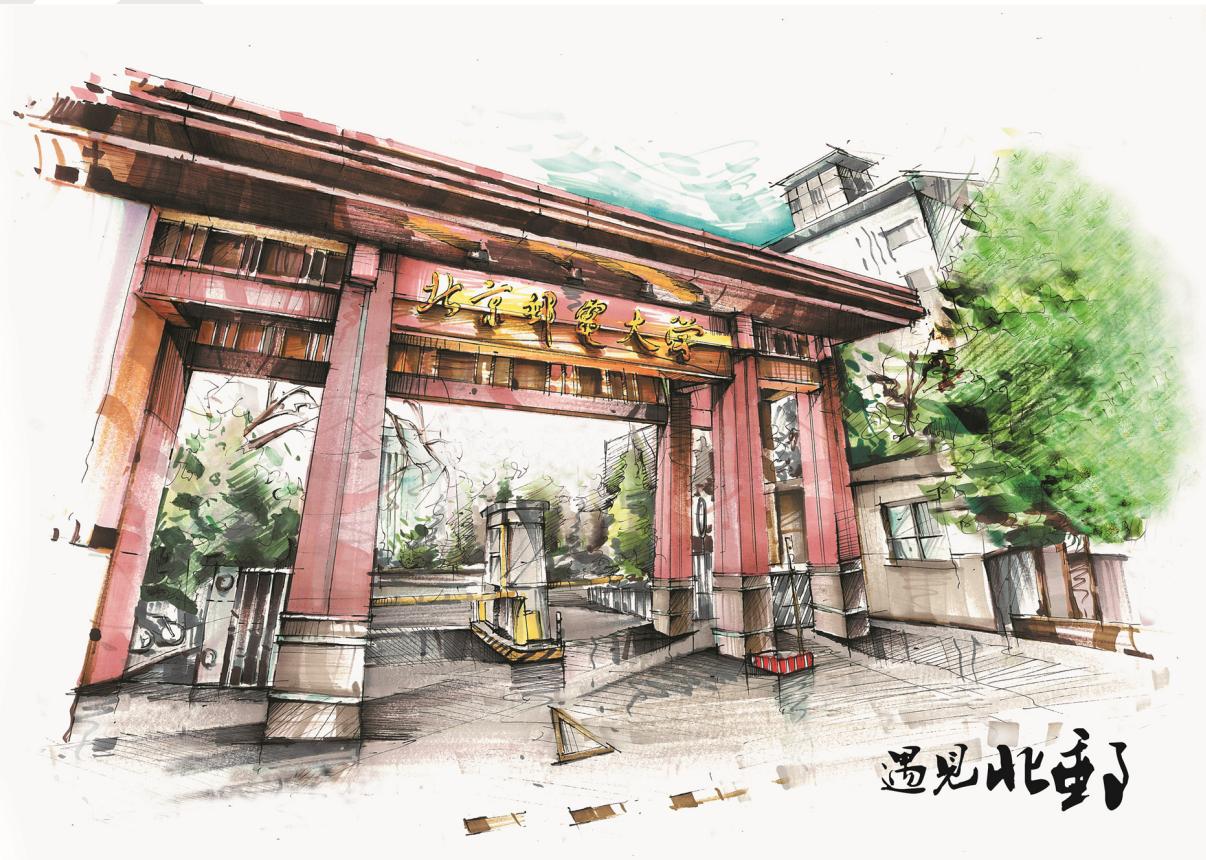


鸟哥的 Linux 私房菜-基础篇 第四版

Linux 学习笔记

YOUNG@BUPT



2019/5/10

Contents

1 第 1 章 Linux 是什么与如何学习	5
1.1 Linux 是什么	5
1.1.1 Linux 是什么	5
1.1.2 Unix 历史	5
1.1.3 GNU 计划、自由软件与开放源代码	5
1.2 Linux 发展	6
1.2.1 发展历程	6
1.2.2 核心版本	6
1.2.3 Linux distributions	7
1.3 Linux 使用场景	7
1.4 Linux 该如何学习	7
2 第 2 章 主机规划与磁盘分区	8
2.1 安装过程说明	8
2.2 虚拟机安装软件	8
2.3 需要了解的知识点	8
3 第 3 章 Linux 安装	9
4 第 4 章首次登入与在线求助	10
4.1 首次登入系统	10
4.2 使用命令行方式进行操作	10
4.3 在线求助 man page 与 info page	10
4.3.1 通用模式	10
4.3.2 man	10
4.3.3 info	11
4.3.4 联机帮助文件	11
4.3.5 一个极轻量级编辑器	12
4.3.6 正确开关机	12
5 第 5 章 Linux 的文件权限与目录配置	13
5.1 用户与群组	13
5.2 文件权限	13
5.2.1 文件属性	13
5.2.2 权限的修改	13
5.2.3 目录与文件的权限含义	14
5.3 目录配置	14
5.3.1 Linux 目录配置	14
5.3.2 目录树	18
5.3.3 绝对路径与相对路径	19
6 第 6 章 Linux 文件与目录管理	20
6.1 目录与路径	20
6.1.1 cd - 切换目录	20
6.1.2 pwd - 显示当前目录	20
6.1.3 mkdir - 创建一个新目录	20
6.1.4 rmdir - 删除一个空目录	20
6.1.5 \$PATH 环境变量	20
6.2 文件与目录管理	20
6.2.1 ls - 目录列表	20
6.2.2 cp - 拷贝文件	21
6.2.3 rm - 删除文件	21
6.2.4 mv - 移动文件或目录	22
6.2.5 basename - 获得文件名称	22
6.2.6 dirname - 获得目录名	22
6.3 文件内容查询	22
6.3.1 cat - 查看文件内容	22

6.3.2	tac - 反向查看文件内容	22
6.3.3	nl - 添加行号显示	22
6.3.4	more/less - 分页查看文件内容	23
6.3.5	head - 查看文件前几行	23
6.3.6	tail - 查看文件后面几行	23
6.3.7	od - octal, decimal, hex, ASCII dump	23
6.3.8	touch - 创建空文件或更新已有文件时间	24
6.4	文件与目录的默认权限	24
6.4.1	umask - 用户创建文件或目录时的权限默认值	24
6.4.2	文件隐藏属性	24
6.4.3	文件的特殊权限	24
6.4.4	stat - 查看文件或操作系统状态	25
6.4.5	file - 查看文件类型	27
6.5	指令与文件的搜索	27
6.5.1	which - 查看某个命令位置	27
6.5.2	whereis - 查找文件, 给出文件目录和手册页	27
6.5.3	locate/updatedb - 基于数据库进行文件查找	27
6.5.4	find - 按照各种规则查找指定文件并在其上执行所需操作	27
7	第 7 章 Linux 磁盘与文件系统管理	29
7.1	Linux 文件系统	29
7.1.1	磁盘组成与分区	29
7.1.2	文件系统特性	29
7.1.3	inode (ext2)	30
7.1.4	与目录树的关系	33
7.1.5	文件的存取与日志式文件系统功能	34
7.1.6	Linux 文件系统的运作	34
7.1.7	挂载点 (mount point)	34
7.1.8	Linux 支持的文件系统	34
7.2	文件系统相关操作	34
7.2.1	磁盘与目录的容量	34
7.2.2	硬链接与符号链接 ln	35
7.3	磁盘的分区、格式化、检验与挂载	35
7.3.1	查看磁盘状态	35
7.3.2	磁盘分区	35
7.3.3	磁盘格式化	35
7.3.4	文件系统检验	35
7.3.5	文件系统挂载与卸除	35
7.3.6	磁盘/文件系统参数修订	36
7.4	设定开机挂载	36
7.4.1	开机挂载/etc/fstab 及 /etc/mtab	36
7.4.2	挂载特殊装置	36
7.5	关于 swap 分区	36
8	第 8 章文件与文件系统的压缩打包与备份	37
8.1	压缩文件类型	37
8.2	常见压缩命令	37
8.2.1	gzip, zcat/zmore/zless/zgrep	37
8.2.2	bzip2, bzcat/bzmore/bzless/bzgrep	37
8.2.3	xz, xzcat/xzmore/xzless/xzgrep	38
8.3	打包指令 tar	38
8.4	xfs 文件系统备份	39
8.5	光盘相关	39
8.6	其他常见工具	40
8.6.1	dd	40
8.6.2	cpio	40
9	vim 编辑器	41

9.1 编辑器的历史	41
9.2 vi 与 vim	41
9.3 vi 的使用	42
9.3.1 简单示例	43
9.3.2 按键说明	43
9.3.3 案例练习	46
9.3.4 其他相关知识	46
9.3.5 区块选择	47
9.3.6 多文件编辑	47
9.3.7 多窗口功能	47
9.3.8 名称补全功能	48
9.3.9 宏	48
9.3.10 书签	48
9.3.11 其他	48
9.3.12 环境设定	49
9.3.13 vim cheat code	49
9.4 其他问题	58
9.4.1 中文编码	58
9.4.2 unix2dos 和 dos2unix	58
9.4.3 编码转换命令	58
9.5 其他示例	58
10 认识与学习 BASH	59
10.1 认识 BASH	59
10.1.1 操作系统与 Shell	59
10.1.2 Shell 简史	59
10.1.3 /etc/shells 文件	59
10.1.4 bash 的功能	59
10.2 查询指令是否为 bash 的内建命令	59
10.3 命令下达和简单的命令行编辑	60
10.4 Shell 的变量功能	60
10.4.1 什么是变量	60
10.4.2 变量的设定与使用	60
10.4.3 环境变量	60
10.5 locale	61
10.5.1 交互读入变量数组和声明	61
10.5.2 ulimit 限制用户使用的系统资源	62
10.5.3 变量内容的删除、取代与替换	62
10.6 命令别名与历史命令	65
10.7 Bash shell 的操作系统	65
10.7.1 路径与命令搜索顺序	65
10.7.2 /etc/issue, /etc/motd	65
10.7.3 bash 的环境配置文件	65
10.7.4 终端设定 stty, set	66
10.8 重定向功能	67
10.9 管道功能	68
10.9.1 cut grep	68
10.9.2 sort wc uniq	68
10.9.3 tee	69
10.9.4 tr col join paste expand	69
10.9.5 split	70
10.9.6 xargs	71
10.9.7 关于减号	71
11 正则表达式与文件格式化处理	72
11.1 什么是正则表达式	72
11.2 基本正则表达式	72
11.2.1 注意语系的问题	72

11.2.2 与 grep 结合使用	72
11.2.3 grep 搜索示例	72
11.2.4 特殊字符汇总	73
11.2.5 sed 工具	74
11.3 扩展正则表达式	75
11.4 文件的格式化与相关处理	75
11.4.1 格式化打印: printf	75
11.4.2 awk	76
11.4.3 文件对比工具	76
12 学习 Shell Scripts	78
12.1 什么是 Shell scripts	78
12.2 简单的 Shell scripts	78
12.2.1 几个简单脚本	78
12.2.2 script 的执行方式	80
12.3 判断式	80
12.3.1 利用判断符号 []	81
12.3.2 shell 脚本的命令行参数	82
12.4 条件判断式	83
12.4.1 if ... then	83
12.4.2 case ... esac	85
12.4.3 函数功能	86
12.5 循环	87
12.5.1 while 和 until 循环	87
12.5.2 for 循环	88
12.5.3 传统 for 循环	90
12.5.4 使用数组及随机数	90
12.6 shell scripts 的调试	91
13 第 13 章 Linux 账号管理与 ACL 权限设定	92
13.1 Linux 账号与群组	92
13.1.1 使用者标识符	92
13.1.2 使用者账号	92
13.1.3 关于群组: 有效与初始群组、groups、newgrp	92
13.2 账号管理	92
13.2.1 添加与删除账号	92
13.2.2 用户功能	93
13.2.3 添加或删除组	93
13.3 主机的权限规划	93
13.3.1 什么是 ACL 及如何支持启动 ACL	93
13.3.2 ACL 设定: getfacl, setfacl	93
13.3.3 setfacl 设定某个文件的 ACL 权限	93
13.3.4 getfacl 取得某个文件的 ACL 设定	93
13.4 使用者身份切换	94
13.4.1 su 身份切换	94
13.4.2 sudo	94
13.5 用户的特殊 shell 与 PAM 模块	94
13.6 Linux 上的用户通信	94
14 附录	95
14.1 使用到的脚本	95
14.1.1 md2pdf.sh	95
14.1.2 head.tex	95
14.1.3 批量 md 文件处理脚本	96
14.1.4 生成的 pdf 添加水印	97

1 第 1 章 Linux 是什么与如何学习

1.1 Linux 是什么

1.1.1 Linux 是什么

- 一套操作系统，与 Windows 并列
- 创建于 1991 年，Linus Torvalds，芬兰赫尔辛基

1.1.2 Unix 历史

- 产生于上世纪 70 年代
- 1969 年以前，Bell/MIT/GE 的 Multics 系统，参考 <http://www.multicians.org/>
- 1969 年，Ken Thompson 的 File Server System(PDP-7, DEC)，发展为 Unics，使用汇编编写，包括的重要概念
 - 一切程序或系统装置都是文件
 - 编写程序的唯一目的就是有效地完成目标
- 1973 年，Unix 诞生，K & R(Ken Thompson & Ritchie) 使用修改后的 B 语言 (C 语言) 编写，AT&T 贝尔实验室
- 1977 年，与加州伯克利 (Berkeley) 大学合作，BSD(Berkeley Software Distribution) 分支产生，Bill Joy(SUN 创始人)
- 1979 年，System V (AT&T) 架构与版权宣告，Unix 开始可以安装于个人计算机
- 1984 年，Minix 开始撰写并于 2 年后诞生，Andrew Tanenbaum(谭宁邦)，Mini Unix，不是免费的
- 1984 年，GNU 计划与 FSF 成立，Richard Mathew Stallman, <http://www.fsf.org/resources>
 - GNU C Compiler
 - Emacs
 - GNU C Library (glibc)
 - Bash Shell
 - GPL (General Public License), copyleft
 - hurd (失败的操作系统内核)
- 1988 年，XFree86，GUI，X Window System + Free + X86
 - MIT 和其他机构于 1984 年推出 X Window System
 - 1988 年推出 XFree86
 - 1994 年融入 Linux 1.0
- 1991 年，芬兰赫尔辛基大学，Linus 在 BBS 上宣称，以 bash gcc 等 GNU 工具写了一个小小的核心程序

1.1.3 GNU 计划、自由软件与开放源代码

1.1.3.1 FSF 核心观念：

- 版权制度是促进社会进步的手段，版权本身不是自然权利 <http://www.gnu.org>
- GNU = GNU's Not Unix
- GPL 中 free 的理解

“Free software” is a matter of liberty, not price. To understand the concept, you should think of “free speech”, not “free beer”. “Free software” refers to the users’ freedom to run, copy, distribute, study, change, and improve the software.

1.1.3.2 自由软件

- 可以
 - 取得软件与源代码
 - 自由复制
 - 自由修改
 - 再发行
 - 回馈社团
- 不可以

- 修改授权
- 单纯贩卖

1.1.3.3 Open Source 开源软件，授权满足如下要求

- 公布原始码且用户具有修改权：用户可以任意的修改与编译程序代码，这点与自由软件差异不大；
- 任意的再散布：该程序代码全部或部份可以被贩卖，且程序代码可成为其他软件的组件之一，作者不应回避具有拥有权或收取其他额外费用。
- 必须允许修改或衍生的作品，且可让再发布的软件使用相似的授权来发表即可。
- 承上，用户可使用与原本软件不同的名称或编号来散布。
- 不可限制某些个人或团体的使用权
- 不可限制某些领域的应用：例如不可限制不能用于商业行为或者是学术行为等特殊领域等等
- 不可限制在某些产品当中，亦即程序代码可以应用于多种不同产品中。
- 不可具有排他条款，例如不可限制本程序代码不能用于教育类的研究中，诸如此类。

1.1.3.4 常见授权方式

- Apache License
- BSD-3/BSD-2
- GPL
- LGPL (Lesser GPL)
- MIT
- Mozilla Public License
- Common Development and Distribution License

1.2 Linux 发展

1.2.1 发展历程

- 从 Minix 中汲取营养
- 386 推出对多任务的支持
- 基于 GNU 自由软件，融合 Minix 文件系统，推出核心程序 v0.02，共享于 FTP 的 Linux 目录

```
Hello everybody out there using minix-
I'm doing a (free) operation system (just a hobby,
won't be big and professional like gnu) for 386(486) AT clones.

I've currently ported bash (1.08) and gcc (1.40),
and things seem to work. This implies that i'll get
something practical within a few months, and I'd like to know
what features most people want. Any suggestions are welcome,
but I won't promise I'll implement them :-)
```

- 基于 POSIX(Portable Operating System Interface, IEEE) 规范修改 Linux
- 虚拟团队的产生
 - 1994 v1.0
 - 1996 v2.0
 - 2011 v3.0
 - 2015 v4.0

1.2.2 核心版本

- 次版本为奇数：development version，如 2.5.xx

- 次版本为偶数：stable version, 如 2.6.xx
- 这种奇偶数使用方式，在 3.0 后失效
- `uname -r` 查看版本信息 <http://www.kernel.org/releases.html>

1.2.3 Linux distributions

- Linux Kernel + GNU Software + Tools (From 3rd Party) + Setup
- Linux 发行版本，均使用同一个 Kernel <http://www.kernel.org>, 遵循 LSB ([Linux Standard Base](#)) 及 FHS ([File System Hierarchy Standard](#))
 - RPM based:
 - [Red Hat](#), 两大版本, Red Hat Linux/Red Hat Enterprise Linux(RHEL), 前者 2003 年停止发布, 由 Fedora Project 取代, 但实验性质太浓, 成功结果会加入 RHEL
 - [SuSE](#)
 - [Fedora](#)
 - [CentOS](#), 由 RHEL 开放的源码编译, 并去掉了 Red Hat 的商标, 生成的合法的发行版本, 并取得了巨大成功。其他克隆版本还有 White Box Enterprise Linux/Scientific Linux 等
 - dpkg based:
 - [Ubuntu](#), 每年两个版本, 分别发布于 4 月和 10 月, 如 2019 年的版本号为 19.04 和 19.10
 - [Debian](#)
 - [gentoo](#)
- Live CD
 - [knoppix](#)

1.3 Linux 使用场景

- 无处不在

1.4 Linux 该如何学习

- 抛弃图形用户界面 (GUI Graphics User Interface, X Window), 使用 CLI (Command Line Interface)
- 从头学起
 - 第 0 章相关知识
 - 安装 Linux
 - 基础技能, 特别要理解权限概念及相关问题
 - 学会 vi
 - 学会某种 Shell 及相应的 Shell Script 编程
 - 能够维护系统
 - 具有网络基础知识
- 实际操作: 安装一个自己拥有的 Linux, 做好数据备份, 大胆操作, 遇到问题要迎头而上, 在解决问题中增长经验, 掌握新的技能
- 解决问题, 查询相关资源
 - Linux 自带的数据: /usr/share/doc 目录
 - [TLDp \(The Linux Documentation Project\)](#), 那些 HOWTOs 非常值得一看
 - [CLDP 中文计划 \(Chinese Linux Documentation Project\)](#), 该项目已终止
 - 各大论坛提问, 注意网络礼仪
 - <https://stackoverflow.com>
 - <https://segmentfault.com>
 - <https://www.csdn.net>
 - <http://bbs.byr.cn>, 北邮人论坛
 - <http://www.newsmth.net> 水木社区
 - 科学上网

2 第 2 章主机规划与磁盘分区

2.1 安装过程说明

- 略，采用虚拟机安装

2.2 虚拟机安装软件

- [VirtualBox](#)
- [VMware](#)
- [Parallel Desktop](#)

2.3 需要了解的知识点

- 硬件设备在系统中对应的文件，均存储在 `/dev` 目录，如 `/dev/sd[a-d]`，对应系统中各个硬盘
- 硬盘如何分区以及分区注意事项
- 什么是 MBR (Master Boot Record)/ GPT (GUID Partition Table)，什么是 EBR (https://en.wikipedia.org/wiki/Extended_boot_record)
- 开机启动过程，了解 BIOS/UEFI(Unified Extensible Firmware Interface)
- 分区与挂载的概念
- 了解 Linux 安装过程（实际安装一遍，不要选择极简模式）

3 第 3 章 Linux 安装

- 略

YOUNG@BUPT

4 第4章首次登入与在线求助

4.1 首次登入系统

- 使用账号密码登陆
- 正常关机或退出操作系统

4.2 使用命令行方式进行操作

- 启动终端，`gnome-terminal`（将其固化在任务栏上，方便下次使用）
- Linux 命令语法：`command [-options] parameter1 parameter2 ...`
 - `command` 命令名称，如 `cd`、`ls` 等
 - 中括号 `[]` 并不存在实际的指令中，其括起来的内容，表示这些内容在实际使用时，为可选项
 - `-options`，命令选项，有长短两种格式，短格式为选项前加 `-` 号，然后跟一个字母；长格式为选项前加 `--` 符号，然后跟一个单词，例如：`ls -a` 和 `ls --all` 效果一样，需要特别说明的是，使用短格式时，如果命令行有多个选项，可以采用精简的方式输入，如：`ls -a -b -l`，可以简化为 `ls -abl`
 - `parameter1 parameter2 ...` 为命令行参数，例如：`ls -a /usr/share/` 中，`-a` 为 option，`/usr/share` 为 `ls` 命令的参数
 - 命令、选项、参数之间，必须以空格分割，多个连续空格会被 shell 解释为一个
 - 最后键入 `[Enter]` 执行命令
 - 如果指令太长，可以使用续行符 `\`，使输入指令延续到下一行
 - Linux 中，大小写是区分的
- 语言支持方面
 - 使用 `locale` 命令查看当前支持的语系
 - 修改语系
 - `LANG=en_US.utf8`
 - `export LC_ALL=en_US.utf8`
 - 以上两个命令，注意 `=` 两边没有空格
 - 强烈建议，如果可能，永远选择 `utf8` 编码
- 输入几个命令练手，`date cal bc`
- 常用热键
 - `tab` 命令补全
 - `ctrl+c` 终止任务
 - `ctrl+d` End of File/End of Input，常见动作为退出
 - `shift+PgUp/PgDn` 翻页查看过往输出
- 注意，要多多查看错误信息，并根据指示解决问题

4.3 在线求助 man page 与 info page

4.3.1 通用模式

- 通用模式，几乎所有命令都有 `--help` 或 `-h` 选项，获取帮助信息

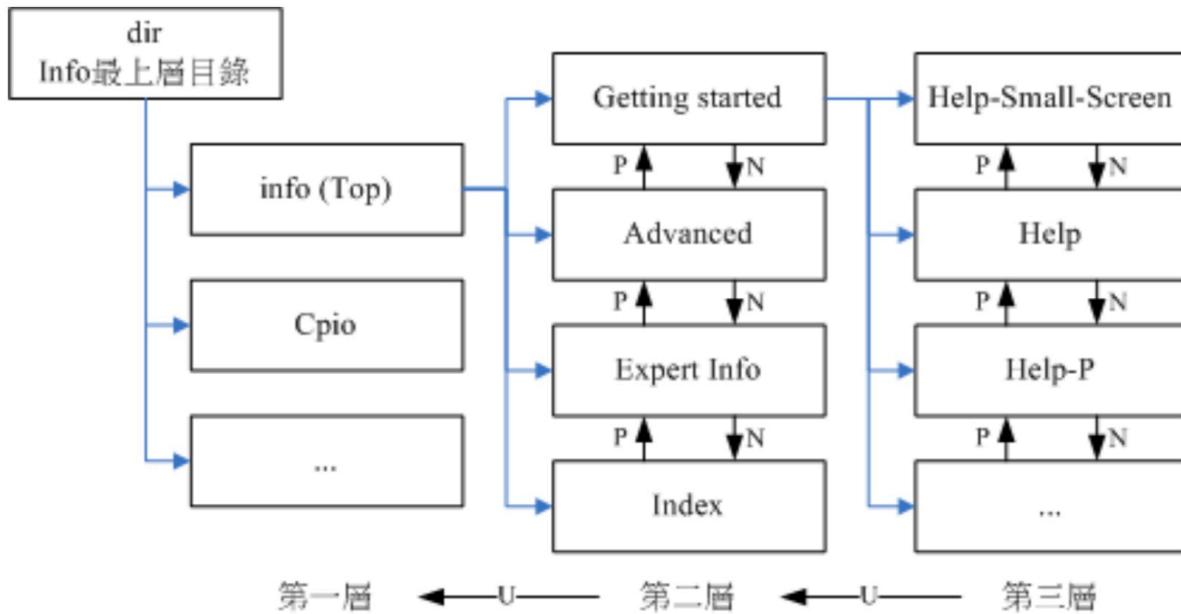
4.3.2 man

- `man command` 获取命令手册页
 - 默认使用 `more` 命令显示手册内容，需要掌握在该模式下的操作命令

- 手册页一般会分成如下部分
 - NAME 简短的指令、数据名称说明
 - SYNOPSIS 简短的指令下达语法 (syntax) 简介
 - DESCRIPTION 较为完整的说明，这部分最好仔细看看！
 - OPTIONS 针对 SYNOPSIS 部分中，有列举的所有可用的选项说明
 - COMMANDS 当这个程序（软件）在执行的时候，可以在此程序（软件）中下达的指令
 - FILES 这个程序或数据所使用或参考或连结到的某些文件
 - SEE ALSO 可以参考的，跟这个指令或数据有相关的其他说明！
 - EXAMPLE 一些可以参考的范例
 - 重点查看 DESCRIPTION 和 OPTIONS 小节，有时候为了快速解决问题，也可以直接跳到 EXAMPLE 部分直接套用前人的成功案例
- 手册系统中的全部资料，根据功能分别存放在不同的目录
 - 1 用户在 shell 环境中可以操作的指令或可执行文件
 - 2 系统核心可呼叫的函数与工具等
 - 3 一些常用的函数 (function) 与函式库 (library)，大部分为 C 的函式库 (libc)
 - 4 装置文件的说明，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 (games)
 - 7 惯例与协议等，例如 Linux 文件系统、网络协议、ASCII code 等等的说明
 - 8 系统管理员可用的管理指令
 - 9 跟 kernel 有关的文件
- 通过查看 man 的配置信息，可以得知这些手册资料在系统中的具体存放位置
- 相关命令
 - `whatis` 等价于 `man -f`
 - `apropos` 等价于 `man -k`

