted an authorization request

85.1 salt.pillar.cmd_json

Execute a command and read the output as JSON. The JSON data is then directly overlaid onto the minion's Pillar data.

`salt.pillar.cmd_json.`**`ext_pillar`**(*minion_id*, *pillar*, *command*)

Execute a command and read the output as JSON

85.2 salt.pillar.cmd_yaml

Execute a command and read the output as YAML. The YAML data is then directly overlaid onto the minion's Pillar data

`salt.pillar.cmd_yaml.`**`ext_pillar`**(*minion_id*, *pillar*, *command*)

Execute a command and read the output as YAML

85.3 salt.pillar.cobbler

A module to pull data from Cobbler via its API into the Pillar dictionary

85.3.1 Configuring the Cobbler ext_pillar

The same cobbler.* parameters are used for both the Cobbler tops and Cobbler pillar modules.

```
ext_pillar:
- cobbler:
    key: cobbler # Nest results within this key. By default, values are not nested.
    only: [parameters] # Add only these keys to pillar.

cobbler.url: https://example.com/cobbler_api #default is http://localhost/cobbler_api
cobbler.user: username # default is no username
cobbler.password: password # default is no password
```

85.3.2 Module Documentation

`salt.pillar.cobbler.`**`ext_pillar`**(*minion_id*, *pillar*, *key=None*, *only=()*)
Read pillar data from Cobbler via its API.

85.4 salt.pillar.django_orm

Generate Pillar data from Django models through the Django ORM

maintainer Micah Hausler <micah.hausler@gmail.com>

maturity new

85.4.1 Configuring the django_orm ext_pillar

To use this module, your Django project must be on the salt master server with database access. This assumes you are using virtualenv with all the project's requirements installed.

```
ext_pillar:
  - django_orm:
      pillar_name: my_application
      project_path: /path/to/project/
      settings_module: my_application.settings
      env_file: /path/to/env/file.sh
      # Optional: If your project is not using the system python,
      # add your virtualenv path below.
      env: /path/to/virtualenv/

      django_app:

        # Required: the app that is included in INSTALLED_APPS
        my_application.clients:

          # Required: the model name
          Client:

            # Required: model field to use as the key in the rendered
            # Pillar. Must be unique; must also be included in the
            # ``fields`` list below.
            name: shortname
            # Optional:
            # See Django's QuerySet documentation for how to use .filter()
            filter:  {'kw': 'args'}

            # Required: a list of field names
            # List items will be used as arguments to the .values() method.
            # See Django's QuerySet documentation for how to use .values()
            fields:
              - field_1
              - field_2
```

This would return pillar data that would look like

```
my_application:
  my_application.clients:
    Client:
      client_1:
        field_1: data_from_field_1
        field_2: data_from_field_2
      client_2:
        field_1: data_from_field_1
        field_2: data_from_field_2
```

As another example, data from multiple database tables can be fetched using Django's regular lookup syntax. Note, using ManyToManyFields will not currently work since the return from values() changes if a ManyToMany is present.

```
ext_pillar:
  - django_orm:
      pillar_name: djangotutorial
      project_path: /path/to/mysite
      settings_module: mysite.settings

      django_app:
        mysite.polls:
          Choices:
            name: poll__question
            fields:
              - poll__question
              - poll__id
              - choice_text
              - votes
```

85.4.2 Module Documentation

`salt.pillar.django_orm.`**`ext_pillar`**(*minion_id*, *pillar*, *pillar_name*, *project_path*, *settings_module*, *django_app*, *env=None*, *env_file=None*, **args*, ***kwargs*)

Connect to a Django database through the ORM and retrieve model fields

Parameters

- **pillar_name** (*str*) – The name of the pillar to be returned
- **project_path** (*str*) – The full path to your Django project (the directory manage.py is in)
- **settings_module** (*str*) – The settings module for your project. This can be found in your manage.py file
- **django_app** (*str*) – A dictionary containing your apps, models, and fields

- **env** (*str*) – The full path to the virtualenv for your Django project
- **env_file** (*str*) – An optional bash file that sets up your environment. The file is run in a subprocess and the changed variables are then added

85.5 salt.pillar.git_pillar

Clone a remote git repository and use the filesystem as a Pillar source

This external Pillar source can be configured in the master config file like so:

```
ext_pillar:
  - git: master git://gitserver/git-pillar.git
```

Note that this is not the same thing as configuring pillar data using the `pillar_roots` parameter. The branch referenced in the `ext_pillar` entry above (`master`), would evaluate to the `base` environment, so this branch needs to contain a `top.sls` with a `base` section in it, like this:

```
base:
  '*':
    - foo
```

To use other environments from the same git repo as git_pillar sources, just add additional lines, like so:

```
ext_pillar:
  - git: master git://gitserver/git-pillar.git
  - git: dev git://gitserver/git-pillar.git
```

In this case, the `dev` branch would need its own `top.sls` with a `dev` section in it, like this:

```
dev:
  '*':
    - bar
```

class `salt.pillar.git_pillar.`**`GitPillar`**`(`*branch*, *repo_location*, *opts*`)`

Deal with the remote git repository for Pillar

`envs``()`

Return a list of refs that can be used as environments

`update``()`

Ensure you are following the latest changes on the remote

Return boolean wether it worked

`salt.pillar.git_pillar.`**`envs`**`(`*branch*, *repo_location*`)`

Return a list of refs that can be used as environments

`salt.pillar.git_pillar.`**`ext_pillar`**`(`*minion_id*, *pillar*, *repo_string*`)`

Execute a command and read the output as YAML

`salt.pillar.git_pillar.`**`update`**`(`*branch*, *repo_location*`)`

Ensure you are following the latest changes on the remote

return boolean whether it worked

85.6 salt.pillar.hiera

Use hiera data as a Pillar source

`salt.pillar.hiera.`**`ext_pillar`**(*minion_id*, *pillar*, *conf*)
Execute hiera and return the data

85.7 salt.pillar.libvirt

Load up the libvirt keys into Pillar for a given minion if said keys have been generated using the libvirt key runner

`salt.pillar.libvirt.`**`ext_pillar`**(*minion_id*, *pillar*, *command*)
Read in the generated libvirt keys

`salt.pillar.libvirt.`**`gen_hyper_keys`**(*minion_id*, *country='US'*, *state='Utah'*, *locality='Salt Lake City'*, *organization='Salted'*)
Generate the keys to be used by libvirt hypervisors, this routine gens the keys and applies them to the pillar for the hypervisor minions

85.8 salt.pillar.mongo

Read Pillar data from a mongodb collection

This module will load a node-specific pillar dictionary from a mongo collection. It uses the node's id for lookups and can load either the whole document, or just a specific field from that document as the pillar dictionary.

85.8.1 Salt Master Mongo Configuration

The module shares the same base mongo connection variables as `salt.returners.mongo_return`. These variables go in your master config file.

- `mongo.db` - The mongo database to connect to. Defaults to `'salt'`.
- `mongo.host` - The mongo host to connect to. Supports replica sets by specifying all hosts in the set, comma-delimited. Defaults to `'salt'`.
- `mongo.port` - The port that the mongo database is running on. Defaults to `27017`.
- `mongo.user` - The username for connecting to mongo. Only required if you are using mongo authentication. Defaults to `''`.
- `mongo.password` - The password for connecting to mongo. Only required if you are using mongo authentication. Defaults to `''`.

85.8.2 Configuring the Mongo ext_pillar

The Mongo ext_pillar takes advantage of the fact that the Salt Master configuration file is yaml. It uses a sub-dictionary of values to adjust specific features of the pillar. This is the explicit single-line dictionary notation for yaml. One may be able to get the easier-to-read multine dict to work correctly with some experimentation.

```
ext_pillar:
  - mongo: {collection: vm, id_field: name, re_pattern: \.example\.com, fields: [customer_id, softwa
```

In the example above, we've decided to use the `vm` collection in the database to store the data. Minion ids are stored in the `name` field on documents in that collection. And, since minion ids are FQDNs in most cases, we'll need to trim the domain name in order to find the minion by hostname in the collection. When we find a minion, return only the `customer_id`, `software`, and `apache_vhosts` fields, as that will contain the data we want for a given node. They will be available directly inside the `pillar` dict in your SLS templates.

85.8.3 Module Documentation

`salt.pillar.mongo.`**`ext_pillar`**(*minion_id*, *pillar*, *collection='pillar'*, *id_field='_id'*, *re_pattern=None*, *re_replace='‘*, *fields=None*)
Connect to a mongo database and read per-node pillar information.

Parameters:

- *collection*: The mongodb collection to read data from. Defaults to `'pillar'`.
- *id_field*: The field in the collection that represents an individual minion id. Defaults to `'_id'`.
- *re_pattern*: If your naming convention in the collection is shorter than the minion id, you can use this to trim the name. *re_pattern* will be used to match the name, and *re_replace* will be used to replace it. Backrefs are supported as they are in the Python standard library. If `None`, no mangling of the name will be performed - the collection will be searched with the entire minion id. Defaults to `None`.
- *re_replace*: Use as the replacement value in node ids matched with *re_pattern*. Defaults to ‘’. Feel free to use backreferences here.
- *fields*: The specific fields in the document to use for the pillar data. If `None`, will use the entire document. If using the entire document, the `_id` field will be converted to string. Be careful with other fields in the document as they must be string serializable. Defaults to `None`.

85.9 salt.pillar.pillar_ldap

Use LDAP data as a Pillar source

This pillar module parses a config file (specified in the salt master config), and executes a series of LDAP searches based on that config. Data returned by these searches is aggregated, with data items found later in the LDAP search order overriding data found earlier on.

The final result set is merged with the pillar data.

`salt.pillar.pillar_ldap.`**`ext_pillar`**(*minion_id*, *pillar*, *config_file*)
Execute LDAP searches and return the aggregated data

85.10 salt.pillar.puppet

Execute an unmodified puppet_node_classifier and read the output as YAML. The YAML data is then directly overlaid onto the minion's Pillar data.

`salt.pillar.puppet.`**`ext_pillar`**(*minion_id*, *pillar*, *command*)
Execute an unmodified puppet_node_classifier and read the output as YAML

85.11 salt.pillar.reclass_adapter

Use the “reclass” database as a Pillar source

This `ext_pillar` plugin provides access to the **reclass** database, such that Pillar data for a specific minion are fetched using **reclass**.

You can find more information about **reclass** at http://reclass.pantsfullofunix.net.

To use the plugin, add it to the `ext_pillar` list in the Salt master config and tell **reclass** by way of a few options how and where to find the inventory:

```
ext_pillar:
    - reclass:
        storage_type: yaml_fs
        inventory_base_uri: /srv/salt
```

This would cause **reclass** to read the inventory from YAML files in `/srv/salt/nodes` and `/srv/salt/classes`.

If you are also using **reclass** as `master_tops` plugin, and you want to avoid having to specify the same information for both, use YAML anchors (take note of the differing data types for `ext_pillar` and `master_tops`):

```
reclass: &reclass
    storage_type: yaml_fs
    inventory_base_uri: /srv/salt
    reclass_source_path: ~/code/reclass

ext_pillar:
    - reclass: *reclass

master_tops:
    reclass: *reclass
```

If you want to run reclass from source, rather than installing it, you can either let the master know via the `PYTHONPATH` environment variable, or by setting the configuration option, like in the example above.

`salt.pillar.reclass_adapter.`**`ext_pillar`**(*minion_id*, *pillar*, ***kwargs*)
    Obtain the Pillar data from **reclass** for the given `minion_id`.

85.12 salt.pillar.virtkey

Accept a key from a hypervisor if the virt runner has already submitted an authorization request

`salt.pillar.virtkey.`**`ext_pillar`**(*hyper_id*, *pillar*, *name*, *key*)
    Accept the key for the VM on the hyper, if authorized.

CHAPTER 86

Master Tops

Salt includes a number of built-in subsystems to generate top file data, they are listed listed at *Full list of builtin master tops modules*.

The source for the built-in Salt master tops can be found here: https://github.com/saltstack/salt/blob/develop/salt/tops

Full list of builtin master tops modules

`cobbler`	Cobbler Tops
`ext_nodes`	External Nodes Classifier
`mongo`	Read tops data from a mongodb collection
`reclass_adapter`	Read tops data from a reclass database

87.1 salt.tops.cobbler

87.1.1 Cobbler Tops

Cobbler Tops is a master tops subsystem used to look up mapping information from Cobbler via its API. The same cobbler.* parameters are used for both the Cobbler tops and Cobbler pillar modules.

```
master_tops:
  cobbler: {}
cobbler.url: https://example.com/cobbler_api #default is http://localhost/cobbler_api
cobbler.user: username # default is no username
cobbler.password: password # default is no password
```

87.1.2 Module Documentation

`salt.tops.cobbler.`**`top`**`(`***kwargs*`)`
Look up top data in Cobbler for a minion.

87.2 salt.tops.ext_nodes

87.2.1 External Nodes Classifier

The External Nodes Classifier is a master tops subsystem used to hook into systems used to provide mapping information used by major configuration management systems. One of the most common external nodes classification system is provided by Cobbler and is called `cobbler-ext-nodes`.

The cobbler-ext-nodes command can be used with this configuration:

```
master_tops:
  ext_nodes: cobbler-ext-nodes
```

It is noteworthy that the Salt system does not directly ingest the data sent from the `cobbler-ext-nodes` command, but converts the data into information that is used by a Salt top file.

`salt.tops.ext_nodes.`**`top`**`(`*`**kwargs`*`)`
Run the command configured

87.3 salt.tops.mongo

Read tops data from a mongodb collection

This module will load tops data from a mongo collection. It uses the node's id for lookups.

87.3.1 Salt Master Mongo Configuration

The module shares the same base mongo connection variables as `salt.returners.mongo_return`. These variables go in your master config file.

- `mongo.db` - The mongo database to connect to. Defaults to `'salt'`.
- `mongo.host` - The mongo host to connect to. Supports replica sets by specifying all hosts in the set, comma-delimited. Defaults to `'salt'`.
- `mongo.port` - The port that the mongo database is running on. Defaults to `27017`.
- `mongo.user` - The username for connecting to mongo. Only required if you are using mongo authentication. Defaults to `''`.
- `mongo.password` - The password for connecting to mongo. Only required if you are using mongo authentication. Defaults to `''`.

87.3.2 Configuring the Mongo Tops Subsystem

```
master_tops:
  mongo:
    collection: tops
    id_field: _id
    re_replace: ""
    re_pattern: \.example\.com
    states_field: states
    environment_field: environment
```

87.3.3 Module Documentation

`salt.tops.mongo.`**`top`**`(`*`**kwargs`*`)`
Connect to a mongo database and read per-node tops data.

Parameters:

- *collection*: The mongodb collection to read data from. Defaults to `'tops'`.
- *id_field*: The field in the collection that represents an individual minion id. Defaults to `'_id'`.
- *re_pattern*: If your naming convention in the collection is shorter than the minion id, you can use this to trim the name. *re_pattern* will be used to match the name, and *re_replace* will be used to replace it. Backrefs are supported as they are in the Python standard library. If `None`, no mangling of the name will be performed - the collection will be searched with the entire minion id. Defaults to `None`.

- *re_replace*: Use as the replacement value in node ids matched with *re_pattern*. Defaults to ''. Feel free to use backreferences here.
- *states_field*: The name of the field providing a list of states.
- *environment_field*: The name of the field providing the environment. Defaults to `environment`.

87.4 salt.tops.reclass_adapter

Read tops data from a reclass database

This *master_tops* plugin provides access to the **reclass** database, such that state information (top data) are retrieved from **reclass**.

You can find more information about **reclass** at http://reclass.pantsfullofunix.net.

To use the plugin, add it to the `master_tops` list in the Salt master config and tell **reclass** by way of a few options how and where to find the inventory:

```
master_tops:
  reclass:
    storage_type: yaml_fs
    inventory_base_uri: /srv/salt
```

This would cause **reclass** to read the inventory from YAML files in `/srv/salt/nodes` and `/srv/salt/classes`.

If you are also using **reclass** as `ext_pillar` plugin, and you want to avoid having to specify the same information for both, use YAML anchors (take note of the differing data types for `ext_pillar` and `master_tops`):

```
reclass: &reclass
  storage_type: yaml_fs
  inventory_base_uri: /srv/salt
  reclass_source_path: ~/code/reclass

ext_pillar:
  - reclass: *reclass

master_tops:
  reclass: *reclass
```

If you want to run reclass from source, rather than installing it, you can either let the master know via the `PYTHONPATH` environment variable, or by setting the configuration option, like in the example above.

`salt.tops.reclass_adapter.`**`top`**`(`****kwargs***`)`

Query **reclass** for the top data (states of the minions).

Salt Runners

See also:

The full list of runners

Salt runners are convenience applications executed with the salt-run command. Where as salt modules are sent out to minions for execution, salt runners are executed on the salt master.

A Salt runner can be a simple client call, or a complex application.

The use for a Salt runner is to build a frontend hook for running sets of commands via Salt or creating special formatted output.

88.1 Writing Salt Runners

Salt runners can be easily written, the work in a similar way to Salt modules except they run on the server side.

A runner is a Python module that contains functions, each public function is a runner that can be executed via the *salt-run* command.

If a Python module named test.py is created in the runners directory and contains a function called `foo` then the function could be called with:

```
# salt-run test.foo
```

88.2 Examples

The best examples of runners can be found in the Salt source:

https://github.com/saltstack/salt/blob/develop/salt/runners

A simple runner that returns a well-formatted list of the minions that are responding to Salt calls would look like this:

```
# Import salt modules
import salt.client

def up():
    '''
    Print a list of all of the minions that are up
    '''
    client = salt.client.LocalClient(__opts__['conf_file'])
    minions = client.cmd('*', 'test.ping', timeout=1)
```

```
for minion in sorted(minions):
    print minion
```

Full list of runner modules

`cache`	Return cached data from minions
`doc`	A runner module to collect and display the inline documentation from the
`fileserver`	Directly manage the salt fileserver plugins
`jobs`	A convenience system to manage jobs, both active and already run
`launchd`	
`lxc`	Control Linux Containers via Salt
`manage`	General management functions for salt, tools like seeing what hosts are up
`network`	Network tools to run from the Master
`search`	Runner frontend to search system
`state`	Execute overstate functions
`thin`	The thin runner is used to manage the salt thin systems.
`virt`	Control virtual machines via Salt
`winrepo`	Runner to manage Windows software repo

89.1 salt.runners.cache

Return cached data from minions

`salt.runners.cache.`**`clear_all`**(*tgt=None*, *expr_form='glob'*)

Clear the cached pillar, grains, and mine data of the targeted minions

CLI Example:

```
salt-run cache.clear_all
```

`salt.runners.cache.`**`clear_grains`**(*tgt=None*, *expr_form='glob'*)

Clear the cached grains data of the targeted minions

CLI Example:

```
salt-run cache.clear_grains
```

`salt.runners.cache.`**`clear_mine`**(*tgt=None*, *expr_form='glob'*)

Clear the cached mine data of the targeted minions

CLI Example:

```
salt-run cache.clear_mine
```

`salt.runners.cache.`**`clear_mine_func`**(*tgt=None*, *expr_form='glob'*, *clear_mine_func=None*)

Clear the cached mine function data of the targeted minions

CLI Example:

```
salt-run cache.clear_mine_func tgt='*',clear_mine_func='network.interfaces'
```

`salt.runners.cache.`**`clear_pillar`**`(`*tgt=None*, *expr_form='glob'*`)`
Clear the cached pillar data of the targeted minions

CLI Example:

```
salt-run cache.clear_pillar
```

`salt.runners.cache.`**`grains`**`(`*tgt=None*, *expr_form='glob'*, ***kwargs*`)`
Return cached grains of the targeted minions

CLI Example:

```
salt-run cache.grains
```

`salt.runners.cache.`**`pillar`**`(`*tgt=None*, *expr_form='glob'*, ***kwargs*`)`
Return cached pillars of the targeted minions

CLI Example:

```
salt-run cache.pillar
```

89.2 salt.runners.doc

A runner module to collect and display the inline documentation from the various module types

`salt.runners.doc.`**`execution`**`()`
Collect all the sys.doc output from each minion and return the aggregate

CLI Example:

```
salt-run doc.execution
```

`salt.runners.doc.`**`runner`**`()`
Return all inline documetation for runner modules

CLI Example:

```
salt-run doc.runner
```

`salt.runners.doc.`**`wheel`**`()`
Return all inline documentation for wheel modules

CLI Example:

```
salt-run doc.wheel
```

89.3 salt.runners.fileserver

Directly manage the salt fileserver plugins

`salt.runners.fileserver.`**`update`**`()`
Execute an update for all of the configured fileserver backends

CLI Example:

```
salt-run fileserver.update
```

89.4 salt.runners.jobs

A convenience system to manage jobs, both active and already run

`salt.runners.jobs.`**`active`**`()`
Return a report on all actively running jobs from a job id centric perspective

CLI Example:

```
salt-run jobs.active
```

`salt.runners.jobs.`**`list_job`**`(`*jid*`)`
List a specific job given by its jid

CLI Example:

```
salt-run jobs.list_job 20130916125524463507
```

`salt.runners.jobs.`**`list_jobs`**`(`*ext_source=None*`)`
List all detectable jobs and associated functions

CLI Example:

```
salt-run jobs.list_jobs
```

`salt.runners.jobs.`**`lookup_jid`**`(`*jid*, *ext_source=None*`)`
Return the printout from a previously executed job

CLI Example:

```
salt-run jobs.lookup_jid 20130916125524463507
```

`salt.runners.jobs.`**`print_job`**`(`*job_id*`)`
Print job available details, including return data.

CLI Example:

```
salt-run jobs.print_job
```

89.5 salt.runners.launchd

`salt.runners.launchd.`**`write_launchd_plist`**`(`*program*`)`
Write a launchd plist for managing salt-master or salt-minion

CLI Example:

```
salt-run launchd.write_launchd_plist salt-master
```

89.6 salt.runners.lxc

Control Linux Containers via Salt

depends lxc execution module

`salt.runners.lxc.`**`find_guest`**(*name*, *quiet=False*)

Returns the host for a container.

```
salt-run lxc.find_guest name
```

`salt.runners.lxc.`**`freeze`**(*name*, *quiet=False*)

Freeze the named container

```
salt-run lxc.freeze name
```

`salt.runners.lxc.`**`info`**(*name*, *quiet=False*)

Returns information about a container.

```
salt-run lxc.info name
```

`salt.runners.lxc.`**`init`**(*name*, *host=None*, ***kwargs*)

Initialize a new container

```
salt-run lxc.init name host=minion_id [cpuset=cgroups_cpuset] \
        [cpushare=cgroups_cpushare] [memory=cgroups_memory] \
        [nic=nic_profile] [profile=lxc_profile] \
        [nic_opts=nic_opts] [start=(true|false)] \
        [seed=(true|false)] [install=(true|false)] \
        [config=minion_config]
```

name Name of the container.

host Minion to start the container on. Required.

cpuset cgroups cpuset.

cpushare cgroups cpu shares.

memory cgroups memory limit, in MB.

nic Network interfaces profile (defined in config or pillar).

profile A LXC profile (defined in config or pillar).

nic_opts Extra options for network interfaces. E.g: {"eth0": {"mac": "aa:bb:cc:dd:ee:ff", "ipv4": "10.1.1.1", "ipv6": "2001:db8::ff00:42:8329"}}

start Start the newly created container.

seed Seed the container with the minion config and autosign its key. Default: true

install If salt-minion is not already installed, install it. Default: true

config Optional config paramers. By default, the id is set to the name of the container.

`salt.runners.lxc.`**`list_`**(*host=None*, *quiet=False*)

List defined containers (running, stopped, and frozen) for the named (or all) host(s).

```
salt-run lxc.list [host=minion_id]
```

`salt.runners.lxc.`**`purge`**(*name*, *delete_key=True*, *quiet=False*)

Purge the named container and delete its minion key if present. WARNING: Destroys all data associated with the container.

```
salt-run lxc.purge name
```

`salt.runners.lxc.`**`start`**(*name*, *quiet=False*)

Start the named container.

```
salt-run lxc.start name
```

`salt.runners.lxc.`**`stop`**(*name*, *quiet=False*)

Stop the named container.

```
salt-run lxc.stop name
```

`salt.runners.lxc.`**`unfreeze`**(*name*, *quiet=False*)

Unfreeze the named container

```
salt-run lxc.unfreeze name
```

89.7 salt.runners.manage

General management functions for salt, tools like seeing what hosts are up and what hosts are down

`salt.runners.manage.`**`bootstrap`**(*version=’develop’*, *script=None*, *hosts=’‘*)

Bootstrap minions with salt-bootstrap

Options: version: git tag of version to install [default: develop] script: Script to execute [default: http://bootstrap.saltstack.org] hosts: Comma separated hosts [example: hosts="host1.local,host2.local"]

CLI Example:

```
salt-run manage.bootstrap hosts="host1,host2"
salt-run manage.bootstrap hosts="host1,host2" version="v0.17"
salt-run manage.bootstrap hosts="host1,host2" version="v0.17" script="https://raw.githubuserconte
```

`salt.runners.manage.`**`bootstrap_psexec`**(*hosts=’‘*, *master=None*, *version=None*, *arch=’win32’*, *installer_url=None*, *username=None*, *password=None*)

Bootstrap Windows minions via PsExec.

hosts Comma separated list of hosts to deploy the Windows Salt minion.

master Address of the Salt master passed as an argument to the installer.

version Point release of installer to download. Defaults to the most recent.

arch Architecture of installer to download. Defaults to win32.

installer_url URL of minion installer executable. Defaults to the latest version from http://docs.saltstack.com/downloads

username Optional user name for login on remote computer.

password Password for optional username. If omitted, PsExec will prompt for one to be entered for each host.

CLI Example:

```
salt-run manage.bootstrap_psexec hosts='host1,host2'
salt-run manage.bootstrap_psexec hosts='host1,host2' version='0.17' username='DOMAIN\Administrat
salt-run manage.bootstrap_psexec hosts='host1,host2' installer_url='http://exampledomain/salt-in
```

`salt.runners.manage.`**`down`**(*removekeys=False*)

Print a list of all the down or unresponsive salt minions Optionally remove keys of down minions

CLI Example:

```
salt-run manage.down
salt-run manage.down removekeys=True
```

`salt.runners.manage.`**`key_regen`**`()`

This routine is used to regenerate all keys in an environment. This is invasive! ALL KEYS IN THE SALT ENVIRONMENT WILL BE REGENERATED!!

The key_regen routine sends a command out to minions to revoke the master key and remove all minion keys, it then removes all keys from the master and prompts the user to restart the master. The minions will all reconnect and keys will be placed in pending.

After the master is restarted and minion keys are in the pending directory execute a salt-key -A command to accept the regenerated minion keys.

The master *must* be restarted within 60 seconds of running this command or the minions will think there is something wrong with the keys and abort.

Only Execute this runner after upgrading minions and master to 0.15.1 or higher!

CLI Example:

```
salt-run manage.key_regen
```

`salt.runners.manage.`**`present`**`()`

Print a list of all minions that are up according to Salt's presence detection, no commands will be sent

CLI Example:

```
salt-run manage.present
```

`salt.runners.manage.`**`safe_accept`**`(`*target*, *expr_form='glob'*`)`

Accept a minion's public key after checking the fingerprint over salt-ssh

CLI Example:

```
salt-run manage.safe_accept my_minion
salt-run manage.safe_accept minion1,minion2 expr_form=list
```

`salt.runners.manage.`**`status`**`(`*output=True*`)`

Print the status of all known salt minions

CLI Example:

```
salt-run manage.status
```

`salt.runners.manage.`**`up`**`()`

Print a list of all of the minions that are up

CLI Example:

```
salt-run manage.up
```

`salt.runners.manage.`**`versions`**`()`

Check the version of active minions

CLI Example:

```
salt-run manage.versions
```

89.8 salt.runners.network

Network tools to run from the Master

`salt.runners.network.`**`wol`**(*mac*, *bcast='255.255.255.255'*, *destport=9*)

Send a "Magic Packet" to wake up a Minion

CLI Example:

```
salt-run network.wol 08-00-27-13-69-77
salt-run network.wol 080027136977 255.255.255.255 7
salt-run network.wol 08:00:27:13:69:77 255.255.255.255 7
```

`salt.runners.network.`**`wollist`**(*maclist*, *bcast='255.255.255.255'*, *destport=9*)

Send a "Magic Packet" to wake up a list of Minions. This list must contain one MAC hardware address per line

CLI Example:

```
salt-run network.wollist '/path/to/maclist'
salt-run network.wollist '/path/to/maclist' 255.255.255.255 7
salt-run network.wollist '/path/to/maclist' 255.255.255.255 7
```

89.9 salt.runners.search

Runner frontend to search system

`salt.runners.search.`**`query`**(*term*)

Query the search system

CLI Example:

```
salt-run search.query foo
```

89.10 salt.runners.state

Execute overstate functions

`salt.runners.state.`**`orchestrate`**(*mods*, *saltenv='base'*, *test=None*, *exclude=None*, *pillar=None*)

New in version 0.17.0.

Execute a state run from the master, used as a powerful orchestration system.

CLI Examples:

```
salt-run state.orchestrate webserver
salt-run state.orchestrate webserver saltenv=dev test=True
```

Changed in version 2014.1.1: Runner renamed from `state.sls` to `state.orchestrate`

`salt.runners.state.`**`over`**(*saltenv='base'*, *os_fn=None*)

New in version 0.11.0.

Execute an overstate sequence to orchestrate the executing of states over a group of systems

CLI Examples:

```
salt-run state.over base /path/to/myoverstate.sls
```

`salt.runners.state.`**`show_stages`**(*saltenv='base'*, *os_fn=None*)

New in version 0.11.0.

Display the OverState's stage data

CLI Examples:

```
salt-run state.show_stages
salt-run state.show_stages saltenv=dev /root/overstate.sls
```

89.11 salt.runners.thin

The thin runner is used to manage the salt thin systems.

Salt Thin is a transport-less version of Salt that can be used to run rouitines in a standalone way. This runner has tools which generate the standalone salt system for easy consumption.

`salt.runners.thin.`**`generate`**`(`*extra_mods='', overwrite=False*`)`

Generate the salt-thin tarball and print the location of the tarball Optional additional mods to include (e.g. mako) can be supplied as a comma delimited string. Permits forcing an overwrite of the output file as well.

CLI Example:

```
salt-run thin.generate
salt-run thin.generate mako
salt-run thin.generate mako,wempy 1
salt-run thin.generate overwrite=1
```

89.12 salt.runners.virt

Control virtual machines via Salt

`salt.runners.virt.`**`force_off`**`(`*name*`)`

Force power down the named virtual machine

`salt.runners.virt.`**`hyper_info`**`(`*hyper=None*`)`

Return information about the hypervisors connected to this master

`salt.runners.virt.`**`init`**`(`*name, cpu, mem, image, hyper=None, seed=True, nic='default', install=True*`)`

Initialize a new vm

`salt.runners.virt.`**`list`**`(`*hyper=None, quiet=False*`)`

List the virtual machines on each hyper

`salt.runners.virt.`**`migrate`**`(`*name, target=''*`)`

Migrate a vm from one hypervisor to another. This routine will just start the migration and display information on how to look up the progress

`salt.runners.virt.`**`next_hyper`**`()`

Return the hypervisor to use for the next autodeployed vm

`salt.runners.virt.`**`pause`**`(`*name*`)`

Pause the named vm

`salt.runners.virt.`**`purge`**`(`*name, delete_key=True*`)`

Destroy the named vm

`salt.runners.virt.`**`query`**`(`*hyper=None, quiet=False*`)`

Query the virtual machines

`salt.runners.virt.`**`reset`**`(`*name*`)`

Force power down and restart an existing vm

`salt.runners.virt.`**`resume`**`(`*name*`)`
Resume a paused vm

`salt.runners.virt.`**`start`**`(`*name*`)`
Start a named virtual machine

`salt.runners.virt.`**`vm_info`**`(`*name*, *quiet=False*`)`
Return the information on the named vm

89.13 salt.runners.winrepo

Runner to manage Windows software repo

`salt.runners.winrepo.`**`genrepo`**`()`
Generate win_repo_cachefile based on sls files in the win_repo

CLI Example:

```
salt-run winrepo.genrepo
```

`salt.runners.winrepo.`**`update_git_repos`**`()`
Checkout git repos containing Windows Software Package Definitions

CLI Example:

```
salt-run winrepo.update_git_repos
```

Full list of builtin wheel modules

`config`	Manage the master configuration file
`file_roots`	Read in files from the file_root and save files to the file root
`key`	Wheel system wrapper for key system
`pillar_roots`	The *pillar_roots* wheel module is used to manage files under the pillar roots directories on the master server.

90.1 salt.wheel.config

Manage the master configuration file

`salt.wheel.config.`**`apply`**`(`*key*, *value*`)`

Set a single key

Note: This will strip comments from your config file

`salt.wheel.config.`**`values`**`()`

Return the raw values of the config file

90.2 salt.wheel.file_roots

Read in files from the file_root and save files to the file root

`salt.wheel.file_roots.`**`find`**`(`*path*, *saltenv='base'*, *env=None*`)`

Return a dict of the files located with the given path and environment

`salt.wheel.file_roots.`**`list_env`**`(`*saltenv='base'*, *env=None*`)`

Return all of the file paths found in an environment

`salt.wheel.file_roots.`**`list_roots`**`()`

Return all of the files names in all available environments

`salt.wheel.file_roots.`**`read`**`(`*path*, *saltenv='base'*, *env=None*`)`

Read the contents of a text file, if the file is binary then

`salt.wheel.file_roots.`**`write`**`(`*data*, *path*, *saltenv='base'*, *index=0*, *env=None*`)`

Write the named file, by default the first file found is written, but the index of the file can be specified to write to a lower priority file root

90.3 salt.wheel.key

Wheel system wrapper for key system

`salt.wheel.key.`**`accept`**`(`*match*`)`
Accept keys based on a glob match

`salt.wheel.key.`**`delete`**`(`*match*`)`
Delete keys based on a glob match

`salt.wheel.key.`**`finger`**`(`*match*`)`
Return the matching key fingerprints

`salt.wheel.key.`**`gen`**`(`*id_=None*, *keysize=2048*`)`
Generate a key pair. No keys are stored on the master, a keypair is returned as a dict containing pub and priv keys

`salt.wheel.key.`**`gen_accept`**`(`*id_*, *keysize=2048*, *force=False*`)`
Generate a key pair then accept the public key. This function returns the key pair in a dict, only the public key is preserved on the master.

`salt.wheel.key.`**`key_str`**`(`*match*`)`
Return the key strings

`salt.wheel.key.`**`list_`**`(`*match*`)`
List all the keys under a named status

`salt.wheel.key.`**`list_all`**`()`
List all the keys

`salt.wheel.key.`**`reject`**`(`*match*`)`
Delete keys based on a glob match

90.4 salt.wheel.pillar_roots

The *pillar_roots* wheel module is used to manage files under the pillar roots directories on the master server.

`salt.wheel.pillar_roots.`**`find`**`(`*path*, *saltenv='base'*, *env=None*`)`
Return a dict of the files located with the given path and environment

`salt.wheel.pillar_roots.`**`list_env`**`(`*saltenv='base'*, *env=None*`)`
Return all of the file paths found in an environment

`salt.wheel.pillar_roots.`**`list_roots`**`()`
Return all of the files names in all available environments

`salt.wheel.pillar_roots.`**`read`**`(`*path*, *saltenv='base'*, *env=None*`)`
Read the contents of a text file, if the file is binary then

`salt.wheel.pillar_roots.`**`write`**`(`*data*, *path*, *saltenv='base'*, *index=0*, *env=None*`)`
Write the named file, by default the first file found is written, but the index of the file can be specified to write to a lower priority file root

Full list of builtin auth modules

`auto`	An "Always Approved" eauth interface to test against, not intended for
`keystone`	Provide authentication using OpenStack Keystone
`ldap`	Provide authentication using simple LDAP binds
`pam`	Authenticate against PAM
`stormpath_mod`	Salt Stormpath Authentication

91.1 salt.auth.auto

An "Always Approved" eauth interface to test against, not intended for production use

`salt.auth.auto.`**`auth`**`(`*username*, *password*`)`
Authenticate!

91.2 salt.auth.keystone

Provide authentication using OpenStack Keystone

depends

- keystoneclient Python module

`salt.auth.keystone.`**`auth`**`(`*username*, *password*`)`
Try and authenticate

`salt.auth.keystone.`**`get_auth_url`**`()`
Try and get the URL from the config, else return localhost

91.3 salt.auth.ldap

Provide authentication using simple LDAP binds

depends

- ldap Python module

`salt.auth.ldap.`**`auth`**`(`*username*, *password*`)`
Authenticate via an LDAP bind

91.4 salt.auth.pam

Authenticate against PAM

Provides an authenticate function that will allow the caller to authenticate a user against the Pluggable Authentication Modules (PAM) on the system.

Implemented using ctypes, so no compilation is necessary.

Note: PAM authentication will not work for the `root` user.

The Python interface to PAM does not support authenticating as `root`.

---

class `salt.auth.pam.`**`PamConv`**

Wrapper class for pam_conv structure

`appdata_ptr`

Structure/Union member

`conv`

Structure/Union member

class `salt.auth.pam.`**`PamHandle`**

Wrapper class for pam_handle_t

`handle`

Structure/Union member

class `salt.auth.pam.`**`PamMessage`**

Wrapper class for pam_message structure

`msg`

Structure/Union member

`msg_style`

Structure/Union member

class `salt.auth.pam.`**`PamResponse`**

Wrapper class for pam_response structure

`resp`

Structure/Union member

`resp_retcode`

Structure/Union member

`salt.auth.pam.`**`auth`**(*username*, *password*, ***kwargs*)

Authenticate via pam

`salt.auth.pam.`**`authenticate`**(*username*, *password*, *service='login'*)

Returns True if the given username and password authenticate for the given service. Returns False otherwise

`username`: the username to authenticate

`password`: the password in plain text

`service`: the PAM service to authenticate against. Defaults to 'login'

91.5 salt.auth.stormpath_mod

Salt Stormpath Authentication

Module to provide authentication using Stormpath as the backend.

depends

- stormpath-sdk Python module

configuration This module requires the development branch of the stormpath-sdk which can be found here: https://github.com/stormpath/stormpath-sdk-python

The following config items are required in the master config:

```
stormpath.api_key_file: <path/to/apiKey.properties>
stormpath.app_url: <Rest url of your Stormpath application>
```

Ensure that your apiKey.properties is readable by the user the Salt Master is running as, but not readable by other system users.

`salt.auth.stormpath_mod.`**`auth`**`(`*username*, *password*`)`

Try and authenticate

CHAPTER 92

Full list of builtin output modules

`grains`	Special outputter for grains
`highstate`	The return data from the Highstate command is a standard data structure which is parsed by the highstate outp
`json_out`	The JSON output module converts the return data into JSON.
`key`	Salt Key makes use of the outputter system to format information sent to the `salt-key` command.
`nested`	Recursively display nested data, this is the default outputter.
`no_out`	Display no output.
`no_return`	Display output for minions that did not return
`overstatestage`	Display clean output of an overstate stage
`pprint_out`	The python pretty print system was the default outputter.
`raw`	The raw outputter outputs the data via the python print function and is shown in a raw state.
`txt`	The txt outputter has been developed to make the output from shell commands on minions appear as they do w
`virt_query`	virt.query outputter
`yaml_out`	Output data in YAML, this outputter defaults to printing in YAML block mode for better readability.

92.1 salt.output.grains

Special outputter for grains

`salt.output.grains.`**`output`**`(`*grains*`)`
Output the grains in a clean way

92.2 salt.output.highstate

The return data from the Highstate command is a standard data structure which is parsed by the highstate outputter to deliver a clean and readable set of information about the HighState run on minions.

Two configurations can be set to modify the highstate outputter. These values can be set in the master config to change the output of the `salt` command or set in the minion config to change the output of the `salt-call` command.

state_verbose: By default *state_verbose* is set to *True*, setting this to *False* will instruct the highstate outputter to omit displaying anything in green, this means that nothing with a result of True and no changes will not be printed

state_output: The highstate outputter has three output modes, *full*, *terse*, *mixed*, and *changes*. The default is set to full, which will display many lines of detailed information for each executed chunk. If the *state_output* option is set to *terse* then the output is greatly simplified and shown in only one line. If *mixed* is used, then terse output will be used unless a state failed, in which case full output will be used. If *changes* is used, then terse output will be used if there was no error and no changes, otherwise full output will be used.

state_tabular: If *state_output* uses the terse output, set this to *True* for an aligned output format. If you wish to use a custom format, this can be set to a string.

`salt.output.highstate.`**`output`**(*data*)

The HighState Outputter is only meant to be used with the state.highstate function, or a function that returns highstate return data.

92.3 salt.output.json_out

The JSON output module converts the return data into JSON.

`salt.output.json_out.`**`output`**(*data*)

Print the output data in JSON

92.4 salt.output.key

Salt Key makes use of the outputter system to format information sent to the `salt-key` command. This outputter is geared towards ingesting very specific data and should only be used with the salt-key command.

`salt.output.key.`**`output`**(*data*)

Read in the dict structure generated by the salt key API methods and print the structure.

92.5 salt.output.nested

Recursively display nested data, this is the default outputter.

class `salt.output.nested.`**`NestDisplay`**

Manage the nested display contents

`display`(*ret*, *indent*, *prefix*, *out*)

Recursively iterate down through data structures to determine output

`salt.output.nested.`**`output`**(*ret*)

Display ret data

92.6 salt.output.no_out

Display no output.

`salt.output.no_out.`**`output`**(*ret*)

Don't display data. Used when you only are interested in the return.

92.7 salt.output.no_return

Display output for minions that did not return

class `salt.output.no_return.`**`NestDisplay`**

Create generator for nested output

display (*ret*, *indent*, *prefix*, *out*)
Recursively iterate down through data structures to determine output

`salt.output.no_return.`**output** (*ret*)
Display ret data

92.8 salt.output.overstatestage

Display clean output of an overstate stage

`salt.output.overstatestage.`**output** (*data*)
Format the data for printing stage information from the overstate system

92.9 salt.output.pprint_out

The python pretty print system was the default outputter. This outputter simply passed the data passed into it through the pprint module.

`salt.output.pprint_out.`**output** (*data*)
Print out via pretty print

92.10 salt.output.raw

The raw outputter outputs the data via the python print function and is shown in a raw state. This was the original outputter used by Salt before the outputter system was developed.

`salt.output.raw.`**output** (*data*)
Rather basic....

92.11 salt.output.txt

The txt outputter has been developed to make the output from shell commands on minions appear as they do when the command is executed on the minion.

`salt.output.txt.`**output** (*data*)
Output the data in lines, very nice for running commands

92.12 salt.output.virt_query

virt.query outputter

`salt.output.virt_query.`**output** (*data*)
Display output for the salt-run virt.query function

92.13 salt.output.yaml_out

Output data in YAML, this outputter defaults to printing in YAML block mode for better readability.

`salt.output.yaml_out.`**`output`**`(`*data*`)`

Print out YAML using the block mode

CHAPTER 93

Full list of Salt Cloud modules

`botocore_aws`	The AWS Cloud Module
`cloudstack`	CloudStack Cloud Module
`digital_ocean`	Digital Ocean Cloud Module
`ec2`	The EC2 Cloud Module
`gce`	Copyright 2013 Google Inc.
`gogrid`	GoGrid Cloud Module
`ibmsce`	IBM SCE Cloud Module
`joyent`	Joyent Cloud Module
`libcloud_aws`	The AWS Cloud Module
`linode`	Linode Cloud Module
`msazure`	Azure Cloud Module
`nova`	OpenStack Nova Cloud Module
`openstack`	OpenStack Cloud Module
`parallels`	Parallels Cloud Module
`rackspace`	Rackspace Cloud Module
`saltify`	Saltify Module ============== The Saltify module is designed to install Salt on a remote machine, virtual
`softlayer`	SoftLayer Cloud Module

93.1 salt.cloud.clouds.botocore_aws

93.1.1 The AWS Cloud Module

The AWS cloud module is used to interact with the Amazon Web Services system.

This module has been replaced by the EC2 cloud module, and is no longer supported. The documentation shown here is for reference only; it is highly recommended to change all usages of this driver over to the EC2 driver.

If this driver is still needed, set up the cloud configuration at `/etc/salt/cloud.providers` or `/etc/salt/cloud.providers.d/aws.conf`:

```
my-aws-botocore-config:
  # The AWS API authentication id
  id: GKTADJGHEIQSXMKKRBJ08H
  # The AWS API authentication key
  key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
  # The ssh keyname to use
  keyname: default
  # The amazon security group
  securitygroup: ssh_open
  # The location of the private key which corresponds to the keyname
  private_key: /root/default.pem
  provider: aws
```

`salt.cloud.clouds.botocore_aws.`**`disable_term_protect`**(*name*, *call=None*)
Disable termination protection on a node

CLI Example:

```
salt-cloud -a disable_term_protect mymachine
```

`salt.cloud.clouds.botocore_aws.`**`enable_term_protect`**(*name*, *call=None*)
Enable termination protection on a node

CLI Example:

```
salt-cloud -a enable_term_protect mymachine
```

`salt.cloud.clouds.botocore_aws.`**`get_configured_provider`**()
Return the first configured instance.

93.2 salt.cloud.clouds.cloudstack

93.2.1 CloudStack Cloud Module

The CloudStack cloud module is used to control access to a CloudStack based Public Cloud.

Use of this module requires the `apikey`, `secretkey`, `host` and `path` parameters.

```
my-cloudstack-cloud-config:
  apikey: <your api key >
  secretkey: <your secret key >
  host: localhost
  path: /client/api
  provider: cloudstack
```

`salt.cloud.clouds.cloudstack.`**`avail_images`**(*conn=None*, *call=None*)
Return a dict of all available VM images on the cloud provider with relevant data

`salt.cloud.clouds.cloudstack.`**`avail_locations`**(*conn=None*, *call=None*)
Return a dict of all available VM locations on the cloud provider with relevant data

`salt.cloud.clouds.cloudstack.`**`avail_sizes`**(*conn=None*, *call=None*)
Return a dict of all available VM images on the cloud provider with relevant data

`salt.cloud.clouds.cloudstack.`**`create`**(*vm_*)
Create a single VM from a data dict

`salt.cloud.clouds.cloudstack.`**`destroy`**(*name*, *conn=None*, *call=None*)
Delete a single VM

`salt.cloud.clouds.cloudstack.`**`get_configured_provider`**()
Return the first configured instance.

`salt.cloud.clouds.cloudstack.`**`get_conn`**()
Return a conn object for the passed VM data

`salt.cloud.clouds.cloudstack.`**`get_image`**(*conn*, *vm_*)
Return the image object to use

`salt.cloud.clouds.cloudstack.`**`get_ip`**(*data*)
Return the IP address of the VM If the VM has public IP as defined by libcloud module then use it Otherwise try to extract the private IP and use that one.

`salt.cloud.clouds.cloudstack.`**`get_key`**()
Returns the ssk private key for VM access

`salt.cloud.clouds.cloudstack.`**`get_keypair`**(*vm_*)
Return the keypair to use

`salt.cloud.clouds.cloudstack.`**`get_location`**(*conn*, *vm_*)
Return the node location to use

`salt.cloud.clouds.cloudstack.`**`get_networkid`**(*vm_*)
Return the networkid to use, only valid for Advanced Zone

`salt.cloud.clouds.cloudstack.`**`get_password`**(*vm_*)
Return the password to use

`salt.cloud.clouds.cloudstack.`**`get_size`**(*conn*, *vm_*)
Return the VM's size object

`salt.cloud.clouds.cloudstack.`**`list_nodes`**(*conn=None*, *call=None*)
Return a list of the VMs that are on the provider

`salt.cloud.clouds.cloudstack.`**`list_nodes_full`**(*conn=None*, *call=None*)
Return a list of the VMs that are on the provider, with all fields

`salt.cloud.clouds.cloudstack.`**`list_nodes_select`**(*conn=None*, *call=None*)
Return a list of the VMs that are on the provider, with select fields

`salt.cloud.clouds.cloudstack.`**`script`**(*vm_*)
Return the script deployment object

`salt.cloud.clouds.cloudstack.`**`show_instance`**(*name*, *call=None*)
Show the details from the provider concerning an instance

93.3 salt.cloud.clouds.digital_ocean

93.3.1 Digital Ocean Cloud Module

The Digital Ocean cloud module is used to control access to the Digital Ocean VPS system.

Use of this module only requires the `api_key` parameter to be set. Set up the cloud configuration at `/etc/salt/cloud.providers` or `/etc/salt/cloud.providers.d/digital_ocean.conf`:

```
my-digital-ocean-config:
  # Digital Ocean account keys
  client_key: wFGEwgregeqw3435gDger
  api_key: GDE43t43REGTrkilg43934t34qT43t4dgegerGEgg
  provider: digital_ocean
```

`salt.cloud.clouds.digital_ocean.`**`avail_images`**(*call=None*)
Return a list of the images that are on the provider

`salt.cloud.clouds.digital_ocean.`**`avail_locations`**(*call=None*)
Return a dict of all available VM locations on the cloud provider with relevant data

`salt.cloud.clouds.digital_ocean.`**`avail_sizes`**(*call=None*)
Return a list of the image sizes that are on the provider

`salt.cloud.clouds.digital_ocean.`**`create`**`(`*vm_*`)`
Create a single VM from a data dict

`salt.cloud.clouds.digital_ocean.`**`create_node`**`(`*args*`)`
Create a node

`salt.cloud.clouds.digital_ocean.`**`destroy`**`(`*name*, *call=None*`)`
Destroy a node. Will check termination protection and warn if enabled.

CLI Example:

```
salt-cloud --destroy mymachine
```

`salt.cloud.clouds.digital_ocean.`**`get_configured_provider`**`()`
Return the first configured instance.

`salt.cloud.clouds.digital_ocean.`**`get_image`**`(`*vm_*`)`
Return the image object to use

`salt.cloud.clouds.digital_ocean.`**`get_keyid`**`(`*keyname*`)`
Return the ID of the keyname

`salt.cloud.clouds.digital_ocean.`**`get_location`**`(`*vm_*`)`
Return the VM's location

`salt.cloud.clouds.digital_ocean.`**`get_size`**`(`*vm_*`)`
Return the VM's size. Used by create_node().

`salt.cloud.clouds.digital_ocean.`**`list_keypairs`**`(`*call=None*`)`
Return a dict of all available VM locations on the cloud provider with relevant data

`salt.cloud.clouds.digital_ocean.`**`list_nodes`**`(`*call=None*`)`
Return a list of the VMs that are on the provider

`salt.cloud.clouds.digital_ocean.`**`list_nodes_full`**`(`*call=None*`)`
Return a list of the VMs that are on the provider

`salt.cloud.clouds.digital_ocean.`**`list_nodes_select`**`(`*call=None*`)`
Return a list of the VMs that are on the provider, with select fields

`salt.cloud.clouds.digital_ocean.`**`query`**`(`*method='droplets'*, *droplet_id=None*, *command=None*, *args=None*`)`
Make a web call to Digital Ocean

`salt.cloud.clouds.digital_ocean.`**`script`**`(`*vm_*`)`
Return the script deployment object

`salt.cloud.clouds.digital_ocean.`**`show_instance`**`(`*name*, *call=None*`)`
Show the details from Digital Ocean concerning a droplet

`salt.cloud.clouds.digital_ocean.`**`show_keypair`**`(`*kwargs=None*, *call=None*`)`
Show the details of an SSH keypair

93.4 salt.cloud.clouds.ec2

93.4.1 The EC2 Cloud Module

The EC2 cloud module is used to interact with the Amazon Elastic Cloud Computing. This driver is highly experimental! Use at your own risk!

