Newest additions first. Page 3 has plenty of space. See other systemd doc for 8x11 portrait format stuff. This turned into landscape to better hold tables.

***Command Equivalents - SysVinit and systemd***

|  |  |  |
| --- | --- | --- |
| **Action** | **SysVinit** | **systemd** |
| Start/stop/restart/reload/status of a service | service ntpd [start | stop | etc... ] | systemctl [ start | stop etc... ] ntpd.service |
| Restart a service only if already running | service ntpd condrestart | systemctl condrestart httpd.service |
| Enable or disable service on startup | chkconfig ntpd [ off | on ] | systemctl [enable | disable ] ntpd.service |
| Is service enabled at startup (this runlevel)? | chkconfig ntpd | systemctl is-enabled ntpd.service |
| List services that can be started or stopped  Used to list all the services and other units | ls /etc/rc.d/init.d/ | systemctl OR  systemctl list-unit-files --type=service OR  ls /lib/systemd/system/\*.service AND  ls /etc/systemd/system/\*.service |
| Print table of services listing runlevels each is configured on or off | chkconfig --list | systemctl list-unit-files --type=service  ls /etc/systemd/system/\*.wants/ |
| Print a table of services that will be started when booting into graphical mode | chkconfig --list | grep 5:on | systemctl list-dependencies graphical.target |
| List what levels this service is config'd on/ off | chkconfig ntpd --list | ls /etc/systemd/system/\*.wants/ntpd.service |
| Create a new service file or modify config | chkconfig ntpd --add | systemctl daemon-reload (this reloads systemd!) |
| Suspend the system | pm-suspend | systemctl suspend |
| Hibernate | pm-hibernate | systemctl hibernate |
| Follow the system log file | tail -f /var/log/messages (or /var/log/syslog) | journalctl -f |
| System halt | telinit 0, poweroff, halt | systemctl isolate poweroff.target | systemctl poweroff |
| Change to Single-user mode | telinit 1, s, single | systemctl isolate rescue.target (or runlevel1.target) |
| Change to Multi-user | telinit 2 | systemctl isolate multi-user.target (or runlevel2.target\*) |
| Change to Multi-user with Network | telinit 3 | systemctl isolate multi-user.target (or runlevel3.target) |
| Change to RunLevel 4 | telinit 4 | systemctl isolate multi-user.target (or runlevel4.target\*) |
| Change to Multi-user, w/ network, x11 | telinit 5 | systemctl isolate graphical.target (or runlevel5.target) |
| Reboot | telinit 6, reboot | systemctl isolate reboot.target | systemctl reboot |
| Emergency Shell | init emergency | emergency.target |
| Check current runlevel | runlevel | runlevel (deprecated) OR systemctl | **grep (script)** |
| Change default runlevel | sed s/^id:.\*:initdefault:/id:3:initdefault:/ | systemctl set-default multi-user.target |
| Set multi-user target on next boot | sed s/^id:.\*:initdefault:/id:3:initdefault:/ | ln -sf /lib/systemd/system/multi-user.target  /etc/systemd/system/default.target |
| Execute a systemd cmd on remote host |  | systemctl dummy.service start -H user@host |
| Check boot time |  | systemd-analyze or systemd-analyze time |
| Kill all processes related to a service |  | systemctl kill dummy |
| Get logs for events for today |  | journalctl --since=today |
| Hostname and other host information |  | hostnamectl |
| Date and time |  | timedatectl |

All recent versions of systemctl assume ".service" if left off. So, 'systemctl start myservicename.service' works like 'systemctl start myservicename'

Default systemd Fedora installs; 0, 1, 3, 5, and 6; have a 1:1 mapping with a specific systemd *target*.

\*\* If you use runlevels 2 or 4 it is suggested that you make a new named systemd *target* as /etc/systemd/system/$YOURTARGET that takes one of the existing runlevels as a base (you can look at /lib/systemd/system/graphical.target as an example), make a directory /etc/systemd/system/$YOURTARGET.wants, and then symlink the additional services that you want to enable into that directory.