4.3.3 info

- 将文件数据拆成一个一个的段落，每个段落用自己的页面来撰写，并且在各个页面还有类似网页的“超链接”来跳到各不同的页面中，每个独立的页面也被称为一个“node”
- 可以将其理解为文本模式的网页数据



4.3.4 联机帮助文件

- /usr/share/doc 目录

- 每 package 一个存放目录

4.3.5 一个极轻量级编辑器

- `nano` 命令
- 面向初学者使用的编辑器，简单易用，可以完成基本的文本编辑功能
- 根据界面最后两行的提示信息操作即可

4.3.6 正确开关机

- `shutdown -h now`
- `shutdown -h 10 'I will shutdown after 10 minutes'`
- `shutdown` 一分钟后关机
- `shutdown -c` 取消关机
- `reboot/halt/poweroff`

5 第5章 Linux 的文件权限与目录配置

彻底解决“Permission denied”问题

5.1 用户与群组

- `/etc/passwd` 存放用户信息
- `/etc/shadow` 存放用户密码信息
- `/etc/group` 存放用户组信息

5.2 文件权限

5.2.1 文件属性

- `ls -l` 的输出，共计 7 部分

```
drwxr-xr-x  2 young staff      64 Mar 19 10:05 model
-rw-r--r--  1 young staff 13450904 May  9  2017 model.h5
-rw-r--r--  1 young staff   3794 May  8  2017 model.json
```

- 第一栏：文件权限，10 个字符
 - 第二栏：文件链接数
 - 第三栏：文件拥有者
 - 第四栏：文件所属组
 - 第五栏：文件大小
 - 第六栏：文件时间
 - 第七栏：文件名称
- 第一栏：表示当前文件的类型与权限
 - 第一个字符：文件类型
 - d：目录，例如上图中文件名为“model”的那一行
 - •：普通文件，例如上图中文件名为“model.h5”那一行
 - l：链接文件 (link file)
 - b：块设备文件，如磁盘
 - c：字符设备文件，如键盘、鼠标
 - 其他九个字符，三位一组，`r` 表示可读、`w` 表示可写、`x` 表示可执行，若无相应权限，则显示`-`
 - 第一组：文件拥有者可具备的权限
 - 第二组：文件拥有者同组用户拥有的权限
 - 第三组：其他人拥有的权限
- 第二栏
 - 对于普通文件，表示该文件的硬链接数
 - 对于目录文件，表示该目录下面的子目录数目
- 第三栏、第四栏：分别表示文件拥有者和文件所属组
- 第五栏：文件大小，默认单位为 Byte
- 第六栏：文件的创建日期或者最近的修改日期
- 第七栏：文件名，如果文件名前有`.`，`ls` 命令默认不会将其列出来

5.2.2 权限的修改

- 默认权限设定，文件权限为 `0666-umask`，目录权限为 `0777-umask`
- 权限的修改命令，`chmod`，带目录修改时，使用参数 `-R`
 - 基于符号进行修改
 - `chmod u+x testfile`
 - `chmod o-x,g=rwx testfile`
 - `chmod a+x,u-w,o=r testfile`

- 基于数字进行修改
 - `chmod 644 testfile`
 - `chmod 755 testfile`
 - `chmod 777 testfile`
- `chgrp` 和 `chown`，修改文件的拥有者或文件所属的组
 - `chgrp group_name file`
 - `chown usr_name file`
 - `chown usr_name:grp_name file`
 - `chown -R usr_name:grp_name file`

5.2.3 目录与文件的权限含义

- 文件
 - `r`：可以读取此文件的实际内容，如读取文本文件的文字内容等，如 `cp/cat/more/less`
 - `w`：可以编辑、新增或者是修改该文件的内容（但不含删除该文件），如 `nano/vi/ed`
 - `x`：该文件具有可以被系统执行的权限
- 目录
 - `r`：具有读取目录结构列表的权限，可以用 `ls` 列出目录内容
 - `w`：具有修改目录结构列表的权限，包括：创建或删除文件或目录（无论文件权限为何），修改文件或目录名称，移动文件或目录
 - `x`：代表用户能否进入该目录成为工作目录
- 做实验，使用 `touch` 命令创建空文件，然后进行各种操作测试

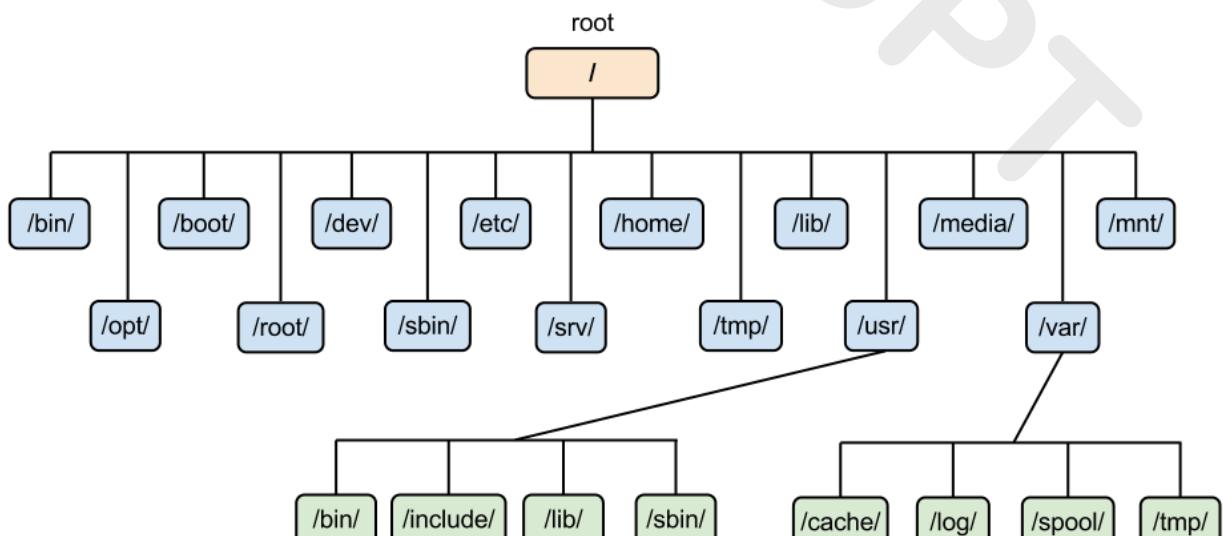
5.3 目录配置

5.3.1 Linux 目录配置

- FHS (Filesystem Hierarchy Standard)

The Filesystem Hierarchy Standard (FHS) defines the directory structure and directory contents in Unix and Unix-like operating systems. It is maintained by the Linux Foundation. The latest version is 3.0, released on 3 June 2015. Currently it is only used by Linux distributions.

http://refspecs.linuxfoundation.org/FHS_3.0/fhs/index.html



-
-

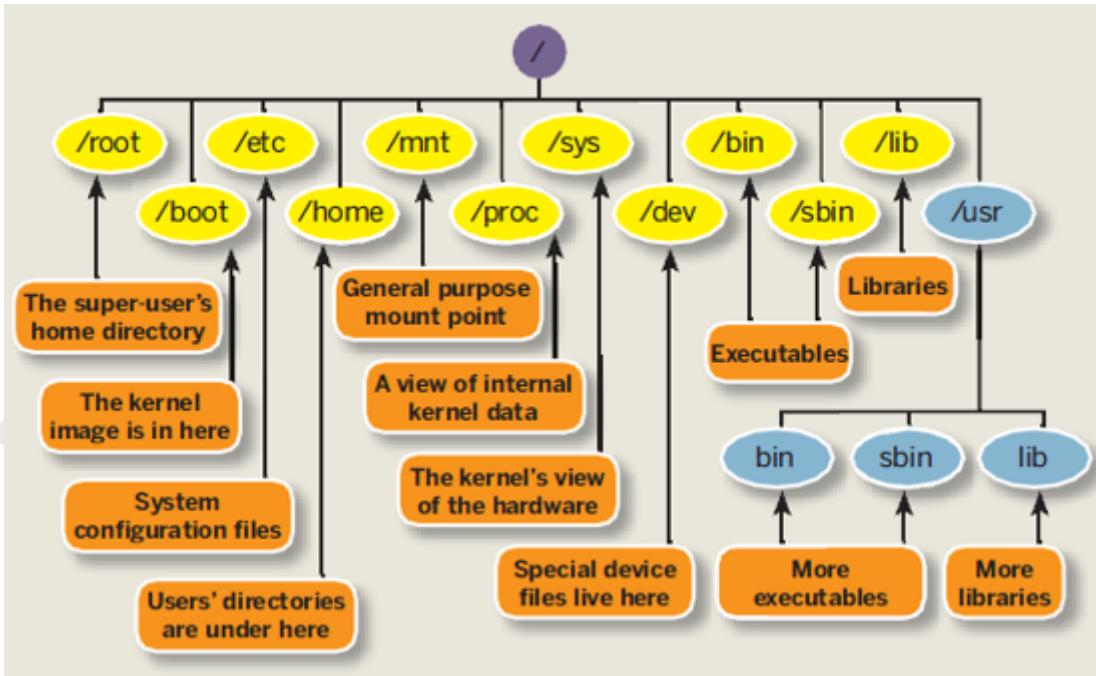


Figure 1: FHS

5.3.1.1 目录分类

分类	可分享的 (shareable)	不可分享的 (unshareable))
不变的 (static)	/usr (软件放置处)	/etc (配置文件)
不变的 (static)	/opt (第三方软件)	/boot (开机和启动用文件)
可变动的 (variable)	/var/mail (邮箱)	/var/run (程序相关)
可变动的 (variable)	/var/spool/news (新闻组)	/var/run (程序相关)

- 说明

- 可分享的: 可以分享给其他系统挂载使用的目录, 包括执行文件与用户的邮件等数据, 是能够分享给网络上其他主机挂载用的目录
- 不可分享的: 自己机器上面运行的配置文件或者是与程序有关的 socket 文件等, 仅与自身机器有关, 不适合分享给其他主机
- 不变的: 不会经常变化的数据, 不随着 distribution 变动。例如函式库、文件说明文件、系统管理员所管理的主机服务配置文件等等
- 可变动的: 经常改变的数据, 例如登录文件、用户接收的新闻组等

5.3.1.2 FHS 建议的三个目录

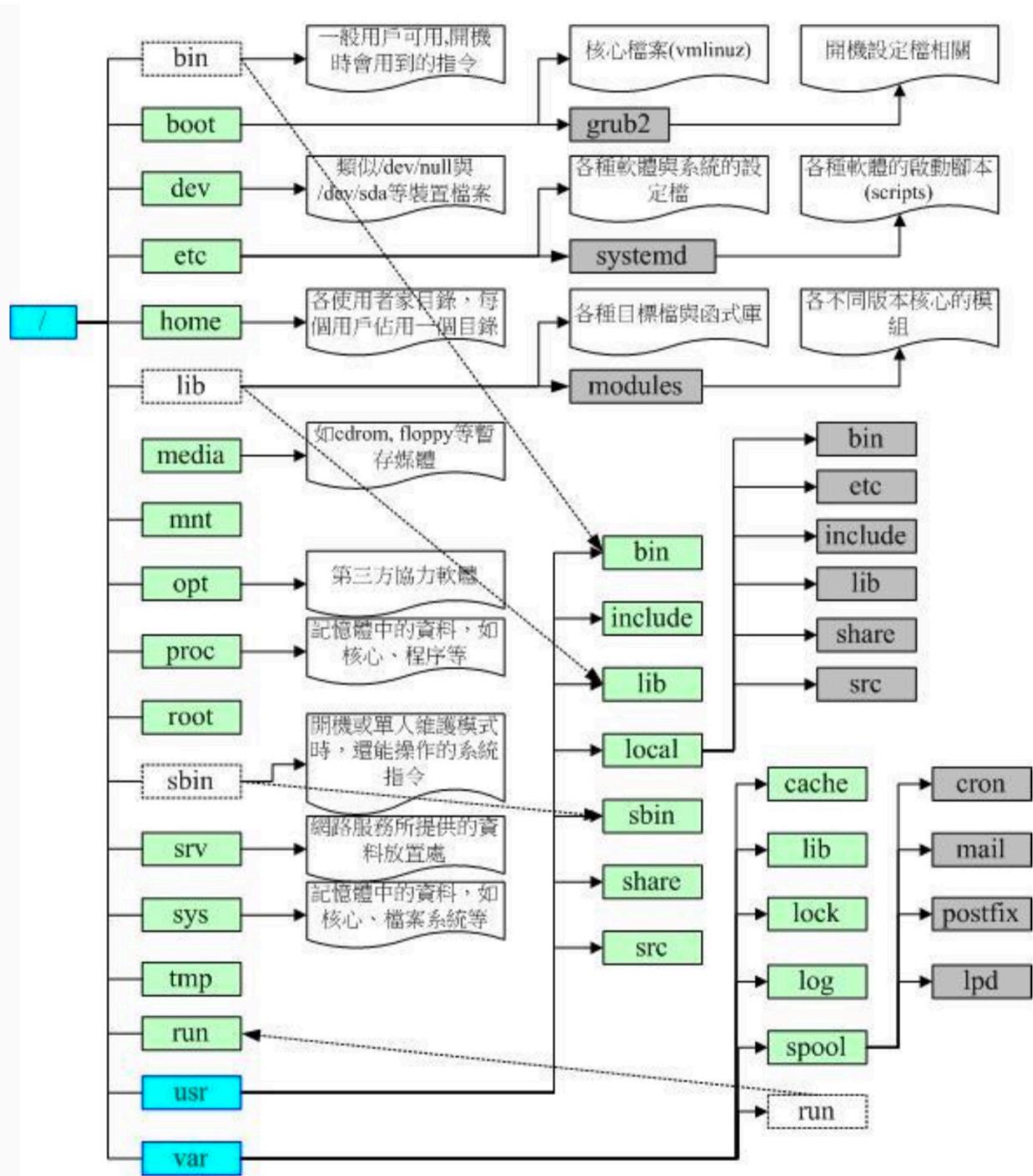
- / (root, 根目录): 与开机系统有关
- /usr (unix software resource): 与软件安装/执行有关
- /var (variable): 与系统运行过程有关

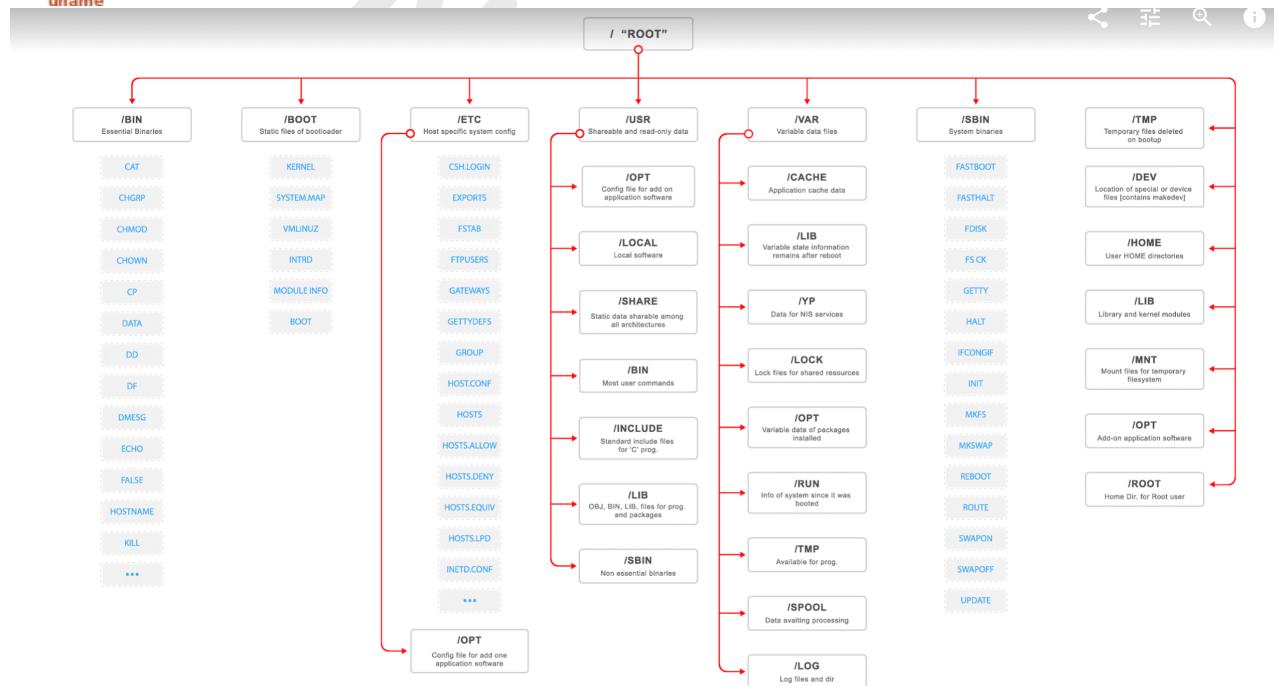
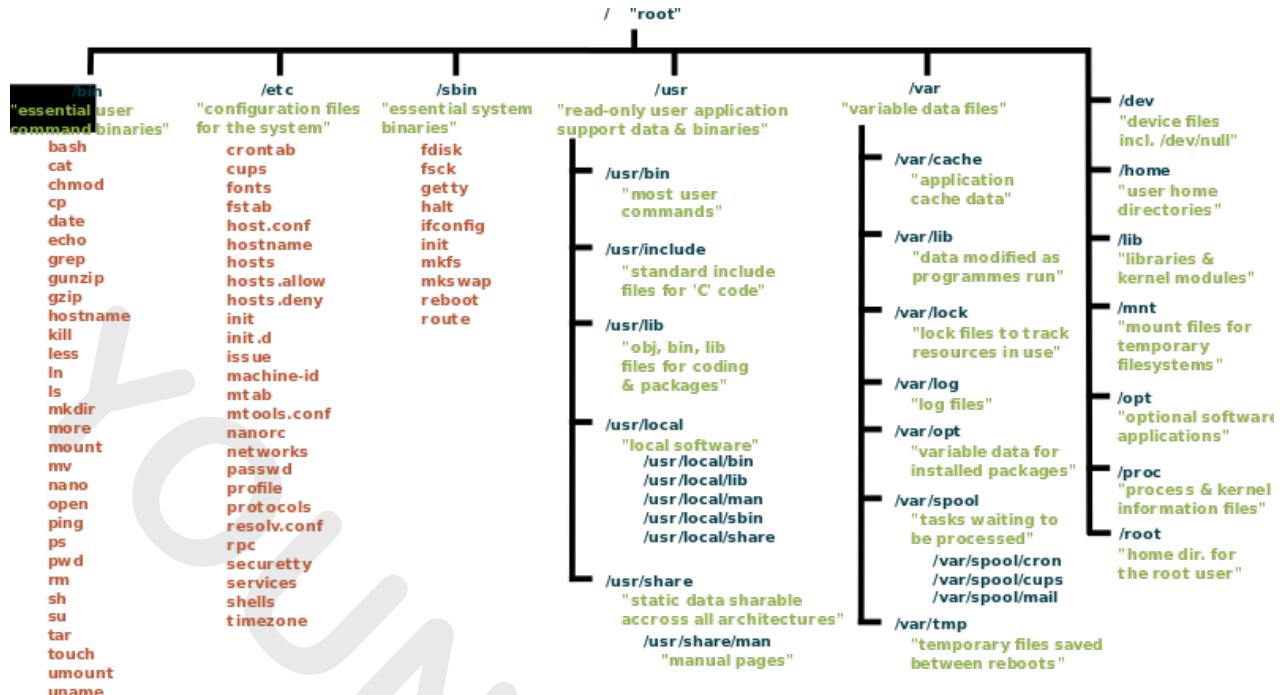
5.3.1.3 目录具体说明

- / 目录：最重要的一个目录，所有的目录都是由根目录衍生出来的，根目录也与开机/还原/系统修复等动作有关
- FHS 要求必须存在的目录
 - /bin : 存放可执行文件，特别地，存放单人维护模式下还能够被操作的指令，如：cat, chmod, chown, date, mv, mkdir, cp, bash 等
 - /boot : 存放开机会使用到的文件，包括 Linux 核心文件以及开机菜单与开机所需配置文件等，Linux kernel 为类似 vmlinuz 的文件
 - /dev : 设备与接口文件，访问该文件，相当于访问某个设备，如 /dev/sd, /dev/zero, /dev/null, /dev/loop, /dev/tty 等
 - /etc : 存放系统几乎所有的配置文件，一般可以让所有用户读取，但仅有 root 才可以修改，FHS 不建议在该目录存放 binary 文件，重要文件有：/etc/passwd, /etc/fstab, /etc/modprobe.d 等，此外，FHS 还规范几个重要目录也存放在该目录下面
 - /etc/opt (必要)：/opt 目录下安装的、第三方软件的相关配置文件，存放在那里
 - /etc/X11/ (建议)：与 X Window 有关的各种配置文件都在这里，尤其是 xorg.conf 这个 X Server 的配置文件
 - /etc/sgml/ (建议)：与 SGML 格式有关的各项配置文件
 - /etc/xml/ (建议)：与 XML 格式有关的各项配置文件
 - /lib : 存放开机时会用到的函数库，以及 /bin 或 /sbin 下的命令会调用的库函数，FHS 还要求必须存在 /lib/modules 目录，存放核心驱动模块
 - /media : 可移除设备如 Floppy/CDROM/DVD/USB Disk 等一般 mnt 在这个目录下，常见目录有 /media/floppy, /media/cdrom 等
 - /mnt : mount 其他设备时使用，比方说：第二块硬盘
 - /opt : 第三方软件安装目录
 - /sbin : 系统 (System) 相关命令，只有 root 才能够使用，如开机、修复和还原文件系统等，如 fdisk, fsck, ifconfig, mkfs 等
 - /srv : Service 的简写，一般用于存储网络服务如 FTP/WWW 等的数据文件，如果这些数据不提供给他人浏览，则建议存放于 /var/lib 目录
 - /tmp : 所有用户和应用都可以存放自己临时数据的目录，系统重新启动时，会自动删除其下所有数据
 - /usr : 第二层 FHS 设定，后续介绍
 - /var : 第二层 FHS 设定，后续介绍
- FHS 建议可以存在的目录
 - /home : 系统默认的用户家目录，可以用 ~ 或者 ~username 来访问自己或某个用户的根目录
 - lib<qualifier> : 存放于 /lib 不同格式的函数库，如 /lib64 等
 - /root : root 的家目录，不在 /home 下的原因是，单人维护模式需要 root 操作，不能因为挂载原因，导致 root 没有家
 - /lost_found : ext2/ext3/ext4 文件系统格式才会产生的一个目录，目的在于当文件系统发生错误时，将一些丢失的片段放置到这个目录下，这些片段类似 windows 下的 chkdsk 找回的文件
 - /proc : 一个虚拟文件系统 (virtual filesystem)，存放的数据都在内存中，例如系统核心、进程信息 (process)、设备状态及网络状态等等。该目录下的数据都在内存中，所以这个目录本身不占任何硬盘空间，比较重要的文件有，/proc/cpuinfo, /proc/dma, /proc/interrupts, /proc/ioports, /proc/net/* 等
 - /sys : 跟 /proc 类似，也是虚拟文件系统，记录核心与系统硬件信息较相关的信息，包括目前已加载的核心模块与核心侦测到的硬件装置信息等等，该目录同样不占硬盘空间

