Linux Documentation

发布 ***1.0***

wenjian

2015 年 02 月 16 日

Contents

作者 闻健 wenjianxue2009@live.cn

CHAPTER 1

文章内容:

1.1 Linux 网络

主要包括 ipv4 和 ipv6 的配置；arp、ndp 的查看；route 的配置与查看。

1.1.1 IP 篇

配置 IP

1、临时配置 **ipv4** 地址 iconfig

命令格式

ifconfig < 网络接口 > <IP 地址 > [<netmask 子网掩码 > <broadcast 广播地址 >]

eg:ifconfig eth0 192.168.0.222

当 IP 地址使用标准 A、B、C 类地址时，广播地址和子网掩码可以省略，系统会自动判断广播地址和子网掩码的值并进行设置。否则必须指出广播地址和子网掩码

ifconfig eth0 10.0.0.222 Mask 255.255.255.0 Broadcast 10.0.0.255

1.1 给接口设置多个 IP

ifconfig eth0:0 192.168.0.250

ifconfig eth1:0 192.168.1.3

ifconfig eth1:1 192.168.2.3

> 警告： 使用 ifconfig 命令设置网络参数会立即生效，但不会修改网络接口配置文件，这将导致所配置的参数在重新启动系统后失效。

2、永久配置 ipv4 地址

2.1 Redhat 系列操作系统

在 /etc/sysconfig/network-scripts 目录下存储网络接口配置文件。每个网络接口有各自的配置文件，配置文件以 ifcfg- 为前缀后接网络接口名。例如，接口 eth0 的配置文件名为 ifcfg-eth0 。下面是 eth0 接口的配置文件。

```
DEVICE=eth0             # 设备名
BROADCAST=192.168.0.255 # 广播地址
HWADDR=00:0c:29:f1:15:8f # MAC 地址
IPADDR=192.168.0.100    # IP 地址
NETMASK=255.255.255.0   # 子网掩码
NETWORK=192.168.0.0     # 网络地址
ONBOOT=yes              # 在系统启动时启用该接口
GATEWAY=192.168.0.1     # 网关地址
TYPE=Ethernet           # 网络接口类型
```

您可以根据需要修改此配置文件 ifcfg-eth0 的配置。如果要设置 eth1 的配置文件，您可以复制 ifcfg-eth0 为 ifcfg-eth1 然后做适当修改，如果要是为接口配置多个 IP 地址，可以将 ifcfg-eth0 拷贝成 ifcfg-eth0:0 等，然后做相应修改

> 警告： 修改完成后记得要重启网络服务 service network restart

2.2 Debian 系列操作系统

在/etc/network/interfaces 文件中存储着各自接口的配置信息，下图是配置示意，可以根据具体环境设置

```
# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
allow-hotplug eth0
iface eth0 inet static
address 10.5.68.6
gateway 10.5.68.1
netmask 255.255.255.0


#auto eth0
#iface eth0 inet dhcp
```

3、临时配置 **ipv6** 地址 ifconfig ens33【接口名称】inet6 add 2001::4/16【ipv6 地址】

4、永久配置 **ipv6** 地址 在配置文件中输入类似于 ipv4 的设置

配置主机名

1、临时修改主机名

hostname xxxxx

或

echo xxxxx > /etc/hostname

或

hostname -F /etc/hostname

2、永久修改主机名

2.1 Redhat 系列操作系统

编辑 /etc/sysconfig/network 文件中的如下配置行：

HOSTNAME=yourhostname

将 yourhostname 修改为您的主机名。配置文件修改完毕，在下次重新启动时就会生效。

> 警告： 不要忘记还需要修改 /etc/hosts 文件中的主机名。

配置 DNS

1、修改 DNS 客户端配置文件

DNS 客户端配置文件为/etc/resolv.conf，使用如下命令添加 DNS 服务器解析的指向。

echo “nameverver 208.67.222.222” > /etc/resolv.conf

表示将 DNS 服务器设置为 208.67.222.222

2、修改 Hosts 表实现静态 DNS 解析

要实现域名解析，即可以使用 DNS 服务器，也可以使用 Hosts 表。Hosts 表配置文件是/etc/hosts

启停网络接口

1、启用接口

ifconfig ethx up

2、停用接口

ifconfig ethx down

查看网络参数配置

1、ifconfig 或者 ifconfig -a

```
root@kali:~/Linux# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0c:29:8a:43:c3
          inet addr:10.5.68.6  Bcast:10.5.68.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe8a:43c3/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:119643 errors:0 dropped:0 overruns:0 frame:0
          TX packets:60270 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:36494948 (34.8 MiB)  TX bytes:19288613 (18.3 MiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:3772157 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3772157 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:525043816 (500.7 MiB)  TX bytes:525043816 (500.7 MiB)
```

输出项目	说明
Link encap	网络接口类型，如以太网或 PPP 等
HWadrr	网卡的 Mac 地址。每一块网卡都有自己的编号，用于在以太网协议下定位网络主机
inet addr	此接口对应的 IP 地址
网络接口状态标志	UP — 网络接口被启用
	RUNNING — 接口正在运行
	BROADCAST — 支持广播 IP 寻址方式
	MULTICAST — 支持多播 IP 寻址方式
	LOOPBACK — 表示本地回环设备接口
MTU	Message transfer unit，此接口所能传输的最大 frame 数
Metric	此接口的 Metric 数，用于引导路由决策
Bcast	广播地址，通常是网络的最后一个 IP 地址
Mask	子网掩码
RX packets	接收的封包总数、错误数、遗失数和溢流数
TX packets	发送的封包总数、错误数、遗失数和溢流数
collisions	冲突数（当多个 NIC 同时使用网线传输数据时会产生冲突）
txqueuelen	指出网络接口可以存储的数据包的个数
RX bytes	与 RX packets 类似，表示接收的具体字节数
TX bytes	与 TX packets 类似，表示发送的具体字节数
Interrupt	网卡使用的中断（IRQ）

2、ip address show

```
root@kali:~/Linux# ip address show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
    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 qlen 1000
    link/ether 00:0c:29:8a:43:c3 brd ff:ff:ff:ff:ff:ff
    inet 10.5.68.6/24 brd 10.5.68.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe8a:43c3/64 scope link
       valid_lft forever preferred_lft forever
```

3、ip link list

```
root@kali:~/Linux# ip link list
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode DEFAULT qlen 1000
    link/ether 00:0c:29:8a:43:c3 brd ff:ff:ff:ff:ff:ff
root@kali:~/Linux#
```

1.1.2 ARP 篇

Linux 查看 arp 表项

1)arp -a (显示结果比较慢)

2)arp -n

3)ip neigh show (显示的结果较为详细)

Linux 查看 NDP

1)ip -6 neigh show

Linux 删除 arp 表项

1)ifconfig eth0 down;ifconfig eth0 up

2)ip neigh delete 1.1.1.1 dev eth0

3)arp -d 10.5.68.1

ARP 扫描

ARP 扫描（ARP 请求风暴）

通讯模式（可能）：

请求 -> 请求 -> 请求 -> 请求 -> 请求 -> 请求 -> 应答 -> 请求 -> 请求 -> 请求...

描述：

网络中出现大量 ARP 请求广播包，几乎都是对网段内的所有主机进行扫描。大量的 ARP 请求广播可能会占用网络带宽资源；ARP 扫描一般为 ARP 攻击的前奏。

出现原因（可能）：

*病毒程序，侦听程序，扫描程序。

*来自和交换机相连的其它主机。

ARP 欺骗防护

1、静态绑定

双向绑定

2、使用 ARP 防护软件

3、定期发送合法的 arp 应答

ARP 欺骗的防护

ARP 欺骗和攻击问题，是企业网络的心腹大患。关于这个问题的讨论已经很深入了，对 ARP 攻击的机理了解的很透彻，各种防范措施也层出不穷。

但问题是，现在真正摆脱 ARP 问题困扰了吗？从用户那里了解到，虽然尝试过各种方法，但这个问题并没有根本解决。原因就在于，目前很多种 ARP 防范措施，一是解决措施的防范能力有限，并不是最根本的办法。二是对网络管理约束很大，不方便不实用，不具备可操作性。三是某些措施对网络传输的效能有损失，网速变慢，带宽浪费，也不可取。

本节通过具体分析一下普遍流行的四种防范 ARP 措施，去了解为什么 ARP 问题始终不能根治。

上篇：四种常见防范 ARP 措施的分析

1、双绑措施

双绑是在路由器和终端上都进行 IP-MAC 绑定的措施，它可以对 ARP 欺骗的两边，伪造网关和截获数据，都具有约束的作用。这是从 ARP 欺骗原理上进行的防范措施，也是最普遍应用的办法。它对付最普通的 ARP 欺骗是有效的。

但双绑的缺陷在于 3 点：

1) 在终端上进行的静态绑定，很容易被升级的 ARP 攻击所捣毁，病毒的一个 ARP–d 命令，就可以使静态绑定完全失效。

2) 在路由器上做 IP-MAC 表的绑定工作，费时费力，是一项繁琐的维护工作。换个网卡或更换 IP，都需要重新配置路由。对于流动性电脑，这个需要随时进行的绑定工作，是网络维护的巨大负担，网管员几乎无法完成。

3) 双绑只是让网络的两端电脑和路由不接收相关 ARP 信息，但是大量的 ARP 攻击数据还是能发出，还要在内网传输，大幅降低内网传输效率，依然会出现问题。

因此，虽然双绑曾经是 ARP 防范的基础措施，但因为防范能力有限，管理太麻烦，现在它的效果越来越有限了。

2、ARP 个人防火墙

在一些杀毒软件中加入了 ARP 个人防火墙的功能，它是通过在终端电脑上对网关进行绑定，保证不受网络中假网关的影响，从而保护自身数据不被窃取的措施。ARP 防火墙使用范围很广，有很多人以为有了防火墙，ARP 攻击就不构成威胁了，其实完全不是那么回事。

ARP 个人防火墙也有很大缺陷：

1) 它不能保证绑定的网关一定是正确的。如果一个网络中已经发生了 ARP 欺骗，有人在伪造网关，那么，ARP 个人防火墙上来就会绑定这个错误的网关，这是具有极大风险的。即使配置中不默认而发出提示，缺乏网络知识的用户恐怕也无所适从。

2)ARP 是网络中的问题，ARP 既能伪造网关，也能截获数据，是个“双头怪”。在个人终端上做 ARP 防范，而不管网关那端如何，这本身就不是一个完整的办法。AR P 个人防火墙起到的作用，就是防止自己的数据不会被盗取，而整个网络的问题，如掉线、卡滞等，ARP 个人防火墙是无能为力的。

因此，ARP 个人防火墙并没有提供可靠的保证。最重要的是，它是跟网络稳定无关的措施，它是个人的，不是网络的。

3、VLAN 和交换机端口绑定

通过划分 VLAN 和交换机端口绑定，以图防范 ARP，也是常用的防范方法。做法是细致地划分 VLAN，减小广播域的范围，使 ARP 在小范围内起作用，而不至于发生大面积影响。同时，一些网管交换机具有 MAC 地址学习的功能，学习完成后，再关闭这个功能，就可以把对应的 MAC 和端口进行绑定，避免了病毒利用 ARP 攻击篡改自身地址。也就是说，把 ARP 攻击中被截获数据的风险解除了。这种方法确实能起到一定的作用。

不过，VLAN 和交换机端口绑定的问题在于：

1)、没有对网关的任何保护，不管如何细分 VLAN，网关一旦被攻击，照样会造成全网上网的掉线和瘫痪。

2) 把每一台电脑都牢牢地固定在一个交换机端口上，这种管理太死板了。这根本不适合移动终端的使用，从办公室到会议室，这台电脑恐怕就无法上网了。在无线应用下，又怎么办呢？还是需要其他的办法。

3) 实施交换机端口绑定，必定要全部采用高级的网管交换机、三层交换机，整个交换网络的造价大大提高。

因为交换网络本身就是无条件支持 ARP 操作的，就是它本身的漏洞造成了 ARP 攻击的可能，它上面的管理手段不是针对 ARP 的。因此，在现有的交换网络上实施 ARP 防范措施，属于以子之矛攻子之盾。而且操作维护复杂，基本上是个费力不讨好的事情。

4、PPPoE

网络下面给每一个用户分配一个帐号、密码，上网时必须通过 PPPoE 认证，这种方法也是防范 ARP 措施的一种。PPPoE 拨号方式对封包进行了二次封装，使其具备了不受 ARP 欺骗影响的使用效果，很多人认为找到了解决 ARP 问题的终极方案。

问题主要集中在效率和实用性上面：

1)PPPoE 需要对封包进行二次封装，在接入设备上再解封装，必然降低了网络传输效率，造成了带宽资源的浪费，要知道在路由等设备上添加 PPPoE Server 的处理效能和电信接入商的 PPPoE Server 可不是一个数量级的。

2)PPPoE 方式下局域网间无法互访，在很多网络都有局域网内部的域控服务器、D NS 服务器、邮件服务器、OA 系统、资料共享、打印共享等等，需要局域网间相互通信的需求，而 PPPoE 方式使这一切都无法使用，是无法被接受的。

3) 不使用 PPPoE，在进行内网访问时，ARP 的问题依然存在，什么都没有解决，网络的稳定性还是不行。

因此，PPPoE 在技术上属于避开底层协议连接，眼不见心不烦，通过牺牲网络效率换取网络稳定。最不能接受的，就是网络只能上网用，内部其他的共享就不能在 PPP oE 下进行了。

通过对以上四种普遍的 ARP 防范方法的分析，我们可以看出，现有 ARP 防范措施都存在问题。这也就是 ARP 即使研究很久很透，但依然在实践中无法彻底解决的原因所在了。

下篇：免疫网络是解决 ARP 最根本的办法

道高一尺魔高一丈，网络问题必定需要网络的方法去解决。目前，欣全向推广的免疫网络就是彻底解决 ARP 问题的最实际的方法。

从技术原理上，彻底解决 ARP 欺骗和攻击，要有三个技术要点。

1) 终端对网关的绑定要坚实可靠，这个绑定能够抵制被病毒捣毁。

2) 接入路由器或网关要对下面终端 IP-MAC 的识别始终保证唯一准确。

3) 网络内要有一个最可依赖的机构，提供对网关 IP-MAC 最强大的保护。它既能够分发正确的网关信息，又能够对出现的假网关信息立即封杀。

免疫网络在这三个问题上，都有专门的技术解决手段，而且这些技术都是厂家欣全向的技术专利。下面我们会详细说明。现在，我们要先做一个免疫网络结构和实施的简单介绍。

免疫网络就是在现有的路由器、交换机、网卡、网线构成的普通交换网络基础上，加入一套安全和管理的解决方案。这样一来，在普通的网络通信中，就融合进了安全和管理的机制，保证了在网络通信过程中具有了安全管控的能力，堵上了普通网络对安全从不设防的先天漏洞。

免疫网络的结构

实施一个免疫网络不是一个很复杂的事，代价并不大。它要做的仅仅是用免疫墙路由器或免疫网关，替换掉现有的宽带接入设备。在免疫墙路由器下，需要自备一台服务器 24 小时运行免疫运营中心。免疫网关不需要，已自带服务器。这就是方案的所需要的硬件调整措施。

软性的网络调整是 IP 规划、分组策略、终端自动安装上网驱动等配置和安装工作，以保证整个的安全管理功能有效地运行。其实这部分工作和网管员对网络日常的管理没有太大区别。

免疫网络的监控中心

免疫网络具有强大的网络基础安全和管理功能，对 ARP 的防范仅是其十分之一不到的能力。但本文谈的是 ARP 问题，所以我们需要回过头来，具体地解释免疫网络对 AR P 欺骗和攻击防范的机理。至于免疫网络更多的强大，可以后续研究。

前述治理 ARP 问题的三个技术要点，终端绑定、网关、机构三个环节，免疫网络分别采用了专门的技术手段。

1) 终端绑定采用了看守式绑定技术。免疫网络需要每一台终端自动安装驱动，不安装或卸载就不能上网。在驱动中的看守式绑定，就是把正确的网关信息存贮在非公开的位置加以保护，任何对网关信息的更改，由于看守程序的严密监控，都是不能成功的，这就完成了对终端绑定牢固可靠的要求。

2) 免疫墙路由器或免疫网关的 ARP 先天免疫技术。在 NAT 转发过程中，由于加入了特殊的机制，免疫墙路由器根本不理会任何对终端 IP-MAC 的 ARP 申告，也就是说，谁都无法欺骗网关。与其他路由器不同，免疫墙路由器没有使用 IP-MAC 的列表进行工作，当然也不需要繁琐的路由器 IP-MAC 表绑定和维护操作。先天免疫，就是不用管也具有这个能力。

3) 保证网关 IP-MAC 始终正确的机构，在免疫网络中是一套安全机制。首先，它能够做到把从路由器中取到的真实网关信息，分发到每一个网内终端，而安装有驱动的终端，只接受这样的信息，其他信息不能接受，保证了网关的唯一正确性。其次，在每一台终端，免疫驱动都会拦截病毒发出的错误网关传播，不使其流窜到网络内

负载均衡三角传输 (DR) 模式之 ARP 禁止响应

echo “1” >/proc/sys/net/ipv4/conf/lo/arp_ignore

echo “2” >/proc/sys/net/ipv4/conf/lo/arp_announce

echo “1” >/proc/sys/net/ipv4/conf/all/arp_ignore

echo “2” >/proc/sys/net/ipv4/conf/all/arp_announce

/sbin/route add -host $VIP dev lo:0

1.1.3 路由篇

查看 Linux 内核路由 (ipv4)