Runlevels 2 and 4 are by default just "multi-user" runlevel3 in systemd until defined otherwise

***systemd Command Overview***

|  |  |
| --- | --- |
| Is systemd is installed on the system? Is it running? | **# systemd-run --version -- # ps -eaf | grep [s]ystemd** |
| List all the available units . [ \*.service, \*.mount, \*.socket, \*.device ] | **# systemctl list-unit-files** |
| List all running units. [ \*.service, \*.mount, \*.socket, \*.device ] | **# systemctl list-units** |
| List all failed units . [ \*.service, \*.mount, \*.socket, \*.device ] | **# systemctl --failed** |
| Analyze the systemd boot process. | **# systemd-analyze** |
| Analyze time taken by each process at boot. | **# systemd-analyze blame** |
| Analyze critical chain at boot (all or a specific service, etc) | **# systemd-analyze critical-chain** (OR **critical-chain httpd.service**) |
|  | "@" = Time after unit is active or started. "+" = Time unit takes to start |

***Using systemd to Manage Mountpoints, Sockets, Devices Just Like Services***

The general systemctl commands that work with services also do the same thing for mountpoints, sockets, and devices (which are seen as service types). Simply specify in the place of "name.service" the proper item, such as tmp.mount, cups.socket, or item.device. The list-unit-files option uses the **--type** directive such as **--type=device** or **--type=socket** accordingly. Starting stopping a mount point simply mounts and unmounts the mountpoint [ **systemctl list-unit-files --type=mount** will list all mountpoints, for example]

**# systemctl list-unit-files --type=socket**

**# systemctl [start | restart | stop | reload | status | is-active | enable | disable | is-enabled | mask | unmask] tmp.mount**

|  |  |
| --- | --- |
| How to enable, disable or check if turned on at boot time (auto start) | **# systemctl [is-active | enable | disable] httpd.service** |
| How to mask (making it impossible to start) or unmask a service | **# systemctl [mask | unmask] httpd.service** |
| List all services (including enabled and disabled). | **# systemctl list-unit-files --type=service** |
| Start, restart, stop, reload and check the status of a service | **# systemctl [start | restart | stop | reload | status] httpd.service** |
| Check all the configuration details of a service | **# systemctl show httpd** |
| Get a list of dependencies for a service | **# systemctl list-dependencies httpd.service** |
| How to a Kill a service using systemctl command. | **# systemctl kill httpd** |
| Is unit enabled or not right now ("is-active" is for the target's config) | **# systemctl is-enabled crond.service** |
| Get current CPU Shares of a Service (default CPUShare = 1024) | **# systemctl show -p CPUShares httpd.service** |
| Increase/decrease CPU share of a process | **# systemctl set-property httpd.service CPUShares=2000** |
| *No unit specified means default.target - Requires=, RequiresOverridable=, Requisite=, RequisiteOverridable=, Wants=, BindsTo= dependencies* | |
| List control groups hierarchically | **# systemd-cgls** |
| List control groups according to CPU, memory, Input and Output | **# systemd-cgtop** |

To enable a service, you must be currently running the target you want the service to start in.

For example, to turn on bluetooth.service in the graphical.target, you have to change to the graphical.target first with **isolate**, then run **enable**.

**systemctl isolate graphical.target** ; **systemctl enable bluetooth.service**

Makes a symlink **/etc/systemd/system/graphical.target.wants/bluetooth.servic**e pointing to **/usr/lib/systemd/system/bluetooth.service**

|  |  |
| --- | --- |
| How to start system rescue mode | **# systemctl rescue** |
| How to enter into emergency mode. | **# systemctl emergency** |
| List current default runlevel in use. | **# systemctl get-default** |
| Start Runlevel 5 aka graphical mode | **# systemctl isolate runlevel5.target** (OR **graphical.target**) |
| Set multiuser mode (runlevel 3) as default | **# systemctl set-default runlevel3.target** (OR **multiuser.target**) |
| *[ This set-default line creates a symlink* ***/etc/systemd/system/default.target*** *pointing to* ***/usr/lib/systemd/system/multiuser.target*** *]* | |
| Reboot, halt, suspend, hibernate or put system in hybrid-sleep | **# systemctl [reboot | halt | suspend | hibernate | hybrid-sleep]** |

Unit and target files in **/usr/lib/systemd/system/** are pointed to by symlinks placed in **/etc/systemd/system/**

