Chapter 1. 计算机概论

1.1.计算机

1、计算机：接受用户输入指令与数据，经过中央处理器的数据与逻辑单元运算处理后，以产生或存储有用的信息。

1.1.1. 五大单元

1、从外观，分为三个部分

- 输入单元：键盘，鼠标，卡片阅读器，扫描器，手写板，触摸屏等
- 中央处理器(CPU)：含有算数逻辑、控制、记忆等单元
- 输出单元：屏幕、打印机

2、五大单元：

- 输入单元
- 输出单元
- CPU 内部的控制单元
- CPU 内部的算术逻辑单元
- 内存

1.1.2. CPU 种类

1、精简指令集：(Reduced Instruction Set Computing,RISC)

- 微指令集较为精简，每个指令的执行时间都很短，完成的操作也很单纯，指令执行的性能较佳
- 若要做复杂事情，就需要多个指令完成
- Sun 公司的 SPARC 系列；IBM 公司的 Power Architecture 系列和 ARM 系列

2、复杂指令集：(Complex Instruction Set Computer,CICS)

- 每个小指令可以执行一些较低阶的硬件操作，指令数目多而复杂，每条指令的长度不同，因为指令执行较为复杂，所以每条指令花费的时间较长。
- 每条个别指令可以处理的工作较为丰富
- AMD；Intel；VIA 等 x86 架构的 CPU

1.1.3. 接口设备

- 存储设备：硬盘、光盘
- 显示设备：显卡
- 网络设备：网卡

1.1.4. 运作流程

1、类比：

- CPU=大脑
- 内存=大脑中的记录区块
- 硬盘=大脑中的记忆区块
- 主板=神经系统
- 各项接口设备=人体与外界通信的手、脚、皮肤、眼睛

1.1.5. 计算机分类

1、超级计算机：用于国防军事，气象预测，太空科技
2、大型计算机：大型企业的主机
3、迷你计算机
4、工作站
5、微型电脑：个人计算机

1.1.6. 计算机上常用的计算单位

1、大小单位：

- 计算机根据有没有通电来记录信息，所以理论上只认 0 与 1
- 1Byte=8bit
- K=1024 (十进制 1000)
- M=1024K (十进制 1000K)
- G=1024M (十进制 1000M)
- T=1024G (十进制 1000G)
- P=1024T (十进制 1000T)
- 硬盘容量 500G 是以十进制来衡量的，即 500*1000*1000*1000B，当转换成 1024 的时候，就只剩 466G 了

2、速度单位

- CPU 的运算速度常用 MHz 或 GHz 之类的单位。

1.2. 个人计算机框架与接口设备

1.2.1. CPU

1.2.2. 内存

1.2.3. 显卡

1.2.4. 硬盘与存储设备

1、计算机系统上的存储设备包括硬盘、软盘、MO、CD、DVD、磁带机、U 盘(闪存)等。最常见的就是硬盘了

2、物理组成：

- 许多的盘片
- 机械手臂
- 磁头
- 主轴马达
- 实际运行时，主轴马达让盘片转动，然后机械手臂可以伸展让读取头在盘上面进行读写操作。
- 单一盘片容量有限，有的硬盘内部会有两个以上的盘片

3、盘上面的数据

- **整个盘片上好像有多个同心圆绘制出的饼图，而由圆心以放射状的方式分割出磁盘的最小存储单位，即扇区，在物理组成方面，每个扇区大小为 512bytes，这个值是不会改变的。**
- 扇区组成一个圆就可以成为磁道，如果实在多硬盘上面，所有的盘片上面的同一个磁道可以组成一个柱面，**柱面也一般是我们分割硬盘时的最小单位了**

4、计算硬盘的存储容量：

- header 数量 x 每个 header 负责的柱面数量 x 每个柱面所含有的扇区数量 x 扇区的容量
- 单位换算为：header x cylinder/header x sector/cylinder x 512 bytes/sector
- 可简化为：Head x Cylinder x Sector x 512Bytes

Chapter 2. Linux 是什么

1、Linux 的内核原型是 1991 年由托瓦兹(Linus Torvalds)写出来的

2.1. Linux 是什么

2.1.1. Linux 是什么

1、Linux 就是一套操作系统

2、由于不同的硬件的功能函数并不相同，例如 IBM 的 Power CPU 和 Intel x86 架构不一样，所以同一套操作系统是无法在不同的硬件平台上运行。

3、Linux 提供了一个完整的操作系统中最底层的硬件控制与资源管理的完整架构，这个架构是沿袭 Unix 良好的传统而来的，所以相当稳定并且功能强大。

2.1.2. UNIX 的历史

1、早在 Linux 出现之前的 20 年，就有一个相当稳定而且成熟的操作系统 UNIX 存在了。

2、历史：

- 早期计算机架构很难使用，指令周期慢，接口也很麻烦，输入设备只有卡片阅读机，输出设备只有打印机，用户也无法与操作系统互动
- 后来，由于硬件与操作系统的改良，可以使用键盘来输入信息。MIT 开发了**分时操作系统**(Compatible Time-Sharing System,CTSS)，它可以让大型主机通过多个终端以连接进入主机，从而利用主机的资源进行运算工作。
- 兼容分时系统是近代操作系统的鼻祖，它可以让多个用户在某一段时间内分别使用 CPU 的资源(CPU 在每个用户的工作之间进行切换)
- Ken Thompson **以汇编语言**开发了一组内核程序，就是 UNIX 的原型。Thompson 的文件系统有两个重要概念
  - **所有的程序或系统装置都是文件**
  - 不管构建编辑器还是附属文件，所写的程序只有一个目的，就是要有效地完成目标。
- 1973 年，UNIX 正式诞生，Ritchie 等人以 **C 语言**写出第一个正式 UNIX 内核。由于 UNIX 是以较高级的 C 语言编写的，相对于汇编语言需要与硬件有密切的配合，高级的 C 语言与硬件的关系就没有这么大了，因此使得 UNIX 很容易被移植到不同的机器上

Chapter 3. Linux如何学习

Chapter 4. 主机规划与磁盘分区

4.1. Linux 与硬件的搭配

各个组件或设备在 Linux 下面都是一个文件

4.1.1. 认识计算机的硬件配置

1、游戏计算机所需要的配置一定比办公用的工作计算机配置更高端
2、性价比的考虑
3、支持性的考虑

4.1.2. 选择与 Linux 搭配的主机配置

1、CPU：对 CPU 要求不高
2、RAM：内存越高对性能的提升越大
3、Hard Disk：
4、VGA(Video Graphics Array)：对显卡基本无要求，有即可
5、Network Interface Card：网卡是服务器上面最重要的组件
6、光盘、键盘、鼠标：能支持就行

4.1.3. 各硬件设备在 Linux 中的文件名

1、在 Linux 系统中，每个设备都被当成一个文件来对待。
2、IDE 接口的硬盘文件即命名为：/dev/hd[a-d] 括号中为 abcd 任意一个
3、几乎所有的硬件设备文件都在/dev 这个目录内，所以你会看到/dev/hda,/dev/fd0 等文件名。

设备	设备在 Linux 内的文件名
IDE 硬盘	/dev/hd[a-d]
SCSI/SATA/USB 硬盘	/dev/sd[a-p]
U 盘	/dev/sd[a-p]
打印机	25 针：/dev/lp[0-2] USB：/dev/usb/lp[0-15]
鼠标	USB：/dev/usb/mouse[0-15] PS2：/dev/psaux
当前 CD ROM/DVD ROM	/dev/cdrom
当前鼠标	/dev/mouse

4.2. 磁盘分区

1、Linux 的设备都是以文件的类型存在。

4.2.1. 磁盘连接的方式与设备文件名的关系

4.2.2. 磁盘组成的复习

1、磁盘主要由盘片，机械手臂，磁头与主轴马达组成
2、盘片上面又可细分为扇区和柱面两种单位，其中扇区每个为 512bytes
3、每个磁盘的第一个扇区特别重要，它记录了整块磁盘的重要信息

- 主引导分区(Master Boot Record, MBR)：可以安装引导加载程序的地方，

有 446bytes。

- MBR 是很重要的，因为当系统开机的时候回主动去读取这个区块的内容，这样系统才会知道你的程序放在那里并且该如何开机。
- 如果要安装多重引导的系统，MBR 这个区块的管理就非常重要了

➢ 分区表(partition table)：记录整块硬盘分区的状态，有 64bytes

4、基本概念总结

➢ 硬盘最基本的组成部分是由坚硬金属材料制成的涂以磁性介质的**盘片**，不同容量硬盘的盘片数不等

➢ 每个盘片有两面，都可记录信息。盘片被分成许多扇形的区域，每个区域叫一个**扇区**，每个扇区可存储 128×2 的 N 次方(N＝0.1.2.3)字节信息。在 DOS 中每扇区是 128×2 的 2 次方＝512 字节

➢ 盘片表面上以盘片中心为圆心，不同半径的同心圆称为**磁道**

➢ 硬盘中，不同盘片相同半径的磁道所组成的圆柱称为**柱面**(一个圆柱体的侧表面称为柱面，当磁头固定不动，盘面转动时，磁头在盘面上勾画出一个同心圆，叫做磁道，上下重叠盘面的磁道构成了一个柱面。)。磁道与柱面都是表示不同半径的圆，在许多场合，磁道和柱面可以互换使用，我们知道，每个磁盘有两个面，每个面都有一个磁头，习惯用磁头号来区分

➢ 当磁头改变位置时，随着盘面的转动，磁头在盘面上勾画出了半径不同的磁道，这些磁道构成了半径不同柱面，而每一个半径不同的柱面可用**柱面号**来标识。

4. 2. 3. 磁盘分区表

1、柱面是文件系统的最小单位，也就是分区的最小单位。

2、我们就是利用参考柱面号码的方式来处理，在分区表所在的 64bytes 容量中，总共分为四组记录区，每组记录区记录了该区段的起始与结束的柱面号码

3、重点信息：

➢ 其实所谓的"分区"只是针对那个 64bytes 的分区表进行设置而已

➢ 硬盘默认的分区表仅能写入四组分区信息

➢ **这四组分区信息我们称为主(Primary)或扩展(Extended)分区**

➢ 分区的最小单位为柱面

4、当系统要写入磁盘时，一定会参考磁盘分区表，才能针对某个分区进行数据的处理。

5、分区的原因：

➢ 数据的安全性：善用分区，可以让数据更安全

➢ 系统的性能考虑：由于分区将数据集中在某个柱面的分段，由于分区将数据集中了，磁盘只会搜索对应书面范围内的数据，提高了数据读取的速度与性能。

6、利用额外的扇区来记录更多的分区信息

➢ **扩展分区的目的是使用额外的扇区来记录分区信息，扩展分区本身并不能拿来格式化**

➢ **由扩展分区继续切出来的分区称为逻辑分区，所有逻辑分区的总和等于扩展分区的总和，因此在分配扩展分区的时候尽可能使用剩余的所有容量，否则那些剩余容量是无法再利用的**

7、注意点：

- **主分区与扩展分区最多可以有 4 个(硬盘的限制)**
- **扩展分区最多只能有一个(操作系统的限制)**
- **逻辑分区是由扩展分区继续切割出来的分区**
- 能够被格式化后作为数据访问的分区为主分区和逻辑分区，扩展分区无法格式化
- 逻辑分区的数量依据操作系统而不同
  - 在 Linux 系统中，IDE 硬盘最多有 59 个逻辑分区(5-63 号)
  - 在 Linux 系统中，SATA 硬盘可以有 11 个逻辑分区(5-15 号)
- 如果扩展分区被破坏，所有逻辑分区将会被删除

4.2.4. 开机流程与主引导分区(MBR)

1、问题引入：

- 没有执行软件的硬件是无用的
- 为了计算机系统的资源合理分配，因此有了操作系统这个系统级软件
- 由于操作系统会控制所有的硬件并且提供内核功能，因此我们的计算机就能够认识硬盘内的文件系统，并且进一步读取硬盘的软件文件与执行该软件来完成各项软件的执行目的。
- **既然操作系统也是软件，那么计算机又是如何识别这个操作系统软件并执行它呢？这就要涉及到开机程序了**

2、BIOS 与 CMOS

- CMOS：记录各项硬件参数切嵌入在主板上面的存储器
- BIOS：一个写入到主板上的韧体，韧体就是写入到硬件上的一个软件程序，BIOS 就是在开机的时候计算机系统会主动执行的第一个程序

3、开机流程：

- BIOS：开机主动执行的韧体，会认识第一个可开机的设备
- MBR：第一个可开机的设备的第一个扇区内的主引导分区块，包含引导加载程序
- 引导加载程序：一支可读取内核文件来执行的软件
- 内核文件：开始操作系统的功能

4、Boot loader：

- 操作系统安装在 MBR 上面的一套软件，由于 MBR 只有 446bytes，因此这个引导加载程序是非常小而完美的。
- bootloader 的主要任务：
  - 提供菜单：用户可以选择不同的开机选项，这也是多重引导的重要功能
  - 载入内核文件：直接指向可开机程序区段来开始操作系统
  - 转交给其他 loader：将引导加载功能转交给其他 loader 负责(引导加载程序还可以安装在每个分区的引导扇区)

5、总结：

- 每个分区都有自己的启动扇区(boot sector)
- 实际可开机的内核文件是放置到各个分区内的
- loader 只会认识自己的系统分区内可开机的内核文件，以及其他的 loader

而已

- loader 可以直接指向或者间接将管理权转交给另一个管理程序

6、为什么要先装 Windows 再装 Linux

- Linux 在安装的时候，可以选择将引导加载程序安装在 MBR 或个别分区的启动扇区，而且 Linux 的 loader 可以手动设置菜单，所以可以在 Linux 的 boot loader 里面加入 Windows 开机的选项
- Windows 在安装的时候，它的安装程序会主动覆盖掉 MBR 以及自己所在分区的启动扇区，没有机会选择，而且没有让我们自己选择菜单的功能。

4.2.5. Linux 安装模式下，磁盘分区的选择(极重要)

1、目录结构：

- 目录结构就是以根目录为主，然后向下呈现分支状的目录结构的一种文件结构。
- **整个目录结构最重要的就是根目录，这个根目录的表示方法为一条斜线'/'**
- 所有的文件都是由根目录"/衍生而来的

2、文件系统与目录树的关系(挂载)

- **所谓"挂载"就是利用一个目录当成进入点，将磁盘分区的数据放在该目录下，也就是说进入该目录就可以读取该分区的意思。**
- 由于 Linux 系统最重要的是根目录，因此根目录一定要挂载到某个分区中
- 其他的目录可以根据用户自己的需求给予挂载到不同的分区。
- 判断某文件在哪个分区(partition)，通过反向追踪即可，哪个"进入点"先被查到，那就是使用的进入点了。

Chapter 5. 安装 CentOS

1、首先，要分区(已有压缩卷上点压缩即可)，得到一个未使用的分区

2、然后，制作 U 盘启动(将 U 盘盘符进行修改，改成一个简单的字符，比如 H)，否则会出现"**/dev/root does not exits**"

3、其次，选择非 UTFI 方式从 U 盘启动，如果从 UTFI 方式从 U 盘启动，会在分区完后，出现"**No valid bootloader target device found See below for details. For a UEFI installation, you must include an EFI System Partition on a GPT-formatted disk, mounted at /boot/efi**"

4、按 TAB 将 CentOS 7 x86_64…那一串改成 H(即你改的 U 盘盘符名字)

Chapter 6. 首次登陆与在线求出 man page

6.1. 首次登陆系统

6.1.1. 首次登陆 centOS 7 图形界面

1、有登陆与离开状态：因为在 Linux 系统中，由于是多人多任务的环境，所有系统随时都有很多任务在进行，因此正确地开关机是很重要的

2、不正常的关机可能会导致文件系统错乱，造成数据损毁

3、通常，Linux 主机都会加挂一个不断电系统

6.1.2. GNOME 的操作与注销

1、GNOME 是一套纯粹自由的计算机软件

2、由于 Linux 是多用户多任务的操作系统，每个人都会有自己的"工作目录"，这个目录是用户可以完全掌握的，所以就称为"用户个人主文件夹"，一般来说，主文件夹都在/home 下面

3、注销：注销并不是关机，只是让你的账号离开系统而已。

6.1.3. KDE 的操作与注销

6.1.4. X Window 与命令行模式的切换

1、Linux 默认的情况下会提供 6 个 Terminal 来让用户登陆，切换的方式为：[Ctrl]+[Atl]+[F1]~[F6]的组合按钮

2、系统会将[F1]~[F6]命名为：tty1~tty6 的操作界面环境。tty:Teletype

3、Linux 默认的登陆模式中，主要分为两种：

- **一种是仅有纯文本界面(所谓运行等级 run level 3)的登陆环境**，在这种环境下，**你可以有 tty1~tty6 的终端界面**，但是并没有图形窗口界面的环境
- 另一种是图形界面的登陆环境(所谓运行等级 run level 5)，在这个环境中，你具有 tty1~tty7。其中 tty7 就是开机玩后默认等待登陆的图形环境
- 如果以纯文本的环境启动 Linux，默认的 tty7 是没有东西的，在这种情况下，可以在 tty1~tty6 的任意一个终端界面使用你的账号登陆后，然后执行命令："startx"
- 要让 startx 生效，需要保证：
  - tty7 没有其他的窗口软件正在运行(即 tty7 必须是空闲的)
  - 你必须已经安装了 X window 系统，并且 X Server 是能够顺利启动的
  - 最好有窗口管理员，例如 GNOME/KDE
  - 启动 X 窗口所需要的服务必须要先启动

- 对于 CentOS7
  - **默认图形化登陆终端界面为:tty1**
  - **纯文本界面为 tty2~tty6**

6.1.5. 在终端界面登陆 linux

1、www login：输入账号，尽量不要用 root，因为 root 有无穷的权利

2、Password：输入密码，输入密码的时候，屏幕不会显示任何字样

3、[username@www ~]$ ：

- ~：代表的是一个变量，root 的主文件夹是/root，那么~就代表/root；hcf 的主文件夹是/home/hcf，那么~就代表/home/hcf
- root 的命令提示符是：#
- 一般用户的命令提示符是：$

4、exit：注销

6.2. 在命令行模式下执行命令

1、窗口管理员或命令行模式都是一组或一支在负责我们所想要完成的命令。

2、命令行模式登陆后所取得的程序被称为 shell，因为这个程序负责最外层的跟用户(我们)通信工作，所以才被戏称为 shell(贝壳)。

6.2.1. 开始执行命令

1、格式：

- [hcf@www ~]$ command [-options] parameter1 parameter2 ...
- 一行命令中第一个输入部分是"命令",或"可执行文件"
- 中括号[]并不存在于实际的命令中，加入参数设置时，通常参数前会带'-'号，有时候会使用参数的完整全名，则参数前带有'--'符号
- parameter12...为依附在 option 后面的参数，或者是 command 的参数
- 命令、-options、参数等以空格区分，不论空几格，shell 都视为一格
- 按下[Enter]键后，该命令就立即执行，[Enter]代表着一行命令的启动
- 命令太长的时候可以使用\[Enter]来转义[Enter]，使命令连续到下一行
- **shell 区分大小写**

2、语言的支援<LANG>

- echo $LANG：显示目前所支持的语言
- LANG=en_US：修改为英文语系
- CentOS7 中，配置 Linux 语言的文件为/etc/locale.conf，通过<localectl>命令进行设置
  - systemd 服务在**启动的时候**读取区域配置文件，完成系统的设置
  - localectl
  - localectl set-locale LANG=zh_CN.utf8
  - localectl status
  - localectl list-locales |grep CN

6.2.2. 基础命令的操作

1、<date>：显示日期与时间的命令

- date <==2016 年 11 月 14 日 星期一 13 时 24 分 00 秒 CST
- date +%Y <==2016
- date +%y <==16
- date +%M <==25(minits 的意思)
- date +%m <==11(month 的意思)
- date +%D <== 11/14/16
- date +%d <==14(day)
- date +%H <==13(hour)
- date +%Y/%m/%d：

- date +%H:%M：

2、<cal>：显示日历的命令
- cal [[month] year]
- cal 2009
- cal 5 12 2009

3、<bc>：简单好用的计算器
- scale=3：设置输出小数点的位数
- quit：离开 bc 回到命令提示符

6.2.3. 重要的热键

1、[Tab]按键
- 命令补全，接在一串命令的第一个命令的后面，则为"命令补全"
- 文件补齐，接在一串命令的第二个命令以后时，则为"文件补齐"

2、[Ctrl]-c 按键
- 中断目前程序

3、[Ctrl]-d
- 代表键盘输入结束
- 想要直接离开文字界面，直接按下[Ctrl]-d 就能够直接离开了。

6.2.4. 错误信息的查看

6.3. Linux 系统的在线求助 man page 与 info page

6.3.1. 〈man〉page

1、用法
- man command
- q：离开 man 的环境
- 代号：

代号	代表内容
1	用户在 shell 环境中可以造作的命令或可执行文件
2	系统内核可调用的函数和工具
3	一些常用的函数与函数库，大部分为 C 的函数库(lic)
4	设备文件的说明，通常在/dev 下的文件
5	配置文件或者是某些文件的格式
6	游戏
7	惯例与协议等
8	系统管理员可用的管理命令
9	跟 kernel 有关的文件
上述内容可以通过命令"man 7 man"来取得更详细的内容	

- man page 的组成

代号	内容说明
NAME	简短的命令、数据名称说明
SYNOPSIS	剪短的命令执行语法(syntax)简介

DESCRIPTION	较为完整的说明，最好细看
OPTIONS	针对 SYNOPSIS 部分中，有列举的所有可用选项说明
COMMANDS	当这个程序在执行时，可以在此程序中执行的命令
FILES	这个程序或数据所使用或参考或连接到的某些文件
SEE ALSO	这个命令或数据具有相关的其他说明
EXAMPLE	一些可以参考的实例
BUGS	是否有相关的错误

- man -f [commandname]：找出与 commandname 完全匹配的命令
- man -k [commandname]：找出包含 commandname 的命令

6.3.2. <info> page

1、用法

- info pate 将文件数据拆成一个个段落，每个段落用自己的页面来撰写
- 在各个页面中还有类似网页的"超链接"来跳跃到不同的页面中
- 每个页面也被称为一个独立的节点

按键	进行工作
B	移动光标到该 info 界面当中的第一个节点处
E	移动光标到该 info 界面当中的最后一个节点处
N	前往下一个节点处
P	前往上一个节点处
U	向上移动一层
S(/)	在 info page 中进行查询
H	显示求助菜单
?	命令一览表
Q	结束这次的 info page

6.4. 超简单文本编辑器：nano

6.5. 正确的关机方法

1、原因：

- 你看不到的屏幕背后其实可能有许多人同时在你的主机上面工作。
- 若不正常关机，则可能造成文件系统的损毁

2、查看系统的使用状态：

- who：查看目前有谁在线
- netstat -a：查看网络的联机状态
- ps -aux：看后台执行的程序

3、通知在线用户关机的时刻

- <sync>：将数据同步写入硬盘中的命令
- <shutdown>：惯用的关机命令
- <reboot>/<halt>/<poweroff>：重启、关机

4、切换执行等级

- init 0：关机
- init 3：纯命令行模式

- init 5：含有图形界面模式
- init 6：重启

Chapter 7. Linux 的文件权限与目录配置

7.1. 用户与用户组

1、文件所有者：(Owner)

- 可以修改文件的权限设置

2、用户组概念：(Group)

- 对文件权限进行设置后，可以使得组内的用户可以看到文件数据，而组外的用户不能看到文件数据
- 每个账号都可以有多个用户组的支持
- 设置用户组，好让组内用户进行分享

3、其他人的概念：(Others)

4、Linux 用户身份与用户组记录的文件

- 默认情况下所有的系统上的账号与一般用户身份，还有 root 的相关信息都是记录在/etc/passwd 文件内。
- 个人密码记录在/etc/shadow 文件内
- 此外 Linux 所有组名都记录在/etc/group 文件内

7.2. Linux 文件权限概念

7.2.1. Linux 文件属性

1、如何查看属性：<ls>

- ls -al：ls 是 list 的意思，参数-al 表示列出所有文件详细的权限与属性(其中 a 表示包含隐藏文件，就是文件名第一个字符是.的文件)

2、显示的事例：drwxr-xr-x 5 HCF staff 170 7 17 19:01 Linux

- 第一列代表类型与权限
  - **第一个字符代表：这个文件是"目录、文件、或链接文件等等"**
    - [d]：目录
    - [-]：文件
    - [l]：链接文件
    - [b]：设备文件里面可供存储的接口设备
    - [c]：表示设备文件里面的串行端口设备，例如键盘、鼠标
  - **接下来的字符中，以三个为一组，均为"rwx"的 3 个参数的组合，其中[r]代表可读，[w]代表可写，[w]代表可执行**
    - 第一组：文件所有者的权限
    - 第二组：同组用户的权限
    - 第三组：其他非本用户组的权限
- 第二列代表：有多少文件名连接到此节点
- 第三列代表：**这个文件(或目录)的所有者的账号**
- 第四列代表：**这个文件的所属用户组**
- 第五列代表：这个文件的容量大小(单位 B)
- 第六列代表：这个文件的创建文件日期或者是最近的修改日期
- 第七列代表：文件名

7.2.2. 如何改变文件属性与权限

1、命令：

- chgrp：change group
- chown：change onwer
- chmod：change ?

2、<chgrp>：改变所属用户组
- 格式：chgrp [groupName] [fileName]
- 要被改变的组名必须要在/etc/group 文件内才行，否则就会显示错误

3、<chown>：改变所有者
- 格式：
  - chown [-R] [userName] [fileName] //[-R]可不加
  - chown [-R] [userName]:[groupName] [fileName] //[-R]可不加
  - chown [-R] [userName].[groupName] [fileNmae] **//最好不用'.'而用':'**
  - -R 参数表示：遍历指定目录下的所有子目录和文件
- 用户必须是已经存在于系统中的账号，也就是在/etc/passwd 这个文件中有记录的用户名称才能改变。
- 通途：
  - 例如，cp 命令会拷贝文件，连同所有属性一起拷贝，因此需要用 chown 或 chgrp 来改变权限及属性

4、<chmod>：改变权限
- 数字类型改变权限：
  - 用数字来代表各个权限
    - r:4
    - w:2
    - x:1
    - 每种身份的分数是各自的三个权限(r、w、x)分数的累加
  - chmod [-R] xyz //xyz 代表三个身份的分数
  - -R：进行递归的持续更改，即连同子目录下的所有文件都会更改
- 符号类型改变文件权限：

chmod	u (User) g (Group) o (Others) a(代表所有身份)	+ (加入) - (除去) = (设置)	r w x	文件或目录

- 例子：
  - chmod u=rwx,go=rx [filename]
  - chmod a-x [filename]

7. 2. 3. 目录与文件的权限意义

1、权限对文件的重要性
- 文件时实际含有数据的地方，包括一般文本文件，数据库内容文件，二进制可执行文件
- r(read)：可读取此文件的实际内容，如读取文本文件的文字内容
- w(write)：可以编辑，新增或者是修改该文件的内容**(不包含删除该文件！)**
- x(execute)：该文件具有可以被系统执行的权限
  - 在 Window 下面的一个文件是否具有执行的能力是通过"扩展名"来判断

的，例如.exe, .bat, .com 等。
    - 在 Linux 下面，文件是否能被执行是由是否具有"x"这个权限来决定的，而根文件名是没有绝对的关系
  - 对于 r、w、x 来说，主要都是针对"文件内容"而言，与文件名的存在与否并没有关系。

2、权限对于**目录**的重要性

- 目录的主要内容是记录文件名列表，文件名与目录具有强烈的关联
- r(read contents in directory)：表示具有读取目录结构列表的权限，表示你可以查询该目录下的文件名数据，因此可以用 ls 命令将目录的内容列表显示出来**(只会显示文件名，详细信息是无法显示的)**
- w(modify contents of directory)：它表示你具有更改该目录结构列表的权限
  - 新建新的文件与目录 (mkdir：新建目录), (touch：新建空文件)
  - 删除已经存在的文件与目录 (rm)
  - 将已存在的文件或目录进行重命名
  - 转移该目录内的文件、目录位置
- x(access directory)：目录的 x 代表的是用户是否能进入该目录称为工作目录的用途。
  - 即能不能进入某一个目录只与该目录的 x 权限有关。
  - 如果你在该目录下不仅有 x 权限，那么你就无法切换到该目录下，也就无法执行该目录下的任何命令，即使你具有该目录的 r 权限
  - 因此要开放目录给任何人浏览时，应该至少也要给予 r 以及 x 的权限

7.2.4. Linux 文件种类与扩展名

1、文件种类：

- 普通文件：第一个字符为[-]，依据内容可分为：
  - 纯文本文件(ASCII)
  - 二进制文件(binary)
  - 数据格式文件(data)
- 目录：第一个字符为[d]
- 链接文件：第一个字符为[l]
- 设备与设备文件：通常集中在/dev 目录中，通常分为两种：
  - 块(block)设备文件：就是一些存储数据，以提供系统随机访问的接口设备，第一个字符为[b]
  - 字符(character)设备文件：一些串行端口的接口设备，例如键盘，鼠标，特征是"一次性读取，不能截断输出"，第一个字符为[c]
- 套接字(sockets)：第一个字符为[s]
  - 被称为数据接口文件
  - 通常被用在网络上的数据连接
- 管道(FIFO,pipe)：第一个字符为[p]

2、Linux 文件扩展名

- 一个 Linux 文件内不能被执行，与它的第一列的 10 个属性有关，与文件名根本一点关系都没有

- 可以被执行跟可以执行成功是不同的
- **基本上 Linux 系统上的文件名只是为了让你了解该文件可能的用途而已，真正的执行与否仍然需要权限的规范**

3、Linux 文件长度规范

- 在 Linux 下，使用默认的 Ext4 文件系统时，针对文件的文件名长度：
  - 单一文件或目录的最大容许文件名为 255 个字符
  - 包含完整路径名称及目录(/)的完整文件名为 4096 个字符

4、Linux 文件名的限制

- 尽量避免使用* ? > < ; & ! [ ] | \ ' " ' ( ) { }这些字符来命名文件或目录

7.3. Linux 目录配置

7.3.1. Linux 目录标准配置：FHS

1、FHS：Filesystem Hierarchy Standard

	可分享的(shareable)	不可分享的(unshareable)
不可变的(static)	/usr(软件放置处)	/etc(配置文件)
	/opt(第三方软件)	/boot(开机与内核文件)
可变动的(variable)	/var/mail(用户邮件信箱)	/var/run(程序相关)
	/var.spool/news(新闻组)	/var/lock(程序相关)

2、FHS 针对目录树架构仅定义出三层目录下面应该放置什么数据：

- /(root,根目录)：与开机系统有关
- /usr(**UNIX software resource**)：与软件安装/执行有关
- /var(variable)：与系统运作过程有关

3、root 在 Linux 中的意义有很多

- 以"账号"的角度来看：root 代表"系统管理员"的身份
- 以"目录"的角度来看，root 代表根目录，即/

4、根目录(/)的意义与内容：

- 根目录是整个系统最重要的目录，因为不但所有目录都是由目录衍生出来的，同时根目录也与开机、还原、系统修复等操作有关。
- **根目录不要放在非常大的分区内，因为越大的分区你会放入的越多的数据，如此一来根目录所在的分区就可能会有较多发生错误的机会**
- 建议根目录(/)越小越好，
- 应用程序所安装的软件最好不要与根目录放在同一个分区内。如此不但性能好，而且根目录所在的文件系统也不容易发生问题。

5、根目录下的子目录：**(注意子目录并不意味着与根目录处于同一个分区，理解挂载的含义)**

- /bin：
  - 系统中有很多放置执行文件的目录，但是/bin 比较特殊，因为/bin 放置的是在单用户维护模式下还能被操作的命令。
  - 在/bin 下面的命令可以被 root 和一般账号所使用，主要有：cat，chmod，chown，date，mv，mkdir，cp，bash 等常用命令
- /boot
  - 主要放置开机会用到的文件，包括 Linux 内核文件以及开机菜单与开机

所需配置文件等

- /dev
  - 在 Linux 系统上，任何设备与接口设备都是以文件的形式存在于这个目录当中的。
  - 访问这个目录下的文件，就等于访问某个设备
  - 重要的文件有：/dev/null, /dev/zero, /dev/tty, /dev/lp*, /dev/hd*, /dev/sd*
- /etc
  - 系统的主要配置文件几乎都放置在这个文件中，比如用户的账号密码，各种服务的起始文件等
  - 一般来说，这个目录下的文件属性是可以让一般用户查阅的，但是只有 root 才有权利修改
  - FHS 建议不要放置可执行文件在这个目录中
- /home
  - 系统默认的用户主文件夹
- /lib
  - 放置在开机时会用到的函数库，以及在/bin 或/sbin 下面的命令会调用的函数库
- /media
  - 放置可删除设备，包括光盘，DVD 等都暂时挂载于此
- /mnt
  - 暂时挂载某些额外的设备
- /opt
  - 给第三方软件放置的目录
- /root
  - 系统管理员(root)的主文件夹
  - 之所以放在这里是因为如果进入单用户维护模式而仅挂载根目录时，该目录就能够拥有 root 的主文件夹，所以我们会希望 root 的主文件夹与根目录放置在同一个分区中
- /sbin
  - 放置开机过程中需要的，里面包括了开机、修复、还原系统所需要的命令
- /srv
  - 放置服务所需要取用的数据
- /tmp
  - 让一般用户或者正在执行的程序暂时放置文件的地方，任何人都能访问。
  - 需要定期清理，FHS 建议开机时将/tmp 的数据都删除

6、必须与根目录处于一个分区的目录

- /etc：配置文件
- /bin：重要执行文件
- /dev：所需要的设备文件
- /lib：执行文件所需的函数库与内核所需的模块
- /sbin：重要的系统执行文件

7.3.2. 目录树

1、主要特性：

- 目录树的起始点为根目录(/, root)
- 每一个目录不仅能使用本地端的文件系统，也可以使用网络上的文件系统
- 每一个文件在此目录中的文件名(包含完整路径)都是独一无二的

7.3.3. 绝对路径与相对路径

1、绝对路径：由根目录(/)开始写起的文件名或目录名称

2、相对路径：相对于目前路径的写法，开头不是(/)就属于相对路径

- . ：代表当前目录，也可以用./来表示
- .. ：代表上一层目录，也可以用../来表示

Chapter 8. Linux 文件与目录管理

8.1. 目录与路径

8.1.1. 相对路径与绝对路径

1、概念：

- 绝对路径：路径的写法一定由根目录/写起
- 相对路径：路径的写法不是由/写起
- 对于文件名的正确性来说，绝对路径的正确性要比较好，在写程序来管理系统的条件下，务必使用绝对路径的写法。

8.1.2. 目录的相关操作

1、重要的特殊目录：

- . 代表此层目录
- .. 代表上层目录
- - 代表前一个工作目录
- ~ 代表"目前用户身份"所在的主文件夹
- ~account 代表 account 这个用于的主文件夹(account 代表账号名)
- 所有的目录下面都会存在两个目录，分别是'.'与'..'

2、常见的处理目录的命令：

- cd：切换目录
- pwd：显示当前目录
- mkdir：新建一个新的目录
- rmdir：删除一个空的目录

3、<cd>：(Change Directory)

- cd [相对路径或绝对路径]

4、<pwd>：(Print Working Directory)

- pwd [-P]
- -P：显示当前的路径，而非连接(link)路径

5、<mkdir>：(Make Directory)

- mkdir [-mp] 目录名称
- -m：配置文件的权限
- -p：帮助你直接将所需的目录(包含上层目录)递归地创建起来

6、<rmdir>：(Remove Directory)

- rmdir [-p] 目录名称
- -p：连同上层"空的"目录一起删除
- 只能删除空目录

8.1.3. 关于执行文件路径的变量：$PATH

- PATH(一定是大写)这个变量的内容是由一堆目录组成，每个目录中间用冒号(:)来隔开，每个目录有"顺序"之分
- 添加路径：PATH="$PATH":[文件路径]
- 存在多个同名命令：PATH 中那个目录先被查询，则那个目录下的命令就先被执行
- 最好不要把'.'添加到 PATH 中

8.2. 文件与目录管理

8.2.1. 查看文件与目录：<ls>

1、格式：

- ls [-aAdfFhilnrRSt] 目录名称...
- ls [--color={never,auto,always}] 目录名称...
- ls [--full-time] 目录名称...

2、参数

- **-a：全部的文件，包括隐藏文件**
- -A：全部文件，包括隐藏文件，但是不包括'.'与'..'这两个目录
- **-d：仅列出当前目录本身，而不是列出目录内的文件数据**
- -f：直接列出结果，而不进行排序(ls 默认会以文件名排序)
- -F：根据文件、目录等信息基于附加数据结构，例如
  - *:代表可执行文件
  - /:代表目录
  - =:代表 socket 文件
  - |:代表 FIFO 文件
- **-h：将文件容量以人类较易读懂的方式(例如 GB,KB 等)**
- -i：列出 inode 号码
- **-l：列出长串数据，包含文件的属性与权限等数据**
- -n：列出 UID 与 GID，而非用户与用户组的名称
- -r：将排序结果反向输出，例如，原本文件名由小到大，反向则为由大到小
- -R：联通子目录内容一起列出来，等于该目录下的所有文件都会显示出来
- -S：以文件容量大小排序，而不是用文件名排序
- -t：以时间排序，而不是用文件名
- --color=never：不要依据文件特性给予颜色显示
- --color=always：显示颜色
- --color=auto：让系统自行依据设置来判断是否给予颜色
- --full-time：以完整时间模式(包含，年月日分)输出
- --time={atime,ctime}：输出访问时间或改变属性时间，而非内容更改时间

8.2.2. 复制、删除和移动：cp、rm、mv

1、<cp>：(copy)

- cp [-adfilprsu] source destination
- cp [options] source1 source2 ... directory
- **-a：相当于-pdr 的意思**
- -d：若源文件为链接属性(link file)，则复制链接文件属性而非文件本身
- -f：强制(force)的意思，若目标文件已经存在且无法开启，则删除后再尝试一次
- **-i：若目标文件已经存在，在覆盖时会先询问操作的进行**
- -l：进行硬链接(hard link)的连接文件创建，而非复制文件本身
- -p：连同文件的属性一起复制过去，而非使用默认属性(备份常用)

- **-r：递归持续复制，用于目录的复制行为**
- -s：复制成为符号链接文件(symbolic link，**软连接**)，即"快捷方式"文件
- -u：若 destination 比 source 旧时才更新 destination
- **如果源文件有两个以上，则最后一个目的文件一定要是"目录"才行**
- 复制别人的数据，必须要有对该文件的 read 权限
- 在默认条件中，cp 源文件与目的文件的权限是不同的，目的文件的所有者通常会是命令操作者本身；由于这个特性，在进行备份的时候，某些需要特别注意的特殊权限文件，例如密码文件(/etc/shadow)以及一些配置文件，就不能直接以 cp 来复制，而必须要加上-a 或者是-p 才能完整复制文件权限的参数才行。
- 另外如果想要复制文件给其他用户，必须要注意文件的权限，否则其他人是无法针对你给予的文件进行修订操作的

2、<rm>：(Remove 移除文件或目录)

- rm [-fir] 文件或目录
- -f：就是 force 的意思，忽略不存在的文件，不会出现警告信息
- -i：互动模式，在删除前会询问用户是否操作
- **-r：递归删除，最常用在目录的删除，这是非常危险的参数**

3、<mv>：(Move 移动文件与目录，改名)

- mv [-fiu] source destination
- mv [options] source1 source2 source3 ...directory
- mv originName newName
- -f：强制的意思，如果目标文件已经存在，不会询问而直接覆盖
- -i：若目标文件已经存在时，就会询问是否覆盖
- -u：若目标文件已经存在，且 source 较新，才会更新(update)

8. 2. 3. 取得路径的文件名与目录名称

1、<basename>：

- basename [相对路径或绝对路径]：返回最后的文件名

2、<dirname>：

- dirname [相对路径或绝对路径]：取得最后文件所在的目录名

8. 3. 文件内容查阅

- cat：由第一行开始显示文件内容
- tac：由最后一行开始显示，可以看出 tac 是 cat 的倒写形式
- nl：显示的时候，顺便输出行号
- more：一页一页的显示文件内容
- less：与 more 类似，但是比 more 更好的是，它可以往前翻阅
- head：只看头几行
- tail：只看结尾几行
- od：以二进制的方式读取文件的内容

8.3.1. 直接查看文件内容

1、<cat>(concatenate)

- cat [-AbEnTv] 文件路径
- -A：相当于-vET 的整合参数，可以列出一些特殊符号，而不是空白而已
- -b：列出行号，对空白
- 标记行号
- -E：将结尾的断行字符$显示出来
- -n：打印出行号，连同空白也会有行号
- -T：将[Tab]按键以^I 显示出来
- -v：列出一些看不出来的特殊字符
- **不太好用，当文件过大时，根本来不及查看就过去了**
- **cat > [newfile] <==创建新文件，从键盘进行输入作文文件内容，[Ctrl]+D 退出键盘的输入**
  - cat 后面不接任何参数，将会进入键盘输入模式，键盘输入什么，就打印什么
  - 因此可以将输入流重定向到文件中
- **cat > [newfile] < [file] <==以 file 作为输入代替键盘输入，新建 newfile 文件**
- **cat > [newfile] <<'结束字符' <==键盘输入，在当前行有且仅有该'结束字符'并按下[Enter]后退出键盘的输入**

2、<tac>(反向表示)

- tac [-AbEnTv] 文件路径
- 与 cat 正好相反

3、<nl>

- nl [-bnw] 文件路径
- -b：指定行号指定的方式：
  - -b a：表示不论是否为空行，同样列出行号，类似于 cat -n
  - -b t：如果有空行，空的那一行不要列出行号
- -n：列出行号的表示方法：
  - -n ln：行号在屏幕的最左上方
  - -n rn：行号在自己字段的最右方显示，且不加 0
  - -n rz：行号在自己字段的最右方显示，且加 0
- -w：行号字段占用的位数

8.3.2. 可翻页查看

1、<more>

- more 文件路径
- 查看时的命令
  - Space：向下翻一页
  - Enter：向下滚动一行
  - /字符串：向下查询该字符串,按 n 继续查询同一个个字符串**(好像并没有用)**
  - :f：立刻显示出文件名以及目前显示的行数

- q：离开 more，不再显示文件内容
- b 或[Ctrl]-b：代表往回反野，只对文件有用，对管道无效

2、<less>

- less 文件路径
- **[command] | less <==当命令输出太多时，以 less 的方式查看命令的输出，非常有用！！！**
- 查看时的命令
  - Space：向下翻动一页
  - Up：向上滚动一行
  - Down：向下滚动一行
  - PageDown：向下翻动一页
  - PageUp：向上翻动一页
  - /字符串：向下查询字符串，按 n 继续向下查询，按 N 反向查询
  - ?字符串：向上查询字符串，按 n 继续向上查询，按 N 反向查询
  - q：离开 less 这个程序

8.3.3. 数据选取

1、<head>

- head 文件路径
- head [-n number] 文件路径
- -n：后面接数字，代表显示几行的意思
  - 若-n 后面接负数，比如-100：则表示，列出前面所有的行，但不包括最后这 100 行
- 没有-n 参数时，默认显示 10 行

2、<tail>

- tail 文件路径
- tail [-n number] 文件路径
- -n：后面接数字，代表显示几行
  - 若后面接'+'加数字，比如+100，则表示，列出所有行，但不包括前面的 100 行，即列出 100 行(包括第 100 行)以后的数据
- -f：表示持续监测后面所接的文件名(可能在查看的过程中，文件会被更新，-f 参数会将更新的内容显示到当前页面)，要等到按下[ctrl]-c 才会结束 tail 监测

8.3.4. 非纯文本文件：<od>

- od [-t TYPE] 文件路径
- -t a：利用默认的字符输出
- -t c：利用 ASCII 字符输出
- -t d[size]：利用十进制来输出数据，每个整数占用 size bytes
- -t f[size]：利用浮点数来输出数据，每个数占用 size bytes
- -t o[size]：利用八进制来输出数据，每个整数占用 size bytes
- -t x[size]：利用十六进制来输出数据，每个整数占用 size bytes

8.3.5. 修改文件时间或创建新文件：touch

1、三个时间：

- modification time(mtime)：当文件的"内容数据"更改时，就会更新这个时间，内容数据指的是文件的内容，而不是文件的属性或权限
- change time(ctime)：当该文件的"状态"改变时，就会更新这个时间，举例来说，像是权限与属性被更改了，都会更新这个时间
- access time(atime)：当"文件的内容被取用"时，就会更新这个读取时间，举例来说，我们用 cat 读取文件时，就会更新这个文件的 atime

2、ls 在默认情况下显示的是 mtime，要显示 atime 以及 ctime，可以加上参数--time=atime 或 --time=ctime

3、<touch>

- touch [-acdmt] 文件路径
- -a：仅修改访问时间**(并伴随 ctime 的修改，并非 touch 的作用)**
- -c：仅修改文件的时间，若该文件不存在**则**
- **创建**新文件**(修改 mtime 与 atime 为当前时间，其中 ctime 是自动修改的，与 touch 无关)**
- -d：后面可以接欲修改的日期而不用目前的日期，也可以用--date="时间或日期"
- -m：仅修改 mtime**(并伴随 ctime 的修改，并非 touch 的作用)**
- -t：后面可以接与修改的时间而不用目前的时间，格式为[YYMMDDHHmm]，例如，2007 年 5 月 12 日 12 点 08 分：0705121208
- ';'可以连续执行命令，执行顺序与书写顺序一致
- ctime 记录的是文件最近的状态被改变的时间，无法被 touch 修改！！！
- touch 最常用的情况：touch 文件路径
  - 当文件不存在时，创建一个空文件
  - 当文件存在时，将文件日期修改为目前日期(mtime 与 atime，**并伴随着 ctime 的自动修改**)

8.4. 文件与目录的默认权限与隐藏权限

1、一个文件有若干属性，包括读写执行(r,w,d)等基本权限，以及是否为目录(d)，文件(-)或者是链接文件(|)等属性。

2、**在 Ext3、Ext4 文件系统下**，我们还可以设置其他的系统隐藏属性，这部分可以使用 **chattr** 来设置，而以 **lsattr** 来查看，最重要的属性就是**设置其不可修改的特性**。这在安全机制方面尤为重要

8.4.1. 文件默认权限：<umask>

1、命令：

- umask　//显示数字形态的权限设置分数
- umask -S　//以符号类型的方式显示
- umask 002　//修改 umask 的后三位

2、默认情况：

- 若用户创建"文件"，则默认没有可执行权限，即只有 r、w 这两个选项，也就是最大值为 666，默认权限为-rw-rw-rw-

- 若用户创建"目录"，则由于 x 与是否可以进入此目录有关，因此默认为所有权限均开放，即为 777，默认权限如下 drwxrwxrwx

3、umask 分数的含义：

- umask 分数指的是：**该默认值需要减掉的权限**
- 比如 umask 为 0022，后三位为 022
  - 新建文件时：(-rw-rw-rw-)-(-----w--w-)=-rw-r--r--
  - 新建目录是：(drwxrwxrwx)-(d----w--w-)=drwxr-xr-x

8.4.2. 文件隐藏属性 chattr，lsattr

1、<chattr>：

- change attributes
- **chattr 只能在 Ext3/Ext4 的文件系统上面生效**
- chattr [+-=][ASacdistu] 文件或目录名称
- +：增加某一个特殊参数，其他原本存在的参数不变
- -：删除某一个特殊参数，其他原本存在的参数不变
- =：仅有后面接的参数
- A：若访问文件(目录)，访问时间 atime 不会被修改
- S：当你进行任何文件的修改，该改动会"同步"写入磁盘
- **a：这个文件将只能增加数据，而不能删除也不能修改数据，只有 root 才能设置这个属性**
- c：自动将文件压缩，读取的时候自动解压缩
- d：当 dump 程序被执行的时候，设置 d 属性将使该文件不会被 dump 备份
- **i：可以让一个文件"不能被删除，改名，设置连接也无法写入或添加数据"，只有 root 才能设置此属性**
- s：如果这个文件被删除，它将会被完全从这个硬盘空间中删除
- u：与 s 相反，使用 u 来配置文件时，如果该文件被删除了，则数据内容其实还存在磁盘中，可以使用来找回文件

2、<lsattr>

- list attributes
- list [-adR] 文件或目录
-
- -a：将隐藏文件的属性也显示出来
- -d：如果接的是目录，列出目录本身的属性而非目录内的文件名
- -R：连同子目录的数据也一并列出来

8.4.3. 文件特殊权限：SUID, GUID, SBIT

1、SetUID

- **s 这个标志出现在文件所有者的 x 权限上**
- **仅针对文件有效**
- SUID 权限仅对二进制程序有效
- 执行者对于该程序需要具有 x 的可执行权限
- 本权限仅在执行该程序的过程中有效

- 执行者将具有该程序所有者的权限
- **执行者会在执行该程序的过程中获取改程序所有者的权限**

2、SetGID

- **s 这个标志出现在文件用户组的 x 权限上**
- **SGID 可以针对文件或目录来设置**
- 对于文件
  - SGID 权限仅对二进制程序有效
  - 执行者对于该程序需要具有 x 的可执行权限
  - 执行者在执行过程中会获得该程序用户组的支持
- 对于目录：
  - 用户若对于此目录具有 r 和 x 权限时，该用户能够进入此目录
  - **用户在此目录下的有效用户组**将会变成该目录的用户组
  - 若用户在此目录下具有 w 的权限，则用户所创建的新文件的用户组与此目录的用户组相同

3、Sticky Bit

- **仅针对目录有效**
- 当用户对于此目录具有 w,x 权限，即具有写入的权限时
- **当用户在该目录下创建文件或目录时，仅有自己与 root 才有权利删除该文件**

4、如何设置 SUID/SGID/SBIT

- 利用命令 chmod 修改，在数字前添加一位，表示这几个权限的情况
- SUID:4
- SGID:2
- SBIT:1
- chmod 4755 filename：-rwsr-xr-x
- 文件或目录权限本身不具有 x 属性时，此时却设置了这三个特殊属性，显然是不对的，因此，此时会在 x 的位置上放置'S'与'T'替换原本会出现's'，'t'的部分。chmod 7666 test ：-rwSrwSrwT

8.4.4. 查看文件类型：<file>

- file [文件路径]
- 可用于判断一个文件是否为二进制文件，二进制文件会显示(ELF 64-bit LSB executable)
- 若是 shell script 则会显示(text executable)

8.5. 命令与文件的查询

8.5.1. 脚本文件名的查询

1、<which>

- which [-a] [command]
- -a：将所有由 PATH 目录中可以找到的命令均列出，而不只是第一个被找到的命令名称
- **这个命令是根据 PATH 这个环境变量所规范的路径去查询"执行文件"的文件名。所有，重点是找出执行文件而已**

8.5.2. 文件名的查找

1、<whereis> (寻找特定文件)

- whereis [-bmsu] [文件或目录路径]
- -b：只找二进制格式的文件
- -m：只找在说明文件 manual 路径下的文件
- -s：只找 source 源文件
- -u：查找不在上述三个选项当中的其他特殊文件
- 感觉 whereis 更加适合用于查找命令，是 which 的高级版本，放在查找文件名这里不太合适，但是它又确实可以查找特定文件，但是无法查找一般文件
- whereis 可以用来查找二进制文件、源码、和 man 手册，故可以推测会到 PAHT，lib，man 手册中进行查询

2、<locate>(寻找一般文件)

- locate [-ir] [keyword(而不需要完整的路径)]
- -i：忽略大小写的差异
- -r：后面可接正则表达式的显示方式
- 特点：只需要写出文件名的部分(可以不记得完整的文件路径)
- locate 是在已创建的数据库/var/lib/mlocate 里面的数据所查找到的，所以不用直接在硬盘当中去访问，速度很快
- 数据库的创建默认是每天执行一次，当你新建文件夹后可能查找不到，或者能够查找到已经被删除的文件
- 可以使用 updatedb 来更新数据库

4、<find>

- find [文件路径] [option] [action]
- -mtime n：列出在 n 天之前的"一天之内"被更改过的文件
- -mtime +n：列出在 n 天之前(不包括 n 天本身)被更改过的文件名
- -mtime -n：列出在 n 天之内(含 n 天本身)被更改过的文件名
- -atime 与-ctime 同上
- find [文件路径 1] -newer [文件路径 2]：找出在文件路径 1 中日期比文件路径 2 要新的文件
- find [文件路径] -user [用户名]：查找在文件路径下属于该用户的文件
- find [文件路径] -group [用户组名]：查找在文件路径下属于该用户组的文件
- find [文件路径] -name [文件名]：查找在文件路径下文件名为 filename 的文件
  - find / -name passwd
  - find . -name "*.java" > file.txt
- find [文件路径] -size [+-][SIZE]：查找比 SIZE 还有大(+)或小(-)的文件，SIZE 的规格有 c：代表 byte，k：代表 1024bytes
- find [文件路径] -type [TYPE]：查找文件类型为 TYPE 的文件，TYPE 可以是：一般正规文件(f)、设备文件(b,c)，目录(d)，链接文件(l)，socket(s)，以及 FIFO(p)等属性

- find /var -type s
- find [文件路径] -perm mode：查找权限刚好等于 mode 的文件
- find [文件路径] -perm -mode：查找权限必须包括 mode 全部(即 4 个数字)的文件
- find [文件路径] -perm +mode：查找权限至少包含 mode 的任一数字的文件
  - find / -perm +7000 <==查找含有 SGD,SUID,SBIT 的属性的文件
  - +7000 ==> ---s--s--t，只要有 s 或 t 就列出，而-7000，是必须包含这三个
- 其他用法参考 P190-P191

8. 6. 权限与命令间的关系(极重要)

1、让用户能进入某目录称为"可工作目录"的基本权限：
- 可使用的命令：如 cd 等切换工作目录的命令
- 目录所需的权限：用户对这个目录至少需要具有 x 的权限
- 额外需求：如果用户想要在这个目录内利用 ls 查阅文件名，则用户对此目录还需要 r 的权限

2、用户在某个目录内读取一个文件的基本权限：
- 可使用的命令：cat,mores,less
- 目录所需的权限：用户对这个目录至少需要具有 x 权限
- 文件所需权限：用户对文件至少需要具有 r 的权限

3、让用户可以修改一个文件的基本权限：
- 可使用的命令：nano、vi 等
- 目录所需的权限：用户在该文件所在的目录至少需要具有 x 权限
- 文件所需权限：用户对该文件至少要有 r,w 权限

4、让一个用户可以创建一个文件的基本权限
- 目录所需权限：用户在该目录要具有 w,x 的权限，创建的重点在 w 权限

5、让用户进入某目录并执行该目录下某个命令的基本权限
- 目录所需权限：用户在该目录至少需要有 x 的权限
- 文件所需权限：用户在该文件至少需要有 x 的权限

8. 7. 纯文本文件与非纯文本文件

1、知道计算机的存储在物理上是二进制的，所以文本文件与二进制文件的区别并不是物理上的，而是逻辑上的。这两者只是在编码层次上有差异。简单来说，文本文件是基于字符编码的文件，常见的编码有 ASCII 编码，UNICODE 编码等等。二进制文件是基于值编码的文件，你可以根据具体应用，指定某个值是什么意思(这样一个过程，可以看作是自定义编码)

2、可以看出文本文件基本上是定长编码的(也有非定长的编码如 UTF-8)。而二进制文件可看成是变长编码的，因为是值编码嘛，多少个比特代表一个值，完全由你决定。大家可能对 BMP 文件比较熟悉，就拿它举例子吧，其头部是较为固定长度的文件头信息，前 2 字节用来记录文件为 BMP 格式，接下来的 8 个字节用来记录文件长度，再接下来的 4 字节用来记录 bmp 文件头的长度。

Chapter 9. Linux 磁盘与文件管理系统

9.1. 认识 EXT2 文件系统

9.1.1. 硬盘组成与分区的复习

1、磁盘的组成

- 圆形盘片
- 机械手臂与机械手臂上的磁头
- 主轴马达，可以转动盘片，让机械手臂的磁头在盘片上读写数据

2、盘片物理组成

- 扇区，最小的物理存储单位，每个扇区 512byte
- 将扇区组成一个圆，那就是柱面，柱面是分区的最小单位
- 第一个扇区最重要，里面有硬盘主引导记录(Masterbootrecord,MBR)以及分区表，其中 MBR 占有 446bytes，而 partition table 则占有 64bytes

3、硬盘在 Linux 中的文件名：

- /dev/sd[a-p][1-15]：SCSI,SATA,USB,Flash 等接口的磁盘文件名
- /dev/hd[a-d][1-63]：为 IDE 接口的磁盘文件名

4、磁盘分区是指：告诉操作系统，这块磁盘在此分区可以访问的区域是由 A 柱面到 B 柱面之间的块

5、指定分区柱面范围的记录在第一个扇区的分区表中，由于分区表只有 64byte，因此只能记录四条分区记录，**这四条分区记录称为主分区或扩展分区，其中扩展分区还可以再分出逻辑分区，而能被格式化的仅有分区和逻辑分区而已**

6、总结

- 主分区与扩展分区最多可以有 4 个(硬盘的限制)
- 扩展分区最多只能有一个(操作系统的限制)
- 逻辑分区是由扩展分区继续分出来的区
- **能够被格式化后作为数据访问的分区为主要分区与逻辑分区，扩展分区无法被格式化**
- 逻辑分区的数量依操作系统而不同，在 Linux 系统中，IDE 硬盘最多有 59 个逻辑分区(5-63 号)，SATA 硬盘则有 11 个分区(5-15 号)

9.1.2. 文件系统特性

1、格式化：因为每种操作系统所设置的文件属性/权限并不相同，为了存放这些文件所需的数据，因此需要将分区格式化，已成为操作系统能够利用的文件系统格式

2、以前，一个分区就只能被格式化称为一个文件系统，LVM 技术可以将一个分区格式化为多个文件系统，也能将多个分区合成一个文件系统。**因此，通常我们可以称呼一个被挂载的数据为一个文件系统而不是一个分区**

3、在 Linux 系统中，**文件权限(rwx)与文件属性(所有者，群组，时间参数等)被存放在 inode 中**，实际的数据放在 data block 中，另外一个超级块(super block)会记录整个文件系统的整体信息，包括 inode 与 block 的总量，使用量，剩余量

4、具体说明：

- super block：记录此文件系统的整体信息，包括 inode/block 的总量，使用量，剩余量，以及文件系统的格式与相关信息等

- inode：记录文件的属性，**一个文件占用一个 inode(大量的小文件可能会造成资源的浪费)**，同时记录文件的数据所在的 block 号码
- block：实际记录文件的内容，**若文件太大时，会占用多个 block**

5、碎片整理：需要碎片整理的原因就是文件写入的 block 太过于离散了，此时文件读取的性能将会变得很差。

9.1.3. Linux 的 Ext2 文件系统（inode）

1、文件系统一开始就将 inode 与 block 规划好了，除非重新格式化，否则 inode 与 block 固定后就不再变动，如果文件系统高达数百 GB，将所有 inode 与 block 放在一起是很不明智的。

2、Ext2 文件系统在格式化时基本上是区分为多个组块，每个组块都有独立的 inode/block/super block 系统。

3、在整体规划中，**文件系统最前面有一个启动扇区，这个启动扇区可以安装引导装载程序，这是个非常重要的设计，因为这样一来，我们就可以把不同的引导装载程序安装到个别的文件系统最前端，而不用覆盖整块硬盘为一个 MBR，才能制作出多重引导的环境**

4、data block

- Ext2 文件系统支持的 block 大小有 1KB,2KB,4KB
- 原则上，block 的大小与数量在格式化完就不能再改变了
- 每个 block 最多只能放一个文件的数据
- 若文件大于 block 的大小，则一个文件会占据多个 block
- 若文件小于 block 的大小，则该 block 剩余的空间将不再被利用
- 较大的 block 可能会产生严重的磁盘容量浪费(大量小文件)
- 较小的 block 可能导致文件系统的不良读写性能(大文件占用很多 block，因此 inode 也要记录更多的 block)

5、inode table 存放的数据

- 该文件的访问模式
- 该文件的所有者组
- 该文件的大小
- 该文件创建或状态改变的时间
- 最近一次读取时间
- 最近修改时间
- 定义文件特性的标志，如 SetUID
- 该文件真正内容的指向

6、inode 的特性：

- 每个 inode 大小固定为 128bytes
- 每个文件都仅占用一个 inode 而已
- 文件系统能够创建的文件数量与 inode 的数量有关
- 系统读取文件时先要找到 inode，并分析 inode 所记录的权限与用户是否符合，若符合才能够开始实际读取 block 的内容

7、**Linux 系统将 inode 记录 block 号码的区域定义为 12 个直接、一个间接、一个双间接与一个三间接记录去，间接是指用 block 来当做记录 block 号码的记录区**

- 以 block=1KB 作为例子，记录一条 block 号码需要 4bytes
- 12 个直接 V1=12*1K=12K
- 1 个间接 V2=256*1K=256K
- 1 个双间接 V3=256*256*1K=256^2K
- 1 个三间接 V4256*256*256*1K=256^3K
- V=V1+V2+V3+V4=16GB