1)route

或

route -n

```
root@kali:~/Linux# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.5.68.1       0.0.0.0         UG    0      0        0 eth0
10.5.68.0       0.0.0.0         255.255.255.0   U     0      0        0 eth0
root@kali:~/Linux#
```

route 命令的输出项说明

输出项	说明
Destination	目标网段或者主机
Gateway	网关地址，”*” 表示目标是本主机所属的网络，不需要路由
Genmask	网络掩码
Flags	标记。一些可能的标记如下：
	U — 路由是活动的
	H — 目标是一个主机
	G — 路由指向网关
	R — 恢复动态路由产生的表项
	D — 由路由的后台程序动态地安装
	M — 由路由的后台程序修改
	! — 拒绝路由
Metric	路由距离，到达指定网络所需的中转数（linux 内核中没有使用）
Ref	路由项引用次数（linux 内核中没有使用）
Use	此路由项被路由软件查找的次数
Iface	该路由表项对应的输出接口

2）ip route show

```
root@kali:~/Linux# ip route show
default via 10.5.68.1 dev eth0
10.5.68.0/24 dev eth0  proto kernel  scope link  src 10.5.68.6
root@kali:~/Linux#
```

3)ip route list table local

ip route list table main

```
root@kali:~/Linux# ip route list table local
broadcast 10.5.68.0 dev eth0  proto kernel  scope link  src 10.5.68.6
local 10.5.68.6 dev eth0  proto kernel  scope host  src 10.5.68.6
broadcast 10.5.68.255 dev eth0  proto kernel  scope link  src 10.5.68.6
broadcast 127.0.0.0 dev lo  proto kernel  scope link  src 127.0.0.1
local 127.0.0.0/8 dev lo  proto kernel  scope host  src 127.0.0.1
local 127.0.0.1 dev lo  proto kernel  scope host  src 127.0.0.1
broadcast 127.255.255.255 dev lo  proto kernel  scope link  src 127.0.0.1
root@kali:~/Linux# ip route list table main
default via 10.5.68.1 dev eth0
10.5.68.0/24 dev eth0  proto kernel  scope link  src 10.5.68.6
root@kali:~/Linux#
```

查看 Linux 内核路由 (ipv6)

1)route -n -6

2)route -A inet6

3)ip -6 route show

4)ip -6 route show dev eth0

添加路由 (ipv4)

添加默认路由

1)ip route add default via 10.5.68.1 dev eth0 table test

添加到主机的路由

2)route add -host 192.168.1.2 dev eth0:0

3)route add -host 10.20.30.148 gw 10.20.30.40

添加到网络的路由

4)route add -net 10.20.30.40 netmask 255.255.255.248 eth0

5)route add -net 10.20.30.48 netmask 255.255.255.248 gw 10.20.30.41

6)route add -net 192.168.1.0/24 eth1

添加路由 (ipv6)

1)ip -6 route add 2000::/3 via 3ffe:ffff:0:f101::1

2)route -A inet6 add 2000::/3 gw 3ffe:ffff:0:f101::1

3)ip -6 route add 2000::/3 dev eth0 metric 1

4)route -A inet6 add 2000::/3 dev eth0

5)route -A inet6 add default gw 2001:250:3000:2:2c0:95ff:fee0:473f

删除路由 (ipv4)

1)route del -host 192.168.1.2 dev eth0:0

2)route del -host 10.20.30.148 gw 10.20.30.40

3)route del -net 10.20.30.40 netmask 255.255.255.248 eth0

4)route del -net 10.20.30.48 netmask 255.255.255.248 gw 10.20.30.41

5)route del -net 192.168.1.0/24 eth1

6)route del default gw 192.168.1.1

删除路由 (ipv6)

1)ip -6 route del 2000::/3 via 3ffe:ffff:0:f101::1

2)route -A inet6 del 2000::/3 gw 3ffe:ffff:0:f101::1

3)ip -6 route del 2000::/3 dev eth0

4)route -A inet6 del 2000::/3 dev eth0

刷新路由表

ip route flush cache

三种路由类型

主机路由

主机路由是路由选择表中指向单个 IP 地址或主机名的路由记录。主机路由的 Flags 字段为 H。例如，在下面的示例中，本地主机通过 IP 地址 192.168.1.1 的路由器到达 IP 地址为 10.0.0.10 的主机。

```
Destination     Gateway         Genmask         Flags  Metric Ref    Use    Iface
-----------     -------         -------         -----  ------ ---    ---    -----
10.0.0.10       192.168.1.1     255.255.255.255 UH     0      0        0    eth0
```

网络路由

网络路由是代表主机可以到达的网络。网络路由的 Flags 字段为 N。例如，在下面的示例中，本地主机将发送到网络 192.19.12 的数据包转发到 IP 地址为 192.168.1.1 的路由器。

```
Destination     Gateway         Genmask         Flags  Metric  Ref    Use    Iface
-----------     -------         -------         -----  -----   ---    ---    -----
192.19.12       192.168.1.1     255.255.255.0   UN      0      0        0    eth0
```

默认路由

当主机不能在路由表中查找到目标主机的 IP 地址或网络路由时，数据包就被发送到默认路由（默认网关）上。默认路由的 Flags 字段为 G。例如，在下面的示例中，默认路由是 IP 地址为 192.168.1.1 的路由器。

```
Destination     Gateway         Genmask Flags  Metric  Ref    Use    Iface
-----------     -------         ------- -----  ------  ---    ---    -----
default         192.168.1.1     0.0.0.0 UG     0       0        0    eth0
```

设置路由转发

1) 临时生效

echo '1' >/proc/sys/net/ipv4/ip_forward

> 警告： 重启后配置失效

2) 永久生效

sysctl -w net.ipv4.ip_forward=1

或

echo “net.ipv4.ip_forward = 1” >>/etc/sysctl.conf

> 警告： 别忘记使用 sysctl -p 是配置生效

3) 查看系统目前支不支持路由转发

sysctl net.ipv4.ip_forward

1.2 Linux 查找 | 过滤命令

1、文件查找（find、locate）

2、内容查找 (grep)

3、内容过滤（cat、cut、tail、head）

1.2.1 文件查找

find

1、find 命令的格式

find 命令用于在文件系统中查找满足条件的文件。find 命令功能强大，提供了相当多的查找条件。find 命令还可以对查找到的文件做操作，如执行 Shell 命令等。

find 命令的格式是：

```
find [<起始目录> ...] [<选项表达式>] [<条件匹配表达式>] [<动作表达式>]
```

其中：

- **<起始目录>**：对每个指定的 **<起始目录>** 递归搜索目录树
  - 若在整个文件系统范围内查找，则起始目录是“/”
  - 若在当前目录下寻找，则起始目录是“.”，省略**<起始目录>**表示当前目录
- **<选项表达式>**：控制 **find** 命令的行为
- **<条件匹配表达式>**：根据匹配条件查找文件
- **<动作表达式>**：指定对查找结果的操作，默认为显示在标准输出（**-print**）

不带任何参数的 find 命令将在屏幕上递归显示当前目录下的文件列表。下面给出一些常用的表达式的解释。

2、选项表达式

表达式	说明
-follow	如果遇到符号链接文件，就跟踪链接所指的文件
-regextype TYPE	指定 -regex 和 -iregex 使用的正则表达式类型，默认为 emacs，还可选择 posix-awk, posix-basic, posix-egrep 和 posix-extended
-depth	查找进入子目录前先查找完当前目录的文件
-mount	查找文件时不跨越文件系统
-xdev	查找文件时不跨越文件系统
-maxdepth LEVELS	设置最大的查找深度
--help	显示 find 命令帮助信息
--version	显示 find 的版本

3、条件匹配表达式

表达式	说明
-name PATTERN	匹配文件名
-iname PATTERN	匹配文件名（忽略大小写）
-lname PATTERN	匹配符号链接文件名
-ilname PATTERN	匹配符号链接文件名（忽略大小写）
-path PATTERN	匹配文件的完整路径（不把 '/' 和 '.' 作为特殊字符）
	PATTERN 使用 Shell 的匹配模式，可以使用 Shell 的通配符（*、？[]），要用"'"或'"括起来
表达式	说明
-regex PATTERN	以正则表达式匹配文件名
-iregex PATTERN	以正则表达式匹配文件名（忽略大小写）
表达式	说明
-amin N	查找 N 分钟以前被访问过的所有文件
-atime N	查找 N 天以前被访问过的所有文件
-cmin N	查找 N 分钟以前文件状态被修改过的所有文件（比如权限修改）
-ctime N	查找 N 天以前文件状态被修改过的所有文件 （比如权限修改）
-mmin N	查找 N 分钟以前文件内容被修改过的所有文件
-mtime N	查找 N 天以前文件内容被修改过的所有文件
-uid N	查找属于 ID 号为 N 用户的所有文件
-gid N	查找属于 ID 号为 N 组的所有文件
-inum N	查找 i-node 是 N 的文件
-links N	查找硬链接数为 N 的文件
-size N[bcwkMG]	查找大小为 N 的文件，b(块)默认单位；c(字节)；w(双字节)

	N 可以有三种输入方式，+N 或 -N 或 N。假设 N 为 20，则：(1) **+20**：表示20以上（21，22，23等）；(2) **-20**：表示20以内（19，18，17等）；(3) **20**：表示正好是20。
表达式	说明
-perm MODE	精确匹配权限模式为 MODE 的文件。MODE：与 chown 命令的书写方式一致，既可以使用字符模式也可以使用8进制模式
-perm -MODE	匹配权限模式至少为 MODE 的文件
表达式	说明
-anewer FILE	查找所有比 FILE 的访问时间新的文件
-cnewer FILE	查找所有比 FILE 的状态修改时间新的文件（比如权限修改）
-newer FILE	查找所有比 FILE 的内容修改时间新的文件
-samefile FILE	查找与 FILE 具有相同 i-node 的文件 （硬链接）
表达式	说明
-fstype TYPE	只查找指定类型的文件系统
-type [bcdpfls]	查找指定类型的文件 [块设备, 字符设备,目录, 管道, 普通文件, 符号链接, socket套接字]
-empty	内容为空的文件
-user NAME	查找用户名为 NAME 的所有文件
-group NAME	查找组名为 NAME 的所有文件
-nouser	文件属于不在 /etc/passwd 文件中的用户
-nogroup	文件属于不在 /etc/group 文件中的组

4、动作表达式

表达式	说明
-print	在标准输出上列出查找结果（每行一个文件）
-print0	在标准输出上列出查找结果（取消间隔符） 同样与 |xargs -0 连用
-fprint FILE	与 -print 一致，只是输出到文件 FILE
-fprint0 FILE	与 print0 一致，只是输出到文件 FILE
-ls	使用 'ls -dils' 在标准输出上列出查找结果
-fls FILE	与 -ls 一致，只是输出到文件 FILE
-prune	忽略对某个目录的查找
-exec COMMAND {} \;	对符合查找条件的文件执行 Linux 命令
-ok COMMAND {} \;	对符合查找条件的文件执行 Linux 命令；与 -exec 不同的是，它会询问用户是否需要执行

5、组合条件表达式

在书写表达式时，可以使用逻辑运算符与、或、非组成的复合条件，并可以用 () 改变默认的操作符优先级。下面以优先级由高到低列出可用的逻辑操作符。若以空格作为各个表达式的间隔符，则各个表示式之间是与关系。

操作符	说明
(EXPR)	改变操作符优先次序，一些 UNIX 版的 find 命令要使用 \(EXPR \) 形式
! EXPR	表示对表达式取反
EXPR1 EXPR2	与逻辑，若 EXPR1 为假，将不再评估 EXPR2
EXPR1 -a EXPR2	与 EXPR1 EXPR2 功能一致
EXPR1 -o EXPR2	逻辑或，若 EXPR1 为真，将不再评估 EXPR2
EXPR1 , EXPR2	若 EXPR1 为假，继续评估 EXPR2

6、find 命令使用举例

```
#################################################################
### find 的版本和使用帮助信息
#################################################################
$ find --help     # 显示 find 命令帮助信息
$ find --version  # 显示 find 的版本
#################################################################
### 不指定匹配表达式，显示所有文件
#################################################################
# 递归显示当前目录的文件列表
$ find
# 递归显示 / 目录的文件列表
$ find /
# 递归显示 / 目录的文件列表（仅限于3层目录）
$ find / -maxdepth 3
# 递归显示 / 目录的文件列表（仅限于 / 文件系统）
$ find / -xdev
# 递归显示 /home、/www、/srv 目录的文件列表
$ find /home /www /srv
#################################################################
### 按文件名/路径名查找
#################################################################
# 查找特定的文件名
$ find -name myfile
$ find -maxdepth 2 -name symfony
# 使用通配符查找特定的文件名
```

```
$ find -name 'd*'
$ find -name '???'
$ find -name '[afd]*'
$ find -iname '[a-z]*'
$ find -name 'ch[0-2][0-9].txt*'
# 匹配文件路径名
$ find -path '*server'
./vbird/server
./server
$ find -path '*server[12]'
./server1
./server2
./server1/server2
./server2/server2
# 以正则表达式匹配文件路径名
$ find -regex '.*'
$ find -regex '.*ch0.*'
./ch01
./ch00
./vbird/server/1000results/ch09-01.jpg
$ find -regex '.*ch[0-9]+'
./ch01
./ch21
./ch00
./ch333
./ch1
./ch41
$ find -regex '.*ch[0-9]+\.txt'
./ch1.txt
./ch24.txt
################################################################
### 按文件属性查找
################################################################
# 只查找普通文件
$ find . -type f
# 只查找符号链接文件
$ find . -type l
# 查找硬连接数大于 1 的文件或目录
$ find /home -links +1
# 查找 /tmp 目录下小于 10M 的文件
$ find /tmp -size -10M
# 查找 /home 目录下大于 1G 的文件
$ find /home -size +1G
# 查找系统中为空的文件或者目录
$ find / -empty
# 查找在 /www 中最后10分钟访问过的文件
$ find /www -amin -10
# 查找在 /www 中最后2天访问过的文件
$ find /www -atime -2
# 查找在 /home 下最近2天内改动过的文件
$ find /home -mtime -2
# 列出被改动过后 2 日内被存取过的文件或目录
$ find /home -used -2
# 列出被改动过后 90 日前被存取过的文件或目录
$ find /home -used +90
# 列出 /home 目录中属于用户 osmond 的文件或目录
$ find /home -user osmond
# 列出 /home 目录中 UID 大于 501 的文件或目录
$ find /home -uid  +501
# 列出 /home 目录中组为 osmond 的文件或目录
$ find /home -group osmond
# 列出 /home 目录中 GID 为 501 的文件或目录
$ find /home -gid 501
# 列出 /home 目录中不属于本地用户的文件或目录
$ find /home -nouser
# 列出 /home 目录中不属于本地组的文件或目录
$ find /home -nogroup
# 精确查找权限为 664 的文件或目录
$ find . -perm 664
# 查找权限至少为 664 的文件或目录
$ find . -perm -664
```

```
#############################################################
### 使用逻辑运算构造复杂表达式
#############################################################
# 查找 /tmp 目录下21天之前访问过的大于 10G 的文件
$ find /tmp -size +10M -a -atime +21
# 查找 / 目录下属主为 jjheng 或 osmond 的文件
$ find / -user jjheng -o -user osmond
# 查找 /tmp 目录下的属主不是 osmond 的文件
$ find /tmp ! -user osmond
# 在 /mnt 下查找 *.txt 且文件系统类型不为 vfat 的文件
$ find /mnt -name '*.txt' ! -fstype vfat
# 在 /tmp 下查找名为 l 开头且类型为符号链接的文件
$ find /tmp -name 'l*' -type l
# 查找以 server 开头的目录名
$ find . -type d -name 'server*'
# 找出 /var/log 目录下所有的前5天修改过的.log 文件
$ find /var/log -name '*.log' -mtime +5
#############################################################
### 按文件样本查找
#############################################################
# 查找所有比 FILE1 的访问时间新的文件
$ find -anewer FILE1
# 查找所有比 FILE2 的访问时间旧的文件
$ find ! -anewer FILE2
# 查找所有比 FILE1 的访问时间新的
# 且比 FILE2 的访问时间旧的文件
$ find -anewer FILE1 ! -anewer FILE2
# 查找所有比 FILE1 的内容修改时间新的文件
$ find -newer FILE1
# 查找所有比 FILE2 的内容修改时间旧的文件
$ find ! -newer FILE2
# 查找所有比 FILE1 的内容修改时间新的
# 且比 FILE2 的内容修改时间旧的文件
$ find -newer FILE1 ! -newer FILE2
# 查找与 FILE 具有相同 i-node 的文件（硬链接）
$ find -samefile FILE -ls
#############################################################
### 对查找到的文件实施命令操作
#############################################################
# 查找并列出当前目录下不安全的文件（世界可读写执行）
$ find . -perm -007 -exec ls -l {} \;
# 查找 logs 目录下的所有的 .log 文件并查看它的详细信息
$ find logs -name "*.log" -type f -exec ls -l {} \;
# 查找当天修改过的普通文件
$ find . -type f -mtime -1 -exec ls -l {} \;
# 查找当前目录下的.php文件并用grep过滤出包含include的行
$ find . -name "*.php" -exec grep "include" {} \; -print
# 查找并删除当前目录及其子目录下所有扩展名为 .tmp 的文件
$ find . -name '*.tmp' -exec rm {} \;
# 在logs目录中查找7天之内未修改过的文件并在删除前询问
$ find logs -type f -mtime +7 -exec  -ok  rm {} \;
# 查询并删除一周以来从未访问过的以 .o 结尾或名为 a.out
# 且不存在于 nfs 文件系统中的所有文件
$ find / ( -name a.out -o -name '*.o' ) -atime +7 \
! -fstype nfs -exec rm {} \;
# 查询并删除当前目录及其子目录下所有的空目录
$ find . -depth -type d -empty -exec rmdir {} \;
# 将default目录下的文件由GBK编码转换为UTF-8编码
# 目录结构不变，转码后的文件保存在utf/default目录下
# From: http://www.xiaojb.com/archives/it/convert-gbk-utf-8.shtml
$ find default -type d -exec mkdir -p utf/{} \;
$ find default -type f -exec iconv -f GBK -t UTF-8 {} -o utf/{} \;
# 下面 find 命令的书写形式均等价
$ find -name \*.sh -exec cp {} /tmp \;
$ find -name '*.sh' -exec cp {} /tmp ';'
$ find -name "*.sh" -exec cp {} /tmp ";"
$ find -name \*.sh -exec cp \{\} /tmp \;
$ find -name '*.sh' -exec cp '{}' /tmp ';'
$ find -name "*.sh" -exec cp "{}" /tmp ";"
#############################################################
### 在查找中排除指定的目录
```

```
############################################################
# 显示当前目录树
$ tree -F -L 2
.
|-- bin/
|   |-- switch-lang.sh*
|   `-- sys2wiki.sh*
|-- book/
|   |-- basic/
|   |-- basic-utf8/
|   |-- basic.zip
|   |-- server/
|   |-- server-utf8/
|   |-- server.zip
|   `-- to-zh-CN-utf8.sh*
`-- bak.sh*
# 显示当前目录下除 book 目录的所有文件
$ find . -name book -prune -o -print
# 查找当前目录下（除了 book 目录）的所有 .sh 文件
$ find . -name book -prune -o -name '*.sh' -print
# 显示当前目录下除 book/server 目录的所有文件
$ find . -path ./book/server -prune -o -print
# 使用绝对路径完成上述任务
$ find /home/osmond -path /home/osmond/book/server -prune -o -print
# 查找当前目录下（除了 book/server 目录）的所有 .sh 文件
$ find . -path ./book/server -prune -o -name '*.sh' -print
# 显示当前目录下除 book/server 和 book/server-utf8 目录的所有文件
$ find . -path './book/server*' -prune -o -print
# 查找当前目录下（除了 book/server 和 book/server-utf8 目录）的所有 .sh 文件
$ find . -path './book/server*' -prune -o -name '*.sh' -print
# 显示当前目录下除 book/server 和 book/basic 目录的所有文件
$ find . \( -path ./book/server -o -path ./book/basic \) -prune -o -print
# 查找当前目录下（除了 book/server 和 book/basic 目录）的所有 .sh 文件
$ find . \( -path ./book/server -o -path ./book/basic \) -prune -o -name '*.sh' -print
```

