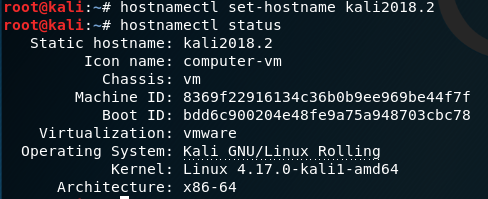
# **调整网络**：

## 修改主机名称

hostnamectl ｛status|set-hostname “hostname”｝





## 网络配置--Nmcli命令基础

Kali linux2018.2 \ Red Hat Enterprise Linux 7 \ CentOS 7 以后，默认的网络服务由 NetworkManager 提供。以前的ifconfig命令会逐渐淘汰。最新的网络管理命令是叫**nmcli**。nmcli比ifconfig命令要强大、复杂的多。nmcli命令管理的三类对象：

device //管理网络接口，是物理设备，也就是网卡

connection //配置网络连接，偏重于逻辑设置，也就是网卡的配件文件

networking //管理网络状态

注意：多个connection可以应用到同一个device，但同一时间只能启用其中一个connection。这样的好处是针对一个网络接口，我们可以设置多个网络连接，比如静态IP和动态IP，再根据需要up相应connection。

### 网络设备（device）的管理

#### 网卡的命名方式

2位的网络设备类型 + 1位的位置扫述 + 1位的编号

* 网络设备类型

en: Ethernet 有线局域网

wl: wlan 无线局域网

ww: wwan无线广域网

。。。

* 位置描述：

o<index>: 集成设备（例如主板自带）的设备索引号

s<slot>: 扩展槽（例如PLC插槽）的索引号

x<MAC>: 基于MAC地址的命名

p<bus>s<slot>: enp2s1

* 编号

以一位数字表示。

示例：

eno1 ： 代表由主板 BIOS 内置的网卡

ens1 ： 代表由主板 BIOS 内置的 PCI-E 界面的网卡

enp2s0 ：代表 PCI-E 界面的独立网卡，可能有多个插孔，因此会有 s0,s1...的编号。

eth0 ： 如果上述的名称都不适用， 就回到原本的默认网卡编号

1. **网卡设备的命名过程：**

第一步：udev, 辅助工具程序/lib/udev/rename\_device, /usr/lib/udev/rules.d/60-net.rules

第二步：biosdevname 会根据/usr/lib/udev/rules.d/71-biosdevname.rules

第三步：通过检测网络接口设备，根据/usr/lib/udev/rules.d/75-net-description

ID\_NET\_NAME\_ONBOARD

ID\_NET\_NAME\_SLOT

ID\_NET\_NAME\_PATH

1. **改回传统命名方式**
2. 编辑/etc/default/grub配置文件

GRUB\_CMDLINE\_LINUX="net.ifnames=0 rhgb quiet"或：修改/boot/grub2/grub.cfg

1. 为grub2生成其配置文件

grub2-mkconfig -o /etc/grub2.cfg

1. 重启系统

#### 网卡管理常用命令

1. 查看网卡信息

# nmcli device //查看所有网卡的状态

DEVICE TYPE STATE CONNECTION

virbr0 bridge connected virbr0

eth0 ethernet connected eth0

eth1 ethernet connected static

eth2 ethernet connected Wired connection 1

lo loopback unmanaged --

virbr0-nic tun unmanaged --

1. 查看网卡详细信息

# nmcli device show

//查看所有网卡的详细信息，后面跟上网卡名可单独查看某块网卡的详细信息,略长，省略：）

1. 激活或关闭网卡

# nmcli device connect eth0

//激活eth1网卡，并自动连接到配置。

# nmcli device disconnect eth0

//断开eth1网卡

1. 修改网卡属性，修改的配置立即生效，并不修改配置文件，重启网卡失效。

# nmcli device modify eth1

1. 删除虚拟网卡，无法删除物理网卡

# nmcli device delete virbr0

### connection 配置网络连接

1. 查看所有连接

# nmcli connection show

NAME UUID TYPE DEVICE

eth0 702eabdc-6623-4d51-b864-f6bfdb521b55 802-3-ethernet eth0

virbr0 40217849-abce-47d2-b2c6-dc5b3f8beadb bridge virbr0

eth1 9c92fad9-6ecb-3e6c-eb4d-8a47c6f50c04 802-3-ethernet --

1. 仅查看活跃连接

# nmcli connection show --active

NAME UUID TYPE DEVICE

eth0 702eabdc-6623-4d51-b864-f6bfdb521b55 802-3-ethernet eth0

virbr0 40217849-abce-47d2-b2c6-dc5b3f8beadb bridge virbr0

1. 创建新连接**INTERNET0**，IP地址通过DHCP自动获取

# nmcli connection add con-name INTERNET0 type ethernet ifname eth0

Connection 'default' (db8ef10f-c848-461a-9098-09501f20d32b) successfully added.

参数

con-name: 指定连接名

type: 指定连接类型

ifname: 指定连接到那块网卡

1. 删除连接

# nmcli connection delete default

Connection 'default' (db8ef10f-c848-461a-9098-09501f20d32b) successfully deleted.

1. 创建新链接static，并指定静态IP地址。

命令完成后，会在/etc/sysconfig/network-scripts/下生成一个配置文件，但是并未生效

# nmcli connection add con-name eth0-static ifname eth0 autoconnect yes type ethernet ipv4.method manual ipv4.addresses 192.168.44.4/24 gw4 192.168.44.2

Connection 'static' (e42075e8-4261-4890-b499-818b96d7411f) successfully added.

# nmcli connection show

NAME UUID TYPE DEVICE

eth0 702eabdc-6623-4d51-b864-f6bfdb521b55 802-3-ethernet eth0

eth1 9c92fad9-6ecb-3e6c-eb4d-8a47c6f50c04 802-3-ethernet eth1

virbr0 40217849-abce-47d2-b2c6-dc5b3f8beadb bridge virbr0

static e42075e8-4261-4890-b499-818b96d7411f 802-3-ethernet --

参数

autoconnect //是否自动链接到网卡，参数（yes/no）。

ipv4.method //设置IP地址是自动获取还是静态指定。

ipv4.addresses //指定静态IPV4地址。

gw4 //指定静态IPV4网关地址。

1. 启动static连接

# nmcli connection up static

Connection successfully activated (D-Bus active path: /org/freedesktop/NetworkManager/ActiveConnection/12)

# nmcli connection show //再次查看连接

NAME UUID TYPE DEVICE

eth0 702eabdc-6623-4d51-b864-f6bfdb521b55 802-3-ethernet eth0

static e42075e8-4261-4890-b499-818b96d7411f 802-3-ethernet eth1

virbr0 40217849-abce-47d2-b2c6-dc5b3f8beadb bridge virbr0

eth1 9c92fad9-6ecb-3e6c-eb4d-8a47c6f50c04 802-3-ethernet --

1. 修改连接配置

# nmcli connection modify "static" ipv4.dns 8.8.8.8

//为static连接添加一个dns地址。

# nmcli connection modify "static" ipv4.addresses 192.168.0.9

//修改static连接的IP地址。

# nmcli con down static

//关闭连接配置。

# nmcli con up static

//启动连接配置生效。

常用参数和网卡配置文件参数的对应关系

nmcli con add/mod 网卡配置文件

ipv4.method manual BOOTPROTO=none

ipv4.method auto BOOTPROTO=dhcp

ipv4.addresses IPADDR、PREFIX、GATEWAY

ipve.dns DNS

ipv4.dns-search example.com DOMAIN=example.com

ipv4.ignore-auto-dns true PEERDNS=no

connection.autoconnect yes ONBOOT=yes

connection.id eth0 NAME=eth0

connection.interface-name eth0 DEVICE=eth0

802-3-ethernet.mac-address HWADDR= . . .

ipv4.ignore-auto-dns yes 手动设置dns

### networking 管理网络状态

# nmcli networking //off关闭网络，on开启网络

connectivity help off on

# nmcli networking connectivity //查看网络状态

full //full启动，none关闭

## Nmcli命令部分示例

### **动态获取IP**

**# nmcli con add con-name eno0 type ethernet ifname eth0**

### **DHCP动态获取IP**

# nmcli con mod eno0 ipv4.method manual | dhcp //修改IP地址是静态还是DHCP

# nmcli con mod eno0 connection.autoconnect yes //修改为自动连接

### ****指定静态IP****

**# nmcli connection add con-name INTERNET0 ifname eth0 autoconnect yes type ethernet ipv4.addresses 192.168.44.4/24 gw4 192.168.44.2**

//一次性建立连接、配置IP和网关。

# nmcli con mod eno0 ipv4.addresses “172.25.X.10/24 172.25.X.254”

//修改IP配置及网关

# nmcli con mod eno0 ipv4.gateway 10.1.0.1 //只修改默认网关

# nmcli con mod eno0 +ipv4.addresses 10.10.10.10/16 //添加第二个IP地址

### DNS设置

# nmcli con mod eno0 ipv4.dns 114.114.114.114 //添加dns1

# nmcli con mod eno0 +ipv4.dns  8.8.8.8 //添加dns2

# nmcli con mod eno0 -ipv4.dns  8.8.8.8 //删除dns

DNS配置文件是/etc/resolv.conf文件，PEERDNS=no 表示当IP通过dhcp自动获取时，dns仍是手动设置，不自动获取。等价于下面命令：

# nmclicon mod “system eth0” ipv4.ignore-auto-dns yes

### 启用/关闭所有的网络连接

# nmcli net on|of

### 禁用网络设备并防止自动激活

# nmcli con dis eno0

### 网络连接图形工具

# nm-connection-editor

### 网络接口配置图形工具

# nmtui

### 网络配置案例

#### 手工修改IP地址

# nmcli connection add con-name INTERNET0 ifname eth0 type ethernet autoconnect yes ipv4.addresses 192.168.44.10/24 gw4 192.168.44.2 ipv4.dns 192.168.44.2

//一次性新建一个连接，设置好IP、子网掩码、网关、DNS。

连接“INTERNET0”(9ab357d8-a2fc-4029-8117-312373e88ae5) 已成功添加。

# nmcli connection show //显示所有连接，INTERNET0已经建立但没有与网卡绑定。

NAME UUID TYPE DEVICE

Wired connection 1 e59fc531-eae7-4486-8b5d-ea9a578450a6 ethernet eth0

INTERNET0 9ab357d8-a2fc-4029-8117-312373e88ae5 ethernet --

# nmcli connection up INTERNET0 //启用INTERNET0连接。

连接已成功激活（D-Bus 活动路径：/org/freedesktop/NetworkManager/ActiveConnection/2）

# nmcli connection show //再次显示所有连接，INTERNET0已经、与网卡绑定。

NAME UUID TYPE DEVICE

INTERNET0 9ab357d8-a2fc-4029-8117-312373e88ae5 ethernet eth0

Wired connection 1 e59fc531-eae7-4486-8b5d-ea9a578450a6 ethernet --

# nmcli device show eth0 //启用网卡的状态。

GENERAL.DEVICE: eth0

GENERAL.TYPE: ethernet

GENERAL.HWADDR: 00:0C:29:99:B6:27

GENERAL.MTU: 1500

GENERAL.STATE: 100 (连接的)

GENERAL.CONNECTION: INTERNET0

GENERAL.CON-PATH: /org/freedesktop/NetworkManager/ActiveConnection/1

WIRED-PROPERTIES.CARRIER: 开

IP4.ADDRESS[1]: 192.168.44.10/24

IP4.ADDRESS[2]: 192.168.44.132/24

IP4.GATEWAY: 192.168.44.2

IP4.ROUTE[1]: dst = 192.168.44.0/24, nh = 0.0.0.0, mt = 100

IP4.ROUTE[2]: dst = 192.168.44.0/24, nh = 0.0.0.0, mt = 100

IP4.ROUTE[3]: dst = 0.0.0.0/0, nh = 192.168.44.2, mt = 100

IP4.DNS[1]: 92.168.44.2

IP4.DOMAIN[1]: localdomain

IP4.WINS[1]: 192.168.44.2

IP6.ADDRESS[1]: fe80::ef4:db96:1dd6:ea2/64

IP6.GATEWAY: --

IP6.ROUTE[1]: dst = fe80::/64, nh = ::, mt = 100

IP6.ROUTE[2]: dst = ff00::/8, nh = ::, mt = 256, table=255

#### 使用nmcli实现bonding

1. 添加bonding接口

# nmcli connection add con-name mybond0 type bond ifname mybond0 mode active-backup

**指定bonding接口的工作模式，active-backup表示主备模式。**bonding的7种工作模式，顺序（0-6）

balance-rr //负载均衡模式，常用模式

active-backup //主备模式

balance-xor //XOR Hash负载分担，和交换机的聚合强制不协商方式配合。

Broadcast //从所有物理接口发送数据包。

802.3ad //表示支持802.3ad协议，和交换机的聚合LACP方式配合。

balance-tlb //根据每个slave的负载情况选择slave发送。

balance-alb //在balance-tlb的基础上增加了rlb。

1. 为bonding0接口添加从属接口

# nmcli con add type bond-slave ifname eth1 master mybond0

# nmcli con add type bond-slave ifname eth2 master mybond0

1. 启动bonding接口

# nmcli connection up bond-slave-eth1 //启动bonding网卡需要先启动从属网卡。

# nmcli connection up bond-slave-eth2

# nmcli con up mybond0 //启动mybond0网卡

# nmcli connection //查看连接信息

NAME UUID TYPE DEVICE

bond-slave-eth1 71c9dad8-7120-4b30-af01-be53337a359e 802-3-ethernet eth1

bond-slave-eth2 9103d9ea-fa2b-4df4-a5f4-3d615c53a94a 802-3-ethernet eth2

eth0 702eabdc-6623-4d51-b864-f6bfdb521b55 802-3-ethernet eth0

mybond0 ac079cc4-6a74-45ed-ab82-9c58a1415984 bond mybond0

virbr0 90d0d99c-4abd-4328-be7d-8d67812a3f68 bridge virbr0

Wired connection 1 2d3af334-7e54-311f-9abd-d05fc0143e43 802-3-ethernet --

eth1 9c92fad9-6ecb-3e6c-eb4d-8a47c6f50c04 802-3-ethernet --

static e42075e8-4261-4890-b499-818b96d7411f 802-3-ethernet --

## 启用SSH服务

1. 编辑ssh的配置文件

# vi /etc/ssh/sshd\_config //修改config文件内容

PermitRootLogin yes //修改内容

PasswordAuthentication yes //修改内容

修改完成示例：

# $OpenBSD: sshd\_config,v 1.103 2018/04/09 20:41:22 tj Exp $

# This is the sshd server system-wide configuration file. See

# sshd\_config(5) for more information.

# This sshd was compiled with PATH=/usr/bin:/bin:/usr/sbin:/sbin

# The strategy used for options in the default sshd\_config shipped with

# OpenSSH is to specify options with their default value where

# possible, but leave them commented. Uncommented options override the

# default value.

#Port 22

#AddressFamily any

#ListenAddress 0.0.0.0

#ListenAddress ::

#HostKey /etc/ssh/ssh\_host\_rsa\_key

#HostKey /etc/ssh/ssh\_host\_ecdsa\_key

#HostKey /etc/ssh/ssh\_host\_ed25519\_key

# Ciphers and keying

#RekeyLimit default none

# Logging

#SyslogFacility AUTH

#LogLevel INFO

# Authentication:

#LoginGraceTime 2m

PermitRootLogin yes

#StrictModes yes

#MaxAuthTries 6

#MaxSessions 10

#PubkeyAuthentication yes

# Expect .ssh/authorized\_keys2 to be disregarded by default in future.

#AuthorizedKeysFile .ssh/authorized\_keys .ssh/authorized\_keys2

#AuthorizedPrincipalsFile none

#AuthorizedKeysCommand none

#AuthorizedKeysCommandUser nobody

# For this to work you will also need host keys in /etc/ssh/ssh\_known\_hosts

#HostbasedAuthentication no

# Change to yes if you don't trust ~/.ssh/known\_hosts for

# HostbasedAuthentication

#IgnoreUserKnownHosts no

# Don't read the user's ~/.rhosts and ~/.shosts files

#IgnoreRhosts yes

# To disable tunneled clear text passwords, change to no here!

PasswordAuthentication yes

#PermitEmptyPasswords no

# Change to yes to enable challenge-response passwords (beware issues with

# some PAM modules and threads)

ChallengeResponseAuthentication no

# Kerberos options

#KerberosAuthentication no

#KerberosOrLocalPasswd yes

#KerberosTicketCleanup yes

#KerberosGetAFSToken no

# GSSAPI options

#GSSAPIAuthentication no

#GSSAPICleanupCredentials yes

#GSSAPIStrictAcceptorCheck yes

#GSSAPIKeyExchange no

# Set this to 'yes' to enable PAM authentication, account processing,

# and session processing. If this is enabled, PAM authentication will

# be allowed through the ChallengeResponseAuthentication and

# PasswordAuthentication. Depending on your PAM configuration,

# PAM authentication via ChallengeResponseAuthentication may bypass

# the setting of "PermitRootLogin without-password".

# If you just want the PAM account and session checks to run without

# PAM authentication, then enable this but set PasswordAuthentication

# and ChallengeResponseAuthentication to 'no'.

UsePAM yes

#AllowAgentForwarding yes

#AllowTcpForwarding yes

#GatewayPorts no

X11Forwarding yes

#X11DisplayOffset 10

#X11UseLocalhost yes

#PermitTTY yes

PrintMotd no

#PrintLastLog yes

#TCPKeepAlive yes

#PermitUserEnvironment no

#Compression delayed

#ClientAliveInterval 0

#ClientAliveCountMax 3

#UseDNS no

#PidFile /var/run/sshd.pid

#MaxStartups 10:30:100

#PermitTunnel no

#ChrootDirectory none

#VersionAddendum none

# no default banner path

#Banner none

# Allow client to pass locale environment variables

AcceptEnv LANG LC\_\*

# override default of no subsystems

Subsystem sftp /usr/lib/openssh/sftp-server

# Example of overriding settings on a per-user basis

#Match User anoncvs

# X11Forwarding no

# AllowTcpForwarding no

# PermitTTY no

# ForceCommand cvs server

1. 启动SSH服务

# systemctl start ssh.service //启动服务

# systemctl status ssh.service //查看SSH服务的状态

● ssh.service - OpenBSD Secure Shell server

Loaded: loaded (/lib/systemd/system/ssh.service; disabled; vendor preset: disabled)

Active: **active (running)** since Tue 2018-09-25 10:17:52 CST; 1min 18s ago

Docs: man:sshd(8)

man:sshd\_config(5)

Process: 3363 ExecStartPre=/usr/sbin/sshd -t (code=exited, status=0/SUCCESS)

Main PID: 3364 (sshd)

Tasks: 1 (limit: 2322)

Memory: 3.6M

CGroup: /system.slice/ssh.service

└─3364 /usr/sbin/sshd -D

9月 25 10:17:52 kali2018-3a systemd[1]: Starting OpenBSD Secure Shell server.

9月 25 10:17:52 kali2018-3a sshd[3364]: Server listening on 0.0.0.0 port 22.

9月 25 10:17:52 kali2018-3a sshd[3364]: Server listening on :: port 22.

9月 25 10:17:52 kali2018-3a systemd[1]: Started OpenBSD Secure Shell server.

1. 添加SSH自启动

# systemctl enable ssh.service //添加ssh开启自动启

Synchronizing state of ssh.service with SysV service script with /lib/systemd/systemd-sysv-install.

Executing: /lib/systemd/systemd-sysv-install enable ssh

Created symlink /etc/systemd/system/sshd.service → /lib/systemd/system/ssh.service.

1. 使用SSH登录工具登录kali

C:\Users\VDragon>ssh root@192.168.44.10

root@192.168.44.10's password:

Linux kali2018-2 4.18.0-kali1-amd64 #1 SMP Debian 4.18.6-1kali1 (2018-09-10) x86\_64

The programs included with the Kali GNU/Linux system are free software;

the exact distribution terms for each program are described in the

individual files in /usr/share/doc/\*/copyright.

Kali GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent

permitted by applicable law.

Last login: Fri Sep 28 08:55:17 2018 from 192.168.44.1

root@kali2018-2:~#

输入用户名、密码后，如果使用SSH连接工具还是连不上kali 2.0，那么要先生成两个密钥：

# ssh-keygen -t dsa -f /etc/ssh/ssh\_host\_dsa\_key

# ssh-keygen -t dsa -f /etc/ssh/ssh\_host\_rsa\_key

执行命令后都会让输入密码，再使用SSH连接工具重新连接kali，成功搞定！

# **系统升级**

网络配置好后，第一件事情就系统升级，系统升级实际上是一个软件包的删除与重装过程，在更新的过程中，使用apt-get命令，apt-get命令会自动删除旧的软件包，以此在系统升级时务必更新软件包的索引文件。

更新步骤分为6步：

* 生新软件包索引文件
* 执行软件包升级
* 解决软件包依赖性升级
* 系统清理
* 自动卸载升级后过时的无用的旧软件包
* 重启系统

在系统升级过程中，原有的系统内核软件会保留在系统中，以便系统升级过程中在使用新的系统内核引导系统出现异常时，使得我们能使用旧的内核继续引导系统。

## 更新软件包索引文件

1. kali系统的更新源

# sudo apt-get install update //生成要更新的文件列表，并不升级文件

注意：执行完上面的命令，就可以进入下一个步骤了，但这里有一些扩展知识，小伙伴们可以了解一下：）

1. 查看kali系统的更新源地址文件

# vim /etc/apt/sources.list

[](https://www.fujieace.com/wp-content/uploads/2017/09/100-1.png?x86494)

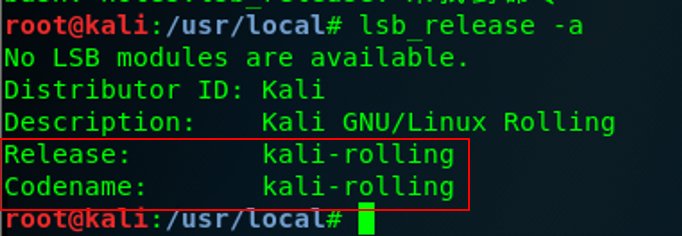
上面这是kali官方的更新源,一个完整的源包括：deb和deb-src；

* 上图源地址是：http://http.kali.org/kali；
* 图中的kali-rolling是kali目前最新的代号。

kali有两个代号(codename)：sana和kali-rolling。打开http://http.kali.org/kali。dists文件夹就是所包含的源。dists文件夹下面的kali-rolling文件夹就是所对应的代号。main non-free contrib ：代表主要软件/收费软件/免费软件什么的......

1. 查看自己的kali linux代号:

#lsb\_release –a



1. 查看操作系统版本

# cat /etc/issue

Kali GNU/Linux Rolling \n \l

# uname -a

Linux kali 4.17.0-kali1-amd64 #1 SMP Debian 4.17.8-1kali1 (2018-07-24) x86\_64 GNU/Linux

# root@kali:~# uname -m

x86\_64

1. 增加或替换掉sources.list文件里面的更新源地址（可选，不建议）

除非官方更新源有问题才执行这个步骤！！！

记录更新源的文件：/etc/apt/sources.list。

# vim /etc/apt/sources.list

默认sources.list中的更新源为官方的，如果下载速度慢，就需要寻找符合自己需要的更新源。百度一下“linux更新源”，选择在自己所在的网中比较快的更新源（教育网、网通、电信等等）。替换sources.list里面的内容。以下是我收集的kali更新源大全，有一些或许是失效的，有一些需要做一些细节上的修改就可以用了（使用一个即可）。

# kali官方源

deb http://http.kali.org/ moto main non-free contrib

deb-src http://http.kali.org/ moto main non-free contrib

deb http://security.kali.org/ moto/updates main contrib non-free

deb-src http://security.kali.org/ moto/updates main contrib non-free

#中科大kali源

deb http://mirrors.ustc.edu.cn/kali kali main non-free contrib

deb-src http://mirrors.ustc.edu.cn/kali kali main non-free contrib

deb http://mirrors.ustc.edu.cn/kali-security kali/updates main contrib non-free

#新加坡kali源

deb http://mirror.nus.edu.sg/kali/kali/ kali main non-free contrib

deb-src http://mirror.nus.edu.sg/kali/kali/ kali main non-free contrib

deb http://security.kali.org/kali-security kali/updates main contrib non-free

deb http://mirror.nus.edu.sg/kali/kali-security kali/updates main contrib non-free

deb-src http://mirror.nus.edu.sg/kali/kali-security kali/updates main contrib non-free

#阿里云kali源

deb http://mirrors.aliyun.com/kali kali main non-free contrib

deb-src http://mirrors.aliyun.com/kali kali main non-free contrib

deb http://mirrors.aliyun.com/kali-security kali/updates main contrib non-free

#163 Kali源

deb http://mirrors.163.com/debian wheezy main non-free contrib

deb-src http://mirrors.163.com/debian wheezy main non-free contrib

deb http://mirrors.163.com/debian wheezy-proposed-updates main non-free contrib

deb-src http://mirrors.163.com/debian wheezy-proposed-updates main non-free contrib

deb-src http://mirrors.163.com/debian-security wheezy/updates main non-free contrib

deb http://mirrors.163.com/debian-security wheezy/updates main non-free contrib

#auto

deb http://http.kali.org/kali kali-rolling main non-free contrib

#中科大

deb http://mirrors.ustc.edu.cn/kali kali-rolling main non-free contrib

deb-src http://mirrors.ustc.edu.cn/kali kali-rolling main non-free contrib

#浙大

deb http://mirrors.zju.edu.cn/kali kali-rolling main contrib non-free

deb-src http://mirrors.zju.edu.cn/kali kali-rolling main contrib non-free

#东软大学

deb http://mirrors.neusoft.edu.cn/kali kali-rolling/main non-free contrib

deb-src http://mirrors.neusoft.edu.cn/kali kali-rolling/main non-free contrib

#重庆大学

deb http://http.kali.org/kali kali-rolling main non-free contrib

deb-src http://http.kali.org/kali kali-rolling main non-free contrib

#官方源

#deb http://http.kali.org/kali kali-rolling main non-free contrib

#deb-src http://http.kali.org/kali kali-rolling main non-free contrib

以下是我目前用的kali linux两个更新源地址：

# kali官方源

deb http://http.kali.org/ kali-rolling main non-free contrib

deb-src http://http.kali.org/ kali-rolling main non-free contrib

#中科大源

deb http://mirrors.ustc.edu.cn/kali kali-rolling main non-free contrib

deb-src http://mirrors.ustc.edu.cn/kali kali-rolling main non-free contrib

我是直接把上面的更新源地址直接加入到sources.list，官方源我也是没有做任何的修改。毕竟源地址多了并没有什么坏处，唯一是在我们更新的时候下载时间或许会久一点，仅此而已。需要用update更新完才能upgrade，此时会从更新源下载并安装

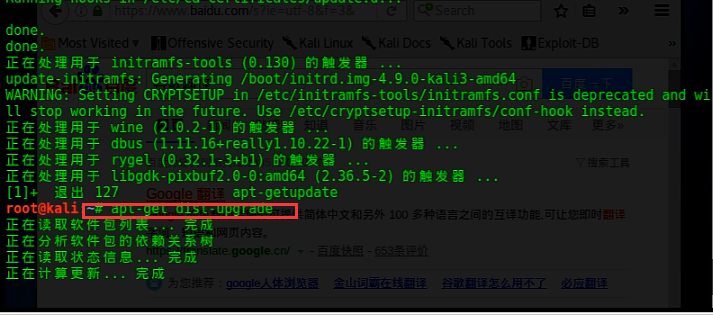
## 执行软件包升级

# apt-get upgrade //使系统升级，这才是真正的软件更新命令

注意：时间很长，有时30分钟，有时整整一天，需要根据系统的情况而定。

## 解决软件包依赖性升级

# apt-get dist-upgrade //解决依赖性问题的升级

[](https://www.fujieace.com/wp-content/uploads/2017/09/1010.png?x86494)

注意：

upgrade和dist-upgradej是有区别的：

upgrade:系统将现有的Package升级,如果有依赖性的问题,而此依赖性需要安装其它新的Package或影响到其它Package的依赖性时,此Package就不会被升级,会保留下来.

dist-upgrade:可以聪明的解决相依性的问题,如果有相依性问题,需要安装/移除新的Package,就会试着去安装/移除它，但有些Package依赖于固定版本的包，如果自动升级后，可能导致这些Package出现问题。因此，dist-upgrade升级具备一定的风险性。

总结：apt-get upgrade 和 apt-get dist-upgrade 本质上是没有什么不同的。只不过：dist-upgrade会识别出当依赖关系改变的情形并作出处理，而upgrade对此情形不处理。

例如软件包 a 原先依赖 b c d，但是在源里面可能已经升级了，现在是 a 依赖 b c e。这种情况下，dist-upgrade 会删除 d 安装 e，并把 a 软件包升级，而 upgrade 会认为依赖关系改变而拒绝升级 a 软件包。

## 系统清理

# apt-get clean

## 自动卸载无用的旧版本软件包

# apt autoremove //自动清除无用的软件包

## 重启系统

# reboot

# 常用软件安装和系统简单配置

## **安装输入法**

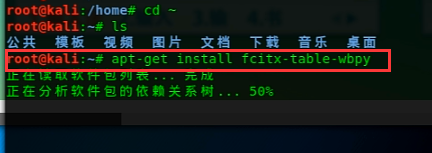
安装拼音、双拼、五笔拼音输入法：fcitx-table-wbpy

1. 打开终端输入命令：

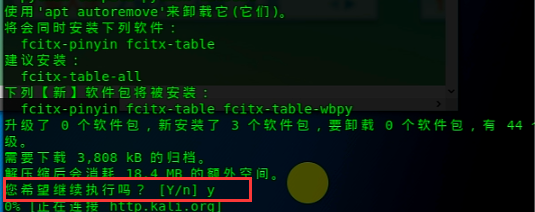
# apt-get install fcitx-table-wbpy

如果上面不好用，请用下面这个命令：

# apt-get install fcitx fcitx-table-wbpy

[](https://www.fujieace.com/wp-content/uploads/2017/09/22-1.png?x86494)

你希望继续执行吗？Y

[](https://www.fujieace.com/wp-content/uploads/2017/09/33.png?x86494)

1. 安装结束后，输入重启命令：

# reboot

重启进入kali系统后，打开浏览器可以用你想要的输入法输出中文；

* 使用Ctrl+空格，可以调出输入法。
* 使用Ctrl+Shift，可以切换输入法。

此输入法共有三种输入法，分别是：拼音、双拼、五笔拼音。

# Linux启动流程

## 第一步： BIOS自检

这步是找到BIOS检查CPU的相关信息、设备启动顺序信息、硬盘信息、内存信息、时钟信息、PnP特性等等，（启动顺很熟悉，每次gost装机的时候，要把开机首选项设置成从U盘启动）。

## 第二步： 读取MBR（Master boot Record）

MBR是位于一扇区，零磁道，0柱面的一个有512B的文件，里面有446B是引导区（PRE-BOOT） ，用来找到活动分区（active）并将活动区读入0×7c00内存，其实被复制到内存的就是我们说的BOOT Loader,准确到电脑上就是GRUB。还有66B是分区表（PARTITION PABLE）这里记录了硬盘信息。(分了几个区之类的）

## 第三步： 加载Boot Loader

Boot Loader的种类有很多，其实是一段小程序，主要任务是初始化硬件设备、建立内存空间的映射图等，从而将系统的软硬件环境带到一个合适的状态，为调用操作系统内核做好一切准备。

### Grub磁盘识别

Grub、Lilo和spfdisk是常见的Loader。Kali linux使用的是Grub。

GRUB启动时，操作系统并没有启动，这时没有根文件系统的概念。GRUB只能识别物理分区。那GRUB是如何识别分区和文件的呢？

存储设备和分区

任何类型的硬盘都是HD,如HD0,HD1。。。。

同一硬盘的不同分区识别为HD0,0 HD0,1。。。

光盘识另为HD-ROM。

GRUB2对设备与分区的命名规则举例如下，看看就能明白。需要说明的是磁盘从"0"开始计数，分区从"1"开始计数。

(fd0) 第一软盘

(hd0) 第一硬盘[大多数U盘与USB接口的移动硬盘以及SD卡也都被当作硬盘看待]

(hd1,1)第二硬盘的第一分区(通用于MBR与GPT分区)

(hd0,msdos2) 第一硬盘的第二MBR分区，也就是传统的DOS分区表

(hd1,msdos5) 第二硬盘的第五MBR分区，也就是第一个逻辑分区

(hd0,gpt1) 第一硬盘的第一GPT分区

(cd) 启动光盘[仅在从光盘启动GRUB时可用]

(cd0) 第一光盘

上面所举的例子仅是最常用的情形，更多高级的设备命名规则请参考GRUB2手册中的"[Naming convention](http://www.gnu.org/software/grub/manual/html_node/Naming-convention.html)"与"[How to specify devices](http://www.gnu.org/software/grub/manual/html_node/Device-syntax.html)"部分。此外，如果你想看看当前系统上有哪些设备可用，可以在GRUB SHELL中使用"ls"命令(可能需要先加载必要的驱动模块)。

### Grub磁盘读取

#### 磁盘块（扇区）访问

磁盘块的命名方法两种：(1)绝对路径表示法，(2)相对路径表示法。举例如下：

(hd1,1)0+1 在第二硬盘的第一分区上，从第"0"个磁盘块(首扇区)起，长度为"1"的连续块。[绝对路径]

(hd1,1)+1 含义与上一个相同，因为当从第"0"个磁盘块(首扇区)起时，"0"可以省略不写。[绝对路径]

+1 在根设备上，从第"0"个磁盘块(首扇区)起，长度为"1"的连续块。[相对路径]

当"root"环境变量等于"(hd1,1)"时，等价于"(hd1,1)0+1"

磁盘块几乎只用于链式引导(chainloader)的场合。更多高级的磁盘块命名规则请参考GRUB2手册中的"[How to specify block lists](http://www.gnu.org/software/grub/manual/html_node/Block-list-syntax.html)"部分。

#### 文件访问

文件的命名方法同样也有两种：(1)绝对路径表示法，(2)相对路径表示法。举例如下：

(fd0)/grldr：第一软盘根目录下的"grldr"文件[绝对路径]

(hd0,gpt1)/boot/vmlinuz：第一硬盘的第一GPT分区"boot"目录下的"vmlinuz"文件[绝对路径]。

/boot/vmlinuz：根设备"boot"目录下的"vmlinuz"文件[相对路径]，当"root"环境变量等于"(hd0,gpt1)"时，等价于"(hd0,gpt1)/boot/vmlinuz"

上面所举的例子仅是最常用的情形，更多高级的文件命名规则请参考GRUB2手册中的"[How to specify files](http://www.gnu.org/software/grub/manual/html_node/File-name-syntax.html)"部分。

### GRUB2启动流程及救援模式

|  |  |
| --- | --- |
| GRUB2在BIOS平台上的常规启动步骤是这样的： | GRUB2在UEFI平台上的常规启动步骤是这样的： |
| * BIOS * boot.img[MBR] * core.img[MBR gap/embedding area/BIOS Boot Partition] * 设置"prefix root cmdpath"环境变量 * 加载"normal.mod"模块[同时还包括它所依赖的 terminal crypto extcmd boot gettext 模块] * 执行"normal $prefix/grub.cfg"命令 | * UEFI * core.img[BOOTX64.EFI/BOOTX86.EFI] * 设置"prefix root cmdpath"环境变量 * 加载"normal.mod"模块[同时还包括它所依赖的 terminal crypto extcmd boot gettext 模块] * 执行"normal $prefix/grub.cfg"命令 |

如果上述步骤全部成功，那么你将进入'普通模式'，一般是显示一个菜单(找到了'$prefix/grub.cfg')，或者直接进入GRUB SHELL(没找到'$prefix/grub.cfg')。在普通模式中，命令模块[command.lst]与加密模块[crypto.lst]会被自动按需载入(无需使用"insmod"命令)，并且可使用完整的GRUB脚本功能。但是其他模块则可能需要明确使用"insmod"命令来载入。

如果在加载"normal.mod"模块这一步出现故障，那么你将进入GRUB2的'救援模式'，而不是常规的'普通模式'。在救援模式中，GRUB只自动设置了"cmdpath prefix root"三个环境变量，并且只能使用"insmod ls set unset"四个命令。只有当额外的模块被加载之后，才可以使用一些其它的命令,变量,解析器,驱动程序。通常来说，进入救援模式可能意味着你的GRUB2没有正确安装。请认真阅读'grub-install --help'的输出选项，并使用正确的选项重新安装。更多细节请参考GRUB2手册中的"[GRUB only offers a rescue shell](http://www.gnu.org/software/grub/manual/html_node/GRUB-only-offers-a-rescue-shell.html)"部分。

#### GRUB2环境变量

GRUB2的环境变量大致可以分为两类，第一类是自动设置的变量，也就是这些变量的初始值由GRUB2自动设置，其值必定存在且不为空。第二类是手动设置的变量，它们没有初始值(或者初始值为空)，需要经过手动明确设置之后才能使用。

大多数有特定含义的环境变量都是附属于特定附加模块的，只有加载了这些模块之后，这些环境变量才变得有意义。所以从模块的角度看，GRUB2的环境变量又可以分为三类：(1)核心变量，GRUB2核心提供的变量，不依赖于任何可加载模块，这样的变量只有"cmdpath prefix root"三个，而且它们的初始值都由GRUB2自动设置。(2)模块变量，绝大多数有特定含义的环境变量都属此类。(3)脚本变量，这是为了方便编写grub.cfg脚本而设置的变量，没有特殊含义，也不依赖于特定模块，与一般的bash脚本中的变量类似。有关GRUB2脚本的完整说明可以参考GRUB2手册中的"[Writing full configuration files directly](http://www.gnu.org/software/grub/manual/html_node/Shell_002dlike-scripting.html)"部分，基本上其语法与bash脚本完全一致，上手非常容易。

下面列出的变量都是有特定含义的变量，这里只列出常用的一些变量，完整的列表可以参考GRUB2手册中的"[Special environment variables](http://www.gnu.org/software/grub/manual/html_node/Special-environment-variables.html)"部分。

?