8、super block
- 记录整个文件系统相关信息的地方
- block 与 inode 的总量
- 未使用与已使用的 inode/block 数量
- block 与 inode 的大小(1/2/4K,128bytes)
- 文件系统的挂载时间，最近一次写入数据的时间，最近一次检验磁盘(fsck)的时间等文件系统的相关信息
- 一个 **validbit** 数值，若此文件系统已经被挂载，则 valid bit 为 0，否则为 1
- **此外每个 block group 都可能含有 super block，但是文件系统应该只有一个 super block，因为这些 block group 的 super block 是作为备份，在 super block 出故障时进行救援的作用**

9、File system Description(文件系统描述说明)
- 描述每个 block group 的开始于结束的 block 号码，以及说明每个区段分别介于哪一个 block 号码之间，这部分能用 dumpe2fs 来查看。

10、block bitmap
- 用于告知哪些 block 是空的，因此系统可以很快地找到可以使用的空间来处置文件
- 同理，若删除文件，要将 block bitmap 中对应的标志标记为未使用

11、inode bitmap
- 与 bolck bitmap 类似，用于告知 inode 的使用状况

12、**<dumpe2fs>**
- dumpe2fs [-bh] 设备文件名
- -b：列出保留为坏道的部分(一般用不到)
- **-h：仅列出 super block 的数据，不会列出其它区段的内容**
- 若不加-h，则上半部分列出 super block 的数据，下半部分列出每个 block group 的数据

13、**<df>**
- 调出目前挂载的设备

9. 1. 4. 与目录树的关系

1、每个文件(不管是一般文件还是目录文件)都会占用一个 inode，且可依据文件内容的大小来分配多个 block 给文件使用，**而目录的内容在记录文件名，一般文件才是实际记录数据内容的地方**

2、目录：
- 在 Linux 下的 Ext2 文件系统新建一个目录，Ext2 会分配一个 inode 与至少一块 block 给该目录
- **inode 记录该目录的相关权限与属性，并可记录分配到那块 block 的号码**

- **block 记录在这个目录下的文件名，与该文件名占用的 inode 号码数据**
- **如果该目录下的文件数太多，导致一个 block 无法容纳下所有文件名与 inode 对照表，此时会基于该目录多一个 block 来继续记录相关的数据**

3、文件：

- 在 Linux 下的 Ext2 新建一个一般文件时，Ext2 会分配一个 inode 与相对于该文件大小的 block 数量给该文件。

4、目录读取树：

- inode 本身并不记录文件名，文件名的记录在目录的 block 中，因此新增/删除/重命名文件名与目录的 w 权限有关。因为文件名存在 block 中，因此，要读取某个文件时，就务必会经过目录的 inode 与 block，然后才能够找到那个待读取文件的 inode 号码，最终才能读取到正确的文件的 block 内的数据。
- 目录树由根目录开始读起，因此系统通过挂载的信息可以找到挂载点的 inode 号码(通常，一个文件系统最顶层的 inode 号码会从 2 开始)，并根据该 inode 读取根目录的 block 内的文件名数据，再一层层往下读到正确的文件名

9.1.5. Ext2/Ext3 文件的访问与日志文件系统的功能

1、新增文件时，文件系统此时的行为：

- 先确定用户对于添加文件的目录是否具有 w 与 x 权限，若有的话才能添加
- 根据 inode bitmap 找到没有使用的 inode 号码，并将新文件的权限/属性写入
- 根据 block bitmap 找到没有使用中的 block 号码，并将实际的数据写入 block 中，且更新 inode 的 block 指向数据
- 将刚才写入的 inode 与 block 数据同步更新到 inode bitmap 与 block bitmap，并更新 super block 的内容

2、一般来说

- inode table 与 data block 称为数据存放区域
- super block、block bitmap 与 inode bitmap 等区段就被称为 meta data(中间数据)，因为这些数据经常变动，每次添加，删除编辑，都可能会影响到这三个数据，因此称为中间数据。

3、数据的不一致状态

- 当文件写入系统时，发生了中断(停电)，可能会导致 metadata 的内容与实际数据存放区产生不一致的情况
- 若此情况发生，系统在重启时会通过 super block 的 valid bit 与文件系统的 state(clean 与否)等状态来判断是否强制进行数据一致性检查
- 但是这样的检查是很费时的！

4、日志文件系统

- 在文件系统中规划处一个块，该块专门记录写入或修订文件时的步骤
- 预备：当系统要写入一个文件时，会现在日志记录块中记录某个文件准备要写入的信息
- 实际写入：开始写入文件的权限与数据，开始更新 meta data

- 结束：完成数据与 meta data 的更新后，在日志记录快中完成该文件的记录

9.1.6. Linux 文件系统的操作

1、所有的数据都得要加载到内存后 CPU 才能够对该数据进行处理

2、如果经常编辑一个很大的文件，在编辑过程中又频繁地要系统来写入磁盘中，由于磁盘写入的速度要比内存慢很多，因此会经常耗在等待硬盘的写入/读取上。

3、为了解决这个问题，Linux 使用异步处理的方式

- 当加载一个文件后，如果该文件没有被改动过，则在内存区段的文件数据会被设置为(clean)
- 但如果内存中的文件数据被更改过了，此时该内存中的数据会被设置为 Dirty，此时所有的操作都还在内存中执行，并没有写入到磁盘中，系统会不定时地将内存中设置为 Dirty 的数据写入磁盘，以保持磁盘与内存数据的一致性

4、Linux 系统上文件系统与内存有非常大的关系

- 系统会将常用的文件数据放置到主存器的缓冲区，以加速文件系统的读/写
- 因此 Linux 的物理内存最后都会被用光，这是正常情况，可以加速系统性能
- **可以手动使用 sync 来强迫内存中设置为 Dirty 的文件回写到磁盘中**
- **若正常关机时，关机命令会主动调用 sync 将内存的数据会写到磁盘中**
- 若不正常关机(如断电，死机或其他不明原因)，由于数据尚未回写到磁盘内，因此重新启动后可能会花很多时间在进行磁盘检验，甚至可能导致文件系统的损毁(非磁盘损坏)

9.1.7. 挂载点(mount point)的意义

- 每个文件都有独立的 inode、block、super block 等信息，这个文件系统要能够链接到目录树才能够被使用
- **将文件系统与目录树结合的操作我们称为挂载**
- 挂载点一定是目录，该目录为进入该文件系统的入口

9.1.8. 其他 Linux 支持的文件系统与 VFS

1、常见的支持文件系统：

- 传统文件系统：ext2/minix/MS-DOS/FAT/iso9660(光盘)
- 日志文件系统：ext3/ReiserFS/Windows'NTFS/IBM'sJFS/SGI'sXFS
- 网络文件系统：NFS/SMBFS

2、Linux VFS：Virtual Filesystem Switch(虚拟文件系统)：整个 Linux 认识的文件系统其实都是 VFS 在进行管理，我们用户并不需要知道每个分区上头的文件系统是什么，VFS 会主动帮我们做好读取操作。

9.2. 文件系统的简单操作

9.2.1. 磁盘与目录的容量：df，du

1、<df>：

- df [-ahikHTm] [目录或文件名]
- -a：列出所有的文件系统，包括系统特有的/proc 等文件系统
- -k：以 KB 的容量显示各文件系统
- -m：以 MB 的容量显示各文件系统
- **-h：以人们交易阅读的 GB、MB、KB 等格式自行显示**
- -H：以 M=1000K 代替 M=1024K 的进位方式
- -T：连同该分区的文件系统名称(如 ext3)也列出
- **-i：不用硬盘容量，而是以 inode 的数量来显示**

2、<du>：

- du [-ahskm] [文件或目录
-]
- -a：列出所有的文件与目录容量，因为默认仅统计目录
- -h：以人们较易阅读的容量格式(G/M)显示
- **-s：列出总量，而不列出每个个别的目录占用容量**
- **-S：不包括子目录下的统计，注意与 du 不加任何参数的区别**
- -k：以 KB 列出容量显示
- -m：以 MB 列出容量显示
- 对于 du [文件或目录名]：
  - 会列出所有的**目录**的容量
  - 比如目录 dir1 中包含有目录 dir11 和 dir12，假设都是空目录，那么显示 dir1 的容量时，会把自身 dir1 的容量，以及其子目录的容量都算上去，也就是 12KB(blocksize=4K)
  - 因此把列出的所有目录容量之和相加不等于总量，因为子目录都被重复计算了
- 对于 du -S [文件或目录名]：
  - 会列出所有的**目录**的容量
  - 并且在计算该目录容量的时候不会加上其子目录的容量，也就是该目录自身所占 block 的大小
  - 因此把所有列出的目录容量之和相加就等于总量

9.2.2. 连接文件：ln

1、种类

- 类似于 Windows 的快捷方式功能的文件，可以让你快速连接到目标文件(或目录)
- hard link：通过文件系统的 inode 连接来产生新文件名而不是产生新文件

2、hard link(硬链接或实际连接)

- 每个文件都会占用一个 inode，文件内容由 inode 的记录来指向
- 想要读取该文件，必须要经过目录记录的文件名来指向到正确的 inode 号码才能读取
- 也就是说，文件名只与目录有关，但是文件内容则与 inode 有关

- **简单地说：hard link 只是在某个目录下新建一条文件名连接到某 inode 号码的关联记录而已**
  - 由于目录的 block 存放的就是文件名与 inode 的关联记录
  - hard link 在某个目录下，也就是该目录的 block 中新建一条文件名与 inode 的关联记录，因此，真实文件的 inode 的关联记录会增加一条
- **两个文件名都会连接到同一个 inode 号码，由 ls -l 查看的链接数就是有多少个文件名连接到这个 inode 号码的意思**
- 最大的好处是安全：如果将任意一个文件名删除，那么 inode 与 bolck 还是存在的，不论用哪个文件名来编辑，最终的结果都将写入到相同的 inode 与 block 中，因此均能进行数据的修改
- 一般来说，用 hard link 设置连接文件，磁盘空间与 inode 树木都不会变，因为 hard link 只是在目录的 block 多写入一个关联数据而已(除非写入该数据后导致当前目录的 block 已满，必须额外分配一个 block)
- 限制：
  - **不能跨文件系统**
  - **不能连接到目录**(会造成环境相当大的复杂度，该目录下的所有目录和文件都得实现硬链接)

3、symbolic link(符号连接，也即是快捷方式)

- 基本上 symbolic link 就是在创建一个独立的文件，这个文件会让数据的读取指向它**连接**的那个**文件名(与 hard link 不同，这里是连接到文件名)**
  - **这里新建的文件指向的是真实文件所在的目录的 block 中的文件名，并通过该文件名关联到真实文件的 inode，此过程中，从文件名到 inode 的关联记录没有增加**
- 因此当源文件被删除后，symbolic link 的文件会无法打开，因为找不到源文件的文件名
- symbolic link 与 Windows 的快捷方式可以划上等号，由于 symbolic link 所创建的文件为一个独立的新的文件，所以会占用掉 inode 与 block

4、命令<ln>

- ln [-sf] [源文件名] [目标文件名]
- **-s：若不加任何参数就进行连接，就是 hard link，加上-s 就是 symbolic link**
- -f：如果目标文件存在时，就主动将目标文件直接删除后再创建

5、基本上 symbolic link 的用途比较广

6、关于目录的连接数量

- 一个空目录至少包含：'本身'，'.'，'..'，而'本身'与'.'是都代表当前目录，'..'代表上一层目录，因此连接数量为 2
- 新建一个目录，上层目录链接数为增加 1

9. 3. 磁盘的分区、格式化、检验与挂载

- 对磁盘进行分区，以新建可用的分区
- 对该分区进行格式化，以创建系统可用的文件系统
- 若想要仔细一点，则可对刚才新建好的文件系统进行检验

- 在 Linux 系统上，需要创建挂载点(即是目录)，并将它挂载上来

9.3.1. 磁盘分区：<fdisk>

- fdisk [设备文件名]
- 输出后面接的设备所有的分区内容
  - 按 m 查看命令介绍
  - d：删除一个分区
  - n：新增一个分区
  - p：在屏幕上显示分区表
    - Device：设备名
    - Boot：表示是否为开机引导模块
    - Start,End：表示这个分区的柱面号码范围
    - Blocks：就是以 1K 为单位的容量
    - ID,System：
  - q：不存储，离开 fdisk 程序
  - w：将刚才的操作写入分区表
- fdisk [-l]
- -l：若仅有 fdisk -l 时，则系统将会把整个系统内能够找到的设备的分区均列出来
- fdisk -l /dev/sda //列出/dev/sda 中的设备

1、新增磁盘分区：(n)

- 1-4 号尚有剩余，且系统未有扩展分区：此时会让你挑选 Primary/Extended 的选项，可以指定 1-4 的号码
- 1-4 号尚有剩余，且系统有扩展分区：此时会让你挑选 Primary/Logical 的选项，若选择 p，则还能指定 1-4 号的号码，若选择 l，则不需要设置号码，系统会自动指定逻辑分区的文件
- 号码
- 1-4 没有剩余，且系统有扩展分区：此时
- 会让你挑选分区类型，直接进入 Logical 的分区形式
- 然后按 w 进行保存
- 需要重启才能够生效，或者输入命令 partprobe 强制让内核重新找一次分区表

9.3.2. 磁盘格式化

1、<mkfs>：

- mkfs [-t 文件系统格式] [设备文件名]
- -t：可接文件系统格式，如 ext3，ext4，vfat 等
- 例如：mkfs -t ext3 /dev/sda10
- mkfs[tab][tab]：查看 mkfs 支持的文件格式

2、由于没有详细指定文件系统的具体选项，因此系统会用默认值来格式化，重要部分有：文件系统的卷标(label)、block 的大小以及 inode 的数量

3、<mke2fs>：

- mke2fs [-b block 大小] [-i block 大小] [-L 卷标] [-cj] [设备文件名]
- -b：设置每个 block 的大小，支持 1024，2048，4096
- -i：多少容量给予一个 inode
- 检查磁盘错误
- -L：后面可接卷标名称
- -j：本来 mke2fs 是 Ext2，加上-j 就加入 journal 变为 Ext3

4、**没有特殊需求就用 mkfs**

9.3.3. 磁盘检验：fsck，badblocks

1、<fsck>：

- fsck [-t 文件系统] [-ACay] [设备名称]
- -t：指定文件系统名称，与 mkfs 一样
- -A：依据/etc/fstab 的内容，将需要的设备扫描一次
- -a：自动修复检查到的有问题的扇区，所以不用一直按 y
- -y：与-a 类似，某些文件系统仅支持-y
- -C：在检验过程中使用一个直方图来显示目前的进度
- fsck[tab][tab]：查看多少文件系统支持 fsck
- **通常只有身为 root 且你的文件系统有问题时才使用这个命令，否则在正常情况下使用此命令，可能会造成对系统的伤害**
- 使用 fsck 时，被检查的分区务必不可挂载到系统上，即需要在卸载的状态

2、<badblocks>：

- badblocks [-svw][设备名称]
- -s：在屏幕上列出进度
- -v：在平明上看到进度
- -w：使用写入的方式来测试，**建议不要使用，尤其是待检查的设备已有文件时**

9.3.4. 磁盘挂载与卸载

1、挂载前确定几件事：

- 单一文件系统不应该被重复挂载不同的挂载点(目录)中
- 单一目录不应该重复挂载多个文件系统
- 作为挂载点的目录理论上应该都是空目录
- 挂载了文件系统后，原目录下的东西就会暂时消失，并不是被覆盖掉，而是被隐藏了，当文件系统被卸载后，这些隐藏的东西就再次显示出来

2、<mount>：

- mount -a
- mount -l
- **mount [-t 文件系统] [-L Label 名称] [-o 额外选项]**
- **[-n] [设备文件名] [挂载点]**
- **-a：依照配置文件/etc/fstab 的数据将所有未挂载的磁盘都挂载上来**
- **-l：单纯输入 mount 会显示目前挂载的信息，加上-l 可显示 Label 名称**
- -t：与 mkfs 的参数非常类似，可加上文件系统种类来指定欲挂载的类型

- -n：在默认情况下，系统会将实际挂载的情况实时写入/etc/mtab 中，以利其他程序的运行。但在某些情况下(例如单用户维护模式)为了避免问题，刻意不写入，此时需要加-n 参数
- -L：系统除了利用设备文件名(/dev/sda5 等等)之外，还可以利用文件系统的卷标名称(Label)来挂载，但是卷标名称必须独一无二(不建议这样用)
- 挂载 Ext2/3/4 文件系统简单的命令：
  - mount [设备名] [挂载点目录名]
- -o：后面可以接一些挂载时额外加上的参数，比方说账号、密码、读写权限等
  - ro，rw：挂载文件系统成为只读(ro)或可读写(rw)
  - async,sync：文件系统是否使用同步写入(sync)，或异步(async)的内存机制，默认为 async
  - auto，noauto：是否允许此分区被以 mount -a 自动挂载
  - dev，nodev：是否允许此分区可以创建设备文件
  - suid，nosuid：是否允许此分区含有 SUID/SGID 的文件格式
  - exec，noexec：是否允许此分区上拥有可执行的 binary 文件
  - user，nouser：是否允许此分区让任何用户执行 mount，一般来说 mount 只有 root 可以进行
  - defaults：默认值为 row，suid，dev，exec，auto，nouser，async
  - **remount：重新挂载，这在系统出错，或重新更新参数时，很有用**

3、<umount>：

- umount [-fn] [**设备文件名或挂载点**]
- -f：强制卸载
- -n：不更新/etc/mtab 的情况下卸载

9.3.5. 磁盘参数的修改

1、<mknod>：

- Linux 下的设备都以文件来代表
- 通过文件的 major 与 minor 数值来替代
- **基本上，Linux 内核 2.6 版本以上，硬件文件名都已经可以被系统自动实时产生了，我们根本不需要手动创建**
- 但是，在某些服务被放到特定目录下时，就需要设置硬盘参数

2、<e2label>：

- 用 mke2fs 对硬盘进行格式化时可以设置 Lable
- 但是在格式化完成后，如果想要修改卷标，需要用 e2label
- **最好用磁盘文件名来挂载，当然，确保卷标唯一的情况下也能用卷标名来挂载，但是不提倡**

3、<tune2fs>：

- tune2fs [-jlL] [设备名称] //设备一般都放在/dev 下
- -l：类似于 dumpe2fs -h 的功能，将 super block 的数据读出来
- -j：将 ext2 文件系统转换为 ext3 文件系统
- -L：类似 e2label 的功能，修改文件系统的 Label

9.4. 开机设置挂载

- 手动处理 mount 不是很人性化，每次开机后都需要再次挂载一此

9.4.1. 开机挂载/etc/fstab 以及/etc/mtab

1、系统挂载的限制：

- 根目录/必须挂载，而且一定要优先于其他 mount point 被挂载起来
- 其他挂载点必须为已新建的目录，可任意指定，但一定要遵守必须的系统目录架构原则
- 所有挂载点在同一时间内，只能挂载一次
- 所有分区在同一时间内，只能挂载一次
- 若进行卸载，必须先将工作目录移动到挂载点(及其子目录)之外

2、/etc/fstab 格式：

- 第一列：磁盘设备文件名或该设备的 Label
- 第二列：挂载点，一定是目录
- 第三列：磁盘分区的文件系统，ext3，ext4，vfat 等等
- 第四列：文件系统参数(在介绍 mount -o 中参数提到过)，一般写 defaults

参数	内容意义
async/sync	设置磁盘是否以异步方式运行，默认 async(性能较佳)
auto/noauto	当下达 mount -a 时，此文件系统是否会被主动测试挂载，默认为 auto
rw/ro	让该分区可以读写或者是只读的形态挂载上来
exec/noexec	限制在此文件系统内是否可以进行"执行"的工作
user/nouser	表示是否允许用户使用 mount 命令来挂载，一般来说，我们不希望一般身份的 user 使用 mount，因为太不安全了，默认为 nouser
suid/nosuid	表示该文件系统是否允许 SUID 的存在
usrquota	启动文件系统支持磁盘配额模式
grpquota	启动文件系统对群组磁盘配额模式的支持
defaults	同时具有 rw,suid,dev,exec,auto,nouser,async 等参数

- 第五列：能否被 dump 备份命令作用
  - 0：不要做 dump 备份
  - 1：要每天进行 dump 备份
  - 2：不定日期的 dump 备份
  - 通常设为 0 或 1
- 第六列：能否以 fsck 检验扇区
  - 0：不要检验
  - 1：最早检验(一般只有根目录设为 1)
  - 2：要检验，但是晚于 1 检验

3、实际文件系统的挂载是记录到/etc/mtab 与/proc/mounts 这两个文件中，每次我们在改动文件系统的挂载时，这两个文件会被更新

4、如果你在/etc/fstab 中输入的数据有误，导致无法顺利开机成功，而进入单

用户维护模式，这时候/是 readonly 的状态，自然无法修改/etc/fstab，也无法更新/etc/mtab，可以利用

- mount -n -o remount,rw /

5、/etc/mtab 文件：

- 该文件记录的是：当前已挂载的分区信息。
- 每当 mount 挂载分区、umount 卸载分区，都会动态更新/etc/mtab
- /etc/mtab 总是保持着当前系统中已挂载的分区信息
- fdisk、df 这类程序，必须要读取/etc/mtab 文件，才能获得当前系统中的分区挂载情况

9.4.2. 特殊设备 loop 挂载（镜像文件不用刻录就能挂载）

1、命令：

- mount -o loop [文件名] [挂载目录]
- mount -o loop /tmp/system.img /mnt

2、**如何挂载 NTFS 格式的 U 盘**

- 安装<ntfs-3g>
  - 先从 ntfs-3g 官网上下载*.tgz 文件
  - 进入*.tgz 文件的目录
  - 用命令：tar -xvf *.tgz 进行解压
  - 进入解压后的目录
  - 首先确保有 C 编译器，若没有的话，./configure 会执行失败，在联网的情况下可以用命令 yum install gcc 来安装 C 编译器
  - 用命令./configure && make && make install
    - &&：代表前一个执行成功就执行后面的命令
    - ;：代表无论前一个命令是否执行成功都会执行后面的命令
- 进行挂载：mount **-t ntfs-3g** [设备文件] [挂载目录]

3、新建大文件以制作 loop 设备文件

- **如果当初在分区，只分出了根目录，假设你已经么有多余的空间可以进行额外的分区，偏偏根目录的空间还很大，此时就可以制作一个大文件，然后将这个文件挂载，如此一来，感觉上就多了一个分区，用途很广**
- 创建大型文件：<dd>
  - dd if=/dev/zero of=/home/loopdev bs=1M count=512
  - if：input file，输入文件，那个/dev/zero 是会一直输出 0 的设备
  - of：output file，将一堆零写入到后面接的文件中
  - bs：每个 block 大小，就想文件系统那样的 block
  - count：多少个 bs 的意思
  - **/dev/zero：是一个输入设备，可以用它来初始化文件，该设备无穷地提供 0，可以使用任何你需要的数目，用于向设备写入字符串 0**
- 创建完之后进行格式化
  - mkfs -t ext3 /home/loopdev
  - 由于是不正常的设备，会提示你"**Proceed anyway(y,n)?**"选择 y 即可
- 最后挂载
  - mount -o loop /home/loopdev /media/cdrom/

9.5. 内存交换空间(swap)的构建

- swap 的功能：就是在应付物理内存不足的情况下所造成的内存扩展记录的功能
- 一般来说，如果硬件的配备足够的话，那么 swap 应该不会被系统所使用到
- 创建 swap 的步骤：
  - 设置一个 swap 分区
  - 创建一个虚拟内存的文件

9.5.1. 使用物理分区构建 swap

1、步骤：

- 分区：先用 fdisk 在磁盘中分一个区给系统作为 swap
- 格式化：用<mkswap> [设备文件名] 对该分区进行格式化
- 使用：将该 swap 设备启动，用<swapon> [设备文件名]
- 查看：<free>查看内存使用情况(若有多个 swap 分区，会显示总用量)
- <swapon> -s：查看目前使用的 swap 设备有哪些
- **<swapoff> [设备文件名]：关掉 swap file**

9.5.2. 使用文件构建 swap

1、步骤：

- dd if=/dev/zero of=/tmp/swap bs=1M count=128
- mkswap /tmp/swap
- swapon /tmp/swap
- free

9.5.3. swap 使用上的限制

- 一般情况下用不到 swap，因为现在的设备物理内存足够大了
- 针对于服务器或者工作站这些常年上线的系统，swap 还是需要的
- 在 2.4.10 版本以后，单一的 swap 已经没有 2GB 的限制了
- 最多仅能创建 32 个 swap
- 由于最 x86_64(64 位)最大内存寻址到 64GB，因此 swap 总量最大也仅能是 64GB

9.6. 文件系统的特殊查看与操作

9.6.1. 磁盘空间浪费问题

1、ll -s：查看所有数据占用的实际 block 数量以及每个文件占用的 block 数量

2、du -s [文件名]

- du -sb [文件名]：以 bytes 方式显示
- du -sm [文件名]：以 KB 显示

9.6.2. 利用 GUN 的 parted 进行分区

1、虽然可以用 fdisk 很快地将分区调整妥当，不过 fdisk 无法支持到 2TB 以上的分区，此时会需要 parted 来处理

2、命令：<parted>

- parted [设备名] [命令 [参数]]
- 命令功能：
  - 新增分区：mkpart [primary|logical|extended] [ext3/4|vfat] [开始] [结束]
  - 分区表：print
  - 删除分区：rm [partition]
- 新增分区例子：parted /dev/sda mkpart logical ext4 19.2GB 19.7GB
- 删除分区例子：parted /dev/sda rm 8
  - 其中编号 8 可以通过 parted /dev/sda print 来查看

Chapter 10. 文件与文件系统的压缩与打包

10.1. 压缩文件的用途与技术

- 文件里面有相当多的空间存在(以二进制序列的角度来看，比如连续多个0，可以视为空间没有被占用)
- 压缩技术就是将这些空间填满，让整个文件占用的容量下降
- 压缩后的文件无法被直接使用，需要解压缩

10.2. Linux 系统常见的压缩命令

1、Linux 环境中，压缩文件案的扩展名大多为".tar",".tar.gz",".tgz",".gz",".Z",".bz2"

- .Z：compress 程序压缩的文件
- .gz：gzip 程序压缩的文件
- .bz2：bzip2 程序压缩的文件
- .tar：tar 程序打包的数据，并没有被压缩过
- .tar.gz：tar 程序打包的文件，其中经过 gzip 的压缩
- .tar.bz2：tar 程序打包的文件，其中经过 bzip2 的压缩

2、其中 compress 已经不再流行，常见的就是 gzip 与 bzip2，但这些仅能针对一个文件或目录进行压缩

3、tar 可以将很多文件打包成一个文件，包括目录。

10.2.1. <Compress>

1、安装:yum install ncompress

2、压缩：

- compress [-rcv] [文件或目录]
- -r：可以连同目录下的文件也同时给予压缩
- -c：将压缩数据输出称为 standard output(输出到屏幕)
- -v：显示压缩后的文件信息以及压缩过程中的一些文件名的变化
- 注意，压缩后，被压缩的源文件会消失，若要保留源文件使用以下命令
  - compress -c [文件名] > [压缩后的文件名]
  - coompress -c man.config > man.config.back.Z

3、解压缩

- uncompress [文件名]

4、**没事别用这个命令**

10.2.2. <gzip>，<zcat>

1、gzip 可以解开 compress，zip，gzip 等软件所压缩的文件，gzip 所新建的压缩文件为*.gz 的文件名

2、压缩：

- gzip [-cdtv#] [文件名]
- -c：将压缩的数据输出到屏幕上，可以通过数据流重定向来处理
- -d：解压缩参数
- -t：可以用来检验一个压缩文件的一致性，看看文件有无错误
- -v：可以显示出原文件/压缩文件的压缩比等信息

- -#：压缩等级，-1 最快，压缩比最差；-9 最慢，但是压缩比最好；默认-6
- 使用与 compress 相同的方法，压缩过程中保留原文件

3、解压缩

- gzip -d [文件名]

4、cat 可以读取纯文本文件，但是 zcat 可以读取纯文本被压缩后的压缩文件

- zcat [文件名(扩展名为.gz)]

10.2.3. <bzip2>，<bzcat>

1、bzip2 用于取代 gzip 并提供更佳的压缩比

2、压缩

- bzip2 [-cdkzv#] [文件名]
- **-k：保留原文件，而不会删除原始的文件**
- 其他 option 与 gzip 相同

3、解压缩

- bzip2 -d [文件名]

10.3. 打包命令：<tar>

1、compress，gzip，bzip2 仅能针对单一文件或目录进行压缩

2、将多个文件或目录包成一个大文件的命令功能，称为打包命令

3、tar 可以将多个目录或文件打包成一个大文件，同时还可以通过 gzip/bzip2 的支持，将该文件同时进行压缩。

4、命令：

- **压缩：tar -jcv -f [压缩创建的文件(*.tar.bz2)] [被压缩的文件或目录 1] [被压缩的文件或目录 2]...**
- **查询：tar -jtv -f [压缩文件(*.tar.bz2)]**
- **解压缩：tar -jxv -f [压缩文件(*.tar.bz2)] -C [欲解压的目录]**
- -c：新建打包文件，可以搭配-v 查看过程中被打包的文件名
- -t：查看打包文件的内容含有那些文件名，终点在查看文件
- -x：解打包或解压缩的功能，可搭配-C 在特定目录解开
- **注意，c t x 是互斥的**
- **-j：通过 bzip2 的支持进行压缩/解压，此时文件名最好为*.tar.bz2**
- -z：通过 gzip 的支持进行压缩/解压，此时文件名最好是*.tar.gz
- -v：在压缩/解压缩过程中，将正在处理的文件名显示出来
- **-f filename：-f 后面接要被处理的文件名，建议-f 单独写一个参数**
- -C [目录名]：用于在解压时指定解压目录
- -p：保留备份数据原本权限与属性，常用语备份(-c)重要的配置文件
- -P：保留绝对路径，即允许备份数据中含有根目录存在之意

5、打包某些目录，但不包这些目录中的某些文件

- **tar -jcv -f [压缩创建的文件(*.tar.bz2)] --exclude=[文件名 1] --exclude[文件名 2]... [被压缩的文件或目录 1] [被压缩的文件或目录 2]**

- **tar -jcv -f /root/system.tar.bz2 --exclude=/root/etc* --exclude=/root/system.tar.bz2 /etc /root**

6、备份比某个时刻还要新的文件

- **tar -jcv -f[压缩创建的文件(*.tar.bz2)] --newer-mtime=[日期] [被压缩的文件 1]...**
- **tar -jcv -f /root.etc.newer.then.passwd.tar.bz2 --newer-mtime="2008/09/29" /etc/***

7、基本名称

- tarfile：仅用 tar 打包而没有进行压缩
  - **tar -cv -f [新建的打包文件名(.tar)] [被打包的文件 1]...**
- tarball：用 tar 打包，并进行了压缩

8、特殊应用，利用管道命令与数据流

- 通过标准输入输出的数据流重定向，以及管道命令的方式，将待处理的文件一边打包一边解压到目标目录中
- 例子：tar -cv -f - /etc | tar -xv -f - -C /tmp
- 注意：输出文件与输入文件都用'-'来代替，且存在'|'，分别代表 standard output，standstard input 与管道命令
- 可以将'-'暂时理解为内存中的一个设备(缓冲区)

10.4. 完整备份工具：dump

1、dump 可以针对整个文件系统进行备份之外还能针对目录来备份

10.4.1. <dump>

1、dump 处理可以备份整个文件系统之外，还可以制定等级

2、待备份的数据为单一文件系统：

- 文件系统可以使用完整的 dump 功能，包括利用 0-9 个数的 level 来备份，同时备份时可以使用挂载点或者设备文件名(如/dev/sda5 之类的设备文件名)来进行备份

3、待备份的数据只是目录：

- 所有备份数据都必须要在目录内
- 且仅能使用 level 0，即仅支持完整备份而已
- 不支持-u 参数，即无法创建/etc/dumpdates 这个 level 备份的时间记录文件

4、常用操作：

- dump [-Suvj] [-level] [-f 备份创建的文件] [待备份数据]
- -S：仅列出后面待备份数据需要多少磁盘空间才能够备份完毕
- -u：将这次 dump 的时间记录到/etc/dumpdates 文件中
- -v：将 dump 的文件过程显示出来
- -j：加入 bzip2 的支持，将数据进行压缩，默认 bzip2 压缩等级为 2
- -level：-0~-9 共 10 个等级
- -f：类似 tar，后面接产生的文件，可接如/dev/st0 等设备文件名
- -W：列出在/etc/fstab 里面的具有 dump 设置的分区是否有备份过

5、测试备份文件系统需要多少容量

- dump -S [待备份数据]

6、备份的简单例子：

- dump -0u -f /root/boot.dump /boot

7、查看有没有文件系统被 dump 备份过

- dump -W

8、备份目录

- dump -0j -f /root/etc.dump.bz2 /etc

9、一般来说，dump 不会使用包含压缩的功能，不过如果想要将备份的空间降低的话，可以加上-j 参数。

10.4.2. <restore>

1、简单用法

- restore -t [-f dumpfile] [-h]：用来查看 dump 文件
- restore -C [-f dumpfile] [-D 挂载点]：比较 dump 与实际文件
- restore -i [-f dumpfiile]：进入互动模式
- restore -r [-f dumpfile]：还原整个文件系统
- t、C、i、r 四个参数互斥
- -h：查看完整备份数据中的 inode 与文件系统 label 信息
- -f：后接要处理的 dump 文件
- -D：与-C 搭配，可以查出后面接的挂载点与 dump 内有不同的文件

2、用 restore 查看 dump 后的备份数据内容

- restore -t -f /root/boot.dump
- 查阅的数据其实显示出的是文件名与源文件的 inode 状态，因此 dump 会参考 inode 的记录，通过这个查询我们也能知道 dump 的内容为何

3、查看文件系统与备份文件之间的差异

- restore -C -f /root/boot.dump -D /boot
- restore -C -f /root/boot.dump **//好像会自动找到对应的文件系统挂载点？**

4、还原文件系统

- 由于 dump 是记录整个文件系统的，因此还原时应该要给予一个全新的文件系统才行
- 首先需要新建分区:fdisk /dev/sda ->n->w...
- 格式化:mkfs -t ext4 /dev/sda11
- 挂载：mount /dev/sda11 /mnt
- **cd /mnt //必须要进入挂载点下**
- restore -r -f /root/boot.dump **//将备份文件中的数据还原到本目录下，因此必须要更改目录到挂载点所在的那个目录才行，这样还原的文件才不会跑错地方**

5、仅还原部分文件的 restore 互动模式

- 某些时候你只是要将备份文件的某个内容找出来而已，并不想要全部解开
- 此时可以进入 restore 的互动模式(interactive mode)
- restore -i -f /root/etc.dump //进入互动模式，会进入到/root/etc.dump 这个文件中，所有操作都是在该文件中的

- help：查看命令
- ls [args]
- cd arg
- pwd
- add [args]：将 file 加入等一下要解压缩的文件列表中
- delete [args]：将 file 从解压缩列表中删除，并非删除文件
- extract：开始将刚才选择的文件列表解压缩
- quit：退出

10.5. 光盘写入工具

1、在命令行模式的刻录行为步骤：

- 先将所需要备份的数据构建成为一个镜像文件(iso)，利用 mkisofs 命令来处理
- 将该镜像文件刻录至光盘或 DVD 当中，利用 cdrecord 命令来处理

10.5.1. <mkisofs>:新建镜像文件

1、我们从 FTP 站下载下来的 Linux 镜像文件(不管是 CD 还是 DVD)都得要继续刻录成为光盘/DVD 后才能进一步的使用，包括安装或更新你的 Linux。同理，想要利用刻录机将你的数据刻录到 DVD 时，也得要将你的数据包成一个镜像文件，这样才能写入 DVD 片中

2、命令：

- mkisofs [-o 镜像文件] [-rv] [-m file] [待备份文件...] [-V vol] -graft-point isodir=systemdir ...
- -o：后接想要产生的镜像文件名
- -r 通过 Rock Ridge 产生支持 UNIX/Linux 的文件数据，可以记录较多的信息。**由于光盘的格式一般为 ISO9660,这种格式一般仅支持旧版 DOS 文件名，即文件名只能以 8.3(文件名 8 个字符，扩展名 3 个字符)的方式存在，加上-r 参数后，那么文件信息能够被记录的比较完整，可包括 UID/GID 与权限等**
- -v：显示构建 ISO 文件的过程
- -m file：-m 为派出文件(exclude)的意思，后面的文件不备份到镜像文件中
- -V vol：新建 Volume
- -graft-point：graft 有转嫁或移植的意思。由于一般默认的情况下，所有要被加到镜像文件中的文件都会被放置到镜像文件中的根目录，如此一来可能会造成刻录后的文件分类不易的情况，所以，可以使用-graft-point 这个参数，用如下方式定义位于镜像文件中的目录
  - /movies/=/srv/movies/ (将 Linux 的/srv/movies 内的文件加至镜像文件中的/movies/目录，注意，两个开头的/分别代表 Linux 的根目录与镜像文件的根目录)

10.5.2. <cdrecord>:光盘刻录工具

10.6. 其他常见的压缩与备份工具

10.6.1. 〈dd〉

1、dd 不但可以制作一个大文件，其最大的功效在于备份，因为 dd 可以读取磁盘设备的内容(几乎是直接读取扇区)，然后将整个设备被分成一个文件

2、命令：

- dd if="input file" of="output file" bs="block size" count="number"
- if：input file，也可以是设备
- of：output file，也可以是设备
- bs：规划的一个 block 的大小，若未指定则默认是 512bytes(一个扇区的大小)
- count：多少个 bs 的意思

3、范例：

- dd if=/etc/passwd of=/tmp/passwd.back
- 感觉上有点像 cp 命令
- dd if=/dev/sda of=/tmp/mbr.back bs=512 count=1 //将自己磁盘的第一个扇区备份下来
- dd if=/dev/sda6 of=/tmp/boot.whole.back //备份整个/dev/sda6

4、用途：将一个分区完整地复制到另一个分区中，由于要赋值启动扇区的区块，因此 cp 或者 tar 这种命令是无法达成需求的。并且新建分区后，不需要进行格式化就能直接进行扇区表面的复制，因为 dd 可以将原本旧的分区中扇区表面的数据整个复制过来，当然连同 super block，boot sector，meta data 等也会全部复制过来。

10.6.2. 〈cpio〉

1、cpio 可以备份任何东西，包括设备文件，不过 cpio 不会主动寻找文件来备份。因此 cpio 需要配合类似 find 等可以找到文件名的命令来告知 cpio 该备份的数据在哪里

2、命令：

- cpio -ovcB > [file|device] 备份
- cpio -ivcdu < [file|device] 还原
- cpio -ivct < [file|device] 查看
- 备份会用到的参数
  - -o：将数据 copy 输出到文件或设备上
  - -B：让默认的 blocks 可以增至 5120bytes，默认是 512bytes，这样做的好处是可以让大文件的存储速度加快
- 还原会用到的参数
  - -i：将数据子文件或设备复制到系统中
  - -d：新建目录，使用 cpio 所备份的数据内容不见得会在同一层目录中，因为我们必须要让 cpio 在还原时可以新建新目录，此时就得要-d 参数
  - -u：自动将较新的文件覆盖较旧的文件
  - -t：需要配合-i 参数，可以查看 cpio 新建的文件或设备的内容
- 可共享的参数

- -v：让存储的过程中文件名可以显示在屏幕上
- -c：一种较新的 portable format 方式存储

3、特点：

- 命令中没有指定需要备份的数据(通过与 find 协作来定位文件)
- cpio 会将数据整个显示到屏幕上，因此可以通过将这些屏幕的数据重新导向(>)一个新的文件

3、例子：

- find /boot | cpio -ocvB > /tmp/boot.cpio

Chapter 11. vim 程序编辑器

11.1. vi 与 vim

1、在 Linux 的系统中使用文本编辑器来编辑你的 Linux 参数配置文件可是一件很重要的事情，因此系统管理员至少应该熟悉一种文本编辑器

2、不同的 Linux distribution 各有其不同的附件软件，如果只会用此种附加软件来控制 Linux 系统时，当接管不同的 Linux distribution 会茫然失措

3、在 Linux 中，绝大部分的配置文件都是以 ASCII 的纯文本形式存在，因此利用简单的文字编辑器就能够修改设置了。

4、纯文本文件：记录的就是 0 与 1，而通过编码系统来将这些 0 与 1 转成我们认识的文字就是了

5、Linux 在命令行界面下的文本编辑器：Emacs，pico，nano，joe，vim

11.1.1. 为什么要学 vim

1、学习 vi 的理由：

- **所有的 UNIX Like 系统都会内置 vi 文本编辑器，其他文本编辑器不一定存在**
- **很多软件的编辑接口都会主动调用 vi(例 crontab，visudo，edquota 等命令)**
- vim 具有程序编辑能力，可以主动以字体颜色辨别语法的正确性，方便程序设计
- 程序简单，编辑速度相当快速

2、vim 是什么：

- 可以将 vim 视为 vi 的高级版本
- vim 可以用颜色或底线方式来显示一些特殊信息，vim 会根据文件的扩展名或者是文件内的开头信息判断该文件的内容而自动调用改程序的语法判断式，再以颜色来显示程序代码与一般信息
- vim 是个程序编辑器

11.2. vi 的使用

1、基本上 vi 分为三种模式：一般模式、编辑模式与命令模式

2、一般模式：

- 以 vi 打开一个文件就进入一般模式了(这是默认模式)
- 在这个模式中，你可以使用上下左右按键来移动光标，可以删除字符或删除整行，也可以复制粘贴你的文件数据

3、编辑模式：

- 在一般模式中，可以进行删除，复制，粘贴等操作，但是却无法编辑文件内容。
- 要等到你按下 i(I),o(O),a(A),r(R)等任何一个字母后才会进入编辑模式
- 通常在 Linux 中，按下这些键时，在界面的左下方会出现 INSERT(ioa)或 REPLACE(r)的字样，此时才可以进行编辑
- 如果要回到一般模式，必须按下 Esc 这个按键即可退出编辑器

4、命令行模式：

- 在一般模式中，输入":" "/" "?"这三个钟的任何一个按钮，就可以将光标

移动到最下面一行，在这个模式中，可以提供你查找数据的操作，而读取、保存、大量替换字符、离开 vi、显示行号等操作则是在此模式中完成的。

5、一般模式与编辑模式以及命令行模式可以相互切换，但是编辑模式与命令行模式之间不可相互切换

11.2.1. 简单执行范例

1、直接输入 vi [文件名]就能进入 vi 的一般模式，vi 后面必须要加上文件名，无论文件名是否存在

2、整个界面分为两部分，上半部与下面一行两者可以视为独立的

3、按下 i 进入编辑模式

- 在一般模式中只要按下 i、o、a 等字符就可以进入编辑模式
- 此时出了 Esc 这个按键之外，其他的按键都可以视作一般的输入了
- 在 vi 里，[Tab]键所得到的结果与空格符所得到的结果是不一样的

4、按下[Esc]键回到一般模式

5、在一般模式中输入":wq"保存后离开 vi

11.2.2. 按键说明<vi>

第一部分：一般模式可用的按钮说明，光标移动，复制粘贴，查找替换等

- h 或向左箭头：光标向左移动一个字符←
- j 或向下箭头：光标向下移动一个字符↓
- k 或向上箭头：光标向上移动一个字符↑
- l 或向右箭头：光标向右移动一个字符→
- 可以配合数字使用，如向右移动 30 个字符 30l 或 30(→)
- [Ctrl]+f：屏幕向下移动一页，相当于[Page Down]按键
- [Ctrl]+b：屏幕向上移动一页，相当于[Page Up]按键
- [Ctrl]+d：屏幕向下移动半页
- [Ctrl]+u：屏幕向上移动半页
- +：光标移动到非空格符的下一行
- -：光标移动到非空格符的上一行
- [n][space]：n 表示数字，再按空格键，光标会向右移动 n 个字符
- 0(数字零)或[home]：移动到这一行的最前面字符处
- $或功能键 end：移动到这一行最后面的字符处
- H：光标移动到屏幕最上方那一行的第一个字符
- M：光标移动到这个屏幕的中央那一行的第一个字符
- L：光标移动到这个屏幕的最下方那一行的第一个字符
- G：移动到这个文件的最后一行
- [n]G：n 为数字，移动到这个文件的第 n 行
- gg：移动到这个文件的第一行，相当于 1G
- [n][Enter]：光标向下移动 n 行
- /[word]：向下寻找一个名为 word 的字符串，支持正则表达式
- ?[word]：向上寻找一个名为 word 的字符串，支持正则表达式
  - n：重复前一个查找操作

- N："反向"进行前一个查找操作

- :[n1],[n2]s/[word1]/[word2]/g：在 n1 与 n2 行之间寻找 word1 这个字符串，并将该字符串替换为 word2，支持正则表达式
- :1,$s/[word1]/[word2]/g：从第一行到最后一行查找 word1 字符串，并将该字符串替换为 word2，支持正则表达式
- :1,$s/[word1]/[word2]/gc：从第一行到最后一行查找 word1 字符串，并将该字符串替换为 word2，且在替换前显示提示字符给用户确认是否替换，支持正则表达式
- x,X：在一行字当中，x 为向后删除一个字符(相当于[Del]键),X 为向前删除一个字符(相当于[Backspace])
- [n]x：连续向后删除 n 个字符
- dd：删除光标所在一整行
- [n]dd：删除光标所在的向下 n 行(包括当前这一行)
- d1G：删除光标所在到第一行的所有数据(包括当前这一行)
- dG：删除光标所在到最后一行的所有数据(包括当前这一行)
- d$：删除光标所在的字符到该行的最后一个字符(包括光标所指向的字符)
- d0(这是零)：删除从光标所在的字符到改行最前面一个字符(不包括光标所指向的字符)
- yy：复制光标所在那一行
- [n]yy：复制光标所在的向下 n 行(包括当前这一行)
- y1G：复制光标所在行到第一行的所有数据(包括当前这一行)
- yG：复制光标所在行到最后一行的数据(包括当前这一行)
- y0(这是零)：复制光标所在那个字符到该行第一个字符的所有数据(不包括光标所指向的字符)
- y$：复制光标所在那个字符到改行最后一个字符的所有数据(包括光标所指向的字符)
- p,P：p 将已复制的数据在光标的下一行粘贴，P 为粘贴在光标的上一行
- J：将光标所在行与下一行的数据结合成同一行
- c：重复删除多个数据，例如向下删除 10 行,10cj
- u：复原(撤销)前一个操作
- [Ctrl]+r：重做上一个操作
- .：重做上一个操作
- [Shift]+3：以暗黄色为底色显示所有指定的字符串
- :nohlsearch：取消[Shift]+3 的显示(no hightlight search)
- **q:：进入命令历史编辑**
- **q/：进入搜索历史编辑**
- **q[a-z]：q 后接任意字母，进入命令记录**
- **针对以上三个：**
  - **可以像编辑缓冲区一样编辑某个命令，然后回车执行**
  - **可以用 ctrl-c 退出历史编辑回到编辑缓冲区，但此时历史编辑窗口不关闭，可以参照之前的命令再自己输入**
  - **用:x 关闭历史编辑并放弃编辑结果回到编辑缓冲区**

- 可以在空命令上回车相当于退出历史编辑区回到编辑缓冲区

第二部分：一般模式切换到编辑模式的可用按钮说明

- **i,I：进入插入模式，i 为从光标所在处插入，I 为在目前所在行的第一个非空格处开始插入**
- **a,A：进入插入模式，a 为从目前光标所在的下一个字符处开始插入，A 为从光标所在行的最后一个字符处开始插入**
- **o,O：进入插入模式，o 为在目前光标所在的下一行处插入新的一行，O 为在目前光标所在处的上一行插入新的一行**
- **s,S：进入插入模式，s 为删除目前光标所在的字符，S 为删除目前光标所在的行**
- r,R：进入替换模式，r 只会替换光标所在的那一个字符一次，R 会一直替换光标所在行的文字，直到按下 Esc
- Esc：退回一般模式

第三部分：一般模式切换到命令行模式的可用按钮说明

- :w：将编辑的数据写入硬盘文件中
- :w!：若文件属性为只读时，强制写入该文件，不过到底能不能写入，还是跟你对该文件的文件属性权限有关
- :q：离开
- :q!：若曾修改过文件，又不想存储，使用"!"为强制离开不保存的意思
- :wq：保存后离开,若为:wq!则代表强制保存并离开
- ZZ：若文件没有变更，则不保存离开，若文件已经变更过，保存后离开
- :w [filename]将编辑文件保存称为另一个文件,注意 w 和文件名中间有空格
- :r [filename]在编辑的数据中，读入另一个文件的数据，即将 filename 这个文件的内容加到光标所在行后面，注意 r 和文件名之间有空格
- :[n1],[n2]w [filename]：将 n1 到 n2 的内容保存成 filename 这个文件，注意 w 和文件名中间有空格，[n2]与 w 之间可以没有空格
- :! [command]：暂时离开 vi 到命令模式下执行 command 的显示结果，例如:! ls /home
- :set nu：显示行号
- :set nonu：取消行号

11.2.3. vim 的保存文件、恢复与打开时的警告信息

1、目前主要的编辑软件都会有"恢复"功能，也即当你的系统因为某些原因导致类似死机的情况时，还可以通过某些特别的机制来让你将之前未保存的数据"救"回来。

2、当我们在使用 vim 编辑器时，vim 会在被编辑的文件的目录下再新建一个名为.filename.swp 的文件

3、情景模拟：

- cd /tmp
- mkdir vimtest

- vim man.config
- [Ctrl]+Z：将 vim 丢到后台执行
- kill -9 %1 <==模拟断线停止
- vim man.config <==会出现警告信息

4、问题一：可能有其他人或程序同时在编辑这个文件
- 原因：由于 Linux 是多人、多任务的环境，因此很有可能很多人在同时编辑这个文件
- 解决方法：找到那个程序或人员，请他讲该 vim 的工作结束，然后你继续处理
- 如果你只是要看该文件的内容并不会有任何修改编辑的行为，可以选择打开称为只读(O)文件，即警告界面反白部分输入英文 o 即可

5、问题二：在 vim 环境中，可能因为某些不明原因导致 vim 中断
- 原因：不正常结束 vim 所产生的结果
- 解决方法：按下 R，此时 vim 会载入*.swp 的内容，让你自己来决定要不要保存，这样就能够救回来你之前未保存的工作，但是那个*.swp 文件不会自动删除，所以离开 vim 后还得自行删除*.swp 文件，免得每次打开都会有警告
- 若确定*.swp 文件无用，那么直接按下 D 删除这个暂存文件，此时 vim 会载入指定文件，并将旧的*.swp 删除后新建新的*.swp 文件

6、警告界面参数介绍：
- [O]pen Read-Only：打开此文件成为只读文件
- [E]dit anyway：用正常的方式打开你要编辑的文件，并不会载入暂存文件的内容，不过很容易出现两个用户相互改变对方文件的问题
- [R]ecover：加载暂存文件的内容，用在你要救回来之前未保存的工作，当你救回来并保存离开 vim 后，还是要手动删除暂存文件
- [D]elete it：你确定暂存文件无用，打开前会删除暂存文件
- [Q]uit：按下 q 就离开 vim，不会进行任何操作回到命令提示符
- [A]bort：忽略这个编辑行为，感觉上与 quit 类似，也会送你回到命令提示符信息

11.3. vim 的功能

1、目前大部分 distributions 都以 vim 替代 vi 的功能了

2、如果使用 vi 后发现界面右下角有显示光标所在的行列号码，说明 vi 已经被 vim 替代了

3、vim 具有颜色显示功能，并且还支持许多程序语法，vim 可以帮助直接进行程序除错(debug)的功能

11.3.1. 块选择(Visual Block)

1、vi 的操作几乎都是以行为单位的

2、<vim>可以处理块范围的数据

3、按键：
- v：字符选择：会将光标经过的地方反白选择
- V：行选择：会将光标经过的行反白选择

- [Ctrl]+v：块选择，可以用长方形的方式选择数据
- y：将反白的地方复制
- d：将反白的地方删除

11. 3. 2. 多文件编辑

1、想要将文件 A 中的数据复制到文件 B 中去，由于 vim 都是独立的，无法在执行 A 文件时执行 nyy 再到 B 文件执行 p。可以使用鼠标圈选，但是会将[Tab]转为空格，并非所预期，这时候可以用多文件编辑

2、按键：

- vim [filename1] [filename2]... <==同时编辑多个文件
- :n：编辑下一个文件
- :N：编辑上一个文件
- :files：列出这个 vim 打开的所有文件

11. 3. 3. 多窗口功能

1、有一个文件非常大，在查阅后面的数据时，想要对照前面的数据，如果用翻页等命令([Ctrl]+f [Ctrl]+b)会显得很麻烦

2、在一般模式中输入命令"sp:"

- 若要打开同一文件，输入"sp"即可
- 若要代开其他文件，输入"sp [filename]"即可

3、窗口切换命令：

- [Ctrl]+w+↓：首先按下[Ctrl]，在按下 w，然后放开两个键，再按下↓切换到下方窗口
- [Ctrl]+w+↑：首先按下[Ctrl]，在按下 w，然后放开两个键，再按下↑切换到上方窗口

4、分屏

1) vim -On file1 file2... <==垂直分屏
2) vim -on file1 file2... <==左右分屏

- Ctrl+w & <==先 Ctrl 再 w，放掉 Ctrl 和 w 再按&，以下操作以此为基准

3) Ctrl+w c <==关闭当前窗口(无法关闭最后一个)
4) Ctrl+w q <==关闭当前窗口(可以关闭最后一个)
5) Ctrl+w s <==上下分割当前打开的文件
6) Ctrl+w v <==左右分割当前打开的文件
7) :sp filename <==上下分割并打开一个新的文件
8) :vsp filename <===左右分割，并打开一个新的文件
9) Ctrl+w l <===把光标移动到右边的屏中
10) Ctrl+w h <===把光标移动到左边的屏中
11) Ctrl+w k <===把光标移动到上边的屏中
12) Ctrl+w j <===把光标移动到下边的屏中
13) Ctrl+w w <===把光标移动到下一个屏中
14) Ctrl+w L <===向右移动屏幕
15) Ctrl+w H <===向左移动屏幕
16) Ctrl+w K <===向上移动屏幕

17) Ctrl+w J <===向下移动屏幕
18) Ctrl+w = <===让所有屏幕都有一样的高度
19) Ctrl+w + <===增加高度
20) Ctrl+w - <==减小高度

11.3.4. vim环境设置与记录:~/.vimrc,~/.viminfo

1、vim 会主动将你曾经做过的行为记录下来，好让你下次可以轻松作业，记录操作的文件就是~/.viminfo
2、整体 vim 的设置值一般放置在/etc/vimrc 这个文件中，不过不建议修改它，但是可以修改~/.vimrc 这个文件(默认不存在，手动创建)

11.3.5. vim常用命令示意图

P289

11.4. 其他vim使用注意事项

11.4.1. 中文编码问题

1、vim 里面可能无法显示正常的中文，这可能是因为编码的问题
2、中文编码有 gb2312，big5，utf8 等
3、vim 的终端界面使用的是统一的 utf8
4、考虑的东西：

- 你的 Linux 系统默认支持的语系数据:/etc/sysconfig/i18n(CentOS7 已改为/etc/locale.conf)
- 你的终端接口(bash)的语系，这与 LANG 变量有关
- 你的文件原本的编码
- 打开终端机的软件，例如在 GNOME 下面的窗口界面

5、情景假设：

- 假设文件编码为 big5，且使用环境是 Linux 的 GNOME，启动的终端界面为 GNOME-terminal 软件，那么可以这样修改：
- LANG=zh_CN.big5
- 然后在终端界面工具栏"终端机"-->"设置字符编码"-->"中文(繁体)(BIG5)"

11.4.2. DOS与Linux的断行字符

1、DOS(Windows 系统)的断行字符为^M$，称为 CR 与 LF 两个符号
2、Linux 系统的断行符号：$，称为 LF 符号
3、Linux 下面命令在开始执行时，判断依据是[Enter]，而 Linux 的[Enter]为 LF 符号，由于 DOS 的断行符号是 CRLF，也就是多一个^M，这样的情况下，如果一个 shell script 的程序文件，将可能造成程序无法执行的状态，因为它会错误判断程序所执行的命令内容。
4、解决方法：

- dos2UNIX [-kn] [file] [newfile]
- UNIX2dos [-kn] [file] [newfile]
- -k：保留该文件原本的 mtime 格式

- -n：保留原本的旧文件，将转换后的内容输出到新文件
- dos2UNIX -k -n man.config man.config.linux

11.4.3. 语系编码转换

1、想要将语系编码进行转换，例如将 big5 编码转换为 utf8

2、<iconv>：转换文件内容的编码格式

- iconv --list
- iconv -f [原本编码] -t [新编码] [file] [-o newfile]
- iconv -f [原本编码] -t [新编码] [file] > [newfile] mac 下不能用-o 参数
- iconv -f big5 -t utf8 vi.big5 -o vi.utf8
- 例子：将繁体中文的 utf8 转为简体中文的 utf8：

  iconv -f utf8 -t big5 vi.utf8 | \
  > iconv -f big 5 -t gb2312 | iconv -f gb2312 -t utf8 -o vi.gb.utf8

3、<convmv>：转换文件名的编码格式

- convmv -f [原本编码] -t [新编码] --notest [file]

11.5. 遇到的问题

11.5.1. vim 中文乱码问题

- vim ~/.vimrc
- 添加：set fileencodings=ucs-bom,utf-8,cp936,gb18030,big5,euc-jp,euc-kr,latin1
- :wq
- 完成！

11.5.2. 完全相同内容的文件在不同路径 vim 显示不同

- 问题：
  - vim /etc/yum.conf <==会以不同颜色高亮显示
  - cp /etc/yum.conf /tmp/yum.conf
  - vim /tmp/yum.conf <==以白色无高亮显示
- 暖神：由于某文件编码了文件路径
- 解决方法：
  - grep -r '/etc/yum.conf' /usr/share/vim
  - 搜索到了对应的文件后，用 vim 进入该文件
  - 找到对应的行，在合适的位置加上 vim /tmp/yum.conf
  - :wq
  - vim /tmp/yum.conf <==会以不同颜色高亮显示

Chapter 12. 认识与学习 bash

12.1. 认识 bash 这个 shell

1、管理整个计算机硬件的其实是操作系统的内核，这个内核是需要被保护的，所以我们一般用户只能通过 shell 来跟内核通信，以让内核达到我们所想要达到的工作

12.1.1. 硬件、内核与 shell

1、只要有操作系统，那么就离不开 shell

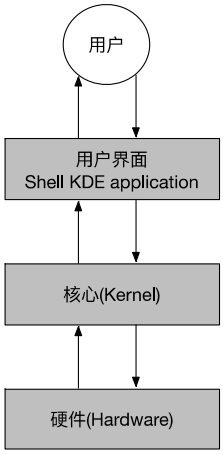
2、我们必须要通过"Shell"将我们输入的命令与内核通信，好让内核可以控制硬件来正确无误地工作

3、操作系统其实是一组软件，由于这组软件在控制整个硬件与管理系统的活动监测，如果这组软件能被用户随意操作，若用户应用不当，将会使整个系统崩溃

4、shell 的功能只是提供用户操作系统的一个接口，因此这个 shell 需要可以调用其他软件才行

5、狭义的 shell：命令行方面的软件，包括 shell 等

6、广义的 shell：包括图形界面的软件，因为图形界面其实也能够操作各种应用程序来调用内核工作



12.1.2. 为什么要学命令行界面的 shell

1、命令行界面的 shell：大家都一样

- 图形化界面虽然亲善，功能虽然强大，但是它是将利用的软件都集成在一起的一组应用程序而已，并非是一个完整的套件，所以某些时候当你升级或者是使用其他套件管理模块时，就会造成设置的困扰了，不同的 distribution 所设计的界面也都不同，也会造成困扰
- 命令行界面的 shell 不同，各家 distribution 使用的 bash 都是一样的，如此就可以轻松转换不同 distributions

2、远程管理：命令行界面就是比较快

- Linux 管理经常需要通过远程联机，而联机时的命令行的传输速度一定比较快，而且较不易出现断线或者是信息外流的问题

3、Linux 的任督二脉：shell

12.1.3. 系统的合法 shell 与/etc/shells 功能

1、早年 UNIX 年代，发展着众多，所以由于 shell 依据发展者的不同就有许多的版本，例如 Bourne SHell(sh)、Sun 默认的 C SHel、商业常用的 K SHell 等等。

2、Linux 使用的这一种版本称为 Bourne Again SHell(bash)，这是 Bourne Shell 的增强版，也是基于 GUN 的架构下开发出来的

3、shell 的历史：

- 第一个流行的 shell：Bourne shell，由 Steven Bourne 发展出来的
- C shell：语法有点类似 C 语言，简称 csh
- 由于 Linux 是由 C 语言编写的，很多程序员使用 C 来开发软件，因此 C shell 相对就很热门了

4、Linux 默认使用 bash

5、**由于系统某些服务在运行过程中回去检查用户所能使用的 shells，而这些 shell 的查询就是借助/etc/shells 这个文件，因此需要将系统上合法的 shell 写入/etc/shells 这个文件中**

