

Linux

知识点梳理及总结



Мэн Цзянин 3530904/90002

GITHUB | [GITHUB.COM/NEKOSILVERFOX](https://github.com/nekosilverfox)

仅供学习与交流使用，禁止商用

目录

一. 简介	5
二. Linux 系统安装	6
三. 注意事项	8
四. 文件处理命令	10
4.1 文件处理命令	10
4.1.1 命令格式与目录处理命令 ls	10
4.1.3 文件处理命令	11
4.1.4 链接命令	12
4.2 权限管理命令	13
4.2.1 权限管理命令 chmod	13
4.2.2 其他权限管理命令 – 对文件/目录而言	14
4.3 文件搜索命令	15
4.3.1 文件搜索命令 find	15
4.3.2 其他搜索命令	16
4.4 帮助命令	17
4.5 用户管理命令	18
4.6 压缩解压命令	19
4.7 网络命令	20
4.8 关机重启命令	22
五. Linux 软件安装	23
5.1 软件包管理简介	23
5.2 RPM 包管理-rpm 命令管理	24
5.3 RPM 包的查询	25
5.4 RPM 包管理-yum 在线管理	27
5.4.1 IP 地址配置和网络 yum 源	27
5.4.2 yum 在线管理-yum 命令	28
5.4.3 yum 在线管理-光盘 yum 源	29
5.5 源码包和 RPM 包区别	30
5.6 源码包管理	31
5.7 脚本安装包	32
5.8 Ubuntu 下 aptitude 包管理器	33
5.9 Ubuntu 下 dpkg 命令	34
六. 用户和用户组管理	36
6.1 用户配置文件-用户信息文件 /etc/passwd	36
6.2 影子文件 /etc/shadow	37
6.3 组信息文件/etc/group 和组密码文件/etc/gshadow	38
6.4 用户管理命令	39
6.4.1 用户添加命令 useradd	39
6.4.2 修改用户密码 passwd	41
6.4.3 修改用户信息 usermod 和修改用户密码状态 chage	42
6.4.4 删除用户 userdel 用户切换命令 su	43
6.5 用户组管理命令	44
七. 权限管理命令	45
7.1 ACL 权限	45
7.1.1 ACL 权限简介与开启	45
7.1.2 查看与设定 ACL 权限	46
7.1.3 最大有效权限与删除 ACL 权限	47

7.1.4 默认 ACL 权限和递归 ACL 权限	48
7.2 文件特殊权限	49
7.2.1 SetUID	49
7.2.2 SetGID	50
7.2.3 Sticky BIT	51
7.3 文件系统属性 chattr 权限	52
7.4 sudo 权限	53
八 . 文件系统管理	54
8.1 回顾分区和文件系统	54
8.2 文件系统常用命令	56
8.2.1 df 命令、du 命令、fsck 命令、dump2fs 命令	56
8.2.2 挂载命令	57
8.2.3 挂载光盘与 U 盘	58
8.2.4 支持 NTFS 文件系统	59
8.3 fdisk 分区	60
8.3.1 fdisk 命令分区过程	60
8.3.2 分区自动挂载与 fstab 文件修复	61
8.4 分配 swap 分区	62
九 Vim 编辑器	63
9.1 Vim 常用操作	63
9.2 Vim 使用技巧	65
十. Shell 概述	66
十一. Shell 脚本的执行方式	67
十二. Shell 脚本在不同环境编写的兼容性问题	68
十三. Bash 的基本功能	69
13.1 历史命令与命令补全	69
13.2 命令别名与常用快捷键	70
13.3 输入输出重定向	71
13.4 多命令顺序执行与管道符	72
13.5 通配符与其他特殊符号	73
十四. Bash 中的变量	74
14.1 用户自定义变量	74
14.2 环境变量	75
14.3 位置参数变量	76
14.4 预定义变量	77
十五. Bash 运算符	78
15.1 数值运算与运算符	78
15.2 变量的测试与内容替换	79
十六. 环境变量配置文件	80
16.1 环境变量配置文件简介	80
16.2 环境变量配置文件作用	81
16.3 其他配置文件和登录信息	82
十七. 正则表达式	83
十八. 字符串截取命令	85
18.1 cut 字段提取命令	85
18.2 printf 命令	86
18.3 awk 命令	87
18.4 sed 命令	89
十九. 字符处理命令	92

二十. 条件判断.....	94
二十一. 流程控制.....	97
21.1 if 语句	97
21.2 case 语句.....	101
21.3 while 循环与 until 循环.....	103
二十二. Linux 服务管理.....	105
22.1 服务简介与分类.....	105
22.2 RPM 包安装服务的管理.....	106
22.2.1 独立服务的管理.....	106
22.2.2 基于 xinetd 的服务 【不推荐】	107
22.3 源码包安装服务的管理.....	108
22.4 服务管理总结	109
二十三. Linux 系统管理.....	110
23.1 进程管理.....	110
23.1.1 进程查看.....	110
23.1.2 终止进程.....	113
23.2 工作管理.....	114
23.3 系统资源查看	115
23.4 定时任务.....	117
二十四. 日志管理	118
24.1 日志管理简介	118
24.2 rsyslogd 日志服务.....	120
24.3 日志轮替.....	122
二十五. 启动管理	123
25.1 CentOS 6x 启动管理.....	123
25.1.1 系统运行级别.....	123
25.1.2 系统启动过程.....	124
25.2 启动引导程序 grub.....	126
25.2.1 Grub 配置文件.....	126
25.2.2 Grub 加密与字符界面分辨率调整	127
25.3 系统修复模式	128
二十六. 备份与恢复.....	129
26.1 备份概述.....	129
26.2 dump 和 restore 命令	130
二十七. GCC	132
27.1 基础.....	132
参数详解.....	132
27.2 使用 ncurses 库编译程序	136
27.3 gcc 命令的常用选项.....	137
二十八. 容器	138
二十九. 内核	152
29.1 前置知识.....	152
29.2 Linux 内核体系结构简析简析	154
三十. 内核模块.....	158
30.1 前置知识.....	158
30.2 相关命令.....	158
30.3 内核模块可以做什么	159
30.4 内核模块编写实例	160
三十一. 内核的安装.....	162

1. 准备工作	162
2. 下载内核	162
3. 配置内核参数	162
4. 编译内核	162
5. 编译模块 (modules)	162
7. 安装模块 (modules)	162
8. 安装内核	163
9. 更新启动程序	163
三十二. Makefile	164
Makefile 简介	164
优点与缺点	164
主要版本	164
从一个简单的例子开始	165
资料来源	170



一. 简介

1. Unix 和 Linux 发展史

- Unix 和 Linux 关系：类似于父子关系， Unix 是父亲， Linux 是儿子
- Linux 是 Unix 的发行版本之一
- Linux 内核和 Linux 操作系统区别：
 - Linux 内核官网：www.kernel.org
 - Linux 的内核很小，从最初的几十 kb 到现在的几十 Mb
 - 所有使用了 Linux 内核写出来的操作系统都叫 Linux 操作系统
 - ◆ Linux 内核是开源免费的，但 Linux 操作系统可能是收费的（服务费）
 - ◆ CentOS 是一个社区版，所以说它是免费的（国内用的很多）

2. 开源软件简介

从服务器端来讲，Linux 的开源软件数量或质量都远远高于 Windows

比如：阿帕奇 – 最主流网站服务器软件

Nginx, MySQL, Python, Samba, Ruby

开源软件特点：

使用的自由 – 绝大多数开源软件免费

研究的自由 – 可以获得软件源代码

散布及改良的自由 – 可以自由传播、改良甚至销售

支撑互联网的开源技术：

LAMP：L – Linux

A – Web 服务器

MySQL – 数据库

PHP – 编程语言

www.netcraft.com – 可以查看一个网站后台使用的系统 OS 和其他信息（该网站使用的技术俗称“踩点”，既通过对服务端发送数据包，然后对返回的数据包进行分析，从而得出结论）

二. Linux 系统安装

1. 磁盘分区

磁盘分区时使用分区编辑器(partition editor)在磁盘上划分几个逻辑部分。碟片一旦划分成数个分区 (Partition)，不同类的目录与文件可以储存进不同的分区

- 主分区：最多 4 个【主分区号：1~4，逻辑分区不得占用主分区号，即使主分区没有 4 个】
- 扩展分区：
 - 最多只能有一个
 - 主分区+扩展分区最多有 4 个
 - 不能写入数据和格式化，只能包含逻辑分区
- 逻辑分区：
 - 可以写入数据和格式化
 - 【逻辑分区号：5~，逻辑分区号不得占用主分区号，所以从 5 开始编号←永远都是】

【主分区和逻辑分区关系：主分区可以包含逻辑分区，就像大柜子包含了小格子】

2. 格式化（高级格式化）

又称逻辑格式化，它是根据用户选定的文件系统（如 FAT16, FAT32, NTFS, EXT2, EXT3, EXT4 等），**在磁盘的特定区域写入特定数据，在分区中划分出一片用于存放文件分配表、目录表等用于文件管理的磁盘空间。**

3. 分区设备文件名

- 设备文件名：
 - /dev/hd1 (IDE 接口硬盘)
 - /dev/sda1 (SCSI 硬盘接口、SATA 硬盘接口)

4. 挂载（也就是 Windows 中的分配盘符[挂载点]）

- 必须分区
 - . 区)
 - swap 分区（交换分区，内存 2 倍，不超过 2GB,）可以理解为虚拟内存。没有盘符，因为 Linux 系统自动调用
- 推荐分区【非必须，但最好这么做】
 - /boot （启动分区，200MB）只储存启动文件，避免分区写满后可能导致系统无法启动的问题

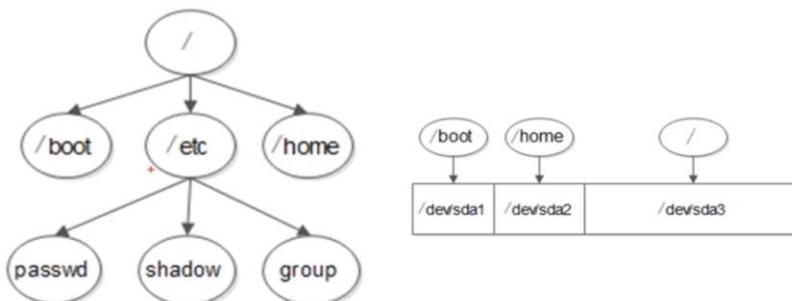


图 1

总结：

- 分区：把大硬盘分为小的逻辑分区
- **格式化：主要目的并不是清空，而是写入文件系统**
- 分区设备文件名：给每个分区定义设备文件名
- 挂载：给每个分区分配挂载点，必须是目录，而且必须是空目录

安装过程中的可选项（软件包）：

- Desktop (桌面)
- Minimal Desktop (最小化桌面)
- Minimal (最小化)
- Basic Server (基本服务器)
- Database Server (数据库服务器)
- Web Server (网页服务器)
- Virtual Host (虚拟主机)
- software development workstation (软件开发工作站)

安装日志：

- /root/install.log 储存来看安装在系统中软件包及其版本信息
- /root/install.log.syslog 储存了安装过程中留下的事件
- /root/anaconda-ks.cfg 以 kickstart 配置文件的格式记录安装过程中设置的选项信息

三．注意事项

- Linux 严格区分大小写（一般为小写，很少出现大写）
- Linux 中所有内容以文件形式保存，包括硬件
 - 硬盘文件是 /dev/sd[a-p]
 - 光盘文件是 /dev/sr0
- 命令行的更改未必会永久生效，根据上条：想命令永久生效，就去修改配置文件
- Windows 靠扩展名区分文件类型，Linux 中没有扩展名的概念，用的是文件权限给予区分（写扩展名是为了使用者能够更好地区分，便于管理）
 - 压缩包 .gz .bz2 .tar.bz2 .tgz
 - 二进制软件包：.rpm
 - 网页文件：.html .php
 - 脚本文件：.sh
 - 配置文件：.conf
- 在 Linux 中分配盘符的过程称为“挂载”
- 比如 U 盘，光盘类设备可能需要手动挂载，不能像 Windows 那样自动化。也就是说：Linux 所有的储存设备都必须在挂载之后才能使用，包括硬盘、U 盘和光盘
- Windows 下的程序不能直接在 Linux 中安装和运行
- **bin 代表 binary，二进制文件。二进制的可执行文件在 Linux 里面就是命令**

目录名	目录作用
/bin/	存放系统命令的目录，普通用户和超级用户都可以执行。不过放在/bin下的命令在单用户模式下也可以执行
/sbin/	保存和系统环境设置相关的命令，只有超级用户可以使用这些命令进行系统环境设置，但是有些命令可以允许普通用户查看
/usr/bin/	存放系统命令的目录，普通用户和超级用户都可以执行。这些命令和系统启动无关，在单用户模式下不能执行
/usr/sbin/	存放根文件系统不必要的系统管理命令，例如多数服务程序。只有超级用户可以使用。大家其实可以注意到Linux的系统，在所有“sbin”目录中保存的命令只有超级用户可以使用，“bin”目录中保存的命令所有用户都可以使用
/boot/	系统启动目录，保存系统启动相关的文件，如内核文件和启动引导程序（grub）文件等
/dev/	设备文件保存位置。我们已经说过Linux中所有内容以文件形式保存，包括硬件。那么这个目录就是用来保存所有硬件设备文件的
/etc/	配置文件保存位置。系统内所有采用默认安装方式（rpm安装）的服务的配置文件全部都保存在这个目录当中，如用户账户和密码，服务的启动脚本，常用服务的配置文件等

- 绝大多数命令都存放在前四个目录中
- **bin 大多数用户都可执行，但 sbin (super) 命令只有管理员才能执行**
- **/usr/bin/在单用户下不能执行**（单用户 – 类似于 Windows 中的“安全模式”，启动最少的服务，做修复用的）

/home/	普通用户的家目录。建立每个用户时，每个用户要有一个默认登录位置，这个位置就是这个用户的家目录，所有普通用户的家目录就是在/home下建立一个和用户名相同的目录。如用户user1的家目录就是/home/user1
/lib/	系统调用的函数库保存位置
/lost+found/	当系统意外崩溃或机器意外关机，而产生一些文件碎片放在这里。当系统启动的过程中fsck工具会检查这里，并修复已经损坏的文件系统。这个目录只在每个分区中出现，例如/lost+found就是根分区的备份恢复目录，/boot/lost+found就是/boot分区的备份恢复目录
/media/	挂载目录。系统建议是用来挂载媒体设备的，例如软盘和光盘
/mnt/	挂载目录，早期Linux中只有这一个挂载目录，并没有细分。现在这个目录系统建议挂载额外设备，如U盘，移动硬盘和其他操作系统的分区
/misc/	挂载目录。系统建议用来挂载NFS服务的共享目录。我们在刚刚已经解释了挂载，童鞋们应该知道只要是一个已经建立的空目录就可以作为挂载点。那么系统虽然准备了三个默认挂载目录/media、/mnt、/misc，但是到底在哪个目录中挂载什么设备都可以由管理员自己决定。例如超哥接触Linux的时候，默认挂载目录只有/mnt一个，所以养成了在/mnt下建立不同目录挂载不同设备的习惯。如/mnt/cdrom挂载光盘，/mnt/usb挂载U盘，这都是可以的
/opt/	第三方安装的软件保存位置。这个目录就是放置和安装其他软件的位置，我手工安装的源码包软件都可以安装到这个目录当中。不过我还是更加习惯把软件放置到/usr/local/目录当中，也就是说/usr/local/目录也可以用来安装软件

/proc/	虚拟文件系统，该目录中的数据并不保存到硬盘当中，而是保存到内存当中。主要保存系统的内核，进程，外部设备状态和网络状态灯。如/proc/cpuinfo是保存CPU信息的，/proc/devices是保存设备驱动的列表的，/proc/filesystems是保存文件系统列表的，/proc/net/是保存网络协议信息的
/sys/	虚拟文件系统。和/proc目录相似，都是保存在内存当中的，主要是保存于内核相关信息的
/root/	超级用户的家目录。普通用户家目录在“/home”下，超级用户家目录直接在“/”下
/srv/	服务数据目录。一些系统服务启动之后，可以在这个目录中保存所需要的数据
/tmp/	临时目录。系统存放临时文件的目录，该目录下所有用户都可以访问和写入。我们建议此目录中不能保存重要数据，最好每次开机都把该目录清空
/usr/	系统软件资源目录。注意usr不是user的缩写，而是“Unix Software Resource”的缩写，所以不是存放用户数据，而是存放系统软件资源的目录。系统中安装的软件大多数保存在这里，所以除了/usr/bin/和/usr/sbin/这两个目录，我在介绍几个/usr/下的二级目录
/var/	动态数据保存位置。主要保存缓存、日志以及软件运行所产生的文件

服务器注意事项

- 远程服务器不允许关机，只能重启 【关机 – 作死行为】
- 重启时应关闭服务 【避免崩溃 – 小心驶得万年船】
- 不要在服务器访问高峰运行高负载命令
- 远程配置防火墙时不要把自己踢出服务器
 - 防火墙其实就是一个过滤器（可理解为渔网网眼大小），是一个门槛。会审查IP地址、端口号、Mac地址和包装数据，并不具备完全防范病毒的功能
- 制定合理的密码规范并定期更新
- **合理分配权限，够用就好，缺少越好**
- 不要把鸡蛋放在一个篮子里，定期备份重要数据和日志

四 . 文件处理命令

4.1 文件处理命令

4.1.1 命令格式与目录处理命令 ls

命令格式：命令 [-选项] [参数]

例：ls -la /etc

说明：

1) 个别命令使用不遵循此格式

2) 当有多个选项时，可以写在一起

3) 简化选项与完整选项

-a 等于 -all

➤ -rw-r--r--

- 文件类型 (- 文件 d 目录 ! 软链接文件)

rw- r-- r--

u g o

u 所有者 g 所属组 o 其他人 ← 对应分组

r 读 w 写 x 执行

ls

命令英文原意：list

命令所在路径：/bin/ls

功能描述：显示目录文件

语法：ls 选项[-ald] [文件或目录]

-a 显示所有文件，包括隐藏文件

-l 详细信息显示

-h 人性化显示数据

-d 查看目录属性

cp

命令所在路径：/bin/cp

执行权限：所有用户

语法：cp -rp [原文件或目录] [目标目录]

-r 复制目录

-p 保留文件属性

\$ cp -r /tmp/China/Hubei /root

将目录/tmp/ China/Hubei 复制到目录/root 下

\$ cp -rp /tmp/China/Shanghai /tmp/China/Beijing

将/tmp/China 目录下的 Shanghai 和 Beijing 目录
复制到 /root 下，保持目录属性

mv

命令英文原意：move

命令所在路径：/bin/mv

执行权限：所有用户

语法：mv [原文件或目录] [目标目录]

功能描述：剪切文件、改名

cd

命令英文原意：change directory

命令所在路径：shell 内置命令

执行权限：所有用户

语法：cd [目录]

功能描述：切换目录

范例：\$ cd /tmp/China/Beijing 切换到指定目录

\$ cd ./xxxxxxxx 相对路径，在当前目录下

\$ cd .. 回到上一级目录

pwd

命令英文原意：print working directory

命令所在路径：/bin/pwd

执行权限：所有用户

语法：pwd

功能描述：显示当前目录

mkdir

命令英文原意：make directories

命令所在路径：/bin/mkdir

执行权限：所有用户

语法：mkdir -p [目录名]

功能描述：创建新目录

-p 递归创建

范例：\$ mkdir -p /tmp/China/Hubei

\$ mkdir /tmp/China/Shanghai /tmp/China/Beijing

rmdir

命令英文原意：remove empty directories

命令所在路径：/bin/rmdir

执行权限：所有用户

语法：rmdir [目录名]

功能描述：删除空目录

范例：\$ rmdir /tmp/China/Shanghai

rm

命令英文原意：remove

命令所在路径：/bin/rm

执行权限：所有用户

语法：rm -rf [文件或目录]

-r 删除目录

-f 强制执行

功能描述：删除文件

范例：

\$ rm /tmp/yum.log → 删除文件/tmp/yum.log

\$ rm -rf /* → 跑路

4.1.3 文件处理命令

touch

命令所在路径: /bin/touch

执行权限: 所有用户

语法: touch [文件名]

功能描述: 创建空文件

范例: \$ touch LinuxNote.list

cat

命令所在路径: /bin/cat

执行权限: 所有用户

语法: cat [文件名]

功能描述: 显示文件内容

-n 显示行号

范例: \$ cat /etc/issue

\$ cat -n /etc/services

tac

命令所在路径: /usr/bin/tac

执行权限: 所有用户

语法: tac [文件名]

功能描述: 显示文件内容 (反向列示)

范例: \$ tac /etc/issue

more

命令所在路径: /bin/more

执行权限: 所有用户

语法: more [文件名]

(空格) 或 f 翻页

(Enter) 换行

q 或 Q 退出

功能描述: 分页显示文件内容

范例: \$ more /etc/services

less

命令所在路径: /usr/bin/less

执行权限: 所有用户

语法: less [文件名]

功能描述: 分页显示文件内容 (可向上翻页)

上箭头↑ 换行

PageUp 翻页

范例: \$ less /etc/services

head

命令所在路径: /usr/bin/head

执行权限: 所有用户

语法: head [文件名]

功能描述: 显示文件前面几行 (默认 10 行)

-n 指定行数

范例: \$ head -n 20 /etc/services

tail

命令所在路径: /usr/bin/tail

执行权限: 所有用户

语法: tail [文件名]

功能描述: 显示文件后面几行 (默认 10 行)

-n 指定行数

-f 动态显示文件末尾内容

范例: \$ tail -n 18 /etc/services

4.1.4 链接命令

ln	软链接特征 - 类似 Windows 快捷方式
命令英文原意: link	1、 lrwxrwxrwx 软链接
命令所在路径: /bin/ln	软链接文件权限都为 rwxrwxrwx
执行权限: 所有用户	2、 文件大小-只是符号链接
语法: ln -s [原文件] [目标文件]	3、 /tmp/issue.soft -> /etc/issue 箭头指向原文件
-s 创建软链接(soft)	
功能描述: 生成链接文件	硬链接特征:
范例: \$ ln -s /etc/issue /tmp/issue.soft	1、 拷贝 cp -p + 同步更新
创建文件/etc/issue 的软链接 /tmp/issue.soft	echo "this is a test" >> /etc/motd
\$ ln /etc/issue /tmp/issue.hard	2、 可通过 i 节点识别
创建文件/etc/issue 的硬链接/tmp/issue.hard	3、 不能跨分区
	4、 不能针对目录使用

4.2.1 权限管理命令 chmod

<p>chmod</p> <p>命令英文原意: change the permissions mode of a file 命令所在路径: /bin/chmod 执行权限: 所有用户 功能描述: 改变文件或目录权限 语法: chmod [{ugoa}{+-=}{rwx}] [文件或目录] chmod [mode=421] [文件或目录] -R 递归修改 u 所有者 g 所属组 o 其他人 </p>	<p>权限的数字表示</p> $\begin{array}{l} r \rightarrow 4 \ (2^2) \\ w \rightarrow 2 \ (2^1) \\ x \rightarrow 1 \ (2^0) \end{array}$ <pre> rwx rw- r-- 7 6 4</pre> <p>范例: \$ chmod g+w testfile 赋予文件 testfile 所属组写权限</p> <p>\$ chmod -R 777 testdir 修改目录 testfile 及其目录下文件为所有用户具有全部权限</p>
--	--

文件目录权限总结

代表字符	权限	对文件的含义	对目录的含义
r	读权限	可以查看文件内容	可以列出目录中的内容
w	写权限	可以修改文件内容	可以在目录中创建、删除文件
x	执行权限	可以执行文件	可以进入目录

- 假如一个目录的权限是 drwxrwxrwx, 但是里面的一个文件 test.a 的权限是: -r--r--r--。是否可以删除 test.a?

可以, 因为目录的权限是 drwxrwxrwx。要注意区分目录权限和文件权限的区别! 文件的 w 权限是指可以修改文件内容, 而不是决定是否可以删除。目录的 w 权限才决定了内部的文件是否可以删除!

4.2.2 其他权限管理命令 – 对文件/目录而言

chown

命令英文原意: change file ownership

命令所在路径: /bin/chown

执行权限: 所有用户

语法: chown [用户] [文件或目录]

功能描述: 改变文件或目录的所有者

范例: \$ chown fox Myfiles

改变文件 Myfiles 的所有者为 fox

chgrp

命令英文原意: change file group ownership

命令所在路径: /bin/chgrp

执行权限: 所有用户

语法: chgrp [用户组] [文件或目录]

功能描述: 改变文件或目录的所属组

范例: \$ chgrp lampbrother Myfiles

改变文件 Myfiles 的所属组为 lampbrother

chmod – 更改文件或目录的权限

chown – 更改文件或目录的所有者

chgrp – 更改文件或目录的所属组

umask

命令英文原意: the user file-creation mask

命令所在路径: Shell 内置命令

执行权限: 所有用户

语法: umask [-S] -S

以 rwx 形式显示新建文件缺省权限(默认权限)

功能描述: 显示、设置文件的缺省权限

范例: \$ umask -S

4.3.1 文件搜索命令 find

<p>find</p> <p>命令所在路径: /bin/find 执行权限: 所有用户 语法: <code>find [搜索范围] [匹配条件]</code> 功能描述: 文件搜索</p> <p>【部分文件名匹配】</p> <p>比如:</p> <pre># find / -name se 在根目录下查找文件名为 se 的</pre> <pre>#find / -name *se 查找以 se 结尾的</pre> <pre>#find / -name se* 查找以 se 开头的</pre> <pre>#find / -name *se* 查找文件名中含有 se 的</pre> <p>【按修改时间】</p> <pre>\$ find /etc -cmin -5 在/etc 下查找 5 分钟内被修改过属性的 <u>文件和目录</u> -amin 访问时间 access -cmin 文件属性 change -mmin 文件内容 modify</pre> <p>-type 根据文件类型查找</p> <p>f 文件 d 目录 l 软链接文件</p> <p>-inum 根据 i 节点查找</p> <p>使用 -i 可以查看 i 节点</p> <p>Linux 使用 i 节点区分了不同的文件, 如果文件名很奇葩, 可以使用 i 结点进行操作</p>	<p>\$ find /etc -name init 在目录/etc 中查找文件 init -iname 不区分大小写</p> <p>\$ find / -size +204800 ← 文件块 在根目录下查找大于 100MB 的文件 +n 大于 -n 小于 n 等于 204800 是文件块, 1 文件块 == 0.5Kb, 1kb == 2 文件块 所以, 100Mb == (100 * 1024)Kb * 2 文件块</p> <p>\$ find /home -user fox 在根目录下查找所有者为 fox 的文件 -group 根据所属组查找</p> <p>\$ find /etc -size +163840 -a -size -204800 在/etc 下查找大于 80MB 小于 100MB 的文件 -a 两个条件同时满足 -o 两个条件满足任意一个即可</p> <p>\$ find /etc -name inittab -exec ls -l {} \; 在/etc 下查找 inittab 文件并显示其详细信息 -exec/-ok 命令 {} \; ← 对搜索结果执行操作</p>
---	--

4.3.2 其他搜索命令

locate

命令所在路径: /usr/bin/locate

执行权限: 所有用户

语法: **locate** 文件名

更新文件资料库: **updatedb**

功能描述: 在文件资料库中查找文件, 类似于 Everything

部分目录中的文件不支持, 例如/tmp

范例: \$ locate inittab

whereis

命令所在路径: /usr/bin/whereis

执行权限: 所有用户

语法: **whereis** [命令名称]

功能描述: 搜索命令所在目录及帮助文档路径

1 – 命令的帮助

5 – 配置文档的帮助

范例: \$ whereis ls

whatis

执行权限: 所有用户

语法: **whatis** [命令名称]

功能描述: 得到一个命令简短的介绍信息

范例: \$ whatis ls

which

命令所在路径: /usr/bin/which

执行权限: 所有用户

语法: **which** [命令名称]

功能描述: 搜索命令所在目录及别名信息

范例: \$ which ls

grep

命令所在路径: /bin/grep

执行权限: 所有用户

语法: **grep** -iv [指定字串] [文件]

功能描述: 在文件中搜寻字串匹配的行并输出

-i 不区分大小写

-v 排除指定字串

范例: # grep mysql /root/install.log

grep -v ^# /etc/inittab ← 排除以#开头的

4.4 帮助命令

man

命令英文原意：manual ← 菜单

命令所在路径：/usr/bin/man

执行权限：所有用户

语法：man [命令或配置文件]

功能描述：获得帮助信息

范例： \$ man ls ← 查看 ls 命令的帮助信息

 \$ man services ← 查看配置文件 services 的帮助信息

help

命令所在路径：Shell 内置命令

执行权限：所有用户

语法：help 命令

功能描述：获得 Shell 内置命令的帮助信息

范例： \$ help umask ← 查看 umask 命令的帮助信息

4.5 用户管理命令

useradd

命令所在路径: /usr/sbin/useradd

执行权限: root

语法: useradd 用户名

功能描述: 添加新用户

范例: \$ useradd yangmi

passwd

命令所在路径: /usr/bin/passwd

执行权限: 所有用户

语法: passwd 用户名

功能描述: 设置用户密码

范例: \$ passwd yangmi

who

命令所在路径: /usr/bin/who

执行权限: 所有用户

语法: who

功能描述: 查看目前所有在线登录用户简要信息

范例: \$ who

```
[root@localhost ~]# who
root      tty1        2020-07-06 15:48
```

tty – 登陆的终端（本机）

pts – 远程登录

w

命令所在路径: /usr/bin/w

执行权限: 所有用户

语法: w

功能描述: 查看目前所有在线登录用户详细信息

范例: \$ w

```
[root@localhost ~]# w
16:28:04 up 41 min, 1 user, load average: 0.03, 0.04, 0.05
USER   TTY      FROM          LOGIN@    IDLE      JCPU     PCPU WHAT
root   tty1                               15:48   4.00s  0.66s  0.05s w
```

up – 系统连续运行的时间

TTY – 登录的终端

load average – 系统负载

IDLE – 用户距上一操作空闲多久了

JCPU – 用户占用 CPU 的总时间 (CPU 时间)

PCPU – 用户当前操作占用 CPU 的时间

WHAT – 当前执行的操作

-bash 等待执行

4.6 压缩解压命令

gzip

命令英文原意：GNU zip

命令所在路径：/bin/gzip

执行权限：所有用户

语法： gzip [文件]

功能描述：压缩文件，压缩完会删除源文件

压缩后文件格式：.gz

gunzip

命令英文原意：GNU unzip

命令所在路径：/bin/gunzip

执行权限：所有用户

语法： gunzip [压缩文件]

功能描述：解压缩.gz 的压缩文件

范例： \$ gunzip Russia.gz

tar

命令所在路径：/bin/tar

执行权限：所有用户

语法： tar 选项[-zcf] [压缩后文件名] [目录]

-c 打包

-v 显示详细信息

-f 指定文件名(file)

-z 打包/解包的同时压缩/解压

功能描述：打包目录

压缩后文件格式：.tar.gz

范例： \$ tar -zcf Japan.tar.gz Japan

将目录 Japan 打包并压缩为.tar.gz 文件

tar

tar 命令解压缩语法：

-x 解包

-v 显示详细信息

-f 指定解压文件

-z 打包/解包的同时压缩/解压

范例： \$ tar -zvxf Japan.tar.gz

zip

命令所在路径：/usr/bin/zip

执行权限：所有用户

语法： zip [-r] [压缩后文件名] [文件或目录]

-r 压缩目录

功能描述：压缩文件或目录

压缩后文件格式：.zip

范例： \$ zip MyFile.zip MyFile 压缩文件

\$ zip -r MyFiles.zip MyFiles 压缩目录

unzip

命令所在路径：/usr/bin/unzip

执行权限：所有用户

语法： unzip [压缩文件]

功能描述：解压.zip 的压缩文件

范例： \$ unzip test.zip

bzip2 【压缩率最高】

命令所在路径：/usr/bin/bzip2

执行权限：所有用户

语法： bzip2 选项[-k] [文件]

-k 产生压缩文件后保留原文件

-j 类似于 tar 中的-z 打包/解包的同时压缩/解压

功能描述：压缩文件

压缩后文件格式：.bz2

范例： \$ bzip2 -k boduo

\$ tar -cjf Japan.tar.bz2 Japan

bunzip2

命令所在路径：/usr/bin/bunzip2

执行权限：所有用户

语法： bunzip2 选项[-k] [压缩文件]

-k 解压缩后保留原文件

功能描述：解压缩

范例： \$ bunzip2 -k Myfiles.bz2

\$ tar -xjf Japan.tar.bz2

4.7 网络命令

write

指令所在路径: /usr/bin/write

执行权限: 所有用户

语法: write [用户名]

功能描述: 给用户发信息, 以 Ctrl+D 保存结束

范例: # write silverfox

mail

命令所在路径: /bin/mail

执行权限: 所有用户

语法: mail [用户名]

功能描述: 查看发送电子邮件

范例: # mail root

wall

命令英文原意: write all

指令所在路径: /usr/bin/wall

执行权限: 所有用户

语法: wall [message]

功能描述: 发广播信息

范例: # wall Yiff yiff yiff!

ping

命令所在路径: /bin/ping

执行权限: 所有用户

语法: ping 选项 IP 地址

-c 指定发送次数

功能描述: 测试网络连通性

范例: # ping 192.168.1.156

ifconfig

命令英文原意: interface configure

命令所在路径: /sbin/ifconfig

执行权限: root

语法: ifconfig 网卡名称 IP 地址

功能描述: 查看和设置网卡信息

范例: # ifconfig eth0 192.168.8.250

traceroute

命令所在路径: /bin/traceroute

英文原意: 路由追踪

执行权限: 所有用户

语法: traceroute <主机名>

功能描述: 显示数据包到主机间的路径

范例: # traceroute www.foxthere.com

*如果 traceroute 出来的全是星号***, 把虚拟机的状态从 NAT 模式改为桥接模式即可

```
[root@localhost ~]# traceroute foxthere.com
traceroute to foxthere.com (172.247.132.226), 30 hops max, 60 byte packets
 1 gateway (192.168.1.1)  0.388 ms  0.761 ms  0.635 ms
 2 * * *
 3 * * *
 4 * * *
 5 * * *
 6 * * *
 7 * kbn-bb3-link.telia.net (62.115.139.169)  286.772 ms *
 8 kbn-bz2-link.telia.net (62.115.123.195)  56.982 ms  nyk-bb3-link.telia.net (213.155.134.50)  220.9
86 ms  220.973 ms
 9 kbn-bn3-link.telia.net (62.115.123.178)  206.063 ms * ash-bb2-link.telia.net (62.115.136.201)  1
48.640 ms
10 las-b24-link.telia.net (62.115.121.220)  202.906 ms  hbg-bb3-link.telia.net (213.155.130.180)  21
0.358 ms  las-b24-link.telia.net (62.115.121.220)  202.623 ms
11 cnservers-ic-344699-las-b24.c.telia.net (62.115.174.231)  216.294 ms  211.272 ms  218.179 ms
12 * * *
13 * * *
14 3tblogs.gdi.1101.core1.ceranetworks.com (23.224.58.2)  207.027 ms  207.508 ms  207.623 ms
15 * cservers-ic-344699-las-b24.c.telia.net (62.115.174.231)  209.150 ms  23.224.58.242 (23.224.58.
242)  208.997 ms
16 * * *
17 * * *
18 * * *
19 23.224.58.242 (23.224.58.242)  209.182 ms *  205.706 ms
```

last

命令所在路径：/usr/bin/last

执行权限：所有用户

语法：last

功能描述：列出**目前与过去**登入系统的用户信息

范例：# last

```
[root@localhost ~]# last
root      tty1                               Mon Jul  6 23:31  still logged in
reboot    system boot  3.10.0-1127.el7. Mon Jul  6 23:30 - 23:32  (00:01)
root      tty1                               Mon Jul  6 15:48 - 21:37  (05:48)
```

lastlog

命令所在路径：/usr/bin/lastlog

执行权限：所有用户

语法：lastlog

功能描述：检查**(某特定)用户上次**登录的时间

范例：# lastlog

lastlog -u 502 ← Linux 中使用 uid 区分用户，就像 b 站

```
[root@localhost ~]# lastlog
Username        Port      From          Latest
root            tty1
bin
daemon
adm
lp
root            Mon Jul  6 23:34:40 +0300 2020
**Never logged in**
```

netstat

命令所在路径：/bin/netstat

执行权限：所有用户

语法：netstat [选项]

功能描述：显示网络相关信息

选项： -t : TCP 协议

-u : UDP 协议

-l : 监听

-r : 路由

-n : 显示 IP 地址和端口号

范例

netstat -tlun 查看本机监听的端口

netstat -an 查看本机所有的网络连接

netstat -rn 查看本机路由表

setup

命令所在路径：/usr/bin/setup

执行权限：root

语法：setup

功能描述：配置网络（只有红帽子能用）

范例：# setup

#注意：CentOS7 中 setup 中没有网络配置功能了，需要用 nmtui

mount

命令位置：/bin/mount ← 本质是一个软连接

执行权限：所有用户

功能描述：挂载设备

命令语法：mount [-t 文件系统] 设备文件名 挂载点

范例：# mount -t iso9660 /dev/sr0 /mnt/cdrom

umount

功能描述：取消挂载点

命令语法：mount 设备文件名/挂载点

范例：# mount /dev/sr0

mount /mnt/cdrom

任意一个即可，否则可能造成重复取消挂载

4.8 关机重启命令

shutdown 命令	其他关机命令
<p>*推荐使用 shutdown 命令，因为该命令会自动终止部分进程，安全性较高</p> <pre>[root@localhost ~]# shutdown [选项] 时间</pre> <p>选项：</p> <ul style="list-style-type: none">-c → 取消前一个关机命令-h → 关机-r → 重启	<pre># halt</pre> <pre># poweroff (安全性最低，相当于直接拔电源)</pre> <pre># init 0</pre>
其他重启命令	其他重启命令
	<pre># reboot</pre> <pre># init 6</pre>

系统运行级别

- 0 关机
- 1 单用户 ← 相当于 Win 中的“安全模式”。只有 root 用户，只启动最基本服务，以便于系统修复
- 2 不完全多用户，不含 NFS 服务 ← 使用 NFS 服务可能会使安全性降低
- 3 完全多用户 (纯字符界面)
- 4 未分配
- 5 图形界面
- 6 重启

修改系统默认运行级别

```
[root@localhost ~]# cat /etc/inittab
```

id:**3**:initdefault: ← 3 处的数字代表默认运行级别，即系统启动时默认的启动方式。

不要删除末尾和中间的冒号！

不要将默认运行级别修改为 0 或者 6!!