上一条命令的返回值，零表示成功，非零表示失败[与bash一样]。由GRUB2自动设置。你只能使用此变量，而不能修改它。

check\_signatures

是否在加载文件时强制验证签名，可以设为'enforce'或'no'

chosen

当前被执行的菜单项名称(紧跟"menuentry"命令之后的字符串或者'--id'选项的参数)，例如'Windows 7'。由GRUB2自动设置。你只应该使用此变量，而不应该修改它。

cmdpath

当前被加载的"core.img"所在目录(绝对路径)。例如：UEFI启动可能是'(hd0,gpt1)/EFI/UBUNTU'或'(cd0)/EFI/BOOT'，BIOS启动可能是'(hd0)'。由GRUB2自动设置。你只应该使用此变量，而不应该修改它。

debug

设为'all'时表示开启调试输出[会显示大量信息,谨慎开启]

default

默认选中第几个菜单项(从'0'开始计数)

fallback

如果默认菜单项启动失败，那么就启动第几个菜单项(从'0'开始计数)

gfxmode

设置"gfxterm"模块所使用的视频模式，可以指定一组由逗号或分号分隔的模式以供逐一尝试：每个模式的格式必须是：'auto'(自动检测),'宽x高','宽x高x色深'之一，并且只能使用VBE标准指定的模式[640x480,800x600,1024x768,1280x1024]x[16,24,32]。可以在GRUB SHELL中使用"videoinfo"命令列出当前所有可用模式。默认值是'auto'。

gfxpayload

设置Linux内核启动时的视频模式，可以指定一组由逗号或分号分隔的模式以供逐一尝试：每个模式的格式必须是：'text'(普通文本模式,不能用于UEFI平台),'keep'(继承"gfxmode"的值),'auto'(自动检测),'宽x高','宽x高x色深'之一，并且只能使用VBE标准指定的模式[640x480,800x600,1024x768,1280x1024]x[16,24,32]。在BIOS平台上的默认值是'text'，在UEFI平台上的默认值是'auto'。除非你想明确设置Linux控制台的分辨率(要求内核必须"CONFIG\_FRAMEBUFFER\_CONSOLE=y")，或者打算在BIOS平台上使用图形控制台(要求内核必须"CONFIG\_FRAMEBUFFER\_CONSOLE=y")，否则不要设置此变量。

gfxterm\_font

设置"gfxterm"模块所使用的字体，默认使用所有可用字体

grub\_cpu

此GRUB所适用的CPU类型。例如：'i386', 'x86\_64'。由GRUB2自动设置。你只应该使用此变量，而不应该修改它。

grub\_platform

此GRUB所适用的平台类型。例如：'pc', 'efi'。由GRUB2自动设置。你只应该使用此变量，而不应该修改它。

lang

设置GRUB2的界面语言，必须搭配"locale\_dir"变量一起使用。简体中文应设为'zh\_CN'。

locale\_dir

设置翻译文件(\*.mo)的目录，通常是'$prefix/locale'，若未明确设置此目录，则禁止国际化。

pager

如果设为'1'，那么每一满屏后暂停输出，等待键盘输入。缺省是''，表示不暂停。

prefix

绝对路径形式的'/boot/grub'目录位置(也就是GRUB2的安装目录)，例如'(hd0,gpt1)/grub'或'(hd0,msdos2)/boot/grub'。初始值由GRUB在启动时根据"grub-install"在安装时提供的信息自动设置。你只应该使用此变量，而不应该修改它。

root

设置"根设备"。任何未指定设备名的文件都视为位于此设备。初始值由GRUB在启动时根据"prefix"变量的值自动设置。在大多数情况下，你不需要修改它。

superusers

设置一组"超级用户"(使用空格/逗号/分号进行分隔)，以开启安全认证的功能。

theme

设置菜单界面的主题风格文件的位置，例如："/boot/grub/themes/starfield/theme.txt"。关于如何定制界面风格(背景图片/字体/颜色/图标等)的细节，可以参考GRUB2手册中的"[Theme file format](http://www.gnu.org/software/grub/manual/html_node/Theme-file-format.html)"部分。

timeout

在启动默认菜单项前，等待键盘输入的秒数。默认值是'5'秒。'0'表示直接启动默认菜单项(不显示菜单)，'-1'表示永远等待。

#### GRUB2命令

对于GRUB-2.0.2版本来说，所有可用的命令有大约200个之多，他们中的绝大多数由各种各样的模块提供。我们没有必要去了解所有这些200个命令，只需要了解一些常用的命令即可(实际上就连官方文档也没有给出全部的命令说明)。更多的命令说明可以参考GRUB2手册中的"[The list of available commands](http://www.gnu.org/software/grub/manual/html_node/Commands.html)"页面中列出的几个二级页面。

menuentry "title" [--class=class …] [--users=users] [--unrestricted] [--hotkey=key] [--id=id] [arg …] { command; … }

定义一个名为"title"的菜单项。当此菜单项被选中时，GRUB将会把环境变量"chosen"的值设为"id"(使用了[--id=id]选项)或"title"(未使用[--id=id]选项)，然后执行花括号中的命令列表，如果列表中最后一个命令执行成功，并且已经载入了一个内核，那么将执行"boot"命令。

可以使用 --class 选项指定菜单项所属的"样式类"。从而可以使用指定的主题样式显示菜单项。

可以使用 --users 选项指定只允许特定的用户访问此菜单项。如果没有使用此选项，则表示允许所有用户访问。

可以使用 --unrestricted 选项指明允许所有用户访问此菜单项。

可以使用 --hotkey 选项设置访问此菜单项的热键(快捷键)。"key"可以是一个单独的字母，或者'backspace','tab','delete'之一。

可以使用 --id 选项为此菜单项设置一个全局唯一的标识符。"id"必须由ASCII字母/数字/下划线组成，且不得以数字开头。

[arg …]是可选的参数列表。你可以把它们理解为命令行参数。实际上"title"也是命令行参数，只不过这个参数是个必须参数而已。这些参数都可以在花括号内的命令列表中使用，"title"对应着"$1"，其余的以此类推。

terminal\_input [--append|--remove] [terminal1] [terminal2] …

如果不带任何选项与参数，则表示列出当前激活的输入终端，以及所有其他可用的输入终端。

可以使用 --append 选项将指定的终端加入到激活的输入终端列表中，所有列表中的终端都可以用于向GRUB提供输入。

可以使用 --remove 选项将指定的终端从激活的输入终端列表中删除。

如果不使用任何选项，但是指定了一个或多个终端参数，则表示将当前激活的输入终端设置为参数指定的终端。

terminal\_output [--append|--remove] [terminal1] [terminal2] …

如果不带任何选项与参数，则表示列出当前激活的输出终端，以及所有其他可用的输出终端。

可以使用 --append 选项将指定的终端加入到激活的输出终端列表中，所有列表中的终端都将接受到GRUB的输出。

可以使用 --remove 选项将指定的终端从激活的输出终端列表中删除。

如果不使用任何选项，但是指定了一个或多个终端参数，则表示将当前激活的输出终端设置为参数指定的终端。

authenticate [userlist]

检查当前用户是否位于"userlist"或环境变量"superusers"中。[注意]如果环境变量"superusers"的值为空，此命令将返回'真'。

background\_color color

设置当前激活的输出终端的背景颜色。"color"可以使用HTML风格的颜色表示法("#RRGGBB"或"#RGB")。

[注意]仅在使用'gfxterm'作为输出终端的时候，才能改变背景色。

background\_image [[--mode 'stretch'|'normal'] file]

将当前激活的输出终端的背景图片设置为"file"文件。除非使用了"--mode 'normal'"选项，否则图片将被自动缩放以填满整个屏幕。

如果不带任何选项与参数，则表示删除背景图片。

[注意]仅在使用'gfxterm'作为输出终端的时候，才能改变背景图片。

boot

启动已经被载入的OS或链式加载器。仅在运行于交互式命令行的时候才是需要的。在一个菜单项结束时是隐含的。

cat [--dos] file

显示文件"file"的内容。如果使用了"--dos"选项，那么"回车/换行符"将被显示为一个简单的换行符。否则，回车符将被显示为一个控制符(<d>)。

chainloader [--force] file

链式加载"file"文件。通常使用磁盘块表示法，例如用'+1'表示当前根分区的第一个扇区。

可以使用 --force 选项强制载入文件，而不管它是否有正确的签名。通常用于加载有缺陷的启动载入器(例如 SCO UnixWare 7.1)。

configfile file

将"file"作为配置文件加载。如果"file"中定义了菜单项，那么立即显示一个包含它们的菜单。

[注意]"file"文件对环境变量所做的任何变更都将在从此文件返回后失效。

cpuid [-l]

检查CPU特性。仅在x86系统上可用。

如果使用了 -l 选项，那么如果CPU是64位则返回真，否则返回假。

drivemap -l|-r|[-s] from\_drive to\_drive

如果不使用任何选项，表示将"from\_drive"映射到"to\_drive"。这主要用于链式加载Windows之类的操作系统，因为它们只能从第一个硬盘启动。出于方便的原因，分区后缀将被忽略，因此你可用安全地将"${root}"作为磁盘使用。

可以使用 -s 选项，执行反向映射，也就是交换这两个磁盘。例如： drivemap -s (hd0) (hd1)

可以使用 -l 选项，列出当前已有的映射。

可以使用 -r 选项，把映射重置为默认值，也就是撤销所有当前已有的映射。

echo [-n] [-e] string …

显示所要求的文本并换行(除非使用了 -n 选项)。如果有多个字符串，依次输出它们，并用空格分隔每一个。

和bash的习惯一样，可以在双引号内使用"${var}"来引用变量的值，也可以使用 -e 选项激活对反斜杠转义符的解释( \\ \a \r \n \t ...)。

export envvar

导出环境变量"envvar"，以使其对于使用"configfile"命令载入的配置文件可见。

false

不做任何事，只返回一个失败的结果。主要用在if/while之类的控制构造中。

gettext string

把"string"翻译为环境变量"lang"指定的语言。MO格式的翻译文件从环境变量"locale\_dir"指定的目录加载。

halt [--no-apm]

关闭计算机。如果指定了 --no-apm 选项，表示不执行APM BIOS调用。否则，计算机使用APM关闭。

help [pattern …]

显示内建命令的帮助信息。如果没有指定"pattern"，那么将显示所有可用命令的简短描述。

如果指定了"pattern"，那么将只显示名字以这些"pattern"开头的命令的详细帮助信息。

initrd file

为以32位协议启动的Linux内核载入一个"initial ramdisk"，并在内存里的Linux设置区域设置合适的参数。

[注意]这个命令必须放在"linux"命令之后使用。

initrd16 file

为以16位协议启动的Linux内核载入一个"initial ramdisk"，并在内存里的Linux设置区域设置合适的参数。

[注意]这个命令必须放在"linux16"命令之后使用。

insmod module

载入名为"module"的GRUB2模块。

linux file …

使用32位启动协议从"file"载入一个Linux内核映像，并将其余的字符作为内核的命令行参数逐字传入。

[注意]使用32位启动协议意味着'vga='启动选项将会失效。如果你希望明确设置一个特定的视频模式，那么应该使用"gfxpayload"环境变量。虽然GRUB可以自动地检测某些'vga='参数，并把它们翻译为合适的"gfxpayload"设置，但是并不建议这样做。

linux16 file …

以传统的16位启动协议从"file"载入一个Linux内核映像，并将其余的字符作为内核的命令行参数逐字传入。这通常用于启动一些遵守Linux启动协议的特殊工具(例如[MEMDISK](http://www.syslinux.org/wiki/index.php/MEMDISK))。

[注意]使用传统的16位启动协议意味着：(1)'vga='启动选项依然有效，(2)不能启动纯64位内核(也就是内核必须要'CONFIG\_IA32\_EMULATION=y'才行)。

loadfont file …

从指定的"file"加载字体，除非使用了绝对路径，否则"file"将被视为"$prefix/fonts/file.pf2"文件。

loopback [-d] device file

将"file"文件映射为"device"回环设备。例如：

loopback loop0 /path/to/image

ls (loop0)/

可以使用 -d 选项，删除先前使用这个命令创建的设备。

ls [arg …]

如果不使用参数，那么列出所有对GRUB已知的设备。

如果参数是包含在括号内的一个设备名，那么列出该设备根目录下的所有文件。

如果参数是以绝对路径给出的目录，那么列出这个目录的内容。

lsfonts

列出已经加载的所有字体

lsmod

列出已经加载的所有模块

normal [file]

进入普通模式，并显示GRUB菜单。[说明]只要当前没有处于救援模式，其实就已经是在普通模式中了，所以通常并不需要明确使用此命令。

在普通模式中，命令模块[command.lst]与加密模块[crypto.lst]会被自动按需载入(无需使用"insmod"命令)，并且可使用完整的GRUB脚本功能。但是其他模块则可能需要明确使用"insmod"命令来载入。

如果给出了"file"参数，那么将从这个文件中读入命令(也就是作为"grub.cfg"的替代)，否则将从"$prefix/grub.cfg"中读入命令(如果存在的话)。你也可以理解为"file"的默认值是'$prefix/grub.cfg'。

可以在普通模式中嵌套调用此命令，以构建一个嵌套的环境。不过一般不这么做，而是使用"configfile"命令来达到这目的。

normal\_exit

退出当前的普通模式。如果这个普通模式实例不是嵌套在另一个普通模式里的话，就会返回到救援模式。

parttool partition commands

对分区表进行各种修改。目前只能作用于MBR分区表(DOS分区表)，而不能用于GPT分区表。目前仅支持以下三种用法：

(1)设置或去掉分区的激活标记(仅对Windows系统有意义)。

例如："parttool (hd0,msdos2) +boot"表示为(hd0,msdos2)分区加上激活标记，而"parttool (hd0,msdos2) -boot"则表示去掉(hd0,msdos2)分区的激活标记。

(2)设置或去掉分区的隐藏标记(仅对Windows系统有意义)。

例如："parttool (hd0,msdos2) +hidden"表示为(hd0,msdos2)分区加上隐藏标记，而"parttool (hd0,msdos2) -hidden"则表示去掉(hd0,msdos2)分区的隐藏标记。

(3)更改分区的类型。其值必须是0x00-0xFF范围内的值。且应该使用'0xNN'格式的十六进制数。

例如："parttool (hd0,msdos2) type=0x83"表示将(hd0,msdos2)分区类型修改为'0x83'(Linux分区)。

password user clear-password

定义一个名为user的用户，并使用明文口令'clear-password'。不建议使用此命令。

password\_pbkdf2 user hashed-password

定义一个名为user的用户，并使用哈希口令'hashed-password'(通过"grub-mkpasswd-pbkdf2"工具生成)。这是建议使用的命令，因为它安全性更高。

probe [--set var] --driver|--partmap|--fs|--fs-uuid|--label device

提取"device"设备的特定信息。如果使用了 --set 选项，则表示将提取的结果保存在"var"变量中，否则将提取的结果直接显示出来。

read [var]

从用户读取一行输入。如果给定环境变量"var"，则把它设为所读取的行(不包括结尾的换行符)。

reboot

重新启动

rmmod module

卸载"module"模块

search [--file|--label|--fs-uuid] [--set [var]] [--no-floppy] name

通过文件[--file]、卷标[--label]、文件系统UUID[--fs-uuid]来查找设备。

如果使用了 --set 选项，那么会将第一个找到的设备设置为环境变量"var"的值。默认的"var"是'root'。

可以使用 --no-floppy 选项来禁止查找软盘设备，因为这些设备非常慢。

set [envvar=value]

将环境变量"envvar"的值设为'value'。如果没有使用参数，则打印出所有环境变量及其值。

source file

直接将"file"文件的内容插入到当前位置。与"configfile"不同，此命令既不切换执行环境，也不会显示一个新的菜单。

* test expression [ expression ]
* 计算"expression"的值，并在结果为真时返回零值，或者在结果为假时返回非零值，主要用在if/while之类的控制构造中。
* 可用的"expression"模式如下(与bash类似)：
* string1 == string2 [string1与string2完全相同]
* string1 != string2 [string1与string2不完全相同]
* string1 < string2 [string1在字母顺序上小于string2]
* string1 <= string2 [string1在字母顺序上小于string2或与string2完全相同]
* string1 > string2 [string1在字母顺序上大于string2]
* string1 >= string2 [string1在字母顺序上大于string2或与string2完全相同]
* integer1 -eq integer2 [integer1等于integer2]
* integer1 -ge integer2 [integer1大于或等于integer2]
* integer1 -gt integer2 [integer1大于integer2]
* integer1 -le integer2 [integer1小于或等于integer2]
* integer1 -lt integer2 [integer1小于integer2]
* integer1 -ne integer2 [integer1不等于integer2]
* prefixinteger1 -pgt prefixinteger2 [剔除非数字字符首部之后，integer1大于integer2]
* prefixinteger1 -plt prefixinteger2 [剔除非数字字符首部之后，integer1小于integer2]
* file1 -nt file2 [file1的修改时间比file2新]
* file1 -ot file2 [file1的修改时间比file2旧]
* -d file [file存在并且是一个目录]
* -e file [file存在]
* -f file [file存在并且不是一个目录]
* -s file [file存在并且文件尺寸大于零]
* -n string [string的长度大于零]
* string [string的长度大于零]
* -z string [string的长度等于零]
* ( expression ) 将expression视为一个整体(分组)
* ! expression 非(NOT)
* expression1 -a expression2 与(AND)
* expression1 -o expression2 或(OR)

true

不做任何事，只返回一个成功的结果。主要用在if/while之类的控制构造中。

unset envvar

撤销环境变量"envvar"

videoinfo [[WxH]xD]

列出所有当前可用的视频模式。如果指定了分辨率(或者还附加了色深)，那么仅显示与其匹配的模式。

#### GRUB2安全

在默认情况下，任何人都可对GRUB访问、选择并编辑任意菜单项，并且可以直接访问GRUB SHELL。为了保证GRUB的安全性，GRUB提供了安全认证机制，要启用认证支持，必须将环境变量"superusers"设置为一组用户名(可用空格/逗号/分号作为分隔符)，这样，将仅允许"superusers"中的用户使用GRUB命令行、编辑菜单项、以及执行任意菜单项。而其他非"superusers"中的用户，只能执行那些没有设置 --users 选项的菜单，以及那些在 --users 选项中包含了该用户的菜单，但不能使用GRUB命令行、编辑菜单项。下面使用一个配置片段举例说明：

set superusers="root"

password\_pbkdf2 root grub.pbkdf2.sha512.10000.biglongstring

password user1 insecure

menuentry "所有人都可以执行此菜单" --unrestricted {

...

}

menuentry "仅允许超级用户执行此菜单" --users "" {

...

}

menuentry "允许 user1 和超级用户执行此菜单" --users user1 {

...

}

有关GRUB2安全的更多详情，请参考GRUB2手册中的"[Security](http://www.gnu.org/software/grub/manual/html_node/Security.html)"部分。

### "grub.cfg"配置

注：本章内容为了便于理解linux启动流程而写的，理解即可，千万不要去死记硬背本节的grub的命令和配置参数，以后用到时，来本节查找妈可！

boot loader 是计算机启动时运行的第一个程序，他负责加载内核并将控制权交给[操作系统](http://lib.csdn.net/base/operatingsystem)的内核，然后内核开始初始化系统的剩余部分。grub是个功能强大的工具，可以加载各种各样的操作系统，grub一个重要的特性就是灵活，他可以理解文件系统和内核的执行方式，所以你可以使用你喜欢的方式加载操作系统，当你用grub引导时你可以用命令行接口模式或者菜单模式，你也可以在菜单模式下转入命令行模式。

大约2002年左右，Yoshinori K. Okuji在PUPA重写了GRUB，使他更清晰，安全，健壮，更强大，PUPA最后把他重命名为GRUB2，把原来版本的grub命名为GRUB Legacy，2005年发行了grub legacy(0.97)，2007 GNU/Linux 开始在小范围内使用GRUB2，到2009年底大多数主要的发行版都开始默认安装GRUB2。虽然grub2是重写的grub，但是他借鉴了grub的一些特征，grub2改进如下：

* 引导配置文件采用新名字grub.cfg，新的语法，加入许多新的命令
* grub.cfg 是用grub-mkconfig自动产生的，由update-grub维护，在执行系统update-grub之前修改的配置都不会生效，这样可以很容易的应对内核升级这样情况。
* grub2中的分区号不再是以0开始，而是以1开始。
* 配置文件是不可写的，而是修改文件后用命令自动更新
* grub2有更可靠的方法在磁盘上有多系统时发现文件和目标内核，可以用命令发现系统设备号或者UUID。
* 更多的文件系统支持，如ext4 hfs ntfs
* grub2可以直接从lvm和raid读取文件
* 在grub中的stage1 stage1.5 stage2已经被取消
* grub2引入很多设备模块，允许core镜像更小
* 支持脚本，包括简单的语法
* 在启动时没有选择菜单的话，按住shift即可强制显示菜单

#### grub2的安装

grub2的组件为grub-pc，同时grub-common也会默认被安装，grub2是过渡期的虚拟组件，会被安全移除。Grub2的相挂钩脚本如下：

# grub-install

安装步骤：

1. 安装grub到设备中，如果需要会创建/boot/grub目录。
2. 他会把\*.mod、\*.lst、\*.img从 /usr/lib/grub/i386-pc/ 目录复制到/boot/grub目录下，他会覆盖已有文件。
3. 然后会调用grub\_probe扫描计算机并收集磁盘和分区信息。
4. 接着调用grub\_mkimage构建一个新的new.img。
5. 最后调用grub\_setup把grub的boot.img写入MBR中，把core.img写进设备的第一个扇区。

注意：grub\_install并不运行grub\_mkconfig

显示版本号

# grub-install –v

#### Grub2镜像

1. boot.img：

在PC BIOS系统中，boot.img是grub启动的第一部分，他被写入MBR（零磁道）或者boot分区，因为PC的boot扇区只有512bytes，所以此文件精确到只有512bytes，他唯一的功能是从本地磁盘读取内核镜像第一个扇区位置然后跳到内核镜像，因为文件大小的限制，boot.img不能识别任何文件系统，所以当安装grub时，grub-setup把内核镜像第一个扇区的位置硬编码进boot.img中。

1. diskboot.img

当从硬盘引导时此镜像作为内核镜像的第一个扇区的内容，他把剩余的内核镜像读入内存中然后启动内核。

cdboot.img 的原理同上

pxeboot.img：当从网络用PXE引导时用次镜像启动内核镜像

1. kernel.img

此镜像包含grub运行时包含的基本工具，框架驱动、文件句柄、环境变量、安全模式命令行解析器等，他可以直接使用，但是编译进所有的内核镜像中了。

1. \*.mod

其余的一切是可以动态加载的一些模块，当需要他们时，他们是被自动加载编译进内核镜像的，但也可以使用insmod手动加载。

1. core.img

这是grub的内核镜像，他被grub-mkimage程序从kernel.img和一些模块动态编译而成，一般情况下他已经包含足够的模块去访问/boot/grub，模块机制可以使得内核镜像保持很小。

#### grub1(grub legacy)的镜像文件

grub1的官方叫“grub legacy”，因为系统总从硬盘的“零磁道（MBR）”开始启动系统，但“零磁道（MBR）”只有512个字节，要想将整个linux系统内核放进去是不可能的，grub1就采用了“两段”启动的机制：

1. stage1

他非常像grub2中的boot.img，他们有一样的功能

1. \*\_stage1\_5

在grub legacy中，stage1\_5的作用是包含足够的系统代码从而可以使更大的stage2从普通系统上读入，他有点像grub2中的core.img但是core.img比他功能强多了，grub legacy可以只使用stage1和stage2而不载入内核，但这时只能使用grub本身、stage1和stage2提供的命令，linux其它的命令当然就没法使用了。但是grub2没有core.img就不能工作。

1. stage2

grub2没有stage2镜像，代替的是：从/boot/grub加载模块

#### grub2配置文件

grub2 包含一个完全修改过的配置文件及目录，menu.list不再使用

grub2采用模块化的设计，主要配置文件是/boot/grub/grub.cfg，但是这个文件的权限是444，连root也不让修改，这是刻意为之的，不要修改文件权限，grub.cfg的内容更新由update-grub命令来更新，其实就是执行grub-mkconfig -o /boot/grub/grub.cfg来更新，此文件会在更新内核自动更新，在/root/grub目录下有很多模块文件，以在按需时自动加载模块。

1. /etc/default/grub

此文件包含menu.list的前半部分内容，以及每行内核配置后面追加的配置，在更新时会合并到grub.cfg中

1. /etc/grub.d/目录

update-grub命令执行时会读取此目录下的文件，并将配置合并至grub.cfg中，grub.cfg中的菜单顺序是由此目录中的文件顺序决定的，文件开头数字较小的会先执行，全英文名字的文件将会在最后执行，自定义选项在40\_custom中定义，或者建立一个新文件，此目录下的文件必须有执行权限才会被update-grub读取，并把配置合并到grub.cfg中。

* 00\_header：配置初始的显示项目，如默认选项，时间限制等，一般由/etc/default/grub导入，一般不需要配置
* 05\_debian\_theme：配置引导画面，文字颜色等主题
* 10\_linux：定位当前操作系统使用中的root设备内核的位置
* 30\_os-prober：用来搜索[Linux](http://lib.csdn.net/base/linux)和其他系统，此脚本中的变量用来指定在/boot/grub/grub.cfg和grub2菜单中的名称显示方式，可以修改/etc/default/grub对比这个脚本
* 40\_custom：用来加入用户自定义的菜单样板，将会在执行update-grub时更新至grub.cfg中
* 41\_custom：判断custom.cfg此配置文件是否存在，如果存在就加载它

### GRUB2实用技巧

#### 定制自己的配置文件

配置文件在/boot/grub/grub.cfg，此文件是只读的不允许手动修改，grub-mkconfig在大多数情况下生成grub.cfg，他适合升级发行版系统，发现可用内核并打算加入菜单中，grub-mkconfig也有一些限制，当打算加入一个普通的菜单选项时可以编辑/etc/grub.d/40\_custom或者/boot/grub/custom.cfg。

/etc/default/grub控制grub-mkconfig的操作，grub里面是以键值对存在的选项，如果值有空格或者其他字符需要用引号引起来，参数如下：

1. GRUB\_DEFAULT：默认的菜单选择项，值可以是数字，默认从0开始，值也可以是title后面的字符串，当值是‘saved’时有特殊含义：默认的菜单选则项会被保存在GRUB\_SAVEDEFAULT中，下次启动时会从这个值启动。当值为saved时可以用grub-set-default和grub-reboot来设置默认启动项，grub-set-default直到下次修改前都有效，grub-reboot下次启动时生效
2. GRUB\_SAVEDEFAULT：只有这个值是true，GRUB\_DEFAULT是saved时才会起作用。
3. GRUB\_TIMEOUT：选择菜单的显示时间，默认是5，值是0表示不显示菜单选项，值是-1表示无限期的等待做出选择
4. GRUB\_HIDDEN\_TIMEOUT：grub2第一次执行时会寻找其他操作系统，如果没有检测到则会将菜单隐藏，如果有其他操作系统才会显示菜单，如果参数大于0，则会等待响应的秒数，但是不会显示菜单，可以按住shift显示菜单。
5. GRUB\_DISTRIBUTOR：菜单中的描述名称，采用命令lsb\_release判断，应该是红帽系列的，如果不是就归为Debian
6. GRUB\_CMDLINE\_LINUX：此行将追加到所有的linux 定义内核行的后面，不论是救援模式还是一般模式
7. GRUB\_CMDLINE\_LINUX\_DEFAULT：次选项只会追加在一般模式后面，同上
8. GRUB\_TERMINAL=console：启用console，默认注释状态
9. GRUB\_DISABLE\_LINUX\_UUID=true：在grub中指定root时可以使用UUID，默认注释
10. GRUB\_GFXMODE=640x480：图形化菜单的分辨率，默认注释
11. GRUB\_DISABLE\_LINUX\_RECOVERY=true：禁止显示救援模式

下面是一个通用于BIOS与UEFI模式的GRUB.cfg配置文件：

由于"$prefix"的值是在"grub-install"安装时确定的，并且嵌入'core.img'中的模块也是随硬件变化的，所以不要只是简单的复制'grub'目录到处使用，而应该在每一个介质上都使用"grub-install"进行安装。

###############################################################################

#(1)本配置文件要求"grub"目录所在分区的卷标必须是"GRUB2"。

#(2)为了保持最大程度的BIOS兼容性，"GRUB2"分区必须位于磁盘的前 137GB 范围。

###############################################################################

# 如果要在windows上安装GRUB2的话，必须首先以管理员身份打开命令提示符，然后运行

# wmic diskdrive list brief

# 命令查看安装目标，最后再使用例如下面这样的命令进行安装：

# grub-install.exe --boot-directory=g: --recheck --target=x86\_64-efi --efi-directory=g: --no-nvram --removable \\.\PHYSICALDRIVE5

# grub-install.exe --boot-directory=g: --recheck --target=i386-pc \\.\PHYSICALDRIVE5

#########################################

## (1)特殊变量 ##

#########################################

#禁止验证签名

set check\_signatures=no

#默认启动第一个菜单项

set default=0

#如果第一个菜单项启动失败，转而启动第二个菜单项

set fallback=1

#优先使用最常规的1024x768分辨率，以保证在不同的屏幕上拥有一致的菜单效果，如果失败再自动匹配分辨率

set gfxmode=1024x768,auto

#使用自己制作的24px的大号字体以避免默认字体太小看不清

set gfxterm\_font=WenQuanYiMicroHeiMono24px

#将GRUB2设置为简体中文界面

set lang=zh\_CN

#指定翻译文件(\*.mo)的目录，若未明确设置此目录，则无法显示中文界面。

set locale\_dir=$prefix/locale

#每一满屏后暂停输出，以免信息太多一闪而过看不清

set pager=1

#开启密码验证功能，并设置一个名为'root'的超级用户

set superusers=root

#设置菜单的超时时间为5秒

set timeout=5

#########################################

## (2)公共模块 ##

#########################################

#两种最流行的磁盘分区格式(partmap.lst)

insmod part\_gpt

insmod part\_msdos

#常见文件系统驱动(fs.lst)

insmod fat

insmod exfat

insmod ntfs

insmod iso9660

insmod ext2

insmod xfs

#一次性加载所有可用的视频驱动

insmod all\_video

#图形模式终端

insmod gfxterm

#背景图片支持

insmod png

#########################################

## (3)公共命令(必须放在模块和变量之后) ##

#########################################

#激活图形模式的输出终端，以允许使用中文和背景图

terminal\_output gfxterm

#设置背景图片

background\_image $prefix/themes/1024x768.png

#加载自己制作的24px的大号字体文件($prefix/fonts/WenQuanYiMicroHeiMono24px.pf2)

loadfont WenQuanYiMicroHeiMono24px

#设置'root'用户的哈希密码[通过"grub-mkpasswd-pbkdf2"工具生成]

password\_pbkdf2 root grub.pbkdf2.sha512.69.7DBCA469F80EA1C0A8A1E2FEBC4F8463.B073C1C89EC1E85309C3D6A1BAFF4356

#########################################

## (4)菜单项 ##

#########################################

menuentry '正常启动(Windows)' --unrestricted {

if [ 'pc' == $grub\_platform ] ; then

if search --file --set /bootmgr ; then

chainloader +1

elif search --file --set /ntldr ; then

chainloader +1

fi

elif [ 'efi' == $grub\_platform ] ; then

if search --file --set /EFI/Microsoft/Boot/bootmgfw.efi ; then

chainloader /EFI/Microsoft/Boot/bootmgfw.efi

fi

fi

}

#http://www.wepe.com.cn/download.html

menuentry '系统救援(WinPE)' --users=root {

if [ 'pc' == $grub\_platform -a -f $prefix/winpe/memdisk ] ; then

if [ -f $prefix/winpe/WinPE.iso ] ; then

linux16 $prefix/winpe/memdisk iso raw

initrd16 $prefix/winpe/WinPE.iso

fi

elif [ 'efi' == $grub\_platform -a -f $prefix/winpe/bootmgfw.efi -a -f $prefix/winpe/BCD ] ; then

if [ -f $prefix/winpe/WinPE.wim -a -f $prefix/winpe/boot.sdi ] ; then

chainloader $prefix/winpe/bootmgfw.efi

fi

fi

}

#https://mirrors.tuna.tsinghua.edu.cn/archlinux/iso/latest/

if [ -f $prefix/linux/archlinux.iso ] ; then

menuentry 'Arch Linux LiveCD [root/空]' --unrestricted {

loopback loop0 $prefix/linux/archlinux.iso

linux (loop0)/arch/boot/x86\_64/vmlinuz img\_label=GRUB2 img\_loop=/grub/linux/archlinux.iso systemd.wants=sshd.service

initrd (loop0)/arch/boot/x86\_64/archiso.img

}

fi

#https://mirrors.163.com/gentoo/releases/amd64/autobuilds/current-install-amd64-minimal/

#因为Gentoo官方的最小安装CD不支持UEFI启动，所以不可用于在UEFI环境中安装Gentoo

if [ -f $prefix/linux/install-amd64-minimal.iso ] ; then

menuentry 'Mini Gentoo LiveCD [root/123]' --unrestricted {

loopback loop0 $prefix/linux/install-amd64-minimal.iso

linux (loop0)/isolinux/gentoo cdroot isoboot=/grub/linux/install-amd64-minimal.iso dosshd nokeymap passwd=123

initrd (loop0)/isolinux/gentoo.igz

}

fi

#https://mirror.umd.edu/calculate/release/

#https://www.calculate-linux.org/main/en/download

# Calculate Linux Scratch 是基于Gentoo制作的 LiveCD ，

#包含编译工具(GCC,Binutils,...)、支持UEFI、支持WiFi，

#既可用作LFS(Linux From Scratch)的宿主系统、也可用于在UEFI环境中安装 Gentoo

if [ -f $prefix/linux/cls.iso ] ; then

menuentry 'Live GCC x64 [root/root]' --unrestricted {

loopback loop0 $prefix/linux/cls.iso

linux (loop0)/boot/vmlinuz root=live iso-scan/filename=/grub/linux/cls.iso vga=current nodevfs noresume nodmraid

initrd (loop0)/boot/initrd

}

fi

#https://mirrors.huaweicloud.com/centos/7/isos/x86\_64/

if [ -f $prefix/linux/CentOS-LiveGNOME.iso ] ; then

menuentry 'CentOS 7.5 GNOME LiveCD [root/空](可退出liveuser后再用root登录)' --unrestricted {

loopback loop0 $prefix/linux/CentOS-LiveGNOME.iso

linux (loop0)/isolinux/vmlinuz0 rd.live.image root=live:CDLABEL=CentOS-7-x86\_64-LiveGNOME-1804 iso-scan/filename=/grub/linux/CentOS-LiveGNOME.iso systemd.wants=sshd.service inst.lang=zh\_CN

initrd (loop0)/isolinux/initrd0.img

}

fi

#https://mirrors.ustc.edu.cn/debian-cd/current-live/amd64/iso-hybrid/

#也适用于'Kali LiveCD [root/toor]' https://cdimage.kali.org/current/

if [ -f $prefix/linux/debian-live-kde.iso ] ; then

menuentry 'Debian 9.5 KDE LiveCD (NO SSH)' --unrestricted {

loopback loop0 $prefix/linux/debian-live-kde.iso

linux (loop0)/live/vmlinuz-4.9.0-7-amd64 boot=live findiso=/grub/linux/debian-live-kde.iso components username=root locales=zh\_CN.UTF-8

initrd (loop0)/live/initrd.img-4.9.0-7-amd64

}

fi

#https://mirrors.aliyun.com/fedora/releases/28/Spins/x86\_64/iso/

#只有 Xfce LiveCD 可以设置中文界面

if [ -f $prefix/linux/Fedora-Xfce-Live.iso ] ; then

menuentry 'Fedora 28 Xfce LiveCD [root/空](可退出liveuser后再用root登录)' --unrestricted {

loopback loop0 $prefix/linux/Fedora-Xfce-Live.iso

linux (loop0)/isolinux/vmlinuz rd.live.image root=live:CDLABEL=Fedora-Xfce-Live-28-1-1 iso-scan/filename=/grub/linux/Fedora-Xfce-Live.iso systemd.wants=sshd.service locale.LANG=zh\_CN.utf8 inst.lang=zh\_CN

initrd (loop0)/isolinux/initrd.img

}

fi

#https://ftp.sjtu.edu.cn/opensuse/distribution/openSUSE-stable/live/

if [ -f $prefix/linux/openSUSE-Leap-KDE-Live.iso ] ; then

menuentry 'openSUSE Leap 15.0 KDE LiveCD [root/空]' --unrestricted {

loopback loop0 $prefix/linux/openSUSE-Leap-KDE-Live.iso

linux (loop0)/boot/x86\_64/loader/linux root=live:CDLABEL=openSUSE\_Leap\_15.0\_KDE\_Live iso-scan/filename=/grub/linux/openSUSE-Leap-KDE-Live.iso systemd.wants=sshd.service lang=zh\_CN

initrd (loop0)/boot/x86\_64/loader/initrd

}

fi

#http://cdimage.ubuntu.com/lubuntu/releases/

#也适用于 KDE neon LiveCD https://files.kde.org/neon/images/neon-userltsedition/current/neon-userltsedition-current.iso

if [ -f $prefix/linux/lubuntu.iso ] ; then

menuentry 'Ubuntu LXQt LiveCD (NO SSH)' --unrestricted {

loopback loop0 $prefix/linux/lubuntu.iso

linux (loop0)/casper/vmlinuz boot=casper iso-scan/filename=/grub/linux/lubuntu.iso username=root locale=zh\_CN keyboard-configuration/layoutcode=us

initrd (loop0)/casper/initrd.lz

}

fi

menuentry '关机' --unrestricted { halt ; }

menuentry '重新启动' --unrestricted { reboot ; }

#https://rhinstaller.github.io/anaconda/boot-options.html

#硬盘安装通用于所有包含"Packages"目录的 CentOS/Fedora ISO映像(Minimal,DVD,Everything)