- **/usr** 目录，存放可分享和不变化的，该目录可以分享给局域网内其他主机来使用，是 **Unix Software Resouce** 而非 user 的缩写，该目录类似于 Windows 下的 **C:\windows** 的一部分 +**C:\Program files** 两个目录的综合体
 - FHS 要求必须存在的目录
 - **/usr/bin**：一般用户能够使用的指令都放在这里，**/bin** 下的命令，有些是该目录的符号链接，FHS 要求该目录下不应该有子目录
 - **/usr/lib**：与 **/lib** 功能相同
 - **/usr/local**：管理员安装自己下载的软件（非 distribution 提供），建议安装到此目录，该目录下也有一套 **bin**, **etc**, **include**, **lib** 的目录结构
 - **/usr/sbin**：系统正常运行时，不必要的系统命令，比如网络服务器命令。基本与 **/sbin** 类似
 - **/usr/share**：存储只读架构的数据文件，也包括共享文件，几乎均为文本文件，此目录下常有如下目录：
 - **/usr/share/man**：联机帮助文件
 - **/usr/share/doc**：软件杂项的文件说明
 - **/usr/share/zoneinfo**：与时区有关的时区文件
 - FHS 建议可以存在的目录
 - **/usr/games**：游戏相关数据
 - **/usr/include**：c/c++ 头文件默认存放目录
 - **/usr/libexec**：存放某些使用者不常用的可执行文件或脚本 (script) 等，例如大部分的 X 窗口底下的操作指令
 - **/usr/lib<qualifier>**：类似 **/lib/**，被其 link
 - **/usr/src**：源代码存放位置，linux kernel 源码，一般存放在 **/usr/src/linux** 下
 - **/var** 目录，存放经常变化的文件，如 cache, log 以及软件运行时产生的文件 (lock file, run file, database file) 等
 - FHS 要求必须存在的目录
 - **/var/cache**：存储缓存数据，比如 apt 命令下载安装的软件包
 - **/var/lib**：程序运行时，数据文件存放目录，如：MySQL 数据库，存放于 **/var/lib/mysql**，rpm 数据库则放在 **/var/lib/rpm** 中
 - **/var/lock**：存放应用的加锁文件，避免单应用同时启动多个进程，加锁文件也有可能存放于 **/run/run/lock** 下，此时 **/var/lock** 为该目录的符号链接
 - **/var/log**：系统日志存放目录，非常重要。重要文件有 **/var/log/messages**, **/var/log/wtmp** (记录登录者信息) 等
 - **/var/mail**：存放用户邮件，也可能是 **/var/spool/mail**
 - **/var/run**：某些程序或者服务启动后，会将其 pid 放置在此目录
 - **/var/spool**：存储一些队列（排队等待处理）数据，这些数据被处理后会被删除，比如邮件 **/var/spool/mail**, **/var/spool/mqueue**，crontab 数据 **/var/spool/cron**

5.3.2 目录树





5.3.3 绝对路径与相对路径

- 绝对路径：由根目录开始写起 V 的文件名或目录名
- 相对路径：相对于当前路径的文件名写法， . 表示当前目录， .. 表示上一级目录

6 第 6 章 Linux 文件与目录管理

6.1 目录与路径

6.1.1 cd - 切换目录

- `cd` 回到自己家目录
- `cd ~` 同上
- `cd ~username` 到某个用户的家目录
- `cd -` 在最近访问过的两个目录之间切换

6.1.2 pwd - 显示当前目录

`pwd` 也是 `shell` 的一个内置命令 `pwd -P` 显示实际目录，而非链接路径

6.1.3 mkdir - 创建一个新目录

- `mkdir dirname` 创建新目录
- `mkdir -p a/b/c/d` 一次创建多层目录
- `mkdir -p {d1,d2}/{a,b,c,d,e}` 同上，创建更复杂的目录结构
- `mkdir -m 711 test` 创建特定权限的目录

6.1.4 rmdir - 删除一个空目录

- `rmdir -p d1/d2/d3` 删除多级空目录结构
- 安全起见，删除空目录时务必使用 `rmdir`，而非 `rm -fr`

6.1.5 \$PATH 环境变量

- 存放执行命令时，系统搜索的目录列表，若其定义的多个目录中都含有同名可执行文件，先搜索到的会被执行，其他的被忽略
- `echo $PATH` 显示环境变量内容
- `PATH=$PATH:/new/path/to/exec; export $PATH` 设置 `$PATH`
- `$PATH` 所定义的目录列表中，为安全起见，一定不要包含当前目录 `.`，否则可能会有如下所示风险
 - 若恶意者在 `/tmp` 下写一个恶意指令，改名为 `ls` 等常用指令名
 - 用户在进入该目录操作时，使用 root 权限执行，不小心执行到
 - 会造成无法预计的损失

6.2 文件与目录管理

6.2.1 ls - 目录列表

- `ls -a`，显示所有文件，一般 alias 成 `la`
- `ls -l`，长列表方式显示，一般 alias 成 `ll`
- `ls --color=auto` 带颜色显示文件
- `ls -alF` 几个选项联用
- `ls -al --full-time` 显示时间信息
- 常见选项与参数
 - `-a`：全部的文件，连同隐藏档（开头为`.`的文件）
 - `-A`：全部的文件，连同隐藏档，但不包括`.`与`..`这两个目

- `-d` : 仅列出目录
- `-f` : 直接列出结果，不进行排序（ls 预设会以档名排序!）
- `-F` : 根据文件、目录等信息，给予附加数据结构，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的方式（例如 GB, KB 等）列出来
- `-i` : 列出 inode 号码
- `-l` : 长数据串行出，包含文件的属性与权限等
- `-n` : 列出 UID 与 GID 而非使用者与群组的名称
- `-r` : 将排序结果反向输出，例如：原本档名由小到大，反向则为由大到小
- `-R` : 连同子目录内容一起列出来
- `-S` : 以文件容量大小排序
- `-t` : 依时间排序
- `--color=never` : 不要依据文件特性给予颜色显示
- `--color=always` : 显示颜色
- `--color=auto` : 让系统自行依据设定来判断是否给予颜色
- `--full-time` : 以完整时间模式（包含年、月、日、时、分）输出
- `--time={atime,ctime}` : 输出 access 时间或改变权限属性时间（ctime）而非内容变更时间（modification time）

6.2.2 cp - 拷贝文件

- `cp src1 src2 ... directory` 拷贝若干文件或目录到指定目录
- `cp -R` 带目录拷贝
- `cp -i` 交互方式
- `cp -a` 相当于-pPR，保留所有权限
- `cp -l` 硬链接方式拷贝
- `cp -s` 符号链接方式拷贝
- 常见选项与参数：
 - `-a` : 相当于 `-dr --preserve=all` 的意思
 - `-d` : 若来源文件为链接文件的属性（link file），则复制链接文件属性而非文件本身
 - `-f` : 为强制（force）的意思，若目标文件已经存在且无法开启，则移除后再尝试一次
 - `-i` : 若目标文件（destination）已经存在时，在覆盖前先询问
 - `-l` : 复制为硬链接（hard link）
 - `-p` : 连同文件的属性（权限、用户、时间）一起复制过去，而非使用默认属性（备份常用）
 - `-r` : 带目录拷贝
 - `-s` : 复制成为符号链接文件（symbolic link）
 - `-u` : destination 比 source 旧才更新 destination，或 destination 不存在的情况下才复制。
 - `--preserve=all` : 除了 `-p` 的权限相关参数外，还加入 SELinux 的属性，links, xattr 等也复制了
 - 最后需要注意的，如果来源档有两个以上，则最后一个目的文件一定要是『目录』

6.2.3 rm - 删除文件

- `rm file_or dirname` 删除文件或目录
- `rm -i` 交互式操作，安全
- `rm -r` 带目录删除
- `rm -fr` 强制带目录删除，linux 下最危险的命令，没有之一
- 常见选项与参数
 - `-f` : 强制删除，不询问
 - `-i` : 互动模式，在删除前会询问

- `-r` : 带目录 s 删除

6.2.4 mv - 移动文件或目录

- `mv source_file target_file` 文件改名
- `mv src1 src2 src3 ... directory` 将多个文件移到指定目录
- 常见选项与参数
 - `-f` : 强制覆盖, 不询问
 - `-i` : 互动模式, 在覆盖前会询问
 - `-u` : 若目标文件已经存在, 并且 source 比较新, 才会更新

6.2.5 basename - 获取文件名称

- strip directory and suffix from filenames
- `basename /usr/bin/sort` 输出“sort”
- `basename include/stdio.h .h` 输出“stdio”
- `basename -s .h include/stdio.h` 输出“stdio”
- `basename -a test/file1.a test1/file3.b` 输出两行 `file1.a file3.b`
- `basename -s .a -a test/file1.a test1/file3.b` 输出两行 `file1 file3.b`

6.2.6 dirname - 获取目录名

- strip last component from file name
- `dirname /usr/bin` 输出“usr”
- `dirname dir1/str dir2/str` 依次输出“dir1”及“dir2”
- `dirname stdio.h` 输出“.”

6.3 文件内容查询

6.3.1 cat - 查看文件内容

- cat 来自 concatenate 单词
- `cat file` 查看文件全部内容
- `cat -n file` 加行号显示文件内容
- `cat -A file` 显示特殊字符
- `cat >> newfile` 一种创建新文件的方法
- 常见选项与参数
 - `-A` : 相当于 `-vET` 的整合选项, 可列出一些特殊字符而不是空白而已;
 - `-b` : 列出行号, 仅针对非空白行做行号显示, 空白行不标行号
 - `-E` : 将结尾的断行字符 \$ 显示出来
 - `-n` : 打印出行号, 连同空白行也会有行号, 与 `-b` 选项不同
 - `-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

6.3.2 tac - 反向查看文件内容

6.3.3 nl - 添加行号显示

- `nl [-bnw] filename` 语法
- `-b` : 指定行号指定的方式, 主要有两种:
 - `-b a` : 表示不论是否为空行, 也同样列出行号 (类似 `cat -n`)

- `-b t` : 如果有空行，空的那一行不要列出行号（默认值）
- `-n` : 列出行号表示的方法，主要有三种：
 - `-n ln` 行号在屏幕的最左方显示
 - `-n rn` 行号在自己字段的最右方显示，不补 0 填充列宽
 - `-n rz` 行号在自己字段的最右方显示，补 0 填充列宽
- `-w` : 行号字段的占用的字符数。

6.3.4 more/less - 分页查看文件内容

- `h` 或 `?` 获得帮助
- 空格 翻页
- `Enter` 翻一行
- `/字符串` 向后搜索字符串
- `? 字符串` 向前搜索字符串
- `n` 重复前一个动作
- `N` 反向重复前一个动作
- `:f` 显示文件名以及目前显示的行数
- `q` 退出
- `b` 向前翻页（对于管道传送的数据无效）
- `g` 跳至第一行
- `G` 跳至最后一行

6.3.5 head - 查看文件前几行

- `head -n number file` 查看前面 number 行
- `head -number file` 同上

6.3.6 tail - 查看文件后面几行

- `tail -n number file` 查看最后 number 行
- `tail -number file` 同上
- `tail +n number file` 查看文件 number 行以后的内容
- `tail -f file` 持续查看 file 文件的尾部内容

6.3.7 od - octal, decimal, hex, ASCII dump

- `od -t TYPE file`
- `-t` : 后面可以接各种“类型 (TYPE)”的输出，例如：
 - `a` : 利用默认的字符来输出
 - `c` : 使用 ASCII 字符来输出
 - `d[size]` : 利用十进制 (decimal) 来输出数据，每个整数占用 size Bytes
 - `f[size]` : 利用浮点数值 (floating) 来输出数据，每个数占用 size Bytes
 - `o[size]` : 利用八进位 (octal) 来输出数据，每个整数占用 size Bytes
 - `x[size]` : 利用十六进制 (hexadecimal) 来输出数据，每个整数占用 size Bytes
- 示例
 - `od -t c /usr/bin/passwd` 使用 ASCII 显示文件内容
 - `od -t oCc /etc/issue` 八进制、ASCII 字符对照表方式显示
 - `od -t xCc /etc/issue` 十六进制、ASCII 字符对照表方式显示
 - `od -t dCc /etc/issue` 十进制、ASCII 字符对照表方式显示

6.3.8 touch - 创建空文件或更新已有文件时间

- 关于时间查看
 - `ls -l /etc/passwd` 显示修改时间
 - `ls -l --time=atime /etc/passwd` 显示访问时间
 - `ls -l --time=ctime /etc/passwd` 显示创建时间
- `touch [-acdmt] file`
 - `-a` : 仅修订 access time
 - `-c` : 若文件不存在则不创建新文件
 - `-d` : 后面可以接欲修订的日期而不用目前的日期，也可以使用 `-date="日期或时间"`
 - `-m` : 仅修改 mtime
 - `-t` : 后面可以接欲修订的时间而不用目前的时间，格式为 [YYYYMMDDhhmm]

6.4 文件与目录的默认权限

6.4.1 umask - 用户创建文件或目录时的权限默认值

- `umask` 显示当前 umask 数值
- `umask -S` 以 Symbolic 方式显示 umask 内容
 - 创建文件默认权限 `0666-umask`
 - 创建目录默认权限 `0777-umask`
- `umask 002` 修改 umask 值

6.4.2 文件隐藏属性

- `chattr [+--] [ASacdstu] 文件或目录名称` 设置文件隐藏属性
 - `+` : 增加一个属性，其他属性不变
 - `-` : 移除一个属性，其他属性不变
 - `=` : 设置恰好等于后面指定的属性
 - `A` : 设置时，存取时间 atime 将不会被修改，避免 I/O 开销 (目前一般使用文件系统挂载参数时使用 noatime 或者 relatime)
 - `S` : 使得文件变动会“同步”写入磁盘中
 - `a` : 文件只能增加数据，不能删除也不能修改数据，只有 root 能设置这属性
 - `c` : 读写时自动压缩和解压缩
 - `d` : 使该文件 (或目录) 不会被 dump 备份
 - `i` : 文件不能被删除、改名、设置链接也无法写入或新增数据，只有 root 能设置此属性
 - `s` : 文件被删除时，无法恢复
 - `u` : 文件被删除时，可以恢复
 - 注: `-i` 选项最为重要
- `lsattr` 显示文件隐藏属性
 - 常见选项与参数
 - `-a` : 将隐藏文件的属性也列出来
 - `-d` : 如果接的是目录，仅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列出来

6.4.3 文件的特殊权限

6.4.3.1 SetUID

- 参见 `/usr/bin/passwd` 文件权限
- 仅对二进制程序有效
- 执行者需要 x 权限
- 执行时具有该程序 owner 的权限

- 该权限仅在程序执行中有效

6.4.3.2 SetGID

- 参见 `/usr/bin/locate` 文件权限
- 对于文件
 - 仅对二进制文件有效
 - 执行时需要 x 权限
 - 执行过程中，获得该程序群组的权限
- 对于目录
 - 用户若对于此目录有 r 与 x 权限时，可以进入此目录
 - 用户在此目录下的有效群组将会变为该目录的群组
 - 如果用户还有 w 权限，使用者创建的新文件的群组与此目录的群组相同

6.4.3.3 Sticky Bit

- 1986 年，4.3BSD 引入
- 目前仅针对目录有效，典型应用场景，`/tmp` 目录
- 当用户对此目录有 w 和 x 权限时
- 用户在此目录下创建的文件，仅有目录拥有者、自己以及 root 才有权利删除该文件
- 历史上，该权限针对文件使用及产生的问题
 - 1974 年，Unix 5th 版本引入
 - 告知操作系统，该程序执行后，将其代码段仍然保留在交换分区中，以便后续快速加载
 - 引发的著名问题，是给设置了 sticky bit、并保留在交换分区的可执行文件打补丁（`patch`）的问题

6.4.3.4 特殊权限位的设定

- 数字方式
 - `chmod 4777 filename` 设置 suid
 - `chmod 2777 filename` 设置 sguid
 - `chmod 1777 filename` 设置 sticky bit
- 符号方式
 - `chmod u+s filename` 设置 suid
 - `chmod g+s filename` 设置 sguid
 - `chmod +t filename` 设置 sticky bit

6.4.4 stat - 查看文件或操作系统状态

- `stat [options] ... FILE ...`
- 这个命令非常有用，因此特别补充于此
- 选项说明
 - `-L, --dereference` : follow links
 - `-f, --file-system` : 显示操作系统状态而不是文件状态
 - `-c --format=FORMAT` : 使用指定格式输出，以换行结束
 - `--printf=FORMAT` : 类似`-format`，但输出无换行符
 - `-t, --terse` : 简洁方式输出
 - `--help` : 输出帮助信息并退出
 - `--version` : 输出版本并退出
 - 针对文件的 FORMAT
 - `%a` : 八进制数字方式输出访问权限
 - `%A` : 以 rwx 方式显示权限
 - `%b` : 分配的 block 数目
 - `%B` : 每 block 所占用的字节数

- `%C` : SELinux 安全上下文字符串
- `%d` : 十进制显示 device number
- `%D` : 十六进制显示 device number
- `%f` : raw mode in hex
- `%F` : 文件类型
- `%g` : 拥有者的组 id
- `%G` : 拥有者的组名
- `%h` : 硬链接数
- `%i` : inode number
- `%m` : mount point
- `%n` : 文件名称
- `%N` : 如果是符号链接，显示其指向的文件
- `%o` : optimal I/O transfer size hint
- `%s` : 字节表示的总大小
- `%t` : major device type in hex, for character/block device special files
- `%T` : minor device type in hex, for character/block device special files
- `%u` : 拥有者的 id
- `%U` : 拥有者用户名
- `%w` : 文件创建日期, 若无则显示 `-`
- `%W` : 文件创建日期, 表示为从 Epoch 开始流逝的秒数, 若未知则显示 `0`
- `%x` : 最后一次访问时间
- `%X` : 最后一次访问时间, 表示为从 Epoch 开始流逝的秒数
- `%y` : 最后一次数据修改时间
- `%Y` : 最后一次数据修改时间, 表示为从 Epoch 开始流逝的秒数
- `%z` : 最后一次状态修改时间
- `%Z` : 最后一次状态修改时间, 表示为从 Epoch 开始流逝的秒数
- 针对文件系统的 FORMAT
 - `%a` : free blocks available to non-superuser
 - `%b` : 文件系统中全部数据块的数目
 - `%c` : 文件系统中全部文件节点的数目
 - `%d` : 空闲文件节点数
 - `%f` : 空闲数据块数
 - `%i` : 十六进制表示的文件系统 ID
 - `%l` : 文件名最大长度
 - `%n` : 文件名
 - `%s` : block size (for faster transfers)
 - `%S` : fundamental block size (for block counts)
 - `%t` : 十六进制表示的文件系统类型
 - `%T` : 方便人类阅读方式表示的文件系统类型
- 示例
 - `stat -c "%a %U:%G" /usr/bin/passwd`
 - `stat -c "%n %w:%x:%y:%z" /usr/bin/passwd`
 - `stat -c "%n %W:%X:%Y:%Z" /usr/bin/passwd`
 - `stat -f -c "%b %i %l %s %s %t %T" /usr/bin/passwd`

6.4.5 file - 查看文件类型

- `file ~/bashrc`
- `file /usr/bin/passwd`
- `file /var/lib/mlocate/mlocate.db`

6.5 指令与文件的搜索

6.5.1 which - 查看某个命令位置

- `which -a` 列出 PATH 环境变量目录中存放的所有指定命令名位置

6.5.2 whereis - 查找文件，给出文件目录和手册页

- `whereis [-bmsu] file` 在特定目录下查找指定文件，给出文件目录和手册页文件
 - `-l` : 列出 whereis 会去查询的目录
 - `-b` : 只查找二进制文件
 - `-m` : 只查找 manual 相关路径
 - `-s` : 只找 source 来源
 - `-u` : 搜寻不在上述三个项目中的其他文件
- 示例
 - `whereis ifconfig`
 - `whereis passwd`

6.5.3 locate/updatedb - 基于数据库进行文件查找

- `updatedb` 更新数据库信息，一般定期自动运行
- `locate keyword` 查找包含 keyword 的文件
- `locate -i keyword` 不区分大小写
- `locate -i -l 5 keyword` 仅列出 5 个结果
- `locate -S` 列出数据库信息

6.5.4 find - 按照各种规则查找指定文件并在其上执行所需操作

- `find [PATH] [option] [action]`
- 选项、参数与示例：
 - 与时间有关的选项
 - `-mtime n` : n 为数字，意义为在 n 天之前的“一天之内”被更动过内容的文件
 - `-mtime +n` : 列出在 n 天之前(不含 n 天本身)被更动过内容的文件名
 - `-mtime -n` : 列出在 n 天之内(含 n 天本身)被更动过内容的文件名
 - `newer file` : file 为一个存在的文件，列出比 file 还要新的文件名
 - 示例
 - `find /var -mtime 0` 搜索过去 24 小时内改动过的文件
 - `find /var -mtime 4` 搜索 4 天前的 24 小时内改动过的文件，也即 4~5 那一天的文件
 - `find /var -mtime +4` 代表大于等于 5 天前的文件
 - `find /var -mtime -4` 代表小于等于 4 天内的文件名
 - `find /etc -newer /etc/passwd` 搜索 /etc 下日期比 passwd 新的文件
- 与使用者或组名有关的参数
 - `find -uid n` 搜索用户 uid 为 n 的文件

- `find -gid n` 搜索用户 gid 为 n 的文件
- `find -user name` 搜索用户名为 name 的文件
- `find -group name` 搜索用户组名为 name 的文件
- `find -nouser` 搜索文件的拥有者不在/etc/passwd 的文件
- `find -nogroup` 搜索文件的拥有者的组不在/etc/group 的文件
- 与文件权限及名称有关的参数
 - `find -name filename` 搜索文件名为 filename 的文件
 - `find -size [+-]SIZE` 搜索比 SIZE 大或小的文件, SIZE 可用 `c/k/m/g` 符号
 - `find / -size +1M` 搜索大于 1M 的文件
 - `find -type TYPE` 搜索文件类型为 TYPE 的文件, TYPE 可用 `f/b/c/d/l/s/p` 等符号
 - `find -perm mode` 搜索文件权限刚好等于 mode 的文件, 如 4755 对应-rwsr-xr-x
 - `find -perm -mode` 搜索文件权限包括全部 mode 权限的文件
 - 如 `find -perm -0744`, 会搜索到权限为 `4755` 的文件
 - `find -perm /mode` 搜索文件权限包含任一 mode 权限子集的文件
 - 如 `find -perm /0755` 会搜索到权限为 `0600` 的文件
- 其他示例
 - `find / -name "*passwd*"` 搜索系统中所有包含 passwd 的文件名
 - `find / -perm /7000` 寻找系统中包含特殊权限的所有文件
- 额外可执行的操作
 - `-exec command` : command 为其他命令
 - `-print` : 将结果输出到屏幕上, 这是默认操作
 - 示例
 - `find /usr/bin -perm /7000 -exec ls -l {} \;` 将搜索到的文件长列表方式显示出来
 - `find . -type d -exec touch {}/test.log \;` 在当前目录下的所有子目录下, 都创建 test.log 文件
 - `find . -name "test.log" -exec rm -f {} \;` 删除刚刚创建的所有 test.log
 - 其中: `{}` 相当于 find 命令的所有返回值, ; 要转义是因为其在 shell 下有特别含义

7 第7章 Linux 磁盘与文件系统管理

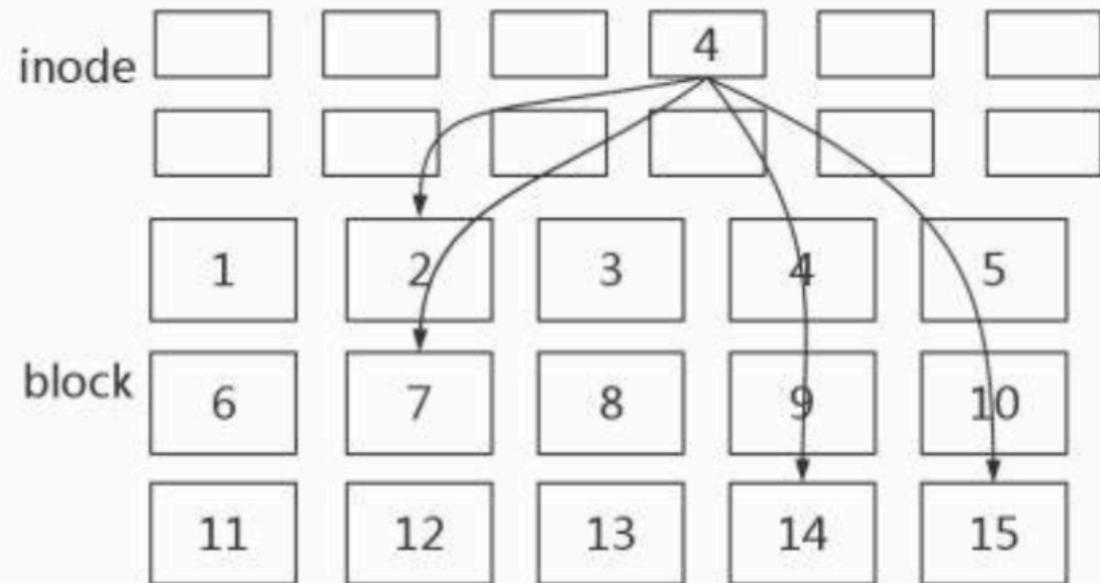
7.1 Linux 文件系统

7.1.1 磁盘组成与分区

- 磁盘设备命名
 - 实体磁盘设备在系统中的命名 `/dev/sd[a-p]`
 - 对于第一块硬盘 `/dev/sda`，其分区则为 `/dev/sda[1-128]`
 - 虚拟机磁盘命名 `/dev/vd[a-p]`
 - 软件磁盘阵列命名 `/dev/md[0-128]`
 - LVM 磁盘命名 `/dev/VGNAME/LVNAME`
- 磁盘分区命令
 - fdisk (MBR)
 - parted/gdisk (GPT)