查询系统运行级别

```
[root@localhost ~]# runlevel
```

退出登录命令

```
[root@localhost ~]# logout
```

说明：在使用完服务器主机，或者远离电脑时一定要退出当前账号！防止他人恶意登录造成企业不必要的损失和
防止自己背不必要黑锅!!!

五. Linux 软件安装

5.1 软件包管理简介

1. 软件包分类

- 源码包
 - a) 脚本安装包
- 二进制包
 - a) RPM 包
 - b) 系统默认包

2. 源码包与二进制包的对比

● 源码包

◆ 优点

- 开源，如果有足够的能力，可以修改源代码
- 可以自由选择所需的功能
- 软件是编译安装，所以更加适合自己的系统，更加稳定也效率更高
- 卸载方便 → 没有依赖性，直接删除整个文件夹就好，没有残留文件

◆ 缺点

- 安装过程步骤较多，尤其安装较大的软件集合时（如 LAMP 环境搭建），容易出现拼写错误
- 编译过程时间较长，安装比二进制安装时间长
- 因为是编译安装，安装过程中一旦报错新手很难解决

● RPM 包/二进制包

◆ 优点

- 包管理系统简单，只通过几个命令就可以实现包的安装、升级、查询和卸载
- 安装速度比源码包安装快的多

◆ 缺点

- 经过编译，不再可以看到源代码
- 功能选择不如源码包灵活
- 依赖性

5.2 RPM 包管理-rpm 命令管理

- RPM 包命名原则

包全名: httpd-2.2.15-15.el6.centos.1.i686.rpm

软件版本	适用的Linux版本
软件包名	发布次数
适合的硬件平台	
-86 32位CPU	
-64 64位CPU	

所有的RPM包都应该是以.rpm结尾

虽然Linux中拓展名可有可无，但是RPM包是给管理员看的，必须为此结尾

- RPM 包依赖性

- 树形依赖 a→b→c
- 环形依赖 a→b→c→a
- 模块依赖: 模块依赖查询网站 www.rpmfind.net

- 包全名与包名

- 包全名, 只有安装和升级时需要使用 (一般来说是未安装过得包)。操作的包是没有安装的软件包时, 使用包全名。而且要注意路径
- 包名: 操作已经安装的软件包时, 使用包名。系统判断是否安装此软件包时搜索 /var/lib/rpm/ 中的数据
- 库

<p>RPM 安装</p> <p>rpm -ivh 包全名</p> <p>选项:</p> <ul style="list-style-type: none">-i (install) 安装-v (verbose) 显示详细信息-h (hash) 显示进度--nodeps 不检测依赖性 (一般不会用)	<p>RPM 包升级</p> <p>rpm -Uvh 包全名</p> <p>选项:</p> <ul style="list-style-type: none">-U (upgrade) 升级
<p>RPM 卸载</p> <p>rpm -e 包名 (系统数据库中搜索已经安装的包)</p> <ul style="list-style-type: none">-e (erase) 卸载	

5.3 RPM 包的查询

查询是否安装

rpm -q 包名

-q 查询 (query)

查询所有安装的 RPM 包

rpm -qa

-a 所有 (all)

查询软件包详细信息

rpm -qi 包名

-i 查询软件信息 (information)

-p 查询未安装包信息 (package)

查询包中文件安装位置

rpm -ql 包名

-l 列表 (list)

-p 查询未安装包的信息 (package)

查询系统文件属于哪个 RPM 包

rpm -qf 系统文件名

-f 查询系统文件属于哪个软件包(file)

查询软件包的依赖性

rpm -qR 包名

-R 查询软件包依赖性 (requires)

-p 查询未安装包信息 (package)

RPM 包校验

rpm -V 已安装的包名

-V 校验指定 RPM 包中的文件 (verify)

校验内容中的 8 个信息的具体内容如下：

- S 文件大小是否改变
- M 文件类型或文件的权限 (rwx) 是否改变
- 5 文件的 MD5 校验和是否改变 (可以看成文件内容是否改变)
- D 设备主从设备号是否改变
- L 文件路径是否改变
- U 文件的所有者是否改变
- G 文件的所有组是否改变
- T 文件的修改时间是否改变

文件类型

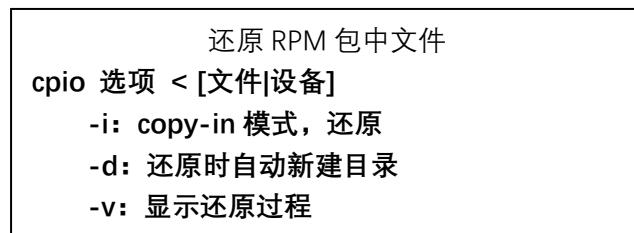
```
[root@localhost /]# rpm -V httpd  
S.5....T. C /etc/httpd/conf/httpd.conf
```

- c 配置文件 (config file)
- d 普通文档 (documentation)
- g “鬼”文件 (ghost file)，很少见，就是该文件不应该被这个 RPM 包包含
- l 授权文件 (license file)
- r 描述文件 (read me)

RPM 包中文件提取

rpm2cpio 包全名 | cpio -idv [文件绝对路径]

- rpm2cpio - 将 rpm 包转换为 cpio 格式的命令
- cpio - 是一个标准工具，它用于创建软件档案文件和从档案文件中提取文件



比如：

```
[root@localhost ~]# rpm -qf /bin/ls          #查询 ls 命令属于哪个软件包
```

```
[root@localhost ~]# mv /bin/ls /tmp/         #造成 ls 命令误删除假象
```

```
[root@localhost ~]# rpm2cpio /mnt/cdrom/Packages/coreutils8.4-19.el6.i686.rpm | cpio -idv ./bin/ls  
#提取 RPM 包中 ls 命令到当前目录的/bin/ls 下
```

```
[root@localhost ~]# cp /root/bin/ls /bin/      #把 ls 命令复制给/bin/目录, 修复文件丢失
```

5.4.1 IP 地址配置和网络 yum 源

1. IP 地址配置

```
[root@localhost ~]# setup #使用 setup 工具  
#注意：CentOS7 中 setup 中没有网络配置功能了，需要用 nmtui
```

```
[root@localhost ~]# vi /etc/sysconfig/network-scripts/ifcfg-eth0 把 ONBOOT="no" 改为 ONBOOT="yes"  
#启动网卡
```

```
[root@localhost ~]# service network restart #重启网络服务
```

2. 网络 yum 源

```
[root@localhost yum.repos.d]# vi /etc/yum.repos.d/CentOS-Base.repo
```

- [base] 容器名称，一定要放在[]中
- name 容器说明，可以自己随便写
- mirrorlist 镜像站点，这个可以注释掉
- baseurl 我们的 yum 源服务器的地址。默认是 CentOS 官方的 yum 源服务器，是可以使用的，如果你觉得慢可以改成你喜欢的 yum 源地址
- enabled 此容器是否生效，如果不写或写成 enable=1 都是生效，写成 enable=0 就是不生效
- gpgcheck 如果是 1 是指 RPM 的数字证书生效，如果是 0 则不生效
- gpgkey 数字证书的公钥文件保存位置。不用修改

5.4.2 yum 在线管理-yum 命令

查询

yum list	查询所有可用软件包列表
yum search 关键字	搜索服务器上所有和关键字有关的包

安装

yum -y install 包名	-y 自动回答 yes
install	安装

升级

yum -y update 包名	-y 自动回答 yes
update	升级

卸载

yum -y remove 包名	-y 自动回答 yes
remove	卸载

yum 软件组管理命令

yum grouplist	列出所有可用的软件组列表
yum groupinstall 软件组名	安装指定软件组，组名可以由 grouplist 查询出来
yum groupremove 软件组名	卸载指定软件组

5.4.3 yum 在线管理-光盘 yum 源

光盘 yum 源搭建步骤

1) 挂载光盘

```
[root@localhost ~]# mount /dev/cdrom /mnt/cdrom/
```

2) 让网络 yum 源文件失效

```
[root@localhost ~]# cd /etc/yum.repos.d/
```

```
[root@localhost yum.repos.d]# mv CentOS-Base.repo \ CentOS-Base.repo.bak 【随便改个名，让系统识别不出来】
```

```
[root@localhost yum.repos.d]# mv CentOS-Debuginfo.repo \ CentOS-Debuginfo.repo.bak
```

```
[root@localhost yum.repos.d]# mv CentOS-Vault.repo \ CentOS-Vault.repo.bak
```

3) 修改光盘 yum 源文件

```
[root@localhost yum.repos.d]# vim CentOS-Media.repo
```

```
[c6-media]
```

```
name=CentOS-$releasever - Media
```

```
baseurl=file:///mnt/cdrom #地址为自己的光盘挂载地址
```

```
# file:///media/cdrom/
```

```
# file:///media/cdrecorder/
```

```
#注释这两个不存在的地址
```

```
gpgcheck=1
```

```
enabled=1
```

```
#把 enabled=0 改为 enabled=1，让这个 yum 源配置文件生效
```

```
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-6
```

5.5 源码包和 RPM 包区别

1. 区别

- 安装之前的区别：概念上的区别
- 安装之后的区别：安装位置不同

2. RPM 包安装位置

▶ 是安装在默认位置中

RPM包默认安装路径	
/etc/	配置文件安装目录
/usr/bin/	可执行的命令安装目录
/usr/lib/	程序所使用的函数库保存位置
/usr/share/doc/	基本的软件使用手册保存位置
/usr/share/man/	帮助文件保存位置

3. 源码包安装位置

安装在指定位置当中，一般是 /usr/local/软件名

4. 安装位置不同带来的影响

- RPM 包安装的服务可以使用系统服务管理命令（service）来管理，例如 RPM 包安装的 apache 的启动方法是：
 - /etc/rc.d/init.d/httpd start
 - service httpd start
- 而源码包安装的服务则不能被服务管理命令管理，因为没有安装到默认路径中。所以只能用绝对路径进行服务的管理，如：
 - /usr/local/apache2/bin/apachectl start

5.6 源码包管理

1、安装准备

- 安装 C 语言编译器 (gcc)
- 下载源码包 <http://mirror.bit.edu.cn/apache/httpd>

2、安装注意事项

- 源代码保存位置：/usr/local/src/
- 软件安装位置：/usr/local/
- 如何确定安装过程报错：
 - 安装过程停止
 - 并出现 error、warning 或 no 的提示

3、源码包安装过程

- 下载源码包
- 解压缩下载的源码包
- 进入解压缩目录
- 安装说明在 **INSTALL** 文件中
- **./configure** 软件配置与检查
 - 定义需要的功能选项。
 - 检测系统环境是否符合安装要求。
 - 把定义好的功能选项和检测系统环境的信息都写入 Makefile 文件，用于后续的编辑。
- make 编译
 - make clean 如果报错，清空编译产生的临时文件
- make install 编译安装

4、源码包的卸载

- 不需要卸载命令，直接删除安装目录即可。不会遗留任何垃圾文件。

可以使用与运算符从而进行命令的组合执行：make && make install

5.7 脚本安装包

1、脚本安装包

- 脚本安装包并不是独立的软件包类型，常见安装的是源码包
- 是人为把安装过程写成了自动安装的脚本，只要执行脚本，定义简单的参数，就可以完成安装
- 非常类似于 Windows 下软件的安装方式。

2、Webmin 的作用

- Webmin 是一个基于 Web 的 Linux 系统管理界面。您就可以通过图形化的方式设置用户帐号、Apache、DNS、文件共享等服务

3、Webmin 安装过程

- 下载软件 <http://sourceforge.net/projects/webadmin/files/webmin/>
- 解压缩，并进入加压缩目录
- 执行安装脚本，按提示完成安装

5.8 Ubuntu 下 aptitude 包管理器

aptitude 与 apt-get 一样, 是 debian 及其衍生系统中功能极其强大的包管理工具。与 apt-get 不同的是, aptitude 在处理依赖问题上更佳一些。举例来说, aptitude 在删除一个包时, 会同时删除本身所依赖的包。这样, 系统中不会残留无用的包, 整个系统更为干净。

- 安装 aptitude: sudo apt-get install aptitude

命令	作用
aptitude update	更新可用的包列表
aptitude safe-upgrade	执行一次安全的升级
aptitude full-upgrade	将系统升级到新的发行版
aptitude install pkgname	安装包
aptitude remove pkgname	删除包
aptitude purge pkgname	删除包及其配置文件
aptitude search string	搜索包
aptitude show pkgname	显示包的详细信息
aptitude clean	删除下载的包文件
aptitude autoclean	仅删除过期的包文件

- 直接输入 aptitude 可直接进入图形化界面

- aptitude 是以树状结构组织的, 相关节点以细线相连. 可以使用方向键, [PageUp] 和 [PageDown] 进行定位.[Return] 或[Enter] 用于展开或收起某个节点. aptitude 的开视图分为四列: 软件包状态和请求动作, 软件包名称, 当前安装版本(或 de><de>nonede>>de>), 最新版本.

5.9 Ubuntu 下 dpkg 命令

命令	作用
dpkg -i package.deb	安装包
dpkg -r package	删除包
dpkg -P package	删除包（包括配置文件）
dpkg -L package	列出与该包关联的文件
dpkg -l package	显示该包的版本
dpkg --unpack package.deb	解开 deb 包的内容
dpkg -S keyword	搜索所属的包内容
dpkg -l	列出当前已安装的包
dpkg -c package.deb	列出 deb 包的内容
dpkg --configure package	配置包

命令	作用
apt-cache search package	搜索包
apt-cache show package	获取包的相关信息，如说明、大小、版本等
sudo apt-get install package	安装包
sudo apt-get install package --reinstall	重新安装包
sudo apt-get -f install	强制安装
sudo apt-get remove package	删除包
sudo apt-get remove package --purge	删除包，包括删除配置文件等
sudo apt-get autoremove	自动删除不需要的包
sudo apt-get update	更新源
sudo apt-get upgrade	更新已安装的包
sudo apt-get dist-upgrade	升级系统
sudo apt-get dselect-upgrade	使用 <code>dselect</code> 升级
apt-cache depends package	了解使用依赖
apt-cache rdepends package	了解某个具体的依赖
sudo apt-get build-deppackage	安装相关的编译环境
apt-get source package	下载该包的源代码
sudo apt-get clean && sudo apt-get autoclean	清理下载文件的存档
sudo apt-get check	检查是否有损坏的依赖

六. 用户和用户组管理

6.1 用户配置文件-用户信息文件 /etc/passwd

1、用户管理简介

- 越是对服务器安全性要求高的服务器，越需要建立合理的用户权限等级制度和服务器操作规范
- 在 Linux 中主要是通过用户配置文件来查看和修改用户信息

2、/etc/passwd

```
[root@localhost ~]# vim /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
```

- 第 1 字段：用户名
- 第 2 字段：密码标志（没有 x 代表没有密码）
- 第 3 字段：UID（用户 ID）
 - 0：超级用户
 - 1-499：系统用户（伪用户，但不能删除，否则会造成系统异常）
 - 500-65535：普通用户
- 第 4 字段：GID（用户初始组 ID）
- 第 5 字段：用户说明
- 第 6 字段：家目录
 - 普通用户：/home/用户名/
 - 超级用户：/root/
- 第 7 字段：登录之后的 Shell（Shell 就是系统的一个解释器）

3、初始组和附加组

- 初始组：就是指用户一登录就立刻拥有这个用户组的相关权限，每个用户的初始组只能有一个，一般就是和这个用户的用户名相同的组名作为这个用户的初始组
- 附加组：指用户可以加入多个其他的用户组，并拥有这些组的权限，附加组可以有多个

4、Shell 是什么

- Shell 就是 Linux 的命令解释器
- 在/etc/passwd 当中，除了标准 Shell 是 /bin/bash 之外，还可以写如/sbin/nologin, /usr/bin/passwd 等，但是别乱改避免登不上

6.2 影子文件 /etc/shadow

```
[root@localhost ~]# vim /etc/shadow
root:$6$ls/JrDZp$.8CIfJjTmzGlckGgeMzd6Ip6LR50q5u60i/w49Ksp244d6rvVXB6Z0vEUQgiJeoRZx53U0GwEYTxEHzMFTMNb0:18449:0:99999:
7:::
bin:*:18353:0:99999:7:::
daemon:*:18353:0:99999:7:::
adm:*:18353:0:99999:7:::
lp:*:18353:0:99999:7:::
sync:*:18353:0:99999:7:::
shutdown:*:18353:0:99999:7:::
```

1、影子文件/etc/shadow

- 第 1 字段：用户名
- 第 2 字段：加密密码
 - 加密算法升级为 SHA512 散列加密算法
 - 如果密码位是“!”或“*”代表没有密码，不能登录
- 第 3 字段：密码最后一次修改日期
 - 使用 1970 年 1 月 1 日作为标准时间，每过一天时间戳 加 1
- 第 4 字段：两次密码的修改间隔时间（和第 3 字段相比）
- 第 5 字段：密码有效期（和第 3 字段相比）
- 第 6 字段：密码修改到期前的警告天数（和第 5 字段相比）
- 第 7 字段：密码过期后的宽限天数（和第 5 字段相比）
 - 0：代表密码过期后立即失效
 - -1：则代表密码永远不会失效
- 第 8 字段：账号失效时间
 - 要用时间戳表示
- 第 9 字段：保留字段

2、时间戳换算

- 把时间戳换算为日期 → date -d "1970-01-01 16066 days"
- 把日期换算为时间戳 → echo \$(((\$date --date="2014/01/06" +%s)/86400+1))

6.3 组信息文件/etc/group 和组密码文件/etc/gshadow

/etc/group/

```
audio:::  
nobody:::  
users:::  
utmp:!:  
utempter:!:!
```

- 第一字段：组名
- 第二字段：组密码标志
- 第三字段：GID
- 第四字段：组中的附加用户

/etc/gshadow/

```
audio:::  
nobody:::  
users:::  
utmp:!:  
utempter:!:!  
input:!:  
systemd-journal:!:!
```

- 第一字段：组名
- 第二字段：组密码
- 第三字段：组管理员用户名
- 第四字段：组中的附加用户

用户信息：

- 用户的家目录：
 - 普通用户：/home/用户名/，所有者和所属组是此用户，权限是 700
`drwx-----. 4 fox fox 4096 Jul 6 23:34 fox`
 - 超级用户：/root/，所有者和所属组都是 root 用户，权限是 550
`dr-xr-x---. 4 root root 4096 Jul 26 16:21 root`
- 用户的邮箱
 - /var/spool/mail/用户名
- 用户模板目录
 - /etc/skel/
 - 在用户模板目录中，可以存放一些说明文件，这样在添加新用户的时候后，新用户的家目录中就会自动的出现这些文件（就像是 GitHub 中的 Readme 文件）

6.4.1 用户添加命令 useradd

useradd 命令格式：

- **useradd [选项] 用户名**
 - -u UID： 手动指定用户的UID号
 - -d 家目录： 手工指定用户的家目录
 - -c 用户说明： 手工指定用户的说明
 - -g 组名： 手工指定用户的初始组（不建议这么做）
 - **-G 组名：** 指定用户的附加组
 - -s shell： 手工指定用户登录shell。默认是/bin/bash

添加默认用户：

```
[root@localhost ~]# useradd lamp
→ [root@localhost ~]# grep "lamp" /etc/passwd
→ [root@localhost ~]# grep "lamp" /etc/shadow
→ [root@localhost ~]# grep "lamp" /etc/group
→ [root@localhost ~]# grep "lamp" /etc/gshadow
→ [root@localhost ~]# ll -d /home/lamp/
→ [root@localhost ~]# ll /var/spool/mail/lamp
```

指定选型添加用户：

```
groupadd lamp1
useradd -u 550 -g lamp1 -G root -d /home/lamp1 -c "test user" -s /bin/bash lamp1
```

用户默认值文件：

/etc/default/useradd

```
# useradd defaults file
GROUP=99    用户默认组
HOME=/home  用户家目录
INACTIVE=-1 密码过期宽限天数（对应/etc/passwd文件）
EXPIRE=      密码失效时间（对应第8字段）
SHELL=/bin/bash  默认shell
SKEL=/etc/skel 模版目录
CREATE_MAIL_SPOOL=yes 是否建立邮箱
```

/etc/login.defs

```
# 
PASS_MAX_DAYS   99999 密码有效期 (s)
PASS_MIN_DAYS   0 密码修改间隔 (s)
PASS_MIN_LEN     5 密码最小位数 (PAM)
PASS_WARN_AGE     7 密码到期警告 (s)

#
# Min/max values for automatic uid selection in useradd
#
UID_MIN           1000
UID_MAX          60000
# System accounts
SYS_UID_MIN        201
SYS_UID_MAX        999      最小和最大UID范围
```

```
CREATE_HOME      yes

# The permission mask is initialized to this value. If not specified,
# the permission mask will be initialized to 022.
UMASK            077

# This enables userdel to remove user groups if no members exist.
#
USERGROUPS_ENAB yes

# Use SHA512 to encrypt password.
ENCRYPT_METHOD  SHA512  加密模式
```

6.4.2 修改用户密码 passwd

passwd 命令格式

● passwd [选项] 用户名

- -S 查询用户密码的密码状态。 仅 root 用户可用
- -l 暂时锁定用户。 仅 root 用户可用
- -u 解锁用户。 仅 root 用户可用
- --stdin 可以通过管道输出的数据作为用户的密码

查看密码状态

```
[root@MiWiFi-R1D-srv ~]# passwd -S fox
fox PS 2020-07-26 0 99999 7 -1 (Password set, SHA512 crypt.)
```

- 用户名密码设定时间 (2020-07-26)
- 密码修改间隔时间 (0)
- 密码有效期 (99999)
- 警告时间 (7)
- 密码不失效 (-1)

锁定和解锁用户

- passwd -l fox (l - lock)
- passwd -u fox (u- unlock)

使用字符串作为用户密码

- echo "123" | passwd --stdin lamp

6.4.3 修改用户信息 usermod 和修改用户密码状态 chage

usermod [选项] 用户名

- -u UID: 修改用户的UID号
- -c “用户说明”: 修改用户的说明信息
- -G 组名: 修改用户的附加组
- -L: 临时锁定用户 (Lock)
- -U: 解锁用户锁定 (Unlock)

比如:

- ◆ [root@localhost ~]# usermod -c "test user" lamp #修改用户的说明
- ◆ [root@localhost ~]# usermod -G root lamp #把lamp用户加入root组
- ◆ [root@localhost ~]# usermod -L lamp #锁定用户
- ◆ [root@localhost ~]# usermod -U lamp #解锁用户

修改密码状态 chage

chage [选项] 用户名

- ◆ -l: 列出用户的详细密码状态
 - ◆ -d 日期: 修改密码最后一次更改日期 (shadow3 字段)
 - ◆ -m 天数: 两次密码修改间隔 (4 字段)
 - ◆ -M 天数: 密码有效期 (5 字段)
 - ◆ -W 天数: 密码过期前警告天数 (6 字段)
 - ◆ -I 天数: 密码过后宽限天数 (7 字段)
 - ◆ -E 日期: 账号失效时间 (8 字段)
- ◆ chage -d 0 lamp

#这个命令其实是把密码修改日期归0了 (shadow 第3字段) #这样用户一登陆就要修改密码

6.4.4 删 除用 户 userdel 用 户切 换命 令 su

删 除用 户

userdel [-r] 用 户名

-r 删 除用 户的 同时删 除用 户家 目录

手 工删 除用 户：

```
v [root@localhost ~]# vi /etc/passwd  
v [root@localhost ~]# vi /etc/shadow  
v [root@localhost ~]# vi /etc/group  
v [root@localhost ~]# vi /etc/gshadow  
v [root@localhost ~]# rm -rf /var/spool/mail/lamp  
v [root@localhost ~]# rm -rf /home/lamp/
```

查 看用 户 ID

id 用 户名

切 换用 户身 份 su

su [选项] 用 户名

选 项：

- : 使用 “-” 代 表连 带用 户的 环境变 量一 起切 换

-c 命 令：仅 执 行一 次命 令， 而 不切 换用 户身 份

```
v [lamp@localhost ~]$ su - root #切 换成 root  
v [lamp@localhost ~]$ su - root -c "useradd user3" #不切 换成 root, 但 是执 行 useradd 命 令添 加 user1 用 户
```

6.5 用户组管理命令

添加用户组

groupadd [选项] 组名

-g GID 指定组 ID, 如果不指定按默认值往后摆

修改用户组

groupmod [选项] 组名

-g GID 指定组 ID

-n 新组名

比如:

groupmod -n newG tg → 将 tg 换成 newG

删除用户组

groupdel 组名

如果这个组有初始用户, 则这个组不能删除。必须移除初始用户。【如果想删除组, 组中不能有附加用户】

附加用户不影响组的删除

把用户添加入组或从组中删除

gpasswd [选项] 组名 ← 操作的是什么就把什么放在最后, 这里操作的是组所以把组放在最后。但 passwd 操作的是用户, 所以把用户放在最后

-a 用户名: 把用户加入组 (add)

-d 用户名: 把用户从组删除 (delete)

不建议更改用户的初始组

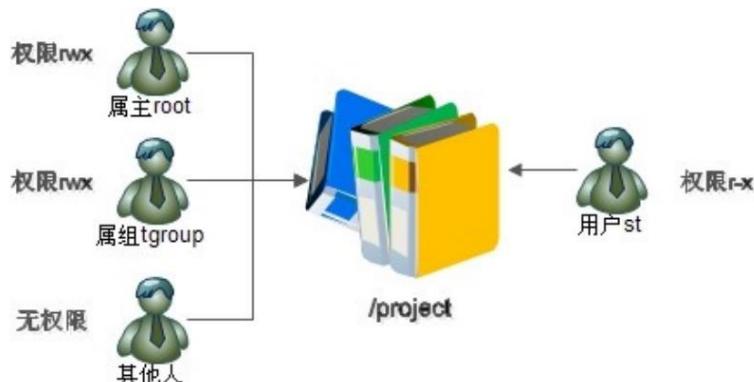
groupXXX 操作的是组

七 . 权限管理命令

7.1 ACL 权限

7.1.1 ACL 权限简介与开启

1. ACL 权限简介



查看分区 ACL 权限是否开启

```
dumpe2fs -h /dev/sda3
```

#dumpe2fs 命令是查询指定分区详细文件系统信息的命令
选项：
-h 仅显示超级块中信息，而不显示磁盘块组的详细信息

临时开启分区 ACL 权限

```
mount -o remount,acl /
```

#重新挂载根分区，并挂载加入 acl 权限

永久开启分区 ACL 权限

```
vi /etc/fstab
```

```
UUID=c2ca6f57-b15c-43ea-bca0-f239083d8bd2 / ext4 defaults,acl 1 1      #加入 acl
```

```
mount -o remount /
```

#重新挂载文件系统或重启动系统，使修改生效

7.1.2 查看与设定 ACL 权限

查看 ACL 命令

```
getfacl 文件名 #查看 ACL 权限
```

设定 ACL 权限

```
setfacl [选项] 文件名
```

- m 设定 ACL 权限
- x 删除指定的 ACL 权限
- b 删除所有的 ACL 权限
- d 设定默认 ACL 权限 (default)
- k 删除默认 ACL 权限
- R 递归设定 ACL 权限 (Recursion)

给用户设定 ACL 权限

```
[root@localhost ~]# useradd zhangsan  
[root@localhost ~]# useradd lisi  
[root@localhost ~]# useradd st  
[root@localhost ~]# groupadd tgroup  
[root@localhost ~]# mkdir /project/  
[root@localhost ~]# chown root:tgroup /project/  
[root@localhost ~]# chmod 770 /project/  
[root@localhost ~]# setfacl -m u:st:rx /project/  
  
#给用户 st 赋予 r-x 权限,  
使用“u:用户名:权限”格式
```

给用户组设定 ACL 权限

```
groupadd tgroup2
```

```
setfacl -m g:tgroup2:rwx project/
```

#为组 tgroup2 分配 ACL 权限。使用“g:组名:权限”格式

7.1.3 最大有效权限与删除 ACL 权限

1. 最大有效权限 mask

- mask 是用来指定最大有效权限的。如果我给用户赋予了 ACL 权限，是需要和 mask 的权限“**相与**”才能得到用户的真正权限

A	B	and
r	r	r
r	-	-
-	r	-
-	-	-

修改最大有效权限

```
setfacl -m m:rx 文件名
```

#设定 mask 权限为 r-x。

使用“**m:权限**”格式

删除 ACL 权限

```
setfacl -x 用户名
```

#删除指定**用户**的 ACL 权限

```
setfacl -x g:组名 文件名
```

#删除指定**用户组**的 ACL 权限

```
setfacl -b 文件名
```

#会删除文件的**所有的** ACL 权限

7.1.4 默认 ACL 权限和递归 ACL 权限

1. 递归 ACL 权限

- 递归是父目录在设定 ACL 权限时，所有的子文件和子目录也会拥有相同的 ACL 权限
- `setfacl -m u:用户名:权限 -R 文件名` （在这里-R 必须放在最后）

2. 默认 ACL 权限

- 默认 ACL 权限的作用是如果给父目录设定了默认 ACL 权限，那么父目录中所有新建的子文件都会继承父目录的 ACL 权限。
- `setfacl -m d:u:用户名:权限 文件名`

7.2 文件特殊权限

7.2.1 SetUID

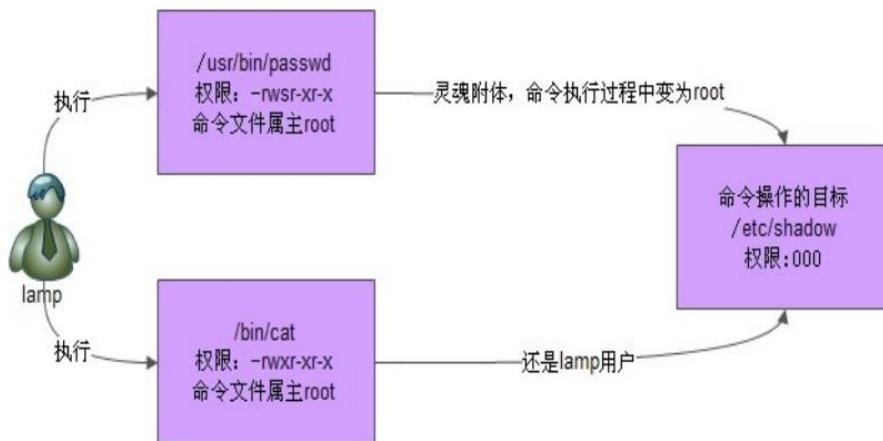
1. SetUID 的功能

- 只有可以执行的**二进制程序**才能设定 SUID 权限
- 命令执行者要对该程序拥有 x (执行) 权限
- 命令执行者在执行该程序时获得该程序文件属主的身份 (在执行程序的过程中灵魂附体为文件的属主)
- SetUID 权限只在该程序执行过程中有效，也就是说身份改变只在程序执行过程中有效
- 综上所说：类似于 Windows 中《以管理员身份运行》
- passwd 命令拥有 SetUID 权限，所以普通用户可以修改自己的密码

```
[root@localhost ~]# ll /usr/bin/passwd  
-rwsr-xr-x. 1 root root 27856 4月 1 06:57 /usr/bin/passwd 高亮红色提示此文件的SetUID权限
```

- cat 命令没有 SetUID 权限，所以说普通用户不能查看/etc/shadow 文件内容

```
[fox@localhost ~]$ ll /bin/cat  
-rwxr-xr-x. 1 root root 54080 8月 20 2019 /bin/cat  
[fox@localhost ~]$ cat /etc/shadow  
cat: /etc/shadow: 权限不够  
[fox@localhost ~]$ ll /etc/shadow  
----- 1 root root 1091 7月 26 16:35 /etc/shadow
```



设定 SetUID 的方法

- 4 代表 SUID
 - chmod 4755 文件名
 - chmod u+**S** 文件名

取消 SetUID 的方法

- 按平时的权限设定来，不写 4 可以
 - chmod 755 文件名
 - chmod u-**S** 文件名

危险的 SetUID

- ◆ 关键目录应严格控制写权限。比如“/”、“/usr”等
- ◆ 用户的密码设置要严格遵守密码三原则
- ◆ 对系统中默认应该具有 SetUID 权限的文件作一列表，定时检查有没有这之外的文件被设置了 SetUID 权限