12.1.4. bash shell 的功能

1、bash 是 GUN 计划中重要的工具软件之一，目前也是 Linux distributions 的标准 shell

2、bash 主要兼容于 sh，并且依据一些用户需求而加强的 shell 版本

3、bash 的优势

- **命令记忆能力：**
  - 在命令行中按上下键就可以找到前/后一个输入的命令
  - 命令记录位置：主文件夹内的.bash_history 中，即~/.bash_history
  - 注意该文件记录的是前一次登陆以前所执行的命令，而至于这一次登陆所执行的命令都被暂存在临时内存中，当你成功注销系统后(exit)，该命令记忆才会记录到该文件中
  - 可以修改环境变量 HISTSIZE 来更改保存的历史命令的条数
- **命令与文件补全功能([Tab]按键的好处)：**
  - [Tab]接在一串命令的第一个字的后面，则为命令补全
  - [Tab]接在一串命令的第二个字以后时，则为文件补齐
- **命令别名设置功能(alial)：**
  - alias lm='ls -al' <==设置别名
  - alias lm <==查看别名
  - unalias lm <==删除别名
- **作业控制、前台、后台控制(job control ,foreground,backbroud)：**
  - **Chapter 17**
- **脚本程序(shell script)：**
  - **Chapter 13**
- **通配符：**

12.1.5. bash shell 的内置命令：type

1、为了方便 shell 操作，bash 内置了很多命令，包括 cd,umask 等

2、type 可以查看命令是属于内置命令还是外部命令

- type [-tpa] [command]
- -t：type 会将 command 以下面这些字眼显示出它的意义：
  - file：表示外部命令
  - alias：表示该命令为命令别名所设置的命令
  - builtin：表示该命令为 bash 内置命令
- -p：如果 command 为外部命令时，才会显示完整文件名
- -a：会由 PATH 变量定义的路径中，将所有含 command 的命令都列出来，包含 alias

12.1.6. 执行的命令

1、[Enter]：开始执行

2、反斜线：

- \[Enter]：对[Enter]进行转义，让[Enter]不再具有开始执行的功能，好让命令在下一行继续输入
- 切记反斜线\和[Enter]必须紧挨着，否则转义的就不是[Enter]了

12.2. shell 的变量功能

12.2.1. 什么是变量

- 简单地说就是让某一个特定字符串代表不固定的内容。
- 变量具有可变性与方便性
- 影响 bash 环境操作的变量：
  - PATH 变量：在任何目录下执行某个命令与 PATH 这个变量有很大关系
    - 例如执行 ls 这个命令，系统就是通过 PATH 这个变量里面的内容所记录的路径顺序来查找命令
- 脚本程序设计(shell script)的好帮手
- 定义：变量就是以一组文字或符号等，来替代一些设置或者是一串保留的数据
- 显示变量：echo

12.2.2. 变量的显示与设置：echo，unset

1、变量的显示：<echo>

- echo $variable
- echo $PATH
- echo ${PATH}

2、变量设置规则

- **在 bash 中，当一个变量名称尚未被设置时，默认的内容是"空"的**
- 变量与变量内容以等号"="来连接
- **等号两边不能接空格！！！，否则会被当做命令，出现 command not found 的错误**
- 变量名只能是英文字母与数字与下划线，但是开头字符不能是数字
- 变量内容若有空格符，可以使用双引号"或者单引号'将变量内容结合起来
  - 双引号内的特殊字符如$等，可以保有原本的特性，在双引号内部的单

引号就是一般的单引号，即 echo "'${PATH}'" <==输出为单引号包围的字符串

- var="lang is $LANG" 则 echo $var ==> lang is en_US
- 单引号内的特殊字符进位一般字符(纯文本)
- ar='lang is $LANG' 则 echo $ ar ==> lang is $LANG
- 即双引号仍然可以保持变量的能容，而单引号内仅能是一般字符，而不会有特殊字符

➢ 可用转义字符\将特殊符号变成一般字符
➢ **在一串命令中，还需要通过其他的命令提供信息，可以使用单反引号``或$(命令)**
- version =$(uname -r)

➢ var=$((运算内容)) <==用于计算表达式的值
➢ 若该变量为增加变量内容时，则可用"$变量名"或${变量
➢ }累加内容
- PATH="$PATH":/home/bin <==红色的双引号是必须的，否则无法区分变量名与添加内容的部分
- PATH=${PATH}:/home/bin

➢ 若该变量需要在其他子进程执行，则需要以 export 来使变量成环境变量
- export PATH

➢ 通常大写字符为系统默认变量，自行设置变量可以使用小写字符

2、取消设置的变量
➢ unset [name]

3、<echo>
➢ -n：可以不断行继续在同一行显示

12. 2. 3. 环境变量的功能

1、环境变量可以帮助我们达到很多功能
➢ 主文件夹的变换
➢ 提示符的显示
➢ 执行文件查找的路径

2、env 查看环境变量与常见环境变量说明
➢ HOME：代表用户的主文件夹，例如可以使用 cd ~回到自己的主文件夹，就是使用这个变量
➢ SHELL：告知目前这个环境使用的是哪个 shell 程序，Linux 默认使用/bin/bash
➢ HISTSIZE：历史命令记录的条数
➢ MAIL：使用 mail 命令在收信时系统会去读取的邮件信箱文件
➢ PATH：执行文件查找路径，目录与目录中间以冒号分隔，由于文件的查找是依据由 PATH 变量内的目录来查询的，所以目录的顺序也是很重要的
➢ LANG：语系数据
➢ RANDOM：随机数变量
- 大多数 distributions 会有随机数生成器,/dev/random 这个文件

- 可以通过$RANDOM 来随机取得整数值，BASH 环境下，这个 RANDOM 的内容介于 0-32767 之间

3、用 set 查看所有变量

- set
- set | less <==以 less 的方式查看输出，因为太多了
- 所有的环境变量与自定义变量都会输出
- 一般来说，勿乱是否为环境变量，只要跟我们目前这个 shell 的操作接口有关的变量，通常都会被设置为大写字符，也就是说，基本上，Linux 默认的情况中，使用{大写字母}来设置的变量一般为系统内定需要的变量
- PS1(提示符设置)
  - \d：可显示出星期日月的格式，如"Mon Feb 2"
  - \H：完整的主机名
  - **\h：仅取主机名字第一个小数点之前的名字**
    - **如何修改这个 hostname???**
  - \t：显示时间，为 24 小时格式的"HH:MM:SS"
  - \T：显示时间，为 12 小时格式的"HH:MM:SS"
  - \A：显示时间，为 24 小时格式的"HH:MM"
  - \@：显示时间，为 12 小时格式的"am\pm"样式
  - \u：用户账号名称
  - \v：BASH 的版本信息
  - \w：完整的工作目录，由根目录写起，但主文件会以~替代
  - \W：利用 basename 函数取得工作目录名称，仅列出最后一个目录名
  - \#：执行的第几个命令
  - \$：提示符，如果是 root，提示符为#，否则就是$
- $
  - $本身也是个变量，代表目前 shell 的线程代号，就是 PID(Process ID)
  - **echo $$ 查询即可**
- ?
  - 上一个命令传回的值
  - 一般来说，成功执行该命令，会传回 0 值
  - 执行错误传回非零值
  - echo $?
- OSTYPE,HOSTTYPE,MACHTYPE(主机硬件与内核的等级)
  - CPU 主要分为 32 位于 64 位
  - 32 位可分为 i386,i586,i686
  - 64 位则为 x86_64
  - 高级的硬件通常会向下兼容旧的软件，但是较高级的软件可能无法再旧机器上安装，比如可以在 x86_64 硬件上安装 i386 的 Linux 操作系统，但是无法再 i686 硬件上安装 x86_64 的 Linux 操作系统

4、export：自定义变量转为环境变量

- **环境变量与自定义变量的区别：在于变量是否会被子进程所继续引用**
- 当你登陆 Linux 并取得一个 bash 后，你的 bash 就是一个独立的进程，被称为 PID，接下来在这个 bash 下面执行的任何命令都是由这个 bash 衍生

出来的，那些被执行的命令就被称为子进程

- 子进程仅会继承父进程的环境变量，子进程不会继承父进程自定义的变量
- export [变量
-]
- export <==不加任何参数，显示所有的环境变量

12.2.4. 影响显示结果的语系变量(locale)

1、命令

- locale -a：查询支持的所有语系
  - 所有语系文件都放在/usr/lib/locale/这个目录中
- locale：查询所有语系变量的设置情况

2、基本上可以逐一设置每个与语系有关的变量数据，但如果其他语系变量都未设置，但设置过 LANG 或者 LC_ALL 时，其他的语系变量就会被这两个变量所替代。因此我们在 Linux 中通常说明仅设置 LANG 这个变量而已，因为它是最主要的设置变量

12.2.5. 变量的有效范围

- 如果在跑程序的时候，有父进程与子进程的不同程序关系时，则变量是否可以被引用与 export 有关。
- 被 export 的变量，我们可以称它为环境变量，环境变量可以被子进程引用，而其他自定义变量内容就不会存在于子进程中(有点类似于全局变量与局部变量的关系)
- 环境变量与 bash 的操作环境意思不太一样，例如，PS1 并不是环境变量，但是这个 PS1 会影响到 bash 的接口(提示符)

12.2.6. 变量键盘读取

1、<read>

- 读取来自键盘输入的变量
- 最常被用于 shell script 的编写中
- read [-pt] variable
- -p：后面可接提示符
- -t：后面可接等待的"秒数"
- read -p [提示符字符串] -t [秒数] [变量名]
- read -p "Please keyin your name: " -t 30 named

2、<declare>/<typeset>

- 声明变量的类型
- declare <==bash 会主动将所有变量名称与内容都调出来，就像 set 一样
- declare [-aixrp] [变量名]
- -a：将后面的变量定义成为数组类型
- -i：将后面的变量定义称为整数类型
- -x：用法与 export 一样，将后面的变量变成环境变量
- -r：将变量设置称为 readonly 类型，该变量不可被更改内容，也不能重

设

- -p：列出变量的类型
- declare +x [变量
-]：可以取消操作
- 如果将变量设置为只读，通常需要注销再登陆才能复原该变量的类型

3、默认的情形

- 变量类型默认为"字符串"，所以若不指定变量类型 1+2 是一个字符串而不是计算式
- bash 环境中的数值运算，默认最多仅能达到整数类型，所以 1/3 是 0

4、数组变量类型：

- ary[0]="small small"
- ary[1]="small"
- ...
- echo ${ary[0]} ${ary[1]}

12.2.7. 与文件系统及程序的限制关系：ulimit

1、bash 可以限制用户的某些系统资源，包括可以打开的文件数量，可以使用的 CPU 时间，可以使用的内存总量

2、<ulimit>

- ulimit [-SHacdfltu] [配额]
- -H：hard limit，严格的限制，必定不能超过这个设置的数值
- -S：soft limit，警告的设置，可以超过这个设置值，但是若超过则有警告信息
- -a：后面不接任何参数，可列出所有的限制额度
- -c：某些进程发生错误时，系统可能会将该进程在内存中的信息写成文件(排错用)，这种文件就被称为内核文件(core file)。此为限制每个内核文件的最大容量
- -f：此 shell 可以创建的最大文件容量(一般可能设置为 2GB)单位为 KB
- -d：进程可以使用的最大断裂内存(segment)容量
- -l：用于锁定(lock)的内存量
- -t：可使用的最大 CPU 时间(单位秒)
- -u：单一用户可以使用的最大进程(process)数量

12.2.8. 变量内容的删除、替代与替换

1、变量内容的删除与替换<echo>

- echo ${变量#关键字}
- echo ${变量##关键字}
- echo ${变量%关键字}
- echo ${变量%%关键字}
- 符号说明：
  - echo 删除模式必不可少，否则后面那一串会被当成 command，从而引发 command not found 的错误
  - $：必不可少，删除模式必须存在

- #：符合替换文字的"最短的"那一个，从前往后匹配
- ##：符合替换文字的"最长的"那一个，从前往后匹配
- %：符合替换文字的"最短的"那一个，从后往前匹配
- %%：符合替换文字的"最长的"那一个，从后往前匹配

2、替换

- echo ${变量/旧字符串/新字符串}
- echo ${变量//旧字符串/新字符串}
- 符号说明：
  - echo 删除模式必不可少，否则后面那一串会被当成 command，从而引发 command not found 的错误
  - $：必不可少，删除模式必须存在
  - /：第一个旧字符串会被新字符串替换
  - //：所有旧字符串会被新字符串替换

3、变量测试与内容替换

变量设置方式	str 没有设置	str 为空字符串""	str 已设置为非空字符串
var=${str-expr}	var=expr	var=""	var=$str
var=${str:-expr}	var=expr	var=expr	var=$str
var=${str+expr}	var=""	var=expr	var=expr
var=${str:+expr}	var=""	var==""	var=expr
var=${str=expr}	str=expr var=expr	str 不变 var=""	str 不变 var=$str
var=${str:=expr}	str=expr var=expr	str=expr var=expr	str 不变 var=$str
var=${str?expr}	expr 输出至 stderr	var=""	var=str
var=${str:?expr}	expr 输出至 stderr	expr 输出至 stderr	var=str

12.3. 命令别名与历史命令

1、<clear>可以清除界面

12.3.1. 命令别名设置：alias, unalias

- <alias>：不加任何参数，查看目前有哪些命令别名
- alias 的配置文件:~/.bashrc
- alias [别名]='[命令参数]'
  - 当命令参数含有空格时，需要将命令参数用"括起来，否则会产生 command not found 的错误
  - alias lm='ls -al'
  - **等号两边不能有空格！！！，因为该操作类似于变量的赋值**
  - 当命令参数没有空格是，可以不用"，但最好还是用吧！
- <unalias> [别名]：将别名去掉

- 命令别名与变量的区别：
  - 命令别名是新创一个新的命令，可以直接执行该命令
  - 变量则需要使用类似 echo ${变量名}才能调用变量内容

12.3.2. 历史命令

1、<history>

- history [n]
- history [-c]
- history [-raw] [histfiles]
- n：代表数字，列出最近 n 调命令
- -c：将目前 shell 中所有 history 内容全部消除
- -a：将目前新增的 history 命令新增入[histfiles]中，若没有[histfiles]参数则默认写入~/.bash_history 中
- -r：将 histfiles 内容读到目前这个 shell 的 history 记忆中
- -w：将目前 history 记忆内容写入 histfiles 中

2、<!>

- ![number]：执行第 number 条命令
- ![command]：由最近的命令向前搜寻命令串开头为 command 的命令，并执行
- !!执行上一条命令

12.4. Bash Shell 的操作环境

12.4.1. 路径与命令查找顺序

1、命令运行顺序

- 以相对/绝对路径执行命令，例如"/bin/ls [- 参数] [文件
-]"，或"./ls [- 参数] [文件
-]"
- 由 alias 找到该命令来执行
- 由 bash 内置(builtin)命令来执行
- 通过$PATH 这个变量的顺序找到第一个命令来执行

12.4.2. bash 的登陆与欢迎信息：/etc/issue, /etc/motd

- 登陆界面的字符串写在**/etc/issue** 里面
- 与$PS1 类似，这个文件的内容可以使用反斜杠作为变量调用
  - \d：本地端时间的日期
  - \l：显示第几个终端机接口
  - \m：显示硬件等级(i386/i486...x86_64)
  - \n：显示主机的网络名称
  - \o：显示 domain name
  - \r：操作系统的版本
  - \t：显示本地端时间的时间
  - \s：操作系统名称(会显示 Linux)

- \S：操作系统名称(会显示 CentOS Linuex 7 (Core))
- \v：操作系统版本

➢ 登陆后取得的信息写在/etc/motd 中

12. 4. 3. bash 的环境配置文件

1、系统有一些环境配置文件的存在，让 bash 在启动时直接读取这些配置文件，以规划好 bash 的操作环境，这些配置文件可以分为全体系统的配置文件以及用户个人偏好配置文件

2、注意，前几小结谈到的命令别名，自定义的变量在你注销 bash 就会失效，如果要保留这些设置必须写入配置文件才行

3、login 与 non-login shell

- login shell：取得 bash 时需要完整的登陆流程，就称为 login shell，例如，你要由 tty1~tty6 登陆，需要输入用户的账号与密码，此时取得的 bash 就称为 login shell
- non-login shell：取得 bash 接口的方法不需要重复登陆的举动，例如，你以 GNOME 登陆 Linux 后，再以图形界面启动终端机，此时终端机接口并没有需要再次输入账号与密码，这个 bash 的环境就称为 non-login shell，你在你原本的 bash 环境下再次执行 bash 这个命令，同样没有输入账号密码，第二个 bash(子进程)也是 non-login shell
- **login shell 与 non-login shell 取得的 bash 读取的配置文件是不一样的**

4、login shell(只会读取两个配置文件)

1) /etc/profile：这是系统的整体设置，最好
2) 要修改这个文件
3) ~/.bash_profile 或~/.bash_login 或~/.profile：属于个人用户设置，你要改自己的数据，就写入这里

5、**/etc/profile(login shell 才会读)**

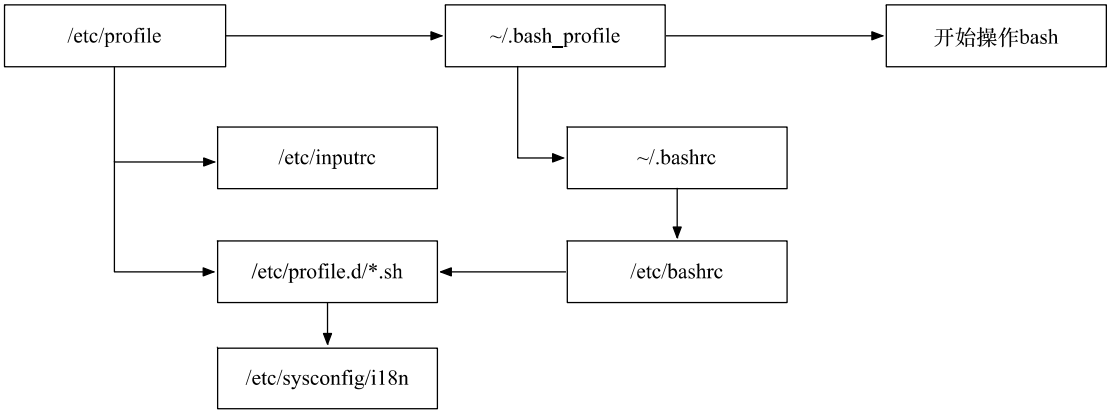
- 每个用户登录取得 bash 时一定会读取的配置文件
- **想要修改所有用户的整体环境，就在这里修改**
- **主要变量：**
  - PATH：会根据 UID 决定 PATH 变量要不要含有 sbin 的系统命令目录
  - MAIL：依据账号设置用户的 mailbox 到/var/spool/mail/账号名
  - USER：根据用户的账号设置此变量内容
  - HOSTNAME：根据主机的 hostname 命令决定此变量的内容
  - HISTSIZE：历史命令记录条数
  - **在 bash 中修改变量的值，在 exit 注销后，重新登录，又会重新读取配置文件的值**
- 而且会依次调入一下数据
  - /etc/inputrc：这个文件并没有被执行，/etc/profile 会主动判断用户有没有自定义输入的按键功能，如果没有的话，/etc/profile 就会决定设置"INPUTRC=/etc/inputrc"这个变量，此文件的内容为热键，[Tab]有无音效等数据
  - /etc/profile.d/*.sh：**只要在/etc/profile.d/这个目录内且扩展名为.sh,另外，用户具有 r 权限，那么该文件就会被/etc/profile 调用。**这个目录

下面的文件规定了 bash 操作接口的颜色，语系,ll 与 ls 命令的命令别名，vi 的命令别名，which 命令别名等。**如果需要帮所有用户设置共享的命令别名时，可以在这个目录下面创建扩展名为.sh 的文件，并将所有需要的数据写入即可**

- /etc/sysconfig/i18n???：这个文件是由/etc/profile.d/lang.sh 调用的。这决定 bash 默认使用何种语系的重要配置文件，文件里最重要的就是 LANG 这个变量的设置

6、**~/.bash_profile(login shell 才会读)**

- bash 在读完了整体环境设置的/etc/profile 并借此调用其他配置文件后，接下来会读取用户个人配置文件，在 login shell 的 bash 环境中，所读取的个人偏好配置文件主要有三种：
  - ~/.bash_profile
  - ~/.bash_login
  - ~/.profile
  - 实际上 bash 的 login shell 设置只会读取上述三个文件中的其中一个，读取顺序按照上述顺序，读取第一个存在的文件



7、<source>(或小数点)：读取环境配置文件的命令

- 由于/etc/profile 与~/.bash_profile 都是在取得 login shell 的时候才会读取的配置文件，所以如果你讲自己的偏好设置写入上述文件后，通常得注销再登陆后设置才会生效，**也可以用 source 直接生效**
- source [配置文件名]
- . [配置文件名]

8、**~/.bashrc(non-login shell 会读)**

- 当你取得 non-login shell 时，该 bash 配置文件仅会读取~/.bashrc 而已
- 此外，~/.bashrc 文件中还会主动调用/etc/bashrc 这个文件(仅限 CentOS)，该文件帮我们的 bash 定义下面的数据：
  - 依据不同的 UID 规定的 umask 的值
    - UID：用户标识号，它与用户名唯一对应，root 的 UID 为 0
  - 依据不同的 UID 规定的提示符(PS1 变量)
  - 调用/etc/profile.d/*.sh

9、其他相关配置文件

- /etc/man.config???(现在是/etc/man_db.conf)：规定了执行 man 的时候该

去哪里查看数据的路径设置

- ~/.bash_history：历史命令记录
- ~/.bash_logout：记录了当注销 bash 后系统再帮我完成什么操作后才离开

12. 4. 4. 终端机环境设置

1、我们可以利用退格来删除命令上的字符，可以用[Ctrl]+C 来强制终止一个命令的运行，当输入错误的时候会有声音警告，这是因为登陆终端机的时候会自动取得一些终端机的输入环境的设置

2、<stty>(setting tty)

- stty -a
- -a：将所有 stty 参数列出来
- 重要参数
  - eof：End of file 的意思，代表输入
  - erase：向后删除字符
  - intr：送出一个 interrupt(中断)的信号给目前正在运行的程序
  - kill：删除在目前命令上的所有文字
  - quit：送出一个 quit 信号给正在运行的程序
  - start：在某个进程停止后，重新启动它的输出
  - stop：停止目前屏幕的输出
  - susp：送出一个 termianl stop 的信号给正在运行的进程
- stty erase ^? ：设置热键

3、<set>：设置整个命令输出/输入的环境

- set [-uvCHhmBx]
- -u：默认不启用，启用后，当使用未设置的变量时，显示错误信息
- -v：默认不启用，启用后，在讯息被输出前，先显示信息的原始内容
- -x：默认不启用，若启用后，在命令被执行前，会显示命令内容(前面有++符号)
- -h：默认启用，与历史命令有关
- -H：默认启用，与历史命令有关
- -m：默认启用，与工作管理有关
- -B：默认启用，与括号[]的作用有关
- -C：默认不启用，使用>等时，若文件存在时，文件不会被覆盖
- echo $- ：显示目前 set 的值
- set [+uvCHhmBx]：可以取消设置

4、其他按键设置功能在/etc/inputrc 文件中

5、默认组合键：

- [Ctrl]+C：终止目前命令
- [Ctrl]+D：输入结束(EOF),例如邮件结束的时候
- [Ctrl]+M：就是[Enger]
- [Ctrl]+S：暂停屏幕的输出
- [Ctrl]+Q：恢复屏幕的输出
- [Ctrl]+U：在命令提示符下，将整行删除
- [Ctrl]+Z：暂停目前命令

12.4.5. 通配符与特殊符号

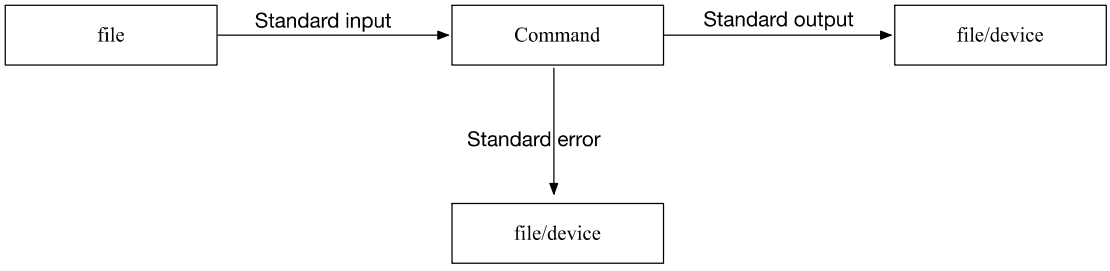
- 通配符
  - *：代表 0 到无穷个任意字符
  - ?：代表一个任意字符
  - []：同样代表一定有一个在括号中的字符，例如[abcd]可能是 a,b,c,d 中的任意一个
  - [-]：若有减号在括号中间时，代表在编码顺序内的所有字符，例如[0-9]代表 0-9 之间的所有数字，如果要包含字符'-'，要使用转义
  - [^]：只要括号中的第一个字符为'^'，那表示原向选择，例如[^abc]代表一定有一个字符，只要是非 a,b,c 的其他字符
- 特殊字符表：
  - #：注释符号
  - \：转义符号，将特殊字符或通配符还原成一般字符
  - |：管道(pipe)，分割两个管道命令的界定
  - ;：连续命令执行分隔符，连续性命令的界定
  - ~：主文件夹
  - $：使用变量前导符，即变量之前需要加的变量替代值
  - &：作业控制(job control)，将命令变成背景下工作
  - !：逻辑运算意义上的非(not)
  - /：目录符号，路径分隔的符号
  - >,>>：数据流重定向，输出导向，分别是"替换"与"累加"
  - <,<<：数据流重定向，输入导向
  - ''：不具有变量置换功能
  - ""：具有变量置换功能
  - ``：可以先执行的命令，也可以用$()
  - ()：在中间为子 shell 的起始与结束
  - {}：在中间为命令块的组合

12.5. 数据流重定向

- **将某个命令执行后应该要出现在屏幕上的数据传输到其他地方，例如文件或者设备**

12.5.1. 什么是数据流重定向

1、命令执行示意图



- 执行一个命令时，这个命令可能会由文件读入数据，经过处理后再将数据输出到屏幕
- standard output 与 standard input 代表标准输出与标准错误输出，这两个

命令都是默认输出到屏幕上面来的

2、standard output 与 standard error output

- 简单地说，标准输出指的是命令执行所传回的正确的信息，而标准错误输出可以理解为命令执行失败后，所传回的信息
- 数据重定向可以将 standard output(stdout)与 standard error output(stdeer)分别传送到其他的文件或设备
  - 标准输入(stdin)：代码为 0，使用<或<<
  - 标准输出(stdout)：代码为 1，使用>或>>
  - 标准错误输出(stdeer)：代码为 2，使用 2>或 2>>

3、命令：

- [完整的命令] > [文件
-]，例如 ll / > /tmp/out.txt
  - **若该文件不存在，系统会自动创建**
  - 若该文件存在，系统会先将这个文件清空，然后再写入
  - 也就是说以>输出到一个已存在的文件中，会覆盖掉原先的内容
- [完整的命令] >> [文件
-]，追加输出，例如 ll / > /tmp/out.txt
- 仅有>或>>则默认代表 1>,1>>
- 将 stdout 于 stderr 分别存入不同的文件
  - [完整命令] > [文件名 1] 2> [文件名 2]

4、/dev/null：垃圾桶黑洞设备遇特殊写法

- /dev/null 这个设备可以吃掉任何导向这个设备的信息
- **如果要讲正确与错误的信息写入同一个文件中去，需要用特殊的写法**
  - [完整命令] > [文件名] 2>&1 <==建议用这个
  - 注意
  - 能用>>&1
  - [完整命令] &> [文件名]

5、standard input：<与<<

- <：将原本需要由键盘输入的数据改由文件内容来代替
  - cat > [文件] **<==利用 cat 创建文件的流程，即使存在会覆盖，输入结束需要利用[Ctrl]+D 来退出输入**
  - cat > [文件] < [需要读入的文件]
- <<：代表结束输入的意思
  - cat > [文件] <<"eof" <==当输入 eof 按下[Enger]时(当前一行不能有其他字符)等效于按下[Ctrl]+D

12. 5. 2. 命令执行的判断依据：；，&& ||

1、；

- 不考虑命令的相关性连续执行命令
- command1 ; command2

2、$?(命令传回码)与&& ||

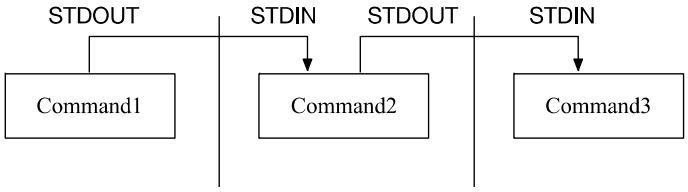
- command1 &&command2
  - 若 command1 执行完毕且正确执行($?=0)，

- 若 command1 执行完毕且为错误($?≠0)，则不执行 command2

- command1 || command2
  - 若 command1 执行完毕且正确执行($?=0)，则不执行 command2
  - 若 command1 执行完毕且为错误($?≠0)，则开始执行 command2
- Linux 下&&与||并没有优先级顺序，命令都是从左到右执行

12.6. 管道命令(pipe)

- bash 命令执行的时候有输出的数据会出现
- 如果这群数据必须要挺过几道手续之后才能得到我们所想要的格式，就会涉及到管道命令
- 管道命令使用的是'|'这个界定符号
- 管道命令与连续执行命令是不一样的
- 这个管道命令'|'**仅能处理有前面一个命令传来的正确信息**，也就是 standard output 的信息，对于 standard error 并没有直接处理的能力
- **每个管道后面接的第一个数据必须是命令，而且这个命令必须能够接受 standard input 的数据才行，例如 less，more，head，tail 等都可以接受 standard input 的管道命令**



12.6.1. 选取命令:cut, grep

1、一般来说，选取信息通常是针对"行"来分析的，并不是整篇信息分析的

2、<cut>

- 这个命令可以将一段信息的某一段"
- "出来，处理的信息以"行"为单位
- cut -d '分隔符' -f fields
- cut -c 字符范围
- **-d：后接分隔字符，与-f 一起使用**
- **-f：根据-d 的分隔字符将一段信息切割成数个字段，用-f 取出第几个字段**
- -c：以字符的单位取出固定字符区间(索引从 1 开始)
- echo $PATH | cut -d ':' -f 5
- export | cut -c 12- <==第 12 个字符(包括)到最后一个字符
- export | cut -c 12-20 <==第 12 个字符(包括)到第 20 个字符(包括)
- cut 在处理多空格相连的数据时，会比较麻烦，因为只能指定准确的分隔符，分隔符不能带有正则表达式

3、<grep>

- grep 分析一行信息，若当前有我们所需要的信息，就将该行拿出来
- grep [-acinvr] [--color==auto] '查找的字符串' filename
- -a：将 binary 文件以 text 文件的方式查找数据
- -c：计算找到'查找字符串' 的次数

- -i：忽略大小写的不同
- n：顺便输出行号
- -v：反向选择，即输出没有'查找字符串'内容的哪一行
- -r：在指定目录中递归查找
- --color==auto：将找到的关键字部分加上颜色
- grep -r [--color==auto] '查找的字符串' [目录名]
- 高级命令参见 grep 的一些高级参数

12.6.2. 排序命令：sort，wc，uniq

1、<sort>

- sort [-fbMnrtuk] [file or stdin]
- -f：忽略大小写的差异
- -b：忽略最前面的空格符部分
- -M：以月份的名字来排序，例如 JAN,DEC 等排序方法
- -n：使用"纯数字"进行排序(默认是以文字类型来排序的)
- -r：反向排序
- -u：就是 uniq，相同的数据中，仅出现一行代表
- -t：分隔符，默认使用[Tab]来分隔
- -k：以哪个区间(field)来进行排序的意思
- cat /etc/passwd | sort <==**默认以第一个字符来排序**
- cat /etc/passwd | sort -t ':' -k 3 <==以':'分隔每一行，输出第三列

2、<uniq>

- uniq [ic]
- -i：忽略大小写的差异
- -c：进行计数
- 排序完了之后想要将重复的数据仅列出一个显示
- last | cut -d ' ' -f 1 | sort |uniq <==与 last | cut -d ' ' -f 1 | sort -u 完全一致
- last | cut -d ' ' -f 1 | sort | uniq -c <==显示数量，必须借助 uniq

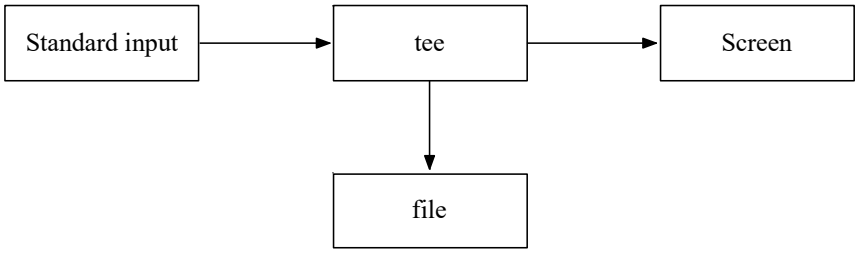
3、<wc>

- wc [-lwm]
- -l：列出行数量
- -w：列出多少字(英文单词)
- -m：列出多少字符

12.6.3. 双重定向：tee

1、>、>>等会将数据流传送给文件或设备，因此除非去读取该文件或设备，否则就无法继续利用这个数据流，如果我们想要将这个数据流的处理过程中将某段信息存下来，可以利用 tee

2、tee 工作流程示意图



3、tee 会将数据流送与文件与屏幕(screen)，输出到屏幕的就是 stdout 可以让下个命令继续处理(>,>>会截断 stdout，从而无法以 stdin 传递给下一个命令)

4、命令：<tee>

- tee [-a] file
- -a：以累加(append)的方式，将数据加入到 file 当中
- command | tee [文件名] | command

12.6.4. 字符串转换命令:tr, col, join, paste, expand

1、<tr>

- tr 可以用来删除一段信息当中的文字，或者是进行文字信息的替换
- tr [-ds] '字符串'
- -d：删除信息当中的'字符串'
- -s：替换掉重复的字符
- last | tr '[a-z]' '[A-Z]'
- 可以用 tr 来出去^M，其中^M 可以用\r 来代替

2、<col>

- col [-xb]
- -x：将 tab 键转换成对等的空格键
- -b：在文字内有反斜杠\时，仅保留反斜杠后接的那个字符

3、<join>

- 处理两个文件之间的数据，主要是将两个文件中有相同数据的那一行加在一起，第二个文件的相同字段不会重复显示
- join [-ti12] file1 file2
- -t：**join 默认空格分隔符分隔数据，并且对比"第一个字段"的数据**，如果两个文件相同，则将两条数据连成一行，**且第一个字段放在第一个**
- -i：忽略大小写差异
- -1(数字)：代表第一个文件要用哪个字段来分析
- -2：代表第二个文件要用那个字段来分析
- 相同的字段会被放在第一个(若字段本身
- 在第一个，也会挪到第一个)
- 特别注意，在使用 join 之前，你所需要处理的文件应该要事先经过排序处理，否则有些对比的项目会被略过

4、<paste>

- 比 join 简单许多，paste 会将两行直接贴在一起，且中间以[tab]键隔开
- paste [-d] file1 file2
- -d：后面可以接分隔符，默认是[tab]

5、<expand>：

- 将[tab]键转换成空格键

- expand [-t] file
- -t：后面可接数字，一般来说一个[tab]按键可用 8 个空格来替换，我们可以自定义一个[tab]代表几个字符
- <unexpand>可以将空格符转换成[tab]

12.6.5. 切割命令：<split>

1、如果文件太大，导致一些携带式设备无法复制的问题
2、split 可以帮你将一个大文件依据文件大小或行数来切割称为小文件
3、split [-bl] file PREFIX

- -b：后面可接欲切割成的文件大小，可加单位例如 bB,kK,mM 等
- -l：以行数来进行切割
- PREFIX：代表前导符，可作为切割文件的前导文字，可以随便取，只要写上前导文字，小文件就会以 xxxaa,xxxab,xxxac 的方式来新建小文件

4、例子

```
dd if=/dev/zero of=/tmp/bigfile bs=1M count=100
cd /tmp; split -b 1m /tmp/bigfile splittest  <==会新建到当前文件夹中
cat splittest* >> splittestback  <==将小文件合成一个大文件
```

12.6.6. 参数代换：<xargs>

12.6.7. 关于减号-的用途

1、管道命令在 bash 的连续处理程序中是相当重要的，在 log file 的分析当中也是相当重要的一环
2、在管道命令中经常会用到前一个命令的 stdout 作文这次的 stdin，偶写命令需要用到文件名来进行处理，该 stdin 与 stdout 可以利用减号-来替代
3、tar -cv -f - /home | tar -xv -f - -C /tmp

Chapter 13. 正则表达式与文件格式化处理

13.1. 什么是正则表达式

13.1.1. 什么是正则表达式

1、正则表达式就是处理字符串的方法，它是以行为单位来进行字符串的处理行为，正则表达式用过一些特殊符号的辅助，可以让用户轻易达到查找、阐述、替换某特定字符串的处理程序

2、正则表达式基本上是一种"表示法"，只要工具程序支持这种表示法，那么该工具程序就可以用来作为正则表达式的字符串处理之用

13.1.2. 正则表达式对于系统管理原的用途

1、增加管理资源的效率

13.1.3. 正则表达式的广泛用途

1、用于邮件服务器，过滤垃圾邮件

2、目前两大邮件服务器软件 sendmail 与 postfix

3、等等其他用途

13.1.4. 正则表达式与 Shell 在 Linux 当中的角色定位

1、正则表达式要背的，其重要程度类似于 99 乘法口诀表

13.1.5. 扩展的正则表达式

1、正则表达式的字符串表示方式依据不同的严谨度而分为基础正则表达式与扩展正则表达式

2、扩展型正则表达式处理简单的一组字符串处理之外，还可以做组的字符串处理，通过特殊字符"("与"|"等的协助来完成

13.2. 基础正则表达式

1、正则表达式是处理字符串的一种表示方法，对字符排序有影响的语系数据就会对正则表达式的结果有影响

2、最简单的字符串选取的工具程序：grep

13.2.1. 语系对正则表达式的影响

1、文件记录的仅有 0 与 1，我们看到的字符文字与数字都是通过编码表转换来的，不同语系的编码数据并不相同，所以会造成数据选取结果的区别。

- LANG=C 时:0 1 2 3 4 … A B C D … Z a b c d …z
- LANG=zh_CN 时：0 1 2 3 4 a A b B … z Z
- 因此当使用[A-Z]时对于两种不同的语系，产生的选取结果是不同的

2、**使用正则表达式时，需要特别留意当时环境的语系为何，否则可能会发现与别人不同的选取结果**

3、一般我们在练习正则表达式时，使用的是兼容于 POSIX 的标准，因此就是用"C"这个语系

4、为了避免语系造成的选取不同问题，可以采用特殊字符

- [:alnum:] <==代表英文大小写字符及数字即 0-9,A-Z,a-z

- [:alpha:] <==代表任何英文大小写字符，即 A-Z,a-z
- [:blank:] <==代表空格键与[Tab]按键
- [:cntrl:] <==代表键盘上面的控制按键，即包括 CR,LF,Tab,Del 等
- [:digit:] <==代表数字而已，即 0-9
- [:graph:] <==除了空格符(空格键与[Tab]按键)外其他所有按键
- [:lower:] <==代表小写字符，即 a-z
- [:print:] <==代表任何可以被打印出来的字符
- [:punct:] <==代表标点符号,即" ; ? ; : # $
- [:upper:] <==代表大写字符，即 A-Z
- [:space:] <==代表会产生空白的字符，包括空格键[Tab]CR 等
- [:xdigit:] <==代表十六进制的数字类型，包括 0-9,A-F,a-f 的数字与字符

13. 2. 2. grep 的一些高级参数

1、命令

- grep [-A] [-B] [--color=auto] '搜寻字符串' [文件名]
- -A：后面可加数字，为 after 的意思，除了列出该行外，后面的 n 行也列出来
- -B：后面可加数字，为 before 的意思，除了列出该行外，前面的 n 行也列出来
- --color=auto 将正确的那个选取的数据列出颜色
- 其他命令参见选取命令:cut,grep

2、grep 在数据中查找一个字符串时，是以整行为单位来进行数据的选取的

13. 2. 3. 基础正则表达式练习

P350-P356

1、[]：

- 代表一个字符，[]里面的字符，只要匹配到其中一个就算匹配
- '[abcd...z]' <==有 abcd...z 其中一个就算匹配
- '[a-z]' <==含义同上
- '[[:lower:]]' <==含义同上

2、[^]

- 代表一个字符，[^]里面的字符，只有所有都不匹配才算匹配成功
- '[^abcd]' <除了 abcd...z 之外才算匹配
- '[^a-z]' <==含义同上

3、行首行尾字符^和&

- **^在[]内部代表反选，在[]外部代表行首，整个的反选用参数-v 来控制**
- '^[a-z]' <==匹配行首任何小写字符
- '\.$' <==匹配行尾的'.'
- '^$' <==代表空白行

4、任意一个字符.与重复字符*

- 通配符的*可以匹配任意(0 或多个)字符
- 但是正则表达式并不是通配符，两者并不等价
- .(小数点)：代表绝对有一个任意字符的意思

- 比如 a*代表：有 0 个到无穷多个 a
- 如果要表示至少含有一个 a：aa*

➢ *(星号)：代表重复前一个 **RE(regular expression)字符** 0 到无穷多次的意思，为组合形态
- 如果要表示通配符那样的意义，可以使用'.*'

5、限定连续 RE 字符范围\{\}
➢ 由于'{'与'}'在 shell 是有特殊意义的(语句块)，因此必须要转义才能使用它
➢ 'a\{2\}' <==匹配出现 2 次 a 的行，**注意，这个与 aaa 也是匹配的**

13. 2. 4. 基础正则表达式

1、正则表达式特殊字符表
➢ ^word：待查找的字符串 word 在行首
➢ word&：待查找的字符串 word 在行尾
➢ .：代表一定有一个任意字符
➢ *：重复零个到无穷多个的前一个字符
➢ [list]：从字符集合的 RE 字符里找到想要选取的字符
➢ [n1-n2]：从字符集合的 RE 字符里找到想要选取的字符范围
➢ [^list]：从字符集合的 RE 字符里面找出不包含这些 RE 字符的的字符串或范围
➢ \{n,m\}：连续 n 到 m 个前一个 RE 字符
- \{n\}：连续 n 个前一个 RE 字符
- \{n,\}：连续 n 个及以上前一个字符

2、ls 不支持正则表达式，但是可以用 ls | grep 来配合使用
3、**注意区别正则表达式的通配符*与一般命令的通配符***

13. 2. 5. <sed>工具

1、sed 本身也是一个管道命令，可以分析 standard input 的，而且 sed 还可以将数据进行替换、新增、选取特定行等功能
2、命令：
➢ sed [-nefr] [动作] [文件] or STD IN | sed [-nefr] [动作]
➢ ***接的动作必须以两个单引号括住***
➢ **-n：使用安静(silent)模式，在一般 sed 中，所有来自 STDIN 的数据一般都会被列到屏幕上，加了参数-n 后，只有经过 sed 特殊处理的那一行才会被列出来**
➢ -e：直接在命令行模式上进行 sed 的动作编辑
➢ -f：直接将 sed 的动作写在一个文件内，-f filenmae 则可以执行 filename 内的 sed 动作
➢ -r：sed 的动作支持的是扩展正则表达式的语法
➢ **-i：直接修改读取的文件内容，而不是由屏幕输出**
➢ 动作说明:[n1 [,n2]]function
- 对于可接字符串的参数，**可以紧挨着或者加个空格，效果一样，建议加个空格，意思更明显一些**
- a：新增，a 的后面可接字符串，而这些字符串会在新的一行出现(目前

行的下一行)

- c：替换，c 的后面可接字符串，这些字符串可以替换 n1,n2 之间的行
- d：删除，后面通常不接任何参数
- i：插入，i 的后面可接字符串，而这些字符串会在新的一行出现(目前行的上一行)
- p：打印，也就是将某个选择的数据打印出来，通常 p 会与参数 sed -n 一起运行
- s：替换，可以直接进行替换的工作，通常这个 s 可以搭配正则表达式，**例如 1,20s/lod/new/g (跟 vim 里面的很像！！！)**

3、例子

- sed '2d' example <==删除 example 文件第二行
- sed '2,$d' example <==删除第二至最后一行**(注意这里$是代表最后一行的意思，而 vim 里面的操作以及正则表达式中$都代表行末)**
- sed '$d' example <==删除 example 文件最后一行
- sed '/test/d' example <==删除 example 文件包含 test 的所有行
- sed 's/test/mytest/g' example <==在整行范围内把 test 替换为 mytest。如果没有 g 标记，则只有每行第一个匹配的 test 被替换成 mytest
- sed -n 's/^test/mytest/p' example <==(-n)选项和 p 标志一起使用表示只打印那些发生替换的行。也就是说，如果某一行开头的 test 被替换成 mytest，就打印它
- sed 's/^192.168.0.1/&localhost/' example <==&符号表示替换换字符串中被找到的部份。所有以 192.168.0.1 开头的行都会被替换成它自己加 localhost，变成 192.168.0.1localhost
- sed -n 's/\(love\)able/\1rs/p' example <==love 被标记为 1，所有 loveable 会被替换成 lovers，而且替换的行会被打印出来
- sed -n '/test/,/check/p' example-----所有在模板 test 和 check 所确定的范围内的行都被打印。**逗号作用：选定行的范围**
- sed -n '5,/^test/p' example-----打印从第五行开始到第一个包含以 test 开始的行之间的所有行
- sed '/test/,/check/s/$/sed test/' example-----对于模板 test 和 west 之间的行，每行的末尾用字符串 sed test 替换

4、sed 可以直接修改文件内容(危险操作)

- 对于一个非常大的文件，要在第 100 行修改内容，此时如果用 vim 会很困难，因为文件太大了，利用 sed 就可以直接修改/替换的功能，甚至不需要 vim 去修改
- sed -i [动作] [文件名]

13.3. 扩展正则表达式

1、RE 字符：

- +：重复一个或一个以上前一个 RE 字符
- ?：0 个或一个前一个 RE 字符
- |：用或(or)的方法找出数个字符串

- ()：找出"组"字符串,(单用()好像没有意义???)
- ()+：多个重复组的判别

13.4. 文件格式化与相关处理

13.4.1. 格式化打印<printf>

1、在很多时候，我们需要将自己的数据给它格式化输出的
2、printf 可以帮助我们将数据输出的结果格式化
3、printf '打印格式' [实际内容]

- **打印格式用单引号(或双引号)括住**
- \a：警告声音输出
- \b：退格键
- \f：清除屏幕
- \n：输出新的一行
- \r：即[Enter]键
- \t：水平的[tab]键
- \v：垂直的[tab]键
- \xNN NN 为两位数的数字，将数字转换为字符
- C 语言程序内常见的变量格式
  - %[n]s <==s 代表 string，即多少个字符
  - %[n]i <==i 代表 integer，即多少整数字数
  - %[N].[n]f <==f 代表 floating(浮点)，N 代表有效位数，n 代表小数点长度，Nn 冲突时(N
  - 足以满足 n 时)，以 n 为准

4、由于 printf 不是管道命令，因此我们要讲文件内容先提取出来给 printf 作为后续的数据才行

- printf '打印格式' $(cat printf.txt)
- printf '打印格式' $(cat printf.txt | grep -v 'Name')

13.4.2. <awk>：好用的数据处理工具

1、相比于 sed**(管道命令)**常常作用于一整行的处理，**awk(管道命令)则比较倾向于将一行分成数个"字段"来处理，因此 awk 相当适合处理小型的数据处理**
2、命令

- awk '条件类型 1{动作 1} 条件类型 2{动作 2}...' [filename]
- **注意 awk 后续的所有动作都是以单引号括住的，而且如果以 print 打印时，非变量的文字部分，包括格式等都需要以双引号的形式定义出来，因为单引号已经是 awk 的命令固定用法了**
- 典型例子：last -n 5 | awk '{print $1 "\t" $3}'
- 每一行的字段都是由变量名称的，那就是$1,$2...其中$0 代表整行

3、awk 处理流程：

- 读入第一行，并将第一行的数据填入$0,$1,...等变量中
- 依据条件类型的限制，判断是否需要进行后面的动作
- 做完所有的动作与条件类型
- 若还有后续的'行'的数据，重复上面的步骤，直到所有数据都读取完为止

4、awk 内置变量：

- NF：每一行($0)拥有的字段总数
- NR：目前 awk 所处理的是第几行
- FS：目前分隔字符，默认是空格键
- 在动作内部引用这些变量不需要用$
- last -n 5 | awk '{print $1 "\t lines: " NR "\t columes: " NF}'

5、awk 逻辑运算符

- >
- <
- >=
- <=
- ==
- !=
- cat /etc/passwd | awk '{FS=":"} $3<10 {print $1 "\t" $3}'
- 注意上面这句，对于第一行来说，分隔符还是空格，分隔符从第二行开始才变成":"，利用 BEGIN 关键字可以预先设定 awk 的变量
- cat /etc/passwd | awk 'BEGIN {FS=":"} $3<10 {print $1 "\t" $3}'
  - BEGIN 和$3<10 作为两个条件类型
  - 除了 BEGIN 还有 END

6、注意事项

- 所有 awk 的动作，即在{}内的动作，如果有需要多个命令辅助时，可以用分号";"间隔，或者直接以[Enter]按键来隔开每个命令
- 与 bash、shell 不同，在 awk 中，内置变量(NF NR FS)可以直接使用，不需要加上$

13. 4. 3.　　文件比较工具

1、文件比较通常是同一个软件的不同版本之间，比较配置文件与源文件的区别，通常是用在 ASCII 纯文本文件的比较上

2、常用的是 diff 和 cmp，其中 cmp 用于比较非纯文本文件，diff 用于比较 ASCII 纯文本文件

3、<diff>

- diff 用在比较两个文件之间的区别，**并且是以行为单位来比较的**，一般是用在 ASCII 纯文本文件的比较上
- diff 通常是用在同一的文件(或软件)的新旧版本区别上
- diff [-bBi] [文件 1] [文件 2]
- [文件 1] [文件 2]可以用-来替换
- -b：忽略一行当中仅有多个空白的区别
- -B：忽略空白行的区别
- -i：忽略大小写的不同
- 文件区别的意义：文件 2 在文件 1 的基础上做出的改动
  - 3a4：[文件 1]的第 3 行的下一行增加'某字符串(会在下面列出)'就与[文件 2]的第 4 行相同了
  - 5c6：[文件 1]的第 5 行，与[文件 2]的第 6 行的内容发生了变动，变动的内容会在下面显示

- **前一个数字代表文件 1 中的行号**
- **中间的字符与 sed 中 function 参数的意思完全相同**
- **后一个数字代表文件 2 中的行号**

4、<cmp>

- 比较两个文件，利用字节单位去比较，因此可以比较二进制文件
- 与 diff 以行为单位进行比较不同
- cmp [-s] [文件 1] [文件 2]
- -s：将所有不同点的字节处都列出来，因为 cmp 默认列出第一个发现的不同点

5、<patch>

- patch 与 diff 有密不可分的关系
- diff 用来分辨两个版本之间的区别
- 升级文件：线比较新旧版本的区别，并将区别文件制作成补丁文件，再由补丁文件更新旧文件即可
- diff **-Naur** passwd.old passwd.new > passwd.patch
- 一般来说，用 diff 制作出来的比较文件通常使用扩展名.patch
- patch -p[N] < [补丁文件]
- patch -R -p[N] < [补丁文件]
- -p：后面的 N 表示取消几层目录的意思
- -R：代表还原，将新的文件还原成原来旧的版本
- 由 diff 制作的.patch 文件包含有新旧文件的信息(包含文件名),因此不必添加新旧文件的文件名，即可实现更新或者还原操作
- "-bash:patch:command not found"：解决方法：yum -y install patch

13. 4. 4. 文件打印准备：<pr>

1、pr [文件
] <==不带任何参数的版本，带有的参数包括"文件时间","文件名","页码"

2、参数太多，自行 man

Chapter 14. 学习 shell script

14.1. 什么是 shell script

1、shell script(程序化脚本)字面意义可以分为两部分

- shell：命令行界面下让我们与系统沟通的一个工具接口
- script：脚本的意思

2、shell script 是利用 shell 的功能所写的一个"程序"，这个程序是使用纯文本文件，将一些 shell 的语法与命令(含外部命令)写在里面，搭配正则表达式，管道命令与数据流重定向等功能，以达到我们所想要的处理目的

3、shell script 就是像早起 DOS 年代的批处理文件(.bat)，最简单的功能就是将许多命令写在一起，让用户很轻易就能一下子处理复杂的操作(执行一个文件"shell script"，就能够一次执行多个命令)

4、shell script 可以提供数组、循环、条件与逻辑判断等重要功能，让用户可以直接以 shell 来编写程序，而不必使用类似 C 语言等传统程序编写的语法

5、shell script 可以简单被看成是批处理文件，也可以被说成是一个程序语言，且这个程序语言由于都是利用 shell 与相关工具命令，所以不需要编译即可执行，且拥有不错的排错(debug)工具

14.1.1. 为什么学习 shell script

1、帮助了解 Linux 的来龙去脉

2、shell script 的功能

- 自动化管理的重要依据
  - 管理一台主机需要每天完成的任务：查询登陆文件，追踪流量，监控用户使用主机状态，主机各项硬件设备状态，主机软件更新查询等
  - 写个简单的程序来帮助每日自动处理分析代替自行手动处理
- 追踪与管理系统的重要工作
  - Linux 系统的服务(services)启动的接口是在/etc/init.d/这个目录下，目录下的所有文件都是 script
- 简单入侵检测功能
  - 当我们系统有异常时，大多会将这些异常记录在系统记录器，也就是我们常提到的"系统注册表文件"
- 连续命令单一化
- 简易的数据处理
  - awk 可以用来处理数据，但是 shell script 功能更强大
- 跨平台支持与学习历程较短
  - 几乎所有的 UNIX Like 都可以跑 shell script，连 Windows 系列也有相关的 script 仿真器

3、虽然 shell script 号称程序，但实际上，shell script 在处理数据的速度上是不太够的，因为 shell script 用的是外部的命令与 bash shell 的一些默认工具，它经常会去调用外部的函数库，因此命令周期上面不比传统的程序语言。

4、shell script 在系统管理上面是很好的一项工具，但在处理大量数值运算上就不太好了

14.1.2. 第一个 script 的编写与执行

1、注意事项

- 命令的执行是从上到下，从左而右地分析与执行
- 命令的执行：命令、参数间的多个空白都会被忽略掉
- 空白行也将被忽略掉，并且[Tab]所得的空白视为空格键
- 如果读到一个[Enter]符号(CR)，就尝试开始执行该行(或该串)命令
- 如果一行内容太多，则可以使用\[Enter]来扩展至下一行
- #可视为批注，任何加在#后面的数据将全部视为批注文字而被忽略

2、如何执行

- **直接命令执行：shell.sh 文件必须要具备可读与可执行(rx)权限，可以使用 chmod a+x [文件名]来获取可执行权限**
  - 绝对路径：使用/home/dmtsai/shell.sh 来执行命令
  - 相对路径：假设工作目录在/home/dmtsai/，则使用./shell.sh 来执行
  - 变量"PATH"功能，将 shell.sh 放在 PATH 指定的目录内，例如~/bin/
- 以 bash 进程来执行，通过 bash shell.sh 或 sh shell.sh 来执行
  - /bin/sh 就是/bin/bash(链接文件)
  - 使用 sh shell.sh 就是告诉系统，我想要直接以 bash 的功能来执行 shell.s
  - 此时你的 shell.sh 只需要有 r 权限即可被执行

3、script 组成分析

- **第一行#!/bin/bash**
  - 声明这个 script 使用 shell 名称
  - 因为我们使用的是 bash，所以必须要以"#!/bin/bash"来声明这个文件内的语法使用 bash 的语法，那么当这个程序被执行时，它就能够加载 bash 的相关环境配置文件(一般来说就是 non-login shell 的~/.bashrc)，并且执行 bash 来使我们下面的命令能够执行
  - 在很多情况中，如果没有设置好第一行，那么该程序很可能会无法执行，因为系统可能无法判断改程序需要使用什么 shell 来执行
- **程序的说明内容**
  - 整个 script 当中，除了第一行的#!用于声明 shell 之外，其他的#都是"批注"的用途
- **主要环境变量的声明**
  - 建议务必将一些重要的环境变量设置好，PATH 和 LANG 是最重要的，如此一来可以让我们这个程序在进行时可以直接执行一些外部命令，而不必写绝对路径
- **程序的主要部分**
- **告知执行结果**
  - 我们可以利用 exit 这个命令来让程序中断，并且传回一个数值给系统，接着利用 echo $?可以得到 0 的值

4、执行，以 sh01.sh 为例

- sh sh01.sh
- bash sh01.sh
- chmod a+x sh01.sh;./sh01.sh

14.1.3. 编写 shell script 的良好习惯

1、在每个 script 的文件头处记录

- script 的功能
- script 的版本信息
- script 的作者与联络方式
- script 的版权声明方式
- script 的 History(历史记录)
- script 内较特殊的命令，使用"绝对路径"的方式来执行
- script 执行时需要的环境变量预先声明与设置

2、程序编写最好使用嵌套格式，最好能以[tab]按键的空格缩排

3、编写 script 的工具最好使用 vim 而不是 vi，因为 vim 会有额外的语法检验机制，能够在第一阶段编写时就能发现语法方面的问题

14.2. 简单的 shell script 练习

14.2.1. 简单范例

1、交互式脚本：变量内容由用户决定

- read -p [提示符字符串] -t [秒数] [变量名]

2、随日期变化：利用日期进行文件的创建

- date=$(date --date='**1 days ago**' **+%Y%m%d**)
- touch ${filename}${date}

3、数值运算：简单的加减乘除

- bash shell 默认仅支持到整数的数据
- declare 定义变量的类型
- var=$((运算内容)) <==没看错，两层括号

14.2.2. script 的执行方式区别(source ，shscript,./script)

1、不同的 script 执行方式会造成不一样的结果，尤其对 bash 的环境影响很大

2、我们还可以用 source (或小数点.)来执行，source 还用于读取配置文件 bash 的环境配置文件

3、各种执行方式的介绍

- 利用直接执行的方式执行 script
  - 使用直接命令执行(绝对\相对\添加到 PATH)或是利用 bash(或 sh)来执行脚本，**script 都会使用一个新的 bash 环境来执行脚本内的命令，也就是说 script 是在子进程的 bash 内执行**
- 利用 source 来执行脚本：**在父进程中执行**
  - **这就是为什么，用 source 来读取配置文件而不是使用 bash**

14.3. 善用判断式

1、条件判断配合$?来决定后续命令是否进行

2、更简单的方式来进行"条件判断":test

14.3.1. 利用<test>命令的测试功能

1、命令介绍

- test -e /dmtsai && echo "exist" || echo "Not exist"
- 关于某个文件名的"文件类型"判断，如 test -e filename
  - **-e：该文件名是否存在**
  - **-f：该文件名是否存在且为文件**
  - **-d：该文件名是否存在且为目录**
  - -b：该文件名是否存在且为一个 block device 设备
  - -c：该文件名是否存在且为一个 character device 设备
  - -S：该文件名是否存在且为一个 Socket 文件
  - -p：该文件名是否存在且为以 FIFO(pipe)文件
  - -L：该文件名是否存在且为一个链接文件
- 关于文件的权限检测，如 test -r filename
  - -r：检测该文件名是否存在且具有"可读"的权限
  - -w：检测该文件名是否存在且具有"可写"的权限
  - -x：检测该文件名是否存在且具有"可执行"的权限
  - -u：检测该文件名是否存在且具有"SUID"的属性
  - -g：检测该文件名是否存在且具有"GUID"的属性
  - -k：检测该文件名是否存在且具有"Sticky bit(SBIT)"的属性
  - -s：检测该文件名是否存在且为"非空白文件"
- 两个文件之间的比较，如 test file1 -nt file2
  - -nt：(newer than) 判断 file1 是否比 file2 新
  - -ot：(older than) 判断 file1 是否比 file2 旧
  - -ef：判断 file1 与 file2 是否为同一文件，可用在判断 hard link 的判断上，主要意义在于判断两个文件是否均指向同一个 inode
- 测试的标志，如 test n1 -eq n2
  - -eq：两数值相等(equal)
  - -ne：两数值不等(not equal)
  - -gt：n1 大于 n1(greater than)
  - -lt：n1 小于 n2(less than)
  - -ge：n1 大于等于 n2(greater than or equal)
  - -le：n1 小于 n2(less than or equal)
- 判定字符串的数据
  - test -z string：判定字符串是否为空，空返回 true
  - test -n string：判断字符串是否为空，非空返回 true
  - test str1=str2：判断 str1 是否等于 str2，**好像有问题，使用中括号正确**
  - test str1!=str2：判断 str1 是否等于 str2，**好像有问题，使用中括号正确**
- 多重条件判定
  - -a：两个条件同时成立，返回 true，如 test -r file1 -a -x file2
  - -o：任一条件成立，返回 true，如 test -r file1 -o -x file2
  - !：反向状态

14. 3. 2. 利用判断符号 []

1、使用中括号必须特别注意，因为中括号用在很多地方，包括通配符与正则表达式，所以如果要在 bash 的语法当中使用括号作为 shell 的判断式时，**必须要注**

意在中括号的两端需要有空格符来分隔

- [□"$HOME"□==□"$MALL"□]
- 括号内的每个组件都需要有空格键来分隔
- 在中括号内的变量，最好都以双引号括起来
- 在中括号内的常量，最好都以单引号括起来

2、中括号的使用方法与 test 几乎一模一样，**只是中括号比较常用在条件判断式子 if...then...fi 的情况**

3、不支持正则表达式???