7.1.2 文件系统特性

- ext [2-4] 系列文件系统
 - superblock: 记录 filesystem 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，同时记录此文件的数据所在的 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
- 索引式文件系统 (indexed allocation), inode 记录多个文件数据的实际存放点，可以一次性读取全部数据，通常不需要磁盘碎片整理

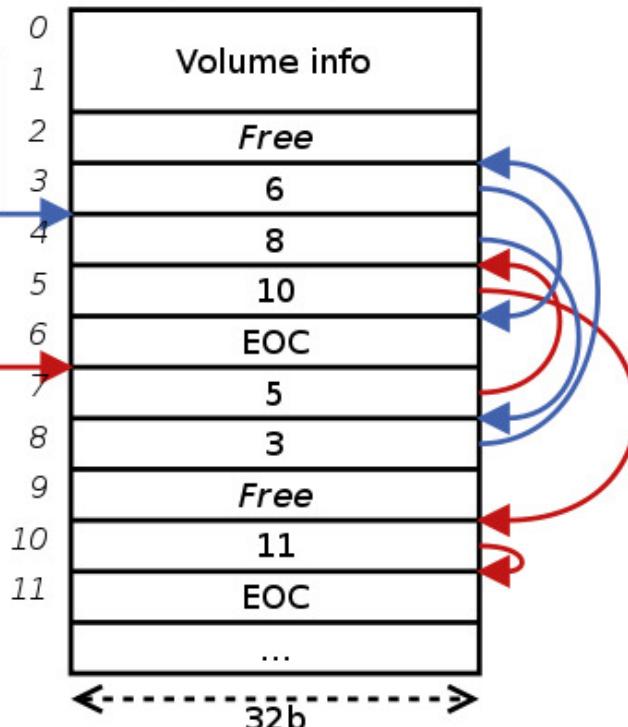


- 过去的 FAT 文件系统，使用链式记录方式，随时间推延，数据读取效率会逐步降低，需要碎片整理

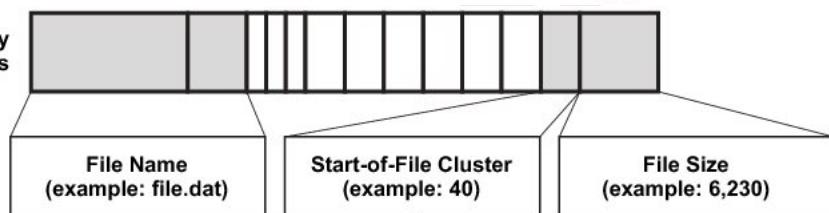
Directory table entry (32B)

Filename (8B)
Extension (3B)
Attributes (1B)
Reserved (1B)
Create time (3B)
Create date (2B)
Last access date (2B)
First cluster # (MSB, 2B)
Last mod. time (2B)
Last mod. date (2B)
First cluster # (LSB, 2B)
File size (4B)

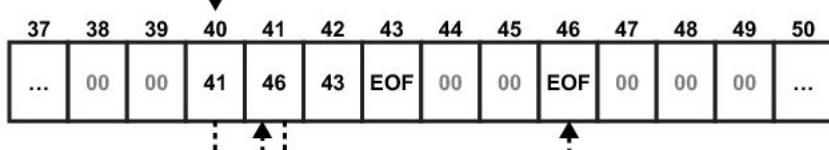
File allocation table



One Directory Entry
32 bytes

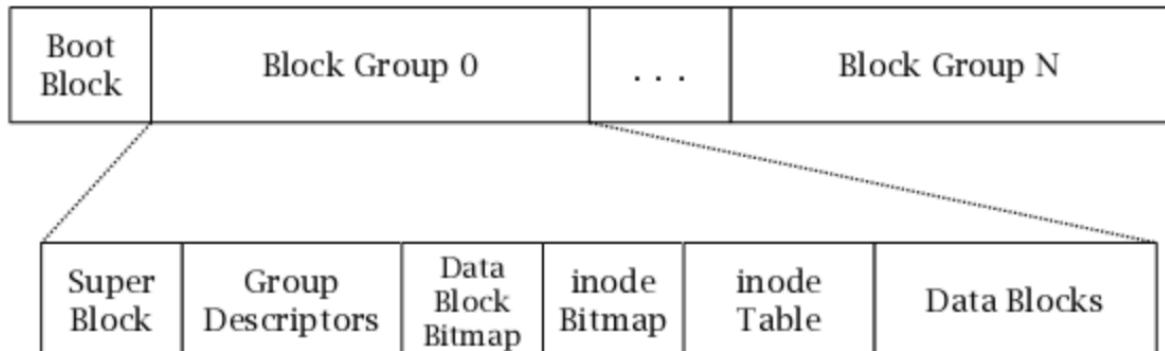


File Allocation Table
4 bytes per entry

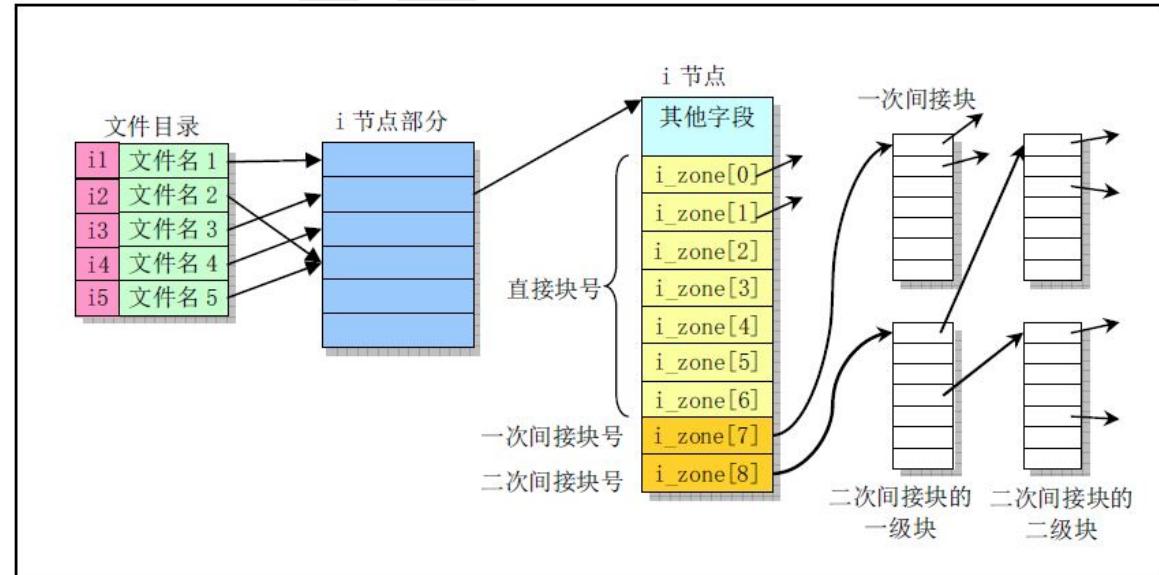


7.1.3 inode (ext2)

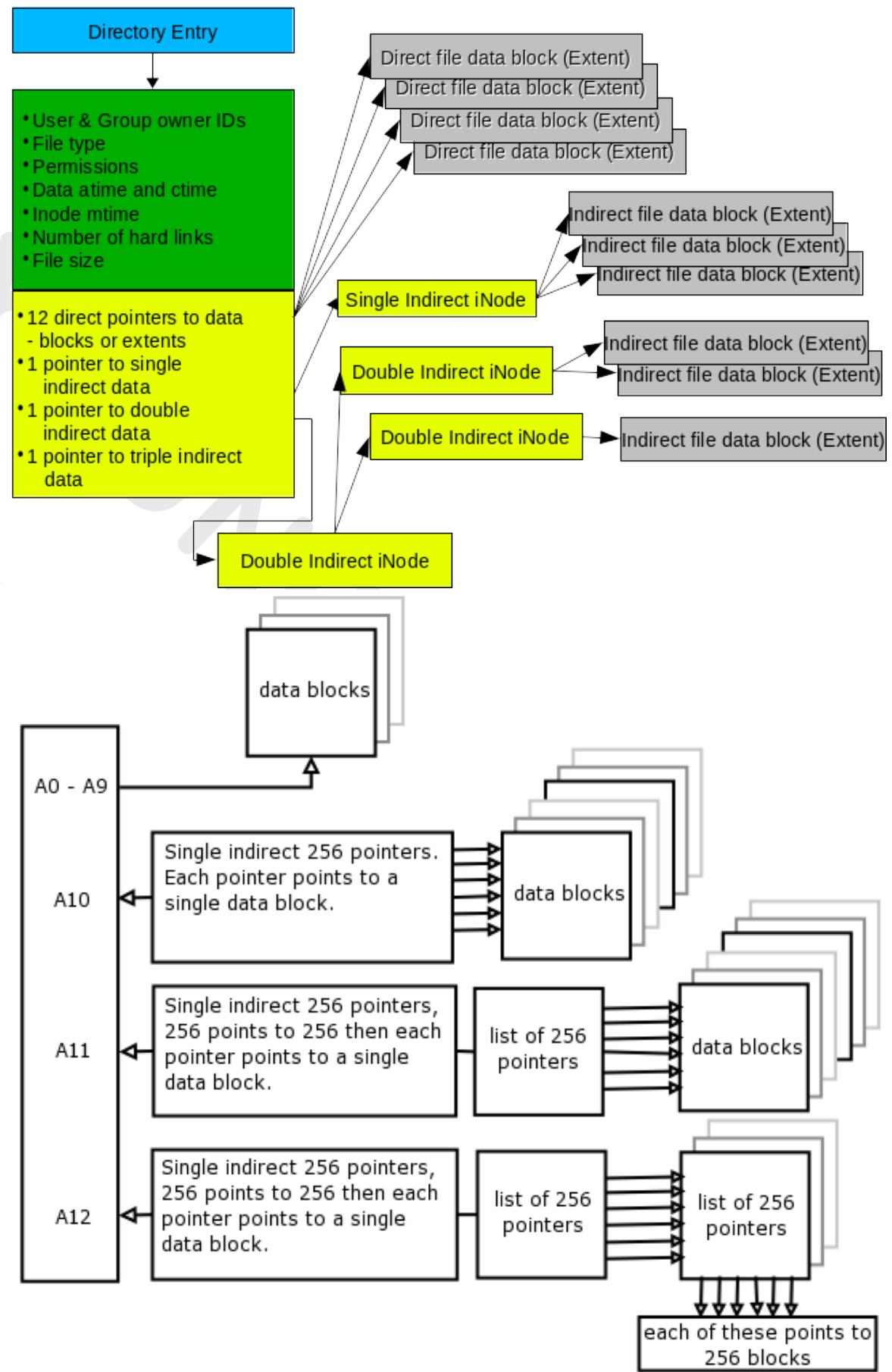
- 文件系统示意图



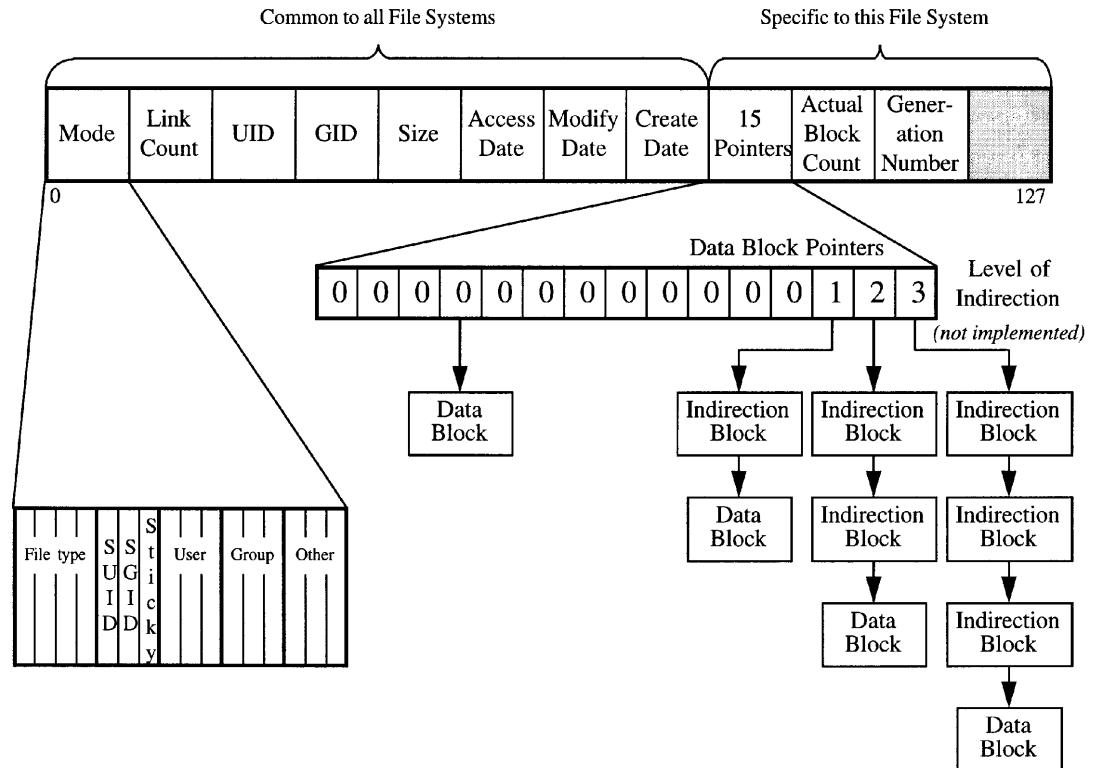
- data block: 真正的数据存储块, 1K/2K/4K 大小
- inode table: 非常重要, 存储如下内容
 - 文件存取模式 (r/w/x)
 - 文件的拥有者与群组 (owner/group)
 - 文件大小
 - 文件的相关时间 (ctime/atime/mtime)
 - 文件的其他 flag (SetUID...)
 - 真正内容的指向 (pointer)
- inode 查找示例图



- inode 结构:



Disk Inode



- 能编码的 block 数计算 (1K block)
 - 直接指向 12 个, $12 \times 1\text{K} = 12\text{K}$
 - 一级间接指向, $1\text{K}/4 \times 1\text{K} = 256\text{K}$
 - 二级间接指向, $1\text{K}/4 \times 1\text{K}/4 \times 1\text{K} = 256 \times 256\text{K} = 64\text{M}$
 - 三级间接指向, $1\text{K}/4 \times 1\text{K}/4 \times 1\text{K}/4 \times 1\text{K} = 16\text{G}$
- SuperBlock: 超级块, 存储
 - block 与 inode 的总量
 - 未使用和已使用 block/inode 数量
 - block 与 inode 的大小 (block: 1/2/4K, inode: 128B/256B)
 - 文件系统的挂载时间, 最近一次写入数据的时间, 最近一次检验磁盘的时间等信息
 - 一个 valid bit 数值, 若被挂载则为 0, 否则为 1
- FileSystem Description (Group Description)
 - 描述每个 block group 的开始与结束的 block 号码, 以及说明每个区段 (SuperBlock, bitmap, inodemap, data block) 分别介于哪个 block 号码之间
- Block Bitmap (Data Block Bitmap): 区块对照表, 通过它可得知哪些 block 被使用, 哪些为空, 存储/删除数据时, 需要同步更新
- inode Bitmap: inode 对照表, 类似 Block Bitmap, 存储 inode 的使用情况
- 一些相关命令
 - `dumpe2fs` : 查看 ext2/ext3/ext4 系列文件系统的相关信息
 - `blkid` : 查看系统被格式化的装置的 id

7.1.4 与目录树的关系

- 创建目录时, 文件系统会分配一个 inode 与至少一块 block 给该目录
 - inode: 记录该目录的权限与属性, 记录分配到的 block 号码
 - block: 记录这个目录下的文件名与该文件名占用的 inode 号码数据
 - 由上可见, ls 命令列目录时, 其大小均为 block size 的整数倍
- 目录树读取: 由根目录读起 (一般为挂载点), 找到其 inode 号码, 读取其 block 内的 inode 及文件名信息, 只要每

一步都具有相应的权限，就一层一层读，直至读到正确的文件名

7.1.5 文件的存取与日志式文件系统功能

- 数据不一致性问题，相关命令
 - `sync`
 - `fsck`
- 日志式文件系统 (Journaling filesystem)
 - 专门开辟日志区，存储文件操作记录，该记录与实际文件系统的操作同步
 - 出现不一致问题时，只需要对相关文件做恢复操作，避免对整个文件系统进行检查，提升效率

7.1.6 Linux 文件系统的运作

- 缓冲区技术
- 大内存能够提升文件系统的性能
- 可能会导致数据不同步问题

7.1.7 挂载点 (mount point)

- 挂载：将新文件系统链接到现有文件系统的目录树中
- 挂载点必须是目录，该目录为进入该文件系统的入口

7.1.8 Linux 支持的文件系统

- 支持的文件系统
 - 传统：exts/minix/MS-DOS/FAT(vfat)/iso9600...
 - 日志式：ext3/ext4/ReiserFS/Window NTFS/IBM JFS/SGI XFS/ZFS
 - 网络文件系统：NFS/SMBFS
- 查看命令
 - `ls -l /lib/modules/$(uname -r)/kernel/fs` 查看所有
 - `cat /proc/filesystems` 查看已加载

7.2 文件系统相关操作

7.2.1 磁盘与目录的容量

- `df`：列出文件系统的整体磁盘使用量
 - `-a`：列出所有文件系统
 - `-k`：以 KBytes 方式显示容量
 - `-m`：以 MBytes 方式显示容量
 - `-h`：以方便阅读的方式，自行选择 KB/MB/GB 格式进行显示
 - `-H`：以 M=1000K 替代 M=1024K 的进位方式
 - `-T`：列出 FS 类型
 - `-i`：以 inode 的方式显示
- `du`：评估文件系统的磁盘使用量
 - `-a`：列出所有文件与目录容量
 - `-h`：以方便阅读的方式，自行选择 KB/MB/GB 格式进行显示
 - `-s`：列出总量，而非每个单独的目录容量大小
 - `-S`：不包括子目录下的总计（存疑）
 - `-k`：以 KBytes 方式显示容量
 - `-m`：以 MBytes 方式显示容量

7.2.2 硬链接与符号链接 ln

- 硬链接: hard link, 多个文件对应同一个 inode, 也即, 仅仅增加了一项链接到某 inode 号的文件关联记录
 - `ln src_file target_file`
 - 不能跨文件系统
 - 不能 link 目录
- 符号链接: symbolic link, 创建独立文件, 这个文件让数据的读取指向它 link 的那个文件, 基本等同于 windows 下的快捷方式
 - `ln [-sf] src_file target_file`
- 关于目录的 link 数目
 - 目录的默认 link 数为 2, 包括 `.` 和 `..` 两个目录
 - 每在目录下创建一个新文件, 目录的 link 数增加 1

7.3 磁盘的分区、格式化、检验与挂载

7.3.1 查看磁盘状态

- `lsblk` 列出系统上所有磁盘列表
- `blkid` 列出设备的 UUID 等参数
- `parted` 列出磁盘的分区表类型与分区信息

7.3.2 磁盘分区

- `fdisk` 针对 MBR 分区表
- `gdisk` 针对 GPT 分区表
- `partprobe` 更新 Linux 核心的分区表信息

7.3.3 磁盘格式化

- `mkfs` 系列命令
 - `mkfs.mkfs.fat mkfs.btrfs mkfs.minix mkfs.cramfs mkfs.msdoc mkfs.ext2 mkfs.vfat mkfs.ext3 mkfs.xfs mkfs.ext4`
 - 例: `mkfs.ext4 [-b size] [-L label] device_name`
 - `-b` : 设定 block 大小, 1K/2K/4K 等
 - `-L` : 设置设备名称

7.3.4 文件系统检验

- `fsck` 系列命令, 类似 mkfs
- `fsck.ext4 [-pf] [-b superblock] device_name`
 - `-p` : 修复过程中, 自动回复 y 命令
 - `-f` : 强制检查
 - `-b` : 手工指明 superblock 的位置, 一般不用

7.3.5 文件系统挂载与卸除

- 注意事项
 - 单一文件系统不应该被重复挂载在不同的挂载点中
 - 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, 理论上都应该是空目录
- `mount` 命令
 - `mount -a`

- 根据 `/etc/fstab` 内容，挂载所有未挂载的磁盘
- `mount [-l]`
 - 查看系统的挂载信息
 - `mount [-t fs_type] LABEL=''` `mount_point`
 - `mount [-t fs_type] UUID=''` `mount_point`
 - `mount [-t fs_type] dev_name mount_point`
 - `-t` : 指定挂载磁盘的文件系统类型，常见的支持类型有：xfs/ext3/ext4/reiserfs/vfat/iso9660/nfs/cifs/smbfs
 - 依据设备标签、ID 或者设备名进行挂载
 - `-n` : 不将挂载情况写入/etc/mtab 文件
 - `-o` : 挂载时额外参数
 - `async, sync`: 异步写入或者同步写入
 - `atime, noatime`: 是否更新文件的 atime
 - `ro, rw`: 只读或读写
 - `auto, noauto`: 是否允许 mount -a 自动挂载
 - `dev, nodev`:
 - `suid, nosuid`:
 - `exec, noexec`:
 - `user, nouser`:
 - 是否允许在该挂载磁盘上进行参数所指定的操作
 - `defaults`: 如下选项的组合
 - `rw, suid, dev, exec, auto, nouser, async`
 - `remount`: 重新挂载
- `umount` 命令，卸除挂载文件
 - `umount [-fn] dev_name` 或 `mount_point`
 - 例：`umount /dev/sda1` 或 `umount /mnt/disk1`
 - `-f` : 强制卸除
 - `-l` : 立即卸除文件系统
 - `-n` : 卸除，但不更新/etc/mtab

7.3.6 磁盘/文件系统参数修订

- `mknod` 创建设备文件
- `tune2fs` 修改 ext4 的 label name 和 UUID

7.4 设定开机挂载

7.4.1 开机挂载/etc/fstab 及/etc/mtab

- `/etc/fstab` 格式
 - 纯文本文件，每行代表一条挂载记录，格式说明如下
 - `[dev_name 或 UUID] [挂载点] [文件系统] [文件系统参数] [dump] [fsck]`

7.4.2 挂载特殊装置

- iso 文件的挂载
 - `mount -o loop ubuntu1804.amd64.iso /mnt/cdrom`
- 制作 iso 文件
 - `dd if=/dev/cdrom of=~/test.iso`

7.5 关于 swap 分区

- 略

8 第 8 章文件与文件系统的压缩打包与备份

GZIP file format specification version 4.3

8.1 压缩文件类型

后缀	类型
*.Z	compress 程序压缩
.zip	zip 程序压缩
.gz	gzip 程序压缩
.bz2	bzip2 程序压缩
.xz	xz 程序压缩
.tar	tar 程序打包，没有压缩
.tar.gz	tar 程序打包，然后 gzip 压缩
.tar.bz2	tar 程序打包，然后 bzip2 压缩
.tar.xz	tar 程序打包，然后 xz 压缩

8.2 常见压缩命令

8.2.1 gzip, zcat/zmore/zless/zgrep

- gzip - 为了取代 compress 而出现
- **gzip [-cdtv#] 文件名**
 - 选项与参数:
 - **-c** : 将压缩的数据输出到屏幕上
 - **-d** : 解压缩
 - **-t** : 检验压缩文件一致性 (看看文件有无错误)
 - **-v** : 显示出压缩文件压缩比等信息
 - **-#** : # 为数字, 表示压缩等级, 1 最快, 压缩比最差、9 最慢, 压缩比最好。默认值为 6
 - 示例
 - gzip -v services
 - gzip -d services.gz
 - gzip -9 -c services > services.gz
 - zcat services.gz (zmore/zless 类似)
 - zgrep -n 'http' services.gz