7.2.2 SetGID

1. SetGID 针对文件的作用

- 只有可执行的二进制程序才能设置 SGID 权限
- 命令执行者要对该程序拥有 x (执行) 权限
- 命令执行在执行程序的时候，组身份升级为该程序文件的属组
- SetGID 权限同样只在该程序执行过程中有效，也就是说组身份改变只在程序执行过程中有效

```
[Fox@localhost ~]# ll /usr/bin/locate  
-rwx--s--x. 1 root slocate 40520 4月 11 2018 /usr/bin/locate  
[Fox@localhost ~]# ll /var/lib/mlocate/mlocate.db  
ls: 无法访问/var/lib/mlocate/mlocate.db: 权限不够  
[Fox@localhost ~]# logout  
  
[root@localhost ~]# ll /usr/bin/locate  
-rwx--s--x. 1 root slocate 40520 4月 11 2018 /usr/bin/locate  
[root@localhost ~]# ll /var/lib/mlocate/mlocate.db  
-rw-r-----. 1 root slocate 1039953 7月 27 12:45 /var/lib/mlocate/mlocate.db
```

- - /usr/bin/locate 是可执行二进制程序，可以赋予 SGID
 - 执行用户 lamp 对/usr/bin/locate 命令拥有执行权限
 - 执行/usr/bin/locate 命令时，组身份会升级为 slocate 组，而 slocate 组对/var/lib/mlocate/mlocate.db 数据库拥有 r 权限，所以普通用户可以使用 locate 命令查询 mlocate.db 数据库
 - 命令结束，lamp 用户的组身份返回为 lamp 组

2. SetGID 针对目录的作用

- 普通用户必须对此目录拥有 r 和 x 权限，才能进入此目录
- 普通用户在此目录中的有效组会变成此目录的属组
- 若普通用户对此目录拥有 w 权限时，新建的文件的默认属组是这个目录的属组

```
drwxr-srwx. 2 root root 4096 7月 27 14:18 uid_test  
drwxr-xr-x. 13 root root 4096 6月 29 18:40 usr  
drwxr-xr-x. 20 root root 4096 7月 18 13:02 var  
[fox@localhost uid_test]$ touch test_file  
[fox@localhost uid_test]$ ll ./test_file  
test test_file  
[fox@localhost uid_test]$ ll ./test_file  
-rw-rw-r--. 1 fox root 0 7月 27 14:19 ./test_file
```

- - [root@localhost ~]# cd /tmp/
 - [root@localhost tmp]# mkdir dtest
 - [root@localhost tmp]# chmod g+s dtest
 - [root@localhost tmp]# ll -d dtest/
 - [root@localhost tmp]# chmod 777 dtest/
 - [root@localhost tmp]# su – lamp
 - [lamp@localhost ~]# cd /tmp/dtest/
 - [lamp@localhost dtest]# touch abc
 - [lamp@localhost dtest]# ll

设定 SetGID

- 2 代表 SGID
 - chmod 2755 文件名
 - chmod g+s 文件名

取消 SetGID

- 按普通设置权限来设定
 - chmod 755 文件名
 - chmod g-S 文件名

7.2.3 Sticky BIT

1. SNIT 黏着位作用

- 粘着位目前**只对目录有效**
- 普通用户对该目录拥有 w 和 x 权限，即普通用户可以在此目录拥有写入权限
- 如果没有粘着位，因为普通用户拥有 w 权限，所以可以删除此目录下所有文件，包括其他用户建立的文件。一旦赋予了粘着位，除了 root 可以删除所有文件，普通用户就算拥有 w 权限，也只能删除自己建立的文件，不能删除其他用户建立的文件

[fox@localhost uid_test]\$ ll -d /tmp
drwxrwxrwt 12 root root 4096 7月 27 14:19 /tmp

设定黏着位

- 1 代表 Sticky BIT
 - chmod 1755 目录名
 - chmod o+t 文件名

取消 SetGID

- 按普通设置权限来设定
 - chmod 777 文件名
 - chmod o-t 文件名

7.3 文件系统属性 chattr 权限

查看文件系统属性

lsattr [选项] 文件名

选项：

- a 显示所有文件和目录
- d 若目标是目录，仅列出目录本身的属性，而不是子文件的

chattr 命令格式

chattr [+ -=][选项] 文件或目录名

- +：增加权限
- ：删除权限
- =：等于某权限

[选项]：

- i
 - 对文件：如果对文件设置 i 属性，那么不允许对文件进行 删除、改名，也不能添加和修改数据。也就是不能做任何修改。
 - 对目录：如果对目录设置 i 属性，那么只能修改目录下文件的数据，但不允许建立和删除文件
- a
 - 对文件：如果对文件设置 a 属性，那么只能在文件中增加数据，但是不能删除也不能修改数据（通过 echo 加入的内容 >> 文件）

```
[fox@localhost uid_test]$ echo 我要写入的内容 是这个 >> test_file
[fox@localhost uid_test]$ cat test_file
我要写入的内容 是这个
[fox@localhost uid_test]$
```
 - 对目录：如果对目录 设置 a 属性，那么只允许在目录中建立文件和修改文件内容（修改文件内容 可以用 vim，可以删除文件内容），但是不允许删除文件

总结：

- chattr 对所有用户生效，包括 root 用户!!! 因为此命令是为了防止误操作的
- 简而言之，对于文件 i 选项封存了所有数据，不得做任何修改。而对目录，同样不可以新建或删除任何文件，但是可以修改已存在文件内容
- 对于文件 a 选项相当于仅封存了已有的文件，仅可以通过 echo 的方式添加，不得修改原有数据。
对于目录，只能建立新文件和修改原有文件内容

7.4 sudo 权限

1. sudo 权限

- root 把本来只能超级用户执行的命令赋予普通用户执行
- sudo 的操作对象是系统命令

2. sudo 使用

```
[root@localhost ~]# visudo #实际修改的是/etc/sudoers 文件
```

```
root ALL=(ALL) ALL ← 第一个 ALL 指的是来源 ip, 也就是哪台计算机被管理  
#用户名 被管理主机的地址=（可使用的身份） 授权命令（绝对路径）
```

```
# %wheel ALL=(ALL) ALL  
#%组名 被管理主机的地址=（可使用的身份） 授权命令（绝对路径）
```

3. 授权 fox 用户可以重启服务器

```
[root@localhost ~]# visudo  
sc ALL= /sbin/shutdown -r now ← 这里的表达越含糊，被赋予的用户权限就越大
```

4. 普通用户执行 sudo 赋予的命令

```
[root@localhost ~]# su – fox  
[fox@localhost ~]$ sudo -l #查看可用的 sudo 命令  
[fox@localhost ~]$ sudo /sbin/shutdown -r now #普通用户执行 sudo 赋予的命令
```

八. 文件系统管理

8.1 回顾分区和文件系统

1. 分区类型

- 主分区：总共最多只能分 4 个
- 扩展分区：只能有一个，也算作主分区的一种，也就是说主分区加扩展分区最多有四个。但是扩展分区不能存储数据和格式化，必须再划分成逻辑分区才能使用（就像大柜子不能直接使用，要分成小的隔断）
- 逻辑分区：逻辑分区是在扩展分区中划分的，如果是 IDE 硬盘，Linux 最多支持 59 个逻辑分区；如果是 SCSI 硬盘，Linux 最多支持 11 个逻辑分区

2. 分区表示方法

- 例一



分区的设备文件名	
主分区1	/dev/sda1
主分区2	/dev/sda2
主分区3	/dev/sda3
扩展分区	/dev/sda4
逻辑分区1	/dev/sda5
逻辑分区2	/dev/sda6
逻辑分区3	/dev/sda7

- 例二



分区的设备文件名	
主分区1	/dev/sda1
扩展分区	/dev/sda2
逻辑分区1	/dev/sda5
逻辑分区2	/dev/sda6
逻辑分区3	/dev/sda7

sda1 → s代表 SATA 硬盘接口

→ a 代表第一块硬盘，由此类推第二块为 b ……

→ 最后的 1234567 代表分区号

3. 文件系统

- ext2：是 ext 文件系统的升级版本，Red Hat Linux7.2 版本以前的系统默认都是 ext2 文件系统。1993 年发布，最大支持 16TB 的分区和最大 2TB 的文件（ $1\text{TB}=1024\text{GB}=1024*1024\text{KB}$ ）
- ext3：ext3 文件系统是 ext2 文件系统的升级版本，最大的区别就是带日志功能，以 在系统突然停止时提高文件系统的可靠性。支持最大 16TB 的分区和最大 2TB 的文件
- ext4：它是 ext3 文件系统的升级版。ext4 在性能、伸缩性和可靠性方面进行了大量改进。EXT4 的变化可以说是翻天覆地的，比如向下兼容 EXT3、最大 1EB 文件系统和 16TB 文件、无限数量子目录、Extents 连续数据块概念、多块分配、延迟分配、持久预分配、快速 FSCK、日志校验、无日志模式、在线碎片整理、inode 增强、默认启用 barrier 等。是 CentOS 6.3 的默认文件系统（ $1\text{EB}=1024\text{PB}=1024*1024\text{TB}$ ）

8.2 文件系统常用命令

8.2.1 df 命令、du 命令、fsck 命令、dump2fs 命令

文件系统查看命令 df

df [选项] [挂载点]

选项:

- a 显示所有的文件系统信息，包括特殊文件系统，如 /proc、/sysfs
- h 使用习惯单位显示容量，如 KB, MB 或 GB 等
- T 显示文件系统类型
- m 以 MB 为单位显示容量
- k 以 KB 为单位显示容量。默认就是以 KB 为单位

统计目录或文件大小

du [选项] [目录或文件名]

选项:

- a 显示每个子文件的磁盘占用量。默认只统计子目录的磁盘占用量
- h 使用习惯单位显示磁盘占用量，如 KB, MB 或 GB 等
- s 统计总占用量，而不列出子目录和子文件的占用量

du 命令和 df 命令的区别:

- df 命令是从文件系统考虑的，不光要考虑文件占用的空间，还要统计被命令或程序占用的空间（最常见的是文件已经删除，但是程序并没有释放空间，但这段空间并不能被系统利用 → 所以说有时候服务器/电脑要定时重启）
- du 命令是面向文件的，只会计算文件或目录占用的空间

显示磁盘状态命令 dump2fs

dump2fs 分区设备文件名

文件系统修复命令 fsck

fsck [选项] 分区设备文件名

选项:

- a: 不用显示用户提示，自动修复文件系统
- y: 自动修复。和-a 作用一致，不过有些文件系统只支持-y

不建议手工使用此命令，有可能会造成系统崩溃

系统在启动的时候会自动执行此命令（基于 lost+found 文件夹汇总的内容）

8.2.2 挂载命令

1. 查询与自动挂载

- **mount [-l]** #查询系统中已经挂载的设备, -l 会显示卷标名称
- **mount -a** #依据配置文件/etc/fstab 的内容, 自动挂载 【这个配置文件千万不能写错】

2. 挂载命令格式

挂载命令格式	
mount [-t 文件系统] [-L 卷标名] \[-o 特殊选项] 设备文件名 挂载点	
选项:	
-t 文件系统: 加入文件系统类型来指定挂载的类型, 可以 ext3、ext4 、iso9660 等文件系统	
-L 卷标名: 挂载指定卷标的分区, 而不是安装设备文件名挂载	
-o 特殊选项: 可以指定挂载的额外选项	

参数	说明
atime/noatime	更新访问时间/不更新访问时间。访问分区文件时, 是否更新文件的访问时间, 默认为更新
async/sync	异步/同步, 默认为异步
auto/noauto	自动/手动, mount -a命令执行时, 是否会自动安装/etc/fstab文件内容挂载, 默认为自动
defaults	定义默认值, 相当于rw,suid,dev,exec,auto,nouser,async这七个选项
exec/noexec	执行/不执行, 设定是否允许在文件系统中执行可执行文件, 默认是exec允许
remount	重新挂载已经挂载的文件系统, 一般用于指定修改特殊权限
rw/ro	读写/只读, 文件系统挂载时, 是否具有读写权限, 默认是rw
suid/nosuid	具有/不具有SUID权限, 设定文件系统是否具有SUID和SGID的权限, 默认是具有
user/nouser	允许/不允许普通用户挂载, 设定文件系统是否允许普通用户挂载, 默认是不允许, 只有root可以挂载分区
usrquota	写入代表文件系统支持用户磁盘配额, 默认不支持
grpquota	写入代表文件系统支持组磁盘配额, 默认不支持

示例:

```
[root@localhost ~]# mount -o remount,noexec /home          #重新挂载/boot 分区, 并使用 noexec 权限
[root@localhost sh]# cd /home
[root@localhost boot]# vi hello.sh
[root@localhost boot]# chmod 755 hello.sh
[root@localhost boot]# ./hello.sh
[root@localhost boot]# mount -o remount,exec /home      #记得改回来啊, 要不会影响系统启动的
```

8.2.3 挂载光盘与 U 盘

1. 挂载光盘

```
[root@localhost ~]# mkdir /mnt/cdrom/          #建立挂载点  
  
[root@localhost ~]# mount -t iso9660 /dev/cdrom /mnt/cdrom/    #挂载光盘, 一下两者二选一  
[root@localhost ~]# mount /dev/sr0 /mnt/cdrom/
```

2. 卸载命令

```
[root@localhost ~]# umount                  #设备文件名或挂载点  
[root@localhost ~]# umount /mnt/cdrom
```

3. 挂载 U 盘

```
[root@localhost ~]# fdisk -l                      #查看 U 盘设备文件名  
[root@localhost ~]# mount -t vfat /dev/sdb1 /mnt/usb/  
注意：Linux 默认是不支持 NTFS 文件系统的
```

8.2.4 支持 NTFS 文件系统

1. 下载 NTFS-3G 插件

<https://www.tuxera.com/community/open-source-ntfs-3g/>

2. 安装 NTFS-3G

```
[root@localhost ~]# tar -zxvf ntfs-3g_ntfsprogs-2013.1.13.tgz #解压
```

```
[root@localhost ~]# cd ntfs-3g_ntfsprogs-2013.1.13 #进入解压目录
```

```
[root@localhost ntfs-3g_ntfsprogs-2013.1.13]# ./configure #编译器准备。没有指定安装目录，安装到默认位置中
```

```
[root@localhost ntfs-3g_ntfsprogs-2013.1.13]# make #编译
```

```
[root@localhost ntfs-3g_ntfsprogs-2013.1.13]# make install #编译安装
```

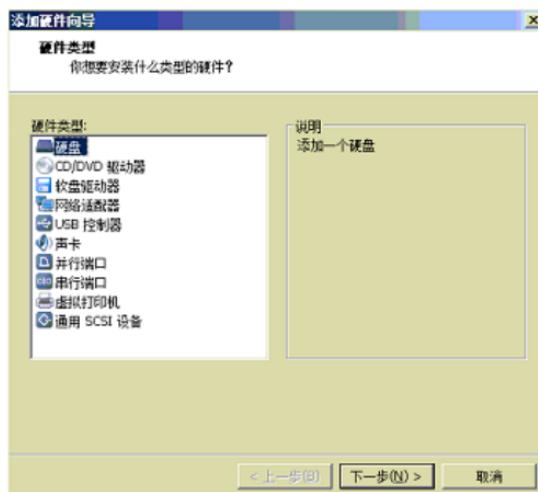
3. 使用

```
[root@localhost ~]# mount -t ntfs-3g 分区设备文件名 挂载点
```

8.3 fdisk 分区

8.3.1 fdisk 命令分区过程

1. 添加新硬盘



2. 查看新硬盘

```
fdisk -l
```

3. 使用 fdisk 命令分区

```
fdisk /dev/sdb
```

fdisk交互指令说明	
命令	说明
a	设置可引导标记
b	编辑bsd磁盘标签
c	设置DOS操作系统兼容标记
d	删除一个分区
l	显示已知的文件系统类型。82为Linux swap分区，83为Linux分区
m	显示帮助菜单
n	新建分区
o	建立空白DOS分区表
p	显示分区列表
q	不保存退出
s	新建空白SUN磁盘标签
t	改变一个分区的系统ID
u	改变显示记录单位
v	验证分区表
w	保存退出
x	附加功能（仅专家）

4. 重新读取分区表信息

```
partprobe #建议每次分区完都执行
```

5. 格式化分区

```
mkfs -t ext4 /dev/sdb1
```

6. 建立挂载点并挂载

```
[root@localhost ~]# mkdir /disk1
```

```
[root@localhost ~]# mount /dev/sdb1 /disk1/
```

8.3.2 分区自动挂载与 fstab 文件修复

1. /etc/fstab 文件

```
UUID=324f0021-04e4-4af7-b936-75357bc8b524 /          ext4    defaults    1 1
UUID=f024278e-b59a-4993-af1e-b2b76a26d2f0 /boot      ext4    defaults    1 2
UUID=f2806290-3490-4eab-bf8a-6707cc9fb8d8 /home      ext4    defaults    1 2
UUID=a466fa15-f74e-4051-82f7-70d17ed3dde1 swap       swap    defaults    0 0
```

- 第一字段：分区设备文件名或 UUID（硬盘通用唯一识别码）
- 第二字段：挂载点
- 第三字段：文件系统名称
- 第四字段：挂载参数
- 第五字段：指定分区是否被 dump 备份，0 代表不备份，1 代表每天备份，2 代表不定期备份
- 第六字段：指定分区是否被 fsck 检测，0 代表不检测，其他数字代表检测的优先级，那么当然 1 的优先级比 2 高
- 我们自己挂载的新硬盘备份的数字不应大于根分区 1

2. 分区自动挂载

```
[root@localhost ~]# vi /etc/fstab
…省略部分输出…
/dev/sdb5 /disk5  ext4    defaults    1 2
```

3. /etc/fstab 文件修复

```
[root@localhost ~]# mount -o remount,rw / ← 前提是根分区没有被搞坏
```

8.4 分配 swap 分区

1. 新建 swap 分区

```
[root@localhost ~]# fdisk /dev/sdb
```

别忘记把分区 ID 改为 82

2、格式化

```
[root@localhost ~]# mkswap /dev/sdb1
```

3、加入 swap 分区

```
[root@localhost ~]# swapon /dev/sdb1      #加入 swap 分区
```

```
[root@localhost ~]# swapoff /dev/sdb1      #取消 swap 分区
```

4、swap 分区开机自动挂载

```
[root@localhost ~]# vi /etc/fstab  
/dev/sdb1    swap    swap    defaults 0 0
```

5、free 命令

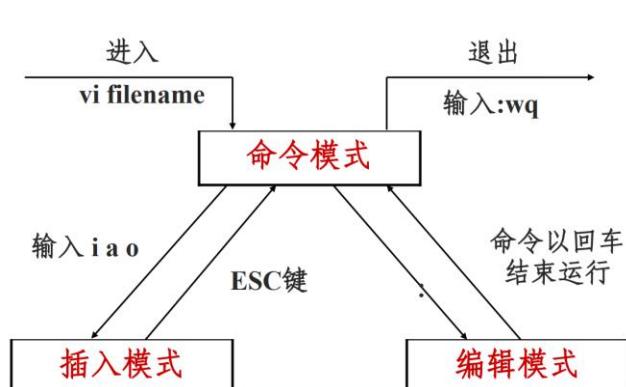
```
[root@localhost ~]# free      #查看内存与 swap 分区使用状况
```

- **cached (缓存)**: 是指把读取出来的数据保存在内存当中，当再次读取时，不用读取硬盘而直接从内存当中读取，加速了数据的读取过程
- **buffer (缓冲)**: 是指在写入数据时，先把分散的写入操作保存到内存当中，当达到一定程度再集中写入硬盘，减少了磁盘碎片和硬盘的反复寻道，加速了数据的写入过程

九 Vim 编辑器

9.1 Vim 常用操作

- Vim 简介：
 - Vim 是一个功能强大的全屏幕文本编辑器，是 Linux/UNIX 上最常用的文本编辑器，它的作用是建立、编辑、显示文本文件。
 - Vim 没有菜单，只有命令
- Vim 工作模式



保存和退出命令

命令	作用
:w	保存修改
:w new_filename	另存为指定文件
:wq	保存修改并退出
ZZ	快捷键，保存修改并退出
:q!	不保存修改退出
:wq!	保存修改并退出（文件所有者及root可使用）

插入命令

命令	作用
a	在光标所在字符后插入
A	在光标所在行尾插入
i	在光标所在字符前插入
I	在光标所在行行首插入
o	在光标下插入新行
O	在光标上插入新行

定位命令

命令	作用
:set nu	设置行号
:set nonu	取消行号
gg	到第一行
G	到最后一行
nG	到第n行
:n	到第n行

命令	作用
\$	移至行尾
0	移至行首

复制和剪切命令

命令	作用
yy	复制当前行
nyy	复制当前行以下n行
dd	剪切当前行
ndd	剪切当前行以下n行
p、P	粘贴在当前光标所在行下或行上

命令	作用
x	删除光标所在处字符
nx	删除光标所在处后n个字符
dd	删除光标所在行，ndd删除n行
dG	删除光标所在行到文件末尾内容
D	删除光标所在处到行尾内容
:n1,n2d	删除指定范围的行

替换和取消命令

命令	作用
r	取代光标所在处字符
R	从光标所在处开始替换字符，按Esc结束
u	取消上一步操作

搜索和搜索替换命令

命令	作用
/string	搜索指定字符串 搜索时忽略大小写 :set ic
n	搜索指定字符串的下一个出现位置
:%s/old/new/g	全文替换指定字符串
:n1,n2s/old/new/g	在一定范围内替换指定字符串

ESC 命令模式

~ 转换大小写	! 外部过滤器	@ 运行宏	# prev ident	\$ 行尾	% 匹配	^ "软" 行首	& 重复 :s	* next ident	(句首) 下一行	"soft" bol	+ 后一行
~ 跳转到标注	1 标注	2	3	4	5	6	7	8	9	0 行首	- 前一行	= 自动 3
Q 切换至 ex模式	W 下一单词	E 词尾	R 替换模式	T back 'till	Y 拷贝行	U 行内命令	I 到行首 插入	O 分段 (前)	P 粘贴 (前)	{ 段首	段尾	
Q 录制宏	W 下一单词	E 词尾	R 替换字符	T 'till	Y 拷贝	U 撤消命令	I 插入模式	O 分段 (后)	P 粘贴 1	{ 段首		段尾
A 在行尾附加	S 删除行并插入	D 删除至行尾	F 行内字符串查找	G 文尾/行号	H 屏幕顶行	J 合并两行	K 帮助	L 屏幕底行	;	· 杂项		· 杂项
a 附加	S 删除字符并插入	D 1,3 删除	F 行内字符串查找	G 附加 6 命令	H 屏幕顶行	J 合并两行	K 帮助	L 屏幕底行	;	· 杂项		· 杂项
Z 退出 4	X 退格	C 修改 1,3 全行末	V 行模式	B 前一单词	N 上一处查找	M 中间行	< 反缩进 3	> 缩进 3	?·	"' 寄存器 1 标识	行首/列	\• 未用!
Z 附加 5	X (字符) 删除	C 1,3 修改	V 可视模式	B 前一单词	N 下一处查找	M 设置标注	, v/T/f/F 反向	, v/T/f/F 反向	?·	' 跳转到行首		
							,	,	/.	· 重复命令		

动作

移动光标, 或者定义操作的范围

主要ex命令:

:w(保存), :q(退出), :q!(不保存退出)

:e f(打开文件 f),

:%s/x/y/g(全局替换 'x'),

:h(帮助 in vim), :new(新建文件 in vim),

备注:

(1) 在拷贝/粘贴/删除 命令前使用 "x(x=a..z,*)

使用命令的寄存器'剪贴板'(如: "ay\$ 拷贝剩余的行内容至寄存器 'a')

(2) 命令前添加数字 多遍重复操作 (e.g.: 2p, d2w, 5i, d4d)

(3) 重复本字符在光标所在行执行操作 (dd = 删除本行, >> = 行首缩进)

(4) ZZ 保存退出, ZQ 不保存退出

(5) zt: 移动光标所在行至屏幕顶端,

zb: 底端, zz: 中间

(6) gg: 文首 (vim only),

gf: 打开光标处的文件名 (vim only)

命令

直接执行的命令, 红色命令 进入编辑模式

后面跟随表示操作范围的指令

操作

特殊功能, 需要额外的输入

后跟字符参数

Q

w,e,b命令

小写(B): quux!foo, bar!, baz!;

大写(B): quux(foo, bar, baz);

vi / vim 键盘图

ESC

模式

命令模式

ESC

命令

模式

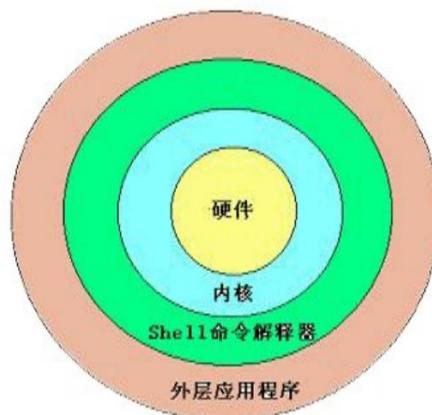
9.2 Vim 使用技巧

创建及打开		标记		定位命令	
vi text.txt	打开text.txt	m一个字母	mark - 添加标记	:set nu	设置行号
vi text.txt +行号	打开text并跳转到行号处执行	`一个字母	转到标记	:set nonu	取消行号
vi text.txt +	打开text 并跳转到文件尾	'一个字母	转到标记处的第一个非空字符	gg	到第一行
					
插入命令		搜索和替换命令		缩排和重复执行	
a	在光标所在字符后插入	/字符串	从光标处搜索指定字符串	>>	向右增加缩进
A	在光标所在行尾插入		按n键向下跳转，N向上	<<	向左减少缩进
i	在光标所在的字符前插入	?字符串	从光标处搜索指定字符串		重复执行
I	在光标所在行的行首插入		按n键向上跳转，N向下		
o	光标下插入新行	(忽略大小写 :set ic)		将某行到某行进行缩进： 按V进入可视模式。 使用nG选中固定行。 使用 . 重复执行按一下执行一次	
O	光标上插入新行	*	向后查找当前光标所在的单词		
		#	向前查找当前光标所在的单词		
		:s/OLD/NEW	将本行或选中内容的一个OLD替换为NEW		
		:s/OLD/NEW/g	将本行或选中内容的所有OLD→NEW		
		:X,Ys/OLD/NEW/g	将X到Y行所有OLD→NEW		
		:%s/OLD/NEW/g	将文件中所有OLD→NEW		
		:%s/OLD/NEW/gc	通过用户确认将OLD→NEW (全部、下一个、取消...)		
		y	yes - 替换		
		n	no - 不替换		
		a	all - 替换所有		
		q	quit - 退出		
		!	last - 替换最后一个、把光标移到行首		
移动					
w	word - 向后移动一个单词	替换		删除命令	
b	back - 向前移动一个单词	r	取代光标所在处单个字符	d (与移动命令配合)	与移动命令配合的删除命令
0	行首	R	从光标处开始取代，按ESC结束	x	删除光标处字符或者选中的字符
^	行首，第一个非空字符	u	取消上一步操作	X	删除光标前字符
\$	行尾			nx	从光标处删除n个字符
- +	向上一行 向下一行			D 或者 d\$	删除光标所在处到行尾内容
^B ^F	向上翻页 向下翻页 (光标到页面两端)			dG	删除光标所在行到文件末尾内容
^U ^D	向上翻页 向下翻页 (光标保持位置)			:起始行.结束行d	删除指定范围行
^Y ^E	向上滚动 向下滚动 (单行移动，光标保持位置)			几种常见的配合使用	
L	转到屏幕底部			dw	光标处删除到单词末尾
z 然后 Enter	光标所在行滚动到屏幕顶部			d0	光标处删除到行尾
z 然后 .	光标所在行滚动到屏幕中央			d}]	光标位置删除到段落结尾
z 然后 -	光标所在行滚动到屏幕底部			d指定行G	光标所在行删除到指定行之间内容
()	上一句 下一句			d'm	光标所在行删除到标记m之间内容
[[]]	上一节 下一节				
{}	上一段 下一段				
H	Head - 仅光标至屏幕顶部	复制和剪切		撤销和恢复撤销	
M	Middle - 仅光标至屏幕中间	y (与移动命令配合)	与移动命令配合的复制	u	undo - 撤销
L	Low - 仅光标至屏幕底部	yy	复制当前行	^r	redo - 恢复撤销
%	一对括号间的快速切换	nyy	复制当前行以下n行		
		dd	剪切当前行		
		ndd	剪切当前行以下n行		
		p	粘贴在光标所在行下		
		P	粘贴在光标所在行上		
文本选择 (可视模式)					
v	可视模式 (从光标开始处选择)	其他命令		保存及退出	
V	可视行模式 (光标经过的完整行)	数字 命令	让该命令重复指定的次数	:w	保存修改
^v	可视块模式 (垂直方向选择文本)			:w 新名字	另存为指定文件
可视模式下可以和移动光标命令连用，用于选择文本					
				:q	不保存退出
				:q!	强行不保存并退出
				:wq 等价于 :x	
				:wq!	强行保存并退出

十. Shell 概述

1. Shell 是什么

- Shell 是一个命令解释器，它为用户提供了一个向 Linux 内核发送请求以便运行程序的界面。系统级程序，用户可以用 Shell 来启动、挂起、停止甚至是编写一些程序。



- Shell 还是一个功能相当强大的编程语言，易编写，易调试，灵活性较强。Shell 是解释执行的脚本语言，在 Shell 中可以直接调用 Linux 系统命令。

2. Shell 的分类

- **Bourne Shell:** 从 1979 起 Unix 就开始使用 Bourne Shell，Bourne Shell 的主文件名为 sh
- C Shell: C Shell 主要在 BSD 版的 Unix 系统中使用，其语法和 C 语言相类似而得名
- Shell 的两种主要语法类型有 Bourne 和 C，这两种语法彼此不兼容。
 - Bourne 家族主要包括 sh、ksh、Bash、psh、zsh
 - C 家族主要包括: csh、tcsh
- **Bash:** Bash 与 sh 兼容，现在使用的 Linux 就是使用 Bash 作为用户的基本 Shell

3. Linux 支持的 Shell

- 在目录/etc/shells

```
[root@localhost ~]# cat /etc/shells
/bin/sh
/bin/bash
/usr/bin/sh
/usr/bin/bash
/bin/tcsh
/bin/csh
```

十一. Shell 脚本的执行方式

1. echo 输出命令

echo [选型] [输出内容] ← 输出内容用单引号或双引号

-e 支持反斜杠控制的字符转换

控制字符	作用
\\\	输出\本身
\a	输出警报音
\b	退格键，也就是向左删除键
\c	取消输出行末的换行符。和“-n”选项一致
\e	ESCAPE键
\f	换页符
\n	换行符
\r	回车键
\t	制表符，也就是Tab键
\v	垂直制表符
\0nnn	按照八进制ASCII码表输出字符。其中0为数字零，nnn是三位八进制数
\xhh	按照十六进制ASCII码表输出字符。其中hh是两位十六进制数

比如：

- 删 除 左 侧 字 符： echo -e "ab\bc"
- 制 表 符 和 换 行 符： echo -e "a\tb\tc\n d\t e\t f"
- 按 照 16 进 制 A S C I I 码 输出： echo -e "\x61\t\x62\t\x63\n\x64\t\x65\t\x66"
- 输 出 颜 色：
 - #30m=黑色，31m=红色，32m=绿色，33m=黄色
 - #34m=蓝色，35m=洋红，36m=青色，37m=白色

2. 第一个脚本

[root@localhost sh]# vi hello.sh

#!/bin/bash ← 第一行不是注释，而是一个 shell 脚本不可缺的

The first program

2020 年 7 月 31 日 12:38:36

echo -e "Hello world"

3. 脚本执行

- 赋予权限，直接运行
 - chmod 755 hello.sh
 - ./hello.sh
- 通过 Bash 直接调用执行脚本
 - bash hello.sh ← 通过 bash 调用执行的可以不赋予执行权限，因为系统是通过 bash 调用的

十二. Shell 脚本在不同环境编写的兼容性问题

- 因为 Windows 中编写和 Linux 编写的程序程序中存在些许不同
 - ◆ `cat -A` ← 查询所有的内容，包括隐藏字符
 - ◆ 在 Linux 中回车符识别为\$符，而在 Windows 中不是^CTRL\$
- 因为回车符符不同，所以 Linux 中执行 Windows 中编写的脚本程序，就会因为回车符的不同导致格式不匹配
- 把格式从 Windows 格式转换为 Unix 格式：
 - ◆ `dos2unix` 文件名
 - ◆ 如果没有安装，使用 `yum` 或 `rpm` 装上就好了

十三. Bash 的基本功能

13.1 历史命令与命令补全

1. 历史命令

历史命令

`history [选项] [历史命令保存文件]`

选项:

-c: 清空历史命令

-w: 把缓存中的历史命令写入历史命令保存文件 `~/.bash_history`

- 历史命令默认会保存 1000 条,可以在环境 变量配置文件`/etc/profile` 中进行修改
- 历史命令的调用
 - ◆ 使用上、下箭头调用以前的历史命令
 - ◆ 使用 “!`n`” 执行第 `n` 条历史命令
 - ◆ 使用 “!!” 重复执行上一条命令
 - ◆ 使用 “!`字符串`” 重复执行最后一条以该字符串开头的命令

2. 命令与文件补全

- 在 Bash 中, 命令与文件补全是非常方便与常用的功能, 我们只要在输入命令或文件时, 按“Tab”键就会自动进行补全

13.2 命令别名与常用快捷键

1. 命令别名

- 设定命令别名 alias 别名='原命令'
- 查询命令别名 alias
- 删除别名 unalias

2. 命令执行时的顺序

- 第一顺位执行 用绝对路径或相对路径执行的命令
- 第二顺位执行 别名
- 第三顺位执行 Bash 的内部命令
- 第四顺位执行 按照\$PATH 环境变量定义的目录查找到的第一个命令

3. 让别名永久生效

- 修改 vi /root/.bashrc
- 用 alias 别名='原命令' 设置的别名在关机重启后就会失效

4. Bash 常用快捷键

快 捷 键	作 用
ctrl+A	把光标移动到命令行开头。如果我们输入的命令过长，想要把光标移动到命令行开头时使用。
ctrl+E	把光标移动到命令行结尾。
ctrl+C	强制终止当前的命令。
ctrl+L	清屏，相当于clear命令。
ctrl+U	删除或剪切光标之前的命令。我输入了一行很长的命令，不用使用退格键一个一个字符的删除，使用这个快捷键会更加方便
ctrl+K	删除或剪切光标之后的内容。
ctrl+Y	粘贴ctrl+U或ctrl+K剪切的内容。
ctrl+R	在历史命令中搜索，按下ctrl+R之后，就会出现搜索界面，只要输入搜索内容，就会从历史命令中搜索。
ctrl+D	退出当前终端。
ctrl+Z	暂停，并放入后台。这个快捷键牵扯工作管理的内容，我们在系统管理章节详细介绍。
ctrl+S	暂停屏幕输出。
ctrl+Q	恢复屏幕输出。

图中应该为小写，大写只是为了便于查看

13.3 输入输出重定向

1. 标准输入输出

设备	设备文件名	文件描述符	类型
键盘	/dev/stdin	0	标准输入
显示器	/dev/stdout	1	标准输出
显示器	/dev/stderr	2	标准错误输出