1)find ./ -type f -name “*.txt” |xargs -i cp {} tmp/

2) 用 shell 查询以 “.” 结尾的文件，并加上后缀 “.ts”

find ./ -name “*.” exec mv {} {}ats ;

3) 查找所有具有 suid 的文件

find / -perm -4100 -exec ls -l {} ;

find / -type f -perm -u=s

4) 查找所有 sgid 的文件

find / -perm -2010 -exec ls -l {} ;

5) 查找同时具有 suid 和 sgid 属性的文件

find / -perm -6110 -exec ls -l {} ;

6)find . -type f -perm 6000 # 完全匹配

find . -type f -perm -6000 # 有 1 的位置必须一样

find . -type f -perm +6000 # 只要有其中一个有 1 的匹配就行

这些 8 进制的权限对比的时候要换成 2 进制形式)

7) 查找含有 sgid 权限位的目录

find / -type f or type d -perm 2000

find / -type f or type d -perm -g=s

8) 查找含有 tickty 权限位的目录

find / -type d -perm 1000

9) 查看危险目录

find / -perm -222 -type d

find / -perm -o+w -tye d

Locate

locate -r “ls$”

locate -r “^ls”

locate -r ‘^/bin.*ls$’

locate -r ‘.*/bin.*<passwd$’ #<passwd 表示以 passwd 单词开头

locate -r ‘.*/bin.*<passwd>’ # 包含 passwd 单词即可

1.2.2 内容查找与过滤

cut

功能说明: 纵向切割出文本指定的部分并写到标准输出，显示文件或 STDIN 数据的指定列

使用 -d 指定区分列的定界符（默认为 TAB）

使用 -f 指定要显示的列

$ cut -d: -f1 /etc/passwd (默认的分隔符是 tab)

$ grep root /etc/passwd | cut -d: -f7

使用 -c 按字符切割 $ cut -c2-5 /usr/share/dict/words

-s : 不打印没有包含分界符的行 (默认的分解符为 tab，可以使用 -d 参数修改)

-b<LIST> : 只列出 <LIST> 指定的字节

-c<LIST> : 只列出 <LIST> 指定的字符

1、cut -b-10 file # 只列出每行的开头到第 10 个字节的内容

2、cut -b2-10 # 列出每行第 2 到第 10 的字节的内容

3、cut -b2-10，15-20 # 列出每行第 2 到第 10 的字节,15 到 20 字节的内容

4、cut -c5- file # 列出每行第 5 个字符到最后的所有内容

5、cut -c5-10 file # 列出每行第 2 到第 10 的字符的内容

6、cut -c5-10,15-20 file # 列出每行 5 到 10 字符，15 到 20 字符的内容

7、cut -f1,3,5 file # 列出每行 1、3、5 字段的内容

8、cut -f2-4 file # 列出每行 2 到 4 字段的内容

9、cut -f1,2-4,6 -d’ ‘ -s file

10、cut 命令 -d 不能和 -c 一起用

11、cut -b8 file # 列出每一行的第 8 个字节的内容

grep

1、grep -R '10.1.198.85' /etc #-R 表示目录递归

2、grep 'test' d*　　# 显示所有以 d 开头的文件中包含 test 的行

3、grep 'test' aa bb cc　　# 显示在 aa，bb，cc 文件中包含 test 的行

4、grep magic /usr/src　　# 显示/usr/src 目录下的文件 (不含子目录) 包含 magic 的行

5、grep -r magic /usr/src　　# 显示/usr/src 目录下的文件 (包含子目录) 包含 magic 的行

6、grep -w pattern files：只匹配整个单词，而不是字符串的一部分 (如匹配'magic'，而不是'magical')，

7、grep '[a-z]{5}' aa # 显示所有包含每行字符串至少有 5 个连续小写字符的字符串的行

8、grep -v -E 'grep|monitor' /etc/passwd # 过滤掉含有 grep 或者 monitor 的行

9、-n 可以打印匹配行的行号显示的结果后面为行号: xxxx (xxx 为匹配行的内容)

10、-s 表示不打印错误信息，但是正常的匹配成功信息还是会打印的，而 -q 则是什么都不打印

grep 的其他参数

-i 表示不区分大小写

-v 表示匹配的但不显示

11、grep -c china dict.txt # 统计包含 china 的行数

ps -ef |grep -A2 -B2 wenjian # 显示匹配行和匹配上的上 2 两行，下两行

12、grep "*test" example.txt # 这里面的通配符 * 不能被 grep 正确处理，grep 默认不识别通配符，只好加行 -E 选项

13、grep test example.txt

grep test <example.txt

cat example.txt | grep test

以上三个都是等效的,grep 可以处理管道输入也可以处理标准输入

14、cat test.sh | grep -n 'echo'

5: echo "very good!"

; 7: echo "good!";

9: echo "pass!";

11: echo "no pass!";

#grep 过滤后输出的行号是原始内容的行号，没有发生改变

15、标准输入优于管道输入 sed -n '1,10p'<test.sh | grep -n 'echo' <testsh.sh

10:echo $total;

18:echo $total;

21: echo "ok";

哈哈，这个 grep 又接受管道输入，又有 testsh.sh 输入，那是不是 2 个都接收呢。刚才说了"<" 运算符会优先，管道还没有发送数据前，grep 绑定了 testsh.sh 输入，这样 sed 命令输出就被抛弃了。这里一定要小心使用

16、grep 使用 Basic regular expression (BR E) 书写匹配模式

egrep 使用 Extended regular expression (ER E) 书写匹配模式，等效于 grep -E

fgrep 不使用任何正则表达式书写匹配模式（以固定字符串对待），执行快速搜索，等效于 grep -F

17、grep -v '^#' myfile

18、grep '[a-z]{5}' myfile

19、egrep '[a-z]{5}' myfile

20、# 如果 west 被匹配，则 es 就被存储到内存中，并标记为 1，然后搜索任意个字符（.*），

这些字符后面紧跟着另外一个 es（1），找到就显示该行。

grep 'w(es)t.*1' myfile

egrep 'w(es)t.*1' myfile

21、通过管道过滤 ls 输出的内容，只显示以 ~ 或 - 或.bak 结尾的行

ls | egrep '(~|-|.bak)$'

tr

other

1、cat -n text.sql

-n: 由 1 开始对所有输出的行进行编号

2、cat -b text.sql

-b：和 -n 相似，只不过对于空行不编号

3、cat -b -s text.sql

-s：当遇到有连续两行以上的空行时，使用一个空行代替

4、cat file1 file2 > files

将两个文件内容合并

Linux tail

tail 命令从指定点开始将 File 参数指定的文件写到标准输出。如果没有指定文件，则会使用标准输入默认在标准输出上显示每个 FILE 的最后 10 行. 如果多于一个 FILE, 会一个接一个地显示, 并在每个文件显示的首部给出文件名. 如果没有 FILE, 或者 FILE 是 -, 那么就从标准输入上读取.

tail /etc/passwd 查看 passwd 文件后十行的内容

tail -2 /etc/passwd 查看 passwd 文件后两行内容

tail -f /etc/passwd 实时查看 passwd 文件后十行内容

tail - 表示从标准终输入读取

tail -n 20 /etc/passwd 显示最后 20 行的内容

tail -c 10 /etc/passwd 显示 passwd 最后 10 个字节

more +10 file1

从第 10 行开始向下显示内容

1.3 Linux Log

日志是安全事件、异常事件分析的入口，打好日志分析基础很重要

1.3.1 syslog

日志系统

1、什么是 syslog

日志的主要用途是系统审计、监测追踪和分析统计。

为了保证 Linux 系统正常运行、准确解决遇到的各种各样的系统问题，认真地读取日志文件是管理员的一项非常重要的任务。

Linux 内核由很多子系统组成，包括网络、文件访问、内存管理等。子系统需要给用户传送一些消息，这些消息内容包括消息的来源及其重要性等。所有的子系统都要把消息送到一个可以维护的公用消息区，于是，就有了 syslog。

syslog 是一个综合的日志记录系统。它的主要功能是：方便日志管理和分类存放日志。syslog 使程序设计者从繁重的、机械的编写日志文件代码的工作中解脱出来，使管理员更好地控制日志的记录过程。在 syslog 出现之前，每个程序都使用自己的日志记录策略。管理员对保存什么信息或是信息存放在哪里没有控制权。

syslog 能设置成根据输出信息的程序或重要程度将信息排序到不同的文件。例如，由于核心信息更重要且需要有规律地阅读以确定问题出在哪里，所以要把核心信息与其他信息分开来，单独定向到一个分离的文件中。

管理员可以通过编辑 /etc/syslog.conf 来配置它们的行为。

2、syslogd 的配置文件

syslogd 的配置文件 /etc/syslog.conf 规定了系统中需要监视的事件和相应的日志的保存位置。使用如下命令：

```
cat  /etc/syslog.conf
```

可以查看此文件的内容为：

```
# Log all kernel messages to the console.
# Logging much else clutters up the screen.
#kern.*                                                 /dev/console

# 将 info 或更高级别的消息送到 /var/log/messages，
# 除了 mail/news/authpriv/cron 以外。
# 其中*是通配符，代表任何设备；none 表示不对任何级别的信息进行记录。
*.info;mail.none;news.none;authpriv.none;cron.none      /var/log/messages

# 将 authpirv 设备的任何级别的信息记录到 /var/log/secure 文件中，
# 这主要是一些和认证、权限使用相关的信息。
authpriv.*                                              /var/log/secure
```

```
# 将 mail 设备中的任何级别的信息记录到 /var/log/maillog 文件中，
# 这主要是和电子邮件相关的信息。
mail.*                                                  -/var/log/maillog


# 将 cron 设备中的任何级别的信息记录到 /var/log/cron 文件中，
# 这主要是和系统中定期执行的任务相关的信息。
cron.*                                                  /var/log/cron

# 将任何设备的 emerg 级别或更高级别的消息发送给所有正在系统上的用户。
*.emerg                                                 *

# 将 uucp 和 news 设备的 crit 级别或更高级别的消息记录到 /var/log/spooler 文件中。
uucp,news.crit                                          /var/log/spooler

# 将和本地系统启动相关的信息记录到 /var/log/boot.log 文件中。
local7.*                                                /var/log/boot.log

# 将 news 设备的 crit 级别的消息记录到 /var/log/news/news.crit 文件中。
news.=crit                                              /var/log/news/news.crit
# 将 news 设备的 err 级别的消息记录到 /var/log/news/news.err 文件中。
news.=err                                               /var/log/news/news.err
# 将 news 设备的 notice 或更高级别的消息记录到 /var/log/news/news.notice 文件中。
news.notice                                             /var/log/news/news.notice
```

该配置文件的每一行的格式如下：

```
facility.priority          action
设备.级别                      动作
```

其中：

1、设备字段用来指定需要监视的事件。它可取的值如下：

设备字段	说明
authpriv	报告认证活动。通常，口令等私有信息不会被记录
cron	报告与cron和at有关的信息
daemon	报告与xinetd有关的信息
kern	报告与内核有关的信息。通常这些信息首先通过klogd传送
lpr	报告与打印服务有关的信息
mail	报告与邮件服务有关的信息
mark	在默认情况下每隔20分钟就会生成一次表示系统还在正常运行的消息。Mark消息很像经常用来确认远程主机是否还在运行的"心跳信号"（Heartbeat）。Mark消息另外的一个用途是用于事后分析，能够帮助系统管理员确定系统死机发生的时间。
news	报告与网络新闻服务有关的信息
syslog	由syslog生成的信息
user	报告由用户程序生成的任何信息，是可编程缺省值
uucp	由UUCP生成的信息
local0-local7	与自定义程序一起使用
*	*代表除了mark之外的所有功能

2、级别字段用于指明与每一种功能有关的级别和优先级。它可取的值如下：

级别字段	说明
emerg	出现紧急情况使得该系统不可用，有些需广播给所有用户
alert	需要立即引起注意的情况

crit	危险情况的警告
err	除了emerg、alert、crit的其他错误
warning	警告信息
notice	需要引起注意的情况，但不如err、warning重要
info	值得报告的消息
debug	由运行于debug模式的程序所产生的消息
none	用于禁止任何消息
*	所有级别，除了none

3、动作字段用于描述对应功能的动作。它可取的值如下：

动作字段	说明
file	指定一个绝对路径的日志文件名记录日志信息
username	发送信息到指定用户，*表示所有用户
device	将信息发送到指定的设备中，如/dev/console
@hostname	将信息发送到可解析的远程主机hostname，且该主机必须正在运行syslogd并可以识别syslog的配置文件

syslog 可以为某一事件指定多个动作，也可以同时指定多个功能和级别，它们之间用分号间隔。

查看日志

1、常见的日志文件

日志文件通常存放在 /var/log 目录下。在该目录下除了包括 syslogd 记录的日志之外，同时还包含所有应用程序的日志。

为了查看日志文件的内容必须要有 root 权限。日志文件中的信息很重要，只能让超级用户有访问这些文件的权限。管理员可以使用下面的命令

```
ls /var/log/
```

查看系统中使用的日志文件，常用的日志文件如表所示。

日志文件	说明
audit/	存储 auditd 审计守护进程的日志目录
conman/	存储 ConMan 串行终端管理守护进程的日志目录
cups/	存储 CUPS 打印系统的日志目录
httpd/	记录 apache 的访问日志和错误日志目录
mail/	存储 mail 日志的目录
news/	存储 INN 新闻系统的日志目录
pm/	存储电源管理的日志目录
ppp/	存储 pppd 的日志目录
prelink/	prelink 的日志目录

samba/	记录 Samba 的每个用户的日志目录
squid/	记录 Squid 的日志目录
vbox/	ISDN 子系统的日志目录
acpid	存储 acpid 高级电源管理守护进程的日志
anaconda.*	Red Hat/CentOS 安装程序 anaconda 的日志，参考 redhat 安装程序 anaconda 分析 [http://www.proxyserve.net/index.php?q=aHR0cDovL3d3dy5pYm0uY29tL2RldmVsb3BlcndvcmtzL2NuL2xpbnV4L2wtYW5hY29uZGEvaW5kZXguaHRtbA%3D%3D]
boot.log	记录系统启动日志
btmp	记录登陆未成功的信息日志
cron	记录守护进程 crond 的日志
dmesg	记录系统启动时的消息日志
lastlog	记录最近几次成功登录的事件和最后一次不成功的登录
maillog	记录邮件系统的日志
messages	由 syslogd 记录的 info 或更高级别的消息日志
rpmpkgs	记录了当前安装的所有 rpm 包
secure	由 syslogd 记录的认证日志
spooler	由 syslogd 记录的 uucp 和 news 的日志
vsftpd.log	记录 vsftpd 的日志
wtmp	一个用户每次登录进入和退出时间的永久记录
yum.log	记录 yum 的日志

