前　言

本书是尚观科技 UEA(嵌入式方向)专用辅导课本。

本书基于 Ubuntu 16.04 LTS(长期支持服务版本)。

本书用于 UEA 方向前期的 linux 基础的教学，并作为学生的辅导练习，本书摘取的常用命令为 UEA 方向常用命令的选项及参数，基本满足 UEA 方向的学习使用。

本书由尚观科技 UEA 方向王毅老师主编，于 2018 年 3 月分完成编辑工作，同期由其他 UEA 方向的老师作校验，并用于尚观科技 UEA 方向内部资料发行，如本书有不尽人意之处，请大家见谅并提供宝贵的意见。谢谢。

目 录

第一章 Linux 系统简介

一、 linux 及其发展

1.1 11969 年，Bell 实验室，根据 Multics 系统加入一些自己的想法利用汇编语言开发了 Unics 系统

1.2 1973 年，由 Ken Tompson 和 Dennis Ritchie 共同用 C 语言实现了 unix 操作系统。

1.3 1977 年，加州伯克利大学的研究小组在获得 Unix 内核后，将其修改为适合自己机器的版本。由此产生了 unix 的重要分支 BSD，他是后来可以安装在 x86 机器上 FreeBSD 的前身。

1.4 1979 年，由于 Unix 良好的移植性和强大的性能，很多公司开始了 Unix 操作系统的开发，AT&T 的 Bell 的 System V ，IBM 的 AIX 等，后来 unix 越发的流行，AT&T 推出了 System V 第七版 Unix，以在个人计算机上安装和运行，但是，它声明了版权，特别提到“不可对学生提供源代码”

1.5 1984 年，芬兰谭宁邦教授的 x86 架构的 Minix 操作系统诞生了

1.6 1984 年，人工智能实验室的理查德 斯托曼创建了 GNU 项目及 FSF 基金会。

1.7 1991 年，芬兰的 linus Trovalds 利用 gcc,bash 等工具写了一个小小的内核程序，这便是 linux 的雏形。后来不断的对内核进行修改完善，最终有了现在的 linux 操作系统。

二、 自由软件之父 Richard M. Stallman(RMS 理查德 斯托曼)

RMS 是自由软件运动的精神领袖，GNU 计划及自由软件基金会 FSF(Free software foundation)的创立者。他为自由软件运动树立了道德，政治及法律框架，被许多人誉为当今自由软件的斗士、伟大的理想主义者。他说 Linux 并不能代表整个操作系统，Linux 只是内核，整个系统还包含数以百计的软件工具和实用程序，这些工具和程序大多是由 GNU 完成的。1983 年，RMS 在新闻发布会上公开发起 GNU 计划，宣布他的目标是创建一套完全自由的操作系统，并附带一份《GNU 宣言》，声称发起该计划的一个重要理由是要重现当年软件接合作互助的团结精神。GNU 其实是“GNU Not UNIX”的缩写。但为了防止不法厂商利用自由软件，使其专有化，RMS 和一群律师起草了广为使用的 GNU 通用公共协议证书(GPL)，创造了 Copyleft 的授权办法，所有的程序都可以被拷贝，修改，出售，但是有一条，就是源代码所有的改进和修改必须向每个用户公开，所有用户都可以获得改动后的源码。它保证了自由软件传播的延续性。目前，尽管他对 GNU 的设想还没有完全实现，但是这个软件系统已经 1000 多个应用程序了

三、 Liux 之父 Linux Torvalds

linux Torvalds 被誉为活着的传奇，它每天的工作都是编程，领导并推动这 Linux 的发展。1991 发布了 linux 内核。 网址：www.kernel.org

查看 Linux 内核版本：Uname -a

2.6.32-573.el6.i686

主版本.次版本.释出版本-修改版本

奇数、偶数版本分类

在 2.6.x 版本以前，托瓦兹将核心的发展趋势分为两股，并根据这两股核心的发展分别给予不同的核心编号，那就是：主次版本的奇偶。

主、次版本为奇数：发展中版本（ development） 如 2.5.xx，这种核心版本主要用在测试与发展、新功能，所以通常这种版本仅有核心开发工程师会使用。如果有新

增的核心程序码，会加到这种版本当中，等到众多工程师测试没问题后，才加入下一版的稳定核心中。

主、次版本为偶数：稳定版本（ stable） 如 2.6.xx，等到核心功能发展成熟后会加到这类的版本中，主要用在一般家用计算机以及企业版本中。 重点在于提供使用者一个相对稳定的 Linux 作业环境平台。

四、 Linux 的吉祥物：企鹅

五、 Linux 的发行版本

- **红帽企业版** Linux（RedHat Enterprise Linux，RHEL）：红帽公司是全球最大的开源技术厂商，RHEL 是全世界内使用最广泛的 Linux 系统。RHEL 系统具有极强的性能与稳定性，并且在全球范围内拥有完善的技术支持。RHEL 系统也是本书、红帽认证以及众多生产环境中使用的系统。
- **社区企业操作系统**（Community Enterprise Operating System，CentOS）：通过把 RHEL 系统重新编译并发布给用户免费使用的 Linux 系统，具有广泛的使用人群。CentOS 当前已被红帽公司“收编”。
- **Fedora**：由红帽公司发布的桌面版系统套件（目前已经不限于桌面版）。用户可免费体验到最新的技术或工具，这些技术或工具在成熟后会被加入到 RHEL 系统中，因此 Fedora 也称为 RHEL 系统的“试验田”。运维人员如果想时刻保持自己的技术领先，就应该多关注此类 Linux 系统的发展变化及新特性，不断改变自己的学习方向。
- **Debian**：稳定性、安全性强，提供了免费的基础支持，可以良好地支持各种硬件架构，以及提供近十万种不同的开源软件，在国外拥有很高的认可度和使用率。
- **Ubuntu**：是一款派生自 Debian 的操作系统，对新款硬件具有极强的兼容能力。Ubuntu 与 Fedora 都是极其出色的 Linux 桌面系统，而且 Ubuntu 也可用于服务器领域。
- **openSUSE**：源自德国的一款著名的 Linux 系统，在全球范围内有着不错的声誉及市场占有率。
- **Gentoo**：具有极高的自定制性，操作复杂，因此适合有经验的人员使用。读者可以在学习完本书后尝试一下该系统。

六、 Linux 应用领域

今天各种场合都有使用各种 Linux 发行版，从嵌入式设备到超级计算机，并且在服务器领域确定了地位，通常服务器使用 LAMP（Linux + Apache + MySQL + PHP）或 LNMP（Linux + Nginx+ MySQL + PHP）组合。

目前 Linux 不仅在家庭与企业中使用，并且在政府中也很受欢迎。

- 巴西联邦政府由于支持 Linux 而世界闻名。
- 有新闻报道俄罗斯军队自己制造的 Linux 发布版的，做为 G.H.ost 项目已经取得成果.
- 印度的 Kerala 联邦计划在向全联邦的高中推广使用 Linux。
- 中华人民共和国为取得技术独立，在龙芯过程中排他性地使用 Linux。
- 在西班牙的一些地区开发了自己的 Linux 发布版，并且在政府与教育领域广泛使用

- Extremadura 地区的 gnuLinEx 和 Andalusia 地区的 Guadalinex。
- 法国和德国同样开始逐步采用 Linux。

七、 Linux vs Windows

目前国内 Linux 更多的是应用于服务器上，而桌面操作系统更多使用的是 Windows。主要区别如下

比较	Windows	Linux
开源	封闭/收费	开源/免费
界面	界面统一，外壳程序固定所有 Windows 程序菜单几乎一致，快捷键也几乎相同	图形界面风格依发布版不同而不同，可能互不兼容。GNU/Linux 的终端机是从 UNIX 传承下来，基本命令和操作方法也几乎一致。
驱动程序	驱动程序丰富，版本更新频繁。默认安装程序里面一般包含有该版本发布时流行的硬件驱动程序，之后所出的新硬件驱动依赖于硬件厂商提供。对于一些老硬件，如果没有了原配的驱动有时很难支持。另外，有时硬件厂商未提供所需版本的 Windows 下的驱动，也会比较头痛。	由志愿者开发，由 Linux 核心开发小组发布，很多硬件厂商基于版权考虑并未提供驱动程序，尽管多数无需手动安装，但是涉及安装则相对复杂，使得新用户面对驱动程序问题（是否存在和安装方法）会一筹莫展。但是在开源开发模式下，许多老硬件尽管在 Windows 下很难支持的也容易找到驱动。HP、Intel、AMD 等硬件厂商逐步不同程度支持开源驱动，问题正在得到缓解。
使用	使用比较简单，容易入门。图形化界面对没有计算机背景知识的用户使用十分有利。	图形界面使用简单，容易入门。文字界面，需要学习才能掌握。
学习	系统构造复杂、变化频繁，且知识、技能淘汰快，深入学习困难。	系统构造简单、稳定，且知识、技能传承性好，深入学习相对容易。
软件	每一种特定功能可能都需要商业软件的支持，需要购买相应的授权。	大部分软件都可以自由获取，同样功能的软件选择较少。

Linux 的优缺点：

优点： 稳定性和高效性：长时间运行（一年）都不宕（dang）机都是正常的事

配置低：数年前的电脑都可以很流畅的运行 linux 系统

安全性：linux 拥有相当庞大的用户和社区支持，很快能发现系统漏洞，并发布安全补丁

缺点： 没有特定的厂商，不会由售后服务类的支持

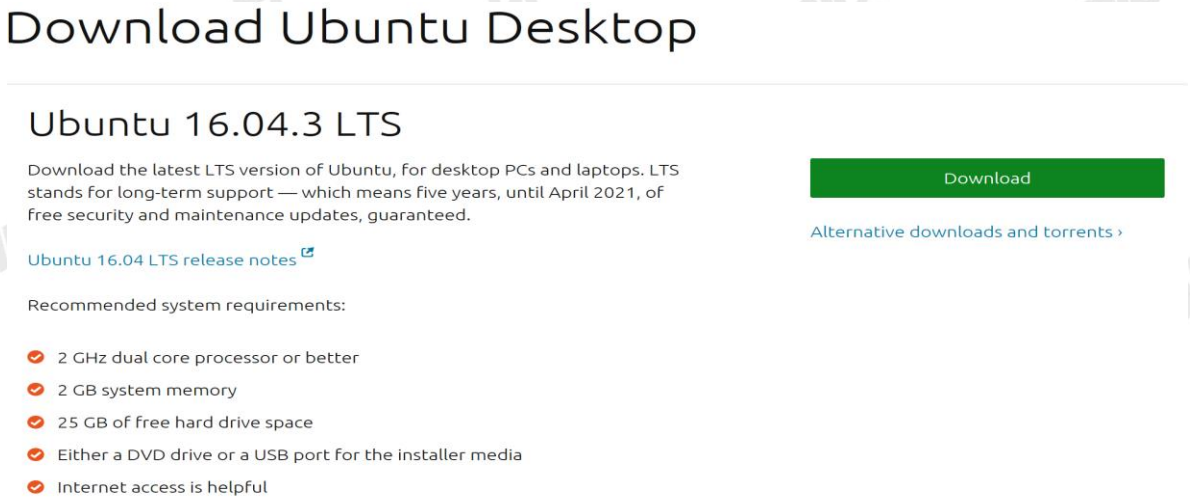
图形界面不够好，和 windows 相差较大

第二章 Linux 安装

一、 Linux 安装

本章节以 Ubuntu 16.04 LTS 桌面版为例为大家介绍 linux 的安装

Ubuntu 16.04 LTS 下载地址 https://www.ubuntu.com/download/desktop



LTS 代表 Long Time Service,长期支持服务版本

二、 Linux 安装步骤

2.1 准备工作

2.1.1 下载 ubuntu-16.04-desktop-amd64.iso

2.1.2 下载 UltraISO 最新版

2.1.3 一个空着的 U 盘（用于作启动盘）

2.2 准备启动盘步骤

2.2.1 下载并安装 UltraISO 软件，安装之后插入 U 盘

2.2.2 打开软件，点击文件打开，找到下载的 Ubuntu 的 ISO 文件，双击打开，完成 ISO 文件的加载

2.2.3 点击启动选项，然后点击写入硬盘映像选项，点击它进入到将要进行操作的界面

2.2.4 如果你插入了 U 盘就可看到

如果你插入多个 U 盘就一定要注意选择自己需要进行操作的 U 盘

2.2.5 格式化

写入方式：USB-HDD+

2.2.6 点击 **写入**,会弹出再次确认是否是你需要写入的 U 盘,点击 **是** 后开始写入,完成了之后我们 就可以使用 U 盘启动盘了

2.2.7 进行 U 盘安装系统之前，还需要设置 BIOS boot 启动选项，默认的是硬盘启动，需要设置为 U 盘启动,不同的主板设置 U 盘启动的方式也不同，在这里就不详述怎么更改 BIOS 设置，大 家查找自己的主板型号然后在网上找相关的设置教程即可

2.2.8 完成 BIOS 设置后就可以插入 U 盘，重启电脑，就可用 U 盘进行 Ubuntu 操作系统的安装了

2.3 (安装前请断掉网络，否则安装时间会大大增加)

2.3.1 安装 Ubuntu 进入安装第一界面

一直等待，等到它自己跳出安装界面后自己从左边的选项里选择中文，选择安装

2.3.2 选择中文→点击继续

2.3.3 “为图形或无线硬件……”（选择或不选择都行）

点击继续

2.3.4 选择“其他选项”

继续

2.3.5 点击“创建新分区表” →继续→选择产生的空闲分区

点击的“+”创建 3 个主要的基础分区

创建 swap 分区：

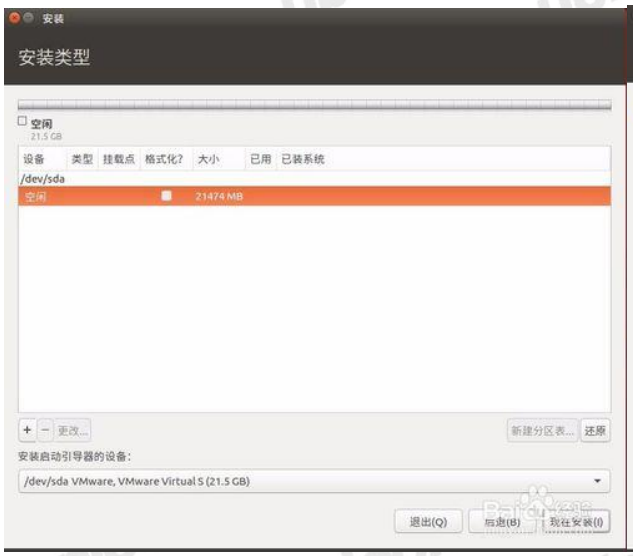
2048MB 逻辑分区 空间起始位置 交换空间

创建 boot 分区：

512MB 逻辑分区 空间起始位置 Ext4 日志文件系统 /boot

创建/分区：

等于全部 主分区 空间起始位置 Ext4 日志文件系统 /



2.3.6 “安装启动引导器的设备”不修改默认

点击“现在安装”→继续→选择时区(Shanghai)→继续→选择键盘布局（使用默认）(汉语)→继续

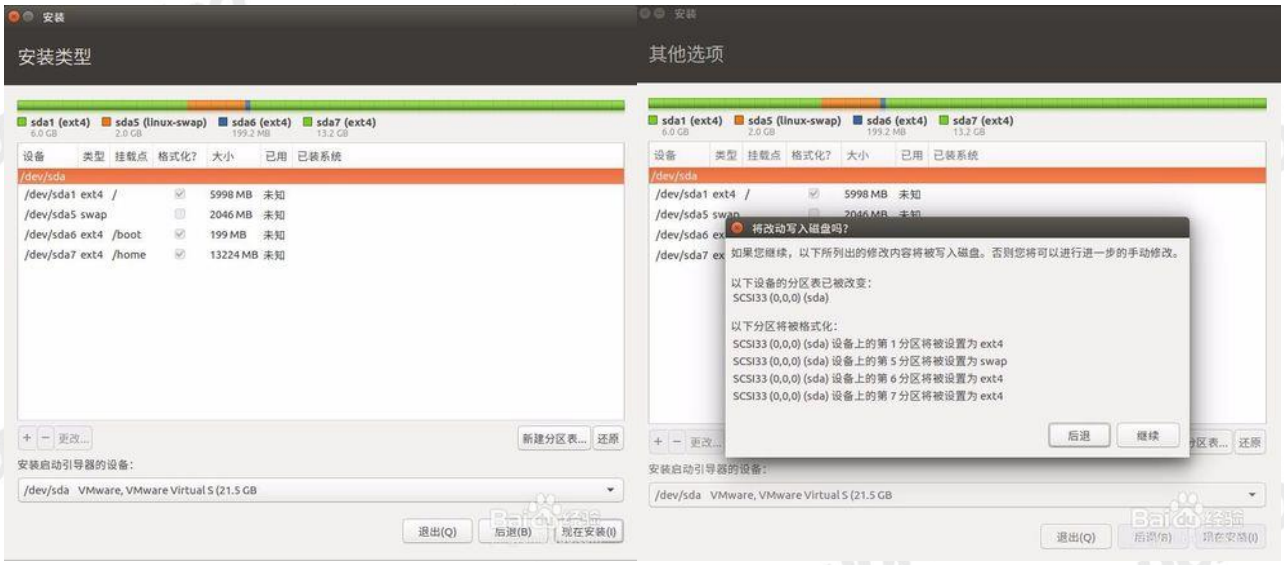
2.3.7 创建用户（计算机名日后可改）

密码要复杂

密码为登录密码请务必牢记

继续

2.3.8 开始安装 等待……


安装
安装类型
sda1 (ext4) sda5 (linux-swap) sda6 (ext4) sda7 (ext4)
设备 类型 挂载点 格式化? 大小 已用 已装系统
/dev/sda
/dev/sda1 ext4 / 5998 MB 未知
/dev/sda5 swap 2046 MB 未知
/dev/sda6 ext4 /boot 199 MB 未知
/dev/sda7 ext4 /home 13224 MB 未知
新建分区表... 还原
安装启动引导器的设备:
/dev/sda VMware, VMware Virtual S (21.5 GB
退出(Q) 后退(B) 现在安装(I)
安装
其他选项
将改动写入磁盘吗?
如果您继续，以下所列出的修改内容将被写入磁盘。否则您将可以进行进一步的手动修改。
以下设备的分区表已被改变:
SCSI33 (0,0,0) (sda)
以下分区将被格式化:
SCSI33 (0,0,0) (sda) 设备上的第 1 分区将被设置为 ext4
SCSI33 (0,0,0) (sda) 设备上的第 5 分区将被设置为 swap
SCSI33 (0,0,0) (sda) 设备上的第 6 分区将被设置为 ext4
SCSI33 (0,0,0) (sda) 设备上的第 7 分区将被设置为 ext4
后退 继续
退出(Q)