2. 输出重定向

- 输出重定向可以看作是本应当输出到屏幕上，但是输出到了文件中（重定向到文件中）

类型	符号	作用
标准输出重定向	命令 > 文件	以覆盖的方式，把命令的正确输出输出到指定的文件或设备当中。
	命令 >> 文件	以追加的方式，把命令的正确输出输出到指定的文件或设备当中。
标准错误输出重定向	错误命令 2>文件	以覆盖的方式，把命令的错误输出输出到指定的文件或设备当中。
	错误命令 2>>文件	以追加的方式，把命令的错误输出输出到指定的文件或设备当中。

正确输出和错误输出同时保存	命令 > 文件 2>&1	以覆盖的方式，把正确输出和错误输出都保存到同一个文件当中。
	命令 >> 文件 2>&1	以追加的方式，把正确输出和错误输出都保存到同一个文件当中。
	命令 &>文件	以覆盖的方式，把正确输出和错误输出都保存到同一个文件当中。
	命令 &>>文件	以追加的方式，把正确输出和错误输出都保存到同一个文件当中。
	命令>>文件1 2>>文件2	把正确的输出追加到文件1中，把错误的输出追加到文件2中。

3. 输入重定向

输入重定向	● 命令<文件 把文件作为命令的输入 ● 命令<<标识符 标识符 ← 遇到相同的标识符停止输入 把标识符之间的内容作为命令的输入
<p>wc [选项] [文件名]</p> <p>选项：</p> <ul style="list-style-type: none">-c 统计字节数-w 统计单词数-l 统计行数	

13.4 多命令顺序执行与管道符

1. 多命令顺序执行

多命令执行符	格式	作用
;	命令1 ; 命令2	多个命令顺序执行，命令之间没有任何逻辑联系
&&	命令1 && 命令2	逻辑与 当命令1正确执行，则命令2才会执行 当命令1执行不正确，则命令2不会执行
	命令1 命令2	逻辑或 当命令1执行不正确，则命令2才会执行 当命令1正确执行，则命令2不会执行

例子：

- [root@localhost ~]# ls ; date ; cd /user ; pwd
- [root@localhost ~]# dd if=输入文件 of=输出文件 bs=字节数 count=个数
选项：

if=输入文件 指定源文件或源设备
of=输出文件 指定目标文件或目标设备
bs=字节数 指定一次输入/输出多少字节，即把这些字节看做一个数据块
count=个数 指定输入/输出多少个数据块

例子： [root@localhost ~]# date ; dd if=/dev/zero of=/root/testfile bs=1k count=100000 ; date

- [root@localhost ~]# ls anaconda-ks.cfg && echo yes
- [root@localhost ~]# ls /root/test || echo no
- [root@localhost ~]# 命令 && echo yes || echo no ← 此命令用于判断命令是否正确执行

2. 管道符

管道符

命令一 | 命令二

解释：用命令一的正确输出作为命令二的操作对象

例子：

```
[root@localhost ~]# ll -a /etc/ | more  
[root@localhost ~]# netstat -an | grep "ESTABLISHED"
```

grep 文件内字符串搜索

grep [选项] "搜索内容" 文件名

选项：

-i: 忽略大小写
-n: 输出行号
-v: 反向查找
--color=auto 搜索出的关键字用颜色显示

13.5 通配符与其他特殊符号

1. 通配符

通配符	作用
?	匹配一个任意字符
*	匹配0个或任意多个任意字符，也就是可以匹配任何内容
[]	匹配中括号中任意一个字符。例如：[abc]代表一定匹配一个字符，或者是a，或者是b，或者是c。
[-]	匹配中括号中任意一个字符，-代表一个范围。例如：[a-z]代表匹配一个小写字母。
[^]	逻辑非，表示匹配不是中括号内的一个字符。例如：[^0-9]代表匹配一个不是数字的字符。

比如：

```
[root@localhost ~]# cd /tmp/  
[root@localhost tmp]# rm -rf *  
[root@localhost tmp]# touch abc  
[root@localhost tmp]# touch abcd  
[root@localhost tmp]# touch 012  
[root@localhost tmp]# touch 0abc  
[root@localhost tmp]# ls ?abc  
[root@localhost tmp]# ls [0-9]*  
[root@localhost tmp]# ls [^0-9]*
```

2. Bash 中其他特殊符号

符号	作用
,,	单引号。在单引号中所有的特殊符号，如“\$”和“`”(反引号)都没有特殊含义。
""	双引号。在双引号中特殊符号都没有特殊含义，但是“\$”、“`”和“\”是例外，拥有“调用变量的值”、“引用命令”和“转义符”的特殊含义。
..	反引号。反引号括起来的内容是系统命令，在Bash中会先执行它。和\$()作用一样，不过推荐使用\$()，因为反引号非常容易看错。
\$()	和反引号作用一样，用来引用系统命令。
#	在Shell脚本中，#开头的行代表注释。
\$	用于调用变量的值，如需要调用变量name的值时，需要用\$name的方式得到变量的值。
\	转义符，跟在\之后的特殊符号将失去特殊含义，变为普通字符。如\\$将输出“\$”符号，而不当做是变量引用。

- 反引号与\$()
 - [root@localhost ~]# echo `ls`
 - [root@localhost ~]# echo \$(date) ← 两者的用法是相同的
- 单引号和双引号
 - [root@localhost ~]# name=sc
 - [root@localhost ~]# echo '\$name' ← 单引号中的所有内容视为字符串
 - [root@localhost ~]# echo "\$name"
 - [root@localhost ~]# echo '\$(date)' ←
 - [root@localhost ~]# echo "\$(date)" ← 双引号中的【\$`\\】具有特殊含义

十四. Bash 中的变量

14.1 用户自定义变量

1. 什么是变量

- 变量是计算机内存的单元，其中存放的值可以改变。当 Shell 脚本需要保存一些信息时，如一个文件名或是一个数字，就把它存放在一个变量中。每个变量有一个名字，所以很容易引用它。使用变量可以保存有用信息，使系统获知用户相关设置，变量也可以用于保存暂时信息

2. 变量的设置规则

- 变量名称可以由字母、数字和下划线组成，但是不能以数字开头。如果变量名是“2name”则是错误的
- 在 Bash 中，变量的默认类型都是字符串型，如果要进行数值运算，则必须指定变量类型为数值型
- 变量用等号连接值，等号左右两侧不能有空格
- 变量的值如果有空格，需要使用单引号或双引号包括
- 在变量的值中，可以使用“\”转义符
- 如果需要增加变量的值，那么可以进行变量值的叠加。不过变量需要用双引号包含“\$变量名”或用\${变量名}包含
 - tmp=&tmp"456
 - tmp=\${tmp}789
- 如果是把命令的结果作为变量值赋予变量，则需要使用`反引号`或 \$() 包含命令
- 环境变量名建议大写，便于区分

3. 变量分类

- 用户自定义变量
- 环境变量：这种变量中主要保存的是和系统操作环境相关的数据
- 位置参数变量：这种变量主要是用来向脚本当中传递参数或数据的，变量名不能自定义，变量作用是固定的（位置参数变量也是预定义变量的一种，因为数量过多所以单独讲解）
- 预定义变量：是 Bash 中已经定义好的变量，变量名不能自定义，变量作用也是固定的

4. 本地变量

- 变量定义
 - [root@localhost ~]# name="shen chao"
- 变量叠加
 - [root@localhost ~]# aa=123
 - [root@localhost ~]# aa="\$aa"456 ← 两种的作用是相同的
 - [root@localhost ~]# aa=\${aa}789
- 变量的调用
 - [root@localhost ~]# echo \$name
- 变量查看
 - [root@localhost ~]# set
- 变量删除
 - [root@localhost ~]# unset name

14.2 环境变量

1. 环境变量是什么

- 用户自定义变量只在当前的 Shell 中生效，而环境变量会在当前 Shell 和这个 Shell 的所有子 Shell 当中生效。如果把环境变量写入相应的配置文件，那么这个环境变量就会在所有的 Shell 中生效

2. 设置环境变量

- 声明变量 `export 变量名=变量值` `export`-输出，出口
- 查询变量 `env`
- 删除变量 `unset 变量名`

3. 常见的系统环境变量

- PATH: 系统查找命令的路径
 - [root@localhost ~]# echo \$PATH

```
[root@localhost ~]# echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/root/bin
```
 - 解释：系统在执行一个命令时会在以上目录中查找，比如说 ls。→ 这就是为什么，系统命令不需要路径也可以执行的原因
- PATH 变量叠加 → 将自定义脚本添加到、PATH 路径中
 - PATH="\$PATH":/root/sh

● PS1: 定义系统提示符的变量

- 查看当前的 PS1 – `echo "$PS1"`

```
root@localhost ~]# echo "$PS1"
[\u@\h \w]\$
```

- 举例：

```
[root@localhost ~]# PS1='[\u@\t \w]\$ '
[root@11:50:02 ~]\$ PS1='[\u@\h \#\W]\$ '
[root@localhost 62~]\$ PS1='[\u@\h \w]\$ '
[root@localhost ~]#
```

4. Bash 环境变量的储存位置

- /etc/bashrc ← 修改 PS1
- /root/.**bashrc**
- /home/用户名/.**bashrc**
- /家目录/bash_profile ← ll、cp 等指令的默认选项写在这里

14.3 位置参数变量

1. 位置参数变量

位置参数变量	作用
\$n	n为数字, \$0代表命令本身, \$1~\$9代表第一到第九个参数, 十以上的参数需要用大括号包含, 如\${10}.
\$*	这个变量代表命令行中所有的参数, \$*把所有的参数看成一个整体
\$@	这个变量也代表命令行中所有的参数, 不过\$@把每个参数区分对待
\$#	这个变量代表命令行中所有参数的个数

例一:

```
1.#!/bin/bash
2. num1=$1 ← 用 num1 代表第一个参数
3. num2=$2 ← 用 num2 代表第二个参数
4. sum=$((num1 + num2)) # 变量 sum 的和是 num1 加 num2
5. echo $sum # 打印变量 sum 的值
```

例二:

```
1.#!/bin/bash
2.echo "A total of $# parameters"      # 使用$#代表所有参数的个数
3.echo "The parameters is: $*"          # 使用$*代表所有的参数 但是看作为一个整体
4.echo "The parameters is: $@"          # 使用$@也代表所有参数 将参数区分对待
```

例三: \$*与\$@的区别

```
1.#!/bin/bash
2.for i in "$*"           # $*中的所有参数看成是一个整体, 所以这个 for 循环只会循环一次
3. do
4.   echo "The parameters is: $i"
5. done
6.
7. x=1
8.
9. for y in "$@"
10. do
11.   echo "The parameter $x is: $y"
12.   x=$((x + 1))
13. done
```

14.4 预定义变量

预定义变量	作用
\$?	最后一次执行的命令的返回状态。如果这个变量的值为0，证明上一个命令正确执行；如果这个变量的值为非0（具体是哪个数，由命令自己来决定），则证明上一个命令执行不正确了。
\$\$	当前进程的进程号（PID）
\$!	后台运行的最后一个进程的进程号（PID）

```
1. #!/bin/bash
2.
3. echo "The current process is $$"
4.      #输出当前进程的 PID。
5.      #这个 PID 就是 variable.sh 这个脚本执行时，生成的进程的 PID
6.
7. find /root -name hello.sh & ← 一个&代表将命令放入后台执行
8.      #使用 find 命令在 root 目录下查找 hello.sh 文件
9.      #符号&的意思是把命令放入后台执行，工作管理我们在系统管理章节会详细介绍
10.
11. echo "The last one Daemon process is $!"
```

2. 接收键盘输入

接收键盘输入

read [选项] [变量名]

选项：

- p “提示信息” 在等待 read 输入时，输出提示信息
- t 秒数 read 命令会一直等待用户输入，使用此选项可以指定等待时间
- n 字符数 read 命令只接受指定的字符数，就会执行
- s 隐藏输入的数据，适用于机密信息的输入

```
1. #!/bin/bash
2.
3. read -t 30 -p "Please input your name: " name      #提示“请输入姓名”并等待 30 秒，把用户的输入保存入变量 name 中
4. echo "Name is $name"
5. echo -e "\n"
6.
7. read -s -t 30 -p "Please enter your age: " age      #年龄是隐私，所以我们用“-s”选项隐藏输入
8. echo -e "Age is $age \n"          ← 顺带学习下两种换行打印方式
9.
10. read -n 1 -t 30 -p "Please select your gender[M/F]: " gender      #使用“-n 1”选项只接收一个输入字符就会执行（都
     不用输入回车）
11. echo "Sex is $gender"
```

十五. Bash 运算符

15.1 数值运算与运算符

1. declare 声明变量类型

```
declare 声明变量类型
declare [+/-][选项] 变量名
选项:
-: 给变量设定类型属性
+: 取消变量的类型属性
-i: 将变量声明为整型 (integer)
-x: 将变量声明为环境变量 之前说过的 export 本质上就是添加了 x 选项
-p: 显示指定变量的被声明的类型
```

2. 数值运算

- 方法一

```
[root@localhost ~]# aa=11
[root@localhost ~]# bb=22
[root@localhost ~]# declare -i cc=$aa+$bb
[root@localhost ~]# echo $cc
33
```

- 方法二 – expr 或 let 数值运算工具 【不推荐】

```
[root@localhost ~]# aa=11      给变量aa和bb赋值
[root@localhost ~]# bb=22
[root@localhost ~]# dd=$((expr $aa + $bb))  dd的值是aa和bb的和,
[root@localhost ~]# echo $dd          注意: =~号左右两侧必须有空格
33
```

- 方法三 - \$((运算式)) 或 \$[] 【推荐】

```
[root@localhost ~]# a=1
[root@localhost ~]# b=2
[root@localhost ~]# c=$(( $a + $b ))
[root@localhost ~]# d=$[ $b + $a ]
[root@localhost ~]# echo " $c \n $d"
3 \n 3
```

← 有没有空格都可以

3. 运算符

优先级	运算符	说明
13	-, +	单目负、单目正
12	!, ~	逻辑非、按位取反或补码
11	* , /, %	乘、除、取模
10	+, -	加、减
9	<<, >>	按位左移、按位右移
8	<=, >=, <, >	小于或等于、大于或等于、小于、大于
7	==, !=	等于、不等于
6	&	按位与
5	^	按位异或
4		按位或
3	&&	逻辑与
2		逻辑或
1	=, +=, - =, *=, /=, %=, &=, ^= =, <=, >=	赋值、运算且赋值

aa=\$(((11+3)*3/2))
#虽然乘和除的优先级高于加，但是通过小括号可以调整运算优先级

bb=\$((14%3))
#14不能被3整除，余数是2

cc=\$((1 && 0))
#逻辑与运算只有想与的两边都为真（不为0），与的结果才是1，否则与的结果是0

15.2 变量的测试与内容替换

变量置换方式	变量y没有设置	变量y为空值	变量y设置值
x=\${y-新值}	x=新值	x为空	x=\$y
x=\${y:-新值}	x=新值	x=新值	x=\$y
x=\${y+新值}	x为空	x=新值	x=新值
x=\${y:+新值}	x为空	x为空	x=新值
x=\${y:=新值}	x=新值 y=新值	x为空 y值不变	x=\$y y值不变
x=\${y?:新值}	x=新值 y=新值	x=新值 y=新值	x=\$y y值不变
x=\${y?新值}	新值输出到标准 错误输出（就是 屏幕）	x为空	x=\$y
x=\${y:?新值}	新值输出到标准 错误输出	新值输出到标准 错误输出	x=\$y

● 测试一

```
[root@localhost ~]# unset y      #删除变量y
[root@localhost ~]# x=${y-new}    #进行测试
[root@localhost ~]# echo $x      #因为变量y不存在, 所以x=new
[root@localhost ~]# new
```

● 测试二

```
[root@localhost ~]# y=""        #给变量y赋值为空
[root@localhost ~]# x=${y-new}    #进行测试
[root@localhost ~]# echo $x      #因为y为空, 所以x也为空
```

● 测试三

```
[root@localhost ~]# y=old      #给变量y赋值
[root@localhost ~]# x=${y-new}    #进行测试
[root@localhost ~]# echo $x      #因为y中有值所以x等于y中的值
[root@localhost ~]# old
```

十六. 环境变量配置文件

16.1 环境变量配置文件简介

1. source 命令

- [root@localhost ~]# source 配置文件
- [root@localhost ~]# .**配置文件** ← 让环境配置文件立刻生效，两个的作用是一样的

2. 环境变量配置文件简介

- 环境变量配置文件中主要是定义对系统的操作环境生效的系统默认环境变量，比如 PATH、HISTSIZE、PS1、HOSTNAME 等 默认环境变量
- /etc/profile
- /etc/profile.d/*.sh
- ~/.bash_profile
- ~/.bashrc
- /etc/bashrc
- 以上文件中，/etc/…… 下的对所有用户生效，而~/ 下的仅对某个用户生效，因为此文件存放于此用户的家目录

16.2 环境变量配置文件作用

- /etc/profile
- /etc/profile.d/*.sh
- ~/.bash_profile
- ~/.bashrc
- /etc/bashrc
- 以上文件中，/etc/…… 下的对所有用户生效，而~/ 下的仅对某个用户生效，因为此文件存放于此用户的家目录

/etc/profile的作用:

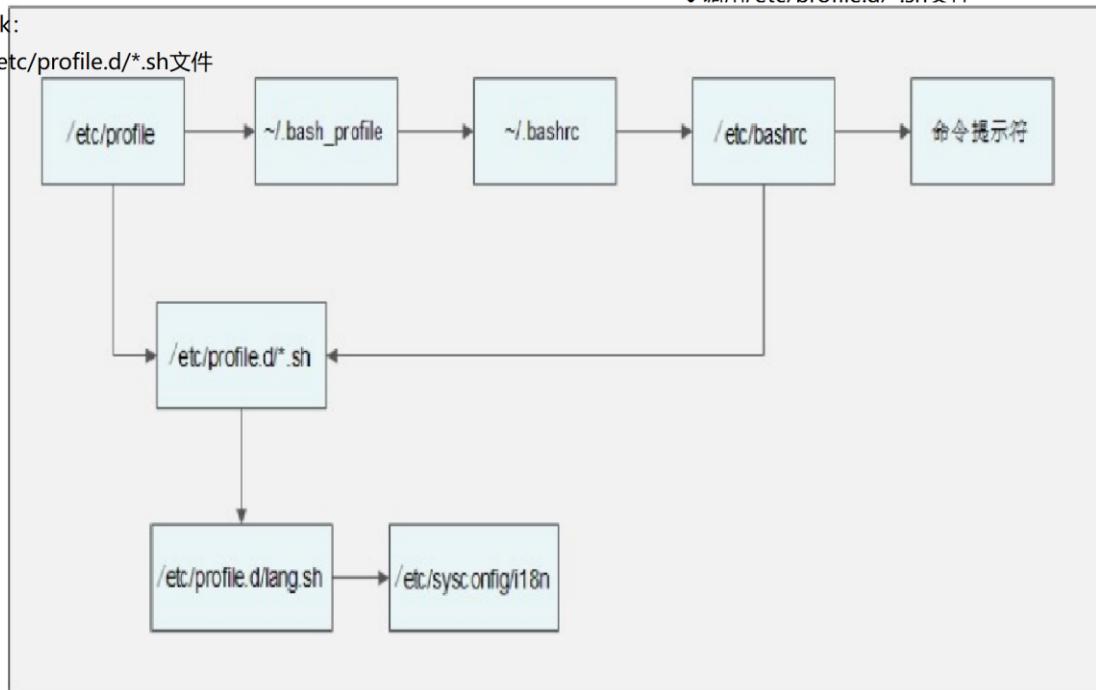
- ◆ USER变量:
- ◆ LOGNAME变量:
- ◆ MAIL变量:
- ◆ PATH变量:
- ◆ HOSTNAME变量:
- ◆ HISTSIZE变量:
- ◆ umask:
- ◆ 调用/etc/profile.d/*.sh文件

~/.bash_profile的作用

- ◆调用了~/.bashrc文件。
- ◆在PATH变量后面加入了“:\$HOME/bin”
- ◆这个目录
- ◆定义默认别名
- ◆调用/etc/bashrc

/etc/bashrc的作用

- ◆PS1变量
- ◆umask
- ◆PATH变量
- ◆调用/etc/profile.d/*.sh文件



16.3 其他配置文件和登录信息

1. 注销时生效的环境变量配置文件

- `~/.bash_logout`

2. 其他配置文件

- `~/bash_history` → 比如在用户退出时将 history 写入相应文件

3. Shell 登录信息

- 本地终端欢迎信息: `/etc/issue`

转义符	作用
<code>\d</code>	显示当前系统日期
<code>\s</code>	显示操作系统名称
<code>\l</code>	显示登录的终端号, 这个比较常用。
<code>\m</code>	显示硬件体系结构, 如i386、i686等
<code>\n</code>	显示主机名
<code>\o</code>	显示域名
<code>\r</code>	显示内核版本
<code>\t</code>	显示当前系统时间
<code>\u</code>	显示当前登录用户的序列号

- 远程终端欢迎信息: `/etc/issue.net`

■ 转义符在`/etc/issue.net`文件中不能使用

■ 是否显示此欢迎信息, 由 ssh 的配置文件 `/etc/ssh/sshd_config` 决定, 加入“Banner /etc/issue.net”行才能显示 (记得重启 SSH 服务)

十七. 正则表达式

1. 正则表达式与通配符

- 正则表达式用来在文件中匹配符合条件的字符串，正则是**包含匹配**。grep、awk、sed 等命令可以支持正则表达式
- 通配符用来匹配符合条件的文件名，通配符是**完全匹配**。ls、find、cp 这些命令不支持正则表达式，所以只能使用 shell 自己的通配符来进行匹配了

2. 正则表达式基础

元字符	作用
*	前一个字符匹配0次或任意多次。
.	匹配除了换行符外任意一个字符。
^	匹配行首。例如：^hello会匹配以hello开头的行。
\$	匹配行尾。例如：hello\$会匹配以hello结尾的行。
[]	匹配中括号中指定的任意一个字符，只匹配一个字符。 例如：[aoeiu] 匹配任意一个元音字母，[0-9] 匹配任意一位数字，[a-z][0-9] 匹配小写字母和一位数字构成的两位字符。
[^]	匹配除中括号的字符以外的任意一个字符。例如：[^0-9] 匹配任意一位非数字字符，[^a-z] 表示任意一位非小写字母。
\	转义符。用于取消讲特殊符号的含义取消。
\{n\}	表示其前面的字符恰好出现n次。例如：[0-9]\{4\} 匹配4位数字，[1][3-8][0-9]\{9\} 匹配手机号码。
\{n, \}	表示其前面的字符出现不小于n次。例如：[0-9]\{2,\} 表示两位及以上的数字。
\{n, m\}	表示其前面的字符至少出现n次，最多出现m次。例如：[a-z]\{6,8\} 匹配6到8位的小写字母。

- “*”前一个字符匹配 0 次，或任意多次
 - grep "a*" test_rule.txt #匹配所有内容，包括空白行
 - grep "aa*" test_rule.txt #匹配至少包含一个 a 的行
 - grep "aaa*" test_rule.txt #匹配最少包含两个连续 a 的字符串
 - grep "aaaaa*" test_rule.txt #则会匹配最少包含四个个连续 a 的字符串
- “.” 匹配除了换行符外任意一个字符
 - grep "s..d" test_rule.txt #“s..d”会匹配在 s 和 d 这两个字母之间一定有两个字符的单词
 - grep "s.*d" test_rule.txt #匹配在 s 和 d 字母之间有任意字符
 - grep ".*" test_rule.txt #匹配所有内容
- “^”匹配行首，“\$”匹配行尾
 - grep "^M" test_rule.txt #匹配以大写“M”开头的行
 - grep "n\$" test_rule.txt #匹配以小写“n”结尾的行
 - grep -n "^\$" test_rule.txt #会匹配空白行
- “[]” 匹配中括号中指定的任意一个字符，只匹配一个字符
 - grep "s[ao]id" test_rule.txt #匹配 s 和 i 字母中，要不是 a、要不是 o
 - grep "[0-9]" test_rule.txt #匹配任意一个数字
 - grep "^*[a-z]" test_rule.txt #匹配用小写字母开头的行
- “[^]” 匹配除中括号的字符以外的任意一个字符
 - grep "^[^a-z]" test_rule.txt #匹配不用小写字母开头的行
 - grep "^[^a-zA-Z]" test_rule.txt #匹配不用字母开头的行
- “\” 转义符
 - grep "\\$" test_rule.txt #匹配使用“.”结尾的行
- “\{n\}”表示其前面的字符恰好出现 n 次和\{n,n\}是一样的
 - grep "a\{3\}" test_rule.txt #匹配 a 字母连续出现三次的字符串
 - grep "[0-9]\{3\}" test_rule.txt #匹配包含连续的三个数字的字符串
- “\{n,\}”表示其前面的字符出现不小于 n 次
 - grep "^[0-9]\{3,\}[a-z]" test_rule.txt #匹配最少用连续三个数字开头的行
- “\{n,m\}”匹配其前面的字符至少出现 n 次，最多出现 m 次
 - grep "sa\{1,3\}i" test_rule.txt #匹配在字母 s 和字母 i 之间有最少一个 a, 最多三个 a

十八. 字符串截取命令

18.1 cut 字段提取命令

cut 截取命令

cut [选项] 文件名

选项:

-f 列号: 提取第几列

-d 分隔符: 按照指定分隔符分割列

cut 是截取列

举例:

```
[root@localhost ~]# vi student.txt
```

ID	Name	gender	Mark
1	Liming	M	86
2	Sc	M	90
3	Gao	M	83

```
[root@localhost ~]# vim student.txt
[root@localhost ~]# cut -f 2 student.txt
Name
Liming
Sc
Gao
[root@localhost ~]# cut -f 2,3 student.txt
Name    gender
Liming   M
Sc      M
Gao     M
[root@localhost ~]# cut -d ":" -f 1,3 /etc/passwd
root:0
bin:1
daemon:2
adm:3
lp:4
sync:5
shutdown:6
```

cut 命令的局限性

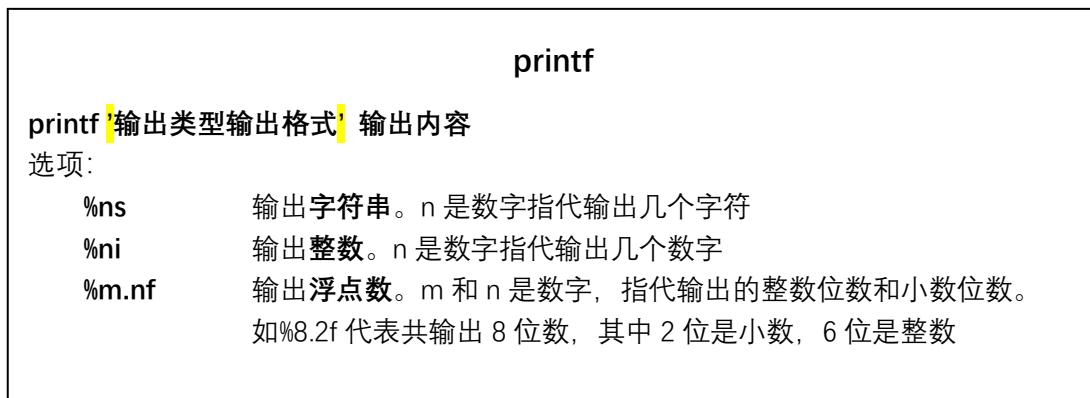
- 如果遇到以空格为分隔的文件或者命令输出，则不能正确的进行分隔
- 比如 df 命令的输出就是用空格分隔的

```
[root@localhost ~]$ df -h
文件系统 容量 已用 可用 已用% 挂载点
/devtmpfs 909M 0 909M 0% /dev
tmpfs 919M 0 919M 0% /dev/shm
tmpfs 919M 9.0M 910M 1% /run
tmpfs 919M 0 919M 0% /sys/fs/cgroup
/dev/sda2 17G 1.9G 14G 12% /
/dev/sda1 190M 117M 60M 67% /boot
/dev/sda3 1.9G 5.8M 1.7G 1% /home
tmpfs 184M 0 184M 0% /run/user/0
[root@localhost ~]$ df -h | cut -d " " -f 2,3
```

● [root@localhost ~]\$ █

18.2 printf 命令

- print 和 printf 的区别
 - 在 awk 命令的输出中支持 print 和 printf 命令
 - **print**: print 会在每个输出之后自动加入一个换行符 (Linux 默认没有 print 命令)
 - **printf**: printf 是标准格式输出命令，并不会自动加入换行符，如果需要换行，需要手工加入换行符
 - 总结：print 会在输出完自动输出回车，而 printf 不会



- 输出格式
 - \a 输出警告声音
 - \b 输出退格键，也就是 Backspace 键
 - \f 清除屏幕
 - \n 换行
 - \r 回车，也就是 Enter 键
 - \t 水平输出退格键，也就是 Tab 键
 - \v 垂直输出退格键，也就是 Tab 键
- 实验一

```
[root@localhost ~]$ printf %s 1 2 3 4 5 6
123456[root@localhost ~]$
[root@localhost ~]$ printf %s %s %s 1 2 3 4 5 6
%$%$123456[root@localhost ~]$
[root@localhost ~]$
[root@localhost ~]$ printf '%s %s %s' 1 2 3 4 5 6
1 2 34 5 6[root@localhost ~]$
[root@localhost ~]$
[root@localhost ~]$ printf '%s %s %s\n' 1 2 3 4 5 6
1 2 3
4 5 6
```

- 实验二
- ```
[root@localhost ~]$ printf '%s' $(cat student.txt)
IDNamegenderMark1LimingM862ScM903GaoM83[root@localhost ~]$
[root@localhost ~]$
[root@localhost ~]$ printf '%s\t %s\t %s\t %s\t %s\t %s\t \n' $(cat student2.txt
)
ID Name PHP Linux MySQL Average
1 Liming 82 95 86 87.66
2 Sc 74 96 87 85.66
3 Gao 9 83 93 91.66
```

## 18.3 awk 命令

### awk

awk [条件 1{动作 1} 条件 2{动作 2}…] 文件名

条件 (Pattern):

一般使用关系表达式作为条件

x > 10 判断变量 x 是否大于 10

x >= 10 大于等于

x <= 10 小于等于

动作 (Action):

格式化输出

流程控制语句

- 可以正确输出以制表符分隔的内容

```
[root@localhost ~]$ awk '{printf $2 "\t" $6 "\n"}' student2.txt
Name Average
Liming 87.66
Sc 85.66
Gao 91.66
```

- 可以正确输出以空格作为分隔的内容

```
[root@localhost ~]$ df -h | awk '{print $1 "\t" $3}'
文件系统 已用
devtmpfs 0
tmpfs 0
tmpfs 9.0M
tmpfs 0
/dev/sda2 1.9G
/dev/sda1 117M
/dev/sda3 5.8M
tmpfs 0
```

- awk 默认第一行是不会按照格式输出的可以添加 BEGIN 进行强行识别

■ BEGIN 表示内容的第一行

```
[root@localhost ~]$ awk 'BEGIN{printf "This is a transcript \n" }
> {printf $2 "\t" $6 "\n"}' student2.txt
This is a transcript
Name Average
Liming 87.66
Sc 85.66
Gao 91.66
```

■ END 表示内容尾

```
[root@localhost ~]$ awk 'END{printf "The End \n" }
> {printf $2 "\t" $6 "\n"}' student2.txt
Name Average
Liming 87.66
Sc 85.66
Gao 91.66
The End
```

- FS 内置变量 → 指定分隔符

```
[root@localhost ~]$ cat /etc/passwd | grep "/bin/bash" | \
> awk 'BEGIN {FS=":"} {printf $1 "\t" $3 "\n"}'
root 0
fox 1000
user1 1002
```

- 关系运算符

```
[root@localhost ~]$ cat student2.txt | grep -v Name | awk '$6 >= 87 {printf $2 " \
\n" }'
Liming
Gao
```

注意：awk 是一个非常强大的指令，这里只是简单介绍

## 18.4 sed 命令

- sed 是一种几乎包括在所有 UNIX 平台（包括 Linux）的轻量级流编辑器。sed 主要是用来将数据进行选取、替换、删除、新增的命令
- sed 可以在不改变文件内容的情况下进行按照条件输出

### sed

sed [选项] '[动作]' 文件名

#### 选项:

- n 一般 sed 命令会把所有数据都输出到屏幕，如果加入此选择，则只会把经过 sed 命令处理的行输出到屏幕
- e 允许对输入数据应用多条 sed 命令编辑，命令之间使用“'XXXX; XXXX'”分隔
- i 用 sed 的修改结果直接修改读取数据的文件，而不是由屏幕输出

#### 动作:

- a \ 追加，在当前行后添加一行或多行。添加多行时，除最后一行外，每行末尾需要用\"代表数据未完结
- c \ 行替换，用 c 后面的字符串替换原数据行，替换多行时，除最后一行外，每行末尾需用\"代表数据未完结
- i \ 插入，在当期行前插入一行或多行。插入多行时，除最后一行外，每行末尾需要用\"代表数据未完结
- d 删 除，删除指定的行
- p 打印，输出指定的行
- s 字串替换，用一个字符串替换另外一个字符串。

格式为“行范 围 s/旧字串/新字串/g”（和 vim 中的替换格式类似）

- 行数据操作

- 查看文件的第二行

```
[root@localhost ~]$ sed '2p' student2.txt
ID Name PHP Linux MySQL Average
1 Liming 82 95 86 87.66
1 Liming 82 95 86 87.66
2 Sc 74 96 87 85.66
3 Gao 9 83 93 91.66

[root@localhost ~]$ sed -n '2p' student2.txt
1 Liming 82 95 86 87.66
```

- 删除第二行到第四行的数据，但不修改文件本身

```
[root@localhost ~]$ sed '2,4d' student2.txt
ID Name PHP Linux MySQL Average
```

- 在第二行后追加 hello

```
[root@localhost ~]$ sed '2a HELLO' student2.txt
ID Name PHP Linux MySQL Average
1 Liming 82 95 86 87.66
HELLO
2 Sc 74 96 87 85.66
3 Gao 9 83 93 91.66
```

- 在第二行前插入两行数据

```
[root@localhost ~]$ sed '2i hello world' student2.txt
ID Name PHP Linux MySQL Average
hello world
1 Liming 82 95 86 87.66
2 Sc 74 96 87 85.66
3 Gao 9 83 93 91.66
```

- 数据替换

```
[root@localhost ~]$ sed '2c No such person' student2.txt
ID Name PHP Linux MySQL Average
No such person
2 Sc 74 96 87 85.66
3 Gao 9 83 93 91.66
```

- 字符串替换: `sed 's/旧字符串/新字符串/g' 文件名`

- 在第四行中, 把 9 换成 74

```
[root@localhost ~]$ sed '4s/9/74/g' student2.txt
ID Name PHP Linux MySQL Average
1 Liming 82 95 86 87.66
2 Sc 99 96 87 85.66
3 Gao 74 83 743 741.66
```

- sed 操作的数据直接写入文件

```
[root@localhost ~]$ cat student2.txt
ID Name PHP Linux MySQL Average
1 Liming 82 95 86 87.66
2 Sc 74 96 87 85.66
3 Gao 9 83 93 91.66
[root@localhost ~]$ sed -i 's/74/99/g' student2.txt
[root@localhost ~]$ cat student2.txt
ID Name PHP Linux MySQL Average
1 Liming 82 95 86 87.66
2 Sc 99 96 87 85.66
3 Gao 9 83 93 91.66
```

- 同时把“Liming”和“Gao”替换为空

```
[root@localhost ~]$ sed -e 's/Liming//g;s/Gao//g' student2.txt
ID Name PHP Linux MySQL Average
1 [REDACTED] 82 95 86 87.66
2 Sc 99 96 87 85.66
3 [REDACTED] 9 83 93 91.66
```

# 十九. 字符处理命令

## 排序命令 sort

sort [选项] 文件名

选项:

-f 忽略大小写

-n 以数值型进行排序, 默认使用字符串型排序

-r 反向排序

-t 指定分隔符, 默认是分隔符是制表符

-k n[,m] 按照指定的字段范围排序。从第 n 字段开始, m 字段结束 (默认到行尾)

- 排序用户信息文件

```
[root@localhost ~]$ sort /etc/passwd
abrt:x:173:173::/etc/abrt:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
apache:x:48:48:Apache:/usr/share/httpd:/sbin/nologin
bin:x:1:1:bin:/bin:/sbin/nologin
chrony:x:997:995::/var/lib/chrony:/sbin/nologin 以字典序输出了
daemon:x:2:2:daemon:/sbin:/sbin/nologin
dbus:x:81:81:System message bus:/sbin/nologin
fox:x:1000:1000:This user Silverfox:/home/fox:/bin/bash
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
```

- 反向排序

```
[root@localhost ~]$ sort -r /etc/passwd
user1:x:1002:1002::/home/user1:/bin/bash
tcpdump:x:72:72::/sbin/nologin
systemd-network:x:192:192:systemd Network Management:/sbin/nologin
sync:x:5:0:sync:/sbin/bin/sync
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
shutdown:x:6:0:shutdown:/sbin/sbin/shutdown
rpc:x:32:32:Rpcbind Daemon:/var/lib/rpcbind:/sbin/nologin
root:x:0:0:root:/root:/bin/bash
postfix:x:89:89::/var/spool/postfix:/sbin/nologin 逆序排序了
polkitd:x:999:998:User for polkitd:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
```

- 指定分隔符是“：“，用第三字段开头，第三字段结尾排序，就是只用第三字段排序

```
[root@localhost ~]$ sort -t ":" -k 3,3 /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
fox:x:1000:1000:This user Silverfox:/home/fox:/bin/bash
user1:x:1002:1002::/home/user1:/bin/bash
operator:x:11:0:operator:/root:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin 默认以字符串型数据排序,
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin 所以会出现这种状况
abrt:x:173:173:/etc/abrt:/sbin/nologin 如果要指定为数值型, 请添加-n
systemd-network:x:192:192:systemd Network Management:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
rpc:x:32:32:Rpcbind Daemon:/var/lib/rpcbind:/sbin/nologin
```

```
[root@localhost ~]$ sort -n -t ":" -k 3,3 /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
```

## 2. 统计命令 wc

### 统计命令

**wc [选项] 文件名**

选项:

-l: 只统计行数

-w: 只统计单词数

-m: 只统计字符数

## 二十. 条件判断

[ -e /root/install.log ] ← 条件判断建议用中括号!! 并且前后要加空格!!