14.3.3. shell script 默认的变量($0, $1)

1、命令可以带有参数，那么 shell script 能不能在脚本文件名后带有参数呢

2、script 针对参数已经设置好一些变量名

- /path/to/scriptname opt1 opt2 opt3 opt4
- $0 $1 $2 $3 $4

3、在 script 内用来调用参数的特殊变量

- $#：代表后接的参数"个数"，不包括$0
- $@：代表"$1"，"$2"，"$3"的意思，每个变量是独立的，用(双引号括起来)
- $*：代表$1c$2c$3c$4，其中 c 为分隔字符，默认为空格键

4、例子

- bash sh07.sh theone haha quot

5、<shift>：造成参数变量号码的偏移

- 初始：
  - /path/to/scriptname opt1 opt2 opt3 opt4 opt5 opt6
  - $0 $1 $2 $3 $4 $5 $6
- shift //默认偏移 1
  - /path/to/scriptname opt2 opt3 opt4 opt5 opt6
  - $0 $1 $2 $3 $4 $5
- shift 3
  - /path/to/scriptname opt5 opt6
  - $0 $1 $2

14.4. 条件判断式

14.4.1. 利用 if...then

1、单层、简单条件判断式

- if [条件判断式]; then
- 当条件判断式成立时，可以进行的命令工作内容
- fi <==将 if 反过来写，结束 if 之意

2、条件判断式写入中括号之中，注意空格

3、可以将多个条件写入一个中括号之内，还可以写多个中括号，中括号之间用&&以及||来隔开

- &&代表 AND
- ||代表 OR

- ["$yn" == "Y" -o "$yn" == "y"] ==> ["$yn" == "Y"]||["$yn" == "y"]
- 之所以这样改是习惯问题，建议这样

4、多重、复杂条件判断式

- if [条件判断式]; then
- 当条件判断式成立时，可以进行的命令工作内容
- else
- 当条件判断式不成立时，可以进行的命令工作内容
- fi

5、更复杂的情况

- if [条件判断式一]; then
- 当条件判断式一成立时，可以进行的命令工作内容
- elif [条件判断式二]; then
- 当条件判断式二成立时，可以进行的命令工作内容
- else
- 当条件判断式一二军不成立时，可以进行的命令工作内容
- fi

6、<netstat>

- netstat -tuln <==获取目前主机有启动的服务
- Local Address(本地主机的 IP 与端口对应)那个字段，它代表的是本机所启动的网络服务
- 若 IP 为 127.0.0.1 则是仅针对本机开放
- 若 IP 为 0.0.0.0 或:::则代表对整个 Internet 开放

14. 4. 2. 利用 case. . . esac 判断

1、语法

- case $变量名称 in
- "第一个变量内容)
- ;;
- "第二个变量内容)
- ;;
- *) <==最后一个变量内容都会用*来代表所有其他值
- exit 1
- ;;
- esac

14. 4. 3. 利用 function 功能

1、语法

- function fname(){}
- 因为 shell script 的执行方式是由上而下，由左而右，因此在 shell script 中的 function 的设置一定要在程序的最前面

2、/etc/init.d function 的源文件???

3、function 也是拥有内置变量的，它的内置变量与 shell script 很类似

- 函数名称：$0

- 后接的变量为：$1,$2...
- 在函数体内的$1 $2...代表的是函数输入参数
- 在函数体外，shell script 内的$1,$2...代表的是脚本运行时附加的参数

14.5. 循环（loop）

14.5.1. while do done,until do done（不定循环）

1、一般来说不定循环最常见的形式：

- 当条件成立时，执行循环，直到条件不成立才停止

```
while [ condition ]
do
    程序段落
done
```

- 条件不成立时，执行循环，直到条件成立才停止

```
until [condition]
do
    程序段落
done
```

14.5.2. for...do...done（固定循环）

1、语法

```
for var in con1 con2 con2...
do
    程序段
done
```

14.5.3. for...do...done 的数值处理

1、语法

```
for((初始值;限制值;执行步长))
do
    程序段
done

for((i=1;i<=100;i=i+1))  #这里可以直接写 i，而不用写$i，i=i+1 可以改为 i++
do
    程序段
done
```

2、注意点：

- 双括号内的变量可以不用$
- 条件可以直接写，不用[]

14.6. shell script 的追踪与调试

1、<sh>或<bash>

- sh(bash) [-nvx] scripts.sh
- -n：不要执行 script，仅查询语法错误

- -v：在执行 script 前，先将 script 的内容输出到屏幕上
- -x：将使用到的 script 内容显示到屏幕上，很有用

Chapter 15. Linux 账号管理与 ACL 权限设置

15.1. Linux 的账号与用户组

15.1.1. 用户标识符：UID 与 GID

1、我们登陆 Linux 主机的时候，输入的是账号，但是 Linux 主机并不会直接认识账号名，它仅认识 ID

2、ID 与账号的对应关系就在/etc/passwd 当中

3、每个文件都有"所有者"与"所属用户组"的属性

15.1.2. 用户账号

1、输入密码后，系统处理后续步骤：

- 先寻找**/etc/passed** 里面是否有你输入的账号，如果没有则跳出，如果有的话则将该账号对应的 UID 与 GID(在**/etc/group** 中)读出来，另外，该账号的主文件夹与 shell 设置也一并读出
- 再来则是核对密码表，这是 Linux 会进入**/etc/shadow** 里面找出对应账号的 UID，然后核对一下你刚才输入的密码与里面的密码是否相符
- 如果一切都 OK 的话，就进入 shell 控管阶段了

2、/etc/passwd 文件结构

- 每一行都代表一个账号，有几行就有几个账号
- 里面有很多账号是系统正常运行所必须的，可以称它们为系统账号，例如 bin，daemon，adm，nobody 等等
- 每一行用":"分隔，共有七个字段
  - **账号名称**：用于对应 UID
  - **密码**：早期 UNXI 系统的密码就是放在这个字段上，但是因为这个文件的特性是所有的程序都能读取，这样一来很容易造成密码数据被窃取后来将这个字段的密码数据改放到/etc/shadow 中，在其位置放置一个"X"
  - **UID**：用户标识符
    - 0：系统管理员
      - 要让其他账号名称也具有 root 权限时，将其 UID 改为 0 即可
      - 一个系统可以有多个 root，但不建议这样做
    - 1-499：系统账号
      - 除了 0 之外，其他的 UID 权限与特性并没有不一样，默认 500 一下的数字让系统作为保留账号只是一个习惯
      - 系统上面启动的服务希望用较小的权限去运行，因此不希望使用 root 身份去执行这些任务，所以我们就得提供这些运行程序的所有者账号才行。
      - 这些系统账号通常是不可登陆的，所以才会有/sbin/nologin 这个特殊的 shell 存在 bash 的环境配置文件
      - 1-99：由 distributions 自行创建的系统账号
      - 100-499：若用户有系统账号需求时，可以使用的账号 UID
      - 500-65535(可登陆账号)：给一般用户的，目前已经支持到 2^32-1 个用户了(4294967295)
  - **GID**：这个与/etc/group 有关，其实/etc/group 和/etc/passwd 差不多，

只是它用来规定组名与 GID 的对应而已

- **用户信息说明列**：这个字段基本没什么用，只是用来解释账号的意义而已，如果提供 finger 的功能时，这个字段可以提供更多的信息
- **主文件夹**：/root /home/liuye
- **Shell**：当用户登录系统后就会取得一个 Shell 来与系统的内核通信以进行用户的操作任务。有一 shell 可以来替代让账号无法取得 shell 环境的登录操作，那就是/sbin/nologin

3、/etc/shadow 文件结构

- 基本上，shadow 同样以":"作为分隔符
- 共有九个字段
  - **(1)账号名称**：必须要与/etc/passwd 相同才行
  - **(2)密码**：经过编码的密码，这个文件的默认权限是-rw-------或者是-r--------，只有 root 才能读写
  - **(3)最近变更密码的日期**：以 1970 年 1 月 1 日开始的累加日期
    - 想要知道某个日期的累积天数:
    - echo $(($(date --date="2016/08/08" +%s)/86400+1))
  - **(4)密码不可被变更的天数(相比于 3)**：密码在最近一次被更改后需要过几天才可以被再次更改
  - **(5)密码需要重新改变的天数(相比于 3)**：你必须要在这个天数内重设置你的密码，**否则这个密码会变为过期特性**
  - **(6)密码需要更改期限前的警告天数(相比于 5)**：根据这个字段发出警告，提醒再过 n 天你的密码就要过期了
  - **(7)密码过期后的账号宽限时间(密码失效日,相比于 5)** ：如果密码过期了，那当你登陆系统时，系统会强制要求你必须要重设密码才能登陆继续使用，这就是密码过期特性
    - 那么这个字段什么用：在密码过期几天后，如果用户还是没有登陆更改密码，那么这个账号的密码将会"失效"，即该账号再也无法使用改密码登陆了，密码过期与密码失效不同
  - **(8)账号失效日期**：这个账号在此字段规定的日期之后，将无法再使用
  - **(9)保留**：最后一个字段是保留的，看以后有没有新功能加入

4、一般用户密码忘记了：利用 root 的身份使用 passwd 来处理即可

- passwd [用户名]

5、root 密码忘记了：重启进入用户维护模式，系统会主动给予 root 权限的 bash 接口，此时再以 passwd 修改密码即可

15. 1. 3. 有效与初始用户组

1、/etc/group 文件结构

- 这个文件每一行代表一个用户组，以":"分隔
- 一共四个字段
  - **用户组名称**
  - **用户组密码**：通常不需要设置，这个设置通常是给"用户组管理员"使用的，目前很少有这个机会设置用户组管理员，同样密码已经移动到/etc/gshadow 中，以'X'替代

- **GID**
- **此用户组支持的账号名称**：一个账号可以加入多个用户组，如果要让某用户加入某用户组，只需要在后面加上",username"即可，不要有空格

2、有效用户组与初始用户组

- 初始用户组：
  - 每个用户在他的/etc/passwd 里面第四列的 GID，这个 GID 就是初始用户组(initial group)
  - 当用户登录系统，立刻就拥有这个用户组的相关权限
  - **非初始用户组：通过在/etc/group 对应组的第四个字段后面添加用户名称从而获取的组**
- 有效用户组：
  - <groups>：查看有效与支持用户组
  - **第一个输出的用户组即为有效用户组，也就是说，以 touch 去新建一个文件，这个文件的用户组就是有效用户组**
  - <newgrp>：有效用户组的切换
    - newgrp [组名]
    - 想要切换的用户组必须是你已经有支持的用户组
    - newgrp 可以更改目前用户的有效用户组，而且以另外一个 shell 来提供这个功能，新的 shell 给予有效 GID 给这个用户
    - exit 后回到原来的环境中，还是原来的有效用户组
- 让一个账号加入不同的用户组
  - 通过系统管理员 root 利用 usermod 帮你加入
  - 通过用户组管理员以 gpasswd 帮你加入他所管理的用户组

3、/etc/gshadow 文件结构

- 如果/etc/gshadow 这个设置没有弄懂的话，newgrp 是无法操作的
- 这个文件以":"作为分隔符来分隔字段
- 几乎与/etc/group 一模一样
- 第二个字段：密码列
  - 如果密码列上是"!"，表示该用户组不具有用户组管理员
- 共四个字段：
  - **用户组名**
  - **密码列**：开头为！表示无合法密码，所以无用户组管理员
  - **用户组管理员账号**(相关信息在 gpasswd 中介绍)
  - **该用户组所属账号**(与/etc/group 内容相同)
- 这个 gshadow 最大的功能就是创建组管理员

15.2. 账号管理

15.2.1. 新增与删除用户：useradd，相关配置文件，passwd，usermod，userdel

1、<useradd>

- useradd [-u UID] [-g 初始用户组] [-G 次要用户组] [-mM]\
- [-c 说明栏] [-d 主文件夹绝对路径] [-s shell] 用户账号名

- -u：后接 UID
- -g：后接用户组名(**初始用户组，initial gropu**)
- -G：这个账号还可以加入的用户组
- -M：强制！不要创建用户主文件夹(系统账号默认值)
- -m：强制！要创建用户主文件夹(一般账号默认值)
- -c：这个就是/etc/passwd 第五列的说明内容，随便设置
- -d：指定某个目录称为主文件夹，而不要使用默认值，**务必使用绝对路径**
- -r：创建一个系统账号，这个账号的 UID 会有限制
- -s：后面接一个 shell，若没有指定则默认是/bin/bash
- -e：后面接一个日期，格式"YYYY-MM-DD"，此选项写入/etc/shadow 第八个字段，账号失效日期的设置选项
- -f：后面接/etc/shadow 的第七个字段，指定密码是否会失效，0 为立即失效，-1 为永远不失效
- 由于系统已经帮我们规定好非常多的默认值了，所以我们可以简单地使用 useradd [账号名]来创建用户即可
- CentOS 这些默认值主要会帮我们处理几个项目：
  - 在/etc/passwd 里面创建一行与账号相关的数据，包括创建 UID/GID/主文件夹等
  - 在/etc/shadow 里面将此账号的密码相关参数填入，但是尚未有密码
  - 在/etc/group 里面加入一个与账号名称一模一样的组名
  - 在/home 下面创建一个与账号同名的目录作为用户主文件夹，权限为 700
  - 由于在/etc/shadow 内仅会有密码参数而不会有加密过的密码数据，因此我们在创建用户账号时，**还需要使用 passwd [账号]来给予密码才算是完成了用户创建的流程**

2、系统账号默认不会创建主文件夹

3、用 useradd 创建账号会更改的文件：
- /etc/passwd,/etc/shadow
- /etc/group,/etc/gshadow
- 用户组的文件夹：/home/账号名称

4、useradd 参考文件：
- 用于设置默认参数
- useradd -D
- 这个数据是由/etc/default/useradd 调用出来的
- 参数介绍
  - GROUP=100：新建账号的初始用户组
    - 系统上面 GID 为 100 即是 users 这个用户组
    - 在 CentOS 上面默认的用户组是与账号名相同的用户组
      - **私有用户组机制**：系统会创建一个与账号一样的用户组给用户作为初始用户组，这种用户组的设置机制会比较有保密性，这是因为用户都有自己的用户组，而且主文件夹的权限将会设置为 700(仅有自己可以进入)，使用这种机制将不会参考 GROUP=100 这

个设置值，代表性的 distributions 有 RHEL，Fedora，CentOS
        - **公用用户组机制**：就是以 GROUP 这个设置值作为新建账号的初始用户组，因此每个账号都属于 users 这个用户组，且默认主文件夹的权限会是"drwxr-xr-x"，由于每个账号都属于 users，因此大家可以互相分享主文件夹内的数据
        - 对于 CentOS，这个设置值不会生效
    - HOME=/home：用户主文件夹的基准目录
    - INACTIVE=-1：密码过期后是否会失效的设置值
        - -1：永不失效
        - 0：立即失效
        - n(n>0)：n 天后失效
    - EXPIRE=: ：账号失效日期，通常不设置此项
    - SHELL=/bin/bash：默认使用这里的 shell 程序文件名
    - SKEL=/etc/skel：用户主文件夹参考基准目录
        - 用户主文件夹内的各项数据都是由/etc/skel 赋值过去的
        - 如果想要新增用户时，该用户的环境变量~/.bashrc 就配置好的话，可以到/etc/skel/.bashrc 去编辑
        - 如果想要新增文件夹，可以新建/etc/sekl/www
    - CREATE_MAIL_SPOOL=yes：创建用户的 mailbox
        - 可以使用 ll /var/spool/mail/liuye

5、/etc/login.defs 文件结构 ：UID/GID/密码等参数

- mailbox 所在目录：/var/spool/mail/用户名
- shadow 密码第 4、5、6 字段内容
    - 通过 PASS_MAX_DAYS 等值来设置
- UID/GID 指定数值
    - UID_MIN：一般账号 UID 最小值
    - UID_MAX：一般账号 UID 最大值
    - SYS_UID_MIN：系统账号 UID 最小值
    - SYS_UID_MAX：系统账号 UID 最大值
- 用户主文件夹设置
    - CREATE_HOME yes
- 用户删除与密码设置
    - USERGROUPS_ENAB yes
    - 如果使用 userdel 去删除一个账号时，且该账号所属的初始用户组已经没有人隶属于该用户组了，那么就删掉该用户组

6、useradd 这个程序在创建 Linux 上的账号至少会参考

- /etc/default/useradd
- /etc/login.defs
- /etc/skel/*

7、<passwd>

- passwd <==修改当前账号的密码, 所有人均可以用来改自己的密码

- passwd [-l] [-u] [--stdin] [-S] [-n 日数] [-x 日数] [-w 日数] [-i 日数] 账号 <==root 的功能
- **--stdin：表示使用 standard input 作为密码，配合'|'可以通过来自前一个管道的数据，作为密码输入，对 shell script 有帮助**
- -l：是 Lock 的意思，会将/etc/shadow 第二列最前面加上！，使密码失效
- -u：与-l 相对，unLock 的意思
- -S：列出密码相关参数，即 shadow 文件内大部分信息
- -n：后接天数，shadow 第 4 字段，表示上一次修改密码之后最少几天后才能修改密码
- -x：后接天数，shadow 第 5 字段，表示上一次修改密码之后，最多几天之内必须修改密码
- -w：后接天数，shadow 第 6 字段，密码过期前的警告天数
- -i：后接天数(1970.1.1 到失效的日期的天数)，shadow 的第 7 字段，密码失效日期
- 可以使用 echo "密码字符串" | passwd --stdin 账号名 <==来改密码，但是可以在/root/.bash_history 中找到，不太安全，通常用于 shell script 大量新建账号中使用

8、<chage>

- chage [-ldEImMW] [账号名]
- -l：列出该账号的详细密码参数
- -d：后面日期，修改 shadow 第 3 字段，最近一次修改密码的日期，格式 YYYY-MM-DD
- -E：后接日期，修改 shadow 第 8 字段，账号失效日，格式 YYYY-MM-DD
- -I：后接天数，修改 shadow 第 7 字段，密码过期后几天密码失效
- -m：后接天数，修改 shadow 第 4 字段，密码改动后最少几天才能再次修改
- -M：后接天数，修改 shadow 第 5 字段，密码改动后最多几天必须进行再次修改
- -W：后接天数，修改 shadow 第 6 字段，密码过期前几天警告日期
- 小技巧：
  - useradd agetest
  - echo "agetest" | passwd --stdin agetest
  - **chage -d 0 agetest ==>"You are required to change your password immediately (root enforced)**
  - 这样会强制要求该用户"agetest"在登陆时修改密码

9、<usermod>

- 我们可以直接到/etc/passwd 或/etc/shadow 去修改相应字段的数据
- 还可以用 usermod 来进行修改
- usermod [-cdegGlsuLU] [用户名]
- -c：后面接账号的说明，即/etc/passwd 第 5 列的说明，可以加入一些账号的说明

- -d：后面接账号的主文件夹，即修改/etc/passwd 的第 6 列
- -e：后接日期，格式 YYYY-MM-DD，/etc/shadow 内的第 8 个字段
- -f：后接天数，/etc/shadow 的第 7 个字段
- -g：后接初始用户组，修改/etc/passwd 第 4 个字段，即 GID 字段
- -G：后接次要用户组，修改这个用户能够支持的用户组，修改/etc/group
- -a：与-G 合用可增加次要用户组的支持而非设置
- -l：后接账号名称，即修改账号名称，/etc/passwd 第一列
- -s：后接 Shell 的实际文件，例如/bin/bash
- -u：后接 UID 数字，修改/etc/passwd 第 3 列数据
- -L：暂时将用户密码冻结，让它无法登陆，其实仅修改/etc/shadow 的密码，添加！
- -U：将/etc/passwd 密码列的！去掉
- **usermod 与 useradd 非常类似，因为 usermod 是用来微调 useradd 增加的用户参数**

10、<userdel>

- 删除用户相关数据，包括
  - /etc/passwd,/etc/shadow
  - /etc/group,/etc/gshadow
  - /home/username,/var/spool/mail/username
- userdel [-r] username
- -r：连同主文件夹也一起删除
- 通常删除账号可以手动将/etc/passwd 与/etc/shadow 里面的账号取消即可，一般而言，如果账号只是暂时不启用的话，那么将/etc/shadow 里面的账号失效日期(第 8 个字段)设置为 0 即可让账号无法使用，但是所有跟账号相关的数据都会留下来

15. 2. 2.　　用户功能

1、<finger>

- 用于查阅用户相关的信息，大部分都是/etc/passwd 这个文件里面的信息
- finger -s [用户名]
- -s：仅列出用户账号，全名，终端机代号与登录时间
- -m：列出与后面接的账号相同者，而不是利用部分对比(包括全名部分)
- 打印的参数解释：
  - Login：用户账号，/etc/passwd 第 1 个字段
  - Name：账号描述，/etc/passwd 第 5 个字段
  - Directory：主文件夹
  - Shell：使用的 shell
  - 登录主机的情况
  - 调查/var/spool/mail 当中的信箱数据
  - 调查~username/.plan 文件，并将该文件取出来说明

2、<chfn>

- change finger 的意思

- chfn [-foph] [用户名]
- chfn [用户名] <== 会要求你输入下列四个参数
- chfn <== 对当前用户进行下列 4 个参数修改
- -f：后接完整账号描述
- -o：办公室号码
- -p：办公室电话号码
- -h：家里电话号码

3、<chsh>
- change shell
- chsh [-ls]
- -l：列出目前系统上可用的 shell，其实就是/etc/shells 的内容
- -s：设置修改自己的 shell

4、**chfn 与 chsh 都能让一般用户修改/etc/passwd 这个系统文件，所以这两个文件(chfn、chsh)的权限是 SUID 的**

5、<id>
- id [用户名]
- 列出 uid，gid 与 groups 的值

15. 2. 3. 新增与删除用户组

1、用户组的内容都与这两个文件有关：/etc/group 与/etc/gshadow，文件内容参详见有效与初始用户组

2、用户组的概念很简单，都是这两个文件的新增、修改与删除，另外还有有效用户组的概念，涉及到命令 newgrp 与 gpasswd

3、<groupadd>
- groupadd [-g gid] [-r] [用户名]
- -g：后面接某个特定 GID，用来直接给予某个 GID
- -r：新建系统用户组
- 不用参数-g 的话 GID 就是非系统用户组的最大 GID+1

4、<groupmod>
- 与 usermod 类似，这个命令仅是在进行 group 相关参数修改
- -g：修改既有的 GID 数字，**不要随意修改这个！**
- -n：修改既有的组名

5、<groupdel>
- 删除用户组
- groupdel [用户组
-]
- 必须要确认/etc/passwd 内的账号没有任何人使用该用户组作为初始用户组才行

6、<gpasswd>
- 系统管理员可以用 gpasswd 来为某个用户组增加一个管理员，用户组管理员可以管理哪些账号可以加入\移出该用户组
- **以下为系统管理员可做的操作**

- gpasswd groupname
- gpasswd [-A user1,...] [-M user3,...] groupname
- gpasswd [-rR] groupname
- gpasswd 后面不接任何参数，只跟用户组名，表示基于 groupname 一个密码(/etc/gshadow)
- -A：将 groupname 的主控权交由后面的用户管理(该用户组的管理员)
- M：将某些账号加入这个用户组
- -r：将 groupname 的密码删除
- -R：让 groupname 的密码栏失效
- 以下为用户组管理员可做的操作
- group [-ad] [用户名] [用户组名]
- -a：将某位用户加入到 groupname 这个用户组当中
- -d：将某位用户删除出 groupname 这个用户组当中

15.2.4. 账号管理实例

1、任务描述：pro1、pro2、pro3 是同一个项目计划的开发人员，想要让着三个用户在同一个目录下面工作，但这三个用户还是各自拥有自己的主文件夹与基本的私有用户组，这个项目在/srv/projecta 目录下开发

2、命令

- groupadd projecta
- useradd -G projecta -c "projecta user" pro1
- useradd -G projecta -c "projecta user" pro2
- useradd -G projecta -c "projecta user" pro3
- echo "password" | passwd --stdin pro1
- echo "password" | passwd --stdin pro2
- echo "password" | passwd --stdin pro3
- mkdir /srv/projecta
- chgrp projecta /srv/projecta
- chmod 2770 /srv/projecta

3、SUID、SGID、SBIT 属性，这里/srv/projecta 目录设置为 SGID，为了让三个用户能够在工程目录下修改对方的文件，并且在该目录/srv/projecta 中的用户的有效用户组为这个目录的用户组

4、问题：如果有一个项目经理，需要查看项目，项目经理不支持这个组，如果将/srv/projecta 目录权限改为 2775，那么其他任何人都可以进来查看了

15.3. 主机的具体权限规划：ACL 的使用

15.3.1. 什么是 ACL

1、ACL 是 Access Control List 的缩写

2、主要目的是提供传统的 owner、group、others、read、write、execute 权限之外的具体权限设置。

3、ACL 可以针对单一用户、单一文件或目录来进行 r、w、x 的权限设置，对于需要特殊权限的使用状况非常有帮助

4、ACL 可以针对以下方面来控制权限

- 用户(user)：针对用户来设置权限
- 用户组(group)：针对用户组来设置其权限
- 默认属性(mask)：还可以在该目录下载新建文件/目录时设置新数据的默认权限

15.3.2. 如何启动 ACL

1、由于 ACL 是传统 UNIX-like 操作系统权限的额外支持项目，因此要使用 ACL 必须要有文件系统的支持

2、在 Centos7 下：

- df <==查看目前挂载的设备
  - /dev/mapper/centos-root /
  - /dev/mapper/centos-root /home
- dumpe2fs -h /dev/mapper/centos-root
  - ...
  - Default mount options: user_xattr acl
  - ...

15.3.3. ACL 的设置技巧：getfacl, setfacl

1、<setfacl>

- 设置某个目录/文件的 ACL 规定
- setfacl [-bkRd] [{-m|-x} acl 参数] [目标文件名]
- **-m：设置后续的 acl 参数给文件使用，不可与-x 合用**
- **-x：删除后续的 acl 设置，不可与-m 合用**
- -b：删除所有 acl 设置参数
- -k：删除默认的 acl 参数
- -R：递归设置 ACL
- -d：设置默认 acl 参数，只对目录有效，在该目录新建的数据会引用此值
- 针对特定用户设置方式：
  - u:[用户账号列表]:[rwx]
  - u::[rwx] <==无用户列表，**代表设置该文件所有者的权限**
  - 例：
    - setfacl -m u:liuye:rx acltest1
    - setfacl -m u:liuye:- acltest1 <==-代表无任何权限
    - setfacl -m u::rwx acltest1
- 针对特定用户组的设置方式：
  - g:[用户组列表]:[rwx]
  - 例：
    - setfacl -m g:mygroup1:rx acl_test1
- 针对有效权限 mask 的设置方式
  - m:[rwx]
  - 例：
    - setfacl -m m:r acl_test1
- 如果一个文件设置了 ACL 参数后，它的权限部分就会多出一个+号

2、<getfacl>

- getfacl [文件名]
- 带有#号的数据代表这个文件的默认属性

3、mask 是用户或用户组所设置的权限必须要存在于 mask 的权限设置范围内才会生效，此即有效权限

- 比如 mask 是 r--
- 某用户的权限是 r-x
- 那么该用户有效的权限就是 r-x 与 r--的合集，即 r--
- 通常将 mask 设置为 rwx，然后再根据不同用户/用户组设置权限

15. 4. 用户身份切换

1、使用一般账号：系统平日操作的好习惯

- 为了安全的缘故，尽量以一般用户身份来操作能够 Linux 的日常作业
- 需要设置系统环境时，才变换身份为 root 进行系统管理

2、用较低权限启动系统服务

- 相对于系统安全，有时候，我们必须要以某些系统账号进行程序的执行

3、软件本身的限制

4、一般用户转变成 root 的方式

- su -
  - 这个命令需要 root 的密码
- sudo [命令]
  - 执行 root 的命令串，由于 sudo 需要实现设置妥当，且 sudo 需要输入用户自己的密码，因此多人共管同一台主机时，sudo 要比 su 来的好

15. 4. 1. <su>

1、su 是最简单的身份切换命令，它可以进行任何身份的切换

2、语法：

- su [-lm] [-c 命令] [用户名]
- **-：单纯使用-,如"su -"表示使用 login-shell 的变量文件读取方式来登陆系统**
- -l：与-类似，但后面需要加欲切换的用户账号(可以不加，就表示 root，与 su - 完全一样)，也是 login-shell 的方式
  - su -l liuye
- -m：与-p 是一样的，表示使用目前的环境设置，而不读取新用户的配置文件
- -c：仅进行一次命令，所以-c 后面可以加上命令，命令必须用""或"括起来
  - su - -c "head -n 3 /etc/shadow"
- **单纯使用 su 切换成 root 身份，读取的变量设置方式为 non-login shell 的方式，这种方式下很多原本的变量不会被改变，切换身份不建议用这种**

3、总结：

- 要完整地切换到新用户的环境，必须使用 su - username 或 su -l username 才会以 login-shell 的方式读取变量设置

- 如果想要执行一次 root 命令，使用 su - -c "命令串"
- 使用 root 切换成为任何用户时，并不需要输入用户的密码

15. 4. 2. <sudo>

1、相对于 su 需要了解切换用户的密码(经常需要 root 的密码)，sudo 的执行仅需要自己的密码即可

2、由于 sudo 可以让你以其他用户的身份执行命令(通常是使用 root 的身份来执行命令)，因此并非所有人都能执行 sudo，而是仅有**/etc/sudoers** 内的用户才能够执行 sudo 命令

3、语法

- sudo [-b] [-u 用户名] [后接要执行的命令]
- -b：后续的命令让系统自己执行，而
- 与目前的 shell 产生影响
- -u：后面可以接欲切换的用户，若无此项则代表切换身份为 root

4、sudo 默认仅有 root 能使用，因为 sudo 的执行流程如下：

- 当用户执行 sudo 时，系统于/etc/sudoers 文件中查找该用户是否有执行 sudo 的权限
- 当用户具有可执行 sudo 的权限后，便让用户输入用户自己的密码来确认
- 若密码输入成功，便开始进行 sudo 后续接的命令(但 root 执行 sudo 时不需要输入密码)
- 若欲切换的身份与执行者身份相同，那也不需要输入密码

5、visudo 与/etc/sudoers

- 若想要使用 sudo 执行属于 root 的权限命令，则 root 需要先使用 visudo 去修改/etc/sudoers，让该账号能够使用全部或部分的 root 命令功能
- 因为/etc/sudoers 是有语法的，如果设置错误那会造成无法使用 sudo 命令的不良后果，因此才使用 visudo 去修改，并在结束离开修改界面时，系统会去检查/etc/sudoers 的语法
- 单一用户可进行 root 所有命令与 sudoers 文件语法
  - visudo
  - root ALL=(ALL) ALL <==找到这句
  - liuye ALL=(ALL) ALL <==新增这句
  - **<1> <2> <3> <4>**
- 上述四个参数的意义：
  - **用户账号<1>**：系统的哪个账号可以使用 sudo 这个命令，默认为 root 这个账号
  - **登陆者的来源主机名<2>**：这个账号由哪台主机连接到本 Linux 主机，意思是这个账号可能是由哪一台网络主机连接过来的，这个设置可以指定客户端计算机(信任用户的意思)，默认 root 可以来自任何一台网络主机
  - **可切换的身份<3>**：这个账号可以切换成什么身份来执行后续的命令，默认 root 可以切换成任何人

- **可执行的命令<4>**：这个命令请务必使用绝对命令编写，默认 root 可以切换任何身份且进行任何命令
- ALL 是特殊的关键字，代表任何身份、主机或命令的意思

6、利用用户组以及免密码的功能处理 visudo

- visudo
- "%wheel ALL=(ALL) ALL" <==找到该行，大约在 84 行左右
- usermod -a -G wheel liuye
- 上面的设置会造成任何加入 wheel 这个用户组的用户就能够使用 sudo 切换身份来操作任何命令

7、设置：让 wheel 组内的用户不需要密码即可使用 sudo

- visudo
- "%wheel ALL=(ALL) NOPASSWD:ALL" <==找到该行，大约在 87 行
- **在最左边加上%代表后面接的是一个用户组的意思**

8、有限制的命令操作

- 下方部分权限，例如让 liuye 仅能帮助 root 修改其他用户的密码
- liuye ALL=(root) /usr/bin/passwd
- 更进一步，不允许修改 root 的密码
- liuye ALL=(root) !/usr/bin/passwd, /usr/bin/passwd [A-Za-z]*, \
- !/usr/bin/passwd root

9、通过别名设置 visudo，如果有多个用户需要加入 sudo 管理员行列，可以通过别名来设置

- visudo
- **User_Alias** ADMPW=pro1,pro2,pro3,myuser1,myuser2
- **Cmnd_Alias** ADMPWCOM=!/usr/bin/passwd, /usr/bin/passwd [A-Za-z]*, \
- !/usr/bin/passwd root
- ADMPW ALL=(root) ADMPWCOM
- **User_Alias** 命令新建一个账号别名，必须要用大写字符来处理
- **Cmnd_Alias** 命令新建命令别名，必须要用大写字符来处理
- **Host_Alias** 命令新建主机别名，必须要用大写字符来处理

10、sudo 的时间间隔问题

- 如果使用同一个账号在短时间内重复操作 sudo 来运行命令，第二次执行 sudo 时并不需要输入自己的密码
- 两次执行在五分钟内，那么再次执行不需要密码
- 使用一般账号，理论上不会用到/sbin,/usr/sbin 等目录内的命令，所以 $PATH 变量不会含有这些目录，因此使用绝对路径比较妥当，间接说明，**使用 sudo 时并不是以 login-shell 的方式，因此 PATH 等环境变量仍旧是当前用户的，而不是 root 的**

15.5. 用户的特殊 shell 与 PAM 模块

1、问题引入：如果想要创建一个仅能使用 mail server 相关邮件服务的账号，而该账号并不能登陆 Linux 主机

15.5.1. 特殊的 shell,/sbin/nologin

1、系统账号的 shell 就是使用/sbin/nologin，重点在于系统账号是不需要登陆的，所以就给予它们"无法登陆"的合法 shell

2、我们所谓的"无法登陆"指的仅是这个用户无法使用 bash 或其他 shell 来登陆系统而已，并不是说这个账号无法使用其他的系统资源

3、如果想要让某个具有/sbin.nologin 的用户知道，它们不能登陆主机，我们可以新建/etc/nologin.txt 这个文件，并且在这个文件内说明不能登陆的原因

- 当使用 su - [用户名]时会出现/etc/nologin.txt 的内容，而不是默认的内容"This account is currently not available."

15.5.2. PAM 模块简介

1、我们经常用不同的机制来判断账号密码，所以弄得一台主机上面拥有多个认证系统，也造成账号密码可能不同步的验证问题，为了解决这个问题，引入了 PAM(Pluggable Authentication Modules,嵌入式模块)的机制

2、PAM 可以说是一套应用程序编程接口(API,Application Programming Interface)，它提供了一连串的验证机制，只要用户将验证阶段的需求告知 PAM 后，PAM 就能够回报用户验证的结果(成功或失败)

3、由于 PAM 仅是一套验证的机制，又可以提供给其他程序调用，因此不论你使用什么程序，都可以使用 PAM 来进行验证

- 只要任何程序有需求时，可以向 PAM 发出验证要求的通知，PAM 经过一连串的验证后，将验证的结果回报给该程序，然后该程序就能利用验证的结果来进行可登陆或显示其他无法使用的信息

15.5.3. PAM 模块设置语法

1、程序调用 PAM 的流程，以执行 passwd 为例

- 用户开始执行/usr/bin/passwd 这支程序，并输入密码
- passwd 调用 PAM 模块进行验证
- PAM 模块会到/etc/pam.d 中寻找与程序(passwd)同名的配置文件
- 根据/etc/pam.d/passwd 内的设置，引用相关 PAM 模块逐步进行验证分析
- 将验证结果(成功失败及其他信息)回传给 passwd 这个程序
- passwd 这只程序会根据 PAM 回传的结果决定下一个操作(重新输入新密码或者通过验证！)

2、**/etc/pam.d**/[command]的文件结构

- 除了第一行声明 PAM 的版本之外，其他任何#都是批注
- 每一行都是一个独立的验证流程
- 每一行可以分为三个字段
  - 验证类别
  - 控制标准
  - PAM 的模块与该模块的参数

3、三个字段详细介绍

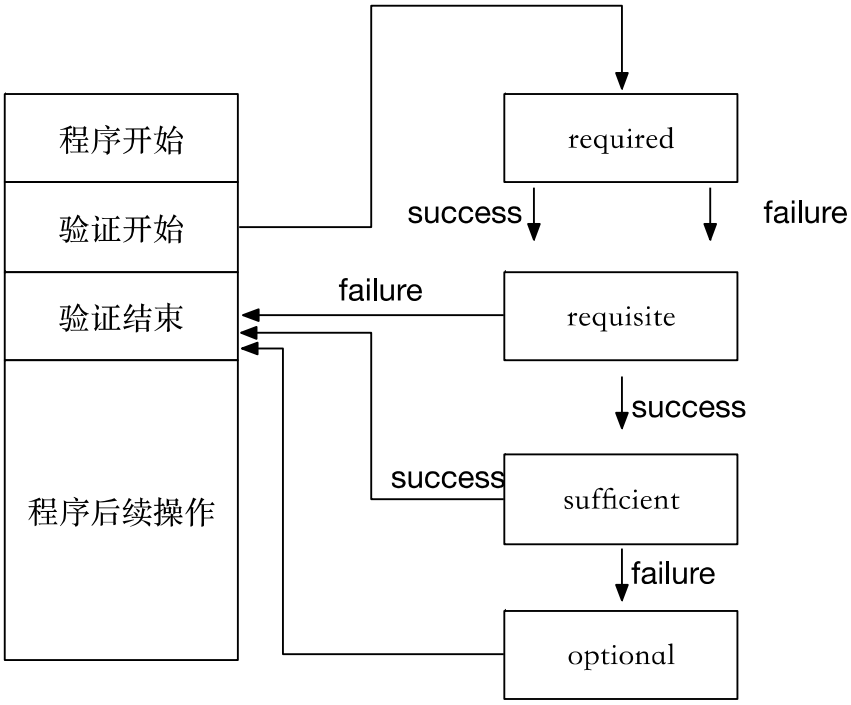
- 第一个字段：验证类型(Type)
  - 验证类型主要分为四种
  - auto：authentication(认证)的缩写

- 这种类型主要用来检验用户的身份验证，这种类型通常是需要密码来检验的
- 所以后续接的模块是用来检验用户身份的

- account：(账号)
  - 账号则大部分在进行 authorization(授权)，这种类型则主要在检验用户是否具有正确的权限
  - 例如，当使用一个过期密码来登陆时，就无法正确登陆了
- session：会议期间的意思???
  - session 管理的就是用户在这次登陆(或使用这个命令)期间 PAM 所给予的环境设置
  - 这个类型通常用于记录用户登陆与注销的信息
- password：密码
  - 主要用于提供验证的修订工作
  - 就是修改密码
- 这四个验证类型通常是有顺序的

➢ 第二个字段：验证的控制标志(control flag)

- 这个字段在管控该验证的放行方式
- 主要分为四种控制方式
- required
  - 此验证成功则带有 success 标志，失败则带有 failure 标志
  - 由于后续的验证流程可以继续进行，因此相当有利于数据的登陆日志，这也是 PAM 最常使用 required 的原因
- requisite
  - 若验证失败，则立刻回报原程序 failure 标志，并终止后续的验证流程
  - 若验证成功则带有 success 的标志并继续后续的验证流程
- sufficient
  - 验证成功则立刻传回 success 给原程序，并终止后续的验证流程
  - 与 requisite 正好相反
- optional
  - 这个模块空间目的大多是在显示信息而已



15. 5. 4. 常用模块简介

1、文件介绍

- /etc/pam.d/*：每个程序个别的 PAM 配置文件
- /lib/security/*：PAM 模块文件的实际放置目录
- /etc/security/*：其他 PAM 环境的配置文件
- /usr/share/doc/pam-*/：详细的 PAM 说明文件

2、常用模块介绍

- pam_securetty.so
  - 这个模块限制系统管理员(root)只能从安全(secure)终端机登陆
  - 什么是终端机?例如 tty1~tty6
  - 安全的终端机设置，写在/etc/securetty 这个文件中，该文件记录所有安全终端机的名称，如 tty1~tty6 等等
- pam_nologin.so
  - 这个模块可以限制一般用户是否能够登陆主机之用，当/etc/nologin 这个文件存在时，所有一般用户均无法再登陆系统了
  - 当这个文件/etc/nologin 存在时，一般用户登陆时会将该文件的内容显示出来，所以正常情况下，该文件是不能存在于系统的
  - 这个模块对于 root 以及已经登陆系统中的一般账号没有影响
- pam_selinux.so
  - SELinux 是个针对程序来进行详细管理权限的功能
  - 由于 SELinux 会影响到用户执行程序的权限，因此我们利用 PAM 模块将 SELinux 暂时关闭，等到验证通过后再予以启动
- pam_console.so
  - 这个模块可以帮助处理一些文件权限的问题，让用户可以通过特殊的终端接口(console)顺利登陆系统

- pam_loginuid.so
  - 为了验证用户的 UID 真的是我们需要的数值，可以用这个模块进行规范
- pam_env.so
  - 用来设置环境变量的模块
- pam_UNIX.so
  - 很复杂且重要的模块
  - 可以用于验证阶段的认证功能
  - 可以用于授权阶段的账号许可证管理
  - 可以用于会以阶段的日志文件记录等
  - 可以用于密码更新阶段的检验
- pam_cracklib.so
  - 用来检验密码的强度，包括密码是否在字典中，密码输入几次失败就断掉此次连接等功能
- pam_limits.so
  - ulimit(与文件系统及程序的限制关系：ulimit)就是这个模块提供的能力

15.5.5. 其他相关文件

1、本节介绍与 PAM 相关的文件

2、/etc/security/limits.conf

- ulimit 功能除了可以通过修改用户的~/.bashrc 配置文件外，还可以通过 PAM 来管理，就是/etc/security/limits.conf 这个文件
- 每一行分为四个字段：
- 第一个字段：账号，或者用户组，若为用户组，则前面需要加上@(与 visudo 不同，这里是用户组前加%)
- 第二个字段：限制的依据，严格(hard)或警告(soft)
- 第三个字段：相关限制
- 第四个字段：限制的值
- liuye soft fsize 9000
- liuye hard fsize 10000
- @group hard maxlogins 1

3、/var/log/secure,/var/log/messages

- 如果发生任何无法登陆或者是产生一些你无法预期的错误，由于 PAM 模块会将数据记载在**/var/log/secure** 当中，所以发生了问题，可以去这个文件查阅

15.6. Linux 主机上的用户信息传递

15.6.1. 查询用户：w，who，last，lastlog

1、查询一个用户的相关数据：id、finger

2、<last>

- 查询用户登陆相关信息
- 现在 last 可以列出从新建之后到目前为止所有登陆者的信息

3、<w>/<who>

- 查询目前谁在系统上

4、<lastlog>

- 查询每个账号最近的登陆时间，lastlog 会读取/var/log/lastlog 文件

15.6.2. 用户对谈：write、mesg、wall

1、<write>

- write [用户名] [用户所在终端接口，可用 who 查看]
- 输入上述命令后进入键盘输入模式，[Ctrl]+D 结束输入

2、<mesg>

- mesg n <==设置为不接受任何消息，但是无法阻止接受来自 root 的消息
- mesg y <==设置为接受消息
- mesg <==查看设置状态

3、<wall>

- wall '消息'
- 广播的形式发送给所有在线的用户

15.6.3. 用户邮件信箱：mail

1、使用 write、wall 只针对在线的用户

2、一般来说 mailbox 都会放在/var/spool/mail 里面

3、用户的邮箱路径：/var/spool/mail/liuye

4、<mail>

- mail username@localhost -s "邮件标题"
- 输入上面的命令后，接下来键入邮件正文
- 结束时最后一行输入小数点'.'即可
- 如果是寄给本机的用户，@localhost 也不用写
- 也可以先将邮件写在文件中，然后利用命令
  - mail username@localhost -s "邮件标题" < [文件名]

5、**解决无法用 mail 发送邮件的问题**

- yum install sendmail
- systemctl start sendmail.service
- systemctl status sendmail.service <==查看启动状态 OK

6、收件相关命令

- mail <==进入 mail 的环境中
- 输入"数字"即可读取数字对应的信件
- h：列出信件标头；如果要查阅 40 封信件标题，输入 h 40
- d：删除后接的信件号码
  - d 1
  - d 10-20
- s：后接信件号码，将信件保存成文件
  - s 5 ~/mail.file
- x：或者输入 exit，表示不作任何操作离开 mail(之前的操作作废)
- q：生效之前的操作，包括 e、d 等

15.7. 手动新增用户

15.7.1. 一些检查工具

1、<pwck>

- 该命令检查/etc/passwd 这个账号配置文件内的信息，与实际的主文件夹是否存在等信息，还可以比较/etc/passwd，/etc/shadow 的信息是否一致

2、<pwconv>

- 该命令将/etc/passwd 内的账号密码移动到/etc/shadow 当中
- 比较/etc/passwd 以及/etc/shadow，若/etc/passwd 内的账号并没有对应的/etc/shadow 密码时，则 pwconv 会去**/etc/login.defs** 取用相关的密码数据
- 若/etc/passwd 内存在加密后的密码数据，则 pwconv 会将该密码列移动到/etc/shadow 内，并将原本/etc/passwd 内相对应的密码变为 x
- 一般来说，如果正常使用 useradd 增加用户时，使用 pwconv 并不会有任何的操作，因为/etc/passwd 与/etc/shadow 并不会有上述两点问题
- 同理，group 的操作就是<gpwconv>

3、<pwunconv>

- 该命令将/etc/shadow 内的密码列数据写回/etc/passwd 中，并且删除/etc/shadow
- **由于会删除/etc/shadow，这个命令别去用**

4、<chpasswd>

- 该命令可以读入未加密前的密码，并且经过加密后，将加密后的密码写入/etc/shadow 中
- 这个命令常用在批量新建账号时
- 它可以由 STDIN 读入数据，每条数据的格式是"username:password"
- echo "liuye:12345678" | chpasswd -m
- -m：使用 MD5 加密方法，否则默认是 DES 加密方法
- 鉴于已经提供了，passwd --stdin 这个命令，因此 chpasswd 用处不大

15.7.2. 特殊账号(纯数字账号)的手工创建

1、必须清楚，账号与用户组

与/etc/group、/etc/shadow、/etc/passwd、/etc/gshadow 有关

2、步骤：

- 先新建所需要的用户组(vim /etc/group)
- 将/etc/group 与/etc/gshadow 同步(grpconv)
- 新建账号的各个属性(vim /etc/passwd)
- 将/etc/passwd 与/etc/shadow 同步(pwconv)
- 新建该账号的密码 (passwd accountname)
- 新建用户主文件夹 (cp -a /etc/sekl /home/accountname)
- 更改用户文件夹的属性(shown -R accountname.group/home/accountname)

15.7.3. 批量新建账号模板(适用于 passwd --stdin 参数)

15.7.4. 批量新建账号的范例

Chapter 16. 磁盘配额(Quota)与高级文件系统管理

16.1. 磁盘配额(Quota)的应用与实践

16.1.1. 什么是Quota

1、在 Linux 系统中，由于是多用于，多任务的环境，所以会有多用户共同使用一个硬盘空间的情况发生，如果其中有少数几个用户大量占掉硬盘空间的话，肯定会影响其他用户

2、Quota 的一般用途

- 针对 WWW server，例如每个人的网页空间的容量限制
- 针对 mail server，例如每个人的邮件空间限制
- 针对 file server，例如每个人最大的可用网络硬盘空间
- 限制某一用户组所能使用的最大磁盘配额
- 限制某一用户的最大磁盘配额
- 以 Link 的方式来使邮件可以作为限制的配额

3、Quota 的使用限制

- 仅能针对整个文件系统
- 内核必须支持 quota，默认为 aquota.user，aquota.group
- Quota 的日志文件
- 只对一般身份用户有效，不能对 root 设置 Quota，因为整个系统都是 root 的

4、Quota 的规范设置选项

- **容量限制或文件数量限制(block 于 inode)**
  - 限制 inode 用量：管理用户可以新建的"文件数量"
  - 限制 block 用量：管理用户磁盘容量的限制，**较常见的为这种方式**
- **soft/hard**
  - 无论 inode/block，限制值都有两个，soft 与 hard
  - 通常 hard 比 soft 要高，例如限制 block，hard 为 500M，soft 为 400M
  - hard：用户的用量绝对不会超过这个限制值，若超过这个值，则系统会锁住该用户的磁盘使用权
  - soft：表示用户在低于 soft 时，可以正常使用磁盘，但若超过 soft 且低于 had 的限值，每次用户登录系统时，系统会主动发出磁盘即将爆满的警告信息，且会给予一个宽限时间，若用户在宽限时间倒数期间内将容量再次低于 soft 限制之下，宽限时间会停止
- **会倒计时的宽限时间**
  - 宽限时间只有在用户的磁盘用量介于 soft 与 hard 之间时，才会出现且会倒数的一个时间
  - 默认宽限时间为 7 天
  - 如果宽限之内没有进行磁盘的管理，那么 soft 限制值会立即代替 hard 来作为 quota 的限制

16.1.2. 一个Quota范例

1、要求

- 账号：myquota1，myquota2，myquota3，myquota4，myquota5
- 账号的磁盘容量限制值：hard=300M，soft=250M

- 用户组的限额：myquotagrp 这个用户组最多仅能使用 1G 容量
- 宽限时间限制：14 天

16.1.3. 实践 Quota 流程 1：文件系统支持

1、步骤

- 首先查看/home 是否是独立的文件系统，因为 Quota 必须要内核与文件系统的支持
- df -h /home
- 如果想要在这次开机中实验 Quota，可以采用如下方式来手动**加入 Quota 的支持，非常重要，否则后续命令都无法用了**
  - **mount -o remount,userquota,grpquota /home**
- 如果想要自动挂载，就写入/etc/fstab 中
  - ... defaults,**usrquota,grpquota**...
  - 因为 defaults 不包含 usrquota，grpquota 这两个参数
- 当你重新挂载时，系统会同步更新/etc/mtab 这个文件，所以你必须确定/etc/mtab 已经加入 usrquota、grpquota 的支持到你所想要设置的文件系统中???

16.1.4. 实践 Quota 流程 2：新建 Quota 配置文件

1、Quota 通过分析整个文件系统中每个用户(用户组)拥有的文件总数与总容量，再将这些数据记录在该文件系统的最顶层目录，然后在配置文件中再使用每个账号(用户组)的限制值去规定磁盘的使用量

2、<quotacheck>

- 扫描文件系统并新建 Quota 的配置文件
- quotacheck [-avugFM] [/mount_point(挂载点)]
- **-a：扫描所有在/etc/mtab 内，含有 quota 支持的文件系统，加上此参数后，挂载点可不必写，因为扫描所有的文件系统了**
- **-u：针对用户扫描文件与目录的使用情况，会新建 aquota.user**
- **-g：针对用户组扫描文件与目录的使用情况会新建 aquota.group**
- **-v：显示扫描过程的信息**
- -f：强制扫描文件系统，并写入新的 quota 配置文件(危险)
- -M：强制以读写的方式扫描文件系统，只有在特殊情况下才会使用
- 只要记得-avug 一起执行即可

3、不要手动编辑 aquota.user 和 aquota.group，这个文件系统进行操作时，操作结果会影响磁盘，会记录到这两个文件中，新建这两个文件要使用 quotacheck 而不要手动编辑

16.1.5. 实践 Quota 流程 3：Quota 启动、关闭与限制值设置

1、<quotaon>

- 启动 quota 的服务
- quotaon [-avug]
- quotaon [-vug] [挂载点]
- -a：根据/etc/mtab 文件系统设置启动有关的 quota，若不加-a，则后面需

要加上特定的文件系统

- -u：针对用户启动 quota(aquota.user)
- -g：针对用户组启动 quota(aquota.group)
- -v：显示启动过程的相关信息
- 这个 quotaon -auvg 命令几乎只在第一次启动 quota 时才需要进行，因为等你重启系统时，系统的/etc/rc.d/rc.sysinit(好像找不到了)这个初始化脚本就会自动执行这个命令了，因此，只需要在这次进行一次即可，将来都不需要自行启动 quota

2、<quotaoff>

- 关闭 quota 的服务
- quotaoff [-a]
- quotaoff [-ug] [挂载点]
- -a：根据/etc/mtab 将全部的文件系统的 quota 都关闭
- -u：仅针对后接的挂载点关闭 user quota
- -g：仅针对后接的挂载点关闭 group quota

3、<edquota>

- 编辑账号/用户组的限制与宽限时间
- edquota [-u username] [-g groupname]
- edquota -t **<==修改宽限时间，不接组名或用户名，进入 vi 界面**
- edquota -p [模板账号] -u [新账号]
- -u：后接账号名称，可进入 quota 的编辑界面(vi)去设置 username 的限制值
- -g：后接组名可进入 quota 的编辑界面(vi)去设置 groupname 的限制值
- -t：可修改宽限时间，可进入 quota 的编辑界面(vi)设置宽限时间限制值
- -p：赋值范本，那个"模板账号"为已存在并已经设置好 quota 的用户，意为将"模板账号"这个人的 quota 限制值复制给新账号
- quota 配置(vi 界面)参数介绍
  - 共有七个字段
  - 文件系统(filesystem)：说明该限制值是针对哪个文件系统
  - 磁盘容量(blocks)：这个数值是 quota 自己算出来的，以 KB 为单位，不要修改
  - soft：磁盘容量的 soft 限制值，单位为 KB
  - hard：block 的 hard 限制值，单位为 KB
  - 文件数量(inodes)：这个是 quota 自己算出来的，不要修改它
  - soft：inode 的 soft 限制值
  - hard：inode 的 hard 限制值
  - 当 soft/hard 为 0 时，表示没有限制的意思

16.1.6. 实践 Quota 流程 4：Quota 限制值报表

1、Quota 的报表主要由两种模式，一种是针对每个个人或用户组的 quota 命令，一个是针对整个文件系统的 repquota 命令

2、<quota>：

- 单一用户的 quota 报表

- quota [-uvs] [用户名]
- quota [-gvs] [用户组名]
- -u：后面接用户名，表示显示出该用户的 quota 限制值，若不接 username，表示显示出执行者的 quota 限制值
- -g：后面可接用户组名，表示显示出该用户组的 quota 限制值
- -v：显示每个用户在文件系统的 quota 值
- -s：使用 1024 为倍数来指定单位，会显示如 M 之类的单位

3、<repquota>

- 针对文件系统的限额做报表
- repquota -a [-vugs]
- -a：直接到/etc/mtab 查询具有 quota 标志的文件系统，并报告 quota 结果
- -v：输出的数据将含有文件系统相关的详细信息
- -u：显示出用户的 quota 限值
- -g：显示出个别用户组的 quota 限值
- -s：使用 M,G 为单位显示结果

16.1.7. 实践 Quota 流程 5：测试与管理

1、例子：

- 用 myquota1 的身份创建 270M 的文件
- dd if=/dev/zero of=bigfile bs=1M count=270 <==预计出现警告信息
- 问题：没有出现警告信息
  - 原因：因为我在配置好 quota 的限值后，重启了一下，导致 quota 没有启动支持(quotaon -auvg，但是这与 P457 说的违背，书上说只要第一次启动，以后系统会自动帮我们启动，但事实上没有启动)
  - 并且在服务打开后，连 root 都没有删除 aquota.user 和 aquota.group 的权限，必须关闭服务后才能删除(quotaoff -ug [挂载点])

2、<warnquota>

- 对超过限额者发出警告信息
- 找不到该命令，也不知道如何安装???