安装
您在什么地方?
Shanghai
后退(B) 继续
安装
键盘布局
选择您的键盘布局:
法罗
法语
法语(刚果民主共和国，刚果(金))
法语(加拿大)
菲律宾语
芬兰语
哈萨克语
汉语
荷兰语
黑山语
捷克
柯尔克孜语(吉尔吉斯语)
克罗地亚
拉脱维亚
老挝语(寮语)
汉语
汉语 - Tibetan
汉语 - Tibetan (with ASCII numerals)
汉语 - Uyghur
在这里输入以测试您的键盘
探测键盘布局
后退(B) 继续


安装
您是谁?
您的姓名: UBUNTUA
您的计算机名: ubuntua-virtual-mach
与其他计算机联络时使用的名称。
选择一个用户名: ubuntua
选择一个密码:
确认您的密码:
自动登录
登录时需要密码
加密我的主目录
后退(B) 继续
安装
欢迎使用 Ubuntu
最新版本的 Ubuntu 快速且具有丰富新特性，用起来比以往更方便。这里有一些值得注意的的新玩意.....
正在安装系统

第三章 Linux 操作命令

一、 系统维护命令

1.1 Linux 系统版本信息：

Ubuntu 版本信息： uname -a

1.2 Linux 关机/重启

在 linux 领域内大多用在服务器上，很少遇到关机的操作。毕竟服务器上跑一个服务是永无止境的，除非特殊情况下，不得已才会关机。

正确的关机流程为：sync > shutdown > reboot > halt

关机指令为：shutdown ，你可以 man shutdown 来看一下帮助文档。

例如你可以运行如下命令关机：

sync 将数据由内存同步到硬盘中。

shutdown 关机指令，你可以 man shutdown 来看一下帮助文档。如你可以运行如下命令关机：

shutdown –h 10 ‘This server will shutdown after 10 mins’ 这个命令告诉大家，计算机将在 10 分钟后关机，并且会显示在登陆用户的当前屏幕中。

Shutdown –h now 立马关机

Shutdown –h 20:25 系统会在今天 20:25 关机

Shutdown –h +10 十分钟后关机

Shutdown –r now 系统立马重启

Shutdown –r +10 系统十分钟后重启

reboot 就是重启，等同于 shutdown –r now

halt 关闭系统，等同于 shutdown –h now 和 poweroff

最后总结一下，不管是重启系统还是关闭系统，首先要运行 **sync** 命令，把内存中的数据写到磁盘中。

关机的命令有 shutdown –h now、halt、poweroff 和 init 0 , 重启系统的命令有 shutdown –r now、reboot、init 6。

1.3 history 显示当前终端下输入命令的历史记录

history -c 清空历史记录

注意：方向键 上下建可以翻出前期输入的命令

1.4 whereis 查询当前系统中的程序所在路径

whereis [-bfmsu][-B <目录>...][-M <目录>...][-S <目录>...][文件...]

参数：

-b 只查找二进制文件。

-B<目录> 只在设置的目录下查找二进制文件。

-f 不显示文件名前的路径名称。

例：使用指令"whereis"查看指令"bash"的位置，输入如下命令：

$ whereis bash

上面的指令执行后，输出信息如下所示：

bash:/bin/bash/etc/bash.bashrc/usr/share/man/man1/bash.1.gz

注意：以上输出信息从左至右分别为查询的程序名、bash 路径、bash 的 man 手册页路径。

1.5 which 指令会在环境变量$PATH 设置的目录里查找符合条件的文件

which [文件...]

参数：

-n<文件名长度> 指定文件名长度，指定的长度必须大于或等于所有文件中最长的文件名。

-p<文件名长度> 与-n 参数相同，但此处的<文件名长度>包括了文件的路径。

-w 指定输出时栏位的宽度。

-V 显示版本信息。

例：使用指令"which"查看指令"bash"的绝对路径，输入如下命令：

```
$ which bash
```

上面的指令执行后，输出信息如下所示：

```
/bin/bash          #bash 可执行程序的绝对路径
```

1.6 df 命令用于显示目前在 Linux 系统上的文件系统的磁盘使用情况统计。

df [选项]... [FILE]...

参数：

文件-a, --all 包含所有的具有 0 Blocks 的文件系统

文件-h, --human-readable 使用人类可读的格式(预设值是不加这个选项的...)

例：显示文件系统的磁盘使用情况统计：

```
# df
Filesystem      1K-blocks      Used Available Use% Mounted on
/dev/sda6       29640780   4320704  23814388  16% /
udev             1536756         4   1536752   1% /dev
tmpfs             617620       888    616732   1% /run
none                5120         0      5120   0% /run/lock
none             1544044       156   1543888   1% /run/shm
```

第一列指定文件系统的名称，第二列指定一个特定的文件系统 1K-块 1K 是 1024 字节为单位的总内存。用和可用列正在使用中，分别指定的内存量。使用列指定使用的内存的百分比，而最后一栏"安装在"指定的文件系统的挂载点。

1.7 du 会显示指定的目录或文件所占用的磁盘空间。

du [-abcDhHklmsSx][-L <符号连接>][-X <文件>][--block-size][--exclude=<目录或文件>][--max-depth=<目录层数>][--help][--version][目录或文件]

参数：

-a 或-all 显示目录中个别文件的大小。

-h 或--human-readable 以 K，M，G 为单位，提高信息的可读性。

例：显示目录或者文件所占空间:

```
# du
608     ./test6
308     ./test4
4       ./scf/lib
4       ./scf/service/deploy/product
4       ./scf/service/deploy/info
12      ./scf/service/deploy
16      ./scf/service
```

1240 .

只显示当前目录下面的子目录的目录大小和当前目录的总的大小，最下面的 1288 为当前目录的总大小

显示指定文件所占空间

```
# du  log2012.log
300     log2012.log
```

1.8 clear 命令用于清除屏幕。

等价与快捷方式： ctrl+L

1.9 fdisk 是一个创建和维护分区表的程序，它兼容 DOS 类型的分区表、BSD 或 SUN 类型的磁盘列表。

fdisk [必要参数][选择参数]

必要参数：

-l 列出素所有分区表

-u 与"-l"搭配使用，显示分区数目

选择参数：

-s<分区编号> 指定分区

-v 版本信息

菜单操作说明

m ： 显示菜单和帮助信息

a ： 活动分区标记/引导分区

d ： 删除分区

l ： 显示分区类型

n ： 新建分区

p ： 显示分区信息

q ： 退出不保存

t ： 设置分区号

v ： 进行分区检查

w ： 保存修改

x ： 扩展应用，高级功能

1.10 获取学习命令的方法：

1.10.1 man 帮助手册

部分	Man 手册的类型
1	用户命令
2	内核系统调用（从用户空间到内核的进入点）
3	库函数
4	特殊文件和设备
5	文件格式和规范
6	游戏
7	规范、标准和其他页面
8	系统管理命令
9	Linux 内核 API（内核调用）

1.10.2 命令 –help

1.11 常用快捷方式：

Alt+ Tab 可以在多个不同终端间切换。

Ctrl + w：删除光标位置前的单词

Ctrl + u：清空行

↑，↓方向键：查看命令历史

Tab：自动补全文件名、目录名和命令等等

Ctrl + c：中止当前命令

Ctrl + d：退出登录 Shell

二、 文件管理命令

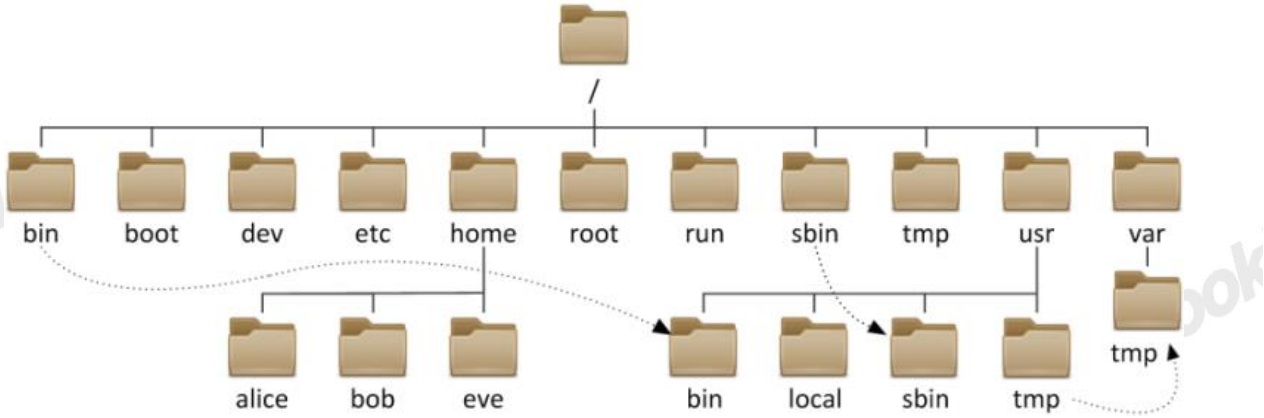
2.1 Linux 系统目录结构

登录系统后，在当前命令窗口下输入命令：

ls /　　你会看到如下图所示:

```
[root@localhost ~]# ls /
bin   dev  home  lost+found  mnt  proc  sbin     srv  tmp  var
boot  etc  lib   media       opt  root  selinux  sys  usr
```

树状目录结构：



以下是对这些目录的解释：

/bin：bin 是 Binary 的缩写，这个目录存放着最经常使用的命令。

/boot：这里存放的是启动 Linux 时使用的一些核心文件，包括一些连接文件以及镜像文件。

/dev ：dev 是 Device(设备)的缩写，该目录下存放的是 Linux 的外部设备，在 Linux 中访问设备的方式和访问文件的方式是相同的。

/etc：这个目录用来存放所有的系统管理所需要的配置文件和子目录。

/home：用户的主目录，在 Linux 中，每个用户都有一个自己的目录，一般该目录名是以用户的账号命名的。

/lib：这个目录里存放着系统最基本的动态连接共享库，其作用类似于 Windows 里的 DLL 文件。几乎所有的应用程序都需要用到这些共享库。

/lost+found：这个目录一般情况下是空的，当系统非法关机后，这里就存放了一些文件。

/media：linux 系统会自动识别一些设备，例如 U 盘、光驱等等，当识别后，linux 会把识别的设备挂载到这个目录下。

/mnt：系统提供该目录是为了让用户临时挂载别的文件系统的，我们可以将光驱挂载在/mnt/上，然后进入该目录就可以查看光驱里的内容了。

/opt：这是给主机额外安装软件所摆放的目录。比如你安装一个 ORACLE 数据库则就可以放到这个目录下。默认是空的。

/proc：这个目录是一个虚拟的目录，它是系统内存的映射，我们可以通过直接访问这个目录来获取系统信息。

/root：该目录为系统管理员，也称作超级权限者的用户主目录。

/sbin：s 就是 Super User 的意思，这里存放的是系统管理员使用的系统管理程序。

/srv：该目录存放一些服务启动之后需要提取的数据。

/tmp：这个目录是用来存放一些临时文件的。

/usr：这是一个非常重要的目录，用户的很多应用程序和文件都放在这个目录下，类似于 windows 下的 program files 目录。

/usr/bin：系统用户使用的应用程序。

/usr/sbin：超级用户使用的比较高级的管理程序和系统守护程序。

/usr/src：内核源代码默认的放置目录。

/var：这个目录中存放着在不断扩充着的东西，我们习惯将那些经常被修改的目录放在这个目录下。包括各种日志文件。

注意：在 linux 系统中，有几个目录是比较重要的，平时需要注意不要误删除或者随意更改内部文件。

/etc： 上边也提到了，这个是系统中的配置文件，如果你更改了该目录下的某个文件可能会导致系统不能启动。

/bin, /sbin, /usr/bin, /usr/sbin: 这是系统预设的执行文件的放置目录，比如 ls 就是在/bin/ls 目录下的。

/var： 这是一个非常重要的目录，系统上跑了很多程序，那么每个程序都会有相应的日志产生，而这些日志就被记录到这个目录下，具体在/var/log 目录下，另外 mail 的预设放置也是在这里。

2.2 ls 命令用于显示指定工作目录下之内容（列出目前工作目录所含之文件及子目录)

ls [-alrtAFR] [name...]

参数 ：

-a 显示所有文件及目录 (ls 内定将文件名或目录名称开头为"."的视为隐藏档，不会列出)

-l 除文件名称外，亦将文件型态、权限、拥有者、文件大小等资讯详细列出

例：列出根目录(\)下的所有目录：

```
# ls /
bin   dev   lib   media  net   root  srv  upload  www
boot  etc   lib64 misc   opt   sbin  sys  usr
home  lost+found  mnt   proc  selinux  tmp  var
```

列出目前工作目录下所有名称是 s 开头的文件，越新的排越后面 ：

```
ls -ltr s*
```

将/bin 目录以下所有目录及文件详细资料列出 ：

```
ls -lR /bin
```

2.3 cd 命令用于切换当前工作目录至 dirName(目录参数)。

其中 dirName 表示法可为绝对路径或相对路径。若目录名称省略，则变换至使用者的 home 目录 (也就是刚 login 时所在的目录)。另外，"~" 也表示为 home 目录 的意思，"." 则是表示目前所在的目录，".." 则表示目前目录位置的上一层目录。

cd [dirName] dirName：要切换的目标目录。

例：跳到 /usr/bin/ : cd /usr/bin

跳到自己的 home 目录 : cd ~

跳到目前目录的上上两层 :cd ../..

跳到的上一次工作的目录 :cd –

2.4 pwd 命令用于显示工作目录。

执行 pwd 指令可立刻得知您目前所在的工作目录的绝对路径名称。

pwd [--help][--version]

参数:

--help 在线帮助。

--version 显示版本信息。

例：查看当前所在目录：

```
# pwd
/root/test                #输出结果
```

2.5 mkdir 命令用于建立名称为 dirName 之子目录。

mkdir [-p] dirName

参数：

-p 确保目录名称存在，不存在的就建一个。

例：在工作目录下，建立一个名为 AAA 的子目录 ：

```
mkdir AAA
```

例：在工作目录下的 BBB 目录中，建立一个名为 Test 的子目录。 若 BBB 目录原本不存在，则建立一个。（注：本例若不加 -p，且原本 BBB 目录不存在，则产生错误。）

```
mkdir -p BBB/Test
```

2.6 touch 命令用于修改文件或者目录的时间属性，包括存取时间和更改时间。若文件不存在，系统会建立一个新的文件。

touch [-acfm][-d<日期时间>][-r<参考文件或目录>] [-t<日期时间>][--help][--version] [文件或目录…]

参数：

a 改变档案的读取时间记录。

m 改变档案的修改时间记录。

c 假如目的档案不存在，不会建立新的档案。与 --no-create 的效果一样。

f 不使用，是为了与其他 unix 系统的相容性而保留。

r 使用参考档的时间记录，与 --file 的效果一样。

d 设定时间与日期，可以使用各种不同的格式。

t 设定档案的时间记录，格式与 date 指令相同。

例:使用指令"touch"修改文件"testfile"的时间属性为当前系统时间，输入如下命令：

```
$ touch testfile                #修改文件的时间属性
```

首先，使用 ls 命令查看 testfile 文件的属性，如下所示：

```
$ ls -l testfile                #查看文件的时间属性
```

原来文件的修改时间为 16:09

```
-rw-r--r-- 1 hdd hdd 55 2011-08-22 16:09 testfile
```

执行指令"touch"修改文件属性以后，并再次查看该文件的时间属性，如下所示：

```
$ touch testfile                #修改文件时间属性为当前系统时间
$ ls -l testfile                #查看文件的时间属性
#修改后文件的时间属性为当前系统时间
-rw-r--r-- 1 hdd hdd 55 2011-08-22 19:53 testfile
```

使用指令"touch"时，如果指定的文件不存在，则将创建一个新的空白文件。例如，在当前目录下，使用该指令创建一个空白文件"file"，输入如下命令：

```
$ touch file                #创建一个名为“file”的新的空白文件
```

2.7 rmdir 命令删除空的目录。

rmdir [-p] dirName

参数：

-p 是当子目录被删除后使它也成为空目录的话，则顺便一并删除。

例:将工作目录下，名为 AAA 的子目录删除 ：

rmdir AAA

例：在工作目录下的 BBB 目录中，删除名为 Test 的子目录。若 Test 删除后，BBB 目录成为空目录，则 BBB 亦予删除。

rmdir -p BBB/Test

2.8 rm 命令用于删除一个文件或者目录。

rm [options] name...

参数：

-i 删除前逐一询问确认。

-f 即使原档案属性设为唯读，亦直接删除，无需逐一确认。

-r 将目录及以下之档案亦逐一删除。

例：删除文件可以直接使用 rm 命令，若删除目录则必须配合选项"-r"，例如：

rm test.txt

rm：是否删除 一般文件 "test.txt"? y

rm homework

rm: 无法删除目录"homework": 是一个目录

rm -r homework

rm：是否删除 目录 "homework"? y

删除当前目录下的所有文件及目录，命令行为：

rm -r *

文件一旦通过 rm 命令删除，则无法恢复，所以必须格外小心地使用该命令。

2.9 cat 命令用于连接文件并打印到标准输出设备上。

cat [-AbeEnstTuv] [--help] [--version] fileName

参数：

-n 或 --number：由 1 开始对所有输出的行数编号。

例：把 textfile1 的文档内容加上行号后输出：

cat -n textfile1

2.10 more 命令类似 cat ，不过会以一页一页的形式显示，更方便使用者逐页阅读。

more [-dlfpcsu] [-num] [+/pattern] [+linenum] [fileNames..]