To use the EC2 cloud module, set up the cloud configuration at `/etc/salt/cloud.providers` or `/etc/salt/cloud.providers.d/ec2.conf`:

```
my-ec2-config:
  # The EC2 API authentication id
  id: GKTADJGHEIQSXMKKRBJ08H
  # The EC2 API authentication key
  key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
  # The ssh keyname to use
  keyname: default
  # The amazon security group
  securitygroup: ssh_open
  # The location of the private key which corresponds to the keyname
  private_key: /root/default.pem

  # Be default, service_url is set to amazonaws.com. If you are using this
  # driver for something other than Amazon EC2, change it here:
  service_url: amazonaws.com

  # The endpoint that is ultimately used is usually formed using the region
  # and the service_url. If you would like to override that entirely, you
  # can explicitly define the endpoint:
  endpoint: myendpoint.example.com:1138/services/Cloud

  provider: ec2
```

`salt.cloud.clouds.ec2.`**`attach_volume`**(*name=None*, *kwargs=None*, *instance_id=None*, *call=None*)
Attach a volume to an instance

`salt.cloud.clouds.ec2.`**`avail_images`**(*kwargs=None*, *call=None*)
Return a dict of all available VM images on the cloud provider.

`salt.cloud.clouds.ec2.`**`avail_locations`**(*call=None*)
List all available locations

`salt.cloud.clouds.ec2.`**`avail_sizes`**(*call=None*)
Return a dict of all available VM sizes on the cloud provider with relevant data. Latest version can be found at:

http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instance-types.html

`salt.cloud.clouds.ec2.`**`block_device_mappings`**(*vm_*)
Return the block device mapping:

```
[{'DeviceName': '/dev/sdb', 'VirtualName': 'ephemeral0'},
  {'DeviceName': '/dev/sdc', 'VirtualName': 'ephemeral1'}]
```

`salt.cloud.clouds.ec2.`**`create`**(*vm_=None*, *call=None*)
Create a single VM from a data dict

`salt.cloud.clouds.ec2.`**`create_attach_volumes`**(*name*, *kwargs*, *call=None*)
Create and attach volumes to created node

`salt.cloud.clouds.ec2.`**`create_keypair`**(*kwargs=None*, *call=None*)
Create an SSH keypair

`salt.cloud.clouds.ec2.`**`create_volume`**(*kwargs=None*, *call=None*)
Create a volume

`salt.cloud.clouds.ec2.`**`del_tags`**(*name*, *kwargs*, *call=None*)
Delete tags for a node

CLI Example:

```
salt-cloud -a del_tags mymachine tag1,tag2,tag3
```

`salt.cloud.clouds.ec2.`**`delete_keypair`**(*kwargs=None*, *call=None*)

Delete an SSH keypair

`salt.cloud.clouds.ec2.`**`delete_volume`**(*name=None*, *kwargs=None*, *instance_id=None*, *call=None*)

Delete a volume

`salt.cloud.clouds.ec2.`**`delvol_on_destroy`**(*name*, *kwargs=None*, *call=None*)

Delete all/specified EBS volumes upon instance termination

CLI Example:

```
salt-cloud -a delvol_on_destroy mymachine
```

`salt.cloud.clouds.ec2.`**`destroy`**(*name*, *call=None*)

Destroy a node. Will check termination protection and warn if enabled.

CLI Example:

```
salt-cloud --destroy mymachine
```

`salt.cloud.clouds.ec2.`**`detach_volume`**(*name=None*, *kwargs=None*, *instance_id=None*, *call=None*)

Detach a volume from an instance

`salt.cloud.clouds.ec2.`**`disable_term_protect`**(*name*, *call=None*)

Disable termination protection on a node

CLI Example:

```
salt-cloud -a disable_term_protect mymachine
```

`salt.cloud.clouds.ec2.`**`enable_term_protect`**(*name*, *call=None*)

Enable termination protection on a node

CLI Example:

```
salt-cloud -a enable_term_protect mymachine
```

`salt.cloud.clouds.ec2.`**`get_availability_zone`**(*vm_*)

Return the availability zone to use

`salt.cloud.clouds.ec2.`**`get_configured_provider`**()

Return the first configured instance.

`salt.cloud.clouds.ec2.`**`get_location`**(*vm_=None*)

Return the EC2 region to use, in this order:

- CLI parameter
- VM parameter
- Cloud profile setting

`salt.cloud.clouds.ec2.`**`get_spot_config`**(*vm_*)

Returns the spot instance configuration for the provided vm

`salt.cloud.clouds.ec2.`**`get_subnetid`**(*vm_*)

Returns the SubnetId to use

`salt.cloud.clouds.ec2.`**`get_tags`**`(`*name=None*, *instance_id=None*, *call=None*, *location=None*`)`
Retrieve tags for a node

`salt.cloud.clouds.ec2.`**`get_tenancy`**`(`*vm_*`)`
Returns the Tenancy to use.

Can be "dedicated" or "default". Cannot be present for spot instances.

`salt.cloud.clouds.ec2.`**`iam_profile`**`(`*vm_*`)`
Return the IAM profile.

The IAM instance profile to associate with the instances. This is either the Amazon Resource Name (ARN) of the instance profile or the name of the role.

Type: String

Default: None

Required: No

Example: arn:aws:iam::111111111111:instance-profile/s3access

Example: s3access

`salt.cloud.clouds.ec2.`**`keepvol_on_destroy`**`(`*name*, *kwargs=None*, *call=None*`)`
Do not delete all/specified EBS volumes upon instance termination

CLI Example:

```
salt-cloud -a keepvol_on_destroy mymachine
```

`salt.cloud.clouds.ec2.`**`keyname`**`(`*vm_*`)`
Return the keyname

`salt.cloud.clouds.ec2.`**`list_availability_zones`**`()`
List all availability zones in the current region

`salt.cloud.clouds.ec2.`**`list_nodes`**`(`*call=None*`)`
Return a list of the VMs that are on the provider

`salt.cloud.clouds.ec2.`**`list_nodes_full`**`(`*location=None*, *call=None*`)`
Return a list of the VMs that are on the provider

`salt.cloud.clouds.ec2.`**`list_nodes_select`**`(`*call=None*`)`
Return a list of the VMs that are on the provider, with select fields

`salt.cloud.clouds.ec2.`**`query`**`(`*params=None*, *setname=None*, *requesturl=None*, *location=None*, *return_url=False*, *return_root=False*`)`

`salt.cloud.clouds.ec2.`**`reboot`**`(`*name*, *call=None*`)`
Reboot a node.

CLI Example:

```
salt-cloud -a reboot mymachine
```

`salt.cloud.clouds.ec2.`**`rename`**`(`*name*, *kwargs*, *call=None*`)`
Properly rename a node. Pass in the new name as "new name".

CLI Example:

```
salt-cloud -a rename mymachine newname=yourmachine
```

`salt.cloud.clouds.ec2.`**`script`**`(`*vm_*`)`
Return the script deployment object

salt.cloud.clouds.ec2.**securitygroup**(*vm_*)
Return the security group

salt.cloud.clouds.ec2.**securitygroupid**(*vm_*)
Returns the SecurityGroupId

salt.cloud.clouds.ec2.**set_tags**(*name*, *tags*, *call=None*, *location=None*, *instance_id=None*)
Set tags for a node

CLI Example:

```
salt-cloud -a set_tags mymachine tag1=somestuff tag2='Other stuff'
```

salt.cloud.clouds.ec2.**show_delvol_on_destroy**(*name*, *kwargs=None*, *call=None*)
Do not delete all/specified EBS volumes upon instance termination

CLI Example:

```
salt-cloud -a show_delvol_on_destroy mymachine
```

salt.cloud.clouds.ec2.**show_image**(*kwargs*, *call=None*)
Show the details from EC2 concerning an AMI

salt.cloud.clouds.ec2.**show_instance**(*name*, *call=None*)
Show the details from EC2 concerning an AMI

salt.cloud.clouds.ec2.**show_keypair**(*kwargs=None*, *call=None*)
Show the details of an SSH keypair

salt.cloud.clouds.ec2.**show_term_protect**(*name=None*, *instance_id=None*, *call=None*, *quiet=False*)
Show the details from EC2 concerning an AMI

salt.cloud.clouds.ec2.**show_volume**(*name=None*, *kwargs=None*, *instance_id=None*, *call=None*)
Show volume details

salt.cloud.clouds.ec2.**ssh_interface**(*vm_*)
Return the ssh_interface type to connect to. Either 'public_ips' (default) or 'private_ips'.

salt.cloud.clouds.ec2.**ssh_username**(*vm_*)
Return the ssh_username. Defaults to a built-in list of users for trying.

salt.cloud.clouds.ec2.**start**(*name*, *call=None*)
Start a node

salt.cloud.clouds.ec2.**stop**(*name*, *call=None*)
Stop a node

93.5 salt.cloud.clouds.gce

Copyright 2013 Google Inc. All Rights Reserved.

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

93.5.1 Google Compute Engine Module

The Google Compute Engine module. This module interfaces with Google Compute Engine. To authenticate to GCE, you will need to create a Service Account.

Setting up Service Account Authentication:

- Go to the Cloud Console at: https://cloud.google.com/console.
- Create or navigate to your desired Project.
- Make sure Google Compute Engine service is enabled under the Services section.
- Go to "APIs and auth" and then the "Registered apps" section.
- Click the "REGISTER APP" button and give it a meaningful name.
- Select "Web Application" and click "Register".
- Select Certificate, then "Generate Certificate"
- Copy the Email Address for inclusion in your /etc/salt/cloud file in the 'service_account_email_address' setting.
- Download the Private Key
- The key that you download is a PKCS12 key. It needs to be converted to the PEM format.
- Convert the key using OpenSSL (the default password is 'notasecret'): C{openssl pkcs12 -in PRIVKEY.p12 -passin pass:notasecret -nodes -nocerts | openssl rsa -out ~/PRIVKEY.pem}
- Add the full path name of the converted private key to your /etc/salt/cloud file as 'service_account_private_key' setting.
- Consider using a more secure location for your private key.

Supported commands: # Create a few instances fro profile_name in /etc/salt/cloud.profiles - salt-cloud -p profile_name inst1 inst2 inst3 # Delete an instance - salt-cloud -d inst1 # Look up data on an instance - salt-cloud -a show_instance inst2 # List available locations (aka 'zones') for provider 'gce' - salt-cloud –list-locations gce # List available instance sizes (aka 'machine types') for provider 'gce' - salt-cloud –list-sizes gce # List available images for provider 'gce' - salt-cloud –list-images gce

```
my-gce-config:
  # The Google Cloud Platform Project ID
  project: google.com:erjohnso
  # The Service ACcount client ID
  service_account_email_address: 1234567890@developer.gserviceaccount.com
  # The location of the private key (PEM format)
  service_account_private_key: /home/erjohnso/PRIVKEY.pem
  provider: gce
```

maintainer Eric Johnson <erjohnso@google.com>

maturity new

depends libcloud >= 0.14.0-beta3

depends pycrypto >= 2.1

`salt.cloud.clouds.gce.`**`avail_images`**`(`*conn=None*`)`

Return a dict of all available VM images on the cloud provider with relevant data

Note that for GCE, there are custom images within the project, but the generic images are in other projects. This returns a dict of images in the project plus images in 'debian-cloud' and 'centos-cloud' (If there is overlap in names, the one in the current project is used.)

`salt.cloud.clouds.gce.`**`avail_locations`**`(`*conn=None*, *call=None*`)`
Return a dict of all available VM locations on the cloud provider with relevant data

`salt.cloud.clouds.gce.`**`avail_sizes`**`(`*conn=None*`)`
Return a dict of available instances sizes (a.k.a machine types) and convert them to something more serializable.

`salt.cloud.clouds.gce.`**`create`**`(`*vm_=None*, *call=None*`)`
Create a single GCE instance from a data dict.

`salt.cloud.clouds.gce.`**`destroy`**`(`*vm_name*, *call=None*`)`
Call 'destroy' on the instance. Can be called with "-a destroy" or -d

`salt.cloud.clouds.gce.`**`get_configured_provider`**`()`
Return the first configured instance.

`salt.cloud.clouds.gce.`**`get_conn`**`()`
Return a conn object for the passed VM data

`salt.cloud.clouds.gce.`**`list_nodes`**`(`*conn=None*, *call=None*`)`
Return a list of the VMs that are on the provider

`salt.cloud.clouds.gce.`**`list_nodes_full`**`(`*conn=None*, *call=None*`)`
Return a list of the VMs that are on the provider, with all fields

`salt.cloud.clouds.gce.`**`list_nodes_select`**`(`*conn=None*, *call=None*`)`
Return a list of the VMs that are on the provider, with select fields

`salt.cloud.clouds.gce.`**`reboot`**`(`*vm_name*, *call=None*`)`
Call GCE 'reset' on the instance.

`salt.cloud.clouds.gce.`**`script`**`(`*vm_*`)`
Return the script deployment object

`salt.cloud.clouds.gce.`**`show_instance`**`(`*vm_name*, *call=None*`)`
Show the details of the existing instance.

93.6 salt.cloud.clouds.gogrid

93.6.1 GoGrid Cloud Module

The GoGrid cloud module. This module interfaces with the gogrid public cloud service. To use Salt Cloud with GoGrid log into the GoGrid web interface and create an api key. Do this by clicking on "My Account" and then going to the API Keys tab.

Set up the cloud configuration at `/etc/salt/cloud.providers` or `/etc/salt/cloud.providers.d/gogrid.conf`:

```
my-gogrid-config:
  # The generated api key to use
  apikey: asdff7896asdh789
  # The apikey's shared secret
  sharedsecret: saltybacon

  provider: gogrid
```

`salt.cloud.clouds.gogrid.`**`avail_images`**`(`*conn=None*, *call=None*`)`
Return a dict of all available VM images on the cloud provider with relevant data

`salt.cloud.clouds.gogrid.`**`avail_sizes`**`(`*conn=None*, *call=None*`)`
Return a dict of all available VM images on the cloud provider with relevant data

`salt.cloud.clouds.gogrid.`**`create`**`(`*vm_*`)`
Create a single VM from a data dict

`salt.cloud.clouds.gogrid.`**`destroy`**`(`*name*, *conn=None*, *call=None*`)`
Delete a single VM

`salt.cloud.clouds.gogrid.`**`get_configured_provider`**`()`
Return the first configured instance.

`salt.cloud.clouds.gogrid.`**`get_conn`**`()`
Return a conn object for the passed VM data

`salt.cloud.clouds.gogrid.`**`get_image`**`(`*conn*, *vm_*`)`
Return the image object to use

`salt.cloud.clouds.gogrid.`**`get_size`**`(`*conn*, *vm_*`)`
Return the VM's size object

`salt.cloud.clouds.gogrid.`**`list_nodes`**`(`*conn=None*, *call=None*`)`
Return a list of the VMs that are on the provider

`salt.cloud.clouds.gogrid.`**`list_nodes_full`**`(`*conn=None*, *call=None*`)`
Return a list of the VMs that are on the provider, with all fields

`salt.cloud.clouds.gogrid.`**`list_nodes_select`**`(`*conn=None*, *call=None*`)`
Return a list of the VMs that are on the provider, with select fields

`salt.cloud.clouds.gogrid.`**`script`**`(`*vm_*`)`
Return the script deployment object

`salt.cloud.clouds.gogrid.`**`show_instance`**`(`*name*, *call=None*`)`
Show the details from the provider concerning an instance

93.7 salt.cloud.clouds.ibmsce

93.7.1 IBM SCE Cloud Module

The IBM SCE cloud module. This module interfaces with the IBM SCE public cloud service. To use Salt Cloud with IBM SCE log into the IBM SCE web interface and create an SSH key.

Set up the cloud configuration at `/etc/salt/cloud.providers` or `/etc/salt/cloud.providers.d/ibmsce.conf`:

```
my-imbsce-config:
  # The generated api key to use
  user: myuser@mycompany.com
  # The user's password
  password: saltybacon
  # The name of the ssh key to use
  ssh_key_name: mykey
  # The ID of the datacenter to use
  location: Raleigh

  provider: ibmsce
```

`salt.cloud.clouds.ibmsce.`**`avail_images`**`(`*conn=None*, *call=None*`)`
Return a dict of all available VM images on the cloud provider with relevant data

`salt.cloud.clouds.ibmsce.`**`avail_locations`**`(`*conn=None*, *call=None*`)`
Return a dict of all available VM locations on the cloud provider with relevant data

`salt.cloud.clouds.ibmsce.`**`avail_sizes`**`(`*conn=None*, *call=None*`)`
Return a dict of all available VM images on the cloud provider with relevant data

`salt.cloud.clouds.ibmsce.`**`create`**`(`*vm_*`)`
Create a single VM from a data dict

`salt.cloud.clouds.ibmsce.`**`destroy`**`(`*name*, *conn=None*, *call=None*`)`
Delete a single VM

`salt.cloud.clouds.ibmsce.`**`get_configured_provider`**`()`
Return the first configured instance.

`salt.cloud.clouds.ibmsce.`**`get_conn`**`()`
Return a conn object for the passed VM data

`salt.cloud.clouds.ibmsce.`**`get_image`**`(`*conn*, *vm_*`)`
Return the image object to use

`salt.cloud.clouds.ibmsce.`**`get_location`**`(`*conn*, *vm_*`)`
Return the location object to use

`salt.cloud.clouds.ibmsce.`**`get_size`**`(`*conn*, *vm_*`)`
Return the VM's size object

`salt.cloud.clouds.ibmsce.`**`list_nodes`**`(`*conn=None*, *call=None*`)`
Return a list of the VMs that are on the provider

`salt.cloud.clouds.ibmsce.`**`list_nodes_full`**`(`*conn=None*, *call=None*`)`
Return a list of the VMs that are on the provider, with all fields

`salt.cloud.clouds.ibmsce.`**`list_nodes_select`**`(`*conn=None*, *call=None*`)`
Return a list of the VMs that are on the provider, with select fields

`salt.cloud.clouds.ibmsce.`**`script`**`(`*vm_*`)`
Return the script deployment object

`salt.cloud.clouds.ibmsce.`**`show_instance`**`(`*name*, *call=None*`)`
Show the details from the provider concerning an instance

`salt.cloud.clouds.ibmsce.`**`ssh_interface`**`(`*vm_*`)`
Return the ssh_interface type to connect to. Either 'public_ips' (default) or 'private_ips'.

93.8 salt.cloud.clouds.joyent

93.8.1 Joyent Cloud Module

The Joyent Cloud module is used to interact with the Joyent cloud system.

Set up the cloud configuration at `/etc/salt/cloud.providers` or `/etc/salt/cloud.providers.d/joyent.conf`:

```
my-joyent-config:
  # The Joyent login user
  user: fred
  # The Joyent user's password
  password: saltybacon
  # The location of the ssh private key that can log into the new VM
  private_key: /root/joyent.pem
  provider: joyent
```

When creating your profiles for the joyent cloud, add the location attribute to the profile, this will automatically get picked up when performing tasks associated with that vm. An example profile might look like:

```
joyent_512:
  provider: my-joyent-config
  size: Extra Small 512 MB
  image: centos-6
  location: us-east-1
```

salt.cloud.clouds.joyent.**avail_images**(*call=None*)
get list of available images

CLI Example:

```
salt-cloud --list-images
```

salt.cloud.clouds.joyent.**avail_locations**(*call=None*)
List all available locations

salt.cloud.clouds.joyent.**avail_sizes**(*call=None*)
get list of available packages

CLI Example:

```
salt-cloud --list-sizes
```

salt.cloud.clouds.joyent.**create**(*vm_*)
Create a single VM from a data dict

CLI Example:

```
salt-cloud -p profile_name vm_name
```

salt.cloud.clouds.joyent.**create_node**(***kwargs*)
convenience function to make the rest api call for node creation.

salt.cloud.clouds.joyent.**delete_key**(*kwargs=None*, *call=None*)
List the keys available

CLI Example:

```
salt-cloud -f delete_key joyent keyname=mykey
```

salt.cloud.clouds.joyent.**destroy**(*name*, *call=None*)
destroy a machine by name

Parameters

- **name** – name given to the machine
- **call** – call value in this case is 'action'

Returns array of booleans , true if successful;ly stopped and true if successfully removed

CLI Example:

```
salt-cloud -d vm_name
```

salt.cloud.clouds.joyent.**get_configured_provider**()
Return the first configured instance.

salt.cloud.clouds.joyent.**get_image**(*vm_*)
Return the image object to use

salt.cloud.clouds.joyent.**get_location**(*vm_=None*)

Return the joyent datacenter to use, in this order:

- CLI parameter
- VM parameter
- Cloud profile setting

`salt.cloud.clouds.joyent.`**`get_location_path`**(*location='us-east-1'*)
create url from location variable :param location: joyent datacenter location :return: url

`salt.cloud.clouds.joyent.`**`get_node`**(*name*)
gets the node from the full node list by name :param name: name of the vm :return: node object

`salt.cloud.clouds.joyent.`**`get_size`**(*vm_*)
Return the VM's size object

`salt.cloud.clouds.joyent.`**`has_method`**(*obj*, *method_name*)
Find if the provided object has a specific method

`salt.cloud.clouds.joyent.`**`import_key`**(*kwargs=None*, *call=None*)
List the keys available

CLI Example:

```
salt-cloud -f import_key joyent keyname=mykey keyfile=/tmp/mykey.pub
```

`salt.cloud.clouds.joyent.`**`joyent_node_state`**(*id_*)
Convert joyent returned state to state common to other datacenter return values for consistency

Parameters **id** – joyent state value

Returns libcloudfuncs state value

`salt.cloud.clouds.joyent.`**`key_list`**(*key='name'*, *items=None*)
convert list to dictionary using the key as the identifier :param key: identifier - must exist in the arrays elements own dictionary :param items: array to iterate over :return: dictionary

`salt.cloud.clouds.joyent.`**`list_keys`**(*kwargs=None*, *call=None*)
List the keys available

`salt.cloud.clouds.joyent.`**`list_nodes`**(*full=False*, *call=None*)
list of nodes, keeping only a brief listing

CLI Example:

```
salt-cloud -Q
```

`salt.cloud.clouds.joyent.`**`list_nodes_full`**(*call=None*)
list of nodes, maintaining all content provided from joyent listings

CLI Example:

```
salt-cloud -F
```

`salt.cloud.clouds.joyent.`**`list_nodes_select`**(*call=None*)
Return a list of the VMs that are on the provider, with select fields

`salt.cloud.clouds.joyent.`**`query`**(*action=None*, *command=None*, *args=None*, *method='GET'*, *data=None*, *headers=None*)
Make a web call to Joyent

`salt.cloud.clouds.joyent.`**`query2`**(*action=None*, *command=None*, *args=None*, *method='GET'*, *location=None*, *data=None*)
Make a web call to Joyent

`salt.cloud.clouds.joyent.`**`reboot`**(*name*, *call=None*)

reboot a machine by name :param name: name given to the machine :param call: call value in this case is 'action' :return: true if successful

CLI Example:

```
salt-cloud -a reboot vm_name
```

`salt.cloud.clouds.joyent.`**`reformat_node`**(*item=None*, *full=False*)

Reformat the returned data from joyent, determine public/private IPs and strip out fields if necessary to provide either full or brief content.

Parameters

- **item** – node dictionary
- **full** – full or brief output

Returns dict

`salt.cloud.clouds.joyent.`**`show_key`**(*kwargs=None*, *call=None*)

List the keys available

`salt.cloud.clouds.joyent.`**`ssh_interface`**(*vm_*)

Return the ssh_interface type to connect to. Either 'public_ips' (default) or 'private_ips'.

`salt.cloud.clouds.joyent.`**`start`**(*name*, *call=None*)

start a machine by name :param name: name given to the machine :param call: call value in this case is 'action' :return: true if successful

CLI Example:

```
salt-cloud -a start vm_name
```

`salt.cloud.clouds.joyent.`**`stop`**(*name*, *call=None*)

stop a machine by name :param name: name given to the machine :param call: call value in this case is 'action' :return: true if successful

CLI Example:

```
salt-cloud -a stop vm_name
```

`salt.cloud.clouds.joyent.`**`take_action`**(*name=None*, *call=None*, *command=None*, *data=None*, *method='GET'*, *location='us-east-1'*)

take action call used by start,stop, reboot :param name: name given to the machine :param call: call value in this case is 'action' :command: api path :data: any data to be passed to the api, must be in json format :method: GET,POST,or DELETE :location: datacenter to execute the command on :return: true if successful

93.9 salt.cloud.clouds.libcloud_aws

93.9.1 The AWS Cloud Module

The AWS cloud module is used to interact with the Amazon Web Services system.

This module has been replaced by the EC2 cloud module, and is no longer supported. The documentation shown here is for reference only; it is highly recommended to change all usages of this driver over to the EC2 driver.

If this driver is still needed, set up the cloud configuration at `/etc/salt/cloud.providers` or `/etc/salt/cloud.providers.d/aws.conf`:

```
my-aws-config:
  # The AWS API authentication id
  id: GKTADJGHEIQSXMKKRBJ08H
  # The AWS API authentication key
  key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
  # The ssh keyname to use
  keyname: default
  # The amazon security group
  securitygroup: ssh_open
  # The location of the private key which corresponds to the keyname
  private_key: /root/default.pem

  provider: aws
```

`salt.cloud.clouds.libcloud_aws.`**`block_device_mappings`**(*vm_*)
Return the block device mapping:

```
[{'DeviceName': '/dev/sdb', 'VirtualName': 'ephemeral0'},
  {'DeviceName': '/dev/sdc', 'VirtualName': 'ephemeral1'}]
```

`salt.cloud.clouds.libcloud_aws.`**`create`**(*vm_*)
Create a single VM from a data dict

`salt.cloud.clouds.libcloud_aws.`**`create_attach_volumes`**(*volumes*, *location*, *data*)
Create and attach volumes to created node

`salt.cloud.clouds.libcloud_aws.`**`del_tags`**(*name*, *kwargs*, *call=None*)
Delete tags for a node

CLI Example:

```
salt-cloud -a del_tags mymachine tag1,tag2,tag3
```

`salt.cloud.clouds.libcloud_aws.`**`destroy`**(*name*)
Wrap core libcloudfuncs destroy method, adding check for termination protection

`salt.cloud.clouds.libcloud_aws.`**`get_availability_zone`**(*conn*, *vm_*)
Return the availability zone to use

`salt.cloud.clouds.libcloud_aws.`**`get_configured_provider`**()
Return the first configured instance.

`salt.cloud.clouds.libcloud_aws.`**`get_conn`**(***kwargs*)
Return a conn object for the passed VM data

`salt.cloud.clouds.libcloud_aws.`**`get_location`**(*vm_=None*)

Return the AWS region to use, in this order:

- CLI parameter
- Cloud profile setting
- Global salt-cloud config

`salt.cloud.clouds.libcloud_aws.`**`get_tags`**(*name*, *call=None*)
Retrieve tags for a node

`salt.cloud.clouds.libcloud_aws.`**`iam_profile`**(*vm_*)
Return the IAM role

`salt.cloud.clouds.libcloud_aws.`**`keyname`**(*vm_*)
Return the keyname

`salt.cloud.clouds.libcloud_aws.`**`rename`**(*name*, *kwargs*, *call=None*)
Properly rename a node. Pass in the new name as "new name".

CLI Example:

```
salt-cloud -a rename mymachine newname=yourmachine
```

`salt.cloud.clouds.libcloud_aws.`**`securitygroup`**(*vm_*)
Return the security group

`salt.cloud.clouds.libcloud_aws.`**`set_tags`**(*name*, *tags*, *call=None*)
Set tags for a node

CLI Example:

```
salt-cloud -a set_tags mymachine tag1=somestuff tag2='Other stuff'
```

`salt.cloud.clouds.libcloud_aws.`**`ssh_interface`**(*vm_*)
Return the ssh_interface type to connect to. Either 'public_ips' (default) or 'private_ips'.

`salt.cloud.clouds.libcloud_aws.`**`ssh_username`**(*vm_*)
Return the ssh_username. Defaults to 'ec2-user'.

`salt.cloud.clouds.libcloud_aws.`**`start`**(*name*, *call=None*)
Start a node

`salt.cloud.clouds.libcloud_aws.`**`stop`**(*name*, *call=None*)
Stop a node

93.10 salt.cloud.clouds.linode

93.10.1 Linode Cloud Module

The Linode cloud module is used to control access to the Linode VPS system

Use of this module only requires the `apikey` parameter.

Set up the cloud configuration at `/etc/salt/cloud.providers` or `/etc/salt/cloud.providers.d/linode.conf`:

```
my-linode-config:
  # Linode account api key
  apikey: JVkbSJDGHSDKUKSDJfhsdklfjgsjdkflhjlsdfffhgdgjkenrtuinv
  provider: linode
```

`salt.cloud.clouds.linode.`**`avail_images`**(*conn=None*, *call=None*)
Return a dict of all available VM images on the cloud provider with relevant data

`salt.cloud.clouds.linode.`**`avail_locations`**(*conn=None*, *call=None*)
Return a dict of all available VM locations on the cloud provider with relevant data

`salt.cloud.clouds.linode.`**`avail_sizes`**(*conn=None*, *call=None*)
Return a dict of all available VM images on the cloud provider with relevant data

`salt.cloud.clouds.linode.`**`create`**(*vm_*)
Create a single VM from a data dict

`salt.cloud.clouds.linode.`**`destroy`**(*name*, *conn=None*, *call=None*)
Delete a single VM

`salt.cloud.clouds.linode.`**`get_configured_provider`**()
Return the first configured instance.

`salt.cloud.clouds.linode.`**`get_conn`**`()`
Return a conn object for the passed VM data

`salt.cloud.clouds.linode.`**`get_image`**`(`*conn*, *vm_*`)`
Return the image object to use

`salt.cloud.clouds.linode.`**`get_location`**`(`*conn*, *vm_*`)`
Return the node location to use

`salt.cloud.clouds.linode.`**`get_password`**`(`*vm_*`)`
Return the password to use

`salt.cloud.clouds.linode.`**`get_size`**`(`*conn*, *vm_*`)`
Return the VM's size object

`salt.cloud.clouds.linode.`**`list_nodes`**`(`*conn=None*, *call=None*`)`
Return a list of the VMs that are on the provider

`salt.cloud.clouds.linode.`**`list_nodes_full`**`(`*conn=None*, *call=None*`)`
Return a list of the VMs that are on the provider, with all fields

`salt.cloud.clouds.linode.`**`list_nodes_select`**`(`*conn=None*, *call=None*`)`
Return a list of the VMs that are on the provider, with select fields

`salt.cloud.clouds.linode.`**`script`**`(`*vm_*`)`
Return the script deployment object

`salt.cloud.clouds.linode.`**`show_instance`**`(`*name*, *call=None*`)`
Show the details from the provider concerning an instance

93.11 salt.cloud.clouds.msazure

93.11.1 Azure Cloud Module

The Azure cloud module is used to control access to Microsoft Azure

Use of this module only requires the `apikey` parameter. Set up the cloud configuration at `/etc/salt/cloud.providers` or `/etc/salt/cloud.providers.d/azure.conf`:

```
my-azure-config:
  provider: azure
  subscription_id: 3287abc8-f98a-c678-3bde-326766fd3617
  certificate_path: /etc/salt/azure.pem
```

Information on creating the pem file to use, and uploading the associated cer file can be found at:

http://www.windowsazure.com/en-us/develop/python/how-to-guides/service-management/

`salt.cloud.clouds.msazure.`**`avail_images`**`(`*conn=None*, *call=None*`)`
List available images for Azure

`salt.cloud.clouds.msazure.`**`avail_locations`**`(`*conn=None*, *call=None*`)`
List available locations for Azure

`salt.cloud.clouds.msazure.`**`avail_sizes`**`(`*call=None*`)`
Because sizes are built into images with Azure, there will be no sizes to return here

`salt.cloud.clouds.msazure.`**`create`**`(`*vm_*`)`
Create a single VM from a data dict

`salt.cloud.clouds.msazure.`**`destroy`**(*name*, *conn=None*, *call=None*)
Destroy a VM

`salt.cloud.clouds.msazure.`**`get_configured_provider`**()
Return the first configured instance.

`salt.cloud.clouds.msazure.`**`get_conn`**()
Return a conn object for the passed VM data

`salt.cloud.clouds.msazure.`**`list_disks`**(*conn=None*, *call=None*)
Destroy a VM

`salt.cloud.clouds.msazure.`**`list_hosted_services`**(*conn=None*, *call=None*)
List VMs on this Azure account, with full information

`salt.cloud.clouds.msazure.`**`list_nodes`**(*conn=None*, *call=None*)
List VMs on this Azure account

`salt.cloud.clouds.msazure.`**`list_nodes_full`**(*conn=None*, *call=None*)
List VMs on this Azure account, with full information

`salt.cloud.clouds.msazure.`**`list_nodes_select`**(*conn=None*, *call=None*)
Return a list of the VMs that are on the provider, with select fields

`salt.cloud.clouds.msazure.`**`list_storage_services`**(*conn=None*, *call=None*)
List VMs on this Azure account, with full information

`salt.cloud.clouds.msazure.`**`script`**(*vm_*)
Return the script deployment object

`salt.cloud.clouds.msazure.`**`show_instance`**(*name*, *call=None*)
Show the details from the provider concerning an instance

93.12 salt.cloud.clouds.nova

93.12.1 OpenStack Nova Cloud Module

PLEASE NOTE: This module is currently in early development, and considered to be experimental and unstable. It is not recommended for production use. Unless you are actively developing code in this module, you should use the OpenStack module instead.

OpenStack is an open source project that is in use by a number a cloud providers, each of which have their own ways of using it.

The OpenStack Nova module for Salt Cloud was bootstrapped from the OpenStack module for Salt Cloud, which uses a libcloud-based connection. The Nova module is designed to use the nova and glance modules already built into Salt.

These modules use the Python novaclient and glanceclient libraries, respectively. In order to use this module, the proper salt configuration must also be in place. This can be specified in the master config, the minion config, a set of grains or a set of pillars.

```
my_openstack_profile:
  keystone.user: admin
  keystone.password: verybadpass
  keystone.tenant: admin
  keystone.auth_url: 'http://127.0.0.1:5000/v2.0/'
```

Note that there is currently a dependency upon netaddr. This can be installed on Debian-based systems by means of the python-netaddr package.

This module currently requires the latest develop branch of Salt to be installed.

This module has been tested to work with HP Cloud and Rackspace. See the documentation for specific options for either of these providers. These examples could be set up in the cloud configuration at `/etc/salt/cloud.providers` or `/etc/salt/cloud.providers.d/openstack.conf`:

```
my-openstack-config:
  # The ID of the minion that will execute the salt nova functions
  auth_minion: myminion
  # The name of the configuration profile to use on said minion
  config_profile: my_openstack_profile

  ssh_key_name: mykey
  # The OpenStack Nova network UUIDs
  networks:
      - fixed:
          - 4402cd51-37ee-435e-a966-8245956dc0e6
      - floating:
          - Ext-Net

  provider: nova
  userdata_file: /tmp/userdata.txt
```

For local installations that only use private IP address ranges, the following option may be useful. Using the old syntax:

```
my-openstack-config:
  # Ignore IP addresses on this network for bootstrap
  ignore_cidr: 192.168.50.0/24
```

`salt.cloud.clouds.nova.`**`avail_images`**`()`
Return a dict of all available VM images on the cloud provider.

`salt.cloud.clouds.nova.`**`avail_locations`**`()`
Would normally return a list of available datacenters (ComputeRegions? Availability Zones?), but those don't seem to be available via the nova client.

`salt.cloud.clouds.nova.`**`avail_sizes`**`()`
Return a dict of all available VM sizes on the cloud provider.

`salt.cloud.clouds.nova.`**`create`**`(`*vm_*`)`
Create a single VM from a data dict

`salt.cloud.clouds.nova.`**`destroy`**`(`*name*, *conn=None*, *call=None*`)`
Delete a single VM

`salt.cloud.clouds.nova.`**`get_configured_provider`**`()`
Return the first configured instance.

`salt.cloud.clouds.nova.`**`get_conn`**`()`
Return a conn object for the passed VM data

`salt.cloud.clouds.nova.`**`get_image`**`(`*conn*, *vm_*`)`
Return the image object to use

`salt.cloud.clouds.nova.`**`get_size`**`(`*conn*, *vm_*`)`
Return the VM's size object

`salt.cloud.clouds.nova.`**`ignore_cidr`**`(`*vm_*, *ip*`)`
Return True if we are to ignore the specified IP. Compatible with IPv4.

`salt.cloud.clouds.nova.`**`list_nodes`**(*call=None*)
Return a list of the VMs that in this location

`salt.cloud.clouds.nova.`**`list_nodes_full`**(*call=None*)
Return a list of the VMs that in this location

`salt.cloud.clouds.nova.`**`list_nodes_select`**(*call=None*)
Return a list of the VMs that are on the provider, with select fields

`salt.cloud.clouds.nova.`**`managedcloud`**(*vm_*)
Determine if we should wait for the managed cloud automation before running. Either 'False' (default) or 'True'.

`salt.cloud.clouds.nova.`**`preferred_ip`**(*vm_*, *ips*)
Return the preferred Internet protocol. Either 'ipv4' (default) or 'ipv6'.

`salt.cloud.clouds.nova.`**`rackconnect`**(*vm_*)
Determine if we should wait for rackconnect automation before running. Either 'False' (default) or 'True'.

`salt.cloud.clouds.nova.`**`reboot`**(*name*, *conn=None*)
Reboot a single VM

`salt.cloud.clouds.nova.`**`script`**(*vm_*)
Return the script deployment object

`salt.cloud.clouds.nova.`**`ssh_interface`**(*vm_*)
Return the ssh_interface type to connect to. Either 'public_ips' (default) or 'private_ips'.

93.13 salt.cloud.clouds.openstack

93.13.1 OpenStack Cloud Module

OpenStack is an open source project that is in use by a number a cloud providers, each of which have their own ways of using it.

OpenStack provides a number of ways to authenticate. This module uses password- based authentication, using auth v2.0. It is likely to start supporting other methods of authentication provided by OpenStack in the future.

Note that there is currently a dependency upon netaddr. This can be installed on Debian-based systems by means of the python-netaddr package.

This module has been tested to work with HP Cloud and Rackspace. See the documentation for specific options for either of these providers. Some examples, using the old cloud configuration syntax, are provided below:

Set up in the cloud configuration at `/etc/salt/cloud.providers` or `/etc/salt/cloud.providers.d/openstack.conf`:

```
my-openstack-config:
  # The OpenStack identity service url
  identity_url: https://region-a.geo-1.identity.hpcloudsvc.com:35357/v2.0/
  # The OpenStack compute region
  compute_region: az-1.region-a.geo-1
  # The OpenStack compute service name
  compute_name: Compute
  # The OpenStack tenant name (not tenant ID)
  tenant: myuser-tenant1
  # The OpenStack user name
  user: myuser
  # The OpenStack keypair name
  ssh_key_name
  # The OpenStack network UUIDs
  networks:
      - fixed:
          - 4402cd51-37ee-435e-a966-8245956dc0e6
      - floating:
          - Ext-Net
  files:
      /path/to/dest.txt:
          /local/path/to/src.txt

  provider: openstack
  userdata_file: /tmp/userdata.txt
```

For in-house Openstack Essex installation, libcloud needs the service_type :

```
my-openstack-config:
  identity_url: 'http://control.openstack.example.org:5000/v2.0/'
  compute_name : Compute Service
  service_type : compute
```

Either a password or an API key must also be specified:

```
my-openstack-password-or-api-config:
  # The OpenStack password
  password: letmein
  # The OpenStack API key
  apikey: 901d3f579h23c8v73q9
```

For local installations that only use private IP address ranges, the following option may be useful. Using the old syntax:

```
my-openstack-config:
  # Ignore IP addresses on this network for bootstrap
  ignore_cidr: 192.168.50.0/24
```

It is possible to upload a small set of files (no more than 5, and nothing too large) to the remote server. Generally this should not be needed, as salt itself can upload to the server after it is spun up, with nowhere near the same restrictions.

```
my-openstack-config:
  files:
      /path/to/dest.txt:
          /local/path/to/src.txt
```

`salt.cloud.clouds.openstack.`**`avail_images`**`(`*conn=None*, *call=None*`)`
Return a dict of all available VM images on the cloud provider with relevant data

`salt.cloud.clouds.openstack.`**`avail_locations`**`(`*conn=None*, *call=None*`)`
Return a dict of all available VM locations on the cloud provider with relevant data

`salt.cloud.clouds.openstack.`**`avail_sizes`**`(`*conn=None*, *call=None*`)`
Return a dict of all available VM images on the cloud provider with relevant data

`salt.cloud.clouds.openstack.`**`create`**`(`*vm_*`)`
Create a single VM from a data dict

`salt.cloud.clouds.openstack.`**`destroy`**`(`*name*, *conn=None*, *call=None*`)`
Delete a single VM

`salt.cloud.clouds.openstack.`**`get_configured_provider`**`()`
Return the first configured instance.

salt.cloud.clouds.openstack.**get_conn**()
Return a conn object for the passed VM data

salt.cloud.clouds.openstack.**get_image**(*conn*, *vm_*)
Return the image object to use

salt.cloud.clouds.openstack.**get_size**(*conn*, *vm_*)
Return the VM's size object

salt.cloud.clouds.openstack.**ignore_cidr**(*vm_*, *ip*)
Return True if we are to ignore the specified IP. Compatible with IPv4.

salt.cloud.clouds.openstack.**list_nodes**(*conn=None*, *call=None*)
Return a list of the VMs that are on the provider

salt.cloud.clouds.openstack.**list_nodes_full**(*conn=None*, *call=None*)
Return a list of the VMs that are on the provider, with all fields

salt.cloud.clouds.openstack.**list_nodes_select**(*conn=None*, *call=None*)
Return a list of the VMs that are on the provider, with select fields

salt.cloud.clouds.openstack.**managedcloud**(*vm_*)
Determine if we should wait for the managed cloud automation before running. Either 'False' (default) or 'True'.

salt.cloud.clouds.openstack.**preferred_ip**(*vm_*, *ips*)
Return the preferred Internet protocol. Either 'ipv4' (default) or 'ipv6'.

salt.cloud.clouds.openstack.**rackconnect**(*vm_*)
Determine if we should wait for rackconnect automation before running. Either 'False' (default) or 'True'.

salt.cloud.clouds.openstack.**reboot**(*name*, *conn=None*)
Reboot a single VM

salt.cloud.clouds.openstack.**script**(*vm_*)
Return the script deployment object

salt.cloud.clouds.openstack.**show_instance**(*name*, *call=None*)
Show the details from the provider concerning an instance

salt.cloud.clouds.openstack.**ssh_interface**(*vm_*)
Return the ssh_interface type to connect to. Either 'public_ips' (default) or 'private_ips'.

93.14 salt.cloud.clouds.parallels

93.14.1 Parallels Cloud Module

The Parallels cloud module is used to control access to cloud providers using the Parallels VPS system.

Set up the cloud configuration at `/etc/salt/cloud.providers` or `/etc/salt/cloud.providers.d/parallels.c`

```
my-parallels-config:
  # Parallels account information
  user: myuser
  password: mypassword
  url: https://api.cloud.xmission.com:4465/paci/v1.0/
  provider: parallels
```

salt.cloud.clouds.parallels.**avail_images**(*call=None*)
Return a list of the images that are on the provider

`salt.cloud.clouds.parallels.`**`create`**(*vm_*)
Create a single VM from a data dict

`salt.cloud.clouds.parallels.`**`create_node`**(*vm_*)
Build and submit the XML to create a node

`salt.cloud.clouds.parallels.`**`destroy`**(*name*, *call=None*)
Destroy a node.

CLI Example:

```
salt-cloud --destroy mymachine
```

`salt.cloud.clouds.parallels.`**`get_configured_provider`**()
Return the first configured instance.

`salt.cloud.clouds.parallels.`**`get_image`**(*vm_*)
Return the image object to use

`salt.cloud.clouds.parallels.`**`list_nodes`**(*call=None*)
Return a list of the VMs that are on the provider

`salt.cloud.clouds.parallels.`**`list_nodes_full`**(*call=None*)
Return a list of the VMs that are on the provider

`salt.cloud.clouds.parallels.`**`list_nodes_select`**(*call=None*)
Return a list of the VMs that are on the provider, with select fields

`salt.cloud.clouds.parallels.`**`query`**(*action=None*, *command=None*, *args=None*, *method='GET'*, *data=None*)
Make a web call to a Parallels provider

`salt.cloud.clouds.parallels.`**`script`**(*vm_*)
Return the script deployment object

`salt.cloud.clouds.parallels.`**`show_image`**(*kwargs*, *call=None*)
Show the details from Parallels concerning an image

`salt.cloud.clouds.parallels.`**`show_instance`**(*name*, *call=None*)
Show the details from Parallels concerning an instance

`salt.cloud.clouds.parallels.`**`start`**(*name*, *call=None*)
Start a node.

CLI Example:

```
salt-cloud -a start mymachine
```

`salt.cloud.clouds.parallels.`**`stop`**(*name*, *call=None*)
Stop a node.

CLI Example:

```
salt-cloud -a stop mymachine
```

`salt.cloud.clouds.parallels.`**`wait_until`**(*name*, *state*, *timeout=300*)
Wait until a specific state has been reached on a node

93.15 salt.cloud.clouds.rackspace

93.15.1 Rackspace Cloud Module

The Rackspace cloud module. This module uses the preferred means to set up a libcloud based cloud module and should be used as the general template for setting up additional libcloud based modules.

Please note that the *rackspace* driver is only intended for 1st gen instances, aka, "the old cloud" at Rackspace. It is required for 1st gen instances, but will *not* work with OpenStack-based instances. Unless you explicitly have a reason to use it, it is highly recommended that you use the *openstack* driver instead.

The rackspace cloud module interfaces with the Rackspace public cloud service and requires that two configuration parameters be set for use, `user` and `apikey`.

Set up the cloud configuration at `/etc/salt/cloud.providers` or `/etc/salt/cloud.providers.d/rackspace.conf`

```
my-rackspace-config:
  provider: rackspace
  # The Rackspace login user
  user: fred
  # The Rackspace user's apikey
  apikey: 901d3f579h23c8v73q9
```

`salt.cloud.clouds.rackspace.`**`avail_images`**(*conn=None*, *call=None*)
Return a dict of all available VM images on the cloud provider with relevant data

`salt.cloud.clouds.rackspace.`**`avail_locations`**(*conn=None*, *call=None*)
Return a dict of all available VM locations on the cloud provider with relevant data

`salt.cloud.clouds.rackspace.`**`avail_sizes`**(*conn=None*, *call=None*)
Return a dict of all available VM images on the cloud provider with relevant data

`salt.cloud.clouds.rackspace.`**`create`**(*vm_*)
Create a single VM from a data dict

`salt.cloud.clouds.rackspace.`**`destroy`**(*name*, *conn=None*, *call=None*)
Delete a single VM

`salt.cloud.clouds.rackspace.`**`get_configured_provider`**()
Return the first configured instance.

`salt.cloud.clouds.rackspace.`**`get_conn`**()
Return a conn object for the passed VM data

`salt.cloud.clouds.rackspace.`**`get_image`**(*conn*, *vm_*)
Return the image object to use

`salt.cloud.clouds.rackspace.`**`get_size`**(*conn*, *vm_*)
Return the VM's size object

`salt.cloud.clouds.rackspace.`**`list_nodes`**(*conn=None*, *call=None*)
Return a list of the VMs that are on the provider

`salt.cloud.clouds.rackspace.`**`list_nodes_full`**(*conn=None*, *call=None*)
Return a list of the VMs that are on the provider, with all fields

`salt.cloud.clouds.rackspace.`**`list_nodes_select`**(*conn=None*, *call=None*)
Return a list of the VMs that are on the provider, with select fields

`salt.cloud.clouds.rackspace.`**`preferred_ip`**(*vm_*, *ips*)
Return the preferred Internet protocol. Either 'ipv4' (default) or 'ipv6'.

`salt.cloud.clouds.rackspace.`**`script`**`(`*vm_*`)`
Return the script deployment object

`salt.cloud.clouds.rackspace.`**`show_instance`**`(`*name*, *call=None*`)`
Show the details from the provider concerning an instance

`salt.cloud.clouds.rackspace.`**`ssh_interface`**`(`*vm_*`)`
Return the ssh_interface type to connect to. Either 'public_ips' (default) or 'private_ips'.

93.16 salt.cloud.clouds.saltify

93.16.1 Saltify Module

The Saltify module is designed to install Salt on a remote machine, virtual or bare metal, using SSH. This module is useful for provisioning machines which are already installed, but not Salted.

Use of this module requires no configuration in the main cloud configuration file. However, profiles must still be configured, as described in the *core config documentation*.

`salt.cloud.clouds.saltify.`**`create`**`(`*vm_*`)`
Provision a single machine

`salt.cloud.clouds.saltify.`**`get_configured_provider`**`()`
Return the first configured instance.

`salt.cloud.clouds.saltify.`**`list_nodes`**`()`
Because this module is not specific to any cloud providers, there will be no nodes to list.

`salt.cloud.clouds.saltify.`**`list_nodes_full`**`()`
Because this module is not specific to any cloud providers, there will be no nodes to list.

`salt.cloud.clouds.saltify.`**`list_nodes_select`**`()`
Because this module is not specific to any cloud providers, there will be no nodes to list.

`salt.cloud.clouds.saltify.`**`script`**`(`*vm_*`)`
Return the script deployment object

93.17 salt.cloud.clouds.softlayer

93.17.1 SoftLayer Cloud Module

The SoftLayer cloud module is used to control access to the SoftLayer VPS system.

Use of this module only requires the `apikey` parameter. Set up the cloud configuration at:

`/etc/salt/cloud.providers` or `/etc/salt/cloud.providers.d/softlayer.conf`:

```
my-softlayer-config:
  # SoftLayer account api key
  user: MYLOGIN
  apikey: JVkbSJDGHSDKUKSDJfhsdklfjgsjdkflhjlsdfffhgdgjkenrtuinv
  provider: softlayer
```

The SoftLayer Python Library needs to be installed in ordere to use the SoftLayer salt.cloud modules. See: https://pypi.python.org/pypi/SoftLayer

depends softlayer

`salt.cloud.clouds.softlayer.`**`avail_images`**(*call=None*)
Return a dict of all available VM images on the cloud provider.

`salt.cloud.clouds.softlayer.`**`avail_locations`**(*call=None*)
List all available locations

`salt.cloud.clouds.softlayer.`**`avail_sizes`**(*call=None*)
Return a dict of all available VM sizes on the cloud provider with relevant data. This data is provided in three dicts.

`salt.cloud.clouds.softlayer.`**`create`**(*vm_*)
Create a single VM from a data dict

`salt.cloud.clouds.softlayer.`**`destroy`**(*name*, *call=None*)
Destroy a node.

CLI Example:

```
salt-cloud --destroy mymachine
```

`salt.cloud.clouds.softlayer.`**`get_configured_provider`**()
Return the first configured instance.

`salt.cloud.clouds.softlayer.`**`get_conn`**(*service='SoftLayer_Virtual_Guest'*)
Return a conn object for the passed VM data

`salt.cloud.clouds.softlayer.`**`get_location`**(*vm_=None*)

Return the location to use, in this order:

- CLI parameter
- VM parameter
- Cloud profile setting

`salt.cloud.clouds.softlayer.`**`list_custom_images`**(*call=None*)
Return a dict of all custom VM images on the cloud provider.

`salt.cloud.clouds.softlayer.`**`list_nodes`**(*call=None*)
Return a list of the VMs that are on the provider

`salt.cloud.clouds.softlayer.`**`list_nodes_full`**(*mask='mask[id]'*, *call=None*)
Return a list of the VMs that are on the provider

`salt.cloud.clouds.softlayer.`**`list_nodes_select`**(*call=None*)
Return a list of the VMs that are on the provider, with select fields

`salt.cloud.clouds.softlayer.`**`list_vlans`**(*call=None*)
List all VLANs associated with the account

`salt.cloud.clouds.softlayer.`**`script`**(*vm_*)
Return the script deployment object

`salt.cloud.clouds.softlayer.`**`show_instance`**(*name*, *call=None*)
Show the details from SoftLayer concerning a guest

Python client API

There are several ways to access Salt programatically.

94.1 Calling Salt from shell scripts

Salt CLI tools can, of course, be called from shell scripts. Reference the help output to see what structured *output formats* are supported. For example:

```
salt '*' disk.usage --out=json
```

94.2 Calling Salt via a REST API

Salt provides a REST API, currently as a separate sister-project. It will be merged into Salt core.

https://github.com/saltstack/salt-api

This API utilizes Salt's Python interface documented below. It is also useful as a reference implementation.