3、<setquota>

- 直接于命令中设置 quota 限额，不通过 vim 的修改
- setquota [-u 用户名] block(soft) block(hard)\
- inode(soft) block(hard) [挂载点]
- setquota [-g 用户组
-] block(soft) block(hard)\
- inode(soft) block(hard) [挂载点]

16.1.8. 不改动既有系统的 Quota 实例

1、/var/spool/mail/目录不是一个独立的分区，但是想要对其进行 quota 的管理

2、由于 quota 是针对文件系统进行设计的，因此无法针对 mail 的使用量给予 quota 的限制

3、想要让/home 与/var/spool/mail 的总量固定，但/home 与/var/spool/mail 不是

同一个文件系统，如何进行整合？

4、解决方法：

- 假设已经有/home 这个独立的分区了
- 将/var/spool/mail 这个目录完整地移动到/home 下面
- 利用 ln -s /home/mail /var/spool/mail 来建立连接数据(soft)
- 将/home 进行 quota 限额配置

16.2. 软件磁盘阵列

16.2.1. 什么是 RAID

1、磁盘阵列的英文全名是 RedundantArrays of Inexpensive Disks(RAID)，即容错廉价磁盘阵列

2、RAID 可以通过一些技术将多个较小的磁盘整合成一个较大的磁盘设备，这个较大的磁盘功能不止是存储而已，它还具有数据保护的功能，整个 RAID 由于选择的等级(level)不同，而使得整合后的磁盘具有不同的功能

3、RAID 等级

- RAID-0(等量模式,stripe)：性能最佳
  - 这种模式如果使用相同型号与容量的磁盘来组成时，效果较佳
  - 这种模式的 RAID 会将磁盘先切出等量的块(举例来说，4KB)，然后一个文件要写入 RAID 时，该文件会依据块的大小切割好后，之后再依序放到各个磁盘里去
  - 由于每个磁盘会交错存放数据，因此当你的数据要写入 RAID 时，数据会被等量放置在各个磁盘上面
  - 越多块磁盘组成的 RAID-0 性能会越好，因为每块负责的数据量就更低了，另外磁盘总容量也变大了，因为每块磁盘的容量最终会加总成为 RAID-0 的总容量
  - 必须承担风险：如果某一块磁盘损毁了，那么文件数据将缺一块，此时这个文件就损毁了，因此 RAID-0 只要有任何一块磁盘损毁，在 RAID 上面的所有数据都会丢失而无法读取
- RAID-1(影像模式,mirror)：完整备份
  - 这种模式也是需要相同的磁盘容量的，最好是一模一样的磁盘
  - 如果是不同容量的磁盘组成的 RAID-1，那么总容量将以最小的那一块磁盘为主
  - 这种模式主要是让同一份数据完整的保存在**两块磁盘**上
  - 例如，有一个 100M 的文件，仅有两块磁盘组成 RAID-1，那么这两块磁盘会同时写入 100M 到它们的存储空间去，因此整体的 RAID 的容量几乎少了 50%
  - 由于两块硬盘内容一模一样，好像镜子映照出来一样，因此称为 mirror(镜像)模式
  - 由于两块磁盘数据一模一样，因此一块损毁时，你的数据还是可以完整保留下来的
  - RAID-1 最大的优点在于数据的备份，由于磁盘容量有一半用在备份，因此总容量会是全部磁盘容量的一半而已
  - RAID-1 的写入性能不佳(只有一个南桥芯片的时候)，读取性能还可以，

因为数据有两份在不同的磁盘上面，如果多个进程在读取同一条数据时，RAID 会自行取得最佳的读取平衡

- RAID-0+1,RAID-1+0
  - 所谓的 RAID-0+1 就是先让两块磁盘组成 RAID-0，并且这样的设置共有两组，然后将这两组 RAID-0 再组成一组 RAID-1。
  - 由于具有 RAID-0 的优点，所以性能得以提升，同时具有 RAID-1 的优点所以数据得以备份，但也由于 RAID-1 的缺点，总容量会少一半用于备份
- RAID-5：性能与数据备份的均衡考虑
  - RAID-5 至**少需要三块以上**的磁盘才能够组成这种类型的磁盘阵列
  - 这种磁盘阵列的数据写入有点类似 RAID-0，不过在每个循环的写入过程中，每块磁盘还会加入一个同为检查数据(Parity)，这个数据会记录其他磁盘的备份数据，用于当时有磁盘损毁时的救援
  - 在每个循环写入时，都会有部分的同位检查码(Parity)被记录起来，并且记录的同位检查码每次都记录在不同的磁盘，因此任何一个磁盘损毁时都能通过其他磁盘的检查码来重建原本磁盘内的数据。
  - 由于有同位检查码，RAID-5 的总容量会是**整体磁盘数量减一块(因为每次循环有一个磁盘用于放置同位检查码)**
  - 当损毁的磁盘数量大于等于两块时，这整组 RAID-5 的数据就损毁了，因为 RAID-5 默认仅能支持一块磁盘的损毁情况
  - 读的性能还行，写的性能较差
- Spare Disk：预备磁盘的功能
  - 当磁盘阵列的磁盘损毁时，就得要将坏掉的磁盘拔除，然后换一块新磁盘，换成新磁盘并且顺利启动磁盘阵列后，磁盘阵列就会主动重建(rebuild)原本坏掉的那块磁盘数据到新的磁盘上，然后你的磁盘阵列上面的数据就复原了，这就是磁盘阵列的优点
  - 为了可以让系统可以实时地在坏掉硬盘时主动重建，因此就需要预备磁盘的辅助
  - 所谓 spare disk 就是一块或多块没有包含在原本磁盘阵列等级中的磁盘，这块磁盘平时并不会被磁盘阵列所使用
  - 当磁盘阵列有任何磁盘损毁时，则这块 spare disk 会被主动拉进磁盘阵列中，并将坏掉的那块硬盘移除磁盘阵列，然后立即重建数据。
  - 若你的磁盘阵列支持热插拔的话，直接将坏掉的那块磁盘拔除换一块新的，再将那块新的设置为 spare disk 就可以了

4、磁盘阵列的优点：

- 数据安全与可靠性：是指当硬件(磁盘)损毁时，数据是否能够安全救援或使用之意
- 读写性能：例如 RAID-0 增强读写性能，让你的系统 I/O 部分得以改善
- 容量：可以让多块磁盘组合起来，故单一文件系统可以有相当大的容量

16.2.2. software，hardware RAID

1、所谓的**硬件磁盘阵列(hardware RAID)**是通过**磁盘阵列卡**来完成数组的目的的

- 磁盘阵列卡上面有一块专门的芯片处理 RAID 的任务，因此在性能方面会

比较好

- 在很多任务时(例如 RAID-5 的同位检查码计算)磁盘阵列并不会重复小号原本系统的 I/O 总线，理论上性能较佳
- 目前一般中高级磁盘阵列卡都支持热插拔，即在不关机的情况下抽换损坏的磁盘，对于系统的复原与数据的可靠性方面非常好用

2、由于磁盘阵列卡动辄上万(现在呢???)，便宜的主板上面附赠的磁盘阵列卡不支持某些高级功能，此外，操作系统也必须要具有磁盘阵列卡的驱动程序，才能够正确识别磁盘阵列所产生的磁盘驱动器，于是发展了软件磁盘阵列

3、所谓的软件磁盘阵列(software RAID)主要通过软件来仿真数组的任务

- 会消耗较多的系统资源，比如 CPU 的运算与 I/O 总线的资源，但是由于目前计算机速度非常快，这些消耗可以忽略不计
- CentOS 提供的软件磁盘阵列为 mdadm 这套软件
  - 这套软件会以分区或磁盘为单位，也就是说不需要两块以上的磁盘，只要有两个以上的分区就能够设计你的磁盘阵列了
  - 此外 mdadm 支持 RAID-0/RAID-1/RAID-5/spare disk 等，而且提供的管理机制还可以达到类似热插拔的功能，可以在线进行分区的抽换，使用上也非常方便

4、**硬件磁盘阵列**在 Linux 下面看起来就是一块实际的大磁盘，**因此磁盘阵列的设备文件名为/dev/sd[a-p]**，因为使用到 SCSI 的模块之故。至于**软件磁盘阵列**则是系统仿真的，因此使用的设备文件名是系统的设备文件，**文件名为/dev/md0 /dev/md1，两者的设备文件名并不相同**

16. 2. 3. 软件磁盘阵列的设置

1、<mdadm>

- mdadm --detail /dev/md0
- mdadm --create --auto=yes /dev/md[0-9] --raid-devices=N\
- --level[015] --spare-devices=N /dev/sdx /dev/hdx
- --create：新建 RAID 的参数
- --auto=yes：决定新建后面接的磁盘阵列设备
- --raid-devices=N：使用几个磁盘作为磁盘阵列设备
- --level[015]：设置这组磁盘阵列的等级，支持很多，建议只用 0,1,5 即可
- --detail：后面所接的那个磁盘阵列设备的详细信息

2、创建 5 个分区，每个大小为 1000M，总共 5000M

- 注意，在创建分区时，如果没有主分区时，需要创建扩展分区，该扩展分区不是一个可以使用的分区，而是所有逻辑分区的总和
- 假如我要创建 5 个逻辑分区，而此时扩展分区尚未创建，那么我们需要创建的扩展分区至少为 5000M+1M，因为有额外的扇区用于记录柱面，所以额外分配 1M
- 创建完后记得 partprobe

3、例子：

- mdadm --create --auto=yes /dev/md0 --level=5 \
- > --raid-devices=4 --spare-devices=1 /dev/sda{5,6,7,8,9}
- mdadm --detail /dev/md0

4、**查阅系统软件磁盘阵列的情况**

- cat /proc/mdstat
  - Personalities: [raid6] [raid5] [raid4]
  - md0: active raid5 sda8[5] sda9[4](S) sda7[2] sda6[1] sda5[0]
  - 3068928 blocks super 1.2 level 5, 512k chunk, algorithm 2 [4/4] [UUUU]
- 第一行指出 md0 为 raid5，且使用了 sda8，sda7，sda6，sda5 四块磁盘，每个设备后面的中括号[]内的数字为此磁盘在 RAID 中的顺序(RaidDevice)，sda9 后面的[s]则代表 spare 的意思
- 第二行：
  - 此阵列拥有 3068928 个 block
  - 使用 raid 等级 5
  - 写入磁盘的小区块(chunk)大小为 512k
  - 使用 algorithm2 磁盘阵列算法
  - [m/n]代表此数组需要 m 个设备，且 n 个设备正常运行
  - 后面的[UUUU]代表四个设备启动情况，U 为正常，_为
  - 正常

5、格式化与挂载使用 RAID
- mkfs -t ext4 /dev/md0
- mkdir /mnt/raid
- mount /dev/md0 /mnt/raid <==挂载

16.2.4. 仿真 RAID 错误的救援模式

1、<mdadm>
- mdadm --manage /dev/md[0-9] [--add 设备] [--remove 设备] \
- > [--fail 设备]
- --add：会将后面的设备加入到 md 中
- --remove：会将后面的设备从这个 md 中删除
- --fail：会将后面的设备设置成为错误的状态

2、仿真步骤
- cp -a /etc /var/log /mnt/raid
- mdadm --manage /dev/md0 --fail /dev/sda8
  - 如果此时 RAID 中有 spare disk 存在，那么会自动修复，用 spare disk 去接替坏掉的磁盘
- mdadm --detail /dev/md0
- cat /proc/mdstat
  - ...sda8[F]... <==8 后面的状态变为[F]
- fdisk 创建一个新的分区，假设为/dev/sda10
- mdadm --manage /dev/md0 --add /dev/sda10 --remove /dev/sda8

16.2.5. 开机启动启动 RAID 并自动挂载

1、software RAID 也有配置文件，这个配置文件在/etc/mdadm.conf 中
- 你只要知道/dev/md[0-9]的 UUID 就能够设置这个文件
- mdadm --detail /dev/md0 | grep -i uuid <==找到 UUID
  - 假设为 7c60c049:57d60814:bd9a77f1:57e49c5b

- vim /etc/mdadm.conf
  - 如下添加
  - ARRAY /dev/md0 UUID=7c60c049:57d60814:bd9a77f1:57e49c5b
- vim /etc/fstab
  - /dev/md0 /mnt/raid ext3 defaults 1 2

16.2.6. 关闭软件 RAID

1、<mdadm>

- umount /dev/md0 <==先卸载
- vim /etc/fstab <==删掉或注释掉添加的那一行
- mdadm --stop /dev/md0
- vim /etc/mdadm.conf <==删掉或注释掉添加的那一行

16.3. 逻辑卷管理器(Logical Volume Manager)

1、LVM 的重点在于可以弹性调整文件系统的容量，而并非在于性能与数据保全上面

2、若需要文件的读写性能或者是数据的可靠性，可以利用 RAID

3、LVM 可以整合多个物理分区在一起，让这些分区看起来就像是一个磁盘一样，而且还可以在将来其他的物理分区或将从这个 LVM 管理的磁盘当中删除

16.3.1. 什么是 LVM：PV,PE,VG,LV 的意义

1、LVM 的做法是讲几个物理的分区(或磁盘)通过软件组合成一块看起来是独立的大磁盘(VG)，然后将这块大磁盘再分为可使用分区(LV)，最终就能够挂载使用了

2、Physical Volume,PV,物理卷

- 我们实际的分区需要调整系统标识符(System ID)成为 8e(LVM 的标识符)，然后再经过 pvcreate 的命令，将它转成 LVM 最底层的物理卷(PV)，之后才能将这些 PV 加以利用
- 调整 System ID 的方法就是通过 fdisk

3、Volume Group,VG,卷用户组

- 所谓的 LVM 大磁盘就是将许多 PV 整合成 VG，**所以 VG 就是 LVM 组合起来的大磁盘**
- 每个 VG 最多今年能包含 65534 个 PE，如果使用 LVM 默认参数，一个 VG 最大可达 256GB 容量

4、Physical Extend,PE,物理扩展块

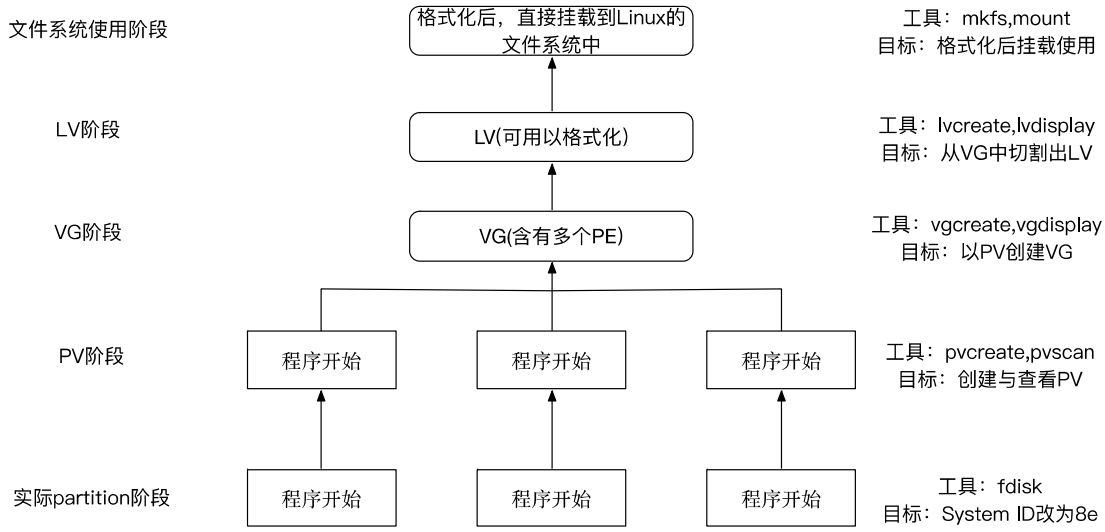
- LVM 默认使用 4MB 的 PE 块，而 LVM 的 VG 最能仅能含有 65534 个 PE，因此默认的 LVM VG 会有 4M*65534=256G
- PE 是 LVM 最小的存储块，我们的文件数据都是由写入 PE 来处理的
- 这个 PE 有点类似文件系统的 block 的大小

5、Logical Volume,LV,逻辑卷

- **最终的 VG 还会被切成 LV，这个 LV 就是最后可以被格式化使用的类似分区**

- LV 不可以随意指定大小，因为 PE 是 LVM 最小存储单位，所以 LV 的大小就与 LV 内的 PE 总数有关
- LV 的设备文件名通常为/dev/vgname/lvname
- 通过 PE 来进行数据转换，将原本 LV 内的 PE 转移到其他设备中以降低 LV 容量，或将其他设备的 PE 加到此 LV 中以加大容量

6、实现流程



7、写入的机制：

- **线性模式(linear)**：加入将/dev/hda1,/dev/hdb1 这两个分区加入到 VG 当中，并且整个 VG 只有一个 LV 时，那么线性模式就是等/dev/hda1 的容量用完之后,/dev/hdb1 的硬盘才会被用到，**建议的模式**
- **交错模式(triped)**：将一条数据拆分成两份，分别写入/dev/hda1 与/dev/hdb1，有点类似 RAID-0,读写性能会比较好
- 基本上 LVM 最主要的用处是在实现一个可以弹性调整容量的文件系统上，而不是在新建一个性能为主的磁盘上，因此 LVM 默认的读写模式是线性模式
- 若使用 triped 模式，当任何一个分区破损时，所有数据都会损坏，如果要强调性能与备份，那么直接使用 RAID 即可，不需要用到 LVM

16. 3. 2. LVM 实作流程

1、LVM 必须要有内核支持且需要安装 lmv2 这个软件

2、要求

- 四个分区，每个分区 1.5G，System ID 设为 8e
- 全部的分区整合成一个 VG，VG 名称位置为 liuyevg，PE 的大小为 16MB
- 全部的 LG 容量都丢给 LV，LV 的名称设置为 liuyelv
- 最终这个 LV 格式化为 ext4 文件系统，且挂载在/mnt/lvm 中

3、新建物理分区，在 fdisk 内部用 t 命令来修改 System ID

4、PV 阶段

- 命令介绍
  - **<pvcreate>**：将物理分区新建成为 PV

- pvcreate [PV 名称]
- **<pvscan>**：查询目前系统里面任何具有 PV 的磁盘
  - pvscan
- **<pvdisplay>**：显示出目前系统上面的 PV 状态
  - pvdisplay
- **<pvmove>**：用于缩小 LV 容量时
  - pvmove [来源 PV] [目标 PV]
  - 将来源 PV 中的 PE 中的数据全部移动到目标 PV 中去
  - **移动后，来源 PV 中将多出 Free PE，目标 PV 将减少等量的 Free PE**
- **<pvremove>**：将 PV 属性删除，让该分区不具备 PV 属性
  - pvremove [PV 名称]

➢ 命令流程

- pvscan
- **pvcreate /dev/sda{5,6,7,8}　<==关键命令**
  - 如果接着上面的 RAID 来做 LVM 的联系，记得删除 RAID，即 mdadm --stop /dev/md0，否则会出现新创建的分区被占用的错误
- pvscan
- pvdisplay

➢ 遇到的问题

- 在做完 RAID 的练习之后，删除了/dev/md0 的情况下(mdadm --stop)
- 执行 fdisk 删除 RAID 的分区，w 后**无论是否 partprobe**，又执行 fdisk 创建分区用于 LVM 的练习，w 后 partprobe，此时发现/dev/md0 又出现了
- 再执行一次 mdadm --stop /dev/md0 即可

5、VG 阶段

➢ 相关命令

- **<vgcreate>**：新建 VG
  - vgcreate [-s N(m|g|t)] [VG 名称] [PV 名称]
  - -s：后接 PE 的大小，单位可以是 m,g,t
- **<vgscan>**：查看系统上面是否有 VG 存在
  - vgscan
- **<vgdisplay>**：显示目前系统上面的 VG 状态
  - vgdisplay
- **<vgextend>**：在 VG 内增加额外的 PV
  - vgextend [VG 名称] [PV 名称]
- **<vgreduce>**：在 VG 内删除 PV
  - vgreduce [VG 名称] [PV 名称]
- **<vgchange>**：设置 VG 是否启动(active)
- **<vgremove>**：删除一个 VG

➢ 与 PV 不同的是，VG 的名字是自定义的，而 PV 的名称就是分区设备的文件名

➢ 命令流程

- **vgcreate -s 16M liuyevg /dev/sda{5,6,7}　<==关键命令**
- vgscan
- pvscan
- vgdisplay
- **vgextend liuyevg /dev/sda8　<==关键命令**

6、LV 阶段

➢ 相关命令

- **<lvcreate>**：新建 LV
  - lvcreate [-L n(m|g|t)] [-n LV 名称] [VG 名称]
  - lvcreate [-l N] [-n LV 名称] [VG 名称]
  - lvcreate [-l N] -s [-n **快照区 LV 名称**] [**被快照的 LV 设备文件名**]
  - **注意，名称使得是名字，没有路径的含义，设备文件名是一个包含路径的文件名**
  - -L：后接容量，容量单位可以是 M,G,T，最小单位为 PE，因此这个数量必须是 PE 的倍数，若不相符，系统会自行计算最接近的容量
  - -l：后接 PE 个数，推荐
  - -n：后接 LV 名称
  - -s：用于新建快照区
- **<lvscan>**：查询系统上面的 LV
  - lvscan
  - 例如虚拟机的 CentOS7：/home /root /swap 就是三个 LV，三个加起来就是一整个 VG
- **<lvdisplay>**：显示系统上面的 LV 状态
  - lvdisplay
- **<lvextend>**：在 LV 里面增加容量
- **<lvreduce>**：在 LV 里面减少容量
- **<lvremove>**：删除一个 LV
- **<lvresize>**：对 LV 进行容量大小的调整
  - lvresize [-L n(m|g|t)] [LV 文件名]
  - lvresize [-l (+|-)N] [LV 文件名]
  - -l：后接 PE 的改变数量，若增加，则在数字前加'+'，若减少，添加'-'

➢ 流程命令

- lvcreate -l 372 -n liuyelv liuyevg
- **会生成/dev/liuyevg/liuyelv 的设备文件**
- lvdisplay

7、文件系统阶段

➢ 流程命令

- mkfs -t ext4 /dev/liuyevg/liuyelv
- mkdir /mnt/lvm
- mount /dev/liuyevg/liuyelv /mnt/lvm
- df

16.3.3. 放大 LV 容量

1、步骤

- 利用 fdisk 设置新的具有 8e system ID 的分区
- 利用 pvcreate 构建 PV
- 利用 vgextend 将 PV 加入到现有的 VG
- 利用<lvresize>将新加入的 PV 内的 PE 加入现有 VG
  - resize2fs [-f] [device] [size]
  - -f：强制进行 resize
  - [size]：可加可不加，如果加上 size 的话，就必须要给一个单位，譬如 M,G，如果没有 size 的话，默认用整个分区的容量来处理
- **通过 resize2fs 将文件系统的容量确实增加**
- 最后一步最重要
  - 因为文件系统在最初格式化的时候就新建了 inode/block/superblock 等信息，要改变这些信息是很难的
  - 但是由于文件系统格式化的时候新建的是多个 block group，因此可以通过在文件系统中增加 block group 的方式来增减文件系统的容量
  - 最后一步针对文件系统进行处理，前几步针对 LVM 的实际容量大小

2、命令流程

- **pvcreate /dev/sda9 <==关键命令**
- pvscan
- **vgextend liuyevg /dev/sda9 <==关键命令**
- vgdisplay <==用于查看空闲的 PE 个数，便于 lvresize
- **lvresize -l +191 /dev/liuyevg/liuyelv <==关键命令**
- **resize2fs /dev/liuyevg/liuyelv <==关键命令，对比减少容量，增加容量的时候不必卸载(umount)该设备**
-

16.3.4. 缩小 LV 容量

1、类似于扩大 LV 容量

2、步骤(假如要从 LV 中删除/dev/sda6 的分区)

- 首先缩小文件系统的容量
  - 先用 pvdisplay 查看该分区的大小，PE 数量等
  - umount /mnt/lvm **<==必须卸载，增加容量的时候不需要卸载**
  - **resize2fs /dev/liuyevg/liuyelv 7330M <==关键命令**
    - 7330 是我用总容量减去/dev/sda6 的容量计算得到的
    - 运行该语句后，会提示你先运行"e2fsck -f /dev/liuyevg/liuyelv"，照做即可
- 再降低 LV 的容量
  - 用上一步 pvdisplay 查到的/dev/sda6 的 PE 数量为 93
  - **lvresize -l -93 /dev/liuyevg/liuyelv <==关键命令**
  - lvdisplay <==可以看到 LV size 确实减少了
  - **pvdisplay <==关键命令，找到被减少的 free PE 放置在哪里，**

在/dev/sda9 中

- **pvmove /dev/sda6 /dev/sda9　<==关键命令**
- pvdisplay　<==可以看到/dev/sda6 中的 PE 全部是 FREE 的
- **vgreduce liuyevg /dev/sda6　<==关键命令**
- pvremove /dev/sda6

16. 3. 5.　　LVM 的系统快照

1、快照就是讲当时的系统信息记录下来，就好像照相记录一般，若将来有任何数据改动了，**则原始数据会被移动到快照区**，没有被改动的区域则由快照区域文件系统共享

2、LVM 的系统快照是非常棒的备份工具，因为它只备份有被改动的数据，文件系统内没有被更改的数据依旧保持在原本的区块内，快照占用的容量非常小

3、由于快照区域被快照 LV 必须要在同一个 VG

4、快照区的新建

- vgdisplay　<==查看还有多少余量(Free PE 的数量)，如果没有的话需要扩展 LV
- **lvcreate -l 93 -s -n liuyess /dev/liuyevg/liuyelv　<==关键命令，会产生名为 /dev/liuyevg/liuyess 的设备文件**
- 快照区可以被挂载，与原本的 LV 一模一样，但是显示的容量会小于该 LV 在 VG 中占有的容量，因为有部分容量用于快照，而这部分容量是不会显示在 LV 中的

5、利用快照区复原系统

- **要复原的数据量不能够高于快照区所能负载的实际容量**，即，如果原始数据被改动的实际数据量比快照区要大，那么快照功能会失效
- 首先要备份快照的数据
  - mkdir -p /backups
  - cd /mnt/snapshot
  - tar -jcv -f /backups/lvm.tar.bz2 *
  - 为什么不直接格式化/dev/liuyevg/liuyelv，然后拷贝/dev/liuyevg/liuyess 到/dev/liuyevg/liuyelv 中？因为有部分是共享的，格式化后，从快照区当然也无法访问到了
- 卸载并删除快照区(因为已经备份过了)
  - umount /mnt/snapshot
  - lvremove /dev/liuyevg/liuyess
- 卸载并格式化被快照的 LV，然后重新挂载，将备份文件拷贝过去
  - umount /mnt/lvm
  - mkfs -t ext4 /dev/liuyevg/liuyelv
  - mount /dev/liuyevg/liuyelv /mnt/lvm
  - tar -jxv -f /backups/lvm.tar.bz2 -C /mnt/lvm　<==指定的必须是目录(挂载点)，而不能指定/dev/liuyevg/liuyelv

6、**换个角度**

- 我们将原本的 liuyelv 当做备份数据，然后将 liuyess 当做实际运作的数据，任何测试的操作都在 liuyess 这个快照区中测试，当测试完毕只需要将快照区删除即可

- 为什么快照区改动数据删除后就没有影响了???，假设初始快照区与原始 LV 一模一样，然后在快照区改动了文件 a，那么原始的文件 a 会被放置到快照区中，新的文件 a 占据了原本的数据，然后此时将快照区删掉，那么原来的文件不就没有了吗???

16.3.6. LVM 相关命令汇整与 LVM 的关闭

1、LVM 相关命令汇整

任务	PV 阶段	VG 阶段	LV 阶段
查找(scan)	pvscan	vgscan	lvscan
新建(create)	pvcreate	vgcreate	lvcreate
显示(display)	pvdisplay	vgdisplay	lvdisplay
增加(extend)		vgextend	lvextend(lvresize)
减少(reduce)		vgexduce	lvreduce(lvresize)
删除(remove)	pvremove	vgremove	lvremove
移动(move)	pvmove		
改变容量(resize)			lvresize
改变属性(attribute)	pvchange	vgchange	lvchange

2、关闭 LVM 设备并删除

- 先卸载系统上的 LVM 文件系统(包括快照与 LV)
- 使用 lvremove 删除 LV
- 使用 vgchange -a [-n VG 名称] <==让这个 VG 不具有 active 的标志
- 使用 vgremove 删除 VG
- 使用 pvremove 删除 PV
- 最后用 fdisk 将 system id 改回来，或删除

Chapter 17. 例行性工作(crontab)

17.1. 什么是例行性工作

- Linux 调度就是通过 crontab 和 at 来实现的

17.1.1. Linux 工作调度的种类:at,cron

1、两种工作调度方式：

- 例行性的：每隔一定周期要来办的事项
- 突发性的：这次做完以后就没有的那种

2、<at>

- at 是个可以处理仅执行一次就结束调度的命令，不过要执行 at 时，必须要有 atd 这个服务的支持才行
- 在某些 distributioins 中，atd 可能默认并没有启动，那么 at 这个命令就会失效(CentOS 是默认启动的)

3、<crontab>

- crontab 这个命令所设置的工作将会循环一直进行下去，可循环的时间为分钟、小时、每周或每年
- crontab 除了可以使用命令执行外，还可以编辑/etc/crontab 来支持
- 让 crontab 生效的服务是 crond

17.1.2. Linux 上常见的例行性工作

1、Linux 会主动帮我们进行一些工作，包括

- 自动进行在线更新(on-line update)
- 自动更新文件名数据库(updatedb)
- 自动进行日志文件分析(所以 root 常常会收到标题为 logwatch 的信件)

2、**日志文件的轮替(log rotate)**

- Linux 会主动将系统所发生的各种信息都记录下来，这就是日志文件
- 由于系统会一直记录日志信息，所以日志文件将会越来越大
- 所以适时将日志文件数据挪一挪，让旧的数据与新的数据分别存放，则可以比较有效地记录日志信息，这就是 log rotate 的任务，这也是系统必要的例行任务

3、**日志文件分析 logwatch 的任务**

- 如果系统发生了软件问题、硬件错误等，绝大部分的错误信息都会被记录到日志在文件中
- 管理员的重要任务之一就是分析日志文件

4、**新建 locate 的数据库**

- locate 命令是通过已经存在的文件名数据库来进行系统中文件名的查询
- 文件名数据库放到/var/lib/mlocate/中
- 系统会主动 updatedb

5、**whatis 数据库的建立**

- 与 locate 数据库类似，whatis 也是个数据库，这个 whatis 是与 man page 有关的一个查询命令
- 要使用 whatis 命令时，必须要拥有 whatis 数据库，这个数据库也是通过

系统的例行性工作调度来自动执行的

6、**RPM 软件日志文件的新建**

- 系统可能会经常更改软件，包括软件的新安装、非经常性更新等，都会造成软件文件名的区别
- 系统帮我们将文件名做个排序的记录

7、**删除临时文件**

- 某些软件在运行中会产生一些临时文件，但是当这个软件关闭时，这些临时文件可能并不会主动被删除
- 系统通过例行性工作调度执行名为 tmpwatch 的命令来删除临时文件

8、**与网络服务有关的分析行为**

- 如果安装了 WWW 服务器软件，Linux 系统通常会主动分析该软件的日志文件
- 同时某些认证的网络信息是否过期的问题，Linux 也会帮你自动检查

17.2. 仅执行一次的工作调度

17.2.1. atd 的启动与 at 的运行方式

1、手动启动 atd 服务的方式

- /etc/init.d/atd restart <==???
- systemctl start atd.service (service atd start) <==网上查到的
- systemctl status atd.service (service atd status) <==查看服务状态
- systemctl stop atd.service (service atd stop) <==网上查到的
- systemctl enable atd.service (chkconfig atd on 已失效) <==开机启动该服务

2、at 的运行方式

- 我们使用 at 这个命令来生成所要运行的工作，并将这个工作一文本文件的方式写入/var/spool/at/目录内，该工作便能等待 atd 这个服务的取用与执行
- 并不是所有人都可以进行 at 工作调度
  - 我们可以利用/etc/at/allow 与/etc/at/deny 这两个文件来进行 at 的使用限制
- at 的工作情况
  - 先寻找/etc/at.allow 文件，写在这个文件中的用户才能使用 at，**没有在这个文件中的用户则不能使用 at(即使没有写在 at.deny)**
  - 如果/etc/at.allow 不存在，就寻找/etc/at.deny 这个文件，**若写在 at.deny 的用户则不能使用 at，没有在 at.deny 文件中的用户可以使用 at**
  - 如果两个文件都不存在，则只有 root 可以使用 at 命令
  - /etc/at.allow 是管理较为严格的方式，而/etc/at.deny 是较为松散的，/etc/at.allow 的优先级更高
  - 通常系统会保留一个空的/etc/at.deny 文件，而没有/etc/at.allow 文件

17.2.2. 实际运行单一的工作调度

1、<at>

- at [-mldv] TIME
- at -c 工作号码
- -m：当 at 的工作完成后，即使没有输出信息，以 email 通知用户改工作已经完成
- -l：at -l 相当于 atq，列出目前系统上面所有该用户的 at 调度
- -d：at -d 相当于 atrm，可以取消一个在 at 调度中的工作
- -v：可以使用较明显的时间格式列出 at 调度中的任务列表
- -c：可以列出后面接的该项工作的实际命令内容
- TIME：时间格式
  - HH:MM
    - 04:00
  - HH:MM YYYY-MM-DD
    - 04:00 2009-03-17
  - HH:MM[am|pm] [Month] [Date]
    - 04pm March 17
  - HH:MM[am|pm] [+ N] [minutes|hours|dars|weeks]
    - now +5 minites
    - 04pm+3days

2、建议最好使用绝对路径来执行命令，比较不会有问题
- 命令的执行与 PATH 变量有关，也与当时的工作目录有关，因此使用绝对路径是比较一劳永逸的方法

3、at 的执行与终端机无关，而所有的 standard output/standard error 都会传送到执行者的 mailbox，因此，在 at 中执行类似"echo $PATH"等命令是无法在屏幕上显示的
- 可以使用 echo "Hello" > /dev/tty1 将信息通过流重定向到终端机显示屏上
- **<tty>与<who>查看目前所属的终端**

4、at 另外一个很棒的优点就是**后台执行(系统后台，而非 bash 后台)**
- 由于在 at 工作调度的使用上，系统会将该项 at 工作独立出你的 bash 环境中，直接交给系统的 atd 程序来接管
- 因此，当你执行了 at 的工作之后就可以立即脱机了，剩下的工作就完全交给 Linux 管理即可。
- 如果有长时间的网络工作，使用 at 可以让你免除网络断线后的困扰

5、at 工作的管理：
- 删除命令：<atq>,<atrm>
  - atq <==查询目前主机上面所有尚未执行的 at 调度
  - atrm [jobID] <==删除工作

6、batch：系统有空时才进行后台任务
- batch 是利用 at 来执行命令的，只是加入一些控制参数而已
- **batch 会在 CPU 工作负载小于 0.8 的时候，才进行你所执行的工作任务**
- **工作负载：CPU 在单一时间点所负责的工作数量**
- **batch 执行方式与 at 一样**
- **batch 也是利用 atq 与 atrm 来管理的**

17.3. 循环执行的例行性工作调度

- 循环执行的例行性工作调度是由 cron(crond)这个系统服务来控制的
- Linux 提供供用户控制例行性工作调度的命令(crontab)

17.3.1. 用户设置

1、与 at 一样，我们可以限制 crontab 的用户账号

- /etc/cron.allow
  - 将可以使用 crontab 的账号写入其中，若不在这个文件内的用户不可用 crontab
- /etc/cron.deny
  - 将不可以使用 crontab 的账号写入其中，若未记录到这个文件当中的用户，就可以使用 crontab
- /etc/cron.allow 的优先级比/etc/cron.deny 要高，一般这两个文件只选择一个来限制，建议只保留一个，系统默认保留/etc/cron.deny

2、**当用户使用 crontab 这个命令来新建工作调度之后，该项工作就会被记录到/var/spool/cron/里面，而且是以账号作为判别的**

- 例如 dmtsai 用户使用 crontab 后，他的工作会被记录到/var/spool/cron/dmtsai 里面，**这是一个文件**

3、cron 执行的每一项工作都会被记录到/var/log/cron 这个日志文件中

4、<crontab>

- crontab [-u 用户名] [-l|-e|-r]
- -u：只有 root 才能进行这个任务，也可以帮其他用户新建/删除 crontab 工作调度
- -e：进入特殊的 vi 界面编辑 crontab 的工作内容
  - 格式：分 时 日 月 周 命令串
  - 分钟：0-59
  - 小时：0-23
  - 日期：1-31
  - 月份：1-12
  - 周：0-7(0 与 7 都代表星期天)
  - *：代表所在字段的任何时候
  - ,：代表分隔时段的意思，例如 3:00 与 6:00==>0 3,6 * * * command
  - -：代表一段时间范围
  - /n：(注意不是反斜线)，例如 ***/5** * * * * command <==每五分钟，与*搭配用
- -l：查阅 crontab 的工作内容
- -r：删除所有的 crontab 的工作内容，若要仅删除一项，用-e 来编辑

17.3.2. 系统的配置文件:/etc/crontab

1、这个 crontab -e 是针对用户的 cron 来设计的，系统的例行性任务存放在配置文件/etc/crontab 中

2、crontab 命令是/usr/bin/crontab 这个可执行文件，/etc/crontab 是纯文本文件

3、/etc/crontab 文件结构

- MAILTO=root：当/etc/crontab 这个文件中的例行性工作的命令发生错误时，或是该工作的执行结果有 STDOUT/STDERR 时，会将错误信息或是屏幕显示的信息传给谁，默认是由系统直接发一封 email 给 root
- PATH：执行文件路径
- SHELL：就是指定哪一个 shell
- 分 时 日 月 周 **用户**
- **命令串**：
  - 用户名：执行后面命令的身份，这与 crontab -e 不同，由于用户自己的 crontab 并不需要指定身份(/var/spool/cron/username)，但是/etc/crontab 里面的当然要指定身份

4、/etc/crontab 的命令串的两种执行方式

- 直接执行命令
  - 分 时 日 月 周 用户名 username [正常的命令]
- 以目录来规划
  - 分 时 日 月 周 用户名 username **run-parts [script 所在目录]**
  - 如果你想让系统**每小时**主动帮你执行某个命令，将该命令写成 script，并将该文件放置到/etc/cron.hourly/目录下即可，**当然这个目录取什么名字，放在哪里都可以自己设定**
  - Linux 系统在每天凌晨 4:02 开始执行例行任务

17. 3. 3. 一些注意事项

1、资源分配不均的问题

- 当大量使用 crontab 的时候，总会有问题发生，最严重的问题就是系统资源分配不均的问题
- 如果每个流程都在同一时间启动，那么在某个时段，系统会相当繁忙，可以手动进行划分
  - 0,5,10,15,20,25,30,35,40,45,50,55 * * * * root CMD0
  - 1,6,11,16,21,26,31,36,41,46,51,56 * * * * root CMD1
  - 2,7,12,17,22,27,32,37,42,47,52,57 * * * * root CMD2
  - 3,8,13,18,23,28,33,38,43,48,53,58 * * * * root CMD3
  - 4,9,14,19,24,29,34,39,44,49,54,59 * * * * root CMD4
  - **使用逗号','的时候，不要有空格符**

2、取消不要的输出项

- 利用流重定向输出到/dev/null

3、安全的检测

- 很多时候，被植入木马都是以例行命令的方式植入的，可以由检查/var/log/cron 的内容来看是否有非你设置的 cron 被执行

4、周与日、月不可同时并存

- 你可以采用以周或日或月为单位的循环，但是不能指定几月几号且为星期几的模式工作，即必须要有*存在

17. 4. 可唤醒停机期间的工作任务

1、如果你的 Linux 主机是作为 24 小时全天、全年无休的服务器之用，那么你只

要有 atd 与 crond 这两个服务来管理你的例行工作调度即可
2、问题：如果每天晚上都要关机，等到白天才启动 Linux 主机，但是默认的工作调度都在凌晨 4:02 进行，这样不就造成工作没人做了？此时需要 **anacron** 了

17.4.1. 什么是 anacron

1、anacron 并不是用来替代 crontab 的，anacron 存在的目的就处理非 24 小时一直启动的 Linux 系统的 crontab 的执行
2、**anacron 并不能指定何时执行某项任务，而是以天为单位或是在开机后立刻执行 anacron 的操作，它会去检测停机期间应该进行但是没有进行的 crontab 任务，并将该任务执行一遍，然后 anacron 就会停止了**
3、anacron 也是通过 crontab 来运行的，因此 anacron 运行时时间通常有两个，一个是系统开机期间运行，一个是写入 crontab 的调度中

17.4.2. anacron 与/etc/anacrontab

1、anacron 是一个程序并非一个服务
2、anacron 的用法

- anacron [-sfn] [job]...
- anacron -u [job]...
- -s：开始连续执行各项工作(job)，会依据时间记录文件的数据判断是否进行
- -f：强制执行，而不去判断时间记录文件的时间戳
- -n：立刻进行未进行的任务，而不延迟等待时间
- -u：仅更新时间记录文件的时间戳，而不进行任何工作
- job：由/etc/anacrontab 定义的各项工作名称

Chapter 18. 程序管理与 SELinux 初探

18.1. 什么是进程

1、在 Linux 系统中，出发任何一个时间时，系统都会将它定义为一个进程，并且基于这个进程一个 ID，称为 PID，同时依据出发这个进程的用户与相关属性关系，给予这个 PID 一组有效的权限设置

18.1.1. 进程与程序(process&program)

1、执行一个程序或命令就可以出发一个事件而取得一个 PID

2、**系统仅认识二进制文件，当我们需要系统工作的时候，需要启动一个二进制文件，那个二进制文件就是程序**

3、程序一般是放在磁盘中，通过用户的执行来触发，触发后会加载到内存中成为一个个体，那就是进程

4、为了操作系统可管理这个进程，因此进程有给予执行者的权限/属性等参数，包括进程所需要的脚本与数据文件数据等，最后再给予一个 PID，系统就是通过 PID 来判断该 process 是否具有权限进行工作

5、**由进程衍生出来的其他进程在一般状态下，会沿用这个进程的相关权限**(例如，在以 root 的身份执行/bin/bash，并在 bash 中执行 touch 命令)

6、总结

- **程序(program)**：通常为二进制程序放置在存储媒介中(硬盘、光盘、软盘、磁带等)，以物理文件的形式存在
- **进程(process)**：程序被触发后，执行者的权限与属性、程序的程序代码所需数据都会被加载到内存中，操作系统并给予这个内存内的单元一个标识符(PID)，可以说，进程就是一个正在运行的程序

7、子进程与父进程

- 例如：登陆系统后取得一个 bash 的 shell，利用 bash 提供的接口去执行另一个命令，例如 touch，那些另外执行的命令也会被触发成为 PID
- 每个进程都有一个 PID，可以通过 Parent PID(PPID)来判断该进程的父进程
- 子进程可以取得父进程的环境变量

8、fork and exec：过程调用的流程

- 子进程和父进程之间的关系挺复杂，最大的复杂点在于进程互相之间的调用，在 Linux 的过程调用中通常称为 fork-and-exec 的流程
- **系统先以 fork 的方式复制一个与父进程相同的暂存进程，这个进程与父进程唯一的区别就是 PID 不同，并且暂存进程还多一个 PPID 的参数，PPID 就是父进程的 PID**
- 然后暂存进程开始以 exec 的方式加载实际要执行的程序，例如新的进程名为 qqq，最终子进程的程序代码就会变成 qqq

7、系统或网络服务：常驻在内存的进程

- 系统每分钟都会去扫描/etc/crontab 以及相关的配置文件，来进行工作调度，是 crond 这个进程锁管理的，它启动后再后台当中一直持续不断运行
- 常驻在内存当中的进程通常都是负责一些系统所提供的功能以服务用户各项任务，**因此这些常驻进程就会被称为服务**

18.1.2. Linux 的多用户、多任务环境

1、在 Linux 下面执行一个命令时，系统会将相关的权限、属性、程序代码与数据均加载到内存，并给予这个单元一个进程标识符(PID)，最终该命令可以进行的任务则与这个 PID 的权限有关

2、**多用户环境**

- Linux 系统上面有多种不同的账号，每种账号都有其特殊的权限，只有一个人具有至高无上的权利，root，其他人都必须受到一些限制
- 每个人进入 Linux 环境设置都可以随着每个人的喜好来设置(~/.bashrc)，因为每个人登陆后取得的 shell 的 PID 不同

3、**多任务行为**

- CPU 每秒能够在各个进程之间进行切换
- CPU 切换到进程的工作与这些工作进入到 CPU 运行的调度(CPU 调度，非 crontab 调度)会影响系统整体性能

4、**多重登陆环境的七个基本终端窗口**

- 在 Linux 当中，默认提供了 6 个命令行界面登陆窗口，以及一个图形界面，你可以使用[Atl]+[F1]~[F7]来切换不同的终端机接口，而且每个终端机接口的登陆者还可以是不同的人(在进程死掉的时候很有用)
- 我们的 Linux 默认会启动 6 个终端机登陆环境的进程

5、**特殊的进程管理行为**

- Linux 几乎可以说绝对不会死机，因为它可以在任何时候，将某个被困住的进程杀死掉
- **当进程死掉后(动弹不得)，利用[Atl]+[F1]~[F7]来切换不同的终端机接口，以 ps -aux 找出刚才错误的进程，然后杀死，就可以回到刚才的终端机接口了**
- **可以这样做的原因是，进程之间是独立的**

6、**bash 环境下的工作管理**

- 在单一的 bash 接口下，进行多个工作：利用&
  - cp file1 file2 **&**
  - **&：执行命令后，这个终端仍然能做其他工作，当这个命令执行完毕后，系统会在你的终端显示完成的消息**

7、**多用户、多任务的系统资源分配问题考虑**

- 由于用户多，当用户达到一定人数后，通常你的机器便需要升级了，因为 CPU 的运算与 RAM 的大小可能就会不够用

18.2. 工作管理

- 这个工作管理是用在 bash 环境下的，也就是说，当我们登陆系统取得 bash shell 之后，在单一终端机下同时进行多个工作的行为管理

18.2.1. 什么是工作管理

1、在进行工作管理的行为中，每个工作都是目前 bash 的子进程，即彼此之间都是有相关性的，我们无法以 job control 的方式由 tty1 的环境去管理 tty2 的 bash

2、为什么要这样做？因为我们可以在/etc/security/limits.conf 中设置用户可以同时登陆的连接数，那么某些用户可能仅能以一个连接来工作

3、前台后台：

- **可以出现在提示符让你操作的环境就称为前台，至于其他工作就可以放入后台去暂停或运行**
- **放入后台的工作想要运行时，它必须不能够与用户互动**
- 放入后台的工作是不能用[Ctrl]+C 来终止的

4、总结：

- **前台**：你可以控制与执行命令的这个环境称为前台工作
- **后台**：可以自动运行的工作，无法用[Ctrl]+C 终止，可使用 bg/fg 调动该工作
- 后台中执行的进程不能等待 termial/shell 的输入(input)，即不能有交互

18.2.2. job control 的管理

1、直接将命令丢到后台执行:&

- 在命令最后加上&代表将命令丢到后台执行
- 此时 bash 会给予这个命令一个工作号码(job number)，后接该命令触发的 PID
- 不能被[Ctrl]+C 中断
- **在后台中执行的命令，如果有 stdout 以及 stderr 时，它的数据依旧是输出到屏幕上面，所以我们会无法看到提示符，命令结束后，必须按下[Enter]才能看到命令提示符**，同时也无法用[Ctrl]+C 中断。解决方法就是利用数据流重定向
  - tar -zpcv -f /tmp/etc.tar.gz /etc > /tmp/log.txt 2>&1 **&**

2、工作号码只与当前 bash 有关，因此 job number 会搭配一个 PID

3、**后台中的工作状态可分为"暂停(stop)"与"运行中(running)"**

3、将工作丢到后台中"暂停":[Ctrl]+Z

- 例如在编辑 vim 中发现有个文件不知道放在哪里，需要到 bash 环境中查找，此时只要暂时将 vim 丢到后台中等待即可

4、<jobs>

- jobs [-lrs]
- -l：除了列出 job number 与命令串之外，同时列出 PID 号码
- -r：仅列出正在后台 run 的工作
- -s：仅列出正在后台中暂停(stop)的工作
- 输出信息中的'+'与'-'号的意义：
  - +：最近被放到后台的工作号码，代表默认的取用工作，即仅输入'fg'时，被拿到前台的工作
  - -：代表最近后第二个被放置到后台的工作号码
  - 超过最后第三个以后，就不会有'+'与'-'号存在了

5、<fg>

- 将后台工作拿到前台来处理
- fg %jobnumber
- fg + <==取出标记为+的工作

- fg - <==取出标记为-的工作
- %jobnumber：jubnumber 为工作号码(数字)，%是可有可无的

6、<bg>

- 让工作在后台下的状态变为运行中
- bg %jobnumber
- bg + <==取出标记为+的工作
- bg - <==取出标记为-的工作
- %jobnumber：jubnumber 为工作号码(数字)，%是可有可无的
- **不能让类似 vim 的工作变为运行中，即便使用该命令会，该工作又立即变为暂停状态**

7、<kill>

- 管理后台当中的工作
- kill [- signal] %jobnumber
- kill -l
- -l：列出目前 kill 能够使用的 signal 有哪些
- - signal：
  - -1：重新读取一次参数的配置文件，类似 reload
  - -2：代表与由键盘输入[Ctrl]+C 同样的操作
  - -9：立刻强制删除一个工作，**通常在强制删除一个不正常的工作时使用**
  - -15：以正常的程序方式终止一项工作，与-9 是不同的，**-15 以正常步骤结束一项工作，这是默认值**
  -
- **与 bg fg 不同，kill 中的%不可省略，因为 kill 默认接 PID**
- 管理进程的操作见进程的管理

18. 2. 3. 脱机管理问题

1、在工作尚未结束的情况下脱机了，工作将会被中断

2、可以利用 at 来处理，因为 at 是将工作放置到**系统的后台**，而与终端机的后台无关

3、也可以利用 nohup 这个命令来处理，nohup 可以让你在脱机或注销系统后，还能够让工作继续工作

4、<nohup>

- nohup [命令与参数] <==在终端机前台工作
- nohup [命令与参数] & <==在终端机后台工作
- nohup 并不支持 bash 内置的命令，因此你的命令必须要是外部命令才行

18. 3. 进程管理

1、为什么进程如此重要

- 首先，我们在操作系统时的各项工作其实都是经过某个 PID 来达成的(包括你的 bash 环境)，因此，能不能进行某项工作就与该进程的权限有关
- 若你的 Linux 系统是个很忙碌的系统，那么当整个系统资源快要被使用完时，是否能找出最小号系统资源的进程，然后删除它，让系统恢复正常

- 若由于某个程序写的不好，导致产生一个有问题的进程在内存中，如何找到它并将它删除
- 若同时有几项工作同时进行，如何设置优先级

18. 3. 1. 进程的查看

1、<ps>

- 将某个时间点的进程运行情况选取下来
- ps aux <==查看系统所有进程数据
- ps -lA <==查看所有系统的数据
- ps axjf <==连同部分进程数状态
- -A：所有的进程均显示出来
- -a：不与 terminal 有关的所有进程
- -u：有效用户相关进程
- x：通常与 a 这个参数一起使用，可列出较完整的信息
- l：较长、较详细地将该 PID 的信息列出
- j：工作的格式(job format)
- -f：做一个更为完整的输出
- **ps -l：查阅自己的 bash 中的程序**
- **ps aux：查阅系统所有运行的程序**
- **ps aux -Z：-Z 参数可以让我们查阅进程的安全上下文，详见** SELinux 网络服务运行范例

2、**ps -l：打印参数介绍**

- **F**：代表这个进程标志(process flags)
  - 若为 4：表示此进程的权限为 root
  - 若为 1：表示此子进程尽可进行复制，而无法实际执行
- **S**：代表这个进程的状态(STAT)
  - R(Running)：该进程正在运行中
  - S(Sleep)：该进程目前正在睡眠状态(idle)，但可以被唤醒(signal)
  - D：不可被唤醒的睡眠状态，通常这个进程可能在等待 I/O 的情况
  - T：停止状态(stop)
  - Z(Zombie)：僵尸状态，进程已经终止但无法被删除至内存外
- **UIP/PID/PPID**：用户标识号/进程 PID 号码/父进程的 PID 号码
- **C**：CPU 使用率，单位为百分比
- **PRI/NI**：Prority/Nice 的缩写，代表此进程被 CPU 所执行的优先级，数值越小代表越快被 CPU 执行
- **ADDR/SZ/WCHAN**：都与内存有关
  - ADDR 是 kernel function，指出该进程在内存的哪个部分，如果是个 running 的进程，一般就会显示"-"
  - SZ 表示此进程用掉多少内存
  - WCHAN 表示目前进程是否运行中，'-'表示正在运行中，'wait'表示等待中
- **TTY**：登陆者的终端机位置，若为远程登录则使用动态终端机接口(pts/n)
- **TIME**：使用掉的 CPU 时间，是此进程实际**花费 CPU 运行的时间**

- **CMD**：造成此程序的出发进程的命令为何

3、**ps aux：打印参数介绍**

- **USER**：该进程所属的用户名
- **PID**：该进程的 PID
- **%CPU**：该进程使用掉的 CPU 资源百分比
- **%MEN**：该进程占用的物理内存百分比
- **VSZ**：该进程使用掉的虚拟内存量
- **RSS**：该进程占用的固定内存量
- **TTY**：该进程是所属终端机，tty1~tty6 是本地，pts/0 是网络连接主机的进程(GNOME 上的 bash)
- **STAT**：该进程目前的状态，状态显示与 ps -l 的 S 标志相同
- **START**：该进程被触发的启动时间
- **TIME**：该进程实际使用 CPU 的时间
- **COMMAND**：该进程的实际命令
- 一般来说 ps aux 会按照 PID 的顺序来排序显示

4、僵尸进程

- 通常造成僵尸进程的成因是因为该进程应该已经执行完毕，或者是因故应该要停止了，但是该进程的父进程却无法完整将该进程结束掉，造成那个进程一直在内存当中
- 通常僵尸进程已经无法控管，而直接是交给 init 这个程序来负责，而 init 是系统第一个执行的程序，它是所有进程的父进程，我们无法杀掉它，因为杀掉他系统就死了，只能通过 reboot 来讲该进程抹去

5、<top>

- 动态查看进程的变化，相对于 ps 是选取一个时间点的进程状态，top 则可以持续检测进程运行的状态
- top [-d 数字] [-bnp]
- -d：后接描述，就是整个进程界面更新的描述，默认 5 秒
- -b：以批次的方式执行 top，通常搭配数据流重定向来讲批处理的结果输出成为文件
- -n：与-b 搭配，需要进行几次 top 输出的结果
- -p：指定某些 PID 来进行检测而已
- 在 top 执行过程中的按键：
  - ?：显示在 top 当中可以输入的按键命令
  - P：以 CPU 的使用资源排序显示
  - M：以内存的使用资源排序显示
  - N：以 PID 来排序
  - T：由该进程使用的 CPU 累积时间排序
  - k：基于某个 PID 一个信号
  - r：基于某个 PID 重新指定一个 nice 值
  - q：离开 top 软件

6、top 显示内容介绍

- **第一行**：

- 目前的时间
- 开机到目前为止所经过的时间
- 已经登录的人数
- 系统在 1，5，15 分钟的平均工作负载

➢ **第二行**：显示的是目前进程的总量，与各个状态下进程的数量
➢ **第三行**：显示的 CPU 整体负载，特别注意 wa，这个代表的是 I/Owait，通常系统变慢都是 I/O 产生的问题比较大
➢ **第四五行**：物理内存与虚拟内存的使用情况，注意 swap 的使用量越少越好，大量 swap 被使用说明系统物理内存不足
➢ **第六行**：top 进程中，输入命令时显示的地方
➢ **第七行以及以后**：每个进程的资源使用情况

- PID：每个进程的 ID
- USER：进程所属用户名称
- PR：Priority，优先顺序，越小优先级越高
- NI：Nice，与 Priority 有关，越小优先级越高
- %CPU：CPU 使用率
- %MEN：内存使用率
- TIME+：CPU 使用时间累加
- COMMAND
- top 默认使用 CPU 使用率作为排序的终点

7、<pstree>

➢ pstree [-A|U] [-up]
➢ -A：各进程树之间的连接以 ASCII 字符来连接**(连接符号是 ASCII 字符)**
➢ -U：各进程树之间的连接以 utf8 码的字符来连接，在某些终端接口下可能会有错误**(连接符号是 utf8 字符，比较圆滑好看)**
➢ -p：同时列出每个进程的 PID
➢ -u：同时列出每个进程所属账号名称

18. 3. 2. 进程的管理

1、程序是如何相互管理的？通过给予进程一个信号(signal)去告知该进程你想要它做什么

2、重要信号介绍

代号	名称	内容
1	SIGHUP	启动被终止的进程，可让该 PID 重新读取自己的配置文件，类似重新启动
2	SIGINT	相当于与用键盘输入[Ctrl]+c 来终止一个进程
9	SIGKILL	代表强制中断一个进程的进行，如果该进程进行到一半，那么尚未完成的部分可能会有"半产品"产生，例如 vim 会有.filename.swp 保留下来
15	SIGTERM	以正常的结束进程来终止该进程，由于是正常的终止，后续的操作会完成，当该进程已经发生问题，即无法正常终止时，该信号是无用的
17	SIGSTOP	相当于用键盘输入[Ctrl]+z 来暂停一个进程的进行

3、命令，详见 job control 的管理

- kill [-signal] PID

4、查找进程 PID 的语法糖：例如查找**执行命令名称为 syslog** 的进程

- ps aux | grep 'syslog' | grep -v 'grep' | awk '{print $2}'
- 首先用 grep 找出含有 syslog 的所有行
- 反选出没有 grep 的行，因为执行 grep 'syslog'的进程也会被查到
- 最后利用 awk '{print $2}'打印找到那行第二个字段
- **实际上还是很麻烦，killall 可以利用命令名称来执行操作，但操作对象是以这个命令启动的所有进程，因此得名 killall**

5、<killall>

- killall [-signal] [命令名称]
- killall [-iIe] [命令名称]
- killall [-iIe] [-signal] [命令名称]
- -i：interactive 的意思，交互式的，若要删除时，会出现提示符给用户
- -e：exact 的意思，表示后面接的 command 要一致，但整个完整的命令不能超过 15 个字符
- -I(大写的 i)：命令名称(可能含参数)忽略大小写
- **killall 删除服务很容易，他可以将系统中所有已某个命令名称启动的进程全部删除**

18. 3. 3. 关于进程的执行顺序

1、Priority 与 Nice 值

- 优先级较高的则在单位时间内运行次数较多，而不需要与较低优先级的进程抢位置
- **PRI 值是由系统动态调整的，用户不能设定**
- PRI 与 PI 的相关性：PRI(new)=PRI(old)+nice
- nice 是可以为负的，用以降低 PRI 的值，即提高优先级
- nice 的范围在-20~19
- 一般用户仅可调整自己进程的 Nice 值，且范围是 0-19(避免一般用户抢占系统资源)
- 一般用户仅可将 nice 调高，即本来为 5，那么调整后只能大于 5
- 调整进程的 nice 值：
  - 一开始执行程序就立即给予一个特定的 nice 值：用 nice 命令
  - 调整某个已经存在的 PID 的 nice 值：renice 命令

2、<nice>

- 新执行的命令即给予新的 nice 值
- nice [-n 数字] [命令]
- -n：后接数字，-20-19
- nice -n 5 tar -jcv -f /tmp/test.tar.bz2 /etc

3、<renice>

- 已存在的进程 nice 值重新调整
- renice [number] PID
- renice 10 9000

18. 3. 4. 系统资源的查看

1、<free>

- free [-b|-k|-m|-g] [-t]
- 直接输入 free 时，显示的是 KB
- -b：bytes
- -m：MB
- -k：KB
- -g：GB
- 在输出的最终结果中显示物理内存与 swap 的总量
- 显示参数介绍：
  - Men 那一行：物理内存
  - Swap：虚拟内存
  - total：总量
  - user：使用量
  - free：剩余可用量
  - shared 与 buffers/cached：被使用的量当中用来作为缓冲以及快取的量
  - buffers：缓冲记忆
  - cached：缓存
  - 一般来说系统会很有效地将所有内存用光，目的是为了让系统的访问性能加速，这一点与 Windows 很不同，因此对于 Linux 系统来说，内存越大越好

2、<uname>

- 查看系统与内核相关信息
- uname [-asrmpi]
- -a：所有系统相关的信息，包括下面的数据都会被列出来
- -s：系统内核名称
- -r：内核的版本
- -m：本系统的硬件名称，例如 i686 或 x86_64
- -p：CPU 的类型，与-m 类似，只是显示 CPU 类型
- -i：硬件的平台，例如 ix86 等

3、<uptime>

- 查看系统启动时间与工作负载
- uptime
- 显示出系统目前已经开机多久的时间，以及 1，5，15 分钟的平均负载(类似 top)，up 可以显示 top 界面最上面的一行

4、<netstat>：

- 跟踪网络，常被用在网络的监控
- netstat -[atunlp]
- -a：将目前系统上所有的连接，监听、Socket 数据都列出来
- -t：列出 tcp 网络数据包的数据
- -u：列出 udp 网络数据包的数据
- -n：不列出进程的服务名称，以端口号(port number)来显示

- -l：列出目前正在网络监听(listen)的服务
- -p：列出该网络服务的进程 PID
- 参数意义：[未摘录]

5、<dmest>

- 分析内核产生的信息
  - 系统在开机的时候，内核会去检测系统的硬件，你的某些硬件到底有没有被识别出来，就与这个时候的检测有关，demsg 能把这些大量的内核信息显示出来
  - 所有的内核检测的信息，无论是开机的时候还是系统运行的时候，反正只要是内核产生的信息都会被记录到内存中的某个保护区段
  - 可以搭配管道命令查看，因为信息太多了
- dmesg | less
- dmesg | grep -i sd <==查看硬盘
- dmesg | grep -i eth <==查看网卡

6、<vmstat>

- 检测系统资源变化
- vmstat 可以检测 CPU/内存/磁盘输入输出状态
- vmstat [-a] [延迟 [总计检测次数] <==CPU/内存相关
  - **如果仅跟延迟，不指明总检测次数，如"vmstat 5"，则检测会一直进行，直到按下[Ctrl]+C**
- vmstat [-fs] <==内存相关
- vmstat [-s 单位] <==设置显示数据的单位
- vmstat [-d] <==与磁盘有关
- vmstat [-p 分区] <==与磁盘有关
- -a：使用 inactive/active 代替 buffer/cache 的内存输出信息
- -f：开机到目前为止系统复制(fork)的进程数
- -s：将一些事件(开机到目前为止)导致的内存变换情况列表说明
- -S：后面可接单位，让显示的数据有单位。例如 K/M 取代 bytes 的容量
- -d：列出磁盘的读写总量统计表
- -p：后面列出分区，可显示该分区的读写总量统计表
- 显示各个字段的意义：
  - 内存字段(procs)：
    - r：等待运行中的进程数量
    - b：不可被唤醒的进程数量
    - **这两个选项越多，代表系统越忙碌**
  - 内存字段(memory)：
    - swpd：虚拟内存被使用的量
    - free：未被使用的内存容量
    - buff：用于缓冲存储器的用量
    - cache：用于高速缓存的用量
    - **这部分与 free 是一样的**
  - 内存交换空间(swap)：
    - si：由磁盘中将程序取出的量

- so：由于内存不足而降没用到的程序写入到磁盘的 swap 的容量
- **如果 si/so 的数值太大，表示内存内的数据经常在磁盘与内存之间传来传去，系统性能会很差**

- 磁盘读写(io)：
  - bi：由磁盘写入的块数量
  - bo：写入到磁盘去的数量
  - **这部分数值越高，代表系统的 I/O 非常忙碌**
- 系统(system)：
  - in：每秒钟被中断的进程次数
  - cs：每秒钟进行的时间切换次数
  - 这两个数值越大，代表系统与接口设备的通信非常频繁，接口设备包括磁盘、网卡、时钟等
- CPU
  - us：非内核层的 CPU 使用状态
  - sy：内核层所使用的 CPU 状态
  - id：闲置的状态
  - wa：等待 I/O 所耗费的 CPU 状态
  - st：被虚拟机所盗用的 CPU 使用状态

18.4. 特殊文件与程序

18.4.1. 具有 SUID/SGID 权限的命令执行状态

1、SUDI 的程序如何被一般用户执行，并具有什么特色

- SUID 权限仅针对二进制程序(binary program)有效
- 执行者对于该程序需要 x 的可执行权限
- 本权限在执行该程序的过程中有效(run-time)
- 执行者将具有该程序所有者(owner)的权限

2、整个 SUID 的权限会生效是由于具有该权限的程序被触发，**而一个程序被触发会变成进程，执行者可以具有所有者权限就是在该程序编程进程的时候**

3、查找 SUID/SGID 的文件："find / -perm +6000"

18.4.2. /proc/*代表的意义

1、进程都是在内存当中的，内存当中的数据都是写入到/proc/*这个目录下的

2、主机上面各个进程的 PID 都是以目录的类型存在于/proc 当中，例如开机执行的第一个进程 init(现在好像是 systemd，这是 CentOS7 的巨大变动)，它的 PID 是 1，这个 PID 所相关的信息都写入在/porc/1/*当中

3、针对整个 Linux 系统相关的参数

文件名	文件内容
/proc/cmdline	加载 kernel 时所执行的相关参数
/proc/cpuinfo	本机的 CPU 的相关信息，包含频率，类型与运算功能
/proc/devices	这个文件记录了系统各个主要设备的主要设备代号，与 mknod 有关
/proc/filesystems	目前系统已经加载的文件系统

/proc/interrupts	目前系统上面的 IRQ 分配状态
/proc/ioports	目前系统上面各个设备所配置的 I/O 地址
/proc/kcore	这个就是内存大小
/proc/loadavg	top，uptime 的三个平均数值
/proc/meminfo	使用 free 列出的内存信息，在这里能够查阅到
/proc/modules	目前 Linux 已经加载的模块列表，也可以想成是驱动程序
/proc/mounts	系统已经挂载的数据，就是用 mount 这个命令调出来的数据
/proc/swaps	系统加载的内存，使用的分区记录在此
/proc/partitions	使用 fidsk -l 会出现所有的分区，在这个文件夹也有记录
/proc/pic	在 PIC 总线上面每个设备的详细情况，可用 lspci 来查阅
/proc/uptime	就是调用 uptime 的时候会出现的信息
/proc/version	内核的版本，就是用 uname -a 显示的内容
/proc/bus/*	一些总线设备，还有 USB 的设备也记录在此

18. 4. 3. 查询已打开文件或已执行程序打开的文件

1、<fuser>

- 通过文件(或文件系统)找出正在使用该文件的程序
- fuser [-umv] [-k [i] [-signal]] file/dir
- -u：除了进程的 PID 外，同时列出该进程的所有者
- -m：后面接的那个文件名会主动提到该文件系统的顶层，对 umount 不成功很有效
- -v：可以列出每个文件与程序还有命令的完整相关性
- -k：找出使用该文件/目录的 PID，并试图以 SIGKILL 这个信号给予该 PID
- -signal 例如-1 -15 等，若不加默认-9
- -v 参数列出的内容介绍(ACCESS)
  - c：此进程在当前的目录下(非子目录)
  - e：可被触发为执行状态
  - f：是一个被打开的文件
  - r：代表顶层目录(root directory)
  - F：该文件被打开了，不过在等待中
  - m：可能为分享的动态函数库
- 例子：
  - fuser -mvu /proc <==查看某文件系统下面有多少进程正在占用该文件系统时，-m 参数很有用
  - fuser -uv /proc <==没有进程会使用/proc 这个目录，而是/proc 下面的文件，因此要加上-m 参数

2、<lsof>

- 列出被进程所打开的文件名
- lsof [-aU] [-u 用户名] [+d]
- -a：多项数据需要"同时成立"才显示结果
- -U：仅列出 Unix like 系统的 socket 文件类型

- -u：后面接 username，列出该用户相关进程所打开的文件
- +d：后面接目录，及找出某个目录下面已经被打开的文件

3、<pidof>

- 找出某个正在执行的进程的 PID
- pidof [-sx] program_name
- -s：仅列出一个 PID 而不列出所有 PID
- -x：同时列出该 program name 可能的 PPID 那个进程的 PID

18.5. SELinux 初探

18.5.1. 什么是 SELinux

1、SELinux 是 Security Enhanced Linux 的缩写，字面上的意思就是安全强化 Linux

2、当初设计的目标：避免资源的误用

- 很多企业界发现，通常系统出现问题的原因大部分在于"内部员工的资源误用"所导致的，实际由外部发动的攻击反而没有那么严重
- SELinux 是整合到内核的一个模块
- 其实 SELinux 是在进行程序、文件等权限设置依据的一个内核模块，由于启动网络服务也是程序，因此刚好也是能够控制网络服务能否访问系统资源的一道关卡

3、传统的文件权限与账号关系：自主访问控制,DAC

- 自主访问控制(Discretionary Access Control,DAC)，基本上，就是依据进程的所有者与文件资源的 rwx 权限来决定有无访问的能力
- root 具有最高的权限：如果某个程序被黑客所得，且该进程属于 root 的权限，那么这个程序就可以在系统上进行任何资源的访问
- 用户可以取得进程来更改文件资源的访问权限

4、以策略规划制定特定程序读取特定文件：委托访问控制,MAC

- SELinux 导入了强制访问控制(Mandatroy Access Contrl,MAC)
- MAC 可以针对特定的进程与特定的文件资源来进行权限的控制，也就是说，即使你是 root，那么在使用不同的进程的时候，你所能取得的权限也未必是 root，而得要看当时该进程的设置而定
- 如此一来，我们针对控制的"主体"变成了"进程"而不是用户，此外主体进程也不能任意使用系统文件资源，因为每个文件资源也有针对该主体进程设置可取用的权限。如此一来控制的项目就多了，所以 SELinux 提供了默认的策略，并在该策略内提供多个规则，让你选择是否启用该控制规则

18.5.2. SELinux 的运行模式

1、SELinux 是通过 MAC 的方式来控管进程，它控制的主体是进程，而目标则是该进程能否读取"文件资源"

2、主体：

- 可以将主体与进程画上等号

3、目标：

- 主体能否访问的"目标资源"一般就是文件系统
- 因此这个目标项目可以与文件系统画上等号

4、策略：

- 由于进程与文件数量庞大，因此 SELinux 会依据某些服务来指定基本的访问安全性策略，这些策略内还会有详细的规则来指定不同的服务开放某些资源的访问与否
- targeted：针对网络服务限制较多，针对本机限制较少，是默认的策略
- strict：完整的 SELinux 限制，限制方面比较严格

5、安全上下文：

- 主体与目标的安全上下文必须一致才能顺利访问
- 安全上下文类似于文件系统的 rwx
- 安全上下文存在于主体进程中与目标文件资源中
  - **进程：进程在内存中，所以安全上下文可以放入内存中**
  - **文件：安全上下文放置到文件的 inode 内**
- **安全上下文可用"ls -Z"查看**
  - ... Identify：role：type...
  - 身份标识(Identify)
    - root：标识 root 的账号身份
    - system_u：标识系统程序方面的标识，通常就是进程
    - user_u：代表的是一般用户账号相关的身份
  - 角色(Role)
    - 通过角色，我们可以知道这个数据是属于程序、文件资源还是用户
    - object_r：代表的是文件或目录等文件资源，最常见
    - system_r：代表进程，一般用户也会被指定成 system_r
    - 以 r 结尾，rule 的意思
  - 类型(Type,最重要)
    - 在默认的 targeted 策略中，Identify 与 Role 基本是不重要的，基本上一个主体进程能不能读取到这个文件资源与类型字段有关，而类型字段在文件与进程的定义不太相同
    - type：在文件资源上面称为类型
    - domain：在主体程序中则称为域
    - domain 需要与 type 搭配，则该程序才能够顺利读取文件资源

6、进程与文件 SELinux type 字段的相关性

身份识别	角色	在 targeted 下的意义
root	system_r	代表供 root 账号登陆时所取得的权限
system_u	system_r	由于为系统账号，因此是非交互式的系统运行程序
user_u	system_r	一般可登陆用户的进程

7、例子：

- ll -Zd /usr/sbin/httpd /var/www/html
- .....system_u:object_r:**httpd_exec_t**.....
- .....system_u:object_r:**httpd_sys_content_t**.....