参数：

-num 一次显示的行数

+num 从第 num 行开始显示

例：从第 20 行开始显示 testfile 之文档内容。

more +20 testfile

常用操作命令

Enter 向下 n 行，需要定义。默认为 1 行

空格键 向下滚动一屏

q 退出 more

2.11 less 与 more 类似，但使用 less 可以随意浏览文件，而 more 仅能向前移动，却不能向后移动，而且 less 在查看之前不会加载整个文件。

less [参数] 文件

参数：

-e 当文件显示结束后，自动离开

-f 强迫打开特殊文件，例如外围设备代号、目录和二进制文件

/字符串：向下搜索"字符串"的功能

?字符串：向上搜索"字符串"的功能
n：重复前一个搜索（与 / 或 ? 有关）
N：反向重复前一个搜索（与 / 或 ? 有关）
Q 退出 less 命令
空格键 滚动一页
回车键 滚动一行
[pagedown]： 向下翻动一页
[pageup]： 向上翻动一页

例:查看文件

less log2013.log

2.12 head 查看文件的前几行，默认是 10 行

head [参数] 文件

参数：

-n 打印多少行

例：查看文件的前 100 行

head -n 100 filename

2.13 tail 查看文件的后几行，默认是 10 行

tail [参数] 文件

参数：

-n 打印多少行

例：查看文件的后 100 行

tail -n 100 filename

2.14 cp 命令主要用于复制文件或目录。

cp [options] source dest 或 cp [options] source... directory

参数：

-a：此选项通常在复制目录时使用，它保留链接、文件属性，并复制目录下的所有内容。其作用等于 dpR 参数组合。

-f：覆盖已经存在的目标文件而不给出提示。

-i：与-f 选项相反，在覆盖目标文件之前给出提示，要求用户确认是否覆盖，回答"y"时目标文件将被覆盖。

-p：除复制文件的内容外，还把修改时间和访问权限也复制到新文件中。

-r：若给出的源文件是一个目录文件，此时将复制该目录下所有的子目录和文件。

例：使用指令"cp"将当前目录"test/"下的所有文件复制到新目录"newtest"下：

$ cp –r test/ newtest

注意：用户使用该指令复制目录时，必须使用参数"-r"或者"-R"。

2.15 mv 命令用来为文件或目录改名、或将文件或目录移入其它位置。

mv [options] source dest 或 mv [options] source... directory

参数：

-i: 若指定目录已有同名文件，则先询问是否覆盖旧文件;

-f: 在 mv 操作要覆盖某已有的目标文件时不给任何指示;

mv 文件名 文件名 将源文件名改为目标文件名

mv 文件名 目录名 将文件移动到目标目录

mv 目录名 目录名 目标目录已存在，将源目录移动到目标目录；目标目录不存在则改名

mv 目录名 文件名 出错

例：将文件 aaa 更名为 bbb :

mv aaa bbb

将 info 目录放入 logs 目录中。注意，如果 logs 目录不存在，则该命令将 info 改名为 logs。

mv info/ logs

2.16 Linux/Unix 的文件调用权限分为三级 : 文件拥有者、群组、其他。利用 chmod 可以藉以控制文件如何被他人所调用。

chmod [-cfvR] [--help] [--version] mode file...

参数

mode : 权限设定字串，格式 :[ugoa...][[+-=][rwxX]...][,...]

其中：u 表示该文件的拥有者，g 表示与该文件的拥有者属于同一个群体(group)者，o 表示其他以外的人，a 表示这三者皆是。+ 表示增加权限、- 表示取消权限、= 表示唯一设定权限。r 表示可读取，w 表示可写入，x 表示可执行，X 表示只有当该文件是个子目录或者该文件已经被设定过为可执行。

其他参数说明：

-R : 对目前目录下的所有文件与子目录进行相同的权限变更(即以递回的方式逐个变更)

例：将文件 file1.txt 设为所有人皆可读取 :

chmod ugo+r file1.txt

将文件 file1.txt 设为所有人皆可读取 :

chmod a+r file1.txt

将文件 file1.txt 与 file2.txt 设为该文件拥有者，与其所属同一个群体者可写入，但其他以外的人则不可写入 :

chmod ug+w,o-w file1.txt file2.txt

将 ex1.py 设定为只有该文件拥有者可以执行 :

chmod u+x ex1.py

将目前目录下的所有文件与子目录皆设为任何人可读取 :

chmod -R a+r *

此外 chmod 也可以用数字来表示权限如 :

chmod 777 file

语法为：chmod abc file

其中 a,b,c 各为一个数字，分别表示 User、Group、及 Other 的权限。r=4，w=2，x=1

若要 rwx 属性则 4+2+1=7；若要 rw-属性则 4+2=6；若要 r-x 属性则 4+1=5。

chmod a=rwx file 和 chmod 777 file 效果相同

chmod ug=rwx,o=x file 和 chmod 771 file 效果相同

若用 chmod 4755 filename 可使此程序具有 root 的权限

2.17 Linux/Unix 是多人多工操作系统，所有的文件皆有拥有者。利用 chown 将指定文件的拥有者改为指定的用户或组，用户可以是用户名或者用户 ID；组可以是组名或者组 ID；文件是以空格分开 的要改变权限的文件列表，支持通配符。

一般来说，这个指令只有是由系统管理者(root)所使用，一般使用者没有权限可以改变别人的文件拥有者，也没有权限可以自己的文件拥有者改设为别人。只有系统管理者(root)才有这样的权限。

使用权限 : root

chown [-cfhvR] [--help] [--version] user[:group] file...

参数 :

user : 新的文件拥有者的使用者 ID

group : 新的文件拥有者的使用者群体(group)

例：将文件 file1.txt 的拥有者设为 users 群体的使用者 jessie :

chown jessie:users file1.txt

将目前目录下的所有文件与子目录的拥有者皆设为 users 群体的使用者 lamport：

chown -R lamport:users *

2.18 chgrp 用于变更文件或目录的所属群组。

在 UNIX 系统家族里，文件或目录权限的掌控以拥有者及所属群组来管理。您可以使用 chgrp 指令去变更文件与目录的所属群组，设置方式采用群组名称或群组识别码皆可。

chgrp [-cfhRv][--help][--version][所属群组][文件或目录...] 或 chgrp [-cfhRv][--help][--reference=<参考文件或目录>][--version][文件或目录...]

参数：

-R 或--recursive 递归处理，将指定目录下的所有文件及子目录一并处理。

--reference=<参考文件或目录> 把指定文件或目录的所属群组全部设成和参考文件或目录的所属群组相同。

例：改变文件的群组属性：

chgrp bin log2012.log

输出： [root@localhost test]# ll

---xrw-r-- 1 root root 302108 11-13 06:03 log2012.log

[root@localhost test]# chgrp -v bin log2012.log

"log2012.log" 的所属组已更改为 bin

[root@localhost test]# ll

---xrw-r-- 1 root bin 302108 11-13 06:03 log2012.log

说明： 将 log2012.log 文件由 root 群组改为 bin 群组

2.19 ln 命令是一个非常重要命令，它的功能是为某一个文件在另外一个位置建立一个同步的链接。当我们需要在不同的目录，用到相同的文件时，我们不需要在每一个需要的目录下都放一个必须相同的文件，我们只要在某个固定的目录，放上该文件，然后在 其它的目录下用 ln 命令链接（link）它就可以，不必重复的占用磁盘空间。

ln [参数][源文件或目录][目标文件或目录]

其中参数的格式为

[-bdfinsvF] [-S backup-suffix] [-V {numbered,existing,simple}]

[--help] [--version] [--]

命令功能 ：

Linux 文件系统中，有所谓的链接(link)，我们可以将其视为档案的别名，而链接又可分为两种 ： 硬链接(hard link)与软链接(symbolic link)，硬链接的意思是一个档案可以有多个名称，而软链接的方式则是产生一个特殊的档案，该档案的内容是指向另一个档案的位置。硬链接是存在同一个文件系统中，而软链接却可以跨越不同的文件系统。

不论是硬链接或软链接都不会将原本的档案复制一份，只会占用非常少量的磁碟空间。

软链接：

1.软链接，以路径的形式存在。类似于 Windows 操作系统中的快捷方式

2.软链接可以 跨文件系统 ，硬链接不可以

3.软链接可以对一个不存在的文件名进行链接

4.软链接可以对目录进行链接

硬链接：

1.硬链接，以文件副本的形式存在。但不占用实际空间。

2.不允许给目录创建硬链接

3.硬链接只有在同一个文件系统中才能创建

参数：

-s 软链接(符号链接)

例：给文件创建软链接，为 log2013.log 文件创建软链接 link2013，如果 log2013.log 丢失，link2013 将失效：

ln -s log2013.log link2013

输出：
```
[root@localhost test]# ll
-rw-r--r-- 1 root bin        61 11-13 06:03 log2013.log
[root@localhost test]# ln -s log2013.log link2013
[root@localhost test]# ll
lrwxrwxrwx 1 root root       11 12-07 16:01 link2013 -> log2013.log
-rw-r--r-- 1 root bin        61 11-13 06:03 log2013.log
```

例：给文件创建硬链接，为 log2013.log 创建硬链接 ln2013，log2013.log 与 ln2013 的各项属性相同

ln log2013.log ln2013

输出：
```
[root@localhost test]# ll
lrwxrwxrwx 1 root root       11 12-07 16:01 link2013 -> log2013.log
-rw-r--r-- 1 root bin        61 11-13 06:03 log2013.log
[root@localhost test]# ln log2013.log ln2013
[root@localhost test]# ll
lrwxrwxrwx 1 root root       11 12-07 16:01 link2013 -> log2013.log
-rw-r--r-- 2 root bin        61 11-13 06:03 ln2013
-rw-r--r-- 2 root bin        61 11-13 06:03 log2013.log
```

三、 用户管理命令

3.1 Linux 用户和用户组管理

Linux 系统是一个多用户多任务的分时操作系统，任何一个要使用系统资源的用户，都必须首先向系统管理员申请一个账号，然后以这个账号的身份进入系统。用户的账号一方面可以帮助系统管理员对使用系统的用户进行跟踪，并控制他们对系统资源的访问；另一方面也可以帮助用户组织文件，并为用户提供安全性保护。

每个用户账号都拥有一个唯一的用户名和各自的口令。

用户在登录时键入正确的用户名和口令后，就能够进入系统和自己的主目录。

实现用户账号的管理，要完成的工作主要有如下几个方面：

用户账号的添加、删除与修改。

用户口令的管理。

用户组的管理。

3.2 Linux 系统用户账号的管理

用户账号的管理工作主要涉及到用户账号的添加、修改和删除。

添加用户账号就是在系统中创建一个新账号，然后为新账号分配用户号、用户组、主目录和登录 Shell 等资源。刚添加的账号是被锁定的，无法使用。

3.2.1 添加新的用户账号使用 useradd 命令，其语法如下：

useradd 选项 用户名

选项:

-c comment 指定一段注释性描述。

-d 目录 指定用户主目录，如果此目录不存在，则同时使用-m 选项，可以创建主目录。

-g 用户组 指定用户所属的用户组。

-G 用户组，用户组 指定用户所属的附加组。

-s Shell 文件 指定用户的登录 Shell。

-u 用户号 指定用户的用户号，如果同时有-o 选项，则可以重复使用其他用户的标识号。

用户名: 指定新账号的登录名。

例 1：# useradd –d /usr/sam -m sam

此命令创建了一个用户 sam，其中-d 和-m 选项用来为登录名 sam 产生一个主目录/usr/sam（/usr 为默认的用户主目录所在的父目录）。

例 2：# useradd -s /bin/sh -g group –G adm,root gem

此命令新建了一个用户 gem，该用户的登录 Shell 是 /bin/sh，它属于 group 用户组，同时又属于 adm 和 root 用户组，其中 group 用户组是其主组。

这里可能新建组：#groupadd group 及 groupadd adm，增加用户账号就是在/etc/passwd 文件中为新用户增加一条记录，同时更新其他系统文件如/etc/shadow, /etc/group 等。

Linux 提供了集成的系统管理工具 userconf，它可以用来对用户账号进行统一管理。

3.2.2 删除帐号

如果一个用户的账号不再使用，可以从系统中删除。删除用户账号就是要将/etc/passwd 等系统文件中的该用户记录删除，必要时还删除用户的主目录。

删除一个已有的用户账号使用 userdel 命令，其格式如下：

userdel 选项 用户名

常用的选项是 -r，它的作用是把用户的主目录一起删除。

例：# userdel -r sam

此命令删除用户 sam 在系统文件中（主要是/etc/passwd, /etc/shadow, /etc/group 等）的记录，同时删除用户的主目录。

3.2.3 修改帐号

修改用户账号就是根据实际情况更改用户的有关属性，如用户号、主目录、用户组、登录 Shell 等。

修改已有用户的信息使用 usermod 命令，其格式如下：

usermod 选项 用户名

常用的选项包括-c, -d, -m, -g, -G, -s, -u 以及-o 等，这些选项的意义与 useradd 命令 中的选项一样，可以为用户指定新的资源值。

3.2.4 用户口令的管理

用户管理的一项重要内容是用户口令的管理。用户账号刚创建时没有口令，但是被系统锁定，无法使用，必须为其指定口令后才可以使用，即使是指定空口令。

指定和修改用户口令的 Shell 命令是 passwd。超级用户可以为自己和其他用户指定口令，普通用户只能用它修改自己的口令。命令的格式为：

passwd 选项 用户名

可使用的选项：

-l 锁定口令，即禁用账号。

-u 口令解锁。

-d 使账号无口令。

-f 强迫用户下次登录时修改口令。

如果默认用户名，则修改当前用户的口令。

例如，假设当前用户是 sam，则下面的命令修改该用户自己的口令：$ passwd

Old password:******

New password:*******

Re-enter new password:*******

如果是超级用户，可以用下列形式指定任何用户的口令：

passwd sam

New password:*******

Re-enter new password:*******

普通用户修改自己的口令时，passwd 命令会先询问原口令，验证后再要求用户输入两遍新口令，如果两次输入的口令一致，则将这个口令指定给用户；而超级用户为用户指定口令时，就不需要知道原口令。

为了系统安全起见，用户应该选择比较复杂的口令，例如最好使用 8 位长的口令，口令中包含有大写、小写字母和数字，并且应该与姓名、生日等不相同。

为用户指定空口令时，执行下列形式的命令：

passwd -d sam

此命令将用户 sam 的口令删除，这样用户 sam 下一次登录时，系统就不再询问口令。

passwd 命令还可以用-l(lock)选项锁定某一用户，使其不能登录，例如：

passwd -l sam

3.3 Linux 系统用户组的管理

每个用户都有一个用户组，系统可以对一个用户组中的所有用户进行集中管理。不同 Linux 系统对用户组的规定有所不同，如 Linux 下的用户属于与它同名的用户组，这个用户组在创建用户时同时创建。

用户组的管理涉及用户组的添加、删除和修改。组的增加、删除和修改实际上就是对/etc/group 文件的更新。

3.3.1 增加一个新的用户组使用 groupadd 命令。其格式如下：

groupadd 选项 用户组

可以使用的选项有：

-g GID 指定新用户组的组标识号（GID）。

-o 一般与-g 选项同时使用，表示新用户组的 GID 可以与系统已有用户组的 GID 相同。

例 1：# groupadd group1

此命令向系统中增加了一个新组 group1，新组的组标识号是在当前已有的最大组标识号 的基础上加 1。

例 2：# groupadd -g 101 group2

此命令向系统中增加了一个新组 group2，同时指定新组的组标识号是 101。

3.3.2 如果要删除一个已有的用户组，使用 groupdel 命令，其格式如下：

groupdel 用户组

例：# groupdel group1

此命令从系统中删除组 group1。

3.3.3 修改用户组的属性使用 groupmod 命令。其语法如下：

groupmod 选项 用户组

常用的选项有：

-g GID 为用户组指定新的组标识号。

-o 与-g 选项同时使用，用户组的新 GID 可以与系统已有用户组的 GID 相同。

-n 新用户组 将用户组的名字改为新名字

例 1：# groupmod -g 102 group2

此命令将组 group2 的组标识号修改为 102。

例 2：# groupmod –g 10000 -n group3 group2

此命令将组 group2 的标识号改为 10000，组名修改为 group3。

3.3.4 如果一个用户同时属于多个用户组，那么用户可以在用户组之间切换，以便具有其他用户组的权限。

用户可以在登录后，使用命令 newgrp 切换到其他用户组，这个命令的参数就是目的用户组。

例：$ newgrp root

这条命令将当前用户切换到 root 用户组，前提条件是 root 用户组确实是该用户的主组或附加组。类似于用户账号的管理，用户组的管理也可以通过集成的系统管理工具来完成。

3.4 与用户账号有关的系统文件

完成用户管理的工作有许多种方法，但是每一种方法实际上都是对有关的系统文件进行修改。与用户和用户组相关的信息都存放在一些系统文件中，这些文件包括/etc/passwd, /etc/shadow, /etc/group 等。

下面分别介绍这些文件的内容。

3.4.1 /etc/passwd 文件是用户管理工作涉及的最重要的一个文件。

Linux 系统中的每个用户都在/etc/passwd 文件中有一个对应的记录行，它记录了这个用户的一些基本属性。

这个文件对所有用户都是可读的。它的内容类似下面的例子：# cat /etc/passwd

root:x:0:0:Superuser:/:

daemon:x:1:1:System daemons:/etc:

bin:x:2:2:Owner of system commands:/bin:

sys:x:3:3:Owner of system files:/usr/sys:

adm:x:4:4:System accounting:/usr/adm:

uucp:x:5:5:UUCP administrator:/usr/lib/uucp:

auth:x:7:21:Authentication administrator:/tcb/files/auth:

cron:x:9:16:Cron daemon:/usr/spool/cron:

listen:x:37:4:Network daemon:/usr/net/nls:

lp:x:71:18:Printer administrator:/usr/spool/lp:

sam:x:200:50:Sam san:/usr/sam:/bin/sh

从上面的例子我们可以看到，/etc/passwd 中一行记录对应着一个用户，每行记录又被冒号(:)分隔为 7 个字段，其格式和具体含义如下：

用户名:口令:用户标识号:组标识号:注释性描述:主目录:登录 Shell

1) "用户名"是代表用户账号的字符串。

通常长度不超过 8 个字符，并且由大小写字母和/或数字组成。登录名中不能有冒号(:)，因为冒号在这里是分隔符。为了兼容起见，登录名中最好不要包含点字符(.)，并且不使用连字符(-)和加号(+)打头。