94.3 Calling Salt from a Python application

Salt provides several entry points for interfacing with Python applications. These entry points are often referred to as `*Client()` APIs.

94.3.1 `opts`

Some clients require access to Salt's `opts` dictionary. (The dictionary representation of the *master* or *minion* config files.)

A common pattern for fetching the `opts` dictionary is to defer to environment variables if they exist or otherwise fetch the config from the default location.

```
import salt.config

master_opts = salt.config.master_config(
    os.environ.get('SALT_MASTER_CONFIG', '/etc/salt/master'))
```

```
minion_opts = salt.config.client_config(
    os.environ.get('SALT_MINION_CONFIG', '/etc/salt/minion'))
```

94.4 Salt's Python interface

94.4.1 LocalClient

class `salt.client.`**`LocalClient`**(*c_path='/etc/salt/master'*, *mopts=None*)

The interface used by the **salt** CLI tool on the Salt Master

`LocalClient` is used to send a command to Salt minions to execute *execution modules* and return the results to the Salt Master.

Importing and using `LocalClient` must be done on the same machine as the Salt Master and it must be done using the same user that the Salt Master is running as. (Unless `external_auth` is configured and authentication credentials are included in the execution).

```
import salt.client

local = salt.client.LocalClient()
local.cmd('*', 'test.fib', [10])
```

`cmd`(*tgt*, *fun*, *arg=()*, *timeout=None*, *expr_form='glob'*, *ret=''*, *kwarg=None*, ***kwargs*)

Synchronously execute a command on targeted minions

The cmd method will execute and wait for the timeout period for all minions to reply, then it will return all minion data at once.

```
>>> import salt.client
>>> local = salt.client.LocalClient()
>>> local.cmd('*', 'cmd.run', ['whoami'])
{'jerry': 'root'}
```

With extra keyword arguments for the command function to be run:

```
local.cmd('*', 'test.arg', ['arg1', 'arg2'], kwarg={'foo': 'bar'})
```

Compound commands can be used for multiple executions in a single publish. Function names and function arguments are provided in separate lists but the index values must correlate and an empty list must be used if no arguments are required.

```
>>> local.cmd('*', [
        'grains.items',
        'sys.doc',
        'cmd.run',
    ],
    [
        [],
        [],
        ['uptime'],
    ])
```

Parameters

- **tgt** (*string or list*) – Which minions to target for the execution. Default is shell glob. Modified by the `expr_form` option.

- **fun** (*string or list of strings*) – The module and function to call on the specified minions of the form `module.function`. For example `test.ping` or `grains.items`.

  Compound commands Multiple functions may be called in a single publish by passing a list of commands. This can dramatically lower overhead and speed up the application communicating with Salt.

  This requires that the `arg` param is a list of lists. The `fun` list and the `arg` list must correlate by index meaning a function that does not take arguments must still have a corresponding empty list at the expected index.

- **arg** (*list or list-of-lists*) – A list of arguments to pass to the remote function. If the function takes no arguments `arg` may be omitted except when executing a compound command.
- **timeout** – Seconds to wait after the last minion returns but before all minions return.
- **expr_form** – The type of `tgt`. Allowed values:
  - `glob` - Bash glob completion - Default
  - `pcre` - Perl style regular expression
  - `list` - Python list of hosts
  - `grain` - Match based on a grain comparison
  - `grain_pcre` - Grain comparison with a regex
  - `pillar` - Pillar data comparison
  - `nodegroup` - Match on nodegroup
  - `range` - Use a Range server for matching
  - `compound` - Pass a compound match string
- **ret** – The returner to use. The value passed can be single returner, or a comma delimited list of returners to call in order on the minions
- **kwarg** – A dictionary with keyword arguments for the function.
- **kwargs** – Optional keyword arguments.

  Authentication credentials may be passed when using `external_auth`.

  - `eauth` - the external_auth backend
  - `username` and `password`
  - `token`

```
>>> local.cmd('*', 'test.ping',
        username='saltdev', password='saltdev', eauth='pam')
```

Returns A dictionary with the result of the execution, keyed by minion ID. A compound command will return a sub-dictionary keyed by function name.

cmd_async (*tgt, fun, arg=(), expr_form='glob', ret='', kwarg=None, **kwargs*)

Asynchronously send a command to connected minions

The function signature is the same as `cmd()` with the following exceptions.

Returns A job ID or 0 on failure.

```
>>> local.cmd_async('*', 'test.sleep', [300])
'20131219215921857715'
```

cmd_iter (*tgt*, *fun*, *arg=()*, *timeout=None*, *expr_form='glob'*, *ret=''*, *kwarg=None*, ***kwargs*)
Yields the individual minion returns as they come in

The function signature is the same as `cmd()` with the following exceptions.

Returns A generator

```
>>> ret = local.cmd_iter('*', 'test.ping')
>>> for i in ret:
...     print i
{'jerry': {'ret': True}}
{'dave': {'ret': True}}
{'stewart': {'ret': True}}
```

cmd_iter_no_block (*tgt*, *fun*, *arg=()*, *timeout=None*, *expr_form='glob'*, *ret=''*, *kwarg=None*, ***kwargs*)
Blocks while waiting for individual minions to return.

The function signature is the same as `cmd()` with the following exceptions.

Returns None until the next minion returns. This allows for actions to be injected in between minion returns.

```
>>> ret = local.cmd_iter('*', 'test.ping')
>>> for i in ret:
...     print i
None
{'jerry': {'ret': True}}
{'dave': {'ret': True}}
None
{'stewart': {'ret': True}}
```

cmd_subset (*tgt*, *fun*, *arg=()*, *expr_form='glob'*, *ret=''*, *kwarg=None*, *sub=3*, *cli=False*, ***kwargs*)
Execute a command on a random subset of the targeted systems

The function signature is the same as `cmd()` with the following exceptions.

Parameters **sub** – The number of systems to execute on

```
>>> SLC.cmd_subset('*', 'test.ping', sub=1)
{'jerry': True}
```

get_cli_returns (*jid*, *minions*, *timeout=None*, *tgt='*'*, *tgt_type='glob'*, *verbose=False*, ***kwargs*)
Starts a watcher looking at the return data for a specified JID

Returns all of the information for the JID

get_event_iter_returns (*jid*, *minions*, *timeout=None*)
Gather the return data from the event system, break hard when timeout is reached.

run_job (*tgt*, *fun*, *arg=()*, *expr_form='glob'*, *ret=''*, *timeout=None*, *kwarg=None*, ***kwargs*)
Asynchronously send a command to connected minions

Prep the job directory and publish a command to any targeted minions.

Returns A dictionary of (validated) `pub_data` or an empty dictionary on failure. The `pub_data` contains the job ID and a list of all minions that are expected to return data.

```
>>> local.run_job('*', 'test.sleep', [300])
{'jid': '20131219215650131543', 'minions': ['jerry']}
```

94.4.2 Salt Caller

class `salt.client.`**`Caller`**(*c_path='/etc/salt/minion'*)

`Caller` is the same interface used by the **salt-call** command-line tool on the Salt Minion.

Importing and using `Caller` must be done on the same machine as a Salt Minion and it must be done using the same user that the Salt Minion is running as.

Usage:

```
import salt.client
caller = salt.client.Caller()
caller.function('test.ping')

# Or call objects directly
caller.sminion.functions['cmd.run']('ls -l')
```

`function`(*fun*, **args*, ***kwargs*)

Call a single salt function

94.4.3 RunnerClient

class `salt.runner.`**`RunnerClient`**(*opts*)

The interface used by the **salt-run** CLI tool on the Salt Master

It executes *runner modules* which run on the Salt Master.

Importing and using `RunnerClient` must be done on the same machine as the Salt Master and it must be done using the same user that the Salt Master is running as.

Salt's `external_auth` can be used to authenticate calls. The eauth user must be authorized to execute runner modules: (`@runner`). Only the `master_call()` below supports eauth.

`async`(*fun*, *low*, *user='UNKNOWN'*)

Execute the runner in a multiprocess and return the event tag to use to watch for the return

`cmd`(*fun*, *arg*, *kwarg=None*)

Execute a runner function

```
>>> opts = salt.config.master_config('/etc/salt/master')
>>> runner = salt.runner.RunnerClient(opts)
>>> runner.cmd('jobs.list_jobs', [])
{
    '20131219215650131543': {
        'Arguments': [300],
        'Function': 'test.sleep',
        'StartTime': '2013, Dec 19 21:56:50.131543',
        'Target': '*',
        'Target-type': 'glob',
        'User': 'saltdev'
    },
    '20131219215921857715': {
        'Arguments': [300],
        'Function': 'test.sleep',
        'StartTime': '2013, Dec 19 21:59:21.857715',
        'Target': '*',
        'Target-type': 'glob',
        'User': 'saltdev'
    },
}
```

get_docs ()
Return a dictionary of functions and the inline documentation for each

low (*fun*, *low*)
Pass in the runner function name and the low data structure

```
runner.low({'fun': 'jobs.lookup_jid', 'jid': '20131219215921857715'})
```

master_call (***kwargs*)
Execute a runner function through the master network interface (eauth).

This function requires that `external_auth` is configured and the user is authorized to execute runner functions: (`@runner`).

```
runner.master_call({
    'fun': 'jobs.list_jobs',
    'username': 'saltdev',
    'password': 'saltdev',
    'eauth': 'pam',
})
```

94.4.4 WheelClient

class `salt.wheel.`**WheelClient** (*opts=None*)
An interface to Salt's wheel modules

Wheel modules interact with various parts of the Salt Master.

Importing and using `WheelClient` must be done on the same machine as the Salt Master and it must be done using the same user that the Salt Master is running as. Unless `external_auth` is configured and the user is authorized to execute wheel functions: (`@wheel`).

call_func (*fun*, ***kwargs*)
Execute a wheel function

```
>>> opts = salt.config.master_config('/etc/salt/master')
>>> wheel = salt.wheel.Wheel(opts)
>>> wheel.call_func('key.list_all')
{'local': ['master.pem', 'master.pub'],
'minions': ['jerry'],
'minions_pre': [],
'minions_rejected': []}
```

get_docs ()
Return a dictionary of functions and the inline documentation for each

master_call (***kwargs*)
Execute a wheel function through the master network interface (eauth).

This function requires that `external_auth` is configured and the user is authorized to execute wheel functions: (`@wheel`).

```
>>> wheel.master_call(**{
    'fun': 'key.finger',
    'match': 'jerry',
    'eauth': 'auto',
    'username': 'saltdev',
    'password': 'saltdev',
})
{'data': {
    '_stamp': '2013-12-19_22:47:44.427338',
    'fun': 'wheel.key.finger',
    'jid': '20131219224744416681',
    'return': {'minions': {'jerry': '5d:f6:79:43:5e:d4:42:3f:57:b8:45:a8:7e:a4:6e:ca'}},
    'success': True,
    'tag': 'salt/wheel/20131219224744416681',
    'user': 'saltdev'
},
'tag': 'salt/wheel/20131219224744416681'}
```

94.4.5 CloudClient

class `salt.cloud.`**`CloudClient`**(*path=None*, *opts=None*, *config_dir=None*)

The client class to wrap cloud interactions

`action`(*fun=None*, *cloudmap=None*, *names=None*, *provider=None*, *instance=None*, *kwargs=None*)

Execute a single action via the cloud plugin backend

Examples:

client.action(fun='show_instance', names=['myinstance']) client.action(fun='show_image', provider='my-ec2-config', kwargs={'image': 'ami-10314d79'})

`create`(*provider*, *names*, ***kwargs*)

Create the named VMs, without using a profile

Example:

client.create(names=['myinstance'], provider='my-ec2-config',

kwargs={'image': 'ami-1624987f', 'size': 'Micro Instance', 'ssh_username': 'ec2-user', 'securitygroup': 'default', 'delvol_on_destroy': True})

`destroy`(*names*)

Destroy the named VMs

`full_query`(*query_type='list_nodes_full'*)

Query all instance information

`list_images`(*provider=None*)

List all available images in configured cloud systems

`list_locations`(*provider=None*)

List all available locations in configured cloud systems

`list_sizes`(*provider=None*)

List all available sizes in configured cloud systems

`low`(*fun*, *low*)

Pass the cloud function and low data structure to run

`profile`(*profile*, *names*, ***kwargs*)

Pass in a profile to create, names is a list of vm names to allocate

`query`(*query_type='list_nodes'*)

Query basic instance information

`select_query`(*query_type='list_nodes_select'*)

Query select instance information

CHAPTER 95

Peer Communication

Salt 0.9.0 introduced the capability for Salt minions to publish commands. The intent of this feature is not for Salt minions to act as independent brokers one with another, but to allow Salt minions to pass commands to each other.

In Salt 0.10.0 the ability to execute runners from the master was added. This allows for the master to return collective data from runners back to the minions via the peer interface.

The peer interface is configured through two options in the master configuration file. For minions to send commands from the master the `peer` configuration is used. To allow for minions to execute runners from the master the `peer_run` configuration is used.

Since this presents a viable security risk by allowing minions access to the master publisher the capability is turned off by default. The minions can be allowed access to the master publisher on a per minion basis based on regular expressions. Minions with specific ids can be allowed access to certain Salt modules and functions.

95.1 Peer Configuration

The configuration is done under the `peer` setting in the Salt master configuration file, here are a number of configuration possibilities.

The simplest approach is to enable all communication for all minions, this is only recommended for very secure environments.

```
peer:
  .*:
    - .*
```

This configuration will allow minions with IDs ending in example.com access to the test, ps, and pkg module functions.

```
peer:
  .*example.com:
    - test.*
    - ps.*
    - pkg.*
```

The configuration logic is simple, a regular expression is passed for matching minion ids, and then a list of expressions matching minion functions is associated with the named minion. For instance, this configuration will also allow minions ending with foo.org access to the publisher.

```
peer:
  .*example.com:
    - test.*
    - ps.*
    - pkg.*
  .*foo.org:
    - test.*
    - ps.*
    - pkg.*
```

95.2 Peer Runner Communication

Configuration to allow minions to execute runners from the master is done via the `peer_run` option on the master. The `peer_run` configuration follows the same logic as the `peer` option. The only difference is that access is granted to runner modules.

To open up access to all minions to all runners:

```
peer_run:
  .*:
    - .*
```

This configuration will allow minions with IDs ending in example.com access to the manage and jobs runner functions.

```
peer_run:
  .*example.com:
    - manage.*
    - jobs.*
```

95.3 Using Peer Communication

The publish module was created to manage peer communication. The publish module comes with a number of functions to execute peer communication in different ways. Currently there are three functions in the publish module. These examples will show how to test the peer system via the salt-call command.

To execute test.ping on all minions:

```
# salt-call publish.publish \* test.ping
```

To execute the manage.up runner:

```
# salt-call publish.runner manage.up
```

To match minions using other matchers, use `expr_form`:

```
# salt-call publish.publish 'webserv* and not G@os:Ubuntu' test.ping expr_form='compound'
```

CHAPTER 96

Client ACL system

The salt client ACL system is a means to allow system users other than root to have access to execute select salt commands on minions from the master.

The client ACL system is configured in the master configuration file via the `client_acl` configuration option. Under the `client_acl` configuration option the users open to send commands are specified and then a list of regular expressions which specify the minion functions which will be made available to specified user. This configuration is much like the `peer` configuration:

```
# Allow thatch to execute anything and allow fred to use ping and pkg
client_acl:
  thatch:
    - .*
  fred:
    - test.*
    - pkg.*
```

96.1 Permission Issues

Directories required for `client_acl` must be modified to be readable by the users specified:

```
chmod 755 /var/cache/salt /var/cache/salt/jobs /var/run/salt
```

Note: In addition to the changes above you will also need to modify the permissions of /var/log/salt and the existing log file. If you do not wish to do this then you must disable logging or Salt will generate errors as it cannot write to the logs as the system users.

If you are upgrading from earlier versions of salt you must also remove any existing user keys and re-start the Salt master:

```
rm /var/cache/salt/.*key
service salt-master restart
```

CHAPTER 97

Salt Syndic

The Salt Syndic interface is a powerful tool which allows for the construction of Salt command topologies. A basic Salt setup has a Salt Master commanding a group of Salt Minions. The Syndic interface is a special passthrough minion, it is run on a master and connects to another master, then the master that the Syndic minion is listening to can control the minions attached to the master running the syndic.

The intent for supporting many layouts is not presented with the intent of supposing the use of any single topology, but to allow a more flexible method of controlling many systems.

97.1 Configuring the Syndic

Since the Syndic only needs to be attached to a higher level master the configuration is very simple. On a master that is running a syndic to connect to a higher level master the `syndic_master` option needs to be set in the master config file. The `syndic_master` option contains the hostname or IP address of the master server that can control the master that the syndic is running on.

The master that the syndic connects to sees the syndic as an ordinary minion, and treats it as such. the higher level master will need to accept the syndic's minion key like any other minion. This master will also need to set the `order_masters` value in the configuration to `True`. The `order_masters` option in the config on the higher level master is very important, to control a syndic extra information needs to be sent with the publications, the `order_masters` option makes sure that the extra data is sent out.

To sum up, you have those configuration options available on the master side:

- `syndic_master`: MasterOfMaster ip/address
- `syndic_master_port`: MasterOfMaster ret_port
- `syndic_log_file`: path to the logfile (absolute or not)
- `syndic_pidfile`: path to the pidfile (absolute or not)

Each Syndic must provide its own `file_roots` directory. Files will not be automatically transferred from the master-master.

97.2 Running the Syndic

The Syndic is a separate daemon that needs to be started on the master that is controlled by a higher master. Starting the Syndic daemon is the same as starting the other Salt daemons.

```
# salt-syndic
```

Note: If you have an exceptionally large infrastructure or many layers of syndics, you may find that the CLI doesn't wait long enough for the syndics to return their events. If you think this is the case, you can set the `syndic_wait` value in the upper master config. The default value is `1`, and should work for the majority of deployments.

CHAPTER 98

File Server Backends

Salt version 0.12.0 introduced the ability for the Salt Master to integrate different file server backends. File server backends allows the Salt file server to act as a transparent bridge to external resources. The primary example of this is the git backend which allows for all of the Salt formulas and files to be maintained in a remote git repository.

The fileserver backend system can accept multiple backends as well. This makes it possible to have the environments listed in the `file_roots` configuration available in addition to other backends, or the ability to mix multiple backends.

This feature is managed by the `fileserver_backend` option in the master config. The desired backend systems are listed in order of search priority:

```
fileserver_backend:
  - roots
  - git
```

With this configuration, the environments and files defined in the `file_roots` parameter will be searched first, if the referenced environment and file is not found then the `git` backend will be searched.

98.1 Environments

The concept of environments is followed in all backend systems. The environments in the classic `roots` backend are defined in the `file_roots` option. Environments map differently based on the backend, for instance the git backend translated branches and tags in git to environments. This makes it easy to define environments in git by just setting a tag or forking a branch.

CHAPTER 99

Dynamic Module Distribution

New in version 0.9.5.

Salt Python modules can be distributed automatically via the Salt file server. Under the root of any environment defined via the `file_roots` option on the master server directories corresponding to the type of module can be used.

Module sync Automatically transfer and load modules, grains, renderers, returners, states, etc from the master to the minions.

The directories are prepended with an underscore:

1. `_modules`
2. `_grains`
3. `_renderers`
4. `_returners`
5. `_states`

The contents of these directories need to be synced over to the minions after Python modules have been created in them. There are a number of ways to sync the modules.

99.1 Sync Via States

The minion configuration contains an option `autoload_dynamic_modules` which defaults to True. This option makes the state system refresh all dynamic modules when states are run. To disable this behavior set `autoload_dynamic_modules` to False in the minion config.

When dynamic modules are autoloaded via states, modules only pertinent to the environments matched in the master's top file are downloaded.

This is important to remember, because modules can be manually loaded from any specific environment that environment specific modules will be loaded when a state run is executed.

99.2 Sync Via the saltutil Module

The saltutil module has a number of functions that can be used to sync all or specific dynamic modules. The saltutil module function `saltutil.sync_all` will sync all module types over to a minion. For more information see: `salt.modules.saltutil`

CHAPTER **100**

File Server Configuration

The Salt file server is a high performance file server written in ZeroMQ. It manages large files quickly and with little overhead, and has been optimized to handle small files in an extremely efficient manner.

The Salt file server is an environment aware file server. This means that files can be allocated within many root directories and accessed by specifying both the file path and the environment to search. The individual environments can span across multiple directory roots to create overlays and to allow for files to be organized in many flexible ways.

100.1 Environments

The Salt file server defaults to the mandatory `base` environment. This environment **MUST** be defined and is used to download files when no environment is specified.

Environments allow for files and sls data to be logically separated, but environments are not isolated from each other. This allows for logical isolation of environments by the engineer using Salt, but also allows for information to be used in multiple environments.

100.2 Directory Overlay

The `environment` setting is a list of directories to publish files from. These directories are searched in order to find the specified file and the first file found is returned.

This means that directory data is prioritized based on the order in which they are listed. In the case of this `file_roots` configuration:

```
file_roots:
  base:
    - /srv/salt/base
    - /srv/salt/failover
```

If a file's URI is `salt://httpd/httpd.conf`, it will first search for the file at `/srv/salt/base/httpd/httpd.conf`. If the file is found there it will be returned. If the file is not found there, then `/srv/salt/failover/httpd/httpd.conf` will be used for the source.

This allows for directories to be overlaid and prioritized based on the order they are defined in the configuration.

It is also possible to have `file_roots` which supports multiple environments:

```
file_roots:
  base:
    - /srv/salt/base
```

```
dev:
  - /srv/salt/dev
  - /srv/salt/base
prod:
  - /srv/salt/prod
  - /srv/salt/base
```

This example ensures that each environment will check the associated environment directory for files first. If a file is not found in the appropriate directory, the system will default to using the base directory.

100.3 Local File Server

New in version 0.9.8.

The file server can be rerouted to run from the minion. This is primarily to enable running Salt states without a Salt master. To use the local file server interface, copy the file server data to the minion and set the file_roots option on the minion to point to the directories copied from the master. Once the minion `file_roots` option has been set, change the `file_client` option to local to make sure that the local file server interface is used.

CHAPTER 101

Salt File Server

Salt comes with a simple file server suitable for distributing files to the Salt minions. The file server is a stateless ZeroMQ server that is built into the Salt master.

The main intent of the Salt file server is to present files for use in the Salt state system. With this said, the Salt file server can be used for any general file transfer from the master to the minions.

101.1 The cp Module

The cp module is the home of minion side file server operations. The cp module is used by the Salt state system, salt-cp and can be used to distribute files presented by the Salt file server.

101.1.1 Environments

Since the file server is made to work with the Salt state system, it supports environments. The environments are defined in the master config file and when referencing an environment the file specified will be based on the root directory of the environment.

101.1.2 get_file

The cp.get_file function can be used on the minion to download a file from the master, the syntax looks like this:

```
# salt '*' cp.get_file salt://vimrc /etc/vimrc
```

This will instruct all Salt minions to download the vimrc file and copy it to /etc/vimrc

Template rendering can be enabled on both the source and destination file names like so:

```
# salt '*' cp.get_file "salt://{{grains.os}}/vimrc" /etc/vimrc template=jinja
```

This example would instruct all Salt minions to download the vimrc from a directory with the same name as their OS grain and copy it to /etc/vimrc

For larger files, the cp.get_file module also supports gzip compression. Because gzip is CPU-intensive, this should only be used in scenarios where the compression ratio is very high (e.g. pretty-printed JSON or YAML files).

Use the *gzip* named argument to enable it. Valid values are 1..9, where 1 is the lightest compression and 9 the heaviest. 1 uses the least CPU on the master (and minion), 9 uses the most.

```
# salt '*' cp.get_file salt://vimrc /etc/vimrc gzip=5
```

Finally, note that by default cp.get_file does *not* create new destination directories if they do not exist. To change this, use the *makedirs* argument:

```
# salt '*' cp.get_file salt://vimrc /etc/vim/vimrc makedirs=True
```

In this example, /etc/vim/ would be created if it didn't already exist.

101.1.3 get_dir

The cp.get_dir function can be used on the minion to download an entire directory from the master. The syntax is very similar to get_file:

```
# salt '*' cp.get_dir salt://etc/apache2 /etc
```

cp.get_dir supports *template* rendering and *gzip* compression arguments just like get_file:

```
# salt '*' cp.get_dir salt://etc/{{pillar.webserver}} /etc gzip=5 template=jinja
```

101.2 File Server Client API

A client API is available which allows for modules and applications to be written which make use of the Salt file server.

The file server uses the same authentication and encryption used by the rest of the Salt system for network communication.

101.2.1 FileClient Class

The FileClient class is used to set up the communication from the minion to the master. When creating a FileClient object the minion configuration needs to be passed in. When using the FileClient from within a minion module the built in `__opts__` data can be passed:

```
import salt.minion

def get_file(path, dest, env='base'):
    '''
    Used to get a single file from the Salt master

    CLI Example:
    salt '*' cp.get_file salt://vimrc /etc/vimrc
    '''
    # Create the FileClient object
    client = salt.minion.FileClient(__opts__)
    # Call get_file
    return client.get_file(path, dest, False, env)
```

Using the FileClient class outside of a minion module where the `__opts__` data is not available, it needs to be generated:

```
import salt.minion
import salt.config
```

```
def get_file(path, dest, env='base'):
    '''
    Used to get a single file from the Salt master
    '''
    # Get the configuration data
    opts = salt.config.minion_config('/etc/salt/minion')
    # Create the FileClient object
    client = salt.minion.FileClient(opts)
    # Call get_file
    return client.get_file(path, dest, False, env)
```

CHAPTER 102

Full list of builtin fileserver modules

`gitfs`	The backend for the git based file server system.
`hgfs`	The backed for the mercurial based file server system.
`roots`	The default file server backend
`s3fs`	The backend for a fileserver based on Amazon S3
`minionfs`	The backend for serving files pushed to master by cp.push (file_recv).

102.1 salt.fileserver.gitfs

The backend for the git based file server system.

After enabling this backend, branches and tags in a remote git repository are exposed to salt as different environments. This feature is managed by the fileserver_backend option in the salt master config.

depends

- gitpython Python module

`salt.fileserver.gitfs.`**`dir_list`**`(`*load*`)`
Return a list of all directories on the master

`salt.fileserver.gitfs.`**`envs`**`(`*ignore_cache=False*`)`
Return a list of refs that can be used as environments

`salt.fileserver.gitfs.`**`file_hash`**`(`*load, fnd*`)`
Return a file hash, the hash type is set in the master config file

`salt.fileserver.gitfs.`**`file_list`**`(`*load*`)`
Return a list of all files on the file server in a specified environment

`salt.fileserver.gitfs.`**`file_list_emptydirs`**`(`*load*`)`
Return a list of all empty directories on the master

`salt.fileserver.gitfs.`**`find_file`**`(`*path, short='base', **kwargs*`)`
Find the first file to match the path and ref, read the file out of git and send the path to the newly cached file

`salt.fileserver.gitfs.`**`init`**`()`
Return the git repo object for this session

`salt.fileserver.gitfs.`**`purge_cache`**`()`
Purge the fileserver cache

`salt.fileserver.gitfs.`**`serve_file`**`(`*load, fnd*`)`
Return a chunk from a file based on the data received

`salt.fileserver.gitfs.`**`update`**`()`
Execute a git pull on all of the repos

102.2 salt.fileserver.hgfs

The backed for the mercurial based file server system.

After enabling this backend, branches, bookmarks, and tags in a remote mercurial repository are exposed to salt as different environments. This feature is managed by the `fileserver_backend` option in the salt master config file.

This fileserver has an additional option `hgfs_branch_method` that will set the desired branch method. Possible values are: `branches`, `bookmarks`, or `mixed`. If using `branches` or `mixed`, the `default` branch will be mapped to `base`.

Changed in version 2014.1.0: (Hydrogen) The `hgfs_base` master config parameter was added, allowing for a branch other than `default` to be used for the `base` environment, and allowing for a `base` environment to be specified when using an `hgfs_branch_method` of `bookmarks`.

depends

- mercurial

`salt.fileserver.hgfs.`**`dir_list`**`(`*load*`)`
Return a list of all directories on the master

`salt.fileserver.hgfs.`**`envs`**`(`*ignore_cache=False*`)`
Return a list of refs that can be used as environments

`salt.fileserver.hgfs.`**`file_hash`**`(`*load*, *fnd*`)`
Return a file hash, the hash type is set in the master config file

`salt.fileserver.hgfs.`**`file_list`**`(`*load*`)`
Return a list of all files on the file server in a specified environment

`salt.fileserver.hgfs.`**`file_list_emptydirs`**`(`*load*`)`
Return a list of all empty directories on the master

`salt.fileserver.hgfs.`**`find_file`**`(`*path*, *short='base'*, ***kwargs*`)`
Find the first file to match the path and ref, read the file out of hg and send the path to the newly cached file

`salt.fileserver.hgfs.`**`init`**`()`
Return the hg repo object for this session

`salt.fileserver.hgfs.`**`purge_cache`**`()`

`salt.fileserver.hgfs.`**`serve_file`**`(`*load*, *fnd*`)`
Return a chunk from a file based on the data received

`salt.fileserver.hgfs.`**`update`**`()`
Execute a hg pull on all of the repos

102.3 salt.fileserver.roots

The default file server backend

Based on the environments in the `file_roots` configuration option.

`salt.fileserver.roots.`**`dir_list`**`(`*load*`)`
Return a list of all directories on the master

`salt.fileserver.roots.`**`envs`**`()`
Return the file server environments

`salt.fileserver.roots.`**`file_hash`**`(`*load*, *fnd*`)`
Return a file hash, the hash type is set in the master config file

`salt.fileserver.roots.`**`file_list`**`(`*load*`)`
Return a list of all files on the file server in a specified environment

`salt.fileserver.roots.`**`file_list_emptydirs`**`(`*load*`)`
Return a list of all empty directories on the master

`salt.fileserver.roots.`**`find_file`**`(`*path*, *saltenv='base'*, *env=None*, ***kwargs*`)`
Search the environment for the relative path

`salt.fileserver.roots.`**`serve_file`**`(`*load*, *fnd*`)`
Return a chunk from a file based on the data received

`salt.fileserver.roots.`**`symlink_list`**`(`*load*`)`
Return a dict of all symlinks based on a given path on the Master

`salt.fileserver.roots.`**`update`**`()`
When we are asked to update (regular interval) lets reap the cache

102.4 salt.fileserver.s3fs

The backend for a fileserver based on Amazon S3

See also:

Salt File Server

This backend exposes directories in S3 buckets as Salt environments. This feature is managed by the `fileserver_backend` option in the Salt Master config.

S3 credentials can be set in the master config file like so:

```
s3.keyid: GKTADJGHEIQSXMKKRBJ08H
s3.key: askdjghsdfjkghWupUjasdflkdfklgjsdfjajkghs
```

Alternatively, if on EC2 these credentials can be automatically loaded from instance metadata.

Additionally, `s3fs` must be included in the `fileserver_backend` config parameter in the master config file:

```
fileserver_backend:
  - s3fs
```

This fileserver supports two modes of operation for the buckets:

1. **A single bucket per environment**

   ```
   s3.buckets:
     production:
       - bucket1
       - bucket2
     staging:
       - bucket3
       - bucket4
   ```

2. **Multiple environments per bucket**

```
s3.buckets:
  - bucket1
  - bucket2
  - bucket3
  - bucket4
```

Note that bucket names must be all lowercase both in the AWS console and in Salt, otherwise you may encounter `SignatureDoesNotMatch` errors.

A multiple-environment bucket must adhere to the following root directory structure:

```
s3://<bucket name>/<environment>/<files>
```

`salt.fileserver.s3fs.`**`dir_list`**`(`*load*`)`
Return a list of all directories on the master

`salt.fileserver.s3fs.`**`envs`**`()`
Return a list of directories within the bucket that can be used as environments.

`salt.fileserver.s3fs.`**`file_hash`**`(`*load, fnd*`)`
Return an MD5 file hash

`salt.fileserver.s3fs.`**`file_list`**`(`*load*`)`
Return a list of all files on the file server in a specified environment

`salt.fileserver.s3fs.`**`file_list_emptydirs`**`(`*load*`)`
Return a list of all empty directories on the master

`salt.fileserver.s3fs.`**`find_file`**`(`*path, saltenv='base', env=None, **kwargs*`)`
Look through the buckets cache file for a match. If the field is found, it is retrieved from S3 only if its cached version is missing, or if the MD5 does not match.

`salt.fileserver.s3fs.`**`serve_file`**`(`*load, fnd*`)`
Return a chunk from a file based on the data received

`salt.fileserver.s3fs.`**`update`**`()`
Update the cache file for the bucket.

102.5 salt.fileserver.minionfs

The backend for serving files pushed to master by cp.push (file_recv).

`file_recv` needs to be enabled in the master config file.

`salt.fileserver.minionfs.`**`dir_list`**`(`*load*`)`
Return a list of all directories on the master

CLI Example:

```
$ salt 'source-minion' cp.push /absolute/path/file  # Push the file to the master
$ salt 'destination-minion' cp.list_master_dirs
destination-minion:
    - .
    - source-minion/absolute
    - source-minion/absolute/path
```

`salt.fileserver.minionfs.`**`envs`**`()`
Return "base" as the file server environment, because there is only one set of minions.

salt.fileserver.minionfs.**file_hash**(*load*, *fnd*)

Return a file hash, the hash type is set in the master config file

salt.fileserver.minionfs.**file_list**(*load*)

Return a list of all files on the file server in a specified environment

salt.fileserver.minionfs.**find_file**(*path*, *env='base'*, ***kwargs*)

Search the environment for the relative path

salt.fileserver.minionfs.**serve_file**(*load*, *fnd*)

Return a chunk from a file based on the data received

CLI Example:

```
$ salt 'source-minion' cp.push /path/to/the/file  # Push the file to the master
$ salt 'destination-minion'  cp.get_file salt://source-minion/path/to/the/file /destination/file
```

salt.fileserver.minionfs.**update**()

When we are asked to update (regular interval) lets reap the cache

CHAPTER 103

Configuration file examples

- Example master configuration file
- Example minion configuration file

103.1 Example master configuration file

```
##### Primary configuration settings #####
##########################################
# This configuration file is used to manage the behavior of the Salt Master
# Values that are commented out but have no space after the comment are
# defaults that need not be set in the config. If there is a space after the
# comment that the value is presented as an example and is not the default.

# Per default, the master will automatically include all config files
# from master.d/*.conf (master.d is a directory in the same directory
# as the main master config file)
#default_include: master.d/*.conf

# The address of the interface to bind to
#interface: 0.0.0.0

# Whether the master should listen for IPv6 connections. If this is set to True,
# the interface option must be adjusted too (for example: "interface: '::'")
#ipv6: False

# The tcp port used by the publisher
#publish_port: 4505

# The user under which the salt master will run. Salt will update all
# permissions to allow the specified user to run the master. The exception is
# the job cache, which must be deleted if this user is changed.  If the
# modified files cause conflicts set verify_env to False.
#user: root

# Max open files
# Each minion connecting to the master uses AT LEAST one file descriptor, the
# master subscription connection. If enough minions connect you might start
# seeing on the console(and then salt-master crashes):
#   Too many open files (tcp_listener.cpp:335)
#   Aborted (core dumped)
#
# By default this value will be the one of `ulimit -Hn`, ie, the hard limit for
# max open files.
#
# If you wish to set a different value than the default one, uncomment and
# configure this setting. Remember that this value CANNOT be higher than the
# hard limit. Raising the hard limit depends on your OS and/or distribution,
# a good way to find the limit is to search the internet for(for example):
#   raise max open files hard limit debian
#
#max_open_files: 100000

# The number of worker threads to start, these threads are used to manage
# return calls made from minions to the master, if the master seems to be
# running slowly, increase the number of threads
#worker_threads: 5

# The port used by the communication interface. The ret (return) port is the
# interface used for the file server, authentication, job returnes, etc.
#ret_port: 4506

# Specify the location of the daemon process ID file
#pidfile: /var/run/salt-master.pid

# The root directory prepended to these options: pki_dir, cachedir,
# sock_dir, log_file, autosign_file, autoreject_file, extension_modules,
# key_logfile, pidfile.
#root_dir: /

# Directory used to store public key data
#pki_dir: /etc/salt/pki/master

# Directory to store job and cache data
#cachedir: /var/cache/salt/master

# Verify and set permissions on configuration directories at startup
#verify_env: True

# Set the number of hours to keep old job information in the job cache
#keep_jobs: 24

# Set the default timeout for the salt command and api, the default is 5
# seconds
#timeout: 5

# The loop_interval option controls the seconds for the master's maintinance
# process check cycle. This process updates file server backends, cleans the
# job cache and executes the scheduler.
#loop_interval: 60

# Set the default outputter used by the salt command. The default is "nested"
#output: nested

# By default output is colored, to disable colored output set the color value
# to False
#color: True
```

```
# Set the directory used to hold unix sockets
#sock_dir: /var/run/salt/master

# The master can take a while to start up when lspci and/or dmidecode is used
# to populate the grains for the master. Enable if you want to see GPU hardware
# data for your master.
#
# enable_gpu_grains: False

# The master maintains a job cache, while this is a great addition it can be
# a burden on the master for larger deployments (over 5000 minions).
# Disabling the job cache will make previously executed jobs unavailable to
# the jobs system and is not generally recommended.
#
#job_cache: True

# Cache minion grains and pillar data in the cachedir.
#minion_data_cache: True

# The master can include configuration from other files. To enable this,
# pass a list of paths to this option. The paths can be either relative or
# absolute; if relative, they are considered to be relative to the directory
# the main master configuration file lives in (this file). Paths can make use
# of shell-style globbing. If no files are matched by a path passed to this
# option then the master will log a warning message.
#
#
# Include a config file from some other path:
#include: /etc/salt/extra_config
#
# Include config from several files and directories:
#include:
#  - /etc/salt/extra_config


#####        Security settings       #####
##########################################
# Enable "open mode", this mode still maintains encryption, but turns off
# authentication, this is only intended for highly secure environments or for
# the situation where your keys end up in a bad state. If you run in open mode
# you do so at your own risk!
#open_mode: False

# Enable auto_accept, this setting will automatically accept all incoming
# public keys from the minions. Note that this is insecure.
#auto_accept: False

# If the autosign_file is specified, incoming keys specified in the
# autosign_file will be automatically accepted. This is insecure.  Regular
# expressions as well as globing lines are supported.
#autosign_file: /etc/salt/autosign.conf

# Works like autosign_file, but instead allows you to specify minion IDs for
# which keys will automatically be rejected. Will override both membership in
# the autosign_file and the auto_accept setting.
#autosign_file: /etc/salt/autosign.conf

# Enable permissive access to the salt keys.  This allows you to run the
# master or minion as root, but have a non-root group be given access to
# your pki_dir.  To make the access explicit, root must belong to the group
# you've given access to.  This is potentially quite insecure.
# If an autosign_file is specified, enabling permissive_pki_access will allow group access
# to that specific file.
#permissive_pki_access: False

# Allow users on the master access to execute specific commands on minions.
# This setting should be treated with care since it opens up execution
# capabilities to non root users. By default this capability is completely
# disabled.
#
#client_acl:
#  larry:
#    - test.ping
#    - network.*
#

# Blacklist any of the following users or modules
#
# This example would blacklist all non sudo users, including root from
# running any commands. It would also blacklist any use of the "cmd"
# module.
# This is completely disabled by default.
#
#client_acl_blacklist:
#  users:
#    - root
#    - '^(?!sudo_).*$'   #  all non sudo users
#  modules:
#    - cmd

# The external auth system uses the Salt auth modules to authenticate and
# validate users to access areas of the Salt system.
#
#external_auth:
#  pam:
#    fred:
#      - test.*
#

# Time (in seconds) for a newly generated token to live. Default: 12 hours
#token_expire: 43200

# Allow minions to push files to the master. This is disabled by default, for
# security purposes.
#file_recv: False

# Set a hard-limit on the size of the files that can be pushed to the master.
# It will be interpreted as megabytes.
# Default: 100
#file_recv_max_size: 100

# Signature verification on messages published from the master.
# This causes the master to cryptographically sign all messages published to its event
# bus, and minions then verify that signature before acting on the message.
#
# This is False by default.
```

```
#
# Note that to facilitate interoperability with masters and minions that are different
# versions, if sign_pub_messages is True but a message is received by a minion with
# no signature, it will still be accepted, and a warning message will be logged.
# Conversely, if sign_pub_messages is False, but a minion receives a signed
# message it will be accepted, the signature will not be checked, and a warning message
# will be logged.  This behavior will go away in Salt 0.17.6 (or Hydrogen RC1, whichever
# comes first) and these two situations will cause minion to throw an exception and
# drop the message.
#
# sign_pub_messages: False

#####    Master Module Management    #####
##########################################
# Manage how master side modules are loaded

# Add any additional locations to look for master runners
#runner_dirs: []

# Enable Cython for master side modules
#cython_enable: False


#####      State System settings     #####
##########################################
# The state system uses a "top" file to tell the minions what environment to
# use and what modules to use. The state_top file is defined relative to the
# root of the base environment as defined in "File Server settings" below.
#state_top: top.sls

# The master_tops option replaces the external_nodes option by creating
# a plugable system for the generation of external top data. The external_nodes
# option is deprecated by the master_tops option.
# To gain the capabilities of the classic external_nodes system, use the
# following configuration:
# master_tops:
#   ext_nodes: <Shell command which returns yaml>
#
#master_tops: {}

# The external_nodes option allows Salt to gather data that would normally be
# placed in a top file. The external_nodes option is the executable that will
# return the ENC data. Remember that Salt will look for external nodes AND top
# files and combine the results if both are enabled!
#external_nodes: None

# The renderer to use on the minions to render the state data
#renderer: yaml_jinja

# The Jinja renderer can strip extra carriage returns and whitespace
# See http://jinja.pocoo.org/docs/api/#high-level-api
#
# If this is set to True the first newline after a Jinja block is removed
# (block, not variable tag!). Defaults to False, corresponds to the Jinja
# environment init variable "trim_blocks".
# jinja_trim_blocks: False
#
# If this is set to True leading spaces and tabs are stripped from the start
# of a line to a block. Defaults to False, corresponds to the Jinja
# environment init variable "lstrip_blocks".
# jinja_lstrip_blocks: False

# The failhard option tells the minions to stop immediately after the first
# failure detected in the state execution, defaults to False
#failhard: False

# The state_verbose and state_output settings can be used to change the way
# state system data is printed to the display. By default all data is printed.
# The state_verbose setting can be set to True or False, when set to False
# all data that has a result of True and no changes will be suppressed.
#state_verbose: True

# The state_output setting changes if the output is the full multi line
# output for each changed state if set to 'full', but if set to 'terse'
# the output will be shortened to a single line.  If set to 'mixed', the output
# will be terse unless a state failed, in which case that output will be full.
#state_output: full


#####      File Server settings      #####
##########################################
# Salt runs a lightweight file server written in zeromq to deliver files to
# minions. This file server is built into the master daemon and does not
# require a dedicated port.

# The file server works on environments passed to the master, each environment
# can have multiple root directories, the subdirectories in the multiple file
# roots cannot match, otherwise the downloaded files will not be able to be
# reliably ensured. A base environment is required to house the top file.
# Example:
# file_roots:
#   base:
#     - /srv/salt/
#   dev:
#     - /srv/salt/dev/services
#     - /srv/salt/dev/states
#   prod:
#     - /srv/salt/prod/services
#     - /srv/salt/prod/states

#file_roots:
#  base:
#    - /srv/salt

# The hash_type is the hash to use when discovering the hash of a file on
# the master server. The default is md5, but sha1, sha224, sha256, sha384
# and sha512 are also supported.
#hash_type: md5

# The buffer size in the file server can be adjusted here:
#file_buffer_size: 1048576

# A regular expression (or a list of expressions) that will be matched
# against the file path before syncing the modules and states to the minions.
# This includes files affected by the file.recurse state.
# For example, if you manage your custom modules and states in subversion
# and don't want all the '.svn' folders and content synced to your minions,
# you could set this to '/\.svn($|/)'. By default nothing is ignored.
#
#file_ignore_regex:
#  - '/\.svn($|/)'
#  - '/\.git($|/)'

# A file glob (or list of file globs) that will be matched against the file
# path before syncing the modules and states to the minions. This is similar
# to file_ignore_regex above, but works on globs instead of regex. By default
# nothing is ignored.
#
# file_ignore_glob:
#  - '*.pyc'
#  - '*/somefolder/*.bak'
#  - '*.swp'

# File Server Backend
# Salt supports a modular fileserver backend system, this system allows
# the salt master to link directly to third party systems to gather and
# manage the files available to minions. Multiple backends can be
# configured and will be searched for the requested file in the order in which
# they are defined here. The default setting only enables the standard backend
# "roots" which uses the "file_roots" option.
#
#fileserver_backend:
#  - roots
#
# To use multiple backends list them in the order they are searched:
#
#fileserver_backend:
#  - git
#  - roots
#
# Uncomment the line below if you do not want the file_server to follow
# symlinks when walking the filesystem tree. This is set to True
# by default. Currently this only applies to the default roots
# fileserver_backend.
#
#fileserver_followsymlinks: False
#
# Uncomment the line below if you do not want symlinks to be
# treated as the files they are pointing to. By default this is set to
# False. By uncommenting the line below, any detected symlink while listing
# files on the Master will not be returned to the Minion.
#
#fileserver_ignoresymlinks: True
#
# By default, the Salt fileserver recurses fully into all defined environments
# to attempt to find files. To limit this behavior so that the fileserver only
# traverses directories with SLS files and special Salt directories like _modules,
# enable the option below. This might be useful for installations where a file root
# has a very large number of files and performance is impacted. Default is False.
#
# fileserver_limit_traversal: False
#
# The fileserver can fire events off every time the fileserver is updated,
# these are disabled by default, but can be easily turned on by setting this
# flag to True
#fileserver_events: False
#
# Git fileserver backend configuration
# When using the git fileserver backend at least one git remote needs to be
# defined. The user running the salt master will need read access to the repo.
#
#gitfs_remotes:
#  - git://github.com/saltstack/salt-states.git
#  - file:///var/git/saltmaster
#
# The gitfs_ssl_verify option specifies whether to ignore ssl certificate
# errors when contacting the gitfs backend. You might want to set this to
# false if you're using a git backend that uses a self-signed certificate but
# keep in mind that setting this flag to anything other than the default of True
# is a security concern, you may want to try using the ssh transport.
#gitfs_ssl_verify: True
#
# The repos will be searched in order to find the file requested by a client
# and the first repo to have the file will return it.
# When using the git backend branches and tags are translated into salt
# environments.
# Note:  file:// repos will be treated as a remote, so refs you want used must
# exist in that repo as *local* refs.
#
# The gitfs_root option gives the ability to serve files from a subdirectory
# within the repository. The path is defined relative to the root of the
# repository and defaults to the repository root.
#gitfs_root: somefolder/otherfolder


#####         Pillar settings        #####
##########################################
# Salt Pillars allow for the building of global data that can be made selectively
# available to different minions based on minion grain filtering. The Salt
# Pillar is laid out in the same fashion as the file server, with environments,
# a top file and sls files. However, pillar data does not need to be in the
# highstate format, and is generally just key/value pairs.

#pillar_roots:
#  base:
#    - /srv/pillar

#ext_pillar:
#  - hiera: /etc/hiera.yaml
#  - cmd_yaml: cat /etc/salt/yaml

# The pillar_gitfs_ssl_verify option specifies whether to ignore ssl certificate
# errors when contacting the pillar gitfs backend. You might want to set this to
# false if you're using a git backend that uses a self-signed certificate but
# keep in mind that setting this flag to anything other than the default of True
# is a security concern, you may want to try using the ssh transport.
#pillar_gitfs_ssl_verify: True

# The pillar_opts option adds the master configuration file data to a dict in
# the pillar called "master". This is used to set simple configurations in the
# master config file that can then be used on minions.
#pillar_opts: True
```

```
#####          Syndic settings       #####
##########################################
# The Salt syndic is used to pass commands through a master from a higher
# master. Using the syndic is simple, if this is a master that will have
# syndic servers(s) below it set the "order_masters" setting to True, if this
# is a master that will be running a syndic daemon for passthrough the
# "syndic_master" setting needs to be set to the location of the master server
# to receive commands from.

# Set the order_masters setting to True if this master will command lower
# masters' syndic interfaces.
#order_masters: False

# If this master will be running a salt syndic daemon, syndic_master tells
# this master where to receive commands from.
#syndic_master: masterofmaster

# This is the 'ret_port' of the MasterOfMaster
#syndic_master_port: 4506

# PID file of the syndic daemon
#syndic_pidfile: /var/run/salt-syndic.pid

# LOG file of the syndic daemon
#syndic_log_file: syndic.log

#####      Peer Publish settings     #####
##########################################
# Salt minions can send commands to other minions, but only if the minion is
# allowed to. By default "Peer Publication" is disabled, and when enabled it
# is enabled for specific minions and specific commands. This allows secure
# compartmentalization of commands based on individual minions.

# The configuration uses regular expressions to match minions and then a list
# of regular expressions to match functions. The following will allow the
# minion authenticated as foo.example.com to execute functions from the test
# and pkg modules.
#
#peer:
#  foo.example.com:
#    - test.*
#    - pkg.*
#
# This will allow all minions to execute all commands:
#
#peer:
#  .*:
#    - .*
#
# This is not recommended, since it would allow anyone who gets root on any
# single minion to instantly have root on all of the minions!

# Minions can also be allowed to execute runners from the salt master.
# Since executing a runner from the minion could be considered a security risk,
# it needs to be enabled. This setting functions just like the peer setting
# except that it opens up runners instead of module functions.
#
```

```
# All peer runner support is turned off by default and must be enabled before
# using. This will enable all peer runners for all minions:
#
#peer_run:
#  .*:
#    - .*
#
# To enable just the manage.up runner for the minion foo.example.com:
#
#peer_run:
#  foo.example.com:
#    - manage.up

#####         Mine settings     #####
##########################################
# Restrict mine.get access from minions. By default any minion has a full access
# to get all mine data from master cache. In acl definion below, only pcre matches
# are allowed.
#
# mine_get:
#   .*:
#     - .*
#
# Example below enables minion foo.example.com to get  'network.interfaces' mine data only
# , minions web* to get all network.* and disk.* mine data and all other minions won't get
# any mine data.
#
# mine_get:
#   foo.example.com:
#     - network.inetrfaces
#   web.*:
#     - network.*
#     - disk.*

#####         Logging settings       #####
##########################################
# The location of the master log file
# The master log can be sent to a regular file, local path name, or network
# location. Remote logging works best when configured to use rsyslogd(8) (e.g.:
# ``file:///dev/log``), with rsyslogd(8) configured for network logging. The URI
# format is: <file|udp|tcp>://<host|socketpath>:<port-if-required>/<log-facility>
#log_file: /var/log/salt/master
#log_file: file:///dev/log
#log_file: udp://loghost:10514

#log_file: /var/log/salt/master
#key_logfile: /var/log/salt/key

# The level of messages to send to the console.
# One of 'garbage', 'trace', 'debug', info', 'warning', 'error', 'critical'.
#log_level: warning

# The level of messages to send to the log file.
# One of 'garbage', 'trace', 'debug', info', 'warning', 'error', 'critical'.
#log_level_logfile: warning

# The date and time format used in log messages. Allowed date/time formating
# can be seen here: http://docs.python.org/library/time.html#time.strftime
```

```
#log_datefmt: ’%H:%M:%S’
#log_datefmt_logfile: ’%Y-%m-%d %H:%M:%S’

# The format of the console logging messages. Allowed formatting options can
# be seen here: http://docs.python.org/library/logging.html#logrecord-attributes
#log_fmt_console: ’[%(levelname)-8s] %(message)s’
#log_fmt_logfile: ’%(asctime)s,%(msecs)03.0f [%(name)-17s][%(levelname)-8s] %(message)s’

# This can be used to control logging levels more specificically.  This
# example sets the main salt library at the ’warning’ level, but sets
# ’salt.modules’ to log at the ’debug’ level:
#   log_granular_levels:
#     ’salt’: ’warning’,
#     ’salt.modules’: ’debug’
#
#log_granular_levels: {}


#####         Node Groups           #####
##########################################
# Node groups allow for logical groupings of minion nodes.
# A group consists of a group name and a compound target.
#
#nodegroups:
#  group1: ’L@foo.domain.com,bar.domain.com,baz.domain.com and bl*.domain.com’
#  group2: ’G@os:Debian and foo.domain.com’


#####     Range Cluster settings     #####
##########################################
# The range server (and optional port) that serves your cluster information
# https://github.com/grierj/range/wiki/Introduction-to-Range-with-YAML-files
#
#range_server: range:80


#####     Windows Software Repo settings #####
##############################################
# Location of the repo on the master
#win_repo: ’/srv/salt/win/repo’

# Location of the master’s repo cache file
#win_repo_mastercachefile: ’/srv/salt/win/repo/winrepo.p’

# List of git repositories to include with the local repo
#win_gitrepos:
#  - ’https://github.com/saltstack/salt-winrepo.git’
```