#网络安装通用于所有包含"images"目录的 CentOS/Fedora ISO映像(Minimal,DVD,Everything,NetInstall)

#https://mirrors.zju.edu.cn/centos/7/isos/x86\_64/

if [ -f $prefix/linux/CentOS-Minimal.iso ] ; then

menuentry '硬盘安装 CentOS [最小安装]' --unrestricted {

loopback loop0 $prefix/linux/CentOS-Minimal.iso

linux (loop0)/isolinux/vmlinuz inst.repo=hd:LABEL=GRUB2:/grub/linux/CentOS-Minimal.iso inst.lang=zh\_CN

initrd (loop0)/isolinux/initrd.img

}

menuentry '网络安装 CentOS 7.x [不支持WiFi]' --unrestricted {

loopback loop0 $prefix/linux/CentOS-Minimal.iso

linux (loop0)/images/pxeboot/vmlinuz inst.repo=https://mirrors.aliyun.com/centos/7/os/x86\_64/ inst.lang=zh\_CN ip=dhcp nameserver=223.6.6.6

initrd (loop0)/images/pxeboot/initrd.img

}

fi

#https://www.debian.org/releases/stable/amd64/ch05s03.html.zh-cn

#https://www.debian.org/releases/stable/amd64/ch06s03.html.zh-cn

#最好将用于硬盘安装的ISO映像(CD/DVD-1.iso)放在文件系统的根目录或第一层子目录中(可简化ISO映像的搜索)

#https://mirrors.163.com/debian/dists/stable/main/installer-amd64/current/images/hd-media/

#https://mirrors.163.com/ubuntu/dists/bionic/main/installer-amd64/current/images/hd-media/

#硬盘安装也适用于 Alternative Ubuntu Server ISO (例如 ubuntu-18.04-server-amd64.iso )

if [ -f $prefix/linux/debian/vmlinuz -a -f $prefix/linux/debian/initrd.gz ] ; then

menuentry '硬盘安装 Debian [自动搜索ISO映像(最好放在根目录)]' --unrestricted {

linux $prefix/linux/debian/vmlinuz priority=low vga=791 locale=zh\_CN

initrd $prefix/linux/debian/initrd.gz

}

fi

#https://mirrors.163.com/debian/dists/stable/main/installer-amd64/current/images/netboot/

#https://mirrors.163.com/ubuntu/dists/bionic/main/installer-amd64/current/images/netboot/

#网络安装也适用于 Ubuntu 的 mini.iso 映像(支持WiFi网络)

if [ -f $prefix/linux/debian/mini.iso ] ; then

menuentry '网络安装 Debian [不支持WiFi]' --unrestricted {

loopback loop0 $prefix/linux/debian/mini.iso

linux (loop0)/linux priority=low vga=791 locale=zh\_CN

initrd (loop0)/initrd.gz

}

fi

#https://doc.opensuse.org/documentation/leap/startup/html/book.opensuse.startup/cha.boot\_parameters.html

#https://ftp.sjtu.edu.cn/opensuse/distribution/openSUSE-stable/iso/

if [ -f $prefix/linux/openSUSE-Leap-DVD.iso ] ; then

menuentry '硬盘安装 openSUSE Leap' --unrestricted {

loopback loop0 $prefix/linux/openSUSE-Leap-DVD.iso

linux (loop0)/boot/x86\_64/loader/linux install=hd:/grub/linux/openSUSE-Leap-DVD.iso lang=zh\_CN

initrd (loop0)/boot/x86\_64/loader/initrd

}

fi

#http://mirrors.nju.edu.cn/opensuse/distribution/openSUSE-stable/iso/

if [ -f $prefix/linux/openSUSE-Leap-NET.iso ] ; then

menuentry '网络安装 openSUSE Leap [支持WiFi]' --unrestricted {

loopback loop0 $prefix/linux/openSUSE-Leap-NET.iso

linux (loop0)/boot/x86\_64/loader/linux install=https://mirrors.aliyun.com/opensuse/distribution/openSUSE-stable/repo/oss/ lang=zh\_CN netsetup=dhcp nameserver=223.6.6.6

initrd (loop0)/boot/x86\_64/loader/initrd

}

fi

#### 重置 kali Linux 的登录密码

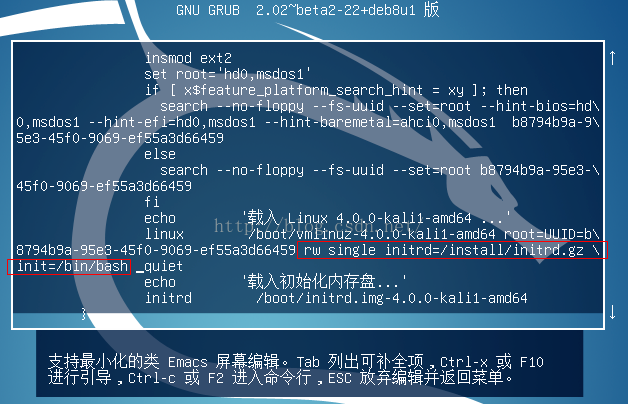
1. 打开kali系统在引导界面选择下面的选项（如图），不要点进去！



1. 然后点击键盘上的e键, 从这里找到 ro initrd=/install/initrd.gz quiet （如图）.



1. 他改成 rw singlw initrd=/install/initrd.gz init=/bin/bash quiet(如图)



1. 然后按Ctrl+x，回到引导界面选择第一项-回车
2. 这时你会进入到root权限的单用户操作界面
3. 输入passwd 回车
4. 这时会出现输入密码的文字，输入密码（密码不会显示），确认输入密码
5. 完了以后重启
6. 这时正常进入系统密码就是你刚刚输入的密码

#### 如何给GRUB2菜单加上背景图

GRUB 在4个地方寻找启动背景图像：

* /etc/default/grub 里的 GRUB\_BACKGROUND 行
* /boot/grub/ 里找到的第一个图像（如果发现多张，将以字母顺序排序）
* /usr/share/desktop-base/grub\_background.sh 中指定的
* /etc/grub.d/05\_debian\_theme 里 WALLPAPER 行列出的

1. 在 /etc/default/grub 文件中指定 GRUB\_BACKGROUND

如果没有在 /etc/default/grub 文件中指定 GRUB\_BACKGROUND 项，理论上 GRUB 应当使用在 /boot/grub/ 文件夹找到的第一个图像文件作为背景。如果 GRUB 在 /boot/grub/ 找到多个图像文件，它会按字母排序并使用第一个图像文件。

1. 在/etc/default/grub/grub\_background.sh 指定图像路径

如果没有在 /etc/default/grub 文件中指定 GRUB\_BACKGROUND 项，而且 /boot/grub/ 目录下没有图像文件，GRUB 将会开始在 /usr/share/desktop-base/grub\_background.sh 文件中指定的图像路径中搜索。Kali Linux 是在这里指定的（每个 Linux 发行版都可能略有区别）。

1. 在 /etc/grub.d/05\_debian\_theme 文件的 WALLPAPER指定图像路径

/etc/grub.d/05\_debian\_theme 文件的 WALLPAPER 一行指定图像，这是 GRUB 搜寻背景图像的最后一个位置。如果在其他部分都没有找到，它将会在这里查找。

理论，我们上述地方任意一个位置都可以更改新的grub开机画面。

我们将采用最简单的方法来改变GRUB2的背景图像：

1. 在kali2018版本中，在/etc/default/grub文件指定了GRUB2的部分配置信息，但没能使用GRUB\_BACKGROUND指定图像路径。

# If you change this file, run 'update-grub' afterwards to update

# /boot/grub/grub.cfg.

# For full documentation of the options in this file, see:

# info -f grub -n 'Simple configuration'

GRUB\_DEFAULT=0

GRUB\_TIMEOUT=5

GRUB\_DISTRIBUTOR=`lsb\_release -i -s 2> /dev/null || echo Debian`

GRUB\_CMDLINE\_LINUX\_DEFAULT="quiet"

GRUB\_CMDLINE\_LINUX="initrd=/install/gtk/initrd.gz"

# Uncomment to enable BadRAM filtering, modify to suit your needs

# This works with Linux (no patch required) and with any kernel that obtains

# the memory map information from GRUB (GNU Mach, kernel of FreeBSD ...)

#GRUB\_BADRAM="0x01234567,0xfefefefe,0x89abcdef,0xefefefef"

# Uncomment to disable graphical terminal (grub-pc only)

#GRUB\_TERMINAL=console

# The resolution used on graphical terminal

# note that you can use only modes which your graphic card supports via VBE

# you can see them in real GRUB with the command `vbeinfo'

#GRUB\_GFXMODE=640x480

# Uncomment if you don't want GRUB to pass "root=UUID=xxx" parameter to Linux

#GRUB\_DISABLE\_LINUX\_UUID=true

# Uncomment to disable generation of recovery mode menu entries

#GRUB\_DISABLE\_RECOVERY="true"

# Uncomment to get a beep at grub start

#GRUB\_INIT\_TUNE="480 440 1"

1. 在/boot/grub/中也没有图像。
2. 找到/usr/share/desktop-base/grub\_background.sh文件，发现grub\_background.sh是一个快捷方式，指向了/etc/alternatives/desktop-grub.sh文件。而/etc/alternatives/desktop-grub.sh文件又是一个软链接，指向了/usr/share/desktop-base/active-theme/grub/grub\_background.sh

# ls -al /usr/share/desktop-base/ grub\_background.sh

lrwxrwxrwx 1 root root 33 9月 23 07:02 grub\_background.sh -> /etc/alternatives/desktop-grub.sh

# ls -al /etc/alternatives/desktop-grub.sh

lrwxrwxrwx 1 root root 60 9月 23 07:02 /etc/alternatives/desktop-grub.sh -> /usr/share/desktop-base/active-theme/grub/grub\_background.sh

# vim /usr/share/desktop-base/active-theme/grub/grub\_background.sh

WALLPAPER=/usr/share/images/desktop-base/desktop-grub.png

COLOR\_NORMAL=white/black

COLOR\_HIGHLIGHT=black/white

1. 替换图片

从上图看出“WALLPAPER=/usr/share/desktop-base/active-theme/grub/grub-4x3.png”，真正的图位置为：/usr/share/desktop-base/active-theme/grub/grub-4x3.png。我们只需要新建一张大小为640\*480、颜色为8位（256色）的图像，把这个图像替换了就成。

1. 更新GRUB配置文件

# update-grub

配置文件在/boot/grub/grub.cfg，此文件是只读的不允许手动修改，update-grub命令用于自动生成/boot/grub/grub.cfg文件。

#### 如何让GRUB2显示中文界面(包括显示中文菜单项)？

由于GRUB2在内部使用UTF-8编码，并且所有文本文件(包括'grub.cfg')也都被假定为使用UTF-8编码，为了避免乱码，请务必以UTF-8编码保存'grub.cfg'文件。

set gfxterm\_font=unicode

set lang=zh\_CN

set locale\_dir=$prefix/locale

insmod gfxterm

terminal\_output gfxterm

loadfont unicode

#### 如何更改GRUB2的字体

如果你认为默认的unicode字体在1024x768或更高分辨率的屏幕上显得太小，或者你认为默认的字体不好看，想换换口味，那么如何自己动手制作一个pf2字体呢？那就要用到"grub-mkfont"工具。下面的示例展示了如何从一个ttc字体(文泉驿等宽微米黑)制作一个24px大小的pf2字体：

grub-mkfont -i1 -n WenQuanYiMicroHeiMono24px -o WenQuanYiMicroHeiMono24px.pf2 -s24 -v wqy-microhei.ttc

将制作好的字体文件(WenQuanYiMicroHeiMono24px.pf2)放到"$prefix/fonts"目录中，修改'grub.cfg'文件中的两行：

set gfxterm\_font=WenQuanYiMicroHeiMono24px

loadfont WenQuanYiMicroHeiMono24px

[注意]你最好使用等宽中文字体(推荐使用文泉驿等宽正黑或者等宽微米黑)，否则可能会让GRUB2的字体间距过大，十分难看。

修改/usr/share/desktop-base/active-theme/grub/grub\_background.sh文件。

#### 如何使用GRUB2引导WindowsPE的ISO文件？

GRUB4DOS有一个非常酷的'磁盘映射'功能，能够用于启动WinPE的ISO文件。其实，将GRUB2配合[MEMDISK](http://www.syslinux.org/wiki/index.php/MEMDISK)工具使用，同样可以引导各种映像文件，包括ISO文件与软/硬盘映像。

首先，你必须安装或者下载"[syslinux](https://www.kernel.org/pub/linux/utils/boot/syslinux/)"软件包，从中提取出"memdisk"文件(可能位于'/usr/share/syslinux/memdisk'或'bios/memdisk/memdisk')，然后将它复制到GRUB2的安装目录中(本文假定其位于"winpe"子目录中，也就是"$prefix/winpe/memdisk")。

然后，将你想要引导的WindowsPE的ISO文件放到某个地方(本文假定其位于"$prefix/winpe/WinPE.ISO")。当然，为了节约磁盘空间，你也可以用gzip对ISO文件进行压缩，不过这个示例中没有这么做。

最后，在'grub.cfg'中加入如下菜单项(如果你对ISO进行了gzip压缩，那么还需要额外再加上"insmod gzio"命令)：

menuentry "[BIOS] Windows PE" --unrestricted {

linux16 $prefix/winpe/memdisk iso raw

initrd16 $prefix/winpe/WinPE.ISO

}

[注意]与GRUB4DOS一样，由于[MEMDISK](http://www.syslinux.org/wiki/index.php/MEMDISK)对各种映像文件的模拟是通过在实模式下拦截BIOS的 INT 13h 与 INT 15h 调用来实现的，所以有很大的局限性：

* 只能用于BIOS模式启动，不能用于UEFI模式启动
* 模拟出来的软盘/光盘/硬盘设备只能被基于实模式的操作系统所识别(DOS,FreeDOS)，不能被基于保护模式的操作系统所识别(Windows,Linux,BSD)
* 从实用的角度来说，只能用于引导WinPE的ISO以及基于DOS/FreeDOS的映像，不能用于引导各种Linux的LiveCD ISO以及微软原版的Windows ISO安装光盘。

[释疑]不要将[MEMDISK](http://www.syslinux.org/wiki/index.php/MEMDISK)与GRUB2的"memdisk.mod"模块混淆，他们是两个毫不相干的东西。"memdisk.mod"模块的作用是为'core.img'提供内存盘支持，其目的是为了让GRUB2能够正确识别启动设备。如果把'core.img'比作Linux内核，那么"memdisk.mod"模块的作用就相当于为'core.img'这个"内核"提供了'initramdisk'，用以加载磁盘与文件系统驱动，从而让GRUB2可以访问磁盘，进而加载其他的模块。但是实际上，在绝大多数情况下，更本不需要使用"memdisk.mod"，因为在'grub-install'的时候，这些驱动已经被嵌入到'core.img'中了。如果你还是不明白"memdisk.mod"模块的作用的话，那就忘记它的存在吧，因为你更本不需要知道有这个东西，就像你不需要了解全部GRUB2模块一样。仅仅是因为这两个东西的名字相同，我才在这里多罗嗦了几句而已。

#### 如何使用GRUB2引导WindowsPE的WIM文件？

由于GRUB2不能在UEFI模式下对ISO文件进行仿真，那么我们应该如何在UEFI模式下引导WindowsPE呢？答案是必须使用WIM格式的WindowsPE。

具体说来就是首先用GRUB2链式加载微软的"bootmgfw.efi"引导管理器，然后再由"bootmgfw.efi"根据BCD文件的指引去启动WindowsPE。

第一步，从例如"[微PE](http://www.wepe.com.cn/)"这样的作品中提取"WEPE64.WIM"与"WEPE.SDI"(BOOT.SDI)文件。

第二步，从Win10的原版安装光盘中提取"bootmgfw.efi"文件(/efi/boot/bootx64.efi)。

第三步，将提取的三个文件放置到一个FAT32或NTFS磁盘分区上，这里假定放到'(hd0,gpt3)/winpe/'目录中。

第四步，仿照下面的命令序列编写一个BCD文件：

bcdedit /createstore BCD

bcdedit /store BCD /create {dddddddd-dddd-dddd-dddd-dddddddddddd} /device

bcdedit /store BCD /set {dddddddd-dddd-dddd-dddd-dddddddddddd} RAMDISKSDIDEVICE BOOT

bcdedit /store BCD /set {dddddddd-dddd-dddd-dddd-dddddddddddd} RAMDISKSDIPATH "\winpe\WEPE.SDI"

bcdedit /store BCD /create {aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa} /d "UEFI Windows PE x64" /application OSLOADER

bcdedit /store BCD /set {aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa} DEVICE RAMDISK=[BOOT]\winpe\WEPE64.WIM,{dddddddd-dddd-dddd-dddd-dddddddddddd}

bcdedit /store BCD /set {aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa} HIGHESTMODE YES

bcdedit /store BCD /set {aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa} OSDEVICE RAMDISK=[BOOT]\winpe\WEPE64.WIM,{dddddddd-dddd-dddd-dddd-dddddddddddd}

bcdedit /store BCD /set {aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa} SYSTEMROOT "\Windows"

bcdedit /store BCD /set {aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa} WINPE YES

bcdedit /store BCD /set {aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa} LOCALE "zh-CN"

bcdedit /store BCD /set {aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa} NX OptIn

bcdedit /store BCD /set {aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa} PAE ForceEnable

bcdedit /store BCD /set {aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa} detecthal Yes

bcdedit /store BCD /set {aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa} DEBUGSTART DISABLE

bcdedit /store BCD /set {aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa} RECOVERYENABLED NO

bcdedit /store BCD /bootems {aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa} OFF

bcdedit /store BCD /ems {aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa} OFF

bcdedit /store BCD /bootdebug {aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa} OFF

bcdedit /store BCD /debug {aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa} OFF

bcdedit /store BCD /event {aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa} OFF

bcdedit /store BCD /create {bootmgr} /d "Windows PE Boot Manager"

bcdedit /store BCD /set {bootmgr} DEVICE BOOT

bcdedit /store BCD /set {bootmgr} LOCALE "zh-CN"

bcdedit /store BCD /set {bootmgr} nointegritychecks Yes

bcdedit /store BCD /displayorder {aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa}

bcdedit /store BCD /default {aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa}

bcdedit /store BCD /timeout 10

第五步，将生成的BCD文件同样放置到'(hd0,gpt3)/winpe/'目录中。

最后，在'grub.cfg'中加入如下菜单项：

menuentry 'UEFI Windows PE Boot Manager' --unrestricted {

chainloader (hd0,gpt3)/winpe/bootmgfw.efi

}

#### 如何使用GRUB2引导(硬盘安装) Gentoo LiveCD 的ISO文件？

以"install-amd64-minimal-\*.iso"为例。

第一步，将ISO中的'/isolinux/{gentoo,gentoo.igz}'、'/image.squashfs'三个文件放到'(hd0,gpt3)/os/gentoo/'目录中；

第二步，将ISO中的'/livecd'放到相同分区(hd0,gpt3)的根目录下；

最后，在'grub.cfg'中加入如下菜单项：

menuentry "Gentoo Minimal Install LiveCD" --unrestricted {

linux (hd0,gpt3)/os/gentoo/gentoo cdroot looptype=squashfs loop=/os/gentoo/image.squashfs

initrd (hd0,gpt3)/os/gentoo/gentoo.igz

}

[提示]'livecd'是寻找'image.squashfs'所在磁盘分区的关键。

[提示] Gentoo LiveCD 亦可使用与其他Linux发行版的LiveCD类似的方法启动(也就是使用"isoboot="参数)。

#### 如何使用GRUB2引导(硬盘安装)各种 Linux LiveCD 的ISO文件？

首先需要说明的是，这里给出的方法，只适用于提供了"img\_loop="或"iso-scan/filename="或"fromiso="或"isoboot="或"isoloop="之类参数的LiveCD。

下面以 Ubuntu 的 LiveCD 为例说明。首先，假定你将ISO文件放在'(hd0,gpt3)/ISO/Ubuntu.iso'；然后，在'grub.cfg'中加入如下菜单项：

menuentry "Ubuntu LiveCD" --unrestricted {

loopback loop0 (hd0,gpt3)/ISO/Ubuntu.iso

linux (loop0)/casper/vmlinuz boot=casper iso-scan/filename=/ISO/Ubuntu.iso

initrd (loop0)/casper/initrd.lz

}

[说明]这里给出的方法，其实就是各种"硬盘安装 XX Linux"的翻版，只不过不再需要将"vmlinuz"与"initrd"从ISO中解压出来而已。

更多其它发行版的实例，请继续阅读下面的内容。

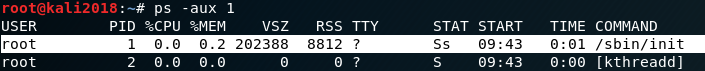
## 第四步：加载内核

系统将解压后的内核放置在内存中（这里说系统到底是哪一个系统呢？）开始调用start\_kernel()函数， 来初始化各种设备，到这里内核环境已经建立起来了。

上面的四步两个版本没差别。

## 第五步：启动systemd进程

从RH7/Kali linux2018.2使用了systemd代替了init，开机速度更快（各个服务平行运行）。可使用ps -aux进行查看



从上图可以看出，系统运行的第一个进程依然是/sbin/init。



查看这个命令的详细情况时，发现init只是一个软链接，指向了/lib/systemd/systemd，这样做的目的是为了保证与旧版linux的兼容性。由此可见，systemd才是Linux运行的第一个命令。systemd的大概流程如下：

### 阶段1

systemd启动后，首先会去三个目录下找相应的配置文件，按优先级从高到底为：

/etc/systemd/System

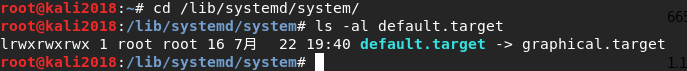
/usr/lib/systemd/system

/lib/systemd/system

优先级高的配置文件会覆盖优先级低的配置文件。

在上述3个目录中寻名为：default.target的目标，通常情况下为：

/lib/systemd/system/default target



default.target又是一个软链接，其指向当前目录下graphical.target，文件内容如下：

[Unit]

Description=Graphical Interface

Documentation=man:systemd.special(7)

Requires=multi-user.target

Wants=display-manager.service

Conflicts=rescue.service rescue.target

After=multi-user.target rescue.service rescue.target display-manager.service

AllowIsolate=yes

文件中配置项 Requires=multi-user.target 将控制权交给 multi-user.target (相当于旧概念的运行级别)。graphical.target常用的命令：

* Description=　：一些描述，显示给用户界面看的，可以是任何字符串，一般是关于服务的说明。
* Documentation=　：指定参考文档的列表，以空格分开的 URI 形式，如http://, https://, file:, info:, man:，这是有顺序的，最好是先解释这个服务的目的是什么，然后是它是如何配置的，再然后是其它文件，这个选项可以多次指定，会将多行的合并，如果指定了一个空的，那么会重置此项，前的配置不在起作用。
* Requires=　：指定此服务依赖的其它服务，如果本服务被激活，那么 Requires 后面的服务也会被激活，反之，如果 Requires 后面的服务被停止或无法启动，则本服务也会停止。这个选项可以指定多次，那么就要求所有指定的服务都被激活。需要注意的是这个选项不影响启动或停止的顺序，启动顺序使用单句的 After= 和 Before= 来配置。例如，如果 foo.service 依赖  bar.serivce，但是只配置了 Requires= 而没有 After= 或 Before=，那么 foo.service 启动时会同时激活 foo.service 和 bar.service。通常使用 Wants= 代替 Requires= 是更好的选择，因为系统会更好的处理服务失败的情况。注意，这种依赖关系，也可以在文件之外来处理，即使用 .requires/ 目录，可以参看上面的说明。
* RequiresOverridable=　：类似上面的 Requires= ，不过这种情况下，只要用户明确要求它启动，才会影响到被依赖的服务，不然服务出错什么的，不会影响被依赖服务的启动。
* Requisite=, RequisiteOverridable=：分别类似上面的两个，不过如果是这个指定服务没有启动，被依赖的服务会不启动，立即失败。
* Wants=　：相对弱化的 Requires= ，这里列出的服务会被启动，但如果无法启动或无法添加到事务处理，并不影响本服务做为一个整体的启动。这是推荐的两个服务关联的方式。这种依赖也可以配置文件外，通过 .wants/ 目录添加，具体可以看上面的说明。
* BindsTo=　：和 Requires= 很像，但是这种情况，如果他后面列出的服务停止运行或崩溃之类的，本服务也会同时停止。
* PartOf=　：又一个类似 Requires= 的选项，但是限制在停止或重启动服务，如果这里列出的服务被停止或重启动，那么本服务也会停止或重启动，注意这个依赖是意向，即本服务停止或重启动，不会影响到这里列出服务的运行状态。
* Conflicts=　：配置一个依赖冲突，如果配置了些项，那么，当一个服务启动时，或停止此处列出的服务，反过来，如果这里列出的服务启动，那么本服务就会停止，即后启动的才起作用。注意，此设置和 After= 和 Before= 是互相独立的。如果服务 A 和 B 冲突，且在 B 启动的时候同时启动，那么有可能会启动失败（两个都是必需的）或修改以修复它（两者之一或两都都不是必需的），后一种情况，会将不需要的依赖删除，或停止冲突。
* Before=　,After=　：配置服务间的启动顺序，比如一个 foo.service 包含了一行 Before=bar.service，那么当他们同时启动时，bar.service 会等待 foo.service 启动完成后才启动。注意这个设置和 Requires= 的相互独立的，同时包含 After= 和 Requires= 也是常见的。此选项可以指定一次以上，这时是按顺序全部启动。
* OnFailure=　 ：列出一个或更多的服务，当本服务启动状态是 failed 的时候，激活这些服务。
* PropagatesReloadTo=　,ReloadPropagatedFrom=　：这两个是列出一些服务，当其它服务 reload 时同时 reload 这个服务，或者反之。
* RequiresMountsFor=　：用空格分开的绝对路径列表，是 Requires= 和 After= 添加的依赖中的 mount 文件需要访问的指定的路径。
* OnFailureIsolate=　：是一个布尔值，如果是真，那么 OnFailure= 后面的服务会进入隔离模式，即所有不是它依赖的服务都会停止。如果只设置一个服务，可以放在 OnFailure= 后，默认值是假。
* IgnoreOnIsolate=　：一个布尔值.如果是真则当隔离其它服务时本服务不会停止（不明白隔离是什么意思，大概在后面）。默认是假。
* IgnoreOnSnapshot=　：一个布尔值.如果是真则本服务不包含快照(snapshots)。对 device 和 snapshot 服务默认为真，其它服务默认为假。
* StopWhenUnneeded=　：一个布尔值。如果是真则当本服务不使用时会停止。 注意，为了尽量减少 systemd 的工作，默认情况下是不会停止不使用的服务的，除非和其它服务冲突，或用户明确要求停止。如果设置了这个选项，那么如果没有其它活动的服务需要此服务，它会自动停止。默认值是假。
* RefuseManualStart=　,RefuseManualStop=　：布尔值。如果设为真值，则此服务只能间接的激活或停止。这种情况下，用户直接启动或停止此服务会被拒绝，只有做为其它的服务依赖关系，由其它服务进行启动或停止才可以。这主要是为了停止用户误操作。默认值是假。
* AllowIsolate=：布尔值。如果是真值，则此服务可以使用 systemctl isolate 命令进行操作。否则会拒绝此操作。最好的办法是不要动这处选项，除非目标服务的行为类似于 SysV 启动系统中的 runlevels。只是一种预防措施，避免系统无法使用的状态。默认值是假。
* DefaultDependencies=：布尔值。如果是真（默认值），一些本服务默认的依赖会隐式的建立，具体是哪些依赖，则于服务的类型决定。比如，对于普通的服务（.service类型），它会确保在系统基本服务启动后才启动本服务，会在系统关机前确保本服务已关闭。一般来说，只有早期开机服务和后期的关机服务，才需要把这个设成假。强烈对大多数普通服务，让这个选项启用即可。如果设成假，也不会禁用所有的隐式依赖，只是禁用那些非必要的。
* JobTimeoutSec=：当一个客户端等待本服务的某个 Job 完成时，所指定的超时时间。如果达到了限制的时间，此 Job 会取消运行，但服务不会更改状态，包括进入“failed”状态。除了设备服务（即.device类型），其它的默认值是0（即没有超时设置）。注意，这个是独立于特定服务所设置的超时设置的（比如对 .service 类型所设置的 Timeout=），它对服务本身没有影响，但特定服务的设置是有影响的（能用来更改服务状态）。
* ConditionPathExists=, ConditionPathExistsGlob=, ConditionPathIsDirectory=, ConditionPathIsSymbolicLink=, ConditionPathIsMountPoint=, ConditionPathIsReadWrite=, ConditionDirectoryNotEmpty=,ConditionFileNotEmpty=, ConditionFileIsExecutable=, ConditionKernelCommandLine=, ConditionVirtualization=, ConditionSecurity=, ConditionCapability=, ConditionHost=, ConditionACPower=,ConditionNull=：这是一组类似的东西。检测特定的条件是不是真值，如果不是真值，服务会略过启动，但是它依赖的服务还是会正常运行的。这个条件测试失败不会让服务进入失败状态。条件是在服务开始运行时检查的。
* ConditionPathExists= 是指定在服务启动时检查指定文件的存在状态。如果指定的绝对路径名不存在，这个条件的结果就是失败。如果绝对路径的带有!前缀，则条件反转，即只有路径不存在时服务才启动。
* ConditionPathExistsGlob=　类似上面的选项，但支持通配符。
* ConditionPathIsDirectory=　判断指定路径是不是目录。
* ConditionPathIsSymbolicLink=　判断指定路径是不是链接。
* ConditionPathIsMountPoint=　判断指定路径是不是一个挂载点。
* ConditionPathIsReadWrite=　多年指定路径是否可读写（即不是做为只读系统挂载的）
* ConditionDirectoryNotEmpty=　判断指定目录是否存在且不为空。
* ConditionFileNotEmpty=　判断指定文件是否是常规文件且不为空（即大小不是0）。
* ConditionFileIsExecutable=　判断指定文件是否是常规文件且可执行。
* 类似的，ConditionKernelCommandLine=是判断有没有指定的内核命令行启动参数（或带有!反之），这个参数必须是一个单词或用=分开的两个单词，前一种情况下，会寻找内核参数是否有此单词或是赋值的左边。后一种情况则必须是赋值的左右同时符合。
* ConditionVirtualization=　是判断是不是在虚拟化环境下执行的服务。这可以是个布尔值以判断是不是任意的虚拟化环境，或者下列的字符串之一： qemu, kvm, vmware, microsoft, oracle, xen, bochs, chroot, openvz, lxc, lxc-libvirt, systemd-nspawn，以判断是不是特定的虚拟化环境，多重嵌套的虚拟化环境，只判断最后一层。可以使用!进行反转判断。
* ConditionSecurity= ：是判断系统是否启用了安全环境，当前仅能识别selinux, apparmor, 和 smack。可以使用!进行反转判断。
* ConditionCapability= 是判断服务管理器绑定的 capability 是否存在。（可以查看其它部分的详细信息。）设置为 capability 的名字，比如 CAP\_MKNOD。可以通过在前面加!反转判断。
* ConditionHost=　是判断主机名 (hostname)或机器ID(machine ID)是否匹配。可以加!反转。
* ConditionACPower=　是判断机器是否在使用交流电源。如果设成 true，而只有至少连接一个交流电源时结果才为真，反过来，设成 false，则不连接所有交流电源时才为真。
* 最后，ConditionNull=　是一个常量性质的判断条件，它应该是布尔值，如果设成 false ，则条件永远失败，反过来则永远成立。
* 如果指定多个条件，则所有条件都需要成立（即条件之间是 AND 的关系）。条件前面可以加上 | 符号，这时条件变成一个触发条件，服务定义了触发条件，那么在满足其它非触发条件和这个触发条件的情况下，服务会至少执行一次。同时指定|和!前缀时，先处理|，后处理!。除了ConditionPathIsSymbolicLink=，其它条件均跟随链接。如果这些条件指定为空，则相当于重置，前面的任何设置都不再起作用。
* SourcePath=：这个服务生成的配置文件所在的路径，这主要是用在生成工具从外部配置文件的格式转换到本地服务的配置格式中。因此，对一般的服务不要使用此选项。
* 服务文件还可能包含一个 [Install] 段，这个段的内容服务的安装信息。它不在 systemd 的运行期间使用。只在使用 systemctl enable 和 systemctl disable 命令启用/禁用服务时有用。
* Alias=：在安装使用应该使用的额外名字（即别名）。名字必须和服务本身有同样的后缀（即同样的类型）。这个选项可以指定多次，所有的名字都起作用，当执行 systemctl enable 命令时，会建立相当的链接。
* WantedBy=, RequiredBy=　：在 .wants/ 或 .requires/ 子目录中为服务建立相应的链接。这样做的效果是当列表中的服务启动，本服务也会启动。 在bar.servic中的 WantedBy=foo.service 　和 Alias=foo.service.wants/bar.service 基本是一个意思。
* Also=：当此服务安装时同时需要安装的附加服务。如果用户请求安装的服务中配置了此项，则systemctl enable 命令执行时会自动安装本项所指定的服务。

### 阶段2

在这个阶段，会启动multi-user.target，这个target为多用户支持设定系统环境。会启动如下两个目录中的目标。

/etc/systemd/system/multi-user.target.wants/

/usr/lib/systemd/system/multi-user.target.wants/

multi-user.target文件内容如下：

[Unit]

Description=Multi-User System

Documentation=man:systemd.special(7)

Requires=basic.target

Conflicts=rescue.service rescue.target

After=basic.target rescue.service rescue.target

AllowIsolate=yes

multi-user.target会将控制权交给 basic.target。

### 阶段3

basi.target用于启动普通服务，特别是图形管理服务，它通过如下两个目录决定那些单元会被启动。

/etc/systemd/system/basic.target.wants/

/usr/lib/systemd/system/basic.target.wants/

basi.target 文件内容如下：

[Unit]

Description=Basic System

Documentation=man:systemd.special(7)

Requires=sysinit.target

After=sysinit.target

Wants=sockets.target timers.target paths.target slices.target

After=sockets.target paths.target slices.target

basi.target 会将控制权交给 sysinit.target

### 阶段4

sysinit.target会启动重要的系统服务例如系统挂载，内存交换空间和设备，内核补充选项等等。它通过如下两个目录决定那些单元会被启动。

/etc/systemd/system/sysinit.target.wants/

/usr/lib/systemd/system/sysinit.target.wants/

sysinit.target文件内容如下：

[Unit]

Description=System Initialization

Documentation=man:systemd.special(7)

Conflicts=emergency.service emergency.target

Wants=local-fs.target swap.target

After=local-fs.target swap.target emergency.service emergency.target

sysinit.target在启动过程中会传递给local-fs.target 和 swap.target。

### 阶段5

local-fs.target 也不能算是阶段5，可能和阶段4一些服务并行启动，这个阶段不会启动用户相关服务，它只处理底层核心服务，这个target会根据/etc/fstab来执行相关磁盘挂载操作。它通过如下一个目录决定那些单元会被启动。

/usr/lib/systemd/system/local-fs.target.wants/

local-fs.target 文件内容如下：

[Unit]

Description=Local File Systems

Documentation=man:systemd.special(7)

DefaultDependencies=no

Conflicts=shutdown.target

After=local-fs-pre.target

OnFailure=emergency.target

OnFailureJobMode=replace-irreversibly

### 总结

个人理解：虽然systemd的引用target的顺序如上，但是真正的启动顺序为从下到上(从第5阶段->第一阶段)，因为下一层作为上一层的依赖而存在，因此先运行的应该是依赖服务。在新版的Linux中，管理单元可以并行启动，系统会为每个运行的进程预先分配一个空的sodkete用于存放进程的运行结果，以解决依赖性问题，至于sodkete的内容可以在以后由进程执行完毕后更新，这样使效率大大提高。其实5个级别的先后顺序关系不大，同时 Systemd 是向下兼容 System V 的。

## Systemd与init主要区别：

### 取消运行级别

在RH7/Kali linux2018.2中取消的0~6级的运行级别的概念，采用了目标的概念，原来“init 0~6”的运行级别命令变成了：

systemctl poweroff

systemctl isolate rescure.target

systemctl isolate multi-usr.target

systemctl isolate graphical.target

systemctl teboot

更改默认系统级别

root@kali2018:~# systemctl set-default multi-user.target

Created symlink /etc/systemd/system/default.target → /lib/systemd/system/multi-user.target.