2) “口令”一些系统中，存放着加密后的用户口令字。

虽然这个字段存放的只是用户口令的加密串，不是明文，但是由于/etc/passwd 文件对 所有用户都可读，所以这仍是一个安全隐患。因此，现在许多 Linux 系统（如 SVR4）都使用了 shadow 技术，把真正的加密后的用户口令字存放到/etc/shadow 文件中，而在/etc/passwd 文件的口令字段中只存放一个特殊的字符，例如“x”或者“*”。

3) “用户标识号”是一个整数，系统内部用它来标识用户。

一般情况下它与用户名是一一对应的。如果几个用户名对应的用户标识号是一样的，系统内部将把它们视为同一个用户，但是它们可以有不同的口令、不同的主目录以及不同的登录 Shell 等。通常用户标识号的取值范围是 0～65 535。0 是超级用户 root 的标识号，1～ 99 由系统保留，作为管理账号，普通用户的标识号从 100 开始。在 Linux 系统中，这个界限是 500。

4) “组标识号”字段记录的是用户所属的用户组。

它对应着/etc/group 文件中的一条记录。

5) “注释性描述”字段记录着用户的一些个人情况。

例如用户的真实姓名、电话、地址等，这个字段并没有什么实际的用途。在不同的 Linux 系统中，这个字段的格式并没有统一。在许多 Linux 系统中，这个字段存放的是一段任意的注释性描述文字，用做 finger 命令的输出。

6) “主目录”，也就是用户的起始工作目录。

它是用户在登录到系统之后所处的目录。在大多数系统中，各用户的主目录都被组织在同一个特定的目录下，而用户主目录的名称就是该用户的登录名。各用户对自己的主目录有读、写、执行（搜索）权限，其他用户对此目录的访问权限则根据具体情况设置。

7) 用户登录后，要启动一个进程，负责将用户的操作传给内核，这个进程是用户登录到系统后运行的命令解释器或某个特定的程序，即 Shell。

Shell 是用户与 Linux 系统之间的接口。Linux 的 Shell 有许多种，每种都有不同的特点。 常用的有 sh(Bourne Shell), csh(C Shell), ksh(Korn Shell), tcsh(TENEX/TOPS-20 type C Shell), bash(Bourne Again Shell)等。

系统管理员可以根据系统情况和用户习惯为用户指定某个 Shell。如果不指定 Shell，那么系统使用 sh 为默认的登录 Shell，即这个字段的值为/bin/sh。

用户的登录 Shell 也可以指定为某个特定的程序（此程序不是一个命令解释器）。利用这一特点，我们可以限制用户只能运行指定的应用程序，在该应用程序运行结束后，用户就自动退出了系统。有些 Linux 系统要求只有那些在系统中登记了的程序才能出现在这个字段中。

8) 系统中有一类用户称为伪用户（psuedo users）。

这些用户在/etc/passwd 文件中也占有一条记录，但是不能登录，因为它们的登录 Shell 为 空。它们的存在主要是方便系统管理，满足相应的系统进程对文件属主的要求。

常见的伪用户如下所示：

伪 用 户 含 义

bin 拥有可执行的用户命令文件

sys 拥有系统文件

adm 拥有帐户文件

nobody NFS 使用

拥有帐户文件

除了上面列出的伪用户外，还有许多标准的伪用户，例如：audit, cron, mail, usenet 等，它们也都各自为相关的进程和文件所需要。

由于/etc/passwd 文件是所有用户都可读的，如果用户的密码太简单或规律比较明显的话，一台普通的计算机就能够很容易地将它破解，因此对安全性要求较高的 Linux 系统都把加密后的口令字分离出来，单独存放在一个文件中，这个文件是/etc/shadow 文件。 有超级用户才拥有该文件读权限，这就保证了用户密码的安全性。

3.4.2 /etc/shadow 中的记录行与/etc/passwd 中的一一对应，它由 pwconv 命令根据/etc/passwd 中的数据自动产生

它的文件格式与/etc/passwd 类似，由若干个字段组成，字段之间用":"隔开。这些字段是：

登录名:加密口令:最后一次修改时间:最小时间间隔:最大时间间隔:警告时间:不活动时间:失效时间:标志

"登录名"是与/etc/passwd 文件中的登录名相一致的用户账号

"口令"字段存放的是加密后的用户口令字，长度为 13 个字符。如果为空，则对应用户没有口令，登录时不需要口令；如果含有不属于集合 { ./0-9A-Za-z }中的字符，则对应的用户不能登录。

"最后一次修改时间"表示的是从某个时刻起，到用户最后一次修改口令时的天数。时间起点对不同的系统可能不一样。例如在 SCO Linux 中，这个时间起点是 1970 年 1 月 1 日。

"最小时间间隔"指的是两次修改口令之间所需的最小天数。

"最大时间间隔"指的是口令保持有效的最大天数。

"警告时间"字段表示的是从系统开始警告用户到用户密码正式失效之间的天数。

"不活动时间"表示的是用户没有登录活动但账号仍能保持有效的最大天数。

"失效时间"字段给出的是一个绝对的天数，如果使用了这个字段，那么就给出相应账号的生存期。期满后，该账号就不再是一个合法的账号，也就不能再用来登录了。

下面是/etc/shadow 的一个例子：

```
# cat /etc/shadow
root:Dnakfw28zf38w:8764:0:168:7:::
daemon:*::0:0::::
bin:*::0:0::::
sys:*::0:0::::
adm:*::0:0::::
uucp:*::0:0::::
nuucp:*::0:0::::
auth:*::0:0::::
cron:*::0:0::::
listen:*::0:0::::
lp:*::0:0::::
sam:EkdiSECLWPdSa:9740:0:0::::
```

3.4.3 用户组的所有信息都存放在/etc/group 文件中。

将用户分组是 Linux 系统中对用户进行管理及控制访问权限的一种手段。每个用户都属于某个用户组；一个组中可以有多个用户，一个用户也可以属于不同的组。当一个用户同时是多个组中的成员时，在/etc/passwd 文件中记录的是用户所属的主组，也就是登录时所属的默认组，而其他组称为附加组。用户要访问属于附加组的文件时，必须首先使用 newgrp 命令使自己成为所要访问的组中的成员。用户组的所有信息都存放在/etc/group 文件中。此文件的格式也类似于/etc/passwd 文件，由冒号(:)隔开若干个字段，这些字段有：

组名:口令:组标识号:组内用户列表

"组名"是用户组的名称，由字母或数字构成。与/etc/passwd 中的登录名一样，组名不应重复。

"口令"字段存放的是用户组加密后的口令字。一般 Linux 系统的用户组都没有口令，即这个字段一般为空，或者是*。

"组标识号"与用户标识号类似，也是一个整数，被系统内部用来标识组。

"组内用户列表"是属于这个组的所有用户的列表/b]，不同用户之间用逗号(,)分隔。这个用户组可能是用户的主组，也可能是附加组。

/etc/group 文件的一个例子如下：

```
root::0:root
bin::2:root,bin
sys::3:root,uucp
adm::4:root,adm
daemon::5:root,daemon
lp::7:root,lp
users::20:root,sam
```

四、 打包管理命令

4.1 Linux tar 命令用于备份文件。

tar 是用来建立，还原备份文件的工具程序，它可以加入，解开备份文件内的文件。

tar [-cdtvxzZ][-b <区块数目>][-C <目的目录>][-f <备份文件>] [文件或目录...]

参数：

-c 或--create 建立新的备份文件。

-C<目的目录>或--directory=<目的目录> 切换到指定的目录。
-d 或--diff 或--compare 对比备份文件内和文件系统上的文件的差异。
-f<备份文件>或--file=<备份文件> 指定备份文件。
-t 或--list 列出备份文件的内容。
-v 或--verbose 显示指令执行过程。
-x 或--extract 或--get 从备份文件中还原文件。
-z 或--gzip 或--ungzip 通过 gzip 指令处理备份文件。
-Z 或--compress 或--uncompress 通过 compress 指令处理备份文件。

例：压缩文件 非打包

```
# touch a.c
# tar -czvf test.tar.gz a.c     //压缩 a.c 文件为 test.tar.gz
```

列出压缩文件内容

```
# tar -tzvf test.tar.gz
-rw-r--r-- root/root         0 2010-05-24 16:51:59 a.c
```

解压文件

```
# tar -xzvf test.tar.gz a.c
```

4.2 Linux zip 命令用于压缩文件。

zip 是个使用广泛的压缩程序，文件经它压缩后会另外产生具有".zip"扩展名的压缩文件。

zip [-KlLrSv][-b <工作目录>][-ll][-t <日期时间>] [压缩文件][文件...] 参数：

-k 使用 MS-DOS 兼容格式的文件名称。
-l 压缩文件时，把 LF 字符置换成 LF+CR 字符。
-ll 压缩文件时，把 LF+CR 字符置换成 LF 字符。
-L 显示版权信息。
-r 递归处理，将指定目录下的所有文件和子目录一并处理。
-S 包含系统和隐藏文件。
-t<日期时间> 把压缩文件的日期设成指定的日期。
-v 显示指令执行过程或显示版本信息。
-<压缩效率> 压缩效率是一个介于 1-9 的数值。

例：将 /home/html/ 这个目录下所有文件和文件夹打包为当前目录下的 html.zip：

```
zip -q -r html.zip /home/html
```

如果在我们在 /home/html 目录下，可以执行以下命令：

```
zip -q -r html.zip *
```

从压缩文件 cp.zip 中删除文件 a.c

```
zip -dv cp.zip a.c
```

4.3 Linux unzip 命令用于解压缩 zip 文件

unzip 为.zip 压缩文件的解压缩程序。

unzip [-cflvz][-qX][.zip 文件][文件][-d <目录>][-x <文件>] 或 unzip [-Z]

参数：

-c 将解压缩的结果显示到屏幕上，并对字符做适当的转换。
-f 更新现有的文件。
-l 显示压缩文件内所包含的文件。
-v 执行是时显示详细的信息。
-z 仅显示压缩文件的备注文字。
-q 执行时不显示任何信息。
-X 解压缩时同时回存文件原来的 UID/GID。

[.zip 文件] 指定.zip 压缩文件。
[文件] 指定要处理.zip 压缩文件中的哪些文件。
-d<目录> 指定文件解压缩后所要存储的目录。
-x<文件> 指定不要处理.zip 压缩文件中的哪些文件。
-Z unzip -Z 等于执行 zipinfo 指令。

例：查看压缩文件中包含的文件：

```
# unzip -l abc.zip
Archive: abc.zip
Length     Date    Time    Name
--------    ----    ----    ----
  94618 05-21-10 20:44   a11.jpg
 202001 05-21-10 20:44   a22.jpg
     16 05-22-10 15:01   11.txt
  46468 05-23-10 10:30   w456.JPG
 140085 03-14-10 21:49   my.asp
--------                   -------
 483188                   5 files
```

五、 进程管理命令

5.1 ps 命令用于显示当前进程 (process) 的状态。

ps [options] [--help]

参数：ps 的参数非常多，在此仅列出几个常用的参数并大略介绍含义

-A 列出所有的行程
-au 显示较详细的资讯
-aux 显示所有包含其他使用者的行程

au(x) 输出格式 ：

USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND
USER: 行程拥有者　　PID: pid
%CPU: 占用的 CPU 使用率　　%MEM: 占用的记忆体使用率
VSZ: 占用的虚拟记忆体大小　　RSS: 占用的记忆体大小
TTY: 虚拟终端的编号　　STAT: 该行程的状态:
D: 不可中断的静止　　R: 正在执行中
S: 静止状态　　T: 暂停执行
Z: 不存在但暂时无法消除　　W: 没有足够的记忆体分页可分配
<: 高优先序的行程　　N: 低优先序的行程
L: 有记忆体分页分配并锁在记忆体内　　START: 行程开始时间
TIME: 执行的时间 COMMAND:所执行的指令

例：# ps -A 显示进程信息

```
PID TTY          TIME CMD
1 ?        00:00:02 init
2 ?        00:00:00 kthreadd
3 ?        00:00:00 migration/0
4 ?        00:00:00 ksoftirqd/0
5 ?        00:00:00 watchdog/0
6 ?        00:00:00 events/0
……省略部分结果
```

```
31374 pts/2    00:00:00 bash
31396 pts/2    00:00:00 ps
```

显示指定用户信息

```
# ps -u root //显示 root 进程用户信息
PID TTY          TIME CMD
1 ?        00:00:02 init
2 ?        00:00:00 kthreadd
3 ?        00:00:00 migration/0
4 ?        00:00:00 ksoftirqd/0
5 ?        00:00:00 watchdog/0
6 ?        00:00:00 events/0
……省略部分结果
31374 pts/2    00:00:00 bash
31397 pts/2    00:00:00 ps
```

显示所有进程信息，连同命令行

```
# ps -ef //显示所有命令，连带命令行
UID        PID PPID C STIME TTY          TIME CMD
root         1     0 0 10:22 ?        00:00:02 /sbin/init
root         2     0 0 10:22 ?        00:00:00 [kthreadd]
root         3     2 0 10:22 ?        00:00:00 [migration/0]
root         4     2 0 10:22 ?        00:00:00 [ksoftirqd/0]
root         5     2 0 10:22 ?        00:00:00 [watchdog/0]
root         6     2 0 10:22 ?        /usr/lib/NetworkManager
……省略部分结果
root     31407 31374 0 17:48 pts/2    00:00:00 ps -ef
```

5.2 kill 命令用于删除执行中的程序或工作。

kill 可将指定的信息送至程序。可将指定程序终止。若仍无法终止该程序，可使用 SIGKILL(9)信息尝试强制删除程序。程序或工作的编号可利用 ps 指令或 jobs 指令查看。

kill [-s <信息名称或编号>][程序]　或　kill [-l <信息编号>]

参数：

-l <信息编号>　若不加<信息编号>选项，则-l 参数会列出全部的信息名称。

-s <信息名称或编号>　指定要送出的信息。

[程序]　[程序]可以是程序的 PID 或是 PGID，也可以是工作编号。

例：杀死进程　# kill 12345

强制杀死进程　# kill -KILL 123456

发送 SIGHUP 信号，可以使用一下信号

```
# kill -HUP pid
```

彻底杀死进程　# kill -9 123456

显示信号　# kill -l

```
 1)SIGHUP       2)SIGINT       3)SIGQUIT      4)SIGILL       5)SIGTRAP
 6)SIGABRT      7)SIGBUS       8)SIGFPE       9)SIGKILL     10)SIGUSR1
11)SIGSEGV     12)SIGUSR2     13)SIGPIPE     14)SIGALRM     15)SIGTERM
16)SIGSTKFLT   17)SIGCHLD     18)SIGCONT     19)SIGSTOP     20)SIGTSTP
21)SIGTTIN     22)SIGTTOU     23)SIGURG      24)SIGXCPU     25)SIGXFSZ
26)SIGVTALRM   27)SIGPROF     28)SIGWINCH    29)SIGIO       30)SIGPWR
```

31)SIGSYS	34)SIGRTMIN	35)SIGRTMIN+1	36)SIGRTMIN+2	37)SIGRTMIN+3
38)SIGRTMIN+4	39)SIGRTMIN+5	40)SIGRTMIN+6	41)SIGRTMIN+7	42)SIGRTMIN+8
43)SIGRTMIN+9	44)SIGRTMIN+10	45)SIGRTMIN+11	46)SIGRTMIN+12	47)SIGRTMIN+13
48)SIGRTMIN+14	49)SIGRTMIN+15	50)SIGRTMAX-14	51)SIGRTMAX-13	52)SIGRTMAX-12
53)SIGRTMAX-11	54)SIGRTMAX-10	55)SIGRTMAX-9	56)SIGRTMAX-8	57)SIGRTMAX-7
58)SIGRTMAX-6	59)SIGRTMAX-5	60)SIGRTMAX-4	61)SIGRTMAX-3	62)SIGRTMAX-2
63)SIGRTMAX-1	64)SIGRTMAX			

杀死指定用户所有进程

```
#kill -9 $(ps -ef | grep hnlinux) //方法一 过滤出 hnlinux 用户进程
```

5.3 top 命令用于实时显示 process 的动态。

```
top [-] [d delay] [n]
```

参数：

d：改变显示的更新速度，或是在交谈式指令列(interactive command)按 s

n：更新的次数，完成后将会退出 top

例:

显示进程信息 # top

设置信息更新次数 #top -n 2 //表示更新两次后终止更新显示

设置信息更新时间 # top -d 3 //表示更新周期为 3 秒

六、特殊命令

6.1 问号？ 匹配一个长度的字符

例：#ls

```
abc.c    bbc.c    acc.c    bcc.c
#mv ?bc.c ../
#ls
acc.c    bcc.c
```

6.2 星号* 匹配任意长度的字符串

例：#ls

```
abc.c    bbc.c    cbc.c    dbc.cpp
#mv *.c
#ls
dbc.cpp
```

6.3 放括号:[]

6.3.1 [...] 匹配方括号中的任意字符

例：#ls

```
abc.c    bbc.c    cbc.c    dbc.c
#mv [ab]bc.c ../
#ls
cbc.c    dbc.c
```

6.3.2 [...-...] ==>匹配方括号中的一个范围

例：#ls

```
abc.c    bbc.c    cbc.c    dbc.c
#mv [a-c]bc.c ../
#ls
dbc.c
```

6.3.3 [^...]==>匹配所有字符，除过方括号中的字符

例：#ls

abc.c bbc.c cbc.c dbc.c

#mv [^a]bc.c ../

#ls

abc.c

6.4 grep 命令用于查找文件里符合条件的字符串。

grep 指令用于查找内容包含指定的范本样式的文件，如果发现某文件的内容符合所指定的范本样式，预设 grep 指令会把含有范本样式的那一列显示出来。若不指定任何文件名称，或是所给予的文件名为"-"，则 grep 指令会从标准输入设备读取数据。

grep [-a] [文件或目录...]

参数：

-a 或--text 不要忽略二进制的数据。

例：在当前目录中，查找后缀有 file 字样的文件中包含 test 字符串的文件，并打印出该字符串的行。此时，可以使用如下命令：

grep test *file

结果如下所示：

$ grep test test* #查找后缀有“test”的文件包含“test”字符串的文件

testfile1:This a Linux testfile! #列出 testfile1 文件中包含 test 字符的行

testfile_2:This is a linux testfile! #列出 testfile_2 文件中包含 test 字符的行

testfile_2:Linux test #列出 testfile_2 文件中包含 test 字符的行

6.5 wc 命令用于计算字数。

利用 wc 指令我们可以计算文件的 Byte 数、字数、或是列数，若不指定文件名称、或是所给予的文件名为"-"，则 wc 指令会从标准输入设备读取数据。

wc [-clw][--help][--version][文件...]