103.2 Example minion configuration file

```
##### Primary configuration settings #####
##########################################

# Per default the minion will automatically include all config files
# from minion.d/*.conf (minion.d is a directory in the same directory
# as the main minion config file).
```

```
#default_include: minion.d/*.conf

# Set the location of the salt master server, if the master server cannot be
# resolved, then the minion will fail to start.
#master: salt

# Set whether the minion should connect to the master via IPv6
#ipv6: False

# Set the number of seconds to wait before attempting to resolve
# the master hostname if name resolution fails. Defaults to 30 seconds.
# Set to zero if the minion should shutdown and not retry.
# retry_dns: 30

# Set the port used by the master reply and authentication server
#master_port: 4506

# The user to run salt
#user: root

# Specify the location of the daemon process ID file
#pidfile: /var/run/salt-minion.pid

# The root directory prepended to these options: pki_dir, cachedir, log_file,
# sock_dir, pidfile.
#root_dir: /

# The directory to store the pki information in
#pki_dir: /etc/salt/pki/minion

# Explicitly declare the id for this minion to use, if left commented the id
# will be the hostname as returned by the python call: socket.getfqdn()
# Since salt uses detached ids it is possible to run multiple minions on the
# same machine but with different ids, this can be useful for salt compute
# clusters.
#id:

# Append a domain to a hostname in the event that it does not exist.  This is
# useful for systems where socket.getfqdn() does not actually result in a
# FQDN (for instance, Solaris).
#append_domain:

# Custom static grains for this minion can be specified here and used in SLS
# files just like all other grains. This example sets 4 custom grains, with
# the 'roles' grain having two values that can be matched against:
#grains:
#  roles:
#    - webserver
#    - memcache
#  deployment: datacenter4
#  cabinet: 13
#  cab_u: 14-15

# Where cache data goes
#cachedir: /var/cache/salt/minion

# Verify and set permissions on configuration directories at startup
#verify_env: True
```

```
# The minion can locally cache the return data from jobs sent to it, this
# can be a good way to keep track of jobs the minion has executed
# (on the minion side). By default this feature is disabled, to enable
# set cache_jobs to True
#cache_jobs: False

# set the directory used to hold unix sockets
#sock_dir: /var/run/salt/minion

# Set the default outputter used by the salt-call command. The default is
# "nested"
#output: nested
#
# By default output is colored, to disable colored output set the color value
# to False
#color: True

# Backup files that are replaced by file.managed and file.recurse under
# 'cachedir'/file_backups relative to their original location and appended
# with a timestamp. The only valid setting is "minion". Disabled by default.
#
# Alternatively this can be specified for each file in state files:
#
# /etc/ssh/sshd_config:
#   file.managed:
#     - source: salt://ssh/sshd_config
#     - backup: minion
#
#backup_mode: minion

# When waiting for a master to accept the minion's public key, salt will
# continuously attempt to reconnect until successful. This is the time, in
# seconds, between those reconnection attempts.
#acceptance_wait_time: 10

# If this is nonzero, the time between reconnection attempts will increase by
# acceptance_wait_time seconds per iteration, up to this maximum. If this is
# set to zero, the time between reconnection attempts will stay constant.
#acceptance_wait_time_max: 0

# If the master rejects the minion's public key, retry instead exiting. Rejected keys
# # will be handled the same as waiting on acceptance.
#rejected_retry: False

# When the master key changes, the minion will try to re-auth itself to receive
# the new master key. In larger environments this can cause a SYN flood on the
# master because all minions try to re-auth immediately. To prevent this and
# have a minion wait for a random amount of time, use this optional parameter.
# The wait-time will be a random number of seconds between
# 0 and the defined value.
#random_reauth_delay: 60

# When waiting for a master to accept the minion's public key, salt will
# continuously attempt to reconnect until successful. This is the timeout value,
# in seconds, for each individual attempt. After this timeout expires, the minion
# will wait for acceptance_wait_time seconds before trying again.
# Unless your master is under unusually heavy load, this should be left at the default.
#auth_timeout: 3
```

```
# If you don’t have any problems with syn-floods, dont bother with the
# three recon_* settings described below, just leave the defaults!
#
# The ZeroMQ pull-socket that binds to the masters publishing interface tries
# to reconnect immediately, if the socket is disconnected (for example if
# the master processes are restarted). In large setups this will have all
# minions reconnect immediately which might flood the master (the ZeroMQ-default
# is usually a 100ms delay). To prevent this, these three recon_* settings
# can be used.
#
# recon_default: the interval in milliseconds that the socket should wait before
#                trying to reconnect to the master (100ms = 1 second)
#
# recon_max: the maximum time a socket should wait. each interval the time to wait
#            is calculated by doubling the previous time. if recon_max is reached,
#            it starts again at recon_default. Short example:
#
#            reconnect 1: the socket will wait 'recon_default' milliseconds
#            reconnect 2: 'recon_default' * 2
#            reconnect 3: ('recon_default' * 2) * 2
#            reconnect 4: value from previous interval * 2
#            reconnect 5: value from previous interval * 2
#            reconnect x: if value >= recon_max, it starts again with recon_default
#
# recon_randomize: generate a random wait time on minion start. The wait time will
#                  be a random value between recon_default and recon_default +
#                  recon_max. Having all minions reconnect with the same recon_default
#                  and recon_max value kind of defeats the purpose of being able to
#                  change these settings. If all minions have the same values and your
#                  setup is quite large (several thousand minions), they will still
#                  flood the master. The desired behaviour is to have timeframe within
#                  all minions try to reconnect.

# Example on how to use these settings:
# The goal: have all minions reconnect within a 60 second timeframe on a disconnect
#
# The settings:
#recon_default: 1000
#recon_max: 59000
#recon_randomize: True
#
# Each minion will have a randomized reconnect value between 'recon_default'
# and 'recon_default + recon_max', which in this example means between 1000ms
# 60000ms (or between 1 and 60 seconds). The generated random-value will be
# doubled after each attempt to reconnect. Lets say the generated random
# value is 11 seconds (or 11000ms).
#
# reconnect 1: wait 11 seconds
# reconnect 2: wait 22 seconds
# reconnect 3: wait 33 seconds
# reconnect 4: wait 44 seconds
# reconnect 5: wait 55 seconds
# reconnect 6: wait time is bigger than 60 seconds (recon_default + recon_max)
# reconnect 7: wait 11 seconds
# reconnect 8: wait 22 seconds
# reconnect 9: wait 33 seconds
# reconnect x: etc.
```

```
#
# In a setup with ~6000 thousand hosts these settings would average the reconnects
# to about 100 per second and all hosts would be reconnected within 60 seconds.
#recon_default: 100
#recon_max: 5000
#recon_randomize: False

# The loop_interval sets how long in seconds the minion will wait between
# evaluating the scheduler and running cleanup tasks. This defaults to a
# sane 60 seconds, but if the minion scheduler needs to be evaluated more
# often lower this value
#loop_interval: 60

# The grains_refresh_every setting allows for a minion to periodically check
# its grains to see if they have changed and, if so, to inform the master
# of the new grains. This operation is moderately expensive, therefore
# care should be taken not to set this value too low.
#
# Note: This value is expressed in __minutes__!
#
# A value of 10 minutes is a reasonable default.
#
# If the value is set to zero, this check is disabled.
#grains_refresh_every = 1

# Cache grains on the minion. Default is False.
# grains_cache: False

# Grains cache expiration, in seconds. If the cache file is older than this
# number of seconds then the grains cache will be dumped and fully re-populated
# with fresh data. Defaults to 5 minutes. Will have no effect if 'grains_cache'
# is not enabled.
# grains_cache_expiration: 300


# When healing, a dns_check is run. This is to make sure that the originally
# resolved dns has not changed. If this is something that does not happen in
# your environment, set this value to False.
#dns_check: True

# Windows platforms lack posix IPC and must rely on slower TCP based inter-
# process communications. Set ipc_mode to 'tcp' on such systems
#ipc_mode: ipc
#
# Overwrite the default tcp ports used by the minion when in tcp mode
#tcp_pub_port: 4510
#tcp_pull_port: 4511

# The minion can include configuration from other files. To enable this,
# pass a list of paths to this option. The paths can be either relative or
# absolute; if relative, they are considered to be relative to the directory
# the main minion configuration file lives in (this file). Paths can make use
# of shell-style globbing. If no files are matched by a path passed to this
# option then the minion will log a warning message.
#
#
# Include a config file from some other path:
# include: /etc/salt/extra_config
```

```
#
# Include config from several files and directories:
#include:
#  - /etc/salt/extra_config
#  - /etc/roles/webserver

#####   Minion module management     #####
##########################################
# Disable specific modules. This allows the admin to limit the level of
# access the master has to the minion
#disable_modules: [cmd,test]
#disable_returners: []
#
# Modules can be loaded from arbitrary paths. This enables the easy deployment
# of third party modules. Modules for returners and minions can be loaded.
# Specify a list of extra directories to search for minion modules and
# returners. These paths must be fully qualified!
#module_dirs: []
#returner_dirs: []
#states_dirs: []
#render_dirs: []
#
# A module provider can be statically overwritten or extended for the minion
# via the providers option, in this case the default module will be
# overwritten by the specified module. In this example the pkg module will
# be provided by the yumpkg5 module instead of the system default.
#
#providers:
#  pkg: yumpkg5
#
# Enable Cython modules searching and loading. (Default: False)
#cython_enable: False
#
#
#
# Specify a max size (in bytes) for modules on import
# this feature is currently only supported on *nix OSs and requires psutil
# modules_max_memory: -1



#####    State Management Settings    #####
###########################################
# The state management system executes all of the state templates on the minion
# to enable more granular control of system state management. The type of
# template and serialization used for state management needs to be configured
# on the minion, the default renderer is yaml_jinja. This is a yaml file
# rendered from a jinja template, the available options are:
# yaml_jinja
# yaml_mako
# yaml_wempy
# json_jinja
# json_mako
# json_wempy
#
#renderer: yaml_jinja
#
# The failhard option tells the minions to stop immediately after the first
# failure detected in the state execution, defaults to False
#failhard: False
#
# autoload_dynamic_modules Turns on automatic loading of modules found in the
# environments on the master. This is turned on by default, to turn of
# autoloading modules when states run set this value to False
#autoload_dynamic_modules: True
#
# clean_dynamic_modules keeps the dynamic modules on the minion in sync with
# the dynamic modules on the master, this means that if a dynamic module is
# not on the master it will be deleted from the minion. By default this is
# enabled and can be disabled by changing this value to False
#clean_dynamic_modules: True
#
# Normally the minion is not isolated to any single environment on the master
# when running states, but the environment can be isolated on the minion side
# by statically setting it. Remember that the recommended way to manage
# environments is to isolate via the top file.
#environment: None
#
# If using the local file directory, then the state top file name needs to be
# defined, by default this is top.sls.
#state_top: top.sls
#
# Run states when the minion daemon starts. To enable, set startup_states to:
# 'highstate' -- Execute state.highstate
# 'sls' -- Read in the sls_list option and execute the named sls files
# 'top' -- Read top_file option and execute based on that file on the Master
#startup_states: ''
#
# list of states to run when the minion starts up if startup_states is 'sls'
#sls_list:
#  - edit.vim
#  - hyper
#
# top file to execute if startup_states is 'top'
#top_file: ''

#####     File Directory Settings    #####
##########################################
# The Salt Minion can redirect all file server operations to a local directory,
# this allows for the same state tree that is on the master to be used if
# copied completely onto the minion. This is a literal copy of the settings on
# the master but used to reference a local directory on the minion.

# Set the file client. The client defaults to looking on the master server for
# files, but can be directed to look at the local file directory setting
# defined below by setting it to local.
#file_client: remote

# The file directory works on environments passed to the minion, each environment
# can have multiple root directories, the subdirectories in the multiple file
# roots cannot match, otherwise the downloaded files will not be able to be
# reliably ensured. A base environment is required to house the top file.
# Example:
# file_roots:
#   base:
#     - /srv/salt/
#   dev:
#     - /srv/salt/dev/services
#     - /srv/salt/dev/states
#   prod:
#     - /srv/salt/prod/services
#     - /srv/salt/prod/states
#
#file_roots:
#  base:
#    - /srv/salt

# By default, the Salt fileserver recurses fully into all defined environments
# to attempt to find files. To limit this behavior so that the fileserver only
# traverses directories with SLS files and special Salt directories like _modules,
# enable the option below. This might be useful for installations where a file root
# has a very large number of files and performance is negatively impacted.
#
# Default is False.
#
# fileserver_limit_traversal: False

# The hash_type is the hash to use when discovering the hash of a file in
# the local fileserver. The default is md5, but sha1, sha224, sha256, sha384
# and sha512 are also supported.
#hash_type: md5

# The Salt pillar is searched for locally if file_client is set to local. If
# this is the case, and pillar data is defined, then the pillar_roots need to
# also be configured on the minion:
#pillar_roots:
#  base:
#    - /srv/pillar

######        Security settings       #####
###########################################
# Enable "open mode", this mode still maintains encryption, but turns off
# authentication, this is only intended for highly secure environments or for
# the situation where your keys end up in a bad state. If you run in open mode
# you do so at your own risk!
#open_mode: False

# Enable permissive access to the salt keys.  This allows you to run the
# master or minion as root, but have a non-root group be given access to
# your pki_dir.  To make the access explicit, root must belong to the group
# you've given access to. This is potentially quite insecure.
#permissive_pki_access: False

# The state_verbose and state_output settings can be used to change the way
# state system data is printed to the display. By default all data is printed.
# The state_verbose setting can be set to True or False, when set to False
# all data that has a result of True and no changes will be suppressed.
#state_verbose: True
#
# The state_output setting changes if the output is the full multi line
# output for each changed state if set to 'full', but if set to 'terse'
# the output will be shortened to a single line.
#state_output: full
#
```

```
# Fingerprint of the master public key to double verify the master is valid,
# the master fingerprint can be found by running "salt-key -F master" on the
# salt master.
#master_finger: ’’

######         Thread settings        #####
###########################################
# Disable multiprocessing support, by default when a minion receives a
# publication a new process is spawned and the command is executed therein.
#multiprocessing: True

#####         Logging settings       #####
##########################################
# The location of the minion log file
# The minion log can be sent to a regular file, local path name, or network
# location. Remote logging works best when configured to use rsyslogd(8) (e.g.:
# ‘‘file:///dev/log‘‘), with rsyslogd(8) configured for network logging. The URI
# format is: <file|udp|tcp>://<host|socketpath>:<port-if-required>/<log-facility>
#log_file: /var/log/salt/minion
#log_file: file:///dev/log
#log_file: udp://loghost:10514
#
#log_file: /var/log/salt/minion
#key_logfile: /var/log/salt/key
#
# The level of messages to send to the console.
# One of ’garbage’, ’trace’, ’debug’, info’, ’warning’, ’error’, ’critical’.
# Default: ’warning’
#log_level: warning
#
# The level of messages to send to the log file.
# One of ’garbage’, ’trace’, ’debug’, info’, ’warning’, ’error’, ’critical’.
# Default: ’warning’
#log_level_logfile:

# The date and time format used in log messages. Allowed date/time formating
# can be seen here: http://docs.python.org/library/time.html#time.strftime
#log_datefmt: ’%H:%M:%S’
#log_datefmt_logfile: ’%Y-%m-%d %H:%M:%S’
#
# The format of the console logging messages. Allowed formatting options can
# be seen here: http://docs.python.org/library/logging.html#logrecord-attributes
#log_fmt_console: ’[%(levelname)-8s] %(message)s’
#log_fmt_logfile: ’%(asctime)s,%(msecs)03.0f [%(name)-17s][%(levelname)-8s] %(message)s’
#
# This can be used to control logging levels more specificically.  This
# example sets the main salt library at the ’warning’ level, but sets
# ’salt.modules’ to log at the ’debug’ level:
#   log_granular_levels:
#     ’salt’: ’warning’,
#     ’salt.modules’: ’debug’
#
#log_granular_levels: {}

######      Module configuration      #####
###########################################
# Salt allows for modules to be passed arbitrary configuration data, any data
# passed here in valid yaml format will be passed on to the salt minion modules
# for use. It is STRONGLY recommended that a naming convention be used in which
# the module name is followed by a . and then the value. Also, all top level
# data must be applied via the yaml dict construct, some examples:
#
# You can specify that all modules should run in test mode:
#test: True
#
# A simple value for the test module:
#test.foo: foo
#
# A list for the test module:
#test.bar: [baz,quo]
#
# A dict for the test module:
#test.baz: {spam: sausage, cheese: bread}


######      Update settings          ######
###########################################
# Using the features in Esky, a salt minion can both run as a frozen app and
# be updated on the fly. These options control how the update process
# (saltutil.update()) behaves.
#
# The url for finding and downloading updates. Disabled by default.
#update_url: False
#
# The list of services to restart after a successful update. Empty by default.
#update_restart_services: []


######      Keepalive settings        ######
############################################
# ZeroMQ now includes support for configuring SO_KEEPALIVE if supported by
# the OS. If connections between the minion and the master pass through
# a state tracking device such as a firewall or VPN gateway, there is
# the risk that it could tear down the connection the master and minion
# without informing either party that their connection has been taken away.
# Enabling TCP Keepalives prevents this from happening.
#
# Overall state of TCP Keepalives, enable (1 or True), disable (0 or False)
# or leave to the OS defaults (-1), on Linux, typically disabled. Default True, enabled.
#tcp_keepalive: True
#
# How long before the first keepalive should be sent in seconds. Default 300
# to send the first keepalive after 5 minutes, OS default (-1) is typically 7200 seconds
# on Linux see /proc/sys/net/ipv4/tcp_keepalive_time.
#tcp_keepalive_idle: 300
#
# How many lost probes are needed to consider the connection lost. Default -1
# to use OS defaults, typically 9 on Linux, see /proc/sys/net/ipv4/tcp_keepalive_probes.
#tcp_keepalive_cnt: -1
#
# How often, in seconds, to send keepalives after the first one. Default -1 to
# use OS defaults, typically 75 seconds on Linux, see
# /proc/sys/net/ipv4/tcp_keepalive_intvl.
#tcp_keepalive_intvl: -1
```

```
######      Windows Software settings ######
############################################
# Location of the repository cache file on the master
#win_repo_cachefile: 'salt://win/repo/winrepo.p'
```

CHAPTER 104

Configuring the Salt Master

The Salt system is amazingly simple and easy to configure, the two components of the Salt system each have a respective configuration file. The **salt-master** is configured via the master configuration file, and the **salt-minion** is configured via the minion configuration file.

See also:

example master configuration file

The configuration file for the salt-master is located at `/etc/salt/master`. The available options are as follows:

104.1 Primary Master Configuration

104.1.1 `interface`

Default: `0.0.0.0` (all interfaces)

The local interface to bind to.

```
interface: 192.168.0.1
```

104.1.2 `publish_port`

Default: `4505`

The network port to set up the publication interface

```
publish_port: 4505
```

104.1.3 `user`

Default: `root`

The user to run the Salt processes

```
user: root
```

104.1.4 `max_open_files`

Default: `max_open_files`

Each minion connecting to the master uses AT LEAST one file descriptor, the master subscription connection. If enough minions connect you might start seeing on the console(and then salt-master crashes):

```
Too many open files (tcp_listener.cpp:335)
Aborted (core dumped)
```

By default this value will be the one of *ulimit -Hn*, i.e., the hard limit for max open files.

If you wish to set a different value than the default one, uncomment and configure this setting. Remember that this value CANNOT be higher than the hard limit. Raising the hard limit depends on your OS and/or distribution, a good way to find the limit is to search the internet for(for example):

```
raise max open files hard limit debian
```

```
max_open_files: 100000
```

104.1.5 `worker_threads`

Default: `5`

The number of threads to start for receiving commands and replies from minions. If minions are stalling on replies because you have many minions, raise the worker_threads value.

Worker threads should not be put below 3 when using the peer system, but can drop down to 1 worker otherwise.

```
worker_threads: 5
```

104.1.6 `ret_port`

Default: `4506`

The port used by the return server, this is the server used by Salt to receive execution returns and command executions.

```
ret_port: 4506
```

104.1.7 `pidfile`

Default: `/var/run/salt-master.pid`

Specify the location of the master pidfile

```
pidfile: /var/run/salt-master.pid
```

104.1.8 `root_dir`

Default: `/`

The system root directory to operate from, change this to make Salt run from an alternative root.

```
root_dir: /
```

Note: This directory is prepended to the following options: `pki_dir`, `cachedir`, `sock_dir`, `log_file`, `autosign_file`, `autoreject_file`, `pidfile`.

104.1.9 `pki_dir`

Default: `/etc/salt/pki`

The directory to store the pki authentication keys.

```
pki_dir: /etc/salt/pki
```

104.1.10 `cachedir`

Default: `/var/cache/salt`

The location used to store cache information, particularly the job information for executed salt commands.

```
cachedir: /var/cache/salt
```

104.1.11 `keep_jobs`

Default: `24`

Set the number of hours to keep old job information

104.1.12 `job_cache`

Default: `True`

The master maintains a job cache, while this is a great addition it can be a burden on the master for larger deployments (over 5000 minions). Disabling the job cache will make previously executed jobs unavailable to the jobs system and is not generally recommended. Normally it is wise to make sure the master has access to a faster IO system or a tmpfs is mounted to the jobs dir

104.1.13 `ext_job_cache`

Default: ‘’

Used to specify a default returner for all minions, when this option is set the specified returner needs to be properly configured and the minions will always default to sending returns to this returner. This will also disable the local job cache on the master

```
ext_job_cache: redis
```

104.1.14 `minion_data_cache`

Default: True

The minion data cache is a cache of information about the minions stored on the master, this information is primarily the pillar and grains data. The data is cached in the Master cachedir under the name of the minion and used to pre determine what minions are expected to reply from executions.

```
minion_cache_dir: True
```

104.1.15 `enforce_mine_cache`

Default: False

By-default when disabling the minion_data_cache mine will stop working since it is based on cached data, by enabling this option we explicitly enabling only the cache for the mine system.

```
enforce_mine_cache: False
```

104.1.16 `sock_dir`

Default: `/tmp/salt-unix`

Set the location to use for creating Unix sockets for master process communication

104.2 Master Security Settings

104.2.1 `open_mode`

Default: `False`

Open mode is a dangerous security feature. One problem encountered with pki authentication systems is that keys can become "mixed up" and authentication begins to fail. Open mode turns off authentication and tells the master to accept all authentication. This will clean up the pki keys received from the minions. Open mode should not be turned on for general use. Open mode should only be used for a short period of time to clean up pki keys. To turn on open mode set this value to `True`.

```
open_mode: False
```

104.2.2 `auto_accept`

Default: `False`

Enable auto_accept. This setting will automatically accept all incoming public keys from minions.

```
auto_accept: False
```

104.2.3 `autosign_file`

Default: `not defined`

If the `autosign_file` is specified incoming keys specified in the autosign_file will be automatically accepted. Matches will be searched for first by string comparison, then by globbing, then by full-string regex matching. This is insecure!

104.2.4 `autoreject_file`

New in version 2014.1.0: (Hydrogen)

Default: `not defined`

Works like `autosign_file`, but instead allows you to specify minion IDs for which keys will automatically be rejected. Will override both membership in the `autosign_file` and the `auto_accept` setting.

104.2.5 `client_acl`

Default: `{}`

Enable user accounts on the master to execute specific modules. These modules can be expressed as regular expressions

```
client_acl:
  fred:
    - test.ping
    - pkg.*
```

104.2.6 `client_acl_blacklist`

Default: `{}`

Blacklist users or modules

This example would blacklist all non sudo users, including root from running any commands. It would also blacklist any use of the "cmd" module.

This is completely disabled by default.

```
client_acl_blacklist:
  users:
    - root
    - '^(?!sudo_).*$'   #  all non sudo users
  modules:
    - cmd
```

104.2.7 `external_auth`

Default: `{}`

The external auth system uses the Salt auth modules to authenticate and validate users to access areas of the Salt system.

```
external_auth:
  pam:
    fred:
      - test.*
```

104.2.8 `token_expire`

Default: `43200`

Time (in seconds) for a newly generated token to live. Default: 12 hours

```
token_expire: 43200
```

104.2.9 `file_recv`

Default: `False`

Allow minions to push files to the master. This is disabled by default, for security purposes.

```
file_recv: False
```

104.3 Master Module Management

104.3.1 `runner_dirs`

Default: `[]`

Set additional directories to search for runner modules

104.3.2 `cython_enable`

Default: `False`

Set to true to enable cython modules (.pyx files) to be compiled on the fly on the Salt master

```
cython_enable: False
```

104.4 Master State System Settings

104.4.1 `state_verbose`

Default: `False`

state_verbose allows for the data returned from the minion to be more verbose. Normally only states that fail or states that have changes are returned, but setting state_verbose to `True` will return all states that were checked

```
state_verbose: True
```

104.4.2 `state_output`

Default: `full`

The state_output setting changes if the output is the full multi line output for each changed state if set to 'full', but if set to 'terse' the output will be shortened to a single line. If set to 'mixed', the output will be terse unless a state failed, in which case that output will be full. If set to 'changes', the output will be full unless the state didn't change.

```
state_output: full
```

104.4.3 `state_top`

Default: `top.sls`

The state system uses a "top" file to tell the minions what environment to use and what modules to use. The state_top file is defined relative to the root of the base environment

```
state_top: top.sls
```

104.4.4 `external_nodes`

Default: None

The external_nodes option allows Salt to gather data that would normally be placed in a top file from and external node controller. The external_nodes option is the executable that will return the ENC data. Remember that Salt will look for external nodes AND top files and combine the results if both are enabled and available!

```
external_nodes: cobbler-ext-nodes
```

104.4.5 `renderer`

Default: `yaml_jinja`

The renderer to use on the minions to render the state data

```
renderer: yaml_jinja
```

104.4.6 `failhard`

Default: `False`

Set the global failhard flag, this informs all states to stop running states at the moment a single state fails

```
failhard: False
```

104.4.7 `yaml_utf8`

Default: `False`

Enable extra yaml render routines for states containing UTF characters

```
yaml_utf8: False
```

104.4.8 `test`

Default: `False`

Set all state calls to only test if they are going to actually make changes or just post what changes are going to be made

```
test: False
```

104.5 Master File Server Settings

104.5.1 `fileserver_backend`

Default:

```
fileserver_backend:
  - roots
```

Salt supports a modular fileserver backend system, this system allows the salt master to link directly to third party systems to gather and manage the files available to minions. Multiple backends can be configured and will be searched for the requested file in the order in which they are defined here. The default setting only enables the standard backend `roots`, which is configured using the `file_roots` option.

Example:

```
fileserver_backend:
  - roots
  - gitfs
```

104.5.2 `file_roots`

Default:

```
base:
  - /srv/salt
```

Salt runs a lightweight file server written in ZeroMQ to deliver files to minions. This file server is built into the master daemon and does not require a dedicated port.

The file server works on environments passed to the master. Each environment can have multiple root directories. The subdirectories in the multiple file roots cannot match, otherwise the downloaded files will not be able to be reliably ensured. A base environment is required to house the top file. Example:

```
file_roots:
  base:
    - /srv/salt
  dev:
    - /srv/salt/dev/services
    - /srv/salt/dev/states
  prod:
    - /srv/salt/prod/services
    - /srv/salt/prod/states
```

104.5.3 `hash_type`

Default: `md5`

The hash_type is the hash to use when discovering the hash of a file on the master server. The default is md5, but sha1, sha224, sha256, sha384 and sha512 are also supported.

```
hash_type: md5
```

104.5.4 `file_buffer_size`

Default: `1048576`

The buffer size in the file server in bytes

```
file_buffer_size: 1048576
```

104.5.5 `file_ignore_regex`

Default: `''`

A regular expression (or a list of expressions) that will be matched against the file path before syncing the modules and states to the minions. This includes files affected by the file.recurse state. For example, if you manage your custom modules and states in subversion and don't want all the '.svn' folders and content synced to your minions, you could set this to '/.svn($|/)'. By default nothing is ignored.

```
file_ignore_regex:
  - '/\.svn($|/)'
  - '/\.git($|/)'
```

104.5.6 `file_ignore_glob`

Default `''`

A file glob (or list of file globs) that will be matched against the file path before syncing the modules and states to the minions. This is similar to file_ignore_regex above, but works on globs instead of regex. By default nothing is ignored.

```
file_ignore_glob:
  - '\*.pyc'
  - '\*/somefolder/\*.bak'
  - '\*.swp'
```

104.5.7 `fileserver_backend`

Default:

```
fileserver_backend:
  - roots
```

Salt supports a modular fileserver backend system, this system allows the salt master to link directly to third party systems to gather and manage the files available to minions. Multiple backends can be configured and will be searched for the requested file in the order in which they are defined here. The default setting only enables the standard backend `roots`, which is configured using the `file_roots` option.

Example:

```
fileserver_backend:
  - roots
  - git
```

104.5.8 `gitfs_provider`

New in version Helium.

Gitfs can be provided by one of two python modules: GitPython or pygit2. If using pygit2, both libgit2 and git itself must also be installed. More information can be found in the `gitfs backend documentation`.

```
gitfs_provider: pygit2
```

104.5.9 `gitfs_remotes`

Default: `[]`

When using the `git` fileserver backend at least one git remote needs to be defined. The user running the salt master will need read access to the repo.

The repos will be searched in order to find the file requested by a client and the first repo to have the file will return it. Branches and tags are translated into salt environments.

```
gitfs_remotes:
  - git://github.com/saltstack/salt-states.git
  - file:///var/git/saltmaster
```

Note: `file://` repos will be treated as a remote, so refs you want used must exist in that repo as *local* refs.

104.5.10 `gitfs_ssl_verify`

Default: `[]`

The `gitfs_ssl_verify` option specifies whether to ignore ssl certificate errors when contacting the gitfs backend. You might want to set this to false if you're using a git backend that uses a self-signed certificate but keep in mind that setting this flag to anything other than the default of True is a security concern, you may want to try using the ssh transport.

```
gitfs_ssl_verify: True
```

104.5.11 `gitfs_root`

Default: `''`

Serve files from a subdirectory within the repository, instead of the root. This is useful when there are files in the repository that should not be available to the Salt fileserver.

```
gitfs_root: somefolder/otherfolder
```

104.5.12 `gitfs_base`

Default: `master`

Defines which branch/tag should be used as the `base` environment.

```
gitfs_base: salt
```

104.5.13 `hgfs_remotes`

New in version 0.17.0.

Default: `[]`

When using the `hg` fileserver backend at least one mercurial remote needs to be defined. The user running the salt master will need read access to the repo.

The repos will be searched in order to find the file requested by a client and the first repo to have the file will return it. Branches and/or bookmarks are translated into salt environments, as defined by the `hgfs_branch_method` parameter.

```
hgfs_remotes:
  - https://username@bitbucket.org/username/reponame
```

104.5.14 `hgfs_branch_method`

New in version 0.17.0.

Default: `branches`

Defines the objects that will be used as fileserver environments.

- `branches` - Only branches and tags will be used
- `bookmarks` - Only bookmarks and tags will be used
- `mixed` - Branches, bookmarks, and tags will be used

```
hgfs_branch_method: mixed
```

Note: Starting in version 2014.1.0 (Hydrogen), the value of the `hgfs_base` parameter defines which branch is used as the `base` environment, allowing for a `base` environment to be used with an `hgfs_branch_method` of `bookmarks`.

Prior to this release, the `default` branch will be used as the `base` environment.

104.5.15 `hgfs_root`

New in version 0.17.0.

Default: `''`

Serve files from a subdirectory within the repository, instead of the root. This is useful when there are files in the repository that should not be available to the Salt fileserver.

```
hgfs_root: somefolder/otherfolder
```

104.5.16 `hgfs_base`

New in version 2014.1.0: (Hydrogen)

Default: `default`

Defines which branch should be used as the `base` environment. Change this if `hgfs_branch_method` is set to `bookmarks` to specify which bookmark should be used as the `base` environment.

```
hgfs_base: salt
```

104.6 Pillar Configuration

104.6.1 `pillar_roots`

Default:

```
base:
  - /srv/pillar
```

Set the environments and directories used to hold pillar sls data. This configuration is the same as `file_roots`:

```
pillar_roots:
  base:
    - /srv/pillar
  dev:
    - /srv/pillar/dev
  prod:
    - /srv/pillar/prod
```

104.6.2 `ext_pillar`

The ext_pillar option allows for any number of external pillar interfaces to be called when populating pillar data. The configuration is based on ext_pillar functions. The available ext_pillar functions can be found herein:

https://github.com/saltstack/salt/blob/develop/salt/pillar

By default, the ext_pillar interface is not configured to run.

Default: `None`

```
ext_pillar:
  - hiera: /etc/hiera.yaml
  - cmd_yaml: cat /etc/salt/yaml
  - reclass:
      inventory_base_uri: /etc/reclass
```

There are additional details at *Pillars*

104.7 Syndic Server Settings

A Salt syndic is a Salt master used to pass commands from a higher Salt master to minions below the syndic. Using the syndic is simple. If this is a master that will have syndic servers(s) below it, set the "order_masters" setting to True. If this is a master that will be running a syndic daemon for passthrough the "syndic_master" setting needs to be set to the location of the master server

Do not not forget that in other word it means that it shares with the local minion it's ID and PKI_DIR.

104.7.1 `order_masters`

Default: `False`

Extra data needs to be sent with publications if the master is controlling a lower level master via a syndic minion. If this is the case the order_masters value must be set to True

```
order_masters: False
```

104.7.2 `syndic_master`

Default: `None`

If this master will be running a salt-syndic to connect to a higher level master, specify the higher level master with this configuration value

```
syndic_master: masterofmasters
```

104.7.3 `syndic_master_port`

Default: `4506`

If this master will be running a salt-syndic to connect to a higher level master, specify the higher level master port with this configuration value

```
syndic_master_port: 4506
```

104.7.4 `syndic_log_file`

Default: `syndic.log`

If this master will be running a salt-syndic to connect to a higher level master, specify the log_file of the syndic daemon.

```
syndic_log_file: salt-syndic.log
```

104.7.5 `syndic_pidfile`

Default: `salt-syndic.pid`

If this master will be running a salt-syndic to connect to a higher level master, specify the pidfile of the syndic daemon.

```
syndic_pidfile: syndic.pid
```

104.8 Peer Publish Settings

Salt minions can send commands to other minions, but only if the minion is allowed to. By default “Peer Publication” is disabled, and when enabled it is enabled for specific minions and specific commands. This allows secure compartmentalization of commands based on individual minions.

104.8.1 `peer`

Default: `{}`

The configuration uses regular expressions to match minions and then a list of regular expressions to match functions. The following will allow the minion authenticated as foo.example.com to execute functions from the test and pkg modules

```
peer:
  foo.example.com:
      - test.*
      - pkg.*
```

This will allow all minions to execute all commands:

```
peer:
  .*:
      - .*
```

This is not recommended, since it would allow anyone who gets root on any single minion to instantly have root on all of the minions!

By adding an additional layer you can limit the target hosts in addition to the accessible commands:

```
peer:
  foo.example.com:
    'db*':
      - test.*
      - pkg.*
```

104.8.2 `peer_run`

Default: `{}`

The peer_run option is used to open up runners on the master to access from the minions. The peer_run configuration matches the format of the peer configuration.

The following example would allow foo.example.com to execute the manage.up runner:

```
peer_run:
  foo.example.com:
      - manage.up
```

104.9 Node Groups

Default: `{}`

Node groups allow for logical groupings of minion nodes. A group consists of a group name and a compound target.

```
nodegroups:
  group1: 'L@foo.domain.com,bar.domain.com,baz.domain.com or bl*.domain.com'
  group2: 'G@os:Debian and foo.domain.com'
```

104.10 Master Logging Settings

104.10.1 `log_file`

Default: `/var/log/salt/master`

The master log can be sent to a regular file, local path name, or network location. See also `log_file`.

Examples:

```
log_file: /var/log/salt/master
```

```
log_file: file:///dev/log
```

```
log_file: udp://loghost:10514
```

104.10.2 `log_level`

Default: `warning`

The level of messages to send to the console. See also `log_level`.

```
log_level: warning
```

104.10.3 `log_level_logfile`

Default: `warning`

The level of messages to send to the log file. See also `log_level_logfile`.

```
log_level_logfile: warning
```

104.10.4 `log_datefmt`

Default: `%H:%M:%S`

The date and time format used in console log messages. See also `log_datefmt`.

```
log_datefmt: '%H:%M:%S'
```

104.10.5 `log_datefmt_logfile`

Default: `%Y-%m-%d %H:%M:%S`

The date and time format used in log file messages. See also `log_datefmt_logfile`.

```
log_datefmt_logfile: '%Y-%m-%d %H:%M:%S'
```

104.10.6 `log_fmt_console`

Default: `[%(levelname)-8s] %(message)s`

The format of the console logging messages. See also `log_fmt_console`.

```
log_fmt_console: '[%(levelname)-8s] %(message)s'
```

104.10.7 `log_fmt_logfile`

Default: `%(asctime)s,%(msecs)03.0f [%(name)-17s][%(levelname)-8s] %(message)s`

The format of the log file logging messages. See also `log_fmt_logfile`.

```
log_fmt_logfile: '%(asctime)s,%(msecs)03.0f [%(name)-17s][%(levelname)-8s] %(message)s'
```

104.10.8 `log_granular_levels`

Default: `{}`

This can be used to control logging levels more specifically. See also `log_granular_levels`.

104.11 Include Configuration

104.11.1 `default_include`

Default: `master.d/*.conf`

The master can include configuration from other files. Per default the master will automatically include all config files from `master.d/*.conf` where `master.d` is relative to the directory of the master configuration file.

104.11.2 `include`

Default: `not defined`

The master can include configuration from other files. To enable this, pass a list of paths to this option. The paths can be either relative or absolute; if relative, they are considered to be relative to the directory the main minion configuration file lives in. Paths can make use of shell-style globbing. If no files are matched by a path passed to this option then the master will log a warning message.

```
# Include files from a master.d directory in the same
# directory as the master config file
include: master.d/*

# Include a single extra file into the configuration
include: /etc/roles/webserver

# Include several files and the master.d directory
include:
  - extra_config
  - master.d/*
  - /etc/roles/webserver
```

CHAPTER 105

Configuring the Salt Minion

The Salt system is amazingly simple and easy to configure, the two components of the Salt system each have a respective configuration file. The **salt-master** is configured via the master configuration file, and the **salt-minion** is configured via the minion configuration file.

See also:

example minion configuration file

The Salt Minion configuration is very simple, typically the only value that needs to be set is the master value so the minion can find its master.

105.1 Minion Primary Configuration

105.1.1 `master`

Default: `salt`

The hostname or ipv4 of the master.

```
master: salt
```

105.1.2 `master_port`

Default: `4506`

The port of the master ret server, this needs to coincide with the ret_port option on the Salt master.

```
master_port: 4506
```

105.1.3 `user`

Default: `root`

The user to run the Salt processes

```
user: root
```

105.1.4 `pidfile`

Default: `/var/run/salt-minion.pid`

The location of the daemon's process ID file

```
pidfile: /var/run/salt-minion.pid
```

105.1.5 `root_dir`

Default: `/`

This directory is prepended to the following options: `pki_dir`, `cachedir`, `log_file`, `sock_dir`, and `pidfile`.

```
root_dir: /
```

105.1.6 `pki_dir`

Default: `/etc/salt/pki`

The directory used to store the minion's public and private keys.

```
pki_dir: /etc/salt/pki
```

105.1.7 `id`

Default: the system's hostname

See also:

Salt Walkthrough

The **Setting up a Salt Minion** section contains detailed information on how the hostname is determined.

Explicitly declare the id for this minion to use. Since Salt uses detached ids it is possible to run multiple minions on the same machine but with different ids.

```
id: foo.bar.com
```

105.1.8 `append_domain`

Default: `None`

Append a domain to a hostname in the event that it does not exist. This is useful for systems where `socket.getfqdn()` does not actually result in a FQDN (for instance, Solaris).

```
append_domain: foo.org
```

105.1.9 `cachedir`

Default: `/var/cache/salt`

The location for minion cache data.

```
cachedir: /var/cache/salt
```

105.1.10 `verify_env`

Default: `True`

Verify and set permissions on configuration directories at startup.

```
verify_env: True
```

Note: When marked as True the verify_env option requires WRITE access to the configuration directory (/etc/salt/). In certain situations such as mounting /etc/salt/ as read-only for templating this will create a stack trace when state.highstate is called.

105.1.11 `cache_jobs`

Default: `False`

The minion can locally cache the return data from jobs sent to it, this can be a good way to keep track of the minion side of the jobs the minion has executed. By default this feature is disabled, to enable set cache_jobs to `True`.

```
cache_jobs: False
```

105.1.12 `sock_dir`

Default: `/var/run/salt/minion`

The directory where Unix sockets will be kept.

```
sock_dir: /var/run/salt/minion
```

105.1.13 `backup_mode`

Default: `[]`

Backup files replaced by file.managed and file.recurse under cachedir.

```
backup_mode: minion
```

105.1.14 `acceptance_wait_time`

Default: `10`

The number of seconds to wait until attempting to re-authenticate with the master.

```
acceptance_wait_time: 10
```

105.1.15 `random_reauth_delay`

When the master key changes, the minion will try to re-auth itself to receive the new master key. In larger environments this can cause a syn-flood on the master because all minions try to re-auth immediately. To prevent this and have a minion wait for a random amount of time, use this optional parameter. The wait-time will be a random number of seconds between 0 and the defined value.

```
random_reauth_delay: 60
```

105.1.16 `acceptance_wait_time_max`

Default: `None`

The maximum number of seconds to wait until attempting to re-authenticate with the master. If set, the wait will increase by acceptance_wait_time seconds each iteration.

```
acceptance_wait_time_max: None
```

105.1.17 `dns_check`

Default: `True`

When healing, a dns_check is run. This is to make sure that the originally resolved dns has not changed. If this is something that does not happen in your environment, set this value to `False`.

```
dns_check: True
```

105.1.18 `ipc_mode`

Default: `ipc`

Windows platforms lack POSIX IPC and must rely on slower TCP based inter- process communications. Set ipc_mode to `tcp` on such systems.

```
ipc_mode: ipc
```

105.1.19 `tcp_pub_port`

Default: `4510`

Publish port used when `ipc_mode` is set to `tcp`.

```
tcp_pub_port: 4510
```

105.1.20 `tcp_pull_port`

Default: `4511`

Pull port used when `ipc_mode` is set to `tcp`.

```
tcp_pull_port: 4511
```

105.2 Minion Module Management

105.2.1 `disable_modules`

Default: `[]` (all modules are enabled by default)

The event may occur in which the administrator desires that a minion should not be able to execute a certain module. The sys module is built into the minion and cannot be disabled.

This setting can also tune the minion, as all modules are loaded into ram disabling modules will lover the minion's ram footprint.

```
disable_modules:
  - test
  - solr
```

105.2.2 `disable_returners`

Default: `[]` (all returners are enabled by default)

If certain returners should be disabled, this is the place

```
disable_returners:
  - mongo_return
```

105.2.3 `module_dirs`

Default: `[]`

A list of extra directories to search for Salt modules

```
module_dirs:
  - /var/lib/salt/modules
```

105.2.4 `returner_dirs`

Default: `[]`

A list of extra directories to search for Salt returners

```
returners_dirs:
  - /var/lib/salt/returners
```

105.2.5 `states_dirs`

Default: `[]`

A list of extra directories to search for Salt states

```
states_dirs:
  - /var/lib/salt/states
```

105.2.6 `render_dirs`

Default: `[]`

A list of extra directories to search for Salt renderers

```
render_dirs:
  - /var/lib/salt/renderers
```

105.2.7 `cython_enable`

Default: `False`

Set this value to true to enable auto-loading and compiling of `.pyx` modules, This setting requires that `gcc` and `cython` are installed on the minion

```
cython_enable: False
```

105.2.8 `providers`

Default: (empty)

A module provider can be statically overwritten or extended for the minion via the `providers` option. This can be done *on an individual basis in an SLS file*, or globally here in the minion config, like below.

```
providers:
  service: systemd
```

105.3 State Management Settings

105.3.1 `renderer`

Default: `yaml_jinja`

The default renderer used for local state executions

```
renderer: yaml_jinja
```

105.3.2 `state_verbose`

Default: `False`

state_verbose allows for the data returned from the minion to be more verbose. Normally only states that fail or states that have changes are returned, but setting state_verbose to `True` will return all states that were checked

```
state_verbose: True
```

105.3.3 `state_output`

Default: `full`

The state_output setting changes if the output is the full multi line output for each changed state if set to 'full', but if set to 'terse' the output will be shortened to a single line.

```
state_output: full
```

105.3.4 `autoload_dynamic_modules`

Default: `True`

autoload_dynamic_modules Turns on automatic loading of modules found in the environments on the master. This is turned on by default, to turn of auto-loading modules when states run set this value to `False`

```
autoload_dynamic_modules: True
```

Default: `True`

clean_dynamic_modules keeps the dynamic modules on the minion in sync with the dynamic modules on the master, this means that if a dynamic module is not on the master it will be deleted from the minion. By default this is enabled and can be disabled by changing this value to `False`

```
clean_dynamic_modules: True
```

105.3.5 `environment`

Default: `None`

Normally the minion is not isolated to any single environment on the master when running states, but the environment can be isolated on the minion side by statically setting it. Remember that the recommended way to manage environments is to isolate via the top file.

```
environment: None
```

105.4 File Directory Settings

105.4.1 `file_client`

Default: `remote`

The client defaults to looking on the master server for files, but can be directed to look on the minion by setting this parameter to `local`.

```
file_client: remote
```

105.4.2 `file_roots`

Default:

```
base:
  - /srv/salt
```

When using a local `file_client`, this parameter is used to setup the fileserver's environments. This parameter operates identically to the `master config parameter of the same name`.

```
file_roots:
  base:
    - /srv/salt
  dev:
    - /srv/salt/dev/services
    - /srv/salt/dev/states
  prod:
    - /srv/salt/prod/services
    - /srv/salt/prod/states
```

105.4.3 `hash_type`

Default: `md5`

The hash_type is the hash to use when discovering the hash of a file on the local fileserver. The default is md5, but sha1, sha224, sha256, sha384 and sha512 are also supported.

```
hash_type: md5
```

105.4.4 `pillar_roots`

Default:

```
base:
  - /srv/pillar
```

When using a local `file_client`, this parameter is used to setup the pillar environments.

```
pillar_roots:
  base:
    - /srv/pillar
  dev:
    - /srv/pillar/dev
  prod:
    - /srv/pillar/prod
```

105.5 Security Settings

105.5.1 `open_mode`

Default: `False`

Open mode can be used to clean out the PKI key received from the Salt master, turn on open mode, restart the minion, then turn off open mode and restart the minion to clean the keys.

```
open_mode: False
```

105.6 Thread Settings

Default: `True`

Disable multiprocessing support by default when a minion receives a publication a new process is spawned and the command is executed therein.

```
multiprocessing: True
```

105.7 Minion Logging Settings

105.7.1 `log_file`

Default: `/var/log/salt/minion`

The minion log can be sent to a regular file, local path name, or network location. See also `log_file`.

Examples:

```
log_file: /var/log/salt/minion
```

```
log_file: file:///dev/log
```

```
log_file: udp://loghost:10514
```

105.7.2 `log_level`

Default: `warning`

The level of messages to send to the console. See also `log_level`.

```
log_level: warning
```

105.7.3 `log_level_logfile`

Default: `warning`

The level of messages to send to the log file. See also `log_level_logfile`.

```
log_level_logfile: warning
```

105.7.4 `log_datefmt`

Default: `%H:%M:%S`

The date and time format used in console log messages. See also `log_datefmt`.

```
log_datefmt: '%H:%M:%S'
```

105.7.5 `log_datefmt_logfile`

Default: `%Y-%m-%d %H:%M:%S`

The date and time format used in log file messages. See also `log_datefmt_logfile`.

```
log_datefmt_logfile: '%Y-%m-%d %H:%M:%S'
```

105.7.6 `log_fmt_console`

Default: `[%(levelname)-8s] %(message)s`

The format of the console logging messages. See also `log_fmt_console`.

```
log_fmt_console: '[%(levelname)-8s] %(message)s'
```

105.7.7 `log_fmt_logfile`

Default: `%(asctime)s,%(msecs)03.0f [%(name)-17s][%(levelname)-8s] %(message)s`

The format of the log file logging messages. See also `log_fmt_logfile`.

```
log_fmt_logfile: '%(asctime)s,%(msecs)03.0f [%(name)-17s][%(levelname)-8s] %(message)s'
```

105.7.8 `log_granular_levels`

Default: `{}`

This can be used to control logging levels more specifically. See also `log_granular_levels`.

105.8 Include Configuration

105.8.1 `default_include`

Default: `minion.d/*.conf`

The minion can include configuration from other files. Per default the minion will automatically include all config files from *minion.d/*.conf* where minion.d is relative to the directory of the minion configuration file.

105.8.2 `include`

Default: `not defined`

The minion can include configuration from other files. To enable this, pass a list of paths to this option. The paths can be either relative or absolute; if relative, they are considered to be relative to the directory the main minion configuration file lives in. Paths can make use of shell-style globbing. If no files are matched by a path passed to this option then the minion will log a warning message.

```
# Include files from a minion.d directory in the same
# directory as the minion config file
include: minion.d/*.conf
```

```
# Include a single extra file into the configuration
include: /etc/roles/webserver

# Include several files and the minion.d directory
include:
  - extra_config
  - minion.d/*
  - /etc/roles/webserver
```

105.9 Frozen Build Update Settings

These options control how `salt.modules.saltutil.update()` works with esky frozen apps. For more information look at https://github.com/cloudmatrix/esky/.

105.9.1 `update_url`

Default: `False` (Update feature is disabled)

The url to use when looking for application updates. Esky depends on directory listings to search for new versions. A webserver running on your Master is a good starting point for most setups.

```
update_url: 'http://salt.example.com/minion-updates'
```

105.9.2 `update_restart_services`

Default: `[]` (service restarting on update is disabled)

A list of services to restart when the minion software is updated. This would typically just be a list containing the minion's service name, but you may have other services that need to go with it.

```
update_restart_services: ['salt-minion']
```

CHAPTER 106

Salt code and internals

Reference documentation on Salt's internal code.

106.1 Contents

106.1.1 Exceptions

Salt-specific exceptions should be thrown as often as possible so the various interfaces to Salt (CLI, API, etc) can handle those errors appropriately and display error messages appropriately.