从上面可以看出systemd使用链接来指向默认的运行级别。在创建新的链接前，可以通过下面命令删除存在的链接：

#rm /etc/systemd/system/default.target

默认启动运行级别3 ：

#ln -sf /lib/systemd/system/multi-user.target /etc/systemd/system/default.target

默认启动运行级别5 ：

#ln -sf /lib/systemd/system/graphical.target/etc/systemd/system/default.target

### /etc/inittab文件取消

systemd不使用/etc/inittab文件（有也是为了兼容）。取而代之的是 /etc/systemd/system/default.target，这是一个软链接指向/usr/lib/systemd/system/graphical.target，当改变默认运行级别的时候，软连接指向的文件也随之改变

### 系统服务管理命令改变

系统服务命令由systemctl命令代替了service命令，详见“4.3.3服务管理命令—systemctl”

## /etc/rc.d/rcX.d脚本

一堆以“S+两数字”和“K+两数字”开关的链接文件。S代表启动服务，K代表关闭服务，后面的两位数据代表优先级别。只有在链接文件存在，系统才会在开机或关机时自动启动或关闭相应服务，用chkconfig命令可以管理链接文件。

chkconfig //服务链接文件管理

--del //删除服务的连接文件

--add //创建服务的连接文件

--list //查看

--level 级别 服务名 开|关

## 开机流程总结



# LINUX防火墙

## 防火墙概述

防火墙的概念：防火墙工作在网络或主机边缘，对进出网络或本机的网络报文按照事先设定好的匹配规则进行检查，对能够被规则所匹配的报文按照规则定义的处理机制进行处理的组件。

防火墙分为网络防火墙和主机防火墙的需要配合使用。

防火墙与防病毒软件不是一个东西，是有区别的，市面上有具备防病毒功能的防火墙，其实是两个防火墙组件与防病毒组件的组合。

防火墙应该工作在操作系统内核和TCP/IP协议栈中，可以对TCP/IP协的标志位进行识别与控制。必须要具备协议解析与控制能力。

应用层网关：一般防火墙工作在OSI网络模型中的TCP/IP层，无法对7层的应用层进行解析和控制。如果需要具备7层协议安全防护功能，必须具备解析7层协议的能力。因此市面上出现了各种针对7层协议进行防护的设备，这种对7层协议进行解析与阻断的组件，我们一般称这为“应用层网关”。出于性能考虑，防火墙与应用层网关应该分开，各司其职，相互配合。7层协议众多，且私有的7层应用协议不断和被开发出来，因此，没有任何7层应用网关可以防护所有7层协议，一般是针对几种特定的7层应用协议进行专门防护，所以7层应用网关一般应用于的专用领域，通用性不如网络防火墙，但最能贴近用户需求。

入侵检测系统(IDS):对所有访问行为进行监视和记录的组件。并且能够根据预先定义的恶意行为进行告警。常见的IDS软件有：

* NIDS软件:snort
* HIDS:OSSEC
* 文件系统入侵检测软件：tripwire

入侵防御系统(IPS):实际上就防火墙与IDS实现连动。IDS检测到恶意行为后自动生成防火墙策略进行阻断。

蜜罐（honepot）:使用虚假目标或大量探针进行数据收集并进行统一的趋势分析的组件。因数据量比较庞大，对计算能力要求比较高。

## Firewalld防火墙工作机制

在新版的linux中，已经默认使用 firewalld 作为防火墙，基于 iptables 的防火墙被默认不启动，但仍然可以继续使用。 RHEL7以后的linux中有几种防火墙共存：firewalld、iptables、ebtables 等，默认使用firewalld 作为防火墙，管理工具是 firewall-cmd。内核版本3.10以后，内核里防火墙的包过滤机制是firewalld，使用firewalld来管理netfilter,不过底层调用的命令仍然是iptables等。因为这几种daemon是冲突的，所以建议禁用其他几种服务，虽然iptables依然存在，但是不建议使用了，建议使用新的 firewalld 服务。

Firewalld提供了支持区域(zone)以及接口安全等级的防火墙管理工具。拥有运行时配置和永久配置选项，支持自定义扩展服务。以前的system-config-firewall防火墙模型是静态的，每次修改都要求防火墙完全重启。这个过程包括内核netfilter防火墙模块的卸载和新配置所需模块的装载等。相反，firewall daemon动态管理防火墙，不需要重启整个防火墙便可应用更改。因而也就没有必要重载所有内核防火墙模块了。

## 防火墙工作流程



Firewalld的9个系统自带的区域：

* 丢弃区域（Drop Zone）：如果使用丢弃区域，任何进入的数据包将被丢弃。这个类似与我们之前使用 iptables -j drop。使用丢弃规则意味着将不存在响应。
* 阻塞区域（Block Zone）：阻塞区域会拒绝进入的网络连接，返回 icmp-host-prohibited，只有服务器已经建立的连接会被通过即只允许由该系统初始化的网络连接。
* 公共区域（Public Zone）：只接受那些被选中的连接，默认只允许 ssh 和 dhcpv6-client。这个 zone 是缺省
* 外部区域（External Zone）：这个区域相当于路由器的启用伪装（masquerading）选项。只有指定的连接会被接受，即 ssh，而其它的连接将被丢弃或者不被接受。
* 隔离区域（DMZ Zone）：如果想要只允许给部分服务能被外部访问，可以在 DMZ 区域中定义。它也拥有只通过被选中连接的特性，即 ssh。
* 工作区域（Work Zone）：在这个区域，我们只能定义内部网络。比如私有网络通信才被允许，只允许 ssh，ipp-client 和 dhcpv6-client。
* 家庭区域（Home Zone）：这个区域专门用于家庭环境。它同样只允许被选中的连接，即 ssh，ipp-client，mdns，samba-client 和 dhcpv6-client。
* 内部区域（Internal Zone）：这个区域和工作区域（Work Zone）类似，只有通过被选中的连接，和 home 区域一样。
* 信任区域（Trusted Zone）：信任区域允许所有网络通信通过。记住：因为 trusted 是最被信任的，即使没有设置任何的服务，那么也是被允许的，因为 trusted 是允许所有连接的。

## Firewall防火墙配置原则

以下要点请牢记，先死记硬下来，以后慢慢理解：

* 运行时配置（临时配置）≠永久配置，要分两次分别配置！
* 运行时配置（临时配置）即时生效，永久配置需要重新载入防火墙服务才能生效！
* 一个源IP只能属于一个区域！

## 防火墙常用命令

### 一般应用

#### 安装防火墙

# apt-get install firewalld

#### 查看firewalld服务是否运行

# systemctl status firewalld

● firewalld.service - firewalld - dynamic firewall daemon

Loaded: loaded (/lib/systemd/system/firewalld.service; disabled; vendor prese

Active: inactive (dead)

Docs: man:firewalld(1)

服务未启动。

#### 获取firewalld状态

# firewall-cmd --state

Running

#### 开启firewalld服务

# systemctl start firewalld.service

# firewall-cmd --state

Running

#### 重启firewalld服务

# systemctl reload firewalld.service

# firewall-cmd --state

Running

#### 关闭firewalld服务

# systemctl stop firewalld.service

# firewall-cmd --state

not running

#### 在不改变状态条件下加载防火墙

# systemctl stop firewalld.service

# firewall-cmd --state

not running

# firewall-cmd --reload

FirewallD is not running

# systemctl start firewalld.service

# firewall-cmd --reload

Success

#### 设置开机自动启动

# systemctl enable firewalld.service

#### 设置关闭开机制动启动

# systemctl disable firewalld.service

#### 启用某个服务

//临时

# firewall-cmd --zone=public --add-service=https

//永久

# firewall-cmd --permanent --zone=public --add-service=https

#### 开启某个端口

//临时

# firewall-cmd --zone=public --add-port=8080-8081/tcp

//永久

# firewall-cmd --permanent --zone=public --add-port=8080-8081/tcp

#### 查看开启的端口和服务

//服务空格隔开 例如 dhcpv6-client https ss

# firewall-cmd --permanent --zone=public --list-services

//端口空格隔开 例如 8080-8081/tcp 8388/tcp 80/tcp

# firewall-cmd --permanent --zone=public --list-ports

#### 修改配置后需要重启服务使其生效

# systemctl restart firewalld.service

#### 查看服务是否生效（例：添加的端口为8080）

# firewall-cmd --zone=public --query-port=8080/tcp

#### 获取所支持区域的列表

# firewall-cmd --get-zones

block dmz drop external home internal public trusted work

#### 获取所有支持的服务

# firewall-cmd --get-services

RH-Satellite-6 amanda-client amanda-k5-client amqp amqps apcupsd bacula bacula-client bgp bitcoin bitcoin-rpc bitcoin-testnet bitcoin-testnet-rpc ceph ceph-mon cfengine cockpit condor-collector ctdb dhcp dhcpv6 dhcpv6-client distcc dns docker-registry docker-swarm dropbox-lansync elasticsearch etcd-client etcd-server finger freeipa-ldap freeipa-ldaps freeipa-replication freeipa-trust ftp ganglia-client ganglia-master git gre high-availability http https imap imaps ipp ipp-client ipsec irc ircs iscsi-target isns jenkins kadmin kerberos kibana klogin kpasswd kprop kshell ldap ldaps libvirt libvirt-tls lightning-network llmnr managesieve matrix mdns minidlna mongodb mosh mountd mqtt mqtt-tls ms-wbt mssql murmur mysql nfs nfs3 nmea-0183 nrpe ntp nut openvpn ovirt-imageio ovirt-storageconsole ovirt-vmconsole plex pmcd pmproxy pmwebapi pmwebapis pop3 pop3s postgresql privoxy proxy-dhcp ptp pulseaudio puppetmaster quassel radius redis rpc-bind rsh rsyncd rtsp salt-master samba samba-client samba-dc sane sip sips slp smtp smtp-submission smtps snmp snmptrap spideroak-lansync squid ssh steam-streaming svdrp svn syncthing syncthing-gui synergy syslog syslog-tls telnet tftp tftp-client tinc tor-socks transmission-client upnp-client vdsm vnc-server wbem-http wbem-https wsman wsmans xdmcp xmpp-bosh xmpp-client xmpp-local xmpp-server zabbix-agent zabbix-server

列出默认有效的服务也可以进入下面的目录查看

# ls /usr/lib/firewalld/services

amanda-client.xml jenkins.xml radius.xml

amanda-k5-client.xml kadmin.xml redis.xml

amqps.xml kerberos.xml RH-Satellite-6.xml

amqp.xml kibana.xml rpc-bind.xml

apcupsd.xml klogin.xml rsh.xml

bacula-client.xml kpasswd.xml rsyncd.xml

bacula.xml kprop.xml rtsp.xml

bgp.xml kshell.xml salt-master.xml

bitcoin-rpc.xml ldaps.xml samba-client.xml

bitcoin-testnet-rpc.xml ldap.xml samba-dc.xml

bitcoin-testnet.xml libvirt-tls.xml samba.xml

bitcoin.xml libvirt.xml sane.xml

ceph-mon.xml lightning-network.xml sips.xml

ceph.xml llmnr.xml sip.xml

cfengine.xml managesieve.xml slp.xml

cockpit.xml matrix.xml smtp-submission.xml

condor-collector.xml mdns.xml smtps.xml

ctdb.xml minidlna.xml smtp.xml

dhcpv6-client.xml mongodb.xml snmptrap.xml

dhcpv6.xml mosh.xml snmp.xml

dhcp.xml mountd.xml spideroak-lansync.xml

distcc.xml mqtt-tls.xml squid.xml

dns.xml mqtt.xml ssh.xml

docker-registry.xml mssql.xml steam-streaming.xml

docker-swarm.xml ms-wbt.xml svdrp.xml

dropbox-lansync.xml murmur.xml svn.xml

elasticsearch.xml mysql.xml syncthing-gui.xml

etcd-client.xml nfs3.xml syncthing.xml

etcd-server.xml nfs.xml synergy.xml

finger.xml nmea-0183.xml syslog-tls.xml

freeipa-ldaps.xml nrpe.xml syslog.xml

freeipa-ldap.xml ntp.xml telnet.xml

freeipa-replication.xml nut.xml tftp-client.xml

freeipa-trust.xml openvpn.xml tftp.xml

ftp.xml ovirt-imageio.xml tinc.xml

ganglia-client.xml ovirt-storageconsole.xml tor-socks.xml

ganglia-master.xml ovirt-vmconsole.xml transmission-client.xml

git.xml plex.xml upnp-client.xml

gre.xml pmcd.xml vdsm.xml

high-availability.xml pmproxy.xml vnc-server.xml

https.xml pmwebapis.xml wbem-https.xml

http.xml pmwebapi.xml wbem-http.xml

imaps.xml pop3s.xml wsmans.xml

imap.xml pop3.xml wsman.xml

ipp-client.xml postgresql.xml xdmcp.xml

ipp.xml privoxy.xml xmpp-bosh.xml

ipsec.xml proxy-dhcp.xml xmpp-client.xml

ircs.xml ptp.xml xmpp-local.xml

irc.xml pulseaudio.xml xmpp-server.xml

iscsi-target.xml puppetmaster.xml zabbix-agent.xml

isns.xml quassel.xml zabbix-server.xm

#### 创建服务

想要创建属于自己的服务，需要在下面的目录自定义它。

例如我想创建一个rtmp的服务端口号为1915. 首先我们我们任选一个服务复制过来改名为rtmp.xml

# cd /etc/firewalld/services/

# cp ssh.xml rtmp.xml

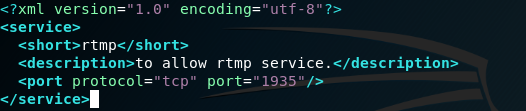
# ls –l /etc/firewalld/services/ rtmp.xml

总用量 4

-rw-r--r-- 1 root root 463 9月 19 13:56 rtmp.xml

然后我们打开创建的ssh.xml 修改以下内容 wq保存退出

# vim /etc/firewalld/services/rtmp.xml



保存退出后,重启 firewalld 服务或者重新加载设置，以激活这些设置，查看是否有的服务

# firewall-cmd --reload

success

# firewall-cmd --get-services |grep rtmp

RH-Satellite-6 amanda-client amanda-k5-client amqp amqps apcupsd bacula bacula-client bgp bitcoin bitcoin-rpc bitcoin-testnet bitcoin-testnet-rpc ceph ceph-mon cfengine cockpit condor-collector ctdb dhcp dhcpv6 dhcpv6-client distcc dns docker-registry docker-swarm dropbox-lansync elasticsearch etcd-client etcd-server finger freeipa-ldap freeipa-ldaps freeipa-replication freeipa-trust ftp ganglia-client ganglia-master git gre high-availability http https imap imaps ipp ipp-client ipsec irc ircs iscsi-target isns jenkins kadmin kerberos kibana klogin kpasswd kprop kshell ldap ldaps libvirt libvirt-tls lightning-network llmnr managesieve matrix mdns minidlna mongodb mosh mountd mqtt mqtt-tls ms-wbt mssql murmur mysql nfs nfs3 nmea-0183 nrpe ntp nut openvpn ovirt-imageio ovirt-storageconsole ovirt-vmconsole plex pmcd pmproxy pmwebapi pmwebapis pop3 pop3s postgresql privoxy proxy-dhcp ptp pulseaudio puppetmaster quassel radius redis rpc-bind rsh rsyncd rtmp rtsp salt-master samba samba-client samba-dc sane sip sips slp smtp smtp-submission smtps snmp snmptrap spideroak-lansync squid ssh steam-streaming svdrp svn syncthing syncthing-gui synergy syslog syslog-tls telnet tftp tftp-client tinc tor-socks transmission-client upnp-client vdsm vnc-server wbem-http wbem-https wsman wsmans xdmcp xmpp-bosh xmpp-client xmpp-local xmpp-server zabbix-agent zabbix-server

#### 获取所有支持icmp的类型(对某种协议字段的支持)

# firewall-cmd --get-icmptypes

address-unreachable bad-header beyond-scope communication-prohibited destination-unreachable echo-reply echo-request failed-policy fragmentation-needed host-precedence-violation host-prohibited host-redirect host-unknown host-unreachable ip-header-bad neighbour-advertisement neighbour-solicitation network-prohibited network-redirect network-unknown network-unreachable no-route packet-too-big parameter-problem port-unreachable precedence-cutoff protocol-unreachable redirect reject-route required-option-missing router-advertisement router-solicitation source-quench source-route-failed time-exceeded timestamp-reply timestamp-request tos-host-redirect tos-host-unreachable tos-network-redirect tos-network-unreachable ttl-zero-during-reassembly ttl-zero-during-transit unknown-header-type unknown-option

#### 列出全部启用的区域的特性（即查询当前防火墙策略）

解释：特性可以是定义的防火墙策略，如：服务、端口和协议的组合、端口/数据报转发、伪装、ICMP 拦截或自定义规则等。

# firewall-cmd --list-all-zones

block

target: %%REJECT%%

icmp-block-inversion: no

interfaces:

sources:

services:

ports:

protocols:

masquerade: no

forward-ports:

source-ports:

icmp-blocks:

rich rules:

dmz

target: default

icmp-block-inversion: no

interfaces:

sources:

services: ssh

ports:

protocols:

masquerade: no

forward-ports:

source-ports:

icmp-blocks:

rich rules:

drop

target: DROP

icmp-block-inversion: no

interfaces:

sources:

services:

ports:

protocols:

masquerade: no

forward-ports:

source-ports:

icmp-blocks:

rich rules:

external

target: default

icmp-block-inversion: no

interfaces:

sources:

services: ssh

ports:

protocols:

masquerade: yes

forward-ports:

source-ports:

icmp-blocks:

rich rules:

home

target: default

icmp-block-inversion: no

interfaces:

sources:

services: ssh mdns samba-client dhcpv6-client

ports:

protocols:

masquerade: no

forward-ports:

source-ports:

icmp-blocks:

rich rules:

internal

target: default

icmp-block-inversion: no

interfaces:

sources:

services: ssh mdns samba-client dhcpv6-client

ports:

protocols:

masquerade: no

forward-ports:

source-ports:

icmp-blocks:

rich rules:

public (active)

target: default

icmp-block-inversion: no

interfaces: eth0 eth1

sources:

services: ssh dhcpv6-client

ports:

protocols:

masquerade: no

forward-ports:

source-ports:

icmp-blocks:

rich rules:

trusted

target: ACCEPT

icmp-block-inversion: no

interfaces:

sources:

services:

ports:

protocols:

masquerade: no

forward-ports:

source-ports:

icmp-blocks:

rich rules:

work

target: default

icmp-block-inversion: no

interfaces:

sources:

services: ssh dhcpv6-client

ports:

protocols:

masquerade: no

forward-ports:

source-ports:

icmp-blocks:

rich rules:

#### 输出区域全部启用的特性。如果省略区域，将显示默认区域的信息

# firewall-cmd --list-all

public (active)

target: default

icmp-block-inversion: no

interfaces: eth0 eth1

sources:

services: ssh dhcpv6-client

ports:

protocols:

masquerade: no

forward-ports:

source-ports:

icmp-blocks:

rich rules:

#### 输出指定区域启动的特性

# firewall-cmd --zone=work --list-all

work

target: default

icmp-block-inversion: no

interfaces:

sources:

services: ssh dhcpv6-client

ports:

protocols:

masquerade: no

forward-ports:

source-ports:

icmp-blocks:

rich rules:

#### 查看默认区域

# firewall-cmd --get-default-zone

public

#### 获取活动的区域

# firewall-cmd --get-active-zones

public

interfaces: eth0 eth1

#### 设置默认区域

# firewall-cmd --set-default-zone=public

Warning: ZONE\_ALREADY\_SET: public

success

#### 根据接口获取区域

即需要查看哪个区域和这个接口绑定即查看某个接口是属于哪个zone的：

firewall-cmd --get-zone-of-interface=接口名

# firewall-cmd --get-zone-of-interface=eth0

public

这条命令将输出接口所属的区域名称。

#### 将接口（网卡）增加到区域

firewall-cmd [--zone=] --add-interface=接口名

# firewall-cmd --add-interface=eth0 //将网卡增加到默认区域

Warning: ZONE\_ALREADY\_SET: 'eth0' already bound to ''

success

# firewall-cmd --get-zone-of-interface=eth0

Public

# firewall-cmd --zone=trusted --add-interface=eth0 //将网卡增加到指定区域

Error: ZONE\_CONFLICT: 'eth0' already bound to a zone //报错，需要先将eth0从别的区域称除，也可以直接使用下面的命令。

如果接口不属于任何区域，接口将被增加到区域。如果区域被省略了，将使用默认区域。接口在重新加载后将重新应用。

#### 修改接口所属区域

firewall-cmd [--zone=] --change-interface=接口名

# firewall-cmd --zone=trusted --change-interface=eth0

Success

# firewall-cmd --get-zone-of-interface=eth0

trusted

这个选项与 --add-interface 选项相似，但是当接口已经存在于另一个区域的时候，该接口将被添加到新的区域。

#### 从区域中删除一个接口

firewall-cmd [--zone=] --remove-interface=接口名

# firewall-cmd --zone=trusted --remove-interface=eth0

success

注：如果某个接口不属于任何 Zone，那么这个接口的所有数据包使用默认的 Zone 的规则

#### 查询区域中是否包含某接口

firewall-cmd [--zone=] --query-interface=接口名

# firewall-cmd --zone=public --query-interface=eth0

Yes

# firewall-cmd --zone=public --query-interface=eth0

yes

如果区域被省略了，将使用默认区域

#### 列举区域中启用的服务

firewall-cmd [ --zone= ] --list-services

# firewall-cmd --zone=work --list-services

ssh dhcpv6-client

# firewall-cmd --list-services

ssh dhcpv6-client

查看 work 区域中启用服务，如果区域被省略了，将使用默认区域。

#### 启用/关闭应急模式

启用应急模式将阻断所有网络连接，以防出现紧急状况。一般用于临时紧急事故处理，如发现网络攻击，但不确定攻击源时，可以启用应急模式，相当于所拔掉所有网线：）

# firewall-cmd --panic-on

success

# firewall-cmd --panic-off

success

# firewall-cmd --query-panic

No

//此命令返回应急模式的状态，如果没有输出。可以使用以下方式获得状态输出：

# firewall-cmd --query-panic && echo "On" || echo "Off"

#### 临时关闭防火墙命令。

重启电脑后，防火墙自动起来。如果网络不通怀疑是防火墙的问题时，可临时关闭防火墙试试！

# systemctl stop firewalld //临时关闭防火墙，重启后防火墙失效

# systemctl start firewalld //关了记得开啊！！重启后防火墙失效

# systemctl disable firewalld //永久关闭防火墙，重启后依然保持关闭状态

# systemctl enable firewalld //永久打开防火墙，重启后依然保持开启状态

其他相关的配置项可以查看 firewall-cmd 的手册页：

# man firewall-cmd

### 处理运行时区域

运行时模式下对区域进行的修改不是永久有效的。重新加载或者重启后修改将失效。

启用区域中的一种服务即给某个区域开启某个服务

firewall-cmd [--zone=区域] --addservice=服务 [--timeout=秒数]

此操作启用区域中的一种服务。如果未指定区域，将使用默认区域。如果设定了超时时间，服务将只启用特定秒数。

//使区域中的public区域中的http服务生效 60 秒

# firewall-cmd --zone=public --add-service=http --timeout=60

success

# firewall-cmd --list-all

public (active)

target: default

icmp-block-inversion: no

interfaces: eth0

sources:

services: ssh dhcpv6-client http

ports:

protocols:

masquerade: no

forward-ports:

source-ports:

icmp-blocks:

rich rules:

等待60秒后，查看public区域中的服务

# firewall-cmd --list-all

public (active)

target: default

icmp-block-inversion: no

interfaces: eth0

sources:

services: ssh dhcpv6-client

ports:

protocols:

masquerade: no

forward-ports:

source-ports:

icmp-blocks:

rich rules:

发现http协议已经不见了。

#### 禁用区域中的某种服务即关闭某个服务

firewall-cmd [--zone=区域] --remove-service=服务

#### 禁用区域中的某种服务

如果未指定区域，将使用默认区域。例:禁止默认区域中的 http 服务:

# firewall-cmd --remove-service=ssh

success

# firewall-cmd --list-all

public (active)

target: default

icmp-block-inversion: no

interfaces: eth0

sources:

services: dhcpv6-client

ports:

protocols:

masquerade: no

forward-ports:

source-ports:

icmp-blocks:

rich rules:

#### 启用区域中的某种服务

如果未指定区域，将使用默认区域。例:禁止默认区域中的 http 服务:

# firewall-cmd --add-service=ssh

success

# firewall-cmd --list-all

public (active)

target: default

icmp-block-inversion: no

interfaces: eth0

sources:

services: dhcpv6-client ssh

ports:

protocols:

masquerade: no

forward-ports:

source-ports:

icmp-blocks:

rich rules:

#### 查询区域中是否启用了特定服务

firewall-cmd [--zone=区域] --query-service=服务

# firewall-cmd --zone=public --query-service=ssh

yes

Yes 表示服务启用，no 表示服务关掉了。

#### 启用区域端口和协议组合

firewall-cmd [--zone=区域] --add-port=portid[-portid]/protocol [--timeout=seconds]

此操作将启用端口和协议的组合。端口可以是一个单独的端口或者是一个端口范围。协议可以是 tcp 或 udp。

# firewall-cmd --zone=public --add-port=8080/tcp

success

# firewall-cmd --zone=public --add-port=25/tcp

success

# firewall-cmd --zone=public --list-all

public (active)

target: default

icmp-block-inversion: no

interfaces: eth0

sources:

services: dhcpv6-client ssh

ports: 8080/tcp 25/tcp

protocols:

masquerade: no

forward-ports:

source-ports:

icmp-blocks:

rich rules:

#### 禁用端口和协议组合

firewall-cmd [--zone=区域] --remove-port=portid[-portid]/protocol

# firewall-cmd --zone=public --remove-port=8080/tcp

success

# firewall-cmd --zone=public --remove-port=25/tcp

success

# firewall-cmd --zone=public --list-all

public (active)

target: default

icmp-block-inversion: no

interfaces: eth0

sources:

services: dhcpv6-client ssh

ports:

protocols:

masquerade: no

forward-ports:

source-ports:

icmp-blocks:

rich rules:

#### 查询区域中是否启用了端口和协议组合

firewall-cmd [--zone=区域] --query-port=portid[-portid]/protocol

# firewall-cmd --zone=public --add-port=22/tcp

success

# firewall-cmd --zone=public --query-port=22/tcp

Yes

//注意：必须协议和端口同时开启，查询状态才为“yes”

#### 启用区域中的 IP 伪装功能

firewall-cmd [--zone=区域] --add-masquerade

此操作启用区域的伪装功能。私有网络的地址将被隐藏并映射到一个公有 IP。这是地址转换的一种形式，常用于路由。由于内核的限制，伪装功能仅可用于 IPv4。

#### 禁用区域中的 IP 伪装

firewall-cmd [--zone=区域] --remove-masquerade

#### 查询区域的伪装状态

firewall-cmd [--zone=区域] --query-masquerade

注意：启用伪装功能的主机同时也需要开启转发服务：

# echo 1 > /proc/sys/net/ipv4/ip\_forward

或

# vi /etc/sysctl.conf 添加如下内容

net.ipv4.ip\_forward = 1

保存退出并执行#sysctl -p 使修改生效

#### 启用区域的 ICMP 阻塞功能

firewall-cmd [--zone=区域] --add-icmp-block=icmp 类型

此操作将启用选中的 Internet 控制报文协议（ICMP）报文进行阻塞。 ICMP 报文可以是请求信息或者创建的应答报文，以及错误应答。

#### 禁止区域的 ICMP 阻塞功能

firewall-cmd [--zone=区域] --remove-icmp-block=icmp 类型

#### 查询区域的 ICMP 阻塞功能

firewall-cmd [--zone=区域] --query-icmp-block=icmp 类型

#### 在区域中启用端口转发或映射

firewall-cmd [--zone= 区 域 ] --add-forward-port=port=portid[-portid]:proto=protocol[ :toport=portid[-portid]][ :toaddres s=address [/mask]]

端口可以映射到另一台主机的同一端口，也可以是同一主机或另一主机的不同端口。端口号可以是一个单独的端口或者是端口范围 - 。协议可以为 tcp 或 udp。目标端口可以是端口号或者是端口范围 - 。目标地址可以是 IPv4 地址。受内核限制，端口转发功能。仅可用于 IPv4。

# firewall-cmd --zone=external --add-forward-port=port=22:proto=tcp:toaddress=211.106.65.50

//意思是凡是来从 external 进来的 22 端口的数据包全部转发到 211.106.65.50

#### 禁止区域的端口转发或者端口映射

firewall-cmd [--zone=] --remove-forward-port=port=portid[-portid]:proto=protocol[ :toport=portid[-portid]][ :toad dress=address [/mask]]

#### 查询区域的端口转发或者端口映射

firewall-cmd [--zone=]--query-forward-port=port=portid[-portid]:proto=protocol[ :toport=portid[-portid]] [ :toaddress=address [/mask]]

### 处理永久区域

永久选项不直接影响运行时的状态。这些选项仅在重载或者重启服务后才生效。为了使用运行时和永久设置，需要分别设置两者。选项--permanent 需要是永久设置的第一个参数。

获取永久选项所支持的服务

# firewall-cmd --permanent --get-services

#### 获取永久选项所支持的 ICMP 类型列表

# firewall-cmd --permanent --get-icmptypes

#### 获取支持的永久区域

# firewall-cmd --permanent --get-zones

#### 配置防火墙在 public 区域打开 http 协议，并保存，以致重启有效

# firewall-cmd --permanent --zone=public --add-service=http

查看永久模式下 public 区域是否打开 http 服务。

# firewall-cmd --permanent --zone=public --query-service=http

#### 在 public 区域开放 8080 端口

# firewall-cmd --permanent --zone=public --add-port=8080/tcp

### 命令行配置富规则

#### 查看富规则：

# firewall-cmd --list-rich-rules

#### 创建富规则：

#### 删除 rich 规则

## 防火墙配置实例

### 富规则中使用伪装功能可以更精确详细的限制：

### 仅允许部分 IP 访问本机服务配置

### 禁止远程 IP 访问 ssh

#### 仅允许部分 IP 访问本机端口配置

#### 创建 rich 规则，可以指定日志的前缀和输出级别

### 端口转发

# Linux内核基础

## Linux工作机制

内核的主要作用：进程管理、内存管理、I/O管理、文件系统、驱动程序、安全相关。

### 内核设计的两大流派：

微内核：内核很小，其它功能（如内存管理等）都做成外部子系统。典型的有windows\Solaris。

单内核：所有必须的功能全部做在一起，较大，都是直接通信，效率较高、耦合度高。典型的有Linux。

### 内核的层次关系



### 内核及内核模块

Linux吸收了微内核的理念，采取了模块化设计，虽然所有核心功能都在一起，但可以根据需要来编译裁剪，这就是内核定制。模块的后缀为.ko(kernel Object内核对象)，实际上就函数库，从名称就可以看出，模块只能被内核调用，用户空间的程序不能直接调用。内核本身不提供任何业务功能，只是提供了系统资源（硬件）的驱动和管理程序，并对外提供了一些程序（函数）接口，供用户选择使用，以便用户不用关心硬件驱动的问题，这些程序接口称为System Call(系统调用)。

### System Call(系统调用)

System Call(系统调用)是Linux内核提供的一个函数库，用于与用户与计算机硬件交互。因为要考虑内核的大小和通用等问题，所以这些函数都是最基本的功能，功能非常简陋，这些功能都不能直接使用，需要用户自己来将这些简陋的功能进行组合，实现自己需要的功能，就像现实生活中面粉、馒头和面条的关系一样，Linux的内核调用只提供面粉，用户需要吃什么需要自己动手做。

### 标准库（glibc）

因为馒头、面条都是非常通用的食品，所以出现了专门做馒头的面粉的人，与之类似，对用常用的系统调用组合，Linux引入了“库”的概念，库工作在用户空间。库虽然运行在用户空间，但它已经集成到内核中，也是内核的组成部分。库提供最常用的功能，典型的代表就是标准库（glibc）。比如，馒头、面条等常用功能由标准库（glibc）来提供，对于杂粮这种不通用的食品，还是需要用户自己调用System Call(系统调用)来实现。标准库（glibc）并不屏蔽System Call(系统调用)。

总结：

因为内核要被众多应用程序提供服务，所以内核必须工作于独立的内存空间且位置不能改变，一旦内核的工作位置改变，其它程序将无法找到内核，程序将无法运行；

在多用户环境下，大家都在用一个内核，如果用户能够对内核进行修改，那么必将影响其它用户的正常使用，所以内核工作的地址空间需要被严密保护起来，用户需要建立自己的用户空间将内核参数写入自己的内存空间，然后调用自己的程序进行处理，这样就不会影响其它用户，这就形成了内核空间和用户空间的概念

因为内核先被加载，所以处于内存的低地址位，通常内核空间是内核工作时在内存中所处的一段地址空间（通常为1G大小）；

内核空间不能被分页，因为内存分页功能是内核加载后由内核给外部提供的功能，内核在没有加载入内存之前根本没有运行，自然就不存在内存分页，内核不可能为自己实现分页功能。

## 系统服务

服务实际上就是一些程序，这些程序默认就专门提供一些功能，比如网页服务、SSH服务、邮件服务等。。。。这些服务往往由Linux操作系统提供，为了区别于普通的应用程序，Linux称这些服务也称为守护进程(daemon)。

守护进程(daemon)是一类在后台运行的特殊进程，用于执行特定的系统任务。很多守护进程在系统引导的时候自动启动，并且一直运行直到系统关闭。另一些只在需要的时候才启动，完成任务后就自动结束。

### 守护进程的分类

独立运行的守护进程

独立运行的守护进程是候linux自动启动的进程，由init脚本负责管理。系统服务都是独立运行的守护进程（如syslogd 和 cron 等）。服务器对特定的端口进行监听，等待[客户端](https://baike.baidu.com/item/%E5%AE%A2%E6%88%B7%E7%AB%AF)的连接。如果客户端产生一个连接请求，守护进程就创建一个子服务器响应这个连接，而主服务器继续监听，并维护多个子服务池等待下一个客户端请求。

#### 超级守护进程（xinetd）和瞬时（短暂）守护进程

从守护进程的工作原理可以看出，系统所运行的每一种服务，都必须监听某个端口，这通常意味着资源浪费。为了解决这个问题，Linux采用了“超级守护进程（xinetd）”的技术，用于管理瞬时（短暂）进程。Xinetd进程能够同时监听多个指定的端口，在接受用户请求时，它能够根据用户请求的端口不同，启动不同的网络服务进程来处理这些用户请求。可以把xinetd看作一个管理启动服务的管理器，它决定把一个客户请求交给那个程序处理，然后启动相应的守护程序。xinetd进程也叫“超级守护进程”，管理瞬时（短暂）守护进程。

### 服务优先级注意事项

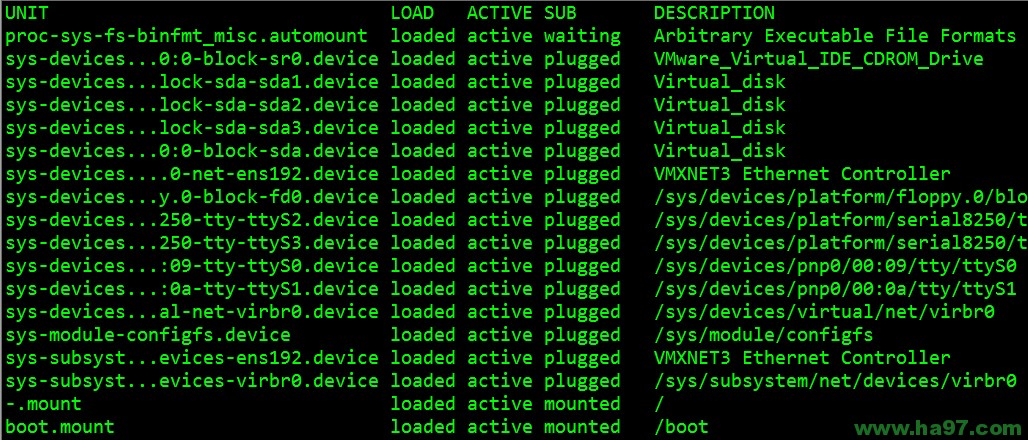
假如X依赖于Y，启动时Y的优先级要高于X，关闭时X的优先级要高于Y。

### 服务管理命令—systemctl

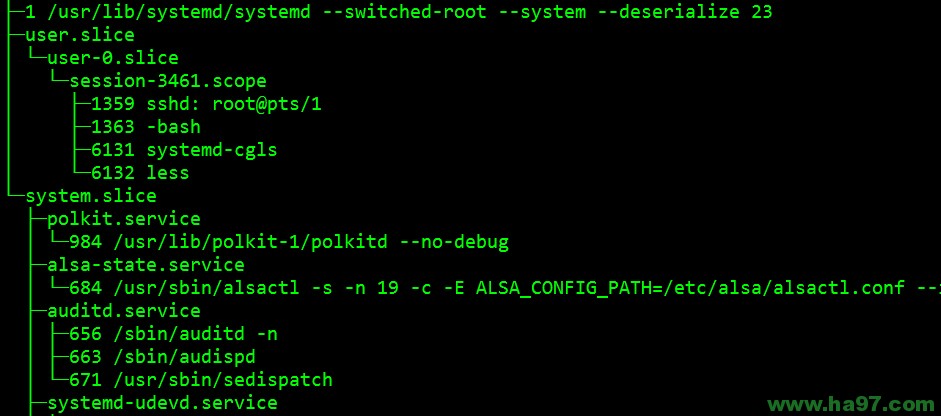
systemctl是主要的工具，它融合之前service和chkconfig的功能于一体。可以使用它永久性或只在当前会话中启用/禁用服务。

* 查看服务状态

systemctl可以列出正在运行的服务状态，如图：

[](http://www.ha97.com/wp-content/uploads/2014/07/systemctl.jpg)

systemd-cgls以树形列出正在运行的进程，它可以递归显示控制组内容。如图：

[](http://www.ha97.com/wp-content/uploads/2014/07/systemd-cgls.jpg)

* 启动/关闭、启用/禁用服务

启动一个服务：systemctl start postfix.service

关闭一个服务：systemctl stop postfix.service

重启一个服务：systemctl restart postfix.service

显示一个服务的状态：systemctl status postfix.service

在开机时启用一个服务：systemctl enable postfix.service

在开机时禁用一个服务：systemctl disable postfix.service

查看服务是否开机启动：systemctl is-enabled postfix.service;echo $?