参数：

-c 或--bytes 或--chars 只显示 Bytes 数。

-l 或--lines 只显示行数。

-w 或--words 只显示字数。

例：在默认的情况下，wc 将计算指定文件的行数、字数，以及字节数。使用的命令为：

wc testfile

先查看 testfile 文件的内容，可以看到：

$ cat testfile

Linux networks are becoming more and more common,but scurity is often an overlooked

issue. Unfortunately, in today’s environment all networks are potential hacker targets,

fro0m tp-secret military research networks to small home LANs.

Linux Network Securty focuses on securing Linux in a networked environment,where the

security of the entire network needs to be considered rather than just isolated machines.

It uses a mix of theory and practicl techniques to teach administrators how to install and use security applications,as well as how the applcations work and why they are necesary.

使用 wc 统计，结果如下：

$ wc testfile # testfile 文件的统计信息

3 92 598 testfile # testfile 文件的行数为 3、单词数 92、字节数 598

其中，3 个数字分别表示 testfile 文件的行数、单词数，以及该文件的字节数。

如果想同时统计多个文件的信息，例如同时统计 testfile、testfile_1、testfile_2，可使用如 下命令：

wc testfile testfile_1 testfile_2 #统计三个文件的信息

输出结果如下：

```
$ wc testfile testfile_1 testfile_2   #统计三个文件的信息
 3 92 598 testfile                    #第一个文件行数为 3、单词数 92、字节数 598
 9 18 78 testfile_1                   #第二个文件的行数为 9、单词数 18、字节数 78
 3 6 32 testfile_2                    #第三个文件的行数为 3、单词数 6、字节数 32
 15 116 708 总用量                      #三个文件总共的行数为 15、单词数 116、字节数 708
```

6.6 | 管道 前面命令的标准输出作为后面命令的标准输入

```
# ifconfig|grep eth0
```

表示 ifconfig 查出来的信息然后过滤出 eth0 的这一行

6.7 > 输出重定向 将 >符号左边命令执行的结果给右边程序

```
echo '123' >test.txt
```

表示将 123 输入到文件 test.txt 中

6.8 < 输入重定向 将< 右边的程序内容发送给左边程序

```
wc < hello.txt
```

计算 hello.txt 文件中的行数，单词数，字节数

6.9 >> 输出重定向

输出重定向追加 - 例如：echo "123" >> test.txt 将 123 追加到文件 test.txt 中

七、 文本编辑器

7.1 Linux vi/vim

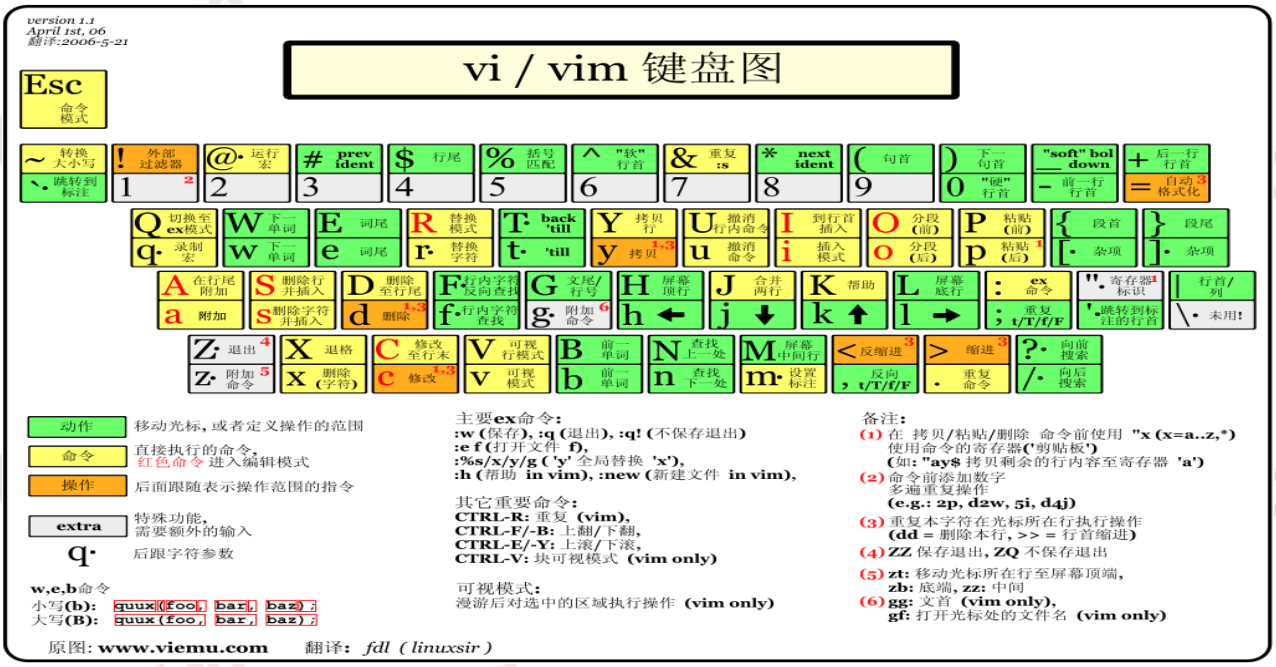
所有的 Unix Like 系统都会内建 vi 文书编辑器，其他的文书编辑器则不一定会存在。但是目前我们使用比较多的是 vim 编辑器。vim 具有程序编辑的能力，可以主动的以字体颜色辨别语法的正确性，方便程序设计。

7.2 什么是 vim?

Vim 是从 vi 发展出来的一个文本编辑器。代码补完、编译及错误跳转等方便编程的功能特别丰富，在程序员中被广泛使用。

简单的来说， vi 是老式的字处理器，不过功能已经很齐全了，但是还是有可以进步的地方。 vim 则可以说是程序开发者的一项很好用的工具。

7.2.1 vim 键盘图：



7.3 vi/vim 的使用

基本上 vi/vim 共分为三种模式，分别是命令模式（Command mode），输入模式（Insert mode）和底线命令模式（Last line mode）。 这三种模式的作用分别是：

7.3.1 命令模式：

用户刚刚启动 vi/vim，便进入了命令模式。此状态下敲击键盘动作会被 Vim 识别为命令，而非输入字符。比如我们此时按下 i，并不会输入一个字符，i 被当作了一个命令。

7.3.2 输入模式

在命令模式下按下 i 就进入了输入模式。

在输入模式中，可以使用以下按键：

字符按键以及 Shift 组合，输入字符

ENTER，回车键，换行

BACK SPACE，退格键，删除光标前一个字符

DEL，删除键，删除光标后一个字符

方向键，在文本中移动光标

HOME/END，移动光标到行首/行尾

Page Up/Page Down，上/下翻页

Insert，切换光标为输入/替换模式，光标将变成竖线/下划线

ESC，退出输入模式，切换到命令模式

7.3.3 底线命令模式

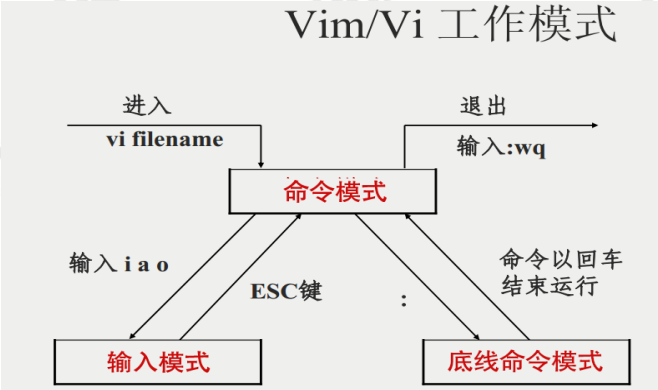
在命令模式下按下:（英文冒号）就进入了底线命令模式。 底线命令模式可以输入单个或多个字符的命令，可用的命令非常多。 在底线命令模式中，基本的命令有（已经省略了冒号）：

q 退出程序

w 保存文件

按 ESC 键可随时退出底线命令模式。

简单的说，我们可以将这三个模式想成底下的图标来表示：



7.3.4 vi/vim 使用实例

使用 vi/vim 进入一般模式

如果你想要使用 vi 来建立一个名为 test.txt 的文件时，你可以这样做：

$ vi uplooking.txt

直接输入 vi 文件名就能够进入 vi 的一般模式了。请注意，记得 vi 后面一定要加文件名，不管该文件存在与否！

按下 i 进入输入模式(也称为编辑模式)，开始编辑文字

在一般模式之中，只要按下 i,o,a，I，O，A 等字符就可以进入输入模式了！

在编辑模式当中，你可以发现在左下角状态栏中会出现 –INSERT- 的字样，那就是可以输入任意字符的提示。这个时候，键盘上除了 **Esc** 这个按键之外，其他的按键都可以视作为一般的输入按钮了，所以你可以进行任何的编辑。

按下 ESC 按钮回到一般模式好了，假设我已经按照上面的样式给他编辑完毕了，那么应该要如何退出呢？是的！没错！就是给他按下 **Esc** 这个按钮即可！马上你就会发现画面左下角的 –INSERT– 不见了！

在一般模式中按下 :wq 储存后离开 vi，这样我们就成功创建了一个 uplooking.txt 的文件。

7.3.5 vi/vim 按键说明

除了上面简易范例的 i, Esc, :wq 之外，其实 vim 还有非常多的按键可以使用。

第一部份：一般模式可用的光标移动、复制粘贴、搜索替换等

移动光标的方法	
h 或 向左箭头键(←)	光标向左移动一个字符
j 或 向下箭头键(↓)	光标向下移动一个字符
k 或 向上箭头键(↑)	光标向上移动一个字符
l 或 向右箭头键(→)	光标向右移动一个字符
如果你将右手放在键盘上的话，你会发现 hjkl 是排列在一起的，因此可以使用这四个按钮来移动光标。 如果想要进行多次移动的话，例如向下移动 30 行，可以使用 "30j" 或 "30↓" 的组合按键， 亦即加上想要进行的次数(数字)后，按下动作即可！	
[Ctrl] + [f]	屏幕『向下』移动一页，相当于 [Page Down]按键 (常用)
[Ctrl] + [b]	屏幕『向上』移动一页，相当于 [Page Up] 按键 (常用)
[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>	那个 n 表示『数字』，例如 20 。按下数字后再按空格键，光标会向右移动这一行的 n 个字符。例如 20<space> 则光标会向后面移动 20 个字符距离。
0 或功能键[Home]	这是数字『 0 』：移动到这一行的最前面字符处 (常用)
$ 或功能键[End]	移动到这一行的最后面字符处(常用)
H	光标移动到这个屏幕的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	移动到这个档案的最后一行(常用)
nG	n 为数字。移动到这个档案的第 n 行。例如 20G 则会移动到这个档案的第 20 行(可配合 :set nu)
gg	移动到这个档案的第一行，相当于 1G 啊！ (常用)

n<Enter>	n 为数字。光标向下移动 n 行(常用)
搜索替换	
/word	向光标之下寻找一个名称为 word 的字符串。例如要在档案内搜寻 vbird 这个字符串，就输入 /vbird 即可！ (常用)
?word	向光标之上寻找一个字符串名称为 word 的字符串。
n	这个 n 是英文按键。代表重复前一个搜寻的动作。举例来说，如果刚刚我们执行 /vbird 去向下搜寻 vbird 这个字符串，则按下 n 后，会向下继续搜寻下一个名称为 vbird 的字符串。如果是执行 ?vbird 的话，那么按下 n 则会向上继续搜寻名称为 vbird 的字符串！
N	这个 N 是英文按键。与 n 刚好相反，为『反向』进行前一个搜寻动作。 例如 /vbird 后，按下 N 则表示『向上』搜寻 vbird 。
使用 /word 配合 n 及 N 是非常有帮助的！可以让你重复的找到一些你搜寻的关键词！	
:n1,n2s/word1/word2/g	n1 与 n2 为数字。在第 n1 与 n2 行之间寻找 word1 这个字符串，并将该字符串取代为 word2 ！举例来说，在 100 到 200 行之间搜寻 vbird 并取代为 VBIRD 则：『:100,200s/vbird/VBIRD/g』。(常用)
:1,$s/word1/word2/g	从第一行到最后一行寻找 word1 字符串，并将该字符串取代为 word2 ！(常用)
:1,$s/word1/word2/gc	从第一行到最后一行寻找 word1 字符串，并将该字符串取代为 word2 ！且在取代前显示提示字符给用户确认 (confirm) 是否需要取代！(常用)
删除、复制与贴上	
x, X	在一行字当中，x 为向后删除一个字符 (相当于 [del] 按键)，X 为向前删除一个字符(相当于 [backspace] 亦即是退格键) (常用)
nx	n 为数字，连续向后删除 n 个字符。举例来说，我要连续删除 10 个字符，『10x』。
dd	删除游标所在的那一整行(常用)
ndd	n 为数字。删除光标所在的向下 n 行，例如 20dd 则是删除 20 行 (常用)
d1G	删除光标所在到第一行的所有数据
dG	删除光标所在到最后一行的所有数据
d$	删除游标所在处，到该行的最后一个字符
d0	那个是数字的 0 ，删除游标所在处，到该行的最前面一个字符
yy	复制游标所在的那一行(常用)
nyy	n 为数字。复制光标所在的向下 n 行，例如 20yy 则是复制 20 行(常用)

y1G	复制游标所在行到第一行的所有数据
yG	复制游标所在行到最后一行的所有数据
y0	复制光标所在的那个字符到该行行首的所有数据
y$	复制光标所在的那个字符到该行行尾的所有数据
p, P	p 为将已复制的数据在光标下一行贴上，P 则为贴在游标上一行！ 举例来说，我目前光标在第 20 行，且已经复制了 10 行数据。则按下 p 后， 那 10 行数据会贴在原本的 20 行之后，亦即由 21 行开始贴。但如果是按下 P 呢？ 那么原本的第 20 行会被推到变成 30 行。 (常用)
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	复原前一个动作。(常用)
[Ctrl]+r	重做上一个动作。(常用)
这个 u 与 [Ctrl]+r 是很常用的指令！一个是复原，另一个则是重做一次～ 利用这两个功能按键，你的编辑，嘿嘿！很快乐的啦！	
.	不要怀疑！这就是小数点！意思是重复前一个动作的意思。 如果你想要重复删除、重复贴上等等动作，按下小数点『.』就好了！ (常用)

第二部份：一般模式切换到编辑模式的可用的按钮说明

进入输入或取代的编辑模式	
i, I	进入输入模式(Insert mode)： i 为『从目前光标所在处输入』， I 为『在目前所在行的第一个非空格符处开始输入』。(常用)
a, A	进入输入模式(Insert mode)： a 为『从目前光标所在的下一个字符处开始输入』， A 为『从光标所在行的最后一个字符处开始输入』。(常用)
o, O	进入输入模式(Insert mode)： 这是英文字母 o 的大小写。o 为『在目前光标所在的下一行处输入新的一行』； O 为在目前光标所在处的上一行输入新的一行！(常用)
r, R	进入取代模式(Replace mode)： r 只会取代光标所在的那一个字符一次；R 会一直取代光标所在的文字，直到按下 ESC 为止；(常用)
上面这些按键中，在 vi 画面的左下角处会出现『--INSERT--』或『--REPLACE--』的字样。 由名称就知道该动作了吧！！特别注意的是，我们上面也提过了，你想要在档案里面输入字符时， 一定要在左下角处看到 INSERT 或 REPLACE 才能输入喔！	

[Esc]	退出编辑模式，回到一般模式中(常用)

第三部份：一般模式切换到指令行模式的可用的按钮说明

指令行的储存、离开等指令	
:w	将编辑的数据写入硬盘档案中(常用)
:w!	若文件属性为『只读』时，强制写入该档案。不过，到底能不能写入， 还是跟你对该档案的档案权限有关啊！
:q	离开 vi (常用)
:q!	若曾修改过档案，又不想储存，使用 ！为强制离开不储存档案。
注意一下啊，那个惊叹号 (!) 在 vi 当中，常常具有『强制』的意思～	
:wq	储存后离开，若为 :wq! 则为强制储存后离开 (常用)
ZZ	这是大写的 Z 喔！若档案没有更动，则不储存离开，若档案已经被更动过，则储存后离开！
:w [filename]	将编辑的数据储存成另一个档案（类似另存新档）
:r [filename]	在编辑的数据中，读入另一个档案的数据。亦即将 『filename』 这个档案内容加到游标所在行后面
:n1,n2 w [filename]	将 n1 到 n2 的内容储存成 filename 这个档案。
:! command	暂时离开 vi 到指令行模式下执行 command 的显示结果！例如『:! ls /home』即可在 vi 当中察看 /home 底下以 ls 输出的档案信息！
vim 环境的变更(配置文件/etc/vimrc 或 ~/.vimrc 建议修改后者)	
:set nu	显示行号，设定之后，会在每一行的前缀显示该行的行号
:set nonu	与 set nu 相反，为取消行号！

第四章 Linux 网络配置

一、 Linux 网络配置

网络环境前提：物理链路畅通，网卡，网线齐全

1.1 IP 地址

IP 地址是一个 32 位的二进制数，通常被分割为 4 个“8 位二进制数”（也就是 4 个字节）。IP 地址通常用“点分十进制”表示成（a.b.c.d）的形式，其中，a,b,c,d 都是 0~255 之间的十进制整数。