Unit files enabled for a specific target will have a symlink in that target's "wants" directory, such as **/etc/systemd/system/multi-user.target.wants**

|  |  |  |  |
| --- | --- | --- | --- |
| Service unit directories hold symlinks to the real unit files like this: | | **/etc/systemd/system/XXXXXX.target.wants/bluetooth.service** | |
| Those symlinks point to the actual service unit files that reside here: | | **/usr/lib/systemd/system/bluetooth.service** | |
|  | | |  |
| The **default.target** file here is a target/runlevel symlink: | **/etc/systemd/system/default.target** | | |
| And the default.target symlink points to the actual target here: | **/usr/lib/systemd/system/XXXXXX.target** | | |

Running **systemctl isolate graphical.target** will not affect the default.target symlink, and merely switches the current runlevel (use **set-default**).

Running **systemctl disable myservice** basically does the same as **rm '/etc/systemd/system/multi-user.target.wants/service.myservice'**

Running **systemctl enable myservice** basically does the same as

**ln -s '/usr/lib/systemd/system/myservice.service' '/etc/systemd/system/multi-user.target.wants/service.myservice'**

The "**target.wants**" directories in **/usr/lib/systemd/system/** hold symlinks to the corresponding runlevel's unit files just like init's **/etc/rc.d/rc#.d/**

A target is itself a unit file, manages other unit files. Defaults are multi-user.target, graphical.target, rescue.target, emergency.target, poweroff.target, and reboot.target

**CASE STUDY- WHAT IS THE DIFF BTW TEE AND SYSTEMD-CAT: EXPLANATION**

Make a wrapper for process which I capture their standard output and pump to the journal via syslog.

Such wrapper already exists and is called systemd-cat.

You may use it as follows:

systemd-cat -t app1 /home/myself/logtest/app1 &

systemd-cat -t app2 /home/myself/logtest/app2 &

The argument to -t is an arbitrary identification string, analogous (equivalent) to syslog's identifier.

It is also possible to use systemd-cat in a shell pipeline like this:

/home/myself/logtest/app1 |& systemd-cat -t app1 &

(The |& is a bash construction to pipeline both stdout and stderr.)

However, the first form is preferable as it avoids spawning an extra process and doing an extra copy of all logged data.

|  |  |  |
| --- | --- | --- |
| **Sysvinit Runlevel** | **Systemd Target** | **Notes** |
| 0 | runlevel0.target, poweroff.target | Halt the system. |
| 1, s, single | runlevel1.target, rescue.target | Single user mode. |
| 2, 4 | runlevel2.target, runlevel4.target, multi-user.target | User-defined/Site-specific runlevels. By default, identical to 3. |
| 3 | runlevel3.target, multi-user.target | Multi-user, non-graphical. Users can usually login via multiple consoles or via the network. |
| 5 | runlevel5.target, graphical.target | Multi-user, graphical. Usually has all the services of runlevel 3 plus a graphical login. |
| 6 | runlevel6.target, reboot.target | Reboot |
| emergency | emergency.target | Emergency shell |

Changing runlevels:

|  |  |  |
| --- | --- | --- |
| **Sysvinit Command** | **Systemd Command** | **Notes** |
| telinit 3 | systemctl isolate multi-user.target  OR systemctl isolate runlevel3.target  OR telinit 3 | Change to multi-user run level. |
| sed s/^id:.\*:initdefault:/id:3:initdefault:/ | ln -sf /lib/systemd/system/multi-user.target /etc/systemd/system/default.target | Set to use multi-user runlevel on next reboot. |

Kernel Options:

The above systemd targets can be used when booting. At the GRUB menu, edit the selection to add "systemd.unit=target" (without the double-quotation marks) as a kernel option where target is one of the above. (For example, "rescue.target".)

Tip: the ".target" extention is optional. The "systemd.unit=rescue" kernel option works the same as "systemd.unit=rescue.target".

**Emergency vs Rescue modes:**

> Hi everyone

>

> I wonder that emergency.target/emergency.service/emergency mode is good for.

>

> Afaics, it doesn't offer anything that rescue mode doesn't also offer.

> I find this situation a bit confusing.