查看已启动的服务列表：systemctl list-unit-files|grep enabled

说明：启用服务就是在当前“runlevel”的配置文件目录/etc/systemd/[system](http://www.ha97.com/tag/system)/multi-user.target.wants/里，建立/usr/lib/systemd/system里面对应服务配置文件的软链接；禁用服务就是删除此软链接。如图：

[](http://www.ha97.com/wp-content/uploads/2014/07/disable-postfix.service.jpg)

查看了/usr/lib/systemd/system 里的文件，语法跟旧版/etc/init.d/里的系统服务脚本完全不同了。

## 模块

### 内核模块概述

Linux 内核的整体结构非常庞大，其包含的组件也非常多。我们怎样把需要的部分都包含在内核中呢？一种方法是把所有需要的功能都编译到 Linux 内核。这会导致两个问题，一是生成的内核会很大，二是如果我们要在现有的内核中新增或删除功能，将不得不重新编译内核。

有没有一种机制使得编译出的内核本身并不需要包含所有功能，而在这些功能需要被使用的时候，其对应的代码可被动态地加载到内核中呢？

Linux 提供了这样的一种机制，这种机制被称为**模块（Module）**，可以实现以上效果。模块具有以下特点。

1) 模块本身不被编译入内核映像，从而控制了内核的大小。

2) 模块一旦被加载，它就和内核中的其他部分完全一样。

### 认识模块

我们在配置内核的时候，在配置菜单看到有一种选项可以选择为<M> <\*> < > 。其中:

* **<M>**表示编译成模块，即modules，这个便于选项编译的代码不编译进内核zImage，而是编译成一个单独的文件，通常为后缀.ko（2.6以上内核版本是.ko，2.6内核之前是.o文件，那么我们可以像软件一样选择安装和卸载这些.ko模块文件。
* **<\*>**表示编译进内核，就是将这一段模块代码编译到了zImage镜像文件去了，在内核启动的时候自动安装执行我们的模块代码，这样做类似于一些安装系统的时候自带的驱动，在安装系统的时候就已经安装好了。
* **< >**表示没有选择，不做任何事情。

所以，**我们所说的内核模块就是linux下的那些.ko文件**。

### [Linux模块的加载和卸载](https://www.cnblogs.com/jjzd/p/6438641.html)

Linux操作系统中模块操作相关命令解释lsmod 查看已经安装好的模块， 也可以查看/proc/modules文件的内容.实际上,lsmod读命令就是通过查看/proc/modules的内容来显示模块信息的.

#### 显示所有已经安装的模块

# root@kali2018:~# lsmod

Module Size Used by

nls\_utf8 16384 1

isofs 49152 1

udf 102400 0

crc\_itu\_t 16384 1 udf

iptable\_filter 16384 1

vmw\_vsock\_vmci\_transport 32768 4

………..

#### 查看某个模块是否安装

# lsmod |grep 'e1000'

e1000 155648 0………..

#### modinfo 显示模块信息

# modinfo e1000

filename: /lib/modules/4.17.0-kali1-amd64/kernel/drivers/net/ethernet/intel/e1000/e1000.ko

version: 7.3.21-k8-NAPI

license: GPL

description: Intel(R) PRO/1000 Network Driver

author: Intel Corporation, <linux.nics@intel.com>

srcversion: EE2DEB2A823235A0461A0C8

alias: pci:v00008086d00002E6Esv\*sd\*bc\*sc\*i\*

alias: pci:v00008086d000010B5sv\*sd\*bc\*sc\*i\*

alias: pci:v00008086d00001099sv\*sd\*bc\*sc\*i\*

alias: pci:v00008086d0000108Asv\*sd\*bc\*sc\*i\*

alias: pci:v00008086d0000107Csv\*sd\*bc\*sc\*i\*

alias: pci:v00008086d0000107Bsv\*sd\*bc\*sc\*i\*

alias: pci:v00008086d0000107Asv\*sd\*bc\*sc\*i\*

。。。。。。。。。

depends:

retpoline: Y

intree: Y

name: e1000

vermagic: 4.17.0-kali1-amd64 SMP mo

#### insmod 加载模块

需要指定完整的路径和模块名字 模块依赖及路径信息，这样子才可以成功加载需要模块。查看模块依赖关系可用modinfo查看

# insmod /lib/modules/2.6.18-398.el5/kernel/drivers/net/pcnet32.ko

#### rmmod 卸载模块

但是内核会认为卸载模块不安全，可以添加命令强制卸载。

# rmmod pcnet32.ko

#### Modprobe综合管理命令

不需要指定路径，它会到默认路径下寻找模块。模块也存在依赖性问题:比如你要加载msdos.ko,需要先加载fat.ko.modprobe查看/lib/module/`uname -r`/modules.dep得知模块的依赖关系

modprobe [-cfr] module\_name

-c :列出目前系统所有的模块

-f:强制加载该模块

-r:类似rmmod 就是删除某个模块

例子

#modprobe cifs

#modprobe -r cifs

#### depmod 检查系统中模块之间的依赖关系

依赖关系信息存于/lib/modules/2.6.18-1.2798/modules.dep中。一般加载模块后，要命令行下先执行 depmod -a后，在系统加载模块时候会查看modules.dep中。

# 输入/输出重定向和管道

学kali linux的目标当然是为了掌握黑客技术和安全技术，如果一个黑客不懂编程,那他就不是一名真正的黑客，充其量也就是一个“脚本小子”（“脚本小子”是指那些只会利用别人的写好的工具进行渗透工作的人）。

注意：对于第七章整章和第八章的内容必须掌握，要不然，看到一个shell脚本就会“懵”了，多看多练。有了基础，再进行复杂的shell脚本的学习，就会事半功倍。把这两章的内容自己再多敲几遍，先掌握这两章的内容，学习是一个循序渐进的过程，先掌握最重点的使用方法，再系统的学习，深纠原理！

在linux系统中，bash的功能非常的丰富与强大。我们可以在bash中编程，可以做到：

1. 编写循环、判断、将结果写到指定文件，形成智能化运维；
2. 可以查询、更改所有的系统设置，实现自动化运维；
3. 最重要的是可以对kali内置的600多款工具进行重新组织，形成自己的黑客工具箱；
4. 可以写批命令、定时计划。甚至可以写游戏、玩游；

。。。。。。

而重定向和管道是bash的基础，如果重定向和管道不熟悉，学习bash几乎是个笑话！！！

## 文件描述符

在学习重定向和管道以前，必须要对文件进行一个初步的了解，在LINUX是遵循“一切都是文件”的理念。存储设备有对应的块设备文件、输出有对应的字符文件、管道有对应的管道文件。。。。。。总之，Linuxu将几乎所有（包含设备）操作都置换成了对文件的操作。

对于内核而言，所有打开的文件都通过文件描述符访问。文件描述符是一个非负整数。

* 当打开或创建一个文件时，内核分配一个文件描述符，返回给进程。从编程的角度来讲，当打开、创建一个文件时，使用open或creat函数生成文件描述符，作为返回值以标识该文件；
* 当读或写一个文件时，内核将该文件的文件描述符返回给进程。从编程的角度来讲，当读、写一个文件时，文件描述符作为参数传递给read或write函数。

按照UNIX系统shell使用惯例：

使用文件描述符0与进程的标准输入相关联；

文件描述符1与标准输出相关联；

文件描述符2与标准出错相关联。

这是各种shell以及很多应用程序使用的惯例，而与UNIX内核无关，如果不遵守这种惯例，那么很多UNIX系统应用程序就不能正常工作。在依从POSIX的应用程序中，0、1、2应当替换成符号常量：

STDIN\_FILENO 0

STDOUT\_FILENO 1

STDERR\_FILENO 3

这些常量都定义在头文件<unistd.h>中。

文件描述符的变化范围0~OPEN\_MAX。早期的UNIX系统实现采用的上限值是19（允许每个进程最多打开20个文件），但现在很多系统则将其增至1024个：

# ulimit -a

core file size (blocks, -c) 0

data seg size (kbytes, -d) unlimited

scheduling priority (-e) 0

file size (blocks, -f) unlimited

pending signals (-i) 7745

max locked memory (kbytes, -l) 16384

max memory size (kbytes, -m) unlimited

open files (-n) 1024

pipe size (512 bytes, -p) 8

POSIX message queues (bytes, -q) 819200

real-time priority (-r) 0

stack size (kbytes, -s) 8192

cpu time (seconds, -t) unlimited

max user processes (-u) 7745

virtual memory (kbytes, -v) unlimited

file locks (-x) unlimited

对于一般情况下，这个数量完全够用，但对于大型服务来说，这个数字明显不够用，比如一个大型网站网页就有几千个，加上同时在线人过万想想会出现什么情况？这就涉及系统调优相关的知识，这里不做介绍。

小结：

程序在运行时会在内存中产生进程，进程会根据需要打开多个文件，这些文件都有自己独一无二的标识，这个标识就叫文件标识符（**fd**:file descriptors），进程通过文件标识符来访问文件。文件第一次打开时，系统会自动分配一个fd,文件访问完毕后，关闭文件时，相应的fd被释放回收。

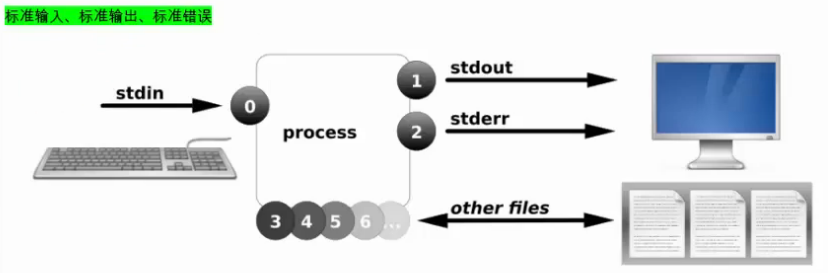
前面介绍过了，有3个文件标识符是被系统占用的，他们名称固定、作用固定，以供所有用户共同使用，它们就文件标识符0、1、2，分别代表标准输入、标准输出、标准错误：

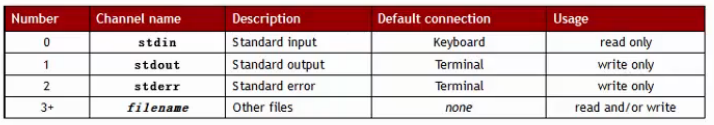
标准输入：文件描述符为0，一舟情况下为键盘和。

标准输出：文件描述符为1，一舟情况下为显示器。

标准错误：文件描述符为2，一舟情况下，错误也是输出到屏幕。

一般情况下，每个 Unix/Linux 命令运行时都会打开三个文件。





## 重定向简介

大多数 UNIX 系统命令从你的终端接受输入并将所产生的输出发送回终端。一个命令通常从一个叫标准输入的地方读取输入，默就是你的终端。同样，一个命令通常将其输出写入到标准输出，默认也是你的终端。

我们可以使用重定向命令，改变输入输出的位置和类型，例如：

# date

2018年 09月 28日 星期五 10:15:45 CST //结果正常输出在屏幕上

# date 1> date.txt //加上重定向后，结果正常输出到时date.txt文件中去了。

# date 2> date.txt //重定标准错误后，结果还是正常输屏幕，这是因为没有错误。

2018年 09月 28日 星期五 10:16:52 CST

# daaate 2> date.txt //人为制造一点错误，输出到指定的data.txt文件中去了。

#

0>或0>>表示：将输入信息重定向或追加到指定位置。

1>或2>>表示：将输出信息重定向或追加到指定位置。

2>或2>>表示：将错误信息重定向或追加到指定位置。

## 重定向命令

重定向命令列表如下：

command > file: 将输出重定向到 file。

command < file: 将输入重定向到 file。

command >> file: 将输出以追加的方式重定向到 file。

n > file: 将文件描述符为 n 的文件重定向到 file。

n >> file: 将文件描述符为 n 的文件以追加的方式重定向到 file。

n >& m: 将输出文件 m 和 n 合并。

n <& m: 将输入文件 m 和 n 合并。

<< tag: 将开始标记 tag 和结束标记 tag 之间的内容作为输入。

需要注意的是文件描述符 0 通常是标准输入（STDIN），1 是标准输出（STDOUT），2 是标准错误输出（STDERR）。

### 输出重定向

重定向一般通过在命令间插入特定的符号来实现。特别的，这些符号的语法如下所示:

command1 > file1 或 command1 1> file1

上面这个命令执行command1然后将输出的内容存入file1。注意：任何file1内的已经存在的内容将被新内容替代。如果要将新内容添加在文件末尾，请使用>>操作符。

例如：

# who > users

执行who 命令，它将命令的完整的输出重定向在用户文件中(users)，执行后，并没有在终端输出信息，这是因为输出已被从默认的标准输出设备（终端）重定向到指定的文件。

你可以使用 cat 命令查看文件内容：

# cat users

\_mbsetupuser console Oct 31 17:35

tianqixin console Oct 31 17:35

tianqixin ttys000 Dec 1 11:33

输出重定向会覆盖文件内容，请看下面的例子：

# echo "菜鸟教程：www.runoob.com" > users

# cat users

菜鸟教程：www.runoob.com

如果不希望文件内容被覆盖，可以使用 >> 追加到文件末尾，例如：

# echo "菜鸟教程：www.runoob.com" >> users

# cat users

菜鸟教程：www.runoob.com

菜鸟教程：www.runoob.com

### 输入重定向

和输出重定向一样，Unix 命令也可以从文件获取输入，语法为：

command1 < file1 或 command1 0< file1

这样，本来需要从键盘获取输入的命令会转移到文件读取内容。

注意：输出重定向是大于号(>)，输入重定向是小于号(<)。

实例：

接着以上实例，我们需要统计 users 文件的行数,执行以下命令：

# wc -l users

2 users

也可以将输入重定向到 users 文件：

# wc -l < users //将users文件做为输入，让wc –l统计行数

2

注意：上面两个例子的结果不同：第一个例子，会输出文件名；第二个不会，因为它仅仅知道从标准输入读取内容。

### 错误重定向

默认情况下，command > file 将 stdout 重定向到 file，command < file 将stdin 重定向到 file。实际上，因为系统默认输入为0，默认输出为1,也就是说：

command1 < file1 等价于 command1 0< file1 //默认输入为0，可省略不写。

command1 > file1 等价于 command1 1> file1 //默认输入为0，可省略不写。

因为标准错误并不系统的默认值，如果希望重定向标准错误，必须指明文件标识符为2。比如，如果希望stderr重定向到file，可以这样写：

command 2> file //2表示标准错误.，>表示重定向覆盖

如果希望 stderr 追加到 file 文件末尾，可以这样写：

command 2>> file //2表示标准错误.，>>表示重定向追加到结尾

如果希望将 stdout 和 stderr 合并后重定向到 file，可以这样写：

command > file 2>& 1 //2>&中间不能有空格，写法是固定的

或者

command >> file 2>&1 //&和1之间有没有空格都可以，但2>&中间不能有空格。

如果希望对 stdin 和 stdout 都重定向，可以这样写：

command < file1 > file2

command 命令将 stdin 重定向到 file1，将 stdout 重定向到 file2。

## Here Document

Here Document 是 Shell 中的一种特殊的重定向方式，用来将输入重定向到一个交互式 Shell 脚本或程序。它的基本的形式如下：

command << delimiter //标识开始，标识名称可以自己随便取，但大家约定俗成的使用EOF

document

delimiter

它的作用是将两个 delimiter 之间的内容(document) 作为输入传递给 command。

注意：

* 结尾的delimiter 一定要顶格写，前面不能有任何字符，后面也不能有任何字符，包括空格和 tab 缩进。
* 开始的delimiter前后的空格会被忽略掉。

实例

在命令行中通过 wc -l 命令计算 Here Document 的行数：

# wc -l << EOF

第一行

第二行

第三行

EOF

3 //输出结果为 3 行

我们也可以将 Here Document 用在脚本中，例如：

#!/bin/bash

cat << EOF

第一行

第二行

第三行

EOF

执行以上脚本，输出结果：

第一行

第二行

第三行

## /dev/null 文件

如果希望执行某个命令，但又不希望在屏幕上显示输出结果，那么可以将输出重定向到 /dev/null：

command > /dev/null

/dev/null 是一个特殊的文件，写入到它的内容都会被丢弃；如果尝试从该文件读取内容，那么什么也读不到。但是/dev/null文件非常有用，将命令的输出重定向到它，会起到"禁止输出"的效果。

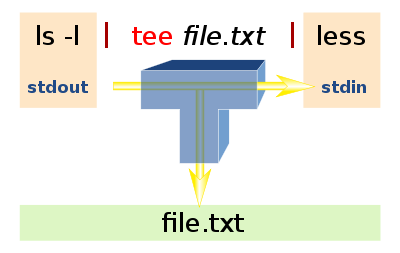
如果希望屏蔽 stdout 和 stderr，可以这样写：

command > /dev/null 2>&1

注意：0 是标准输入（STDIN），1 是标准输出（STDOUT），2 是标准错误输出（STDERR）*。*

## Tee管道

**tee命令**用于将数据重定向到文件，另一方面还可以提供一份重定向数据的副本作为后续命令的stdin。简单的说就是把数据重定向到给定文件和屏幕上。



存在缓存机制，每1024个字节将输出一次。若从管道接收输入数据，应该是缓冲区满，才将数据转存到指定的文件中。若文件内容不到1024个字节，则接收完从标准输入设备读入的数据后，将刷新一次缓冲区，并转存数据到指定文件。

1. 语法

tee(选项)(参数)

1. 选项

-a：向文件中重定向时使用追加模式；

-i：忽略中断（interrupt）信号。

1. 参数

文件：指定输出重定向的文件。

例：在终端打印stdout同时重定向到文件中：

# ls | tee out.txt

//屏幕上输出，同时在out.txt文件中记录

公共

模板

视频

图片

文档

下载

音乐

桌面

# cat out.txt

公共

模板

视频

图片

文档

下载

音乐

桌面

# ls | tee out.txt | cat –n

//屏幕上输出，同时在out.txt文件中记录，最后将加上行号显示，注意，先记录到out.txt文件，然后才加的行号，所以out.txt中并不显示号。

1 out.txt

2 公共

3 模板

4 视频

5 图片

6 文档

7 下载

8 音乐

9 桌面

# cat out.txt

out.txt

公共

模板

视频

图片

文档

下载

音乐

桌面

## 小结

command > file 2>&1

command >> file 2>&1

放在>后面的&，表示重定向的目标不是一个文件，而是一个文件描述符。

换言之2>1代表将stderr重定向到当前路径下文件名为1的文件中，而2>&1代表将stderr重定向到文件描述符为1的文件(即/dev/stdout)中。

而&>file是一种特殊的用法，也可以写成>&file，二者的意思完全相同，都等价于

>file 2>&1

此处&>或者>&视作整体，分开没有单独的含义

**示例分析：**

# find /etc -name .bashrc > list 2>&1

//我想问为什么不能调下顺序,比如这样

# find /etc -name .bashrc 2>&1 > list

因为从左到右有顺序的

第一种

xxx > list 2>&1

先将要输出到stdout的内容重定向到文件，此时文件list就是这个程序的stdout，再将stderr重定向到stdout，也就是文件list

第二种

xxx 2>&1 > list

先将要输出到stderr的内容重定向到stdout，此时会产生一个stdout的拷贝，作为程序的stderr，而程序原本要输出到stdout的内容，依然是对接在stdout原始文件上，因此第二步重定向stdout，对stdout的拷贝不产生任何影响

对于上面 '2>&1'，举个例子，比如说:

# find /etc -names "\*.txt" >list 2>&1

从左往右执行，执行到 >list，此时的 stdout 为 list；而执行到 2>&1，表示 stderr 重定向到 stdout，这里也就是 list 文件。

因为 **[ find /etc -names "\*.txt" ]** 这条命令是错误的( -names 应该是 -name)。本来要输出到终端屏幕的错误信息:

find: unknown predicate `-names`

被重定向到了 stdout 也就是 list 文件中，所以屏幕不会出现错误信息，而是打印到了 list 文件中。

## 管道

前面介绍的重定向的目标主要是把输入输出结果的重新指定一个地方，并且重定向的最终输出对象一般为文件，也无法实现跨进程间的通信，只能在本进程打开文件间进行数据交换。有没办法将输入输出结果传送给另外一个命令（进程）继续进一步处理呢？答案是肯定的，那就是管道！

### ****管道的概念：****

管道是一种最基本的IPC机制，作用于有血缘关系的进程之间，完成数据传递。有如下特质：

1. 其本质是一个伪文件(实为内核缓冲区)
2. 由两个文件描述符引用，一个表示读端，一个表示写端。
3. 规定数据从管道的写端流入管道，从读端流出。

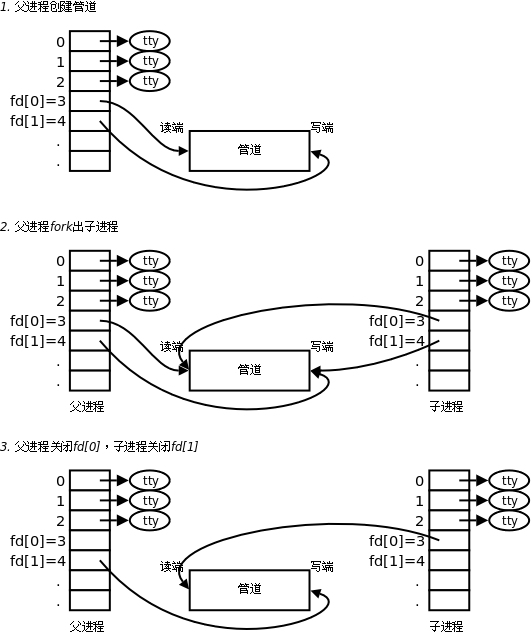
管道的原理:

管道实为内核使用环形队列机制，借助内核缓冲区(4k)实现。

管道的局限性：

1. 数据自己读不能自己写。
2. 数据一旦被读走，便不在管道中存在，不可反复读取。
3. 由于管道采用半双工通信方式。因此，数据只能在一个方向上流动。
4. 只能在有公共祖先的进程间使用管道。

### 管道创建过程



1. 父进程调用pipe函数创建管道，得到两个文件描述符fd[0]、fd[1]指向管道的读端和写端。
2. 父进程调用fork创建子进程，那么子进程也有两个文件描述符指向同一管道。
3. 父进程关闭管道读端，子进程关闭管道写端。父进程可以向管道中写入数据，子进程将管道中的数据读出。由于管道是利用环形队列实现的，数据从写端流入管道，从读端流出，这样就实现了进程间通信。

### 管道的应用实例

管道可以轻易地连接两个毫不相关的程序，把一个程序的结果交给另一个来处理，甚至，不停地交接处理。下面我通过几个例来学习管道的使用方法。

#### 从执行结果中过滤关键字。

# ps –ef |grep ‘python’

* ps是linux中非常强大的进程查看工具，其中-e为显示所有进程，-f为全格式显示。
* grep是一个非常高效的查询工具，可以查询文本中带有某关键字的行。
* “|”是管道符号。

这个命令的功能是，查询带有关键字 python 的进程。会一条一条列出。

单使用ps –ef会列出所有的进程，但是我们不需要看到所有的进程，我们只需要看我们想要的进程有没有存在于后台，所以用管道符，把所有的进程信息传给grep程序，帮我们搜索我们想要的进程信息。简直是神器，比如我们想要查看ssh服务是否开启，就可以运行ps –ef |grep ssh

#### 转接大小写后保存到指定位置

我们将/etc/issue文件中的内容转换为大写后保存至/tmp/issue.out文件中。其中有几个关键点：

* 首先是获取issue文件的内容，很简单，我们用cat /etc/issue命令（Issue文件保存的是系统登录前的提示信息），
* 其次是转换为大写，我们用转换命令tr来实现，tr [a-z] [A-Z]。最后用重定向保存到文件即可。

命令：

# cat /etc/issue |tr [a-z] [A-Z] > /tmp/issue.out

将当前系统登录用户的信息转换为大写后保存至/tmp/who.out文件中

获取当前用户的信息我们用who命令。

# who | tr 'a-z' 'A-Z' > /tmp/who.out

#### 自动邮件发送

自动化运维环境中，当系统遇到突发事件后，系统自动发邮件通知系统管理员，我们同样可以用管道或bash实现这个功能。

例如：

给root发邮件，要求邮件标题为 help ，邮件正文如下：

Hello, I am 用户名,The system version is here,please help me to check it ,thanks!

操作系统版本信息

关键点：

* 用户名要进行替换；
* 操作系统版本信息也要替换；
* 最后还有个回车符。

我们需要用到命令：

echo $USER //输出用户名

uname –a //输出操作系统版本信息

\n //回车符

我们组合一下如何写邮件内容：

# echo -e 'Hello, I am ' $USER ',The system version is here,please help me to check it,thanks!\n' `uname -a`

echo –e会对后面的\n进行解释，不然只会当成字符处理。字符要加单引号，避免出现问题。$USER不能加引号，\n必须在引号里。Uname –a是个命令，所以我们要加反引号。

接下来是发邮件的命令

格式是这样的mail –s “邮件标题” 收件人 内容

我们用管道符来传入内容进去

# echo -e 'Hello, I am '$USER ',The system version is here,please help me to check it,thanks!\n' `uname -a`|mail -s "help" root

#### 100以内数的总和

这是个非常的经典例子。看到这道题，你会想怎么去做？写个循环i++ j++？还是用公式（1+100）\*n/2？

我们的bash可以非常巧妙的解决这个问题。

首先我们要知道linux 中有个计算器，叫bc，我们先来看看他的用法。

1. 直接输入bc进入程序

# bc

bc 1.07.1

Copyright 1991-1994, 1997, 1998, 2000, 2004, 2006, 2008, 2012-2017 Free Software Foundation, Inc.

This is free software with ABSOLUTELY NO WARRANTY.

For details type `warranty'.

6+1

7

quit

1. 扔给bc一条算式

# echo '61+9' | bc

70

看到第二种用法，我们或许就意识到了，bc可以直接计算输入给他的字符串！是不是很有趣？

那我们能不能拿到一个1+2+3+..+99+100的字符串呢？我们知道echo {1..100}可以生成1 2 3 4 5 … 100的字符串，中间以空格隔开。

# echo {1..100}

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

等等..用空格隔开？把空格换成加号不就行了么？我们来：

# echo {1..100} | tr ' ' '+'

1+2+3+4+5+6+7+8+9+10+11+12+13+14+15+16+17+18+19+20+21+22+23+24+25+26+27+28+29+30+31+32+33+34+35+36+37+38+39+40+41+42+43+44+45+46+47+48+49+50+51+52+53+54+55+56+57+58+59+60+61+62+63+64+65+66+67+68+69+70+71+72+73+74+75+76+77+78+79+80+81+82+83+84+85+86+87+88+89+90+91+92+93+94+95+96+97+98+99+100

看我们拿到了什么？最后问题就简单了：）

# echo {1..100} | tr ' ' '+' | bc

5050

看看~是不是很厉害呢？管道符是linux 中重要的符号，更多管道符的用法让小伙伴们慢慢探索吧！

# Bash shell脚本实例编程

## Shell 脚本简介

Shell 是一个用 C 语言编写的程序，它是用户使用 Linux 的桥梁。Shell 既是一种命令语言，又是一种程序设计语言。

Shell也是指一个应用程序，这个应用程序提供了一个界面，用户通过这个界面访问操作系统内核的服务，就是我们每次打开linux看到的那个黑忽忽的命令行窗口。

Ken Thompson 的 sh 是第一种 Unix Shell，Windows Explorer 是一个典型的图形界面 Shell。

Shell 脚本（shell script），是一种为 shell 编写的脚本程序。业界所说的 shell 通常都是指 shell 脚本，shell 和 shell script 是两个不同的概念。由于习惯的原因，简洁起见，本文出现的 "shell编程" 都是指 shell 脚本编程，不是指开发 shell 自身。

脚本其实就是短小的程序、用来让计算机自动化完成一系列工作的程序，这类程序可以用文本编辑器修改，不需要编译，通常是解释运行的。

## Shell 环境

### Shell分类

Shell 编程跟 java、php 编程一样，只要有一个能编写代码的文本编辑器和一个能解释执行的脚本解释器就可以了。

Linux 的 Shell 种类众多，常见的有：

* Bourne Shell（/usr/bin/sh或/bin/sh）
* Bourne Again Shell（/bin/bash）
* C Shell（/usr/bin/csh）
* K Shell（/usr/bin/ksh）
* Shell for Root（/sbin/sh）
* ……

**sh(全称 Bourne Shell)**: 是UNIX最初使用的 shell，而且在每种 UNIX 上都可以使用。Bourne Shell 在 shell 编程方面相当优秀，但在处理与用户的交互方面做得不如其他几种 shell。

**bash（全称 Bourne Again Shell）**: LinuxOS 默认的，它是 Bourne Shell 的扩展。 与 Bourne Shell 完全兼容，并且在 Bourne Shell 的基础上增加了很多特性。可以提供命令补全，命令编辑和命令历史等功能。它还包含了很多 C Shell 和 Korn Shell 中的优点，有灵活和强大的编辑接口，同时又很友好的用户界面。

**csh(全称 C Shell)**: 是一种比 Bourne Shell更适合的变种 Shell，它的语法与 C 语言很相似。

**Tcsh**: 是 Linux 提供的 C Shell 的一个扩展版本。Tcsh 包括命令行编辑，可编程单词补全，拼写校正，历史命令替换，作业控制和类似 C 语言的语法，他不仅和 Bash Shell 提示符兼容，而且还提供比 Bash Shell 更多的提示符参数。

**ksh (全称 Korn Shell)**: 集合了 C Shell 和 Bourne Shell 的优点并且和 Bourne Shell 完全兼容。

**pdksh**: 是 Linux 系统提供的 ksh 的扩展。pdksh 支持人物控制，可以在命令行上挂起，后台执行，唤醒或终止程序。

本文关注的是 Bash，也就是 Bourne Again Shell，由于易用和免费，Bash 在日常工作中被广泛使用。同时，Bash 也是大多数Linux 系统默认的 Shell。

在一般情况下，人们并不区分 Bourne Shell 和 Bourne Again Shell，所以，像 **#!/bin/sh**，它同样也可以改为 **#!/bin/bash**。

#! 告诉系统其后路径所指定的程序即是解释此脚本文件的 Shell 程序。

### 切换用户时的shell环境

采用“su”切换用户时，加不加小横线（-），对于shell环境是有很大区别的。在linux中，有6个配置文件对shell的有影响：

系统级：

/etc/profile

/etc/bashrc

用户级：

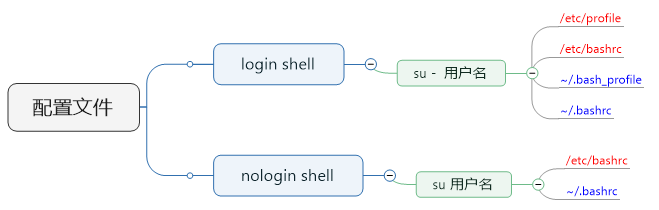
~/.bash\_profile

~/.bashrc

退出时执行：

~/.bash\_logout

~/.bash\_history



对于退出时，基本流程都是一样的，首先更新~/.bash\_history，然后执行~/.bash\_logout。但切换用户时，加不加小横线（-），加载的配置文件是不一样的，如上图所示，所以在切换用户时，如果不是有特殊要求，小横线（-）一定要加上！

如果改变了shell的配置文件，一般情况下，传统教材中会说重启sell后生效，其实不用那么麻烦，我们只需要在当前shell中把配置脚本重新执行一下就行了，命令如下：

# source ~/.bashrc

## Shell命令基础

### 最基本的连接符

1. “;”可以连续执行多个命令，只是命令的排序，无逻辑判断能力。

# cd /etc ; ls -al

总用量 1604

drwxr-xr-x 190 root root 12288 10月 3 11:53 .

drwxr-xr-x 24 root root 4096 9月 28 10:49 ..

-rw-r--r-- 1 root root 2981 4月 12 18:37 adduser.conf

-rw-r--r-- 1 root root 16 9月 23 06:47 adjtime

-rw-r--r-- 1 root root 185 4月 12 18:42 aliases

drwxr-xr-x 2 root root 20480 9月 25 15:08 alternatives

drwxr-xr-x 2 root root 4096 9月 23 00:04 amap

…

1. “&&”具备逻辑（真）判断能力

# cd /home/44444; ls //不管/home/44444是否存在，ls依然执行，显示当前目录下的文件

bash: cd: /home/44444: 没有那个文件或目录

adduser.conf inetsim profile.d

adjtime init protocols

aliases init.d proxychains.conf

alternatives initramfs-tools pulse

amap inputrc python

anacrontab insserv.conf.d python2.7

apache2 iproute2 python3

。。。。

# cd /home/44444 && ls //如果home/44444存在，ls才被执行，如果home/44444不存在，ls不执行。

bash: cd: /home/44444: 没有那个文件或目录

1. “||”具备逻辑（假）判断能力

# cd /home/44444 && echo '目录存在' ||echo '目录不存在！！！'

bash: cd: /home/44444: 没有那个文件或目录

目录不存在！！！

# cd /home && echo '目录存在' || echo '目录不存在！！！'

目录存在

///home/222/333/aaa/bbbb不存在，就执行mkdir -p /home/222/333/aaa/bbbb命令。

# ls /home/222/333/aaa/bbbb || mkdir -p /home/222/333/aaa/bbbb

ls: 无法访问'/home/222/333/aaa/bbbb': 没有那个文件或目录

# ls /home/222/333/aaa/bbbb

//用ping命令判断一个主机是否存活。

# ping -c1 192.168.44.2 && echo 'is up' || echo 'is down'

PING 192.168.44.2 (192.168.44.2) 56(84) bytes of data.