8.2.2 bzip2, bzcat/bzmore/bzless/bzgrep

- bzip2 - 为了取代 gzip 而开发
- **bzip2 [-cdkzv#] 文件名**
 - 选项与参数:
 - **-c** : 将压缩的过程产生的数据输出到屏幕上
 - **-d** : 解压缩
 - **-k** : 保留源文件
 - **-z** : 压缩的参数 (默认值, 可以不加)
 - **-v** : 显示出压缩文件压缩比等信息
 - **-#** : 与 gzip 一样
 - 示例
 - bzip2 -v services
 - bzcat services.bz2
 - bzip2 -d services.bz2
 - bzip2 -9 -c services > services.bz2

8.2.3 xz, xzcat/xzmore/xzless/xzgrep

- xz - bzip2 的后继
- **xz [-dtlkc#] 文件名**
 - 选项与参数:
 - **-d** : 解压缩
 - **-t** : 测试压缩文件的完整性
 - **-l** : 列出压缩文件的相关信息
 - **-k** : 保留原本文件
 - **-c** : 将数据由屏幕上输出
 - **-#** : 同前
 - 示例
 - xz -v services
 - xz -l services.xz
 - xzcat services.xz
 - xz -d services.xz
 - xz -k services

8.3 打包指令 tar

- 常见使用方式
 - **tar [-z|-j|-J] [cv] [-f 欲创建文件名] filename...** 压缩
 - **tar [-z|-j|-J] [xv] [-f 已有文件名] [-C 目录]** 解压
 - **tar [-z|-j|-J] [tv] [-f 已有文件名]** 查看
- 选项与参数
 - **-c** : 建立打包文件, 可搭配 **-v** 来察看过程中被打包的文件名
 - **-t** : 察看打包文件的内容
 - **-x** : 解打包或解压缩, 可以搭配 **-C** 在特定目录解开
 - 注意: **-c, -t, -x** 不可同时出现在一串指令列中。
 - **-z** : 使用 gzip 进行压缩/解压缩, 文件名一般为 *.tar.gz
 - **-j** : 使用 bzip2 进行压缩/解压缩, 文件名一般为 *.tar.bz2
 - **-J** : 使用 xz 进行压缩/解压缩, 文件名一般为 *.tar.xz
 - 注意: **-z, -j, -J** 不可以同时出现在一串指令列中
 - **-v** : 在压缩/解压缩时, 将处理的文件名显示出来
 - **-f filename** : **-f** 后面要立刻接要被处理的文件名, 可以 **-f** 单独写一个选项, 以防忘记
 - **-C 目录** : 解压缩到指定目录
 - 其他可能会用到的选项
 - **-p** : 保留备份数据的原来权限与属性
 - **-P** : 保留绝对路径, 亦即允许备份数据中含有根目录存在
 - **--exclude=FILE** : 压缩过程中, 不要将 FILE 打包
- 示例
 - 备份/etc 目录
 - **tar -zpcv -f /root/etc.tar.gz /etc**
 - **tar -jpcv -f /root/etc.tar.gz /etc**
 - **tar -Jpcv -f /root/etc.tar.gz /etc**
 - 查看 tar 文件的数据内容
 - **tar -jtv -f /root/etc.tar.bz2** 注意: 为了安全, 查看时会移除文件名开头的'/'
 - **tar -jpPcv -f /root/etc.and.root.tar.bz2 /etc**
 - **tar -jtf /root/etc.and.root.tar.bz2**
 - 备份时存储原始路径, 查看确认, 解压时注意安全
 - 解压文件
 - **tar -jxv -f /root/etc.tar.bz2** 当前目录解压

- `tar -jxv -f /root/etc.tar.bz2 -C /tmp` 解压到指定目录
- 解出指定文件
 - `tar -jtv -f /root/etc.tar.bz2 | grep 'shadow'` 找到特定文件名
 - `tar -jxv -f /root/etc.tar.bz2 etc/shadow` 指定文件时，用刚刚找到的文件名
- 有选择的打包
 - `tar -jcv -f /root/system.tar.bz2 --exclude=/root/etc* --exclude=/root/system.tar.bz2 /etc /root`
注意使用了 `--exclude` 选项
- 仅备份较新的文件
 - `find /etc -newer /etc/passwd` 查到日期信息
 - `tar -jcv -f /root/etc.newer.then.passwd.tar.bz2 --newer-mtime="2015/06/17" /etc/*`
只备份某日期之后的文件
- 备份到磁带 `tar -cv -f /dev/st0 /home/root/etc`
- 使用管道示例 `tar -cvf - /etc | tar -xvf -` 将/etc 下文件拷贝到当前目录
 - 第一个 `-` 表示 standard output
 - 第二个 `-` 表示 standard input

8.4 xfs 文件系统备份

- 略

8.5 光盘相关

- 创建 iso 文件
 - `mkisofs [-o 输出文件] [-Jrv] [-V vol] [-m file] 待处理文件... -graft-point isodir=systemdir ...`
 - 选项与参数:
 - `-o` : 指定输出文件
 - `-J` : 产生较兼容于 windows 机器的文件名结构，可增加文件名长度到 64 个 unicode 字符
 - `-r` : Rock Ridge 光盘文件系统标准 (针对 UNIX/Linux 系统的 ISO-9660 文件系统)
 - 常见光盘文件系统标准有 `iso9660`、`UDF`、`Joliet`、`ROMEO`、`HFS`、`CD-RFS`、`Rock Ridge` 等
 - `-v` : 显示创建 ISO 文件的详细过程
 - `-V vol` : 建立 Volume, 有点像 Windows 看到的 CD title 的东西
 - `-m file` : `-m` 为排除文件 (exclude) 的意思，后面的文件不备份到映像档中，也能使用 * 通配符
 - `-graft-point` : 设定目录映射
 - 常见使用方式
 - `mkisofs -r -v -o /tmp/system.img /root /home /etc` 将三个目录文件全部创建在目标映像文件的同一目录下，如果有重名文件，处理很麻烦
 - `mkisofs -r -V 'linux_file' -o /tmp/system.img -m /root/etc -graft-point /root=/root /home=/ho` 分开处理
 - `mount -o loop /tmp/system.img /mnt` mount 映像文件
 - 创建开机启动映像文件 (书中示例，未测试)
 - `isoinfo -d -i /home/CentOS-7-x86_64-Minimal-1503-01.iso` 查看文件信息
 - `mount /home/CentOS-7-x86_64-Minimal-1503-01.iso /mnt` mount 到/mnt
 - `mkdir /srv/newcd` 创建工作目录
 - `rsync -a /mnt/ /srv/newcd` 将数据同步过去 (务必保留权限属性)
 - `cd /srv/newcd`
 - `mkisofs -o /custom.iso -b isolinux/isolinux.bin -c isolinux/boot.cat -no-emul-boot -V 'Cent0`
 - 光盘刻录 `cdrecord` 类似命令，略

8.6 其他常见工具

8.6.1 dd

- 作用 - convert and copy a file
- `dd if="input_file" of="output_file" bs="block_size" count="number"`
 - 选项与参数:
 - `if` : input file, 也可以是 device
 - `of` : output file, 也可以是 device
 - `bs` : 数据块大小, 默认值为 512 字节
 - `count` : 数据块数量
 - 示例:
 - `dd if=/etc/passwd of=/tmp/passwd.backup` 备份 passwd 文件
 - `dd if=/dev/sr0 of=/tmp/system.iso` 从磁带制作映像
 - `dd if=/tmp/system.iso of=/dev/sda` 从映像到磁盘

8.6.2 cpio

- 作用 - copy files to and from archives
- `cpio -ocvB > [file|device]` 备份
- `cpio -ivcd < [file|device]` 还原
- `cpio -ivct < [file|device]` 察看
- 选项与参数
 - `-o` : 将数据拷贝到文件或设备
 - `-B` : 将默认的 Block 大小增至 5120 字节
- 还原会使用到的选项与参数:
 - `-i` : 将数据从文件或设备复制到系统中
 - `-d` : 自动建立目录
 - `-u` : 自动的用较新的文件覆盖较旧的文件
 - `-t` : 配合 `-i` 选项, “察看”以 cpio 建立的文件或装置的内容
- 一些可共享的选项与参数:
 - `-v` : 让储存的过程中文件名可以在屏幕上显示
 - `-c` : 一种较新的 portable format 方式储存
- 示例
 - `find boot | cpio -ocvB > /tmp/boot.cpio`
 - `find / | cpio -ocvB > /dev/st0` 备份
 - `cpio -idvc < /dev/st0` 还原

9 vim 编辑器

GNU 下的 joke - <https://www.gnu.org/fun/>

关于编辑器 ed 的 joke - <https://www.gnu.org/fun/jokes/ed-msg.html>

9.1 编辑器的历史

- ed - 最古老的编辑器，由 Unix 之父 Ken Thompson 编写

- 受来自于加州伯克利大学的 qed 编辑器的影响
- 第一次应用了正则表达式 (regular expression)
- 是一种行编辑器

- 一个示例

```
# ed greeting
0      # 因为新创建文件，所以读入了0个字节
a      # 进入编辑模式(append)
hello world, eveyone.    # 输入一行文本
.      # 回到命令模式
1p    # 显示(print)第一行
hello world, eveyone.
1c    # 最后一个词写错了，修改(change)第一行
hello world, everyone.
.      # 回到命令模式
1p    # 重新显示第一行，这回发现无误
hello world, everyone.
q      # 退出(quit)
?      # ? 表示没有保存或者命令不认识
w      # 保存(write)
23    # 提示写了23个字节
q      # 退出
```

- ex - ed 的扩展，vi 即构建在 ex 之上

- vi - 硬件的发展，使得全屏幕编辑器被开发出来

- 有 pico, nano, joe, jed, jove 等，但使用的最广泛的就是 vi 和 emacs
- 1976 年，伯克利大学的 Bill Joy 开发了 vi，作为 ex 的 visual interface，用户一次可以编辑一屏的文本

- vim - 随着技术进步和需求增多，vi 的各种变体不断涌现

- 知名的有 nvi, elvis, vim, vile, yzis 等
- 移植性最好、特性最多、使用最广的即为 vim，开发者是 Bram Moolenaar
- 起初，VIM 表示 **Vi IMitation** 的意思，1992 年，1.22 版的 vim 被移植到了 UNIX 和 MS-DOS 上，此后，VIM 全名变为 **Vi IMproved**

9.2 vi 与 vim

- Linux 系统中，绝大多数要处理的文件是纯文本文件，文本文件的编辑效率，直接影响到我们的工作效率

- 学习 vim 必要性

- 所有 linux 系统都内置此命令
- vim 是很多软件的内置编辑器
- 编辑功能极其强大
- 体积小，启动快

- vim 的版本历史，参见 [vim-history](#)

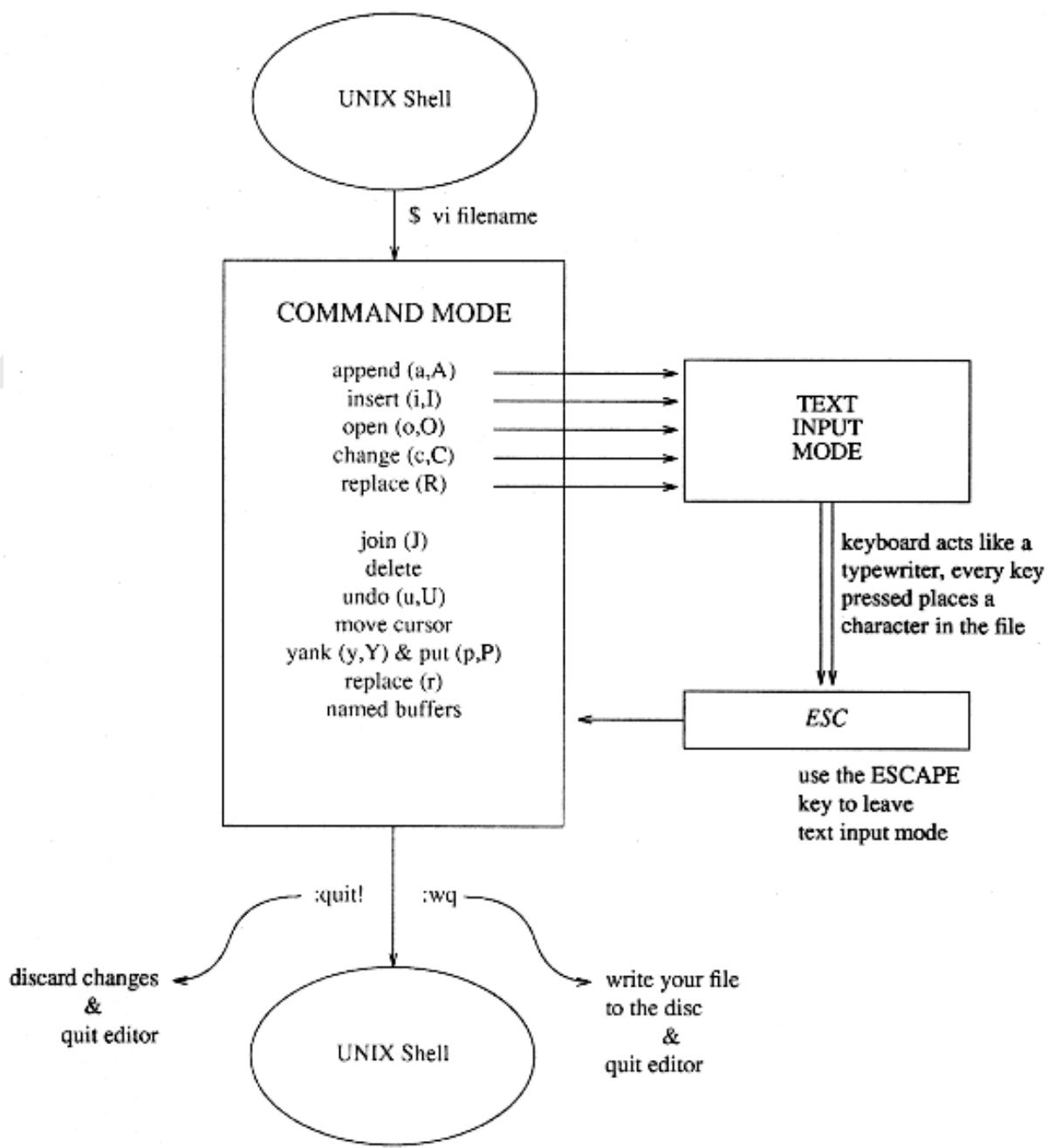
时间	版本	里程碑
2018 May 17	Vim 8.1	

时间	版本	里程碑
2016 Sep 12	Vim 8.0	Asynchronous I/O, channels, Jobs, Timers, Partials, Lambda and Closure, Packages
2013 Aug 10	Vim 7.4	A new, faster regex engine
2010 Aug 15	Vim 7.3	Persistent undo/redo, Blowfish encryption, Lua/Python support
2008 Aug 09	Vim 7.2	Floating point in scripts
2006 May 07	Vim 7.0	Spell Checking, multi-tabpage, code completion
2001 Sep 26	Vim 6.0	Folding (and more), plugin, multilingual
1998 Feb 19	Vim 5.0	Syntax highlighting, scripting
1996 May 29	Vim 4.0	Graphical User Interface (Robert Webb)
1994 Aug 12	Vim 3.0	Support for multiple buffers and windows.
1992	Vim 1.22	Port to Unix. vim now competes with vi. This was when VIM became Vi IMproved .
1991 Nov 02	Vim 1.14	First release (on Fred Fish disk #591).

- vim 的老家 - www.vim.org

9.3 vi 的使用

- 理解三种基本模式
 - command mode: 命令模式, 此为默认模式, vi 打开文件时, 即为此模式, 该模式下, 输入的所有内容, 会被理解为命令 (移动光标、删除字符和行、复制、粘贴), 而非编辑内容
 - insert mode: 插入模式, 真正的编辑模式, 在命令模式下输入 `i I o O a A r R` 中任一键, 均可进入编辑模式 (注意编辑器左下角的状态栏信息变化), 然后进行实际编辑操作, 输入 `Esc` 回到命令模式
 - command line mode: 命令行模式, 也叫最后行模式, 命令模式下, 输入 `: / ?` 中任一键, 光标会移动到最底下那一行, 可供命令输入 (搜索内容、读取或存储文件、字符替换、退出 vi 等), 输入 `Esc` 回到命令模式



9.3.1 简单示例

- `vi filename` 打开文件
- `i` 进入编辑模式
- 编辑文件
- `esc` 回到命令模式
- `:` 到最后行模式
- `wq!` 存盘退出或者 `q!` 不存盘退出

9.3.2 按键说明

9.3.2.1 命令模式

- 光标移动

- 命令列表

命令	动作
h 或左箭头 (<left>)</left>	光标向左移动一个字符
j 或下箭头 (<down>)</down>	光标向下移动一个字符
k 或上箭头 (<up>)</up>	光标向上移动一个字符
l 或右箭头 (<right>)</right>	光标向右移动一个字符
备注	想要进行多次移动，例如向下移动 30 列，可以使用 “30j” 或 “30 <down>” 的组合按键</down>
[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列
-	光标移动到非空格符的上一列
n	n 表示『数字』，按下数字后再按空格键，光标会向右移动这一列的 n 个字符
0 或功能键 [Home]	移动到这一列的最前面字符处
\$ 或功能键 [End]	移动到这一列的最后面字符处
H	光标移动到这个屏幕的最上方那一列的第一个字符
M	光标移动到这个屏幕的中央那一列的第一个字符
L	光标移动到这个屏幕的最下方那一列的第一个字符
G	移动到这个文件的最后一行
nG	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列，相当于 1G
gg=G	对当前文档进行格式化
n	n 为数字。光标向下移动 n 行
(,)	跳到前/后一句的句首
{, }	跳到前/后一段的段首
f	在当前行搜索指定字符，并将光标停留在匹配字符上
t	在当前行搜索指定字符，并将光标停留在匹配字符的左边
%	搜索与当前字符匹配的位置（比如括号对），并跳到匹配处

- vi 中 字 的概念

- 广义：两个空格之间的任何内容为一个字
- 狭义：空格是字的定界符，并且所有标点和非字母字符均被当成一个字
- 示例：`printf("Hello, world\n");`
 - 广义 - 两个字
 - 狭义 - 11 个字

命令	动作
w, W	跳到下一个词的词首（狭义/广义）
b, B	跳到上一个词的词首（狭义/广义）
e, E	跳到下一个词的词尾（狭义/广义）

- 查找与替换

命令	动作
/word	从光标处向后查找 word 字符串
?word	从光标处向前查找 word 字符串
n	重复前一个动作
N	反向重复前一个动作
:n1,n2s/word1/word2/g	在 n1 到 n2 行之间，查找所有 word1，并替换为 word2
:1,\$ /word1/word2/g	从第 1 行到最后一行，查找所有 word1，并替换为 word2
:1,\$ /word1/word2/gc	从第 1 行到最后一行，查找所有 word1，并替换为 word2，但每次替换前要求用户确认

- 删除、复制、粘贴

命令	动作
x, X	x = del, X = backspace
nx	删除 n 个字符
dd	删除光标所在行
ndd	删除 n 行
d1G	删除光标所在行到第一行的所有数据
dG	删除光标所在行到最后一行的所有数据
d\$	删除光标所在位置到该行的最后一个字符
d0	删除光标处到该行的第一个字符
D	删除从光标所在位置直到行尾的所有字符
yy	拷贝当前行
nyy	拷贝 n 行
y1G	拷贝从当前行到第一行的所有数据
yG	拷贝当前行到最后一行的所有数据
y0	拷贝光标所在的字符到当前行首的所有字符
y\$	拷贝光标所在的字符到当前行尾的所有字符
yw	拷贝当前单词
yl	拷贝当前字符
p	将拷贝的数据粘贴到当前行后
P	将拷贝的数据粘贴到当前行前
J	将当前行与下一行合并
c	重复删除多个数据, 例如 10cj, 向下删除 10 行
cw	替换一个单词 (狭义)
cW	替换一个单词 (广义)
cc	替换整行
C, c\$	替换到行尾
u	undo
ctrl + r	redo
.	重复前一个动作
ctrl+l	重画屏幕, 刷新屏幕
ctrl+g	显示当前编辑文本的状态

- 补充:

- `["char"] [n]y [<cursor_movement>|y|w|l]` 拷贝特定内容到指定寄存器, 其中 yy, yw, yl 的解释见上表
 - "ayy 拷贝当前行内容到 a 寄存器
 - "a10yy 拷贝 10 行内容到 a 寄存器
- `["char"]p|P` 从寄存器粘贴内容到当前行前 | 后
- 可用寄存器字符为 `a-zA-Z0-9`
- 使用 `:reg` 查看寄存器内容
- `:help registers`

9.3.2.2 编辑模式

命令	动作
i, I	i 为『在当前光标所在处插入』, I 为『在目前所在行的第一个非空格符处开始插入』
a, A	a 为『从当前光标的下一个字符处开始插入』, A 为『从光标所在行的行尾开始插入』
o, O	o 在目前光标的下一行插入新的一行, O 在当前光标的上一行插入新的一行
r, R	r 取代光标所在的那个字符一次, R 一直对光标处字符进行替换, 直至 esc 回命令模式

9.3.2.3 最后行模式

- 说明，以下列出的命令，均为实际所需键入的字符序列，例如：`:w`，表示在命令模式下，依次键入`:` `w`

命令	动作
<code>:w</code>	将数据写入文件
<code>:w!</code>	强制写入（如果文件为只读，但拥有 owner 权限，vi 会删除原来文件，并创建一个新的）
<code>:q</code>	退出 vi
<code>:q!</code>	强制退出，放弃修改
<code>:wq</code>	存盘退出
<code>ZZ</code>	存盘退出
<code>:w filename</code>	存为另一个文件
<code>:r filename</code>	在当前光标行下，读入 filename 文件内容
<code>:n1,n2 w filename</code>	将 n1 到 n2 行的内容存储为 filename
<code>:f filename</code>	将当前文件重命名为 filename
<code>:f</code>	显示当前文件名称和状态
<code>:n1,n2 m n3</code>	将 n1 到 n2 行的内容移动到 n3 行的后面
<code>:n1,n2 t n3</code>	将 n1 到 n2 行的内容拷贝到 n3 行的后面
<code>:! command</code>	暂时离开 vi，到 shell 中执行 command，回车后返回 vi
<code>:set nonu</code>	set nonumber 的缩写，不显示行号
<code>:set nu</code>	set number 的缩写，显示行号

• 本模式下光标的移动

- 数值用来指定绝对行号
 - `:1` 跳转到第 1 行
- 字符`.` 表示光标所在行的行号
- 字符`$` 表示正文最后一行的行号
- 字符`%` 表示所有行
 - `..+5` 表示定位到当前行往下的第 5 行
 - `:$` 跳转到最后一行

9.3.3 案例练习

- 完成下文中描述的所有操作（书中案例）
 - 请在`/tmp`这个目录下建立一个名为`vitest`的目录
 - 进入`vitest`这个目录当中
 - 将`/etc/man_db.conf`复制到本目录
 - 使用 vi 打开本目录下的`man_db.conf`这个文件
 - 在 vi 中设定行号
 - 移动到第 43 行，向右移动 59 个字符，请问你看到的小括号内是哪个文字？
 - 移动到第一行，并且向下搜寻一下`gzip`这个字符串，请问他在第几行？
 - 接下来，将 29 到 41 行之间的 小写 man 字符串 改为 大写 MAN 字符串，并且 一个一个挑选 是否需要修改，如何下达指令？如果在挑选过程中一直按`y`，结果会在最后一行出现改变了几个 man 呢？
 - 修改完之后，突然反悔了，要全部复原，有哪些方法？
 - 复制 66 到 71 这 6 行的内容（含有`MANDB_MAP`），并且贴到最后一行之后
 - 113 到 128 行之间的开头为`#`符号的批注数据我不要了，要如何删除？
 - 将这个文件另存成一个`man.test.config`的文件名
 - 去到第 25 行，并且删除 15 个字符，结果出现的第一个单字是`*`什么？
 - 在第一行新增一行，该行内容输入`I am a student...`
 - 储存后离开

9.3.4 其他相关知识

- 关于备份文件
 - 禁止生成备份文件，`set nobackup`

- 允许生成备份文件, `set backup`
- 设置备份文件存放目录, `set backupdir=/path/to/dir`
- 关于临时交换文件
 - 确保编辑工作不丢失
 - 非正常退出 vi 时, 会在机器里留存一个 `.swp` 文件, 存储的是 vim 缓冲区中的内容
 - 每次意外退出时, 不会覆盖旧交换文件, 而是生成新的交换文件
 - 示例, 若编辑文件 `file.txt`, 第一次生成的交换文件为 `.file.txt.swp`, 第二次生成 `.file.txt.swo`, 第三次则为 `.file.txt.swn`, 然后是 `.file.txt.swm`, 依此类推
 - `vim -r` 查看所有 swp 文件
 - `vim -r filename` 恢复文件, 使用上次意外退出没有保存的修改, 覆盖 filename 文件内容, 最后使用 rm 删除交换文件
 - 禁止生成 swp 文件, `set noswapfile`
 - 恢复生成 swp 文件, `set swapfile`
 - 设置临时交换文件的存放目录, `set directory=/path/to/dir`
 - 默认设置, 交换文件每隔 4000 毫秒或者 200 字符保存一次, 如果修改, 可
 - `set updatetime=2000`
 - `set updatecount=400`
- 关于 undo 文件
 - `set nounofile`
 - `set undodir=/path/to/dir`