>

> Can anyone enlighten me why we need two different modes which are

> basically the same?

Emergency mode means the only running process is a shell, besides

systemd itself.

rescue mode means all early-boot services have been started, all

mounts have been established, and then gives you a shell.

emergency mode is a bit like booting with init=/bin/sh except that you

actually have systemd up properly, and can start services bit-by-bit

if you like. emergency mode is also entered if fsck fails, as in that

case rescue mode is unachievable, as we cannot mount the failed

disks...

rescue mode is usually what admins want to boot into for single-user

maintainance tasks. The only reason when you want to boot into

emergency mode instead if you have some fuckup with your disks.

Note that sysvinit had the very same distinction, though the feature

is little-known. Booting sysvinit with "emergency" on the kernel

cmdline would give you just sysvinit as PID1 plus /bin/sh forked

off. Booting sysvinit with "1" on the kernel cmdline would give you

runlevel 1 services including mounts, and then a shell. In fact,

systemd understand the same kernel cmdline keywords and does the right

thing..

| **Action** | **Debian Init** | **Ubuntu Upstart** | **Red Hat/CentOS Init** | **Systemd** |
| --- | --- | --- | --- | --- |
| Start service | /etc/init.d/nginx start | initctl start nginx  /etc/init.d/nginx start | /etc/init.d/nginx start | systemctl start nginx.service |
| Stop service | /etc/init.d/nginx stop | initctl stop nginx  /etc/init.d/nginx stop | /etc/init.d/nginx stop | systemctl stop nginx.service |
| Restart service | /etc/init.d/nginx restart | initctl restart nginx  /etc/init.d/nginx restart | /etc/init.d/nginx restart | systemctl restart nginx.service |
| Enable service at boot / startup | update-rc.d nginx defaults | edit /etc/init/nginx.conf,  add start on local-filesystems line  update-rc.d nginx defaults | chkconfig nginx on  ntsysv | systemctl enable nginx.service |
| Disable service at startup / boot | update-rc.d nginx remove | echo 'manual' > /etc/init/nginx.override  update-rc.d nginx remove | chkconfig nginx off  ntsysv | systemctl disable nginx.service |
| List all available services | ls /etc/init.d/ | initctl list | chkconfig --list  ntsysv | systemctl list-units -t service --all |
| Get a service status | /etc/init.d/nginx status | initctl status nginx | /etc/init.d/nginx status | systemctl status nginx.service |

Press the power button on your system, and after few moments you see the Linux login prompt.

Have you ever wondered what happens behind the scenes from the time you press the power button until the Linux login prompt appears?

The following are the 6 high level stages of a typical Linux boot process.  
  


**1. BIOS**

* BIOS stands for Basic Input/Output System
* Performs some system integrity checks
* Searches, loads, and executes the boot loader program.
* It looks for boot loader in floppy, cd-rom, or hard drive. You can press a key (typically F12 of F2, but it depends on your system) during the BIOS startup to change the boot sequence.
* Once the boot loader program is detected and loaded into the memory, BIOS gives the control to it.
* So, in simple terms BIOS loads and executes the MBR boot loader.

**2. MBR**

* MBR stands for Master Boot Record.
* It is located in the 1st sector of the bootable disk. Typically /dev/hda, or /dev/sda
* MBR is less than 512 bytes in size. This has three components 1) primary boot loader info in 1st 446 bytes 2) partition table info in next 64 bytes 3) mbr validation check in last 2 bytes.
* It contains information about GRUB (or LILO in old systems).
* So, in simple terms MBR loads and executes the GRUB boot loader.

**3. GRUB**

* GRUB stands for Grand Unified Bootloader.
* If you have multiple kernel images installed on your system, you can choose which one to be executed.
* GRUB displays a splash screen, waits for few seconds, if you don’t enter anything, it loads the default kernel image as specified in the grub configuration file.
* GRUB has the knowledge of the filesystem (the older Linux loader LILO didn’t understand filesystem).
* Grub configuration file is /boot/grub/grub.conf (/etc/grub.conf is a link to this). The following is sample grub.conf of CentOS.