- 首先触发一个可执行文件，那就是具有 httpd_exec_t 这个类型的/usr/sbin/httpd 这个文件
- 该文件的类型会让这个文件所造成的主体进程具有 httpd 这个域，我们的策略针对这个域已经制定了许多规则，包括这个域可以读取的目标资源类型
- 由于 httpd domain 被设置为可读取 http_sys_content_t 这个类型的文件(Object)，因此你的网页放置到/var/www/html/目录下，就能够被 httpd 那个进程读取了
- 但最终能不能读到正确的数据，还得要看 rwx 是否符合 Linux 权限的规范

18.5.3. SELinux 的启动、关闭与查看

1、SELinux 三种模式
- enforcing：强制模式，代表 SELinux 正在运行中，并且已经开始限制 domain/type 了
- permissive：宽容模式，代表 SELinux 正在运行中，不过仅会有警告信息并不会实际限制 domain/type 的访问
- disabled：关闭，SELinux 并没有实际运行

2、<getenforce>
- 显示出目前的模式

3、<sestatus>
- 查看 SELinux 的策略
- -v：检查列于/etc/sestatus.conf 内的文件与程序的安全上下文内容
- -b：将目前策略的规则布尔值列出，即某些规则(rule)是否要启动(1/0)之意

4、SELinux 的配置文件:/etc/selinux/config

5、SELinux 的启动与关闭
- 改变了策略需要重新启动
- 从 enforcing 或 permissive 模式改成 disabled 模式，或者由 disabled 模式改成 enforcing 或 permissive 模式，也需要重新启动
- 由于 SELinux 是整合到内核中去的，只可以在 SELinux 运行下切换成为强制(enforcing)或许可(permissive)模式，不能够直接关闭 SELinux

6、若你处于 Enforcing 模式，但是由于一些设置的问题导致 SELinux 让某些服务无法正常运行，你可以将 Enforcing 的模式改为许可(permissive)的模式，让 SELinux 只会警告无法顺利连接的信息，而不是直接抵挡主体进程的读取权限
- 转换命令：<setenforce>
  - setenforce [0|1]
  - 0：转换成 permissive 宽容模式
  - 1：转换成 Enforcing 强制模式

18.5.4. SELinux 网络服务运行范例

1、步骤
- systemctl start httpd.service (service httpd start) <==启动 httpd 服务
- pstree -upU | grep httpd <==查看相关的进程

- ps aux -Z | grep httpd <==-Z 可以让我们查阅安全上下文
- echo "This is my first web page." > /var/www/html/index.html
- 打开浏览器输入"http://127.0.0.1"
- 此时你的浏览器会通过 httpd 这个进程拥有的 httpd_t 这个 domain 去读取 /var/www/html/index.html 这个文件
- ll -Z /var/www/html/index.html <== ...httpd_sys_content_t，这个类型是 httpd_t 可读取的，这是 targeted 策略里面设置的
- cd <==回到 root 的主文件夹
- echo "My 2nd web page..." > index.html
- rm /var/www/html/index.html
- mv index.html /var/www/html
- 此时打开浏览器输入"http://127.0.0.1"，会得到错误信息
- ll -Z /var/www/html/index.html <==...admin_home_t，这个类型无法被 httpd_t 访问，因此被拒绝了
- **chcon -t httpd_sys_content_t /var/www/html/index.html** <==修改上下文类型
- chcon --reference=/etc/passwd /var/www/html/index.html <==参照/etc/passwd 的安全上下文类型修改目标文件的类型

2、<chcon>

- **通过直接指定的方式来处理安全上下文的类型数据**
- chcon [-R] [-t type] [-u user] [-r role] 文件
- chcon [-R] --reverence=范例文件 文件
- -R：连同该目录下的子目录也同时被修改，递归
- -t：后面接安装上下文的类型字段，例如 httpd_sys_content_t
- -u：后面接身份标识符，例如 system_u
- -r：后面接角色，例如 system_r

3、<restorecon>

- **默认的安全上下文，非常好用**
- restorecon [-Rv] [文件或目录]
- -R：连同子目录一起修改
- -v：将过程显示到屏幕上

18.5.5. SELinux 所需的服务

1、由于 SELinux 是整合到内核的一个内核功能，因此几乎不需要启动什么额外的服务来打开 SELinux 的，开机完成后，SELinux 就启动了。**但需要某些服务记录错误信息，几乎所有 SELinux 相关的程序都会以 se 开头，服务也是如此**

2、setroubleshoot：将错误信息写入/var/log/messages

3、auditd：将详细的数据写入/var/log/audit/audit.log

18.5.6. SELinux 的策略与规则管理

1、策略查阅

- <seinfo> ???没有这个命令
  - seinfo [-Atrub]

- -A：列出 SELinux 的状态、规则布尔值，身份识别，角色，类型等所有信息
- -t：列出 SELinux 的所有类型(type)种类
- -r：列出 SELinux 的所有角色(role)种类
- -u：列出 SELinux 的所有身份标识(user)种类
- -b：列出所有规则的种类(布尔值)

➢ <sesearch>???没有这个命令
- [未摘录]

➢ <getsebool>
- 布尔值的查询
- getsebool [-a] [布尔值条款]
- -a：列出目前系统上面所有布尔值条款设置为开启或关闭值

➢ <setsebool>
- 布尔值的修改
- setsebool [-P] 布尔值=[0|1]
- -P：将设置写入配置文件，该设置数据将会生效
- setsebool -P httpd_enable_homedirs=0

➢ <semanage>
- **默认目录的安全上下文的查询与修改，这就是 restorecon 修改的依据**
- semanage [login|user|port|interface|**fcontext**|translation] -l
- semanage fcontext -{a|d|m} [-frst] file_spec
- -l：查询的意思
- fcontext 主要用在安全上下文方面的用途，也是唯一可能会用的参数
- -a：增加
- -m：修改
- -d：删除

Chapter 19. 认识系统服务

19.1. 什么是 daemon 与服务(service)

1、常驻内存中的进程，且可以提供一些系统或网络功能，那就是服务

2、系统为了某些功能必须要提供一些服务(不论是系统本身还是网络方面)，这个服务就称为 service。但是 service 的提供总是需要进程的运行，所以实现这个 service 的**程序就称为 daemon**，举例来说，实现循环型例行性工作调度服务的程序为 cornd 这个 daemon

3、不必区分 daemon 与 service，可以将这两者视为相同，因为达成某个服务是需要一个 daemon 在后台运行，没有这个 daemon 就不会有 service

19.1.1. daemon 的主要分类

1、依据 daemon 的启动与管理方式来区分，可以将 daemon 分为可独立启动的 **stand_alone** 与通过一个 **super daemon 来统一管理**的服务这两大类

2、stand_alone：此 daemon 可以自行单独启动服务

- 这种类型的 daemon 可以自行启动而不必通过其他机制的管理
- daemon 启动并加载到内存后就一直占用内存与系统资源
- 最大的优点：**因为一直在内存内持续提供服务，因此对于发生客户端的请求时，stand_alone 的 daemon 响应速度较快**
- 常见的有：WWW 的 daemon(httpd)，FTP 的 daemon(vsftpd)
- **(CentOS7 以前)可以用/etc/init.d/[daemon**
- **]来启动(启动方式)**，例如
  - /etc/init.d/xinetd restart
  - /etc/init.d/httpd start

3、super daemon：一个特殊的 daemon 来统一管理

- 这种类型的 daemon 启动方式则是通过一个统一的 daemon 来负责唤起服务，这个特殊的 daemon 就被称为 super daemon，**该 super daemon 本身是一个 stand alone**
- **早期的 super daemon 是 inetd，后来被 xinetd 取代**
- 当没有客户端的请求时，各项服务都是未启动的情况，等到有来自客户端的请求时，super daemon 才唤醒相应的服务，当客户端请求结束后，被唤醒的服务会关闭并释放系统资源
- 这种机制的好处：
  - 由于 super daemon 负责唤醒各项服务，因此 suuper daemon 可以具有安全控管的机制，就是类似网络防火墙的功能
  - 由于服务在客户端的连接结束后就关闭，因此不会一直占用系统资源
- 缺点：相应客户端的速度较慢
- **不能用/etc/init.d/[daemon**
- **]来启动**

4、窗口类型的解说：

- 这两种方式如何选择，取决于主机的工作负荷以及实际的用途
- 例如主机用来作为 WWW 服务器的，那么 httpd 自然就以 stand alone 的启动方式较佳
- 个别窗口负责单一服务的 stand alone

- 统一窗口负责各种业务的 super daemon
- 多线程(multi-threaded)(???为什么叫线程)：一个服务会负责好几个进程，对于每个登陆者，会有一个单独的 daemon 来进行服务
- 单线程(single-threaded) (???为什么叫线程)：一个服务负责一个进程，对于所有登陆者，共享同一个 daemon 的服务

5、daemon 工作形态的类型

- signal-control：通过信号来管理，只要有任何客户端的请求进来，它就会立即启动去处理，例如打印机服务
- interval-control：每隔一段时间就主动去执行某项工作，所以你要做的就是在配置文件中指定服务要进行的时间与工作，atd 与 crond 就属于这一类型的 daemon(每分钟检测一次配置文件)

6、daemon 的命名规则

- 通常在服务的名称之后会加上一个 d，例如 at 与 cron 这两个服务，它的程序文件名会被取为 atd 与 crond，d 代表 daemon 的意思

19.1.2. 服务与端口的对应

1、系统所有的功能都是某些程序所提供的，而进程是通过程序触发产生的

2、IP 就代表你的主机在因特网上面的门牌号码，端口号(port)可以想象成你家门牌上面的第几层楼。IP 与 port 就是因特网连接的最重要机制之一

3、举例：

- http://ftp.isu.edu.cn/
- ftp://ftp.isu.edu.cn/
- 这两个网址都是指向 ftp.isu.edu.cn 这个 FTP 网站，但是浏览器的结果却是不一样的，因为我们指向了不同的服务，一个是 http 这个 WWW 服务，一个是 ftp 这个文件传输服务

4、**为了统一整个因特网的端口号对应服务的功能，好让所有主机都使用相同的机制来提供服务于请求服务，所有就有了"协议"，也就是说，有些约定俗成的服务都放在同一个端口号上面**

- 例如网址栏上面的 http 会让浏览器向 WWW 服务器的 80 端口号进行连接的请求，而 WWW 服务器也会将 httpd 这个软件激活在 port80，两者才能够达成成功连接

5、系统上面让服务与端口号对应在一起的设置：/etc/services

- 第一字段：daemon 名称
- 第二字段：该 daemon 所使用的**端口号**与**网络数据包协议**，数据包协议主要为可靠连接的 TCP 数据包以及较快速但为非面向连接的 UDP 数据包
- 第三字段：该服务的说明
- 不建议修改该文件，除非你要假设一个地下网站，否则使用原先的设置即可

19.1.3. daemon 的启动脚本与启动方式

1、提供某个服务的 daemon 虽然只是一个程序(进程)而已，但是这个 daemon 的启动还是需要执行文件、配置文件、执行环境等

2、通常 distribution 都会记录每一个 daemon 启动后所取得进程的 PID 并放

在/var/run/这个目录下

3、在启动服务之前，可能要自行处理一下 daemon 能够顺利执行的环境是否正确，即要启动一个 daemon 考虑的事情很多，并非单纯执行一个进程就够了。为了解决该问题，**distribution 通常给我们一个简单的 shell script 来进行启动的功能，该 script 可以进行环境的检测、配置文件的分析、PID 文件的放置，以及相关重要交换文件的锁住(lock)等操作，你只要执行该 script，上述的操作就会执行，最终就能顺利且简单地启动这个 daemon**

4、配置文件

- /etc/init.d/*：启动脚本放置处(在 CentOS7 后已经改变了)
- /etc/sysconfig/*：各服务的初始化环境配置文件
- /etc/xinetd.conf,/etc/xinetd.d/*：super daemon 配置文件
- /etc/*：各服务各自的配置文件
- /var/lib/*：各服务产生的数据库
- /var/run/*：各服务程序的 PID 记录处
- 为了管理时不影响到其他进程，因此 daemon 通常会将自己的 PID 记录一份到/var/run 当中，例如日志文件的 PID 记录就在/var/run/syslog.pid 这个文件中

5、用法

- **从/etc/init.d/*启动服务的用法已经在 CentOS7 中取消，因此不做介绍**
- **CentOS7 为了向下兼容了，保留了 service 的用法，但会重新定向到新的 systemctl 工具**
- <service>：不推荐用，尽量用心的工具 systemctl
- service [service name] [start|stop|status|restart...]
- service --status-all <==列出系统所有 stand alone 服务状态

6、super daemon 的启动方式：**xinetd(超级进程)**

- super daemon 本身是一个 stand alone，其启动方式与其他 stand alone 是一样的
- super daemon 所管理的其他 daemon 启动就不同了，必须哟啊在配置文件中设置为启动该 daemon 才行，配置文件就是/etc/xinetd.d/*
  - 本来是没有该程序的，需要先安装：yum install xinetd
- 配置文件中的"disable=yes"代表取消此项服务的启动
- 也就是说先修改/etc/xinetd.d/下面的配置文件，然后再重新启动 xinetd 即可

19.2. 解析 super daemon 的配置文件

1、super daemon 是一个总管进程，这个 super daemon 是 xinetd 这一个进程所实现的，xinetd 可以进行安全性或者是其他管理机制的控制

2、一些对客户端开放较多权限的服务(如 telnet)或者本身不具有管理机制或防火墙机制的服务就可以通过 xinetd 来管理

3、xinetd 的配置文件：/etc/xinetd.conf

19.2.1. 默认配置文件：xinetd.conf

1、文件内容介绍

```
defaults{
        log_type=SYSLOG daemon info  <==日志文件的记录服务类型
        log_on_failure=HOST    <==发生错误时需要记录的信息为主机
        log_on_success=PID HOST DURATION EXIT <==成功启动或登录时记录的
        信息

        cps=50 10  <==同一秒内最大连接数为 50，若超过则暂停 10 秒
        instances=50  <==同一服务的最大同时连接数
        per_source=10  <==同一来源的客户端的最大连接数

        v6only=no  <==是否仅允许 IPV6

        groups=yes
        umask=002
}
includedir /etc/xinetd.d  <==更多的设置值在/etc/xinetd.d 那个目录内
```

2、为什么/etc/xinetd.conf 称为默认配置文件，因为如果启动某个 super daemon 管理的服务，但是该服务的设置并没有指定上述这些项目，那么该服务的设置值就以上述的默认值为主

- **即，一个服务最多可以有 50 个同时连接，但每秒钟发起的"新"连接最多仅能有 50 条，若超过 50 条则该服务会暂停 10 秒，同一个来源的用户最多仅能达成 10 条连接，登陆的成功与失败所记录的信息并不相同**

3、更多的服务参数文件都在/etc/xinetd.d 里面，文件内容形如：

```
service <service_name>
{
        <attribute>  <assign_op>  <value>  <value> ...
        ...
}
```

- 第一行都有一个 service
- attribute 是一些 xinetd 的管理参数
- assign_op 则是参数的设置方法
- assign_op 的主要设置形式为：
  - =：表示后面设置参数就是这样
  - +=：表示后面的设置为在原来的设置里面加入新的参数
  - -=：表示后面的设置为在原来的参数中设置这里输入的参数

4、attribute 表，详见 P559

19.2.2. 一个简单的 rsync 范例设置

1、(CentOS7 上安装的 rsync 好像不由 xinetd 管理，而且在/etc/xinetd.d/目录中加入了 rsync 配置文件后，重启 xinetd 也不会使得 rsync 服务由 xinetd 管理)

19.3. 服务的防火墙管理 xinetd,TCP Wrappers

1、一般来说，系统的防火墙分析主要可以通过数据包过滤或者通过软件分析，Linux 默认提供一个软件分析的工具，即/etc/hosts.deny 和/etc/hosts.allow 这两个配置文件

19.3.1. /etc/hosts.allow,/etc/hosts.deny 管理

1、at 的使用可以通过修改/etc/at.{allow|deny}来管理，crontab 使用/etc/cron.{allow|deny}来管理

2、/etc/hosts.{allow|deny}管理某些进程是否能够接受或者拒绝来自因特网的连接

3、任何以 xinetd 管理的服务都可以通过/etc/hosts.allow,/etc/hosts.deny 来设置防火墙

4、防火墙，简单地说就是针对源 IP 或域进行允许或拒绝的设置，以决定改连接是否能够成功实现连接，/etc/xinetd.d/目录下的配置文件中的 no_access,only_from 也可以进行这方面的防火墙设置，不过使用/etc/hosts.allow,/etc/hosts.deny 则更容易集中管理

5、/etc/hosts.{allow|deny}也是/usr/sbin/tcpd 的配置文件，这个/usr/sbin/tcpd 则是用来分析进入系统的 TCP 网络数据包的一个软件

- TCP 是一种相面连接的网络连接数据包，包括 www,email,ftp 等都是使用 TCP 数据包来实现连接的
- TCP 数据包的头文件主要记录了来源于目的主机的 IP 与 port，因此通过分析 TCP 数据包并搭配/etc/hosts.{allow,deny}的规则比较，就可以决定该连接是否能够进入我们的主机

6、TCP Wrapper 控管的是

- 源 IP 或/与整个域的 IP 网段
- port

7、**基本上只要一个服务受到 xinetd 管理，或者该服务的程序支持 TCP Wrappers 函数的功能时，那么该服务的防火墙的设置就能以/etc/hosts.{allow,deny}来处理**

8、查询一个服务的程序是否支持 TCP Wrappers

- ldd $(which sshd httpd)
- **ldd：查询某个程序的动态函数库支持状态**
- 当出现 tcp wrappers 所提供的 libwraps.so 这个函数库文件，就可以说明该服务支持 TCP Wrappers

9、配置文件语法

- 这两个配置文件的语法都是一样的
- <service(program_name)> : <IP, domain, hostname> : <action>
- 两个重点：
  - 一是找出想要管理的程序的文件名
  - 二是写下你想要放行或者是抵挡的 IP 或域
  - rsync : 127.0.0.100 127.0.0.200 : deny
  - 或
  - rsync : 127.0.0.100 : deny

- rsync : 127.0.0.200 : deny

- hosts.allow 中的 IP 与网段默认可通行，第三列的 allow 可省略
- hosts.deny 中的 IP 与网段默认不可通行，第三列的 deny 可省略
- **以 /etc/hosts.allow 为优先，若分析到 IP 或网段并没有记录在 /etc/hosts.allow，则以 /etc/hosts.deny 来判断 (/etc/at.{allow|deny}和/etc/cron.{allow|deny}只有一个会生效)**
- 惯例：
  - 允许进入的卸载/etc/hosts.allow 当中
  - 不允许进入的写在/etc/hosts.deny 中

19.3.2. TCP Wrappers 特殊功能

1、当其他人扫描我的 rsync port 时，TCP Wrappers 就将它的 IP 记住，以作为将来的查询与认证之用

2、<spawn>

3、<twist>

19.4. 系统开启的服务

19.4.1. 查看系统启动的服务

1、查看目前系统开启的网络服务

- netstat -tulp

2、找出所有的监听网络的服务

- netstat -lnp

19.4.2. 设置开机后立即启动服务的方法

1、systemctl，详见 System

2、**<chkconfig>已经不推荐使用，CentOS7 中对它的调用会将请求转发到 systemctl**

- 我们在启动 Linux 系统时，可以进入不同的模式，这模式我们称为执行等级(run level)。不同的执行等级有不同的功能与服务
- chkconfig --list [服务名称]
- chkconfig [--level [0123456]] [服务名称] [on|off]
- --list：仅将目前的各项服务状态栏显示出来
- --level：设置某个服务在该 level 下启动或关闭
- chkconfig [--add|--del] [服务名称]
- --add：增加一个服务名称给 chkconfig 来管理，该服务名称必须在/etc/init.d/内
  - vim /etc/init.d/liuyed
  - #! /bin/bash
  - **# chkconfig: 35 80 70 <==这句是有用的，并非当做注释，冒号与 chkconfig 之间不要有空格**
  - # description: 范例
  - echo "Nothign"
- --del：删除一个给 chkconfig 管理的服务

3、<ntsysv>：类图形界面管理模式

- ntsysv [--level <levels>]
- --level：后面可接不同的 run level，例如 ntsysv --level 35
- 一般直接输入 ntsysv 就进入管理界面了
  - '*'代表开机默认会启动
  - 上下键：在中间的放扩中，在各个服务中移动
  - 空格键：用来选择你所需要的服务
  - tab 键：在方框，OK，Cancel 之间移动
  - F1 键：显示该服务的说明

4、Linux 主机如何开机：

- 打开计算机电源，开始读取 BIOS 并进行主机的自我测试
- 通过 BIOS 取得第一个可开机设备，读取主要开机区(MBR)取得启动装载程序
- 通过启动装载程序的设置，取得 kernel 并加载内存检测系统硬件
- 内核主动调用 init 进程**(CentOS7 之后改为调用 systemd 进程)以下都变?**
- init 进行开始执行系统初始化(/etc/rc.d/rc.sysinit)
- 根据 init 的设置进行 daemonstart(/etc/rc.d/rc[0-6].d/*
- 加载本机设置(/etc/rc.d/rc.local)

Chapter 20. 认识与分析日志文件

20.1. 什么是日志文件

1、详细而确实地分析以及备份系统的日志文件是一个系统管理员应该要进行的任务之一

2、日志文件：记录系统活动信息的几个文件，记录系统在什么时候由哪个进程做了什么样的行为时，发生了何种的事件

3、日志文件的重要性

- 解决系统方面的错误
- 解决网络服务的问题
  - 由于网络服务的各种问题通常都会被写入特别的日志文件，只需要查询日志文件就会知道出了什么差错
  - 例如无法启动邮件服务器(mailsend)，查询/var/log/maillog 通常可以找到不错的答案
- 过往时间记录簿(很重要)
  - 例如 WWW 服务(apache 软件)在某个时刻流量特别大，想要查找原因，可以通过日志文件去找出该时段是哪些 IP 在联机与查询的网页数据是什么

4、Linux 常见的日志文件名

- 日志文件可以帮助我们了解很多系统重要的事件，包括登陆者的部分信息，因此日志文件的权限通常设置为仅有 root 能够读取
- /var/log/cron：例行性工作调度
- /var/log/dmesg：记录系统在开机的时候内核检测过程所产生的各项信息
- /var/log/lastlog：记录系统上面所有账号最近一次登陆系统的相关信息
- /var/log/maillog 或/var/log/mail/*：记录邮件的往来信息，主要记录 sendmail(SMTP 协议提供者)与 dovecot(POP3 协议提供者)所产生的信息
  - SMTP：发信所使用的协议
  - POP3：收信所使用的协议
  - 这是两套不同的协议
- /var/log/messages：**相当重要**，几乎系统发生的错误信息(或重要信息)都会记录在这个文件中，如果系统发生莫名错误，这个文件一定要查阅
- /var/log/secure：只要牵涉到需要输入账号密码的软件，那么当登陆时(不管登陆正确或错误)都会被记录在此文件中
- /var/log/wtmp,/var/log/faillog：这两个文件记录正确登陆系统者的账户信息(wtmp)，与错误登陆系统者时所使用的账户信息(faillog)
- /var/log/httpd/*,/var/log/news/*,/var/log/samba/*：不同的网络服务会使用它们自己的日志文件案来记载它们自己产生的各项信息

5、日志文件所需相关服务(daemon)与进程

- 日志文件产生的方式
  - 由软件开发商自行定义写入的日志文件与相关格式，例如 WWW 软件 apache 就是这样处理的
  - 由 Linux distribution 提供的日志文件管理服务来统一管理，例如 CentOS 提供 syslogd 这个服务来统一管理日志文件，CentOS6 之后改成了 rsyslogd

- 除了 syslogd 之外，内核也需要额外的登录服务来记录内核产生的各项信息，专门记录内核信息的日志文件服务就是 klogd
- 因此日志文件所需的服务主要就是 syslogd 与 klogd
- 通过 logrotate(日志文件轮替)来自动化处理日志文件容量与更新问题
  - 所谓 logrotate，基本上就是将旧的日志文件更改名称，新建一个空的日志文件，重新开始记录，避免读取大文件效率不高的问题
  - 旧的日志文件保留一段时间(几个月)后系统将它自动砍掉，避免占用硬盘空间
- 总结，针对日志文件所需的功能，需要的服务与程序有：
  - syslogd：主要记录系统与网络等服务的嘻嘻
  - klogd：主要记录内核产生的各项信息
  - logrotate：主要进行日志文件的轮替功能

20.2. syslogd：记录日志文件的服务

20.2.1. 日志文件内容的一般格式

1、记录的几个重要数据

- 事件发生的日期与时间
- 发生此事件的主机名
- 启动此时间的服务名称(如 samba，xinetd 等)或函数名称

20.2.2. syslogd 的配置文件：/etc/syslog.conf

注意：Centos6 以后改为/etc/rsyslog.conf

1、就是 syslogd 这个 daemon 的配置文件

2、syslogd 可以负责主机产生的各个信息的记录，而这些信息本身是有"严重等级"之分的，而且这些数据最终要传送到哪个文件去是可以修改的(每个 Linux distributions 放置日志文件
可能会有所区别)

3、基本上，syslog 针对各种服务与信息记录在某些文件的配置文件就是/etc/rsyslog.conf，**这个文件规定了什么服务的什么等级以及需要被记录在哪里(设备或文件)**

4、设置语法

- # 服务名称[.=!]信息等级　　信息记录的文件名或设备或主机
  - **'.'：代表比后面还要高的等级(含该等级)，较常用**
  - '.='：代表所需要的等级就是后面接的等级
  - '.!'：代表不等于，即除了该等级外的其他等级都记录
- mail.info　/var/log/maillog_info
- # mail 服务产生大于等于 info 等级的信息，记录到/var/log/maillog_info 中

5、syslog 设置的主要服务表

服务类型	说明
auth(authpriv)	主要与认证有关的机制，例 login,ssh,su 等需要账号/密码
cron	理性工作调度 cron/at 等生成信息日志的地方
daemon	与各个 daemon 有关的信息
kern	就是内核产生信息的地方

lpr	打印相关的信息
mail	只要与邮件收发有关的信息
news	与新闻组服务器有关的东西
syslog	就是 syslogd 这个程序本身生成的信息
user,uucp,local0-7	与 UNIX like 机器本身有关的一些信息

- 以上都是 syslog 自行定制的服务名称，软件开发商可以通过调用上述的服务名称来记录他们的软件
- 例如 sendmail，postfix 与 dovecot 都是与邮件有关的软件，这些软件在设计日志文件记录时，都会主动调用 syslogd 内的 mail 服务名称(LOG_MAIL)

6、信息等级，分为 7 个等级

等级	等级名称	说明
1	info	仅是一些基本的信息说明
2	notice	除了 info 外还需要注意的一些信息内容
3	warning(warn)	警示的信息，可能有问题，但是还不至于影响到 daemon 的运行，**基本上 info，notice，warn 这三个信息都是在告知一些基本信息而已，还不至于造成一些系统运行的困扰**
4	err(error)	一些重大的错误信息，例如配置文件的某些设置值造成该服务无法启动的信息说明，通常通过 err 的错误告知
5	crit	比 error 还要严重的错误信息，这个 crit 是临界点(critical)的缩写
6	alert	警告，已经很有问题的等级，比 crit 还严重
7	emerg(panic)	意指系统已经几乎要死机的状态，通常大概只有硬件出现问题导致整个内核无法顺利运行才会出现该信息

- 除此之外，还有两个特殊的等级，debug(错误检测等级)与 none(不需要登陆等级)两个

7、信息记录的文件名或设备或主机

- 文件的绝对路径：通常就是放在/var/log 里的文件
- 打印机或其他设备：例如/dev/lp0 这个打印机设备
- 用户名称：显示给用户
- 远程主机：例如@www.vbird.tsai，当然要对方主机也能支持才行
- '*'：代表目前在线的所有人，类似 wall 这个命令的意义

20. 2. 3. 日志文件的安全性设置

1、**我们可以通过一个隐藏的属性来设置你的日志文件成为只可以增加数据但是不能被删除的状态**

2、**syslog 的日志文件只要被编辑过就无法继续记录，例如 vim 退出:wq**

3、将一个文件以 chattr 设置 i 这个属性，连 root 都不能删掉，而且也不能新增数据，设置成 a 这个属性可以增加而不能被删除

- chattr +a [文件名]

4、新手可以暂时不用这设置，仅使用默认值即可，a 参数会导致 logrotate 无法移动该日志的文件名，会造成很大的困扰(尽管可以通过 logrotate 的配置文件来解决)

20.2.4. 日志文件服务器的设置

1、场景：有 10 台 Linux 主机，每一台负责一个网络服务，你想了解每台主机的状态，需要经常登陆 10 台主机去查阅日志文件，**此时，我们可以让某一个主机成为日志文件服务器，用它来记录 10 台 linux 主机的信息**

2、CentOS 本身具有日志文件服务器的功能，只是默认没有启动该功能，Linux 主机会启动一个端口来监听，默认端口就是 UDP 的 514

3、可以通过修改 syslogd 的启动配置文件/etc/sysconfig/syslog**(CentOS6 以后改成了/etc/sysconfig/rsyslog)**

20.3. 日志文件的轮替（logrotate）

1、**syslog 利用的是 daemon 的方式来启动的，当有需求的时候立刻就会被执行，logrotate 是挂载 cron 下面的**

20.3.1. logrotate 的配置文件

1、logrotate 的配置文件：

- /etc/logrotate.conf：主要参数文件
- /etc/logrotate.d/ ：目录，该目录中所有的文件都会被读入/etc/logrotate.conf 中来进行

2、文件名的更替

- messages==>messages.1==>messages.2==>messages.3==>删除

3、/etc/logrotate.conf 内容介绍

- weekly：默认每个礼拜进行一次 rotate
- rotate 4：保留 4 个日志文件，即(无后缀、.1、.2、.3)
- create：由于旧日志文件被重名，于是新建一个新的日志文件来继续存储
- **include /etc/logrotate.d：将该目录下的文件读进来执行**

4、其实/etc/logrotate.d 就是由/etc/logrotate.conf 规划出来的目录，因此，我们其实可以将所有的数据都写入/etc/logrotate.conf 即可，但是这样一来，这个文件会变得很复杂，如果独立出来一个目录，每个以 RPM 打包方式所新建服务的日志文件轮替设置，就可以独自称为一个文件

5、一般来说，/etc/logrotate.conf 是默认的轮替状态而已，各个服务都可以拥有自己的日志文件轮替设置

6、配置/etc/logrotate.d/syslog/目录下的文件

- 例子:P587
- 文件名：被处理的日志文件绝对路径文件名，使用空格符分隔多个日志文件
- 参数：上述文件名进行轮替的参数用{}括起来
- 可调用外部命令来进行额外的命令执行，这个设置需要与 sharedscripts...endscript 设置一起使用才行，即实现日志文件的安全性设置需要的功能所做的配置

- prerotate：在启动 logrotate 之前进行的命令，例如修改日志文件的属性等操作
- postrotate：在做完 logrotate 之后进行的命令，例如重启(kill -HUP)某个服务

```
sharedscripts
prerotate
    /usr/bin/chattr -a /var/log/messages
endscript
sharedscripts
postrotate
    /bin/kill -HUP `cat /var/run/syslogd.pid 2> /dev/null` 2> /dev/null || true
    /bin/kill -HUP `cat /var/run/rsyslogd.pid 2> /dev/null` 2> /dev/null || true
    /usr/bin/chattr +a /var/log/message
endscript
```

20.3.2. 实际测试 logrotate 的操作

1、<logrotate>

- logrotate [-vf] [日志文件]
- -v：启动显示模式，会显示 logrotate 运行的过程
- -f：不论是否符合配置文件的数据，强制每个日志文件都进行 rotate 操作

Chapter 21. 启动流程、模块管理与 Loader

- 这部分内容出版日期在 2010 年之前，因此可能相对落后，阅读请务必带入该书发布的时间
- 这部分针对的是 sysvint，init 系统的发展具体详见系统设置工具(网络与打印机)与硬件检测

21.1. Linux 的启动流程分析

21.1.1. 启动流程一览

1、启动的过程中，引导装载程序(Boot Loader)使用的软件可能不一样

- 例如各大 Linux distributions 的主流为 grub
- 早期 Linux 默认使用 LILO

2、当你按下电源按键后计算机硬件会自动读取 BIOS 来加载硬件信息以及进行硬件系统的自我测试，之后系统会主动读取第一个可启动的设备(由 BIOS 设置)，此时就可以读入引导装载程序了

3、引导装载程序可以指定使用哪个内核文件来启动，并实际加载内和到内存当中解压缩与执行，此时内核就能够开始在内存内活动，并检测所有硬件信息与加载适当的驱动程序来使这部分主机开始运行，等到内核检测硬件与加载驱动程序完毕后，操作系统就开始在你的 PC 上面跑了

4、系统的启动过程如下：

- 加载 BIOS 的硬件信息与进行自我测试，并依据设置取得第一个可启动设备
- 读取并执行第一个启动设备内 MBR 的 boot Loader(即 brub,spfdisk 等程序)
- 依据 boot loader 的设置加载 Kernel，Kernel 会开始检测硬件与加载驱动程序
- 在硬件驱动成功后，Kernel 会主动调用 init 进程，而 init 会取得 run-level 信息
- init 执行/etc/rc.d/rc.sysinit 文件来准备软件执行操作环境(如网络、时区等)
- init 执行 run-level 的各个服务的启动(script 方式)
- init 执行/etc/rc.d/rc.local 文件
- init 执行终端模拟机程序 mingetty 来启动 login 进程，最后等待用户登录

21.1.2. BIOS, boot loader 与 Kernel 加载

1、BIOS，开机自我测试与 MBR

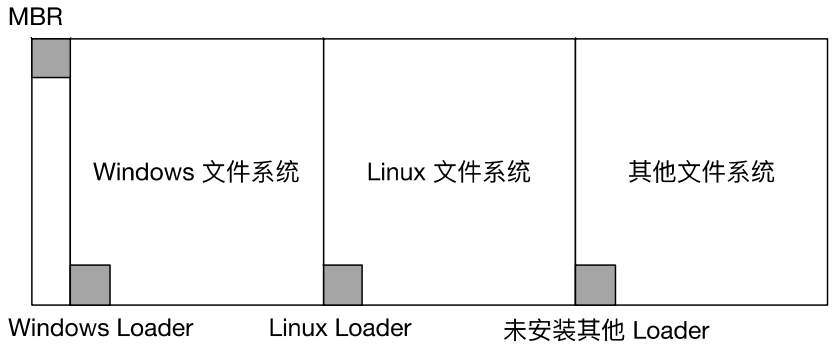
- 启动整个系统首先要让系统加载 BIOS(Basic Input Output System)，并通过 BIOS 程序加载 CMOS 信息，并通过 CMOS 内的设置值取得主机的各项硬件配置，例如 CPU 与接口设备的通信频率，启动设备的查找顺序，硬盘的大小与类型，系统时间，各周边总线是否启用 Plugand Play(PnP，即插即用设备)、各接口设备的 I/O 地址以及与 CPU 通信的 IRQ 中断等信息
- 在取得这些信息后，BIOS 还会进行开机自检(Power-on Self Test,POST)，然后开始执行硬件检测的初始化，并配置 PnP 设备，之后再定义出可启动的设备顺序，接下来就会开始进行启动设备的数据读取了
- 由于我们的系统软件大多放置在硬盘，所以 BIOS 会指定启动的设备好让我们可以读取磁盘中的操作系统内核文件，但由于不同的操作系统的文

件系统格式不同，因此我们必须要以一个引导装载程序来处理内核文件加载(load)的问题，因此这个程序就被称为 BootLoader，该程序一般装载启动设备的第一个扇区，即 MBR

- BIOS 如何读取 MBR 内的 loader？通过 INT 13 中断功能来读取 MBR，即只要 BIOS 能够检测到你的磁盘(无论是 SATA 接口还是 IDE 接口)，那就有办法通过 INT 13 这条信道来读取该磁盘的第一个扇区内的 MBR，这样 boot loader 就能被执行

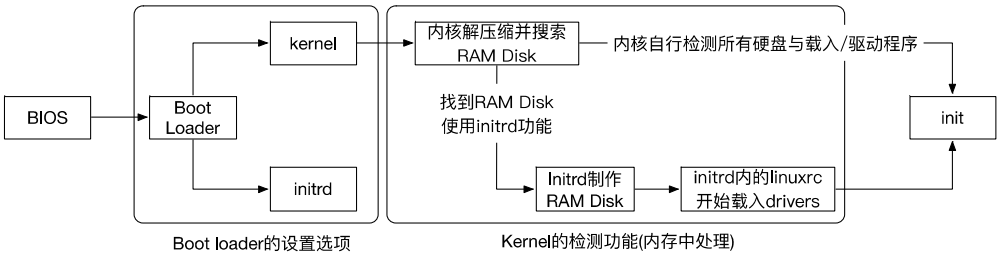
2、Boot Loader 的功能

- 必须要使用自己的 loader 才能加载属于自己的操作系统内核，而系统的 MBR 只有一个，如何在一部主机上面安装 Windows 与 Linux 呢
- 每个文件系统(filesystem 或 partition)都会保留一块引导扇区(boot sector)提供操作系统安装 boot loader，而操作系统默认都会安装一份 loader 到它的根目录所在的文件系统的 boot sector 上
- 每个操作系统默认会安装一套 boot loader 到它自己的文件系统中
  - 对于 Linux，你可以选择将 boot loader 安装到 MBR 去，也可以不选择
  - 对于 Windows，默认主动将 MBR 与 boot sector 都装上一份 boot loader



  -
- boot loader 主要功能：
  - 提供菜单：用户可以选择不同的启动选项，这也是多重引导的重要功能
  - 加载内核文件：直接指向可启动的程序区段来开始操作系统
  - 转交其他 loader：将引导装载功能交给其他 loader 负责
  -
  - 由于 Windows 的 loader 默认不具有控制权转交功能，因此你不能使用 Windows 的 loader 来加载 Linux 的 loader，因此要先安装 Windows 再安装 Linux(Linux 的 boot loder 会覆盖原有的 MBR 上的对应 Windows 的 boot loader)
  - 我们的 MBR 使用 Linux 的 grub 这个引导装载程序，并且里面假设有三个菜单，第一个菜单直接指向 Linux 的内核文件并直接加载内核文件来启动，第二个菜单可以将引导装载程序控制权交给 Windows 来管理，此时 Windows 会接管启动流程，第三个菜单使用 Linux 在 boot sector 内的引导装载程序，此时就会跳出另一个 grub 菜单
  - 最终 boot loader 的功能就是加载 kernel 文件

3、加载内核检测硬件与 initrd 功能

- 当通过 boot loader 的管理二开始读取内核文件后，Linux 就会将内核解压缩到内存当中，并且利用内核功能，开始测试与驱动各个周边设备，包括存储设备，CPU，网卡，声卡等。此时内核会以自己的功能重新检测一次硬件，而不一定会使用 BIOS 检测到的硬件信息，也就是说，内核此时接管 BIOS 后的工作
- 内和文件一般被放置到/boot 里面
- **ls --format=single-column -F /boot**
- 为了硬件开发商与其他内核功能开发者的便利，因此 Linux 内核是可以通过动态加载内核模块的(可以想象成驱动程序)，这些内核模块就放在/lib/modules 目录内
- **由于模块放置到磁盘根目录内(记得/lib 必须与/放在同一个分区)，因此在启动过程中，内核必须要挂载根目录，这样才能够读取内核模块提供加载驱动程序的功能**，而且为了担心影响到磁盘内的文件系统，启动过程中根目录是以只读的方式来挂载的
- **一般来说，非必要的功能可以编译成为模块的内核功能，目前的 Linux distributions 都会讲它编译成模块，因此 USB,SATA,SCSI 等磁盘设备的驱动程序通常都是以模块的方式来存在的**
- 问题是：内核根本不认识 SATA 磁盘，因此要加载 SATA 磁盘的驱动程序，否则根本无法挂载根目录，但是 SATA 的驱动程序在/lib/modules 内，你根本无法挂载根目录，又如何读取到/lib/modules/内的驱动程序，在这个情况下，Linux 是无法顺利启动的，可以通过虚拟文件系统来处理这个问题
- 虚拟文件系统(InitiaRAM Disk)一般使用文件名为/boot/initrd，这个文件的特色是，它能够通过 boot loader 加载到内存中，然后这个文件会被解压缩并且在内存当中仿真成一个根目录，且此仿真在内存当中的文件系统能够提供一个可执行的程序，通过改程序来加载启动过程中最需要的内核模块，**通常这些模块就是 USB,RAID,LVM,SCSI 等文件系统与磁盘接口的驱动程序**，等载入完毕后，会帮助内核重新调用/sbin/init 来开始后续正常启动流程
- 

21.1.3. 第一个进程 init 及配置文件/etc/inittab 与 runlevel

1、在内核加载完毕进行完硬件检测与驱动程序加载后，此时你的主机硬件应该已经准备就绪了，此时内核会主动调用第一个进程，那就是/sbin/init(init 的 PID 号码是 1)

2、/sbin/init 最主要的同能就是准备软件执行的环境，包括系统的主机名，网络设置，语系处理，文件系统格式以及其他服务的启动

3、run level：执行等级

- 0：halt(系统直接关机)
- 1：single user mode(单用户维护模式，用在系统出问题时的维护)
- 2：Multi-user, without NFS(类似下面的 runlevel3，但是无 NFS 服务)
- 3：Full multil-user mode(完整含有网络功能的纯文本模式)
- 4：unused(系统保留功能)
- 5：X11(与 runlevel3 类似，但是加载使用 X Windows)
- 6：reboot(重新启动)
- 由于 runlevel 0/4/6 不是关机重启就是系统保留，因此不能将默认的 run level 设置为这三个值

4、run level 的配置文件**/etc/inittab**

- CentOS6.5 只有 **initdefault** 这一个字段了，其余的设置被写入到了不同的文件中，在/etc/inittab 的注释中都有说明，自行查看即可

21.1.4. init 处理系统初始化流程(/etc/rc.d/rc.sysinit)

- **rc.d：run control directory**

1、大致流程

- **取得网络环境与主机类型**：读取网络配置文件/etc/sysconfig/network，取得主机名(HOSTNAME)与默认网关(gateway)
- **测试与挂载内存设备/proc 以及 USB 设备/sys**：还会检测系统是否具有 usb 的设备，若有则会主动加载 usb 的驱动程序，并尝试挂载 usb 的文件系统
- **决定是否启动 SELinux**
- **启动系统的随机数生成器**
- **设置终端机(console)字体**
- **设置现实与启动过程中的欢迎界面(textbanner)**
- **设置系统时间(clock)与时区设置**：需读入/etc/sysconfig/clock 设置值
- **接口设备的检测与 Plug and Play(PnP)参数的测试**：根据内核在启动时检测的结果/proc/sys/kernel/modprobe)开始进行/ide/scsi/网络/音效等接口设备的检测
- **用户自定义模块的加载**：用户可以在/etc/sysconfig/modules/*.modules 中加入自定义的模块，此时会被加载到系统当中，**类似于 systemd 的 unit 单元**
- **加载内核的相关设置**：系统会主动去读取/etc/sysctl.conf 这个文件的设置值
- **设置主机名与初始化电源管理模块(ACPI)**
- **初始化软件磁盘阵列**：主要通过/etc/mdadm.conf 来设置
- **初始化 LVM 文件系统功能**
- **以 fsck 检验磁盘文件系统**：会进行 filesystem check
- **进行磁盘配额 quota 的转换**
- **重新以可读写模式挂载系统磁盘**
- **启动 quota 功能**：所以我们不需要自定义 quotaon 操作
- **启动系统伪随机数生成器**
- **清除启动过程当中的临时文件**

- 将启动相关信息加载/var/log/dmesg 文件中

21.1.5. 启动系统服务与相关启动配置文(/etc/rc.d/rcN&/etc/sysconfig)

1、加载内核让这个系统准备接受命令来工作，再经过/etc/rc.d/rc.sysinit 的系统模块与相关硬件信息的初始化后，Linux 系统应该可以顺利工作了

2、我们还需要启动系统所需要的各项服务，这样主机才能提供相关的网络或者主机功能，这时，根据/etc/inittab 里面的 run level 值，就可以决定启动的服务选项了

3、通过/etc/rc.d/rc 这个命令来处理相关任务，例如默认的 run level 为 5

- 通过外部第一号参数($1)来取得想要执行的脚本目录，即由/etc/rc.d/rc **5** 可以取得/etc/rc**5**.d/这个目录来准备处理相关的脚本程序
- 找到/etc/rc5.dK??*开头的文件，并进行/etc/rc5.d/K??*stop 操作
- 找到/etc/rc5.dS??*开头的文件，并进行/etc/rc5.d/S??*start 操作

4、/etc/rcX.d/目录的文件解析：

- X 代表运行级别(runlevel)
- 文件名全以 Sxx 或 Kxx 开头，xx 为数字
- 全部是连接文件，连接到 stand alone 服务启动的目录/etc/init.d/去

21.1.6. 用户自定义开机启动程序(/etc/rc.d/rc.local)

1、在完成默认的 runlevel 指定的各项服务的启动后，我们如果还有其他操作想要完成，将它直接写到/etc/rc.d/rc.local，那么该工作在启动的时候被自动加载

21.1.7. 根据/etc/inittab 的设置加载终端机或 X Windows 界面

1、现在这段设置不在/etc/inittab 中，不知道放在哪里了

21.1.8. 启动过程会用到的主要配置文件

1、/sbin/init 运行过程中会执行许多脚本，包括/etc/rc.d/rc.sysinit 以及/etc/rc.d/rc，这些脚本都会用到相当多的系统配置文件，**这些启动过程中会用到的配置文件则大多放置在/etc/sysconfig/目录下**

2、/etc/modprobe.conf

3、/etc/sysconfig/*

21.1.9. Run level 的切换

1、依据启动是否进入不同 run level 的设置

- 要每次启动都执行某个默认的 run level，则需要修改/etc/inittab 内的设置选项
- 如果仅只是暂时更改系统的 run level 时，则使用 init [0-6]来进行 run level 的更改，下次重启时，依旧以/etc/inittab 为准

2、在 run level 为 5 时，执行 init 3 时

- 比较/etc/rc3.d/以及/etc/rc5.d 内的 K 与 S 开头的文件
- 在新的 run level 即是/etc/rc3.d/内有多的 K 开头的文件，则予以关闭
- 在新的 run level 即使/etc/rc3.d/内有多的 S 开头的文件，则予以启动

3、<runlevel>可以查看目前的 run level 等级

21.2. 内核与内核模块

1、在整个启动过程中，是否能够成功驱动我们主机的硬件配备是内核(kernel)的工作，而内核一般是压缩文件，因此在使用内核之前，就要将它解压缩后才能加载内存当中

2、为了应付日新月异的硬件，目前的内核都是可读取模块化程序的功能，即所谓的 modules(模块化)的功能，所谓模块化可以将它想象成一个"插件"

3、**内核与内核模块放置位置**：

- 内核：/boot/vmlinuz 或/boot/vmlinuz-version
- 内核解压缩所需 RAMDisk：/boot/initrd(/boot/initrd-version)
- 内核模块：/lib/modules/version/kernel 或/lib/modules/$(uname -r)/kernel
- 内核源码：/usr/src/linux 或/usr/src/kernels
- 内核版本：/proc/version
- 系统内核功能：/proc/sys/kernel

4、若有个新的硬件，但操作系统不支持，怎么办

- 重新编译内核，并加入最新的硬件驱动程序源码
- 将硬件的驱动程序编译成为模块，在启东时加入该模块

21.2.1. 内核模块与依赖性

1、/lib/modules/$(uname -r)/kernel 中的目录

- arch：与硬件平台有关的选项，例如 CPU 等级
- crypto：内核所支持的加密技术，例如 md5 或者是 des
- drivers：一些硬件的驱动程序，例如显卡，网卡，PCI 相关硬件
- fs：内核所支持的文件系统，如 vfat，reiserfs，nfs 等
- lib：一些函数库
- net：与网络有关的各项协议数据，还有防火墙模块(net/ipv4/netfilter/*)
- sound：与音效有关的各项模块

2、检查/lib/modules/$(uname -r)/modules.dep 这个文件，该文件记录了内核支持的模块各项依赖性

- <depmod>
- depmod [-Ane]
- -A：不加任何参数是，depmod 会主动分析目前内核的模块，并重新写入/lib/modules/$(uname -r)/modules.dep 当中，若加入-A 参数，则 depmod 会查找比 modules.dep 内容还要新的模块，如果找到了，才会更新
- -n：不写入 modules.dep，而是将结果输出到屏幕上(stdout)
- -e：显示出目前已加载的不可执行的模块名称

21.2.2. 内核模块的查看

1、<lsmod>

- 查看目前内核加载了多少模块
- lsmod
- 显示的内容

- 模块名称
- 模块大小
- 此模块是否被其他模块使用，若有，则将其全部列出

2、<modinfo>

- 查阅模块的具体信息
- modinfo [-adln] [module_name|filename]
- -a：仅列出作者名称
- -d：仅列出该 modules 的说明
- -l：仅列出授权
- -n：仅列出该模块的详细路径

21.2.3. 内核模块的加载与删除

1、<insmod>

- 完全由用户自行加载一个完整文件名的模块，并不会主动分析模块依赖性
- insmod [/full/path/module_name] [parameters]

2、<rmmod>

- rmmod [-fw] module_name
- -f：强制删掉该模块，不论是否被使用
- -w：若该模块正在被使用，则 rmmod 会等待该模块被使用完毕后删除它

3、<modprobe>

- **推荐使用，会直接查找 modules.dep 记录，所以可以克服模块依赖性的问题，而且还不需要模块的详细路径**
- modprobe [-lcfr] module_name
- -c：列出目前系统的所有模块
- -l：列出在/lib/modules/`uname -r`/kernel 当中所有模块完整文件名
- -f：强制加载该模块
- -r：类似 rmmod，删除某个模块

21.2.4. 内核模块的额外参数设置：/etc/modprobe.conf

1、找不到该文件???