2、查看文本日志文件

绝大多数日志文件是纯文本文件，每一行就是一个消息。只要是在 Linux 下能够处理纯文本的工具都能用来查看日志文件。可以使用 cat、tac、more、less、tail 和 grep 进行查看。

下面以 /var/log/message s 为例，说明其日志文件的格式。

该文件中每一行表示一个消息，而且都由四个域的固定格式组成：

时间标签 (Timestamp)：表示消息发出的日期和时间。

主机名 (Hostname)：表示生成消息的计算机的名字。

生成消息的子系统的名字：可以是“Kernel”，表示消息来自内核或者是进程的名字，表示发出消息的程序的名字。

在方括号里的是进程的 PID。

消息 (Me ssage)，即消息的内容。

例如：

```
# syslog 发出的消息，说明了守护进程已经在 Dec 16，03:32:41 重新启动了。
Dec 16 03:32:41 cnetos5 syslogd 1.4.1: restart.
# 在 Dec 19，00:20:56 启动了内核日志 klogd
Dec 19 00:20:56 cnetos5 kernel: klogd 1.4.1, log source = /proc/kmsg started.
# 在 Dec 19，00:21:01 启动了xinetd
Dec 19 00:21:01 cnetos5 xinetd[2418]: xinetd Version 2.3.14 started with libwrap
  loadavg labeled-networking options compiled in.
```

可以看出，实际上在 /var/log/message 文件中的消息都不是特别重要或紧急的。

3、查看非文本日志文件

也有一些日志文件是二进制文件，需要使用相应的命令进行读取。

1)lastlog

使用 lastlog 命令来检查某特定用户上次登录的时间，并格式化输出上次登录日志 /var/log/lastlog 的内容。例如：

```
# lastlog
Username         Port     From             Latest
root             pts/0    192.168.0.77     Wed Dec 19 02:11:14 +0800 2007
bin                                        **Never logged in**
......
osmond           pts/0    192.168.0.77     Wed Dec 19 07:37:34 +0800 2007
```

2)last

last 命令往回搜索 /var/log/wtmp 来显示自从文件第一次创建以来登录过的用户。例如：

```
# last
osmond   pts/0        192.168.0.77     Wed Dec 19 07:37   still logged in
osmond   pts/0        192.168.0.77     Wed Dec 19 02:19 - 07:14  (04:54)
root     pts/0        192.168.0.77     Wed Dec 19 02:11 - 02:17  (00:06)
osmond   pts/0        192.168.0.77     Wed Dec 19 00:43 - 02:11  (01:27)
reboot   system boot  2.6.18-53.el5    Wed Dec 19 00:20         (32+16:26)
root     tty1                          Fri Dec 14 20:33 - down   (15:11)
reboot   system boot  2.6.18-53.el5    Sun Dec  9 01:08          (00:05)

wtmp begins Sun Dec  9 01:08:00 2007
```

3)lastb

lastb 命令搜索 /var/log/btm p 来显示登录未成功的信息。例如：

```
# lastb
osmond   ssh:notty    192.168.0.77     Sat Dec 15 17:24 - 17:24  (00:00)
rroot    tty1                          Tue Dec 11 06:40 - 06:40  (00:00)

btmp begins Tue Dec 11 06:40:57 2007
```

4）who

who 命令查询 wtm p 文件并报告当前登录的每个用户。who 命令的缺省输出包括用户名、终端类型、登录日期及远程主机。例如：

```
$ who
root     tty1         2007-12-20 16:49
osmond   pts/0        2007-12-19 07:37 (192.168.0.77)
```

> 警告： last 与 lastlog 的区别

last 指的是最近登录的记录，且只记录有登录记录的用户，没有登录的不记录，可能一个用户有多次登录记录，也包括正在登录的记录 (正在登录的现实 still login)

lastlog 记录最近一次用户登录的记录每个用户只有一条记录，切包括哪些没有登录的 (显示 Nerver Login)

> 警告： utmp 和 wtmp 的区别

/var/run/utmp – database of currently logged-in users

/var/log/wtmp – database of past user logins

5) 其他查询命令

w 命令查询 utmp 文件并显示当前系统中每个用户和它所运行的进程信息

ac 命令根据当前的/var/log/wtmp 文件中的登录进入和退出来报告用户连结的时间（小时）

users 用单独的一行打印出当前登录的用户，每个显示的用户名对应一个登录会话

1.3.2 日志回滚

为什么使用日志滚动

所有的日志文件都会随着时间的推移和访问次数的增加而迅速增长，因此必须对日志文件进行定期清理以免造成磁盘空间的不必要的浪费。同时也加快了管理员查看日志所用的时间，因为打开小文件的速度比打开大文件的速度要快。

logrotate

Linux 下有一个专门的日志滚动处理程序 logrotate 能够自动完成日志的压缩、备份、删除、和日志邮寄等工作。每个日志文件都可被设置成每日，每周或每月处理，也能在文件太大时立即处理。一般把 logrota te 加入到系统每天执行的计划任务中，这样就省得管理员自己去处理了。

其命令格式为：

选项说明如下：

- -d：详细显示指令执行过程，便于排错或了解程序执行的情况。
- -f：强行启动记录文件维护操作，即使 logrota te 指令认为无需要亦然。
- -m comm and：指定发送邮件的程序，默认为 /usr/bin/m ail。
- -s sta te file：使用指定的状态文件。
- -v：在执行日志滚动时显示详细信息。
- -?：显示命令帮助。
- –usage：显示使用摘要信息。

<configfile> 是 logrotate 命令的配置文件的路径。

logrotate 的配置文件

管理员可以在 logrota te 的配置文件中设置日志的滚动周期，日志的备份数目，以及如何备份日志等等。

- logrota te 默认的主配置文件是 /etc/logrotate.conf
- /etc/logrotate.d 的目录下的文件，这些文件被 include 到主配置文件 /etc/logrotate.conf 中

在这些文件中可以使用如下的配置语句。

配置语句	功能
compress	对滚动的旧日志文件使用 gzip 压缩。
nocompress	不压缩滚动的旧日志文件。
copytruncate	用在处于打开状态的日志文件，将当前日志备份并截断。
nocopytruncate	备份日志文件但是不截断。
create mode owner group	滚动日志时使用指定的文件模式创建新的日志文件。
nocreate	不创建新的日志文件。
delaycompress	和 compress 一起使用，转储的日志文件到下一次滚动时才压缩。
nodelaycompress	覆盖 delaycompress 选项，转储同时压缩。
ifempty	即使是空文件也滚动日志，这个是 logrotate 的缺省选项。
notifempty	如果是空文件的话，不滚动日志
errors address	将滚动日志时的错误信息发送到指定的 Email 地址。
mail address	把转储的日志文件发送到指定的 E-mail 地址。
nomail	转储时不发送日志文件。
olddir directory	将滚动的旧日志文件存储到指定的目录，必须和当前日志文件在同一个文件系统。
noolddir	将滚动的旧日志文件和当前日志文件放在同一个目录下。
prerotate/endscript	在滚动日志以前需要执行的命令可以放入此语句括号内，这两个关键字必须单独成行。
postrotate/endscript	在滚动日志以后需要执行的命令可以放入此语句括号内，这两个关键字必须单独成行。
daily	指定日志滚动周期为每天。
weekly	指定日志滚动周期为每周。
monthly	指定日志滚动周期为每月。
rotate n	指定日志文件删除之前日志滚动的备份次数，0 指没有备份，5 指保留 5 个备份
tabootext [+] list	让 logrotate 不转储指定扩展名的文件，缺省的扩展名是：.rpmorig, .rpmsave, dpkg-dist, .dpkg-old, .dpkg-new, .disabled, ,v, .swp, .rpmnew, 和 ~。
size n	当日志文件到达指定的大小时才进行日志滚动，n 可以指定 bytes(缺省)，或使用 G/M/k 后缀单位。

1. 在 /etc/logrotate.conf 中可以使用以上的配置语句设置全局值

2. 在 /etc/logrotate.conf 中使用 include 语句包含的配置文件中也可以使用上述的配置语句，被 include 的配置文件中的语句会覆盖 /etc/logrotate.conf 中的配置

3. 为指定的文件配置日志滚动使用如下的语法

```
# 注释

/full/path/to/logfile {

    配置语句1
    ……
    配置语句n

}
```

1.4 Linux VIM

主要内容包括 vim 的安装配置，常用技巧

1.4.1 安装配置篇

vim 的安装，vim 插件的安装，语法高亮、项目视图配置

vim 的安装

1、Debian 系列

apt-get install vim

2、Redhat 系列

yum install vim

vim 插件管理器安装

最好也要安装 vim-scripts 和 vim-addon-manager 这两个是插件管理器安装好之后，可以方便的在线进行插件的安装

1、Debian 系列

apt-get install vim-scripts vim-addon-manager

2、Redhat 系列

yum install vim-scripts vim-addon-manager

3、安装插件

vim-addons install taglist —安装 taglist 插件

vim 的配置

语法高亮

vim 版本 5 之后就支持语法高亮显示

1、首先查看 vim 版本, 如果符合要求则进行下一步

2、编辑 vim 配置文件 vimrc vim /etc/vim/vimrc，这是系统中公共的 vim 配置文件，对所有用户都有效

1) 打开 vimrc，添加以下语句来使得语法高亮显示

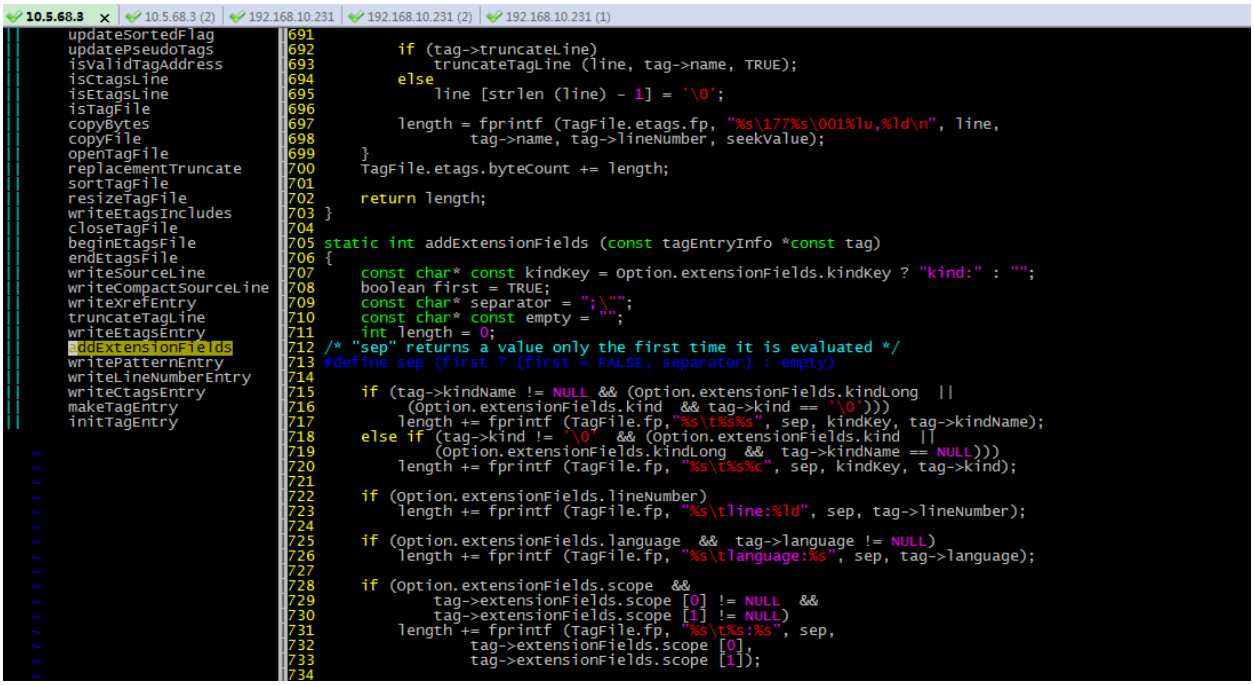
syntax on

2) 如果此时语法还是没有高亮显示，那么在/etc 目录下的 profile 文件中添加以下语句

export TERM=xterm-color

项目视图

项目试图就是这种：



1、下载安装 Exuberant Ctags

2、假设 ctags 可执行文件安装于:/usr/local/bin 目录下

在 vimrc 中加入

let Tlist_Ctags_Cmd='/usr/local/bin/ctags'

或者在.bashrc 中加入

Export Tlist_Ctags_Cmd='/usr/local/bin/ctags'

3、然后用 vim-add-manager 下载安装 Tasklist 插件

vim-addons install taglist

4、在 vim normal 模式下执行命令：TlistToggle

5、退出项目试图就在 normal 模式下再次执行 TlistToggle 即可

语法提示和自动完成

1、下载 pythoncomplete.vim 并将其放在 <Vim 安装目录 >/<$VIMRUNTIME>/autoload/目录下 (一般为：/usr/share/vim/vim 版本/autoload)

2、在 vimrc 中添加

filetype plugin on

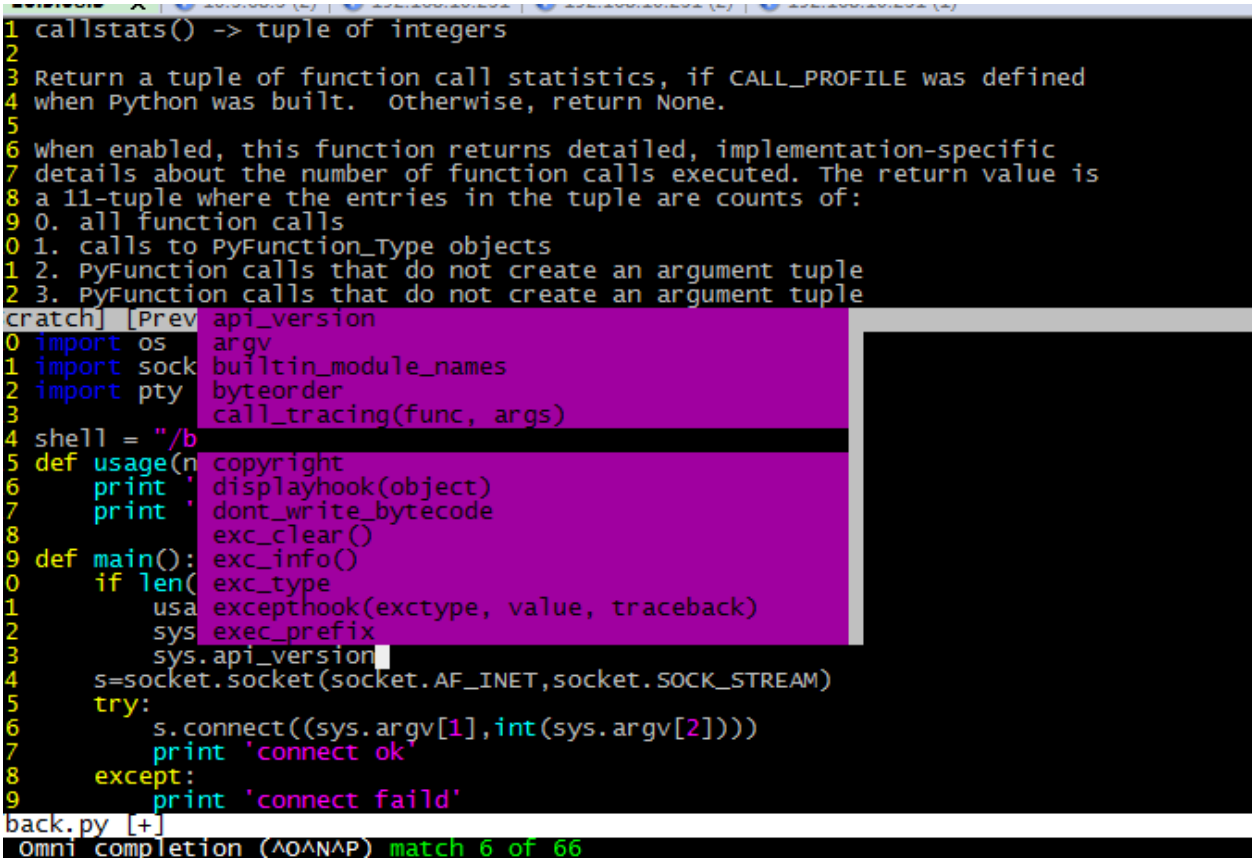
set ofu=syntaxcomplet

autocmd FileType python

set omnifunc=pythoncomplete

autocmd FileType python runtime! autoload/pythoncomplete.vim

3、然后在用 vim 编辑 python 代码文件时候通过 ctrl-x ctrl-o 或者 ctrl+n 来打开文法提示上下文菜单, 如下图所示:



FAQ

1、vim 的配置文件默认在什么位置

vim 启动的时候会读取配置文件，配置文件一般位置为:/etc/vim/vimrc

/etc/vim/gvimrc —适用于 Gui VIM

另外还有一个配置文件：/usr/share/vim/vim73/debian.vim

这个配置文件不建议直接修改，而是修改 vimrc, 因为 vimrc 中的配置文件会直接覆盖 debian.vim 的配置

2、vim 插件的位置

2.1 语法类的插件一般位置为：

/usr/share/vim/vim73/syntax (作用：用于识别不同类型语言的代码文件，为语法高亮显示提供支持)

(如果 vim 不能自动识别语法文件，还需要在 vim 中执行命令：set filetype=language，如:set filetype=python。)

2.2 标准插件的位置：

/usr/share/vim/vim73/plugin ，vim 每次启动的时候都会加载这个文件夹内的文件

2.3 编译类的插件位置：

/usr/share/vim/vim73/compiler 顾名思义就是为能够在 Vim 直接编译某些语言编写的程序提供支持，其他的插件也都位于

/usr/share/vim/vim73 目录下的相应目录

3.vimrc 配置文件中常用的设置

set nocompatible " explictly get out of vi-compatible mode

set background=dark " we plan to use a dark background

syntax on " syntax highlighting on

set number " turn on line numbers

set ruler "always show current position along the bottom

set incsearch " do highlight as you type you search phrase

set ignorecase " case insensitive by default

set smartcase " if there are caps, go case-sensitive

colorscheme macvim " the color scheme I am using now

" 1 tab == 4 spaces

set shiftwidth=4

set tabstop=4

" Enable filetype plugins

filetype on

filetype plugin on

filetype indent on

"设置 python 自动补充

set ofu=syntaxcomplet

autocmd FileType python

set omnifunc=pythoncomplete

autocmd FileType python runtime! autoload/pythoncomplete.vim

" Set utf8 as standard encoding and en_US as the standard language

set encoding=utf8

set langmenu=zh_CN.UTF-8

"设置缩进

set autoindent " same level indent

set smartindent " next level indent

set expandtab

set tabstop=4

set shiftwidth=4

set softtabstop=4

1.4.2 系统篇

Vi 简介

Vi 是“Visual interface”的简称，它可以执行输出、删除、查找、替换、块操作等众多文本操作，而且用户可以根据自己的需要对其进行定制，这是其他编辑程序所没有的。

Vi 不是一个排版程序，它不像 M$ Word 或 WPS 那样可以对字体、格式、段落等其他属性进行编排，它只是一个文本编辑程序。

Vi 是全屏幕文本编辑器，它没有菜单，只有命令。

进入 Vi

在命令行下键入 vi 即可进入 Vi 界面。还有如下几种进入方法：

1、打开或新建文件filename，并将光标置于第一行首

```
vi filename
```

2、打开文件filename，并将光标置于第n行首

```
vi +n filename
```

3、打开文件filename，并将光标置于最后一行首

```
vi + filename
```

4、打开文件filename，并将光标置于第一个与pattern匹配的串处

```
vi +/pattern filename
```

5、打开上次用vi编辑时发生系统崩溃，恢复filename

```
vi -r filename
```

Vi 的 3 种运行模式

Vi 有 3 种基本工作模式：普通 (norm al) 模式、插入 (insert) 模式和命令行 (comm and-line 或 Cm dline) 模式。

进入 Vi 之后，首先进入的就是普通模式，进入普通模式后 Vi 等待编辑命令输入而不是文本输入，也就是说这时输入的字母都将作为命令来解释。在普通 (norm al) 模式里，你可以输入所有的普通编辑命令。普通模式亦称为命令 (comm and) 模式。

进入普通模式后光标停在屏幕第一行首位上（用 _ 表示），其余各行的行首均有一个“~”符号，表示该行为空行。最后一行是状态行，显示出当前正在编辑的文件名及其状态。如果是 [New File]，则表示该文件是一个新建的文件。如果输入 Vi 带文件名后，文件已在系统中存在，则在屏幕上显示出该文件的内容，并且光标停在第一行的首位，在状态行显示出该文件的文件名、行数和字符数

在普通模式下输入插入命令 i、附加命令 a、打开命令 o、修改命令 c、取代命令 r 或替换命令 s 都可以进入插入模式。在插入模式下，用户输入的任何字符都被 Vi 当作文件内容保存起来，并将其显示在屏幕上。在文本输入过程中，若想回到命令行模式下，按 Esc 键即可。

在普通模式下，执行 Ex 命令使用:，查找使用? 和 /，调用过滤命令使用!。多数文件管理命令都是在此模式下执行的。末行命令执行完后，Vi 自动回到普通模式。

关于这 3 种模式的转换如图所示。

Vi 三种模式之间的转换示意图

普通模式下的操作

1. 进入插入模式

命令	说明
i	从光标所在位置前开始插入文本
I	该命令是将光标移到当前行的行首，然后在其前插入文本
a	用于在光标当前所在位置之后追加新文本
A	将把光标挪到所在行的行尾，从那里开始插入新文本
o	将在光标所在行的下面新开一行，并将光标置于该行的行首，等待输入文本
O	在光标所在行的上面插入一行，并将光标置于该行的行首，等待输入文本

2、光标定位

命令	说明
G	将光标移至最后一行行首
nG	光标移至第 n 行首
n+	光标下移 n 行
n−	光标上移 n 行
n$	光标移至第 n 行尾
0	移动到光标所在行的行首
$	移动到光标所在行的行尾
^	移动到光标所在行的第一个字符（非空格）
h、j、k、l	分别用于光标左移、下移、上移、右移一个字符
H	分别用于光标左移、下移、上移、右移一个字符
M	将光标移至屏幕显示文件的中间行的行首
L	将光标移至当前屏幕的最底行的行首

3、替换和删除

命令	说明
rc	用字符c替换光标所指向的当前字符
nrc	用字符c替换光标所指向的前n个字符
x	删除光标处的字符
nx	删除从光标所在位置开始向右的n个字符
dw	删除一个单词。若光标处在某个词的中间，则从光标所在位置开始删至词尾并连同空格
ndw	删除n个指定的单词
db	删除光标所在位置之前的一个词
ndb	删除光标所在位置之前的n个词
dd	删除光标所在的整行
ndd	删除当前行及其后n-1行的内容
dG	删除光标所在位置到最后一行的所有内容
d1G	删除光标所在位置到第一行的所有内容
d$	删除光标所在位置到当前行的末尾的内容
d0	删除光标所在位置到当前行的开始的内容

4、复制和粘贴

命令	说明
yy	将当前行的内容复制到缓冲区
nyy	将当前开始的n行内容复制到缓冲区
yG	将当前光标所在位置到最后一行的所有内容复制到缓冲区
y1G	将当前光标所在位置到第一行的所有内容复制到缓冲区
y$	将当前光标所在位置到当前行的末尾的内容复制到缓冲区
y0	将当前光标所在位置到当前行的开始的内容复制到缓冲区
p	将缓冲区的内容写出到光标所在的位置

5、搜索字符串

命令	说明
/str	往右移动到有 str 的地方
?str	往左移动到有 str 的地方
n	向相同的方向移动到有str的地方
N	向相反的方向移动到有str的地方

6、撤销和重复

命令	说明
u	取消前一次的误操作或不合适的操作对文件造成的影响，使之恢复到这种误操作或不合适操作被执行之前的状态
.	再执行一次前面刚完成的某个命令

7、退出 Vi

命令	说明
ZZ	存盘退出
ZQ	不保存退出

命令行模式下的操作

1、跳行

命令	说明
:n	直接输入要移动到的行号即可实现跳行

2、字符串搜索、替换

命令	说明
:/str/	从当前光标开始往右移动到有 str 的地方
:?str?	从当前光标开始往左移动到有 str 的地方
:/str/w file	将包含有str的行写到文件file中
:/str1/,/str2/w file	将从str1开始到str2结束的内容写入file文件中
:s/str1/str2/	将找到的第1个str1替换为str2
:s/str1/str2/g	将找到的所有的str1替换为str2
:n1,n2s/str1/str2/g	将从n1行到n2行找到的所有的str1替换为str2
:1,.s/str1/str2/g	将从第1行到当前位置的所有的str1替换为str2
:.,$s/str1/str2/g	将从当前位置到结尾的所有的str1替换为str2
:1,$s/str1/str2/gc	将从第1行到到最后一行的所有的str1替换为str2，并在替换前询问

vi 支持基本的正则表达式 Basic regular expre ssion (BRE)，上述命令中的 str 和 str1 可以使用正则表达式进行搜索。

3、文本的复制、移动和删除

命令	说明
:n1,n2 co n3	将从n1开始到n2为止的所有内容复制到n3后面
:n1,n2 m n3	将从n1开始到n2为止的所有内容移动到n3后面
:d	删除当前行
:nd	删除从当前行开始的n行
:n1,n2 d	删除从n1开始到n2为止的所有内容
:.,$d	删除从当前行到结尾的所有内容
:/str1/,/str2/d	删除从str1开始到str2为止的所有内容

4、文件相关

命令	说明
:w	将当前编辑的内容存盘
:w file	将当前编辑的内容写到 file 文件中
:n1,n2w file	将从 n1 开始到 n2 结束的行写到 file 文件中
:nw file	将第 n 行写到 file 文件中
:1,.w file	将从第 1 行起到光标当前位置的所有内容写到 file 文件中
:.,$w file	将从光标当前位置起到文件结尾的所有内容写到 file 文件中
:r file	打开另一个文件 file
:e file	新建 file 文件
:f file	把当前文件改名为 file 文件

5、执行 Shell 命令

命令	说明
:!Cmd	运行Shell命令Cmd
:n1,n2 w ! Cmd	将n1到n2行的内容作为Cmd命令的输入，如果不指定n1和n2，则将整个文件的内容作为命令Cmd的输入
:r ! Cmd	将命令运行的结果写入当前行位置

6、设置 Vi 环境

命令	说明
:set autoindent	缩进每一行，使之与前一行相同。常用于程序的编写
:set noautoindent	取消缩进
:set number	在编辑文件时显示行号
:set nonumber	不显示行号
:set ruler	在屏幕底部显示光标所在的行、列位置
:set noruler	不显示光标所在的行、列位置
:set tabstop=value	设置显示制表符的空格字符个数
:set wrapmargin=value	设置显示器的右页边。当输入进入所设置的页边时，编辑器自动回车换行
:set	显示设置的所有选项
:set all	显示所有可以设置的选项

7、退出 Vi

命令	说明
:q	退出Vi
:wq	保存退出Vi
:q!	不保存退出Vi

1.4.3 技巧篇

vim 常用技巧，值得去看

常用技巧

1、如何去掉 vim 打开的文件中的 ^M

:%s/r/

其他可以参考的方法

tr -d “\r” < src >dest

tr -d “\015” dest

cat filename1 | tr -d “^V^M” > newfile;

sed -e “s/^V^M//” filename >

:%s/^M$//g

strings A>B

(windows rn 0d0a;linux n 0a)

2、vim 跨文件复制

1) 用 vim 打开一个文件，例如：original.trace

2) 在普通模式下，输入：”:sp”（不含引号）横向切分一个窗口，或者”:vsp” 纵向切分一个窗口，敲入命令后，你将看到两个窗口打开的是同一个文件

3) 在普通模式下，输入：”:e new.trace”，在其中一个窗口里打开另一个文件

4) 切换到含有源文件（original.trace）的窗口，在普通模式下，把光标移到你需要复制内容的起始行，然后输入你想复制的行的数量（从光标所在行往下计算），在行数后面接着输入 yy，这样就将内容复制到临时寄存器里了（在普通模式下 ctrl+w，再按一下 w，可以在两个窗口之间切换）

5) 切换到目标文件（new.trace）窗口，把光标移到你接收复制内容的起始行，按一下 p，就完成复制了。

3、3、修改文件的编码

1) 查看当前 vim 打开文件内容的编码

:set fileencoding

2) 设置新编码

:set fileencoding=utf-8

4、连续删除 1，到 n 行

:1,10d

5、vim 连续注释多行

:1,10s/^/#/g

6、取消连续多行的注释

:1,10s/#/^/g

7、多行整体向右移动若干个 tab

:1,10>>

8、多行整体向 ← 移动若干个 tab

:1,10<<

9、vim 插入的命令

a 在光标之后插入

i 在光标之前插入

o 在下行插入

O 在上行插入

10、定位到第一行

gg

:1

11、定位到最后一行

G

:$

12、删除文件所有内容 :%d

13、无插件 Vim 编程技巧

http://coolshell.cn/articles/11312.html

14、vim 替换

vi/vim 中可以使用：s 命令来替换字符串。该命令有很多种不同细节使用方法，可以实现复杂的功能

:s/vivian/sky/ 替换当前行第一个 vivian 为 sky

:s/vivian/sky/g 替换当前行所有 vivian 为 sky

:n,$s/vivian/sky/ 替换第 n 行开始到最后一行中每一行的第一个 vivian 为 sky

:n,$s/vivian/sky/g 替换第 n 行开始到最后一行中每一行所有 vivian 为 sky

:s#vivian/#sky/# 替换当前行第一个 vivian/ 为 sky/

:%s+/oradata/apras/+/user01/apras1+ （使用 + 来替换 / ）：/oradata/apras/替换成/user01/apras1/

:s/vivian/sky/ 替换当前行第一个 vivian 为 sky

:s/vivian/sky/g 替换当前行所有 vivian 为 sky

:%s/vivian/sky/ 替换每一行的第一个 vivian 为 sky

:%s/vivian/sky/g 替换每一行中所有 vivian 为 sky

s/vivian/sky/gc 替换当前行所有 vivian 为 sky , 在替换之前进行替换确认，c 是 confirm 的缩写

1.5 大话 SSH

你需要掌握的

1、ssh 安全配置

2、ssh 隧道

3、ssl 端口转发

1.5.1 ssh 安全配置

1、sshd_confg 安全配置

1) 禁止 root 登录

2) 禁止空口令登录

3) 禁止 kndown host 登录

4) 禁止端口转发

5) 使用 1024 位公钥 (默认是 768)

6) 只允许指定用户 | 用户组登录

2、ssh 登录限制配置思路

1) 使用 tcpwrapper

你可在 hosts.allow 中设：

sshd: 192.168.0.1

sshd: ALL: deny

只允许 192.168.0.1 主机登录

2)iptables

iptables -I INPUT -p tcp –dport 22 -j DROP

iptables -I INPUT -p tcp –dport 22 -s 192.168.0.0/32 -j ACCEPT

(注：用 -I 命令，且顺序不能颠倒...)

3) 使用 pam 模块

修改 /etc/pam.d/sshd

auth required pam_listfile.so item=user sense=allow file=/etc/sshusers onerr=fail

将你要的用户写进/etc/sshusers ， 如：

echo “zhangsan” >> /etc/sshusers

设定登录黑名单

vi /etc/pam.d/sshd

增加

auth required / lib/ security/ pam_listfile.so item=user sense=deny file=/ etc/ sshd_user_deny_list onerr=succeed

所有/etc/sshd_user_deny_list 里面的用户被拒绝 ssh 登录

4) 使用 sshd_config 配置文件

允许或者禁止某个用户 (组) 通过 ssh 登录

在/etc/ssh/sshd_conf 添加

AllowUsers 用户名

或者

AllowGroups 组名

或者

DenyUsers 用户名

3、ssh 暴力破解防护

1) 限制 IP

iptables、tcpwraper 都可以做

2) 限制用户 (组)

sshd 可做

pam（黑名单、白名单）

3) 只允许公钥登录

4) 指定尝试密码次数

vi /etc/sshd_config

修必 MaxAuthTries 3

4、ssh 中间人攻击原理与防护

1) 原理

dh 算法是这样的，A 和 B 先共享 p,q。然后 A 生成随机数 a, 计算 E1=(p^a)modq 发给 B，B 生成随机数 b, 计算 E2=(p^b)modq 发给 A；A 计算 K=E2^a=(p^ab)modq,B 计算 K=E1^b=(p^ba)modq，就共享了 K 这个密钥吧，

C 假设是中间人，他知道 E1 和 E2,p,q

他也生成一个随机数 c，计算 Ec=(p^c)modq 给 A 和 B

这个时候 E1，E2 被截获没有发给对方，都在中间被 C 拦截住了

A 收到 Ec 以为是 E2,B 收到 Ec 以为是 E1

A 计算 Kac=Ec^a=p^ca,B 计算 Kbc=Ec^b=p^cb

C 就可以中间人了

E1、E2、p、q 这些信息全部用服务器公钥加密传输

2) 防护

注意 server 端公钥指纹的变化

5、公钥自动登录设置

ssh-keygen 和 ssh-copy-id 实现 ssh 自动登录

本实验的前提是：两者都是 linux 系统，且一端是 ssh-server(192.168.11.164)，一端是 ssh-client(192.168.11.103)

在 ssh-client 执行如下操作

1)ssh-keygen -t rsa ===> 这条命令用来生成 rsa 密钥对（一个公钥，一个私钥，.pub 结尾的是公钥，这里 -t 的参数就是指定使用的算法，主要有三种 rsa dsa 用于 ssh2 ，rsa1 用于 ssh1，这里以 ssh2 为例实验）

在 ssh-client 当前用户的家目录下面的.ssh 目录下面会出现两个文件 id_rsa id_rsa.pub

2)ssh-copy-id -i id_rsa.pub root@192.168.11.164

出现如下：

root@192.168.11.164's password:

Now try logging into the machine, with "ssh 'root@192.168.11.164'", and check in:

.ssh/authorized_keys

to make sure we haven't added extra keys that you weren't expecting.