#boot=/dev/sda

default=0

timeout=5

splashimage=(hd0,0)/boot/grub/splash.xpm.gz

hiddenmenu

title CentOS (2.6.18-194.el5PAE)

root (hd0,0)

kernel /boot/vmlinuz-2.6.18-194.el5PAE ro root=LABEL=/

initrd /boot/initrd-2.6.18-194.el5PAE.img

* As you notice from the above info, it contains kernel and initrd image.
* So, in simple terms GRUB just loads and executes Kernel and initrd images.

**4. Kernel**

* Mounts the root file system as specified in the “root=” in grub.conf
* Kernel executes the /sbin/init program
* Since init was the 1st program to be executed by Linux Kernel, it has the process id (PID) of 1. Do a ‘ps -ef | grep init’ and check the pid.
* initrd stands for Initial RAM Disk.
* initrd is used by kernel as temporary root file system until kernel is booted and the real root file system is mounted. It also contains necessary drivers compiled inside, which helps it to access the hard drive partitions, and other hardware.

**5. Init**

* Looks at the /etc/inittab file to decide the Linux run level.
* Following are the available run levels
  + 0 – halt
  + 1 – Single user mode
  + 2 – Multiuser, without NFS
  + 3 – Full multiuser mode
  + 4 – unused
  + 5 – X11
  + 6 – reboot
* Init identifies the default initlevel from /etc/inittab and uses that to load all appropriate program.
* Execute ‘grep initdefault /etc/inittab’ on your system to identify the default run level
* If you want to get into trouble, you can set the default run level to 0 or 6. Since you know what 0 and 6 means, probably you might not do that.
* Typically you would set the default run level to either 3 or 5.

**6. Runlevel programs**

* When the Linux system is booting up, you might see various services getting started. For example, it might say “starting sendmail …. OK”. Those are the runlevel programs, executed from the run level directory as defined by your run level.
* Depending on your default init level setting, the system will execute the programs from one of the following directories.
  + Run level 0 – /etc/rc.d/rc0.d/
  + Run level 1 – /etc/rc.d/rc1.d/
  + Run level 2 – /etc/rc.d/rc2.d/
  + Run level 3 – /etc/rc.d/rc3.d/
  + Run level 4 – /etc/rc.d/rc4.d/
  + Run level 5 – /etc/rc.d/rc5.d/
  + Run level 6 – /etc/rc.d/rc6.d/
* Please note that there are also symbolic links available for these directory under /etc directly. So, /etc/rc0.d is linked to /etc/rc.d/rc0.d.
* Under the /etc/rc.d/rc\*.d/ directories, you would see programs that start with S and K.
* Programs starts with S are used during startup. S for startup.
* Programs starts with K are used during shutdown. K for kill.
* There are numbers right next to S and K in the program names. Those are the sequence number in which the programs should be started or killed.
* For example, S12syslog is to start the syslog deamon, which has the sequence number of 12. S80sendmail is to start the sendmail daemon, which has the sequence number of 80. So, syslog program will be started before sendmail.

There you have it. That is what happens during the Linux boot process.

***Contents of the systemd Package***

**Installed programs:** bootctl, busctl, coredumpctl, halt, hostnamectl, init, journalctl, kernel-install, localectl, loginctl, machinectl, networkctl, poweroff, reboot, runlevel, shutdown, systemctl, systemd-analyze, systemd-ask-password, systemd-cat, systemd-cgls, systemd-cgtop, systemd-delta, systemd-detect-virt, systemd-escape, systemd-hwdb, systemd-inhibit, systemd-machine-id-setup, systemd-mount, systemd-notify, systemd-nspawn, systemd-path, systemd-resolve, systemd-run, systemd-socket-activate, systemd-stdio-bridge, systemd-tmpfiles, systemd-tty-ask-password-agent, telinit, timedatectl, and udevadm

**Installed libraries:** libnss\_myhostname.so.2, libnss\_mymachines.so.2, libnss\_resolve.so.2, libnss\_systemd.so.2, libsystemd.so, libsystemd-shared-231.so, and libudev.so