### 1. 按照文件类型进行判断

| 测试选项  | 作用                               |
|-------|----------------------------------|
| -b 文件 | 判断该文件是否存在，并且是否为块设备文件（是块设备文件为真）   |
| -c 文件 | 判断该文件是否存在，并且是否为字符设备文件（是字符设备文件为真） |
| -d 文件 | 判断该文件是否存在，并且是否为目录文件（是目录为真）       |
| -e 文件 | 判断该文件是否存在（存在为真）                  |
| -f 文件 | 判断该文件是否存在，并且是否为普通文件（是普通文件为真）     |
| -L 文件 | 判断该文件是否存在，并且是否为符号链接文件（是符号链接文件为真） |
| -p 文件 | 判断该文件是否存在，并且是否为管道文件（是管道文件为真）     |
| -s 文件 | 判断该文件是否存在，并且是否为非空（非空为真）          |
| -S 文件 | 判断该文件是否存在，并且是否为套接字文件（是套接字文件为真）   |

两种判断格式：

- test -e /root/install.log
- [ -e /root/install.log ] ← 注意两侧的空格，如果没有，系统会无法识别命令!!!
- [ -d /root ] && echo "yes" || echo "no"
  - 第一个判断命令如果正确执行，则打印“yes”，否则打印“no”

### 2. 按照文件权限进行判断

| 测试选项  | 作用                                   |
|-------|--------------------------------------|
| -r 文件 | 判断该文件是否存在，并且是否该文件拥有读权限（有读权限为真）       |
| -w 文件 | 判断该文件是否存在，并且是否该文件拥有写权限（有写权限为真）       |
| -x 文件 | 判断该文件是否存在，并且是否该文件拥有执行权限（有执行权限为真）     |
| -u 文件 | 判断该文件是否存在，并且是否该文件拥有SUID权限（有SUID权限为真） |
| -g 文件 | 判断该文件是否存在，并且是否该文件拥有SGID权限（有SGID权限为真） |
| -k 文件 | 判断该文件是否存在，并且是否该文件拥有SBit权限（有SBit权限为真） |

- 注意：这样判断的文件是否有某权限，只要有一个用户有即为真！
  - 也就是说 -rw-r—r-- 为真，但是普通用户并没有写权限！

### 3. 两个文件之间进行比较

| 测试选项        | 作用                                                       |
|-------------|----------------------------------------------------------|
| 文件1 -nt 文件2 | 判断文件1的修改时间是否比文件2的新（如果新则为真）                               |
| 文件1 -ot 文件2 | 判断文件1的修改时间是否比文件2的旧（如果旧则为真）                               |
| 文件1 -ef 文件2 | 判断文件1是否和文件2的Inode号一致，可以理解为两个文件是否为同一个文件。这个判断用于判断硬链接是很好的方法 |

- ln /root/student.txt /tmp/stu.txt #创建个硬链接吧
- [ /root/student.txt -ef /tmp/stu.txt ] && echo "yes" || echo "no" yes #用 test 测试下，果然很有用

### 4. 两个整数之间进行比较

| 测试选项        | 作用                     |
|-------------|------------------------|
| 整数1 -eq 整数2 | 判断整数1是否和整数2相等（相等为真）    |
| 整数1 -ne 整数2 | 判断整数1是否和整数2不相等（不相等位置）  |
| 整数1 -gt 整数2 | 判断整数1是否大于整数2（大于为真）     |
| 整数1 -lt 整数2 | 判断整数1是否小于整数2（小于位置）     |
| 整数1 -ge 整数2 | 判断整数1是否大于等于整数2（大于等于为真） |
| 整数1 -le 整数2 | 判断整数1是否小于等于整数2（小于等于为真） |

- [ 23 -ge 22 ] && echo "yes" || echo "no"
  - yes #判断 23 是否大于等于 22，当然是了
- [ 23 -le 22 ] && echo "yes" || echo "no"
  - no #判断 23 是否小于等于 22，当然不是了

## 5. 字符串的判断

| 测试选项         | 作用                       |
|--------------|--------------------------|
| -z 字符串       | 判断字符串是否为空（为空返回真）         |
| -n 字符串       | 判断字符串是否为非空（非空返回真）        |
| 字符串1 == 字符串2 | 判断字符串1是否和字符串2相等（相等返回真）   |
| 字符串1 != 字符串2 | 判断字符串1是否和字符串2不相等（不相等返回真） |

- name=sc #给 name 变量赋值  

```
[-z "$name"] && echo "yes" || echo "no"
```

no #判断 name 变量是否为空，因为不为空，所以返回 no
- aa=11  
bb=22 #给变量 aa 和变量 bb 赋值  

```
["$aa" == "bb"] && echo "yes" || echo "no"
```

no #判断两个变量的值是否相等，明显不相等，所以返回 no

## 6. 多重条件判断

| 测试选项       | 作用                        |
|------------|---------------------------|
| 判断1 -a 判断2 | 逻辑与，判断1和判断2都成立，最终的结果才为真   |
| 判断1 -o 判断2 | 逻辑或，判断1和判断2有一个成立，最终的结果就为真 |
| ! 判断       | 逻辑非，使原始的判断式取反             |

- aa=11  

```
[-n "$aa" -a "$aa" -gt 23] && echo "yes" || echo "no"
```

no #判断变量 aa 是否有值，同时判断变量 aa 的是否大于 23  
#因为变量 aa 的值不大于 23，所以即使第一个判断值为真，返回的结果也是假
- aa=24  

```
[-n "$aa" -a "$aa" -gt 23] && echo "yes" || echo "no"
```

yes

# 二十一. 流程控制

## 21.1 if 语句

### 1. 单分支 if 条件语句

```
1. if [条件判断式];then ← [] 两侧要有空格
2. 程序
3. fi
```

或者

```
1. if [条件判断式] ← [] 两侧要有空格
2. then
3. 程序
4. fi
```

单分支条件语句需要注意的几个点

- if 语句使用 fi 结尾，和一般语言使用大括号结尾不同
- [ 条件判断式 ] 就是使用 test 命令判断，所以中括号和条件判断式之间必须有空格
- then 后面跟符合条件之后执行的程序，可以放在[]之后，用“;”分割。也可以换行写入，就不需要“;”了

例子：

```
1.#!/bin/bash
2. # 统计根分区使用率
3. # 2020 年 8 月 3 日 10:58:44
4.
5.
6. # 把根分区使用率赋给 rate ↓截取第五列 ↓以百分号为分割，截取第一列
7. rate=$(df -h | grep "/dev/sda2" | awk '{print $5}' | cut -d "%" -f 1)
8.
9. if [$rate -ge 10]
10. then
11. echo "-----"
12. echo "Warning! The rate of /dev/sda2 is $rate"
13. echo "$(date)"
14. echo "-----"
15. fi
```

## 2. 双分支 if 条件语句

```
1. if [条件判断式]
2. then
3. 条件成立时, 执行的程序
4. else
5. 条件不成立时, 执行的另一个程序
6. fi
```

例子一:

备份 MySQL 数据库

```
1.#!/bin/bash
2. # 备份 MySQL 数据库
3. # 2020 年 8 月 3 日 11:29:26
4.
5. # 同步系统时间
6. ntpdate asia.pool.ntp.org $> /dev/null # 类似于回收站
7.
8. # 将当前系统时间赋给 date
9. date=$(date +%y%m%d)
10.
11. # 统计 MySQL 数据库大小, 并将结果赋给 size
12. size=$(du -sh /var/lib/mysql)
13.
14. if [-d /tmp/dbbak]
15. then
16. echo "Data: $date" > /tmp/dbbak/dbinfo.txt
17. echo "Size: $size" >> /tmp/dbbak/dbinfo.txt
18. cd /tmp/dbbak
19. tar -zcf mysql-lib-$date.tar.gz /var/lib/mysql dbinfo.txt $> /dev/null
20. rm -rf /tmp/dbbak/dbinfo.txt
21.
22. else
23. mkdir /tmp/dbbak
24. echo "Data: $date" > /tmp/dbbak/dbinfo.txt
25. echo "Size: $size" >> /tmp/dbbak/dbinfo.txt
26. cd /tmp/dbbak
27. tar -zcf mysql-lib-$date.tar.gz /var/lib/mysql dbinfo.txt $> /dev/null
28. rm -rf /tmp/dbbak/dbinfo.txt
29. fi
```

## 例二：

### 判断 Apache 是否启动

```
1. #!/bin/bash
2. # 判断 Apache 是否启动
3. # 2020 年 8 月 3 日 11:45:14
4.
5. # 使用 nmap 命令扫描服务器，并截取 Apache 服务状态，赋予变量 port
6. # 如果无法找到命令，请安装 nmap 命令
7. port=$(nmap -sT 192.168.0.6 | grep tcp | grep http | awk '{print $2}')
8.
9. if ["$port" == "open"]
10. then
11. echo "$(date) Apache-httd is running" >> /tmp/autostart-acc.log
12.
13. else
14. # 适用于 rpm 包安装的阿帕奇
15. #/etc/rc.d/init.d/httpd start $> /dev/null
16.
17. # 适用于源码包安装的阿帕奇
18. /usr/local/apache/bin/apachectl start &> /dev/null
19.
20. echo "$(date) restart Apache-httd" >> /tmp/autostart-err.log
21. fi
```

### 3. 多分支 if 条件语句

```
1. if [条件判断式 1]
2. then
3. 当条件判断式 1 成立时，执行程序 1
4.
5. elif [条件判断式 2]
6. then
7. 当条件判断式 2 成立时，执行程序 2
8. ...省略更多条件...
9.
10. else
11. 当所有条件都不成立时，最后执行此程序
12. fi
```

#### 判断用户输入的是什么文件

```
1. #!/bin/bash
2. # 判断用户输入的是什么文件
3.
4. # 键盘接受输入，赋予变量 file_name
5. read -t 30 -p "Please input the file name: " file_name
6.
7. # 判断 file_name 是否为空
8. if [-z "$file_name"]
9. then
10. echo "Error! The file name can not be empty"
11.
12. # 判断 file 的值是否存在
13. elif [! -e "$file_name"]
14. then
15. echo "Not a file"
16. exit 2
17.
18. # 判断是否为普通文件
19. elif [-f "$file_name"]
20. then
21. echo "$file_name is a regular file"
22. exit 3
23.
24. elif [-d "$file_name"]
25. then
26. echo "$file_name is a directory"
27. exit 4
28.
29. else
30. echo "$file_name is other file"
31.
32. fi
```

## 21.2 case 语句

1. case 语句和 if…elif…else 语句一样都是多分支条件语句，不过和 if 多分支条件语句不同的是，case 语句只能判断一种条件关系，而 if 语句可以判断多种条件关系

```
1. case $变量名 in
2. "值 1")
3. 如果变量的值等于值 1，则执行程序 1
4. ;;
5.
6. "值 2")
7. 如果变量的值等于值 2，则执行程序 2
8. ;;
9. ...省略其他分支...
10.
11. *)
12. 如果变量的值都不是以上的值，则执行此程序
13. ;;
14. esac
```

## 判断用户输入

```
1. #!/bin/bash
2. # 判断用户输入的是什么文件
3. # 2020 年 8 月 3 日 13:36:09
4.
5. # 键盘接受输入，赋予变量 file_name
6. read -t 30 -p "Please input the file name: " file_name
7.
8. # 判断 file_name 是否为空
9. if [-z "$file_name"]
10. then
11. echo "Error! The file name can not be empty"
12.
13. # 判断 file 的值是否存在
14. elif [! -e "$file_name"]
15. then
16. echo "Not a file"
17. exit 2
18.
19. # 判断是否为普通文件
20. elif [-f "$file_name"]
21. then
22. echo "$file_name is a regular file"
23. exit 3
24.
25. elif [-d "$file_name"]
26. then
27. echo "$file_name is a directory"
28. exit 4
29.
30. else
31. echo "$file_name is other file"
32.
33. fi
```

## 21.3 while 循环与 until 循环

1. while 循环是不定循环，也称作条件循环。只要条件判断式成立，循环就会一直继续，直到条件判断式不成立，循环才会停止。这就和 for 的固定循环不太一样了

```
1. while [条件判断式]
2. do
3. 程序
4. done
```

小程序：

```
1.#!/bin/bash
2. #从 1 加到 100
3.
4. i=1
5. s=0
6. while [$i -le 100]
7. #如果变量 i 的值小于等于 100，则执行循环
8. do
9. s=$(($s+$i))
10. i=$(($i+1))
11. done
12. echo "The sum is: $s"
```

- until 循环，和 while 循环相反，until 循环时只要条件判断式不成立则进行循环，并执行循环程序。一旦循环条件成立，则终止循环

```
1. until [条件判断式]
2. do
3. 程序
4. done
```

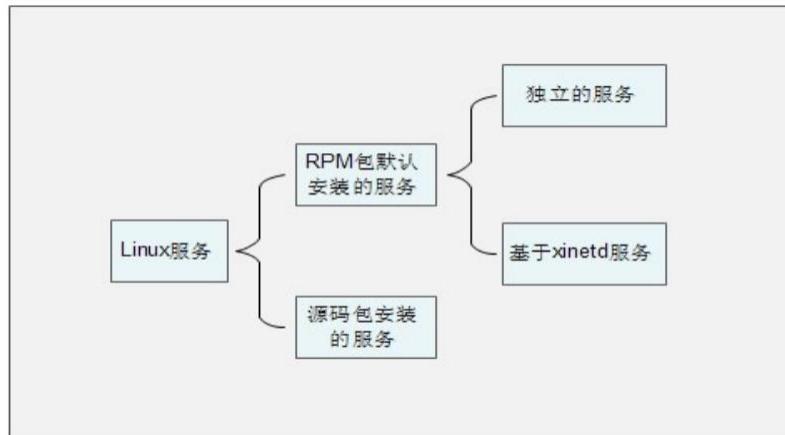
### 小程序：

```
1.#!/bin/bash
2. #从 1 加到 100
3.
4. i=1
5. s=0
6. until [$i -gt 100]
7. #循环直到变量 i 的值大于 100，就停止循环
8. do
9. s=$(($s+$i))
10. i=$(($i+1))
11. done
12. echo "The sum is: $s"
```

# 二十二. Linux 服务管理

## 22.1 服务简介与分类

### 1. 服务的分类



- 目前，因为安全性的问题，基于 xinetd 的服务已经越来越少，并且不建议使用
- 启动与自启动
  - 服务启动：就是在当前系统中让服务运行，并提供功能
  - 服务自启动：自启动是指让服务在系统开机或重启之后，随着系统的启动而自动启动服务
- 查询已安装的服务
  - PRM 包安装的服务
    - ◆ `chkconfig --list` #查看服务自启动状态，可以看到所有 RPM 包安装的服务

```
[root@localhost ~]$ chkconfig --list
注: 该输出结果只显示 SysV 服务, 并不包含
原生 systemd 服务。SysV 配置数据
可能被原生 systemd 配置覆盖。
要列出 systemd 服务, 请执行 'systemctl list-unit-files'。
查看在具体 target 启用的服务请执行
'systemctl list-dependencies [target]'。
netconsole 0:关 1:关 2:关 3:关 4:关 5:关 6:关
network 0:关 1:关 2:开 3:开 4:开 5:开 6:关
◆ 您在 /var/spool/mail/root 中有新邮件
◆ 注意: 在 CentOS 7 中, 要查看服务, 请使用 systemctl list-unit-files
```
    - 源码包安装的服务
      - ◆ 查看服务安装位置，一般是 /usr/local/ 下
  - PRM 包和源码包安装服务的区别
    - RPM 安装服务和源码包安装服务的区别就是安装位置的不同
      - ◆ 源码包安装在指定位置，一般是 /usr/local/
      - ◆ RPM 包安装在默认位置中

## 22.2 RPM 包安装服务的管理

### 22.2.1 独立服务的管理

#### 1. RPM 包安装服务的位置

- /etc/init.d/: 启动脚本位置
- /etc/sysconfig/: 初始化环境配置文件位置
- /etc/: 配置文件位置
- /etc/xinetd.conf: xinetd 配置文件
- /etc/xinetd.d/: 基于 xinetd 服务的启动脚本
- /var/lib/: 服务产生的数据放在这里
- /var/log/: 日志

#### 2. 独立服务的启动

- /etc/init.d/ 独立服务名 start|stop|status|restart|

```
[root@localhost ~]$ ll /etc/init.d/
总用量 40
-rw-r--r--. 1 root root 18281 8月 19 2019 functions
-rwxr-xr-x. 1 root root 4569 8月 19 2019 netconsole
-rwxr-xr-x. 1 root root 7928 8月 19 2019 network
-rw-r--r--. 1 root root 1160 4月 1 04:30 README
```

- service 独立服务名 start|stop|restart||status ← 系统重启后会失效，永久生效请写入配置文件

#### 3. 独立服务的自启动 (CentOS 6 中)

- chkconfig [-l 运行级别] [独立服务名] [on|off]
- 修改/etc/rc.d/rc.local 文件 【推荐】
- 使用 ntsysv 命令管理自启动

## 22.2.2 基于 xinetd 的服务 【不推荐】

注意：本节说明的服务适用于 CentOS 6，后续发行版本会有差异

### 1. 安装 xinetd 与 telnet

- yum -y install xinetd
- yum -y install telnet-server ← 因为潜在的安全隐患，系统默认并不会自带

### 2. xinetd 服务的启动

```
[root@localhost ~]# vi /etc/xinetd.d/telnet
service telnet ↓服务的名称为 telnet
{
 flags = REUSE ← 标志为 REUSE, 设定 TCP/IP socket 可重用
 socket_type = stream ← 使用 TCP 协议数据包
 wait = no ← 允许多个连接同时连接
 user = root ← 启动服务的用户为
 root server = /usr/sbin/in.telnetd ← 服务的启动程序
 log_on_failure += USERID ← 登陆失败后, 记录用户的
 ID disable = no ← 服务不启动
}
```

### 3. 重启 xinetd 服务

```
service xinetd restart
```

### 4. xinetd 服务的自启动

```
chkconfig telnet on
ntsysv
```

## 22.3 源码包安装服务的管理

### 1. 源码包安装服务的启动

- 使用绝对路径，调用启动脚本来启动。不同的源码包的启动脚本不同。可以查看源码包的安装说明，查看启动脚本的方法
  - 比如 Apache: /usr/local/apache/bin/apachectl start|stop

### 2. 源码包服务的自启动

```
[root@localhost ~]$ vim /etc/rc.local

#!/bin/bash
THIS FILE IS ADDED FOR COMPATIBILITY PURPOSES

It is highly advisable to create own systemd services or udev rules
to run scripts during boot instead of using this file.

In contrast to previous versions due to parallel execution during boot
this script will NOT be run after all other services.

Please note that you must run 'chmod +x /etc/rc.d/rc.local' to ensure
that this script will be executed during boot.

touch /var/lock/subsys/local

/usr/local/apache/bin/apachectl start 加入这一行
~
~
~
```

### 3. 让源码包服务被服务管理命令识别 【对新手不推荐】

- 让源码包的 apache 服务能被 service 命令管理启动

In -s /usr/local/apache/bin/apachectl /etc/init.d/apache ← 本质上是一个软连接

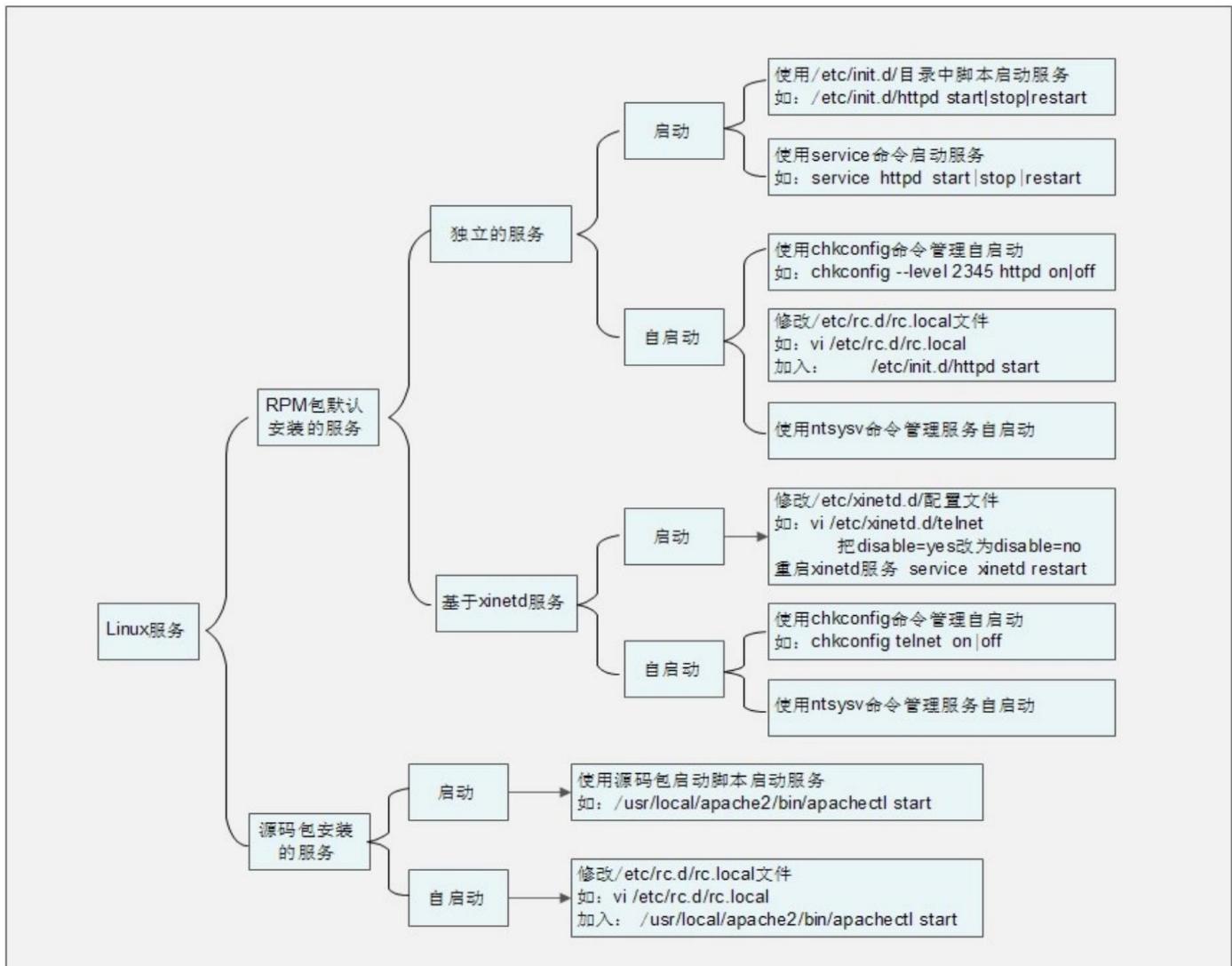
- 让源码包的 apache 服务能被 chkconfig 与 ntsysv 命令管理自启动

- 1. 修改配置文件

```
1. vi /etc/init.d/apache
2. # chkconfig: 35 86 76 ←这些运行级别不与系统现有服务冲突，这个注释不能省，而且必须是这个格式！！
3. 在 /etc/rv?.rc 里保存了系统的运行级别
4. [root@localhost ~]$ ll /etc/rc
 rc0.d/ rc1.d/ rc2.d/ rc3.d/ rc4.d/ rc5.d/ rc6.d/ rc.d/ rc.local
5. #指定 httpd 脚本可以被 chkconfig 命令管理。格式是: chkconfig: 运行级别 启动顺序 关闭顺序
6. # description: source package apache
7. #说明, 内容随意
```

- 2. 使用 chkconfig –add apache 将 Apache 加入到系统自动管理当中, --del 删除

## 22.4 服务管理总结



# 二十三. Linux 系统管理

## 23.1 进程管理

### 23.1.1 进程查看

#### 1. 进程简介

- 进程是正在执行的一个程序或命令，每一个进程都是一个运行的实体，都有自己的地址空间，并占用一定的系统资源

#### 2. 进程管理的作用

- 判断服务器健康状态
- 查看系统中所有进程
- 杀死进程

#### 3. 查看系统中的所有进程

- `ps aux` # 查看系统中所有进程，使用 BSD 操作系统格式

```
[root@localhost ~]$ ps aux
USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND
```

- `ps -le` # 查看系统中所有的进程，使用 Linux 标准命令格式

```
[root@localhost ~]$ ps -le
F S UID PID PPID C PRI NI ADDR SZ WCHAN TTY TIME CMD
```

- ✓ USER 该进程是由哪个用户产生的
- ✓ PID 进程的 ID 号
- ✓ %CPU 该进程占用 CPU 资源的百分比，占用越高，进程越耗费资源
- ✓ %MEM 该进程占用物理内存的百分比，占用越高，进程越耗费资源
- ✓ VSZ 该进程占用虚拟内存的大小，单位 KB
- ✓ RSS 该进程占用实际物理内存的大小，单位 KB
- ✓ TTY 该进程是在哪个终端中运行的。其中 `tty1-tty7` 代表本地控制台终端，`tty1-tty6` 是本地的字符界面终端，`tty7` 是图形终端。`pts/0-255` 代表虚拟终端
- ✓ STAT 进程状态。常见的状态有：
  - ◆ R：运行
  - ◆ S：睡眠
  - ◆ T：停止状态
  - ◆ s：包含子进程
  - ◆ +：位于后台
- ✓ START 该进程的启动时间
- ✓ TIME 该进程占用 CPU 的运算时间，注意不是系统时间
- ✓ COMMAND 产生此进程的命令名

#### 4. 查看系统健康状态

##### 查看系统健康状态

###### top [选项]

选项:

-d 秒数: 指定 top 命令每隔几秒更新。默认是 3 秒在 top 命令的交互模式当中可以执行的命令:

- ? 或 h: 显示交互模式的帮助
- P: 以 CPU 使用率排序, 默认就是此项
- M: 以内存的使用率排序
- N: 以 PID 排序
- q: 退出 top

重点关注的几个地方:

```
top - 12:15:09 up 1:50, 1 user, load average: 0.00, 0.01, 0.05
Tasks: 141 total, 2 running, 139 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.0 us, 0.0 sy, 0.0 ni, 100.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem : 1881348 total, 1517372 free, 164064 used, 199912 buff/cache
KiB Swap: 1047548 total, 1047548 free, 0 used. 1557124 avail Mem

 PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
 9 root 20 0 0 0 0 S 0.3 0.0 0:01.64 rcu_sched
 673 root 20 0 305500 6548 5152 S 0.3 0.3 0:10.92 vmtoolsd
 1 root 20 0 193828 6992 4164 S 0.0 0.4 0:04.43 systemd
 2 root 20 0 0 0 0 S 0.0 0.0 0:00.04 kthreadd
 4 root 0 -20 0 0 0 S 0.0 0.0 0:00.00 kworker/0:0H
 6 root 20 0 0 0 0 S 0.0 0.0 0:00.16 ksoftirqd/0
```

## 第一行信息为任务队列信息

| 内容                             | 说明                                                       |
|--------------------------------|----------------------------------------------------------|
| 12:26:46                       | 系统当前时间                                                   |
| up 1 day, 13:32                | 系统的运行时间, 本机已经运行1天13小时32分钟                                |
| 2 users                        | 当前登录了两个用户                                                |
| load average: 0.00, 0.00, 0.00 | 系统在之前1分钟, 5分钟, 15分钟的平均负载。一般认为小于1时, 负载较小。如果大于1, 系统已经超出负荷。 |

## 第二行为进程信息

| 内容              | 说明                     |
|-----------------|------------------------|
| Tasks: 95 total | 系统中的进程总数               |
| 1 running       | 正在运行的进程数               |
| 94 sleeping     | 睡眠的进程                  |
| 0 stopped       | 正在停止的进程                |
| 0 zombie        | 僵尸进程。如果不是0, 需要手工检查僵尸进程 |

## 第三行为 CPU 信息

| 内容             | 说明                                                    |
|----------------|-------------------------------------------------------|
| Cpu(s): 0.1%us | 用户模式占用的CPU百分比                                         |
| 0.1%sy         | 系统模式占用的CPU百分比                                         |
| 0.0%ni         | 改变过优先级的用户进程占用的CPU百分比                                  |
| 99.7%id        | 空闲CPU的CPU百分比                                          |
| 0.1%wa         | 等待输入/输出的进程的占用CPU百分比                                   |
| 0.0%hi         | 硬中断请求服务占用的CPU百分比                                      |
| 0.1%si         | 软中断请求服务占用的CPU百分比                                      |
| 0.0%st         | st (Steal time) 虚拟时间百分比。就是当有虚拟机时, 虚拟CPU等待实际CPU的时间百分比。 |

## 第四行为物理内存信息

| 内容                 | 说明                                                  |
|--------------------|-----------------------------------------------------|
| Mem: 625344k total | 物理内存的总量, 单位KB                                       |
| 571504k used       | 已经使用的物理内存数量                                         |
| 53840k free        | 空闲的物理内存数量, 我们使用的是虚拟机, 总共只分配了628MB内存, 所以只有53MB的空闲内存了 |
| 65800k buffers     | 作为缓冲的内存数量                                           |

## 第五行为交换分区 (swap) 信息

| 内容                  | 说明             |
|---------------------|----------------|
| Swap: 524280k total | 交换分区(虚拟内存)的总大小 |
| 0k used             | 已经使用的交互分区的大小   |
| 524280k free        | 空闲交换分区的大小      |
| 409280k cached      | 作为缓存的交互分区的大小   |

## 5. 查看进程树

| 查看进程树                                                      |
|------------------------------------------------------------|
| <b>pstree [选项]</b>                                         |
| 选项:                                                        |
| -p: 显示进程的 PID                                              |
| -u: 显示进程的所属用户                                              |
| redhat7 默认没有预装, 请安装 tree 和 psmisc 命令<br>单使用 tree 可以查看目录树结构 |

```
[root@localhost ~]$ pstree
systemd--NetworkManager--dhclient
 | 2*[{NetworkManager}]
 |
 -VGAAuthService
 -abrt-watch-log
 -abrt
 -agetty
 -atd
 -auditd--{auditd}
 -chronyd
 -crond
 -dbus-daemon--{dbus-daemon}
 -firewalld--{firewalld}
 -irqbalance
 -lsmrd
 -lvmtd
 -master--pickup
 qmgr
```

## 23.1.2 终止进程

- `kill -l` # 查看可用进程号

| 信号代号 | 信号名称    | 说明                                                                                                      |
|------|---------|---------------------------------------------------------------------------------------------------------|
| 1    | SIGHUP  | 该信号让进程立即关闭，然后重新读取配置文件之后重启。                                                                              |
| 2    | SIGINT  | 程序终止信号，用于终止前台进程。相当于输出 <code>ctrl+c</code> 快捷键。                                                          |
| 8    | SIGFPE  | 在发生致命的算术运算错误时发出。不仅包括浮点运算错误，还包括溢出及除数为0等其它所有的算术的错误。                                                       |
| 9    | SIGKILL | 用来立即结束程序的运行。本信号不能被阻塞、处理和忽略。一般用于强制终止进程。                                                                  |
| 14   | SIGALRM | 时钟定时信号，计算的是实际的时间或时钟时间。 <code>alarm</code> 函数使用该信号。                                                      |
| 15   | SIGTERM | 正常结束进程的信号， <code>kill</code> 命令的默认信号。有时如果进程已经发生问题，这个信号是无法正常终止进程的，我们才会尝试 <code>SIGKILL</code> 信号，也就是信号9。 |
| 18   | SIGCONT | 该信号可以让暂停的进程恢复执行，本信号不能被阻断。                                                                               |
| 19   | SIGSTOP | 该信号可以暂停前台进程，相当于输入 <code>ctrl+z</code> 快捷键。本信号不能被阻断。                                                     |

- `kill -1 24564` # 重启进程
- `kill -9 22342` # 强制杀死进程

### killall 命令

`killall [选项][信号] 进程名` #按照进程名杀死进程  
选项：

- i：交互式，询问是否要杀死某个进程
- l：忽略进程名的大小写

### pkill 命令

`pkill [选项] [信号] 进程名` #按照进程名终止进程  
选项：

- t 终端号：按照终端号踢出用户

- 按照终端号踢出用户
  - `w` # 使用 w 命令查询本机已经登录的用户
  - `pkill -t -9 pts/1` # 强制杀死从 pts/1 虚拟终端登录的进程

## 23.2 工作管理

### 1. 把工作放入后台

- tar -czf etc.tar.gz /etc & ← &表示放入后台执行，命令在后台运行，没有终止
- top ← 在命令执行过程中，按下 Ctrl+z 快捷键，放入后台，但在后台命令被暂停(挂起)