例：点分十进 IP 地址（100.4.5.6），实际上是 32 位二进制（01100100.00000100.00000101.00000110）。

ip 地址的配置

ip 地址的分类：5 类：所有地址形式采用点分十进制

A 类地址 ==》子网掩码： 255.0.0.0 ==》广域网

1.0.0.0 - 127.255.255.255

10.0.0.0-10.255.255.255 私有地址

B 类地址 ==》子网掩码： 255.255.0.0 ==》城域网

128.0.0.0 -- 191.255.255.255

172.16.0.0-172.31.255.255 私有地址

C 类地址 ==》子网掩码 255.255.255.0 ==》局域网

192.0.0.0 -- 223.255.555.255

192.168.0.0-192.168.255.255 私有地址

D 类地址 ==》没有子网掩码 ==》广播地址

224.0.0.0 --- 239.255.255.255

E 类地址 ==》没有子网掩码 ==》实验预留地址

240.0.0.0 -255.255.255.255

注：1. 每一个字节都为 0 的地址（“0.0.0.0”）对应于当前主机；

2. IP 地址中的每一个字节都为 1 的 IP 地址（“255．255．255．255”）是当前子网的广播地址；

3. IP 地址中凡是以“11110”开头的 E 类 IP 地址都保留用于将来和实验使用。

4. IP 地址中不能以十进制“127”作为开头，该类地址中数字 127．0．0．1 到 127．255．255．255 用于回路测试，如：127.0.0.1 可以代表本机 IP 地址，用“http://127.0.0.1”就可以测试本机中配置的 Web 服务器。

5. 网络 ID 的第一个 8 位组也不能全置为“0”，全“0”表示本地网络。

1.2 子网掩码：根据后边 0 的个数可以查看当前网络最大的主机数

比如 24 后有 8 个 0 我们当前的网络最大可以容纳的主机数为 255-2 个

1.3 ifconfig 查看网络配置(临时网络配置)

1.3.1 ifconfig eth0 192.168.0.xxx/24 up ==>临时配置 ip 地址

1.3.2 ping 192.168.0.1 ==》测试网关

ping 192.168.0.xxx ==>ttl = xx time = xxx 网络联通

1.4 网关的配置

route add default gw 192.168.0.1 ==》配置默认路由

route -n ===》查看路由配置

rotue del default gw 192.168.0.1 ==》删除默认路由

1.5 DNS 的配置

gedit /etc/resolv.conf ==> 配置文件的路径

nameserver 61.134.1.4 ==> 陕西省的 DNS 解析服务器

nameserver 8.8.8.8 ==> google 的 DNS 解析服务器

nameserver 114.114.114.114 ==> 国内首家云安全 DNS(由多个基础电信运营商与南京信风共建共享)

二、 Linux Ubuntu 软件安装

2.1 Ubuntu 软件离线安装

dpkg 命令是 Debian Linux 系统用来安装、创建和管理软件包的实用工具。

dpkg(选项)(参数)

选项:

-i：安装软件包；

-r：删除软件包；

-P：删除软件包的同时删除其配置文件；

-L：显示于软件包关联的文件；

-l：显示已安装软件包列表；

--unpack：解开软件包；

-c：显示软件包内文件列表；

--confiugre：配置软件包。

参数:

deb 软件包：指定要操作的.deb 软件包。

例:

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

注：Linux 系统软件的离线安装大部分都存在依赖包的问题，部分以来问题不容易解决。

2.2 apt-get 命令是 Debian Linux 发行版中的 APT 软件包管理工具。所有基于 Debian 的发行都使用这个包管理系统。deb 包可以把一个应用的文件包在一起，大体就如同 Windows 上的安装文件。

apt-get(选项)(参数)

选项

-c：指定配置文件。

参数

管理指令：对 APT 软件包的管理操作；

软件包：指定要操纵的软件包。

例：使用 apt-get 命令的第一步就是引入必需的软件库，Debian 的软件库也就是所有 Debian 软 件包的集合，它们存在互联网上的一些公共站点上。把它们的地址加入，apt-get 就能搜索到我们想要的软件。

/etc/apt/sources.list 是存放这些地址列表的配置文件，其格式如下：

deb [web 或 ftp 地址] [发行版名字] [main/contrib/non-free]

我们常用的 Ubuntu 就是一个基于 Debian 的发行，我们使用 apt-get 命令获取这个列表，以下是我整理的常用命令：

在修改/etc/apt/sources.list 或者/etc/apt/preferences 之后运行该命令。此外您需要定期运行这一命令以确保您的软件包列表是最新的： apt-get update

安装一个新软件包： apt-get install packagename

卸载一个已安装的软件包（保留配置文件）： apt-get remove packagename

卸载一个已安装的软件包（删除配置文件）： apt-get –purge remove packagename

将安装的软件的备份也删除，不会影响软件的使用的： apt-get clean

更新所有已安装的软件包： apt-get upgrade

将系统升级到新版本： apt-get dist-upgrade

定期运行这个命令来清除那些已经卸载的软件包的.deb 文件：apt-get autoclean 通过这种方式，您可以释放大量的磁盘空间。如果您的需求十分迫切，可以使用 apt-get clean 以释放更多空间。

第五章 简单服务器搭建

一、 NFS(网络文件系统)

NFS（网络文件系统）服务可以将远程 Linux 系统上的文件共享资源挂载到本地主机的目录上，从而使得本地主机（Linux 客户端）基于 TCP/IP 协议，像使用本地主机上的资源那样读写远程 Linux 系统上的共享文件。

1.1 NFS 客服端的安装

```
sudo apt-get install nfs-common
```

1.2 NFS 服务器端的安装

```
sudo apt-get install nfs-kernel-server
```

1.3 NFS 服务器端的配置

```
sudo vim /etc/exports
```

默认情况下里面没有任何内容。我们可以按照“共享目录的路径 允许访问的 NFS 客户端（共享权限参数）”的格式，定义要共享的目录与相应的权限。

例如，如果想要把/nfsfile 目录共享给 192.168.10.0/24 网段内的所有主机，让这些主机都拥有读写权限，在将数据写入到 NFS 服务器的硬盘中后才会结束操作，最大限度保证数据不丢失，以及把来访客户端 root 管理员映射为本地的匿名用户等，则可以按照下面命令中的格式，将表中的参数写到 NFS 服务程序的配置文件中。

用于配置 NFS 服务程序配置文件的参数

参数	作用
ro	只读
rw	读写
root_squash	当 NFS 客户端以 root 管理员访问时，映射为 NFS 服务器的匿名用户
no_root_squash	当 NFS 客户端以 root 管理员访问时，映射为 NFS 服务器的 root 管理员
all_squash	无论 NFS 客户端使用什么账户访问，均映射为 NFS 服务器的匿名用户
sync	同时将数据写入到内存与硬盘中，保证不丢失数据
async	优先将数据保存到内存，然后再写入硬盘；这样效率更高，但可能会丢失数据

请注意，NFS 客户端地址与权限之间没有空格。

1.4 启动 NFS 服务器

```
/etc/init.d/nfs-kernel-server restart
```

1.5 NFS 客户端的配置步骤也十分简单。先使用 showmount 命令（以及必要的参数，见表 12-8）查询 NFS 服务器的远程共享信息，其输出格式为“共享的目录名称 允许使用客户端地址”。

表 showmount 命令中可用的参数以及作用

参数	作用
-e	显示 NFS 服务器的共享列表
-a	显示本机挂载的文件资源的情况
-v	显示版本号

例： [root@linuxprobe ~]# showmount -e 192.168.10.10

Export list for 192.168.10.10:

/nfsfile 192.168.10.*

然后在 NFS 客户端创建一个挂载目录。使用 mount 命令，在命令后面写上服务器的 IP 地址、服务器上的共享目录以及要挂载到本地系统（即客户端）的目录。

[root@linuxprobe ~]# mkdir /nfsfile

[root@linuxprobe ~]# mount nfs 192.168.10.10:/nfsfile /nfsfile

挂载成功后就应该能够顺利地看到在执行前面的操作时写入的文件内容了。

1.6 mount 命令是经常会使用到的命令，它用于挂载 Linux 系统外的文件。

mount [-hV]

mount -a [-fFnrsvw] [-t vfstype]

mount [-fnrsvw] [-o options [,...]] device | dir

mount [-fnrsvw] [-t vfstype] [-o options] device dir

参数：

-V：显示程序版本

-h：显示辅助讯息

-v：显示较讯息，通常和 -f 用来除错。

-a：将 /etc/fstab 中定义的所有档案系统挂上。

-F：这个命令通常和 -a 一起使用，它会为每一个 mount 的动作产生一个行程负责执行。在系统需要挂上大量 NFS 档案系统时可以加快挂上的动作。

-f：通常用在除错的用途。它会使 mount 并不执行实际挂上的动作，而是模拟整个挂上的过程。通常会和 -v 一起使用。

-n：一般而言，mount 在挂上后会在 /etc/mtab 中写入一笔资料。但在系统中没有可写入档案系统存在的情况下可以用这个选项取消这个动作。

-s-r：等于 -o ro

-w：等于 -o rw

-L：将含有特定标签的硬盘分割挂上。

-U：将档案分割序号为 的档案系统挂下。-L 和 -U 必须在/proc/partition 这种档案存在时才有意义。

-t：指定档案系统的型态，通常不必指定。mount 会自动选择正确的型态。

-o async：打开非同步模式，所有的档案读写动作都会用非同步模式执行。

-o sync：在同步模式下执行。

-o atime、-o noatime：当 atime 打开时，系统会在每次读取档案时更新档案的『上一次调用时间』。当我们使用 flash 档案系统时可能会选项把这个选项关闭以减少写入的次数。

-o auto、-o noauto：打开/关闭自动挂上模式。

-o defaults:使用预设的选项 rw, suid, dev, exec, auto, nouser, and async.

-o dev、-o nodev-o exec、-o noexec 允许执行档被执行。

-o suid、-o nosuid：允许执行档在 root 权限下执行。

-o user、-o nouser：使用者可以执行 mount/umount 的动作。

-o remount：将一个已经挂下的档案系统重新用不同的方式挂上。例如原先是唯读的系统，现在用可读写的模式重新挂上。

-o ro：用唯读模式挂上。

-o rw：用可读写模式挂上。

-o loop=：使用 loop 模式用来将一个档案当成硬盘分割挂上系统。

例：

将 /dev/hda1 挂在 /mnt 之下。

#mount /dev/hda1 /mnt

将 /dev/hda1 用唯读模式挂在 /mnt 之下。

#mount -o ro /dev/hda1 /mnt

将 /tmp/image.iso 这个光碟的 image 档使用 loop 模式挂在 /mnt/cdrom 之下。用这种方法可以将一般网络上可以找到的 Linux 光 碟 ISO 档在不烧录成光碟的情况下检视其内容。

#mount -o loop /tmp/image.iso /mnt/cdrom

1.7 umount 可卸除目前挂在 Linux 目录中的文件系统。

umount [-ahnrv][-t <文件系统类型>][文件系统]

参数：

-a 卸除/etc/mtab 中记录的所有文件系统。

-h 显示帮助。

-n 卸除时不要将信息存入/etc/mtab 文件中。

-r 若无法成功卸除，则尝试以只读的方式重新挂入文件系统。

-t<文件系统类型> 仅卸除选项中所指定的文件系统。

-v 执行时显示详细的信息。

[文件系统] 除了直接指定文件系统外，也可以用设备名称或挂入点来表示文件系统。

例：下面两条命令分别通过设备名和挂载点卸载文件系统，同时输出详细信息：

```
# umount -v /dev/sda1              通过设备名卸载
/dev/sda1 umounted
# umount -v /mnt/mymount/          通过挂载点卸载
/tmp/diskboot.img umounted
```

如果设备正忙，卸载即告失败。卸载失败的常见原因是，某个打开的 shell 当前目录为挂载点里的某个目录：

```
# umount -v /mnt/mymount/
umount: /mnt/mymount: device is busy
umount: /mnt/mymount: device is busy
```

对付系统文件正忙的方法是执行延迟卸载：

```
umount -vl /mnt/mymount/         执行延迟卸载
```

延迟卸载（lazy unmount）会立即卸载目录树里的文件系统，等到设备不再繁忙时才清理所有相关资源。

二、 FTP(File Transfer Protocol，FTP 文件传输协议)

2.1 文件传输协议

一般来讲，人们将计算机联网的首要目的就是获取资料，而文件传输是一种非常重要的获取资料的方式。今天的互联网是由几千万台个人计算机、工作站、服务器、小型机、大型机、巨型机等具有不同型号、不同架构的物理设备共同组成的，而且即便是个人计算机，也可能会装有 Windows、Linux、UNIX、Mac 等不同的操作系统。为了能够在如此复杂多样的设备之间解决问题解决文件传输问题，文件传输协议（FTP）应运而生。

FTP 是一种在互联网中进行文件传输的协议，基于客户端/服务器模式，默认使用 20、21 号端口，其中端口 20（数据端口）用于进行数据传输，端口 21（命令端口）用于接受客户端发出的相关 FTP 命令与参数。FTP 服务器普遍部署于内网中，具有容易搭建、方便管理的特点。而且有些 FTP 客户端工具还可以支持文件的多点下载以及断点续传技术，因此 FTP 服务得到了广大用户的青睐。FTP 协议的传输拓扑如图所示。

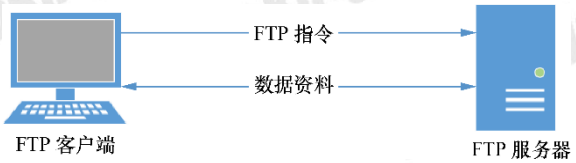


图 FTP 协议的传输拓扑

FTP 服务器是按照 FTP 协议在互联网上提供文件存储和访问服务的主机，FTP 客户端则是向服务器发送连接请求，以建立数据传输链路的主机。FTP 协议有下面两种工作模式。

主动模式：FTP 服务器主动向客户端发起连接请求。

被动模式：FTP 服务器等待客户端发起连接请求（FTP 的默认工作模式）。

2.2 FTP 客户端的安装

```
sudo apt-get install ftp
```

2.3 FTP 服务器端的安装

sudo apt-get install vsftpd

2.4 FTP 服务器的配置

vsftpd 服务程序的主配置文件（/etc/vsftpd/vsftpd.conf）内容总长度达到 123 行，但其中大多数参数在开头都添加了井号（#），从而成为注释信息，大家没有必要在注释信息上花费太多的时间。

表中罗列了 vsftpd 服务程序主配置文件中常用的参数以及作用。当前大家只需要简单了解即可，在后续的实验中将演示这些参数的用法，以帮助大家熟悉并掌握。

vsftpd 服务程序常用的参数以及作用

参数	作用
listen=[YES|NO]	是否以独立运行的方式监听服务
listen_address=IP 地址	设置要监听的 IP 地址
listen_port=21	设置 FTP 服务的监听端口
download_enable＝[YES|NO]	是否允许下载文件
userlist_enable=[YES|NO] userlist_deny=[YES|NO]	设置用户列表为“允许”还是“禁止”操作
max_clients=0	最大客户端连接数，0 为不限制
max_per_ip=0	同一 IP 地址的最大连接数，0 为不限制
anonymous_enable=[YES|NO]	是否允许匿名用户访问
anon_upload_enable=[YES|NO]	是否允许匿名用户上传文件
anon_umask=022	匿名用户上传文件的 umask 值
anon_root=/var/ftp	匿名用户的 FTP 根目录
anon_mkdir_write_enable=[YES|NO]	是否允许匿名用户创建目录
anon_other_write_enable=[YES|NO]	是否开放匿名用户的其他写入权限（包括重命名、删除等操作权限）
anon_max_rate=0	匿名用户的最大传输速率（字节/秒），0 为不限制
local_enable=[YES|NO]	是否允许本地用户登录 FTP
local_umask=022	本地用户上传文件的 umask 值
local_root=/var/ftp	本地用户的 FTP 根目录
chroot_local_user=[YES|NO]	是否将用户权限禁锢在 FTP 目录，以确保安全
local_max_rate=0	本地用户最大传输速率（字节/秒），0 为不限制

2.5 vsftpd 服务程序

vsftpd 作为更加安全的文件传输的服务程序，允许用户以三种认证模式登录到 FTP 服务器上。

匿名开放模式：是一种最不安全的认证模式，任何人都可以无需密码验证而直接登录到 FTP 服务器。

本地用户模式：是通过 Linux 系统本地的账户密码信息进行认证的模式，相较于匿名开放模式更安全，而且配置起来也很简单。但是如果被黑客破解了账户的信息，就可以畅通无阻地登录 FTP 服务器，从而完全控制整台服务器。

虚拟用户模式：是这三种模式中最安全的一种认证模式，它需要为 FTP 服务单独建立用户数据库文件，虚拟出用来进行口令验证的账户信息，而这些账户信息在服务器系统中实际上是不存在的，仅供 FTP 服务程序进行认证使用。这样，即使黑客破解了账户信息也无法登录服务器，从而有效降低了破坏范围和影响。

2.6 匿名开放模式

前文提到，在 vsftpd 服务程序中，匿名开放模式是最不安全的一种认证模式。任何人都可以无需密码验证而直接登录到 FTP 服务器。这种模式一般用来访问不重要的公开文件（在生产环境中尽量不

要存放重要文件）。当然，如果采用第 8 章中介绍的防火墙管理工具（如 Tcp_wrappers 服务程序）将 vsftpd 服务程序允许访问的主机范围设置为企业内网，也可以提供基本的安全性。

vsftpd 服务程序默认开启了匿名开放模式，我们需要做的就是开放匿名用户的上传、下载文件的权限，以及让匿名用户创建、删除、更名文件的权限。需要注意的是，针对匿名用户放开这些权限会带来潜在危险，我们只是为了在 Linux 系统中练习配置 vsftpd 服务程序而放开了这些权限，不建议在生产环境中如此行事。表罗列了可以向匿名用户开放的权限参数以及作用。

可以向匿名用户开放的权限参数以及作用

参数	作用
anonymous_enable=YES	允许匿名访问模式
anon_umask=022	匿名用户上传文件的 umask 值
anon_upload_enable=YES	允许匿名用户上传文件
anon_mkdir_write_enable=YES	允许匿名用户创建目录
anon_other_write_enable=YES	允许匿名用户修改目录名称或删除目录

```
[root@linuxprobe ~]# vim /etc/vsftpd/vsftpd.conf
1 anonymous_enable=YES
2 anon_umask=022
3 anon_upload_enable=YES
4 anon_mkdir_write_enable=YES
5 anon_other_write_enable=YES
6 local_enable=YES
7 write_enable=YES
8 local_umask=022
9 dirmessage_enable=YES
10 xferlog_enable=YES
11 connect_from_port_20=YES
12 xferlog_std_format=YES
13 listen=NO
14 listen_ipv6=YES
15 pam_service_name=vsftpd
16 userlist_enable=YES
17 tcp_wrappers=YES
```

在 vsftpd 服务程序的主配置文件中正确填写参数，然后保存并退出。还需要重启 vsftpd 服务程序，让新的配置参数生效。

2.7 重启服务

```
/etc/init.d/vsftpd       restart
```

2.8 客户端登陆服务器端

现在就可以在客户端执行 ftp 命令连接到远程的 FTP 服务器了。在 vsftpd 服务程序的匿名开放认证模式下，其账户统一为 ftp，密码为 ftp。而且在连接到 FTP 服务器后，默认访问的是/var/ftp 目录。我们可以切换到该目录下的 pub 目录中，然后尝试创建一个新的目录文件，以检验是否拥有写入权限：

```
[root@linuxprobe ~]# ftp 192.168.10.10
Connected to 192.168.10.10 (192.168.10.10).
220 (vsFTPd 3.0.2)
Name (192.168.10.10:root): ftp
331 Please specify the password.
Password:ftp
```

230 Login successful.

Remote system type is UNIX.