**Installed directories:** /etc/binfmt.d, /etc/init.d, /etc/kernel, /etc/modules-load.d, /etc/sysctl.d, /etc/systemd, /etc/tmpfiles.d, /etc/udev, /etc/xdg/systemd, /lib/systemd, /lib/udev, /usr/include/systemd, /usr/lib/binfmt.d, /usr/lib/kernel, /usr/lib/modules-load.d, /usr/lib/sysctl.d, /usr/lib/systemd, /usr/lib/tmpfiles.d, /usr/share/doc/systemd-234, /usr/share/factory, /usr/share/systemd, /var/lib/systemd, and /var/log/journal

|  |  |
| --- | --- |
| **bootctl** | Query the firmware and boot manager settings |
| **busctl** | Review logs and monitor the D-Bus bus |
| **coredumpctl** | Retrieve coredumps from the systemd Journal |
| **halt** | Normally invokes **shutdown** with the *-h* option, except when already in run-level 0, then it tells the kernel to halt the system; it notes in the file /var/log/wtmp that the system is being brought down |
| **hostnamectl** | Query and change the system hostname and related settings |
| **init** | The first process to be started when the kernel has initialized the hardware which takes over the boot process and starts all the proceses it is instructed to |
| **journalctl** | Query the contents of the systemd Journal |
| **kernel-install** | Add and remove kernel and initramfs images to and from /boot |
| **localectl** | Query and change the system locale and keyboard layout settings |
| **loginctl** | Review logs and control the state of the systemd Login Manager |
| **machinectl** | Review logs and control the state of the systemd Virtual Machine and Container Registration Manager |
| **networkctl** | Review logs and state of the network links as seen by systemd-networkd |
| **poweroff** | Tells the kernel to halt the system and switch off the computer (see **halt**) |
| **reboot** | Tells the kernel to reboot the system (see **halt**) |
| **runlevel** | Reports the previous and the current run-level, as noted in the last run-level record in /var/run/utmp |
| **shutdown** | Brings the system down in a secure way, signaling all processes and notifying all logged-in users |
| **systemctl** | Review logs and control the state of the systemd system and service manager |
| **systemd-analyze** | Determine system boot-up performance of the current boot |
| **systemd-ask-password** | Query a system password or passphrase from the user, using a question message specified on the command line |
| **systemd-cat** | Connect STDOUT and STDERR of a process with the Journal |
| **systemd-cgls** | Recursively shows the contents of the selected Linux control group hierarchy in a tree |
| **systemd-cgtop** | Shows the top control groups of the local Linux control group hierarchy, ordered by their CPU, memory and disk I/O load |
| **systemd-delta** | Identify and compare configuration files in /etc that override default counterparts in /usr |
| **systemd-detect-virt** | Detects execution in a virtualized environment |
| **systemd-escape** | Escape strings for inclusion in systemd unit names |
| **systemd-hwdb** | Manage hardware database (hwdb) |
| **systemd-inhibit** | Execute a program with a shutdown, sleep or idle inhibitor lock taken |
| **systemd-machine-id-setup** | Used by system installer tools to initialize the machine ID stored in /etc/machine-id at install time with a randomly generated ID |
| **systemd-mount** | A tool to temporarily mount or auto-mount a drive. |
| **systemd-notify** | Used by daemon scripts to notify the init system about status changes |
| **systemd-nspawn** | Run a command or OS in a light-weight namespace container |
| **systemd-path** | Query system and user paths |
| **systemd-resolve** | Resolve domain names, IPV4 and IPv6 addresses, DNS resource records, and services |
| **systemd-run** | Create and start a transient .service or a .scope unit and run the specified command in it |
| **systemd-socket-activate** | A tool to listen on socket devices and launch a process upon connection. |
| **systemd-tmpfiles** | Creates, deletes and cleans up volatile and temporary files and directories, based on the configuration file format and location specified in tmpfiles.d directories |
| **systemd-tty-ask-password-agent** | Used to list or process pending systemd password requests |
| **telinit** | Tells **init** which run-level to change to |
| **timedatectl** | Query and change the system clock and its settings |
| **udevadm** | Generic Udev administration tool: controls the udevd daemon, provides info from the Udev database, monitors uevents, waits for uevents to finish, tests Udev configuration, and triggers uevents for a given device |
| **libsystemd** | systemd utility library |
| **libudev** | A library to access Udev device information |

from http://www.linuxfromscratch.org/lfs/view/systemd/chapter06/systemd.html

**Processes and cgroups**

systemd organizes processes with cgroups, this is a Linux kernel feature to limit, police and account the resource usage of certain processes (actually process groups). Compared to other approaches like the ‘**nice**’ command or **/etc/security/limits.conf**, cgroups are more flexible.