`salt.exceptions`	This module is a central location for all salt exceptions

salt.exceptions

This module is a central location for all salt exceptions

exception `salt.exceptions.`**`AuthenticationError`**
If sha256 signature fails during decryption

exception `salt.exceptions.`**`AuthorizationError`**
Thrown when runner or wheel execution fails due to permissions

exception `salt.exceptions.`**`CommandExecutionError`**
Used when a module runs a command which returns an error and wants to show the user the output gracefully instead of dying

exception `salt.exceptions.`**`CommandNotFoundError`**
Used in modules or grains when a required binary is not available

exception `salt.exceptions.`**`EauthAuthenticationError`**
Thrown when eauth authentication fails

exception `salt.exceptions.`**`LoaderError`**
Problems loading the right renderer

exception `salt.exceptions.`**`MasterExit`**
Rise when the master exits

exception `salt.exceptions.`**`MinionError`**
Minion problems reading uris such as salt:// or http://

exception `salt.exceptions.`**`PkgParseError`**

Used when of the pkg modules cannot correctly parse the output from the CLI tool (pacman, yum, apt, aptitude, etc)

exception `salt.exceptions.`**`SaltClientError`**

Problem reading the master root key

exception `salt.exceptions.`**`SaltException`**

Base exception class; all Salt-specific exceptions should subclass this

exception `salt.exceptions.`**`SaltInvocationError`**

Used when the wrong number of arguments are sent to modules or invalid arguments are specified on the command line

exception `salt.exceptions.`**`SaltMasterError`**

Problem reading the master root key

exception `salt.exceptions.`**`SaltRenderError`**(*error*, *line_num=None*, *buf=’‘*, *marker=’ <======================’*, *trace=None*)

Used when a renderer needs to raise an explicit error. If a line number and buffer string are passed, get_context will be invoked to get the location of the error.

exception `salt.exceptions.`**`SaltReqTimeoutError`**

Thrown when a salt master request call fails to return within the timeout

exception `salt.exceptions.`**`SaltRunnerError`**

Problem in runner

exception `salt.exceptions.`**`SaltSystemExit`**(*code=0*, *msg=None*)

This exception is raised when an unsolvable problem is found. There’s nothing else to do, salt should just exit.

exception `salt.exceptions.`**`SaltWheelError`**

Problem in wheel

exception `salt.exceptions.`**`TimedProcTimeoutError`**

Thrown when a timed subprocess does not terminate within the timeout, or if the specified timeout is not an int or a float

exception `salt.exceptions.`**`TokenAuthenticationError`**

Thrown when token authentication fails

106.1.2 salt.exceptions

This module is a central location for all salt exceptions

exception `salt.exceptions.`**`AuthenticationError`**

If sha256 signature fails during decryption

exception `salt.exceptions.`**`AuthorizationError`**

Thrown when runner or wheel execution fails due to permissions

exception `salt.exceptions.`**`CommandExecutionError`**

Used when a module runs a command which returns an error and wants to show the user the output gracefully instead of dying

exception `salt.exceptions.`**`CommandNotFoundError`**

Used in modules or grains when a required binary is not available

exception `salt.exceptions.`**`EauthAuthenticationError`**

Thrown when eauth authentication fails

exception `salt.exceptions.`**`LoaderError`**
Problems loading the right renderer

exception `salt.exceptions.`**`MasterExit`**
Rise when the master exits

exception `salt.exceptions.`**`MinionError`**
Minion problems reading uris such as salt:// or http://

exception `salt.exceptions.`**`PkgParseError`**
Used when of the pkg modules cannot correctly parse the output from the CLI tool (pacman, yum, apt, aptitude, etc)

exception `salt.exceptions.`**`SaltClientError`**
Problem reading the master root key

exception `salt.exceptions.`**`SaltException`**
Base exception class; all Salt-specific exceptions should subclass this

exception `salt.exceptions.`**`SaltInvocationError`**
Used when the wrong number of arguments are sent to modules or invalid arguments are specified on the command line

exception `salt.exceptions.`**`SaltMasterError`**
Problem reading the master root key

exception `salt.exceptions.`**`SaltRenderError`**(*error*, *line_num=None*, *buf=''*, *marker=' <======================'*, *trace=None*)
Used when a renderer needs to raise an explicit error. If a line number and buffer string are passed, get_context will be invoked to get the location of the error.

exception `salt.exceptions.`**`SaltReqTimeoutError`**
Thrown when a salt master request call fails to return within the timeout

exception `salt.exceptions.`**`SaltRunnerError`**
Problem in runner

exception `salt.exceptions.`**`SaltSystemExit`**(*code=0*, *msg=None*)
This exception is raised when an unsolvable problem is found. There's nothing else to do, salt should just exit.

exception `salt.exceptions.`**`SaltWheelError`**
Problem in wheel

exception `salt.exceptions.`**`TimedProcTimeoutError`**
Thrown when a timed subprocess does not terminate within the timeout, or if the specified timeout is not an int or a float

exception `salt.exceptions.`**`TokenAuthenticationError`**
Thrown when token authentication fails

CHAPTER 107

Network Topology

Salt is based on a powerful, asynchronous, network topology using ZeroMQ. Many ZeroMQ systems are in place to enable communication. The central idea is to have the fastest communication possible.

107.1 Servers

The Salt Master runs 2 network services. First is the ZeroMQ PUB system. This service by default runs on port `4505` and can be configured via the `publish_port` option in the master configuration.

Second is the ZeroMQ REP system. This is a separate interface used for all bi-directional communication with minions. By default this system binds to port `4506` and can be configured via the `ret_port` option in the master.

107.2 PUB/SUB

The commands sent out via the salt client are broadcast out to the minions via ZeroMQ PUB/SUB. This is done by allowing the minions to maintain a connection back to the Salt Master and then all connections are informed to download the command data at once. The command data is kept extremely small (usually less than 1K) so it is not a burden on the network.

107.3 Return

The PUB/SUB system is a one way communication, so once a publish is sent out the PUB interface on the master has no further communication with the minion. The minion, after running the command, then sends the command's return data back to the master via the `ret_port`.

CHAPTER 108

Windows Software Repository

The Salt Windows Software Repository provides a package manager and software repository similar to what is provided by yum and apt on Linux.

It permits the installation of software using the installers on remote windows machines. In many senses, the operation is similar to that of the other package managers salt is aware of:

- the `pkg.installed` and similar states work on Windows.
- the `pkg.install` and similar module functions work on Windows.
- each windows machine needs to have `pkg.refresh_db` executed against it to pick up the latest version of the package database.

High level differences to yum and apt are:

- The repository metadata (sls files) is hosted through either salt or git.
- Packages can be downloaded from within the salt repository, a git repository or from http(s) or ftp urls.
- No dependencies are managed. Dependencies between packages needs to be managed manually.

108.1 Operation

The install state/module function of the windows package manager works roughly as follows:

1. Execute `pkg.list_pkgs` and store the result
2. Check if any action needs to be taken. (ie compare required package and version against `pkg.list_pkgs` results)
3. If so, run the installer command.
4. Execute `pkg.list_pkgs` and compare to the result stored from before installation.
5. Sucess/Failure/Changes will be reported based on the differences between the original and final `pkg.list_pkgs` results.

If there are any problems in using the package manager it is likely to be due to the data in your sls files not matching the difference between the pre and post `pkg.list_pkgs` results.

108.2 Usage

By default, the Windows software repository is found at `/srv/salt/win/repo` This can be changed in the master config file (default location is `/etc/salt/master`) by modifying the `win_repo` variable. Each piece of software should have its own directory which contains the installers and a package definition file. This package definition file is a YAML file named `init.sls`.

The package definition file should look similar to this example for Firefox: `/srv/salt/win/repo/firefox/init.sls`

```
firefox:
  17.0.1:
    installer: 'salt://win/repo/firefox/English/Firefox Setup 17.0.1.exe'
    full_name: Mozilla Firefox 17.0.1 (x86 en-US)
    locale: en_US
    reboot: False
    install_flags: ' -ms'
    uninstaller: '%ProgramFiles(x86)%/Mozilla Firefox/uninstall/helper.exe'
    uninstall_flags: ' /S'
  16.0.2:
    installer: 'salt://win/repo/firefox/English/Firefox Setup 16.0.2.exe'
    full_name: Mozilla Firefox 16.0.2 (x86 en-US)
    locale: en_US
    reboot: False
    install_flags: ' -ms'
    uninstaller: '%ProgramFiles(x86)%/Mozilla Firefox/uninstall/helper.exe'
    uninstall_flags: ' /S'
  15.0.1:
    installer: 'salt://win/repo/firefox/English/Firefox Setup 15.0.1.exe'
    full_name: Mozilla Firefox 15.0.1 (x86 en-US)
    locale: en_US
    reboot: False
    install_flags: ' -ms'
    uninstaller: '%ProgramFiles(x86)%/Mozilla Firefox/uninstall/helper.exe'
    uninstall_flags: ' /S'
```

More examples can be found here: https://github.com/saltstack/salt-winrepo

The version number and `full_name` need to match the output from `pkg.list_pkgs` so that the status can be verfied when running highstate. Note: It is still possible to successfully install packages using `pkg.install` even if they don't match which can make this hard to troubleshoot.

```
salt 'test-2008' pkg.list_pkgs
test-2008
    ----------
    7-Zip 9.20 (x64 edition):
        9.20.00.0
    Microsoft .NET Framework 4 Client Profile:
        4.0.30319,4.0.30319
    Microsoft .NET Framework 4 Extended:
        4.0.30319,4.0.30319
    Microsoft Visual C++ 2008 Redistributable - x64 9.0.21022:
        9.0.21022
    Mozilla Firefox 17.0.1 (x86 en-US):
        17.0.1
    Mozilla Maintenance Service:
        17.0.1
    NSClient++ (x64):
        0.3.8.76
    Notepad++:
        6.4.2
    Salt Minion 0.16.0:
        0.16.0
```

If any of these preinstalled packages already exist in winrepo the full_name will be automatically renamed to their package name during the next update (running highstate or installing another package).

```
test-2008:
    ----------
    7zip:
        9.20.00.0
    Microsoft .NET Framework 4 Client Profile:
        4.0.30319,4.0.30319
    Microsoft .NET Framework 4 Extended:
        4.0.30319,4.0.30319
    Microsoft Visual C++ 2008 Redistributable - x64 9.0.21022:
        9.0.21022
    Mozilla Maintenance Service:
        17.0.1
    Notepad++:
        6.4.2
    Salt Minion 0.16.0:
        0.16.0
    firefox:
        17.0.1
    nsclient:
        0.3.9.328
```

Add `msiexec: True` if using an MSI installer requiring the use of `msiexec /i` to install and `msiexec /x` to uninstall.

The `install_flags` and `uninstall_flags` are flags passed to the software installer to cause it to perform a silent install. These can often be found by adding `/?` or `/h` when running the installer from the command line. A great resource for finding these silent install flags can be found on the WPKG project's wiki:

```
7zip:
  9.20.00.0:
    installer: salt://win/repo/7zip/7z920-x64.msi
    full_name: 7-Zip 9.20 (x64 edition)
    reboot: False
    install_flags: ' /q '
    msiexec: True
    uninstaller: salt://win/repo/7zip/7z920-x64.msi
    uninstall_flags: ' /qn'
```

108.3 Generate Repo Cache File

Once the sls file has been created, generate the repository cache file with the winrepo runner:

```
salt-run winrepo.genrepo
```

Then update the repository cache file on your minions, exactly how it's done for the Linux package managers:

```
salt '*' pkg.refresh_db
```

108.4 Install Windows Software

Now you can query the available version of Firefox using the Salt pkg module.

```
salt '*' pkg.available_version firefox
```

```
{'davewindows': {'15.0.1': 'Mozilla Firefox 15.0.1 (x86 en-US)',
                 '16.0.2': 'Mozilla Firefox 16.0.2 (x86 en-US)',
                 '17.0.1': 'Mozilla Firefox 17.0.1 (x86 en-US)'}}
```

As you can see, there are three versions of Firefox available for installation.

```
salt '*' pkg.install firefox
```

The above line will install the latest version of Firefox.

```
salt '*' pkg.install firefox version=16.0.2
```

The above line will install version 16.0.2 of Firefox.

If a different version of the package is already installed it will be replaced with the version in winrepo (only if the package itself supports live updating)

108.5 Uninstall Windows Software

Uninstall software using the pkg module:

```
salt '*' pkg.remove firefox
```

```
salt '*' pkg.purge firefox
```

`pkg.purge` just executes `pkg.remove` on Windows. At some point in the future `pkg.purge` may direct the installer to remove all configs and settings for software packages that support that option.

108.6 Standalone Minion Salt Windows Repo Module

In order to facilitate managing a Salt Windows software repo with Salt on a Standalone Minion on Windows, a new module named winrepo has been added to Salt. winrepo matches what is available in the salt runner and allows you to manage the Windows software repo contents. Example: `salt '*' winrepo.genrepo`

108.7 Git Hosted Repo

Windows software package definitions can also be hosted in one or more git repositories. The default repo is one hosted on Github.com by SaltStack,Inc., which includes package definitions for open source software. This repo points to the HTTP or ftp locations of the installer files. Anyone is welcome to send a pull request to this repo to add new package definitions. Browse the repo here: https://github.com/saltstack/salt-winrepo .

Configure which git repos the master can search for package definitions by modifying or extending the `win_gitrepos` configuration option list in the master config.

Checkout each git repo in `win_gitrepos`, compile your package repository cache and then refresh each minion's package cache:

```
salt-run winrepo.update_git_repos
salt-run winrepo.genrepo
salt '*' pkg.refresh_db
```

108.8 Troubleshooting

108.8.1 Incorrect name/version

If the package seems to install properly, but salt reports a failure then it is likely you have a version or `full_name` mismatch.

Check the exact `full_name` and version used by the package. Use `pkg.list_pkgs` to check that the names and version exactly match what is installed.

108.8.2 Changes to sls files not being picked up

Ensure you have (re)generated the repository cache file and then updated the repository cache on the relevant minions:

```
salt-run winrepo.genrepo
salt 'MINION' pkg.refresh_db
```

108.8.3 Packages management under Windows 2003

On windows server 2003, you need to install optional windows component "wmi windows installer provider" to have full list of installed packages. If you don't have this, salt-minion can't report some installed software.

CHAPTER 109

Command Line Reference

Salt can be controlled by a command line client by the root user on the Salt master. The Salt command line client uses the Salt client API to communicate with the Salt master server. The Salt client is straightforward and simple to use.

Using the Salt client commands can be easily sent to the minions.

Each of these commands accepts an explicit *–config* option to point to either the master or minion configuration file. If this option is not provided and the default configuration file does not exist then Salt falls back to use the environment variables `SALT_MASTER_CONFIG` and `SALT_MINION_CONFIG`.

See also:

Configuring Salt

109.1 Using the Salt Command

The Salt command needs a few components to send information to the Salt minions. The target minions need to be defined, the function to call and any arguments the function requires.

109.1.1 Defining the Target Minions

The first argument passed to salt, defines the target minions, the target minions are accessed via their hostname. The default target type is a bash glob:

```
salt '*foo.com' sys.doc
```

Salt can also define the target minions with regular expressions:

```
salt -E '.*' cmd.run 'ls -l | grep foo'
```

Or to explicitly list hosts, salt can take a list:

```
salt -L foo.bar.baz,quo.qux cmd.run 'ps aux | grep foo'
```

109.1.2 More Powerful Targets

The simple target specifications, glob, regex and list will cover many use cases, and for some will cover all use cases, but more powerful options exist.

Targeting with Grains

The Grains interface was built into Salt to allow minions to be targeted by system properties. So minions running on a particular operating system can be called to execute a function, or a specific kernel.

Calling via a grain is done by passing the -G option to salt, specifying a grain and a glob expression to match the value of the grain. The syntax for the target is the grain key followed by a globexpression: "os:Arch*".

```
salt -G 'os:Fedora' test.ping
```

Will return True from all of the minions running Fedora.

To discover what grains are available and what the values are, execute the grains.item salt function:

```
salt '*' grains.items
```

Targeting with Executions

As of 0.8.8 targeting with executions is still under heavy development and this documentation is written to reference the behavior of execution matching in the future.

Execution matching allows for a primary function to be executed, and then based on the return of the primary function the main function is executed.

Execution matching allows for matching minions based on any arbitrary running data on the minions.

Compound Targeting

New in version 0.9.5.

Multiple target interfaces can be used in conjunction to determine the command targets. These targets can then be combined using and or or statements. This is well defined with an example:

```
salt -C 'G@os:Debian and webser* or E@db.*' test.ping
```

In this example any minion who's id starts with `webser` and is running Debian, or any minion who's id starts with db will be matched.

The type of matcher defaults to glob, but can be specified with the corresponding letter followed by the `@` symbol. In the above example a grain is used with `G@` as well as a regular expression with `E@`. The `webser*` target does not need to be prefaced with a target type specifier because it is a glob.

Node Group Targeting

New in version 0.9.5.

Often the convenience of having a predefined group of minions to execute targets on is desired. This can be accomplished with the new nodegroups feature. Nodegroups allow for predefined compound targets to be declared in the master configuration file:

```
nodegroups:
  group1: 'L@foo.domain.com,bar.domain.com,baz.domain.com and bl*.domain.com'
  group2: 'G@os:Debian and foo.domain.com'
```

109.1.3 Calling the Function

The function to call on the specified target is placed after the target specification.

New in version 0.9.8.

Functions may also accept arguments, space-delimited:

```
salt '*' cmd.exec_code python 'import sys; print sys.version'
```

Optional, keyword arguments are also supported:

```
salt '*' pip.install salt timeout=5 upgrade=True
```

They are always in the form of `kwarg=argument`.

Arguments are formatted as YAML:

```
salt '*' cmd.run 'echo "Hello: $FIRST_NAME"' env='{FIRST_NAME: "Joe"}'
```

Note: dictionaries must have curly braces around them (like the `env` keyword argument above). This was changed in 0.15.1: in the above example, the first argument used to be parsed as the dictionary `{'echo "Hello': '$FIRST_NAME"'}`. This was generally not the expected behavior.

If you want to test what parameters are actually passed to a module, use the `test.arg_repr` command:

```
salt '*' test.arg_repr 'echo "Hello: $FIRST_NAME"' env='{FIRST_NAME: "Joe"}'
```

Finding available minion functions

The Salt functions are self documenting, all of the function documentation can be retried from the minions via the `sys.doc()` function:

```
salt '*' sys.doc
```

109.1.4 Compound Command Execution

If a series of commands needs to be sent to a single target specification then the commands can be sent in a single publish. This can make gathering groups of information faster, and lowers the stress on the network for repeated commands.

Compound command execution works by sending a list of functions and arguments instead of sending a single function and argument. The functions are executed on the minion in the order they are defined on the command line, and then the data from all of the commands are returned in a dictionary. This means that the set of commands are called in a predictable way, and the returned data can be easily interpreted.

Executing compound commands if done by passing a comma delimited list of functions, followed by a comma delimited list of arguments:

```
salt '*' cmd.run,test.ping,test.echo 'cat /proc/cpuinfo',,foo
```

The trick to look out for here, is that if a function is being passed no arguments, then there needs to be a placeholder for the absent arguments. This is why in the above example, there are two commas right next to each other. `test.ping` takes no arguments, so we need to add another comma, otherwise Salt would attempt to pass "foo" to `test.ping`.

If you need to pass arguments that include commas, then make sure you add spaces around the commas that separate arguments. For example:

```
salt '*' cmd.run,test.ping,test.echo 'echo "1,2,3"' , , foo
```

You may change the arguments separator using the `--args-separator` option:

```
salt --args-separator=:: '*' some.fun,test.echo params with , comma :: foo
```

salt

110.1 Synopsis

salt '*' [options] sys.doc

salt -E '.*' [options] sys.doc cmd

salt -G 'os:Arch.*' [options] test.ping

salt -C 'G@os:Arch.* and webserv* or G@kernel:FreeBSD' [options] test.ping

110.2 Description

Salt allows for commands to be executed across a swath of remote systems in parallel. This means that remote systems can be both controlled and queried with ease.

110.3 Options

`--version`
Print the version of Salt that is running.

`--versions-report`
Show program's dependencies and version number, and then exit

`-h, --help`
Show the help message and exit

`-c` `CONFIG_DIR`, **`--config-dir`**`=CONFIG_dir`
The location of the Salt configuration directory. This directory contains the configuration files for Salt master and minions. The default location on most systems is `/etc/salt`.

`-t` `TIMEOUT`, **`--timeout`**`=TIMEOUT`
The timeout in seconds to wait for replies from the Salt minions. The timeout number specifies how long the command line client will wait to query the minions and check on running jobs. Default: 5

`-s, --static`
By default as of version 0.9.8 the salt command returns data to the console as it is received from minions, but previous releases would return data only after all data was received. To only return the data with a hard timeout and after all minions have returned then use the static option.

--async
Instead of waiting for the job to run on minions only print the jod id of the started execution and complete.

--state-output=STATE_OUTPUT
New in version 0.17.

Override the configured state_output value for minion output. Default: full

--subset=SUBSET
Execute the routine on a random subset of the targeted minions. The minions will be verified that they have the named function before executing.

-v VERBOSE**, --verbose**
Turn on verbosity for the salt call, this will cause the salt command to print out extra data like the job id.

-b BATCH**, --batch-size**=BATCH
Instead of executing on all targeted minions at once, execute on a progressive set of minions. This option takes an argument in the form of an explicit number of minions to execute at once, or a percentage of minions to execute on.

-a EAUTH**, --auth**=EAUTH
Pass in an external authentication medium to validate against. The credentials will be prompted for. Can be used with the -T option.

-T, --make-token
Used in conjunction with the -a option. This creates a token that allows for the authenticated user to send commands without needing to re-authenticate.

--return=RETURNER
Chose an alternative returner to call on the minion, if an alternative returner is used then the return will not come back to the command line but will be sent to the specified return system.

-d, --doc, --documentation
Return the documentation for the module functions available on the minions

--args-separator=ARGS_SEPARATOR
Set the special argument used as a delimiter between command arguments of compound commands. This is useful when one wants to pass commas as arguments to some of the commands in a compound command.

110.3.1 Logging Options

Logging options which override any settings defined on the configuration files.

-l LOG_LEVEL**, --log-level**=LOG_LEVEL
Console logging log level. One of `all`, `garbage`, `trace`, `debug`, `info`, `warning`, `error`, `quiet`. Default: `warning`.

--log-file=LOG_FILE
Log file path. Default: /var/log/salt/master.

--log-file-level=LOG_LEVEL_LOGFILE
Logfile logging log level. One of `all`, `garbage`, `trace`, `debug`, `info`, `warning`, `error`, `quiet`. Default: `warning`.

110.3.2 Target Selection

-E, --pcre
The target expression will be interpreted as a PCRE regular expression rather than a shell glob.

-L, --list
The target expression will be interpreted as a comma-delimited list; example: server1.foo.bar,server2.foo.bar,example7.quo.qux

-G, --grain
The target expression matches values returned by the Salt grains system on the minions. The target expression is in the format of '<grain value>:<glob expression>'; example: 'os:Arch*'

This was changed in version 0.9.8 to accept glob expressions instead of regular expression. To use regular expression matching with grains, use the –grain-pcre option.

--grain-pcre
The target expression matches values returned by the Salt grains system on the minions. The target expression is in the format of '<grain value>:< regular expression>'; example: 'os:Arch.*'

-N, --nodegroup
Use a predefined compound target defined in the Salt master configuration file.

-R, --range
Instead of using shell globs to evaluate the target, use a range expression to identify targets. Range expressions look like %cluster.

Using the Range option requires that a range server is set up and the location of the range server is referenced in the master configuration file.

-C, --compound
Utilize many target definitions to make the call very granular. This option takes a group of targets separated by `and` or `or`. The default matcher is a glob as usual. If something other than a glob is used, preface it with the letter denoting the type; example: 'webserv* and G@os:Debian or E@db*' Make sure that the compound target is encapsulated in quotes.

-X, --exsel
Instead of using shell globs, use the return code of a function.

-I, --pillar
Instead of using shell globs to evaluate the target, use a pillar value to identify targets. The syntax for the target is the pillar key followed by a glob expression: "role:production*"

-S, --ipcidr
Match based on Subnet (CIDR notation) or IPv4 address.

110.3.3 Output Options

--out
Pass in an alternative outputter to display the return of data. This outputter can be any of the available outputters:

> `grains`, `highstate`, `json`, `key`, `overstatestage`, `pprint`, `raw`, `txt`, `yaml`

Some outputters are formatted only for data returned from specific functions; for instance, the `grains` outputter will not work for non-grains data.

If an outputter is used that does not support the data passed into it, then Salt will fall back on the `pprint` outputter and display the return data using the Python `pprint` standard library module.

Note: If using `--out=json`, you will probably want `--static` as well. Without the static option, you will get a JSON string for each minion. This is due to using an iterative outputter. So if you want to feed it to a JSON parser, use `--static` as well.

--out-indent OUTPUT_INDENT, **--output-indent** OUTPUT_INDENT
Print the output indented by the provided value in spaces. Negative values disable indentation. Only applicable in outputters that support indentation.

--out-file=OUTPUT_FILE, **--output-file**=OUTPUT_FILE
Write the output to the specified file.

--no-color
Disable all colored output

--force-color
Force colored output

110.4 See also

salt(7) salt-master(1) salt-minion(1)

CHAPTER 111

`salt-master`

The Salt master daemon, used to control the Salt minions

111.1 Synopsis

salt-master [options]

111.2 Description

The master daemon controls the Salt minions

111.3 Options

`--version`
Print the version of Salt that is running.

`--versions-report`
Show program's dependencies and version number, and then exit

`-h, --help`
Show the help message and exit

`-c` `CONFIG_DIR`, **`--config-dir`**`=CONFIG_dir`
The location of the Salt configuration directory. This directory contains the configuration files for Salt master and minions. The default location on most systems is `/etc/salt`.

`-u` `USER`, **`--user`**`=USER`
Specify user to run salt-master

`-d, --daemon`
Run salt-master as a daemon

`--pid-file` `PIDFILE`
Specify the location of the pidfile. Default: /var/run/salt-master.pid

111.3.1 Logging Options

Logging options which override any settings defined on the configuration files.

`-l` `LOG_LEVEL`, **`--log-level`**`=LOG_LEVEL`
Console logging log level. One of `all`, `garbage`, `trace`, `debug`, `info`, `warning`, `error`, `quiet`. Default: `warning`.

`--log-file``=LOG_FILE`
Log file path. Default: /var/log/salt/master.

`--log-file-level``=LOG_LEVEL_LOGFILE`
Logfile logging log level. One of `all`, `garbage`, `trace`, `debug`, `info`, `warning`, `error`, `quiet`. Default: `warning`.

111.4 See also

`salt(1)` *`salt(7)`* *`salt-minion(1)`*

CHAPTER 112

salt-minion

The Salt minion daemon, receives commands from a remote Salt master.

112.1 Synopsis

salt-minion [options]

112.2 Description

The Salt minion receives commands from the central Salt master and replies with the results of said commands.

112.3 Options

--version
Print the version of Salt that is running.

--versions-report
Show program's dependencies and version number, and then exit

-h, --help
Show the help message and exit

-c `CONFIG_DIR`**, --config-dir**=`CONFIG_dir`
The location of the Salt configuration directory. This directory contains the configuration files for Salt master and minions. The default location on most systems is `/etc/salt`.

-u `USER`**, --user**=`USER`
Specify user to run salt-minion

-d, --daemon
Run salt-minion as a daemon

--pid-file `PIDFILE`
Specify the location of the pidfile. Default: /var/run/salt-minion.pid

112.3.1 Logging Options

Logging options which override any settings defined on the configuration files.

-l `LOG_LEVEL`**, --log-level**=`LOG_LEVEL`
Console logging log level. One of `all`, `garbage`, `trace`, `debug`, `info`, `warning`, `error`, `quiet`. Default: `warning`.

--log-file=`LOG_FILE`
Log file path. Default: /var/log/salt/minion.

--log-file-level=`LOG_LEVEL_LOGFILE`
Logfile logging log level. One of `all`, `garbage`, `trace`, `debug`, `info`, `warning`, `error`, `quiet`. Default: `warning`.

112.4 See also

`salt(1)` *`salt(7)`* *`salt-master(1)`*

CHAPTER 113

salt-key

113.1 Synopsis

salt-key [options]

113.2 Description

Salt-key executes simple management of Salt server public keys used for authentication.

113.3 Options

`--version`
Print the version of Salt that is running.

`--versions-report`
Show program's dependencies and version number, and then exit

`-h, --help`
Show the help message and exit

`-c` `CONFIG_DIR`**`, --config-dir`**`=CONFIG_dir`
The location of the Salt configuration directory. This directory contains the configuration files for Salt master and minions. The default location on most systems is `/etc/salt`.

`-q, --quiet`
Suppress output

`-y, --yes`
Answer 'Yes' to all questions presented, defaults to False

113.3.1 Logging Options

Logging options which override any settings defined on the configuration files.

`--log-file``=LOG_FILE`
Log file path. Default: /var/log/salt/minion.

--log-file-level=`LOG_LEVEL_LOGFILE`
Logfile logging log level. One of `all`, `garbage`, `trace`, `debug`, `info`, `warning`, `error`, `quiet`. Default: `warning`.

113.3.2 Output Options

--out
Pass in an alternative outputter to display the return of data. This outputter can be any of the available outputters:

> `grains`, `highstate`, `json`, `key`, `overstatestage`, `pprint`, `raw`, `txt`, `yaml`

Some outputters are formatted only for data returned from specific functions; for instance, the `grains` outputter will not work for non-grains data.

If an outputter is used that does not support the data passed into it, then Salt will fall back on the `pprint` outputter and display the return data using the Python `pprint` standard library module.

Note: If using `--out=json`, you will probably want `--static` as well. Without the static option, you will get a JSON string for each minion. This is due to using an iterative outputter. So if you want to feed it to a JSON parser, use `--static` as well.

--out-indent `OUTPUT_INDENT`, **--output-indent** `OUTPUT_INDENT`
Print the output indented by the provided value in spaces. Negative values disable indentation. Only applicable in outputters that support indentation.

--out-file=`OUTPUT_FILE`, **--output-file**=`OUTPUT_FILE`
Write the output to the specified file.

--no-color
Disable all colored output

--force-color
Force colored output

113.3.3 Actions

-l `ARG`, **--list**=`ARG`
List the public keys. The args `pre`, `un`, and `unaccepted` will list unaccepted/unsigned keys. `acc` or `accepted` will list accepted/signed keys. `rej` or `rejected` will list rejected keys. Finally, `all` will list all keys.

-L, --list-all
List all public keys. (Deprecated: use `--list all`)

-a `ACCEPT`, **--accept**=`ACCEPT`
Accept the specified public key (use –include-all to match rejected keys in addition to pending keys). Globs are supported.

-A, --accept-all
Accepts all pending keys.

-r `REJECT`, **--reject**=`REJECT`
Reject the specified public key (use –include-all to match accepted keys in addition to pending keys). Globs are supported.

-R, --reject-all
Rejects all pending keys.

--include-all
Include non-pending keys when accepting/rejecting.

-p PRINT, **--print**=PRINT
Print the specified public key.

-P, --print-all
Print all public keys

-d DELETE, **--delete**=DELETE
Delete the specified key. Globs are supported.

-D, --delete-all
Delete all keys.

-f FINGER, **--finger**=FINGER
Print the specified key's fingerprint.

-F, --finger-all
Print all keys' fingerprints.

113.3.4 Key Generation Options

--gen-keys=GEN_KEYS
Set a name to generate a keypair for use with salt

--gen-keys-dir=GEN_KEYS_DIR
Set the directory to save the generated keypair. Only works with 'gen_keys_dir' option; default is the current directory.

--keysize=KEYSIZE
Set the keysize for the generated key, only works with the '–gen-keys' option, the key size must be 2048 or higher, otherwise it will be rounded up to 2048. The default is 2048.

113.4 See also

salt(7) salt-master(1) salt-minion(1)

CHAPTER 114

salt-cp

Copy a file to a set of systems

114.1 Synopsis

```
salt-cp '*' [ options ] SOURCE DEST
```

```
salt-cp -E '.*' [ options ] SOURCE DEST
```

```
salt-cp -G 'os:Arch.*' [ options ] SOURCE DEST
```

114.2 Description

Salt copy copies a local file out to all of the Salt minions matched by the given target.

114.3 Options

`--version`
Print the version of Salt that is running.

`--versions-report`
Show program's dependencies and version number, and then exit

`-h, --help`
Show the help message and exit

`-c` `CONFIG_DIR`, **`--config-dir`**`=CONFIG_dir`
The location of the Salt configuration directory. This directory contains the configuration files for Salt master and minions. The default location on most systems is `/etc/salt`.

`-t` `TIMEOUT`, **`--timeout`**`=TIMEOUT`
The timeout in seconds to wait for replies from the Salt minions. The timeout number specifies how long the command line client will wait to query the minions and check on running jobs. Default: 5

114.3.1 Logging Options

Logging options which override any settings defined on the configuration files.

-l `LOG_LEVEL`**, --log-level**=`LOG_LEVEL`
Console logging log level. One of `all`, `garbage`, `trace`, `debug`, `info`, `warning`, `error`, `quiet`. Default: `warning`.

--log-file=`LOG_FILE`
Log file path. Default: /var/log/salt/master.

--log-file-level=`LOG_LEVEL_LOGFILE`
Logfile logging log level. One of `all`, `garbage`, `trace`, `debug`, `info`, `warning`, `error`, `quiet`. Default: `warning`.

114.3.2 Target Selection

-E, --pcre
The target expression will be interpreted as a PCRE regular expression rather than a shell glob.

-L, --list
The target expression will be interpreted as a comma-delimited list; example: server1.foo.bar,server2.foo.bar,example7.quo.qux

-G, --grain
The target expression matches values returned by the Salt grains system on the minions. The target expression is in the format of '<grain value>:<glob expression>'; example: 'os:Arch*'

This was changed in version 0.9.8 to accept glob expressions instead of regular expression. To use regular expression matching with grains, use the –grain-pcre option.

--grain-pcre
The target expression matches values returned by the Salt grains system on the minions. The target expression is in the format of '<grain value>:< regular expression>'; example: 'os:Arch.*'

-N, --nodegroup
Use a predefined compound target defined in the Salt master configuration file.

-R, --range
Instead of using shell globs to evaluate the target, use a range expression to identify targets. Range expressions look like %cluster.

Using the Range option requires that a range server is set up and the location of the range server is referenced in the master configuration file.

114.4 See also

salt(1) salt-master(1) salt-minion(1)

CHAPTER 115

salt-call

115.1 Synopsis

```
salt-call [options]
```

115.2 Description

The salt-call command is used to run module functions locally on a minion instead of executing them from the master.

115.3 Options

--version
Print the version of Salt that is running.

--versions-report
Show program's dependencies and version number, and then exit

-h, --help
Show the help message and exit

-c `CONFIG_DIR`**, --config-dir**=`CONFIG_dir`
The location of the Salt configuration directory. This directory contains the configuration files for Salt master and minions. The default location on most systems is `/etc/salt`.

-g, --grains
Return the information generated by the Salt grains

-m `MODULE_DIRS`**, --module-dirs**=`MODULE_DIRS`
Specify an additional directories to pull modules from, multiple directories can be delimited by commas

-d, --doc, --documentation
Return the documentation for the specified module or for all modules if none are specified

--master=`MASTER`
Specify the master to use. The minion must be authenticated with the master. If this option is omitted, the master options from the minion config will be used. If multi masters are set up the first listed master that responds will be used.

`--return` `RETURNER`
Set salt-call to pass the return data to one or many returner interfaces. To use many returner interfaces specify a comma delimited list of returners.

`--local`
Run salt-call locally, as if there was no master running.

115.3.1 Logging Options

Logging options which override any settings defined on the configuration files.

`-l` `LOG_LEVEL`, **`--log-level`**`=LOG_LEVEL`
Console logging log level. One of `all`, `garbage`, `trace`, `debug`, `info`, `warning`, `error`, `quiet`. Default: `info`.

`--log-file``=LOG_FILE`
Log file path. Default: /var/log/salt/minion.

`--log-file-level``=LOG_LEVEL_LOGFILE`
Logfile logging log level. One of `all`, `garbage`, `trace`, `debug`, `info`, `warning`, `error`, `quiet`. Default: `info`.

115.3.2 Output Options

`--out`
Pass in an alternative outputter to display the return of data. This outputter can be any of the available outputters:

> `grains`, `highstate`, `json`, `key`, `overstatestage`, `pprint`, `raw`, `txt`, `yaml`

Some outputters are formatted only for data returned from specific functions; for instance, the `grains` outputter will not work for non-grains data.

If an outputter is used that does not support the data passed into it, then Salt will fall back on the `pprint` outputter and display the return data using the Python `pprint` standard library module.

Note: If using `--out=json`, you will probably want `--static` as well. Without the static option, you will get a JSON string for each minion. This is due to using an iterative outputter. So if you want to feed it to a JSON parser, use `--static` as well.

`--out-indent` `OUTPUT_INDENT`, **`--output-indent`** `OUTPUT_INDENT`
Print the output indented by the provided value in spaces. Negative values disable indentation. Only applicable in outputters that support indentation.

`--out-file``=OUTPUT_FILE`, **`--output-file`**`=OUTPUT_FILE`
Write the output to the specified file.

`--no-color`
Disable all colored output

`--force-color`
Force colored output

115.4 See also

`salt(1)` *`salt-master(1)`* *`salt-minion(1)`*

CHAPTER 116

`salt-run`

Execute a Salt runner

116.1 Synopsis

```
salt-run RUNNER
```

116.2 Description

salt-run is the frontend command for executing `Salt Runners`. Salt runners are simple modules used to execute convenience functions on the master

116.3 Options

`--version`
Print the version of Salt that is running.

`--versions-report`
Show program's dependencies and version number, and then exit

`-h, --help`
Show the help message and exit

`-c` `CONFIG_DIR`, **`--config-dir`**`=CONFIG_dir`
The location of the Salt configuration directory. This directory contains the configuration files for Salt master and minions. The default location on most systems is `/etc/salt`.

`-t` `TIMEOUT`, **`--timeout`**`=TIMEOUT`
The timeout in seconds to wait for replies from the Salt minions. The timeout number specifies how long the command line client will wait to query the minions and check on running jobs. Default: 1

`-d, --doc, --documentation`
Display documentation for runners, pass a module or a runner to see documentation on only that module/runner.

116.3.1 Logging Options

Logging options which override any settings defined on the configuration files.

-l `LOG_LEVEL`, **--log-level**=`LOG_LEVEL`
Console logging log level. One of `all`, `garbage`, `trace`, `debug`, `info`, `warning`, `error`, `quiet`. Default: `warning`.

--log-file=`LOG_FILE`
Log file path. Default: /var/log/salt/master.

--log-file-level=`LOG_LEVEL_LOGFILE`
Logfile logging log level. One of `all`, `garbage`, `trace`, `debug`, `info`, `warning`, `error`, `quiet`. Default: `warning`.

116.4 See also

salt(1) *salt-master(1)* *salt-minion(1)*

CHAPTER 117

salt-ssh

117.1 Synopsis

salt-ssh '*' [options] sys.doc

salt-ssh -E '.*' [options] sys.doc cmd

117.2 Description

Salt SSH allows for salt routines to be executed using only SSH for transport

117.3 Options

`-r, --raw, --raw-shell`
Execute a raw shell command.

`--roster-file`
Define which roster system to use, this defines if a database backend, scanner, or custom roster system is used. Default is the flat file roster.

`--refresh, --refresh-cache`
Force a refresh of the master side data cache of the target's data. This is needed if a target's grains have been changed and the auto refresh timeframe has not been reached.

`--max-procs`
Set the number of concurrent minions to communicate with. This value defines how many processes are opened up at a time to manage connections, the more running process the faster communication should be, default is 25.

`--passwd`
Set the default password to attempt to use when authenticating.

`--key-deploy`
Set this flag to attempt to deploy the authorized ssh key with all minions. This combined with –passwd can make initial deployment of keys very fast and easy.

`--version`
Print the version of Salt that is running.

--versions-report
Show program's dependencies and version number, and then exit

-h, --help
Show the help message and exit

-c CONFIG_DIR, **--config-dir**=CONFIG_dir
The location of the Salt configuration directory. This directory contains the configuration files for Salt master and minions. The default location on most systems is `/etc/salt`.

117.3.1 Target Selection

-E, --pcre
The target expression will be interpreted as a PCRE regular expression rather than a shell glob.

-L, --list
The target expression will be interpreted as a comma-delimited list; example: server1.foo.bar,server2.foo.bar,example7.quo.qux

-G, --grain
The target expression matches values returned by the Salt grains system on the minions. The target expression is in the format of '<grain value>:<glob expression>'; example: 'os:Arch*'

This was changed in version 0.9.8 to accept glob expressions instead of regular expression. To use regular expression matching with grains, use the –grain-pcre option.

--grain-pcre
The target expression matches values returned by the Salt grains system on the minions. The target expression is in the format of '<grain value>:< regular expression>'; example: 'os:Arch.*'

-N, --nodegroup
Use a predefined compound target defined in the Salt master configuration file.

-R, --range
Instead of using shell globs to evaluate the target, use a range expression to identify targets. Range expressions look like %cluster.

Using the Range option requires that a range server is set up and the location of the range server is referenced in the master configuration file.

117.3.2 Logging Options

Logging options which override any settings defined on the configuration files.

-l LOG_LEVEL, **--log-level**=LOG_LEVEL
Console logging log level. One of `all`, `garbage`, `trace`, `debug`, `info`, `warning`, `error`, `quiet`. Default: `warning`.

--log-file=LOG_FILE
Log file path. Default: /var/log/salt/ssh.

--log-file-level=LOG_LEVEL_LOGFILE
Logfile logging log level. One of `all`, `garbage`, `trace`, `debug`, `info`, `warning`, `error`, `quiet`. Default: `warning`.

117.3.3 Output Options

`--out`

Pass in an alternative outputter to display the return of data. This outputter can be any of the available outputters:

> `grains`, `highstate`, `json`, `key`, `overstatestage`, `pprint`, `raw`, `txt`, `yaml`

Some outputters are formatted only for data returned from specific functions; for instance, the `grains` outputter will not work for non-grains data.

If an outputter is used that does not support the data passed into it, then Salt will fall back on the `pprint` outputter and display the return data using the Python `pprint` standard library module.

Note: If using `--out=json`, you will probably want `--static` as well. Without the static option, you will get a JSON string for each minion. This is due to using an iterative outputter. So if you want to feed it to a JSON parser, use `--static` as well.

`--out-indent` `OUTPUT_INDENT`, **`--output-indent`** `OUTPUT_INDENT`

Print the output indented by the provided value in spaces. Negative values disable indentation. Only applicable in outputters that support indentation.

`--out-file``=OUTPUT_FILE`, **`--output-file`**`=OUTPUT_FILE`

Write the output to the specified file.

`--no-color`

Disable all colored output

`--force-color`

Force colored output

117.4 See also

`salt(7)` *`salt-master(1)`* *`salt-minion(1)`*

CHAPTER 118

`salt-syndic`

The Salt syndic daemon, a special minion that passes through commands from a higher master

118.1 Synopsis

salt-syndic [options]

118.2 Description

The Salt syndic daemon, a special minion that passes through commands from a higher master.

118.3 Options

`--version`
Print the version of Salt that is running.

`--versions-report`
Show program's dependencies and version number, and then exit

`-h, --help`
Show the help message and exit

`-c` `CONFIG_DIR`, **`--config-dir`**`=CONFIG_dir`
The location of the Salt configuration directory. This directory contains the configuration files for Salt master and minions. The default location on most systems is `/etc/salt`.

`-u` `USER`, **`--user`**`=USER`
Specify user to run salt-syndic

`-d, --daemon`
Run salt-syndic as a daemon

`--pid-file` `PIDFILE`
Specify the location of the pidfile. Default: /var/run/salt-syndic.pid

118.3.1 Logging Options

Logging options which override any settings defined on the configuration files.

`-l` `LOG_LEVEL`, **`--log-level`**`=LOG_LEVEL`
Console logging log level. One of `all`, `garbage`, `trace`, `debug`, `info`, `warning`, `error`, `quiet`. Default: `warning`.

`--log-file``=LOG_FILE`
Log file path. Default: /var/log/salt/master.

`--log-file-level``=LOG_LEVEL_LOGFILE`
Logfile logging log level. One of `all`, `garbage`, `trace`, `debug`, `info`, `warning`, `error`, `quiet`. Default: `warning`.

118.4 See also

`salt(1)` *`salt-master(1)`* *`salt-minion(1)`*

salt-cloud

Provision virtual machines in the cloud with Salt

119.1 Synopsis

```
salt-cloud -m /etc/salt/cloud.map

salt-cloud -p PROFILE NAME

salt-cloud -p PROFILE NAME1 NAME2 NAME3 NAME4 NAME5 NAME6
```

119.2 Description

Salt Cloud is the system used to provision virtual machines on various public clouds via a cleanly controlled profile and mapping system.

119.3 Options

`-h, --help`
Print a usage message briefly summarizing these command-line options.

`-p` `PROFILE`, **`--profile`**`=PROFILE`
Select a single profile to build the named cloud VMs from. The profile must be defined in the specified profiles file.

`-m` `MAP`, **`--map`**`=MAP`
Specify a map file to use. If used without any other options, this option will ensure that all of the mapped VMs are created. If the named VM already exists then it will be skipped.

`-H, --hard`
When specifying a map file, the default behavior is to ensure that all of the VMs specified in the map file are created. If the –hard option is set, then any VMs that exist on configured cloud providers that are not specified in the map file will be destroyed. Be advised that this can be a destructive operation and should be used with care.

`-d, --destroy`
Pass in the name(s) of VMs to destroy, salt-cloud will search the configured cloud providers for the specified

names and destroy the VMs. Be advised that this is a destructive operation and should be used with care. Can be used in conjunction with the -m option to specify a map of VMs to be deleted.

`-P, --parallel`
Normally when building many cloud VMs they are executed serially. The -P option will run each cloud vm build in a separate process allowing for large groups of VMs to be build at once.

Be advised that some cloud provider's systems don't seem to be well suited for this influx of vm creation. When creating large groups of VMs watch the cloud provider carefully.

`-Q, --query`
Execute a query and print out information about all cloud VMs. Can be used in conjunction with -m to display only information about the specified map.

`-F, --full-query`
Execute a query and print out all available information about all cloud VMs. Can be used in conjunction with -m to display only information about the specified map.

`-S, --select-query`
Execute a query and print out selected information about all cloud VMs. Can be used in conjunction with -m to display only information about the specified map.

`--list-images`
Display a list of images available in configured cloud providers. Pass the cloud provider that available images are desired on, aka "linode", or pass "all" to list images for all configured cloud providers.

`--list-sizes`
Display a list of sizes available in configured cloud providers. Pass the cloud provider that available sizes are desired on, aka "aws", or pass "all" to list sizes for all configured cloud providers

`-C` `CLOUD_CONFIG`, **`--cloud-config`**`=CLOUD_CONFIG`
Specify an alternative location for the salt cloud configuration file. Default location is /etc/salt/cloud.

`-M` `MASTER_CONFIG`, **`--master-config`**`=MASTER_CONFIG`
Specify an alternative location for the salt master configuration file. The salt master configuration file is used to determine how to handle the minion RSA keys. Default location is /etc/salt/master.

`-V` `VM_CONFIG`, **`--profiles`**`=VM_CONFIG`, **`--vm_config`**`=VM_CONFIG`
Specify an alternative location for the salt cloud profiles file. Default location is /etc/salt/cloud.profiles.

`--raw-out`
Print the output from the salt command in raw python form, this is suitable for re-reading the output into an executing python script with eval.

`--text-out`
Print the output from the salt command in the same form the shell would.

`--yaml-out`
Print the output from the salt command in yaml.

`--json-out`
Print the output from the salt command in json.

`--no-color`
Disable all colored output.

119.4 Examples

To create 4 VMs named web1, web2, db1 and db2 from specified profiles:

```
salt-cloud -p fedora_rackspace web1 web2 db1 db2
```

To read in a map file and create all VMs specified therein:

```
salt-cloud -m /path/to/cloud.map
```

To read in a map file and create all VMs specified therein in parallel:

```
salt-cloud -m /path/to/cloud.map -P
```

To delete any VMs specified in the map file:

```
salt-cloud -m /path/to/cloud.map -d
```

To delete any VMs NOT specified in the map file:

```
salt-cloud -m /path/to/cloud.map -H
```

To display the status of all VMs specified in the map file:

```
salt-cloud -m /path/to/cloud.map -Q
```

119.5 See also

`salt-cloud(7)` *`salt(7)`* *`salt-master(1)`* *`salt-minion(1)`*

Release notes and upgrade instructions

120.1 Salt 0.10.0 Release Notes

release 2012-06-16

0.10.0 has arrived! This release comes with MANY bug fixes, and new capabilities which greatly enhance performance and reliability. This release is primarily a bug fix release with many new tests and many repaired bugs. This release also introduces a few new key features which were brought in primarily to repair bugs and some limitations found in some of the components of the original architecture.

120.1.1 Major Features

Event System

The Salt Master now comes equipped with a new event system. This event system has replaced some of the back end of the Salt client and offers the beginning of a system which will make plugging external applications into Salt. The event system relies on a local ZeroMQ publish socket and other processes can connect to this socket and listen for events. The new events can be easily managed via Salt's event library.

Unprivileged User Updates

Some enhancements have been added to Salt for running as a user other than root. These new additions should make switching the user that the Salt Master is running as very painless, simply change the `user` option in the master configuration and restart the master, Salt will take care of all of the particulars for you.

Peer Runner Execution

Salt has long had the peer communication system used to allow minions to send commands via the salt master. 0.10.0 adds a new capability here, now the master can be configured to allow for minions to execute Salt runners via the `peer_run` option in the salt master configuration.

YAML Parsing Updates

In the past the YAML parser for sls files would return the incorrect numbers when the file mode was set with a preceding 0. The YAML parser used in Salt has been modified to no longer convert these number into octal but to keep them as the correct value so that sls files can be a little cleaner to write.

120.1.2 State Call Data Files

It was requested that the minion keep a local cache of the most recent executed state run. This has been added and now with state runs the data is stored in a msgpack file in the minion's cachedir.

120.1.3 Turning Off the Job Cache

A new option has been added to the master configuration file. In previous releases the Salt client would look over the Salt job cache to read in the minion return data. With the addition of the event system the Salt client can now watch for events directly from the master worker processes.

This means that the job cache is no longer a hard requirement. Keep in mind though, that turning off the job cache means that historic job execution data cannot be retrieved.

120.1.4 Test Updates

Minion Swarms Are Faster

To continue our efforts with testing Salt's ability to scale the minionswarm script has been updated. The minionswarm can now start up minions much faster than it could before and comes with a new feature allowing modules to be disabled, thus lowering the minion's footprint when making a swarm. These new updates have allows us to test

```
# python minionswarm.py -m 20 --master salt-master
```

120.1.5 Many Fixes

To get a good idea for the number of bugfixes this release offers take a look at the closed tickets for 0.10.0, this is a very substantial update:

https://github.com/saltstack/salt/issues?milestone=12&state=closed

Master and Minion Stability Fixes

As Salt deployments grow new ways to break Salt are discovered. 0.10.0 comes with a number of fixes for the minions and master greatly improving Salt stability.

120.2 Salt 0.10.1 Release Notes

release 2012-06-19

120.3 Salt 0.10.2 Release Notes

release 2012-07-30

0.10.2 is out! This release comes with enhancements to the pillar interface, cleaner ways to access the salt-call capabilities in the API, minion data caching and the event system has been added to salt minions.

There have also been updates to the ZeroMQ functions, many more tests (thanks to sponsors, the code sprint and many contributors) and a swath of bug fixes.

120.3.1 Major Features

Ext Pillar Modules

The ranks of available Salt modules directories sees a new member in 0.10.2. With the popularity of pillar a higher demand has arisen for `ext_pillar` interfaces to be more like regular Salt module additions. Now ext_pillar interfaces can be added in the same way as other modules, just drop it into the pillar directory in the salt source.

Minion Events

In 0.10.0 an event system was added to the Salt master. 0.10.2 adds the event system to the minions as well. Now event can be published on a local minion as well.

The minions can also send events back up to the master. This means that Salt is able to communicate individual events from the minions back up to the Master which are not associated with command.

Minion Data Caching

When pillar was introduced the landscape for available data was greatly enhanced. The minion's began sending grain data back to the master on a regular basis.

The new config option on the master called `minion_data_cache` instructs the Salt master to maintain a cache of the minion's grains and pillar data in the cachedir. This option is turned off by default to avoid hitting the disk more, but when enabled the cache is used to make grain matching from the salt command more powerful, since the minions that will match can be predetermined.

Backup Files

By default all files replaced by the file.managed and file.recurse states we simply deleted. 0.10.2 adds a new option. By setting the backup option to `minion` the files are backed up before they are replaced.

The backed up files are located in the cachedir under the file_backup directory. On a default system this will be at: `/var/cache/salt/file_backup`

Configuration files

`salt-master` and `salt-minion` automatically load additional configuration files from `master.d/*.conf` respective `minion.d/*.conf` where `master.d/minion.d` is a directory in the same directory as the main configuration file.