Using binary mode to transfer files.

ftp> cd pub

250 Directory successfully changed.

ftp> mkdir files

550 Permission denied.

系统显示拒绝创建目录！因为 pub 目录的用户属于 root，且权限为 755，我们不能进行修改。如果要想服务器上传文件，则需要在/var/ftp 中创建 upload 目录，并将权限修改为 777，这样我们就可以想服务器上传文件了。

三、 SSH(远程主机服务)

3.1 SSH 服务软件安装

sudo apt-get install openssh*

3.2 sshd 服务

SSH（Secure Shell）是一种能够以安全的方式提供远程登录的协议，也是目前远程管理 Linux 系统的首选方式。在此之前，一般使用 FTP 或 Telnet 来进行远程登录。但是因为它们以明文的形式在网络中传输账户密码和数据信息，因此很不安全，很容易受到黑客发起的中间人攻击，这轻则篡改传输的数据信息，重则直接抓取服务器的账户密码。

想要使用 SSH 协议来远程管理 Linux 系统，则需要部署配置 sshd 服务程序。sshd 是基于 SSH 协议开发的一款远程管理服务程序，不仅使用起来方便快捷，而且能够提供两种安全验证的方法：

基于口令的验证—用账户和密码来验证登录；

基于密钥的验证—需要在本地生成密钥对，然后把密钥对中的公钥上传至服务器，并与服务器中的公钥进行比较；该方式相较来说更安全。

sshd 服务的配置信息保存在/etc/ssh/sshd_config 文件中。sshd 服务配置文件中包含的重要参数如表中所示。

sshd 服务配置文件中包含的参数以及作用

参数	作用
Port 22	默认的 sshd 服务端口
ListenAddress 0.0.0.0	设定 sshd 服务器监听的 IP 地址
Protocol 2	SSH 协议的版本号
HostKey /etc/ssh/ssh_host_key	SSH 协议版本为 1 时，DES 私钥存放的位置
HostKey /etc/ssh/ssh_host_rsa_key	SSH 协议版本为 2 时，RSA 私钥存放的位置
HostKey /etc/ssh/ssh_host_dsa_key	SSH 协议版本为 2 时，DSA 私钥存放的位置
PermitRootLogin yes	设定是否允许 root 管理员直接登录
StrictModes yes	当远程用户的私钥改变时直接拒绝连接
MaxAuthTries 6	最大密码尝试次数
MaxSessions 10	最大终端数
PasswordAuthentication yes	是否允许密码验证
PermitEmptyPasswords no	是否允许空密码登录（很不安全）

3.3 sshd 服务重启

/etc/init.d/sshd restart

3.4 sshd 服务器的登陆

[root@linuxprobe ~]# ssh 192.168.10.10

root@192.168.10.10's password:此处输入远程主机用户的密码

Permission denied, please try again.

3.5 远程传输命令

scp（secure copy）是一个基于 SSH 协议在网络之间进行安全传输的命令，其格式为“scp [参数] 本地文件 远程帐户@远程 IP 地址:远程目录”。

```
[root@linuxprobe ~]# echo "Welcome to LinuxProbe.Com" > readme.txt
[root@linuxprobe ~]# scp /root/readme.txt 192.168.10.20:/home
root@192.168.10.20's password:此处输入远程服务器中用户的密码
readme.txt 100% 26 0.0KB/s 00:00
```

scp 命令还可以把远程主机上的文件下载到本地主机，其命令格式为“scp [参数] 远程用户@远程 IP 地址:远程文件 本地目录”。例如，可以把远程主机的系统版本信息文件下载过来，这样就无须先登录远程主机，再进行文件传送了，也就省去了很多周折。

```
[root@linuxprobe ~]# scp 192.168.10.20:/etc/redhat-release /root
root@192.168.10.20's password:此处输入远程服务器中 root 管理员的密码
redhat-release 100% 52 0.1KB/s 00:00
[root@linuxprobe ~]# cat redhat-release
Red Hat Enterprise Linux Server release 7.0 (Maipo)
```

第六章 Linux Shell 编程

一、 Shell 简介

1.1 shell

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

Shell 是指一种应用程序，这个应用程序提供了一个界面，用户通过这个界面访问操作系统内核的服务。

Ken Thompson 的 sh 是第一种 Unix Shell，Windows Explorer 是一个典型的图形界面 Shell。

Shell 在线工具

1.2 Shell 脚本

Shell 脚本（shell script），是一种为 shell 编写的脚本程序。 业界所说的 shell 通常都是指 shell 脚本，但读者朋友要知道，shell 和 shell script 是两个不同的概念。由于习惯的原 因，简洁起见，本文出现的 "shell 编程" 都是指 shell 脚本编程，不是指开发 shell 自身。

1.3 Shell 环境

Shell 编程跟 java、php 编程一样，只要有一个能编写代码的文本编辑器和一个能解释执行的脚本解释器就可以了。

Linux 的 Shell 种类众多，常见的有：

Bourne Shell（/usr/bin/sh 或/bin/sh）

Bourne Again Shell（/bin/bash）

C Shell（/usr/bin/csh）

K Shell（/usr/bin/ksh）

Shell for Root（/sbin/sh）

……

而我们学习的是 Bash，也就是 Bourne Again Shell，由于易用和免费，Bash 在日常工作中被广泛使用。同时，Bash 也是大多数 Linux 系统默认的 Shell。在一般情况下，人们并不区分 Bourne Shell 和 Bourne Again Shell，所以，像 #!/bin/sh，它同样也可以改为 #!/bin/bash。

1.4 sh/bash/csh/Tcsh/ksh 等 shell 的区别

sh(全称 Bourne Shell): 是 UNIX 最初使用的 shell，而且在每种 UNIX 上都可以使用。

Bourne Shell 在 shell 编程方面相当优秀，但在处理与用户的交互方面做得不如其他几种 shell。

bash（全称 Bourne Again Shell）: LinuxOS 默认的，它是 Bourne Shell 的扩展。与 Bourne Shell 完全兼容，并且在 Bourne Shell 的基础上增加了很多特性。可以提供命令补全，命令编辑和命令历史等功能。它还包含了很多 C Shell 和 Korn Shell 中的优点，有灵活和强大的编辑接口， 同时又很友好的用户界面。

csh(全称 C Shell): 是一种比 Bourne Shell 更适合的变种 Shell，它的语法与 C 语言很相似。

Tcsh: 是 Linux 提供的 C Shell 的一个扩展版本。Tcsh 包括命令行编辑，可编程单词补全， 拼写校正，历史命令替换，作业控制和类似 C 语言的语法，他不仅和 Bash Shell 提示符兼容，而 且还提供比 Bash Shell 更多的提示符参数。

ksh (全称 Korn Shell): 集合了 C Shell 和 Bourne Shell 的优点并且和 Bourne Shell 完全 兼容。

1.5 编写第一个 shell 脚本

打开文本编辑器(可以使用 vi/vim 命令来创建文件)，新建一个文件 test.sh，扩展名为 sh（sh 代表 shell），扩展名并不影响脚本执行，见名知意就好，如果你用 php 写 shell 脚本，扩展名就用 php 好了。 输入一些代码，第一行一般是这样：

例：#!/bin/bash

```
echo "Hello World !"
```

运行结果：

#! 是一个约定的标记，它告诉系统这个脚本需要什么解释器来执行，即使用哪一种 Shell。echo 命令用于向窗口输出文本。

1.6 运行 Shell 脚本有两种方法：

1.6.1 作为可执行程序

将上面的代码保存为 test.sh，并 cd 到相应目录：

```
chmod +x ./test.sh  #使脚本具有执行权限
./test.sh  #执行脚本
```

注意，一定要写成 ./test.sh，而不是 test.sh，运行其它二进制的程序也一样，直接写 test.sh，linux 系统会去 PATH 里寻找有没有叫 test.sh 的，而只有 /bin, /sbin, /usr/bin，/usr/sbin 等在 PATH 里，你的当前目录通常不在 PATH 里，所以写成 test.sh 是会找不到命令的，要用 ./test.sh 告诉系统说，就在当前目录找。

1.6.2 作为解释器参数

这种运行方式是，直接运行解释器，其参数就是 shell 脚本的文件名，如：

```
/bin/sh test.sh
```

这种方式运行的脚本，不需要在第一行指定解释器信息，写了也没用。

二、 Shell 变量

2.1 shell 变量

定义变量时，变量名不加美元符号（$，PHP 语言中变量需要），如：your_name="runoob.com"

注意，变量名和等号之间不能有空格，这可能和你熟悉的所有编程语言都不一样。同时，变量名的命

2.1.1 名须遵循如下规则：

命名只能使用英文字母，数字和下划线，首个字符不能以数字开头。

中间不能有空格，可以使用下划线（_）。

不能使用标点符号。

不能使用 bash 里的关键字（可用 help 命令查看保留关键字）。

有效的 Shell 变量名示例如下：

```
RUNOOB
LD_LIBRARY_PATH
_var
var2
```

无效的变量命名：

```
?var=123
user*name=runoob
```

以上语句将 /etc 下目录的文件名循环出来。

2.1.2 使用变量

使用一个定义过的变量，只要在变量名前面加美元符号即可，如：

```
your_name="qinjx"
echo $your_name
echo ${your_name}
```

变量名外面的花括号是可选的，加不加都行，加花括号是为了帮助解释器识别变量的边界，比如下面这种情况：

```
for skill in Ada Coffe Action Java; do
    echo "I am good at ${skill}Script"
done
```

如果不给 skill 变量加花括号，写成 echo "I am good at $skillScript"，解释器就会把$skillScript 当成一个变量（其值为空），代码执行结果就不是我们期望的样子了。 推荐给所有变量加上花括号，这是个好的编程习惯。

已定义的变量，可以被重新定义，如：

```
your_name="tom"
echo $your_name
your_name="alibaba"
echo $your_name
```

这样写是合法的，但注意，第二次赋值的时候不能写$your_name="alibaba"，使用变量的时候才$符号。

2.1.3 只读变量

使用 readonly 命令可以将变量定义为只读变量，只读变量的值不能被改变。

下面的例子尝试更改只读变量，结果报错：

```
#!/bin/bash
myUrl="uplooking"
readonly myUrl
myUrl="uplooking.com"
```

运行脚本，结果如下：

```
/bin/sh: NAME: This variable is read only.
```

2.1.4 删除变量

使用 unset 命令可以删除变量。语法：

```
unset variable_name
```

变量被删除后不能再次使用。unset 命令不能删除只读变量。

例：

```
#!/bin/sh
myUrl="uplooking"
unset myUrl
echo $myUrl
```

以上实例执行将没有任何输出。

2.1.5 变量类型

运行 shell 时，会同时存在三种变量：

1) 局部变量 局部变量在脚本或命令中定义，仅在当前 shell 实例中有效，其他 shell 启动的程序不能访问局部变量。

2) 环境变量 所有的程序，包括 shell 启动的程序，都能访问环境变量，有些程序需要环境变量来保证其正常运行。必要的时候 shell 脚本也可以定义环境变量。

3) shell 变量 shell 变量是由 shell 程序设置的特殊变量。shell 变量中有一部分是环境变量，有一部分是局部变量，这些变量保证了 shell 的正常运行

2.2 Shell 字符串

字符串是 shell 编程中最常用最有用的数据类型（除了数字和字符串，也没啥其它类型好用了），字符串可以用单引号，也可以用双引号，也可以不用引号。

2.2.1 单引号

```
str='this is a string'
```

单引号字符串的限制：

单引号里的任何字符都会原样输出，单引号字符串中的变量是无效的；

单引号字串中不能出现单引号（对单引号使用转义符后也不行）。

2.2.2 双引号

```
your_name='qinjx'
str="Hello, I know your are \"$your_name\"! \n"
```

双引号的优点：

双引号里可以有变量

双引号里可以出现转义字符

2.2.3 获取字符串长度

```
string="abcd"
echo ${#string} #输出 4
```

2.3 Shell 注释

以"#"开头的行就是注释，会被解释器忽略。

sh 里没有多行注释，只能每一行加一个#号。只能像这样：

```
#--------------------------------------------
# 这是一个注释
# author：King
# site：www.uplooking.com
#--------------------------------------------
```

三、 Shell 传递参数

我们可以在执行 Shell 脚本时，向脚本传递参数，脚本内获取参数的格式为：$n。n 代表一个数字，1 为执行脚本的第一个参数，2 为执行脚本的第二个参数，以此类推……

例：以下实例我们向脚本传递三个参数，并分别输出，其中 $0 为执行的文件名：

```
#!/bin/bash
echo "Shell 传递参数实例！";
echo "执行的文件名：$0";
echo "第一个参数为：$1";
echo "第二个参数为：$2";
echo "第三个参数为：$3";
```

为脚本设置可执行权限，并执行脚本，输出结果如下所示：

```
$ chmod +x test.sh
$ ./test.sh 1 2 3
执行的文件名：./test.sh
第一个参数为：1
第二个参数为：2
第三个参数为：3
```

另外，还有几个特殊字符用来处理参数：

$# 传递到脚本的参数个数

$* 以一个单字符串显示所有向脚本传递的参数。

如"$*"用「"」括起来的情况、以"$1 $2 … $n"的形式输出所有参数。

$$ 脚本运行的当前进程 ID 号

$! 后台运行的最后一个进程的 ID 号

$@ 与$*相同，但是使用时加引号，并在引号中返回每个参数。

如"$@"用「"」括起来的情况、以"$1" "$2" … "$n" 的形式输出所有参数。

$- 显示 Shell 使用的当前选项，与 set 命令功能相同。

$? 显示最后命令的退出状态。0 表示没有错误，其他任何值表明有错误。

```
#!/bin/bash
echo "Shell 传递参数实例！";
```

```
echo "第一个参数为：$1";
echo "参数个数为：$#";
echo "传递的参数作为一个字符串显示：$*";
```

执行脚本，输出结果如下所示：

```
$ chmod +x test.sh
$ ./test.sh 1 2 3
Shell 传递参数实例！
第一个参数为：1
参数个数为：3
传递的参数作为一个字符串显示：1 2 3
```

$* 与 $@ 区别：

相同点：都是引用所有参数。

不同点：只有在双引号中体现出来。假设在脚本运行时写了三个参数 1、2、3，，则 "*" 等价于 "1 2 3"（传递了一个参数），而 "@" 等价于 "1" "2" "3"（传递了三个参数）。

```
echo "-- \$* 演示 ---"
for i in "$*"; do
echo $i
done
echo "-- \$@ 演示 ---"
for i in "$@"; do
echo $i
done
```

执行脚本，输出结果如下所示：

```
$ chmod +x test.sh
$ ./test.sh 1 2 3
-- $* 演示 ---
1 2 3
-- $@ 演示 ---
1
2
3
```

四、 数组

数组中可以存放多个值。Bash Shell 只支持一维数组（不支持多维数组），初始化时不需要定义数组大小。 与大部分编程语言类似，数组元素的下标由 0 开始。

4.1 数组的表示方法

Shell 数组用括号来表示，元素用"空格"符号分割开，语法格式如下：

```
array_name=(value1 ... valuen)
```

例：

```
#!/bin/bash
my_array=(A B "C" D)
```

我们也可以使用下标来定义数组:

```
array_name[0]=value0
array_name[1]=value1
array_name[2]=value2
```

4.2 读取数组

读取数组元素值的一般格式是：${array_name[index]}

例：#!/bin/bash

```
my_array=(A B "C" D)
echo "第一个元素为: ${my_array[0]}"
echo "第二个元素为: ${my_array[1]}"
echo "第三个元素为: ${my_array[2]}"
echo "第四个元素为: ${my_array[3]}"
```

执行脚本，输出结果如下所示：

使用@ 或 * 可以获取数组中的所有元素，例如：

```
my_array[0]=A
my_array[1]=B
my_array[2]=C
my_array[3]=D
echo "数组的元素为: ${my_array[*]}"
echo "数组的元素为: ${my_array[@]}"
```

执行脚本，输出结果如下所示：

```
$ chmod +x test.sh
$ ./test.sh
```

数组的元素为: A B C D

数组的元素为: A B C D

4.3 获取数组的长度

获取数组长度的方法与获取字符串长度的方法相同，例如：

```
#!/bin/bash
my_array[0]=A
my_array[1]=B
my_array[2]=C
my_array[3]=D
echo "数组元素个数为: ${#my_array[*]}"
echo "数组元素个数为: ${#my_array[@]}"
```

执行脚本，输出结果如下所示：

```
$ chmod +x test.sh
$ ./test.sh
```

数组元素个数为: 4

数组元素个数为: 4

五、 Shell 基本运算符

Shell 和其他编程语言一样，支持多种运算符，包括：算数运算符，关系运算符，布尔运算符，字符串运算符，文件测试运算符。原生 bash 不支持简单的数学运算，但是可以通过其他命令来实现，例如 awk 和 expr，expr 最常用。 expr 是一款表达式计算工具，使用它能完成表达式的求值操作。

例：两个数相加(注意使用的是反引号 ` 而不是单引号 '):

```
#!/bin/bash
val=`expr 2 + 2`
echo "两数之和为 : $val"
```