9.3.5 区块选择

命令	动作
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	区块选择, 以列的方式进行编辑
y	拷贝选择内容
d	删除选择内容
p	复制选择内容

- visual mode : 可视化模式
 - 命令模式下键入 `ctrl+v` 进入该模式
 - 在该模式下, 可以按照列的方式操作文本区域
 - 输入 `Esc` 回到命令模式

9.3.6 多文件编辑

命令	动作
:files	查看目前 vim 打开的所有文件
:n	编辑下一个文件
:N	编辑上一个文件

9.3.7 多窗口功能

命令	动作
:sp [filename]	当前文件分窗口同步编辑, 或者在新窗口打开一个新文件 filename
ctrl+w j (或 ↓)	先按下 ctrl 不放, 再按下 w 后放开所有的按键, 然后再按下 j (或向下箭头键), 则光标可移动到下方的窗口
ctrl+w k (或 ↑)	同上, 不过光标是移动到上方的窗口

命令	动作
ctrl+w q	结束离开

9.3.8 名称补全功能

命令	动作
ctrl+x ctrl+n	利用目前正在编辑的文件内容作为关键词，予以补全
ctrl+x ctrl+f	利用当前目录的文件名作为关键词，予以补全
ctrl+x ctrl+o	以扩展名作为语法补充，以 vim 内建的关键词，予以补全

9.3.9 宏

- 命令模式下输入 `q<reg>` (指的是 `a-zA-Z0-9` 共 37 个寄存器其中之一)，启动宏录制
- 然后可以进行任何操作，包括模式间切换
- 在命令模式下输入 `q` 结束宏录制
- 使用 `@<reg>` 执行该宏
- `:help q`

9.3.10 书签

- 命令模式下输入 `m<reg>` 制作书签
- 使用 `'<reg>` 或 反单引号 `<reg>` 跳转到指定书签处
- `:help m`

9.3.11 其他

- `:s//, :g//, :g!//`
 - 语法 `:[range]s|g|v/{pattern}/[cmd]`
 - 配合正则表达式，这三个命令常常能完成非常复杂的任务
 - `:[range]s/{pattern}/[cmd]` 替换操作
 - `:[range]g[lobal]/{pattern}/[cmd]` 查找匹配模式的行
 - `:g!//` 用于查找不匹配模式的行，等价于 `:v//`
 - 说明及示例
 - `[cmd]` 默认是 ex 的命令 `:p` 即在屏幕上打印输出
 - 如果要执行一个非 ex 的命令，可以使用 `:normal` 命令，如下
 - `:g/{pattern}/normal {command}`
 - `:g/pat/s//PAT/g` 将替换所有的 pat 为 PAT，相当于 `:%s/pat/PAT/g`
- `help :s`
- `help :g`
- 插件 (plug-in)
 - vim 有自己的脚本语言，同时也支持使用 Perl/Python/Tcl/Ruby 等语言编写插件
 - 配置文件为 `~/.vimrc`，配置文件存放目录位置 `~/.vim`
 - vim 官方网址 <https://www.vim.org>，那里有成千上万的插件
 - 有很多的第三方插件管理器，比如 [Vundle](#)、[vim-plug](#) 以及比较新的[vire](#)
 - vim 8 引入了[新的包管理机制](#)，未来也许会替代目前大部分的包管理器
- 配色
 - `:colorscheme delek` 设定使用某种配色方案
 - 也可简写为 `:colo delek`

- vi 安装时，已经默认安装了一些配色方案，可以试着到 `/usr/share/vim/vim72/colors` 类似的目录去查看
- 在 [VIM Color Scheme Test](#) 页面可以查看各种配色方案的效果，该页面也提供了配色方案下载的链接
- 自动缩进（本应属于编辑功能，放这里单独讨论）
 - 打开自动缩进选项后，vim 会自动控制缩进
 - 对于选择的块，使用 `>>` 和 `<<` 可以很方便地调整其缩进
 - 使用 `=` 命令，可以对选择程序进行排版
 - `help cindent`
 - `help autoindent`
 - `help smartindent`
- 代码折叠
 - `:help fold`

9.3.12 环境设定

- 定制 vim 编辑环境，三种方式
 - 运行 vim 时，使用 `set` 命令设定
 - 通过环境变量 `EXINIT` 进行设定
 - 使用全局配置文件 `/etc/vimrc`，个人配置文件为 `~/.vimrc` 或 `~/.exrc`
- 一些常见设置

设置	功能
<code>:set nu</code>	设定行号
<code>:set nonu</code>	取消行号
<code>:set hlsearch</code>	设定高亮搜索
<code>:set nohlsearch</code>	取消高亮搜索
<code>:set autoindent</code>	设置自动缩进
<code>:set noautoindent</code>	取消自动缩进
<code>:set ruler</code>	在右下角显示状态栏说明
<code>:set showmode</code>	是否显示“-INSERT-”之类的左下角状态栏
<code>:set backspace=(012)</code>	进入编辑模式时，可以任意删除原有字符还是只能删除新输入的字符
<code>:set all</code>	显示所有环境参数设定数值
<code>:set</code>	显示与系统默认值不同的设定数值，也即用户设定过的参数
<code>:syntax on</code>	语法加亮
<code>:syntax off</code>	取消语法加亮
<code>:set bg=dark</code>	设置背景的颜色色调
<code>:set bg=light</code>	同上

- 以上内容都可以固化到 `~/.vimrc` 中
- 由于一台机器中可能会安装 vim 的多个版本，为了得知当前的 vim 安装位置，可以在 `最后行方式` 使用 `echo $VIM` 获取其配置文件的存放位置，使用 `echo $HOME` 获取 vim 的安装主目录

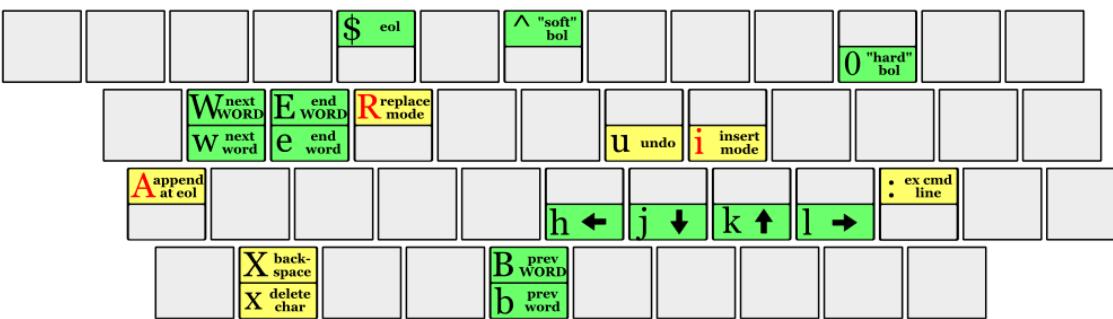
9.3.13 vim cheat code

- 一个 step by step 的 cheat code 版本
 - basic editing

vi/vim lesson 1 - basic editing

motion moves the cursor, or defines the range for an operator
command direct action command, if red, it enters insert mode

ESC
normal mode



Basics:

h **j** **k** **l** are vi/vim cursor keys – use them as they are much closer than regular cursor keys!

Use **i** to enter insert mode, cursor turns from a block into a vertical line, and you can type in text. Use **Esc** to return to normal mode.

Use **X** to delete the current character, or **X** to delete the one to the left

Use **A** to go insert text at the end of the line (wherever you are in the line!)

(Note: insert mode is actually very similar to a regular editor, you can use cursor/navigation keys, backspace, delete...)

Extras:

u to undo the last action – traditional vi has a single level, while vim supports unlimited undo (CTRL - **R** to redo)

0 jumps directly to the beginning of the line, **\$** to the end, and **^** to the first non-blank

Use **w** **b** **e** to move along 'words'. A 'word' is a sequence of all alphanumeric or punctuation signs: **quux(foo, bar, baz)**

Use **W** **B** **E** to move along WORDS. A 'WORD' is a sequence of any non-blank characters: **quux(foo, bar, baz)**

Use **R** to enter insert mode with an overstrike cursor, which types over existing characters.

: **w** and press enter to save, **:** **q** and enter to quit.

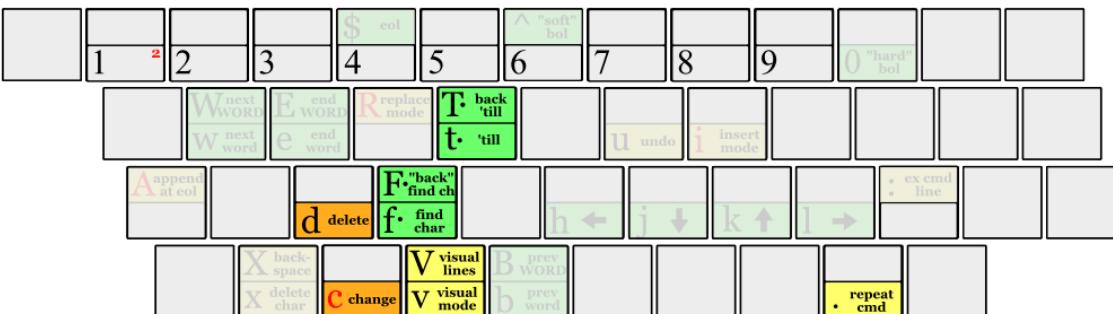
For the rest of the tutorial & a full cheat sheet, go to www.viemu.com - home of ViEmu, vi/vim emulation for Microsoft Visual Studio

- operators & repetition

vi/vim lesson 2 - operators & repetition

learned in previous lessons
motion moves the cursor, or defines the range for an operator
command direct action command, if red, it enters insert mode
operator requires a motion afterwards, operates between cursor & destination

ESC
normal mode



Basics:

f, followed by another key, moves the cursor to the next instance of that character on the current line. **F** does the same backwards.

t and **T** do the same, but they stop right before the character.

d(delete), followed, by any motion deletes the text between the cursor and that motion's destination **d w**, **d f** ...).

C(change) does the same, but leaves you in insert mode.

Some motions, such as **J** and **K**, are linewise – deletion includes the full start/end lines.

. repeats the last editing action: text input, delete or change, etc... motion is recalculated at the new place.

Extras:

Prepend a count to any command/motion to repeat it that number of times:

d 2 w to delete up to the second word.

d 2 t . to delete up to but not including the second comma.

2 i repeats the text after you press (Esc) to finish the input session.

Repeat operator (**c c** or **d d**) to operate on the current line.

Only in vim, **V** enters visual mode. Move around with motions, the text will be highlighted. Press an operator to operate on that selection.

V enters visual-lines mode – like **V**, but selecting whole lines.

CTRL - **V** selects rectangular blocks.

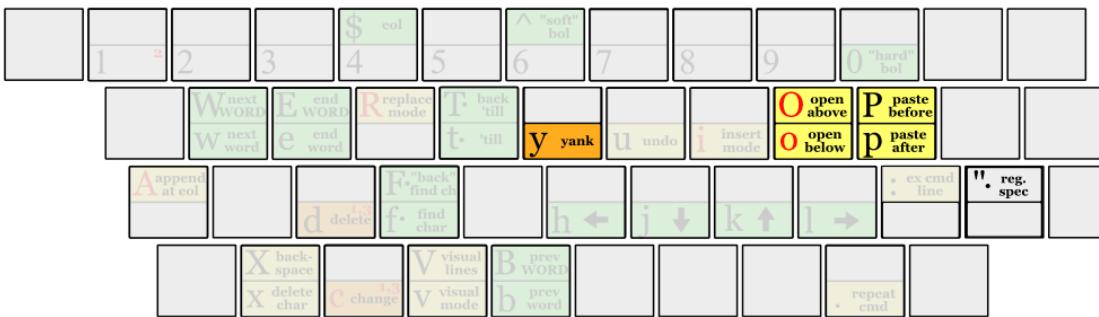
For the rest of the tutorial & a full cheat sheet, go to www.viemu.com - home of ViEmu, vi/vim emulation for Microsoft Visual Studio

- yank & paste

vi/vim lesson 3 - yank & paste

	learned in previous lessons
	motion moves the cursor, or defines the range for an operator
	command direct action command, if red, it enters insert mode
	operator requires a motion afterwards, operates between cursor & destination
	extra special functions, requires extra input

Esc
normal mode



Basics

Use **Y** followed by any motion to 'yank' (copy).

Use **P** to paste after (if charwise, to the right, if linewise, below).

Use **P** to paste before.

Y Y copies the current line.

Y also works in visual mode.

Text deleted with **D**, **C**, **X**... is also copied!

Extras

" and an **[a]-[z]** character before any yank/delete/paste command chooses a register.

An **[A]-[Z]** register before yank/delete means "append-copy".

"* or "++ select the system clipboard.

I enters insert mode in a new empty line below the current one.

O does the same above the current line.

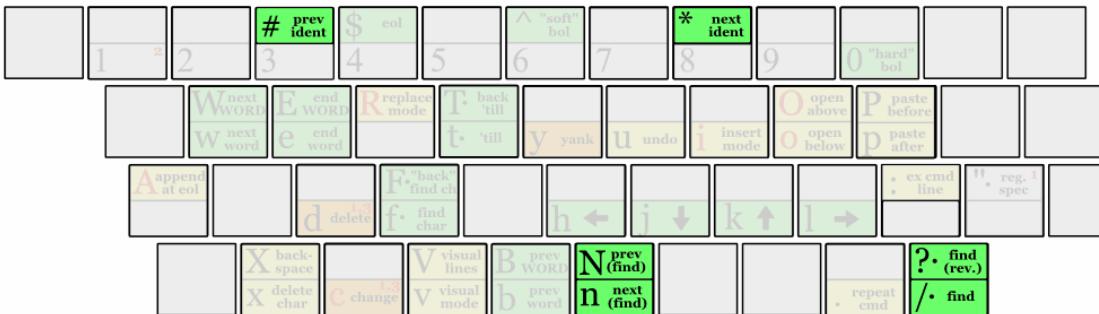
For the rest of the tutorial & a full cheat sheet, go to www.viemu.com - home of ViEmu, vi/vim emulation for Microsoft Visual Studio

- searching

vi/vim lesson 4 - searching

	learned in previous lessons
	motion moves the cursor, or defines the range for an operator
	command direct action command, if red, it enters insert mode
	operator requires a motion afterwards, operates between cursor & destination
	extra special functions, requires extra input

Esc
normal mode



Basics:

is the basic search motion – type the text you are searching for after the slash, and then press return. Being a motion, you can use this after an operator, or in visual mode.

? does the same, backwards.

N repeats the last search in the same direction, **N** repeats it in the reverse direction

Be careful, because the search target is interpreted as a regular expression: **a*b** means zero or more 'a's followed by a 'b', **^abc** means 'abc' at the beginning of a line, **[0-9]** looks for the next digit, etc...

Extras:

The following very useful motions work only in vim:

g searches forward for the next instance of the identifier under the cursor.

g does the same backwards.

For the rest of the tutorial & a full cheat sheet, go to www.viemu.com - home of ViEmu, vi/vim emulation for Microsoft Visual Studio

- marks & macros

vi/vim lesson 5 - marks & macros

	learned in previous lessons
	moves the cursor, or defines the range for an operator
	direct action command, if red, it enters insert mode
	operator, if red, it enters insert mode, requires a motion afterwards, operates between cursor & destination
	extra special functions, requires extra input

Marks:
Use **m** followed by an **a** - **z** character to set a mark.
Use **g** followed by a character to go to that mark.
Use **g** and a character to go to the first non-blank in that line.
A - **Z** marks are global, **a** - **z** per-buffer.
g refers to the position of the last modification.

Macros:
Use **Q** followed by an **a** - **z** character to start recording.
Use **Q** afterwards to stop recording.
@ followed by a character replays that macro.
@@ to repeat the last macro played.

For the rest of the tutorial & a full cheat sheet, go to www.viemu.com - home of ViEmu, vi/vim emulation for Microsoft Visual Studio

- various motions

% jumps between matching pairs of '{', '}', '[', etc...
H M L jump directly to the top/middle/bottom of the screen.
G jumps to the end of the file, or to the line # typed before it.
- + jump to the previous/next line.
K, not technically a motion, jumps to the help for the word under the cursor: vim help, man page under unix, etc...

C and **D** jump to the beginning/end of the current sentence.
G and **H** jump to the previous/next empty line.
I I jumps to the previous '{' in column 0.
J J jumps to the next '{' in column 0.

For the rest of the tutorial & a full cheat sheet, go to www.viemu.com - home of ViEmu, vi/vim emulation for Microsoft Visual Studio

- various commands

vi/vim lesson 7 - various commands

 learned in previous lessons
 motion moves the cursor, or defines the range for an operator
 command direct action command, requires a motion afterwards, operates between cursor & destination
 operator requires a motion afterwards, operates between cursor & destination
 extra special functions, requires extra input

Esc normal mode															
~ toggle case	1	2	3	4	5	6	7	8	9	- prev line	+ next line	= auto format			
* goto mark			@ play macro	# prev ident	\$ eol	% goto match	^ "soft" bol	*	(begin sentence) end sentence					
			W next WORD	E end WORD	R replace mode	T back 'till	Y yank line	U undo	I insert mode	O open above	P paste before	{ begin parag.	}	end parag.	
			g record macro	w next word	e end word	r replace char	t 'till	v yank	u undo	i insert mode	o open above	p paste after	{ begin parag.	}	end parag.
			A append at col	S subst line	D delete to col	F back find ch	G eol goto ln	H screen top	J join lines	K help	L screen bottom	* ex cmd line	H * reg spec		
			append at col	subst line	delete to col	back find ch	eol goto ln	screen top	join lines	help	screen bottom	ex cmd line	reg spec		
			S subst char	d delete	f find char			j down	k up	l right					
			X back-space	C change to col	V visual lines	B prev WORD	N prev (find)	M screen mid'l	< un-indent	> indent	? find (rev.)				
			X delete char	C change	V visual mode	B prev word	N next (find)	M set mark			/ find				

Basics:

- [J] joins the current line with the next one, or all the lines in the current visual selection.
- [R] followed by any character replaces the current character with that one.
- [C] is shorthand for [c] [S], changes to end of line.
- [D] is shorthand for [d] [S], deletes to end of line.
- [Y] is shorthand for [y] [y], yanks the whole line.
- [S] deletes the character under the cursor and enters insert mode.
- [S] clears the current line and enters insert mode.

Extras:

- [>] and a motion to indent one or more lines.
 - [<] and a motion to unindent.
 - [=] and a motion to reformat a range of text.
- All of them work in visual mode, or can be repeated ([>] [>], etc...) to operate on the current line.
- [~] toggles the case of the character under the cursor.

Now go grab the full cheat sheet and learn the rest.
Start with [I] [a] and [e]. Piece of cake!

For the rest of the tutorial & a full cheat sheet, go to www.viemu.com home of ViEmu, vi/vim emulation for Microsoft Visual Studio

• 其他 jpg 版本

version 1.1
April 1st, 06

vi / vim graphical cheat sheet

Esc normal mode															
~ toggle case	1	2	3	4	5	6	7	8	9	0 "hard" bol	- prev line	+ next line	= auto format		
* goto mark			@ external filter	# play macro	\$ prev ident	% eol	^ match	& repeat :s	*	(begin sentence) end sentence	"soft" bol			
			Q ex mode	W next WORD	E end WORD	R replace mode	T back 'till	Y yank line	U undo line	I insert at bol	O open above	P paste before	{ begin parag.	}	end parag.
			Q record macro	W next word	E end word	R replace char	T 'till	y yank	U undo	I insert mode	O open below	P paste after	{ misc	}	misc
			A append at col	S subst line	D delete to col	F "back" find ch	G eol goto ln	H screen top	J join lines	K help	L screen bottom	: ex cmd line	H * reg spec		
			A append	S subst char	D delete	F find char	G extra cmd	h left	j down	k up	l right	:	repeat t/T/U/F	' goto mk. bol	* not used
			Z quit	X back-space	C change to col	V visual lines	B prev WORD	N (find)	M screen mid'l	< un-indent	> indent	? find (rev.)			
			Z extra cmd	X delete char	C change	V visual mode	B prev word	N next (find)	M set mark	reverse t/T/f/F	repeat cmd	/ find			

motion moves the cursor, or defines the range for an operator

command direct action command, if red, it enters insert mode

operator requires a motion afterwards, operates between cursor & destination

extra special functions, requires extra input

Q commands with a dot need a char argument afterwards

bol = beginning of line, eol = end of line, mk = mark, yank = copy

words: `suux{foo, bar, baz}`

WORDs: `suux{foo, bar, baz}`

Main command line commands ('ex'):

:w (save), :q (quit), :q! (quit w/o saving)
:e f (open file),

:%s/x/y/g (replace 'x' by 'y' filewide),

:h (help in vim), :new (new file in vim),

Other important commands:

CTRL-R: redo (vim),

CTRL-F/B: page up/down,

CTRL-E/-Y: scroll line up/down,

CTRL-V: block-visual mode (vim only)

Visual mode:

Move around and type operator to act on selected region (vim only)

Notes:

(1) use "x before a yank/paste/del command to use that register ('clipboard') (x=a..z, "e.g.: "ay\$ to copy rest of line to reg 'a')

(2) type in a number before any action to repeat it that number of times (e.g.: 2p, d2w, 5i, d4j)

(3) duplicate operator to act on current line (dd = delete line, >> = indent line)

(4) ZZ to save & quit, ZQ to quit w/o saving

(5) zt: scroll cursor to top,

zb: bottom, zz: center

(6) gg: top of file (vim only),
gf: open file under cursor (vim only)

For a graphical vi/vim tutorial & more tips, go to www.viemu.com - home of ViEmu, vi/vim emulation for Microsoft Visual Studio

53

Modes & Controls		Inserting Text	Other
Command Mode	ESC (commands preceded by :)	i Insert before cursor	u Undo last change
Insertion Mode	Entered on insertion or change	a Append after cursor	J Join lines
Starting VI (command line)		I Insert before line	nJ Join next n lines
vi <filename>	Edit <i>filename</i>	A Append after line	.
vi -r <filename>	Edit last version of <i>filename</i> after crash	o Add new line after current line	Repeat last command
vi +n <filename>	Edit <i>filename</i> at line <i>n</i>	O Add new line before current line	U Undo all changes to line
vi +<filename>	Edit <i>filename</i> at end of file	r Overwrite one character	:N Open split screen
vi +str <filename>	Edit <i>filename</i> at first occurrence of <i>str</i>	R Overwrite many characters	v Visual mode
In insertion mode the following should be preceded by ESC:		:r <file> Reads <i>file</i> and inserts it after this line	ctrl + c Escape insert mode
:w	Save	p Put after the position or line	
:x	Save & Exit	P Put before the position or line	
:q	Exit if no changes made	C Rewrite the whole line	
:q!	Exit & discard any changes		
Cursor Navigation		Deleting Text	
h or <	Cursor left	x Delete character to right of cursor	
j or ▼	Cursor down	X Delete character to left of cursor	
k or ▲	Cursor up	D Delete the rest of line	
l or ►	Cursor right	dd or :d Delete current line	
w	Next word	ndw Deletes the next n words	
W	Next blank delimited word	ndb Deletes the previous n words	
b	Start of word	ndd Deletes n lines starting with current	
B	Start of blank delimited word	:x,yd Delete lines x through y	
e	End of word	:r <file> Reads <i>file</i> and inserts it after this line	
E	End of blank delimited word	d{nav_cmd} Overwrite many characters	
(Back a sentence	:r <file> Reads <i>file</i> and inserts it after this line	
)	Forward a sentence		
{	Back a paragraph		
}	Forward a paragraph		
0	Beginning of line		
\$	End of the line		
1G	Start of file		
G	End of file		
:n	<i>n</i> th line of file		
f<char>	Forward to <i>char</i>		
F<char>	Back to <i>char</i>		
H	Top of screen		
M	Middle of screen		
L	Bottom of screen		
%	Matching bracket		
gg	Start of document		
Searching		Replacing	
/string		:s/pt/str/flag	Replace pattern with string
?string		Flags	
n		g	Replace all occurrences of pattern
N		c	Confirm replaces
:set ic		&	Repeat last :s command
:set noic			
:set nu			
:x,yg/str			
:g/str/cmd			
*			
#			



Download this Help Sheet now at gosquared.com/liquidicity
Put it on your wall Referenced from <http://www.lagmonster.org/docs/vi.html>

© 2010 Go Squared Ltd.