21.3. Boot Loader:Grub

21.3.1. boot loader 的两个 stage

1、MBR 是整个硬盘的第一个 sector 内的一个块，大小只有 446bytes，但 loader 光是程序代码都不止 446bytes 的容量，为了解决这个问题，Linux 将 boot loader 的程序代码执行与设置加载分成两个阶段(stage)

- Stage1：执行 boot loader 主程序
  - 这个主程序必须安装在启动区，即 MBR 或 boot sector
  - 通常 MBR 仅安装 boot loader 最小主程序，并没有安装 loader 配置文件
- Stage2：主程序加载配置文件
  - 通过 boot loader 加载所有配置文件与相关环境参数文件(包括文件系统定义与主要配置文件 menu.lst)，一般来说，配置文件都在/boot 下面

21.3.2. grub 的配置文件/boot/grub/menu.lst 与菜单类型

1、grub 的优点

- 认识与支持较多的文件系统，并且可以使用 grub 的主程序直接在文件系统中查找内核文件名
- **启动的时候可以自行编辑与修改启动设置选项，类似 bash 的命令模式**
- 可以动态查找配置文件，而不需要在修改配置文件后重新安装 grub，即我们只要修改完/boot/brub/menu.lst 里头的设置后，下次启动就生效了
- **上面三点就是 Stage 1 与 Stage 2 分别安装在 MBR(主程序)与文件系统当中(配置文件与定义文件)的原因**

2、硬盘与分区在 grub 中的代号

- 安装在 MBR 的 grub 主程序最重要的任务之一就是从磁盘当中加载内核文件，以让内核能够顺利驱动整个系统的硬件
- grub 对硬盘的代号设置与传统的 Linux 磁盘代号完全不同
  - (hd0,0)
  - 硬盘代号以小括号"()"括起来
  - 硬盘以 hd 表示，后面接一组数字
  - **以"查找顺序"作为硬盘的编号，而不是依照硬盘扁平电缆的排序(重要)**
  - 第一个查找到的硬盘为 0 号，第二个为 1 号...
  - 每块硬盘的第一个分区代号为 0，依次类推

3、/boot/grub/menu.lst 配置文件

- title 前四行都是 grub 的整体设置，包括默认的等待时间与默认的启动选项，还有显示的界面特性等，title 后面的才是指定启动的内核文件或者 boot loader 控制权
- default=0
  - 要与 title 作为对照，在配置文件里有几个 title，启动的时候就有几个菜单可以选择
  - 由于 grub 起始号码为 0，因此 default=0 代表使用第一个 title 选项来启动的意思
  - default 的意思是如果在读秒时间结束前都没有按键，grub 默认使用此 title 选项来启动
- timeout=5
  - 启动会进行读秒，如果在 5 秒钟内没有按下任何键，就会使用上面提到的 default 后面接的 title 选项来启动
- splashimage=(hd0,0)/grub/splash.xpm.gz
  - 这个文件提供后台图示
- hiddenmenu
  - 启动时是否需要显示菜单，默认 CentOS 是不显示菜单的，若果想要显示，就注释掉这个设置

4、直接指定内核启动：

- 既然要指定内核启动，当然要找到内核文件，此时有可能还需要用到 initrd 的 RAM Disk 配置文件，但目前启动尚未完成，所以必须以 grub 的硬盘识别方式找出完整的 kernel 与 initrd 文件名才行
- 参数解释

- root：代表的是内核文件放置的那个分区，不是根目录
- kernel：kernel 后面接的则是内核的文件名，而在文件名后面接的则是内核的参数
- initrd：就是前面提到的 initrd 制作出 RAM Disk 文件名

5、利用 chain loader 的方式转交控制权

- 仅是将控制权交给下一个 boot loader 而已

21.3.3. initrd 的重要性与创建新 initrd 文件

1、initrd 的目的在于提供启动过程中所需要的最重要的内核模块，以让系统启动过程可以顺利完成

2、需要 initrd 的原因：

- 因为内核模块放置于/lib/modules/$(uname -r)/kernel 当中，这些模块必须要根目录'/'被挂载时才能够被读取
- 但如果内核本身不具有磁盘的驱动程序时，当然无法挂载根目录，也就没有办法读取驱动程序，因此造成两难的地步

3、initrd 作用

- initrd 可以将/lib/modules/...内的启动过程当中一定需要的模块打包成一个文件(文件名就是 initrd)
- 然后在启动时通过主机的 INT13 硬件功能将该文件读出来解压缩，并且 initrd 在内存内会仿真成为根目录
- 由于此虚拟文件系统(Initial RAM Disk)主要包含磁盘与文件系统的模块，因此我们的内核最后就能够认识实际的磁盘，于是就能进行根目录的挂载
- initrd 内所包含的模块大多与启动过程有关，而主要以文件系统以及硬盘模块(如 usb、SCSI 等)为主

4、需要 initrd 的时刻

- 根目录所在磁盘为 SATA、USB 或 SCSI 等接口
- 根目录所在文件系统为 LVM、RAID 等特殊格式
- 根目录所在文件系统为非传统 Linux"认识"的文件系统
- 其他必须要在内核加载时提供的模块

5、一般来说，各个 distribution 提供的内核都会附上 initrd 文件，如果想要重制 initrd 文件的话，可以使用 mkinitrd 来处理

6、<mkinitrd>

- mkinitrd [-v] [--with=模块名称] [initrd 文件名] [内核版本]

21.3.4. 测试与安装 grub

1、<grub>

- root (hdx,x), x 为数字：选择含有 grub 目录的那个分区代号，例如 root (hd0,0)
- find /boot/grub/stage：找到安装信息文件
  - 若 boot 是独立的分区，那么 find /grub/stage
- find /boot/vmlinuz-xx.xxx.xxx：看看能否找到内核文件
  - 若 boot 是独立的分区，那么 find /vmlinuz-xx.xxx.xxx

- setup (hdx,x),x 为数字：将 grub 安装在 boot sector 中
- setup (hdx), x 为数字：将 grub 安装在 MBR 中

2、总结

- 如果从其他 boot loader 转成 grub 时，得先用 grub-install 安装 grub 配置文件
- 开始编辑 menu.lst 这个重要的配置文件
- 通过 grub 来将主程序安装到系统中，如 MBR(hd0)或 bootsector(hd0,0)等

21.3.5. 启动前的额外功能修改

1、在 boot loader 选择的菜单界面

- e：启动前编辑命令
- a：修改内核参数
- c：进入 grub shell

2、在启动菜单将光标移动任意一个菜单(在上一步按下 e)，按下 e，将会显示出 menu.lst 里面设置的东西，此时

- e：进入 gurb shell 的编辑界面
- o：在光标下面再新增一行
- d：将光标所在行删除

21.3.6. 关于内核功能当中的 vga 设置

1、tty1~tty6 除了 80*24 分辨率外，还能有其他分辨率的支持，前提是内核必须支持 FRAMEBUFFER_CONSOLE 这个内核功能参数才行

2、查看是否支持 FRAMEBUFFER_CONSOLE

- grep 'FRAMEBUFFER_CONSOLE' /boot/config<这里按 Tab>
- 若出现 CONFIG_FRAMEBUFFER_CONSOLE=y 则说明支持

3、分辨率/色度对照表表

颜色彩度/分辨率	640x480	800x600	1024x768	1280x1024	bit
256	769	771	773	775	8 bit
32768	784	787	790	793	15 bit
65536	785	788	791	794	16 bit
16.8M	786	789	792	795	32 bit

- 若想要终端机屏幕分辨率调到 1024x768 且颜色深度为 15bit 时，就要指定 vga=790 这个数字

21.3.7. BIOS 无法读取大硬盘的问题

1、如果用旧主板来安插大容量硬盘时，可能由于系统 BIOS 或其他问题，导致 BIOS 无法判断该硬盘的容量，此时你的系统读取可能会有问题

2、

21.3.8. 为某个菜单加上密码

1、某种情况，你需要为某个菜单做个密码，而且必须是加密过的，否则找到/boot/grub/menu.lst 就可以找到启动密码了

2、利用 grub 提供 md5 编码来处理

- grub-md5-crypt

- 输入该命令会让你输入密码两次，随后生成一串字符，将其复制，包括最开始的'$'与结束的'.'(如果有的话)，即生成那行的所有字符
- 在/boot/grub/menu.lst 中你需要加密的选项的 title 下方添加一行(这个选项一定要处于 title 下面的第一行)：
- password --md5 [刚复制的那串字符]

3、其实上述方法仍然可以通过在选项处按 e 进入编辑界面，用 d 删掉 password 这一行，再按 b 直接启动

4、**只有通过整体的 password(放在所有 title 之前)，然后在 title 下面的第一行设置 lock，才能完全防破解，接用户想要编辑也得输入密码**

- 在 title 下方没有加 lock 的选项仍然可以进入编辑模式哦

21.4. 启动过程的问题解决

1、进入 runlevel1(单用户维护模式)去处理

21.4.1. 忘记 root 密码的解决之道

1、忘记 root 密码是可以补救的

- 只要能够进入并挂载/，然后重新设置 root 的密码即可
- 因为在启动流程中，若强制内核进入 runlevel 1 时，默认是不需要密码即可取得一个 root 的 shell 来救援的

2、步骤

- 重新启动
- 进入 grub 菜单后，在要你进入的菜单上面按'e'进入编辑模式，将光标移动到 kernel 上方点 e 进入编辑界面，在最后加上 single 即可
- grub edit> kernel /vmlinuz-xxxxxxxxx ro root=LABEL=/ rhgb quiet **single**
- 在按 b 就能够启动进入单用户模式了
- 进入单用户模式后，系统会以 root 的权限直接给你一个 shell，你就能执行 passwd 这个命令来重建 root 的密码，然后直接 init 5 就可以切换为 X 窗口界面，或者 init 3 进入文本界面

21.4.2. init 配置文件错误

1、唯一一个很难挽救 root 丢失的情况，就是/etc/inittab 这个文件设置错误导致无法启动，因为 runlevel 0- runlevel 6 都会读取/etc/inittab 配置文件

2、解决办法：告诉内核不要执行 init，改用 bash，直接略过 init

- 重新启动
- 进入 grub 菜单后，在要你进入的菜单上面按'e'进入编辑模式，将光标移动到 kernel 上方点 e 进入编辑界面，在最后加上 single 即可
- grub edit> kernel /vmlinuz-xxxxx ro root=LABEL=/ rhgb quiet **init=/bin/bash**
- 于是，我们指定了内核调用的第一个进程(init)变成了/bin/bash，因此/sbin/init 就不会被执行
- 根据启动流程的说明，此时虽然可以用 root 取得 bash 工作，但是除了根目录外，其他目录都没有挂载
- mount -o remount,rw /
- mount -a

- 进行救援工作，救援完毕后 reboot 重启即可

21.4.3. BIOS 磁盘对应的问题

1、将系统装在不同的磁盘上，这样 MBR 与 boot sector 就都有启动装载程序了

2、但是 **grub 对磁盘的设备代号使用的是检测到的顺序**，也就是说，你调整了 BIOS 磁盘启动顺序后，**原先在/boot/grub/menu.lst 内的设备代号就可能会对应到错误的磁盘上了**

3、解决方法：

- 通过/boot/grub/device.map 这个文件来写死每个设备对 grub 磁盘代号的对应关系
  - (fd0) /dev/fd0
  - (hd0) /dev/hda
- grub-install --recheck /dev/hda1 <==或者用该命令来更新.map 文件

21.4.4. 因文件错误而无法启动

1、一般会出现提示你可以用 root 来进行补救，那就照做即可，然后用 root 的身份来进行文件的更正

21.4.5. 利用 chroot 切换到另一块硬盘工作

1、change root directory 的意思：暂时将根目录移动到某个目录下，然后处理某个问题，最后再离开该 root 回到原本的系统当中

2、假设有问题的 Linux 磁盘在/dev/hdb1 上面

- / -> /dev/hdb1
- /var -> /dev/hdb2
- /home -> /dev/hdb3
- /usr -> /dev/hdb5
- 可以在另一个没问题的 Linux 下面新建一个文件，例如/chroot
- 将/dev/hdb{1,2,3,5}全部挂载到该目录下面，即
- /chroot/ -> /dev/hdb1
- /chroot/var/ -> /dev/hdb2
- /chroot/home/ -> /dev/hdb3
- /chroot/usr/ -> /dev/hdb5
- 全部挂载完毕后，再输入 chroot /chroot

21.5. Grub2 简介

1、参考 http://forum.ubuntu.org.cn/viewtopic.php?t=466590

21.5.1. Grub2 模块

1、命令模块[command.lst]

- 提供了各种不同的功能，类似标准 Unix 命令，一共将近 100 个
- 例如：cat cpuid echo halt lspci chainloader initrd linux password ...

2、加密模块[crypto.lst]

- 提供了各种数据完整性校验与密码算法支持，一共 20 多个
- 例如：gcry_rijndael crc64 gcry_md5 ...

3、文件系统模块[fs.lst]

- 提供了访问各种文件系统的功能，一共 30 多个
- 例如：btrfs cpio exfat ext2 fat iso9660 ntfs tar xfs zfs ...

4、分区模块[partmap.lst]
- 提供了识别各种分区格式的功能，一共 10 多个
- 例如：part_bsd part_gpt part_msdos ...

5、分区工具[parttool.lst]
- 提供了操作各种分区格式的功能，目前只有 msdospart 这一个

6、终端模块[terminal.lst]
- 提供了各种不同终端的支持，一共不到 10 个
- 例如：serial gfxterm vga_text at_keyboard ...

7、视频模块[video.lst]
- 提供了各种不同的视频模式支持，一共 6 个
- 例如：vga vbe efi_gop efi_uga ...

8、其他模块
- 所有未在上述分类文件中列出的模块都归为这一类，一共将近 100 个
- 值得关注的有以下几个：
  - "all_video"可用于一次性加载当前所有可用的视频模块
  - "gfxmenu"可用于提供主题支持
  - "jpeg png tga"可用于提供特定格式的背景图片支持
  - "xzio gzio lzopio"可用于提供特定压缩格式支持(常配合"initrd"命令使用)
  - "memdisk"可用于提供内存盘支持，常用于配合 MEMDISK 工具引导各种镜像文件(ISO 镜像/软盘镜像/硬盘镜像)。

21.5.2. Grub2 救援模式

21.5.3. Grub2 命名规则

1、**磁盘从"0"开始计数，分区从"1"开始计数**

2、设备与分区
- (fd0)：第一软盘
- (hd0)：第一硬盘[大多数 U 盘与 USB 接口的移动硬盘以及 SD 卡也都被当作硬盘看待]
- (hd1,1)：第二硬盘的第一分区(通用于 MBR 与 GPT 分区)
- (hd0,msdos2)：第一硬盘的第二 MBR 分区，也就是传统的 DOS 分区表
- (hd1,msdos5)：第二硬盘的第五 MBR 分区，也就是第一个逻辑分区
- (hd0,gpt1)：第一硬盘的第一 GPT 分区
- (cd)：启动光盘[仅在从光盘启动 GRUB 时可用]
- (cd0)：第一光盘
- (memdisk)：内存盘[只能有一个]

3、文件
- (fd0)/grldr：第一软盘根目录下的"grldr"文件[绝对路径]
- (hd0,gpt1)/boot/vmlinuz：第一硬盘的第一 GPT 分区"boot"目录下的"vmlinuz"文件[绝对路径]
- /boot/vmlinuz：根设备"boot"目录下的"vmlinuz"文件[相对路径]，当"root"环境变量等于"(hd0,gpt1)"时，等价于"(hd0,gpt1)/boot/vmlinuz"

4、磁盘块

- (hd1,1)0+1：在第二硬盘的第一分区上，从第"0"个磁盘块(首扇区)起，长度为"1"的连续块。[绝对路径]
- (hd1,1)+1：含义与上一个相同，因为当从第"0"个磁盘块(首扇区)起时，"0"可以省略不写。[绝对路径]
- +1：在根设备上，从第"0"个磁盘块(首扇区)起，长度为"1"的连续块。[相对路径]，当"root"环境变量等于"(hd1,1)"时，等价于"(hd1,1)0+1"

21.5.4. Grub2 环境变量

1、GRUB2 的环境变量大致可以分为两类

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

2、大多数有特定含义的环境变量都是附属于特定附加模块的，只有加载了这些模块之后，这些环境变量才变得有意义。所以从模块的角度看，GRUB2 的环境变量又可以分为三类：

- **核心变量**：GRUB2 核心提供的变量，不依赖于任何可加载模块，这样的变量只有"**cmdpath prefix root**"三个，且它们的初始值都由 GRUB2 自动设置
- **模块变量**：绝大多数有特定含义的环境变量都属此类
- **脚本变量**：这是为了方便编写 grub.cfg 脚本而设置的变量，没有特殊含义，也不依赖于特定模块，与一般的 bash 脚本中的变量类似。

3、各变量详解

- **?**：上一条命令的返回值，零表示成功，非零表示失败[与 bash 一样]。由 GRUB2 自动设置。你只能使用此变量，而不能修改它
- **check_signatures**：是否在加载文件时强制验证签名，可以设为'yes'或'no'
- **chosen**：当前被执行的菜单项名称(紧跟"menuentry"命令之后的字符串或者'--id'选项的参数)，例如'Windows 7'。由 GRUB2 自动设置。你只应该使用此变量，而不应该修改它
- **cmdpath**：当前被加载的"core.img"所在目录(绝对路径)。例如：UEFI 启动可能是'(hd0,gpt1)/EFI/UBUNTU'或'(cd0)/EFI/BOOT'，BIOS 启动可能是'(hd0)'。由 GRUB2 自动设置。你只应该使用此变量，而不应该修改它
- **debug**：设为'all'时表示开启调试输出[会显示大量信息,谨慎开启]
- **default**：默认选中第几个菜单项(从'0'开始计数)
- **fallback**：如果默认菜单项启动失败，那么就启动第几个菜单项(从'0'开始计数)
- **gfxmode**：设置"gfxterm"模块所使用的视频模式，可以指定一组由逗号或分号分隔的模式以供逐一尝试：每个模式的格式必须是：'auto'(自动检测),'宽 x 高','宽 x 高 x 色深'之一，并且只能使用 VBE 标准指定的模式[640x480,800x600,1024x768,1280x1024]x[16,24,32]。可以在 GRUB SHELL 中使用"videoinfo"命令列出当前所有可用模式。默认值是'auto'
- **gfxpayload**：设置 Linux 内核启动时的视频模式，可以指定一组由逗号或

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

- **gfxterm_font**：设置"gfxterm"模块所使用的字体，默认使用所有可用字体
- **grub_cpu**：此 GRUB 所适用的 CPU 类型。例如：'i386', 'x86_64'。由 GRUB2 自动设置。你只应该使用此变量，而不应该修改它
- **grub_platform**：此 GRUB 所适用的平台类型。例如：'pc', 'efi'。由 GRUB2 自动设置。你只应该使用此变量，而不应该修改它。
- **lang**：设置 GRUB2 的界面语言，必须搭配"locale_dir"变量一起使用。简体中文应设为'zh_CN'
- **locale_dir**：设置翻译文件(*.mo)的目录，通常是'$prefix/locale'，若未明确设置此目录，则禁止国际化
- **pager**：如果设为'1'，那么每一满屏后暂停输出，等待键盘输入。缺省是''，表示不暂停
- **prefix**：绝对路径形式的'/boot/grub'目录位置(也就是 GRUB2 的安装目录)，例如'(hd0,gpt1)/grub'或'(hd0,msdos2)/boot/grub'。初始值由 GRUB 在启动时根据"grub-install"在安装时提供的信息自动设置。你只应该使用此变量，而不应该修改它
- **root**：设置"根设备"。任何未指定设备名的文件都视为位于此设备。初始值由 GRUB 在启动时根据"prefix"变量的值自动设置。在大多数情况下，你都需要修改它
- **superusers**：设置一组"超级用户"(使用空格/逗号/分号进行分隔)，以开启安全认证的功能
- **theme**：设置菜单界面的主题风格文件的位置，例如："/boot/grub/themes/starfield/theme.txt"。关于如何定制界面风格(背景图片/字体/颜色/图标等)的细节，可以参考 GRUB2 手册中的"Theme file format"部分
- **timeout**：在启动默认菜单项前，等待键盘输入的秒数。默认值是'5'秒。'0'表示直接启动默认菜单项(不显示菜单)，'-1'表示永远等待。

21.5.5. Grub2 命令(grub 环境下而非 bash 下的命令)

1、<menuentry>

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

- --class：指定菜单项所属的"样式类"。从而可以使用指定的主题样式显示菜单项
- --users：指定只允许特定的用户访问此菜单项。如果没有使用此选项，则表示允许所有用户访问
- --unrestricted：指明允许所有用户访问此菜单项
- --hotkey：设置访问此菜单项的热键(快捷键)。"key"可以是一个单独的字母，或者'backspace','tab','delete'之一
- --id：为此菜单项设置一个全局唯一的标识符。"id"必须由 ASCII 字母/数字/下划线组成，且不得以数字开头
- [arg …]：可选的参数列表。你可以把它们理解为命令行参数。实际上"title"也是命令行参数，只不过这个参数是个必须参数而已。这些参数都可以在花括号内的命令列表中使用，"title"对应着"$1"，其余的以此类推。

2、<terminal_input>
- terminal_input [--append|--remove] [terminal1] [terminal2] …
- --append：指定的终端加入到激活的输入终端列表中，所有列表中的终端都可以用于向 GRUB 提供输入
- --remove：指定的终端从激活的输入终端列表中删除
- 如果不使用任何选项，但是指定了一个或多个终端参数，则表示将当前激活的输入终端设置为参数指定的终端。

3、<terminal_output>
- terminal_output [--append|--remove] [terminal1] [terminal2] …
- 如果不带任何选项与参数，则表示列出当前激活的输出终端，以及所有其他可用的输出终端
- --append：将指定的终端加入到激活的输出终端列表中，所有列表中的终端都将接受到 GRUB 的输出
- --remove：选项将指定的终端从激活的输出终端列表中删除
- 如果不使用任何选项，但是指定了一个或多个终端参数，则表示将当前激活的输出终端设置为参数指定的终端。

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

5、<linux16>
- linux16 file …
- 以传统的 16 位启动协议从"file"载入一个 Linux 内核映像，并将其余的字符作为内核的命令行参数逐字传入。这通常用于启动一些遵守 Linux 启动协议的特殊工具(例如 MEMDISK)
- [注意]使用传统的 16 位启动协议意味着：(1)'vga='启动选项依然有效，(2)不能启动纯 64 位内核(也就是内核必须要'CONFIG_IA32_EMULATION=y'才行)。

6、<initrd>

- initrd file
- 为以 32 位协议启动的 Linux 内核载入一个"initial ramdisk"，并在内存里的 Linux 设置区域设置合适的参数。
- [注意]这个命令必须放在"linux"命令之后使用。

7、<initrd16>

- initrd16 file
- 为以 16 位协议启动的 Linux 内核载入一个"initial ramdisk"，并在内存里的 Linux 设置区域设置合适的参数。

8、<search>

- search [--file|--label|--fs-uuid] [--set=[var]] [--no-floppy] name
- 通过文件[--file]、卷标[--label]、文件系统 UUID[--fs-uuid]来查找设备
- 如果使用了 --set 选项，那么会将第一个找到的设备设置为环境变量"var"的值。默认的"var"是'root'。
- 可以使用 --no-floppy 选项来禁止查找软盘设备，因为这些设备非常慢。

9、<password_pbkdf2>

- password_pbkdf2 user hashed-password
- 定义一个名为 user 的用户，并使用哈希口令'hashed-password'(通过**"grub2-mkpasswd-pbkdf2"(bash 中的命令，密码有效部分参考** Grub2 安全**的示例)**工具生成)。这是建议使用的命令，因为它安全性更高。

21.5.6. Grub2 安全

1、在默认情况下，GRUB 对于所有可以在物理上进入控制台的人都是可访问的。任何人都可以选择并编辑任意菜单项，并且可以直接访问 GRUB SHELL

2、要启用认证支持，必须将环境变量"superusers"设置为一组用户名(可用空格/逗号/分号作为分隔符)，这样，将仅允许"superusers"中的用户使用 GRUB 命令行、编辑菜单项、以及执行任意菜单项。而其他非"superusers"中的用户，只能执行那些没有设置 --users 选项的菜单，以及那些在 --users 选项中包含了该用户的菜单，但不能使用 GRUB 命令行、编辑菜单项

3、下面使用一个配置片段举例说明：

```
set superusers="root"
password_pbkdf2 root grub.pbkdf2.sha512.10000.biglongstring
password user1 insecure

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

menuentry "仅允许超级用户执行此菜单" --users "" {
    ...
}

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

1、grub2 的特性

- 图形接口
- 使用了模块机制，通过动态加载需要的模块来扩展功能
- 支持脚本语言，例如条件判断，循环、变量和函数
- 支持 rescue 模式，可用于系统无法引导的情况
- 国际化语言。包括支持非 ASCII 的字符集和类似 gettext 的消息分类，字体，图形控制台等等
- 有一个灵活的命令行接口。如果没有配置文件存在，GRUB 会自动进入命令模式
- 针对文件系统、文件、设备、驱动、终端、命令、分区表、os loder 的模块化、层次化、基于对象的框架
- 支持多种文件系统格式
- 可访问已经安装的设备上的数据
- 支持自动解压

2、设备的命名

- grub2 同样以 fd 表示软盘，hd 表示硬盘(包含 IDE 和 SCSI 硬盘)。设备是从 0 开始编号，分区则是从 1 开始，主分区从 1-4，逻辑分区从 5 开始
- 而 grub 分区编号是从 0 开始的。

3、查看 UUID

- sudo blkid
- ls -l /dev/disk/by-uuid

Chapter 22. 系统设置工具(网络与打印机)与硬件检测

22.1. CentOS 系统设置工具:setup

1、系统设置出了使用手动的方式编辑配置文件外(例如/etc/inttab,/etc/fstab 等)，Red Hat 系统的 RHEL，CentOS 以及 Fedora 还有提供一支总和程序来管理，即 setup 命令

2、<setup>

- 在启用 systemd 后(例如 CentOS7)，功能大大减弱
- Authentication configuration：与用户身份认证有关的设置，包括本机的账号与利用远程服务器提供的账号来登陆本机等功能的设置
- Firewall configuration：
- Keyboard configuration：
- Networkconfiguration
- System services
- Timezone configuration
- X configuration

Chapter 23. 软件安装：源码与 Tarball

23.1. 开放源码的软件安装与升级简介

1、在 Windows 上面的软件，你无法修改该软件的源代码

2、Linux 上面的软件都是经过 GPL 的授权，所以每个软件几乎都提供源代码，并且你可以自行修改该程序代码，以符合你个人的需求

23.1.1. 什么是开放源码、编译程序与可执行文件

1、在 Linux 上面，一个文件能不能被执行看的是有没有可执行权限(但有 x 权限并不意味着你一定能执行它哦)

2、Linux 系统上面真正识别的可执行文件是二进制文件，例如/usr/bin/passwd /bin/touch 这些文件即为二进制文件

3、对于 shell script：只是利用 shell(例如 bash)这个程序的功能进行一些判断式，而最终执行的除了 bash 提供的功能外，仍是调用一些已经编译好的二进制程序来执行

4、**判断一个文件是否为二进制文件：file，见**查看文件类型：<file>

5、我们如何写一个二进制程序

- 首先就用一般的文本处理器编写，例如 vim
  - **写完的程序就是所谓的源代码**
  - 这个程序代码文件其实就是一般的纯文本文件
- 然后将这个文件"编译"成操作系统看得懂的二进制程序
  - 编译自然需要编译程序来做，经过编译程序的编译与链接之后，就会生成一个可执行的二进制程序
  - 在 Linux 上面最标准的程序语言为 C，所以使用 C 的语法进行源代码的书写，写完之后，以 Linux 上标准的 C 语言编译程序 gcc 这个程序来编译，就可以制作一个可执行的二进制程序
  - **事实上，在编译过程中还会生成所谓的目标文件(Object file)，这些文件以*.o 的扩展名形式存在，C 语言的源代码文件通常以*.c 作为扩展名**
  - **此外，有时候我们会在程序中引用，调用其他的外部子程序，或者利用其他软件提供的"函数功能"，这个时候就必须在编译过程中将该函数库加进去，如此一来，编译程序就可以将所有的程序代码与函数库做一个链接(LINK)以生成正确的执行文件**
- 流程如图
-

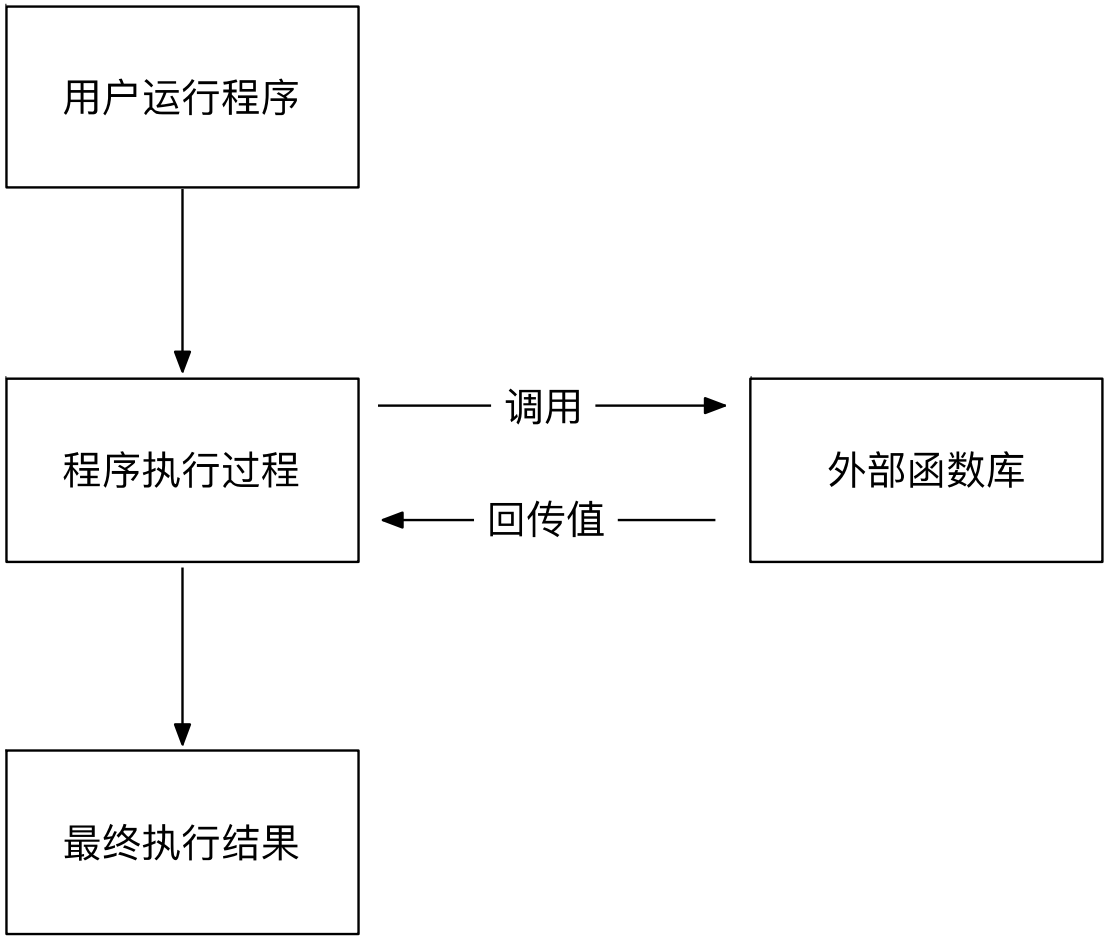
6、总结

- 开放源码：就是程序代码，写给人类看的程序语言，但机器并不认识，所以无法执行
- 编译程序：将程序代码转译成为机器看得懂的语言，类似翻译者的角色
- 可执行文件：经过编译程序变成二进制程序后机器看得懂所以可以执行的文件

23.1.2. 什么是函数库

1、例如 Linux 系统上通常提供一个可以进行身份验证的模块，PAM 模块

2、函数库又分为动态与静态函数库
3、程序执行情况(有调用外部函数库的程序)



4、函数库：类似子程序的角色，可以被调用来执行的一段功能函数

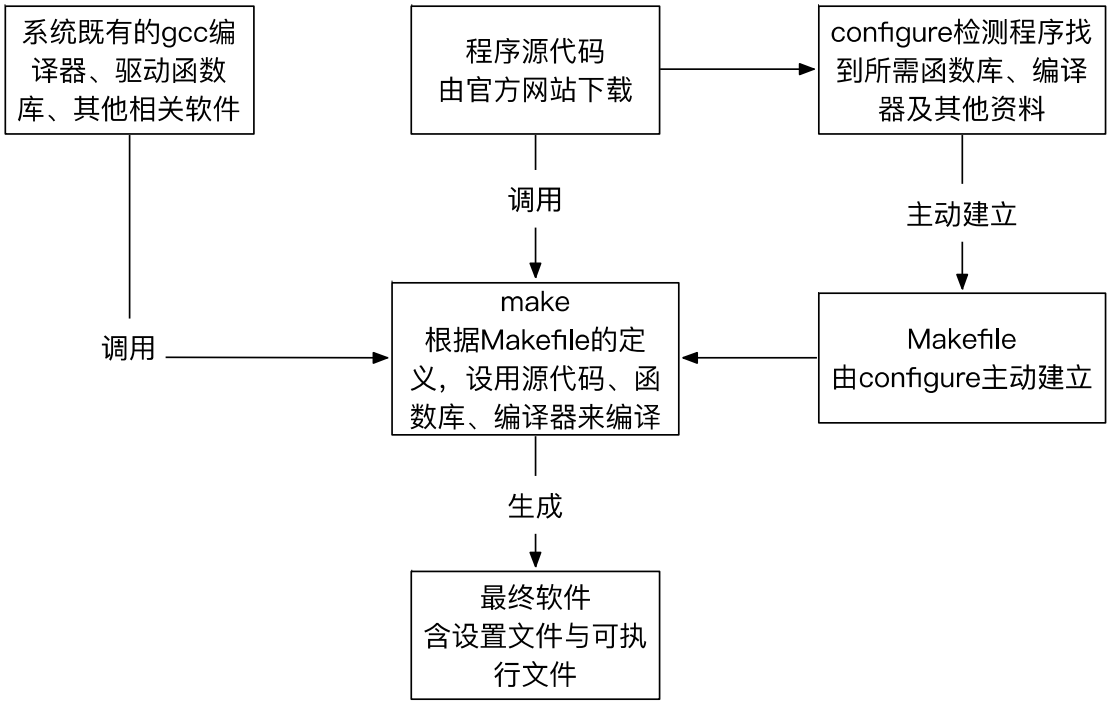
23.1.3. 什么是 make 与 configure

1、事实上，使用 gcc 的编译程序来进行编译的过程并不简单
- 因为一套软件并不会仅有一个程序，而是有一堆程序代码文件
- 所以除了每个主程序与子程序均需要写上一条编译过程的命令外，还需要写上最终的链接程序
- 大型程序会有相当多的程序需要编译，非常麻烦

2、<make>
- 使用 make 这个命令的相关功能来进行编译过程的命令简化
- 当执行 make 时，make 会在当前目录下搜索 Makefile(or makefile)这个文本文件，而 Makefile 里面记录了源码如何编译的详细信息
- make 会自动判别源码是否经过了变动而自动更新执行文件
- makefile 如何写？
  - 通常软件开发商都会写一个**检测程序来检测用户的操作环境**，以及该操作环境是否有软件开发商所需要的其他功能，该检测程序检测完毕后，就会主动新建这个 makefile 的规则文件，**通常这个检测程序的文件名为 configure 或 config**

- 为什么要检测操作环境？
  - 不同版本的内核所使用的系统调用可能不同，每个软件需要的相关函数库也不同，同时开发商不会仅针对 Linux 开发，而是会针对整个 UNIX-Like 做开发，因此他必须检测该操作系统平台有没有提供何时的编译程序才行，所以必须要检测环境
- 检测程序会检测的数据：
  - 是否有合适的编译程序可以编译本软件的程序代码
  - 是否已经存在本软件所需的函数库或其他需要的相关软件
  - 操作系统平台是否适合本软件，包括 Linux 的内核版本
  - 内核的头定义文件(header include)是否存在(驱动程序必须要的检测)

3、整个流程



23.1.4. 什么是 Tarball 的软件

1、Tarball 文件由来

- 源代码就是一些写满了程序代码的纯文本文件
- 纯文本文件在网络上其实是很浪费带宽的一种文件格式
- 如果能够将这些源码通过文件的打包与压缩技术来将文件的数量与容量减小，不但用户容易下载，软件开发商的网站带宽也能节省很多，这就是 Tarball 文件的由来

2、Tarball 包含的内容

- 源代码文件
- 检测程序文件(可能使 configure 或 config 文件名)
- 本软件的简易说明与安装说明(INSTALL 或 README)

23.1.5. 如何安装与升级软件

1、需要升级的原因

- 需要新的功能，但旧有主机的旧版本软件并没有

- 旧版本软件上面可能有安全隐患，需要更新到新版本软件
- 旧版本软件执行性能不佳，或者执行的能力不能让管理者满足

2、更新的方法

- 直接以源码通过编译来安装与升级
  - 直接以 Tarball 在自己的机器上进行检测、编译、安装与设置等操作来升级
  - 具有很高的可选择性，但操作麻烦一点
- 直接以编译好的二进制程序来安装与升级
  - Linux distribution 厂商针对自己的操作平台先进行编译等过程，再将编译好的二进制程序释出，省略了检测与编译等繁杂的过程，较为容易
  - 这个预编译机制存在于很多 distribution，包括 Red Hat 系统(含 Fedora/CentOS 系列)开发的 RPM 软件管理机制与 yum 在线更新模式，Debian 使用 dpkg 软件管理机制与 APT 在线更新模式

3、如何安装一个软件的 Tarball

- 将 Tarball 由厂商的网页下载下来--1
- 将 Tarball 解压缩，生成很多的源码文件--2
- 开始以 gcc 进行源码的编译(会生成目标文件)--3
- 开始以 gcc 进行函数库、主程序、子程序的链接，以形成主要的二进制文件--4
- 将上述的二进制文件以及相关的配置文件安装至自己的主机上面--5
- 3、4 两步可以通过 make 这个命令来简化

23.2. 使用传统程序语言进行编译的简单范例

23.2.1. 单一程序：打印 Hello World

23.2.2. 主程序、子程序链接：子程序的编译

1、程序的编译与链接

- gcc -c thanks.c thanks2.c ==>生成 thanks.o,thanks2.o
- gcc -o thanks thanks.o thanks2.o =>>生成 thanks(可执行文件)
- 每个源文件(*.c 文件)进行单独编译，编译时提供三个表：未解决符号表、导出符号表、地址重定向表

23.2.3. 调用外部函数库：加入链接的函数库

23.2.4. gcc 的简易用法(编译、参数与链接)

1、<gcc>

- gcc -c hello.c ==>自动生成 hello.o 这个文件
- gcc -O hello.c -c ==>自动生成 hello.o 这个文件，并且进行优化
- gcc sin.c -lm -L/usr/lib -I/usr/include
  - -lm：指的是 libm.so 或 libm.a 这个函数库文件
  - -L：后面接的路径是上面那个函数库的搜索目录
  - -I：后面接的是源码 include 文件的所在目录
- gcc -o hello hello.c ==>生成要输出的 binary file 文件名

- gcc -o hello hello.c -Wall ==>加入 Wall 后，程序编译更严谨
- gcc hello.c
  - 如果直接以 gcc 编译源码，并且没有加上任何参数，则执行文件的文件名会被自动设置为 a.out，可以直接执行这个文

23.3. 用 make 进行宏编译

23.3.1. 为什么要用 make

1、make 的好处

- 简化编译时所需要执行的命令
- 若在编译完成之后，修改了某个源码文件，则 make 仅会针对被修改的文件进行编译，其他的目标文件则不会更改
- 可以依照相依性来更新执行文件

23.3.2. makefile 的基本语法与变量

1、makefile 的基本规则：

- [目标]:[目标文件 1] [目标文件 2]...
- <tab> gcc -o [欲新建的可执行文件] [目标文件 1] [目标文件 2]...

2、例如

```
target1: main.o haha.o sin_value.o cos_value.o
    gcc -o main main.o haha.o sin_value.o cos_value.o -lm
```

- make
- make target1 <==让其作为 make 的后接参数

3、变量功能来简化