### 2. 查看后台工作

- jobs [-l] ← -l 显示工作的 PID

```
[root@localhost ~]$ jobs
[1]- 已停止 top “-” 代表倒数第二个放入后台的工作
[2]+ 已停止 tar -zcf tmp.tar.gz / “+” 代表最近一个放入后台的工作，也是工作恢复时，默认恢复的工作
```

↑ 工作号

### 3. 将后台暂停的工作恢复到前台执行

- fg %工作号 ← %可以省略，但是注意工作号和 PID 的区别

### 4. 将后台暂停的工作恢复到后台执行

- bg %工作号
- 注意：后台恢复执行的命令，是不能和前台有交互的，否则不能恢复到后台执行

## 22.3 系统资源查看

### vmstat 命令监控系统资源

vmstat [刷新延时 刷新次数]

例如： [root@localhost proc]# vmstat 1 3

```
[root@localhost ~]$ vmstat
procs -----memory----- ---swap-- -----io---- -system-- -----cpu-----
 r b swpd free buff cache si so bi bo in cs us sy id wa st
 1 0 0 1209664 42824 449500 0 0 4 1 8 8 0 0 100 0 0
```

### dmesg 开机时内核检测信息

dmesg

例如： [root@localhost proc]# dmesg | grep CPU

```
[root@localhost ~]$ dmesg | grep CPU
[0.000000] smpboot: Allowing 8 CPUs, 0 hotplug CPUs
[0.000000] setup_percpu: NR_CPUS:5120 nr_cpumask_bits:8 nr_cpu_ids:8 nr_node_ids:1
[0.000000] SLUB: HWalign=64, Order=0-3, MinObjects=0, CPUs=8, Nodes=1
[0.000000] RCU restricting CPUs from NR_CPUS=5120 to nr_cpu_ids=8.
```

### free 命令查看内存使用状态

free [-b|-k|-m|-g]

选项：

- b：以字节为单位显示
- k：以 KB 为单位显示， 默认就是以 KB 为单位显示
- m：以 MB 为单位显示
- g：以 GB 为单位显示

```
[root@localhost ~]$ free -m
 total used free shared buff/cache available
Mem: 1837 174 1181 9 480 1470
Swap: 1022 0 1022
```

- 缓存和缓冲的区别

- 简单来说缓存（cache）是用来加速数据从硬盘中“读取”的，而缓冲（buffer）是用来加速数据“写入”硬盘的

## uptime 命令

### uptime

# 显示系统的启动时间和平均负载，也就是 top 命令的第一行。w 命令也可以看到这个数据

```
[root@localhost ~]$ uptime
13:17:07 up 2:52, 1 user, load average: 0.00, 0.01, 0.05
[root@localhost ~]$ w
13:19:02 up 2:54, 1 user, load average: 0.00, 0.01, 0.05
USER TTY FROM LOGIN@ IDLE JCPU PCPU WHAT
root pts/0 192.168.0.7 10:25 6.00s 1.07s 0.02s w
```

### 查看 CPU 信息

```
cat /proc/cpuinfo
```

### 判断当前系统的位数

```
file /bin/ls
```

随便一个调用 bash 的命令即可

```
[root@localhost ~]$ file /bin/ls
/bin/ls: ELF [64-bit] LSB executable, x86-64, ve
es shared libs), for GNU/Linux 2.6.32, BuildID
1c5d7dfd9, stripped
```

### 查看系统与内核相关信息

```
uname [选项]
```

选项：

- a: 查看系统所有相关信息
- r: 查看内核版本
- s: 查看内核名称

```
[root@localhost ~]$ uname -ars
Linux localhost.localdomain 3.10.0-1127.el7.x86_64 #1 SMP Tue Mar 31 23:36:51 UTC
20
[root@localhost ~]$ uname -a
Linux localhost.localdomain 3.10.0-1127.el7.x86_64 #1 SMP Tue Mar 31 23:36:51 UTC
20
[root@localhost ~]$ uname -r
3.10.0-1127.el7.x86_64
[root@localhost ~]$ uname -s
```

### 列出进程打开或使用的文件信息

```
lsof [选项]
```

选项：

- c 字符串： 只列出以字符串开头的进程打开的文件
- u 用户名： 只列出某个用户的进程打开的文件
- p pid： 列出某个 PID 进程打开的文件

### 查询当前 Linux 系统的发行版本

```
lsb_release -a
```

## 23.4 定时任务

### 1. crond 服务管理与访问控制

- service crond restart
- chkconfig crond on

| 用户的 crontab 设置                   |  |  |
|----------------------------------|--|--|
| <b>crontab [选项]</b>              |  |  |
| 选项:                              |  |  |
| -e: 编辑 crontab 定时任务              |  |  |
| -l: 查询 crontab 任务                |  |  |
| -r: 删除当前用户所有的 crontab 任务         |  |  |
| # 进入 crontab 编辑界面。会打开 vim 编辑你的工作 |  |  |
| ***** 执行的任务                      |  |  |

| 项目      | 含义         | 范围              |
|---------|------------|-----------------|
| 第一个 “*” | 一小时当中的第几分钟 | 0-59            |
| 第二个 “*” | 一天当中的第几小时  | 0-23            |
| 第三个 “*” | 一个月当中的第几天  | 1-31            |
| 第四个 “*” | 一年当中的第几月   | 1-12            |
| 第五个 “*” | 一周当中的星期几   | 0-7 (0和7都代表星期日) |

| 特殊符号 | 含义                                                              |
|------|-----------------------------------------------------------------|
| *    | 代表任何时间。比如第一个 “*” 就代表一小时内每分钟都执行一次的意思。                            |
| ,    | 代表不连续的时间。比如 “0 8,12,16 * * * 命令”，就代表在每天的8点0分，12点0分，16点0分都执行一次命令 |
| -    | 代表连续的时间范围。比如 “0 5 * * 1-6命令”，代表在周一到周六的凌晨5点0分执行命令                |
| */n  | 代表每隔多久执行一次。比如 “*/10 * * * * 命令”，代表每隔10分钟就执行一遍命令                 |

| 时间              | 含义                                                               |
|-----------------|------------------------------------------------------------------|
| 45 22 * * * 命令  | 在22点45分执行命令                                                      |
| 0 17 * * 1 命令   | 每周1的17点0分执行命令                                                    |
| 0 5 1,15 * * 命令 | 每月1号和15号的凌晨5点0分执行命令                                              |
| 40 4 * * 1-5 命令 | 每周一到周五的凌晨4点40分执行命令                                               |
| */10 4 * * * 命令 | 每天的凌晨4点，每隔10分钟执行一次命令                                             |
| 0 0 1,15 * 1 命令 | 每月1号和15号，每周1的0点0分都会执行命令。注意：星期几和几号最好不要同时出现，因为他们定义的都是天。非常容易让管理员混乱。 |

### 举例

\*/5 \* \* \* \* /bin/echo "11" >> /tmp/test

5 5 \* \* 2 /sbin/shutdown -r now

0 5 1,10,15 \* \* /root/sh/autobak.sh

# 二十四. 日志管理

## 24.1 日志管理简介

### 1. 日志服务

- 在 CentOS 6.x 中日志服务已经由 **rsyslogd** 取代了原先的 syslogd 服务。rsyslogd 日志服务更加先进，功能更多。但是不论该服务的使用，还是日志文件的格式其实都是和 syslogd 服务相兼容的，所以学习起来基本和 syslogd 服务一致
- rsyslogd 的新特点
  - 基于 TCP 网络协议传输日志信息
  - 更安全的网络传输方式
  - 有日志消息的及时分析框架
  - 后台数据库
  - 配置文件中可以写简单的逻辑判断
  - 与 syslog 配置文件相兼容

### ● 确定服务启动

- ps aux | grep rsyslogd # 查看服务是否启动

```
[root@localhost ~]$ ps aux | grep rsyslogd
root 1077 0.5 0.3 216424 6688 ? Ss1 10:24 0:00 /usr/sbin/rsyslogd -n
root 1687 0.0 0.0 112828 980 pts/0 S+ 10:26 0:00 grep --color=auto rsyslogd
```

- chkconfig –list | grep rsyslog # 查看服务是否自启动

◆ 注意在 CentOS 7 中，要查看系统的服务要使用 `systemctl list-unit-files`

```
[root@localhost ~]$ chkconfig --list | grep rsyslog
注：该输出结果只显示 SysV 服务，并不包含
原生 systemd 服务。SysV 配置数据
可能被原生 systemd 配置覆盖。

要列出 systemd 服务，请执行 'systemctl list-unit-files'。
查看在具体 target 启用的服务请执行
'systemctl list-dependencies [target]'.
```

◆

```
[root@localhost ~]$ systemctl list-unit-files | grep rsyslog
rsyslog.service enabled
```

## 2. 常见日志的作用

| 日志文件             | 说 明                                                                                                                                                                                                        |
|------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| /var/log/cron    | 记录了系统定时任务相关的日志。                                                                                                                                                                                            |
| /var/log/cups/   | 记录打印信息的日志                                                                                                                                                                                                  |
| /var/log/dmesg   | 记录了系统在开机时内核自检的信息。也可以使用 dmesg 命令直接查看内核自检信息。                                                                                                                                                                 |
| /var/log/btmp    | 记录错误登录的日志。这个文件是二进制文件，不能直接 vi 查看，而要使用 lastb 命令查看，命令如下：<br><pre>[root@localhost log]# lastb<br/>root          tty1              Tue Jun  4 22:38 - 22:38  (00:00)<br/>#有人在6月4日22:38使用root用户，在本地终端1登录错误</pre> |
| /var/log/lastlog | 记录系统中所有用户最后一次的登录时间的日志。这个文件也是二进制文件，不能直接 vi，而要使用 lastlog 命令查看。                                                                                                                                               |

|                  |                                                                                        |
|------------------|----------------------------------------------------------------------------------------|
| /var/log/maillog | 记录邮件信息。                                                                                |
| /var/log/message | 记录系统重要信息的日志。这个日志文件中会记录Linux系统的绝大多数重要信息，如果系统出现问题时，首先要检查的就应该是这个日志文件。                     |
| /var/log/secure  | 记录验证和授权方面的信息，只要涉及账户和密码的程序都会记录比如说系统的登录，ssh的登录，su切换用户，sudo授权，甚至添加用户和修改用户密码都会记录在这个日志文件中。  |
| /var/log/wtmp    | 永久记录所有用户的登录、注销信息，同时记录系统的启动、重启关机事件。同样这个文件也是一个二进制文件，不能直接vi，而需要使用last命令来查看。               |
| /var/run/utmp    | 记录当前已经登录的用户的信息。这个文件会随着用户的登录和注销而不断变化，只记录当前登录用户的信息。同样这个文件不能直接vi，而要使用w, who, users等命令来查询。 |

- 除了系统默认的日志之外，采用 RPM 方式安装的系统服务也会默认把日志记录在 /var/log/ 目录中（源码包安装的服务日志 是在源码包指定目录中）。不过这些日志不是由 rsyslogd 服务来记录和管理的，而是各个服务使用自己的日志管理文档来记录自身日志

| 日志文件            | 说 明                             |
|-----------------|---------------------------------|
| /var/log/httpd/ | RPM包安装的 <b>apache</b> 服务的默认日志目录 |
| /var/log/mail/  | RPM包安装的邮件服务的额外日志目录              |
| /var/log/samba/ | RPM包安装的 <b>samba</b> 服务的日志目录    |
| /var/log/sssd/  | 守护进程安全服务目录                      |

## 24.2 rsyslogd 日志服务

### 1. 日志文件格式

- 基本日志格式包含以下 4 列
  - ◆ 事件产生的时间
  - ◆ 发生事件的服务器主机名
  - ◆ 发生事件的服务器或程序名
  - ◆ 事件的具体信息

### 2. ./etc/rsyslog.conf 配置文件

|                 |                         |
|-----------------|-------------------------|
| authpriv.*      | /var/log/secure         |
| #服务名称[连接符号]日志等级 | 日志记录位置                  |
| #认证相关服务.所有日志等级  | 记录在 /var/log/secure 日志中 |

### 2. 服务名称

| 服务名称           | 说 明                                                                    |
|----------------|------------------------------------------------------------------------|
| auth           | 安全和认证相关消息（不推荐使用authpriv替代）                                             |
| authpriv       | 安全和认证相关消息（私有的）                                                         |
| cron           | 系统定时任务cront和at产生的日志                                                    |
| daemon         | 和各个守护进程相关的日志                                                           |
| ftp            | ftp守护进程产生的日志                                                           |
| kern           | 内核产生的日志（不是用户进程产生的）                                                     |
| local0-local17 | 为本地使用预留的服务                                                             |
| lpr            | 打印产生的日志                                                                |
| mail           | 邮件收发信息                                                                 |
| news           | 与新闻服务器相关的日志                                                            |
| syslog         | 有syslogd服务产生的日志信息（虽然服务名称已经改为rsyslogd，但是很多配置都还是沿用了syslogd的，这里并没有修改服务名）。 |
| user           | 用户等级类别的日志信息                                                            |
| uucp           | uucp子系统的日志信息，uucp是早期linux系统进行数据传递的协议，后来也常用在新闻组服务中。                     |

### 3. 连接符号

- 连接符号可以识别为
  - ◆ “\*”代表所有日志等级，比如：“authpriv.\*”代表 authpriv 认证信息服务产生的日志，所有的日志等级都记录
  - ◆ “.”代表只要比后面的等级高的（包含该等级）日志都记录下来。比如：“cron.info”代表 cron 服务产生的日志，只要日志等级大于等于 info 级别，就记录
  - ◆ “.=”代表只记录所需等级的日志，其他等级的都不记录。比如：“\*.=emerg”代表人和日志服务产生的日志，只要等级是 emerg 等级就记录。这种用法及少见，了解就好
  - ◆ “!”代表不等于，也就是除了该等级的日志外，其他等级的日志都记录

### 日志等级

| 等级名称    | 说 明                                |
|---------|------------------------------------|
| debug   | 一般的调试信息说明                          |
| info    | 基本的通知信息                            |
| notice  | 普通信息，但是有一定的重要性                     |
| warning | 警告信息，但是还不回影响到服务或系统的运行              |
| err     | 错误信息，一般达到err等级的信息以及可以影响到服务或系统的运行了。 |
| crit    | 临界状况信息，比err等级还要严重                  |
| alert   | 警告状态信息，比crit还要严重。必须立即采取行动          |
| emerg   | 疼痛等级信息，系统已经无法使用了                   |

### 日志记录位置

- 日志文件的绝对路径，如“/var/log/secure”
- 系统设备文件，如“/dev/lp0”
- 转发给远程主机，如“@192.168.0.210:514”
- 用户名，如“root”
- 忽略或丢弃日志，如“~”

## 24.3 日志轮替

### 1. 日志文件的命名规则

- 如果配置文件中拥有“dateext”参数，那么日志会用日期作为日志文件的后缀，例如“secure-20130605”。这样的话日志文件名不会重叠，所以也就不需要日志文件的改名，只需要保存指定的日志个数，删除多余的日志文件即可
- 如果配置文件中没有“dateext”参数，那么日志文件就需要进行改名了。当第一次进行日志轮替时，当前的“secure”日志会自动改名为“secure.1”，然后新建“secure”日志，用来保存新的日志。当第二次进行日志轮替时，“secure.1”会自动改名为“secure.2”，当前的“secure”日志会自动改名为“secure.1”，然后也会新建“secure”日志，用来保存新的日志，以此类推

### 2. logrotate 配置文件

| 参数                      | 参数说明                                            |
|-------------------------|-------------------------------------------------|
| daily                   | 日志的轮替周期是每天                                      |
| weekly                  | 日志的轮替周期是每周                                      |
| monthly                 | 日志的轮替周期是每月                                      |
| rotate 数字               | 保留的日志文件的个数。0指没有备份                               |
| compress                | 日志轮替时，旧的日志进行压缩                                  |
| create mode owner group | 建立新日志，同时指定新日志的权限与所有者和所属组。如create 0600 root utmp |

|              |                                                |
|--------------|------------------------------------------------|
| mail address | 当日志轮替时，输出内容通过邮件发送到指定的邮件地址。如mail shenc@lamp.net |
| missingok    | 如果日志不存在，则忽略该日志的警告信息                            |
| notifempty   | 如果日志为空文件，则不进行日志轮替                              |
| minsize 大小   | 日志轮替的最小值。也就是日志一定要达到这个最小值才会轮替，否则就算时间达到也不轮替      |
| size 大小      | 日志只有大于指定大小才进行日志轮替，而不是按照时间轮替。如size 100k         |
| dateext      | 使用日期作为日志轮替文件的后缀。如secure-20130605               |

- 如果要修改配置文件，注意配置文件中的“就近原则”

### 3. 把 apache 日志加入轮替

```
vim /etc/logrotate.conf
/usr/local/apache/logs/access_log {
 daily
 create
 rotate 30
}
```

#### logrotate 命令

##### logrotate [选项] 配置文件名

如果此命令没有选项，则会按照配置文件中的条件进行日志轮替

-v：显示日志轮替过程。加了-v 选项，会显示日志的轮替的过程

-f：强制进行日志轮替。不管日志轮替的条件是否已经符合，强制配置文件中所有的日志进行轮替

# 二十五. 启动管理

## 25.1 CentOS 6.x 启动管理

### 25.1.1 系统运行级别

#### 1. 运行级别

| 运行级别 | 含 义                              |
|------|----------------------------------|
| 0    | 关机                               |
| 1    | 单用户模式，可以想象为windows的安全模式，主要用于系统修复 |
| 2    | 不完全的命令行模式，不含NFS服务                |
| 3    | 完全的命令行模式，就是标准字符界面                |
| 4    | 系统保留                             |
| 5    | 图形模式                             |
| 6    | 重启动                              |

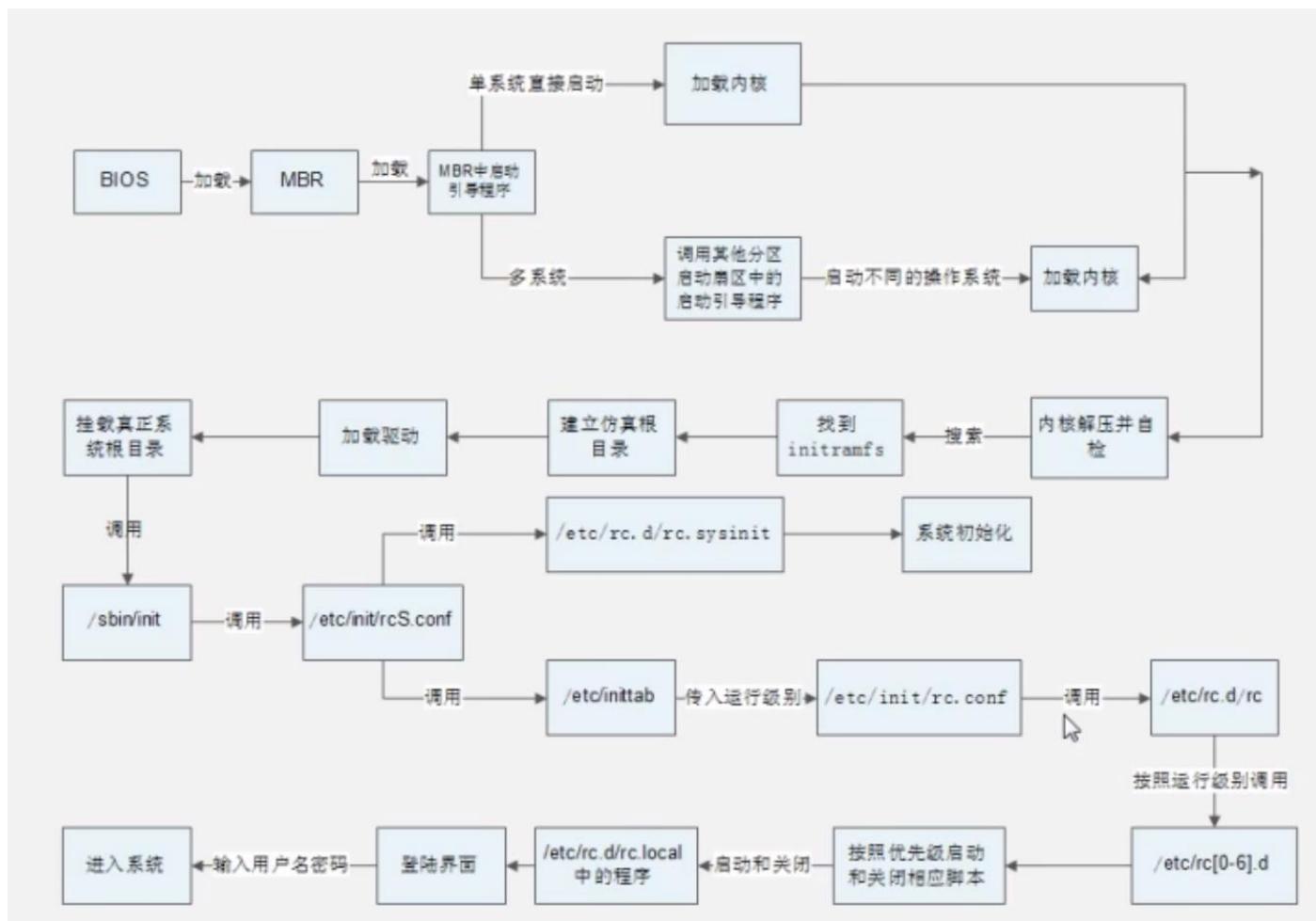
#### 2. 运行级别命令

- runlevel # 查看运行级别命令
- init 运行级别 # 改变运行级别命令

#### 3. 系统默认运行级别

- 系统开机后直接进入哪个运行级别
  - ✧ [root@localhost ~]# vim /etc/inittab
  - ✧ id:3:initdefault:

## 25.1.2 系统启动过程



### initramfs 内存文件系统

- CentOS 6.x 中使用 initramfs 内存文件系统取代了 CentOS 5.x 中的 initrd RAM Disk。他们的作用类似，可以通过启动引导程序加载到内存中，然后加载启动过程中所需要的内核模块，比如 USB、SATA、SCSI 硬盘的驱动和 LVM、RAID 文件系统的驱动

```
◆ mkdir /tmp/initramfs #建立测试目录
◆ cp /boot/initramfs-2.6.32-279.el6.i686.img /tmp/initramfs/ #复制 initramfs 文件
◆ cd /tmp/initramfs/
◆ file initramfs-2.6.32-279.el6.i686.img
◆ mv initramfs-2.6.32-279.el6.i686.img initramfs-2.6.32-279.el6.i686.img.gz #修改文件的后缀名为.gz
◆ gunzip initramfs-2.6.32-279.el6.i686.img.gz #解压缩
◆ file initramfs-2.6.32-279.el6.i686.img
◆ cpio -ivcd < initramfs-2.6.32-279.el6.i686.img #解压缩
```

## 调用/etc/init/rc?.conf 配置文件

- 主要功能是两个：
  - 先调用/etc/rc.d/rc.sysinit，然后由 /etc/rc.d/rc.sysinit 配置文件进行 Linux 系统初始化
  - 然后再调用/etc/inittab，然后由/etc/inittab 配置文件确定系统的默认运行级别

## 由/etc/rc.d/rc.sysinit 初始化

1. 获得网络环境
2. 挂载设备
3. 开机启动画面 Plymouth (取替了过往的 RHGB)
4. 判断是否启用 SELinux
5. 显示于开机过程中的欢迎画面
6. 初始化硬件
7. 用户自定义模块的加载
8. 配置内核的参数
9. 设置主机名
10. 同步存储器
11. 设备映射器及相关的初始化
12. 初始化软件磁盘阵列 (RAID)
13. 初始化 LVM 的文件系统功能
14. 检验磁盘文件系统 (fsck)
15. 设置磁盘配额(quota)
16. 重新以可读写模式挂载系统磁盘
17. 更新 quota (非必要)
18. 启动系统虚拟随机数生成器
19. 配置机器 (非必要)
20. 清除开机过程当中的临时文件
21. 创建 ICE 目录
22. 启动交换分区 (swap)
23. 将开机信息写入/var/log/dmesg 文件中

## 调用/etc/rc.d/rc 文件

- 运行级别参数传入/etc/rc.d/rc 这个脚本之后，由这个脚本文件按照不同的运行级别启动/etc/rc[0-6].d/目录中的相应的程序
  - /etc/rc3.d/K?? 开头的文件 (??是数字)，会按照数字顺序**依次关闭**
  - /etc/rc3.d/S?? 开头的文件 (??是数字)，会按照数字顺序**依次启动**

## 25.2 启动引导程序 grub

### 25.2.1 Grub 配置文件

#### 1. grub 中分区表示

| 硬盘        | 分区      | Linux中设备文件名 | Grub中设备文件名 |
|-----------|---------|-------------|------------|
| 第一块SCSI硬盘 | 第一个主分区  | /dev/sda1   | hd(0, 0)   |
|           | 第二个主分区  | /dev/sda2   | hd(0, 1)   |
|           | 扩展分区    | /dev/sda3   | hd(0, 2)   |
|           | 第一个逻辑分区 | /dev/sda5   | hd(0, 4)   |
| 第二块SCSI硬盘 | 第一个主分区  | /dev/sdb1   | hd(1, 0)   |
|           | 第二个主分区  | /dev/sdb2   | hd(1, 1)   |
|           | 扩展分区    | /dev/sdb3   | hd(1, 2)   |
|           | 第一个逻辑   | /dev/sdb5   | hd(1, 4)   |

#### 2. grub 配置文件

- vim /boot/grub/grub.conf
  - default=0           默认启动第一个系统
  - timeout=5           等待时间， 默认是 5 秒
  - splashimage=(hd0,0)/grub/splash.xpm.gz       这里是指定 grub 启动时的背景图像文件的保存位置的
  - hiddenmenu           隐藏菜单
  - title CentOS (2.6.32-279.el6.i686)       title 就是标题的意思
  - root (hd0,0)        是指启动程序的保存分区
  - kernel /vmlinuz-2.6.32-279.el6.i686 ro  
root=UUID=b9a7a1a8-767f-4a87-8a2b-a535edb362c9  
rd\_NO\_LUKS KEYBOARDTYPE=pc KEYTABLE=us  
rd\_NO\_MD crashkernel=auto LANG=zh\_CN.UTF-8  
rd\_NO\_LVM rd\_NO\_DM rhgb quiet        定义内核加载时的选项
  - initrd /initramfs-2.6.32-279.el6.i686.img    指定了 initramfs 内存文件系统镜像文件的所在位置

## 25.2.2 Grub 加密与字符界面分辨率调整

### 1. grub 加密

- CentOS 6
  - grub-md5-crypt → 生成加密字符串
  - vim /boot/grub/grub.conf
  - default=0
  - timeout=5
  - password --md5 加密后的字符串** ← 加入这条说明
  - splashimage=(hd0,0)/grub/splash.xpm.gz
  - hiddenmenu
  - …后续内容省略…
- CentOS 7
  - xfsgrub

### 2. 纯字符界面分辨率调整

#### 1). 查询内核是否支持分辨率修改 →

```
grep "CONFIG_FRAMEBUFFER_CONSOLE" /boot/config-2.6.32-279.el6.i686
```

| 色深  | 640×480 | 800×600 | 1024×768 | 1280×1024 |
|-----|---------|---------|----------|-----------|
| 8位  | 769     | 771     | 773      | 775       |
| 15位 | 784     | 787     | 790      | 793       |
| 16位 | 785     | 788     | 791      | 794       |
| 32位 | 786     | 789     | 792      | 795       |

- 在一些版本的 Linux 当中，可能不支持十进制的分辨率，修改为十六进制即可

#### 2). 在配置文件中加入方框内的东西即可

```
vi /boot/grub/grub.conf
kernel /vmlinuz-2.6.32-279.el6.i686 ro
root=UUID=b9a7a1a8-767f-4a87-8a2b-
a535edb362c9 rd_NO_LUKS
KEYBOARDTYPE=pc KEYTABLE=us
rd_NO_MD crashkernel=auto LANG=zh_CN.UTF-
8 rd_NO_LVM rd_NO_DM rhgb quiet vga=791
```

## 25.3 系统修复模式

### 1. 单用户模式

- 单用户模式常见的错误修复

- 忘记 root 密码
- 修改系统默认运行级别

### 2. 光盘修复模式

- 重要系统文件丢失，导致系统无法正常启动

```
bash-4.1# chroot /mnt/sysimage #改变主目录
sh-4.1# cd /root

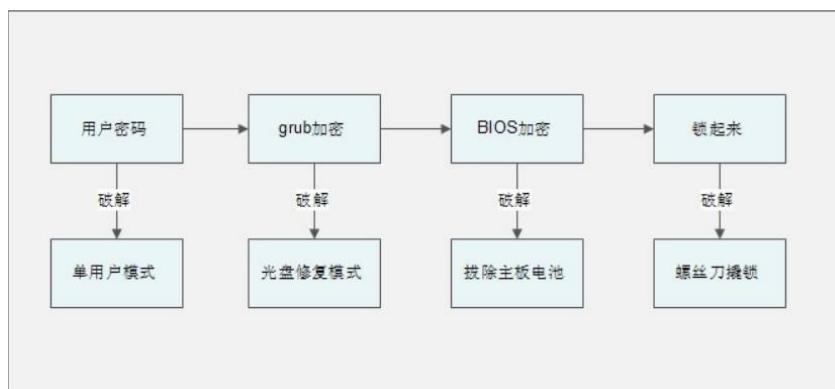
sh-4.1# rpm -qf /etc/inittab #查询下/etc/inittab 文件属于哪个包
sh-4.1# mkdir /mnt/cdrom #建立挂载点

sh-4.1# mount /dev/sr0 /mnt/cdrom #挂载光盘

sh-4.1# rpm2cpio \ /mnt/cdrom/Packages/initscripts-8.45.3-1.i386.rpm \ | cpio -idv ./etc/inittab
 #提取 inittab 文件到当前目录

sh-4.1# cp etc/inittab /etc/inittab #复制 inittab 文件到指定位置
```

### 3. Linux 的安全性



# 二十六. 备份与恢复

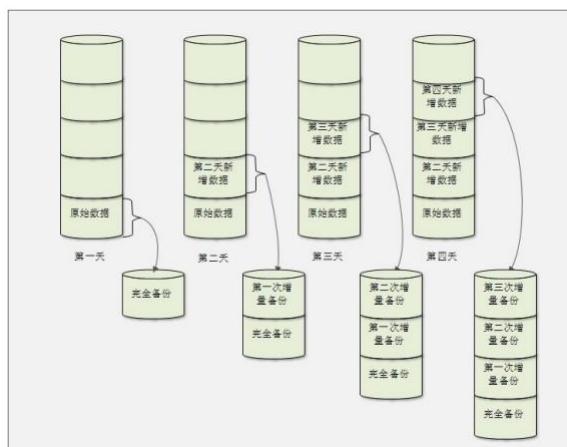
## 26.1 备份概述

### 1. Linux 系统需要备份的数据

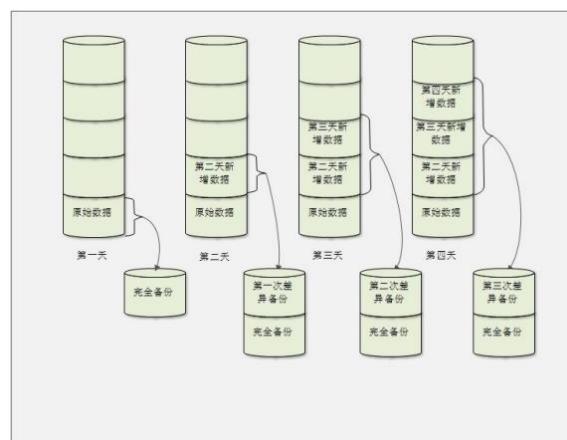
- /root/目录
- /home/目录
- /var/spool/mail/目录
- /etc/目录
- 其他目录：
- 安装服务的数据
  - apache 需要备份的数据
    - ◆ 配置文件
    - ◆ 网页主目录
    - ◆ 日志文件
  - mysql 需要备份的数据
    - ◆ 源码包安装的 mysql: /usr/local/mysql/data/
    - ◆ RPM 包安装的 mysql: /var/lib/mysql/

### 2. 备份策略

- 完全备份：完全备份就是指把所有需要备份的数据全部备份，当然完全备份可以备份整块硬盘，整个分区或某个具体的目录
- 增量备份



- 差异备份



## 26.2 dump 和 restore 命令

### 1. dump 命令

#### dump 命令

dump [选项] 备份之后的文件名 原文件或目录

选项：

- level：就是我们说的 0-9 十个备份级别 → 使用的时候用 -数字，-lever 只是便于说明
- f 文件名：指定备份之后的文件名
- u：备份成功之后，把备份时间记录在/etc/dumpdates 文件
- v：显示备份过程中更多的输出信息
- j：调用 bzlib 库压缩备份文件，其实就是把备份文件压缩为.bz2 格式，默认压缩等级是 2
- W：显示允许被 dump 的分区的备份等级及备份时间

#### ● 备份分区

```
dump -0uj -f /root/boot.bak.bz2 /boot/ #备份命令。先执行一次完全备份，并压缩和更新备份时间
cat /etc/dumpdates #查看备份时间文件
cp install.log /boot/ #复制日志文件到/boot 分区
dump -1uj -f /root/boot.bak1.bz2 /boot/ #增量备份/boot 分区，并压缩
dump -W #查询分区的备份时间及备份级别的
```

#### ● 备份文件或目录

```
dump -0j -f /root/etc.dump.bz2 /etc/
#完全备份/etc/目录，只能使用 0 级别进行完全备份，而不再支持增量备份
```

### 2. restore 命令

#### restore 命令

restore [模式选项] [选项]

模式选项：restore 命令常用的模式有以下四种，这四个模式不能混用。

- C：比较备份数据和实际数据的变化
- i：进入交互模式，手工选择需要恢复的文件。
- t：查看模式，用于查看备份文件中拥有哪些数据。
- r：还原模式，用于数据还原。

选项：

- f：指定备份文件的文件名

- 比较备份数据和实际数据的变化

```
mv /boot/vmlinuz-2.6.32-279.el6.i686 /boot/vmlinuz-2.6.32-279.el6.i686.bak
#把/boot 目录中内核镜像文件改个名字
restore -C -f /root/boot.bak.bz2
#restore 发现内核镜像文件丢失
```

- 查看模式

```
restore -t -f boot.bak.bz2
```

- 还原模式

```
#还原 boot.bak.bz2 分区备份
```

```
#先还原完全备份的数据
```

```
mkdir boot.test
```

```
cd boot.test/ restore -r -f /root/boot.bak.bz2 #解压缩
```

```
restore -r -f /root/boot.bak1.bz2 #恢复增量备份数据
```

```
restore -r -f etc.dump.bz2 #还原/etc/目录的备份 etc.dump.bz2
```

```
#还原 etc.dump.bz2 备份
```

# 二十七. GCC

## 27.1 基础

gcc 与 g++ 分别是 gnu 的 c & c++ 编译器 gcc/g++ 在执行编译工作的时候，总共需要 4 步：

- 1、预处理,生成 .i 的文件[预处理器 cpp]
- 2、将预处理后的文件转换成汇编语言, 生成文件 .s [编译器 egcs]
- 3、有汇编变为目标代码(机器代码)生成 .o 的文件[汇编器 as]
- 4、连接目标代码, 生成可执行程序 [链接器 ld]