这样当你在当前的 ssh-client 用户环境中去以 root 身份登录 192.168.11.164 这台机器的时候，会不用输入密码而自动登录的

当你登录到 192.168.11.164 这台机器后，你会发现在 root 会用的 ~/.ssh 目录下面有一个名为 authorized_keys 的文件，这个文件存储有 ssh-client 发送的公钥信息

当然你也可以选择以不同用户身份自动登录进入 ssh-server，只要将对应的公钥信息拷贝至相应的用户的.ssh/authorized_keys 文件中即可

3、ssh-copy-id 应注意的小地方

Default public key: ssh-copy-id uses ~/.ssh/identity.pub as the default public key file (i.e when no value is passed to option -i). Instead, I wish it uses id_dsa.pub, or id_rsa.pub, or identity.pub as default keys. i.e If any one of them exist, it should copy that to the remote-host. If two or three of them exist, it should copy identity.pub as default. The agent has no identities: When the ssh-agent is running and the ssh-add -L returns "The agent has no identities" (i.e no keys are added to the ssh-agent), the ssh-copy-id will still copy the message "The agent has no identities" to the remote-host's authorized_keys entry. Duplicate entry in authorized_keys: I wish ssh-copy-id validates duplicate entry on the remote-host's authorized_keys. If you execute ssh-copy-id multiple times on the local-host, it will keep appending the same key on the remote-host's authorized_keys file without checking for duplicates. Even with duplicate entries everything works as expected. But, I would like to have my authorized_keys file clutter free.

1.6 Linux 进程管理

主要内容包括：后台进程、任务、进程查看工具使用、性能工具使用

1.6.1 进程与作业

1. 掌握进程的相关概念
2. 熟悉 Linux 中进程的类型和启动方式
3. 学会查看和杀死进程
4. 理解何谓作业控制
5. 掌握实施作业控制的常用命令

进程概述

进程的概念

进程（Proce ss）是一个程序在其自身的虚拟地址空间中的一次执行活动。之所以要创建进程，就是为了使多个程序可以并发的执行，从而提高系统的资源利用率和吞吐量。

进程和程序的概念不同，下面是对这两个概念的比较

- 程序只是一个静态的指令集合；而进程是一个程序的动态执行过程，它具有生命期，是动态的产生和消亡的。
- 进程是资源申请、调度和独立运行的单位，因此，它使用系统中的运行资源；而程序不能申请系统资源、不能被系统调度、也不能作为独立运行的单位，因此，它不占用系统的运行资源。
- 程序和进程无一一对应的关系。一方面一个程序可以由多个进程所共用，即一个程序在运行过程中可以产生多个进程；另一方面，一个进程在生命期内可以顺序的执行若干个程序。

Linux 操作系统是多任务的，如果一个应用程序需要几个进程并发地协调运行来完成相关工作，系统会安排这些进程并发运行，同时完成对这些进程的调度和管理任务，包括 CPU、内存、存储器等系统资源的分配。

Linux 中的进程

在 Linux 系统中总是有很多进程同时在运行，每一个进程都有一个识别号，叫做 PID（Proce ss ID），用以区分不同的进程。系统启动后的第一个进程是 init，它的 PID 是 1。init 是唯一一个由系统内核直接运行的进程。新的进程可以用系统调用 fork 来产生，就是从一个已经存在的旧进程中分出一个新进程来，旧的进程是新产生的进程的父进程，新进程是产生它的进程的子进程，除了 init 之外，每一个进程都有父进程。当系统启动以后，init 进程会创建 login 进程等待用户登录系统，login 进程是 init 进程的子进程。当用户登录系统后，login 进程就会为用户启动 shell 进程，shell 进程就是 login 进程的子进程，而此后用户运行的进程都是由 shell 衍生出来的。

init 进程是所有进程的发起者和控制者。因为在任何基于 Unix 的系统（比如 linux）中，它都是第一个运行的进程，所以 init 进程的编号（ProcessID，PID）永远是 1。如果 init 出现了问题，系统的其余部分也就随之而垮掉了。

init 进程有两个作用。

- 第一个作用是扮演终结父进程的角色。因为 init 进程永远不会被终止，所以系统总是可以确信它的存在，并在必要的时候以它为参照。如果某个进程在它衍生出来的全部子进程结束之前被终止，就会出现必须以 init 为参照的情况。此时那些失去了父进程的子进程就都会以 init 作为它们的父进程。快速执行一下 ps-af 命令，可以列出许多父进程 ID（ParentProcessID，PPID）为 1 的进程来。
- init 的第二个角色是在进入某个特定的运行级别（Runlevel）时运行相应的程序，以此对各种运行级别进行管理。它的这个作用是由/etc/inittab 文件定义的

除了进程识别号外，每个进程还有另外四个识别号。它们是实际用户识别号（real user ID）、实际组识别号以及有效用户识别号（e ffe ct user ID），和有效组识别号（e ffe ct group ID）。实际用户识别号和实际组识别号的作用是识别正在运行此进程的用户和组。一个进程的实际用户识别号和实际组识别号就是运行此进程的用户的识别号（UID）和组的识别号（GID）。有效用户识别号和有效组识别号的作用是确定一个进程对其访问的文件的权限和优先权。除了产生进程的程序被设置 UID 位和 GID 位之外，一般有效用户识别号和有效组识别号和实际用户识别号及实际组识别号相同。如果程序被设置了 UID 位或 GID 位，则此进程相应的有效用户识别号和有效组识别号，将和运行此进程的文件的所属用户的 UID 或所属组的 GID 相同。

例如，一个可执行文件/usr/bin/passwd，其所属用户是 root（UID 为 0），此文件被设置了 UID 位。则当一个 UID 为 500、GID 为 501 的用户执行此命令时，产生的进程的实际用户识别号和实际组识别号分别是 500 和 501，而其有效用户识别号是 0，有效组识别号是 501。

所有这些设计都是为了在一个多用户、多任务的操作系统中，所有用户的工作都能够安全可靠地进行，这也是 Linux 操作系统的优秀性所在。

进程的类型

可以将运行在 Linux 系统中的进程分为三种不同的类型：

- 交互进程：由一个 Shell 启动的进程。交互进程既可以在前台运行，也可以在后台运行。
- 批处理进程：不与特定的终端相关联，提交到等待队列种顺序执行的进程。
- 守护进程：在 Linux 在启动时初始化，需要时运行于后台的进程。

以上三种进程各有各的特点、作用和不同的使用场合。

进程的启动方式

启动一个进程有两个主要途径　手工启动和调度启动。

1、手工启动：由用户输入命令，直接启动一个进程便是手工启动进程。手工启动进程又可以分为前台启动和后台启动。

> I . 前台启动：是手工启动一个进程的最常用的方式。一般地，用户键入一个命令“ls -l”，这就已经启动了一个进程，而且是一个前台的进程。
>
> II. 后台启动：直接从后台手工启动一个进程用得比较少一些，除非是该进程甚为耗时，且用户也不急着需要结果的时候。假设用户要启动一个需要长时间运行的格式化文本文件的进程。为了不使整个 shell 在耗时进程的运行过程中都处于“瘫痪”状态，从后台启动这个进程是明智的选择。在后台启动一个进程，可以在命令行后使用 & 命令，例如：# ls –R / >list &

2、调度启动方式是事先进行设置，根据用户要求让系统自行启动。如安排周期性任务

进程的属性

进程 ID（PID)：是唯一的数值，用来区分进程；

父进程和父进程的 ID（PPID)；

启动进程的用户 ID（UID）和所归属的组（GID)；

进程状态：状态分为运行 R、休眠 S、僵尸 Z；

进程执行的优先级；

进程所连接的终端名；

进程资源占用：比如占用资源大小（内存、CPU 占用量)；

进程优先级

进程优先级的范围:-19~20

系统第一个进程是 init ，由内核启动，后续的所有进程都是她的子进程

一般用户的 nice 值是 0 ～ 19

?root 用户可用的 nice 值是 -20 ～ 19

PRI 是系统动态决定的，虽然 nice 值可以影响 PRI，但是最终的 PRI 还是要经过系统分析后才能决定。

nice 值可以被改变……

? 在启动进程时:

$ nice -n 5 command

? 在启动后：

$ renice 5 PID

只有根用户才能降低 nice 值（提高优先性）

linux 上进程有 5 种状态

1. 运行 (正在运行或在运行队列中等待)
2. 中断 (休眠中, 受阻, 在等待某个条件的形成或接受到信号)
3. 不可中断 (收到信号不唤醒和不可运行, 进程必须等待直到有中断发生)
4. 僵死 (进程已终止, 但进程描述符存在, 直到父进程调用 wait4() 系统调用后释放)
5. 停止 (进程收到 SIGSTOP, SIGSTP, SIGTIN, SIGTOU 信号后停止运行运行)

父进程和子进程

他们的关系是管理和被管理的关系，当父进程终止时，子进程也随之而终止。但子进程终止，父进程并不一定终止。比如 httpd 服务器运行时，我们可以杀掉其子进程，父进程并不会因为子进程的终止而终止。

守护进程简介

内容提要

1. 守护进程的概念
2. 超级服务器的引入
3. 熟悉常见的守护进程

什么是守护进程　Linux 系统在启动时就启动很多进程（例如：init 进程、等待用户登录的进程 login、等待 FTP 客户连接的 vsftpd 等），这些进程向本地和网络用户提供了 Linux 的系统功能接口，直接面向应用程序和用户。将这些进程称为守护进程（daem on）。守护进程是指在后台运行而又没有终端或登录 shell 与之结合在一起的进程。由于此类程序运行在后台，除非程序异常终止或者人为终止，否则它们将一直运行下去直至系统关闭。一般地，守护进程在系统引导装入时启动，在系统关闭时终止。一个实际运行中的系统一般会有多个这样的守护进程在运行。Windows 系统中的守护进程被称为“服务”。

按照服务类型可以分为如下两类：

- 系统守护进程：如 a td、cron、lpd、syslogd、login 等。
- 网络守护进程：如 sshd、httpd、sendm ail、xine td 等。

网络守护进程　在 Client/Server 模型中，服务器监听（Listen）在一个特定的端口上等待客户的连接。连接成功之后客户机与服务器通过端口进行数据通讯。

守护进程的工作就是打开一个端口，并且等待（Listen）进入的连接。如果客户提请了一个连接，守护进程就创建（fork）子进程来响应此连接，而父进程继续监听更多的服务请求。正因为如此，每个守护进程都可以处理多个客户服务请求。

超级守护进程　运行 Linux 的计算机一般都作为服务器使用，提供了许多不同协议的服务。但是，一台繁忙的服务器也可能是专用于某个任务的，比如传输邮件、响应 DNS 请求等。

从守护进程的概念，我们可以看出，对于系统所要提供的每一种服务，都必须运行一个监听某个端口连接发生的守护程序。无论如何，在提供多种服务的 Linux 系统中，系统内存中同时运行二、三十种不同的守护进程是对资源的浪费。解决这个问题的方法是使用超级服务器（SuperServer）。

几乎所有的类 UNIX 系统都运行了一个“超级服务器”，它为众多服务创建套接字（Socke t），并且使用 Socke t 系统调用同时监听多个端口。当远程系统请求一个服务时，超级服务器监听到这个请求后会产生该端口的服务器程序为客户提供服务。使用最广泛的超级服务器程序是 xinetd ，即“扩展网络守护进程”。

xinetd 在运行时读取 /etc 下文本配置文件，在文件中指出超级服务器需要监听的端口以及在数据包到达端口时需要启动的程序。xinetd 具有更先进的配置模式和更好的安全性。

守护进程的运行方式　由于引入了超级服务器，因此守护进程有如下两种运行方式：

1、独立运行的（stand-alone）守护进程

- 独立运行的守护进程由 init 脚本负责管理
- 独立运行的守护进程的脚本存放在 /e tc/init.d/ 目录下
- 所有的系统服务都是独立运行的。如：cron、syslogd 等。

2、由超级服务器（SuperServer）运行的守护进程

- 要运行的守护进程由 ine td/xine td 启动
- 由 xine td 管理的守护进程的配置文件存放在 /e tc/xine td.d/ 目录下, 默认的 xine td 的主配置文件是/etc/xine td.conf
- ine td/xine td 本身是独立运行的守护进程

为了节省资源，引入了超级服务器用于监控网络服务，如 telne t、talk 等。使用超级服务器启动网络服务虽然可以节省资源，但是对于服务量很大的守护进程（如 HTTP 服务、FTP 服务）将影响到其他服务的运行，同时也影响所提供服务的响应速度。为此，某些常用的知名网络服务的守护进程需要单独启动。

哪些守护进程可以使用超级服务器启动?

几乎所有的网络服务程序都可以由超级服务器来启动，而具体提供哪些服务将由 /e tc/service s 文件指出。这个文件中说明了超级服务器可提供服务的端口号和名字。

查看守护进程树　可以使用 pstree 命令查看守护进程树。例如：

```
$ pstree
init─┬─acpid
     ├─apache2─────5*[apache2]
     ├─atd
     ├─cron
     ├─dd
     ├─events/0
     ├─6*[getty]
     ├─gpm
     ├─khelper
     ├─kjournald
     ├─klogd
     ├─ksoftirqd/0
     ├─kswapd0
     ├─kthread─┬─aio/0
     │         ├─kacpid
     │         ├─kblockd/0
     │         ├─kgameportd
     │         ├─khubd
     │         ├─kseriod
     │         ├─2*[pdflush]
     │         ├─reiserfs/0
     │         └─scsi_eh_0
     ├─master─┬─pickup
     │        └─qmgr
     ├─mdadm
     ├─migration/0
     ├─mysqld_safe─┬─logger
     │             └─mysqld─────8*[{mysqld}]
     ├─nmbd
     ├─proftpd
     ├─shpchpd_event
     ├─smbd─────smbd
     ├─sshd─────sshd─────sshd─────bash─────pstree
     ├─syslogd
     ├─udevd
     ├─watchdog/0
     └─xinetd
```

守护进程的启用和停止　1、独立运行的守护进程的启用和停止

1)/etc/init.d/server-name start|stop|restart|reload

2)service server-name start|stop|restart|reload

2、由超级服务器运行的守护进程的启用和停止

1) 修改 /etc/xine td.d/ 目录下的相关文件

- 启用服务，使用 disable = no 选项
- 停用服务，使用 disable = yes 选项

2) 重新启动超级服务器

/etc/init.d/xinetd restart

service xinetd restart

3、使用 chkconfig 管理启动脚本

可以使用 chk config 命令检查、设置系统的各种服务。此命令实际上是通过操控 /etc/rc[0-6].d 目录下的符号链接文件对系统的各种服务进行管理。

可以使用 chk config 命令检查、设置系统的各种服务。此命令实际上是通过操控 /etc/rc[0-6].d 目录下的符号链接文件对系统的各种服务进行管理。

chk config 命令具有如下功能：

添加指定的新服务

清除指定的服务

显示由 chk config 管理的服务

改变服务的运行级别

检查指定服务的启动状态

chk config 命令的格式如下：

```
# chkconfig –list [server-name]
# chkconfig –add server-name
# chkconfig –del server-name
# chkconfig [–level <levels>] server-name <on|off|reset|resetpriorities>
```

其中：

server-name：是由 chk config 命令管理的服务的名字。

–list：显示由 chk config 管理的所有服务。

–level <levels>：指定某服务要在哪个运行级别中开启或关闭，<levels> 的范围在 0-6 之间。

–add：添加由 chk config 进行管理的指定服务。

–del：删除由 chk config 进行管理的指定服务。

on|off：在指定的运行级别，开启或关闭服务。不指定运行级别时，默认的运行级别是 3、4、5。

reset：在指定的运行级别，重置该服务，使其状态返回到操作系统启动时的默认状态。

例如：

1) 查看指定的服务在所有运行级别的运行状态。

```
# chkconfig –list sendmail
```

) 显示由 chk config 管理的所有服务。

```
# chkconfig –list
```

3) 添加一个由 chk config 管理的服务。

```
# chkconfig –add httpd
```

4) 更改指定服务在指定运行级别的运行状态。

```
# chkconfig –level 35 httpd on
# chkconfig httpd on
# chkconfig –level 4 sendmail off
```

5) 启动或停用由 xine td 运行的服务

```
# chkconfig rsync on # 相当于配置文件中的 “disable = no”
# chkconfig rsync off # 相当于配置文件中的 “disable = yes”
```