64 bytes from 192.168.44.2: icmp\_seq=1 ttl=128 time=0.096 ms

--- 192.168.44.2 ping statistics ---

1 packets transmitted, 1 received, 0% packet loss, time 0ms

rtt min/avg/max/mdev = 0.096/0.096/0.096/0.000 ms

is up

//用ping命令判断一个主机是否存活。并除去显示信息。&>表示混合重定向，即输出和错误都重定向。

# ping -c1 192.168.44.2 &>/dev/null && echo 'is up' || echo 'is down'

is up

### shell的“元字符”（通配符）与“原字符”

“元字符”与“原字符”：

* “元字符”代表不是原来的意思的字符，具有特别意思的字符，例如“\*”字面上就一个星号，但在shell中，表示“0到任意多个字符”，\*.txt就表示以.txt结尾的所有文件。
* “原字符”代表就是原来的意思的字符，没有特别意思的字符，例如“a”就是字母a，没有其它的意思。

1. 最常用的元字符：

\*：任意多个字符；

？：任意一个字符；

[]：括号中的任意一个字符；^表示取反。

()：在子sell中执行；

{}：集合

注：有很多元字符，这里只举几个例子，以后在使用过程中逐渐掌握，不要死记硬背！

例如：

# mkdir dy

# cd dy

# touch dzp dzp2 dzp3 dzp34 dzpcang dzpbols

# ls

dzp dzp2 dzp3 dzp34 dzpbols dzpcang

# ls dzp

dzp

# ls dzp\*

dzp dzp2 dzp3 dzp34 dzpbols dzpcang

# ls dzp?

dzp2 dzp3

# ls dzp[0-9]

dzp2 dzp3

# ls dzp[0-9][0-9]

dzp34

# ls dzp[^0-9]

ls: cannot access dzp[^0-9]: No such file or directory

# touch dzpa

# ls dzp[^0-9]

dzpa

# ls dzp[^0-9]\*

dzpa dzpbols dzpcang

1. Bash中的其他特殊符号

‘’:单引号。在单引号中的所有“元字符”都回归“原字符”，失去特殊意义；

“”:双引号。除“$”和“\”保留元字符以外，其它的都回归“原字符”，失去特殊意义；

`` :反引号。反引号里面的内容当成命令来执行，推荐使用$()，因为反引号容易看错；

$() :和反引号一样，用来想用系统命令；

# :以#号开关的行表示注释；

$ :用于调用变量的值，如需要调用名为name变量的值，则采用$name;

\：转义符号，实现“元字符”与“原字符”的相互转换。

实例：

# aa=123

# echo $aa //$是元字符，表示引用变量的值。这条命令的含义：打印变量aa的值。

123

# echo '$aa' //单引号内，所有元字符都转换为原字符，失去特殊意义。

$aa

# echo "$aa" //双引号内，$和\这两元字符特殊意义。其它元字符失去作用

123

# aa=ls //定义一个名aa的变量，因为后面的ls没有用反引号或$()等系统命令引用符号，所以被当成普通字符处理，这里的ls为原字符，就是原来的意思，不是系统命令。

# echo "$aa" //””符号在linux中原本表示引用变量名，是一个元字符，这条命令的意义为：显示名为aa的变量的值。

ls

# aa=`ls` //``符号在linux中原本表示引用系统命令，是一个元字符，这条命令的意义为：执行系统命令ls，把结果保存在名为aa的变量中。

# echo "$aa"

dzp

dzp2

dzp3

dzp34

dzpa

dzpbols

dzpcang

# bb=$(ls) //$()符号在linux中原本表示引用系统命令，是一个元字符，这条命令的意义为：执行系统命令ls，把结果保存在名为bb的变量中。

# echo $bb //$符号在linux中原本表示引用变量名，是一个元字符，这条命令的意义为：显示名为bb的变量的值。

dzp dzp2 dzp3 dzp34 dzpa dzpbols dzpcang

[root@localhost dy]# echo $(date)

Sat Jul 15 17:23:31 CST 2017

# echo \$bb //$符号在linux中原本表示引用变量，是一个元字符，在前面加上转义符（\）后，回归原字符，直接打印$这个符号，失去了特殊含义。

$bb

### Bash的快捷键

1. 编辑命令

Ctrl + a ：移到命令行首

Ctrl + e ：移到命令行尾

Ctrl + f ：按字符前移（右向）

Ctrl + b ：按字符后移（左向）

Alt + f ：按单词前移（右向）

Alt + b ：按单词后移（左向）

Ctrl + xx：在命令行首和光标之间移动

Ctrl + u ：从光标处删除至命令行首

Ctrl + k ：从光标处删除至命令行尾

Ctrl + w ：从光标处删除至字首

Alt + d ：从光标处删除至字尾

Ctrl + d ：删除光标处的字符

Ctrl + h ：删除光标前的字符

Ctrl + y ：粘贴至光标后

Alt + c ：从光标处更改为首字母大写的单词

Alt + u ：从光标处更改为全部大写的单词

Alt + l ：从光标处更改为全部小写的单词

Ctrl + t ：交换光标处和之前的字符

Alt + t ：交换光标处和之前的单词

Alt + Backspace：从光标处删除至字首，与 Ctrl + w 相同类似，分隔符有些差别。

1. 重新执行命令

Ctrl + r：逆向搜索命令历史

Ctrl + g：从历史搜索模式退出

Ctrl + p：历史中的上一条命令

Ctrl + n：历史中的下一条命令

Alt + .：使用上一条命令的最后一个参数

1. 控制命令

Ctrl + l：清屏

Ctrl + o：执行当前命令，并选择上一条命令

Ctrl + s：阻止屏幕输出

Ctrl + q：允许屏幕输出

Ctrl + c：终止前台正在运行的命令，对后台运行的进程无效。

Ctrl + z：挂起命令

1. Bang (!) 命令

!!：执行上一条命令

!blah：执行最近的以 blah 开头的命令。

!blah:p：仅打印输出，而不执行

!$：上一条命令的最后一个参数，与 Alt + . 相同

!$:p：打印输出 !$ 的内容

!\*：上一条命令的所有参数

!\*:p：打印输出 !\* 的内容

^blah：删除上一条命令中的 blah

^blah^foo：将上一条命令中的 blah 替换为 foo

^blah^foo^：将上一条命令中所有的 blah 都替换为 foo

### 前台和后台运行

1. 命令

主要是5个命令：

1. &

最经常被用到,这个用在一个命令的最后，可以把这个命令放到后台执行

1. ctrl + z

可以将一个正在前台执行的命令放到后台，并且暂停

1. jobs

查看当前有多少在后台运行的命令

1. fg %n

将后台中的命令调至前台继续运行

如果后台中有多个命令，可以用 fg%jobnumber将选中的命令调出，%jobnumber是通过jobs命令查到的后台正在执行的命令的序号(不是pid)

1. bg %n

将一个在后台暂停的命令，变成继续执行

注:"n"为jobs查看到的进程编号.

1. 具体使用

* 进程前台和后台的切换

当用bash启动一个程序,你可以用在程序后加&,实现程序在后台运行；

例如: 原本想输入sleep 8000 &，但如果你忘了输入&，输成了sleep 8000但你又不想把程序停止后重新启动，你可以这么做:

# sleep 8000

^Z //按Ctrl + Z暂停前台工作并放到后台，这样你就可以达到和sleep 8000 &一样的效果了

[1]+ 已停止 sleep 8000

# jobs //然后用jobs命令查看在后台一共启动了多少程序,你可以看到你的sleep 8000在其中，它的状态是Stoped的,作业ID号为1.

[1]+ 已停止 sleep 8000

# fg %1 //如果需要把任务回到前台，只需要fg %1

sleep 8000

* 暂停一个进程，执行其它进程后再切换回来

# vim /etc/dns2tcpd.conf

^Z //按Ctrl + Z暂停前台工作并放到后台.

[1]+ 已停止 vim /etc/dns2tcpd.conf

# ip addr

1: lo: <LOOPBACK,UP,LOWER\_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000

link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

inet 127.0.0.1/8 scope host lo

valid\_lft forever preferred\_lft forever

inet6 ::1/128 scope host

valid\_lft forever preferred\_lft forever

2: eth0: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc pfifo\_fast state UP group default qlen 1000

link/ether 00:0c:29:03:38:e5 brd ff:ff:ff:ff:ff:ff

inet 192.168.44.135/24 brd 192.168.44.255 scope global dynamic noprefixroute eth0

valid\_lft 1271sec preferred\_lft 1271sec

inet 192.168.44.10/24 brd 192.168.44.255 scope global secondary noprefixroute eth0

valid\_lft forever preferred\_lft forever

inet6 fe80::961f:dbf3:f392:5887/64 scope link noprefixroute

valid\_lft forever preferred\_lft forever

# jobs //然后用jobs命令查看在后台一共启动了多少程序,发现作业ID号为1

[1]+ 已停止 vim /etc/dns2tcpd.conf

# fg %1

当你有些工作比如updatedb, find等需要一定时间,这些工作可以考虑让它在后台运行,而你可以用bash做其它工作。

### echo设置颜色

先学装X技能，自定义字体的显示颜色，将显示结果的不同部分用不同颜色显示，将大大提升用户体验。

echo要变换颜色的时候，要使用参数-e格式:

echo -e "\033[字背景颜色;字体颜色m字符串 \033[0m"

例如:

# echo -e "\033[40;31m显示红色 \033[0m"

其中40的位置代表底色, 31的位置是代表字的颜色，显示结果如下图所示：



注：

1、字背景颜色和文字颜色之间是英文的“""”

2、文字颜色后面有个m

3、字符串前后可以没有空格，如果有的话，输出也是同样有空格

格式是固定的，不要部我为什么：）

\033[ ; m …… \033[0m

让字体变为红色并且不停的闪烁

# echo -e "\033[31m \33[05m 请确认是否要停止当前的squid进程,输入 [Y/N] \033[0m"

或者

#echo -e "\033[31m \033[05m 请确认是否要停止当前的squid进程,输入 [Y/N] \033[0m"

下面看几个例子：

echo -e "\033[30m 黑色字 \033[0m"

echo -e "\033[31m 红色字 \033[0m"

echo -e "\033[32m 绿色字 \033[0m"

echo -e "\033[33m 黄色字 \033[0m"

echo -e "\033[34m 蓝色字 \033[0m"

echo -e "\033[35m 紫色字 \033[0m"

echo -e "\033[36m 天蓝字 \033[0m"

echo -e "\033[37m 白色字 \033[0m"

echo -e "\033[40;37m 黑底白字 \033[0m"

echo -e "\033[41;37m 红底白字 \033[0m"

echo -e "\033[42;37m 绿底白字 \033[0m"

echo -e "\033[43;37m 黄底白字 \033[0m"

echo -e "\033[44;37m 蓝底白字 \033[0m"

echo -e "\033[45;37m 紫底白字 \033[0m"

echo -e "\033[46;37m 天蓝底白字 \033[0m"

echo -e "\033[47;30m 白底黑字 \033[0m"

字背景颜色范围:

30-----------39:

30:黑 31:红 32:绿 33:黄 34:蓝色 35:紫色 36:深绿 37:白色\33[0m 关闭所有属性 \33[1m 设置高亮度 \33[4m 下划线 \33[5m 闪烁 \33[7m 反显 \33[8m 消隐 \33[30m -- \33[37m 设置前景色 \33[40m -- \33[47m 设置背景色 \33[nA 光标上移n行 \33[nB 光标下移n行 \33[nC 光标右移n行 \33[nD 光标左移n行 \33[y;xH设置光标位置 \33[2J 清屏 \33[K 清除从光标到行尾的内容 \33[s 保存光标位置 \33[u 恢复光标位置 \33[?25l 隐藏光标 \33[?25h 显示光标

40----49:

40:黑 41:深红 42:绿 43:黄色 44:蓝色 45:紫色 46:深绿 47:白色字颜色。

## 第一个shell脚本

### 创建脚本

打开文本编辑器(可以使用 vi/vim 命令来创建文件)，新建一个文件 test.sh，扩展名为 sh（sh代表shell），扩展名并不影响脚本执行，见名知意就好，如果你用 php 写 shell 脚本，扩展名就用 php 好了。

输入一些代码，第一行一般是这样：

#!/bin/bash  
echo "Hello World !"

* #! 是一个约定的标记，它告诉系统这个脚本需要什么解释器来执行，即使用哪一种 Shell。
* echo 命令用于向窗口输出文本。

### 运行 Shell 脚本的3种方法

下面这句话ubuntu（linux）下 source、sh、bash、./ 执行脚本的区别是什么？

1. **Source和“.”命令用法：**

source FileName

作用:在当前bash环境下读取并执行 FileName 中的命令。该 filename 文件可以"无执行权限"。注：该命令通常用命令 **.** 来替代。

1. **sh、bash的命令用法：**

sh FileName

或

bash FileName

作用:打开一个子shell来读取并执行 FileName 中命令。该 filename 文件可以"无执行权限"。

注：运行一个shell脚本时会启动另一个命令解释器。

1. **./的命令用法：**

./FileName

作用: 打开一个子shell来读取并执行 FileName 中命令，该 filename 文件需要 "执行权限"。

注：运行一个 shell 脚本时会启动相应的命令解释器。

## Shell脚本编程基础

从本节开始，我们要开始正式的shell编程之旅了，本章将简单介绍shell编程必备的基础知识。

### shell变量

本节将了解以下内容：

1. 什么是shell变量?
2. 变量的类型
3. 变量的定义及赋值方式
4. 变量的运算
5. 变量"内容"的删除和替换
6. i++与++i

#### shell变量

用一个固定的字符串去表示不固定的内容

#### 变量的类型

1. 自定义变量

* 定义变量：

变量名=变量值

注意：变量名必须以字母或下划线开头，区分大小写

例如：

# ip1=192.168.2.115 //定义变量ip1,并赋值为192.168.2.115

# echo ${ip1} //打印变量ip1的值，执行结果为：192.168.2.115

* 引用变量：

$变量名 或 ${变量名}

* 查看变量：

echo $变量名 set(所有变量：包括自定义变量和环境变量)

* 取消变量：

unset 变量名

* 作用范围：

仅在当前 shell 中有效

1. 环境变量

* 定义环境变量：

方法一：

export back\_dir1=/home/backup

方法二：

export back\_dir1 //将自定义变量转换成环境变量

* 引用环境变量：

$变量名 或 ${变量名} //推荐方式

* 查看环境变量：

echo ${变量名} //推荐

$变量名

env //查看所有环境变量

env |grep back\_dir2 //查看含有指定关键字的环境变量

unset 变量名 //取消环境变量

* 变量作用范围：

在当前 shell 和子 shell 有效

自定义变量相当于C语言中的局部变量，环境变量当于C语言中的全局变量！！！

1. 位置变量

$1 $2 $3 $4 $5 $6 $7 $8 $9 ${10}

1. 预定义变量

$0 脚本名

$\* 所有的参数

$@ 所有的参数

$# 参数的个数

$$ 当前进程的 PID

$! 上一个后台进程的 PID

$? 上一个命令的返回值 0 表示成功

示例 ：

# vim test.sh

echo "第 2 个位置参数是$2"

echo "第 1 个位置参数是$1"

echo "第 4 个位置参数是$4"

echo "所有参数是: $\*"

echo "所有参数是: $@"

echo "参数的个数是: $#"

echo "当前进程的 PID 是: $$"

echo '$1='$1

echo '$2='$2

echo '$3='$3

echo '$\*='$\*

echo '$@='$@

echo '$#='$#

echo '$$='$$

1. 了解$\*和$@区别

$\* 和 $@ 都表示传递给函数或脚本的所有参数，它们有细微的差别：

* 不被双引号(" ")包含时，都以"$1" "$2" … "$n" 的形式输出所有参数。
* 被双引号(" ")包含时，"$\*" 会将所有的参数作为一个整体，以"$1 $2 … $n"的形式输出所有参数；"$@" 会将各个参数分开，以"$1" "$2" … "$n" 的形式输出所有参数。

下面的例子可以清楚的看到 $\* 和 $@ 的区别：

#!/bin/bash

echo "print each param from \$\*"

for var in $\*

do

echo "$var"

done

echo "print each param from \$@"

for var in $@

do

echo "$var"

done

echo "print each param from \"\$\*\""

for var in "$\*"

do

echo "$var"

done

echo "print each param from \"\$@\""

for var in "$@"

do

echo "$var"

done

执行 ./test.sh "a" "b" "c" "d"，看到下面的结果：

print each param from $\*

a

b

c

d

print each param from $@

a

b

c

d

print each param from "$\*"

a b c d

print each param from "$@"

a

b

c

d

示例 ：

# vim ping.sh

#!/bin/bash

ping -c2 $1&>/dev/null

if [ $?= 0 ];then

echo "host $1 is ok"

else

echo "host $1 is fail"

fi

# chmod a+x ping.sh

# ./ping.sh 192.168.2.25

#### 变量的定义及赋值方式

1. 显式赋值

变量名=变量值

示例：

# ip1=192.168.1.251 school="BeiJing 1000phone"

# today1=`date +%F` today2=$(date +%F)

1. read 从键盘读入变量值

read 变量名

read -p "提示信息: " 变量名

read -t 5 -p "提示信息: " 变量名

read -n 2 变量名

示例：

# vim first.sh

back\_dir1=/var/backupread -p "请输入你的备份目录: " back\_dir2

echo $back\_dir1echo $back\_dir2

# chmod a+x first.sh

# sh first.sh

示例：

# vim ping2.sh

#!/bin/bash

read -p "Input IP: " ip

ping -c2 $ip &>/dev/null

if [ $?= 0 ];then

echo "host $ip is ok"

else

echo "host $ip is fail"

fi

# chmod a+x ping2.sh

# ./ping.sh

定义或引用变量时注意事项：

" "弱引用

' '强引用

# school=1000phone

# echo "${school} is good"

1000phone is good

# echo '${school} is good'

${school} is good

` `命令替换等价于$()，反引号中的shell命令会被先执行。

# touch `date +%F`\_file1.txt

# touch $(date +%F)\_file2.txt

# disk\_free3="df -Ph |grep '/$' |awk '{print $4}'" //执行结果可能不是预期的

# disk\_free4=$(df -Ph |grep '/$' |awk '{print $4}')

# disk\_free5=`df -Ph |grep '/$' |awk '{print $4}'`

#### 变量的运算

1. 整数运算

* 方法一：expr

**expr命令**是linux内置的表达式计算工具，使用它完成表达式的求值操作。

expr 1 + 2

expr $num1 + $num2

注意：除号的需要转义（+ - \\* / % \*\*）

* 方法二：$(())

echo $(($num1+$num2))

运算符号不变（+ - \* / % \*\*）

# echo $((num1+num2))

# echo $((5-3\*2))

# echo $(((5-3)\*2))

# echo $((2\*\*3));sum=$((1+2))

# echo $sum

* 方法三：$[]

# echo $[5+2]

# echo $[5\*\*2]

运算符号不变（+ -\* / % \*\*）

* 方法四：let

let sum=2+3; echo $sum

let i++; echo $i

运算符号不变（+ -\* / % \*\*）

1. 小数运算

# echo "2\*4" |bc

# echo "2^4" |bc

# echo "scale=2;6/4" |bc

# awk 'BEGIN{print 1/2}'

# echo "print 5.0/2" |python

#### 变量"内容"的删除和替换（扩展）

1. 内容"的删除

# url=www.sina.com.cn

# echo ${#url} //获取变量值的长度

15

# echo ${url} //标准查看

[www.sina.com.cn](http://www.sina.com.cn)

# echo ${url#\*.} //从前往后删除，最短匹配，以第一个点（含点）为标记，将前面的字符全部删除

sina.com.cn

# echo ${url##\*.} //从前往后，最长匹配,贪婪匹配,以最后一个点（.）为标记，将前面的字符（含.）全部删除

cn

# url=www.sina.com.cn

# echo ${url}

[www.sina.com.cn](http://www.sina.com.cn)

# echo ${url%.\*} //从后往前找到第一个点（.），并将后面的字符（含.）全部删除。最短匹配。

[www.sina.com](http://www.sina.com)

# echo ${url%%.\*} //从后往前找到最一个点（.），并将后面的字符（含.）全部删除。贪婪匹配。

www

# url=www.sina.com.cn

# echo ${url#a.}

[www.sina.com.cn](http://www.sina.com.cn)

# echo ${url#\*sina.}

com.cn

# echo $HOSTNAME

tianyun.1000phone.com

# echo ${HOSTNAME%%.\*}

tianyun

1. 索引及切片

# url=www.sina.com.cn

# echo ${url:0:5}

# echo ${url:5:5}

# echo ${url:5}

1. 内容"的替换

# url=www.sina.com.cn

# echo ${url/sina/baidu}

www.baidu.com.cn

# url=www.sina.com.cn

# echo ${url/n/N} //最短匹配，只有第一个n都被替换为大写

www.siNa.com.cn

# echo ${url//n/N} //贪婪匹配，所有n都被替换为大写

[www.siNa.com.cN](http://www.siNa.com.cN)

1. 变量的替代

# unset var1

# echo ${var1}

# echo ${var1-aaaaa}

aaaaa

# var2=111

# echo ${var2-bbbbb}

111

# var3=

# echo ${var3-ccccc}

小结：${变量名-新的变量值}

* 变量没有被定义：会使用“新的变量值“ 替代，即创建变量并赋值
* 变量定义了但没有被赋值（空值）：不会被替代

# unset var1

# unset var2

# unset var3

# var2=

# var3=111

# echo ${var1:-aaaa}

aaaa

# echo ${var2:-aaaa}

aaaa

# echo ${var3:-aaaa}

111

小结：${变量名:-新的变量值}

* 变量没有被定义或者没有被赋值（空值）：都会使用“新的变量值“ 替代
* 变量有有明确的赋值：不会被替代

动手实验，了解下面命令的含义

# echo ${var3+aaaa}

# echo ${var3:+aaaa} //$var为空时,不变，不为空时，把string赋值给var，

# echo ${var3=aaaa}

# echo ${var3:=aaaa} //$var为空时,把string赋值给var，不为空时不变

# echo ${var3?aaaa}

# echo ${var3:?aaaa} //$var为空时,把string输出到标准错误中，不为空时不变

* ${var:-string} 和 ${var:=string}一样：

若变量var为空或者未定义,则用在命令行中用string来替换${var:-string}；当变量var不为空时,则用变量var的值来替换${var:-string}

# echo $a

# echo ${a:-bcc}

bcc

# echo $a

# a=ajax

# echo ${a:-bcc}

ajax

# unset a

# echo $a

# echo ${a:=bbc}

bbc

# echo $a

bbc

发现${var:-string}和${var:=string}比较 后者发现$var为空时,把string赋值给了var后者是一种赋值默认值的常见做法

* ${var:+string}

规则和上面的完全相反即只有当var不是空的时候才替换成string,若var为空时则不替换或者说是替换成变量var的值,即空值

# a=ajax

# echo $a

ajax

# echo ${a:+bbc}

bbc

# echo $a

ajax

# unset a

# echo $a

# echo ${a:+bbc}

#

* ${var:?string}

替换规则:若变量var不为空,则用变量var的值来替换${var:?string}若变量var为空,则把string输出到标准错误中,并从脚本中退出。可利用此特性来检查是否设置了变量的值

# echo $a

# echo ${a:?bbc}

-bash: a: bbc

# a=ajax

# echo ${a:?bbc}

ajax

# a=ajax

# echo ${a:-`date`}

ajax

# unset a

# echo ${a:-`date`}

2017年 02月 21日 星期二 10:13:46 CST

# echo ${a:-$(date)}

2017年 02月 21日 星期二 10:13:59 CST

# b=bbc

# echo ${a:-$b}

bbc

#### i++ 和++i（了解）

1. 对变量的值的影响：

# i=1

# let i++

# echo $i

2

# j=1

# let ++j

# echo $j

2

1. 对表达式的值的影响：

# unset i

# unset j

# i=1

# j=1

# let x=i++ //先赋值，再运算

# let y=++j //先运算，再赋值

# echo $x

1

# echo $y

2

# echo $i

2

# echo $j

2

### 条件判断语句—if

#### Shell 条件测试3种格式

1. test 条件表达式 //test是linux命令
2. **[** 条件表达式 ] //[其实就是test命令,]是固定的格式
3. **[[** 条件表达式 ]] //[[也是test命令,可以使用正则表达式

关于条件表表达式的用法，建议大家可以使用man test命令，查看帮助文档。

#### 条件表达式的3种类型

1. 文件测试

* 使用格式

[ 操作符 文件或目录 ] //[其实就是test命令,所以两边的空格一定要存在！

示例：

# test -d /home //用test命令的-d选项，测试/home是否为一个目录。

# echo $? //$?表示上一次的返回结果，0为真，非0为假。

0

# test -d /home11111

# echo $?

1

# [ -d /home ] //换种表示方式，测试/home是否为一个目录。

* 常见的文件测试选项

[ -e dir|file ] //文件或目录是否存在。

[-d dir] //是否为一个目录。

[ -f file ] //是否存在，而且是文件

[ -r file ] //当前用户对该文件是否有读权限，对root无效

[ -x file ] //当前用户对该文件是否有执行权限，对root无效

[ -w file ] //当前用户对该文件是否有写权限，对root无效

[ -L file ] //文件是否为链接文件

示例：

# [ ! -d /ccc ] && mkdir /ccc //如果/ccc目录不存在为真，就创建目录。&&为逻辑真

# [ -d /ccc ] || mkdir /ccc //如果/ccc目录存在为假，就创建目录。||为逻辑假

1. 数值比较[ 整数1 操作符整数2 ]

* Bash风格的比较符号

[ 1 -gt 10 ] 大于

[ 1 -lt 10 ] 小于

[ 1 -eq 10 ] 等于

[ 1 -ne 10 ] 不等于

[ 1 -ge 10 ] 大于等于

[ 1 -le 10 ] 小于等于

示例：

# disk\_use=$(df -P |grep '/$' |awk '{print $5}' |awk-F% '{print $1}')

# [ $disk\_use -gt 90 ] && echo "war......"

# [ $disk\_use -gt 60 ] && echo "war......"

war......

# id -u

0

# [ $(id -u) -eq 0 ] && echo "当前是超级用户"

当前是超级用户

$[ $UID -eq0 ] && echo "当前是超级用户" || echo "you不是超级用户"

you不是超级用户

* C语言风格的数值比较

# ((1<2));echo $?

0

# ((1==2));echo $?

1

# ((1>2));echo $?

1

# ((1>=2));echo $?

1

# ((1<=2));echo $?

0

# ((1!=2));echo $?

0

# ((`id -u`>0));echo $?

1

# (($UID==0));echo $?

0

1. 字符串比较

重要提示：字符比较一定要使用双引号

# [ "$USER"= "root"];echo $? //USER是环境变量，表示当前用户名

0

# [ "$USER"=="root"];echo $? //注意这里使用的比较符号是“==”

0

# BBB="" //定义变量BBB，值为空

# echo ${#BBB} //测试变量BBB的长度，因为值为空，所以长度为0

0

# [ -z"$BBB" ] //测试变量BBB字符长度是否为0

# echo $?

0

# [ -n"$BBB" ] //测试变量BBB字符长度是否不为0

# echo $?

1

# var1=111 //定义变量var1，值为111

# var2= //定义变量var2，值为pwa

//var3变量没有定义

# echo ${#var1} //测试变量var1的字符长度

3

# echo ${#var2} //测试变量var2的字符长度

0

# echo ${#var3} //测试变量var3的字符长度

0

# [ -z "$var1" ];echo $? //测试变量var1的是否为0长度

1

# [ -z "$var2" ];echo $? //测试变量var2的是否为0长度

0

# [ -z "$var3" ];echo $? //测试变量var3的是否为0长度

0

# [ -n "$var1" ];echo $? //测试变量var1的是否不为空

0

# [ -n "$var2" ];echo $? //测试变量var2的是否不为空

1

# [ -n "$var3" ];echo $? //测试变量var3的是否不为空

1

小结：变量为空或未定义：长度都为0

# [ "$USER" = "root" ];echo $? //测试当前用户是否是root? USER是环境变量

0

# [ "$USER" = "alice" ];echo $? //测试当前用户是否是alice? USER是环境变量

1

# [ "$USER" != "alice" ];echo $? //测试当前用户是否不是alice? 不是为真。

0

# [ 1 -lt 2 -a 5 -gt 10 ];echo $? //1>2并且5>10，是否为真？-a表示and

1

# [ 1 -lt 2 -o 5 -gt 10 ];echo $? //1>2或者5>10，是否为真？-o表示or

0

# [[ 1 -lt 2 && 5 -gt 10 ]];echo $? //1>2并且5>10，是否为真？&&表示and

1

# [[ 1 -lt 2 ||5 -gt 10 ]];echo $? //1>2或者5>10，是否为真？||表示or

0

# [ "$USER" = "root" ];echo $?

0

//下面的语句：当前用户名是否以字母r开关？=~表示正则表达式匹配符，^r表示以字母r开头。正则表达式我们将在以后详细介绍。

# [ "$USER" =~ ^r ];echo $?

bash: [: =~: binary operator expected //直接报错

2 //通常情况下：为真返回0，为假返回1，这里返回值为2，表示程序运行异常

# [[ "$USER" =~ ^r ]];echo $? //使用正则，要使用双中括号，运行正常了

0

#### 实用技巧及案例

1. 判断变量是不是数字：

# num10=123

# num20=ssss1114ss

# [[ "$num10" =~ ^[0-9]+$ ]];echo $?

0

# [[ "$num20" =~ ^[0-9]+$ ]];echo $?

1

案例1：

# cat test02.sh

脚本全文

#!/bin/bash

#判断用户输入的是否是数字

read -p"请输入一个数值: " num

if [[ ! "$num" =~ ^[0-9]+$ ]];then

echo "你输入的不是数字，程序退出!!!"

exit

fi

案例2：

# cat test03.sh

脚本全文如下：

#!/bin/bash #判断用户输入的是否是数字

read -p "请输入一个数值: " num

while :

do

if [[ $num =~ ^[0-9]+$ ]];then

break

else

read -p "不是数字，请重新输入数值: " num

fi

done

echo "你输入的数字是: $num"

1. 调试脚本：

# sh -n 02.sh 仅调试syntax error

# sh -vx 02.sh 以调试的方式执行，查询整个执行过程

#### 总结

1. 分支结构特殊运算符号总结

() 子shell中执行

(()) C语言风格的数值比较及运算

$() 命令替换，相当于反引号的作用

$(()) 整数运算

{} 列表

${} 引用变量，也可以用双引号

[] 条件测试，不支持正则=~

[[]] 条件测试，支持正则=~

$[] 整数运算

1. 执行脚本方式的复习：

# ./01.sh 需要执行权限在子shell中执行

# bash 01.sh 不需要执行权限在子shell中执行

# .01.sh 不需要执行权限在当前shell中执行

# source 01.sh 不需要执行权限在当前shell中执行

提示：通常修改系统配置文件中如/etc/profile 的PATH等变量后，想要使之在当前shell中生效时，可以source方式执行脚本，不必重启机器。

1. if流程控制结构总结：

* 单分支结构

if 条件测试then

命令序列

Fi

* 双分支结构

if 条件测试then

命令序列

else

命令序列

fi

* 多分支结构

if 条件测试1then

命令序列

elif 条件测试2then

命令序列

elif 条件测试3 then

命令序列

...

else

命令序列

fi

示例：

read -p "确认开始安装KVM [y]: " kvm\_install

if [ ! "${kvm\_install}" = "y" ];then

echo -e "$red\_col输入不正确! $reset\_col"

exit

### 分支结构—case … esac和select

#### Case … esac分支结构

|  |  |
| --- | --- |
| case 变量 in  模式1)  命令序列1  ;;  模式2)  命令序列2  ;;  模式3)  命令序列3  ;;  \*) 无匹配后命令序列  esac |  |

动手操作：

# vim linux\_version.sh

下面是脚本的全文，作用是判断linux操作系统的类型。

#! /bin/bash

#show linux OS version.

if [ -e /etc/redhat-release ]; then

Version=`cat /etc/redhat\_release |cut -d' ' -f1 `

else

Version=`cat /etc/issue |cut -d' ' -f1 `

fi

echo "The operating system you are using is： $Version"

case $Version in

Red|RED|red|CentOS|centOS|centos)

P\_M= "yun"

;;

Kali|kali|KALI|Debian|debian|DEBIAN|ubuntu|Ubuntu|UBUNTU)

P\_M="apt-get"

;;

\*)

echo "Sory, dose not supposr this OS!! "

exit 1

esac

echo "The recommended installation command is: $P\_M"

下面是执行结果：

# ./linux\_version.sh

The operating system you are using is： Kali

The recommended installation command is: apt-get

不要试图在脚本以外查看脚本内的变量值！！！！如下：

# echo "$P\_M"

# echo "$version"

在case中用通配符：

# vim case\_tongpeifu.sh

脚本全文如下：

#!/bin/bash

while true

do

echo "Hit a key, then hit return."

read KeyPress

case "$KeyPress" in

"X" ) exit;;

[[:lower:]] ) echo "Lowecaoe letter";;

[[:upper:]] ) echo "Opper letter";;

[0-9] ) echo "Digit";;

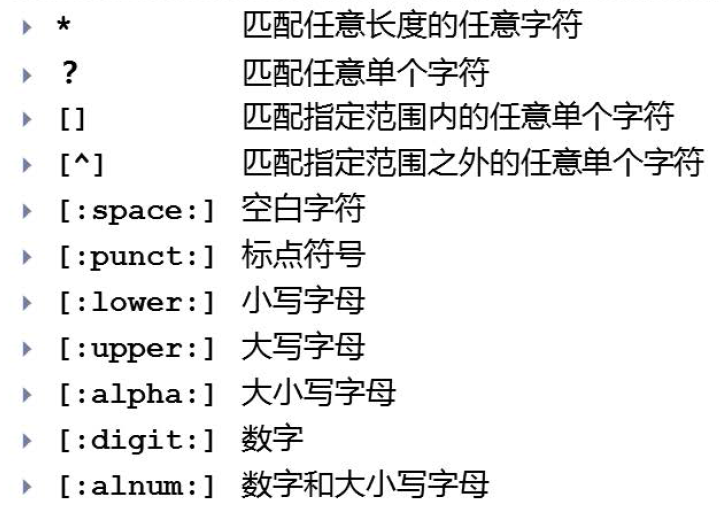
\* ) echo "Punctuation,whitespace,orother";;

esac

done

exit

支持的通配符如下：



### 循环结构

#### for循环

循环次数是固定的

for 语法结构

Shell风格：

for 变量名 [ in 取值列表 ]

do

循环体

done

C语言风格：

for ((初值;条件;步长))

do

循环体

done

案例1: ping测试主机

# cat ip.txt

172.16.8.100

172.16.8.4

172.16.100.254

案例2: 通过用户列表文件创建用户

# cat user.txt

zhuzhu1

yang1

zhang2

#### while循环

循环次数不一定是固定的 可以固定 可以不固定

while语句结构

while 条件测试

do

循环体、

done

当条件测试成立（条件测试为真），执行循环体

#### until循环

until语法结构

until 条件测试

do

循环体

done

当条件测试成立（条件测试为假），执行循环体

### 影响程序的Shell内置命令

true ：人工干预，使用命令使条件为真（0）

alse ：人工干预，使用命令使条件为假（非0）

exit ：退出整个程序

break ：结束当前循环，或跳出本层循环

continue ：忽略本次循环剩余的代码，直接进行下一次循环

shift ：使位置参数向左移动，默认移动1位，可以使用shift 2

对于shell 脚本编程基础知识就介绍这么多，如果对shell 脚本完全不会的小伙伴，可能无法继续下面的内容，建议先去学习第七章整章和第八章的内容；我们在以后的章节中，将采用“脚本+逐行注释”的方式来开始kali linux的学习。

# bash 进阶知识

## 进程并发与控制

### 复习FD

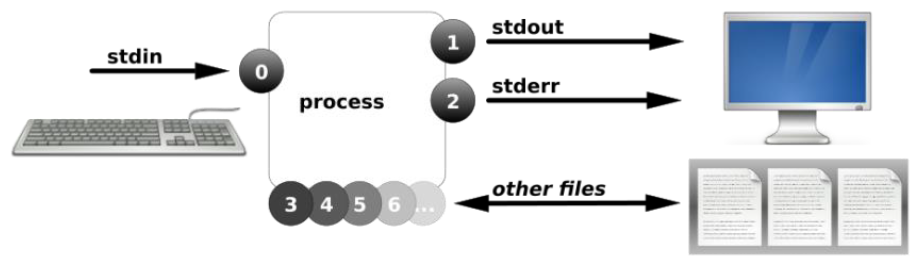
File Descriptors （FD，文件描述符）或文件句柄：

进程使用文件描述符来管理打开的文件。

# ls /proc/$$/fd //查看当前进程的FD, $$表示当前进程。

0 1 2 3 255

0, 1和2,我们已经知道是：标准输入（standard input）,标准输出（standard output）,标准错误（standard error）。



# ls -al /proc/$$/fd

total 0

lr-x------ 1 root root 64 Sep 6 13:32 0 -> /dev/pts/0

lrwx------ 1 root root 64 Sep 6 13:32 1 -> /dev/pts/0

lrwx------ 1 root root 64 Sep 6 13:32 2 -> /dev/pts/0

lrwx------ 1 root root 64 Sep 6 15:38 255 -> /dev/pts/0

#### 使用FD打开指定文件

我们新建一个文件，并手工指定6号FD打开新建的文件。

# touch /file1

# exec 6<> /file1 //打开文件

# ls -al /proc/$$/fd

total 0

lr-x------ 1 root root 64 Sep 6 13:32 0 -> /dev/pts/0

lrwx------ 1 root root 64 Sep 6 13:32 1 -> /dev/pts/0

lrwx------ 1 root root 64 Sep 6 13:32 2 -> /dev/pts/0

lrwx------ 1 root root 64 Sep 6 15:38 255 -> /dev/pts/0

lrwx------ 1 root root 64 Sep 6 13:32 6 -> /file1

#### 奇迹一：通过FD修改文件

通过操作FD来改变物理文件的内容。

# echo "hello FD" > /proc/$$/fd/6

# cat /proc/$$/fd/6

hello FD

# cat /file1

hello FD

#### 奇迹二：通过FD恢复文件

通过FD来恢复被删除物理文件。

# rm -rf /file1

# ls -al /proc/$$/fd

total 0

lr-x------ 1 root root 64 Sep 6 13:32 0 -> /dev/pts/0

lrwx------ 1 root root 64 Sep 6 13:32 1 -> /dev/pts/0

lrwx------ 1 root root 64 Sep 6 13:32 2 -> /dev/pts/0

lrwx------ 1 root root 64 Sep 6 15:38 255 -> /dev/pts/0

lrwx------ 1 root root 64 Sep 6 13:32 6 -> /file1 (deleted)

通过上面的命令显示结果可以看出，物理文件（存储在硬盘上）已经被删除。

注意：虽然存储在硬盘上的物理文件被删除了，当用户打开硬盘上的一个文件时，是将硬盘上的文件复制一份到内存中，由于当前用户打开的FD还没有关闭，FD指向内存中的某一块地址空间，这块内存地址空间中存储的正是我们以前打开file1文件。所以，虽然硬盘上的物理文件被删除了，但内存中的文件还在，所以我们可以利用这个机制恢复物理文件。

# cat /proc/$$/fd/6

hello FD

# cat /proc/$$/fd/6 > /file1

# cat /file1

hello FD

# ls -al /proc/$$/fd

总用量 0

dr-x------ 2 root root 0 10月 7 10:27 .

dr-xr-xr-x 9 root root 0 10月 7 10:27 ..