## 参数详解

### -x language filename

设定文件所使用的语言, 使后缀名无效, 对以后的多个有效。也就是根据约定 C 语言的后缀名称是 .c 的, 而 C++ 的后缀名是 .C 或者 .cpp, 如果你很个性, 决定你的 C 代码文件的后缀名是 .pig 哈哈, 那你就要用这个参数, 这个参数对他后面的文件名都起作用, 除非到了下一个参数的使用。可以使用的参数吗有下面的这些: 'c', 'objective-c', 'c-header', 'c++', 'cpp-output', 'assembler', 与 'assembler-with-cpp'。

看到英文, 应该可以理解的。

例子用法:

```
gcc -x c hello.pig
```

### -x none filename

关掉上一个选项, 也就是让 gcc 根据文件名后缀, 自动识别文件类型。

例子用法:

```
gcc -x c hello.pig -x none hello2.c
```

### -c

只激活预处理,编译,和汇编,也就是他只把程序做成 obj 文件

例子用法:

```
gcc -c hello.c
```

他将生成 .o 的 obj 文件

### -S

只激活预处理和编译, 就是指把文件编译成为汇编代码。

例子用法:

```
gcc -S hello.c
```

他将生成 .s 的汇编代码, 你可以用文本编辑器察看。

## -E

只激活预处理,这个不生成文件, 你需要把它重定向到一个输出文件里面。

例子用法:

```
gcc -E hello.c > pianoapan.txt
gcc -E hello.c | more
```

慢慢看吧, 一个 hello word 也要与处理成 800 行的代码。

## -O

制定目标名称, 默认的时候, gcc 编译出来的文件是 a.out, 很难听, 如果你和我有同感, 改掉它, 哈哈。

例子用法:

```
gcc -o hello.exe hello.c (123456789)
gcc -o hello.asm -S hello.c
```

## -pipe

使用管道代替编译中临时文件, 在使用非 gnu 汇编工具的时候, 可能有些问题。

```
gcc -pipe -o hello.exe hello.c
```

## -ansi

关闭 gnu c 中与 ansi c 不兼容的特性, 激活 ansi c 的专有特性(包括禁止一些 asm inline typeof 关键字, 以及 UNIX,vax 等预处理宏)。

## -fno-asm

此选项实现 ansi 选项的功能的一部分, 它禁止将 asm, inline 和 typeof 用作关键字。

## -fno-strict-prototype

只对 g++ 起作用, 使用这个选项, g++ 将对不带参数的函数, 都认为是没有显式的对参数的个数和类型说明, 而不是没有参数。

而 gcc 无论是否使用这个参数, 都将对没有带参数的函数, 认为城没有显式说明的类型。

## -fthis-is-variable

就是向传统 c++ 看齐, 可以使用 this 当一般变量使用。

## -fcond-mismatch

允许条件表达式的第二和第三参数类型不匹配, 表达式的值将为 void 类型。

## -funsigned-char 、 -fno-signed-char、 -fsigned-char 、 -fno-unsigned-char

这四个参数是对 char 类型进行设置, 决定将 char 类型设置成 unsigned char(前两个参数)或者 signed char(后两个参数)。

### **-include file**

包含某个代码,简单来说,就是便以某个文件,需要另一个文件的时候,就可以用它设定,功能就相当于在代码中使用 `#include<filename>`。

例子用法:

```
gcc hello.c -include /root/pianopan.h
```

### **-imacros file**

将 file 文件的宏, 扩展到 gcc/g++ 的输入文件, 宏定义本身并不出现在输入文件中。

### **-Dmacro**

相当于 C 语言中的 `#define macro`

### **-Dmacro=defn**

相当于 C 语言中的 `#define macro=defn`

### **-Umacro**

相当于 C 语言中的 `#undef macro`

### **-undef**

取消对任何非标准宏的定义

### **-I dir**

在你是用 `#include "file"` 的时候, gcc/g++ 会先在当前目录查找你所制定的头文件, 如果没有找到, 他回到默认的头文件目录找, 如果使用 -I 制定了目录, 他会先在你所制定的目录查找, 然后再按常规的顺序去找。

对于 `#include<file>`, gcc/g++ 会到 -I 制定的目录查找, 查找不到, 然后将到系统的默认的头文件目录查找。

### **-I-**

就是取消前一个参数的功能, 所以一般在 -I dir 之后使用。

### **-idirafter dir**

在 -I 的目录里面查找失败, 讲到这个目录里面查找。

### **-iprefix prefix 、 -iwithprefix dir**

一般一起使用, 当 -I 的目录查找失败, 会到 prefix+dir 下查找

### **-nostdinc**

使编译器不再系统默认的头文件目录里面找头文件, 一般和 -I 联合使用, 明确限定头文件的位置。

#### **-nostdin C++**

规定不在 g++ 指定的标准路径中搜索，但仍在其他路径中搜索，此选项在创 libg++ 库使用。

#### **-C**

在预处理的时候，不删除注释信息，一般和-E 使用，有时候分析程序，用这个很方便的。

#### **-M**

生成文件关联的信息。包含目标文件所依赖的所有源代码你可以用 `gcc -M hello.c` 来测试一下，很简单。

#### **-MM**

和上面的那个一样，但是它将忽略由 `#include<file>` 造成的依赖关系

#### **-MD**

和-M 相同，但是输出将导入到.d 的文件里面

#### **-MMD**

和 -MM 相同，但是输出将导入到 .d 的文件里面。

#### **-Wa,option**

此选项传递 option 给汇编程序；如果 option 中间有逗号，就将 option 分成多个选项，然后传递给会汇编程序。

#### **-Wl,option**

此选项传递 option 给连接程序；如果 option 中间有逗号，就将 option 分成多个选项，然后传递给会连接程序。

#### **-llibrary**

制定编译的时候使用的库

#### 例子用法

```
gcc -lcurses hello.c
```

## 27.2 使用 ncurses 库编译程序

### -Ldir

制定编译的时候，搜索库的路径。比如你自己的库，可以用它制定目录，不然编译器将只在标准库的目录找。这个 dir 就是目录的名称。

### -O0 、 -O1 、 -O2 、 -O3

编译器的优化选项的 4 个级别，-O0 表示没有优化，-O1 为默认值，-O3 优化级别最高。

### -g

只是编译器，在编译的时候，产生调试信息。

### -gstabs

此选项以 stabs 格式声称调试信息，但是不包括 gdb 调试信息。

### -gstabs+

此选项以 stabs 格式声称调试信息，并且包含仅供 gdb 使用的额外调试信息。

### -ggdb

此选项将尽可能的生成 gdb 的可以使用的调试信息。

### -static

此选项将禁止使用动态库，所以，编译出来的东西，一般都很大，也不需要什么动态连接库，就可以运行。

### -share

此选项将尽量使用动态库，所以生成文件比较小，但是需要系统由动态库。

### -traditional

试图让编译器支持传统的 C 语言特性。

GCC 是 GNU 的 C 和 C++ 编译器。实际上，GCC 能够编译三种语言：C、C++ 和 Object C（C 语言的一种面向对象扩展）。利用 gcc 命令可同时编译并连接 C 和 C++ 源程序。

使用逆向优化：

### -fprofile-generate -fprofile-use

-fprofile-generate 和 -fprofile-use，两者需要搭配使用

1. 首先使用 -fprofile-generate -o EXE\_NAME 生成可执行文件
2. 运行编译出来的可执行文件
3. 使用-fprofile-use -o EXE\_NAME 进行再次编译，此时将使用反馈文件进行负反馈优化

## 27.3 gcc 命令的常用选项

| 选项           | 解释                                               |
|--------------|--------------------------------------------------|
| -ansi        | 只支持 ANSI 标准的 C 语法。这一选项将禁止 GNU C 的某些特色，如 asm 关键词。 |
| -c           | 只编译并生成目标文件。                                      |
| -DMACRO      | 以字符串"1"定义 MACRO 宏。                               |
| -DMACRO=DEFN | 以字符串"DEFN"定义 MACRO 宏。                            |
| -E           | 只运行 C 预编译器。                                      |
| -g           | 生成调试信息。GNU 调试器可利用该信息。                            |
| -IDIRECTORY  | 指定额外的头文件搜索路径 DIRECTORY。                          |
| -LDIRECTORY  | 指定额外的函数库搜索路径 DIRECTORY。                          |
| -ILIBRARY    | 连接时搜索指定的函数库 LIBRARY。                             |
| -m486        | 针对 486 进行代码优化。                                   |
| -o FILE      | 生成指定的输出文件。用在生成可执行文件时。                            |
| -O0          | 不进行优化处理。                                         |
| -O 或 -O1     | 优化生成代码。                                          |
| -O2          | 进一步优化。                                           |
| -O3          | 比 -O2 更进一步优化，包括 inline 函数。                       |
| -shared      | 生成共享目标文件。通常用在建立共享库时。                             |
| -static      | 禁止使用共享连接。                                        |
| -UMACRO      | 取消对 MACRO 宏的定义。                                  |
| -w -Wall     | 不生成任何警告信息。 生成所有警告信息。                             |

# 二十八. 向量

## 可以自动矢量化的代码

使用 gcc version 4.3.4 的 gcc 编译器，编译下面的示例代码：

```
1. int main(){
2.
3. const unsigned int ArraySize = 10000000;
4.
5. float* a = new float[ArraySize];
6.
7. float* b = new float[ArraySize];
8.
9. float* c = new float[ArraySize];
10.
11. for (unsigned int j = 0; j < 200 ; j++) // some repetitions
12.
13. for (unsigned int i = 0; i < ArraySize; ++ i)
14.
15. c[i] = a[i] * b[i];
16.
17. }
```

编译并运行上面的代码：

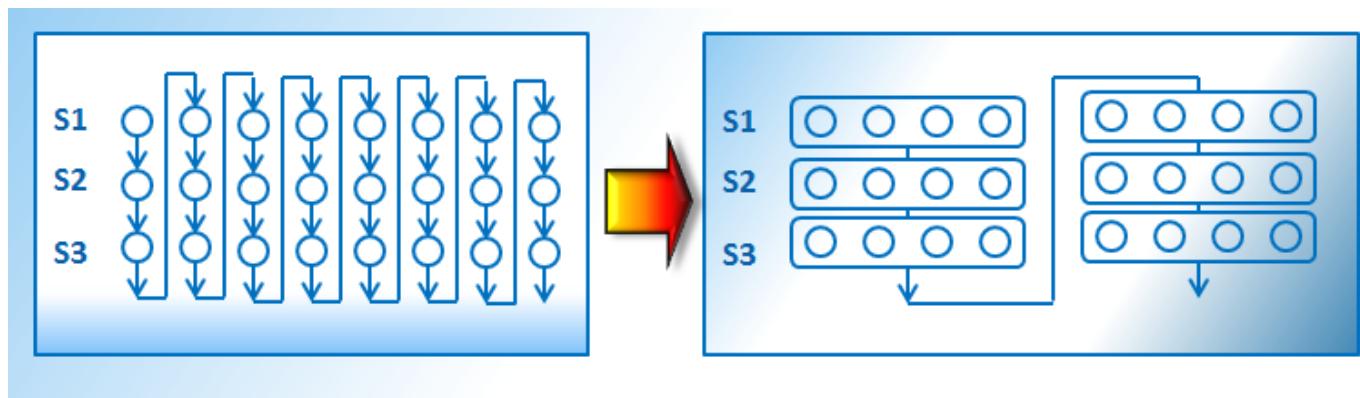
```
g++ vectorisable_example.cpp -o vectorisable_example -O2 time ./vectorisable_example
```

```
g++ vectorisable_example.cpp -o vectorisable_example_vect -O2 -ftree-vectorize time ./vectorisable_example_vect
```

第二种编译方式，产生的可执行程序比第一种产生的可执行程序运行的更快？我们深入研究下，看看为什么他就这么快？

## 简介

从上世纪九十年代后期开始，英特尔就已经将单指令多数据（SIMD）机器指令集成到其商品 CPU 产品线中。这些 SIMD 指令允许一次将相同的数学运算（乘，加...）应用于 2, 4 或甚至更多的数据值上。这组数据值也被称为“vector”（不要与代数中的 vector 混淆）。理论上矢量计算的性能增益等于 CPU 可以容纳的矢量单位的数量。



尽管 SIMD 指令集已经集成到普通 CPU 中相当长一段时间了，但是这些扩展功能（SIMD 指令集）只能通过在 C 或者 C++ 代码中通过准汇编语言进行使用。像 GNU 编译器系列（GCC）这样的现代编译器现在可以将常规的 C 或 C ++ 源代码转换成向量操作。这个过程被称为自动矢量化（**auto-vectorization**）。

这篇文章，我们将会介绍必要的软件工具和编程技术，并进一步提供利用 GCC 自动矢量化功能的示例源代码。

## 硬件

要估算自动矢量化代码可以带来多大的性能提升，首先需要了解您正在使用的 CPU 的能力以及 CPU 可以处理多少个 Vector。因此，运行以下命令：

```
$ cat /proc/cpuinfo...
flags: fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush
dts acpi mmx fxsr sse sse2ss ht tm pbe syscall nx rdtscp lm constant_tsc arch_perfmon
pebs bts rep_good nopl xtopology nonstop_tsc aperfmpf perf pni pclmulqdq dtes64 monitor
ds_cpl vmx est tm2sse3cx16 xtpr pdcmssse4_1 sse4_2x2apic popcnt xsaveavxlahf_lm ida
arat epb xsaveopt pln pts dts tpr_shadow vnmi flexpriority ept vpid...
```

在 CPU 属性中，您将找到有关 SIMD 功能的信息（以粗体突出显示）。若信息中包含下表列出的字符串，则您的 CPU 支持(SIMD)功能：

| Name    | Register Size | Amount of floats | Amount of doubles |
|---------|---------------|------------------|-------------------|
| mmx (*) | 64 bit        | 0                | 0                 |
| sse2    | 128 bit       | 4                | 2                 |
| ssse3   | 128 bit       | 4                | 2                 |
| sse4_1  | 128 bit       | 4                | 2                 |
| sse4_2  | 128 bit       | 4                | 2                 |
| avx     | 256 bit       | 8                | 4                 |

- 英特尔处理器上的第一个矢量单元只能容纳两个迭代器。

所有 Intel 和 AMD 64 位 CPU 都至少支持 sse2 指令集。 AVX 指令集已于 2011 年推出，采用 Sandy Bridge 架构，可在最新一代 Mac Book Pro 笔记本电脑上使用。

AVX 指令集可以同时对四个双精度浮点值应用数学运算。因此，与未使用矢量单位的版本相比，应用程序可以实现的最大性能增益是 4 倍。

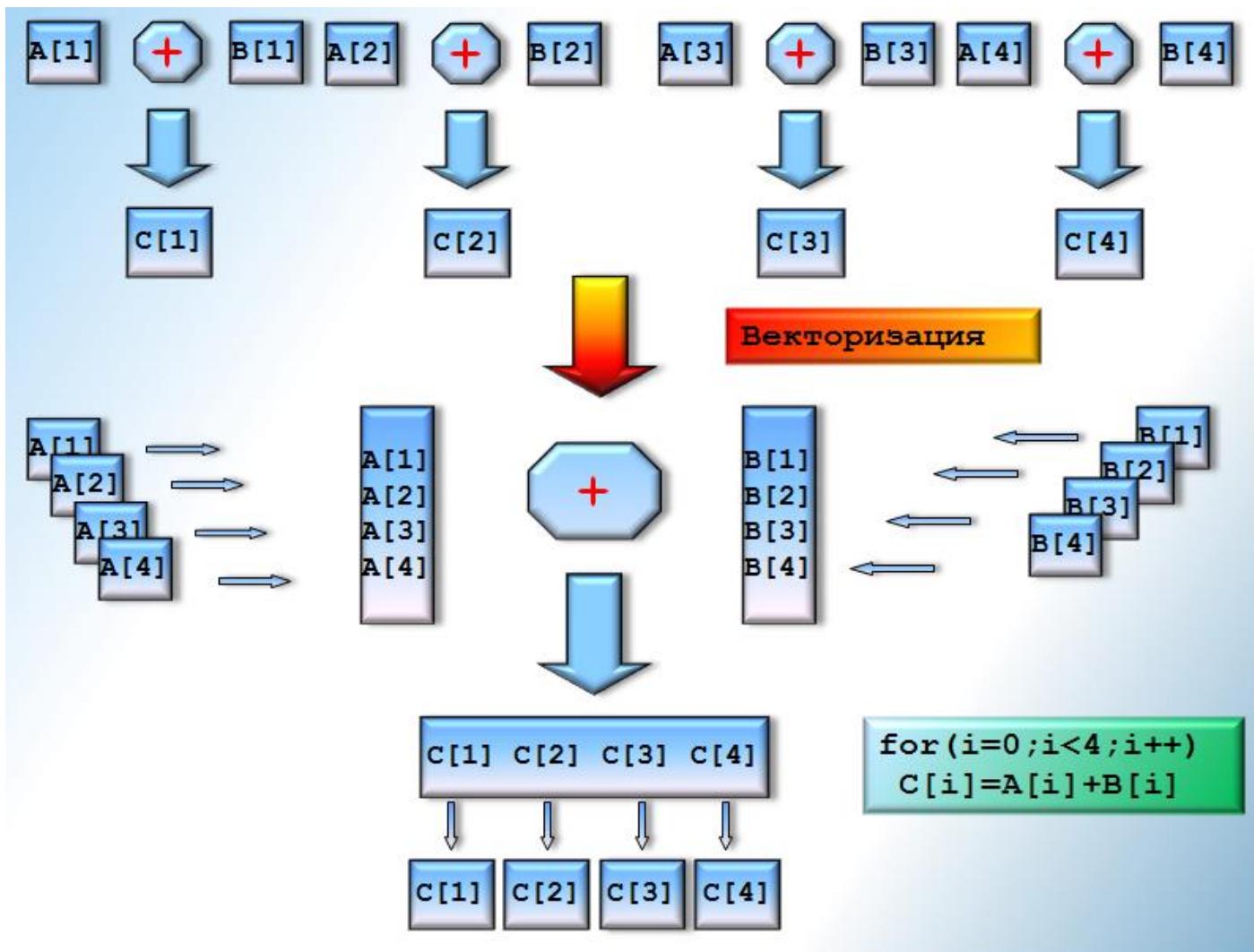
Vector 单位大小的持续增大，反应了是计算机微体系结构演化的新趋势。十年前，计算机的时钟频率几乎每年翻一番。计算机的时钟频率已经达到了极限，制造商转而开始提供越来越大的矢量单位，并且在每个芯片中提供越来越多的核心数量。

## 哪些代码可以被 auto-vectorized ?

为了将计算分布到 CPU 的矢量单元，编译器必须对源代码的依赖性和相互影响有一个很好的理解。但最重要的是，编译器必须能够检测出那些代码段可以使用 SIMD 指令进行优化。最简单的情况是对数组执行计算的循环：

```
for (unsigned int i = 0; i < ArraySize; ++ i)
{
 c[i] = a[i] * b[i];
}
```

使用等于或者高于 gcc461(在 slc5\_amd64\_gcc461\_scram 体系结构的服务器中可用) 版本的 gcc 编译器，编译的时候，一定要带上 `-fno-tree-vectorize`。



**注意:**从自动矢量化中获益最多的循环是包含数学运算的循环。只是迭代对象集合的循环不会获利，在大多数情况下甚至不能自动矢量化。

一旦你使用选项`-ftree-vectorizer-verbose = 7`编译你的代码，GCC 将会给你一个关于你的程序中所有循环的详细报告，以及它们是否已经被自动矢量化了。以下报告是成功对循环进行向量化的结果：

```
autovect.cpp:66: note: vect_model_load_cost: aligned.

autovect.cpp:66: note: vect_get_data_access_cost: inside_cost = 1, outside_cost = 0.

autovect.cpp:66: note: vect_model_load_cost: aligned.

autovect.cpp:66: note: vect_get_data_access_cost: inside_cost = 2, outside_cost = 0.

autovect.cpp:66: note: vect_model_store_cost: aligned.

autovect.cpp:66: note: vect_get_data_access_cost: inside_cost = 3, outside_cost = 0.

autovect.cpp:66: note: vect_model_load_cost: aligned.

autovect.cpp:66: note: vect_model_load_cost: inside_cost = 1, outside_cost = 0 .

autovect.cpp:66: note: vect_model_load_cost: aligned.
```

```
autovect.cpp:66: note: vect_model_load_cost: inside_cost = 1, outside_cost = 0 .

autovect.cpp:66: note: vect_model_simple_cost: inside_cost = 1, outside_cost = 0 .

autovect.cpp:66: note: vect_model_simple_cost: inside_cost = 1, outside_cost = 1 .

autovect.cpp:66: note: vect_model_store_cost: aligned.

autovect.cpp:66: note: vect_model_store_cost: inside_cost = 1, outside_cost = 0 .

autovect.cpp:66: note: Cost model analysis:

Vector inside of loop cost: 5

Vector outside of loop cost: 11

Scalar iteration cost: 5

Scalar outside cost: 0

prologue iterations: 0

epilogue iterations: 2

Calculated minimum iters for profitability: 3

autovect.cpp:66: note: Profitability threshold = 3

autovect.cpp:66: note: Profitability threshold is 3 loop iterations.

autovect.cpp:66: note: LOOP VECTORIZED.
```

若一个循环不能被向量化, GCC 将会给出原因:

## 让 GCC 自动对你的代码进行向量化

想要让 GCC 对自己的代码进行自动向量化, 需要使用最新的编译器, 建议使用至少 GCC 4.6.1 版本的编译器, 通常来说, 版本越新越好。

## 编译器标示

想要打开 auto-vectorization, 使用标示:

`-ftree-vectorize`

如果您使用优化级别`-O3` 或进行编译, 则隐式启用此选项。

想要得到哪些 loop 是已经被 auto-vectorized 和哪些 loops 没有被成功矢量化以及原因, 可以使用选项:

```
-ftree-vectorizer-verbose=7
```

关于如何阅读这个输出，请看下面的 Real-World 代码示例部分。

某些循环结构（例如浮点数的减法）只能在允许编译器改变数学运算顺序的情况下进行矢量化。要做到这一点，需要使用选项：

```
-ffast-math
```

如果您使用优化级别`-Ofast`，则会隐式启用该选项。请注意，`fast-math` 选项会修正修改浮点运算中的错误操作。详情请看 <http://gcc.gnu.org/onlinedocs/gcc-4.1.1/gcc/Optimize-Options.html>

此外，您可以明确指出编译器在生成二进制文件时应使用哪一个 SIMD 指令集。在编译 x86-64 架构时，GCC 默认使用 SSE2 指令集。如果要启用 AVX 指令集，请使用编译器标志：

```
-mavx
```

需要注意的是，要运行二进制文件的机器必须支持 AVX 指令集。您还可以让通过下面的选项，让编译器自己决定使用机器中的哪个指令集：

```
-march=native
```

若你想要使用 C++11 的特性，例如：lambda 表达式，一定要确认开启了新的 C++ Standard：

```
-std=gnu++0x
```

## 启用 auto-vectorization 最佳实践

### 使用数组结构

想要启用自动向量化，编译器不仅需要能够分析循环迭代，还需要能够分析访问内存的模式。嵌套数组的复杂类型很难或不可能由编译器自动进行矢量化。建议使用简单的 c 数组 or `std :: arrays`（在 C++ 11 中引入）来保存数据，并允许编译器以连续的方式访问数据。这也将使您的程序能够利用 CPU 的各种缓存。

如果你需要可变数组，建议使用 `std :: vector` 并获取指向第一个元素的指针来访问循环中的元素：

```
std::vector myData(23);

double * pMyData = &myData.front();

...
pMyData[i] += 5;
```

## 限制分支

尽力限制 for 循环内的分支。GCC 能够将一些 if 语句翻译成向量化的代码，但很有限。这样做时，将评估所有可能的分支，并丢弃未采用分支的值。

## 尽量简化代码

如果在 for 循环中有复杂的计算，您希望 GCC 对其进行矢量化，可以考虑使用 C++ 11 中的 lambda 表达式，将它们分解为更简单的代码。在这里可以找到 C++11 新功能的介绍：[http://en.wikipedia.org/wiki/Anonymous\\_function#C.2B.2B](http://en.wikipedia.org/wiki/Anonymous_function#C.2B.2B)

```
auto kernel_square =

[] // capture nothing

(double const& a) // take 1 parameter by reference

->double // lambda function returns the square

{

return (a * a);

};
```

请注意，此代码将由 GCC 自动矢量化。对 lambda 函数的调用不会像常规函数调用那样，造成特别大的开销，因为代码会被完全内联。

另一个更全面的例子是 lambda 函数中的 for 循环。这个循环也会被 GCC 自动矢量化：

```
1. // Defining a compute kernel to encapsulate a specific computation
2.
3. auto kernel_multiply =
4.
5. [&cFixedMultiply] // capture the constant by reference of the scope of the lambda expression
6.
7. (DataArray const& a, DataArray const& b, DataArray & res) // take 3 parameters by reference
8.
9. ->void // lambda function returns void
10.
11. {
12.
13. // simple loop vectorized
14.
15. for(unsigned int i = 0; i < a.size(); ++ i)
16.
```

```
17. {
18.
19. res[i] = a[i] * b[i];
20.
21. }
22.
23.};
```

## 不要调用外部函数

---

调用外部函数，如 `exp()`，`log()` 等等，会导致循环不能被向量化。因此，您必须决定循环中的数学运算是否足够，从而将循环分解。这意味着，在第一个循环中，计算调用 `exp()` 时，需要使用的参数，并将此结果存储在临时数组中。GCC 会自动矢量化这个循环。第二个循环将简单地执行对 `exp()` 的调用并存储结果。

如果要调用的函数是由您控制的，请尽量尝试将此函数指定为 C++ 11 lambda 表达式。可以在这里

## 在循环中使用普通的整数计数器

---

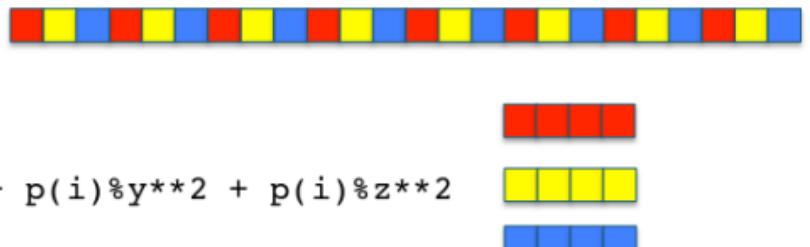
使用普通的整数计数器来构建 `for` 循环，而不是 `std::iterator`，因为这些使得 GCC 难以分析内存访问和循环的属性，如迭代计数。

```
1. for (vector::iterator it = y.begin(); it != y.end(); it++)
2.
3. {
4.
5. (*it) += 23;
6.
7. }
8. for (unsigned int i = 0; i < y.size(); ++i)
9.
10. {
11.
12. y[i] += 23;
13.
14. }
```

## 数组结构 (单)

```
type coords
 real :: x, y, z
end type
type (coords) :: p(100)
real dsquared(100)
```

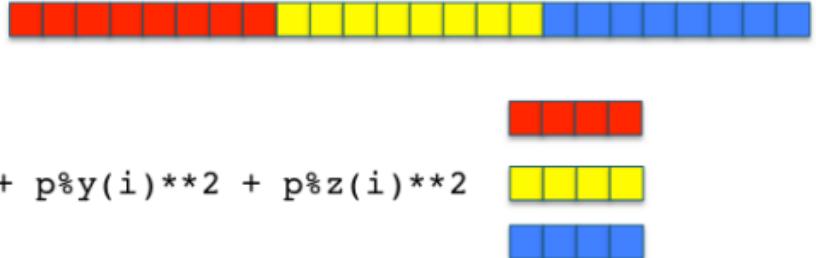
```
do i=1,100
 dsquared(i) = p(i)%x**2 + p(i)%y**2 + p(i)%z**2
end do
```



## 数组结构 (多)

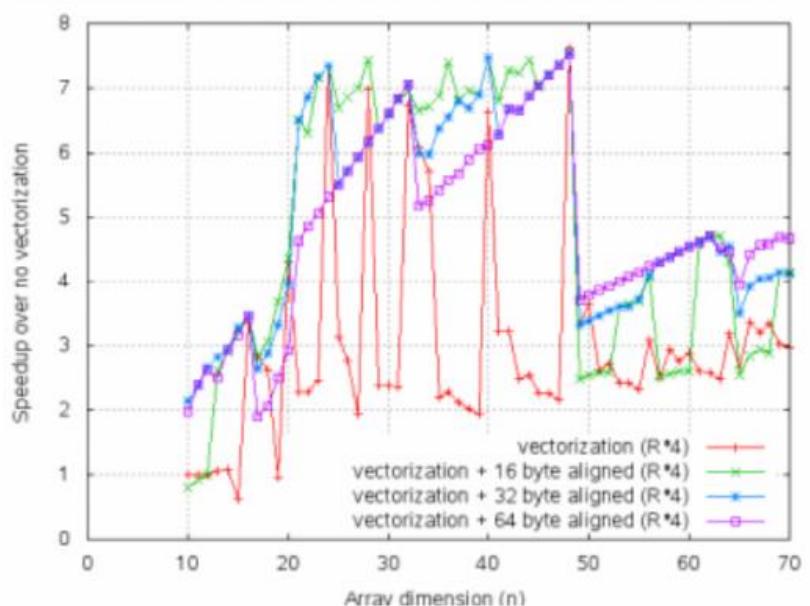
```
type coords
 real :: x(100), y(100), z(100)
end type
type (coords) :: p
real dsquared(100)
```

```
do i=1,100
 dsquared(i) = p%x(i)**2 + p%y(i)**2 + p%z(i)**2
end do
```



## 在整齐排列数据下 («Наивное» выравнивание данных)

```
real, allocatable :: a(:,:), b(:,:), c(:,:)
!dir$ attributes align : 32 :: a,b,c
...
allocate (a(npadded,n))
allocate (b(npadded,n))
allocate (c(npadded,n))
...
do j=1,n
 do k=1,n
 !dir$ vector aligned
 do i=1,npadded
 c(i,j) = c(i,j) &
 + a(i,k) * b(k,j)
 end do
 end do
end do
!.... Ignore c(n+1:npadded,:)
```



## 综合代码示例

下面的代码给出了一些关于如何用 GCC 编写自动矢量化 C ++ 代码的例子。复制并粘贴源代码并使用下面命令进行编译即可：

```
1. /*
2.
3. Compile with (use at least gcc 4.7):
4. g++ -Ofast -ftree-vectorizer-verbose=7 -march=native -std=c++11 -o autovect autovect.cpp
5.
6. */
7.
8. #include <math.h>
9.
10. #include <string>
11. #include <iostream>
12. #include <array>
13. #include <vector>
14.
15. // Structure-Of-Array to hold coordinates
16. struct Vector3
17. {
18. std::vector<double> x;
19. std::vector<double> y;
20. std::vector<double> z;
21.
22. // final result of the distance calculation
23. std::vector<double> distance;
24.
25. void add(double _x, double _y, double _z)
26. {
27. x.push_back(_x);
28. y.push_back(_y);
29. z.push_back(_z);
30. distance.push_back(0.0f);
31. }
32.
33. };
34.
35.
36. int main()
37. {
38.
39. // Fixed Size Arrays
40.
41. typedef std::array<double, 10> DataArray;
42.
```

```

43. DataArray vect_a = { 0,1,2,3,4,5,6,7,8,9 };
44. DataArray vect_b = { 0.5,1,2,3,4,5,6,7,8,9 };
45. DataArray vect_res_plain = { 0,0,0,0,0,0,0,0,0,0 };
46. DataArray vect_res_lambda = { 0,0,0,0,0,0,0,0,0,0 };
47.
48. constexpr double cFixedMultiply = 23.5f;
49.
50. // simple loop vectorized
51. // -- auto-vectorized --
52. for(unsigned int i = 0; i < vect_a.size(); ++ i)
53. {
54. vect_res_plain[i] = vect_a[i] + vect_b[i];
55. }
56.
57. // Defining a compute kernel to encapsulate a specific computation
58. auto kernel_multiply =
59. [&cFixedMultiply] // capture the constant by reference of the scope of the lambda expression
60. (DataArray const& a, DataArray const& b, DataArray & res) // take 3 parameters by reference
61. ->void // lambda function returns void
62. {
63. // simple loop vectorized
64. // -- auto-vectorized --
65. for(unsigned int i = 0; i < a.size(); ++ i)
66. {
67. res[i] = a[i] * b[i] * cFixedMultiply;
68. }
69. };
70.
71. // call the lambda function
72. // this call is autovectorized
73. kernel_multiply (vect_a, vect_b, vect_res_lambda);
74.
75.
76. // This kernel will be called multiple times and performs the quadrature
77. auto kernel_square =
78. [] // capture nothing
79. (double const& a) // take 1 parameter by reference
80. ->double // lambda function returns the square
81. {
82. return (a * a);
83. };
84.
85. // create struct and fill with dummy values
86. Vector3 v3;
87. for (unsigned int i = 0; i < 50 ; ++ i)
88. {
89. v3.add(i * 1.1, i * 2.2, i* 3.3);

```

```

90. }
91.
92. // store the size in a local variable. This is needed to GCG
93. // can estimate the loop iterations
94. const unsigned int size = v3.x.size();
95.
96. // -- auto-vectorized --
97. for (unsigned int i = 0; i < size; ++ i)
98. {
99. v3.distance[i] = sqrt(kernel_square(v3.x[i]) + kernel_square(v3.y[i]) + kernel_square(
100. v3.z[i]));
101. }
102. // output the result, so GCC won't optimize the calculations away
103. std::cout << std::endl << "Computation on std::array" << std::endl;
104. for(unsigned int i = 0; i < vect_a.size(); ++ i)
105. {
106. std::cout << vect_res_plain[i] << std::endl;
107. std::cout << vect_res_lambda[i] << std::endl;
108. }
109.
110. std::cout << std::endl << "Computation on Structure-of-
 Array with variable sized std::vectors" << std::endl;
111. for(unsigned int i = 0; i < v3.x.size(); ++ i)
112. {
113. std::cout << "sqrt(" << v3.x[i] << "^2 + " << v3.y[i] << "^2 + " << v3.z[i] << "^2) = "
114. << v3.distance[i] << std::endl;
115. }
116.
117. return 0;
118. }
```

**SSE (Streaming SIMD Extensions)** 是英特尔在 AMD 的 3D Now 发布一年之后，在其计算机芯片 Pentium III 中引入的指令集，是继 MMX 的扩展指令集。SSE 指令集提供了 70 条新指令。AMD 后来在 Athlon XP 中加入了对这个新指令集的支持。

## SSE2

SSE2 是 Intel 在 Pentium 4 处理器的最初版本中引入的，但是 AMD 后来在 Opteron 和 Athlon 64 处理器中也加入了 SSE2 的支持。SSE2 指令集添加了对 64 位双精度浮点数的支持，以及对整型数据的支持，也就是说这个指令集中所有的 MMX 指令都是多余的了，同时也避免了占用浮点数寄存器。这个指令集还增加了对 CPU 缓存的控制指令。AMD 对它的扩展增加了 8 个 XMM 寄存器，但是需要切换到 64 位模式 (x86-64/AMD64) 才可以使用这些寄存器。Intel 后来在其 Intel 64 架构中也增加了对 x86-64 的支持。

## SSE3

SSE3 是 Intel 在 Pentium 4 处理器的 Prescott 核心中引入的第三代 SIMD 指令集，AMD 在 Athlon 64 的第五个版本，Venice 核心中也加入了 SSE3 的支持。这个指令集扩展的指令包含寄存器的局部位之间的运算，例如高位和低位之间的加减运算；浮点数到整数的转换，以及对超线程技术的支持。