管理开机时守护进程的的启用状态　若要管理守护进程在计算机启动过程中是否启动，Redhat 系列操作系统可以使用 ntsysv。

执行如下命令运行 ntsysv：

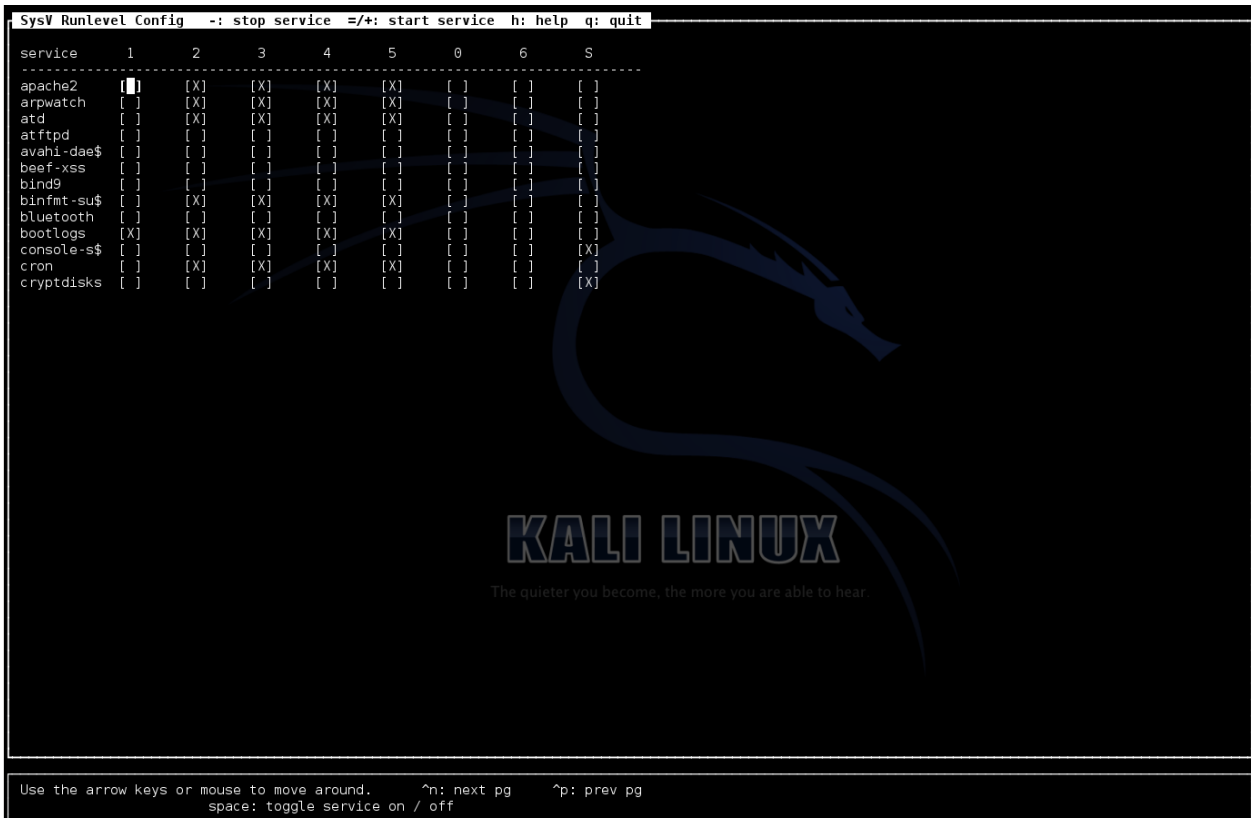
```
# ntsysv
```

使用 ntsysv 管理开机时守护进程的的启用状态

可以使用上下方向键移动光标选择操作对象，使用空格键激活或终止服务（[*] 表示激活；[ ] 表示终止）。操作结束单击“确定”按钮结束（也可以单击“取消”按钮取消操作）。这样在下次启动机器时，修改将生效。

Debian 系列操作系统可以使用 sysv-rc-conf

#sysv-rc-conf



作业控制

什么是作业控制

作业控制是指控制当前正在运行的进程的行为，也称为进程控制。作业控制是 Shell 的一个特性，使用户能在多个独立进程间进行切换。例如，用户可以挂起一个正在运行的进程，稍后再恢复它的运行。bash 记录所有启动的进程并保持对所有已启动的进程的跟踪，在每一个正在运行的进程的生命期内的任何时候，用户可以任意地挂起进程或重新启动进程恢复运行。

例如，当用户使用 Vi 编辑一个文本文件，并需要中止编辑做其他事情时，利用作业控制，用户可以让编辑器暂时挂起，返回 Shell 提示符开始做其他的事情。其他事情做完以后，用户可以重新启动挂起的编辑器，返回到刚才中止的地方，就像用户从来没有离开编辑器一样。这只是一个例子，作业控制还有许多其他实际的用途。

实施作业控制的常用命令

下表列出了作业控制的常用命令或操作快捷键。

命令或快捷键	功能说明
cmd &	命令后的&符号表示将该命令放到后台运行，以免霸占终端
<Ctrl+d>	终止一个正在前台运行的进程（含有正常含义）
<Ctrl+c>	终止一个正在前台运行的进程（含有强行含义）
<Ctrl+z>	挂起一个正在前台运行的进程
jobs	显示后台作业和被挂起的进程
bg	重新启动一个挂起的作业，并且在后台运行
fg	把一个在后台运行的作业放到前台来运行

这些命令经常用于用户需要在后台运行，而却意外地把它放到了前台启动运行的时候。当一个命令在前台被启动运行时，它会禁止用户与 Shell 的交互，直到该命令结束。由于大多数命令的执行都能很快完成，所以一般情况下不会有什么问题。但是如果要运行的命令要花费很长时间的话，我们通常会把它放到后台，以便能在前台继续输入其他命令。此时，上面的命令就会派上用场了。

在进行作业控制时经常使用如下的作业标识符：

作业标识符	说明
%N	第N号作业
%S	以字符串S开头的被命令行调用的作业
%?S	包含字符串S的被命令行调用的作业
%+	默认作业(前台最后结束的作业，或后台最后启动的作业)，等同于 %%
%-	第二默认作业

作业控制举例

下面举一个简单的例子说明作业控制命令的使用。

```
# 列出所有正在运行的作业
$ jobs
# 在前台运行睡眠进程
$ sleep 100000
# 使用 Ctrl+z 挂起
[1]+ Stopped sleep 100000
# 在前台运行睡眠进程
$ sleep 200000
# 使用 Ctrl+z 挂起
[2]+ Stopped sleep 200000
# 在后台运行睡眠进程
$ sleep 300000 &
[3] 8941
# 运行 cat 命令
```

```
$ cat >example
This is a example.
# 使用 Ctrl+z 挂起
[4]+ Stopped cat >example
# 列出所有正在运行的作业
# 第 1 列是作业号，第 2 列中的 + 表示默认作业；-表示第二默认作业，第 3 列是作业状态
$ jobs
[1] Stopped sleep 100000
[2]- Stopped sleep 200000
[3] Running sleep 300000 &
[4]+ Stopped cat >example
# 列出所有正在运行的作业，同时列出进程 PID
$ jobs -l
[1] 8939 Stopped sleep 100000
[2]- 8940 Stopped sleep 200000
[3] 8941 Running sleep 300000 &
[4]+ 8942 Stopped cat >example
# 将第二默认作业（以 -标识）在后台继续运行
$ bg %-
[2]- sleep 200000 &
$ jobs -l
[1]- 8939 Stopped sleep 100000
[2] 8940 Running sleep 200000 &
[3] 8941 Running sleep 300000 &
[4]+ 8942 Stopped cat >example
# 将 1 号作业在后台继续运行
$ bg %1
[1]- sleep 100000 &
$ jobs -l
[1] 8939 Running sleep 100000 &
[2] 8940 Running sleep 200000 &
[3]- 8941 Running sleep 300000 &
[4]+ 8942 Stopped cat >example
# 将默认作业（以 + 标识）在前台继续运行
# fg 等同于 fg %+；bg 等同于 bg %+
$ fg
```

```
cat >example
# 使用 Ctrl+d 结束进程
$ jobs -l
[1] 8939 Running sleep 100000 &
[2]- 8940 Running sleep 200000 &
[3]+ 8941 Running sleep 300000 &
# 杀死 1 号作业
$ kill %1
$ jobs -l
[1] 8939 Terminated sleep 100000
[2]- 8940 Running sleep 200000 &
[3]+ 8941 Running sleep 300000 &
# 杀死默认作业（以 + 标识）
$ kill %+
$ jobs -l
[2]- 8940 Running sleep 200000 &
[3]+ 8941 Terminated sleep 300000
```

断了远程连接后继续执行后台 job

我们常常有这样的需求：想退出 secureCRT 后，能够继续跑自己的进程。为什么会有这样的需求？作为系统管理员，经常遇到这样的问题，用 telnet/ssh 登录了远程的 Linux 服务器，需要运行了一些耗时较长的任务，例如批量 ping 一些网段之类，有时候却由于网络的不稳定导致任务中途失败，或者需要中途离开，总不会在等它结束吧，如果你退出 SSH 登陆的话，那么你的任务也会被终止了，岂不是白费精。力了？如何让命令或者任务在后台自己的运行，可以有很多方式实现，向大家都不陌生了，例如 nohup，setsid 和 screen 等等，我就简单说说吧。

在我们通过 SSH 登陆服务器后，一般来说，所做的操作或者命令的输入都是属 sshd 下的 shell 的子进程，例如打开个 SSH 终端，输入 ping www.163.com >>output.txt &，然后查看进程情况：

```
$ ps -ef|grep ping
sszheng 27491 27467 0 10:20 pts/0 00:00:00 ping www.163.com
sszheng 27535 27467 0 11:40 pts/0 00:00:00 grep ping
```

很显然它是 shell 的子进程，命令由一个子 shell 在后台执行，当前 shell（27467）立即取得控制等候用户输入，所以我的 grep 就可以使用了。后台命令和当前 shell 的执行是并行的，他们没有互相的依赖、等待关系，所以是异步的并行。现在问题来了，如果 ssh 退出了，bash 结束了，那么这个工作过程如何呢？后台执行的能否继续下去？

这里涉及到两个问题，就是退出 ssh 后，在我们 exit 执行的 shell 时候，会不会向我们后台的 jobs 发送 SIGHUP 信号呢？如果发送了 SIGHUP 信号，那么所有该 shell 下运行的进程都会被终止，也就是所希望的后台执行没有实现。在 shell 的 options 中，有 huponexit 这个选项，意思就是退出 shell 时候，是否发送这个 SIGHUP 信号？

```
$ shopt
cdable_vars off
```

cdspell off

checkhash off

checkwinsize off

cmdhist on

dotglob off

execfail off

expand_aliases on

extdebug off

extglob off

extquote on

failglob off

force_fignore on

gnu_errfmt off

histappend off

histreedit off

histverify off

hostcomplete on

huponexit off

interactive_comments on

lithist off

login_shell on

mailwarn off

no_empty_cmd_completion off

nocaseglob off

nocasematch off

nullglob off

progcomp on

promptvars on

restricted_shell off

shift_verbose off

sourcepath on

xpg_echo off

上面的默认选项中，huponexit off，这个情况时候，当你退出 shell 时候，后台的程序还会继续运行，但是这个是全局选项，有时候我们往往希望退出 shell 后，shell 发起的进程相应结束了，而不是一直运行，因为有时候你可能开了很多子进程，没有时间去一一关闭吧?? 往往这个选项是建议打开的。

huponexit 打开后，所以后台进行的 jobs，在 shell 退出后就会相应退出了，但是针对我们特定的任务时候，我们可以对它进行单独操作，可以有下面集中方法。

1、nohup

nohup 的用途就是让提交的命令忽略 hangup 信号，使用方法：

$nohup ping www.163.com & 如果没有重定向输入和输出的话，标准输出和标准错误缺省会被重定向到 nohup.out 文件中。一般像示例一样，加上"&" 来将命令同时放入后台运行，也可用">filename 2>&1" 来更改缺省的重定向文件名。退出 shell 后，ping 会继续运行，直到命令执行结束。

$ ps -ef |grep ping

sszheng 5377 5311 0 16:51 pts/1 00:00:00 ping www.163.com

sszheng 5379 5311 0 16:51 pts/1 00:00:00 grep ping

退出 shell 后，重新登陆查看，ping 进程依然在执行，只不过他的 PPID 变成了 1，也就是被 init 所管理的孤儿进程了，稍后说一下孤儿进程。

$ ps -ef |grep ping

sszheng 5377 1 0 16:51 ? 00:00:00 ping www.163.com

sszheng 5389 5383 0 16:52 pts/0 00:00:00 grep ping

2、setsid

> nohup 是通过忽略 HUP 信号来使进程避免中途被中断，也可以用另一种方法，进程是不属于接受 HUP 信号的终端的 shell 子进程，那么自然也就不会受到 HUP 信号的影响了，真是白猫黑猫，抓到老鼠就是好猫，呵呵，废话多了

shell 提供了 setsid 这个方法

$setsid ping www.163.com & >>163.txt

$ ps -ef |grep ping

sszheng 5377 1 0 16:51 ? 00:00:00 ping www.163.com

sszheng 5395 1 0 16:56 ? 00:00:00 ping www.163.com

sszheng 5397 5383 0 16:57 pts/0 00:00:00 grep ping

大家应该注意到，上一个示例中，ping 的父进程是 5311，当它的父进程退出后，它才被 init（PID=1）收养，而 setsid 直接把 ping(pid=5395) 给 init 了，那么就无所谓的 shell 退出影响了。

3、(&)

再提一下关于 subshell 的使用，我们知道，将一个或多个命名包含在“()”中就能让这些命令在子 shell 中运行中，当我们将"&" 也放入“()”内之后，我们就会发现所提交的作业并不在作业列表中，也就是说，是无法通过 jobs 来查看的。看看下面的进程 id 就知道了：

$（ping www.163.com &）

$ ps -ef |grep ping

sszheng 5377 1 0 16:51 ? 00:00:00 ping www.163.com

sszheng 5395 1 0 16:56 ? 00:00:00 ping www.163.com

sszheng 5401 1 0 17:03 pts/0 00:00:00 ping www.163.com

sszheng 5403 5383 0 17:03 pts/0 00:00:00 grep ping

可以看到，执行的 5401 的父进程是 init 了，这样子也可以达到忽略 hup 信号的目的了。

说到这里，相信大家都略明白后台执行的方法了，简单说下原理：bash 进程终止后，init 进程会接管父进程留下的这些“孤儿进程”，所以 PPID 是 1 了，孤儿进程不是僵尸进程，下面是他们的概念和区别

- 僵尸进程：一个子进程在其父进程还没有调用 wait() 或 waitpid() 的情况下退出。这个子进程就是僵尸进程。
- 孤儿进程：一个父进程退出，而它的一个或多个子进程还在运行，那么那些子进程将成为孤儿进程。孤儿进程将被 init 进程 (进程号为 1) 所收养，并由 init 进程对它们完成状态收集工作。

1.6.2 工具篇

工欲善其事必先利其器，go！

ps

ps 为我们提供了进程的一次性的查看，它所提供的查看结果并不动态连续的；如果想对进程时间监控，应该用 top 工具；

使用 ps，我们：

—可以确定有哪些进程正在执行和执行的状态

—进程是否结束、进程有没有僵死

—哪些进程占用了过多的系统资源等

ps 默认显示当前终端进程

-a 包括所有终端的进程

-x 包括不属于终端的进程

-u 打印进程所有者信息

-f 选项显示进程的父进程

-e 含义同 -a

-o 属性…选项显示定制的信息：pid、comm、%cpu、%mem、state、tty、euser、ruser

-H 显示树状结构，表示程序间的相互关系

-c 列出程序时，显示每个程序真正的指令名称，而不包含路径，参数或常驻服务的标示。

-u xxx 显示指定用户的进程信息

-t< 终端机编号 **>** 指定终端机编号，并列出属于该终端机的程序的状况。

[wenjian@r520-85 ~]$ ps aux |grep httpd

USER PID %CPU %MEM VSZ RSS tty STAT TIME COMMAND

wenjian 10383 0.0 0.0 4624 1628 ? Ss Aug01 0:05 httpd -k start

wenjian 10385 0.0 0.0 4768 1800 ? S Aug01 0:00 httpd -k start

wenjian 10386 0.0 0.0 4768 1796 ? S Aug01 0:00 httpd -k start

wenjian 10388 0.0 0.0 4768 1808 ? S Aug01 0:00 httpd -k start

wenjian 11706 0.0 0.0 4768 1804 ? S Aug01 0:00 httpd -k start

wenjian 11730 0.0 0.0 4768 1808 ? S Aug01 0:00 httpd -k start

wenjian 11741 0.0 0.0 4768 1800 ? S Aug01 0:00 httpd -k start

wenjian 22231 0.0 0.0 4768 1804 ? S Aug01 0:00 httpd -k start

wenjian 33806 0.0 0.0 4768 1764 ? S 09:10 0:00 httpd -k start

wenjian 43835 0.0 0.0 4628 1372 ? S 09:22 0:00 httpd -k start

wenjian 43926 0.0 0.0 4768 1788 ? S Aug01 0:00 httpd -k start

wenjian 45652 0.0 0.0 103240 876 pts/0 S+ 09:24 0:00 grep httpd

USER 域指明了是哪个用户启动了这个命令

PID 表示进程号

%CPU 表示进程占用了多少 CPU

%MEM 表示进程占用了多少内存

VSZ 表示如果一个程序完全驻留在内存的话需要占用多少内存空间

RSS 表示目前进程实际占用的内存大小

TTY 表示进程从哪一个终端启动，不是从终端启动的进程则显示为?