Salt Key Verification

A number of users complained that they had inadvertently deleted the wrong salt authentication keys. 0.10.2 now displays what keys are going to be deleted and verifies that they are the keys that are intended for deletion.

Key auto-signing

If `autosign_file` is specified in the configuration file incoming keys will be compared to the list of keynames in `autosign_file`. Regular expressions as well as globbing is supported.

The file must only be writable by the user otherwise the file will be ignored. To relax the permission and allow group write access set the `permissive_pki_access` option.

Module changes

Improved OpenBSD support

New modules for managing services and packages were provided by Joshua Elsasser to further improve the support for OpenBSD.

Existing modules like the *disk* module were also improved to support OpenBSD.

SQL Modules

The MySQL and PostgreSQL modules have both received a number of additions thanks to the work of Avi Marcus and Roman Imankulov.

ZFS Support on FreeBSD

A new ZFS module has been added by Kurtis Velarde for FreeBSD supporting various ZFS operations like creating, extending or removing zpools.

Augeas

A new Augeas module by Ulrich Dangel for editing and verifying config files.

Native Debian Service module

The support for the Debian was further improved with an new service module for Debian by Ahmad Khayyat supporting *disable* and *enable*.

Cassandra

Cassandra support has been added by Adam Garside. Currently only status and diagnostic information are supported.

Networking

The networking support for *RHEL* has been improved and supports bonding support as well as zeroconf configuration.

Monit

Basic monit support by Kurtis Velarde to control services via monit.

nzbget

Basic support for controlling nzbget by Joseph Hall

Bluetooth

Baisc `bluez` support for managing and controlling Bluetooth devices. Supports scanning as well as pairing/unpairing by Joseph Hall.

120.3.2 Test Updates

Consistency Testing

Another testing script has been added. A bug was found in pillar when many minions generated pillar data at the same time. The new `consist.py` script is the tests directory was created to reproduce bugs where data should always be consistent.

120.3.3 Many Fixes

To get a good idea for the number of bugfixes this release offers take a look at the closed tickets for 0.10.2, this is a very substantial update:

https://github.com/saltstack/salt/issues?milestone=24&page=1&state=closed

Master and Minion Stability Fixes

As Salt deployments grow new ways to break Salt are discovered. 0.10.2 comes with a number of fixes for the minions and master greatly improving Salt stability.

120.4 Salt 0.10.3 Release Notes

release 2012-09-30

The latest taste of Salt has come, this release has many fixes and feature additions. Modifications have been made to make ZeroMQ connections more reliable, the beginning of the ACL system is in place, a new command line parsing system has been added, dynamic module distribution has become more environment aware, the new *master_finger* option and many more!

120.4.1 Major Features

ACL System

The new ACL system has been introduced. The ACL system allows for system users other than root to execute salt commands. Users can be allowed to execute specific commands in the same way that minions are opened up to the peer system.

The configuration value to open up the ACL system is called `client_acl` and is configured like so:

```
client_acl:
  fred:
    - test..*
    - pkg.list_pkgs
```

Where *fred* is allowed access to functions in the test module and to the `pkg.list_pkgs` function.

Master Finger Option

The *master_finger* option has been added to improve the security of minion provisioning. The *master_finger* option allows for the fingerprint of the master public key to be set in the configuration file to double verify that the master is valid. This option was added in response to a motivation to pre-authenticate the master when provisioning new minions to help prevent man in the middle attacks in some situations.

Salt Key Fingerprint Generation

The ability to generate fingerprints of keys used by Salt has been added to `salt-key`. The new option *finger* accepts the name of the key to generate and display a fingerprint for.

```
salt-key -F master
```

Will display the fingerprints for the master public and private keys.

Parsing System

Pedro Algavio, aka s0undt3ch, has added a substantial update to the command line parsing system that makes the help message output much cleaner and easier to search through. Salt parsers now have *–versions-report* besides usual *–version* info which you can provide when reporting any issues found.

Key Generation

We have reduced the requirements needed for *salt-key* to generate minion keys. You're no longer required to have salt configured and it's common directories created just to generate keys. This might prove useful if you're batch creating keys to pre-load on minions.

Startup States

A few configuration options have been added which allow for states to be run when the minion daemon starts. This can be a great advantage when deploying with Salt because the minion can apply states right when it first runs. To use startup states set the `startup_states` configuration option on the minion to *highstate*.

New Exclude Declaration

Some users have asked about adding the ability to ensure that other sls files or ids are excluded from a state run. The exclude statement will delete all of the data loaded from the specified sls file or will delete the specified id:

```
exclude:
  - sls: http
  - id: /etc/vimrc
```

Max Open Files

While we're currently unable to properly handle ZeroMQ's abort signals when the max open files is reached, due to the way that's handled on ZeroMQ's, we have minimized the chances of this happening without at least warning the user.

More State Output Options

Some major changes have been made to the state output system. In the past state return data was printed in a very verbose fashion and only states that failed or made changes were printed by default. Now two options can be passed to the master and minion configuration files to change the behavior of the state output. State output can be set to verbose (default) or non-verbose with the `state_verbose` option:

```
state_verbose: False
```

It is noteworthy that the state_verbose option used to be set to *False* by default but has been changed to *True* by default in 0.10.3 due to many requests for the change.

Te next option to be aware of new and called `state_output`. This option allows for the state output to be set to *full* (default) or *terse*.

The *full* output is the standard state output, but the new *terse* output will print only one line per state making the output much easier to follow when executing a large state system.

```
state_output: terse
```

state.file.append Improvements

The salt state *file.append()* tries *not* to append existing text. Previously the matching check was being made line by line. While this kind of check might be enough for most cases, if the text being appended was multi-line, the check would not work properly. This issue is now properly handled, the match is done as a whole ignoring any white space addition or removal except inside commas. For those thinking that, in order to properly match over multiple lines, salt will load the whole file into memory, that's not true. For most cases this is not important but an erroneous order to read a 4GB file, if not properly handled, like salt does, could make salt chew that amount of memory. Salt has a buffered file reader which will keep in memory a maximum of 256KB and iterates over the file in chunks of 32KB to test for the match, more than enough, if not, explain your usage on a ticket. With this change, also *salt.modules.file.contains()*, *salt.modules.file.contains_regex()*, *salt.modules.file.contains_glob()* and *salt.utils.find* now do the searching and/or matching using the buffered chunks approach explained above.

Two new keyword arguments were also added, *makedirs* and *source*. The first, *makedirs* will create the necessary directories in order to append to the specified file, of course, it only applies if we're trying to append to a non-existing file on a non-existing directory:

```
/tmp/salttest/file-append-makedirs:
    file.append:
        text: foo
        makedirs: True
```

The second, *source*, allows one to append the contents of a file instead of specifying the text.

```
/tmp/salttest/file-append-source:

file.append:
    - source: salt://testfile
```

120.4.2 Security Fix

A timing vulnerability was uncovered in the code which decrypts the AES messages sent over the network. This has been fixed and upgrading is strongly recommended.

120.5 Salt 0.10.4 Release Notes

release 2012-10-23

Salt 0.10.4 is a monumental release for the Salt team, with two new module systems, many additions to allow granular access to Salt, improved platform support and much more.

This release is also exciting because we have been able to shorten the release cycle back to under a month. We are working hard to keep up the aggressive pace and look forward to having releases happen more frequently!

This release also includes a serious security fix and all users are very strongly recommended to upgrade. As usual, upgrade the master first, and then the minion to ensure that the process is smooth.

120.5.1 Major Features

External Authentication System

The new external authentication system allows for Salt to pass through authentication to any authentication system to determine if a user has permission to execute a Salt command. The Unix PAM system is the first supported system with more to come!

The external authentication system allows for specific users to be granted access to execute specific functions on specific minions. Access is configured in the master configuration file, and uses the new access control system:

```
external_auth:
  pam:
    thatch:
      - 'web*':
        - test.*
        - network.*
```

The configuration above allows the user *thatch* to execute functions in the test and network modules on minions that match the web* target.

Access Control System

All Salt systems can now be configured to grant access to non-administrative users in a granular way. The old configuration continues to work. Specific functions can be opened up to specific minions from specific users in the case of external auth and client ACLs, and for specific minions in the case of the peer system.

Access controls are configured like this:

```
client_acl:
  fred:
    - web\*:
      - pkg.list_pkgs
      - test.*
      - apache.*
```

Target by Network

A new matcher has been added to the system which allows for minions to be targeted by network. This new matcher can be called with the *-S* flag on the command line and is available in all places that the matcher system is available. Using it is simple:

```
$ salt -S '192.168.1.0/24' test.ping
$ salt -S '192.168.1.100' test.ping
```

Nodegroup Nesting

Previously a nodegroup was limited by not being able to include another nodegroup, this restraint has been lifted and now nodegroups will be expanded within other nodegroups with the *N@* classifier.

Salt Key Delete by Glob

The ability to delete minion keys by glob has been added to `salt-key`. To delete all minion keys whose minion name starts with 'web':

```
$ salt-key -d 'web*'
```

Master Tops System

The *external_nodes* system has been upgraded to allow for modular subsystems to be used to generate the top file data for a highstate run.

The *external_nodes* option still works but will be deprecated in the future in favor of the new *master_tops* option.

Example of using *master_tops*:

```
master_tops:
  ext_nodes: cobbler-external-nodes
```

Next Level Solaris Support

A lot of work has been put into improved Solaris support by Romeo Theriault. Packaging modules (pkgadd/pkgrm and pkgutil) and states, cron support and user and group management have all been added and improved upon. These additions along with SMF (Service Management Facility) service support and improved Solaris grain detection in 0.10.3 add up to Salt becoming a great tool to manage Solaris servers with.

120.5.2 Security

A vulnerability in the security handshake was found and has been repaired, old minions should be able to connect to a new master, so as usual, the master should be updated first and then the minions.

Pillar Updates

The pillar communication has been updated to add some extra levels of verification so that the intended minion is the only one allowed to gather the data. Once all minions and the master are updated to salt 0.10.4 please activate pillar *2* by changing the *pillar_version* in the master config to *2*. This will be set to *2* by default in a future release.

120.6 Salt 0.10.5 Release Notes

release 2012-11-15

Salt 0.10.5 is ready, and comes with some great new features. A few more interfaces have been modularized, like the outputter system. The job cache system has been made more powerful and can now store and retrieve jobs archived in external databases. The returner system has been extended to allow minions to easily retrieve data from a returner interface.

As usual, this is an exciting release, with many noteworthy additions!

120.6.1 Major Features

External Job Cache

The external job cache is a system which allows for a returner interface to also act as a job cache. This system is intended to allow users to store job information in a central location for longer periods of time and to make the act of looking up information from jobs executed on other minions easier.

Currently the external job cache is supported via the mongo and redis returners:

```
ext_job_cache: redis
redis.host: salt
```

Once the external job cache is turned on the new *ret* module can be used on the minions to retrieve return information from the job cache. This can be a great way for minions to respond and react to other minions.

OpenStack Additions

OpenStack integration with Salt has been moving forward at a blistering pace. The new *nova*, *glance* and *keystone* modules represent the beginning of ongoing OpenStack integration.

The Salt team has had many conversations with core OpenStack developers and is working on linking to OpenStack in powerful new ways.

Wheel System

A new API was added to the Salt Master which allows the master to be managed via an external API. This new system allows Salt API to easily hook into the Salt Master and manage configs, modify the state tree, manage the pillar and more. The main motivation for the wheel system is to enable features needed in the upcoming web UI so users can manage the master just as easily as they manage minions.

The wheel system has also been hooked into the external auth system. This allows specific users to have granular access to manage components of the Salt Master.

Render Pipes

Jack Kuan has added a substantial new feature. The render pipes system allows Salt to treat the render system like unix pipes. This new system enables sls files to be passed through specific render engines. While the default renderer is still recommended, different engines can now be more easily merged. So to pipe the output of Mako used in YAML use this shebang line:

#!mako|yaml

Salt Key Overhaul

The Salt Key system was originally developed as only a CLI interface, but as time went on it was pressed into becoming a clumsy API. This release marks a complete overhaul of Salt Key. Salt Key has been rewritten to function purely from an API and to use the outputter system. The benefit here is that the outputter system works much more cleanly with Salt Key now, and the internals of Salt Key can be used much more cleanly.

Modular Outputters

The outputter system is now loaded in a modular way. This means that output systems can be more easily added by dropping a python file down on the master that contains the function *output*.

Gzip from Fileserver

Gzip compression has been added as an option to the cp.get_file and cp.get_dir commands. This will make file transfers more efficient and faster, especially over slower network links.

Unified Module Configuration

In past releases of Salt, the minions needed to be configured for certain modules to function. This was difficult because it required pre-configuring the minions. 0.10.5 changes this by making all module configs on minions search the master config file for values.

Now if a single database server is needed, then it can be defined in the master config and all minions will become aware of the configuration value.

Salt Call Enhancements

The `salt-call` command has been updated in a few ways. Now, `salt-call` can take the –return option to send the data to a returner. Also, `salt-call` now reports executions in the minion proc system, this allows the master to be aware of the operation salt-call is running.

Death to pub_refresh and sub_timeout

The old configuration values *pub_refresh* and *sub_timeout* have been removed. These options were in place to alleviate problems found in earlier versions of ZeroMQ which have since been fixed. The continued use of these options has proven to cause problems with message passing and have been completely removed.

Git Revision Versions

When running Salt directly from git (for testing or development, of course) it has been difficult to know exactly what code is being executed. The new versioning system will detect the git revision when building and how many commits have been made since the last release. A release from git will look like this:

0.10.4-736-gec74d69

Svn Module Addition

Anthony Cornehl (twinshadow) contributed a module that adds Subversion support to Salt. This great addition helps round out Salt's VCS support.

120.6.2 Noteworthy Changes

Arch Linux Defaults to Systemd

Arch Linux recently changed to use systemd by default and discontinued support for init scripts. Salt has followed suit and defaults to systemd now for managing services in Arch.

Salt, Salt Cloud and Openstack

With the releases of Salt 0.10.5 and Salt Cloud 0.8.2, OpenStack becomes the first (non-OS) piece of software to include support both on the user level (with Salt Cloud) and the admin level (with Salt). We are excited to continue to extend support of other platforms at this level.

120.7 Salt 0.11.0 Release Notes

release 2012-12-14

Salt 0.11.0 is here, with some highly sought after and exciting features. These features include the new overstate system, the reactor system, a new state run scope component called __context__, the beginning of the search system (still needs a great deal of work), multiple package states, the MySQL returner and a better system to arbitrarily reference outputters.

It is also noteworthy that we are changing how we mark release numbers. For the life of the project we have been pushing every release with features and fixes as point releases. We will now be releasing point releases for only bug fixes on a more regular basis and major feature releases on a slightly less regular basis. This means that the next release will be a bugfix only release with a version number of 0.11.1. The next feature release will be named 0.12.0 and will mark the end of life for the 0.11 series.

120.7.1 Major Features

OverState

The overstate system is a simple way to manage rolling state executions across many minions. The overstate allows for a state to depend on the successful completion of another state.

Reactor System

The new reactor system allows for a reactive logic engine to be created which can respond to events within a salted environment. The reactor system uses sls files to match events fired on the master with actions, enabling Salt to react to problems in an infrastructure.

Your load-balanced group of webservers is under extra load? Spin up a new VM and add it to the group. Your fileserver is filling up? Send a notification to your sysadmin on call. The possibilities are endless!

Module Context

A new component has been added to the module loader system. The module context is a data structure that can hold objects for a given scope within the module.

This allows for components that are initialized to be stored in a persistent context which can greatly speed up ongoing connections. Right now the best example can be found in the *cp* execution module.

Multiple Package Management

A long desired feature has been added to package management. By definition Salt States have always installed packages one at a time. On most platforms this is not the fastest way to install packages. Erik Johnson, aka terminalmage, has modified the package modules for many providers and added new capabilities to install groups of packages. These package groups can be defined as a list of packages available in repository servers:

```
python_pkgs:
  pkg.installed:
    - pkgs:
      - python-mako
      - whoosh
      - python-git
```

or specify based on the location of specific packages:

```
python_pkgs:
  pkg.installed:
    - sources:
      - python-mako: http://some-rpms.org/python-mako.rpm
      - whoosh: salt://whoosh/whoosh.rpm
      - python-git: ftp://companyserver.net/python-git.rpm
```

Search System

The bones to the search system have been added. This is a very basic interface that allows for search backends to be added as search modules. The first supported search module is the whoosh search backend. Right now only the basic paths for the search system are in place, making this very experimental. Further development will involve improving the search routines and index routines for whoosh and other search backends.

The search system has been made to allow for searching through all of the state and pillar files, configuration files and all return data from minion executions.

120.7.2 Notable Changes

All previous versions of Salt have shared many directories between the master and minion. The default locations for keys, cached data and sockets has been shared by master and minion. This has created serious problems with running a master and a minion on the same systems. 0.11.0 changes the defaults to be separate directories. Salt will also attempt to migrate all of the old key data into the correct new directories, but if it is not successful it may need to be done manually. If your keys exhibit issues after updating make sure that they have been moved from `/etc/salt/pki` to `/etc/salt/pki/{master,minion}`.

The old setup will look like this:

```
/etc/salt/pki
|-- master.pem
|-- master.pub
|-- minions
|   `-- ragnarok.saltstack.net
|-- minions_pre
|-- minion.pem
|-- minion.pub
|-- minion_master.pub
|-- minions_pre
`-- minions_rejected
```

With the accepted minion keys in `/etc/salt/pki/minions`, the new setup places the accepted minion keys in `/etc/salt/pki/master/minions`.

```
/etc/salt/pki
|-- master
|   |-- master.pem
|   |-- master.pub
|   |-- minions
|   |   `-- ragnarok.saltstack.net
|   |-- minions_pre
|   `-- minions_rejected
|-- minion
|   |-- minion.pem
|   |-- minion.pub
|   `-- minion_master.pub
```

120.8 Salt 0.11.1 Release Notes

release 2012-12-19

120.9 Salt 0.12.0 Release Notes

release 2013-01-15

Another feature release of Salt is here! Some exciting additions are included with more ways to make salt modular and even easier management of the salt file server.

120.9.1 Major Features

Modular Fileserver Backend

The new modular fileserver backend allows for any external system to be used as a salt file server. The main benefit here is that it is now possible to tell the master to directly use a git remote location, or many git remote locations, automatically mapping git branches and tags to salt environments.

Windows is First Class!

A new Salt Windows installer is now available! Much work has been put in to improve Windows support. With this much easier method of getting Salt on your Windows machines, we hope even more development and progress will occur. Please file bug reports on the Salt GitHub repo issue tracker so we can continue improving.

One thing that is missing on Windows that Salt uses extensively is a software package manager and a software package repository. The Salt pkg state allows sys admins to install software across their infrastructure and across operating systems. Software on Windows can now be managed in the same way. The SaltStack team built a package manager that interfaces with the standard Salt pkg module to allow for installing and removing software on Windows. In addition, a software package repository has been built on top of the Salt fileserver. A small YAML file provides the information necessary for the package manager to install and remove software.

An interesting feature of the new Salt Windows software package repository is that one or more remote git repositories can supplement the master's local repository. The repository can point to software on the master's fileserver or on an HTTP, HTTPS, or ftp server.

New Default Outputter

Salt displays data to the terminal via the outputter system. For a long time the default outputter for Salt has been the python pretty print library. While this has been a generally reasonable outputter, it did have many failings. The new default outputter is called "nested", it recursively scans return data structures and prints them out cleanly.

If the result of the new nested outputter is not desired any other outputter can be used via the –out option, or the output option can be set in the master and minion configs to change the default outputter.

Internal Scheduler

The internal Salt scheduler is a new capability which allows for functions to be executed at given intervals on the minion, and for runners to be executed at given intervals on the master. The scheduler allows for sequences such as executing state runs (locally on the minion or remotely via an overstate) or continually gathering system data to be run at given intervals.

The configuration is simple, add the schedule option to the master or minion config and specify jobs to run, this in the master config will execute the state.over runner every 60 minutes:

```
schedule:
  overstate:
    function: state.over
    minutes: 60
```

This example for the minion configuration will execute a highstate every 30 minutes:

```
schedule:
  highstate:
    function: state.highstate
    minutes: 30
```

Optional DSL for SLS Formulas

Jack Kuan, our renderer expert, has created something that is astonishing. Salt, now comes with an optional Python based DSL, this is a very powerful interface that makes writing SLS files in pure python easier than it was with the raw py renderer. As usual this can be used with the renderer shebang line, so a single sls can be written with the DSL if pure python power is needed while keeping other sls files simple with YAML.

Set Grains Remotely

A new execution function and state module have been added that allows for grains to be set on the minion. Now grains can be set via a remote execution or via states. Use the *grains.present* state or the *grains.setval* execution functions.

Gentoo Additions

Major additions to Gentoo specific components have been made. The encompasses executions modules and states ranging from supporting the make.conf file to tools like layman.

120.10 Salt 0.12.1 Release Notes

release 2013-01-21

120.11 Salt 0.13.0 Release Notes

release 2013-02-12

The lucky number 13 has turned the corner! From CLI notifications when quitting a salt command, to substantial improvements on Windows, Salt 0.13.0 has arrived!

120.11.1 Major Features

Improved file.recurse Performance

The file.recurse system has been deployed and used in a vast array of situations. Fixes to the file state and module have led towards opening up new ways of running file.recurse to make it faster. Now the file.recurse state will download fewer files and will run substantially faster.

Windows Improvements

Minion stability on Windows has improved. Many file operations, including file.recurse, have been fixed and improved. The network module works better, to include network.interfaces. Both 32bit and 64bit installers are now available.

Nodegroup Targeting in Peer System

In the past, nodegroups were not available for targeting via the peer system. This has been fixed, allowing the new nodegroup expr_form argument for the publish.publish function:

> salt-call publish.publish group1 test.ping expr_form=nodegroup

Blacklist Additions

Additions allowing more granular blacklisting are available in 0.13.0. The ability to blacklist users and functions in client_acl have been added, as well as the ability to exclude state formulas from the command line.

Command Line Pillar Embedding

Pillar data can now be embedded on the command line when calling `state.sls` and `state.highstate`. This allows for on the fly changes or settings to pillar and makes parameterizing state formulas even easier. This is done via the keyword argument:

```
salt '*' state.highstate pillar='{"cheese": "spam"}'
```

The above example will extend the existing pillar to hold the `cheese` key with a value of `spam`. If the `cheese` key is already specified in the minion's pillar then it will be overwritten.

CLI Notifications

In the past hitting ctrl-C and quitting from the `salt` command would just drop to a shell prompt, this caused confusion with users who expected the remote executions to also quit. Now a message is displayed showing what command can be used to track the execution and what the job id is for the execution.

Version Specification in Multiple-Package States

Versions can now be specified within multiple-package `pkg.installed` states. An example can be found below:

```
mypkgs:
  pkg.installed:
    - pkgs:
      - foo
      - bar: 1.2.3-4
      - baz
```

120.11.2 Noteworthy Changes

The configuration subsystem in Salt has been overhauled to make the `opts` dict used by Salt applications more portable, the problem is that this is an incompatible change with salt-cloud, and salt-cloud will need to be updated to the latest git to work with Salt 0.13.0. Salt Cloud 0.8.5 will also require Salt 0.13.0 or later to function.

The SaltStack team is sorry for the inconvenience here, we work hard to make sure these sorts of things do not happen, but sometimes hard changes get in.

120.12 Salt 0.13.1 Release Notes

release 2013-02-15

120.13 Salt 0.13.2 Release Notes

release 2013-03-13

120.14 Salt 0.13.3 Release Notes

release 2013-03-18

120.15 Salt 0.14.0 Release Notes

release 2013-03-23

Salt 0.14.0 is here! This release was held up primarily by PyCon, Scale and illness, but has arrived! 0.14.0 comes with many new features and is breaking ground for Salt in the area of cloud management with the introduction of Salt providing basic cloud controller functionality.

120.15.1 Major Features

Salt - As a Cloud Controller

This is the first primitive inroad to using Salt as a cloud controller is available in 0.14.0. Be advised that this is alpha, only tested in a few very small environments.

The cloud controller is built using kvm and libvirt for the hypervisors. Hypervisors are autodetected as minions and only need to have libvirt running and kvm installed to function. The features of the Salt cloud controller are as follows:

- Basic vm discovery and reporting
- Creation of new virtual machines
- Seeding virtual machines with Salt via qemu-nbd or libguestfs
- Live migration (shared and non shared storage)
- Delete existing VMs

It is noteworthy that this feature is still Alpha, meaning that all rights are reserved to change the interface if needs be in future releases!

Libvirt State

One of the problems with libvirt is management of certificates needed for live migration and cross communication between hypervisors. The new `libvirt` state makes the Salt Master hold a CA and manage the signing and distribution of keys onto hypervisors, just add a call to the libvirt state in the sls formulas used to set up a hypervisor:

```
libvirt_keys:
  libvirt.keys
```

New get Functions

An easier way to manage data has been introduced. The pillar, grains and config execution modules have been extended with the new `get` function. This function works much in the same way as the get method in a python dict, but with an enhancement, nested dict components can be extracted using a *:* delimiter.

If a structure like this is in pillar:

```
foo:
  bar:
    baz: quo
```

Extracting it from the raw pillar in an sls formula or file template is done this way:

```
{{ pillar['foo']['bar']['baz'] }}
```

Now with the new get function the data can be safely gathered and a default can be set allowing the template to fall back if the value is not available:

```
{{ salt['pillar.get']('foo:bar:baz', 'qux') }}
```

This makes handling nested structures much easier, and defaults can be cleanly set. This new function is being used extensively in the new formulae repository of salt sls formulas.

120.16 Salt 0.14.1 Release Notes

release 2013-04-13

120.17 Salt 0.15.0 Release Notes

release 2013-05-03

The many new features of Salt 0.15.0 have arrived! Salt 0.15.0 comes with many smaller features and a few larger ones.

These features range from better debugging tools to the new Salt Mine system.

120.17.1 Major Features

The Salt Mine

First there was the peer system, allowing for commands to be executed from a minion to other minions to gather data live. Then there was the external job cache for storing and accessing long term data. Now the middle ground is being filled in with the Salt Mine. The Salt Mine is a system used to execute functions on a regular basis on minions and then store only the most recent data from the functions on the master, then the data is looked up via targets.

The mine caches data that is public to all minions, so when a minion posts data to the mine all other minions can see it.

IPV6 Support

0.13.0 saw the addition of initial IPV6 support but errors were encountered and it needed to be stripped out. This time the code covers more cases and must be explicitly enabled. But the support is much more extensive than before.

Copy Files From Minions to the Master

Minions have long been able to copy files down from the master file server, but until now files could not be easily copied from the minion up to the master.

A new function called `cp.push` can push files from the minions up to the master server. The uploaded files are then cached on the master in the master cachedir for each minion.

Better Template Debugging

Template errors have long been a burden when writing states and pillar. 0.15.0 will now send the compiled template data to the debug log, this makes tracking down the intermittent stage templates much easier. So running state.sls or state.highstate with *-l debug* will now print out the rendered templates in the debug information.

State Event Firing

The state system is now more closely tied to the master's event bus. Now when a state fails the failure will be fired on the master event bus so that the reactor can respond to it.

Major Syndic Updates

The Syndic system has been basically re-written. Now it runs in a completely asynchronous way and functions primarily as an event broker. This means that the events fired on the syndic are now pushed up to the higher level master instead of the old method used which waited for the client libraries to return.

This makes the syndic much more accurate and powerful, it also means that all events fired on the syndic master make it up the pipe as well making a reactor on the higher level master able to react to minions further downstream.

Peer System Updates

The Peer System has been updated to run using the client libraries instead of firing directly over the publish bus. This makes the peer system much more consistent and reliable.

Minion Key Revocation

In the past when a minion was decommissioned the key needed to be manually deleted on the master, but now a function on the minion can be used to revoke the calling minion's key:

```
$ salt-call saltutil.revoke_auth
```

Function Return Codes

Functions can now be assigned numeric return codes to determine if the function executed successfully. While not all functions have been given return codes, many have and it is an ongoing effort to fill out all functions that might return a non-zero return code.

Functions in Overstate

The overstate system was originally created to just manage the execution of states, but with the addition of return codes to functions, requisite logic can now be used with respect to the overstate. This means that an overstate stage can now run single functions instead of just state executions.

Pillar Error Reporting

Previously if errors surfaced in pillar, then the pillar would consist of only an empty dict. Now all data that was successfully rendered stays in pillar and the render error is also made available. If errors are found in the pillar, states will refuse to run.

Using Cached State Data

Sometimes states are executed purely to maintain a specific state rather than to update states with new configs. This is grounds for the new cached state system. By adding *cache=True* to a state call the state will not be generated fresh from the master but the last state data to be generated will be used. If no previous state data is available then fresh data will be generated.

Monitoring States

The new monitoring states system has been started. This is very young but allows for states to be used to configure monitoring routines. So far only one monitoring state is available, the `disk.status` state. As more capabilities are added to Salt UI the monitoring capabilities of Salt will continue to be expanded.

120.18 Salt 0.15.1 Release Notes

release 2013-05-08

The 0.15.1 release has been posted, this release includes fixes to a number of bugs in 0.15.1 and a three security patches.

120.18.1 Security Updates

A number of security issues have been resolved via the 0.15.1 release.

Path Injection in Minion IDs

Salt masters did not properly validate the id of a connecting minion. This can lead to an attacker uploading files to the master in arbitrary locations. In particular this can be used to bypass the manual validation of new unknown minions. Exploiting this vulnerability does not require authentication.

This issue affects all known versions of Salt.

This issue was reported by Ronald Volgers.

Patch

The issue is fixed in Salt 0.15.1. Updated packages are available in the usual locations.

Specific commits:

https://github.com/saltstack/salt/commit/5427b9438e452a5a8910d9128c6aafb45d8fd5d3

https://github.com/saltstack/salt/commit/7560908ee62351769c3cd43b03d74c1ca772cc52

https://github.com/saltstack/salt/commit/e200b8a7ff53780124e08d2bdefde7587e52bfca

RSA Key Generation Fault

RSA key generation was done incorrectly, leading to very insecure keys. It is recommended to regenerate all RSA keys.

This issue can be used to impersonate Salt masters or minions, or decrypt any transferred data.

This issue can only be exploited by attackers who are able to observe or modify traffic between Salt minions and the legitimate Salt master.

A tool was included in 0.15.1 to assist in mass key regeneration, the manage.regen_keys runner.

This issue affects all known versions of Salt.

This issue was reported by Ronald Volgers.

Patch

The issue is fixed in Salt 0.15.1. Updated packages are available in the usual locations.

Specific commits:

https://github.com/saltstack/salt/commit/5dd304276ba5745ec21fc1e6686a0b28da29e6fc

Command Injection Via ext_pillar

Arbitrary shell commands could be executed on the master by an authenticated minion through options passed when requesting a pillar.

Ext pillar options have been restricted to only allow safe external pillars to be called when prompted by the minion.

This issue affects Salt versions from 0.14.0 to 0.15.0.

This issue was reported by Ronald Volgers.

Patch

The issue is fixed in Salt 0.15.1. Updated packages are available in the usual locations.

Specific commits:

https://github.com/saltstack/salt/commit/43d8c16bd26159d827d1a945c83ac28159ec5865

120.19 Salt 0.15.2 Release Notes

release 2013-05-29

120.20 Salt 0.15.3 Release Notes

release 2013-06-01

120.21 Salt 0.16.0 Release Notes

release 2013-07-01

The 0.16.0 release is an exciting one, with new features in master redundancy, and a new, powerful requisite.

120.21.1 Major Features

Multi-Master

This new capability allows for a minion to be actively connected to multiple salt masters at the same time. This allows for multiple masters to send out commands to minions and for minions to automatically reconnect to masters that have gone down. A tutorial is available to help get started here:

Multi Master Tutorial

Prereq, the New Requisite

The new *prereq* requisite is very powerful! It allows for states to execute based on a state that is expected to make changes in the future. This allows for a change on the system to be preempted by another execution. A good example is needing to shut down a service before modifying files associated with it, allowing, for instance, a webserver to be shut down allowing a load balancer to stop sending requests while server side code is updated. In this case, the prereq will only run if changes are expected to happen in the prerequired state, and the prerequired state will always run after the prereq state and only if the prereq state succeeds.

Peer System Improvements

The peer system has been revamped to make it more reliable, faster, and like the rest of Salt, async. The peer calls when an updated minion and master are used together will be much faster!

Relative Includes

The ability to include an sls relative to the defined sls has been added, the new syntax id documented here:

Includes

More State Output Options

The `state_output` option in the past only supported *full* and *terse*, 0.16.0 add the *mixed* and *changes* modes further refining how states are sent to users' eyes.

Improved Windows Support

Support for Salt on Windows continues to improve. Software management on Windows has become more seamless with Linux/UNIX/BSD software management. Installed software is now recognized by the short names defined in the *repository SLS*. This makes it possible to run `salt '*' pkg.version firefox` and get back results from Windows and non-Windows minions alike.

When templating files on Windows, Salt will now correctly use Windows appropriate line endings. This makes it much easier to edit and consume files on Windows.

When using the cmd state the `shell` option now allows for specifying Windows Powershell as an alternate shell to execute cmd.run and cmd.script. This opens up Salt to all the power of Windows Powershell and its advanced Windows management capabilities.

Several fixes and optimizations were added for the Windows networking modules, especially when working with IPv6.

A system module was added that makes it easy to restart and shutdown Windows minions.

The Salt Minion will now look for its config file in `c:\salt\conf` by default. This means that it's no longer necessary to specify the `-c` option to specify the location of the config file when starting the Salt Minion on Windows in a terminal.

Muliple Targets for pkg.removed, pkg.purged States

Both `pkg.removed` and `pkg.purged` now support the `pkgs` argument, which allow for multiple packages to be targeted in a single state. This, as in `pkg.installed`, helps speed up these states by reducing the number of times that the package management tools (apt, yum, etc.) need to be run.

Random Times in Cron States

The temporal parameters in `cron.present` states (minute, hour, etc.) can now be randomized by using `random` instead of a specific value. For example, by using the `random` keyword in the `minute` parameter of a cron state, the same cron job can be pushed to hundreds or thousands of hosts, and they would each use a randomly-generated minute. This can be helpful when the cron job accesses a network resource, and it is not desirable for all hosts to run the job concurrently.

```
/path/to/cron/script:
  cron.present:
    - user: root
    - minute: random
    - hour: 2
```

Since Salt assumes a value of `*` for unspecified temporal parameters, adding a parameter to the state and setting it to `random` will change that value from `*` to a randomized numeric value. However, if that field in the cron entry on the minion already contains a numeric value, then using the `random` keyword will not modify it.

Confirmation Prompt on Key Acceptance

When accepting new keys with `salt-key -a minion-id` or `salt-key -A`, there is now a prompt that will show the affected keys and ask for confirmation before proceeding. This prompt can be bypassed using the `-y` or `--yes` command line argument, as with other `salt-key` commands.

Support for Setting Password Hashes on BSD Minions

FreeBSD, NetBSD, and OpenBSD all now support setting passwords in `user.present` states.

120.22 Salt 0.16.1 Release Notes

release 2013-07-29

120.23 Salt 0.16.2 Release Notes

release 2013-08-01

Version 0.16.2 is a bugfix release for *0.16.0*, and contains a number of fixes.

120.23.1 Windows

- Only allow Administrator's group and SYSTEM user access to C:\salt. This eliminates a race condition where a non-admin user could modify a template or managed file before it is executed by the minion (which is running as an elevated user), thus avoiding a potential escalation of privileges. (issue 6361)

120.23.2 Grains

- Fixed detection of `virtual` grain on OpenVZ hardware nodes
- Gracefully handle lsb_release data when it is enclosed in quotes

- LSB grains are now prefixed with `lsb_distrib_` instead of simply `lsb_`. The old naming is not preserved, so SLS may be affected.
- Improved grains detection on MacOS

120.23.3 Pillar

- Don't try to load *git_pillar* if not enabled in master config (issue 6052)
- Functions `pillar.item` and `pillar.items` added for parity with `grains.item`/`grains.items`. The old function `pillar.data` is preserved for backwards compatibility.
- Fixed minion traceback when Pillar SLS is malformed (issue 5910)

120.23.4 Peer Publishing

- More gracefully handle improperly quoted publish commands (issue 5958)
- Fixed traceback when timeout specified via the CLI fo `publish.publish`, `publish.full_data` (issue 5959)
- Fixed unintended change in output of `publish.publish` (issue 5928)

120.23.5 Minion

- Fixed salt-key usage in minionswarm script
- Quieted warning about **SALT_MINION_CONFIG** environment variable on minion startup and for CLI commands run via `salt-call` (issue 5956)
- Added minion config parameter `random_reauth_delay` to stagger re-auth attempts when the minion is waiting for the master to approve its public key. This helps prevent SYN flooding in larger environments.

120.23.6 User/Group Management

- Implement previously-ignored `unique` option for `user.present` states in FreeBSD
- Report in state output when a `group.present` state attempts to use a gid in use by another group
- Fixed regression that prevents a `user.present` state to set the password hash to the system default (i.e. an unset password)
- Fixed multiple `group.present` states with the same group (issue 6439)

120.23.7 File Management

- Fixed file.mkdir setting incorrect permissions (issue 6033)
- Fixed cleanup of source files for templates when `/tmp` is in file_roots (issue 6118)
- Fixed caching of zero-byte files when a non-empty file was previously cached at the same path
- Added HTTP authentication support to the cp module (issue 5641)
- Diffs are now suppressed when binary files are changed

120.23.8 Package/Repository Management

- Fixed traceback when there is only one target for `pkg.latest` states
- Fixed regression in detection of virtual packages (apt)
- Limit number of pkg database refreshes to once per `state.sls`/`state.highstate`
- YUM: Allow 32-bit packages with arches other than i686 to be managed on 64-bit systems (issue 6299)
- Fixed incorrect reporting in pkgrepo.managed states (issue 5517)
- Fixed 32-bit binary package installs on 64-bit RHEL-based distros, and added proper support for 32-bit packages on 64-bit Debian-based distros (issue 6303)
- Fixed issue where requisites were inadvertently being put into YUM repo files (issue 6471)

120.23.9 Service Management

- Fixed inaccurate reporting of results in `service.running` states when the service fails to start (issue 5894)
- Fixed handling of custom initscripts in RHEL-based distros so that they are immediately available, negating the need for a second state run to manage the service that the initscript controls

120.23.10 Networking

- Function network.hwaddr renamed to `network.hw_addr` to match `network.ip_addrs` and `network.ip_addrs6`. All three functions also now work without the underscore in the name, as well.
- Fixed traceback in `bridge.show` when interface is not present (issue 6326)

120.23.11 SSH

- Fixed incorrect result reporting for some `ssh_known_hosts.present` states
- Fixed inaccurate reporting when `ssh_auth.present` states are run with `test=True`, when rsa/dss is used for the `enc` param instead of ssh-rsa/ssh-dss (issue 5374)

120.23.12 pip

- Properly handle `-f` lines in pip freeze output
- Fixed regression in pip.installed states with specifying a requirements file (issue 6003)
- Fixed use of `editable` argument in `pip.installed` states (issue 6025)
- Deprecated `runas` parameter in execution function calls, in favor of `user`

120.23.13 MySQL

- Allow specification of *MySQL* connection arguments via the CLI, overriding/bypassing minion config params
- Allow `mysql_user.present` states to set a passwordless login (issue 5550)
- Fixed endless loop when `mysql.processlist` is run (issue 6297)

120.23.14 PostgreSQL

- Fixed traceback in `postgres.user_list` (issue 6352)

120.23.15 Miscellaneous

- Don't allow *npm states* to be used if *npm module* is not available
- Fixed `alternatives.install` states for which the target is a symlink (issue 6162)
- Fixed traceback in *sysbench module* (issue 6175)
- Fixed traceback in job cache
- Fixed tempfile cleanup for windows
- Fixed issue where SLS files using the *pydsl renderer* were not being run
- Fixed issue where returners were being passed incorrect information (issue 5518)
- Fixed traceback when numeric args are passed to `cmd.script` states
- Fixed bug causing `cp.get_dir` to return more directories than expected (issue 6048)
- Fixed traceback when `supervisord.running` states are run with `test=True` (issue 6053)
- Fixed tracebacks when Salt encounters problems running rbenv (issue 5888)
- Only make the *monit module* available if monit binary is present (issue 5871)
- Fixed incorrect behavior of `img.mount_image`
- Fixed traceback in `tomcat.deploy_war` in Windows
- Don't re-write /etc/fstab if mount fails
- Fixed tracebacks when Salt encounters problems running gem (issue 5886)
- Fixed incorrect behavior of `selinux.boolean` states (issue 5912)
- *RabbitMQ*: Quote passwords to avoid symbols being interpolated by the shell (issue 6338)
- Fixed tracebacks in `extfs.mkfs` and `extfs.tune` (issue 6462)
- Fixed a regression with the `module.run` state where the `m_name` and `m_fun` arguments were being ignored (issue 6464)

120.24 Salt 0.16.3 Release Notes

release 2013-08-09

Version 0.16.3 is another bugfix release for *0.16.0*. The changes include:

- Various documentation fixes
- Fix proc directory regression (issue 6502)
- Properly detect Linaro Linux (issue 6496)
- Fix regressions in `mount.mounted` (issue 6522, issue 6545)
- Skip malformed state requisites (issue 6521)
- Fix regression in gitfs from bad import

- Fix for watching prereq states (including recursive requisite error) (issue 6057)
- Fix mod_watch not overriding prereq (issue 6520)
- Don't allow functions which compile states to be called within states (issue 5623)
- Return error for malformed top.sls (issue 6544)
- Fix traceback in `mysql.query`
- Fix regression in binary package installation for 64-bit packages on Debian-based Linux distros (issue 6563)
- Fix traceback caused by running `cp.push` without having set `file_recv` in the master config file
- Fix scheduler configuration in pillar (issue 6201)

120.25 Salt 0.16.4 Release Notes

release 2013-09-07

Version 0.16.4 is another bugfix release for *0.16.0*, likely to be the last before 0.17.0 is released. The changes include:

- Multiple documentation improvements/additions
- Added the `osfinger` and `osarch` grains
- Properly handle 32-bit packages for debian32 on x86_64 (issue 6607)
- Fix regression in yum package installation in CentOS 5 (issue 6677)
- Fix bug in `hg.latest` state that would erroneously delete directories (issue 6661)
- Fix bug related to pid not existing for `ps.top` (issue 6679)
- Fix regression in `MySQL returner` (issue 6695)
- Fix IP addresses grains (`ipv4` and `ipv6`) to include all addresses (issue 6656)
- Fix regression preventing authenticated FTP (issue 6733)
- Fix setting password for windows users (issue 6824)
- Fix `file.contains` on values YAML parses as non-string (issue 6817)
- Fix `file.get_gid`, `file.get_uid`, and `file.chown` for broken symlinks (issue 6826)
- Fix comment for service reloads in service state (issue 6851)

120.26 Salt 0.17.0 Release Notes

release 2013-09-26

The 0.17.0 release is a very exciting release of Salt, this brings to Salt some very powerful new features and advances. The advances range from the state system to the test suite, covering new transport capabilities and making states easier and more powerful, to extending Salt Virt and much more!

The 0.17.0 release will also be the last release of Salt to follow the old 0.XX.X numbering system, the next release of Salt will change the numbering to be date based following this format:

<Year>.<Month>.<Minor>

So if the release happens in November of 2013 the number will be 13.11.0, the first bugfix release will be 13.11.1 and so forth.

120.26.1 Major Features

Halite

The new Halite web GUI is now available on PyPI. A great deal of work has been put into Halite to make it fully event driven and amazingly fast. The Halite UI can be started from within the Salt Master (after being installed from PyPI), or standalone, and does not require an external database to run. It is very lightweight!

This initial release of Halite is primarily the framework for the UI and the communication systems, making it easy to extend and build the UI up. It presently supports watching the event bus and firing commands over Salt.

At this time, Halite is not available as a package, but installation documentation is available at: http://docs.saltstack.com/topics/tutorials/halite.html

Halite is, like the rest of Salt, Open Source!

Much more will be coming in the future of Halite!

Salt SSH

The new `salt-ssh` command has been added to Salt. This system allows for remote execution and states to be run over ssh. The benefit here being, that salt can run relying only on the ssh agent, rather than requiring a minion to be deployed.

The `salt-ssh` system runs states in a compatible way as Salt and states created and run with salt-ssh can be moved over to a standard salt deployment without modification.

Since this is the initial release of salt-ssh, there is plenty of room for improvement, but it is fully operational, not just a bootstrap tool.

Rosters

Salt is designed to have the minions be aware of the master and the master does not need to be aware of the location of the minions. The new salt roster system was created and designed to facilitate listing the targets for salt-ssh.

The roster system, like most of Salt, is a plugin system, allowing for the list of systems to target to be derived from any pluggable backend. The rosters shipping with 0.17.0 are flat and scan. Flat is a file which is read in via the salt render system and the scan roster does simple network scanning to discover ssh servers.

State Auto Order

This is a major change in how states are evaluated in Salt. State Auto Order is a new feature that makes states get evaluated and executed in the order in which they are defined in the sls file. This feature makes it very easy to see the finite order in which things will be executed, making Salt now, fully imperative AND fully declarative.

The requisite system still takes precedence over the order in which states are defined, so no existing states should break with this change. But this new feature can be turned off by setting `state_auto_order:  False` in the master config, thus reverting to the old lexicographical order.

state.sls Runner

The `state.sls` runner has been created to allow for a more powerful system for orchestrating state runs and function calls across the salt minions. This new system uses the state system for organizing executions.

This allows for states to be defined that are executed on the master to call states on minions via `salt-run state.sls`.

Salt Thin

Salt Thin is an exciting new component of Salt, this is the ability to execute Salt routines without any transport mechanisms installed, it is a pure python subset of Salt.

Salt Thin does not have any networking capability, but can be dropped into any system with Python installed and then `salt-call` can be called directly. The Salt Thin system, is used by the `salt-ssh` command, but can still be used to just drop salt somewhere for easy use.

Event Namespacing

Events have been updated to be much more flexible. The tags in events have all been namespaced allowing easier tracking of event names.

Mercurial Fileserver Backend

The popular git fileserver backend has been joined by the mercurial fileserver backend, allowing the state tree to be managed entirely via mercurial.

External Logging Handlers

The external logging handler system allows for Salt to directly hook into any external logging system. Currently supported are sentry and logstash.

Jenkins Testing

The testing systems in Salt have been greatly enhanced, tests for salt are now executed, via jenkins.saltstack.com, across many supported platforms. Jenkins calls out to salt-cloud to create virtual machines on Rackspace, then the minion on the virtual machine checks into the master running on Jenkins where a state run is executed that sets up the minion to run tests and executes the test suite.

This now automates the sequence of running platform tests and allows for continuous destructive tests to be run.

Salt Testing Project

The testing libraries for salt have been moved out of the main salt code base and into a standalone codebase. This has been done to ease the use of the testing systems being used in salt based projects other than Salt itself.

StormPath External Authentication

The external auth system now supports the fantastic Stormpath cloud based authentication system.

LXC Support

Extensive additions have been added to Salt for LXC support. This included the backend libs for managing LXC containers. Addition into the salt-virt system is still in the works.

Mac OS X User/Group Support

Salt is now able to manage users and groups on Minions running Mac OS X. However, at this time user passwords cannot be managed.

Django ORM External Pillar

Pillar data can now be derived from Django managed databases.

Fixes from RC to release

- Multiple documentation fixes
- Add multiple source files + templating for `file.append` (issue 6905)
- Support sysctl configuration files in systemd>=207 (issue 7351)
- Add `file.search` and `file.replace`
- Fix cross-calling execution functions in provider overrides
- Fix locale override for postgres (issue 4543)
- Fix Raspbian identification for service/pkg support (issue 7371)
- Fix `cp.push` file corruption (issue 6495)
- Fix ALT Linux password hash specification (issue 3474)
- Multiple salt-ssh-related fixes and improvements

120.27 Salt 0.17.1 Release Notes

release 2013-10-17

Note: THIS RELEASE IS NOT COMPATIBLE WITH PREVIOUS VERSIONS. If you update your master to 0.17.1, you must update your minions as well. Sorry for the inconvenience – this is a result of one of the security fixes listed below.

The 0.17.1 release comes with a number of improvements to salt-ssh, many bugfixes, and a number of security updates.

Salt SSH has been improved to be faster, more featureful and more secure. Since the original release of Salt SSH was primarily a proof of concept, it has been very exciting to see its rapid adoption. We appreciate the willingness of security experts to review Salt SSH and help discover oversights and ensure that security issues only exist for such a tiny window of time.

120.27.1 SSH Enhancements

Shell Improvements

Improvements to Salt SSH's communication have been added that improve routine execution regardless of the target system's login shell.

Performance

Deployment of routines is now faster and takes fewer commands to execute.

120.27.2 Security Updates

Be advised that these security issues all apply to a small subset of Salt users and mostly apply to Salt SSH.

Insufficient Argument Validation

This issue allowed for a user with limited privileges to embed executions inside of routines to execute routines that should be restricted. This applies to users using external auth or client ACL and opening up specific routines.

Be advised that these patches address the direct issue. Additional commits have been applied to help mitigate this issue from resurfacing.

CVE

CVE-2013-4435

Affected Versions

0.15.0 - 0.17.0

Patches

https://github.com/saltstack/salt/commit/6d8ef68b605fd63c36bb8ed96122a75ad2e80269
https://github.com/saltstack/salt/commit/ebdef37b7e5d2b95a01d34b211c61c61da67e46a
https://github.com/saltstack/salt/commit/7f190ff890e47cdd591d9d7cefa5126574660824
https://github.com/saltstack/salt/commit/8e5afe59cef6743fe5dbd510dcf463dbdfca1ced
https://github.com/saltstack/salt/commit/aca78f314481082862e96d4f0c1b75fa382bb885
https://github.com/saltstack/salt/commit/6a9752cdb1e8df2c9505ea910434c79d132eb1e2
https://github.com/saltstack/salt/commit/b73677435ba54ecfc93c1c2d840a7f9ba6f53410
https://github.com/saltstack/salt/commit/07972eb0a6f985749a55d8d4a2e471596591c80d
https://github.com/saltstack/salt/commit/1e3f197726aa13ac5c3f2416000089f477f489b5

Found By

Feth Arezki, of Majerti

MITM SSH attack in salt-ssh

SSH host keys were being accepted by default and not enforced on future SSH connections. These patches set SSH host key checking by default and can be overridden by passing the -i flag to *salt-ssh*.

CVE

CVE-2013-4436

Affected Versions

0.17.0

Found By

Michael Scherer, Red Hat

Insecure Usage of /tmp in salt-ssh

The initial release of salt-ssh used the /tmp directory in an insecure way. These patches not only secure usage of files under /tmp in salt-ssh, but also add checksum validation for all packages sent into the now secure locations on target systems.

CVE

CVE-2013-4438

Affected Versions

0.17.0

Patches

https://github.com/saltstack/salt/commit/aa4bb77ef230758cad84381dde0ec660d2dc340a
https://github.com/saltstack/salt/commit/8f92b6b2cb2e4ec3af8783eb6bf4ff06f5a352cf
https://github.com/saltstack/salt/commit/c58e56811d5a50c908df0597a0ba0b643b45ebfd
https://github.com/saltstack/salt/commit/0359db9b46e47614cff35a66ea6a6a76846885d2
https://github.com/saltstack/salt/commit/4348392860e0fd43701c331ac3e681cf1a8c17b0
https://github.com/saltstack/salt/commit/664d1a1cac05602fad2693f6f97092d98a72bf61
https://github.com/saltstack/salt/commit/bab92775a576e28ff9db262f32db9cf2375bba87
https://github.com/saltstack/salt/commit/c6d34f1acf64900a3c87a2d37618ff414e5a704e

Found By

Michael Scherer, Red Hat

YAML Calling Unsafe Loading Routine

It has been argued that this is not a valid security issue, as the YAML loading that was happening was only being called after an initial gateway filter in Salt has already safely loaded the YAML and would fail if non-safe routines were embedded. Nonetheless, the CVE was filed and patches applied.