```
LIBS = -lm
OBJS = main.o haha.o sin_value.o cos_value.o
target1: ${OBJS}
    gcc -o main ${OBJS} ${LIBS}
target2:
    rm -f main ${OBJS}
```

- 一般来说 target1 与 target2...通常写成"main"，"clean"，"install"
- 变量与变量内容以"="隔开，同时两边可以具有空格
- 变量左边不可以有<tab>
- 变量与变量在"="两边不能具有":"
- 在习惯上，变量以大写字母为主
- 运用变量时，以${变量}或$(变量)使用
- 在该 shell 的环境变量是可以被套用的，例如提到的 CFLAGS 这个变量
- 在命令模式也可以定义变量

23.4. Tarball 的管理与建议

1、Tarball 的安装是可以跨平台的，因为 C 语言的程序代码在各个平台上面是可以共用的

23.4.1. 使用源码管理软件所需要的基础软件

1、基础软件

- gcc 或 cc 等 C 语言编译程序，gcc 是首选
- make 及 autoconfig 等软件
- 需要 Kernel 提供的 Library 以及相关的 include 文件

23.4.2. Tarball 安装的基本步骤

1、详细步骤说明

- **取得源文件**：将 tarball 文件在/usr/local/src 目录下解压
- **取得步骤流程**：进入新建立的目录下面，去查阅 INSTALL 与 README 等相关文件内容
- **相关属相软件安装**：根据 INSTALL/README 的内容查看并安装好一些相关软件(非必要，已经有了就不用再装了)
- **建立 makefile**：已自动检测程序(configure/config)检测操作环境，并建立 Makefile 这个文件
- **编译**：以 make 这个程序并使用该目录下的 makefile 作为它的参数配置文件，来进行 make 的操作
- **安装**：以 make 这个程序，并以 makefile 这个参数配置文件，根据 **install 这个目标**的指定来安装到正确的路径
- makefile 里面的目标最常见的就是 clean 与 install

2、通用命令步骤

- tar -xv -f [*.tar] -C [欲解压的目录]
- tar -jxv -f [*.tar.bz2] -C [欲解压的目录]
- cd [欲解压的目录]
- **./configure**
- make clean <==一般没什么用，清除上一次遗留的临时文件
- **make**
- **make install**
- make clean <==清除本次安装的临时文件

23.4.3. 一般 Tarball 软件安装的建议事项(如何删除、升级)

1、默认情况下，Linux distribution 发布安装的软件大多是在/usr 里面的

2、用户自行安装的软件**建议**放在/usr/local 里面，方便管理

3、几乎每个软件都会提供在线帮助的服务，那就是 info 与 man 的功能

- man 会去搜索/usr/local/man 里面的说明文件
- 如果我们将软件安装在/usr/local 下面的话，那么自然安装完成后，软件的说明文件就会被找到

4、因此我们建议将自己安装的软件放置在/usr/local 下，置于源码就放在/usr/local/src 里面(src 为 source 的缩写)

5、软件内容放置

- 默认的安装软件路径：
  - /etc
  - /usr/lib
  - /usr/bin
  - /usr/share/man
  - 大致上是摆放在 etc，lib，bin，man 等目录中，分别代表配置文件，

函数库，可执行文件，在线帮助文档

- 默认放在/usr/local 里面
  - /usr/local/etc
  - /usr/local/bin
  - /usr/local/lib
  - /usr/local/man
  - 所有的软件都会放在上述四个目录，想要删除或升级的时候比较麻烦因为难以查找，大家都混在一起了
- 选择单独的目录，以 appache 为例：/usr/local/appche
  - /usr/local/appche/etc
  - /usr/local/appche/bin
  - /usr/local/appche/lib
  - /usr/local/appche/man
  - 如此一来，删除软件就非常简单，只要删除该目录/usr/local/appche 即可**(当然实际的安装还得根据 makefile 里面的 install 信息才能知道，也就是说未必全在该目录中)**
  - 这个方式虽然有利于软件的删除，但是该命令的执行却需要绝对路径因为原先的 PATH 无法定位到该路径，或者更改 PATH
  - **如何指定要安装的目录呢?**
    - ./configure **--prefix=/usr/local/appache**
    - 如果没有--prefix=/usr/local/appache，那么默认为/usr/local

23.4.4. 一个简单的范例(利用 ntp 来示范)

23.4.5. 利用 patch 更新源码

1、利用 diff 与 patch 命令，命令参数详见文件比较工具

23.5. 函数库

1、在 Linux 中，函数库是很重要的一个项目，因为很多软件之间都会互相使用彼此提供的函数库来进行特殊功能的运行，例如很多需要验证身份的程序都习惯用 PAM 这个模块提供的验证机制来实作，很多网络联机机制则习惯利用 SSL 函数库来进行联机加密的机制

23.5.1. 动态与静态函数库

1、静态函数库特色

- **扩展名(.a)**：通常为 libxxx.a
- **编译行为**：这些函数库在编译的时候会直接整合到执行程序当中，所以利用静态函数库编译成的文件会比较大一些
- **独立的执行状态**：这类函数库最大的优点就是编译成功的可执行文件可以独立执行，而不需要再想外部要求读取函数库的内容
- **升级难易度**：虽然可执行文件可以独立执行，但是因为函数库是直接整合到可执行文件中，因此若函数库升级时，整个可执行文件必须要重新编译才能将新版的函数库整合到程序当中，也就是说，在升级方面，只要函数库升级了，所有需将此升级的函数库纳入的程序都需要重新编译

2、动态函数库特色

- **扩展名(.so)**：通常为 libxxx.so
- **编译行为**：动态函数库在编译的时候，在程序里面只有一个"指向"的位置而已，也就是说动态函数库的内容并没有被整合到可执行文件里面，而是当可执行文件要使用到函数库的机制时，程序才会去读取函数库来使用，由于并不包含函数库，所以文件会较小一点
- **独立执行的状态**：这类型的函数库所编译出来的程序不能被独立执行，因为当我们使用到函数库的机制时，程序才会去读取函数库，所以函数库文件必须要存在才行，而且，函数库的所在目录也不能改变，因为可执行文件里面仅有"指标"，因此动态函数库的路径不能随意移动或删除
- **升级难易度**：当函数库升级后，可执行文件根本不需要重新编译，因为可执行文件会直接指向新的函数库文件(前提是新旧函数库文件名相同)

3、目前的 Linux distribution 比较倾向于使用动态函数库

4、函数库放置位置

- /usr/lib
- /lib
- /lib/modules　<==kernel 的函数库

23.5.2. ldconfig 与/etc/ld.so.conf(l 是 L 的小写)

1、内存的访问速度是硬盘的好几倍，如果我们将常用的动态函数库优先加载内存当中(缓存，cache)，如此一来，软件要使用动态函数库时，就不需要从硬盘里面读出，可以增加动态函数库的读取速度，需要 ldconfig 与/etc/ld.so.conf 的协助

2、如何将动态函数库加载到内存

- 首先，必须在/etc/ld.so.conf 里面写下想要读入高速缓存当中的动态函数库所在的**目录，注意是目录，不是文件**
- 接下来利用 ldconfig 这个可执行文件将/etc/ld.so.conf 的数据读入缓存中
- 同时也将数据记录一份在/etc/ld.so.cache 文件中

3、<ldconfig>

- ldconfig [-f conf] [-C cache]
- ldconfig [-p]
- -p conf：conf 指的是文件名，也就是说，使用 conf 作为 library 函数库的取得路径，而不以/etc/ld.so.conf 为默认值
- -C cache：cache 指的是某个文件名，也就是说，使用 cache 作为缓存暂存的函数库资料，而
- 以/etc/ld.so.cache 为默认值
- -p：列出目前的所有函数库数据内容(/etc/ld.so.cache 内的数据)

23.5.3. 动态函数库解析：ldd

1、<ldd>

- 判断某个可执行的二进制文件含有什么动态函数库
- ldd [-vdr] [filename]
- -v：列出所有内容信息
- -d：将数据有丢失的 link 点显示出来

- -r：将 ELF 有关的错误内容显示出来

23.6. 检验软件的正确性

1、下载的软件可能本身就有问题

2、我们可以通过每个文件独特的指纹验证数据

- 每个文件的内容与文件大小不同，所以如果一个文件被修改之后，必然会有部分信息不一样

3、<md5sum><sha1sum>

- md5sum/sha1sum [-bct] [文件名]
- md5sum/sha1sum [--status|--warn] --check [文件名]
- -b：使用二进制的读取方式，默认为 Window/DOS 文件类型的读取方式
- -c：检验文件指纹
- -t：以文本类型来读取指纹

4、用 md5sum 与 sha1sum 来计算指纹，与网上提供的值进行对比，来进行检验

Chapter 24. 软件安装：RPM、SRPM 与 YUM 功能

24.1. 软件管理器简介

1、厂商可以在它们的系统上面编译好我们用户所需要的软件，然后将这个编译好的可执行软件直接发布给用户来安装，类似于 Windows 的安装方式

2、如果在安装的时候添加一些与这些程序相关的信息，将它建成数据库，就可以进行安装，反安装，升级与验证等相关功能(类似 Windows 下面的添加安装程序)

3、在 Linux 上至少有两种常见的软件管理器：RPM 与 Debian 的 dpkg

4、CentOS 主要以 RPM 为主

24.1.1. Linux 界的量大主流：RPM 与 DPKG

1、客户端取得这个文件后，只要通过特定的命令来安装，那么该软件文件就会按照内部的脚本来检测相关的前驱软件是否存在，若安装的环境符合需求，那就会开始安装

2、dpkg

- 这个机制最早由 Debian Linux 社区所开发出来的
- 只要派生于 Debian 的 Linux distribution 大多使用 dpkg 这个机制来安装软件，包括 B2D,Ubuntu 等

3、RPM

- 这个机制最早是由 Red Hat 社区开发出来的，很多 distribution 就使用这个机制来作为软件安装的管理方式，包括 Fedora，CentOS，SuSE

4、无论 dpkg 还是 rpm，这些机制或多或少都会有软件属性依赖的问题，我们将依赖属性的数据做成列表

5、目前新的 Linux 开发商都提供在线升级机制，通过这个机制，原版光盘只有在第一次安装时需要用到而已，其他时候只要有网络，你就能取得原本开发生所提供的任何软件。

distribution	软件管理机制	使用命令	在线升级机制(命令)
Red Hat/Fedora	RPM	rpm,rpmbuild	YUM(yum)
Debian/Ubuntu	DPKG	dpkg	APT(apt-get)

6、CentOS 系统使用的软件管理机制是 RPM 机制，用来作为在线升级的方式则是 yum

24.1.2. 什么是 RPM 与 SRPM

1、RPM 全名是"RedHat Package Manager"，简称 RPM

2、RPM 最大的特点就是将你要安装的软件先编译过，并且打包称为 RPM 机制的安装包，通过包装好的软件里头默认的数据库记录这个软件要安装的时候必须具备的依赖属性软件，当安装在你 Linux 主机时，RPM 会先依照软件里头的数据查询 Linux 主机的依赖属性软件是否满足，若满足则予以安装，若不满足则不予安装。安装的时候讲该软件的信息整个写入 RPM 的数据库中，以便未来的查询、验证与反安装

3、优点

- 由于已经编译完成并且打包完毕，所有软件传输与安装上很方便(不需要重新再编译)

- 由于软件的信息都已经记录在 Linux 主机的数据库上，很方便查询、升级与反安装

4、困扰

- 由于 RPM 文件是已经包装好的数据(编译完成)，**该软件几乎只能安装在原本默认的硬件与操作系统版本中**
- 通常不同的 distribution 所发布的 RPM 文件并不能用在其他 distribution 上
- 相同的 distribution 的不同版本之间也无法互通，例如 CentOS4.x 的 RPM 文件就无法直接应用在 CentOS5.x

5、要求

- 软件安装的环境必须与打包时的环境需求一致或相当
- 需要满足软件的依赖属性需求
- 反安装时需要特别小心，最底层的软件不可先删除，否则可能造成整个系统的问题

6、SRPM，即 Source RPM，这个 RPM 文件里面含有源代码，这个 SRPM 所提供的软件内容并没有经过编译，它提供的是源代码

- 通常 SRPM 的扩展名为***.src.rpm
- 虽然 SRPM 虽然内容是源代码，但是它仍然含有该软件所需要的依赖软件说明以及所有 RPM 文件所提供的数据
- 与 RPM 不同的是，SRPM 提供了参数设置文件，即 configure 与 makefile

7、使用 SRPM 安装的步骤

- 先将该软件以 RPM 管理的方式编译，此时 SRPM 会被编译成 RPM 文件
- 然后将编译完成的 RPM 文件安装到 Linux 系统当中

8、总结

- RPM 文件必须要在相同的 Linux 环境下才能安装
- SRPM 是源代码的格式，可以通过修改 SRPM 内的参数设置文件，然后重新编译生成能适合我们 Linux 环境的 RPM 文件

文件格式	文件名格式	直接安装与否	内含程序类型	可否修改参数并编译
RPM	xxx.rpm	可	已编译	不可
SRPM	xxx.src.rpm	不可	未编译的源代码	可

24.1.3. 什么是 i386、i586、i686、noarch、x86_64

1、软件安装包文件名的解析

rp-pppoe - 3.1 - 5 .i386 .rpm

软件名称 软件的版本信息 发布次数 适合的硬件平台 扩展名

- 软件名称：就是软件的名称咯
- 版本信息：分为主版本与次版本，以上面为例，主板本就是 3，主版本下改动部分源码内容而释出一个新的版本，就是次版本，以上面为例就是 1
- 发布版本次数：编译的次数
  - 为什么要重复编译：可能有 bug 或者安全上的顾虑
- 操作硬件平台：由于 RPM 可以适用在不同的操作平台上，但是不同平台的设置参数是有所区别的，我们可以针对较高级的 CPU 来进行优化参数的设置，这样才能够使用高级别 CPU 带来的硬件加速功能

平台名称	适合平台说明

i386	几乎适用于所有 x86 平台，不论是旧的 pentum 或者是新的 Inter Core2 与 K8 系列的 CPU 都能正常工作，i 指的是 Inter 兼容 CPU 的意思，386 就是 CPU 级别
i586	针对 586 级别设计的计算机进行优化编译，包括 P-I MMX CPU 以及 AMD K5 K6 系列的 CPU 等
i686	在 P- Ⅱ 以后的 Intel 系列的 CPU 及 K7 以后级别的 CPU 都属于 686 级别
x86_64	针对 64 位 CPU 进行优化编译设置，包括 Intel Core2 以上级别的 CPU，以及 AMD 的 Athlon64 以后级别的 CPU，都属于这一类型的硬件平台
noarch	没有任何硬件等级上的限制，一般来说这种类型的 RPM 文件里面应该没有二进制程序，较常见出现的就是属于 shell script 方面的软件

-

24.1.4. RPM 的优点

1、RPM 内含已经编译过的程序与设置文件等数据，可以让用户免除重新编译的困扰

2、RPM 在被安装之前，会先检查系统的硬盘容量，操作系统版本，可避免文件被错误安装

3、RPM 文件本身提供软件版本信息，依赖属性软件名称、软件用途说明、软件所含文件等信息，便于了解软件

4、RPM 管理的方式使用数据库记录 RPM 文件的相关参数，便于升级、删除、查询与验证

24.1.5. RPM 属性依赖的解决方式：YUM 在线升级

1、为了重复利用既有的软件功能，因此很多软件都会以函数库的方式释出部分功能，以方便其他软件的调用应用，例如 PAM 模块的验证功能

2、目前 distribution 在发布软件时，都会将软件的内容分为一般使用与开发使用(development)两大类

- pam-x.x.rpm
- pam-devel-x.x.rpm

3、YUM 机制

- CentOS 先将发布的软件放置到 YUM 服务器内，然后分析这些软件的依赖属性问题，将软件内的信息记录下来
- 然后将这些信息分析后记录成软件相关性的清单列表，这些列表数据与软件所在的位置可以称为容器
- 当客户端有软件安装的需求时，客户端主机会主动向网络上面的 yum 服务器的容器网址下载清单列表，然后通过清单列表的数据与本机 RPM 数据库已存放的软件数据相比较，就能够一口气安装所有需要的具有依赖属性的软件了

4、流程

- 当客户端有升级、安装的需求时，yum 会向容器要求清单的更新，等到清单更新到本机的/var/cache/yum 里面后，等一下更新时就会用这个本机清单与本机的 RPM 数据库进行比较，这样就知道下载什么软件
- 接下来 yum 会跑到容器服务器下载所需要的软件，然后通过 RPM 机制开始安装软件

24.2. RPM 软件管理程序：rpm

24.2.1. RPM 默认安装的路径

1、一般来说，RPM 类型的文件在安装的时候，会先去读取文件内记载的设置参数内容，然后将该数据用来比较 Linux 系统环境，找出是否有属性依赖的软件尚未安装的问题

2、若环境检查合格了，那么 RPM 文件就开始被安装到 Linux 系统上，安装完毕后，该软件相关的信息就会被写入/var/lib/rpm 目录下的数据库文件中，这个目录很重要，未来如果有任何软件升级的需求，版本之间的比较就是来自于这个数据库，而如果你想要查询系统已经安装的软件，也得从这里查

3、文件放置目录表

/etc	一些设置文件放置的目录
/usr/bin	一些可执行文件
/usr/lib	一些程序使用的动态函数库
/usr/share/doc	一些基本的软件使用手册与帮助文档
/usr/share/man	一些 man page 文件

24.2.2. RPM 安装(install)

1、<rpm>

- rpm -ivh package_name
- -i：install
- -v：查看更详细的安装信息画面
- -h：以安装信息栏显示安装进度
- **以下为其他可执行参数**
- --nodeps：
  - 使用时机：当软件发生属性依赖问题而无法安装，但你执意安装
  - 危险性：可能造成该软件无法使用
- --replacefiles：
  - 使用时机：如果在安装过程中出现了"某个文件已经被安装在你的系统上面"的信息，或出现版本不合的信息时，可以使用这个参数来覆盖文件
  - 危险性：覆盖操作是无法复原的
- --replacepkgs：
  - 使用时机：重新安装某个已经安装过的软件
- --force：
  - 使用时机：这个参数其实就是--replacefiles 与--replacepkgs 的综合体
- --test：
  - 使用时机：想要测试一下软件是否可以被安装到用户的 Linux 环境当中，找出是否有属性依赖的问题
  - rpm -ivh pkgname.i386.rpm --test
- --justdb：
  - 使用时机：由于 RPM 数据库损坏或者故障时，可使用这个参数来更新软件在数据库内的信息
- --nosignature：
  - 使用时机：想要略过数字验证检查

- --prefix [新路径]
  - 使用时机：想要将软件安装到其他非正规目录
- --noscripts
  - 使用时机：不想让软件在安装过程中自行执行某些系统命令
- **通常，用好-ivh 参数就足够了**

24.2.3. RPM 升级与更新(upgrade/freshen)

1、以-Uvh 或-Fvh 来升级即可

- -Uvh：如果后面接的软件即使没有安装过，则系统将以直接安装，若后面接的软件有安装过旧版，则系统自动更新至新版
- -Fvh：如果后面接的软件并未安装到你的 Linux 系统上，则该软件不会被安装，亦只有已安装到你 Linux 系统内的软件会被"升级"

24.2.4. RPM 查询(query)

1、<rpm>

- rpm -qa
- rpm -q[licdR] [已安装的软件名称]
- rpm -qf [存在于系统上面的某个文件名]
- rpm -qp[licdR] [未安装的某个文件名称]
- -q：仅查询，后面接的软件名称是否有安装
- -qa：列出所有的已经安装在本机 Linux 系统上面的所有软件名称
- -qi：列出该软件的详细信息(information)，包含开发商、版本与说明等
- -ql：列出该软件所有的文件与目录所在的完整文件名(list)
- -qc：列出该软件的所有设置文件(找出/etc/下面的文件名而已)
- -qd：列出该软件的所有帮助文件(找出与 man 有关的文件而已)
- -qR：列出与该软件有关的依赖软件所含的文件(Required 的意思)
- -qf：由后面接的文件名找出该文件属于哪一个已安装的软件

2、查询主要分为两种：

- 查询已安装到系统上面的软件信息，这部分的信息由/var/lib/rpm/所提供
- 查询 rpm 文件内容，由 RPM 文件内找出一些要写入数据库内的信息，需要使用 rpm -qp(p 是 package 的意思)

3 例子：

- rpm -qpR [*.rpm 文件(包含路径)] ==>查询该 RPM 文件的需求文件
- rpm -ql [软件名(不包含路径)]

4、注意：

- 查询本机上面的 RPM 软件相关信息时，不需要加版本名称，只需要软件名即可，因为它会由/var/lib/rpm 这个数据库里面去查找
- 查询某个 RPM 文件就不同了，需要列出整个完整的文件名，包括后缀，路径等

24.2.5. RPM 验证与数字证书(Verify/Signature)

1、验证(Verify)的功能主要在于提供系统管理员一个有用的管理机制，其作用方式是使用/var/lib/rpm 下面的数据库内容来比较目前 Linux 系统的环境下的所有

软件文件

2、<rpm>

- rpm -Va
- rpm -V [已安装的软件名称]
- rpm -Vp [某个 RPM 文件的文件名]
- rpm -Vf [在系统上面的某个文件]
- -V：后面加的是软件名称，若该软件所含的文件被改动过，才会列出来
- -Va：列出目前系统上所有可能被改动过的文件
- -Vp：后面加的是文件名称，列出该软件内可能被改动过的文件
- -Vf：列出某个文件是否被改动过

3、被改动过的标记，在文件名前会有一个标识符

- **SM5DLUGT** [**cdgir**] filename
- **S**(file Size differs)：文件的容量大小被改变
- **M**(Modediffers)：文件的类型或文件的属性(rwx)被改变
- **5**(MD5)：MD5 这一种指纹码的内容已经不同
- **D**(Device major/minor number mis-match)：设备的主/次代码已经改变
- **L**(readLink(2) path mis-match)：Link 路径已被改变
- **U**(User ownership differs)：文件的所有者已被改变
- **G**(Group ownership differs)：文件的所属用户组已被改变
- **T**(mTime differs)：文件的 modify time 时间已被改变
- **c**(config file)：设置文档
- **d**(documentation)：文档
- **g**(ghost file)："鬼"文件，通常是该文件不被某个软件所包含，较少发生
- **l**(licensefile)：授权文件
- **r**(read me)：自述文件
- 例如 S.5....T. c /etc/logrotate.conf
- **一般来说设置文件改过是很正常的**

4、数字证书<没懂>

- 可以利用 md5 指纹码来检查软件的正确性，详见检验软件的正确性
- 就像自己的签名一样，软件开发商原厂所推出的软件也会有一个厂商自己的证书系统，只是这个证书被数字化了
- 厂商以数字证书系统产生一个专属于该软件的证书，并将该证书的公钥(public key)发布，当你要安装一个 RPM 文件时
  - **首先，必须安装原厂发布的公钥文件，对于每一个软件都有一个公钥文件**
  - **实际安装原厂 RPM 软件时，rpm 命令会去读取 RPM 文件的证书信息，与本机系统内的证书信息比较**
  - **若证书相同则予以安装，若找不到相关证书信息，就会给予警告并且停止安装**
- CentOS 使用的数字证书系统为 GNU 计划的 GnuPG(GNU Privacy Guard,GPG)
  - GPG 可以通过复杂运算算出独一无二的专属金钥系统或者是数字证书系统

- CentOS 的数字证书位于:/etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7

24.2.6. 卸载 RPM 与重建数据库(erase/rebuilddb)

1、卸载就是将软件解除安装，解安装一定要由最上层往下解除

2、<rpm>

- rpm -e [软件名称]
- rpm --rebuilddb <==重建数据库

24.3. SRPM 的使用:rpmbuild

1、找不到该命令，也无法安装

- 解决方法:
- yum provides rpmbuild <==找到包含该命令的软件，然后安装该软件

24.3.1. 利用默认值安装 SRPM 文件(--rebuid/--recoompile)

1、如果下载了 SRPM 文件，又不想修改这个文件的源代码与相关设置，可以直接安装吗？可以，利用 rpmbuild 配合选项即可

2、<rmpbuild>

- --rebuild：这个选项会将后面的 SRPM 进行编译与打包的操作，最终生成 RPM 文件，但是产生的 RPM 文件并没有安装到系统上，当你使用--rebuild 的时候，通常会出现一行字体，即编译后的 RPM 文件路径，这个文件可以用来安装
- --recompile：这个动作会直接编译、打包、并且安装
- 这两个参数都没有修改过 SRPM 内的设置值，仅是通过再次编译来产生 RPM 可安装软件文件而已
- 一般来说，如果编译操作顺利的话，编译过程中产生的临时文件都会被自动删除，如果发生任何错误，则该临时文件会被保留在系统上，等待用户的调试操作

24.3.2. SRPM 使用的路径与需要的软件

1、对于 CentOS，工作目录为/usr/src/redhat/

- /usr/src/redhat/SPECS：这个目录当中放置的是该软件的设置文件
- /usr/src/redhat/SOURCES：这个目录当中放置的是该软件的源文件(*.tar.gz)以及 config 这个设置文件
- /usr/src/redhat/BUILD：编译过程中，有些暂存的数据会放置在这个目录中
- /usr/src/redhat/RPMS：经过编译后，并且顺利编译成功之后，将打包完成的文件放置在这个目录当中，里面包含了 i386,i586,i686,noarch 等次目录
- /usr/src/redhat/SRPMS：与 RPMS 内相似，这里放置的就是 SRPM 封装的文件
- 此外，在编译过程中，可能会发生不明的错误或者是设置的错误，这时候就会在/tmp 下面产生一个相对应的错误文档，可以根据该文档进行除

错工作

2、SRPM 需要重新编译，因此需要 make 与其相关程序，以及 gcc、c 等其他的编译用的程序语言来进行编译

24.3.3. 设置文件的主要内容(*.spec)

1、<未完成>，找不到该目录：/usr/src/redhat/

24.4. YUM 在线升级机制

1、yum 通过分析 RPM 的标题数据后，根据各软件的相关性限制做出属性依赖时的解决方案，然后可以自动处理软件的依赖属性问题，以解决软件安装或删除与升级的问题

2、由于 distribution 必须要先释出软件，然后将软件置于 yum 服务器上面，以提供客户端来要求安装或升级之用，因此想要使用 yum 的功能时，必须要先找到适合的 yum server 才行，每个 yum server 可能会提供许多不同的软件功能，就是"容器"

3、事实上，CentOS 发布软件时已经制作出多部镜像站点(mirror site)提供全世界的软件更新之用，理论上我们不用任何设置，只需要连上 Internet 就可以使用 yum

24.4.1. 利用 yum 进行查询、安装、升级与删除功能

1、<yum>

- 查询功能：yum [list|info|search|provides|whatprovides] 参数
- yum [option] [查询工作项目] [相关参数]
- -y：当 yum 要等待用户输入时，这个选项可以自动提供 yes 的响应
- --installroot=/some/path：将软件安装在指定目录，而不是用默认路径
- search：搜索某个软件名称或者是描述的重要关键字
- list：列出目前 yum 所管理的所有软件名称与版本，类似于 rpm -qa，但是 yum list 列出的是 yum 服务器所支持的所有软件，rpm -qa 列出的是本地已安装的软件
- 例子：
  - yum search raid <==搜索与磁盘阵列(raid)相关的软件
  - yum info mdadm <==找出 mdadm 这个软件的功能为何
  - yum list <==列出 yum 服务器上面提供的所有软件名称
  - yum list updates <==列出服务器上可供本机升级的软件
  - yum provides passwd <==列出提供 passwd 这个文件的软件有哪些
- 安装/升级功能：yum [install|update] [软件]
- yum [option] [查询工作项目] [相关参数]
- install：后面接要安装的软件
- update：后面接要升级的软件，若要整个系统都升级，就直接 update
- 删除功能：yum [remove] [软件]
- yum remove [软件名称]

24.4.2. yum 的设置文件

1、虽然 yum 是你主机能够联网就能直接用的，不过由于 CentOS 的镜像站点可能会选错，比如，我们在北京，但是 CentOS 的镜像站点却选到了台湾或者日本，此时网速是非常慢的

2、配置文件/etc/yum.repos.d/CentOS-Base.repo

- [base]：容器名字，中括号一定要存在，名字可以随便取，但是不能有两个相同名字的容器，否则 yum 不知道去哪里找
- name：只是说明一下这个容器的意义，重要性不高
- mirrorlist：列出这个容器可以使用额镜像站点
- baseurl：最重要，因为后面接的就是容器的实际网址，mirrorlist 是由 yum 程序自行去找镜像站点，baseurl 则是指定固定的一个容器网址
- enable=1：让容器启动，不启动为 0
- gpgcheck=1：指定是否需要查阅 RPM 文件内的数字证书
- pgpkey：数字证书公钥文件所在位置，使用默认值即可

3、<yum>

- yum repolist all <==列出目前 yum server 所使用的容器有哪些
- yum clean [package|headers|all]
- packages：将已下载的软件文件删除
- headers：将下载的软件文件头删除
- all：将所有容器数据都删除

24.4.3. yum 的软件组功能

1、如要想要安装一个大型项目，可以通过 yum 软件组功能

2、<yum>

- yum [组功能] [软件组]
- grouplist：列出所有可使用的组列表，例如 Development Tools 之类的
- groupinfo：后面接 group name，则可了解该 group 内含的所有组名称
- groupinstall：可以安装一整组软件
- groupremove：删除某个组
- 例:
  - yum grouplist <==查看目前容器与本机上面的可用于安装过的软件组有哪些
  - yum groupinstall "Development Tools"
  - yum groupinstall "KDE Plasma Workspaces"

24.4.4. 全系统自动升级

1、<yum>

- yum -y update
- 可以配合 crond 例行性工作
  - vim /etc/crontab
  - 0 3 * * * root /usr/bin/yum -y update
  - -y 参数很重要，否则会一直卡在询问是否继续安装更新的地方

24.5. 管理的选择：RPM 还是 Tarball

1、优先选择原厂的 RPM 功能
2、选择软件官方网站发布的 RPM 或者是提供的容器网址
3、利用 Tarball 安装特殊软件：某些特殊的软件并不会特别帮你制作 RPM 文件
4、用 Tarball 测试新版软件

Chapter 25. X Window设置介绍

1、在 Linux 上头的图形界面称为 X Window System，简称为 X 或 X11

25.1. 什么是X Window System

1、UNIX Like 操作系统不是只是能进行服务器的架设而已，在排版，制图，多媒体应用上也是有需求的，这些需求都要用到图形界面的操作

2、X 的由来：X 在 W(indow)的后面，因此人们就戏称这一版的窗口界面为 X

3、X 窗口系统是能够跨网络与跨操作系统平台的

25.1.1. X Window的发展历史

1、X Window 系统最早是由 MIT 在 1984 年发展出来的，开发者希望这个窗口界面不要与硬件有强烈的相关性，如果与硬件相关性高就等于一个操作系统了，如此一来应用性会受限

2、总结

- 在 UNIX Like 上面的图形界面(GUI)被称为 X 或 X11
- X11 是一个软件而不是一个操作系统
- X11 是利用网络架构来进行图形界面的运行与绘制
- 较著名的 X 版本为 X11R6 版本，目前大部分的 X 都由这一版演化而来
- 现在大部分 distribution 使用的 X 都是由 Xorg 基金会所提供的 X11 软件
- X11 使用的是 MIT 授权，为类似于 GPL 的自由软件授权方式

25.1.2. 主要组件:X Server/X Client/Window Manager/Display Manager

1、组件功能：

- X Server：硬件管理、屏幕绘制与提供字体功能
  - 包括键盘、鼠标、手写板、显示器、屏幕分辨率、色彩深度、显卡、与显示的字体等
  - 每部客户端主机都需要安装 X Server，而服务器端则是提供 X Client 软件，以提供客户端绘图所需要的数据
- X Client：负责 X Server 要求的"事件"的处理
  - X Client 最重要的工作就是处理来自 X Server 的操作，产生绘图的数据，因此也称呼 X Client 为 X Application(X 应用程序)
  - 每个 X Client 并不知道其他 X Client 的存在，因此如果两个以上 X Client 同时存在，两者并不知道对方传了什么数据给 X Server，因此 X Client 的绘图经常会相互重叠而产生困扰
  - X Client 并不需要知道 X Server 的硬件配备与操作系统
- X Window Manager：特殊的 X Client，负责管理所有的 X Client 软件
  - 提供许多的控制元素，包括任务栏，背景桌面的设置等
  - 管理虚拟桌面
  - 提供窗口控制参数，包括窗口的大小，窗口的重叠显示，窗口的移动窗口的最小化等
  - 常见的窗口管理器全名：
    - GNOME
    - KDE

- twm
- XFCE

- Display Manager：提供登陆需求

25.1.3. X Window 的启动流程

1、要启动 X Window System，必须要先启动管理硬件与绘图的 X Server，然后再加载 X Client

2、基本上，目前都是用 Window Manager 来管理窗口界面风格的

3、**在文字界面启动 X：通过命令 startx**

- Linux 是多人多任务的操作系统，因此 X 窗口也可以根据不同的用户而有不同的设置
- startx 是一个 shell script
- startx 最重要的任务就是找出用户或者是系统默认的 X Server 与 X Client 的设置文件，而用户也能够使用 startx 外界参数来取代设置文件的内容
- startx [X client 参数] -- [X server 参数]
- start 后面接的两个参数以两个减号"--"隔开，前面是 X Client 的设置，后面是 X Server 的设置

4、start 找到的设置值可用顺序

- X Server 参数：
  - 使用 startx 后接的参数
  - 若无参数，则查找用户根目录的文件，即~/.xserverrc
  - 若无上述两者，则使用/etc/X11/xinit/xserverrc
  - 若无上述三者，则单纯执行/usr/bin/X
- X Client 参数：
  - 使用 startx 后面接的参数
  - 若无参数，则寻找用户主文件夹的文件，即~/.xinitrc
  - 若无上述两者，则使用/etc/X11/xinit/xinitrc
  - 若无上述三者，则单纯执行 sterm

5、由 startx 调用执行的 xinit

- xinit [client option] -- [server or display option]
- client option 与 server option 由 startx 去找出来
- 通过 startx 找到适当的 xinitrc 与 xserverrc 后，交给 xinit
- 为什么要通过 startx?因为我们必须取得一些参数，而 startx 可以帮助我们快速找到这些参数而不必手动输入
- 若单纯执行 xinit，则系统默认的 X Client 与 X Server 是这样的：
  - xinit xterm -geometry +1+1 -n login -display :0 -- X :0
  - xterm 是 X 窗口下面的终端虚拟机，后面接的参数是这儿终端机的位置与登陆与否
  - 最后接一个-display :0 表示这个虚拟机启动在第":0"号的 X 显示界面
  - 我们的 Linux 可以同时启动多个 X，第一个 X 的界面是:0，即是 tty7，第二个 X 则是:1，即 tty8

6、X 启动的端口

- 在文字界面启动 X 时，直接使用 startx 来找到 X Server 与 X Client 的参数或设置文件，然后再调用 xinit 来启动 X 窗口系统
- xinit 现在入 X server 到默认的:0 这个显示接口(默认在 tty7)，然后再加载 X client 到这个 X 显示接口上，而 X client 通常就是 GNOME 或 KDE
- 既然 X 是而已跨网络的，那 X 启动的端口是几号呢
  - CentOS 考虑 X 窗口是在本机上面运行的，因此将端口改为插槽文件(socket)，事实上，X Server 应该要启动一个 port 6000 来与 X Client 进行沟通的

X 窗口系统	显示接口号码	默认终端机	网络监听接口
第一个 X	hostname:0	tty7	port 6000
第二个 X	hostname:1	tty8	port 6001

  - 因为主机上面的 X 可能有多个同时存在，当我们在启动 X Server/Client 时，应该要注明该 X Server/Client 主要提供或接受来自哪个 display 的 port number 才行

25.1.4. X 启动流程测试

1、<未完成>：CentOS7 下命令不起作用

25.1.5. 我是否需要启动 X Window System

1、如果 Linux 主机定位为网络服务器的话，那么由于 Linux 里面的主要服务的设置文件都是纯文本的格式文件，相当容易设置，不需要 X Window 存在

2、如果 Linux 主机作为台式机来用，那么 X Window 是相当重要的

3、考虑的因素

- 稳定性：X Window 仅为 Linux 上面的一个软件，虽然目前 X Window 已经整合地相当好了，但是任何设计都会有 bug，X 也不例外，因此系统稳定性可能会下降
- 性能：启用了 X 造成一些系统资源的损耗，可能会降低性能

25.2. X Server 设置文件解析与设置

1、基本上，X Server 的设置文件都默认放在/etc/X11 目录下，而相关的显示模块或总模块主要放在/usr/lib/xorg/modules(CentOS6.5 找不到)下面，比较重要的是字体文件与芯片组，放置位置为：

- 提供的屏幕字体:/usr/share/X11/fonts/
- 显卡的芯片组:/usr/lib/xorg/modules/drivers

2、<未完成>，找不到提到的那些配置文件

25.3. 显卡驱动程序安装范例

<未完成>

Chapter 26. Linux 备份策略

26.1. 备份要点

1、备份是个很重要的工作

26.1.1. 备份资料的考虑

1、备份是系统损毁时等待救援的救星，需要重新安装系统时，备份的好坏会影响到系统恢复的进度

2、系统有可能由于不预期的伤害而导致系统发生错误

- 系统可能因为
- 预期的硬件损坏而导致系统崩溃

3、造成系统损毁的问题

- 硬件问题
- 软件问题
  - 用户操作不当
  - 安全攻击事件

4、主机角色不同，备份任务也不同

5、备份因素考虑

- 备份哪些文件
- 选择什么备份的设备
- 考虑备份的方式
- 备份的频率
- 备份使用的工具

26.1.2. 备份哪些 Linux 数据

1、一般来说备份最重要的文件，而不是整体系统

2、有必要备份的文件，可以粗分为两类

- 一类是系统基本设置信息
- 一类则是类似网络服务的内容数据

3、操作系统本身需要备份的文件

- /etc
- /home
- /var/spool/mail
- /boot
- /root
- /usr/local 或/opt(如果你自行安装过其他的套件)

4、网络服务的数据方面

- 软件本身的配置文件：例如/etc 整个目录，/usr/local 整个目录
- 软件服务提供的数据，以 WWW 以及 MySQL 为例
  - WWW 数据：/var/www /srv/www 以及用户的主文件夹
  - MySQL：/var/lib/mysql
- 其他在 Linux 主机上面提供服务的数据库文件

5、根据 3 与 4，推荐要备份的目录

- /boot
- /etc

- /home
- /root
- /usr/local(或者/opt 以及/srv 等)
- /var(其中的临时目录可不备份)

6、不需要备份的目录

- /dev：备份与否均可
- /proc：真的不需要备份
- /mnt 与/media：没有在这个目录内放置你自己系统的东西，则不需备份
- /tmp：临时文件，不需备份

26.1.3. 选择备份设备

1、远程备份系统

- 将你的数据备份到其他地方去，比如将 A 地某主机的数据备份到 B 地某主机上
- 缺点就是带宽要求较大，因此仅选择备份最重要的数据

2、存储设备的考虑

- 备份速度要求：硬盘
- 存储容量：磁带
- 经费与数据可靠性：DVD(可保存 10 年左右)

3、设备代号

- 光驱：/dev/cdrom(/dev/sdX 或/dev/hdX)
- 磁带机：/dev/st0,/dev/ht0
- 软盘驱动器：/dev/fd0,/dev/fd1
- 硬盘：/dev/hd[a-d][1-16](IDE),/dev/sd[a-p][1-16]
- 移动硬盘：/dev/lp[0-2]

26.2. 备份的种类、频率与工具的选择

1、备份的方式

- 增量备份
- 差异备份

26.2.1. 完整备份的增量备份

1、还原的考虑：

- 若是完整备份，若硬件出现问题导致系统损毁时，只要将完整备份拿出来，这个给他倒回硬盘去，所有事情就搞定了
- 有时候(例如使用 dd 命令)甚至连系统都不需要重新安装
- 很多企业用来提供重要服务的主机都会使用完整备份，若提供的服务真的非常重要时，甚至会架设一部一模一样的机器，这样一来，若是原本的机器出问题，就立刻将备份的机器拿出来接管，使企业的网络服务不会中断
- **完整备份就是将根目录'/'整个系统全部备份下来的意思，在某些场合下，完整备份也可以是备份一个文件系统，比如/dev/sda1，/dev/md0 等**

2、增量备份的原则

- 系统用的越久，数据量就会越大，如此一来，完整备份所需要花费的时间与存储设备的使用就会相当麻烦，所以完整备份并不会也不太可能每天进行
- 增量备份：
- 系统在进行完第一次完整备份之后，经过一段时间的运行，比较系统与备份文件之间的差异，仅备份有差异的文件而已
- 比如
  - 星期一做了完整备份，星期二到星期四都做了增量备份，那么星期五的数据则是星期一的完整备份加星期二的增量备份加星期三的增量备份加星期四的增量备份
  - 因此还原起来比较麻烦，必须要还原星期一的完整备份，然后还原星期二的增量备份，再依序还原星期三星期四...
- 增量备份使用的备份软件
  - dd，cpio，dump，restore 等
  - dd：可以直接读取磁盘的扇区而不理会文件系统，是相当良好的备份工具，缺点就是慢很多
  - cpio：能够备份所有文件名，不过得要配合 find 或其他找文件名的命令才能处理妥当
  - dd 与 cpio 可以完整备份
  - dump：可以直接进行增量备份

26. 2. 2. 完整备份的差异备份

1、差异备份与增量备份有点类似，也是需要进行第一次的完整备份后才能进行，只是**差异备份的每次备份都是与原始的完整备份比较的结果，**因此系统运行得越久，离完整备份的时间越长，那么该次差异备份的数据可能就会越大

2、差异备份常用的工具与增量备份差不多，因为都需要完整备份

- 可以通过 rsync 来进行镜像备份
- rsync -av [源目录] [目标目录]

3、差异备份所使用的磁盘容量可能会比增量备份来的大，但是差异备份的还原比较快，只需要还原完整备份与最近一次的差异备份即可

26. 2. 3. 关键数据备份

1、如果仅备份关键数据，那么由于系统的绝大部分可执行文件都可以后来重新安装，因此若你的系统不是因为硬件问题，而是因为软件问题而导致系统被攻破或损毁时，直接获取最新的 Linux distribution，然后重新安装，然后再将系统数据(如账号/密码/主文件夹)与服务数据一个个填回去，那你的系统就还原了

2、**备份关键数据最麻烦的地方就是在还原**，上述的还原方式是你必须要很熟悉系统的运行，否则还原要花费很多时间，这就是仅备份关键数据的缺点

3、**可以使用 tar，配合 date，利用"``"或"$()"来自动生成包含日期的文件名**

- tar -jpcvf mysql.`data +%Y-%m-%d`.tar.bz2 /var/lib/mysql

26. 3. 鸟哥的备份策略

1、主机硬件：使用一个独立的文件系统来存储备份数据，此文件系统挂载到/backup 当中

2、每日进行：目前仅备份 MySQL 数据库
3、每周进行：包括/home，/var，/etc，/boot，/usr/local 等目录与特殊服务的目录
4、自动处理：这方面利用/etc/crontab 来自动提供备份的进行
5、远程备份：每月定期将数据分别刻录到光盘上面和使用网络传输到另一台机器上面

Chapter 27. Linux 内核编译与管理

27.1. 编译前的任务：认识内核与取得内核源代码

1、内核是整个操作系统的最底层，它负责了整个硬件的驱动以及提供各种系统所需的内核功能，包括防火墙机制，是否支持 LVM 或 Quota 等文件系统。

27.1.1. 什么是内核(kernel)

1、kernel：

- 计算机真正在工作的东西其实是"硬件"，而控制这些硬件的就是内核
- 要想让计算机进行的工作，内核必须要提供支持
- 内核就是系统上面一个文件而已，这个文件包含了驱动主机各项硬件的检测程序与驱动模块
- 当系统读完 BIOS 并加载 MBR 内的引导装载程序后，就能加载内核到内存当中，然后内核开始检测硬件，挂载根目录并取得内核模块来驱动所有的硬件，之后调用/sbin/init 就能够依序启动所有系统所需要的服务了
- 内核文件通常被放置在/boot/vmlinuz 中
- 一台主机上面可以拥有多个内核文件，只是开机的时候仅能选择一个来加载而已

2、内核模块的用途：

- 由于硬件更新速度太快，如果内核比较旧，那么换了新的硬件后这个内核可能就无法支持了
- 模块化设置：将一些不常用的类似驱动程序的内容独立出内核，编译成为模块，然后，内核可以在系统正常运行的过程当中加载这个模块到内核的支持，如此一来，我们在不需要改动内核的前提下，只要编译出适当的内核模块并加载它，就行了
- **模块放置位置：/lib/modules/$(uname -r)/kernel/当中**

3、内核编译：

- 内核其实是一个文件，这个文件是通过源代码编译而成的，内核是直接被读入到内存当中的，所以要将它编译成系统可以认识的数据才行
- 我们必须得到内核的源代码，然后利用 22 章 Tarball 安装方式提到的编译概念来实现内核的编译才行

4、关于驱动程序：

- 驱动程序开发的工作上面来说，应该是属于硬件发展厂商的问题

27.1.2. 更新内核的目的

1、内核是操作系统中最早被加载到内存中的，它包含了所有可以让硬件与软件工作的信息

2、内核编译的重点在于你要你的 Linux 做什么，那些没有必要的工作，干脆不要加在你的内核当中了，免得内核太庞大

3、Linux 内核特色与默认内核对终端用户的角色

- Linux 内核有几个主要的特色：
  - 可以随时随各人洗好而改动
  - 版本改动次数很频繁
- 除非你有特殊的需求，否则一次编译成功就可以，不需要随时保持最新

的内核版本，而且也没必要(编译一次要很久)

- 对于普通用户，由于系统已经将内核编译得相当适合普通用户了，因此一般入门的用户基本上不太需要编译内核

4、内核编译的可能目的

- 新功能的需求
- 内核太过臃肿
- 与硬件搭配的稳定性
- 其他需求
- 如果系统已经运行很久了，而且也没有什么大问题，加上你又不增加冷萌的硬件，那么建议不需要编译内核，因为重新编译内核的最主要目的就是想让系统变得更稳定

5、由于内核的主要工作是控制硬件，所以编译内核之前，先了解一下硬件配备与这台主机将用于做什么，内核越简单越好，所以只要将这台主机想要的功能编进去即可

27. 1. 3. 内核的版本

1、主要的版本定义为：[主][次].[发布]-[修改]的样式

2、大多数使用那种偶数的内核版本(2.6.x 是稳定版本，而 2.5.x 是测试版本)

3、2.6.x 与 2.4.x 是两个具有相当大差异的内核版本，两者之间使用到的函数库基本上已经不相同了，因此如果是 2.4.xx 就升级到 2.4.xx 的最新版，而不是是 2.6.xx 的版本

4、2.4.xx 与 2.6.xx 比较中，并不是 2.6.xx 就一定比 2.4.xx 更新，两种版本都在进行维护和升级中

27. 1. 4. 内核源代码的取得方式

1、根据 distribution 去挑选的内核源代码来源主要有：

- 原本 distribution 提供的内核源代码文件
  - 各主要 distribution 推出产品时，其实已经都附上内核源代码了
  - 利用原本的 distribution 提供的源代码，我们可以轻易了解到各项设置参数的参数值，因此编译难度会比较低
- 取得最新的稳定版本内核源代码
  - 如果是站在更新驱动程序的立场来看，使用新的内核会比较好
- 保留原本设置：利用 patch 升级内核源代码
  - 每一次内核发布时，除了发布完整的内核压缩文件之外，还会发布该版本与前一版本的差异性补丁文件，**每个内核的补丁仅有针对前一版本的内核来分析而已**(想要从 2.6.27 升级到 2.6.30，就必须要下载 patch-2.6.28、patch-2.6.29、patch-2.6.30，然后依序进行 patch)
  - 如果想要升级 2.6.30 的修改版本到 2.6.30.3，由于修改版本是针对 2.6.30 来制作的，因此只需要下载 patch-2.6.30.3 来直接将 2.6.30 提升到 2.6.30.3 即可(如果要从 2.6.30.2 升级到 2.6.30.3，若没有对应的补丁，那么只能还原到 2.6.30，在用 patch-2.6.30.3 来升级)

27.1.5. 内核源代码的解压缩/安装/观察

1、内核源代码下的此目录

- arch：与硬件平台有关的选项，大部分指的是 CPU 的类型，例如 x86,x86_64,Xen 虚拟机等
- block：与区块设备较相关的设置数据，区块数据通常指的是大量存储媒介，包括类似 ext3 等文件系统的支持是否允许等
- crypto：内核支持的加密技术，如 md5 或者是 des
- Documentation：与内核有关的一堆帮助文档
- drivers：一些硬件的驱动程序，例如显卡、网卡、PCI 相关硬件等
- firmware：一些旧式硬件的微指令(固件)数据
- fs：内核所支持的文件系统，例如 vfat，reiserfs，nfs 等
- include：一些可让其他过程调用的头(header)定义数据
- init
- ipc：定义 Linux 操作系统内各程序的通信
- kernel：定义内核的程序、内核状态、线程、程序的调度、进程的信号灯
- mm：与内存单元有关的各项数据，包括 swap 与虚拟内存等
- net：与网络有关的各项协议数据，还有防火墙模块等
- security：包括 SELinux 等在内的安全性设置
- sound：与音效有关的各项模块
- virt：与虚拟化机器有关的信息，目前内核支持的是 KVM(Kernel base Virtual Machine)

27.2. 内核编译前的处理与内核功能选择

27.2.1. 硬件环境查看与内核功能要求

1、</proc/cpuinfo>和<lspci>

27.2.2. 保持干净源代码：make mrproper

1、make mrproper：这个操作会将你以前进行过的内核功能选择文件也删除，几乎只有第一次执行内核编译才进行这个操作

2、make clean：仅删除类似目标文件之类的编译过程生成的中间文件，而不会删除设置文件

27.2.3. 开始挑选内核功能：make XXconfig

1、在/boot/目录下存在一个名为 config-xxx 的文件，这个文件就是内核功能列表文件

2、<未完成>

Chapter 28. init 系统简介

1、近年来，Linux 系统的 init 进程经历了两次重大的演进，传统的 sysvinit 已经淡出历史舞台，新的 init 系统 UpStart 和 systemd 各有特点，而越来越多的 Linux 发行版采纳了 systemd。本节简要介绍了这三种 init 系统的使用和原理，每个 Linux 系统管理员和系统软件开发者都应该了解它们，以便更好地管理系统和开发应用。

28.1. 什么是 init 系统, init 系统的历史和现状

1、Linux 操作系统的启动首先从 BIOS 开始，接下来进入 boot loader，由 bootloader 载入内核，进行内核初始化。内核初始化的最后一步就是启动 pid 为 1 的 init 进程。这个进程是系统的第一个进程。它负责产生其他所有用户进程。

2、init 以守护进程方式存在，是所有其他进程的祖先。init 进程非常独特，能够完成其他进程无法完成的任务。

3、Init 系统能够定义、管理和控制 init 进程的行为。它负责组织和运行许多独立的或相关的始化工作(因此被称为 init 系统)，从而让计算机系统进入某种用户预订的运行模式。

4、仅仅将内核运行起来是毫无实际用途的，必须由 init 系统将系统代入可操作状态。比如启动外壳 shell 后，便有了人机交互，这样就可以让计算机执行一些预订程序完成有实际意义的任务。或者启动 X 图形系统以便提供更佳的人机界面，更加高效的完成任务。这里，字符界面的 shell 或者 X 系统都是一种预设的运行模式。

5、大多数 Linux 发行版的 init 系统是和 System V 相兼容的，被称为 sysvinit。这是人们最熟悉的 init 系统。一些发行版如 Slackware 采用的是 BSD 风格 init 系统，这种风格使用较少，本文不再涉及。其他的发行版如 Gentoo 是自己定制的。Ubuntu 和 RHEL 采用 upstart 替代了传统的 sysvinit。而 Fedora 从版本 15 开始使用了一个被称为 systemd 的新 init 系统。

6、可以看到不同的发行版采用了不同的 init 实现，本系列文章就是打算讲述三个主要的 init 系统：sysvinit，UpStart 和 systemd。了解它们各自的设计特点，并简要介绍它们的使用。

7、在 Linux 主要应用于服务器和 PC 机的时代，SysVinit 运行非常良好，概念简单清晰。它主要依赖于 Shell 脚本，这就决定了它的最大弱点：启动太慢。在很少重新启动的 Server 上，这个缺点并不重要。而当 Linux 被应用到移动终端设备的时候，启动慢就成了一个大问题。为了更快地启动，人们开始改进 sysvinit，先后出现了 upstart 和 systemd 这两个主要的新一代 init 系统。Upstart 已经开发了 8 年多，在不少系统中已经替换 sysvinit。Systemd 出现较晚，但发展更快，大有取代 upstart 的趋势。

9、**setup 和 ntsysv 工具还是保留了，但是功能已大大减弱，以前 ntsysv 工具可以控制所有系统服务的自启动，现在只能控制少部分服务**

28.2. sysvinit

1、sysvinit 就是 system V 风格的 init 系统，顾名思义，它源于 System V 系列 UNIX。它提供了比 BSD 风格 init 系统更高的灵活性。是已经风行了几十年的 UNIX init 系统，一直被各类 Linux 发行版所采用

28.2.1. 运行级别

1、Sysvinit 检查/etc/inittab 文件中是否含有'initdefault'项。这告诉 init 系统是否有一个默认运行模式。如果没有默认的运行模式，那么用户将进入系统控制台，手动决定进入何种运行模式。

- 0：关机
- 1：单用户模式，往往用于系统故障之后的排错和恢复
- 2：多用户状态(没有 NFS)
- 3：字符界面的 shell
- 4：系统未使用，保留
- 5：GUI 模式
- 6：重启
- S：往往用于系统故障之后的排错和恢复

28.2.2. sysvinit 运行顺序

1、首先，sysvinit 需要读取/etc/inittab 文件。分析这个文件的内容，它获得以下一些配置信息：

- **系统需要进入的 runlevel**
- 捕获组合件的定义
- 定义电源 fail/restore 脚本
- 启动 getty 和虚拟控制台

2、得到配置信息后，sysvinit 顺序地执行以下这些步骤，从而将系统初始化为预订的 runlevel X(X 为运行级别，即 0123456S)

- /etc/rc.d/rc.sysinit
- /etc/rc.d/rc 和/etc/rc.d/rcX.d/ (X 代表运行级别 0-6)
- /etc/rc.d/rc.local
- X Display Manager(如果需要的话)

3、首先，运行 rc.sysinit 以便执行一些重要的系统初始化任务。在 RedHat 公司的 RHEL5 中(RHEL6 已经使用 UpStart 了)，rc.sysinit 主要完成以下工作：

- 激活 udev 和 selinux
- 设置定义在/etc/sysctl.conf 中的内核参数
- 设置系统时钟
- 加载 keymaps
- 使能交换分区
- 设置主机名(hostname)
- 根分区检查和 remount
- 激活 RAID 和 LVM 设备
- 开启磁盘配额
- 检查并挂载所有文件系统
- 清除过期的 locks 和 PID 文件

4、完成了以上这些工作之后，sysvinit 开始运行/etc/rc.d/rc 脚本。根据不同的 runlevel，rc 脚本将打开对应该 runlevel 的 rcX.d 目录(X 就是 runlevel)，找到并运行存放在该目录下的所有启动脚本。每个 runlevel X 都有一个这样的目录，目录名为/etc/rc.d/rcX.d。

5、在这些目录下存放着很多不同的脚本。文件名以 S 开头的脚本就是启动时应该运行的脚本，S 后面跟的数字定义了这些脚本的执行顺序。在/etc/rc.d/rcX.d 目录下的脚本其实都是一些软链接文件，真实的脚本文件存放在/etc/init.d 目录下

6、当所有的初始化脚本执行完毕。Sysvinit 运行/etc/rc.d/rc.local 脚本，rc.local 是 Linux 留给用户进行个性化设置的地方。您可以把自己私人想设置和启动的东西放到这里，一台 Linux Server 的用户一般不止一个，所以才有这样的考虑。

28.2.3. sysvinit 和系统关闭

1、Sysvinit 不仅需要负责初始化系统，还需要负责关闭系统。在系统关闭时，为了保证数据的一致性，需要小心地按顺序进行结束和清理工作。

2、比如应该先停止对文件系统有读写操作的服务，然后再 umount 文件系统。否则数据就会丢失。

3、这种顺序的控制这也是依靠/etc/rc.d/rcX.d/目录下所有脚本的命名规则来控制的，在该目录下所有以 K 开头的脚本都将在关闭系统时调用，字母 K 之后的数字定义了它们的执行顺序。

4、这些脚本负责安全地停止服务或者其他的关闭工作。

28.2.4. Sysvinit 的管理和控制功能

1、原始的 sysvinit 软件包包含了一系列的控制启动，运行和关闭所有其他程序的工具，如下：

- halt：停止系统
- init：这个就是 sysvinit 本身的 init 进程实体，以 pid1 身份运行，**是所有用户进程的父进程。最主要的作用是在启动过程中使用/etc/inittab 文件创建进程**
- killall5：就是 SystemV 的 killall 命令。向除自己的会话(session)进程之外的其它进程发出信号，所以不能杀死当前使用的 shell
- last：回溯/var/log/wtmp 文件(或者-f 选项指定的文件)，显示自从这个文件建立以来，所有用户的登录情况
- lastb：作用和 last 差不多，默认情况下使用/var/log/btmp 文件，显示所有失败登录企图
- mesg：控制其它用户对用户终端的访问
- pidof：找出程序的进程识别号(pid)，输出到标准输出设备
- poweroff：等于 shutdown -h –p，或者 telinit 0。关闭系统并切断电源
- reboot 等于 shutdown –r 或者 telinit 6。重启系统。
- runlevel：读取系统的登录记录文件(一般是/var/run/utmp)把以前和当前的系统运行级输出到标准输出设备。
- shutdown：以一种安全的方式终止系统，所有正在登录的用户都会收到系统将要终止通知，并且不准新的登录。
- sulogin：当系统进入单用户模式时，被 init 调用。当接收到启动加载程序传递的-b 选项时，init 也会调用 sulogin。
- telinit：实际是 init 的一个连接，用来向 init 传送单字符参数和信号。
- utmpdump：以一种用户友好的格式向标准输出设备显示/var/run/utmp

文件的内容。

- wall：向所有有信息权限的登录用户发送消息。

2、不同的 Linux 发行版在这些 sysvinit 的基本工具基础上又开发了一些辅助工具用来简化 init 系统的管理工作。比如 RedHat 的 RHEL 在 sysvinit 的基础上开发了 initscripts 软件包，包含了大量的启动脚本(如 rc.sysinit)，还提供了 service，chkconfig 等命令行工具，甚至一套图形化界面来管理 init 系统。其他的 Linux 发行版也有各自的 initscript 或其他名字的 init 软件包来简化 sysvinit 的管理。

28.2.5. Sysvinit 的小结

1、**Sysvinit 的优点是概念简单**。Service 开发人员只需要编写启动和停止脚本，概念非常清楚；将 service 添加/删除到某个 runlevel 时，只需要执行一些创建/删除软连接文件的基本操作；这些都不需要学习额外的知识或特殊的定义语法(UpStart 和 Systemd 都需要用户学习新的定义系统初始化行为的语言)。

2、**其次，sysvinit 的另一个重要优点是确定的执行顺序**：脚本严格按照启动数字的大小顺序执行，一个执行完毕再执行下一个，这非常有益于错误排查。UpStart 和 systemd 支持并发启动，导致没有人可以确定地了解具体的启动顺序，排错不易。

3、**但是串行地执行脚本导致 sysvinit 运行效率较慢**，在新的 IT 环境下，启动快慢成为一个重要问题。此外动态设备加载等 Linux 新特性也暴露出 sysvinit 设计的一些问题。针对这些问题，人们开始想办法改进 sysvinit，以便加快启动时间，并解决 sysvinit 自身的设计问题。

28.3. UpStart

28.4. Systemd

28.4.1. Systemd 的简介

1、Systemd 是 Linux 系统中最新的初始化系统(init)，它主要的设计目标是克服 sysvinit 固有的缺点，提高系统的启动速度。systemd 和 ubuntu 的 upstart 是竞争对手，预计会取代 UpStart，实际上在作者写作本文时，已经有消息称 Ubuntu 也将采用 systemd 作为其标准的系统初始化系统。

28.4.2. Systemd 的特点

- Systemd 的很多概念来源于苹果 Mac OS 操作系统上的 launchd，不过 launchd 专用于苹果系统，因此长期未能获得应有的广泛关注。Systemd 借鉴了很多 launchd 的思想，它的重要特性如下：

1、**同 SysVinit 和 LSB init scripts 兼容**

- Systemd 是一个"新来的"，Linux 上的很多应用程序并没有来得及为它做相应的改变。和 UpStart 一样，systemd 引入了新的配置方式，对应用程序的开发也有一些新的要求。如果 systemd 想替代目前正在运行的初始化系统，就必须和现有程序兼容。任何一个 Linux 发行版都很难为了采用 systemd 而在短时间内将所有的服务代码都修改一遍。
- Systemd 提供了和 Sysvinit 以及 LSB initscripts 兼容的特性。系统中已经存在的服务和进程无需修改。这降低了系统向 systemd 迁移的成本，使得 systemd 替换现有初始化系统成为可能。

2、**更快的启动速度**

- Systemd 提供了比 UpStart 更激进的并行启动能力，采用了 socket / D-Bus activation 等技术启动服务。一个显而易见的结果就是：更快的启动速度。
- 为了减少系统启动时间，systemd 的目标是：
  - 尽可能启动更少的进程
  - 尽可能将更多进程并行启动
- 同样地，UpStart 也试图实现这两个目标。UpStart 采用事件驱动机制，服务可以暂不启动，当需要的时候才通过事件触发其启动，这符合第一个设计目标；此外，不相干的服务可以并行启动，这也实现了第二个目标
- **Systemd 能够更进一步提高并发性，即便对于那些 UpStart 认为存在相互依赖而必须串行的服务**，比如 Avahi 和 D-Bus 也可以并发启动
- **Systemd 所有的任务都同时并发执行，总的启动时间被进一步降低**

3、**Systemd 提供按需启动能力**

- 当 sysvinit 系统初始化的时候，**它会将所有可能用到的后台服务进程全部启动运行。并且系统必须等待所有的服务都启动就绪之后，才允许用户登录。这种做法有两个缺点：首先是启动时间过长；其次是系统资源浪费**
- 某些服务很可能在很长一段时间内，甚至整个服务器运行期间都没有被使用过。比如 CUPS，打印服务在多数服务器上很少被真正使用到。您可能没有想到，在很多服务器上 SSHD 也是很少被真正访问到的。花费在启动这些服务上的时间是不必要的；同样，花费在这些服务上的系统资源也是一种浪费
- Systemd 可以提供按需启动的能力，**只有在某个服务被真正请求的时候才启动它。当该服务结束，systemd 可以关闭它，等待下次需要时再次启动它**

4、**Systemd 采用 Linux 的 Cgroup 特性跟踪和管理进程的生命周期**

- init 系统的一个重要职责就是负责跟踪和管理服务进程的生命周期。它不仅可以启动一个服务，也必须也能够停止服务。这看上去没有什么特别的，然而在真正用代码实现的时候，您或许会发现停止服务比一开始想的要困难。
- 服务进程一般都会作为精灵进程(daemon)在后台运行，为此服务程序有时候会派生(fork)两次。在 UpStart 中，需要在配置文件中正确地配置 expect 小节。这样 UpStart 通过对 fork 系统调用进行计数，从而获知真正的精灵进程的 PID 号，如下
- **P1--(fork)-->P1`--(fork)-->P1``**
- 如果 UpStart 找错了，将 p1`作为服务进程的 Pid，那么停止服务的时候，UpStart 会试图杀死 p1`进程，而真正的 p1``进程则继续执行。换句话说该服务就失去控制了。
- 还有更加特殊的情况。比如，一个 CGI 程序会派生两次，从而脱离了和 Apache 的父子关系。当 Apache 进程被停止后，该 CGI 程序还在继续运行。而我们希望服务停止后，所有由它所启动的相关进程也被停止。
- 为了处理这类问题，UpStart 通过 strace 来跟踪 fork、exit 等系统调用，

但是这种方法很笨拙，且缺乏可扩展性。systemd 则利用了 Linux 内核的特性即 CGroup 来完成跟踪的任务。当停止服务时，通过查询 CGroup，systemd 可以确保找到所有的相关进程，从而干净地停止服务。

- CGroup 已经出现了很久，它主要用来实现系统资源配额管理。CGroup 提供了类似文件系统的接口，使用方便。当进程创建子进程时，子进程会继承父进程的 CGroup。因此无论服务如何启动新的子进程，所有的这些相关进程都会属于同一个 CGroup，systemd 只需要简单地遍历指定的 CGroup 即可正确地找到所有的相关进程，将它们一一停止即可。

5、**启动挂载点和自动挂载的管理**

- 传统的 Linux 系统中，用户可以用/etc/fstab 文件来维护固定的文件系统挂载点。这些挂载点在系统启动过程中被自动挂载，一旦启动过程结束，这些挂载点就会确保存在。这些挂载点都是对系统运行至关重要的文件系统，比如 HOME 目录。和 sysvinit 一样，Systemd 管理这些挂载点，以便能够在系统启动时自动挂载它们。Systemd 还兼容/etc/fstab 文件，您可以继续使用该文件管理挂载点。
- 有时候用户还需要动态挂载点，比如打算访问 DVD 内容时，才临时执行挂载以便访问其中的内容，而不访问光盘时该挂载点被取消(umount)，以便节约资源。传统地，人们依赖 autofs 服务来实现这种功能。
- Systemd 内建了自动挂载服务，无需另外安装 autofs 服务，可以直接使用 systemd 提供的自动挂载管理能力来实现 autofs 的功能。

6、**实现事务性依赖关系管理**

- 系统启动过程是由很多的独立工作共同组成的，这些工作之间可能存在依赖关系，比如挂载一个 NFS 文件系统必须依赖网络能够正常工作。Systemd 虽然能够最大限度地并发执行很多有依赖关系的工作，但是类似"挂载 NFS"和"启动网络"这样的工作还是存在天生的先后依赖关系，无法并发执行。对于这些任务，systemd 维护一个"事务一致性"的概念，保证所有相关的服务都可以正常启动而不会出现互相依赖，以至于死锁的情况。

7、**能够对系统进行快照和恢复**

- systemd 支持按需启动，因此系统的运行状态是动态变化的，人们无法准确地知道系统当前运行了哪些服务。Systemd 快照提供了一种将当前系统运行状态保存并恢复的能力。
- 比如系统当前正运行服务 A 和 B，可以用 systemd 命令行对当前系统运行状况创建快照。然后将进程 A 停止，或者做其他的任意的对系统的改变，比如启动新的进程 C。在这些改变之后，运行 systemd 的快照恢复命令，就可立即将系统恢复到快照时刻的状态，即只有服务 A，B 在运行。一个可能的应用场景是调试：比如服务器出现一些异常，为了调试用户将当前状态保存为快照，然后可以进行任意的操作，比如停止服务等等。等调试结束，恢复快照即可。
- 这个快照功能目前在 systemd 中并不完善，似乎开发人员也没有特别关注它，因此有报告指出它还存在一些使用上的问题，使用时尚需慎重。

8、**日志服务**

- **systemd 自带日志服务 journald，该日志服务的设计初衷是克服现有的**

syslog 服务的缺点，比如

- syslog 不安全，消息的内容无法验证。每一个本地进程都可以声称自己是 Apache PID 4711，而 syslog 也就相信并保存到磁盘上。
- 数据没有严格的格式，非常随意。自动化的日志分析器需要分析人类语言字符串来识别消息。一方面此类分析困难低效；此外日志格式的变化会导致分析代码需要更新甚至重写。

➢ **Systemd Journal 用二进制格式保存所有日志信息，用户使用 journalctl 命令来查看日志信息。无需自己编写复杂脆弱的字符串分析处理程序**

➢ **Systemd Journal 的优点如下**

- 简单性：代码少，依赖少，抽象开销最小。
- 零维护：日志是除错和监控系统的核心功能，因此它自己不能再产生问题。举例说，自动管理磁盘空间，避免由于日志的不断产生而将磁盘空间耗尽。
- 移植性：日志文件应该在所有类型的 Linux 系统上可用，无论它使用的何种 CPU 或者字节序。
- 性能：添加和浏览日志非常快。
- 最小资源占用：日志数据文件需要较小。
- 统一化：各种不同的日志存储技术应该统一起来，将所有的可记录事件保存在同一个数据存储中。所以日志内容的全局上下文都会被保存并且可供日后查询。例如一条固件记录后通常会跟随一条内核记录，最终还会有一条用户态记录。重要的是当保存到硬盘上时这三者之间的关系不会丢失。Syslog 将不同的信息保存到不同的文件中，分析的时候很难确定哪些条目是相关的。
- 扩展性：日志的适用范围很广，从嵌入式设备到超级计算机集群都可以满足需求。
- 安全性：日志文件是可以验证的，让无法检测的修改不再可能。

28.4.3. Systemd 的基本概念

1、**单元的概念**

➢ 系统初始化需要做的事情非常多。需要启动后台服务，比如启动 SSHD 服务；需要做配置工作，比如挂载文件系统。**这个过程中的每一步都被 systemd 抽象为一个配置单元，即 unit。**可以认为一个服务是一个配置单元；一个挂载点是一个配置单元；一个交换分区的配置是一个配置单元；等等。systemd 将配置单元归纳为以下一些不同的类型。然而，systemd 正在快速发展，新功能不断增加。所以配置单元类型可能在不久的将来继续增加。

➢ service：代表一个后台服务进程，比如 mysqld。这是最常用的一类。

➢ socket：此类配置单元封装系统和互联网中的一个套接字。当下，systemd 支持流式、数据报和连续包的 AF_INET、AF_INET6、AF_UNIX socket。每一个套接字配置单元都有一个相应的服务配置单元。相应的服务在第一个"连接"进入套接字时就会启动(例如：nscd.socket 在有新连接后便启动 nscd.service)。

- device：此类配置单元封装一个存在于 Linux 设备树中的设备。每一个使用 udev 规则标记的设备都将会在 systemd 中作为一个设备配置单元出现。
- mount：此类配置单元封装文件系统结构层次中的一个挂载点。Systemd 将对这个挂载点进行监控和管理。比如可以在启动时自动将其挂载；可以在某些条件下自动卸载。Systemd 会将/etc/fstab 中的条目都转换为挂载点，并在开机时处理。
- automount：此类配置单元封装系统结构层次中的一个自挂载点。每一个自挂载配置单元对应一个挂载配置单元 ，当该自动挂载点被访问时，systemd 执行挂载点中定义的挂载行为。
- swap: 和挂载配置单元类似，交换配置单元用来管理交换分区。用户可以用交换配置单元来定义系统中的交换分区，可以让这些交换分区在启动时被激活。
- target ：此类配置单元为其他配置单元进行逻辑分组。它们本身实际上并不做什么，只是引用其他配置单元而已。这样便可以对配置单元做一个统一的控制。这样就可以实现大家都已经非常熟悉的运行级别概念。比如想让系统进入图形化模式，需要运行许多服务和配置命令，这些操作都由一个个的配置单元表示，将所有这些配置单元组合为一个目标(target)，就表示需要将这些配置单元全部执行一遍以便进入目标所代表的系统运行状态。(例如：multi-user.target 相当于在传统使用 SysV 的系统中运行级别 5)
- timer：定时器配置单元用来定时触发用户定义的操作，这类配置单元取代了 atd、crond 等传统的定时服务。
- snapshot：与 target 配置单元相似，快照是一组配置单元。它保存了系统当前的运行状态。
- 每个配置单元都有一个对应的配置文件，系统管理员的任务就是编写和维护这些不同的配置文件，比如一个 MySQL 服务对应一个 mysql.service 文件。这种配置文件的语法非常简单，用户不需要再编写和维护复杂的系统 5 脚本了。

2、**依赖关系**

- 虽然 systemd 将大量的启动工作解除了依赖，使得它们可以并发启动。但还是存在有些任务，它们之间存在天生的依赖，不能用"套接字激活"(socket activation)、D-Bus activation 和 autofs 三大方法来解除依赖(三大方法详情见后续描述)。比如：挂载必须等待挂载点在文件系统中被创建；挂载也必须等待相应的物理设备就绪。为了解决这类依赖问题，systemd 的配置单元之间可以彼此定义依赖关系。
- Systemd 用配置单元定义文件中的关键字来描述配置单元之间的依赖关系。比如：unit A 依赖 unit B，可以在 unit B 的定义中用"**require A**"来表示。这样 systemd 就会保证先启动 A 再启动 B。

3、**Systemd 事务**

- Systemd 能保证事务完整性。Systemd 的事务概念和数据库中的有所不同，**主要是为了保证多个依赖的配置单元之间没有环形引用**
- 存在循环依赖，那么 systemd 将无法启动任意一个服务。此时 systemd 将会尝试解决这个问题，因为配置单元之间的依赖关系有两种：required

是强依赖；want 则是弱依赖，systemd 将去掉 wants 关键字指定的依赖看看是否能打破循环。如果无法修复，systemd 会报错。

- Systemd 能够自动检测和修复这类配置错误，极大地减轻了管理员的排错负担。

4、**Target 和运行级别**

- **systemd 用目标(target)替代了运行级别的概念**，提供了更大的灵活性，如您可以继承一个已有的目标，并添加其它服务，来创建自己的目标。下表列举了 systemd 下的目标和常见 runlevel 的对应关系：

运行级别	Systemd 目标	备注
0	runlevel0.target,poweroff.target	关闭系统
1,s,single	runlevel1.target,rescue.target	单用户模式
2,4	runlevel2.target,runlevel4.target, multi-user.target	用户定义/域特定运行级别，默认等同于 3
3	runlevel3.target,multi-user.target	多用户，非图形化，用户可通过多个控制台或网络登录
5	runlevel5.target,graphical.target	多用户，图形化，通常为所有运行级别 3 的服务外加图形化登录
6	runlevel6.target,reboot.target	重启
emergency	emergency.target	紧急 shell

- systemctl isolate runlevel5.target <==init 5

5、**Systemd 的并发启动原理**

- 如前所述，在 Systemd 中，所有的服务都并发启动，比如 Avahi、D-Bus、livirtd、X11、HAL 可以同时启动。乍一看，这似乎有点儿问题，比如 Avahi 需要 syslog 的服务，Avahi 和 syslog 同时启动，假设 Avahi 的启动比较快，所以 syslog 还没有准备好，可是 Avahi 又需要记录日志，这岂不是会出现问题？
- Systemd 的开发人员仔细研究了服务之间相互依赖的本质问题，发现所谓依赖可以分为三个具体的类型，而每一个类型实际上都可以通过相应的技术解除依赖关系。
- **并发启动原理之一：解决 socket 依赖**
  - 绝大多数的服务依赖是套接字依赖。比如服务 A 通过一个套接字端口 S1 提供自己的服务，其他的服务如果需要服务 A，则需要连接 S1。因此如果服务 A 尚未启动，S1 就不存在，其他的服务就会得到启动错误。所以传统地，人们需要先启动服务 A，等待它进入就绪状态，再启动其他需要它的服务。Systemd 认为，只要我们预先把 S1 建立好，那么其他所有的服务就可以同时启动而无需等待服务 A 来创建 S1 了。如果服务 A 尚未启动，那么其他进程向 S1 发送的服务请求实际上会被 Linux 操作系统缓存，其他进程会在这个请求的地方等待。一旦服务 A 启动就绪，就可以立即处理缓存的请求，一切都开始正常运行。
  - 那么服务如何使用由 init 进程创建的套接字呢？
  - Linux 操作系统有一个特性，当进程调用 fork 或者 exec 创建子进程之后，所有在父进程中被打开的文件句柄 (file descriptor) 都被子进程所继承。

套接字也是一种文件句柄，进程 A 可以创建一个套接字，此后当进程 A 调用 exec 启动一个新的子进程时，只要确保该套接字的 close_on_exec 标志位被清空，那么新的子进程就可以继承这个套接字。子进程看到的套接字和父进程创建的套接字是同一个系统套接字，就仿佛这个套接字是子进程自己创建的一样，没有任何区别。

- 这个特性以前被一个叫做 inetd 的系统服务所利用。Inetd 进程会负责监控一些常用套接字端口，比如 Telnet，当该端口有连接请求时，inetd 才启动 telnetd 进程，并把有连接的套接字传递给新的 telnetd 进程进行处理。这样，当系统没有 telnet 客户端连接时，就不需要启动 telnetd 进程。Inetd 可以代理很多的网络服务，这样就可以节约很多的系统负载和内存资源，只有当有真正的连接请求时才启动相应服务，并把套接字传递给相应的服务进程。
- 和 inetd 类似，systemd 是所有其他进程的父进程，它可以先建立所有需要的套接字，然后在调用 exec 的时候将该套接字传递给新的服务进程，而新进程直接使用该套接字进行服务即可。

➢ **并发启动原理之二：解决 D-Bus 依赖**

- D-Bus 是 desktop-bus 的简称，是一个低延迟、低开销、高可用性的进程间通信机制。它越来越多地用于应用程序之间通信，也用于应用程序和操作系统内核之间的通信。很多现代的服务进程都使用 D-Bus 取代套接字作为进程间通信机制，对外提供服务。比如简化 Linux 网络配置的 NetworkManager 服务就使用 D-Bus 和其他的应用程序或者服务进行交互：邮件客户端软件 evolution 可以通过 D-Bus 从 NetworkManager 服务获取网络状态的改变，以便做出相应的处理。
- D-Bus 支持所谓"bus activation"功能。如果服务 A 需要使用服务 B 的 D-Bus 服务，而服务 B 并没有运行，则 D-Bus 可以在服务 A 请求服务 B 的 D-Bus 时自动启动服务 B。而服务 A 发出的请求会被 D-Bus 缓存，服务 A 会等待服务 B 启动就绪。利用这个特性，依赖 D-Bus 的服务就可以实现并行启动。

➢ **并发启动原理之三：解决文件系统依赖**

- 系统启动过程中，文件系统相关的活动是最耗时的，比如挂载文件系统，对文件系统进行磁盘检查(fsck)，磁盘配额检查等都是非常耗时的操作。在等待这些工作完成的同时，系统处于空闲状态。那些想使用文件系统的服务似乎必须等待文件系统初始化完成才可以启动。但是 systemd 发现这种依赖也是可以避免的。
- Systemd 参考了 autofs 的设计思路，使得依赖文件系统的服务和文件系统本身初始化两者可以并发工作。autofs 可以监测到某个文件系统挂载点真正被访问到的时候才触发挂载操作，这是通过内核 automounter 模块的支持而实现的。比如一个 open() 系统调用作用在"/misc/cd/file1"的时候，/misc/cd 尚未执行挂载操作，此时 open()调用被挂起等待，Linux 内核通知 autofs，autofs 执行挂载。这时候，控制权返回给 open()系统调用，并正常打开文件。
- Systemd 集成了 autofs 的实现，对于系统中的挂载点，比如/home，当系统启动的时候，systemd 为其创建一个临时的自动挂载点。在这个时

刻/home 真正的挂载设备尚未启动好，真正的挂载操作还没有执行，文件系统检测也还没有完成。可是那些依赖该目录的进程已经可以并发启动，他们的 open()操作被内建在 systemd 中的 autofs 捕获，将该 open()调用挂起(可中断睡眠状态)。然后等待真正的挂载操作完成，文件系统检测也完成后，systemd 将该自动挂载点替换为真正的挂载点，并让 open()调用返回。由此，实现了那些依赖于文件系统的服务和文件系统本身同时并发启动。

- 当然对于"/"根目录的依赖实际上一定还是要串行执行，因为 systemd 自己也存放在/之下，必须等待系统根目录挂载检查好。
- 不过对于类似/home 等挂载点，这种并发可以提高系统的启动速度，尤其是当/home 是远程的 NFS 节点，或者是加密盘等，需要耗费较长的时间才可以准备就绪的情况下，因为并发启动，这段时间内，系统并不是完全无事可做，而是可以利用这段空余时间做更多的启动进程的事情，总的来说就缩短了系统启动时间。

28.4.4. Systemd 的使用

1、系统软件开发人员：开发人员需要了解 systemd 的更多细节。比如您打算开发一个新的系统服务，就必须了解如何让这个服务能够被 systemd 管理。这需要您注意以下这些要点：

- 后台服务进程代码不需要执行两次派生来实现后台精灵进程，只需要实现服务本身的主循环即可。
- 不要调用 setsid()，交给 systemd 处理
- 不再需要维护 pid 文件。
- Systemd 提供了日志功能，服务进程只需要输出到 stderr 即可，无需使用 syslog。
- 处理信号 SIGTERM，这个信号的唯一正确作用就是停止当前服务，不要做其他的事情。
- SIGHUP 信号的作用是重启服务。
- 需要套接字的服务，不要自己创建套接字，让 systemd 传入套接字。
- 使用 sd_notify()函数通知 systemd 服务自己的状态改变。一般地，当服务初始化结束，进入服务就绪状态时，可以调用它。

2、Unit 文件的编写

- 对于开发者来说，工作量最大的部分应该是编写配置单元文件，定义所需要的单元。
- 举例来说，开发人员开发了一个新的服务程序，比如 httpd，就需要为其编写一个配置单元文件以便该服务可以被 systemd 管理，类似 UpStart 的工作配置文件。在该文件中定义服务启动的命令行语法，以及和其他服务的依赖关系等。
- 此外我们之前已经了解到，systemd 的功能繁多，不仅用来管理服务，还可以管理挂载点，定义定时任务等。这些工作都是由编辑相应的配置单元文件完成的。我在这里给出几个配置单元文件的例子。

```
#cat /etc/system/system/sshd.service
[Unit]
```

Description=OpenSSH server daemon
[Service]
EnvironmentFile=/etc/sysconfig/sshd
ExecStartPre=/usr/sbin/sshd-keygen
ExecStart=/usrsbin/sshd –D $OPTIONS
ExecReload=/bin/kill –HUP $MAINPID
KillMode=process
Restart=on-failure
RestartSec=42s
[Install]
WantedBy=multi-user.target

- 文件分为三个小节。第一个是[Unit]部分，这里仅仅有一个描述信息。第二部分是 Service 定义，其中，ExecStartPre 定义启动服务之前应该运行的命令；ExecStart 定义启动服务的具体命令行语法。第三部分是[Install]，WangtedBy 表明这个服务是在多用户模式下所需要的。

#cat multi-user.target
[Unit]
Description=Multi-User System
Documentation=man.systemd.special(7)
Requires=basic.target
Conflicts=rescue.service rescure.target
After=basic.target rescue.service rescue.target
AllowIsolate=yes
[Install]
Alias=default.target

- 第一部分中的 Requires 定义表明 multi-user.target 启动的时候 basic.target 也必须被启动；另外 basic.target 停止的时候，multi-user.target 也必须停止。如果您接着查看 basic.target 文件，会发现它又指定了 sysinit.target 等其他的单元必须随之启动。同样 sysinit.target 也会包含其他的单元。采用这样的层层链接的结构，最终所有需要支持多用户模式的组件服务都会被初始化启动好。
- 在[Install]小节中有 Alias 定义，即定义本单元的别名，这样在运行 systemctl 的时候就可以使用这个别名来引用本单元。这里的别名是 default.target，比 multi-user.target 要简单一些。。。
- 此外在/etc/systemd/system 目录下还可以看到诸如*.wants 的目录，放在该目录下的配置单元文件等同于在[Unit]小节中的 wants 关键字，即本单元启动时，还需要启动这些单元。比如您可以简单地把您自己写的 foo.service 文件放入 multi-user.target.wants 目录下，这样每次都会被默认启动了。

#cat sys-kernel-debug.mount
[Unit]
Description=Debug File Syste
DefaultDependencies=no
ConditionPathExists=/sys/kernel/debug
Before=sysinit.target

[Mount]
What=debugfs
Where=/sys/kernel/debug
Type=debugfs

- 这个配置单元文件定义了一个挂载点。挂载配置单元文件有一个[Mount]配置小节，里面配置了 What，Where 和 Type 三个数据项。这都是挂载命令所必须的，例子中的配置等同于下面这个挂载命令：
- mount –t debugfs /sys/kernel/debug debugfs
- 配置单元文件的编写需要很多的学习，必须参考 systemd 附带的 man 等文档进行深入学习。希望通过上面几个小例子，大家已经了解配置单元文件的作用和一般写法了。

3、系统管理员

- systemd 的主要命令行工具是<systemctl>

Sysvinit 命令	Systemd 命令	备注
service foo start	systemctl start foo.service	启动一个服务(不会重启现有的)
service foo stop	systemctl stop foo.service	停止一个服务(不会重启现有的)
service foo restart	systemctl restart foo.service	停止并启动一个服务
service foo reload	systemctl reload foo.service	当支持时，重新装载配置文件而不中断等待操作
service foo condrestart	systemctl condrestart foo.service	如果服务正在运行那么重启它
service foo status	systemctl status foo.service	汇报服务是否正在运行
ls /etc/rc.d/init.d/	systemctl list-unit-files --type=service	用来列出可以启动或停止的服务列表
chkconfig foo on	systemctl enable foo.service	下次启动时或满足其他触发条件时设置服务为启动(开机启动)
chkconfig foo off	systemctl disable foo.service	在下次启动时或满足其他触发条件设置服务为仅用(开机不启动)
chkconfig foo	systemctl is-enabled foo.service	用来检查一个服务在当前环境被配置为启用还是禁用
chkconfig --list	systemctl list-unit-files --type=service	输出在各个运行级别下服务的启用和禁用情况
chkconfig foo --list	ls /etc/systemd/system/ *.wants/foo.service	用来列出该服务在哪些运行级别下启用和禁用
chkconfig foo --add	systemctl daemon-reload	创建新服务文件或变更设置时使用
telinit 3	systemctl isolate multi-user.target	改变至多用户运行级别

命令	操作
systemctl reboot	重启机器
systemctl poweroff	关机
systemctl suspend	待机
systemctl hibernate	休眠
systemctl hybrid-sleep	混合休眠模式(同时休眠到硬盘并待机)
systemctl get-default	获取当前默认的 target(runlevel)

systemctl set-default Target.target	设置默认的 target(runlevel)

28.4.5. Systemd 小结

1、systemd 的最大特点有两个

- 令人惊奇的激进的并发启动能力，极大地提高了系统启动速度
- 用 CGroup 统计跟踪子进程，干净可靠

2、此外，和其前任不同的地方在于，systemd 已经不仅仅是一个初始化系统了

3、Systemd 出色地替代了 sysvinit 的所有功能，但它并未就此自满。因为 init 进程是系统所有进程的父进程这样的特殊性，systemd 非常适合提供曾经由其他服务提供的功能，比如定时任务(以前由 crond 完成)会话管理(以前由 ConsoleKit/PolKit 等管理)。仅仅从本文皮毛一样的介绍来看，Systemd 已经管得很多了，可它还在不断发展。它将逐渐成为一个多功能的系统环境，能够处理非常多的系统管理任务，有人甚至将它看作一个操作系统。

4、好的一点是，这非常有助于标准化 Linux 的管理！从前，不同的 Linux 发行版各行其事，使用不同方法管理系统，从来也不会互相妥协。比如如何将系统进入休眠状态，不同的系统有不同的解决方案，即便是同一个 Linux 系统，也存在不同的方法，比如一个有趣的讨论：如何让 Ubuntu 系统休眠，可以使用底层的 /sys/power/state 接口，也可以使用诸如 pm-utility 等高层接口。存在这么多种不同的方法做一件事情对像我这样的普通用户而言可不是件有趣的事情。systemd 提供统一的电源管理命令接口，这件事情的意义就类似全世界的人都说统一的语言，我们再也不需要学习外语了，多么美好！

5、如果所有的 Linux 发行版都采纳了 systemd，那么系统管理任务便可以很大程度上实现标准化。此外 systemd 有个很棒的承诺：接口保持稳定，不会再轻易改动。对于软件开发人员来说，这是多么体贴又让人感动的承诺啊

涉及到但未曾学到的命令：

- depmod -a：加载驱动程序
- lsmod：查看已经加载的驱动程序
- seq n1 n2：输出从 n1 到 n2 的整数，a=$(seq 1 100)

<没懂>
<未完成>

名词：
Linux distribution：Linux 发行版，例如 CentOS，Ubuntu，Fedora 等等