VIM

Cursor movement

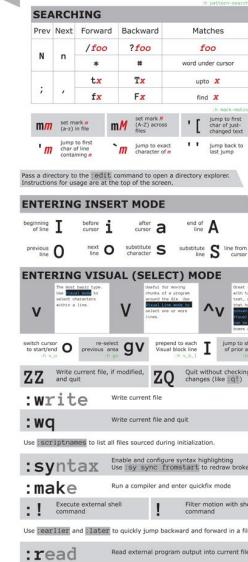
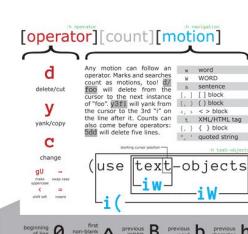
h - move left
 j - move down
 k - move up
 l - move right
 w - jump by start of words (punctuation considered words)
 W - jump by words (spaces separate words)
 e - jump to end of words (punctuation considered words)
 E - jump to end of words (no punctuation)
 b - jump backward by words (punctuation considered words)
 B - jump backward by words (no punctuation)
 0 - (zero) start of line
 ^ - first non-blank character of line
 \$ - end of line
 G - Go To command (prefix with number - 5G goes to line 5)
 Note: Prefix a cursor movement command with a number to repeat it. For example, 4j moves down 4 lines.

Insert Mode - Inserting/Appending text

i - start insert mode at cursor
 I - insert at the beginning of the line
 a - append after the cursor
 A - append at the end of the line
 o - open (append) blank line below current line (no need to press return)
 O - open blank line above current line
 ea - append at end of word
 Esc - exit insert mode

Editing

r - replace a single character (does not use insert mode)
 J - join line below to the current one
 cc - change (replace) an entire line
 cw - change (replace) to the end of word
 c\$ - change (replace) to the end of line
 s - delete character at cursor and substitute text
 S - delete line at cursor and substitute text (same as cc)
 xp - transpose two letters (delete and paste, technically)
 u - undo
 . - repeat last command



Marking text (visual mode)

v - start visual mode, mark lines, then do command (such as y-yanke)
 V - start Linewise visual mode
 o - move to other end of marked area
 Ctrl+v - start visual block mode
 O - move to Other corner of block
 aw - mark a word
 ab - a () block (with braces)
 aB - a {} block (with brackets)
 ib - inner () block
 iB - inner {} block
 Esc - exit visual mode

Visual commands

> - shift right
 < - shift left
 y - yankee (copy) marked text
 d - delete marked text
 ~ - switch case

Cut and Paste

yy - yankee (copy) a line
 2yy - yankee 2 lines
 yw - yankee word
 y\$ - yankee to end of line
 p - put (paste) the clipboard after cursor
 P - put (paste) before cursor
 dd - delete (cut) a line
 dw - delete (cut) the current word
 x - delete (cut) current character

Exiting

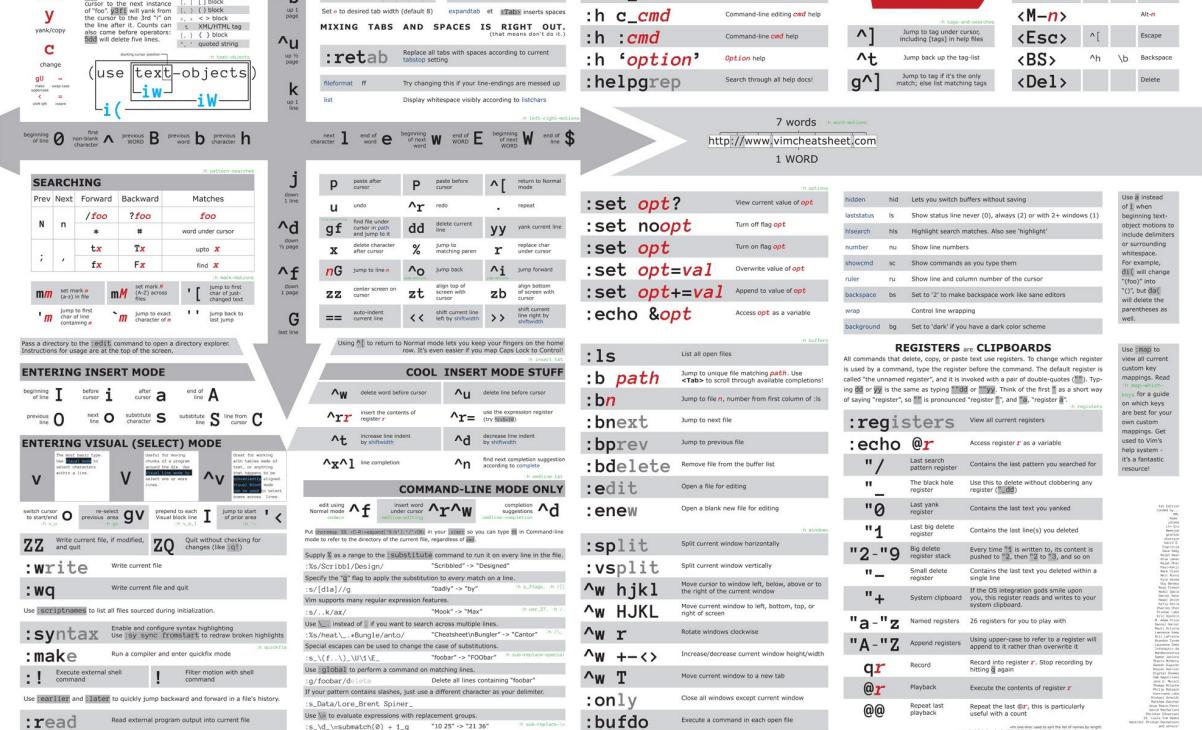
:w - write (save) the file, but don't exit
 :wq - write (save) and quit
 :q - quit (fails if anything has changed)
 :q! - quit and throw away changes

Search/Replace

/pattern - search for pattern
 ?pattern - search backward for pattern
 n - repeat search in same direction
 N - repeat search in opposite direction
 :%s/old/new/g - replace all old with new throughout file
 :%s/old/new/gc - replace all old with new throughout file with confirmations

Working with multiple files

:e filename - Edit a file in a new buffer
 :bnext (or :bn) - go to next buffer
 :bprev (of :bp) - go to previous buffer
 :bd - delete a buffer (close a file)
 :sp filename - Open a file in a new buffer and split window
 ctrl+ws - Split windows
 ctrl+ww - switch between windows
 ctrl+wq - Quit a window
 ctrl+vv - Split windows vertically



Vim Cheat Sheet

Cursor movement	Editing	Cut and paste
h - move cursor left	r - replace a single character	yy - yank (copy) a line
j - move cursor down	J - join line below to the current one	2yy - yank (copy) 2 lines
k - move cursor up	cc - change (replace) entire line	yw - yank (copy) word
l - move cursor right	cw - change (replace) to the end of the word	y\$ - yank (copy) to end of line
w - jump forwards to the start of a word	c\$ - change (replace) to the end of the line	p - put (paste) the clipboard after cursor
W - jump forwards to the start of a word (words can contain punctuation)	s - delete character and substitute text	P - put (paste) before cursor
e - jump forwards to the end of a word	S - delete line and substitute text (same as cc)	dd - delete (cut) a line
E - jump forwards to the end of a word (words can contain punctuation)	xp - transpose two letters (delete and paste)	2dd - delete (cut) 2 lines
b - jump backwards to the start of a word	u - undo	dw - delete (cut) word
B - jump backwards to the start of a word (words can contain punctuation)	Ctrl + r - redo	D - delete (cut) to the end of the line
0 - jump to the start of the line	. - repeat last command	d\$ - delete (cut) to the end of the line
^ - jump to the first non-blank character of the line		x - delete (cut) character

Marking text (visual mode)	Exiting
v - start visual mode, mark lines, then do a command (like y-yank)	:w - write (save) the file, but don't exit

- pdf 版本

vi / vim graphical cheat sheet

Esc normal mode															
~ toggle case	! external filter	@ play macro	# prev ident	\$ col	% goto match	^ "soft" bol	& repeat :s	* next ident	(begin sentence) end sentence	"soft" bol down	+ next line			
` goto mark	1 append	2 subst char	3 delete <small>1,3</small>	4 fwd to char	5 extra ⁶ cmds	6	7	8	9	0 "hard" bol	- prev line	= auto ³ format			
Q ex mode	W next WORD	E end WORD	R replace mode	T back 'till	Y yank line	U undo line	I insert at bol	O open above	P paste before	{ begin parag.	}	end parag.			
q record macro	w next word	e end word	r replace char	t 'till	y yank <small>1,3</small>	u undo	i insert mode	o open below	p paste <small>1</small> after	[misc]	misc			
A append at eol	S subst line	D delete to eol	F "back" fwd	G soft goto ln	H screen top	J join lines	K help	L screen bottom	: ex cmd line	!! reg spec	! bol/ goto col				
a append	s subst char	d delete <small>1,3</small>	f fwd to char	g extra ⁶ cmds	h ←	j ↓	k ↑	l →	;	repeat <small>1, T/U/F</small>	\ not used				
Z extra ⁴ quit	X back-space	C change to eol	V visual lines	B prev WORD	N prev (find)	M screen mid'l	< un ³ indent	> indent <small>3</small>	? find rev.						
Z extra ⁵ cmds	X delete char	C change <small>1,3</small>	V visual mode	b prev word	n next (find)	m set mark	,	, reverse <small>1, T/U/F</small>	/ find						
motion moves the cursor, or defines the range for an operator															
command direct action command, if red, it enters insert mode requires a motion afterwards, operates between cursor & destination															
operator special functions, requires extra input commands with a dot need a char argument afterwards															
extra Q . beginning of line, eol = end of line mk = mark, yank = copy words: res : base v. offset WORDs: res : base v. offset															
Main command line commands ('ex'): :w (save), :q (quit), :q! (quit w/o saving) :e f (open file f) :%s/x/y/g (replace 'x' by 'y' filewide), :h (help in vim), :new (new file in vim),															
Other important commands: CTRL-R: redo (vim), CTRL-F/B: page up/down, CTRL-E/Y: scroll line up/down, CTRL-V: block-visual mode (vim only)															
Visual mode: Move around and type operator to act on selected region (vim only)															
Notes: (1) use "x before a yank/paste/del command to use that register ('clipboard') (x=a, z, *) (e.g., "ay8 to copy rest of line to reg 'a') (2) type in a number before any action to repeat it that number of times (e.g.: 2p, d2w, 5i, d4j) (3) duplicate operator to act on current line (dd = delete line, >> = indent line) (4) ZZ to save & quit, ZQ to quit w/o saving (5) zt: scroll cursor to top, zb: bottom, zz: center (6) gg: top of file (vim only), gf: open file under cursor (vim only)															

Based on SVG version at http://www.viemu.com/a_vi_vim_graphical_cheat_sheet_tutorial.html

9.4 其他问题

9.4.1 中文编码

- `LANG=zh_CN.utf-8`
- `export LANG`

9.4.2 unix2dos 和 dos2unix

- Windows 下，文本文件的换行符为 `CR + LF`，使用 vi 编辑器可以看到 `^M` 字样
- Unix 下，文本文件的换行符仅为 `LF`
- 两者之间的转换，可以使用 `dos2unix` 或 `unix2dos` 命令
- 安装
 - `apt-get install tofrodos`
 - `cd /usr/bin`
 - `ln -s fromdos dos2unix`
 - `ln -s todos unix2dos`

9.4.3 编码转换命令

- `iconv --list` 列表显示所有可用的编码方式
- `iconv -f old_encoding -t new_encoding filename [-o newfile]`
 - `-f` : from, 原有编码方式
 - `-t` : to, 目标编码方式
- 示例
 - `iconv -f gbk -t utf8 testfile -o newfile`

9.5 其他示例

- 如下，将其中的 (1), (2), (3) ... (100)，分别替换为 (2), (3), ... (101)

(1) first line
(2) second line
...
(100) 100th line

 - 解决方案
 - `:%s/(\(\d+\))/\="(.submatch(1)+1)."/g`
- 有一个 log 系统对于输出行长度有限制，因此在输出很长的 log 时需要断行，在断行时以单行的 `-$-` 标记，现在的
需求是把这些行连起来，在 vim 中也可以很方便地做到
 - `:%s/\n-\$-\n//g`
 - `:g/-\$-/norm ddkJx`
- 连续插入 36 个等号
 - 按 `ESC` 进入命令模式，输入 `36i` 再按 `ESC` 即可
- 在多行开始插入 `//`
 - 移动光标到需要注释掉的第一行开头，然后
 - 按 `Ctrl-v` (如果开启了 vim 的 mswin 行为，则 `Ctrl-v` 表示粘贴，这时需要用 `Ctrl-q` 代替) 进入 Visual
blockwise 模式，这个模式是 Visual mode 的一种，可以进行块选择。然后按 `j` 选择上所有需要注释行的行
首 (看起来效果是选择了第一列)，输入 `I//` 再按 `ESC` 就可以在每一行开头插入 `//`

10 认识与学习 BASH

10.1 认识 BASH

10.1.1 操作系统与 Shell

- 操作系统是一组软件，控制整个硬件与管理系统的活动监测，合适地使用这些功能，才能发挥计算机的功能
- Shell 实质是一个应用程序，如同包裹着操作系统能力的一层外壳，可以接收用户发出的命令，并调用系统核心能力来完成所需工作

10.1.2 Shell 简史

- Bourne shell: Steven Bourne 开发，第一个 Shell，可简称为 sh
- C shell: 伯克利大学的 Bill Joy 基于 BSD Unix 的 shell 开发，语法类似 C 语言，也称为 csh，随着 SUN 公司的主机销售，流传甚广
- ash:
- zsh:

10.1.3 /etc/shells 文件

- 标明哪些 shell 是系统的合法 shell
- bash 是目前 linux 下最流行的 shell，bash 是 Bourne Again SHell 的简称

10.1.4 bash 的功能

- 兼容 sh，并进行了增强
- 命令行历史编辑功能
 - 使用上下键查看过去键入的命令
 - 这些命令存储在 `~/.bash_history` 文件中
- 命令与文件补全
 - 使用 Tab 键补全
 - 可以补全命令或者文件名
 - 默认安装 `bash-completion`，甚至可以进行选项/参数的补全
- alias: alias `ll='ls -al'` 给命令起别名，取消时用 `unalias ll`
- 任务控制 - job control
 - `jobs`
 - `fg`
 - `bg`
 - `ctrl+z`
 - `ctrl+c`
 - `kill %1`
- 脚本编程 - 下一章讲解，非常重要的功能
- 通配符 - wildcard

10.2 查询指令是否为 bash 的内建命令

- `type` 命令
- 选项与参数
 - 不加任何选项与参数时，`type` 会显示出 name 是外部指令还是 bash 内建指令
 - `-t`：当加入 `-t` 参数时，`type` 会将 name 以下这些字眼显示出他的意义：
 - file：表示为外部指令
 - alias：表示该指令为命令别名所设定的名称
 - builtin：表示该指令为 bash 内建的指令功能
 - `-p`：如果后面接的 name 为外部指令时，才会显示完整文件名

- `-a` : 会将 PATH 变量定义的路径中，所有含 name 的指令都列出来，包含 alias

10.3 命令下达和简单的命令行编辑

- 输入命令后，回车执行
- 如果命令行长度太长，需要折行，为清晰起见，可以使用 `\` 续行符，将一行输入不了的内容，在下一行继续输入，最后回车执行即可
- 非常有用的命令行编辑快捷键
 - `ctrl+u` 从光标处向前删除指令字符串
 - `ctrl+k` 从光标处向后删除指令字符串
 - `ctrl+a` 光标移动到当前行首
 - `ctrl+e` 光标移动到当前行尾

10.4 Shell 的变量功能

10.4.1 什么是变量

- 变化的量，一般以一组文字或符号等，来代表一些设定或者一串保留的数据

10.4.2 变量的设定与使用

- 变量的使用
 - `echo $variable` 显示变量 `variable` 的值
- 设定变量及规则
 - `variable=value` 注意等号两边没有空格
 - 变量名称只能是英文字母与数字，但不能以数字开头
 - 含有空格的变量，可以用双引号或单引号括起来
 - 双引号时，不影响特殊内容的解释，如
 - `var="lang is $LANG"` , `echo $var` 时，显示 `lang is zh_CN.UTF-8`
 - 单引号时，所包括内容均当做普通文本处理，如
 - `var='lang is $LANG'` , `echo $var` 时，显示 `lang is $LANG`
 - `\` 可以将特殊字符，如 `ENTER`，`$`，`\`，`_`，`!` 等，转义为普通字符
 - 反单引号或 `$(())`，可以完成动态给变量赋值的功能，如
 - `ver=$(uname -r)` 或者 `ver='uname -a'`
 - 然后 `echo $var`，输出 `3.10.0-229.el7.x86_64` 类似的信息
 - 两者区别
 - ‘输入简便，但不支持命令嵌套’
 - `$(())` 稍微麻烦，但支持命令的嵌套
- 如果变量需要在其他子程序执行，需要使用 `export` 将其输出到当前环境中，`export $var`
 - 仔细学习书中示例
- 通常以大写字母表示系统默认变量，小写字母表示自定义变量
- 取消变量的方法为使用 `unset`，如 `unset var`

10.4.3 环境变量

10.4.3.1 查看环境变量

- `env` 查看目前 shell 下的所有环境变量，常见环境变量有
 - `HOME` 用户的家目录。使用 `cd ~` 或者 `cd` 回到用户家目录时，就是取用该变量，很多程序都可能会取用到这个变量的值
 - `SHELL` 目前环境使用的 SHELL, Linux 默认使用 `/bin/bash`
 - `HISTSIZE` 与历史命令有关，系统存储历史命令的条数
 - `MAIL` 使用 `mail` 指令收信时，系统会去读取邮件信箱文件 (mailbox)

- `PATH` 执行文件的搜索路径，目录与目录中间以冒号 (:) 分隔
- `LANG` 系统的语言设定
- `RANDOM` 随机数生成器，也即 `/dev/random` 文件，每次 `echo $RANDOM` 均可获得一个 0~32767 之间的随机变量
- `set` 查看所有变量（环境变量和自定义变量），比如
 - `PS1` `PS` 表示 `Prompt String`，`PS1` 即为命令提示符，可以使用 `man bash` 查看先关说明
 - `\d` : 显示“星期月日”的日期格式，如：“Mon Feb 2”
 - `\H` : 完整的主机名
 - `\h` : 取主机名在第一个小数点之前的名字
 - `\t` : 显示时间，为 24 小时格式的“HH:MM:SS”
 - `\T` : 显示时间，为 12 小时格式的“HH:MM:SS”
 - `\A` : 显示时间，为 24 小时格式的“HH:MM”
 - `\@` : 显示时间，为 12 小时格式的“am/pm”样式
 - `\u` : 目前使用者的账号名
 - `\v` : BASH 的版本信息，如鸟哥的测试主机版本为 4.2.46(1)-release，仅取“4.2”显示
 - `\w` : 完整的工作目录名称，由根目录写起的目录名称，但家目录会以 `~` 取代
 - `\W` : 利用 basename 函数取得工作目录名称，所以仅会列出最后一个目录名
 - `\#` : 下达的第几个指令
 - `\$` : 提示字符，如果是 root 用户，提示字符为 `#`，否则就是 `$`
 - `$` 当前 shell 线程的 pid，使用 `echo $$` 显示
 - `?` 上个执行命令的返回值，使用 `echo $?` 显示，命令执行成功时，一般返回 `0`，否则为非 `0`
- `export` 自定义变量转成环境变量
 - 环境变量和自定义变量的区别：该变量是否会被子进程使用。
 - 子进程仅会继承父进程的环境变量，不会继承自定义变量
 - 如果希望某个变量被子进程使用，使用 `export var_name` 将其导出即可（其实质是讲自定义变量写入到环境变量区）
 - 使用 `export` 可查看当前所有的导出变量
 - 简单理解（不准确，类比）
 - 环境变量 = 全局变量
 - 自定义变量 = 局部变量

10.5 locale

- `locale -a` 查看当前系统支持的所有语言种类
- `locale` 查看当前使用的语言种类
- 不同语言的具体支撑文件，存放在 `/usr/lib/locale` 目录
- 系统的语言配置文件，参见 `/etc/locale.conf`
- 简单起见，仅设置 `LANG` 和 `LC_ALL` 即可，记得使用 `export` 命令予以导出

10.5.1 交互读入变量数组和声明

- `read` shell 内置命令，从标准输入读内容到变量
 - `read [-pt] variable`
 - `-p` : 后面跟提示字符串
 - `-t` : 后面跟等待的秒数
- `typeset/declare`
 - 用于声明变量，均为 shell 保留字
 - `declare [-aixr] variable`
 - `-a` : 将后面的 variable 定义为数组类型
 - `-i` : 将后面的 variable 定义为整型数字类型

- `-x` : 同 `export`, 将后面的 variable 变为环境变量
- `-r` : 变量设定为只读类型, 无法修改, 也无法 `unset`
- 示例:
 - `declare -i sum=100+300+50`
 - `echo $sum` 输出 450
 - `declare -x sum` 将其转为环境变量
 - `declare +x sum` 取消 sum 的环境变量类型
 - 注意: 默认声明变量时, 类型为字符串
- 数组变量类型, 暂略
 - `var[1] = "small min"`
 - `var[2] = "big min"`
 - `var[3] = "nice min"`
 - `echo "${var[1]}, ${var[2]}, ${var[3]}"`

10.5.2 ulimit 限制用户使用的系统资源

- `ulimit [-SHacdfltu] [配额]`
- 选项与参数:
 - `-H` : hard limit, 严格设定, 不能超过这个设定的数值
 - `-S` : soft limit, 警告设定, 可以超过, 但是若超过则有警告, 比如说: soft 可设定为 80 而 hard 设定为 100, 那么你可以使用到 90 (因为没有超过 100) 但介于 80~100 之间时, 系统会有警告
 - `-a` : 后面不接任何选项与参数, 可列出所有的限制额度
 - `-c` : 当某些程序发生错误时, 系统可能会将该程序在内存中的信息写成文件 (除错用), 这种文件就被称为核心文件 (core file, 写文件动作叫做 core dumped), 此为限制每个核心文件的最大容量
 - `-f` : 此 shell 可以建立的最大文件容量 (一般可能设定为 2GB) 单位为 Kbytes
 - `-d` : 程序可使用的最大断裂内存 (segment) 容量
 - `-l` : 可用于锁定 (lock) 的内存量
 - `-t` : 可使用的最大 CPU 时间 (单位为秒)
 - `-u` : 单一用户可以使用的最大进程 (process) 数量
- 示例
 - `ulimit -f 10240`
 - `ulimit -a | grep 'file size'`
 - `dd if=/dev/zero of=123 bs=1M count=20`

10.5.3 变量内容的删除、取代与替换

替换方式	含义
<code> \${VAR}</code>	基本替换, 花括号限定变量名的开始和结束
条件设定部分	
<code> \${VAR-WORD}</code>	若 <code>VAR</code> 未设定, 则返回 <code>WORD</code> 的值。不改变变量的值
<code> \${VAR:-WORD}</code>	若 <code>VAR</code> 未设定或为空值, 则返回 <code>WORD</code> 的值; 否则返回 <code>VAR</code> 的值。不改变变量的值
<code> \${VAR=WORD}</code>	若 <code>VAR</code> 未设定, 则返回 <code>WORD</code> 的值, 同时将 <code>WORD</code> 的值赋给 <code>VAR</code>
<code> \${VAR:=WORD}</code>	若 <code>VAR</code> 未设定或为空值, 则返回 <code>WORD</code> 的值, 同时将 <code>WORD</code> 的值赋给 <code>VAR</code>
<code> \${VAR+WORD}</code>	若 <code>VAR</code> 已设定, 则返回 <code>WORD</code> ; 否则返回空。不改变变量的值
<code> \${VAR:+WORD}</code>	若 <code>VAR</code> 为非空值, 则返回 <code>WORD</code> ; 否则返回空。不改变变量的值