lrwx------ 1 root root 64 10月 7 10:27 0 -> /dev/pts/0

lrwx------ 1 root root 64 10月 7 10:27 1 -> /dev/pts/0

lrwx------ 1 root root 64 10月 7 10:27 2 -> /dev/pts/0

lrwx------ 1 root root 64 10月 7 10:26 255 -> /dev/pts/0

lrwx------ 1 root root 64 10月 7 10:27 6 -> '/file1 (deleted)'

FD指向的文件依然显示被删除，这是因为，虽然我们恢复了文件的内容，但由于每一个文件都有一个唯一的“i结点”，我们恢复的这个文件的“i结点”与以前的文件不同，得不到FD的认可：），也就是说：我们只是将文件的内容恢复到了一个新文件中，并不是正在恢复了原来那个文件！

#### 关闭FD

FD是非常宝贵的资源，使用完以后一定要记得释放！

# exec 6<&-

# ls -al /proc/$$/fd

total 0

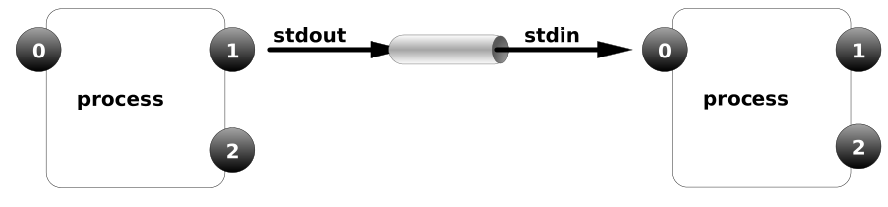
lr-x------ 1 root root 64 Sep 6 13:32 0 -> /dev/pts/0

lrwx------ 1 root root 64 Sep 6 13:32 1 -> /dev/pts/0

lrwx------ 1 root root 64 Sep 6 13:32 2 -> /dev/pts/0

lrwx------ 1 root root 64 Sep 6 15:38 255 -> /dev/pts/0

### 再谈管道



1. 匿名管道

# rpm -qa |grep bash

1. 命名管道

# mkfifo /tmp/tmpfifo

# file /tmp/tmpfifo

/tmp/tmpfifo: fifo (named pipe)

# tty

/dev/pts/0

# rpm -qa > /tmp/tmpfifo

# tty

/dev/pts/1

# grep bash /tmp/tmpfifo

bash-4.1.2-14.el6.x86\_64

### 进程并发与控制实例

#### 案例1：多文件处理

#!/bin/bash

exec 7<> /etc/hosts

exec 8<> /etc/sysconfig/network

while read -u 7 line

do

echo $line read -u 8 line2 echo $line2

done

exec 7<&-

exec 8<&-

#### 案例2：并发 ping\_multi\_thread1.sh

#### 案例3：并发 ping\_multi\_thread2.sh

## 用实例玩数组

### 数组变量

普通数组：只能使用整数作为数组索引

关联数组：可以使用字符串作为数组索引

#### 普通数组

1. 定义数组：

方法一： 一次赋一个值

数组名[下标]=变量值

# array1[0]=pear

# array1[1]=apple

# array1[2]=orange

# array1[3]=peach

方法二： 一次赋多个值

# array2=(tom jack alice)

# array3=(`cat /etc/passwd`) //希望是将该文件中的每一个行作为一个元数赋值给数组

# array4=(`ls /var/ftp/Shell/for\*`)

# array5=(tom jack alice "bash shell")

# colors=($red $blue $green $recolor)

# array5=(1 2 3 4 5 6 7 "linux shell" [20]=puppet)

1. 查看数组：

# declare -a

declare -a array1='([0]="pear" [1]="apple" [2]="orange" [3]="peach")'

declare -a array2='([0]="tom" [1]="jack" [2]="alice")'

1. 访问数组元数：

# echo ${array1[0]} //访问数组中的第一个元数

# echo ${array1[@]} //访问数组中所有元数 等同于 echo ${array1[\*]}

# echo ${#array1[@]} //统计数组元数的个数

# echo ${!array2[@]} //获取数组元数的索引

# echo ${array1[@]:1} //从数组下标1开始

# echo ${array1[@]:1:2} //从数组下标1开始，访问两个元素

1. 遍历数组：

* 方法一： 通过数组元数的个数进行遍历 (不推荐使用，也不介绍了)
* 方法二： 通过数组元数的索引进行遍历

#### 关联数组

1. 定义关联数组：

申明关联数组变量，必须使用declare -A关键字

# declare -A ass\_array1

# declare -A ass\_array2

方法一： 一次赋一个值

数组名[索引]=变量值

# ass\_array1[index1]=pear

# ass\_array1[index2]=apple

# ass\_array1[index3]=orange

# ass\_array1[index4]=peach

方法二： 一次赋多个值

# ass\_array2=([index1]=tom [index2]=jack [index3]=alice [index4]='bash shell')

1. 查看数组：

# declare -A

declare -A ass\_array1='([index4]="peach" [index1]="pear" [index2]="apple" [index3]="orange" )'

declare -A ass\_array2='([index4]="bash shell" [index1]="tom" [index2]="jack" [index3]="alice" )'

1. 访问数组元数：

# echo ${ass\_array2[index2]} //访问数组中的第二个元数

# echo ${ass\_array2[@]} //访问数组中所有元数 等同于 echo ${array1[\*]}

# echo ${#ass\_array2[@]} // 获得数组元数的个数

# echo ${!ass\_array2[@]} // 获得数组元数的索引

1. 遍历数组：

方法： 通过数组元数的索引进行遍历

自己动手：

1. 使用关联数组统计文件/etc/passwd中用户使用的不同类型shell的数量

2. 使用关联数组按扩展名统计指定目录中文件的数量

## 用实例玩函数

### 函数的作用

完成特定功能的代码片段（块）在shell中定义函数可以使用代码模块化，便于复用代码函数必须先定义才可以使用一

传参$1,$2

变量local

返回值return $?

### 定义函数

1. 方法一

函数名**() {**函数要实现的功能代码**}**

方法二

function 函数名 {

函数要实现的功能代码

}

### 调用函数

函数名

函数名参数1 参数2

### 自己动手：

#### 编写系统初始化脚本

配置YUM

安装虚拟化软件

安装cherrytree

安装ntfs-3g

按q键退出程序

#### 编写虚拟机管理脚本

安装虚拟机

关闭虚拟机

打开虚拟机

克隆虚拟机（可以指定克隆的数量）

按q键退出程序

#### 编写系统管理工具箱

查看内存的使用情况

查看磁盘的使用情况

查看系统的负载

按q键退出程序

......

注：

1. 所有脚本必须有选择菜单
2. 所有的危险操作均需给出提示

## 非交互式excepct函数

# 正则表达式

## 正则表达式中的元字符

正则表达式（regular expression, RE）是一种字符模式，用于在查找过程中匹配指定的字符。在大多数程序里，正则表达式都被置于两个正斜杠之间；例如/l[oO]ve/就是由正斜杠界定的正则表达式，它将匹配被查找的行中任何位置出现的相同模式。在正则表达式中，元字符是最重要的概念。

所谓的元字符，就是一些有特殊含义的字符，比如ro**\*t**中的 **\***就是表示前面的字符o出现0到多次（可以一次都不出现）的意思。如果要查找字符串中的“\*”符号，则需要对“\*”进行转义，即在其前加一个“\”，如ro\\*t最终匹配“ro\*t”。这些具有特殊含义的字符，我们统称“元字符”。

正则表达式的作用

1. Linux正则表达式 grep,sed,awk
2. 大量的字符串文件需要进⾏配置，⽽且是非交互式的
3. 过滤相关的字符串，匹配字符串，打印字符串

正则表达式注意事项

1. 正则表达式应用⾮常广泛，存在于各种语言中，例如：php,python,java等。
2. 正则表达式和通配符特殊字符是有本质区别的。
3. 要想学好 grep、sed、awk 首先就要掌握正则表达式。

### 基础正则表达式

元字符具备特殊的含义，不能直接显示或匹配，若要显示匹配这些特殊字符，必须首先使字符"转义"，即，将反斜杠字符“\”放在它们前面。下表列出了正则表达式中的特殊字符：

#### 基础符号

|  |  |  |
| --- | --- | --- |
| **字符** | **含义** | **描述** |
| **$** | 匹配行尾 | 匹配输入字符串的结尾位置。如果设置了 RegExp 对象的Multiline 属性，则 $ 也匹配 '\n' 或 '\r'。要匹配 $ 字符本身，请使用 \$。 |
| **()** | 创建子表达式 | 标记一个子表达式的开始和结束位置。子表达式可以获取供以后使用。要匹配这些字符，请使用 \( 和 \)。 |
| **.** | 任意一个字符 | 匹配除换行符 \n 之外的任何单字符。要匹配 . ，请使用 \. 。 |
| **[** | 任选一个字符 | 标记一个中括号表达式的开始。要匹配 [，请使用 \[。 |
| **\** | 转义符 | 将下一个字符转义，标记为或特殊字符、或原义字符、或向后引用、或八进制转义符。例如，'n' 匹配字符'n'。'\n' 匹配换行符。序列 '\\' 匹配 "\"，而 '\(' 则匹配 "("。 |
| **^** |  |  |
| **|** |  | 意思为或，指明两项或多项之间的一个选择。要匹配 |，请使用 \|。 |

#### 限定符

限定符用来指定一个给定组件必须要出现多少次才能满足匹配。它们不能独立存在，必须在前面有相应的“原字符或表达式”。限定符总共有4种：\*、+、?、{}，其中{}又有3种形式：{n}、{n,}、{n,m}，最终出现6种。

|  |  |  |
| --- | --- | --- |
| **字符** | **含义** | **描述** |
| **\*** | 匹配到任意次 | 匹配前面的子表达式零次或多次。要匹配 \* 字符，请使用 \\*。例如，zo\* 能匹配 "z" 以及 "zoo"。\*等价于{0,}。 |
| **+** | 匹配1到任意多次 | 匹配前面的子表达式一次或多次。要匹配 + 字符，请使用 \+。例如，'zo+' 能匹配 "zo" 以及 "zoo"，但不能匹配 "z"。+ 等价于 {1,}。 |
| **?** | 匹配0到1次 | 前面的子表达式零次或一次。要匹配 ？ 字符，请使用 \？。例如，"do(es)?" 可以匹配 "do" 、 "does" 中的 "does" 、 "doxy" 中的 "do" 。? 等价于 {0,1}。 |
| **{n}** | 重复次数 | n 是一个非负整数。匹配确定的 n 次。例如，'o{2}' 不能匹配 "Bob" 中的 'o'，但是能匹配 "food" 中的两个 o。 |
| **{n,}** | 至少重复次数 | n 是一个非负整数。至少匹配n 次。例如，'o{2,}' 不能匹配 "Bob" 中的 'o'，但能匹配 "foooood" 中的所有 o。'o{1,}' 等价于 'o+'。'o{0,}' 则等价于 'o\*'。 |
| **{n,m}** | 指定重复次数的范围 | m 和 n 均为非负整数，其中n <= m。最少匹配n次且最多匹配m次。例如，"o{1,3}" 将匹配 "fooooood" 中的前三个 o。'o{0,1}' 等价于 'o?'。请注意在逗号和两个数之间不能有空格。 |

**注意：**

**1） \*、+限定符都是贪婪的，因为它们会尽可能多的匹配文字，只有在它们的后面加上一个?就可以实现非贪婪或最小匹配。**

2） 限定符不能独立存在，它们出现在范围表达式之后,应用于整个范围表达式。

#### 定位符

定位符使您能够将正则表达式固定到行首或行尾。它们还使您能够创建这样的正则表达式，这些正则表达式出现在一个单词内、在一个单词的开头或者一个单词的结尾。

定位符用来描述字符串或单词的边界，^ 和 $ 分别指字符串的开始与结束，\b 描述单词的前或后边界，\B 表示非单词边界。

正则表达式的定位符有：

|  |  |  |
| --- | --- | --- |
| **字符** | **含义** | **描述** |
| **^** | 匹配行首 | 匹配输入字符串开始的位置。如果设置了 RegExp 对象的 Multiline 属性，^ 还会与 \n 或 \r 之后的位置匹配。 |
| **$** | 匹配行尾 | 匹配输入字符串结尾的位置。如果设置了 RegExp 对象的 Multiline 属性，$ 还会与 \n 或 \r 之前的位置匹配。 |
| **\b** | 单词边界 | 匹配一个单词边界，即字与空格间的位置。 |
| **\B** | 单词中任意位置 | 非单词边界匹配。 |

**注意**：不能将限定符与定位符一起使用。由于在紧靠换行或者单词边界的前后不能有一个以上位置，因此不允许诸如“^\*”之类的表达式。

#### 扩展字符

|  |  |  |
| --- | --- | --- |
| **字符** | **含义** | **描述** |
| **>[[:space:]]** | 空格 | 空格 |
| **[[:digit:]]** | 数字 | 等价于：[0-9] |
| **[[:lower:]]** | 小写字母 | 等价于：[a-z] |
| **[[:upper:]]** | 大写字母 | 等价于：[A-Z] |
| **[[:alpha:]]** | 所有字母 | 等价于：[a-Z] |

### 正则表达式实战

如果grep匹配后不显示颜色，请执行以下命令：

# vim /etc/bash.bashrc

在文件最后加入如下两条命令：

alias egrep="egrep --color"

alias grep="grep --color"

然后生重新载入一下bashr 配置文件：

# source /etc/bash.bashrc

测试一下：

# ifconfig eth0 | grep "inet" | grep -E "[0-9]{1,3}.[0-9]{1,3}."



# ifconfig eth0 | grep "inet" | grep -E "[0-9]{1,3}\.[0-9]{1,3}\."



开始疯狂的练习吧！！！

# cat /test

I am v-dragrn!

I learning linux.

test

I like badminton ball ,billiard ball and chinese chess!

my blog **is** http:*//liangweilinux.blog.51cto.com*

our site **is** http:*//www.xuliangwei.com*

my qq “过客” **is** 109614185

not 1096141888885.

//过滤以m开头的行

# grep "^m" test

//排除空行, 并打印行号

# grep -vn "^$" test

//匹配任意一个字符,不包括空行

# grep "." test

//.匹配所有

# grep ".\*" test

//匹配单个任意字符

# grep "xuliangw.i" test

//以点结尾的

# grep "\.$" test

//精确匹配到

# grep -o "8\*" test

//匹配有abc的行

# grep "[abc]" test

//匹配数字所在的行"[^0-9]"

# grep "[0-9]" test

//匹配所有⼩写字母[^a-z]

# grep "[a-z]" test

//重复0三次

# grep "0\{3\}" test

//重复3个000不用转义符

# grep -E "(000){3}" test

//重复数字8, 3-5次

# grep -E "8{3,5}" test.txt

//至少1次或1次以上

# grep -E "8{1,}" test

# nmcli device show | egrep "[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}"

IP4.ADDRESS[1]: 192.168.44.10/24

IP4.ADDRESS[2]: 192.168.44.128/24

IP4.GATEWAY: 192.168.44.2

IP4.ROUTE[1]: dst = 192.168.44.0/24, nh = 0.0.0.0, mt = 100

IP4.ROUTE[2]: dst = 192.168.44.0/24, nh = 0.0.0.0, mt = 100

IP4.ROUTE[3]: dst = 0.0.0.0/0, nh = 192.168.44.2, mt = 100

IP4.DNS[1]: 192.168.44.2

IP4.WINS[1]: 192.168.44.2

IP4.ADDRESS[1]: 192.168.220.236/24

IP4.GATEWAY: 192.168.220.1

IP4.ROUTE[1]: dst = 0.0.0.0/0, nh = 192.168.220.1, mt = 101

IP4.ROUTE[2]: dst = 192.168.220.0/24, nh = 0.0.0.0, mt = 101

IP4.DNS[1]: 202.106.0.20

IP4.DNS[2]: 114.114.114.114

IP4.ADDRESS[1]: 192.168.220.237/24

IP4.GATEWAY: 192.168.220.1

## sed文本处理器

sed是一个流编辑器,非交互式的编辑器,它一次处理一行内容。如果传统的文本编辑器打开一个文本时，是将整个文本一次性全部载入内存，假如要打开一个20G的日志文件，将会是一个漫长的过程，并且可能导致服务器停止响应。而基于sed文本处理器,不关心文件的大小，它一次只载入一行进行处理，处理完毕后再载入下一行，所以永远不用担心文件大小的问题。

1. sed处理文件时，把当前处理的行存储在临时缓冲区中，称为“模式空间”（pattern space）
2. 接着用 sed 命令处理缓冲区中的内容，处理完成后，把缓冲区的内容送往屏幕。
3. 接着处理下⼀行，这样不断重复，直到文件末尾。

Sed在编辑文件时，文件内容并没有改变，除非你使用重定向选项（-i）存储输出，原文件才会被修改。

sed要用来自动编辑一个或多个文件；简化对文件的反复操作；编写转换程序等。

### sed命令格式

sed [options] 'command' file(s)

### sed正则使用

与 grep⼀样，sed 在文件中查找模式时也可以使用正则表达式(RE)和各种元字符。正则表达式是括在斜杠间，用于查找和替换，以下是sed⽀持的元字符。

使用基本元字符集 , $, ., \*, [], [], < >, (), {}

使用扩展元字符集 ?, +, { }, |, ( )

使用扩展元字符的方式 + sed -r

### sed命令示例

sed选项参数

-e 允许多项编辑

-n 取消默认的输出

-i 直接修对应文件

-r 支持扩展元字符

sed 命令参数

a 在当前行后添加一行或多行

c 在当前行进行替换修改

d 在当前行进行删除操作

i 在当前行之前插入文本

p 打印匹配的行或指定行

n 读入下一输入行，从下和条命令进行处理

! 对所选行以外的所有行应用命令

h 把模式空间里的内容重定向到暂存缓冲区

H 把模式空间里的内容追加到暂存缓冲区

g 取出暂存缓冲区的内容，将其复制到模式空间，覆盖该处原有内容

G 取出暂存缓冲区的内容，将其复制到模式空间，追加在原有内容后面

多重编辑选项 e

//先删除行,然后管道给后面的sed进行替换

# sed '1,9d' passwd |sed 's#root#alex#g'

//使用-e进行多次编辑修改操作,注意不需要管道了

# sed -e '1,9d' -e 's#root#alex#g' passwd

打印命令 p

//打印匹配halt的行

# sed -n '/halt/p' passwd

halt:x:7:0:halt:/sbin:/sbin/halt

//打印第二行的内容

# sed -n '2p' passwd

bin:x:1:1:bin:/bin:/sbin/nologin

//打印最后一行

# sed -n '$p' passwd

追加命令 a

//给30行添加配置 \t tab键(需要转义) \n 换行符

# sed -i '30a listen 80;' passwd

修改命令 c

//指定某行进行内容替换

# sed -i '7c SELINUX=Disabled' /etc/selinux/config

//正则匹配对应内容, 然后进行替换

# sed -i '/^SELINUX=/cSELINUX=Disabled' /etc/selinux/config

//非交互式修改指定的配置文件

# sed -ri '/UseDNS/cUseDNS no' /etc/ssh/sshd\_config

# sed -ri '/GSSAPIAuthentication/c#GSSAPIAuthentication no' /etc/ssh/

sshd\_config

# sed -ri '/^SELINUX=/cSELINUX=disabled' /etc/selinux/config

删除命令 d

//指定删除第三行, 但不会改变文件内容

# sed '3d' passwd

# sed '3{d}' passwd

//从第三行删除到最后一行

# sed '3,$d' passwd

//删除最后一行

# sed '$d' passwd

//删除所有的行

# sed '1,$d' passwd

//匹配正则进行该行删除

# sed /mail/d passwd

插⼊命令 i

//在文件的某一行上面添加内容

# sed -i '30i listen 80;' passwd

写文件命令 w

*//*将匹配到的行写入到新文件中

# sed -**n** '/root/w newfile' passwd

*//*将*passwd*文件的第⼆二行写入到*newfile*中

# sed -**n** '2w newfile' passwd

获取下一行命令 n

*//*匹配root的行, 删除root行的下一列

# sed '/root/{n;d}' passwd

//替换匹配root行的下一列

# sed '/root/{n; s/bin/test/}' passwd

暂存和取用命令 h H g G

//将第一行的写入到暂存区, 替换最后一行的内容

# sed '1h;$g' /etc/hosts

//将第一行的写入到暂存区, 在最后一行调用暂存区的内容

# sed '1h;$G' /etc/hosts

//将第一行的内容删除但保留至暂存区, 在最后一行调用暂存区内容追加至于尾部

# sed -r '1{h;d};$G' /etc/hosts

//将第一行的内容写⼊至暂存区, 从第二行开始进行重定向替换

# sed -r '1h;2,$g' /etc/hosts

//将第一行重定向至暂存区, 2-3行追加至暂存区, 最后追加调用暂存区的内容

# sed -r '1h; 2,3H; $G' /etc/hosts

反向选择命令 !

//除了第三行,其他全部删除

# sed -r '3!d' /etc/hosts

**sed**匹配替换

s 替换命令标志

g 行内全局替换

I 忽略替换大小写

替换命令 s

//替换每行出现的第一个root

# sed 's/root/alice/' passwd

//替换以root开头的行

# sed 's/^root/alice/' passwd

//查找匹配到的行,在匹配的行后面添加内容

# sed -r 's/[0-9][0-9]$/& .5/' passwd

//匹配包含有root的行进行替换

# sed -r 's/root/alice/g' passwd

//匹配包含有root的行进行替换,忽略大小写

# sed -r 's/root/alice/gi' /etc/passwd

//后向引用

# sed -r 's#(Roo)#\1-alice#g' passwd

# ifconfig eth0|sed -n '2p'|sed -r 's#(^.\*et) (.\*) (net.\*$)#\2#g'

//示例

# vim a.txt

/etc/abc/456

etc

//删除文本中的内容,需加转义

# sed -r '\/etc\/abc\/456/d' a.txt

//如果碰到/符号, 建议使用#符替换

# sed -r 's#/etc/abc/456#/dev/null#g' a.txt

# sed -r 's@/etc/abc/456@/dev/null@' a.txt

删除文件

//删除配置文件中#号开头的注释行, 如果碰到tab或空格将无法删除

# sed '/^#/d' file

//删除配置文件中含有tab键或空格的注释行

# sed -r '/^[ \t]\*#/d' file

//删除无内容空行

# sed -r '/^[ \t]\*$/d' file

//删除注释行及空行

# sed -r '/^[ \t]\*#/d; /^[ \t]\*$/d' /etc/vsftpd/vsftpd.conf

# sed -r '/^[ \t]\*#|^[ \t]\*$/d' /etc/vsftpd/vsftpd.conf

# sed -r '/^[ \t]\*($|#)/d' /etc/vsftpd/vsftpd.conf

给文件行添加注释

//将第二行到第六行加上注释信息

# sed '2,6s/^/#/' passwd

//将第二行到第六行最前面添加#注释符

# sed -r '2,6s/.\*/#&/' passwd

//添加#注释符

# sed -r '3,$ s/^#\*/#/' passwd

# sed -r '30,50s/^[ \t]\*#\*/#/' /etc/nginx.conf

# sed -r '2,8s/^[ \t#]\*/#/' /etc/nginx.conf

## awk文本处理工具

awk是一个专门用于处理文本的**编程语言**，之所以叫awk，是因为取了三位创始人[Alfred Aho](http://en.wikipedia.org/wiki/Alfred_Aho)、[Peter Weinberger](http://en.wikipedia.org/wiki/Peter_J._Weinberger)和[Brian Kernighan](http://en.wikipedia.org/wiki/Brian_Kernighan)的Family Name的首字符。awk是贝尔实验室1977年搞出来的文本编辑神器，我估计80后的年轻朋友可能对awk/sed这类上古神器有点陌生了，但是越是经典的老东西，干货越多，越能学到其中的精髓。时经40多年，awk依然是所有Linux发行版本默认集成的最重要的工具之一，在基于命令行的文本处理工具中，还没有取代它的工具。

awk 的处理文本和数据的方式如下:

1.逐行扫描文件，从第一行扫描到最后一行

2.寻找匹配特定模式的行,在行上进行操作

3.如果没有指定处理动作,则把匹配的行显示到标准输出

4.如果没有指定模式，则所有被操作的行都被处理

### awk 的两种形式语法格式

awk [options] 'commands' filenames

awk [options] -f awk-script-file filenames

options

# awk 'BEGIN{print 1/2} {print "ok"} END{print "Game Over"}' /etc/hosts

0.5

ok

ok

ok

ok

ok

ok

ok

Game Over

# cat /etc/hosts

127.0.0.1 localhost

127.0.1.1 kali2018-2.localdomain kali2018-2

# The following lines are desirable for IPv6 capable hosts

::1 localhost ip6-localhost ip6-loopback

ff02::1 ip6-allnodes

ff02::2 ip6-allrouters

想想，办什么打印7个ok？因为/etc/hosts文件有7行（含空行）。

### awk命令格式

//awk 'pattern' filename 匹配文件

# awk '/root/' /etc/passwd

//awk '{action}' filename 对文件行进行动作处理

# awk -F: '{print $1}' /etc/passwd

//awk 'pattern {action}' filename 匹配+处理动作

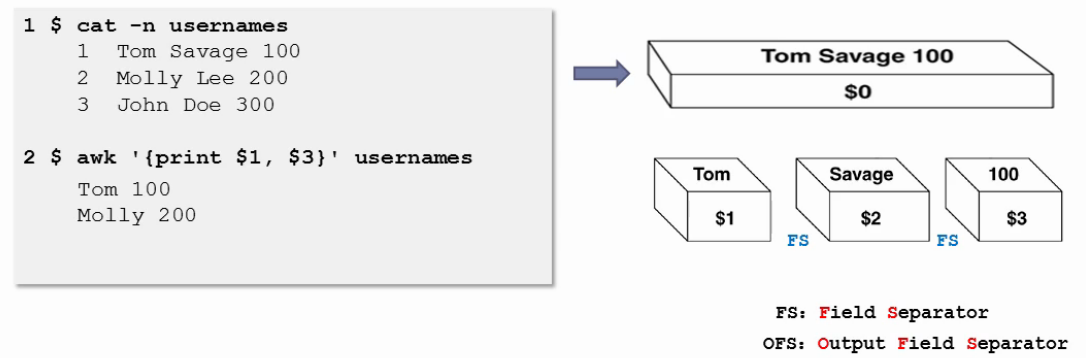
# awk -F ':' '/root/ {print $1,$3}' /etc/passwd

# awk 'BEGIN{FS=":"} /root/{print $1,$3}' /etc/passwd

//command |awk 'pattern {action}' //判断大于多少则输出什么内容

# df |awk '/\/$/ {if ($3>50000) print $4}'

### awk工作原理

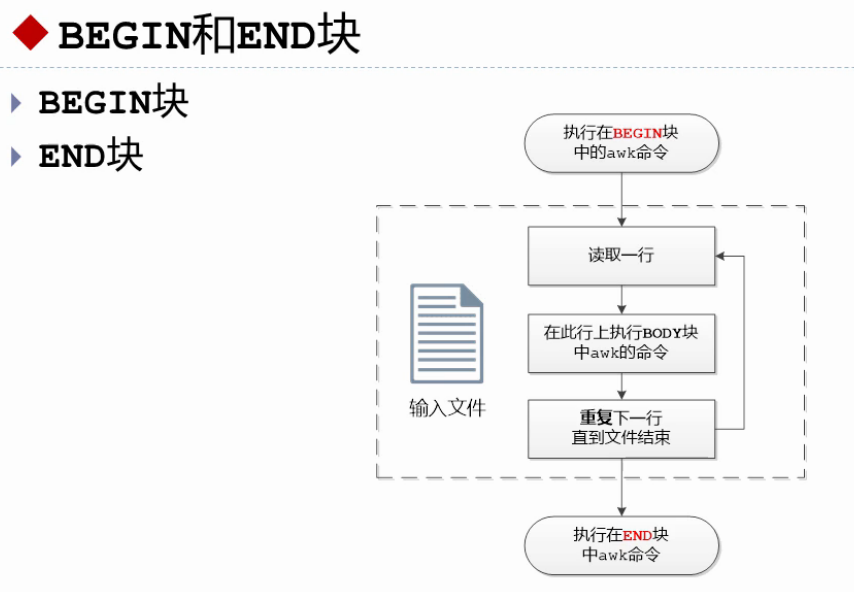


# awk -F: '{print $1,$3}' /etc/passwd

-F 定义输⼊字段分隔符，默认的分隔符是空格或tab键

BEGIN{} {} END{}

行处理前 行处理 行处理后



1.awk使用一行作为输入，并将这一行赋给内部变量 $0 ,每一行也可称为一个记录，以换行符结束;

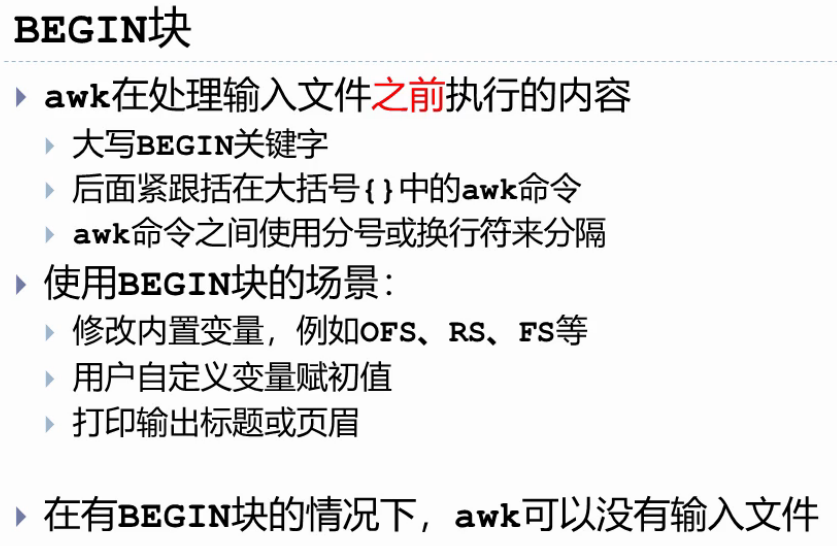
2.awk进行字段分解，每个字段存储在已编号的变量中，从 $1 开始;

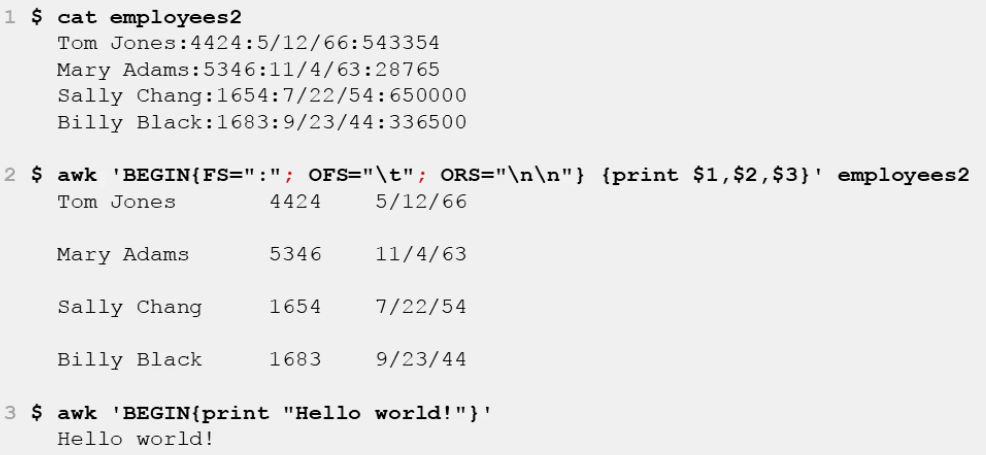
3.awk默认情况下的分隔符是空格,由内部变量 FS 来确定字段分隔符。初始 FS 为空格或tab

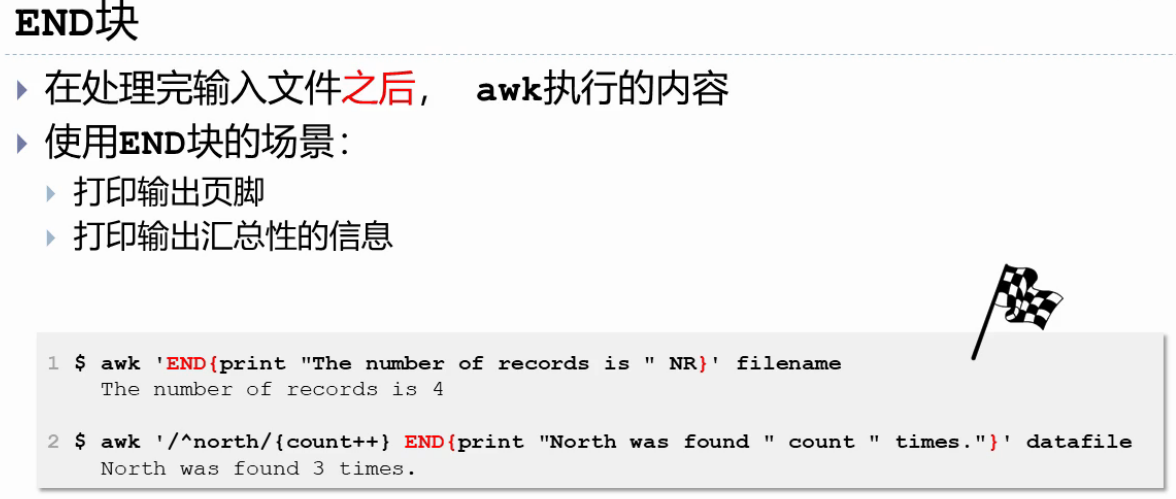
4.awk打印字段,将以设置的方法使用 print 函数打印

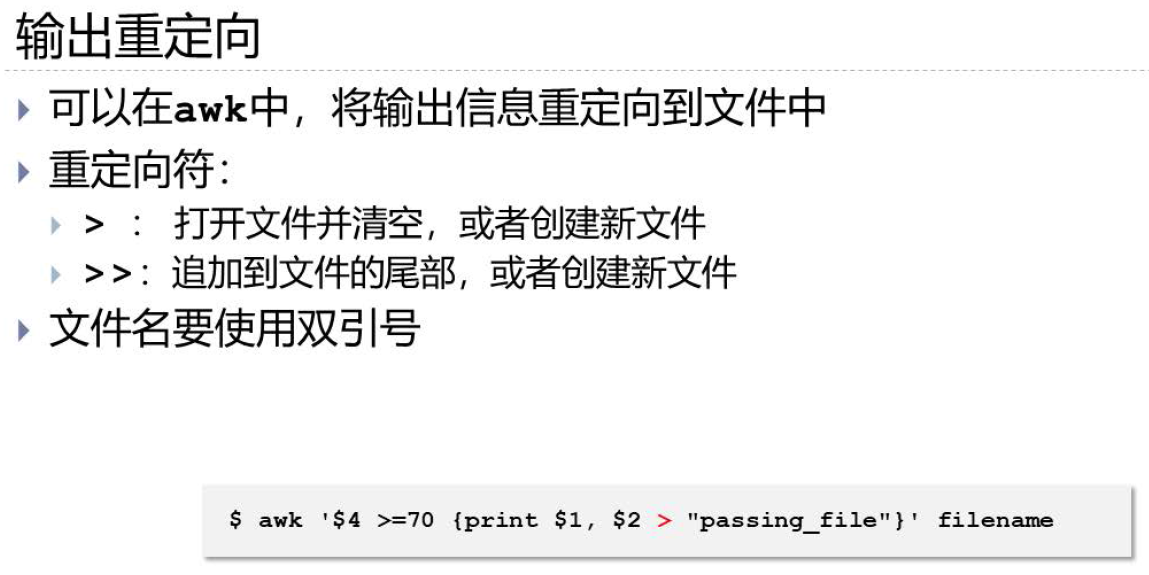
5.awk在打印的字段间加上空格，因为 $1,$3 之间有一个逗号。逗号它映射为另一个内部变量，称为输出字段分隔符 OFS 默认为空格

6.awk输出之后，将从文件中获取另一行，并将其存储在 $0 中，覆盖原来的内容，然后将新的字符串分隔成字段并进行处理。该过程将持续到所有行处理完毕











### awk的输出格式

1. 检出格式变量：OFMT、OFS、ORS

OFMT数字的输出格式，默认值是%.6g output format for umbers

OFS输出字段分隔符，默认是一个空格 output field separator

ORS输出记录分隔符，默认是一个换行符 output record separator

# echo 1.123456789|awk '{print $1; print 1.123456789; print OFMT}'

1.123456789

1.12346

%.6g

# echo 1.123456789|awk '{OFMT="%.2f";print $1; print 1.123456789}'

1.123456789

1.12

# echo -e "a b c\n1 2 3"|awk '{OFS="-"; ORS="\*"; print $1, $2, $3}'

a-b-c\*1-2-3\*root

* 上例中的内置变量$1默认为字符型，所以全部输出
* 打印“1.123456789”没有用双引号，默认为数值，%.6g表示显示6位有效果数字，所以显示6位数字（含整数），并且进行了四舍五入；
* 第二条命令中，将OFMT指定为%.2f，其中的“2f”表示两位小数；
* 第三条命令中重新指定了OFS为“-”，ORS为“\*”。