STAT 表示当前进程的状态

- D 不可中断 Uninterruptible（usually IO）
- R 正在运行，或在队列中的进程
- S 处于休眠状态
- T 停止或被追踪
- Z 僵尸进程
- W 进入内存交换（从内核 2.6 开始无效）
- X 死掉的进程
- < 高优先级
- n 低优先级
- s 包含子进程
- \+ 位于后台的进程组

top

1、top 参数详解：

top - 10:07:59 up 60 days, 18:02, 3 users, load average: 1.58, 1.84, 1.30

Tasks: 517 total, 1 running, 516 sleeping, 0 stopped, 0 zombie

Cpu(s): 6.2%us, 0.3%sy, 0.0%ni, 93.5%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st

Mem: 32830828k total, 24500432k used, 8330396k free, 307940k buffers

Swap: 35061752k total, 12516k used, 35049236k free, 6253084k cached

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND

11517 radius 20 0 9228m 5.2g 10m S 93.3 16.5 1908:32 /home/radius/jdk1.7.0_25/bin/java -Dserver=MemoryDB -Dserver.home=/home/radius/memdb2_wangxl -Ddb.poolname=

11140 radius 20 0 8969m 5.2g 10m S 47.8 16.5 1909:07 /home/radius/jdk1.7.0_25/bin/java -Dserver=MemoryDB -Dserver.home=/home/radius/memdb_wangxl -Ddb.poolname=m

36802 radius 20 0 1396m 45m 10m S 1.6 0.1 105:36.43 radius_3rj 10 d

26440 radius 20 0 1228m 995m 672 S 1.3 3.1 293:47.93 ./lm_rj2 -xx -d

41554 wenjian 20 0 9056 1408 1008 S 1.3 0.0 65:24.49 x_agent -c x_agent.ini -l 2 -e -d 2

**统计信息区 **

前五行是系统整体的统计信息。第一行是任务队列信息，同 uptime 命令的执行结果。其内容如下

10:07:59 当前时间

up 60 days, 18:02 系统运行时间，格式为时: 分

3 user 当前登录用户数

load average: 1.58, 1.84, 1.30 系统负载，即任务队列的平均长度。

三个数值分别为 1 分钟、5 分钟、15 分钟前到现在的平均值。

第二、三行为进程和 CPU 的信息。当有多个 CPU 时，这些内容可能会超过两行。内容如下：

Tasks: 29 total 进程总数

1 running 正在运行的进程数

28 sleeping 睡眠的进程数

0 stopped 停止的进程数

0 zombie 僵尸进程数

Cpu(s): 0.3% us 用户空间占用 CPU 百分比

1.0% sy 内核空间占用 CPU 百分比

0.0% ni 用户进程空间内改变过优先级的进程占用 CPU 百分比

98.7% id 空闲 CPU 百分比

0.0% wa 等待输入输出的 CPU 时间百分比

0.0% hi

0.0% si

最后两行为内存信息。内容如下：

Mem: 191272k total 物理内存总量

173656k used 使用的物理内存总量

17616k free 空闲内存总量

22052k buffers 用作内核缓存的内存量

Swap: 192772k total 交换区总量

0k used 使用的交换区总量

192772k free 空闲交换区总量

123988k cached 缓冲的交换区总量。

内存中的内容被换出到交换区，而后又被换入到内存，但使用过的交换区尚未被覆盖，

该数值即为这些内容已存在于内存中的交换区的大小。

相应的内存再次被换出时可不必再对交换区写入。

PID 表示进程的进程号

USER 表示进程的用户

PR 表示进程的优先级

NI nice 值。负值表示高优先级，正值表示低优先级 (-19 - 20)

VIRT 进程使用的虚拟内存总量，单位 kb。VIRT=SWAP+RES

RES 进程使用的、未被换出的物理内存大小，单位 kb。RES=CODE+DATA

SWAP 进程使用的虚拟内存中，被换出的大小，单位 kb。

SHR 共享内存大小，单位 kb

%CPU 上次更新到现在的 CPU 时间占用百分比

%MEM 进程使用的物理内存百分比

TIME+ 进程使用的 CPU 时间总计，单位 1/100 秒

COMMAND 表示启动进程的命令

2、top 中快捷键

> 向后翻页

< 向前翻页

k 干掉一个进程，按下 k 命令之后，输入要 kill 的进程 pid 回车即可

h 查看帮助

r 调整进程优先级，输入完命令之后按提示操作即可

s 改变刷新的时间间隔，输入完命令之后按提示操作即可

r 调整一个进程的优先级

q 退出 top

- t 显示摘要信息开关.
- m 显示内存信息开关.
- A 分类显示系统不同资源的使用，有助于快速识别系统中资源消耗多的任务

z 彩色/黑白显示开关

u 查看指定按下 u 命令之后，界面提示输入用户名，输入即可

W 将当前的设置写入 ~/.toprc

一个进程状态后面 < 表示优先级高

N 表示优先级低

3、top 常用参数

-u 只显示指定用户的进程

-b 批处理模式，可以使用这个参数将 top 输出重定向到文件中

-n top 隔多久执行一次，一般和 b 结合用

-p pid 监控指定 pid 的进程

-u 监视指定用户的进程信息

pgrep

1、pgrep -U wenjian -l

显示出所有用户名为 wenjian 的进程 ID 和对应的 daemon 程序

2、pgrep -l httpd

显示 httpd 的程序 (包括主进程和子进程) 的 pid

3、pgrep -lo httpd

列出最早启动的 apache 进程 ID，也就是主进程的 PID

-l 表示显示 pid 对应的程序名

4、pgrep -f httpd 和 pgrep httpd 效果一样

f 参数可以匹配 command 中的关键字

5、pgrep -ln httpd

列出最新启动的 apache 进程 ID，-l 参数用来显示进程名称；

lsof

lsof（list open files）是一个列出当前系统打开文件的工具。在 linux 环境下，任何事物都以文件的形式存在，通过文件不仅仅可以访问常规数据，还可以访问网络连接和硬件。

在终端下输入 lsof 即可显示系统打开的文件，因为 lsof 需要访问核心内存和各种文件，所以必须以 root 用户的身份运行它才能够充分地发挥其功能。

COMMAND PID USER FD TYPE DEVICE SIZE NODE NAME

init 1 root cwd DIR 3,3 1024 2 /

init 1 root rtd DIR 3,3 1024 2 /

init 1 root txt REG 3,3 38432 1763452 /sbin/init

init 1 root mem REG 3,3 106114 1091620 /lib/libdl-2.6.so

init 1 root mem REG 3,3 7560696 1091614 /lib/libc-2.6.so

init 1 root mem REG 3,3 79460 1091669 /lib/libselinux.so.1

init 1 root mem REG 3,3 223280 1091668 /lib/libsepol.so.1

init 1 root mem REG 3,3 564136 1091607 /lib/ld-2.6.so

init 1 root 10u FIFO 0,15 1309 /dev/initctl

每行显示一个打开的文件，若不指定条件默认将显示所有进程打开的所有文件。lsof 输出各列信息的意义如下：

COMMAND：进程的名称

PID：进程标识符

USER：进程所有者

FD：文件描述符，应用程序通过文件描述符识别该文件。如 cwd、txt 等

TYPE：文件类型，如 DIR、REG 等

DEVICE：指定磁盘的名称

SIZE：文件的大小

NODE：索引节点（文件在磁盘上的标识）

NAME：打开文件的确切名称

其中 FD 列中的文件描述符 cwd 值表示应用程序的当前工作目录，这是该应用程序启动的目录，除非它本身对这个目录进行更改。

txt 类型的文件是程序代码，如应用程序二进制文件本身或共享库，如上列表中显示的 /sbin/init 程序

其次数值表示应用程序的文件描述符，这是打开该文件时返回的一个整数。如上的最后一行文件/dev/initctl，其文件描述符为 10

u 表示该文件被打开并处于读取/写入模式，而不是只读? 或只写 (w) 模式。

同时还有大写的 W 表示该应用程序具有对整个文件的写锁. 该文件描述符用于确保每次只能打开一个应用程序实例

初始打开每个应用程序时，都具有三个文件描述符，从 0 到 2，分别表示标准输入、输出和错误流。所以大多数应用程序所打开的文件的 FD 都是从 3 开始

与 FD 列相比，Type 列则比较直观。文件和目录分别称为 REG 和 DIR。而 CHR 和 BLK，分别表示字符和块设备；或者 UNIX、FIFO 和 IPv4，分别表示 UNIX 域套接字、先进先出 (FIFO) 队列和网际协议 (IP) 套接字。

lsof 常见的用法是查找应用程序打开的文件的名称和数目。可用于查找出某个特定应用程序将日志数据记录到何处，或者正在跟踪某个问题。例如，linux 限制了进程能够打开文件的数目。通常这个数值很大，所以不会产生问题，并且在需要时，应用程序可以请求更大的值（直到某个上限）。如果你怀疑应用程序耗尽了文件描述符，那么可以使用 lsof 统计打开的文件数目，以进行验证。lsof 语法格式是：

lsof ［options］filename

1、常用的参数列表：

选项	说明
-h	显示使用帮助信息
-a	表示所列出的选项是"与"逻辑，都必须满足时才显示结果
-R	显示进程的 PPID 列
-l	不将 UID 转化为用户登录名
-n	不将 IP 转换为主机名
-P	不将服务端口号转化为服务名称
-u Username/UID	显示由属于指定用户的进程打开的文件
-g gid	显示属于指定组的进程打开的文件
-d FD	显示指定文件描述符（file descriptors）的进程
-c string	显示命令列中包含指定字符串 string 的进程打开的文件
-c /string/	与上面的功能相同，/ / 中可以使用正则表达式
+d Dirname	显示指定目录下被进程打开的文件
+D Dirname	与上面的功能相同，但是会搜索目录下的所有子目录
-i	显示所有网络进程打开的文件

下面着重谈一下 -i 选项，它可跟如下参数进行输出限制：

```
-i [46][protocol][@hostname|hostaddr][:service|port]
```

其中：

- **4** -- IPv4
- **6** -- IPv6
- **protocol** -- TCP 或 UDP
- **hostname** -- 网络主机名
- **hostaddr** -- IP 地址
- **service** -- /etc/service 中的服务名
- **port** -- 服务端口号

下面给出几个使用 -i 选项的例子

- **-i 6** -- 仅限于 IPv6
- **-i TCP:25** -- TCP 且端口号为 25
- **-i @1.2.3.4** -- IPv4 地址为 1.2.3.4
- **-i @[3ffe:1ebc::1]:1234** -- IPv6 地址为 3ffe:1ebc::1，端口号为 1234
- **-i UDP:who** - UDP 协议的 who 服务端口
- **-i TCP@lsof.itap:513** -- TCP 协议的 513 端口，主机名为 lsof.itap

2、常用输出项

COMMAND	进程的名称
PID	进程标识符
USER	进程所有者
FD	文件描述符，应用程序通过文件描述符识别该文件
TYPE	文件类型
DEVICE	磁盘的名称
SIZE	文件的大小
NODE	索引节点（文件在磁盘上的标识）
NAME	打开文件的确切名称

3、常见的文件描述符

cwd	程序的当前工作目录
rtd	根目录
txt	程序文本（包括代码和数据）
mem	内存映像文件
n	n为数值，应用程序的文件描述符，这是打开该文件时返回的一个整数。（0 到 2，分别表示标准输入、标准输出和标准错误输出） n 后的 r 表示打开的文件只读；w 表示打开的文件只写；u 表示打开的文件可读写。

4、常见的文件类型（TYPE）

REG	普通文件
LINK	符号链接文件
DIR	目录
CHR	字符设备
BLK	块设备
FIFO	先进先出队列
unix	UNIX 域套接字
sock	不可知域套接字
inet	Internet域套接字
IPv4	IPv4 套接字
IPv6	IPv6 网络文件

5、lsof 使用实例

1) 查找谁在使用文件系统

在卸载文件系统时，如果该文件系统中有任何打开的文件，操作通常将会失败。那么通过 lsof 可以找出那些进程在使用当前要卸载的文件系统，如下：

lsof /GTES11/

COMMAND PID USER FD TYPE DEVICE SIZE NODE NAME

bash 4208 root cwd DIR 3,1 4096 2 /GTES11/

vim 4230 root cwd DIR 3,1 4096 2 /GTES11/

2) 查看 22 端口现在运行的情况

lsof -i :22

COMMAND PID USER FD TYPE DEVICE SIZE NODE NAME

sshd 1409 root 3u IPv6 5678 TCP *:ssh (LISTEN)

3) 查看所属 root 用户进程所打开的文件类型为 txt 的文件:

lsof -a -u root -d txt

COMMAND PID USER FD TYPE DEVICE SIZE NODE NAME

init 1 root txt REG 3,3 38432 1763452 /sbin/init

mingetty 1632 root txt REG 3,3 14366 1763337 /sbin/mingetty

mingetty 1633 root txt REG 3,3 14366 1763337 /sbin/mingetty

mingetty 1634 root txt REG 3,3 14366 1763337 /sbin/mingetty

mingetty 1635 root txt REG 3,3 14366 1763337 /sbin/mingetty

mingetty 1636 root txt REG 3,3 14366 1763337 /sbin/mingetty

mingetty 1637 root txt REG 3,3 14366 1763337 /sbin/mingetty

kdm 1638 root txt REG 3,3 132548 1428194 /usr/bin/kdm

X 1670 root txt REG 3,3 1716396 1428336 /usr/bin/Xorg

kdm 1671 root txt REG 3,3 132548 1428194 /usr/bin/kdm

startkde 2427 root txt REG 3,3 645408 1544195 /bin/bash

在这个示例中，用户 root 正在其/GTES11 目录中进行一些操作。一个 bash 是实例正在运行，并且它当前的目录为/GTES11，另一个则显示的是 vim 正在编辑/GTES11 下的文件。要成功地卸载/GTES11，应该在通知用户以确保情况正常之后，中止这些进程。这个示例说明了应用程序的当前工作目录非常重要，因为它仍保持着文件资源，并且可以防止文件系统被卸载。这就是为什么大部分守护进程（后台进程）将它们的目录更改为根目录、或服务特定的目录（如 sendmail 示例中的 /var/spool/mqueue）的原因，以避免该守护进程阻止卸载不相关的文件系统。

3) 恢复删除的日志文件

当 Linux 计算机受到入侵时，常见的情况是日志文件被删除，以掩盖攻击者的踪迹。管理错误也可能导致意外删除重要的文件，比如在清理旧日志时，意外地删除了数据库的活动事务日志。有时可以通过 lsof 来恢复这些文件。

当进程打开了某个文件时，只要该进程保持打开该文件，即使将其删除，它依然存在于磁盘中。这意味着，进程并不知道文件已经被删除，它仍然可以向打开该文件时提供给它的文件描述符进行读取和写入。除了该进程之外，这个文件是不可见的，因为已经删除了其相应的目录索引节点。

在/proc 目录下，其中包含了反映内核和进程树的各种文件。/proc 目录挂载的是在内存中所映射的一块区域，所以这些文件和目录并不存在于磁盘中，因此当我们对这些文件进行读取和写入时，实际上是在从内存中获取相关信息。大多数与 lsof 相关的信息都存储于以进程的 PID 命名的目录中，即 /proc/1234 中包含的

是 PID 为 1234 的进程的信息。每个进程目录中存在着各种文件，它们可以使得应用程序简单地了解进程的内存空间、文件描述符列表、指向磁盘上的文件的符号链接和其他系统信息。lsof 程序使用该信息和其他关于内核内部状态的信息来产生其输出。所以 lsof 可以显示进程的文件描述符和相关的文件名等信息。也就是我们通过访问进程的文件描述符可以找到该文件的相关信息

当系统中的某个文件被意外地删除了，只要这个时候系统中还有进程正在访问该文件，那么我们就可以通过 lsof 从/proc 目录下恢复该文件的内容。假如由于误操作将/var/log/messages 文件删除掉了，那么这时要将/var/log/messages 文件恢复的方法如下：

首先使用 lsof 来查看当前是否有进程打开/var/logmessages 文件，如下：

lsof |grep /var/log/messages

syslogd 1283 root 2w REG 3,3 5381017 1773647 /var/log/messages (deleted)

从上面的信息可以看到 PID 1283（syslogd）打开文件的文件描述符为 2。同时还可以看到/var/log/messages 已经标记被删除了。因此我们可以在 /proc/1283/fd/2 （fd 下的每个以数字命名的文件表示进程对应的文件描述符）中查看相应的信息，如下:

```
# head -n 10 /proc/1283/fd/2
```

Aug 4 13:50:15 holmes86 syslogd 1.4.1: restart.

Aug 4 13:50:15 holmes86 kernel: klogd 1.4.1, log source = /proc/kmsg started.

Aug 4 13:50:15 holmes86 kernel: Linux version 2.6.22.1-8 (root@everestbuilder.linux-ren.org) (gcc version 4.2.0) #1 SMP Wed Jul 18 11:18:32 EDT 2007

Aug 4 13:50:15 holmes86 kernel: BIOS-provided physical RAM map:

Aug 4 13:50:15 holmes86 kernel: BIOS-e820: 0000000000000000 - 000000000009f000 (usable)

Aug 4 13:50:15 holmes86 kernel: BIOS-e820: 000000000009f000 - 00000000000a0000 (reserved)

从上面的信息可以看出，查看 /proc/1283/fd/2 就可以得到所要恢复的数据。如果可以通过文件描述符查看相应的数据，那么就可以使用 I/O 重定向将其复制到文件中，如:

cat /proc/1283/fd/2 > /var/log/messages

对于许多应用程序，尤其是日志文件和数据库，这种恢复删除文件的方法非常有用。

kill 与 killall

kill 和 pid 在一起

1、kill -9 pid

-9 表示强制终止信号

2、kill -s 9 pid

效果与 1 中等效

kill -s SIGKILL #SIGKILL 与 9 信号等效

3、kill -HUP pid # 重启进程

kill -l HUP pid

kill -1 pid

三者等效

4、kill -TERM PPID

kill -15 PPID

两者等效

给父进程发送一个 TERM 信号，试图杀死它和它的子进程。

两者使用形式差不多，效果也差不多，写在一起

killall 与程序名一起

1、killall 程序名