CVE

CVE-2013-4438

Patches

https://github.com/saltstack/salt/commit/339b0a51befae6b6b218ebcb55daa9cd3329a1c5

Found By

Michael Scherer, Red Hat

Failure to Drop Supplementary Group on Salt Master

If a salt master was started as a non-root user by the root user, root's groups would still be applied to the running process. This fix changes the process to have only the groups of the running user.

CVE

CVE not considered necessary by submitter.

Affected Versions

0.11.0 - 0.17.0

Patches

https://github.com/saltstack/salt/commit/b89fa9135822d029795ab1eecd68cce2d1ced715

Found By

Michael Scherer, Red Hat

Failure to Validate Minions Posting Data

This issue allowed a minion to pose as another authorized minion when posting data such as the mine data. All minions now pass through the id challenge before posting such data.

CVE

CVE-2013-4439

Affected Versions

0.15.0 - 0.17.0

Patches

https://github.com/saltstack/salt/commit/7b850ff3d07ef6782888914ac4556c01e8a1c482
https://github.com/saltstack/salt/commit/151759b2a1e1c6ce29277aa81b054219147f80fd

Found By

David Anderson

120.27.3 Fix Reference

Version 0.17.1 is the first bugfix release for *0.17.0*. The changes include:

- Fix symbolic links in thin.tgz (issue 7482)
- Pass env through to file.patch state (issue 7452)
- Service provider fixes and reporting improvements (issue 7361)
- Add `--priv` option for specifying salt-ssh private key
- Fix salt-thin's salt-call on setuptools installations (issue 7516)
- Fix salt-ssh to support passwords with spaces (issue 7480)
- Fix regression in wildcard includes (issue 7455)
- Fix salt-call outputter regression (issue 7456)
- Fix custom returner support for startup states (issue 7540)
- Fix value handling in augeas (issue 7605)
- Fix regression in apt (issue 7624)
- Fix minion ID guessing to use `socket.getfqdn()` first (issue 7558)
- Add minion ID caching (issue 7558)
- Fix salt-key race condition (issue 7304)
- Add `--include-all` flag to salt-key (issue 7399)
- Fix custom grains in pillar (part of issue 5716, issue 6083)
- Fix race condition in salt-key (issue 7304)
- Fix regression in minion ID guessing, prioritize `socket.getfqdn()` (issue 7558)
- Cache minion ID on first guess (issue 7558)
- Allow trailing slash in `file.directory` state
- Fix reporting of file_roots in pillar return (issue 5449 and issue 5951)
- Remove pillar matching for mine.get (issue 7197)
- Sanitize args for multiple execution modules
- Fix yumpkg mod_repo functions to filter hidden args (issue 7656)
- Fix conflicting IDs in state includes (issue 7526)
- Fix mysql_grants.absent string formatting issue (issue 7827)
- Fix postgres.version so it won't return None (issue 7695)
- Fix for trailing slashes in mount.mounted state
- Fix rogue AttributErrors in the outputter system (issue 7845)
- Fix for incorrect ssh key encodings resulting in incorrect key added (issue 7718)

- Fix for pillar/grains naming regression in python renderer (issue 7693)
- Fix args/kwargs handling in the scheduler (issue 7422)
- Fix logfile handling for *file://*, *tcp://* and *udp://* (issue 7754)
- Fix error handling in config file parsing (issue 6714)
- Fix RVM using sudo when running as non-root user (issue 2193)
- Fix client ACL and underlying logging bugs (issue 7706)
- Fix scheduler bug with returner (issue 7367)
- Fix user management bug related to default groups (issue 7690)
- Fix various salt-ssh bugs (issue 7528)
- Many various documentation fixes

120.28 Salt 0.17.2 Release Notes

release 2013-11-14

Version 0.17.2 is another bugfix release for *0.17.0*. The changes include:

- Add ability to delete key with grains.delval (issue 7872)
- Fix possible state compiler stack trace (issue 5767)
- Fix architecture regression in yumpkg (issue 7813)
- Use correct `ps` on Debian to prevent truncating (issue 5646)
- Fix grains targeting for new grains (issue 5737)
- Fix bug with merging in git_pillar (issue 6992)
- Fix print_jobs duplicate results
- Fix apt version specification for pkg.install
- Fix possible KeyError from ext_job_cache missing option
- Fix auto_order for `- names` states (issue 7649)
- Fix regression in new gitfs installs (directory not found error)
- Fix escape pipe issue on Windows for file.recurse (issue 7967)
- Fix fileclient in case of master restart (issue 7987)
- Try to output warning if CLI command malformed (issue 6538)
- Fix `--out=quiet` to actually be quiet (issue 8000)
- Fix for state.sls in salt-ssh (issue 7991)
- Fix for MySQL grants ordering issue (issue 5817)
- Fix traceback for certain missing CLI args (issue 8016)
- Add ability to disable lspci queries on master (issue 4906)
- Fail if sls defined in topfile does not exist (issue 5998)
- Add ability to downgrade MySQL grants (issue 6606)

- Fix ssh_auth.absent traceback (issue 8043)
- Add upstart detection for Debian/Raspbian (issue 8039)
- Fix ID-related issues (issue 8052, issue 8050, and others)
- Fix for jinja rendering issues (issue 8066 and issue 8079)
- Fix argument parsing in salt-ssh (issue 7928)
- Fix some GPU detection instances (issue 6945)
- Fix bug preventing includes from other environments in SLS files
- Fix for kwargs with dashes (issue 8102)
- Fix salt.utils.which for windows '.exe' (issue 7904)
- Fix apache.adduser without apachectl (issue 8123)
- Fix issue with evaluating `test` kwarg in states (issue 7788)
- Fix regression in `salt.client.Caller()` (issue 8078)
- Fix apt-key silent failure
- Fix bug where cmd.script would try to run even if caching failed (issue 7601)
- Fix apt `pkg.latest` regression (issue 8067)
- Fix for mine data not being updated (issue 8144)
- Fix for noarch packages in yum
- Fix a Xen detection edge case (issue 7839)
- Fix windows `__opts__` dictionary persistence (issue 7714)
- Fix version generation for when it's part of another git repo (issue 8090)
- Fix _handle_iorder stacktrace so that the real syntax error is shown (issue 8114 and issue 7905)
- Fix `git.latest` state when a commit SHA is used (issue 8163)
- Fix various small bugs in yumpkg.py (issue 8201)
- Fix for specifying identify file in git.latest (issue 8094)
- Fix for `--output-file` CLI arg (issue 8205)
- Add ability to specify shutdown time for system.shutdown (issue 7833)
- Fix for salt version using non-salt git repo info (issue 8266)
- Add additional hints at impact of `pkgrepo` states when `test=True` (issue 8247)
- Fix for salt-ssh files not being owned by root (issue 8216)
- Fix retry logic and error handling in fileserver (related to issue 7755)
- Fix file.replace with `test=True` (issue 8279)
- Add flag for limiting file traversal in fileserver (issue 6928)
- Fix for extra mine processes (issue 5729)
- Fix for unloading custom modules (issue 7691)
- Fix for salt-ssh opts (issue 8005 and issue 8271)
- Fix compound matcher for grains (issue 7944)

- Improve error reporting in ebuild module (related to issue 5393)
- Add `dir_mode` to `file.managed` (issue 7860)
- Improve traceroute support for FreeBSD and OS X (issue 4927)
- Fix for matching minions under syndics (issue 7671)
- Improve exception handling for missing ID (issue 8259)
- Fix grain mismatch for ScientificLinux (issue 8338)
- Add configuration option for minion_id_caching
- Fix open mode auth errors (issue 8402)

120.29 Salt 0.17.3 Release Notes

release 2013-12-08

Note: 0.17.3 had some regressions which were promptly fixed in the 0.17.4 release. Please use 0.17.4 instead.

Version 0.17.3 is another bugfix release for *0.17.0*. The changes include:

- Fix some jinja render errors (issue 8418)
- Fix `file.replace` state changing file ownership (issue 8399)
- Fix state ordering with the PyDSL renderer (issue 8446)
- Fix for new npm version (issue 8517)
- Fix for pip state requiring `name` even with requirements file (issue 8519)
- Fix yum logging to open terminals (issue 3855)
- Add sane maxrunning defaults for scheduler (issue 8563)
- Fix states duplicate key detection (issue 8053)
- Fix SUSE patch level reporting (issue 8428)
- Fix managed file creation umask (issue 8590)
- Fix logstash exception (issue 8635)
- Improve argument exception handling for salt command (issue 8016)
- Fix pecl success reporting (issue 8750)
- Fix launchctl module exceptions (issue 8759)
- Fix argument order in pw_user module
- Add warnings for failing grains (issue 8690)
- Fix hgfs problems caused by connections left open (issue 8811 and issue 8810)
- Add Debian iptables default for iptables-persistent package (issue 8889)
- Fix installation of packages with dots in pkg name (issue 8614)
- Fix noarch package installation on CentOS 6 (issue 8945)
- Fix portage_config.enforce_nice_config (issue 8252)
- Fix salt.util.copyfile umask usage (issue 8590)

- Fix rescheduling of failed jobs (issue 8941)
- Fix pkg on Amazon Linux (uses yumpkg5 now) (issue 8226)
- Fix conflicting options in postgres module (issue 8717)
- Fix ps modules for psutil >= 0.3.0 (issue 7432)
- Fix postgres module to return False on failure (issue 8778)
- Fix argument passing for args with pound signs (issue 8585)
- Fix pid of salt CLi command showing in status.pid output (issue 8720)
- Fix rvm to run gem as the correct user (issue 8951)
- Fix namespace issue in win_file module (issue 9060)
- Fix masterless state paths on windows (issue 9021)
- Fix timeout option in master config (issue 9040)

120.30 Salt 0.17.4 Release Notes

release 2013-12-10

Version 0.17.4 is another bugfix release for *0.17.0*. The changes include:

- Fix some jinja render errors (issue 8418)
- Fix `file.replace` state changing file ownership (issue 8399)
- Fix state ordering with the PyDSL renderer (issue 8446)
- Fix for new npm version (issue 8517)
- Fix for pip state requiring `name` even with requirements file (issue 8519)
- Fix yum logging to open terminals (issue 3855)
- Add sane maxrunning defaults for scheduler (issue 8563)
- Fix states duplicate key detection (issue 8053)
- Fix SUSE patch level reporting (issue 8428)
- Fix managed file creation umask (issue 8590)
- Fix logstash exception (issue 8635)
- Improve argument exception handling for salt command (issue 8016)
- Fix pecl success reporting (issue 8750)
- Fix launchctl module exceptions (issue 8759)
- Fix argument order in pw_user module
- Add warnings for failing grains (issue 8690)
- Fix hgfs problems caused by connections left open (issue 8811 and issue 8810)
- Add Debian iptables default for iptables-persistent package (issue 8889)
- Fix installation of packages with dots in pkg name (issue 8614)
- Fix noarch package installation on CentOS 6 (issue 8945)

- Fix portage_config.enforce_nice_config (issue 8252)
- Fix salt.util.copyfile umask usage (issue 8590)
- Fix rescheduling of failed jobs (issue 8941)
- Fix pkg on Amazon Linux (uses yumpkg5 now) (issue 8226)
- Fix conflicting options in postgres module (issue 8717)
- Fix ps modules for psutil >= 0.3.0 (issue 7432)
- Fix postgres module to return False on failure (issue 8778)
- Fix argument passing for args with pound signs (issue 8585)
- Fix pid of salt CLi command showing in status.pid output (issue 8720)
- Fix rvm to run gem as the correct user (issue 8951)
- Fix namespace issue in win_file module (issue 9060)
- Fix masterless state paths on windows (issue 9021)
- Fix timeout option in master config (issue 9040)

120.31 Salt 0.6.0 release notes

The Salt remote execution manager has reached initial functionality! Salt is a management application which can be used to execute commands on remote sets of servers.

The whole idea behind Salt is to create a system where a group of servers can be remotely controlled from a single master, not only can commands be executed on remote systems, but salt can also be used to gather information about your server environment.

Unlike similar systems, like Func and MCollective, Salt is extremely simple to setup and use, the entire application is contained in a single package, and the master and minion daemons require no running dependencies in the way that Func requires Certmaster and MCollective requires activeMQ.

Salt also manages authentication and encryption. Rather than using SSL for encryption, salt manages encryption on a payload level, so the data sent across the network is encrypted with fast AES encryption, and authentication uses RSA keys. This means that Salt is fast, secure, and very efficient.

Messaging in Salt is executed with ZeroMQ, so the message passing interface is built into salt and does not require an external ZeroMQ server. This also adds speed to Salt since there is no additional bloat on the networking layer, and ZeroMQ has already proven itself as a very fast networking system.

The remote execution in Salt is "Lazy Execution", in that once the command is sent the requesting network connection is closed. This makes it easier to detach the execution from the calling process on the master, it also means that replies are cached, so that information gathered from historic commands can be queried in the future.

Salt also allows users to make execution modules in Python. Writers of these modules should also be pleased to know that they have access to the impressive information gathered from PuppetLabs' Facter application, making Salt module more flexible. In the future I hope to also allow Salt to group servers based on Facter information as well.

All in all Salt is fast, efficient and clean, can be used from a simple command line client or through an API, uses message queue technology to make network execution extremely fast, and encryption is handled in a very fast and efficient manner. Salt is also VERY easy to use and VERY easy to extend.

You can find the source code for Salt on my GitHub page, I have also set up a few wiki pages explaining how to use and set up Salt. If you are using Arch Linux there is a package available in the Arch Linux AUR.

Salt 0.6.0 Source: https://github.com/downloads/saltstack/salt/salt-0.6.0.tar.gz

GitHub page: https://github.com/saltstack/salt

Wiki: https://github.com/saltstack/salt/wiki

Arch Linux Package: https://aur.archlinux.org/packages.php?ID=47512

I am very open to contributions, for instance I need packages for more Linux distributions as well as BSD packages and testers.

Give Salt a try, this is the initial release and is not a 1.0 quality release, but it has been working well for me! I am eager to get your feedback!

120.32 Salt 0.7.0 release notes

I am pleased to announce the release of Salt 0.7.0!

This release marks what is the first stable release of salt, 0.7.0 should be suitable for general use.

0.7.0 Brings the following new features to Salt:

- Integration with Facter data from puppet labs
- Allow for matching minions from the salt client via Facter information
- Minion job threading, many jobs can be executed from the master at once
- Preview of master clustering support - Still experimental
- Introduce new minion modules for stats, virtualization, service management and more
- Add extensive logging to the master and minion daemons
- Add sys.reload_functions for dynamic function reloading
- Greatly improve authentication
- Introduce the saltkey command for managing public keys
- Begin backend development preparatory to introducing butter
- Addition of man pages for the core commands
- Extended and cleaned configuration

0.7.0 Fixes the following major bugs:

- Fix crash in minions when matching failed
- Fix configuration file lookups for the local client
- Repair communication bugs in encryption
- Numerous fixes in the minion modules

The next release of Salt should see the following features:

- Stabilize the cluster support
- Introduce a remote client for salt command tiers
- salt-ftp system for distributed file copies
- Initial support for “butter”

Coming up next is a higher level management framework for salt called Butter. I want salt to stay as a simple and effective communication framework, and allow for more complicated executions to be managed via Butter.

Right now Butter is being developed to act as a cloud controller using salt as the communication layer, but features like system monitoring and advanced configuration control (a puppet manager) are also in the pipe.

Special thanks to Joseph Hall for the status and network modules, and thanks to Matthias Teege for tracking down some configuration bugs!

Salt can be downloaded from the following locations;

Source Tarball:

https://github.com/downloads/saltstack/salt/salt-0.7.0.tar.gz

Arch Linux Package:

https://aur.archlinux.org/packages.php?ID=47512

Please enjoy the latest Salt release!

120.33 Salt 0.8.0 release notes

Salt 0.8.0 is ready for general consumption! The source tarball is available on GitHub for download:

https://github.com/downloads/saltstack/salt/salt-0.8.0.tar.gz

A lot of work has gone into salt since the last release just 2 weeks ago, and salt has improved a great deal. A swath of new features are here along with performance and threading improvements!

The main new features of salt 0.8.0 are:

Salt-cp

Cython minion modules

Dynamic returners

Faster return handling

Lowered required Python version to 2.6

Advanced minion threading

Configurable minion modules

120.33.1 Salt-cp -

The salt-cp command introduces the ability to copy simple files via salt to targeted servers. Using salt-cp is very simple, just call salt-cp with a target specification, the source file(s) and where to copy the files on the minions. For instance:

# salt-cp '*' /etc/hosts /etc/hosts

Will copy the local /etc/hosts file to all of the minions.

Salt-cp is very young, in the future more advanced features will be added, and the functionality will much more closely resemble the cp command.

120.33.2 Cython minion modules -

Cython is an amazing tool used to compile Python modules down to c. This is arguably the fastest way to run Python code, and since pyzmq requires cython, adding support to salt for cython adds no new dependencies.

Cython minion modules allow minion modules to be written in cython and therefore executed in compiled c. Simply write the salt module in cython and use the file extension ".pyx" and the minion module will be compiled when the minion is started. An example cython module is included in the main distribution called cytest.pyx:

https://github.com/saltstack/salt/blob/develop/salt/modules/cytest.pyx

120.33.3 Dynamic Returners -

By default salt returns command data back to the salt master, but now salt can return command data to any system. This is enabled via the new returners modules feature for salt. The returners modules take the return data and sends it to a specific module. The returner modules work like minion modules, so any returner can be added to the minions.

This means that a custom data returner can be added to communicate the return data so anything from MySQL, Redis, MongoDB and more!

There are 2 simple stock returners in the returners directory:

https://github.com/saltstack/salt/blob/develop/salt/returners

The documentation on writing returners will be added to the wiki shortly, and returners can be written in pure Python, or in cython.

120.33.4 Configurable Minion Modules -

Minion modules may need to be configured, now the options passed to the minion configuration file can be accessed inside of the minion modules via the __opt__ dict.

Information on how to use this simple addition has been added to the wiki: https://github.com/thatch45/salt/wiki/Writing-Salt-Modules

The test module has an example of using the __opts__ dict, and how to set default options:

https://github.com/saltstack/salt/blob/develop/salt/modules/test.py

120.33.5 Advanced Minion Threading:

In 0.7.0 the minion would block after receiving a command from the master, now the minion will spawn a thread or multiprocess. By default Python threads are used because for general use they have proved to be faster, but the minion can now be configured to use the Python multiprocessing module instead. Using multiprocessing will cause executions that are CPU bound or would otherwise exploit the negative aspects of the Python GIL to run faster and more reliably, but simple calls will still be faster with Python threading. The configuration option can be found in the minion configuration file:

https://github.com/saltstack/salt/blob/develop/conf/minion

Lowered Supported Python to 2.6 -

The requirement for Python 2.7 has been removed to support Python 2.6. I have received requests to take the minimum Python version back to 2.4, but unfortunately this will not be possible, since the ZeroMQ Python bindings do not support Python 2.4.

Salt 0.8.0 is a very major update, it also changes the network protocol slightly which makes communication with older salt daemons impossible, your master and minions need to be upgraded together! I could use some help bringing salt to the people! Right now I only have packages for Arch Linux, Fedora 14 and Gentoo. We need packages for Debian and people willing to help test on more platforms. We also need help writing more minion modules and returner modules. If you want to contribute to salt please hop on the mailing list and send in patches, make a fork on GitHub and send in pull requests! If you want to help but are not sure where you can, please email me directly or post tot he mailing list!

I hope you enjoy salt, while it is not yet 1.0 salt is completely viable and usable!

-Thomas S. Hatch

120.34 Salt 0.8.7 release notes

It has been a month since salt 0.8.0, and it has been a long month! But Salt is still coming along strong. 0.8.7 has a lot of changes and a lot of updates. This update makes Salt's ZeroMQ back end better, strips Facter from the dependencies, and introduces interfaces to handle more capabilities.

Many of the major updates are in the background, but the changes should shine through to the surface. A number of the new features are still a little thin, but the back end to support expansion is in place.

I also recently gave a presentation to the Utah Python users group in Salt Lake City, the slides from this presentation are available here: https://github.com/downloads/saltstack/salt/Salt.pdf

The video from this presentation will be available shortly.

The major new features and changes in Salt 0.8.7 are:

- Revamp ZeroMQ topology on the master for better scalability
- State enforcement
- Dynamic state enforcement managers
- Extract the module loader into salt.loader
- Make Job ids more granular
- Replace Facter functionality with the new salt grains interface
- Support for "virtual" salt modules
- Introduce the salt-call command
- Better debugging for minion modules

The new ZeroMQ topology allows for better scalability, this will be required by the need to execute massive file transfers to multiple machines in parallel and state management. The new ZeroMQ topology is available in the aforementioned presentation.

0.8.7 introduces the capability to declare states, this is similar to the capabilities of Puppet. States in salt are declared via state data structures. This system is very young, but the core feature set is available. Salt states work around rendering files which represent Salt high data. More on the Salt state system will be documented in the near future.

The system for loading salt modules has been pulled out of the minion class to be a standalone module, this has enabled more dynamic loading of Salt modules and enables many of the updates in 0.8.7 –

https://github.com/saltstack/salt/blob/develop/salt/loader.py

Salt Job ids are now microsecond precise, this was needed to repair a race condition unveiled by the speed improvements in the new ZeroMQ topology.

The new grains interface replaces the functionality of Facter, the idea behind grains differs from Facter in that the grains are only used for static system data, dynamic data needs to be derived from a call to a salt module. This makes grains much faster to use, since the grains data is generated when the minion starts.

Virtual salt modules allows for a salt module to be presented as something other than its module name. The idea here is that based on information from the minion decisions about which module should be presented can be made. The best example is the pacman module. The pacman module will only load on Arch Linux minions, and will be called pkg. Similarly the yum module will be presented as pkg when the minion starts on a Fedora/RedHat system.

The new salt-call command allows for minion modules to be executed from the minion. This means that on the minion a salt module can be executed, this is a great tool for testing Salt modules. The salt-call command can also be used to view the grains data.

In previous releases when a minion module threw an exception very little data was returned to the master. Now the stack trace from the failure is returned making debugging of minion modules MUCH easier.

Salt is nearing the goal of 1.0, where the core feature set and capability is complete!

Salt 0.8.7 can be downloaded from GitHub here: https://github.com/downloads/saltstack/salt/salt-0.8.7.tar.gz

-Thomas S Hatch

120.35 Salt 0.8.8 release notes

Salt 0.8.8 is here! This release adds a great deal of code and some serious new features. The latest release can be downloaded here: https://github.com/downloads/saltstack/salt/salt-0.8.8.tar.gz

Improved Documentation has been set up for salt using sphinx thanks to the efforts of Seth House. This new documentation system will act as the back end to the salt website which is still under heavy development. The new sphinx documentation system has also been used to greatly clean up the salt manpages. The salt 7 manpage in particular now contains extensive information which was previously only in the wiki. The new documentation can be found at: http://thatch45.github.com/salt-www/ We still have a lot to add, and when the domain is set up I will post another announcement.

More additions have been made to the ZeroMQ setup, particularly in the realm of file transfers. Salt 0.8.8 introduces a built in, stateless, encrypted file server which allows salt minions to download files from the salt master using the same encryption system used for all other salt communications. The main motivation for the salt file server has been to facilitate the new salt state system.

Much of the salt code has been cleaned up and a new cleaner logging system has been introduced thanks to the efforts of Pedro Algarvio. These additions will allow for much more flexible logging to be executed by salt, and fixed a great deal of my poor spelling in the salt docstrings! Pedro Algarvio has also cleaned up the API, making it easier to embed salt into another application.

The biggest addition to salt found in 0.8.8 is the new state system. The salt module system has received a new front end which allows salt to be used as a configuration management system. The configuration management system allows for system configuration to be defined in data structures. The configuration management system, or as it is called in salt, the "salt state system" supports many of the features found in other configuration managers, but allows for system states to be written in a far simpler format, executes at blazing speeds, and operates via the salt minion matching system. The state system also operates within the normal scope of salt, and requires no additional configuration to use.

The salt state system can enforce the following states with many more to come: Packages Files Services Executing commands Hosts

The system used to define the salt states is based on a data structure, the data structure used to define the salt states has been made to be as easy to use as possible. The data structure is defined by default using a YAML file rendered via a Jinja template. This means that the state definition language supports all of the data structures that YAML supports, and all of the programming constructs and logic that Jinja supports. If the user does not like YAML or Jinja the states can be defined in yaml-mako, json-jinja, or json-mako. The system used to render the states is completely dynamic, and any rendering system can be added to the capabilities of Salt, this means that a rendering system that renders XML data in a cheetah template, or whatever you can imagine, can be easily added to the capabilities of salt.

The salt state system also supports isolated environments, as well as matching code from several environments to a single salt minion.

The feature base for Salt has grown quite a bit since my last serious documentation push. As we approach 0.9.0 the goals are becoming very clear, and the documentation needs a lot of work. The main goals for 0.9.0 are to further refine the state system, fix any bugs we find, get Salt running on as many platforms as we can, and get the documentation filled out. There is a lot more to come as Salt moves forward to encapsulate a much larger scope, while maintaining supreme usability and simplicity.

If you would like a more complete overview of Salt please watch the Salt presentation: Flash Video: http://blip.tv/thomas-s-hatch/salt-0-8-7-presentation-5180182 OGV Video Download: http://blip.tv/file/get/Thatch45-Salt087Presentation416.ogv Slides: https://github.com/downloads/saltstack/salt/Salt.pdf

-Thomas S Hatch

120.36 Salt 0.8.9 Release Notes

Salt 0.8.9 has finally arrived! Unfortunately this is much later than I had hoped to release 0.8.9, life has been very crazy over the last month. But despite challenges, Salt has moved forward!

This release, as expected, adds few new features and many refinements. One of the most exciting aspect of this release is that the development community for salt has grown a great deal and much of the code is from contributors.

Also, I have filled out the documentation a great deal. So information on States is properly documented, and much of the documentation that was out of date has been filled in.

120.36.1 Download!

The Salt source can be downloaded from the salt GitHub site:

https://github.com/downloads/saltstack/salt/salt-0.8.9.tar.gz

Or from PyPI:

http://pypi.python.org/packages/source/s/salt/salt-0.8.9.tar.gz

Here s the md5sum:

7d5aca4633bc22f59045f59e82f43b56

For instructions on how to set up Salt please see the *Installation* instructions.

120.36.2 New Features

Salt Run

A big feature is the addition of Salt run, the `salt-run` command allows for master side execution modules to be made that gather specific information or execute custom routines from the master.

Documentation for salt-run can be found here:

http://saltstack.org/ref/runners.html

Refined Outputters

One problem often complained about in salt was the fact that the output was so messy. Thanks to help from Jeff Schroeder a cleaner interface for the command output for the Salt CLI has been made. This new interface makes adding new printout formats easy and additions to the capabilities of minion modules makes it possible to set the printout mode or `outputter` for functions in minion modules.

Cross Calling Salt Modules

Salt modules can now call each other, the `__salt__` dict has been added to the predefined references in minion modules. This new feature is documented in the modules documentation:

http://saltstack.org/ref/modules/index.html

Watch Option Added to Salt State System

Now in Salt states you can set the watch option, this will allow watch enabled states to change based on a change in the other defined states. This is similar to subscribe and notify statements in puppet.

Root Dir Option

Travis Cline has added the ability to define the option `root_dir` which allows the salt minion to operate in a subdir. This is a strong move in supporting the minion running as an unprivileged user

Config Files Defined in Variables

Thanks again to Travis Cline, the master and minion configuration file locations can be defined in environment variables now.

120.36.3 New Modules

Quite a few new modules, states, returners and runners have been made.

New Minion Modules

apt

Support for apt-get has been added, this adds greatly improved Debian and Ubuntu support to Salt!

useradd and groupadd

Support for manipulating users and groups on Unix-like systems.

moosefs

Initial support for reporting on aspects of the distributed file system, MooseFS. For more information on MooseFS please see: http://moosefs.org

Thanks to Joseph Hall for his work on MooseFS support.

mount

Manage mounts and the fstab.

puppet

Execute puppet on remote systems.

shadow

Manipulate and manage the user password file.

ssh

Interact with ssh keys.

New States

user and group

Support for managing users and groups in Salt States.

mount

Enforce mounts and the fstab.

New Returners

mongo_return

Send the return information to a MongoDB server.

New Runners

manage

Display minions that are up or down.

120.37 Salt 0.9.0 Release Notes

release 2011-08-27

Salt 0.9.0 is here. This is an exciting release, 0.9.0 includes the new network topology features allowing peer salt commands and masters of masters via the syndic interface.

0.9.0 also introduces many more modules, improvements to the API and improvements to the ZeroMQ systems.

120.37.1 Download!

The Salt source can be downloaded from the salt GitHub site:

https://github.com/downloads/saltstack/salt/salt-0.9.0.tar.gz

Or from PyPI:

http://pypi.python.org/packages/source/s/salt/salt-0.9.0.tar.gz

Here is the md5sum:

9a925da04981e65a0f237f2e77ddab37

For instructions on how to set up Salt please see the *Installation* instructions.

120.37.2 New Features

Salt Syndic

The new *Syndic interface* allows a master to be commanded via another higher level salt master. This is a powerful solution allowing a master control structure to exist, allowing salt to scale to much larger levels then before.

Peer Communication

0.9.0 introduces the capability for a minion to call a publication on the master and receive the return from another set of minions. This allows salt to act as a communication channel between minions and as a general infrastructure message bus.

Peer communication is turned off by default but can be enabled via the `peer` option in the master configuration file. Documentation on the new *Peer interface*.

Easily Extensible API

The minion and master classes have been redesigned to allow for specialized minion and master servers to be easily created. An example on how this is done for the master can be found in the `master.py` salt module:

https://github.com/saltstack/salt/blob/develop/salt/master.py

The `Master` class extends the `SMaster` class and set up the main master server.

The minion functions can now also be easily added to another application via the `SMinion` class, this class can be found in the `minion.py` module:

https://github.com/saltstack/salt/blob/develop/salt/minion.py

Cleaner Key Management

This release changes some of the key naming to allow for multiple master keys to be held based on the type of minion gathering the master key.

The -d option has also been added to the salt-key command allowing for easy removal of accepted public keys.

The –gen-keys option is now available as well for salt-key, this allows for a salt specific RSA key pair to be easily generated from the command line.

Improved 0MQ Master Workers

The 0MQ worker system has been further refined to be faster and more robust. This new system has been able to handle a much larger load than the previous setup. The new system uses the IPC protocol in 0MQ instead of TCP.

120.37.3 New Modules

Quite a few new modules have been made.

New Minion Modules

apache

Work directly with apache servers, great for managing balanced web servers

cron

Read out the contents of a systems crontabs

mdadm

Module to manage raid devices in Linux, appears as the `raid` module

mysql

Gather simple data from MySQL databases

ps

Extensive utilities for managing processes

publish

Used by the peer interface to allow minions to make publications

120.38 Salt 0.9.1 Release Notes

release 2011-08-29

120.39 Salt 0.9.2 Release Notes

release 2011-09-17

Salt 0.9.2 has arrived! 0.9.2 is primarily a bugfix release, the exciting component in 0.9.2 is greatly improved support for salt states. All of the salt states interfaces have been more thoroughly tested and the new salt-states git repo is growing with example of how to use states.

This release introduces salt states for early developers and testers to start helping us clean up the states interface and make it ready for the world!

0.9.2 also fixes a number of bugs found on Python 2.6.

120.39.1 Download!

The Salt source can be downloaded from the salt GitHub site:

https://github.com/downloads/saltstack/salt/salt-0.9.2.tar.gz

Or from PyPI:

http://pypi.python.org/packages/source/s/salt/salt-0.9.2.tar.gz

For instructions on how to set up Salt please see the *Installation* instructions.

120.39.2 New Features

Salt-Call Additions

The salt-call command has received an overhaul, it now hooks into the outputter system so command output looks clean, and the logging system has been hooked into salt-call, so the -l option allows the logging output from salt minion functions to be displayed.

The end result is that the salt-call command can execute the state system and return clean output:

```
# salt-call state.highstate
```

State System Fixes

The state system has been tested and better refined. As of this release the state system is ready for early testers to start playing with. If you are interested in working with the state system please check out the (still very small) salt-states GitHub repo:

https://github.com/thatch45/salt-states

This git repo is the active development branch for determining how a clean salt-state database should look and act. Since the salt state system is still very young a lot of help is still needed here. Please fork the salt-states repo and help us develop a truly large and scalable system for configuration management!

120.39.3 Notable Bug Fixes

Python 2.6 String Formatting

Python 2.6 does not support format strings without an index identifier, all of them have been repaired.

Cython Loading Disabled by Default

Cython loading requires a development tool chain to be installed on the minion, requiring this by default can cause problems for most Salt deployments. If Cython auto loading is desired it will need to be turned on in the minion config.

120.40 Salt 0.9.3 Release Notes

release 2011-11-05

Salt 0.9.3 is finally arrived. This is another big step forward for Salt, new features range from proper FreeBSD support to fixing issues seen when attaching a minion to a master over the Internet.

The biggest improvements in 0.9.3 though can be found in the state system, it has progressed from something ready for early testers to a system ready to compete with platforms such as Puppet and Chef. The backbone of the state system has been greatly refined and many new features are available.

120.40.1 Download!

The Salt source can be downloaded from the salt GitHub site:

https://github.com/downloads/saltstack/salt/salt-0.9.3.tar.gz

Or from PyPI:

http://pypi.python.org/packages/source/s/salt/salt-0.9.3.tar.gz

For instructions on how to set up Salt please see the *Installation* instructions.

120.40.2 New Features

WAN Support

Recently more people have been testing Salt minions connecting to Salt Masters over the Internet. It was found that Minions would commonly loose their connection to the master when working over the internet. The minions can now detect if the connection has been lost and reconnect to the master, making WAN connections much more reliable.

State System Fixes

Substantial testing has gone into the state system and it is ready for real world usage. A great deal has been added to the documentation for states and the modules and functions available to states have been cleanly documented.

A number of State System bugs have also been founds and repaired, the output from the state system has also been refined to be extremely clear and concise.

Error reporting has also been introduced, issues found in sls files will now be clearly reported when executing Salt States.

Extend Declaration

The Salt States have also gained the `extend` declaration. This declaration allows for states to be cleanly modified in a post environment. Simply said, if there is an apache.sls file that declares the apache service, then another sls can include apache and then extend it:

```
include:
  - apache

extend:
  apache:
    service:
      - require:
        - pkg: mod_python

mod_python:
  pkg:
    - installed
```

The notable behavior with the extend functionality is that it literally extends or overwrites a declaration set up in another sls module. This means that Salt will behave as though the modifications were made directly to the apache sls. This ensures that the apache service in this example is directly tied to all requirements.

Highstate Structure Specification

This release comes with a clear specification of the Highstate data structure that is used to declare Salt States. This specification explains everything that can be declared in the Salt SLS modules.

The specification is extremely simple, and illustrates how Salt has been able to fulfill the requirements of a central configuration manager within a simple and easy to understand format and specification.

SheBang Renderer Switch

It came to our attention that having many renderers means that there may be a situation where more than one State Renderer should be available within a single State Tree.

The method chosen to accomplish this was something already familiar to developers and systems administrators, a SheBang. The Python State Renderer displays this new capability.

Python State Renderer

Until now Salt States could only be declared in yaml or json using Jinja or Mako. A new, very powerful, renderer has been added, making it possible to write Salt States in pure Python:

```
#!py

def run():
    '''
    Install the python-mako package
    '''
    return {'include': ['python'],
            'python-mako': {'pkg': ['installed']}}
```

This renderer is used by making a run function that returns the Highstate data structure. Any capabilities of Python can be used in pure Python sls modules.

This example of a pure Python sls module is the same as this example in yaml:

```
include:
  - python

python-mako:
  pkg:
    - installed
```

FreeBSD Support

Additional support has been added for FreeBSD, this is Salt's first branch out of the Linux world and proves the viability of Salt on non-Linux platforms.

Salt remote execution already worked on FreeBSD, and should work without issue on any Unix-like platform. But this support comes in the form of package management and user support, so Salt States also work on FreeBSD now.

The new freebsdpkg module provides package management support for FreeBSD and the new pw_user and pw_group provide user and group management.

120.40.3 Module and State Additions

Cron Support

Support for managing the system crontab has been added, declaring a cron state can be done easily:

```
date > /tmp/datestamp:
  cron:
    - present
    - user: fred
    - minute: 5
    - hour: 3
```

File State Additions

The file state has been given a number of new features, primarily the directory, recurse, symlink and absent functions.

file.directory Make sure that a directory exists and has the right permissions.

```
/srv/foo:
  file:
    - directory
    - user: root
    - group: root
    - mode: 1755
```

file.symlink Make a symlink.

```
/var/lib/www:
  file:
    - symlink
    - target: /srv/www
    - force: True
```

file.recurse The recurse state function will recursively download a directory on the master file server and place it on the minion. Any change in the files on the master will be pushed to the minion. The recurse function is very powerful and has been tested by pushing out the full Linux kernel source.

```
/opt/code:
  file:
    - recurse
    - source: salt://linux
```

file.absent Make sure that the file is not on the system, recursively deletes directories, files and symlinks.

```
/etc/httpd/conf.d/somebogusfile.conf:
  file:
    - absent
```

Sysctl Module and State

The sysctl module and state allows for sysctl components in the kernel to be managed easily. the sysctl module contains the following functions:

sysctl.show Return a list of sysctl parameters for this minion

sysctl.get Return a single sysctl parameter for this minion

sysctl.assign Assign a single sysctl parameter for this minion

sysctl.persist Assign and persist a simple sysctl parameter for this minion

The sysctl state allows for sysctl parameters to be assigned:

```
vm.swappiness:
  sysctl:
    - present
    - value: 20
```

Kernel Module Management

A module for managing Linux kernel modules has been added. The new functions are as follows:

kmod.available Return a list of all available kernel modules

kmod.check_available Check to see if the specified kernel module is available

kmod.lsmod Return a dict containing information about currently loaded modules

kmod.load Load the specified kernel module

kmod.remove Unload the specified kernel module

The kmod state can enforce modules be either present or absent:

```
kvm_intel:
  kmod:
    - present
```

Ssh Authorized Keys

The ssh_auth state can distribute ssh authorized keys out to minions. Ssh authorized keys can be present or absent.

```
AAAAB3NzaC1kc3MAAACBAL0sQ9fJ5bYTEyYvlRBsJdDOo49CNfhlWHWXQRqul6rwL4KIuPrhY7hBw0tV7UNC7J9IZRNO4iGod9C+
  ssh_auth:
    - present
    - user: frank
    - enc: dsa
    - comment: 'Frank's key'
```

120.41 Salt 0.9.4 Release Notes

release 2011-11-27

Salt 0.9.4 has arrived. This is a critical update that repairs a number of key bugs found in 0.9.3. But this update is not without feature additions as well! 0.9.4 adds support for Gentoo portage to the pkg module and state system. Also there are 2 major new state additions, the failhard option and the ability to set up finite state ordering with the `order` option.

This release also sees our largest increase in community contributions. These contributors have and continue to be the life blood of the Salt project, and the team continues to grow. I want to put out a big thanks to our new and existing contributors.

120.41.1 Download!

The Salt source can be downloaded from the salt GitHub site:

https://github.com/downloads/saltstack/salt/salt-0.9.4.tar.gz

Or from PyPI:

http://pypi.python.org/packages/source/s/salt/salt-0.9.4.tar.gz

For instructions on how to set up Salt please see the *Installation* instructions.

120.41.2 New Features

Failhard State Option

Normally, when a state fails Salt continues to execute the remainder of the defined states and will only refuse to execute states that require the failed state.

But the situation may exist, where you would want all state execution to stop if a single state execution fails. The capability to do this is called `failing hard`.

State Level Failhard

A single state can have a failhard set, this means that if this individual state fails that all state execution will immediately stop. This is a great thing to do if there is a state that sets up a critical config file and setting a require for each state that reads the config would be cumbersome. A good example of this would be setting up a package manager early on:

```
/etc/yum.repos.d/company.repo:
  file:
    - managed
    - source: salt://company/yumrepo.conf
    - user: root
    - group: root
    - mode: 644
    - order: 1
    - failhard: True
```

In this situation, the yum repo is going to be configured before other states, and if it fails to lay down the config file, than no other states will be executed.

Global Failhard

It may be desired to have failhard be applied to every state that is executed, if this is the case, then failhard can be set in the master configuration file. Setting failhard in the master configuration file will result in failing hard when any minion gathering states from the master have a state fail.

This is NOT the default behavior, normally Salt will only fail states that require a failed state.

Using the global failhard is generally not recommended, since it can result in states not being executed or even checked. It can also be confusing to see states failhard if an admin is not actively aware that the failhard has been set.

To use the global failhard set failhard: True in the master configuration

Finite Ordering of State Execution

When creating salt sls files, it is often important to ensure that they run in a specific order. While states will always execute in the same order, that order is not necessarily defined the way you want it.

A few tools exist in Salt to set up the correct state ordering, these tools consist of requisite declarations and order options.

The Order Option

Before using the order option, remember that the majority of state ordering should be done with requisite statements, and that a requisite statement will override an order option.

The order option is used by adding an order number to a state declaration with the option *order*:

```
vim:
  pkg:
    - installed
    - order: 1
```

By adding the order option to *1* this ensures that the vim package will be installed in tandem with any other state declaration set to the order *1*.

Any state declared without an order option will be executed after all states with order options are executed.

But this construct can only handle ordering states from the beginning. Sometimes you may want to send a state to the end of the line, to do this set the order to last:

```
vim:
  pkg:
    - installed
    - order: last
```

Substantial testing has gone into the state system and it is ready for real world usage. A great deal has been added to the documentation for states and the modules and functions available to states have been cleanly documented.

A number of State System bugs have also been founds and repaired, the output from the state system has also been refined to be extremely clear and concise.

Error reporting has also been introduced, issues found in sls files will now be clearly reported when executing Salt States.

Gentoo Support

Additional experimental support has been added for Gentoo. This is found in the contribution from Doug Renn, aka nestegg.

120.42 Salt 0.9.5 Release Notes

release 2012-01-15

Salt 0.9.5 is one of the largest steps forward in the development of Salt.

0.9.5 comes with many milestones, this release has seen the community of developers grow out to an international team of 46 code contributors and has many feature additions, feature enhancements, bug fixes and speed improvements.

> **Warning:** Be sure to *read the upgrade instructions* about the switch to msgpack before upgrading!

120.42.1 Community

Nothing has proven to have more value to the development of Salt that the outstanding community that has been growing at such a great pace around Salt. This has proven not only that Salt has great value, but also the expandability of Salt is as exponential as I originally intended.

0.9.5 has received over 600 additional commits since 0.9.4 with a swath of new committers. The following individuals have contributed to the development of 0.9.5:

- Aaron Bull Schaefer
- Antti Kaihola
- Bas Tichelaar
- Brad Barden
- Brian Wagner
- Byron Clark
- Chris Scheller
- Christer Edwards
- Clint Savage
- Corey Quinn
- David Boucha
- Eivind Uggedal
- Eric Poelke
- Evan Borgstrom
- Jed Glazner
- Jeff Schroeder
- Jeffrey C. Ollie
- Jonas Buckner

- Kent Tenney
- Martin Schnabel
- Maxim Burgerhout
- Mitch Anderson
- Nathaniel Whiteinge
- Seth House
- Thomas S Hatch
- Thomas Schreiber
- Tor Hveem
- lzyeval
- syphernl

This makes 21 new developers since 0.9.4 was released!

To keep up with the growing community follow Salt on Ohloh (http://www.ohloh.net/p/salt), to join the Salt development community, fork Salt on Github, and get coding (https://github.com/saltstack/salt)!

120.42.2 Major Features

SPEED! Pickle to msgpack

For a few months now we have been talking about moving away from Python pickles for network serialization, but a preferred serialization format had not yet been found. After an extensive performance testing period involving everything from JSON to protocol buffers, a clear winner emerged. Message Pack (http://msgpack.org/) proved to not only be the fastest and most compact, but also the most "salt like". Message Pack is simple, and the code involved is very small. The msgpack library for Python has been added directly to Salt.

This move introduces a few changes to Salt. First off, Salt is no longer a "noarch" package, since the msgpack lib is written in C. Salt 0.9.5 will also have compatibility issues with 0.9.4 with the default configuration.

We have gone through great lengths to avoid backwards compatibility issues with Salt, but changing the serialization medium was going to create issues regardless. Salt 0.9.5 is somewhat backwards compatible with earlier minions. A 0.9.5 master can command older minions, but only if the `serial` config value in the master is set to `pickle`. This will tell the master to publish messages in pickle format and will allow the master to receive messages in both msgpack and pickle formats.

Therefore **the suggested methods for upgrading** are either to just upgrade everything at once, or:

1. Upgrade the master to 0.9.5
2. Set `serial` to `pickle` in the master config
3. Upgrade the minions
4. Remove the `serial` option from the master config

Since pickles can be used as a security exploit the ability for a master to accept pickles from minions at all will be removed in a future release.

C Bindings for YAML

All of the YAML rendering is now done with the YAML C bindings. This speeds up all of the sls files when running states.

Experimental Windows Support

David Boucha has worked tirelessly to bring initial support to Salt for Microsoft Windows operating systems. Right now the Salt Minion can run as a native Windows service and accept commands.

In the weeks and months to come Windows will receive the full treatment and will have support for Salt States and more robust support for managing Windows systems. This is a big step forward for Salt to move entirely outside of the Unix world, and proves Salt is a viable cross platform solution. Big Thanks to Dave for his contribution here!

Dynamic Module Distribution

Many Salt users have expressed the desire to have Salt distribute in-house modules, states, renderers, returners, and grains. This support has been added in a number of ways:

Modules via States

Now when salt modules are deployed to a minion via the state system as a file, then the modules will be automatically loaded into the active running minion - no restart required - and into the active running state. So custom state modules can be deployed and used in the same state run.

Modules via Module Environment Directories

Under the file_roots each environment can now have directories that are used to deploy large groups of modules. These directories sync modules at the beginning of a state run on the minion, or can be manually synced via the Salt module `salt.modules.saltutil.sync_all`.

The directories are named:

- `_modules`
- `_states`
- `_grains`
- `_renderers`
- `_returners`

The modules are pushed to their respective scopes on the minions.

Module Reloading

Modules can now be reloaded without restarting the minion, this is done by calling the `salt.modules.sys.reload_modules` function.

But wait, there's more! Now when a salt module of any type is added via states the modules will be automatically reloaded, allowing for modules to be laid down with states and then immediately used.

Finally, all modules are reloaded when modules are dynamically distributed from the salt master.

Enable / Disable Added to Service

A great deal of demand has existed for adding the capability to set services to be started at boot in the service module. This feature also comes with an overhaul of the service modules and initial systemd support.

This means that the `service state` can now accept `- enable: True` to make sure a service is enabled at boot, and `- enable: False` to make sure it is disabled.

Compound Target

A new target type has been added to the lineup, the compound target. In previous versions the desired minions could only be targeted via a single specific target type, but now many target specifications can be declared.

These targets can also be separated by and/or operators, so certain properties can be used to omit a node:

```
salt -C 'webserv* and G@os:Debian or E@db.*' test.ping
```

will match all minions with ids starting with webserv via a glob and minions matching the `os:Debian` grain. Or minions that match the `db.*` regular expression.

Node Groups

Often the convenience of having a predefined group of minions to execute targets on is desired. This can be accomplished with the new nodegroups feature. Nodegroups allow for predefined compound targets to be declared in the master configuration file:

```
nodegroups:
  group1: 'L@foo.domain.com,bar.domain.com,baz.domain.com and bl*.domain.com'
  group2: 'G@os:Debian and foo.domain.com'
```

And then used via the `-N` option:

```
salt -N group1 test.ping
```

Minion Side Data Store

The data module introduces the initial approach into storing persistent data on the minions, specific to the minions. This allows for data to be stored on minions that can be accessed from the master or from the minion.

The Minion datastore is young, and will eventually provide an interface similar to a more mature key/value pair server.

Major Grains Improvement

The Salt grains have been overhauled to include a massive amount of extra data. this includes hardware data, os data and salt specific data.

Salt -Q is Useful Now

In the past the salt query system, which would display the data from recent executions would be displayed in pure Python, and it was unreadable.

0.9.5 has added the outputter system to the `-Q` option, thus enabling the salt query system to return readable output.

120.42.3 Packaging Updates

Huge strides have been made in packaging Salt for distributions. These additions are thanks to our wonderful community where the work to set up packages has proceeded tirelessly.

FreeBSD

Salt on FreeBSD? There a port for that:

http://www.freebsd.org/cgi/cvsweb.cgi/ports/sysutils/salt/pkg-descr

This port was developed and added by Christer Edwards. This also marks the first time Salt has been included in an upstream packaging system!

Fedora and Red Hat Enterprise

Salt packages have been prepared for inclusion in the Fedora Project and in EPEL for Red Hat Enterprise 5 and 6. These packages are the result of the efforts made by Clint Savage (herlo).

Debian/Ubuntu

A team of many contributors have assisted in developing packages for Debian and Ubuntu. Salt is still actively seeking inclusion in upstream Debian and Ubuntu and the package data that has been prepared is being pushed through the needed channels for inclusion.

These packages have been prepared with the help of:

- Corey
- Aaron Toponce
- and‘

More to Come

We are actively seeking inclusion in more distributions. Primarily getting Salt into Gentoo, SUSE, OpenBSD and preparing Solaris support are all turning into higher priorities.

120.42.4 Refinement

Salt continues to be refined into a faster, more stable and more usable application. 0.9.5 comes with more debug logging, more bug fixes and more complete support.

More Testing, More BugFixes

0.9.5 comes with more bugfixes due to more testing than any previous release. The growing community and the introduction a a dedicated QA environment have unearthed many issues that were hiding under the covers. This has further refined and cleaned the state interface, taking care of things from minor visual issues to repairing misleading data.

Custom Exceptions

A custom exception module has been added to throw salt specific exceptions. This allows Salt to give much more granular error information.

New Modules

`data`

The new data module manages a persistent datastore on the minion. Big thanks to bastichelaar for his help refining this module

`freebsdkmod`

FreeBSD kernel modules can now be managed in the same way Salt handles Linux kernel modules.

This module was contributed thanks to the efforts of Christer Edwards

`gentoo_service`

Support has been added for managing services in Gentoo. Now Gentoo services can be started, stopped, restarted, enabled, disabled and viewed.

`pip`

The pip module introduces management for pip installed applications. Thanks goes to whitinge for the addition of the pip module

`rh_service`

The rh_service module enables Red Hat and Fedora specific service management. Now Red Hat like systems come with extensive management of the classic init system used by Red Hat

`saltutil`

The saltutil module has been added as a place to hold functions used in the maintenance and management of salt itself. Saltutil is used to salt the salt minion. The saltutil module is presently used only to sync extension modules from the master server.

`systemd`

Systemd support has been added to Salt, now systems using this next generation init system are supported on systems running systemd.

`virtualenv`

The virtualenv module has been added to allow salt to create virtual Python environments. Thanks goes to whitinge for the addition of the virtualenv module

`win_disk`

Support for gathering disk information on Microsoft Windows minions The windows modules come courtesy of Utah_Dave

`win_service`

The win_service module adds service support to Salt for Microsoft Windows services

`win_useradd`

Salt can now manage local users on Microsoft Windows Systems

`yumpkg5`

The yumpkg module introduces in 0.9.4 uses the yum API to interact with the yum package manager. Unfortunately, on Red Hat 5 systems salt does not have access to the yum API because the yum API is running under Python 2.4 and Salt needs to run under Python 2.6.

The yumpkg5 module bypasses this issue by shelling out to yum on systems where the yum API is not available.

New States

`mysql_database`

The new mysql_database state adds the ability to systems running a mysql server to manage the existence of mysql databases.

The mysql states are thanks to syphernl

`mysql_user`

The mysql_user state enables mysql user management.

`virtualenv`

The virtualenv state can manage the state of Python virtual environments. Thanks to Whitinge for the virtualenv state

New Returners

`cassandra_returner`

A returner allowing Salt to send data to a cassandra server. Thanks to Byron Clark for contributing this returner

120.43 Salt 0.9.6 Release Notes

release 2012-01-21

Salt 0.9.6 is a release targeting a few bugs and changes. This is primarily targeting an issue found in the names declaration in the state system. But a few other bugs were also repaired, like missing support for grains in extmods.