1. print命令

# date

2018年 10月 14日 星期日 18:22:02 CST

# date | awk '{print "Month: " $2 "\nYear: ", $1}'

Month: 10月

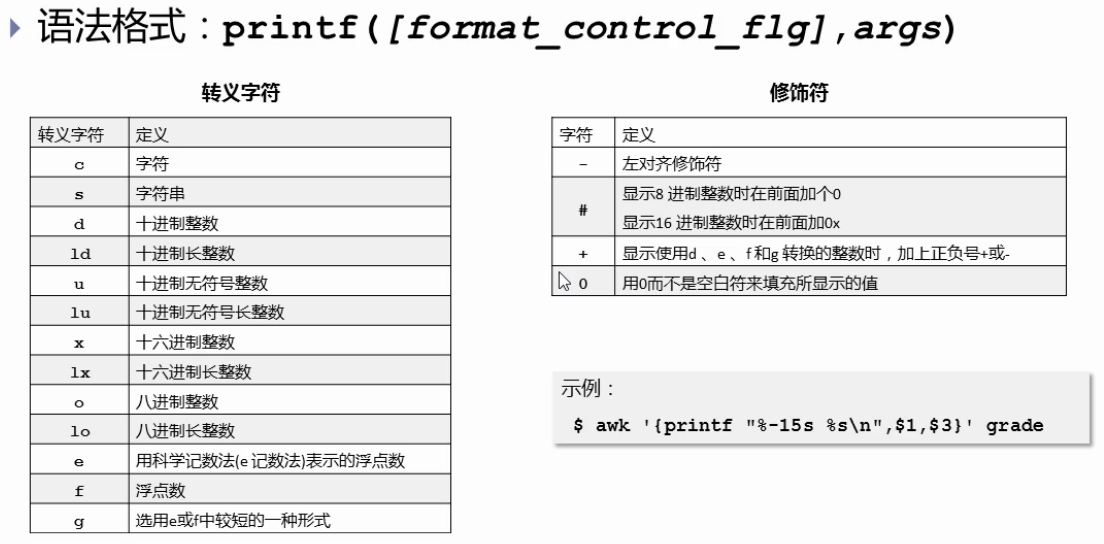
Year: 2018年



上面是print命令中的元字符，不用死记硬背，可用man手册查看。

1. printf函数

在有些命令中，print命令无法满足格式需求，这时就可以用到printf函数。



# awk '{printf "%-15s %s\n" , $1, $3}' /usr/scripts/test/usernames

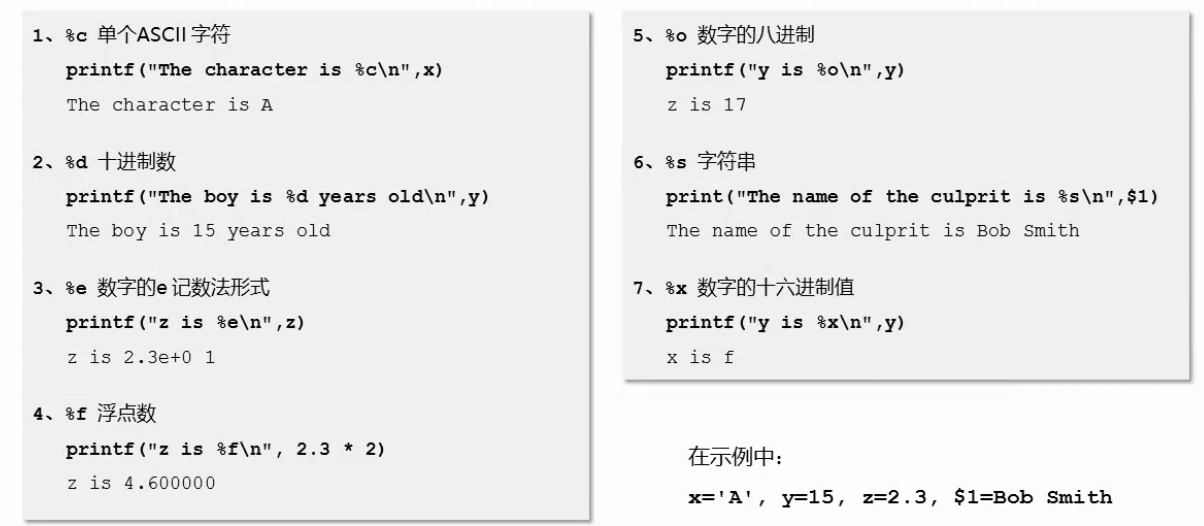
Tom 100

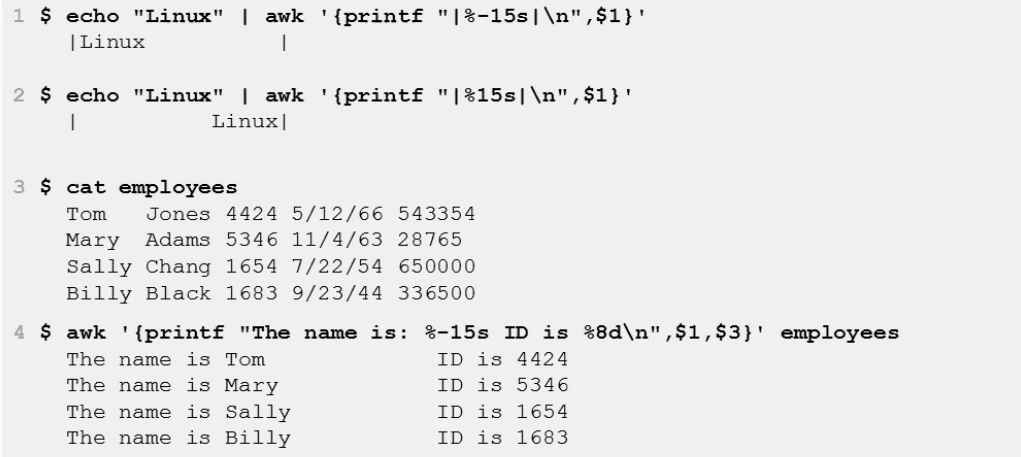
Molly 200

John 300

* %-15”中的“-”表示左对齐并符占15个字符宽度，“%15”表示占15个字符宽度且右对齐;
* Printf函数不会自动换行，所以要手工加一个换行符“\n”。

Printf函数的示例：





### awk内部变量

$0 保存当前记录（行）的内容

# awk '{print $0}' /etc/passwd

NR 控制输⼊的总记录数（即行数）

//打印整个文本的行号

# awk '{print NR,$0}' /etc/passwd

//打印文件的前三行

# awk 'NR<=3' /etc/passwd

FNR 记录输入文件的编号

# awk '{print FNR,$0}' /etc/passwd /etc/hosts

NF 保存每一条记录的字段数

# awk -F ":" '{print $1,$NF}' /etc/passwd

FS 指定字段分割符,默认空格

//以冒号作为字段分隔符

# awk -F: '/root/{print $1, $3}' /etc/passwd

# awk 'BEGIN{FS=":"} {print $1,$3}' /etc/passwd

//以空格冒号tab作为字段分割

# awk -F'[ :\t]' '{print $1,$2,$3}' /etc/passwd

OFS 输出字段分隔符

//符号,映射为OFS,初始情况下OFS变量是空格

# awk -F: '/root/{print $1,$2,$3,$4}' /etc/passwd

# awk 'BEGIN{FS=":"; OFS="+++"} /^root/{print $1,$2,}' /etc/passwd

RS 输入记录分隔符，默认为换行符

# awk -F: 'BEGIN{RS=" "} {print $0}' /etc/hosts

ORS 将文件每一行合并为一行,以空格为分割

# awk -F: 'BEGIN{ORS=" "} {print $0}' /etc/hosts

//通常情况下让输出分隔符为换行符, 然后依次打印响应的字段

# awk -F ":" 'BEGIN{OFS="\n"}{print $1,$2,$3,$4,$5,$6,$7}' passwd

//将RS记录值标记为冒号分割, 打印后相当于将每行的内容进行切割

# awk 'BEGIN{RS=":"}{print $0}' passwd

print 格式化输出函数

# date|awk '{print $2,"5⽉月份""\n",$NF,"今年"}'

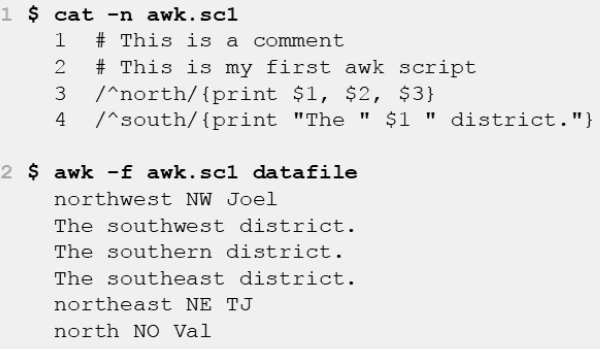
# awk -F: '{print "用户是:" $1 "\t 用户uid: " $3 "\t 用户gid:" $4}' /etc/passwd

printf 函数

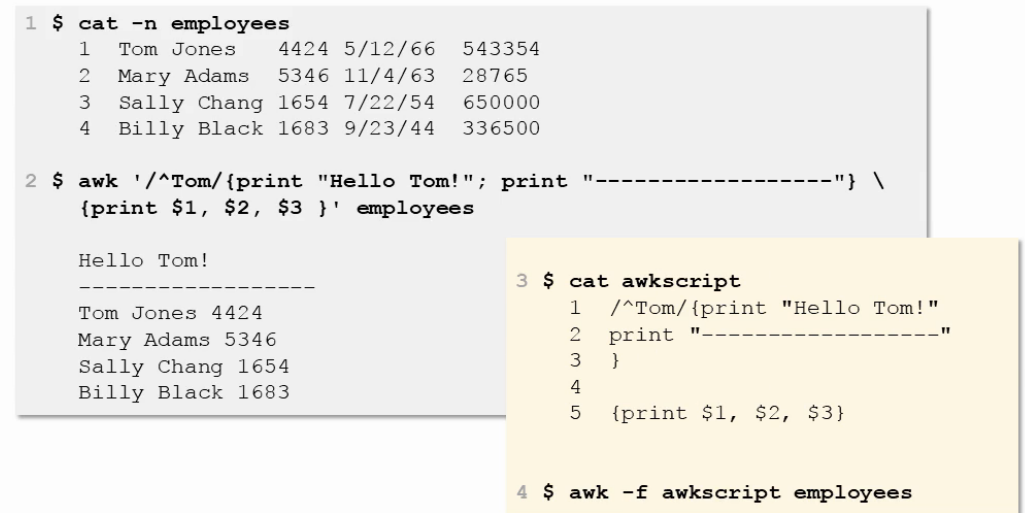
# awk -F: '{printf "%-15s %-10s %-15s\n", $1, $2, $3}' /etc/passwd

* %s 字符类型
* %d 数值类型
* 15 占 15 字符
* 表示左对齐，默认是右对齐
* Printf 默认不会在行尾自动换行，加\n

### 缩写awk脚本



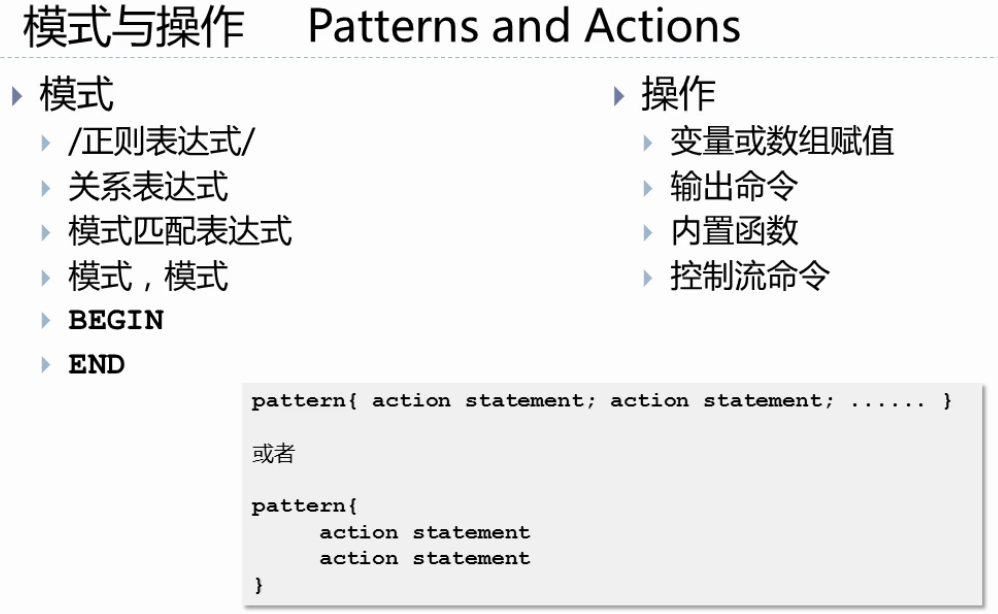
### Awk调用外部脚本



### awk模式和动作（操作）

awk语句都由模式和动作组成。模式和动作都是可选的，可以没有，如果模式和动作都没指定，那么awk将默认执行print命令。

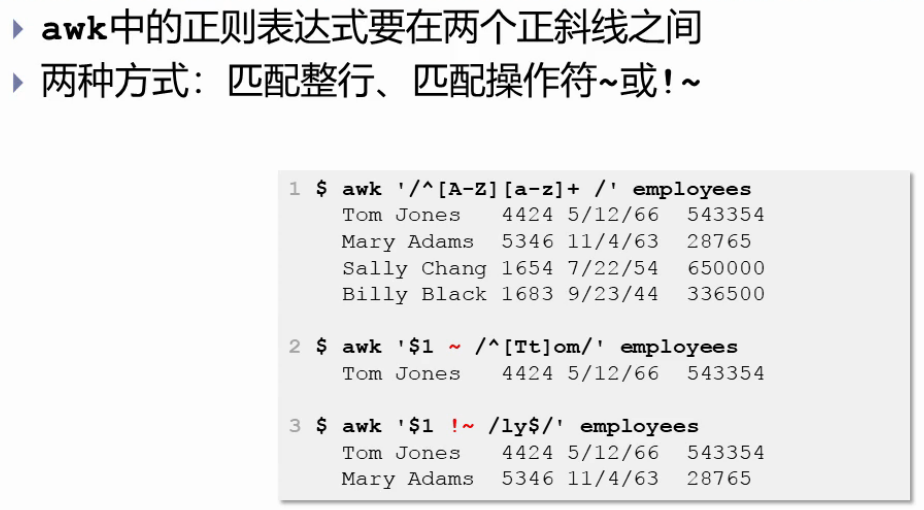
可以简单把模式理解为条件，动作理解成操作。



模式部分决定动作语句何时触发及触发事件。

如果省略模式部分，动作将时刻保持执行状态。模式可以是条件语句、复合语句或正则表达式。

#### 正则表达式



//匹配记录（整行）

# awk '/^root/' /etc/passwd //匹配整行

# awk '$0 ~ /^root/' /etc/passwd //$0表示整行，匹配整行

//匹配字段：匹配操作符（~ !~）

# awk '$1 ~ /^root/' /etc/passwd //第一个字段以 root开头的行

# awk '!/^root/' /etc/passwd //第一个字段，不是root开头的行

# awk '$1 ~ !/^root/' /etc/passwd //第一个字段，不是root开头的行（等价于上一条）

#### 比较表达式

比较表达式采用对文本进行比较，只有当条件为真，才执行指定的动作。

比较表达式使用关系运算符，用于比较数字与字符串。

关系运算符：

|  |  |  |
| --- | --- | --- |
| **算符** | **含义** | **示例** |
| < | 小于 | x<y |
| <= | 小于或等于 | x<=y |
| == | 等于 | x==y |
| != | 不等于 | x!=y |
| >= | 大于等 | x>=y |
| > | 大于 | x>y |

示例：

//uid为0的列出来

# awk -F ":" '$3==0' /etc/passwd

//uid小于10的全部列出来

# awk -F: '$3 < 10' /etc/passwd

//用户登陆的shell等于/bin/bash

# awk -F: '$7 == "/bin/bash" ' /etc/passwd

//第一列为alice的列出来

# awk -F: '$1 == "alice" ' /etc/passwd

//为alice的用户列出来

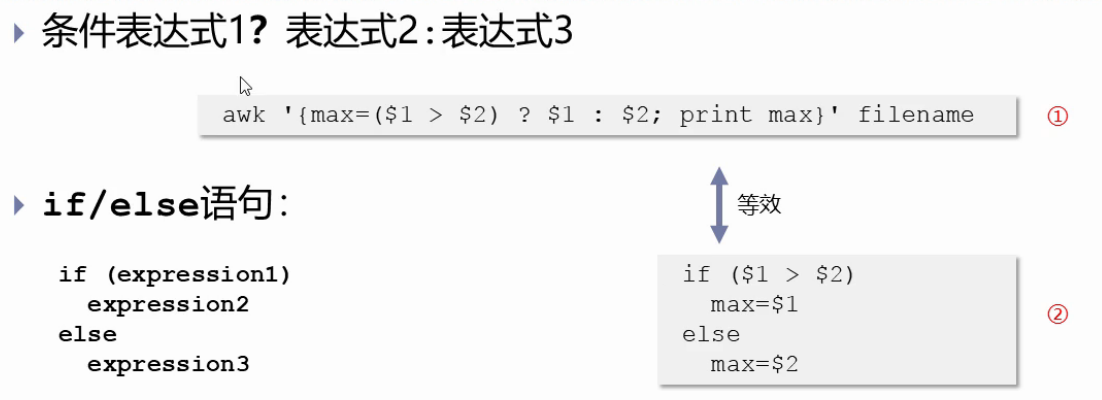
# awk -F: '$1 ~ /alice/ ' /etc/passwd

# awk -F: '$1 !~ /alice/ ' /etc/passwd

//磁盘使用率大于多少则，则打印可用的值

# df |awk '/\/$/'|awk '$3>1000000 {print $4}'

#### 条件表达式



# awk -F: '$3>300 {print $0}' /etc/passwd

# awk -F: '{if($3>300) print $0}' /etc/passwd

# awk -F: '{if($3>5555){print $3} else {print $1}}' /etc/passwd

#### 运算表达式



# awk -F: '$3 \* 10 > 500000' /etc/passwd

# awk -F: 'BEGIN{OFS="--"} {if($3\*10>50000) {print $1,$3}} END {print "打印ok"}' etc/passwd

# awk '/southem/{print $5 + 10}' datafile

# awk '/southem/{print $5 + 10.56}' datafile

# awk '/southem/{print $8 - 10}' datafile

# awk '/southem/{print $8 / 2 }' datafile

# awk '/southem/{print $8 \* 2 }' datafile

# awk '/southem/{print $8 % 2 }' datafile

#### 逻辑操作符和复合模式



&&逻辑与 || 逻辑或 !逻辑非

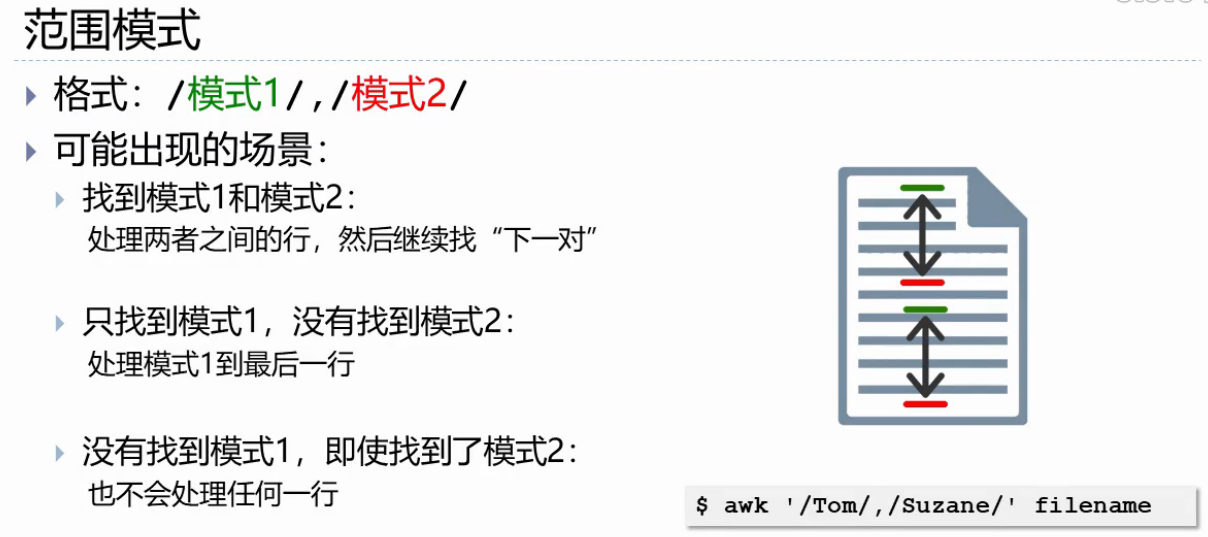
//匹配用户名为root并且打印uid小于15的行

# awk -F: '$1~/root/ && $3<=15' /etc/passwd

//匹配用户名为root或uid大于5000

# awk -F: '$1~/root/ || $3>=5000' /etc/passwd

### awk的范围模式



### awk示例

#### awk示例1

# awk '/west/' datafile

# awk '/^north/' datafile

# awk '$3 ~ /^north/' datafile

# awk '/^(no|so)/' datafile

# awk '{print $3,$2}' datafile

# awk '{print $3 $2}' datafile

# awk '{print $0}' datafile

# awk '{print "Number of fields: "NF}' datafile

# awk '/northeast/{print $3,$2}' datafile

# awk '/^[ns]/{print $1}' datafile

# awk '$5 ~ /\. [7-9]+/' datafile

# awk '$2 !~ /E/{print $1,$2}' datafile

# awk '$3 ~ /^Joel/{print $3 "is a nice boy."}' datafile

# awk '$8 ~ /[0-9][0-9]$/{print $8}' datafile

# awk '$4 ~ /Chin$/{print "The price is $" $8 "."}' datafile

# awk '/Tj/{print $0}' datafile

# awk -F: '{print "Number of fields: "NF}' /etc/passwd

# awk -F"[ :]" '{print NF}' /etc/passwd

#### awk示例2

# cat b.txt

bgx xuliangwei:is a:good boy!

# awk '{print NF}' b.txt

4

# awk -F ':' '{print NF}' b.txt

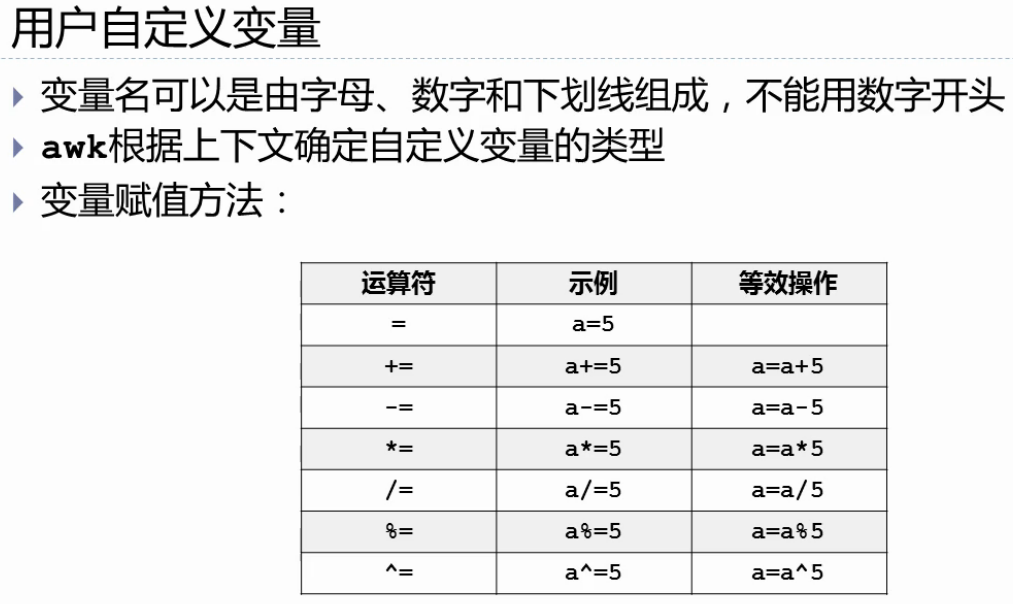
3

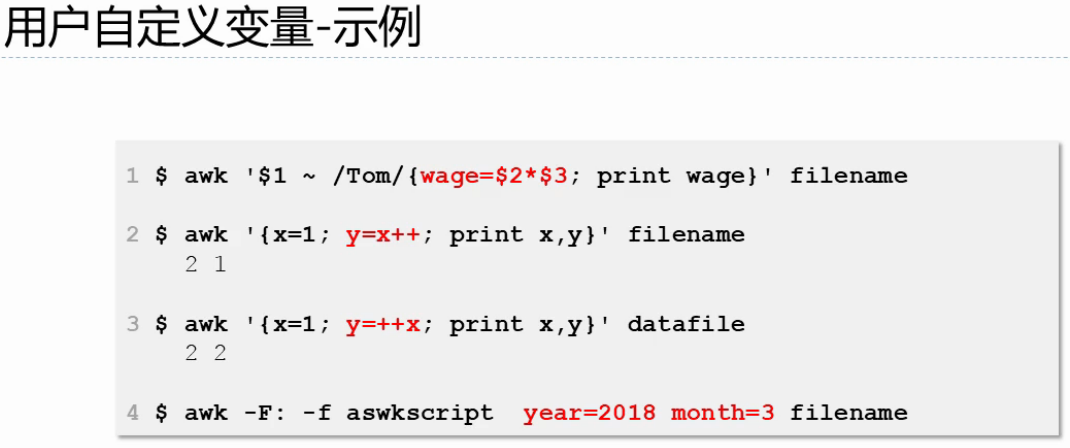
# awk -F"[ :]" '{print NF}' b.txt

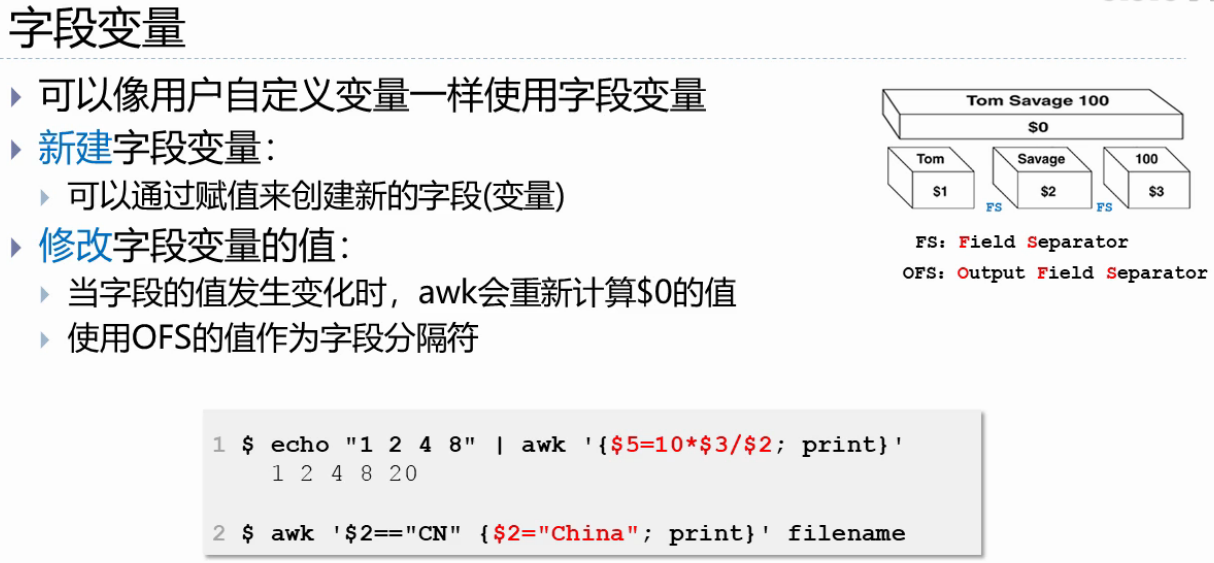
6

### awk中的变量

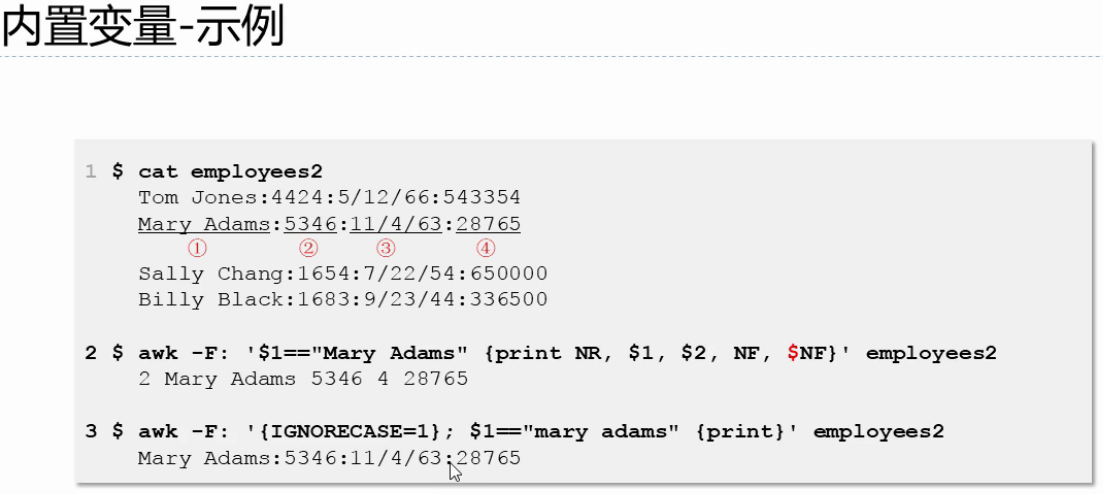












### awk条件判断

#### if语句

1. 格式一:

{ if(表达式)｛语句;语句;... ｝}

//打印当前管理员用户名称

# awk -F: '{ if($3==0){print $1 "is adminisitrator"} }' /etc/passwd

//统计系统用户数量

# awk -F: '{ if($3>0 && $3<1000){i++}} END {print i}' /etc/passwd

//统计普通用户数量

# awk -F: '{ if($3>1000){i++}} END {print i}' /etc/passwd

1. 格式二：

if...else 语句格式: {if(表达式)｛语句;语句;... ｝else{语句;语句;...}}

# awk -F: '{if($3==0){print $1} else {print $7}}' /etc/passwd

# awk -F: '{if($3==0) {count++} else{i++} }' /etc/passwd

# awk -F: '{if($3==0){count++} else{i++}} END{print " 管理员个数: "count ; print "系统用户数: "i}' /etc/passwd

1. 格式三：if...else if...else 语句格式:

{if(表达式 1)｛语句;语句；... ｝else if(表达式 2)｛语句;语句；. .. ｝else｛语句;语句；... ｝}

# awk -F: '{ if($3==0){i++} else if($3>0 && $3<1000){j++} else if($3>1000) {k++}} END {print i;print j;print k}' /etc/passwd

# awk -F: '{ if($3==0){i++} else if($3>0 && $3<1000){j++} else if($3>1000) {k++}} END {print "管理员个数"i; print "系统用户个数" j; print "系统用户个 数" }' /etc/passwd

管理员个数1

系统用户个数29

系统用户个数69

### Awk循环语句

#### while循环

# awk 'BEGIN{ i=1; while(i<=10){print i; i++} }'

# awk -F: '{i=1; while(i<=NF){print $i; i++}}' /etc/passwd

# awk -F: '{i=1; while(i<=10) {print $0; i++}}' /etc/passwd

#cat b.txt

111 222

333 444 555

666 777 888 999

# awk '{i=1; while(i<=NF){print $i; i++}}' b.txt

#### for循环

//C 风格 for

# awk 'BEGIN{for(i=1;i<=5;i++){print i} }'

//将每行打印 10 次

# awk -F: '{ for(i=1;i<=10;i++) {print $0} }' passwd

# awk -F: '{ for(i=1;i<=10;i++) {print $0} }' passwd

# awk -F: '{ for(i=1;i<=NF;i++) {print $i} }' passwd

### awk数组实战

# awk -F: '{username[++i]=$1} END{print username[1]}' /etc/passwd

# awk -F: '{username[i++]=$1} END{print username[1]}' /etc/passwd

# awk -F: '{username[i++]=$1} END{print username[0]}' /etc/passwd

注意:将需要统计的某个字段作为数组的索引，最后对索引进行遍历

#### 按索引遍历

# awk -F: '{username[x++]=$1} END{for(i in username) {print i,username[i]} }' /etc/passwd

# awk -F: '{username[++x]=$1} END{for(i in username) {print i,username[i]} }' /etc/passwd

1. 统计/etc/passwd 中各种类型 shell 的数量

# awk -F: '{shells[$NF]++} END{ for(i in shells){print i,shells[i]} }' /etc/passwd

1. 网站访问状态统计<当前时实状态ss>

# ss -an|awk '/:80/{tcp[$2]++} END {for(i in tcp){print i,tcp[i]}}'

1. 统计当前访问的每个IP的数量<当前时实状态 netstat,ss>

# ss -an|awk -F ':' '/:80/{ips[$(NF-1)]++} END {for(i in ips){print i,ips[i]}}'

### awk数组案例

Nginx 日志分析，日志格式如下：

'$remote\_addr - $remote\_user [$time\_local] "$request" ''$status $body\_bytes\_sent "$http\_referer" ''"$http\_user\_agent" "$http\_x\_forwarded\_for"';

52.55.21.59 - - [25/Jan/2018:14:55:36 +0800] "GET /feed/ HTTP/1.1" 404 162 "https://www.google.com/" "Opera/9.80 (Macintosh; Intel Mac OS X 10.6.8; U; de) Presto/2.9.168 Version/11.52" "-"

1. 统计2018年01⽉月25日,当天的PV量

# grep "25/Jan/2018" log.bjstack.log |wc -l

# awk "/25\/Jan\/2018/" log.bjstack.log |wc -l

# awk '/25\/Jan\/2018/ {ips[$1]++} END {for(i in ips) {sum+=ips[i]} {print sum}}'log.bjstack.log

//统计15-19点的pv量

# awk '$4>="[25/Jan/2018:15:00:00" && $4<="[25/Jan/2018:19:00:00 {print $0}"' log.bjstack.log |wc -l

1. 统计2018年01⽉月25日,一天内访问最多的10个IP

# awk '/25\/Jan\/2018/ {ips[$1]++} END {for(i in ips){ print ips[i],i}}' log.bjstack.log |sort -rn|head

//统计15-19点访问次数最多的10个IP

# awk '$4>="[25/Jan/2018:15:00:00" && $4<="[25/Jan/2018:19:00:00"' log.bjstack.log |awk '{ips[$1]++} END {for(i in ips){print ips[i],i}}'|sort -rn|head

1. 统计2018年01⽉月25日,访问大于100次的IP

# awk '/25\/Jan\/2018/ {ips[$1]++} END {for(i in ips){if(ips[i]>10){print i,ips[i]}}}' log.bjstack.log

1. 统计2018年01⽉月25日,访问最多的10个页面($request top 10)

# awk '/25\/Jan\/2018/ {request[$7]++} END {for(i in request){print request[i],i}}' log.bjstack.log |sort -rn|head

1. 统计2018年01⽉月25日,每个URL访问内容总大小($body\_bytes\_sent)

# awk '/25\/Jan\/2018/ {request[$7]++;size[$7]+=$10} END {for(i in request){print request[i],i,size[i]}}' log.bjstack.log |sort -rn|head

1. 统计2018年01⽉月25日,每个IP访问状态码数量($status)

# awk '{ip\_code[$1 " " $9]++} END {for(i in ip\_code){print ip\_code[i],i}}' log.bjstack.log|sort -rn|head

1. 统计2018年01⽉月25日,访问状态码为404及出现的次数($status)

# grep "404" log.bjstack.log |wc -l

# awk '{if($9=="404") code[$9]++} END {for(i in code){print i,code[i]}}' log.bjstack.log

1. 统计2018年01⽉月25日,8:30-9:00访问状态码是404

# awk '$4>="[25/Jan/2018:15:00:00" && $4<="[25/Jan/2018:19:00:00" && $9=="404" {code[$9]++} END {for(i in code){print i,code[i]}}' log.bjstack.log

# awk '$9=="404" {code[$9]++} END {for(i in code){print i,code[i]}}'log.bjstack.log

1. 统计2018年01⽉月25日,各种状态码数量

# awk '{code[$9]++} END {for(i in code){print i,code[i]}}' log.bjstack.log

# awk '{if($9>=100 && $9<200) {i++}

else if ($9>=200 && $9<300) {j++}

else if ($9>=300 && $9<400) {k++}

else if ($9>=400 && $9<500) {n++}

else if($9>=500) {p++}}

END{print i,j,k,n,p,i+j+k+n+p}' log.bjstack.log

# awk '{if($9>=100 && $9<200) {i++}

else if ($9>=200 && $9<300) {j++}

else if ($9>=300 && $9<400) {k++}

else if ($9>=400 && $9<500) {n++}

else if($9>=500) {p++}}

END{print i?i:0,j?j:0,k?k:0,n?n:0,p?p:0,i+j+k+n+p}' log.bjstack.log

注意：对于第七、八、九十章的内容必须掌握，要不然，看到一个shell脚本就会“懵”了，多看多练。有了以上的基础，再进行复杂的shell脚本的学习，就会事半功倍。目前只要掌握这么多，学习是一个循序渐进的过程，先掌握最重点的使用方法，再系统的学习，深纠原理！

# 信息收集-bash编程

我们将用大量的实例，将kali中安全工具灵活组合，形成自己独有的“神器”。