执行脚本，输出结果如下所示：

两数之和为 : 4

两点注意：

表达式和运算符之间要有空格，例如 2+2 是不对的，必须写成 2 + 2，这与我们熟悉的大多数编程语言不一样。

完整的表达式要被 `` 包含，注意这个字符不是常用的单引号，在 Esc 键下边。

5.1 算术运算符

下表列出了常用的算术运算符，假定变量 a 为 10，变量 b 为 20：

运算符	说明	举例
+	加法	`expr $a + $b` 结果为 30。
-	减法	`expr $a - $b` 结果为 -10。
*	乘法	`expr $a \* $b` 结果为 200。
/	除法	`expr $b / $a` 结果为 2。
%	取余	`expr $b % $a` 结果为 0。
=	赋值	a=$b 将把变量 b 的值赋给 a。
==	相等。用于比较两个数字，相同则返回 true。	[$a == $b] 返回 false。
!=	不相等。用于比较两个数字，不相同则返回 true。	[$a != $b] 返回 true。

注意：条件表达式要放在方括号之间，并且要有空格，例如: [$a==$b] 是错误的，必须写成 [$a == $b]。

例：算术运算符实例如下：

```
#!/bin/bash
a=10
      5.2  b=20
val=`expr $a + $b`
      echo "a + b : $val"
val=`expr $a \* $b`
      echo "a * b : $val"
val=`expr $b / $a`
      echo "b / a : $val"
if [ $a == $b ]；then
      echo "a 等于 b"
fi
if [ $a != $b ]；then
      echo "a 不等于 b"
fi
执行脚本，输出结果如下所示：
a + b : 30
a * b : 200
b / a : 2
a 不等于 b
```

注意：乘号(*)前边必须加反斜杠(\)才能实现乘法运算；if...then...fi 是条件语句，后续将 会讲解。

5.2 关系运算符

关系运算符只支持数字，不支持字符串，除非字符串的值是数字。

下表列出了常用的关系运算符，假定变量 a 为 10，变量 b 为 20：

运算符	说明	举例
-eq	检测两个数是否相等，相等返回 true。	[$a -eq $b] 返回 false。
-ne	检测两个数是否相等，不相等返回 true。	[$a -ne $b] 返回 true。
-gt	检测左边的数是否大于右边的，如果是，则返回 true。	[$a -gt $b] 返回 false。
-lt	检测左边的数是否小于右边的，如果是，则返回 true。	[$a -lt $b] 返回 true。
-ge	检测左边的数是否大于等于右边的，如果是，则返回 true。	[$a -ge $b] 返回 false。
-le	检测左边的数是否小于等于右边的，如果是，则返回 true。	[$a -le $b] 返回 true。

例:关系运算符实例如下：

```
#!/bin/bash
a=10
b=20
if [ $a -eq $b ];then
      echo "$a -eq $b : a 等于 b"
else
      echo "$a -eq $b: a 不等于 b"
fi
if [ $a -gt $b ];then
      echo "$a -gt $b: a 大于 b"
else
      echo "$a -gt $b: a 不大于 b"
fi
if [ $a -le $b ];then
     echo "$a -le $b: a 小于或等于 b"
else
    echo "$a -le $b: a 大于 b"
fi
执行脚本，输出结果如下所示：
10 -eq 20: a 不等于 b
10 -gt 20: a 不大于 b
10 -le 20: a 小于或等于 b
```

5.3 布尔运算符

下表列出了常用的布尔运算符，假定变量 a 为 10，变量 b 为 20：

运算符	说明	举例

!	非运算，表达式为 true 则返回 false，否则返回 true。	[! false] 返回 true。
-o	或运算，有一个表达式为 true 则返回 true。	[$a -lt 20 -o $b -gt 100] 返回 true。
-a	与运算，两个表达式都为 true 才返回 true。	[$a -lt 20 -a $b -gt 100] 返回 false。

例：布尔运算符实例如下：

```
#!/bin/bash
a=10
b=20
if [ $a != $b ];then
    echo "$a != $b : a 不等于 b"
else
    echo "$a != $b: a 等于 b"
fi
if [ $a -lt 100 -a $b -gt 15 ];then
    echo "$a 小于 100 且 $b 大于 15：返回 true"
else
    echo "$a 小于 100 且 $b 大于 15：返回 false"
fi
if [ $a -lt 100 -o $b -gt 100 ];then
    echo "$a 小于 100 或 $b 大于 100：返回 true"
else
    echo "$a 小于 100 或 $b 大于 100：返回 false"
fi
if [ $a -lt 5 -o $b -gt 100 ];then
    echo "$a 小于 5 或 $b 大于 100：返回 true"
else
    echo "$a 小于 5 或 $b 大于 100：返回 false"
fi
```

执行脚本，输出结果如下所示：

```
10 != 20 : a 不等于 b
10 小于 100 且 20 大于 15：返回 true
10 小于 100 或 20 大于 100：返回 true
10 小于 5 或 20 大于 100：返回 false
```

5.4 逻辑运算符

下表列出了常用的逻辑运算符，假定变量 a 为 10，变量 b 为 20:

运算符	说明	举例
&&	逻辑的 AND	[[$a -lt 100 && $b -gt 100]] 返回 false
||	逻辑的 OR	[[$a -lt 100 || $b -gt 100]] 返回 true

逻辑运算符实例如下：

```
#!/bin/bash
```

```
a=10
b=20
if [[ $a -lt 100 && $b -gt 100 ]];then
    echo "返回 true"
else
    echo "返回 false"
fi
if [[ $a -lt 100 || $b -gt 100 ]];then
    echo "返回 true"
else
    echo "返回 false"
fi
```

执行脚本，输出结果如下所示：

```
返回 false
返回 true
```

5.5 字符串运算符

下表列出了常用的字符串运算符：假定变量 a 为 "abc"，变量 b 为 "efg"

运算符	说明	举例
=	检测两个字符串是否相等，相等返回 true。	[$a = $b] 返回 false。
!=	检测两个字符串是否相等，不相等返回 true。	[$a != $b] 返回 true。
-z	检测字符串长度是否为 0，为 0 返回 true。	[-z $a] 返回 false。
-n	检测字符串长度是否为 0，不为 0 返回 true。	[-n $a] 返回 true。
Str	检测字符串是否为空，不为空返回 true。	[$a] 返回 true。

字符串运算符实例如下：

```
#!/bin/bash
a="abc"
b="efg"
if [ $a = $b ];then
    echo "$a = $b : a 等于 b"
else
    echo "$a = $b: a 不等于 b"
fi
if [ -z $a ];then
    echo "-z $a : 字符串长度为 0"
else
    echo "-z $a : 字符串长度不为 0"
fi
if [ $a ];then
    echo "$a : 字符串不为空"
else
```

```
    echo "$a : 字符串为空"
fi
执行脚本，输出结果如下所示：
abc = efg: a 不等于 b
-z abc : 字符串长度不为 0
abc : 字符串不为空
```

5.6 文件测试运算符

文件测试运算符用于检测 Unix 文件的各种属性。 属性检测描述如下：

操作符	说明	举例
-b file	检测文件是否是块设备文件，如果是，则返回 true。	[-b $file] 返回 false。
-c file	检测文件是否是字符设备文件，如果是，则返回 true。	[-c $file] 返回 false。
-d file	检测文件是否是目录，如果是，则返回 true。	[-d $file] 返回 false。
-f file	检测文件是否是普通文件（既不是目录，也不是设备文件），如果是，则返回 true。	[-f $file] 返回 true。
-g file	检测文件是否设置了 SGID 位，如果是，则返回 true。	[-g $file] 返回 false。
-k file	检测文件是否设置了粘着位(Sticky Bit)，如果是，则返回 true。	[-k $file] 返回 false。
-p file	检测文件是否是有名管道，如果是，则返回 true。	[-p $file] 返回 false。
-u file	检测文件是否设置了 SUID 位，如果是，则返回 true。	[-u $file] 返回 false。
-r file	检测文件是否可读，如果是，则返回 true。	[-r $file] 返回 true。
-w file	检测文件是否可写，如果是，则返回 true。	[-w $file] 返回 true。
-x file	检测文件是否可执行，如果是，则返回 true。	[-x $file] 返回 true。
-s file	检测文件是否为空（文件大小是否大于 0），不为空返回 true。	[-s $file] 返回 true。
-e file	检测文件（包括目录）是否存在，如果是，则返回 true。	[-e $file] 返回 true。

例:变量 file 表示文件"/var/www/runoob/test.sh"，它的大小为 100 字节，具有 rwx 权限。下面的代码，将检测该文件的各种属性：

```
#!/bin/bash
file="/var/www/runoob/test.sh"
if [ -r $file ];then
    echo "文件可读"
else
    echo "文件不可读"
fi
if [ -w $file ];then
    echo "文件可写"
else
```

```
    echo "文件不可写"
fi
if [ -x $file ];then
    echo "文件可执行"
else
    echo "文件不可执行"
fi
if [ -f $file ];then
    echo "文件为普通文件"
else
    echo "文件为特殊文件"
fi
if [ -d $file ];then
    echo "文件是个目录"
else
    echo "文件不是个目录"
fi
if [ -s $file ];then
    echo "文件不为空"
else
    echo "文件为空"
fi
if [ -e $file ];then
    echo "文件存在"
else
    echo "文件不存在"
fi
```

执行脚本，输出结果如下所示：

```
文件可读
文件可写
文件可执行
文件为普通文件
文件不是个目录
文件不为空
文件存在
```

六、 Shell 流程控制

6.1 if 语句语法格式：

```
if condition
then
command1
    …
commandN
fi
```

写成一行（适用于终端命令提示符）：

```
if [ $(ps -ef | grep -c "ssh") -gt 1 ]; then echo "true"; fi
```

末尾的 fi 就是 if 倒过来拼写，后面还会遇到类似的。

6.2 if else 语法格式：

```
if condition
then
command1
    ...
commandN
else
command
fi
```

6.3 if else-if else 语法格式：

```
if condition1 ; then
    command1
elif condition2 ; then
command2
else
commandN
fi
```

以下实例判断两个变量是否相等：

```
a=10
b=20
if [ $a == $b ]
then
    echo "a 等于 b"
elif [ $a -gt $b ]
then
    echo "a 大于 b"
elif [ $a -lt $b ]
then
    echo "a 小于 b"
else
    echo "没有符合的条件"
fi
输出结果：
a 小于 b
```

6.4 for 循环

for 循环一般格式为：

```
for var in item1 item2 ... itemN
do
    command1
...
commandN
done
```

写成一行：

```
for var in item1 item2 ... itemN; do command1; command2… done;
```

当变量值在列表里，for 循环即执行一次所有命令，使用变量名获取列表中的当前取值。命令可为任何有效的 shell 命令和语句。in 列表可以包含替换、字符串和文件名。 in 列表是可选的，如果不用它，for 循环使用命令行的位置参数。

例：顺序输出当前列表中的数字：

```
For loop in 1 2 3 4 5
do
    echo "The value is: $loop"
done
输出结果：
The value is: 1
The value is: 2
The value is: 3
The value is: 4
The value is: 5
顺序输出字符串中的字符：
for str in 'This is a string'
do
    echo $str
done
输出结果：
This is a string
```

6.5 while 语句

while 循环用于不断执行一系列命令，也用于从输入文件中读取数据；命令通常为测试条件。其格式为：

```
while condition
do
command
done
```

以下是一个基本的 while 循环，测试条件是：如果 int 小于等于 5，那么条件返回真。int 从 0 开始，每次循环处理时，int 加 1。运行上述脚本，返回数字 1 到 5，然后终止。

```
#!/bin/sh
int=1
while(( $int<=5 ))
do
    echo $int
    let "int++"
done
运行脚本，输出：
1
2
3
4
5
```

使用中使用了 Bash let 命令，它用于执行一个或多个表达式，变量计算中不需要加上 $ 来表示变量，具体可查阅：Bash let 命令。

6.6 until 循环

until 循环执行一系列命令直至条件为真时停止。 until 循环与 while 循环在处理方式上刚好反。一般 while 循环优于 until 循环，但在某些时候—也只是极少数情况下，until 循环更加有用。

until 语法格式:

```
until condition
do
        command
done
```

条件可为任意测试条件，测试发生在循环末尾，因此循环至少执行一次—请注意这一点。

6.7 case

Shell case 语句为多选择语句。可以用 case 语句匹配一个值与一个模式，如果匹配成功，执行相匹配的命令。case 语句格式如下：

```
case 值 in
模式 1)
command1
        ...
commandN
;;
模式 2）
command1
       ...
commandN
        ;;
esac
```

case 工作方式如上所示。取值后面必须为单词 in，每一模式必须以右括号结束。取值可以为变量或常数。匹配发现取值符合某一模式后，其间所有命令开始执行直至 ;;。取值将检测匹配的每一个模式。一旦模式匹配，则执行完匹配模式相应命令后不再继续其他模式。如果无一匹配模式，使用星号 * 捕获该值，再执行后面的命令。

下面的脚本提示输入 1 到 4，与每一种模式进行匹配：

```
echo '输入 1 到 4 之间的数字:'
echo '你输入的数字为:'
read aNum
case $aNum in
    1)  echo '你选择了 1'
    ;;
    2)  echo '你选择了 2'
    ;;
    3)  echo '你选择了 3'
    ;;
    4)  echo '你选择了 4'
    ;;
    *)  echo '你没有输入 1 到 4 之间的数字'
    ;;
esac
```

输入不同的内容，会有不同的结果，例如：

```
输入 1 到 4 之间的数字:
你输入的数字为:
3
你选择了 3
```

6.8 跳出循环

在循环过程中，有时候需要在未达到循环结束条件时强制跳出循环，Shell 使用两个命令实现现该功能：break 和 continue。

6.8.1 break 命令允许跳出所有循环（终止执行后面的所有循环）。

下面的例子中，脚本进入死循环直至用户输入数字大于 5。要跳出这个循环，返回到 shell 提示符下，需要使用 break 命令。

```
#!/bin/bash
while :
do
    echo -n "输入 1 到 5 之间的数字:"
    read aNum
    case $aNum in
        1|2|3|4|5) echo "你输入的数字为 $aNum!"
        ;;
        *) echo "你输入的数字不是 1 到 5 之间的！游戏结束"
            break
        ;;
    esac
done
执行以上代码，输出结果为：
输入 1 到 5 之间的数字:3
你输入的数字为 3!
输入 1 到 5 之间的数字:7
你输入的数字不是 1 到 5 之间的！游戏结束
```

6.8.2 continue 命令与 break 命令类似，只有一点差别，它不会跳出所有循环，仅跳出当前循环。

对上面的例子进行修改：

```
#!/bin/bash
while :
do
    echo -n "输入 1 到 5 之间的数字: "
    read aNum
    case $aNum in
        1|2|3|4|5) echo "你输入的数字为 $aNum!"
        ;;
        *) echo "你输入的数字不是 1 到 5 之间的!"
            continue
            echo "游戏结束"
        ;;
    esac
done
```

运行代码发现，当输入大于 5 的数字时，该例中的循环不会结束，语句 echo "游戏结束" 永远不会被执行。

6.8.3 esac

case 的语法和 C family 语言差别很大，它需要一个 esac（就是 case 反过来）作为结束标记，每个 case 分支用右圆括号，用两个分号表示 break。

七、 Shell 函数

linux shell 可以用户定义函数，然后在 shell 脚本中可以随便调用。

7.1 shell 中函数的定义格式如下：

```
[ function ] funname [()]
{
    action;
    [return int;]
}
```

说明：

1、可以带 function fun() 定义，也可以直接 fun() 定义,不带任何参数。

2、参数返回，可以显示加：return 返回，如果不加，将以最后一条命令运行结果，作为返回值。return 后跟数值 n(0-255)

下面的例子定义了一个函数并进行调用：

```
#!/bin/bash
demoFun(){
    echo "这是我的第一个 shell 函数!"
}
echo "-----函数开始执行-----"
demoFun
echo "-----函数执行完毕-----"
输出结果：
-----函数开始执行-----
这是我的第一个 shell 函数!
-----函数执行完毕-----
```

下面定义一个带有 return 语句的函数：

```
#!/bin/bash
funWithReturn(){
    echo "这个函数会对输入的两个数字进行相加运算..."
    echo "输入第一个数字: "
    read aNum
    echo "输入第二个数字: "
    read anotherNum
    echo "两个数字分别为 $aNum 和 $anotherNum !"
    return $(($aNum+$anotherNum))
}
funWithReturn
echo "输入的两个数字之和为 $? !"
输出结果：
```

```
这个函数会对输入的两个数字进行相加运算...
输入第一个数字:
1
输入第二个数字:
2
两个数字分别为 1 和 2 !
输入的两个数字之和为 3 !
```

函数返回值在调用该函数后通过 $? 来获得。

注意：所有函数在使用前必须定义。这意味着必须将函数放在脚本开始部分，直至 shell 解释器首次发现它时，才可以使用。调用函数仅使用其函数名即可。

7.2 函数参数

在 Shell 中，调用函数时可以向其传递参数。在函数体内部，通过 $n 的形式来获取参数的值，例如，$1 表示第一个参数，$2 表示第二个参数...

带参数的函数示例：

```
#!/bin/bash
funWithParam(){
    echo "第一个参数为 $1 !"
    echo "第二个参数为 $2 !"
    echo "第十个参数为 $10 !"
    echo "第十个参数为 ${10} !"
    echo "第十一个参数为 ${11} !"
    echo "参数总数有 $# 个!"
    echo "作为一个字符串输出所有参数 $* !"
}
funWithParam 1 2 3 4 5 6 7 8 9 34 73
输出结果：
第一个参数为 1 !
第二个参数为 2 !
第十个参数为 10 !
第十个参数为 34 !
第十一个参数为 73 !
参数总数有 11 个!
作为一个字符串输出所有参数 1 2 3 4 5 6 7 8 9 34 73 !
```

注意，$10 不能获取第十个参数，获取第十个参数需要${10}。当 n>=10 时，需要使用${n}来获取参数。

7.3 函数调试

7.3.1 通过 echo 方式

功能: 最简单的调试方法，可以在任何怀疑出错的地方用 echo 打印变量

场合: 所有怀疑可能有问题的地方

示例: echo $VAR

7.3.2 -n

读一遍脚本中的命令但不执行，用于检查脚本中的语法错误

7.3.3 -v

一边执行脚本，一边将执行过的脚本命令打印到标准错误输出

7.3.4 -x

提供跟踪执行信息，将执行的每一条命令和结果依次打印出来
使用这些选项有三种方法：
一是在命令行提供参数 $ sh -x ./script.sh
二是在脚本开头提供参数 #! /bin/sh –x
三是在脚本中用 set 命令启用或禁用参数

```
#! /bin/sh
if [ -z "$1" ]; then
set -x
echo "ERROR: Insufficient Args."
exit 1
```