Due to a conflict in distribution packaging msgpack will no longer be bundled with Salt, and is required as a dependency.

120.43.1 New Features

HTTP and ftp support in files.managed

Now under the source option in the file.managed state a HTTP or ftp address can be used instead of a file located on the salt master.

Allow Multiple Returners

Now the returner interface can define multiple returners, and will also return data back to the master, making the process less ambiguous.

Minion Memory Improvements

A number of modules have been taken out of the minion if the underlying systems required by said modules are not present on the minion system. A number of other modules need to be stripped out in this same way which should continue to make the minion more efficient.

Minions Can Locally Cache Return Data

A new option, cache_jobs, has been added to the minion to allow for all of the historically run jobs to cache on the minion, allowing for looking up historic returns. By default cache_jobs is set to False.

Pure Python Template Support For file.managed

Templates in the file.managed state can now be defined in a Python script. This script needs to have a run function that returns the string that needs to be in the named file.

120.44 Salt 0.9.7 Release Notes

release 2012-02-15

Salt 0.9.7 is here! The latest iteration of Salt brings more features and many fixes. This release is a great refinement over 0.9.6, adding many conveniences under the hood, as well as some features that make working with Salt much better.

A few highlights include the new Job system, refinements to the requisite system in states, the `mod_init` interface for states, external node classification, search path to managed files in the file state, and refinements and additions to dynamic module loading.

0.9.7 also introduces the long developed (and oft changed) unit test framework and the initial unit tests.

120.44.1 Major Features

Salt Jobs Interface

The new jobs interface makes the management of running executions much cleaner and more transparent. Building on the existing execution framework the jobs system allows clear introspection into the active running state of the running Salt interface.

The Jobs interface is centered in the new minion side proc system. The minions now store msgpack serialized files under `/var/cache/salt/proc`. These files keep track of the active state of processes on the minion.

Functions in the saltutil Module

A number of functions have been added to the saltutil module to manage and view the jobs:

`running` - Returns the data of all running jobs that are found in the proc directory.

`find_job` - Returns specific data about a certain job based on job id.

`signal_job` - Allows for a given jid to be sent a signal.

`term_job` - Sends a termination signal (`SIGTERM, 15`) to the process controlling the specified job.

`kill_job` Sends a kill signal (`SIGKILL, 9`) to the process controlling the specified job.

The jobs Runner

A convenience runner front end and reporting system has been added as well. The jobs runner contains functions to make viewing data easier and cleaner.

The jobs runner contains a number of functions...

active

The active function runs `saltutil.running` on all minions and formats the return data about all running jobs in a much more usable and compact format. The active function will also compare jobs that have returned and jobs that are still running, making it easier to see what systems have completed a job and what systems are still being waited on.

lookup_jid

When jobs are executed the return data is sent back to the master and cached. By default is is cached for 24 hours, but this can be configured via the `keep_jobs` option in the master configuration.

Using the `lookup_jid` runner will display the same return data that the initial job invocation with the salt command would display.

list_jobs

Before finding a historic job, it may be required to find the job id. `list_jobs` will parse the cached execution data and display all of the job data for jobs that have already, or partially returned.

External Node Classification

Salt can now use external node classifiers like Cobbler's `cobbler-ext-nodes`.

Salt uses specific data from the external node classifier. In particular the classes value denotes which sls modules to run, and the environment value sets to another environment.

An external node classification can be set in the master configuration file via the `external_nodes` option: http://salt.readthedocs.org/en/latest/ref/configuration/master.html#external-nodes

External nodes are loaded in addition to the top files. If it is intended to only use external nodes, do not deploy any top files.

State Mod Init System

An issue arose with the pkg state. Every time a package was run Salt would need to refresh the package database. This made systems with slower package metadata refresh speeds much slower to work with. To alleviate this issue the `mod_init` interface has been added to salt states.

The `mod_init` interface is a function that can be added to a state file. This function is called with the first state called. In the case of the pkg state, the `mod_init` function sets up a tag which makes the package database only refresh on the first attempt to install a package.

In a nutshell, the `mod_init` interface allows a state to run any command that only needs to be run once, or can be used to set up an environment for working with the state.

Source File Search Path

The file state continues to be refined, adding speed and capabilities. This release adds the ability to pass a list to the source option. This list is then iterated over until the source file is found, and the first found file is used.

The new syntax looks like this:

```
/etc/httpd/conf/httpd.conf:
  file:
    - managed
    - source:
      - salt://httpd/httpd.conf
      - http://myserver/httpd.conf: md5=8c1fe119e6f1fd96bc06614473509bf1
```

The source option can take sources in the list from the salt file server as well as an arbitrary web source. If using an arbitrary web source the checksum needs to be passed as well for file verification.

Refinements to the Requisite System

A few discrepancies were still lingering in the requisite system, in particular, it was not possible to have a `require` and a `watch` requisite declared in the same state declaration.

This issue has been alleviated, as well as making the requisite system run more quickly.

Initial Unit Testing Framework

Because of the module system, and the need to test real scenarios, the development of a viable unit testing system has been difficult, but unit testing has finally arrived. Only a small amount of unit testing coverage has been developed, much more coverage will be in place soon.

A huge thanks goes out to those who have helped with unit testing, and the contributions that have been made to get us where we are. Without these contributions unit tests would still be in the dark.

Compound Targets Expanded

Originally only support for `and` and `or` were available in the compound target. 0.9.7 adds the capability to negate compound targets with `not`.

Nodegroups in the Top File

Previously the nodegroups defined in the master configuration file could not be used to match nodes for states. The nodegroups support has been expanded and the nodegroups defined in the master configuration can now be used to match minions in the top file.

120.45 Salt 0.9.8 Release Notes

release 2012-03-21

Salt 0.9.8 is a big step forward, with many additions and enhancements, as well as a number of precursors to advanced future developments.

This version of Salt adds much more power to the command line, making the old hard timeout issues a thing of the past and adds keyword argument support. These additions are also available in the salt client API, making the available API tools much more powerful.

The new pillar system allows for data to be stored on the master and assigned to minions in a granular way similar to the state system. It also allows flexibility for users who want to keep data out of their state tree similar to ‘external lookup’ functionality in other tools.

A new way to extend requisites was added, the “requisite in” statement. This makes adding requires or watch statements to external state decs much easier.

Additions to requisites making them much more powerful have been added as well as improved error checking for sls files in the state system. A new provider system has been added to allow for redirecting what modules run in the background for individual states.

Support for OpenSUSE has been added and support for Solaris has begun serious development. Windows support has been significantly enhanced as well.

The matcher and target systems have received a great deal of attention. The default behavior of grain matching has changed slightly to reflect the rest of salt and the compound matcher system has been refined.

A number of impressive features with keyword arguments have been added to both the CLI and to the state system. This makes states much more powerful and flexible while maintaining the simple configuration everyone loves.

The new batch size capability allows for executions to be rolled through a group of targeted minions a percentage or specific number at a time. This was added to prevent the "thundering herd" problem when targeting large numbers of minions for things like service restarts or file downloads.

120.45.1 Upgrade Considerations

Upgrade Issues

There was a previously missed oversight which could cause a newer minion to crash an older master. That oversight has been resolved so the version incompatibility issue will no longer occur. When upgrading to 0.9.8 make sure to upgrade the master first, followed by the minions.

Debian/Ubuntu Packages

The original Debian/Ubuntu packages were called salt and included all salt applications. New packages in the ppa are split by function. If an old salt package is installed then it should be manually removed and the new split packages need to be freshly installed.

On the master:

```
# apt-get purge salt
# apt-get install salt-{master,minion}
```

On the minions:

```
# apt-get purge salt
# apt-get install salt-minion
```

And on any Syndics:

```
# apt-get install salt-syndic
```

The official Salt PPA for Ubuntu is located at: https://launchpad.net/~saltstack/+archive/salt

120.45.2 Major Features

Pillar

Pillar offers an interface to declare variable data on the master that is then assigned to the minions. The pillar data is made available to all modules, states, sls files etc. It is compiled on the master and is declared using the existing renderer system. This means that learning pillar should be fairly trivial to those already familiar with salt states.

CLI Additions

The `salt` command has received a serious overhaul and is more powerful than ever. Data is returned to the terminal as it is received, and the salt command will now wait for all running minions to return data before stopping. This makes adding very large *–timeout* arguments completely unnecessary and gets rid of long running operations returning empty `{}` when the timeout is exceeded.

When calling salt via sudo, the user originally running salt is saved to the log for auditing purposes. This makes it easy to see who ran what by just looking through the minion logs.

The *salt-key* command gained the *-D* and *–delete-all* arguments for removing all keys. Be careful with this one!

Running States Without a Master

The addition of running states without a salt-master has been added to 0.9.8. This feature allows for the unmodified salt state tree to be read locally from a minion. The result is that the UNMODIFIED state tree has just become portable, allowing minions to have a local copy of states or to manage states without a master entirely.

This is accomplished via the new file client interface in Salt that allows for the `salt://` URI to be redirected to custom interfaces. This means that there are now two interfaces for the salt file server, calling the master or looking in a local, minion defined `file_roots`.

This new feature can be used by modifying the minion config to point to a local `file_roots` and setting the `file_client` option to `local`.

Keyword Arguments and States

State modules now accept the `**kwargs` argument. This results in all data in a sls file assigned to a state being made available to the state function.

This passes data in a transparent way back to the modules executing the logic. In particular, this allows adding arguments to the `pkg.install` module that enable more advanced and granular controls with respect to what the state is capable of.

An example of this along with the new debconf module for installing ldap client packages on Debian:

```
ldap-client-packages:
  pkg:
    - debconf: salt://debconf/ldap-client.ans
    - installed
    - names:
      - nslcd
      - libpam-ldapd
      - libnss-ldapd
```

Keyword Arguments and the CLI

In the past it was required that all arguments be passed in the proper order to the *salt* and *salt-call* commands. As of 0.9.8, keyword arguments can be passed in the form of `kwarg=argument`.

```
# salt -G 'type:dev' git.clone \
    repository=https://github.com/saltstack/salt.git cwd=/tmp/salt user=jeff
```

Matcher Refinements and Changes

A number of fixes and changes have been applied to the Matcher system. The most noteworthy is the change in the grain matcher. The grain matcher used to use a regular expression to match the passed data to a grain, but now defaults to a shell glob like the majority of match interfaces in Salt. A new option is available that still uses the old style regex matching to grain data called `grain-pcre`. To use regex matching in compound matches use the letter *P*.

For example, this would match any ArchLinux or Fedora minions:

```
# salt --grain-pcre 'os:(Arch:Fed).*' test.ping
```

And the associated compound matcher suitable for `top.sls` is *P*:

```
P@os:(Arch|Fed).*
```

NOTE: Changing the grains matcher from pcre to glob is backwards incompatible.

Support has been added for matching minions with Yahoo's range library. This is handled by passing range syntax with *-R* or *–range* arguments to salt.

More information at: https://github.com/grierj/range/wiki/Introduction-to-Range-with-YAML-files

Requisite "in"

A new means to updating requisite statements has been added to make adding watchers and requires to external states easier. Before 0.9.8 the only way to extend the states that were watched by a state outside of the sls was to use an extend statement:

```
include:
  - http
extend:
  apache:
    service:
      - watch:
        - pkg: tomcat

tomcat:
  pkg:
    - installed
```

But the new `Requisite in` statement allows for easier extends for requisites:

```
include:
  - http

tomcat:
  pkg:
    - installed
    - watch_in:
      - service: apache
```

Requisite in is part of the extend system, so still remember to always include the sls that is being extended!

Providers

Salt predetermines what modules should be mapped to what uses based on the properties of a system. These determinations are generally made for modules that provide things like package and service management. The apt module maps to pkg on Debian and the yum module maps to pkg on Fedora for instance.

Sometimes in states, it may be necessary for a non-default module to be used for the desired functionality. For instance, an Arch Linux system may have been set up with systemd support. Instead of using the default service module detected for Arch Linux, the systemd module can be used:

```
http:
  service:
    - running
    - enable: True
    - provider: systemd
```

Default providers can also be defined in the minion config file:

```
providers:
  service: systemd
```

When default providers are passed in the minion config, then those providers will be applied to all functionality in Salt, this means that the functions called by the minion will use these modules, as well as states.

Requisite Glob Matching

Requisites can now be defined with glob expansion. This means that if there are many requisites, they can be defined on a single line.

To watch all files in a directory:

```
http:
  service:
    - running
    - enable: True
    - watch:
      - file: /etc/http/conf.d/*
```

This example will watch all defined files that match the glob `/etc/http/conf.d/*`

Batch Size

The new batch size option allows commands to be executed while maintaining that only so many hosts are executing the command at one time. This option can take a percentage or a finite number:

```
salt '*' -b 10 test.ping

salt -G 'os:RedHat' --batch-size 25% apache.signal restart
```

This will only run test.ping on 10 of the targeted minions at a time and then restart apache on 25% of the minions matching `os:RedHat` at a time and work through them all until the task is complete. This makes jobs like rolling web server restarts behind a load balancer or doing maintenance on BSD firewalls using carp much easier with salt.

Module Updates

This is a list of notable, but non-exhaustive updates with new and existing modules.

Windows support has seen a flurry of support this release cycle. We've gained all new *file*, *network*, and *shadow* modules. Please note that these are still a work in progress.

For our ruby users, new *rvm* and *gem* modules have been added along with the *associated states*

The *virt* module gained basic Xen support.

The *yum* module gained Scientific Linux support.

The `pkg` module on Debian, Ubuntu, and derivatives force apt to run in a non-interactive mode. This prevents issues when package installation waits for confirmation.

A *pkg* module for OpenSUSE's zypper was added.

The *service* module on Ubuntu natively supports upstart.

A new *debconf* module was contributed by our community for more advanced control over deb package deployments on Debian based distributions.

The *mysql.user* state and *mysql* module gained a *password_hash* argument.

The *cmd* module and state gained a *shell* keyword argument for specifying a shell other than `/bin/sh` on Linux / Unix systems.

New *git* and *mercurial* modules have been added for fans of distributed version control.

120.45.3 In Progress Development

Master Side State Compiling

While we feel strongly that the advantages gained with minion side state compiling are very critical, it does prevent certain features that may be desired. 0.9.8 has support for initial master side state compiling, but many more components still need to be developed, it is hoped that these can be finished for 0.9.9.

The goal is that states can be compiled on both the master and the minion allowing for compilation to be split between master and minion. Why will this be great? It will allow storing sensitive data on the master and sending it to some minions without all minions having access to it. This will be good for handling ssl certificates on front-end web servers for instance.

Solaris Support

Salt 0.9.8 sees the introduction of basic Solaris support. The daemon runs well, but grains and more of the modules need updating and testing.

Windows Support

Salt states on windows are now much more viable thanks to contributions from our community! States for file, service, local user, and local group management are more fully fleshed out along with network and disk modules. Windows users can also now manage registry entries using the new "reg" module.

120.46 Salt 0.9.9 Release Notes

release 2012-04-27

0.9.9 is out and comes with some serious bug fixes and even more serious features. This release is the last major feature release before 1.0.0 and could be considered the 1.0.0 release candidate.

A few updates include more advanced kwargs support, the ability for salt states to more safely configure a running salt minion, better job directory management and the new state test interface.

Many new tests have been added as well, including the new minion swarm test that allows for easier testing of Salt working with large groups of minions. This means that if you have experienced stability issues with Salt before, particularly in larger deployments, that these bugs have been tested for, found, and killed.

120.46.1 Major Features

State Test Interface

Until 0.9.9 the only option when running states to see what was going to be changed was to print out the highstate with state.show_highstate and manually look it over. But now states can be run to discover what is going to be changed.

Passing the option `test=True` to many of the state functions will now cause the salt state system to only check for what is going to be changed and report on those changes.

```
salt '*' state.highstate test=True
```

Now states that would have made changes report them back in yellow.

State Syntax Update

A shorthand syntax has been added to sls files, and it will be the default syntax in documentation going forward. The old syntax is still fully supported and will not be deprecated, but it is recommended to move to the new syntax in the future. This change moves the state function up into the state name using a dot notation. This is in-line with how state functions are generally referred to as well:

The new way:

```
/etc/sudoers:
  file.present:
    - source: salt://sudo/sudoers
    - user: root
    - mode: 400
```

Use and Use_in Requisites

Two new requisite statements are available in 0.9.9. The use and use_in requisite and requisite-in allow for the transparent duplication of data between states. When a state "uses" another state it copies the other state's arguments as defaults. This was created in direct response to the new network state, and allows for many network interfaces to be configured in the same way easily. A simple example:

```
root_file:
  file.absent:
    - name: /tmp/nothing
    - user: root
    - mode: 644
    - group: root
    - use_in:
      - file: /etc/vimrc

fred_file:
  file.absent:
    - name: /tmp/nothing
    - user: fred
    - group: marketing
    - mode: 660

/files/marketing/district7.rst:
  file.present:
    - source: salt://marketing/district7.rst
    - template: jinja
    - use:
      - file: fred_file

/etc/vimrc:
  file.present:
    - source: salt://edit/vimrc
```

This makes the 2 lower state decs inherit the options from their respectively "used" state decs.

Network State

The new network state allows for the configuration of network devices via salt states and the ip salt module. This addition has been given to the project by Jeff Hutchins and Bret Palsson from Jive Communications.

Currently the only network configuration backend available is for Red Hat based systems, like Red Hat Enterprise, CentOS, and Fedora.

Exponential Jobs

Originally the jobs executed were stored on the master in the format: `<cachedir>/jobs/jid/{minion ids}` But this format restricted the number of jobs in the cache to the number of subdirectories allowed on the filesystem. Ext3 for instance limits subdirectories to 32000. To combat this the new format for 0.9.9 is: `<cachedir>/jobs/jid_hash[:2]/jid_hash[2:]/{minion ids}` So that now the number of maximum jobs that can be run before the cleanup cycle hits the job directory is substantially higher.

ssh_auth Additions

The original ssh_auth state was limited to accepting only arguments to apply to a public key, and the key itself. This was restrictive due to the way the we learned that many people were using the state, so the key section has been expanded to accept options and arguments to the key that over ride arguments passed in the state. This gives substantial power to using ssh_auth with names:

```
sshkeys:
  ssh_auth:
    - present
    - user: backup
    - enc: ssh-dss
    - options:
      - option1="value1"
      - option2="value2 flag2"
    - comment: backup
    - names:
      - AAAAB3NzaC1yc2EAAAABIwAAAQEAlyE26SMFFVY5YJvnL7AF5CRTPtAigSW1U887ASfBt6FDa7Qr1YdO5ochiLoz8aSi
      - AAAAB3NzaC1yc2EAAAABIwAAAQEAlyE26SMFFVY5YJvnL7AF5CRTPtAigSW1U887ASfBt6FDa7Qr1YdO5ochiLoz8aSi
      - ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEAlyE26SMFFVY5YJvnL7AF5CRTPtAigSW1U887ASfBt6FDa7Qr1YdO5och
      - ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEAlyE26SMFFVY5YJvnL7AF5CRTPtAigSW1U887ASfBt6FDa7Qr1YdO5och
      - option3="value3",option4="value4 flag4" ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEAlyE26SMFFVY5YJvn
      - option3="value3" ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEAlyE26SMFFVY5YJvnL7AF5CRTPtAigSW1U887ASf
```

LocalClient Additions

To follow up the recent additions in 0.9.8 of additional kwargs support, 0.9.9 also adds the capability to send kwargs into commands via a dict. This addition to the LocalClient api can be used like so:

```
import salt.client

client = salt.client.LocalClient('/etc/salt/master')
ret = client.cmd('*', 'cmd.run', ['ls -l'], kwarg={'cwd': '/etc'})
```

This update has been added to all cmd methods in the LocalClient class.

Better Self Salting

One problem faced with running Salt states, is that it has been difficult to manage the Salt minion via states, this is due to the fact that if the minion is called to restart while a state run is happening then the state run would be killed. 0.9.9 slightly changes the process scope of the state runs, so now when salt is executing states it can safely restart the salt-minion daemon.

In addition to daemonizing the state run, the apt module also daemonizes. This update makes it possible to cleanly update the salt-minion package on Debian/Ubuntu systems without leaving apt in an inconsistent state or killing the active minion process mid-execution.

Wildcards for SLS Modules

Now, when including sls modules in include statements or in the top file, shell globs can be used. This can greatly simplify listing matched sls modules in the top file and include statements:

```
base:
  '*':
    - files*
    - core*

include:
  - users.dev.*
  - apache.ser*
```

External Pillar

Since the pillar data is just, data, it does not need to come expressly from the pillar interface. The external pillar system allows for hooks to be added making it possible to extract pillar data from any arbitrary external interface. The external pillar interface is configured via the `ext_pillar` option. Currently interfaces exist to gather external pillar data via hiera or via a shell command that sends yaml data to the terminal:

```
ext_pillar:
  - cmd_yaml: cat /etc/salt/ext.yaml
  - hiera: /etc/hirea.yaml
```

The initial external pillar interfaces and extra interfaces can be added to the file salt/pillar.py, it is planned to add more external pillar interfaces. If the need arises a new module loader interface will be created in the future to manage external pillar interfaces.

Single State Executions

The new state.single function allows for single states to be cleanly executed. This is a great tool for setting up a small group of states on a system or for testing out the behavior of single states:

```
salt '*' state.single user.present name=wade uid=2000
```

The test interface functions here as well, so changes can also be tested against as:

```
salt '*' state.single user.present name=wade uid=2000 test=True
```

120.46.2 New Tests

A few exciting new test interfaces have been added, the minion swarm allows not only testing of larger loads, but also allows users to see how Salt behaves with large groups of minions without having to create a large deployment.

Minion Swarm

The minion swarm test system allows for large groups of minions to be tested against easily without requiring large numbers of servers or virtual machines. The minion swarm creates as many minions as a system can handle and roots them in the /tmp directory and connects them to a master.

The benefit here is that we were able to replicate issues that happen only when there are large numbers of minions. A number of elusive bugs which were causing stability issues in masters and minions have since been hunted down. Bugs that used to take careful watch by users over several days can now be reliably replicated in minutes, and fixed in minutes.

Using the swarm is easy, make sure a master is up for the swarm to connect to, and then use the minionswarm.py script in the tests directory to spin up as many minions as you want. Remember, this is a fork bomb, don't spin up more than your hardware can handle!

```
python minionswarm.py -m 20 --master salt-master
```

Shell Tests

The new Shell testing system allows us to test the behavior of commands executed from a high level. This allows for the high level testing of salt runners and commands like salt-key.

Client Tests

Tests have been added to test the aspects of the client APIs and ensure that the client calls work, and that they manage passed data, in a desirable way.

120.47 Salt 2014.1.0 Release Notes - Codename Hydrogen

Note: Due to a change in master to minion communication, 2014.1.0 minions are not compatible with older-version masters. Please upgrade masters first. More info on backwards-compatibility policy *here*, under the "Upgrading Salt" subheading.

release 2014-02-24

The 2014.1.0 release of Salt is a major release which not only increases stability but also brings new capabilities in virtualization, cloud integration, and more. This release brings a great focus on the expansion of testing making roughly double the coverage in the Salt tests, and comes with many new features.

2014.1.0 is the first release to follow the new date-based release naming system.

120.47.1 Major Features

Salt Cloud Merged into Salt

Salt Cloud is a tool for provisioning salted minions across various cloud providers. Prior to this release, Salt Cloud was a separate project but this marks its full integration with the Salt distribution. A Getting Started guide and additional documentation for Salt Cloud can be found *here*:

Google Compute Engine

Alongside Salt Cloud comes new support for the Google Compute Engine. Salt Stack can now deploy and control GCE virtual machines and the application stacks that they run.

For more information on Salt Stack and GCE, please see this blog post.

Documentation for Salt and GCE can be found *here*.

Salt Virt

Salt Virt is a cloud controller that supports virtual machine deployment, inspection, migration and integration with many aspects of Salt.

Salt Virt has undergone a major overhaul with this release and now supports many more features and includes a number of critical improvements.

Docker Integration

Salt now ships with `states` and an `execution module` to manage Docker containers.

Substantial Testing Expansion

Salt continues to increase its unit/regression test coverage. This release includes over 300 new tests.

BSD Package Management

BSD package management has been entirely rewritten. FreeBSD 9 and older now default to using pkg_add, while FreeBSD 10 and newer will use pkgng. FreeBSD 9 can be forced to use pkgng, however, by specifying the following option in the minion config file:

```
providers:
  pkg: pkgng
```

In addition, support for installing software from the ports tree has been added. See the documentation for the ports `state` and `execution module` for more information.

Network Management for Debian/Ubuntu

Initial support for management of network interfaces on Debian-based distros has been added. See the documentation for the `network state` and the `debian_ip` for more information.

IPv6 Support for iptables State/Module

The iptables `state` and `module` now have IPv6 support. A new parameter `family` has been added to the states and execution functions, to distinguish between IPv4 and IPv6. The default value for this parameter is `ipv4`, specifying `ipv6` will use ip6tables to manage firewall rules.

GitFS Improvements

Several performance improvements have been made to the `Git fileserver backend`. Additionally, file states can now use any any SHA1 commit hash as a fileserver environment:

```
/etc/httpd/httpd.conf:
  file.managed:
    - source: salt://webserver/files/httpd.conf
    - saltenv: 45af879
```

This applies to the functions in the `cp module` as well:

```
salt '*' cp.get_file salt://readme.txt /tmp/readme.txt saltenv=45af879
```

MinionFS

This new fileserver backend allows files which have been pushed from the minion to the master (using `cp.push`) to be served up from the salt fileserver. The path for these files takes the following format:

```
salt://minion-id/path/to/file
```

`minion-id` is the id of the "source" minion, the one from which the files were pushed to the master. `/path/to/file` is the full path of the file.

The *MinionFS Walkthrough* contains a more thorough example of how to use this backend.

saltenv

To distinguish between fileserver environments and execution functions which deal with environment variables, fileserver environments are now specified using the `saltenv` parameter. `env` will continue to work, but is deprecated and will be removed in a future release.

Grains Caching

A caching layer has been added to the Grains system, which can help speed up minion startup. Disabled by default, it can be enabled by setting the minion config option `grains_cache`:

```
grains_cache: True

# Seconds before grains cache is considered to be stale.
grains_cache_expiration: 300
```

If set to `True`, the grains loader will read from/write to a msgpack-serialized file containing the grains data.

Additional command-line parameters have been added to salt-call, mainly for testing purposes:

- `--skip-grains` will completely bypass the grains loader when salt-call is invoked.
- `--refresh-grains-cache` will force the grains loader to bypass the grains cache and refresh the grains, writing a new grains cache file.

Improved Command Logging Control

When using the `cmd module`, either on the CLI or when developing Salt execution modules, a new keyword argument `output_loglevel` allows for greater control over how (or even if) the command and its output are logged. For example:

```
salt '*' cmd.run 'tail /var/log/messages' output_loglevel=debug
```

The package management modules (`apt`, `yumpkg`, etc.) have been updated to log the copious output generated from these commands at loglevel `debug`.

Note: To keep a command from being logged, `output_loglevel=quiet` can be used.

Prior to this release, this could be done using `quiet=True`. This argument is still supported, but will be removed in a future Salt release.

PagerDuty Support

Initial support for firing events via PagerDuty has been added. See the documentation for the `pagerduty` module.

Virtual Terminal

Sometimes the subprocess module is not good enough, and, in fact, not even `askpass` is. This virtual terminal is still in it's infant childhood, needs quite some love, and was originally created to replace `askpass`, but, while developing it, it immediately proved that it could do so much more. It's currently used by salt-cloud when bootstrapping salt on clouds which require the use of a password.

Proxy Minions¬ ————-¬ ¬ Initial basic support for Proxy Minions is in this release. Documentation can¬ be found `here`.¬ ¬ Proxy minions are a developing feature in Salt that enables control of¬ devices that cannot run a minion. Examples include network gear like switches¬ and routers that run a proprietary OS but offer an API, or "dumb"¬ devices that just don't have the horsepower or ability to handle a Python VM.¬ ¬ Proxy minions can be difficult to write, so a simple REST-based example proxy¬ is included. A Python bottle-based webserver can be found at¬ http://github.com/cro/salt-proxy-rest as an endpoint for this proxy.¬

This is an ALPHA-quality feature. There are a number of issues with it currently, mostly centering around process control, logging, and inability to work in a masterless configuration.

Additional Bugfixes (Release Candidate Period)

Below are many of the fixes that were implemented in salt during the release candidate phase.

- Fix mount.mounted leaving conflicting entries in fstab (issue 7079)
- Fix mysql returner serialization to use json (issue 9590)
- Fix `ZMQError: Operation cannot be accomplished in current state` errors (issue 6306)
- Rbenv and ruby improvements
- Fix quoting issues with mysql port (issue 9568)
- Update mount module/state to support multiple swap partitions (issue 9520)
- Fix `archive` state to work with `bsdtar`
- Clarify logs for minion ID caching

- Add numeric revision support to git state (issue 9718)
- Update `master_uri` with `master_ip` (issue 9694)
- Add comment to Debian `mod_repo` (issue 9923)
- Fix potential undefined loop variable in rabbitmq state (issue 8703)
- Fix for salt-virt runner to delete key on VM deletion
- Fix for `salt-run -d` to limit results to specific runner or function (issue 9975)
- Add tracebacks to jinja renderer when applicable (issue 10010)
- Fix parsing in monit module (issue 10041)
- Fix highstate output from syndic minions (issue 9732)
- Quiet logging when dealing with passwords/hashes (issue 10000)
- Fix for multiple remotes in git_pillar (issue 9932)
- Fix npm installed command (issue 10109)
- Add safeguards for utf8 errors in zcbuildout module
- Fix compound commands (issue 9746)
- Add systemd notification when master is started
- Many doc improvements

120.48 Salt 2014.1.1 Release Notes

release 2014-03-18

Version 2014.1.1 is another bugfix release for *2014.1.0*. The changes include:

- Various doc fixes, including up-to-date Salt Cloud installation documentation.
- Renamed state.sls runner to state.orchestrate, to reduce confusion with the `state.sls` execution function
- Fix various bugs in the `dig` module (issue 10367)
- Add retry for query on certain EC2 status codes (issue 10154)
- Fix various bugs in mongodb_user state module (issue 10430)
- Fix permissions on `~/.salt_token` (issue 10422)
- Add PyObjects support
- Fix launchctl module crash with missing files
- Fix `saltutil.find_job` for Windows (issue 10581)
- Fix OS detection for OpenSolaris (issue 10601)
- Fix broken salt-ssh key_deploy
- Add support for multiline cron comments (issue 10721)
- Fix timezone module for Arch (issue 10789)
- Fix symlink support for `file.recurse` (issue 10809)
- Fix multi-master bugs (issue 10732 and issue 10969)

- Fix file.patch to error when source file is unavailable (issue 10380)
- Fix pkg to handle packages set as `purge` in `pkg.installed` (issue 10719)
- Add `zmqversion` grain
- Fix highstate summary for masterless minions (issue 10945)
- Fix `saltutil.find_job` for 2014.1 masters talking to 0.17 minions (issue 11020)
- Fix `file.recurse` states with trailing slashes in source (issue 11002)
- Fix `pkg states` to allow `pkgname.x86_64` (issue 7306)
- Make `iptables states` set a default table for flush (issue 11037)
- Added iptables `--reject-with` after final iptables call in `iptables states` (issue:*10757*)
- Fix improper passing of "family" in `iptables states` (issue 10774)
- Fix traceback in `iptables.insert` states (issue 10988)
- Fix zombie processes (issue 10867 and others)
- Fix batch mode to obey `--return` settings (issue 9146)
- Fix localclient issue that was causing batch mode breakage (issue 11094, issue 10470, and others)
- Multiple salt-ssh fixes
- FreeBSD: look in /usr/local/etc/salt for configuration by default, if installed using `pip --editable`.
- Add a `skip_suggestions` parameter to pkg.installed states which allows pre-flight check to be skipped (issue 11106)
- Fixed tag-based gitfs fileserver environments regression (issue 10956)
- Yum: fix cache of available pkgs not cleared when repos are changed (issue 11001)
- Yum: fix for plugin-provided repositories (i.e. RHN/Spacewalk) (issue 11145)
- Fix regression in `chocolatey.bootstrap` (issue 10541)
- Fix fail on unknown target in `jobs runner` (issue 11151)
- Don't log errors for commands which are expected to sometimes exit with non-zero exit status (issue 11154, issue 11090)
- Fix `test=True` CLI override of config option (issue 10877)
- Log sysctl key listing at loglevel TRACE (issue 10931)

120.49 Salt 2014.1.2 Release Notes

release 2014-04-15

Version 2014.1.2 is another bugfix release for *2014.1.0*. The changes include:

- Fix username detection when su'ed to root on FreeBSD (issue 11628)
- Fix minionfs backend for file.recurse states
- Fix 32-bit packages of different arches than the CPU arch, on 32-bit RHEL/CentOS (issue 11822)
- Fix bug with specifying alternate home dir on user creation (FreeBSD) (issue 11790)
- Don't reload `site` module on module refresh for MacOS

- Fix regression with running execution functions in Pillar SLS (issue 11453)
- Fix some modules missing from Windows installer
- Don't log an error for yum commands that return nonzero exit status on non-failure (issue 11645)
- Fix bug in rabbitmq state (issue 8703)
- Fix missing ssh config options (issue 10604)
- Fix top.sls ordering (issue 10810 and issue 11691)
- Fix `salt-key --list all` (issue 10982)
- Fix win_servermanager install/remove function (issue 11038)
- Fix interaction with tokens when running commands as root (issue 11223)
- Fix overstate bug with `find_job` and `**kwargs` (issue 10503)
- Fix `saltenv` for `aptpkg.mod_repo` from `pkgrepo` state
- Fix environment issue causing file caching problems (issue 11189)
- Fix bug in `__parse_key` in registry state (issue 11408)
- Add minion auth retry on rejection (issue 10763)
- Fix publish_session updating the encryption key (issue 11493)
- Fix for bad `AssertionError` raised by GitPython (issue 11473)
- Fix `debian_ip` to allow disabling and enabling networking on Ubuntu (issue 11164)
- Fix potential memory leak caused by saved (and unused) events (issue 11582)
- Fix exception handling in the MySQL module (issue 11616)
- Fix environment-related error (issue 11534)
- Include `psutil` on Windows
- Add `file.replace` and `file.search` to Windows (issue 11471)
- Add additional `file` module helpers to Windows (issue 11235)
- Add `pid` to netstat output on Windows (issue 10782)
- Fix Windows not caching new versions of installers in winrepo (issue 10597)
- Fix hardcoded md5 hashing
- Fix kwargs in salt-ssh (issue 11609)
- Fix file backup timestamps (issue 11745)
- Fix stacktrace on `sys.doc` with invalid eauth (issue 11293)
- Fix `git.latest` with `test=True` (issue 11595)
- Fix `file.check_perms` hardcoded `follow_symlinks` (issue 11387)
- Fix certain `pkg` states for RHEL5/Cent5 machines (issue 11719)

120.50 Salt 2014.1.2 Release Notes

release 2014-04-15

Version 2014.1.3 is another bugfix release for *2014.1.0*. It was created as a hotfix for a regression found in 2014.1.2, which was not distributed. The only change made was as follows:

- Fix regression that caused `saltutil.find_job` to fail, causing premature terminations of salt CLI commands.

Changes in the not-distributed 2014.1.2, also included in 2014.1.3:

- Fix username detection when su'ed to root on FreeBSD (issue 11628)
- Fix minionfs backend for file.recurse states
- Fix 32-bit packages of different arches than the CPU arch, on 32-bit RHEL/CentOS (issue 11822)
- Fix bug with specifying alternate home dir on user creation (FreeBSD) (issue 11790)
- Don't reload `site` module on module refresh for MacOS
- Fix regression with running execution functions in Pillar SLS (issue 11453)
- Fix some modules missing from Windows installer
- Don't log an error for yum commands that return nonzero exit status on non-failure (issue 11645)
- Fix bug in rabbitmq state (issue 8703)
- Fix missing ssh config options (issue 10604)
- Fix top.sls ordering (issue 10810 and issue 11691)
- Fix `salt-key --list all` (issue 10982)
- Fix win_servermanager install/remove function (issue 11038)
- Fix interaction with tokens when running commands as root (issue 11223)
- Fix overstate bug with `find_job` and `**kwargs` (issue 10503)
- Fix `saltenv` for `aptpkg.mod_repo` from `pkgrepo` state
- Fix environment issue causing file caching problems (issue 11189)
- Fix bug in `__parse_key` in registry state (issue 11408)
- Add minion auth retry on rejection (issue 10763)
- Fix publish_session updating the encryption key (issue 11493)
- Fix for bad `AssertionError` raised by GitPython (issue 11473)
- Fix `debian_ip` to allow disabling and enabling networking on Ubuntu (issue 11164)
- Fix potential memory leak caused by saved (and unused) events (issue 11582)
- Fix exception handling in the MySQL module (issue 11616)
- Fix environment-related error (issue 11534)
- Include `psutil` on Windows
- Add `file.replace` and `file.search` to Windows (issue 11471)
- Add additional `file` module helpers to Windows (issue 11235)
- Add `pid` to netstat output on Windows (issue 10782)

- Fix Windows not caching new versions of installers in winrepo (issue 10597)
- Fix hardcoded md5 hashing
- Fix kwargs in salt-ssh (issue 11609)
- Fix file backup timestamps (issue 11745)
- Fix stacktrace on `sys.doc` with invalid eauth (issue 11293)
- Fix `git.latest` with `test=True` (issue 11595)
- Fix `file.check_perms` hardcoded `follow_symlinks` (issue 11387)
- Fix certain `pkg` states for RHEL5/Cent5 machines (issue 11719)

120.51 Salt 2014.1.4 Release Notes

release 2014-05-05

Version 2014.1.4 is another bugfix release for *2014.1.0*. Changes include:

- Fix setup.py dependency issue (issue 12031)
- Fix handling for IOErrors under certain circumstances (issue 11783 and issue 11853)
- Fix fatal exception when `/proc/1/cgroup` is not readable (issue 11619)
- Fix os grains for OpenSolaris (issue 11907)
- Fix `lvs.zero` module argument pass-through (issue 9001)
- Fix bug in `debian_ip` interaction with `network.system` state (issue 11164)
- Remove bad binary package verification code (issue 12177)
- Fix traceback in solaris package installation (issue 12237)
- Fix `file.directory` state symlink handling (issue 12209)
- Remove `external_ip` grain
- Fix `file.managed` makedirs issues (issue 10446)
- Fix hang on non-existent Windows drive letter for `file` module (issue 9880)
- Fix salt minion caching all users on the server (issue 9743)

See also:

`Legacy salt-cloud release docs`

Salt Module Index

a

salt.auth.auto, 927
salt.auth.keystone, 927
salt.auth.ldap, 927
salt.auth.pam, 928
salt.auth.stormpath_mod, 929

c

salt.cloud.clouds.botocore_aws, 935
salt.cloud.clouds.cloudstack, 936
salt.cloud.clouds.digital_ocean, 937
salt.cloud.clouds.ec2, 938
salt.cloud.clouds.gce, 942
salt.cloud.clouds.gogrid, 944
salt.cloud.clouds.ibmsce, 945
salt.cloud.clouds.joyent, 946
salt.cloud.clouds.libcloud_aws, 949
salt.cloud.clouds.linode, 951
salt.cloud.clouds.msazure, 952
salt.cloud.clouds.nova, 953
salt.cloud.clouds.openstack, 955
salt.cloud.clouds.parallels, 957
salt.cloud.clouds.rackspace, 959
salt.cloud.clouds.saltify, 960
salt.cloud.clouds.softlayer, 960

e

salt.exceptions, 1044

f

salt.fileserver.gitfs, 987
salt.fileserver.hgfs, 988
salt.fileserver.minionfs, 990
salt.fileserver.roots, 988
salt.fileserver.s3fs, 989

l

salt.log.handlers.logstash_mod, 349
salt.log.handlers.sentry_mod, 352

m

salt.modules.aliases, 367
salt.modules.alternatives, 368
salt.modules.apache, 369
salt.modules.aptpkg, 371
salt.modules.archive, 376
salt.modules.at, 378
salt.modules.augeas_cfg, 379
salt.modules.aws_sqs, 380
salt.modules.bluez, 381
salt.modules.brew, 383
salt.modules.bridge, 385
salt.modules.bsd_shadow, 386
salt.modules.cassandra, 387
salt.modules.chocolatey, 388
salt.modules.cloud, 391
salt.modules.cmdmod, 392
salt.modules.composer, 395
salt.modules.config, 396
salt.modules.cp, 397
salt.modules.cron, 400
salt.modules.daemontools, 401
salt.modules.darwin_sysctl, 403
salt.modules.data, 403
salt.modules.ddns, 404
salt.modules.debconfmod, 405
salt.modules.debian_ip, 406
salt.modules.debian_service, 407
salt.modules.dig, 409
salt.modules.disk, 410
salt.modules.djangomod, 410
salt.modules.dnsmasq, 411
salt.modules.dnsutil, 412
salt.modules.dockerio, 413
salt.modules.dpkg, 425
salt.modules.ebuild, 426
salt.modules.eix, 431
salt.modules.eselect, 431
salt.modules.event, 432
salt.modules.extfs, 432

salt.modules.file, 433
salt.modules.freebsd_sysctl, 448
salt.modules.freebsdjail, 449
salt.modules.freebsdkmod, 450
salt.modules.freebsdpkg, 450
salt.modules.freebsdports, 454
salt.modules.freebsdservice, 455
salt.modules.gem, 457
salt.modules.gentoo_service, 459
salt.modules.gentoolkitmod, 460
salt.modules.git, 462
salt.modules.glance, 468
salt.modules.gnomedesktop, 469
salt.modules.grains, 470
salt.modules.groupadd, 473
salt.modules.grub_legacy, 474
salt.modules.guestfs, 474
salt.modules.hg, 475
salt.modules.hosts, 476
salt.modules.htpasswd, 477
salt.modules.img, 477
salt.modules.iptables, 478
salt.modules.key, 482
salt.modules.keyboard, 482
salt.modules.keystone, 482
salt.modules.kmod, 488
salt.modules.launchctl, 489
salt.modules.layman, 490
salt.modules.ldapmod, 490
salt.modules.linux_acl, 491
salt.modules.linux_lvm, 492
salt.modules.linux_sysctl, 493
salt.modules.localemod, 494
salt.modules.locate, 494
salt.modules.logrotate, 495
salt.modules.lvs, 496
salt.modules.lxc, 498
salt.modules.mac_group, 500
salt.modules.mac_user, 501
salt.modules.makeconf, 502
salt.modules.match, 509
salt.modules.mdadm, 510
salt.modules.memcached, 511
salt.modules.mine, 513
salt.modules.modjk, 513
salt.modules.mongodb, 517
salt.modules.monit, 518
salt.modules.moosefs, 518
salt.modules.mount, 519
salt.modules.munin, 520
salt.modules.mysql, 521
salt.modules.netbsd_sysctl, 528
salt.modules.netbsdservice, 528
salt.modules.network, 530
salt.modules.nfs3, 532
salt.modules.nginx, 532
salt.modules.nova, 533
salt.modules.npm, 536
salt.modules.omapi, 537
salt.modules.openbsdpkg, 538
salt.modules.openbsdservice, 539
salt.modules.openstack_config, 540
salt.modules.osxdesktop, 540
salt.modules.pacman, 541
salt.modules.pagerduty, 544
salt.modules.pam, 544
salt.modules.parted, 545
salt.modules.pecl, 548
salt.modules.pillar, 549
salt.modules.pip, 550
salt.modules.pkg, 363
salt.modules.pkg_resource, 553
salt.modules.pkgin, 554
salt.modules.pkgng, 556
salt.modules.pkgutil, 567
salt.modules.portage_config, 569
salt.modules.postgres, 570
salt.modules.poudriere, 575
salt.modules.powerpath, 576
salt.modules.ps, 576
salt.modules.publish, 580
salt.modules.puppet, 581
salt.modules.pw_group, 582
salt.modules.pw_user, 582
salt.modules.qemu_img, 584
salt.modules.qemu_nbd, 584
salt.modules.quota, 585
salt.modules.rabbitmq, 586
salt.modules.rbenv, 590
salt.modules.rdp, 591
salt.modules.reg, 591
salt.modules.ret, 592
salt.modules.rh_ip, 593
salt.modules.rh_service, 594
salt.modules.riak, 596
salt.modules.rpm, 597
salt.modules.rvm, 597
salt.modules.s3, 600
salt.modules.saltcloudmod, 602
salt.modules.saltutil, 602
salt.modules.seed, 605
salt.modules.selinux, 605
salt.modules.service, 606
salt.modules.shadow, 607
salt.modules.smartos_imgadm, 608
salt.modules.smartos_vmadm, 609
salt.modules.smf, 611
salt.modules.solaris_group, 613

salt.modules.solaris_shadow, 613
salt.modules.solaris_user, 614
salt.modules.solarispkg, 616
salt.modules.solr, 619
salt.modules.sqlite3, 625
salt.modules.ssh, 626
salt.modules.state, 627
salt.modules.status, 630
salt.modules.supervisord, 632
salt.modules.svn, 634
salt.modules.sysbench, 637
salt.modules.sysmod, 638
salt.modules.system, 639
salt.modules.systemd, 640
salt.modules.test, 642
salt.modules.timezone, 645
salt.modules.tls, 645
salt.modules.tomcat, 648
salt.modules.upstart, 652
salt.modules.useradd, 654
salt.modules.virt, 656
salt.modules.virtualenv_mod, 663
salt.modules.win_autoruns, 664
salt.modules.win_disk, 664
salt.modules.win_dns_client, 664
salt.modules.win_file, 665
salt.modules.win_firewall, 667
salt.modules.win_groupadd, 667
salt.modules.win_ip, 668
salt.modules.win_network, 670
salt.modules.win_ntp, 672
salt.modules.win_path, 672
salt.modules.win_pkg, 673
salt.modules.win_repo, 675
salt.modules.win_servermanager, 675
salt.modules.win_service, 676
salt.modules.win_shadow, 678
salt.modules.win_status, 678
salt.modules.win_system, 679
salt.modules.win_timezone, 681
salt.modules.win_useradd, 682
salt.modules.xapi, 683
salt.modules.xmpp, 688
salt.modules.yumpkg, 688
salt.modules.zcbuildout, 694
salt.modules.zfs, 696
salt.modules.zpool, 697
salt.modules.zypper, 698

o

salt.output.grains, 931
salt.output.highstate, 931
salt.output.json_out, 932
salt.output.key, 932
salt.output.nested, 932
salt.output.no_out, 932
salt.output.no_return, 932
salt.output.overstatestage, 933
salt.output.pprint_out, 933
salt.output.raw, 933
salt.output.txt, 933
salt.output.virt_query, 933
salt.output.yaml_out, 934

p

salt.pillar.cmd_json, 899
salt.pillar.cmd_yaml, 899
salt.pillar.cobbler, 899
salt.pillar.django_orm, 900
salt.pillar.git_pillar, 902
salt.pillar.hiera, 902
salt.pillar.libvirt, 903
salt.pillar.mongo, 903
salt.pillar.pillar_ldap, 904
salt.pillar.puppet, 904
salt.pillar.reclass_adapter, 904
salt.pillar.virtkey, 905

r

salt.renderers.jinja, 885
salt.renderers.json, 885
salt.renderers.mako, 885
salt.renderers.py, 886
salt.renderers.pydsl, 886
salt.renderers.stateconf, 891
salt.renderers.wempy, 895
salt.renderers.yaml, 895
salt.returners.carbon_return, 703
salt.returners.cassandra_return, 703
salt.returners.couchdb_return, 704
salt.returners.local, 704
salt.returners.memcache_return, 704
salt.returners.mongo_future_return, 705
salt.returners.mongo_return, 706
salt.returners.mysql, 706
salt.returners.postgres, 707
salt.returners.redis_return, 709
salt.returners.sentry_return, 709
salt.returners.smtp_return, 710
salt.returners.sqlite3_return, 710
salt.returners.syslog_return, 711
salt.runners.cache, 915
salt.runners.doc, 916
salt.runners.fileserver, 916
salt.runners.jobs, 917
salt.runners.launchd, 917
salt.runners.lxc, 917
salt.runners.manage, 919

`salt.runners.network`, 920
`salt.runners.search`, 921
`salt.runners.state`, 921
`salt.runners.thin`, 922
`salt.runners.virt`, 922
`salt.runners.winrepo`, 923

s

`salt.states.alias`, 775
`salt.states.alternatives`, 775
`salt.states.apt`, 776
`salt.states.archive`, 776
`salt.states.augeas`, 777
`salt.states.aws_sqs`, 778
`salt.states.cloud`, 778
`salt.states.cmd`, 779
`salt.states.composer`, 785
`salt.states.cron`, 786
`salt.states.ddns`, 789
`salt.states.debconfmod`, 789
`salt.states.disk`, 791
`salt.states.dockerio`, 791
`salt.states.eselect`, 794
`salt.states.file`, 795
`salt.states.gem`, 809
`salt.states.git`, 810
`salt.states.gnomedesktop`, 811
`salt.states.grains`, 813
`salt.states.group`, 814
`salt.states.hg`, 814
`salt.states.host`, 815
`salt.states.iptables`, 816
`salt.states.keyboard`, 818
`salt.states.keystone`, 819
`salt.states.kmod`, 821
`salt.states.layman`, 821
`salt.states.libvirt`, 822
`salt.states.locale`, 822
`salt.states.lvm`, 822
`salt.states.lvs_server`, 823
`salt.states.lvs_service`, 824
`salt.states.makeconf`, 824
`salt.states.mdadm`, 825
`salt.states.memcached`, 826
`salt.states.modjk_worker`, 826
`salt.states.module`, 827
`salt.states.mongodb_database`, 828
`salt.states.mongodb_user`, 829
`salt.states.mount`, 829
`salt.states.mysql_database`, 830
`salt.states.mysql_grants`, 831
`salt.states.mysql_user`, 832
`salt.states.network`, 833
`salt.states.npm`, 836
`salt.states.ntp`, 837
`salt.states.openstack_config`, 838
`salt.states.pagerduty`, 838
`salt.states.pecl`, 839
`salt.states.pip_state`, 839
`salt.states.pkg`, 841
`salt.states.pkgng`, 844
`salt.states.pkgrepo`, 845
`salt.states.portage_config`, 847
`salt.states.ports`, 847
`salt.states.postgres_database`, 848
`salt.states.postgres_group`, 849
`salt.states.postgres_user`, 850
`salt.states.powerpath`, 852
`salt.states.process`, 852
`salt.states.quota`, 852
`salt.states.rabbitmq_cluster`, 853
`salt.states.rabbitmq_plugin`, 853
`salt.states.rabbitmq_policy`, 854
`salt.states.rabbitmq_user`, 854
`salt.states.rabbitmq_vhost`, 855
`salt.states.rbenv`, 856
`salt.states.rdp`, 857
`salt.states.reg`, 857
`salt.states.rvm`, 857
`salt.states.saltmod`, 859
`salt.states.selinux`, 861
`salt.states.service`, 861
`salt.states.ssh_auth`, 862
`salt.states.ssh_known_hosts`, 864
`salt.states.stateconf`, 865
`salt.states.status`, 865
`salt.states.supervisord`, 865
`salt.states.svn`, 866
`salt.states.sysctl`, 867
`salt.states.timezone`, 867
`salt.states.tomcat`, 868
`salt.states.user`, 869
`salt.states.virtualenv_mod`, 871
`salt.states.win_dns_client`, 872
`salt.states.win_firewall`, 872
`salt.states.win_network`, 872
`salt.states.win_path`, 873
`salt.states.win_servermanager`, 874
`salt.states.win_system`, 874
`salt.states.xmpp`, 875
`salt.states.zcbuildout`, 875

t

`salt.tops.cobbler`, 909
`salt.tops.ext_nodes`, 909
`salt.tops.mongo`, 910
`salt.tops.reclass_adapter`, 911

W

`salt.wheel.config`, 925
`salt.wheel.file_roots`, 925
`salt.wheel.key`, 926
`salt.wheel.pillar_roots`, 926