## SSSE3

SSSE3 是 Intel 针对 SSE3 指令集的一次额外扩展，最早内置于 Core 2 Duo 处理器中。

## SSE4

SSE4 是 Intel 在 Penryn 核心的 Core 2 Duo 与 Core 2 Solo 处理器时，新增的 47 条新多媒体指令集，现在 SSE4 版本更新至 SSE4.2。

## SSE4a

AMD 也开发了属于自己的 SSE4a 多媒体指令集，并内置在 Athlon II 与 Opteron 等 K10 架构处理器中，不过 SSE4a 无法与 Intel 的 SSE4 系列指令集兼容。目前 AMD 新一代处理器已支持 Intel 的 SSE4.1、SSE4.2 指令集。

## SSE5

SSE5 是 AMD 为了打破 Intel 垄断在处理器指令集的独霸地位所提出的，SSE5 初期规划将加入超过 100 条新指令，

其中最引人注目的就是三操作数指令（3-Operand Instructions）及熔合乘法累积（Fused Multiply Accumulate）。其中，三操作数指令让处理器可将一个数学或逻辑库，套用到操作数或输入资料。借由增加操作数的数量，一个 x86 指令能处理二至三笔资料，SSE5 允许将多个简单指令汇整成一个指令，达到更有效率的指令处理模式。提升为三运算指令的运算能力，是少数 RISC 架构的水准。熔合乘法累积让允许创建新的指令，有效率地运行各种复杂的运算。熔合乘法累积可结合乘法与加法运算，透过单一指令运行多笔重复计算。透过简化代码，让系统能迅速运行绘图着色、快速照片着色、音场音效，以及复杂向量演算等性能密集的应用作业。目前 AMD 已放弃下一代 Bulldozer 核心内置 SSE5 指令集，改内置 Intel 授权 SSE4 系列指令集。

## AVX

AVX（Advanced Vector Extensions）是 Intel 的 SSE 延伸架构，如 IA16 至 IA32 般的把寄存器 XMM 128bit 提升至 YMM 256bit，以增加一倍的运算效率。

# 二十九. 内核

## 29.1 前置知识

**Linux** 内核的任务:

- 1.从技术层面讲，内核是硬件与软件之间的一个中间层。作用是将应用层序的请求传递给硬件，并充当底层驱动程序，对系统中的各种设备和组件进行寻址。
- 2.从应用程序的层面讲，应用程序与硬件没有联系，只与内核有联系，内核是应用程序知道的层次中的最底层。在实际工作中内核抽象了相关细节。
- 3.内核是一个资源管理程序。负责将可用的共享资源(CPU 时间、磁盘空间、网络连接等)分配得到各个系统进程。
- 4.内核就像一个库，提供了一组面向系统的命令。系统调用对于应用程序来说，就像调用普通函数一样。

**内核实现策略:**

- 1.微内核。最基本的功能由中央内核（微内核）实现。所有其他的功能都委托给一些独立进程，这些进程通过明确定义的通信接口与中心内核通信。
- 2.宏内核。内核的所有代码，包括子系统（如内存管理、文件管理、设备驱动程序）都打包到一个文件中。内核中的每一个函数都可以访问到内核中所有其他部分。目前支持模块的动态装卸(裁剪)。**Linux** 内核就是基于这个策略实现的。

**哪些地方用到了内核机制？**

- 1.进程（在 **cpu** 的虚拟内存中分配地址空间，各个进程的地址空间完全独立；同时执行的进程数最多不超过 **cpu** 数目）之间进行通信，需要使用特定的内核机制。
- 2.进程间切换(同时执行的进程数最多不超过 **cpu** 数目)，也需要用到内核机制。

进程切换也需要像 **FreeRTOS** 任务切换一样保存状态，并将进程置于闲置状态/恢复状态。

3. 进程的调度。确认哪个进程运行多长的时间。

## Linux 进程

1. 采用层次结构，每个进程都依赖于一个父进程。内核启动 `init` 程序作为第一个进程。该进程负责进一步的系统初始化操作。`init` 进程是进程树的根，所有的进程都直接或者间接起源于该进程。
2. 通过 `pstree` 命令查询。实际上得系统第一个进程是 `systemd`，而不是 `init`（这也是疑问点）
3. 系统中每一个进程都有一个唯一标识符(**ID**)，用户（或其他进程）可以使用 **ID** 来访问进程。

## Linux 内核源代码的目录结构

Linux 内核源代码包括三个主要部分：

1. 内核核心代码，包括第 3 章所描述的各个子系统和子模块，以及其它的支撑子系统，例如电源管理、Linux 初始化等
2. 其它非核心代码，例如库文件（因为 Linux 内核是一个自包含的内核，即内核不依赖其它的任何软件，自己就可以编译通过）、固件集合、KVM（虚拟机技术）等
3. 编译脚本、配置文件、帮助文档、版权说明等辅助性文件

使用 `ls` 命令看到的内核源代码的顶层目录结构，具体描述如下。

`include/` ---- 内核头文件，需要提供给外部模块（例如用户空间代码）使用。

`kernel/` ---- Linux 内核的核心代码，包含了 3.2 小节所描述的进程调度子系统，以及和进程调度相关的模块。

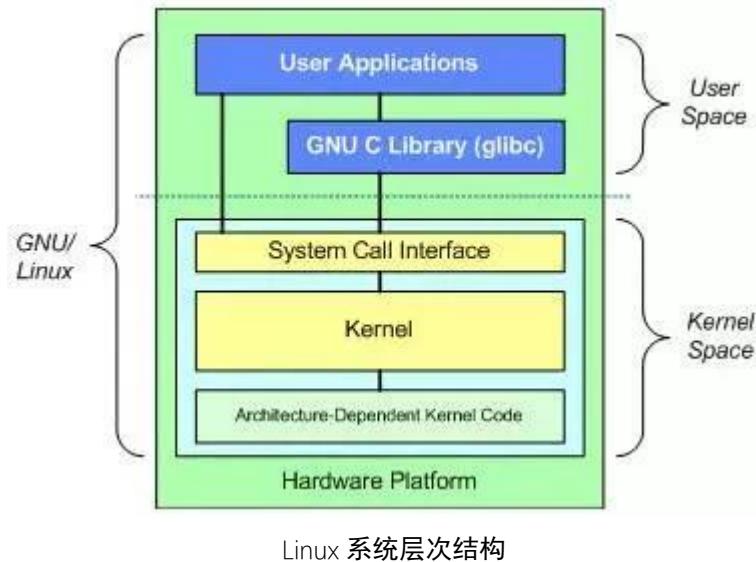
`mm/` ---- 内存管理子系统（3.3 小节）。

`fs/` ---- VFS 子系统（3.4 小节）。

`net/` ---- 不包括网络设备驱动的网络子系统（3.5 小节）。

`ipc/` ---- IPC（进程间通信）子系统。

## 29.2 Linux 内核体系结构简析简析



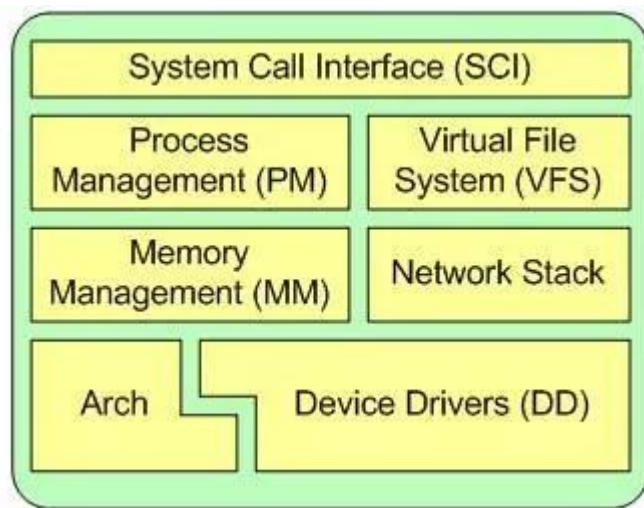
最上面是用户（或应用程序）空间。这是用户应用程序执行的地方。用户空间之下是内核空间，Linux 内核正是位于这里。GNU C Library （glibc）也在这里。它提供了连接内核的系统调用接口，还提供了在用户空间应用程序和内核之间进行转换的机制。这点非常重要，因为内核和用户空间的应用程序使用的是不同的保护地址空间。每个用户空间的进程都使用自己的虚拟地址空间，而内核则占用单独的地址空间。

Linux 内核可以进一步划分成 3 层。最上面是系统调用接口，它实现了一些基本的功能，例如 `read` 和 `write`。系统调用接口之下是内核代码，可以更精确地定义为独立于体系结构的内核代码。这些代码是 Linux 所支持的所有处理器体系结构所通用的。在这些代码之下是依赖于体系结构的代码，构成了通常称为 BSP（Board Support Package）的部分。这些代码用作给定体系结构的处理器和特定于平台的代码。

Linux 内核实现了很多重要的体系结构属性。在或高或低的层次上，内核被划分为多个子系统。Linux 也可以看作是一个整体，因为它会将所有这些基本服务都集成到内核中。这与微内核的体系结构不同，后者会提供一些基本的服务，例如通信、I/O、内存和进程管理，更具体的服务都是插入到微内核层中的。每种内核都有自己的优点，不过这里并不对此进行讨论。

随着时间的流逝，Linux 内核在内存和 CPU 使用方面具有较高的效率，并且非常稳定。但是对于 Linux 来说，最为有趣的是在这种大小和复杂性的前提下，依然具有良好的可移植性。Linux 编译后可在大量处理器和具有不同体系结构约束和需求的平台上运行。一个例子是 Linux 可以在一个具有内存管理单元（MMU）的处理器上运行，也可以在那些不提供 MMU 的处理器上运行。

Linux 内核的 uClinux 移植提供了对非 MMU 的支持。



Linux 内核体系结构

Linux 内核的主要组件有：系统调用接口、进程管理、内存管理、虚拟文件系统、网络堆栈、设备驱动程序、硬件架构的相关代码。

### (1) 系统调用接口

**SCI** 层提供了某些机制执行从用户空间到内核的函数调用。正如前面讨论的一样，这个接口依赖于体系结构，甚至在相同的处理器家族内也是如此。**SCI** 实际上是一个非常有用的函数调用多路复用和多路分解服务。在 `./linux/kernel` 中您可以找到 **SCI** 的实现，并在 `./linux/arch` 中找到依赖于体系结构的部分。

### (2) 进程管理

进程管理的重点是进程的执行。在内核中，这些进程称为线程，代表了单独的处理器虚拟化（线程代码、数据、堆栈和 **CPU** 寄存器）。在用户空间，通常使用进程 这个术语，不过 **Linux** 实现并没有区分这两个概念（进程和线程）。内核通过 **SCI** 提供了一个应用程序编程接口（**API**）来创建一个新进程（`fork`、`exec` 或 `Portable Operating System Interface [POSIX]` 函数），停止进程（`kill`、`exit`），并在它们之间进行通信和同步（`signal` 或者 `POSIX` 机制）。

进程管理还包括处理活动进程之间共享 **CPU** 的需求。内核实现了一种新型的调度算法，不管有多少个线程在竞争 **CPU**，这种算法都可以在固定时间内进行操作。这种算法就称为 **O(1)** 调度程序，这个名字就表示它调度多个线程所使用的时间和调度一个线程所使用的时间是相同的。**O(1)** 调度程序也可以支持多处理器（称为对称多处理器或 **SMP**）。您可以在 `./linux/kernel` 中找到进程管理的源代码，在 `./linux/arch` 中可以找到依赖于体系结构的源代码。

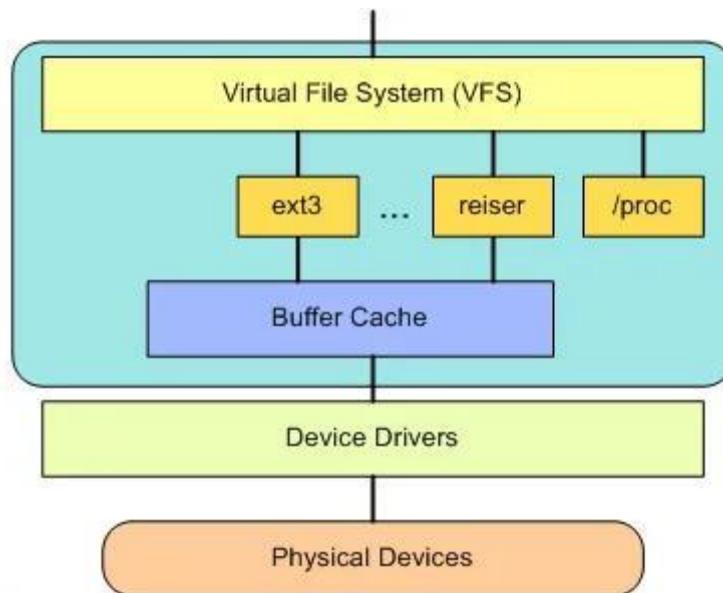
### (3) 内存管理

内核所管理的另外一个重要资源是内存。为了提高效率，如果由硬件管理虚拟内存，内存是按照所谓的内存页 方式进行管理的（对于大部分体系结构来说都是 4KB）。Linux 包括了管理可用内存的方式，以及物理和虚拟映射所使用的硬件机制。不过内存管理要管理的可不止 4KB 缓冲区。

Linux 提供了对 4KB 缓冲区的抽象，例如 slab 分配器。这种内存管理模式使用 4KB 缓冲区为基数，然后从中分配结构，并跟踪内存页使用情况，比如哪些内存页是满的，哪些页面没有完全使用，哪些页面为空。这样就允许该模式根据系统需要来动态调整内存使用。为了支持多个用户使用内存，有时会出现可用内存被消耗光的情况。由于这个原因，页面可以移出内存并放入磁盘中。这个过程称为交换，因为页面会被从内存交换到硬盘上。内存管理的源代码可以在 ./linux/mm 中找到。

#### (4) 虚拟文件系统

虚拟文件系统（VFS）是 Linux 内核中非常有用的一个方面，因为它为文件系统提供了一个通用的接口抽象。VFS 在 SCI 和内核所支持的文件系统之间提供了一个交换层（请参看图 4）。



Linux 文件系统层次结构

在 VFS 上面，是对诸如 open、close、read 和 write 之类的函数的一个通用 API 抽象。在 VFS 下面是文件系统抽象，它定义了上层函数的实现方式。它们是给定文件系统（超过 50 个）的插件。文件系统的源代码可以在 ./linux/fs 中找到。文件系统层之下是缓冲区缓存，它为文件系统层提供了一个通用函数集（与具体文件系统无关）。这个缓存层通过将数据保留一段时间（或者随即预先读取数据以便在需要时就可用）优化了对物理设备的访问。缓冲区缓存之下是设备驱动程序，它实现了特定物理设备的接口。

## (5) 网络堆栈

网络堆栈在设计上遵循模拟协议本身的分层体系结构。回想一下，Internet Protocol (IP) 是传输协议（通常称为传输控制协议或 TCP）下面的核心网络层协议。TCP 上面是 socket 层，它是通过 SCI 进行调用的。socket 层是网络子系统的标准 API，它为各种网络协议提供了一个用户接口。从原始帧访问到 IP 协议数据单元 (PDU)，再到 TCP 和 User Datagram Protocol (UDP)，socket 层提供了一种标准化的方法来管理连接，并在各个终点之间移动数据。内核中网络源代码可以在 `./linux/net` 中找到。

## (6) 设备驱动程序

Linux 内核中有大量代码都在设备驱动程序中，它们能够运转特定的硬件设备。Linux 源码树提供了一个驱动程序子目录，这个目录又进一步划分为各种支持设备，例如 Bluetooth、I2C、serial 等。设备驱动程序的代码可以在 `./linux/drivers` 中找到。

## (7) 依赖体系结构的代码

尽管 Linux 很大程度上独立于所运行的体系结构，但是有些元素则必须考虑体系结构才能正常操作并实现更高效率。`./linux/arch` 子目录定义了内核源代码中依赖于体系结构的部分，其中包含了各种特定于体系结构的子目录（共同组成了 BSP）。对于一个典型的桌面系统来说，使用的是 x86 目录。每个体系结构子目录都包含了很多其他子目录，每个子目录都关注内核中的一个特定方面，例如引导、内核、内存管理等。这些依赖体系结构的代码可以在 `./linux/arch` 中找到。

如果 Linux 内核的可移植性和效率还不够好，Linux 还提供了其他一些特性，它们无法划分到上面的分类中。作为一个生产操作系统和开源软件，Linux 是测试新协议及其增强的良好平台。Linux 支持大量网络协议，包括典型的 TCP/IP，以及高速网络的扩展（大于 1 Gigabit Ethernet [GbE] 和 10 GbE）。Linux 也可以支持诸如流控制传输协议 (SCTP) 之类的协议，它提供了许多比 TCP 更高级的特性（是传输层协议的接替者）。

Linux 还是一个动态内核，支持动态添加或删除软件组件。被称为动态可加载内核模块，它们可以在引导时根据需要（当前特定设备需要这个模块）或在任何时候由用户插入。

Linux 最新的一个增强是可以用作其他操作系统的操作系统（称为系统管理程序）。最近，对内核进行了修改，称为基于内核的虚拟机 (KVM)。这个修改为用户空间启用了一个新的接口，它可以允许其他操作系统在启用了 KVM 的内核之上运行。除了运行 Linux 的其他实例之外，Microsoft Windows 也可以进行虚拟化。唯一的限制是底层处理器必须支持新的虚拟化指令。

# 三十. 内核模块

## 30.1 前置知识

### 什么是 Linux 内核模块?

内核模块是可以根据需要加载到内核中或从内核中卸载的代码块，因此无需重启就可以扩展内核的功能。事实上，除非用户使用类似 `lsmod` 这样的命令来查询模块信息，否则用户不太可能知道内核发生的任何变化。

需要知道的重要一点是，在你的 Linux 系统上总会有很多可用的模块，并且如果你可以深入其中了解到很多细节。`lsmod` 的主要用途之一是在系统不能正常工作时检查模块。然而，大多数情况下，模块会根据需要加载的，而且用户不需要知道它们如何运作。

### 显示内核模块

显示内核模块最简单的方法是使用 `lsmod` 命令。虽然这个命令包含了很多细节，但输出却是非常用户友好。

在上面的输出中：

- Module 显示每个模块的名称
- Size 显示每个模块的大小（并不是它们占的内存大小）
- Used by 显示每个模块被使用的次数和使用它们的模块

## 30.2 相关命令

Linux 提供了几条用于罗列、加载及卸载、测试，以及检查模块状态的命令。

- `depmod` —— 生成 `modules.dep` 和映射文件
- `insmod` —— 一个往 Linux 内核插入模块的程序
- `lsmod` —— 显示 Linux 内核中模块状态
- `modinfo` —— 显示 Linux 内核模块信息
- `modprobe` —— 添加或移除 Linux 内核模块
- `rmmmod` —— 一个从 Linux 内核移除模块的程序

### 显示内置的内核模块

正如前文所说，`lsmod` 命令是显示内核模块最方便的命令。然而，也有其他方式可以显示它们。`modules.builtin` 文件中列出了所有构建在内核中的模块，在 `modprobe` 命令尝试添加文件中的模块时会使用它。注意，以下命令中的 `$(uname -r)` 提供了内核版本的名称。

### 30.3 内核模块可以做什么

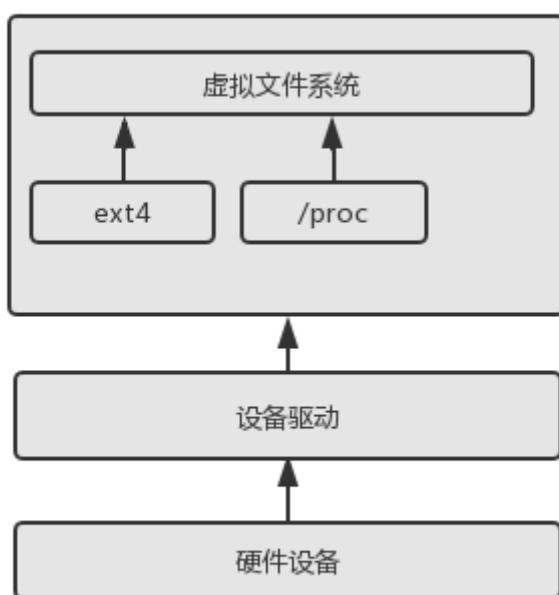
Linux 内核下面的代码几乎都是用内核模块完成开发的，我们换一个角度来看，内核模块看起来就像是一个应用程序的一段代码，这个点我觉得应该很容易理解，但是它有跟应用程序又不一样，我们知道内核空间和用户空间地址段是不一样的，内核模块编写需要非常谨慎，因为稍微不注意就会可能引起系统的崩溃，我这样说大家应该能理解吧，应用空间运行的应用程序有问题，崩溃了没关系，那只是它自己挂掉了，但是内核模块不一样，如果内核模块有调用它的地方，比如音频的 ALSA 驱动，如果崩溃了，那么整个系统看语音部分就不能工作了。

设备驱动使用内核模块编程，就比如一个触摸屏驱动，在设备模型章节我们好好聊下这个事情，但是我们这里就提一下，一个设备需要依赖一个驱动，没有驱动设备是不可能正常工作了，所以内核模块在系统下的驱动都是用内核模块编写的，Linux 内核下面有非常多的设备，当然就需要非常多的驱动。

文件系统驱动，说到设备文件不得不提一句，Linux 下的所有都可以看做是文件，不同的文件系统还有一个统一的虚拟文件系统来管理，这些都是需要内核模块来完成的。

系统调用，用户空间的程序需要调用内核服务，就必须要使用系统调用，正常的系统调用比如关机，读写文件，如果你需要新建一个系统调用也是可以的，那就需要你这个大神自己写个内核模块。

我大概就知道这么多，可能还有些没写出来的，也是内核模块编程范围的，比如网络编程等。



## 30.4 内核模块编写实例

### 1. 编写 .c 和 Makefiles

#### kernel\_lab6.c

```
1. #include <linux/init.h> // 包含了对宏的支持
2. #include <linux/kernel.h> // 包含了内核的一些函数
3. #include <linux/module.h> // 对内核模块机制的一个支持
4.
5. #define DRIVER_AUTHOR "Meng Jianing"
6. #define DRIVER_DESC "Kernel in lab6"
7. #define DRIVER_VERSION "0.01"
8.
9. // 入口函数
10. // __init 宏的作用是告诉编译器，相关的函数或者变量用于初始化。并在初始化结束释放这段内存
11.
12. static int __init lab_init(void)
13. {
14. // 一共有 8 个级别，最低的是类似于一些调试信息 最高的级别是系统警告，比如可能造成系统崩溃的信
15. printk(KERN_ALERT "Hello World\n");
16. return 0;
17. }
18.
19. static void __exit lab_exit(void)
20. {
21. printk(KERN_ALERT "Goodbye World\n");
22. }
23.
24. // 入口点和出口点，用来引导内核
25. module_init(lab_init);
26. module_exit(lab_exit);
27.
28. // 许可证声明
29. MODULE_LICENSE("GPL");
30. MODULE_AUTHOR(DRIVER_AUTHOR);
31. MODULE_DESCRIPTION(DRIVER_DESC);
32. MODULE_VERSION(DRIVER_VERSION);
```

# Makefile

```
1. obj-m += kernel_lab6.o
2. startstop-objs := start.o stop.o
3.
4. CURRENT_PATH:=$(shell pwd)
5. LINUX_KERNEL:=$(shell uname -r)
6. LINUX_KERNEL_PATH:=/usr/src/linux-headers-$(LINUX_KERNEL)
7.
8. all:
9. make -C $(LINUX_KERNEL_PATH) M=$(CURRENT_PATH) modules
10. clean:
11. make -C $(LINUX_KERNEL_PATH) M=$(CURRENT_PATH) clean
```

- 使用 `make` 命令进行编译（为了生成.ko 文件）

```
fox@ubuntu:~/lab/lab6/manyFiles$ ls
kernel_lab6.c Makefile start.c stop.c
fox@ubuntu:~/lab/lab6/manyFiles$ make
make -C /usr/src/linux-headers-4.15.0-122-generic M=/home/fox/lab/lab6/manyFiles modules
make[1]: Entering directory '/usr/src/linux-headers-4.15.0-122-generic'
 CC [M] /home/fox/lab/lab6/manyFiles/kernel_lab6.o
 Building modules, stage 2.
 MODPOST 1 modules
 CC /home/fox/lab/lab6/manyFiles/kernel_lab6.mod.o
 LD [M] /home/fox/lab/lab6/manyFiles/kernel_lab6.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.15.0-122-generic'
fox@ubuntu:~/Lab/lab6/manyFiles$ ls
kernel_lab6.c kernel_lab6.mod.c kernel_lab6.o modules.order start.c
kernel_lab6.ko kernel_lab6.mod.o Makefile Module.symvers stop.c
```

- 安装内核模块: `sudo insmod kernel_lab6.ko`
- 通过 `lsmod` 命令查看当前的内核模块

```
fox@ubuntu:~/lab/lab6/manyFiles$ sudo insmod kernel_lab6.ko
fox@ubuntu:~/lab/lab6/manyFiles$ lsmod
Module size Used by
kernel_lab6 16384 0
rfcomm 77824 0
bnep 20480 2
snd_ens1371 28672 2
```

- 删除内核模块 `sudo rmmod kernel_lab6`
- 通过 `dmesg` 命令来查看系统日志信息

```
[15153.148654] Goodbye World
[15214.624322] Hello World [In Many files]
[15259.856013] Goodbye World [In Many files]
[15943.435979] Hello World [In Many files]
[15991.506594] Goodbye World [In Many files]
```

- 使用 `make clean` 清除编译产生的文件

# 三十一. 内核的安装

## 1. 准备工作

```
1 sudo apt-get update
2 sudo apt-get install build-essential
3 sudo apt-get install libncurses5-dev
4 sudo apt-get build-dep linux-image-$(uname -r)
5 sudo -i
```

## 2. 下载内核

由于加载源代码和解压会导致大量的附加文件，所以最好建立一个单独的工作目录。

```
1 mkdir /usr/local/src/kernel && cd /usr/local/src/kernel
2 apt-get source linux-image-4.4.0-47-generic
```

## 3. 配置内核参数

```
1 # 1. 复制当前机心的配置参数和
2 # 在此配置的基础上进行修改
3 make oldconfig
4 # 2. 拷贝参数
5 make menuconfig
```

## 4. 编译内核

```
1 make
```

## 5. 编译模块 (modules)

```
1 make modules
```

## 7. 安装模块 (modules)

```
1 make modules_install
```

## 8. 安装内核

```
1 make install
```

## 9. 更新启动程序

```
1 update-grub
```

# 三十二. Makefile

## Makefile 简介

在软件开发中，make 通常被视为一种软件构建工具。该工具主要经由读取一种名为“makefile”或“Makefile”的文件来实现软件的自动化建构。它会通过一种被称之为“target”概念来检查相关文件之间的依赖关系，这种依赖关系的检查系统非常简单，主要通过对比文件的修改时间来实现。在大多数情况下，我们主要用它来编译源代码，生成结果代码，然后把结果代码连接起来生成可执行文件或者库文件。

## 优点与缺点

与大多数古老的 Unix 工具一样，make 也分别有着人数众多的拥护者和反对者。它在适应现代大型软件项目方面有着许许多多的问题。但是，依然有很多人坚定地认为（包括我）它能应付绝大多数常见的情况，而且使用非常的简单，功能强大，表达清楚。无论如何，make 如今仍然被用来编译很多完整的操作系统，而且它的那些“更为现代”的替代品们在基本操作上与它没有太大差别。

当然，随着现代的集成开发环境（IDE）的诞生，特别是非 Unix 的平台上，很多程序员不再手动管理依靠关系检查，甚至不用去管哪些文件是这个项目的一部分，而是把这些任务交给了他们的开发环境去做。类似的，很多现代的编程语言有自己专属的、能高效配置依赖关系的方法（譬如 Ant）。

## 主要版本

make 程序经历过各方多次的改写与重写，各方都依据自己的需要做了一些特定的改良。目前市面上主要流行有以下几种版本：

- GNU make:

GNU make 对 make 的标准功能（通过 clean-room 工程）进行了重新改写，并加入作者自认

为值得加入的新功能，常和 GNU 编译系统一起被使用，是大多数 GNU Linux 默认安装的工具。

- BSD make:

该版本是从 Adam de Boor 制作的版本上发展起来的。它在编译目标时有并发计算的能力。主要应用于 FreeBSD, NetBSD 和 OpenBSD 这些系统。

- Microsoft nmake:

该版本主要用于微软的 Windows 系统中，需要注意的是，微软的 nmake 与 Unix 项目中的 nmake 是两种不同的东西，千万不要混淆。

## 从一个简单的例子开始

我们可以用 K&R C 中 4.5 那个例子来做个说明。在这个例子中，我们会看到一份主程序代码 (main.c)、三份函数代码(getop.c、stack.c、getch.c)以及一个头文件(calc.h)。通常情况下，我们需要这样编译它：

```
gcc -o calc main.c getch.c getop.c stack.c
```

如果没有 makefile，在开发+调试程序的过程中，我们就需要不断地重复输入上面这条编译命令，要不就是通过终端的历史功能不停地按上下键来寻找最近执行过的命令。这样做两个缺陷：

1. 一旦终端历史记录被丢失，我们就不得不从头开始；
2. 任何时候只要我们修改了其中一个文件，上述编译命令就会重新编译所有的文件，当文件足够多时这样的编译会非常耗时。

那么 Makefile 又能做什么呢？我们先来看一个最简单的 makefile 文件：

```
calc: main.c getch.c getop.c stack.c
```

```
gcc -o calc main.c getch.c getop.c stack.c
```

现在你看到的就是一个最基本的 Makefile 语句，它主要分成了三个部分，第一行冒号之前的 calc，我们称之为目标（target），被认为是这条语句所要处理的对象，具体到这里就是我们所要编译的这个程序 calc。冒号后面的部分（main.c getch.c getop.c stack.c），我们称之为依赖关系表，也就是编译 calc 所需要的文件，这些文件只要有一个发生了变化，就会触发该语句的第三部分，我们称其为命令部分，相信你也看得出这就是一条编译命令。现在我们只要将上面这两行语句写入一个名为 Makefile 或者 makefile 的文件，然后在终端中输入 make 命令，就会看到它按照我们的设定去编译程序了。

请注意，在第二行的“gcc”命令之前必须要有一个 tab 缩进。语法规定 Makefile 中的任何命令之前都必须要有一个 tab 缩进，否则 make 就会报错。

接下来，让我们来解决一下效率方面的问题，先初步修改一下上面的代码：

```
1 cc = gcc
2 prom = calc
3 src = main.c getch.c getop.c stack.c
4
5 $(prom): $(src)
6 $(cc) -o $(prom) $(src)
```

如你所见，我们在上述代码中定义了三个常量 cc、prom 以及 src。它们分别告诉了 make 我们要使用的编译器、要编译的目标以及源文件。这样一来，今后我们要修改这三者中的任何一项，只需要修改常量的定义即可，而不用再去管后面的代码部分了。

请注意，很多教程将这里的 cc、prom 和 src 称之为变量，个人认为这是不妥当的，因为它们在整个文件的执行过程中并不是可更改的，作用也仅仅是字符串替换而已，非常类似于 C 语言中的宏定义。或者说，事实上它就是一个宏。

但我们现在依然还是没能解决当我们只修改一个文件时就要全部重新编译的问题。而且如果我们修改的是 calc.h 文件，make 就无法察觉到变化了（所以有必要为头文件专门设置一个常量，并将其加

入到依赖关系表中）。下面，我们来想一想如何解决这个问题。考虑到在标准的编译过程中，源文件往往是先被编译成目标文件，然后再由目标文件连接成可执行文件的。我们可以利用这一点来调整一下这些文件之间的依赖关系：

```
1 cc = gcc
2 prom = calc
3 deps = calc.h
4 obj = main.o getch.o getop.o stack.o
5
6 $(prom): $(obj)
7 $(cc) -o $(prom) $(obj)
8
9 main.o: main.c $(deps)
10 $(cc) -c main.c
11
12 getch.o: getch.c $(deps)
13 $(cc) -c getch.c
14
15 getop.o: getop.c $(deps)
16 $(cc) -c getop.c
17
18 stack.o: stack.c $(deps)
19 $(cc) -c stack.c
```

这样一来，上面的问题显然是解决了，但同时我们又让代码变得非常啰嗦，啰嗦往往伴随着低效率，是不祥之兆。经过再度观察，我们发现所有.c都会被编译成相同名称的.o文件。我们可以根据该特点再对其做进一步的简化：

```
1 cc = gcc
2 prom = calc
3 deps = calc.h
4 obj = main.o getch.o getop.o stack.o
5
6 $(prom): $(obj)
7 $(cc) -o $(prom) $(obj)
8
9 %.o: %.c $(deps)
10 $(cc) -c $< -o $@
```

在这里，我们用到了几个特殊的宏。首先是%.o: %.c，这是一个模式规则，表示所有的.o目标都依赖于与它同名的.c文件（当然还有deps中列出的头文件）。再来就是命令部分的 @，其中^），具体到我们这里就是%.c。而\$@代表的是当前语句的目标，即%.o。这样一来，make命令就会自动将所有的.c源文件编译成同名的.o文件。不用我们一项一项去指定了。整个代码自然简洁了许多。

到目前为止，我们已经有了一个不错的 makefile，至少用来维护这个小型工程是没有什么问题了。

当然，如果要进一步增加上面这个项目的可扩展性，我们就会需要用到一些 Makefile 中的伪目标和函数规则了。例如，如果我们想增加自动清理编译结果的功能就可以为其定义一个带伪目标的规则；

```
1 cc = gcc
2 prom = calc
3 deps = calc.h
4 obj = main.o getch.o getop.o stack.o
5
6 $(prom): $(obj)
7 $(cc) -o $(prom) $(obj)
8
9 %.o: %.c $(deps)
10 $(cc) -c $< -o $@
11
12 clean:
13 rm -rf $(obj) $(prom)
```

有了上面最后两行代码，当我们在终端中执行 make clean 命令时，它就会去删除该工程生成的所有编译文件。

另外，如果我们需要往工程中添加一个.c 或.h，可能同时就要再手动为 obj 常量再添加第一个.o 文件，如果这列表很长，代码会非常难看，为此，我们需要用到 Makefile 中的函数，这里我们演示两个：

```
1 cc = gcc
2 prom = calc
3 deps = $(shell find ./ -name "*.h")
4 src = $(shell find ./ -name "*.c")
5 obj = $(src:.c=%.o)
6
7 $(prom): $(obj)
8 $(cc) -o $(prom) $(obj)
9
10 %.o: %.c $(deps)
11 $(cc) -c $< -o $@
12
13 clean:
14 rm -rf $(obj) $(prom)
```

其中， shell 函数主要用于执行 shell 命令， 具体到这里就是找出当前目录下所有的.c 和.h 文件。而 \$(src:.c=.o) 则是一个字符替换函数， 它会将 src 所有的.c 字串替换成.o， 实际上就等于列出了所有.c 文件要编译的结果。有了这两个设定， 无论我们今后在该工程加入多少.c 和.h 文件， Makefile 都能自动将其纳入到工程中来



## 资料来源

(网页). 兄弟连细说 Linux 视频 <https://www.bilibili.com/video/BV1mW411i7Qf?p>

一文看懂 Linux 内核! Linux 内核架构和工作原理详解 <https://blog.csdn.net/jinking01/article/details/104547290>