Control groups can be used in multiple ways:

* create and manage them on the fly using tools like **cgcreate, cgexec, cgclassify** etc
* the “rules engine daemon”, to automatically move certain users/groups/commands to groups (**/etc/cgrules.conf** and **/usr/lib/systemd/system/cgconfig.service**)
* through other software such as Linux Containers (LXC) virtualization

So Control Groups are two things: **(A)** *a way to hierarchally group and label processes*, and **(B)** *a way to then apply resource limits* to these groups. systemd only requires the former (A), and not the latter (B).

You can see the sue of cgroups with the ps command, which has been updated to show cgroups. Run this command to see which service owns which processes:

|  |
| --- |
| $ **ps** xawf -eo pid,user,cgroup,args  PID USER CGROUP COMMAND  2 root - **[**kthreadd**]**  3 root - \\_ **[**ksoftirqd**/**0**]**  **[**...**]**  4281 root - \\_ **[**flush-8:0**]**  1 root name=systemd:**/**systemd-1 **/**sbin**/**init |

**Systemd Command Examples**

Where the binaries and libraries of systemd and systemctl?

# whereis systemd

systemd: /usr/lib/systemd /etc/systemd /usr/share/systemd /usr/share/man/man1/systemd.1.gz

# whereis systemctl

systemctl: /usr/bin/systemctl /usr/share/man/man1/systemctl.1.gz

Welcome to emergency mode! After logging in, type "journalctl -xb" to view

system logs, "systemctl reboot" to reboot, "systemctl default" to try again

to boot into default mode.

Runlevels refresh:

Runlevel 0 : Shut down and Power off the system

Runlevel 1 : Rescue/Maintainance Mode

Runlevel 3 : multiuser, no-graphic system

Runlevel 4 : multiuser, no-graphic system

Runlevel 5 : multiuser, graphical system

Runlevel 6 : Shutdown and Reboot the machine

When masking or unmasking, another way is to do it - make the alias appear alive, but pointed to /dev/null instead of a service, socket, etc., file.

**# ln -s '/dev/null' '/etc/systemd/system/cups.socket'**

**About systemctl kill - why not just use traditional kill?**

--kill-who=

When used with kill, choose which processes to send a signal to. Must be one of main, control or all to select whether to kill only the main process, the control process or all processes of the unit. The main process of the unit is the one that defines the life-time of it. A control process of a unit is one that is invoked by the manager to induce state changes of it. For example, all processes started due to the ExecStartPre=, ExecStop= or ExecReload= settings of service units are control processes. Note that there is only one control process per unit at a time, as only one state change is executed at a time. For services of type Type=forking, the initial process started by the manager for ExecStart= is a control process, while the process ultimately forked off by that one is then considered the main process of the unit (if it can be determined). This is different for service units of other types, where the process forked off by the manager for ExecStart= is always the main process itself. A service unit consists of zero or one main process, zero or one control process plus any number of additional processes. Not all unit types manage processes of these types however. For example, for mount units, control processes are defined (which are the invocations of /usr/bin/mount and /usr/bin/umount), but no main process is defined. If omitted, defaults to all.

-s, --signal=

When used with kill, choose which signal to send to selected processes. Must be one of the well-known signal specifiers such as SIGTERM, SIGINT or SIGSTOP. If omitted, defaults to SIGTERM.

When you set CPUShare for a service, a directory with the name of service is created (httpd.service.d) with a file 90-CPUShares.conf for CPUShare Limit information. You may view the file as:

**# vi /etc/systemd/system/httpd.service.d/90-CPUShares.conf**

[Service]

CPUShares=2000

*Shows units required and wanted by the specified unit. This recursively lists units following the Requires=, RequiresOverridable=, Requisite=, RequisiteOverridable=, Wants=, BindsTo= dependencies. If no unit is specified, default.target is implied.*

*By default, only target units are recursively expanded. When --all is passed, all other units are recursively expanded as well.*