替换方式	含义
<code> \${VAR?MESSAGE}</code>	若 <code>VAR</code> 未设定，则将 <code>MESSAGE</code> 的值输出到标准错误和标准输出，同时 shell 也显示出 <code>VAR</code> 的名字；否则返回 <code>VAR</code> 的值
<code> \${VAR:?MESSAGE}</code>	若 <code>VAR</code> 未设定或为空值，则将 <code>MESSAGE</code> 的值输出到标准错误和标准输出，同时 shell 也显示出 <code>VAR</code> 的名字；否则返回 <code>VAR</code> 的值
删除和替换部分	
<code> \${VAR##WORD}</code>	返回删除掉 <code>WORD</code> 的最长匹配之后的字符串，从左到右
<code> \${VAR%WORD}</code>	返回删除掉 <code>WORD</code> 的最短匹配之后的字符串，从右到左
<code> \${VAR%%WORD}</code>	返回删除掉 <code>WORD</code> 的最长匹配之后的字符串，从右到左
<code> \${VAR:offset:length}</code>	返回从第 <code>offset</code> 个字符开始的长度为 <code>length</code> 的子字符串
<code> \${VAR/s/t}</code>	将第一个 <code>s</code> 的最长匹配替换为 <code>t</code> 后返回
<code> \${VAR//s/t}</code>	将所有 <code>s</code> 的最长匹配替换为 <code>t</code> 后返回
<code> \${VAR#WORD}</code>	返回删除掉 <code>WORD</code> 的最短匹配之后的字符串，从左到右
其他	
<code> \${#VAR}</code>	返回 <code>VAR</code> 的长度。如果 <code>VAR</code> 是 <code>*</code> 或 <code>@</code> ，则返回 <code>\$@</code> 中元素的个数

- 示例 1(删除)

- `path=${PATH}`
- `echo $path`
 - 输出 `/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/home/dmtsai/bin`
- `echo ${path#//*local/bin:}`
 - 输出 `/usr/bin:/usr/local/sbin:/usr/sbin:/home/dmtsai/bin`
 - 也即 `/usr/local/bin: /usr/bin:/usr/local/sbin:/usr/sbin:/home/dmtsai/bin`

- 示例 2(删除)

- `echo ${path#/*:}`
 - 输出 `/usr/bin:/usr/local/sbin:/usr/sbin:/home/dmtsai/bin`
 - 也即 `/usr/local/bin: /usr/bin:/usr/local/sbin:/usr/sbin:/home/dmtsai/bin`
- `echo ${path##/*:}`
 - 输出 `/home/dmtsai/bin`
 - 也即 `/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin: /home/dmtsai/bin`

- 示例 3(删除)

- `echo ${path%:*bin}`
 - 输出 `/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/home/dmtsai/.local/bin`
 - 也即 `/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/home/dmtsai/.local/bin :/home/dmtsai/bin`
- `echo ${path%::*bin}`
 - 输出 `/usr/local/bin`
 - 也即 `/usr/local/bin :/usr/bin:/usr/local/sbin:/usr/sbin:/home/dmtsai/bin`

- 示例 4(删除)

- `MAIL=/var/spool/mail/young`
- `echo $MAIL` 输出 `/var/spool/mail/young`
- `echo ${MAIL##/*/}`
 - 输出 `young`，从左边最长匹配删除，结果是只保留用户名
- `echo ${MAIL%/*}`
 - 输出 `/var/spool/mail`，从右边最短匹配删除，结果是保留路径名

- 示例 5(替换)
 - `echo ${path/sbin/SBIN}` , 将 \$path 中的 第一个 `sbin` , 替换成 `SBIN`
 - `echo ${path//sbin/SBIN}` , 将 \$path 中的 所有 `sbin` , 都替换成 `SBIN`
- 变量的测试与内容替换
 - 有时, 需要基于某种 判断 比如说某个变量是否存在或者被设定, 来决定不同的动作 (设定不同的变量值)
 - 如下表所示

变量设定方式	str 没有设定	str 为空字符串	str 已设定非为空字符串
<code>var=\${str-expr}</code>	<code>var=expr</code>	<code>var=</code>	<code>var= \$str</code>
<code>var=\${str:-expr}</code>	<code>var=expr</code>	<code>var=expr</code>	<code>var= \$str</code>
<code>var=\${str+expr}</code>	<code>var=</code>	<code>var=expr</code>	<code>var=expr</code>
<code>var=\${str:+expr}</code>	<code>var=</code>	<code>var=</code>	<code>var=expr</code>
<code>var=\${str:expr}</code>	<code>str=expr var=expr</code>	<code>str 不变 var=</code>	<code>str 不变 var= \$str</code>
<code>var=\${str:=expr}</code>	<code>str=expr var=expr</code>	<code>str=expr var=expr</code>	<code>str 不变 var= \$str</code>
<code>var=\${str?expr}</code>	<code>expr -> stderr</code>	<code>var=</code>	<code>var= \$str</code>
<code>var=\${str:?expr}</code>	<code>expr -> stderr</code>	<code>expr -> stderr</code>	<code>var= \$str</code>

- 示例 1, 检查是否存在某个变量
 - `unset username`
 - `echo ${username}` 输出空白, 因为 `username` 没有设定
 - `username=${username-root}`
 - `echo ${username}` 输出 `root` , 因为 `username` 没有设定
- 示例 2, 检查空字符串
 - `username=""` 设定为空字符串
 - `echo ${username}` 输出空白, 因为 `username` 为空字符串
 - `username=${username-root}`
 - `echo ${username}` 输出空白, 因为 `username` 为空字符串
- 示例 3, 使用 `=` 号, 会更改原变量的内容
 - `unset str; var=${str=newvar}`
 - `echo "var=${var}, str=${str}"`
 - 输出 `var=newvar, str=newvar` 因为 str 不存在
- 示例 4, 使用 `=` 号, 继续
 - `str="oldvar"; var=${str=newvar}`
 - `echo "var=${var}, str=${str}"`
 - 输出 `var=oldvar, str=oldvar` 因为 str 存在
- 示例 5, 使用 `?` 做条件判断
 - `unset str; var=${str?无此变量}`
 - 运行结果 `bash: str: 无此变量`
 - `str="oldvar"; var=${str?novar}`
 - `echo "var=${var}, str=${str}"`
 - 输出 `var=oldvar, str=oldvar` 因为 str 存在
- 助记
 - `#` 是去掉左边 (键盘上 `#` 在 `$` 的左边)
 - `%` 是去掉右边 (键盘上 `%` 在 `$` 的右边)
 - 单一符号是最短匹配
 - 两个符号是最长匹配
 - 带冒号是检查未设定和空值, 不带冒号只检查未设定
 - 以上内容参见 `man bash` 的 EXPANSION 小节
- 以上的各种操作, 也可以使用 shell 编程中的条件控制语句 `if...then...` 来完成

10.6 命令别名与历史命令

- alias, unalias
 - `alias` 查看当前别名设定
 - `alias ll='ls -l'` 设定一个别名
 - `unalias ll` 取消一个别名设定
- history
 - `history` 查看历史命令
 - `alias h='history'` 一般会如此设定，减少输入字符数
 - `history [n]` 显示最近输入的 n 条命令
 - `history -c` 清空最近的全部 history 内容
 - `history [-raw] histfiles`
 - `-a` : 将目前新增的 history 指令添加到 histfiles 中, 若没有加 histfiles, 则默认写入 `~/.bash_history`
 - `-r` : 将 histfiles 的内容读到目前这个 shell 的 history 记忆中
 - `-w` : 将目前的 history 记忆内容写入 histfiles 中, 如果没有加 histfiles, 则默认写入 `~/.bash_history`
 - 历史命令的记忆条数, 由环境变量 `HISTFILESIZE` 指定
 - 历史命令的执行
 - 使用 `!number`, 可以执行编号为 number 的命令
 - 使用 `!command`, 可以向前搜索命令前缀为 command 的命令并执行
 - 使用 `!!`, 可以重复执行刚刚执行过的命令
 - 多个登录时, 只有最后注销的那个 bash 的历史命令, 会写入 `~/.bash_history` 文件
 - 考虑如何为历史命令添加时间戳功能

10.7 Bash shell 的操作系统

10.7.1 路径与命令搜索顺序

- 命令的执行顺序
 - 以相对/绝对路径执行命令, 如 `/bin/ls` 或 `./test.sh`
 - 由 alias 找到该指令来执行
 - 由 bash 内建的命令来执行
 - 在 \$PATH 路径列表中依次寻找匹配的命令

10.7.2 /etc/issue, /etc/motd

- issue, 本地登录提示信息, 文件中的代码意义如下 `\d` : 本地时间的日期 `\l` : 显示第几个终端机接口 `\m` : 显示硬件的等级 (`i386/i486/i586/i686...`) `\n` : 显示主机的网络名称 `\o` : 显示 domain name `\r` : 操作系统的版本 (相当于 `uname -r`) `\t` : 显示本地时间 `\s` : 操作系统的名称 * `\v` : 操作系统的版本
- issue.net 远程登录提示信息
- /etc/motd 其内部的信息也可以被登录者看到 (Message Of ToDay)
- 建议试着修改这些文件内容, 查看效果

10.7.3 bash 的环境配置文件

- login 与 non-login shell
 - login shell: 启动 bash 前需要完整登录流程的
 - 读取的配置文件有
 - `/etc/profile` 全局配置文件
 - `~/.bash_profile` 或 `~/.bash_login` 或 `~/.profile` 个人设定文件

- non-login shell: 不使用登录方式运行的 shell 窗口, 比如说登录后, 启动一个 bash, 或者在一个 bash 中, 启动一个 bash 子 shell
- `/etc/profile` 说明
 - 内置的变量设定, 一般包括
 - `PATH` : 命令搜索路径
 - `MAIL` : 用户的 mailbox 目录, 一般为 `/var/spool/mail/账号名`
 - `USER` : 存储用户名
 - `HOSTNAME` : 根据主机的 `hostname` 命令决定
 - `HISTSIZE` : 历史命令记录条数
 - `umask` : root 默认为 022, 而一般用户为 002
 - `/etc/profile.d/*.sh`
 - 注意后缀为 `.sh` 的才会被使用
 - 分成多个文件存储不同设置, 目的是更清晰和便于维护
 - `/etc/locale.conf` 由 `/etc/profile.d/lang.sh` 引用, 所包含的重要变量为 `LANG/LC_ALL`
 - `/usr/share/bash-completion/completions/*`
 - 增强 bash 命令的参数/选项补足功能
 - 由 `/etc/profile.d/bash_completion.sh` 引用
- `~/.bash_profile` bash 读取 `/etc/profile` 后, 会按照以下顺序读取个人偏好配置文件, 注意: 如果多个文件存在时, bash 只会读取其中之一
 1. `~/.bash_prifile`
 2. `~/.bash_login`
 3. `~/.profile`
- `source` 读入环境配置文件的指令
 - `source` 是 shell 内置命令
 - `source` 命令可以使用符号 `.` 来替代
 - `source` 可以将配置文件的内容读入到 `当前 shell` 环境 中
- `~/.bashrc` (non-login shell 会读取)
 - non-login shell 仅会读取该配置文件
 - `/etc/bashrc`
- 其他相关配置文件
 - `/etc/man_db.conf` 定义了 man page 的查找路径
 - `~/.bash_history` 存储历史命令
 - `~/.bash_logout` 定义了当注销 bash 时, 系统会做哪些后续
- 以上相关内容, 可参见 `/usr/share/doc/bash/examples/startup-files`
 - 如果没有该文件, 使用 `apt-get install bash-doc` 安装

10.7.4 终端设定 stty, set

- `stty [-a]` 查看终端设置 (setting tty)
 - `-a` : 列出所有的终端参数
 - 输出示例

```

speed 38400 baud; rows 26; columns 93; line = 0;
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D; eol = M-^?; eol2 = M-^?;
swtch = <undef>; start = ^Q; stop = ^S; susp = ^Z; rprnt = ^R; werase = ^W; lnext = ^V;
discard = ^O; min = 1; time = 0;
)parenb -parodd -cmspar cs8 -hupcl -cstopb cread -clocal -crtsets
-ignbrk -brkint -ignpar -parmrk -inpck -istrip -inlcr -ignocr icrnl ixon -ixoff -iuclc ixany
imaxbel -iutf8
opost -olcuc -ocrnl onlcr -onocr -onlret -ofill -ofdel nl0 cr0 tab0 bs0 vt0 ff0
isig icanon iexten echo echoe echok -echonl -noflsh -xcase -tostop -echoprt echoctl echoke
-flusho -extproc

```

- 重要内容
 - `intr` : 送出 interrupt 信号给当前运行的程序
 - `quit` : 送出一个 quit 信号给当前运行的程序
 - `erase` : 向后删除字符
 - `kill` : 删除在目前命令行上的所有文字
 - `eof` : End Of File, 表示结束输入
 - `start` : 某个程序停止后, 重新启动它的 output
 - `stop` : 停止目前屏幕的输出
 - `susp` : 发送 terminal stop 信号给当前运行的程序
- 重新设置
 - `stty erase ^h` 设置使用 `ctrl+h` 来进行字符删除
- set 命令进行各种设置
 - `set [-uvCHhmBx]`
 - `-u` : 默认不启用。若启用后, 当使用未设定变量时, 会显示错误信息
 - `-v` : 默认不启用。若启用后, 在信息被输出前, 会先显示信息的原始内容
 - `-x` : 默认不启用。若启用后, 在指令被执行前, 会显示指令内容 (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。与中括号 [] 的作用有关
 - `-C` : 默认不启用。若使用 > 重定向等, 则若文件存在时, 该文件不会被覆盖
 - 示例
 - `echo $-` 显示目前的默认设定, 可能输出 `himBH`
 - `set -u; echo $notexist` 输出 `bash: notexist: unbound variable`
 - `set -x` 经常用来调试脚本 ##### 通配符与特殊符号

符号	意义
*	代表”0个到无穷多个”任意字符
?	代表“一定有一个”任意字符
[]	代表“任意一个在括号内”的字符。例如 [abcd] 代表”a, b, c, d 中任意一个字符”
[-]	减号在中括号内, 代表“在编码顺序内的所有字符”。例如 [0-9] 代表 0 到 9 之间的所有数字
[^]	若中括号内的第一个字符为指数符号 (^), 表示“反向选择”, 例如 [^abc] 代表一定有一个字符, 只要不是 a, b, c 即可
#	批注符号: 这个最常被使用在 script 当中, 视为说明在其后的数据均不执行
\	转义符号: 将“特殊字符或通配符”还原成一般字符
	管道 (pipe) 符号: 分隔两个管线命令的界定 (后两节介绍)
;	命令行上多个命令时的分隔符
~	用户的家目录
\$	取用变量前导符
&	工作控制 (job control), 将命令转为后台工作
!	逻辑运算意义上的“非”
/	路径分隔符
>, >>	数据流重定向: 输出重定向, 分别是“覆盖”与“追加”
<, <<	数据流重定向: 输入重定向
”	单引号, 不具有变量替换的功能
””	双引号, 具有变量替换的功能

‘ | 两个” ” 中间为可以先执行的指令, 也可使用 \$(() () | 在中间为子 shell 的起始与结束 {} | 在中间为命令区块的组合

10.8 重定向功能

- standard input: stdin, 文件句柄为 0, 使用 < 或者 <<
- standard output: stdout, 文件句柄为 1, 使用 > 或者 >>
- standard error output: stderr, 文件句柄为 2, 使用 2> 或者 2>>
 - 1>: 以覆盖的方法将“正确的数据”输出到指定的文件或设备上

- 1>>: 以追加的方法将“正确的数据”输出到指定的文件或设备上
- 2> : 以覆盖的方法将“错误的数据”输出到指定的文件或设备上
- 2>>: 以追加的方法将“错误的数据”输出到指定的文件或设备上
- /dev/null
 - `/dev/null` 可以理解为系统的黑洞，任何输出到这里的信息，都会被完全忽略
 - 示例
 - `find /home -name .bashrc 2> /dev/null`
 - 两个数据流的输出写入到同一个文件
 - `find /home -name .bashrc > list 2> list` 能写入，但是顺序不保证正确
 - `find /home -name .bashrc > list 2>&1` 正确
 - `find /home -name .bashrc &> list` 正确
- 命令执行方面
 - 使用 分号 分割，可以在一个命令行上输入多个命令，顺序执行
 - 条件执行命令
 - `cmd1 && cmd2`，当 cmd1 执行成功 (`$? = 0`)，才会执行 cmd2
 - `cmd1 || cmd2`，当 cmd1 执行错误 (`$? != 0`)，才会执行 cmd2

10.9 管道功能

- | , 管道符号
- cmd1 | cmd2 , 将 cmd1 的 stdout 作为 cmd2 的 stdin
- 管道功能不理会前一个命令的 stderr 输出

10.9.1 cut grep

- cut 按指定分隔符切割文本
 - 选项与参数:
 - `-d` : 后面接分隔字符，常与 `-f` 一起使用
 - `-f` : 依据 -d 的分隔字符将一段信息分为数段，用 -f 取出指定段
 - `-c` : 以字符 (characters) 的单位取出特定字符区间
 - 示例
 - `echo ${PATH} | cut -d ':' -f 5` 取出第五个路径
 - `echo ${PATH} | cut -d ':' -f 3,5` 取出第三个和第五个路径
 - `export | cut -c 12-` 取前 12 列之后的所有字符串
 - `last | cut -d ' ' -f 1`
- grep 查找特定字符串，非常强大，支持使用正则表达式查找
 - 选项与参数:
 - `-a` : 以 text 方式搜寻二进制文件中的数据
 - `-c` : 计算找到‘搜索字符串’的次数
 - `-i` : 忽略大小写
 - `-n` : 输出行号
 - `-v` : 反向选择，即显示没有‘搜寻字符串’内容的那一行
 - `--color=auto` : 将找到的关键词部分加上颜色显示
 - 示例
 - `last | grep 'root'`
 - `last | grep -v 'root'`
 - `last | grep 'root' | cut -d ' ' -f1`
 - `grep --color=auto 'MANPATH' /etc/man_db.conf`

10.9.2 sort wc uniq

- sort 排序

- `sort [-fbMnrtuk] [file or stdin]`
- 选项与参数:
 - `-f` : 忽略大小写
 - `-b` : 忽略前导空格符
 - `-M` : 以月份的名字来排序
 - `-n` : 作为“数字”进行排序 (默认是字符串排序)
 - `-r` : 反向排序
 - `-u` : 去重, 即 `uniq`, 相同的数据, 仅保留一行
 - `-t` : 设定分隔符, 默认分隔符是 `[tab]` 键
 - `-k` : 以指定区间 (field) 来进行排序
- 示例
 - `cat /etc/passwd | sort`
 - `cat /etc/passwd | sort -t ':' -k 3`
 - `last | cut -d ' ' -f1 | sort`
- `uniq` 去重
 - `uniq [-ic]`
 - 选项与参数:
 - `-i` : 忽略大小写
 - `-c` : 进行计数
 - 示例
 - `last | cut -d ' ' -f1 | sort | uniq`
 - `last | cut -d ' ' -f1 | sort | uniq -c`
- `wc` 字符及行数统计
 - `wc [-lwm]`
 - 选项与参数:
 - `-l` : 仅显示行数
 - `-w` : 仅显示多少字 (英文单字)
 - `-m` : 仅显示字符数
 - 示例
 - `cat /etc/man_db.conf | wc`
 - `last | grep [a-zA-Z] | grep -v 'wtmp' | grep -v \'reboot\' | grep -v 'unknown' | wc -l`

10.9.3 tee

- 功能是将数据流同时发送到文件和屏幕 (`stdout`)
- `tee [-a] file`
- 选项与参数:
 - `-a` : 以追加 (append) 的方式, 将数据写入 file
- 示例
 - `last | tee last.list | cut -d " " -f1`
 - `ls -l /home | tee ~/homefile | more`
 - `ls -l / | tee -a ~/homefile | more`

10.9.4 tr col join paste expand

- `tr` 字符替换或删除
 - `tr [-ds] SET1 ...`
 - 选项与参数:
 - `-d` : 删除所处理数据流中的 SET1 字符串
 - `-s` : 字符去重
 - 示例
 - `last | tr '[a-z]' '[A-Z]' 小写转大写`

- `cat /etc/passwd | tr -d ':'` 删除 : s 字符
 - `cp /etc/passwd ~/passwd && unix2dos ~/passwd`
 - `file /etc/passwd ~/passwd`
 - `cat ~/passwd | tr -d '\r' > ~/passwd.linux` 删除回车、换行字符序列中的回车
- col 用户过滤控制字符，常用来将 tab 替换为对等的空格
 - `col [-xb]`
 - 选项与参数:
 - `-x` : 将 tab 键转换成对等的空格键
 - `-b` : 过滤掉所有的控制字符，包括 RLF 和 HRLF
 - 示例
 - `man man | col -b > man_help` 过滤掉所有控制字符
 - `cat /etc/man_db.conf | col -x | cat -A | more`
 - tab 展开为空格，并用 `cat -A` 确认操作成功
- join 两个文件中，基于相同数据，以行的方式拼接两个文件
 - 处理方式: 比对“第一个字段”的数据，如果两个文件相同，则将两笔数据拼接成一行，且第一个字段放在第一个位置
 - `join [-ti12] file1 file2`
 - 选项与参数:
 - `-t` : 设定分隔字符，默认以空格符分隔数据
 - `-i` : 忽略大小写
 - `-1` : 数字 1，表示“以第一个文件中的字段作为基准来分析”
 - `-2` : 数字 2，表示“以第二个文件中的字段作为基准来分析”
 - 示例
 - `head -n 3 /etc/passwd /etc/shadow`
 - `join -t ':' /etc/passwd /etc/shadow | head -n 3`
 - `head -n 3 /etc/passwd /etc/group`
 - `join -t ':' -1 4 /etc/passwd -2 3 /etc/group | head -n 3`
- paste 行对行的方式，拼接两个文件，连接处以 tab 分隔
 - `paste [-d] file1 file2`
 - 选项与参数:
 - `-d` : 设定分隔字符。默认分隔符是 [tab]
 - `-` : 如果 file 部分写成 -，表示处理来自 standard input 的数据
 - 示例
 - `paste /etc/passwd /etc/shadow` 拼接两个文件，tab 分隔
 - `cat /etc/group|paste /etc/passwd /etc/shadow -|head -n 3`
- expand 将 tab 转换为空格
 - `expand [-t] file`
 - 选项与参数:
 - `-t` : 设定使用多少个空格来替换 [tab]，默认 tab 键用 8 个空格取代
 - 示例
 - `grep '^MANPATH' /etc/man_db.conf | head -n 3`
 - `grep '^MANPATH' /etc/man_db.conf | head -n 3 | cat -A`
 - `grep '^MANPATH' /etc/man_db.conf | head -n 3 | expand -t 6 - | cat -A`

10.9.5 split

- 将大文件切割成小文件，便于传输和存储
 - `split [-bl] file PREFIX`
 - 选项与参数:
 - `-b` : 设定切割成的文件大小，可用 b, k, m 等设定大小单位
 - `-l` : 以行数来进行分隔

- `PREFIX` : 设定分隔文件名的前缀
- 示例
 - `split -b 300k /etc/services services` 切割
 - `cat services* >> servicesback` 合并
 - `ls -al / | split -l 10 - lsroot` 管道的使用, 10 行一个文件

10.9.6 xargs

- args 是 arguments 的意思, 处理命令的参数, 功能是读入 stdin 的数据, 并且以空格符或者换行符作为分隔, 将读入内容分隔为 arguments
- 这个命令存在的原因是, 有些命令不支持管道功能, 可以使用 xargs 为其提供 stdin 输入
- xargs [-0epn] command
 - 选项与参数:
 - `-0` : 如果输入的 stdin 含有特殊字符, 例如 ‘, , 空格键时, 该参数可将其还原成一般字符
 - `-e` : 设定 EOF (end of file), 后接字符串, 要求 xargs 遇到该字符串时, 结束处理
 - `-p` : 执行每个指令的 argument 时, 要求使用者确认
 - `-n` : 后接次数, 设定每次 command 指令执行时, 使用几个参数
 - 当 xargs 后面没有接任何的指令时, 默认以 echo 来进行输出
- 示例
 - 希望取出 passwd 中前三行的用户名, 并用 id 命令显示每个用户的信息
 - `id $(cut -d ':' -f 1 /etc/passwd | head -n 3)` 错误
 - `cut -d ':' -f 1 /etc/passwd | head -n 3 | id` 一样达不到目的
 - `cut -d ':' -f 1 /etc/passwd | head -n 3 | xargs -n 1 id` 没问题
 - 同上, 但是每一次执行 id 命令时, 要求确认
 - `cut -d ':' -f 1 /etc/passwd | head -n 3 | xargs -p -n 1 id`

10.9.7 关于减号

- 管道命令中, 有时为了省略文件名称, 可以使用 `-` 来替代 stdin 和 stdout
- 示例
 - `tar -cvf - /home | tar -xvf - -C /tmp/` 前一个减号替代输出文件名, 后一个减号替代输入文件名

11 正则表达式与文件格式化处理

11.1 什么是正则表达式

- Regular Expression(RE), 正则表达式
- 是处理字符串的方法, 通过特殊符号(人为规定)的辅助, 帮助使用者完成字符串 搜索/删除/替代 的操作
- 在 unix 世界得到广泛支持, 各种编程语言中, 也有相应的开发库 API
- 分为基本正则表达式(Basic RE)与扩展正则表达式(Extended RE)
- 正则表达式与通配符是两种东西, 通配符是 shell 提供的接口, 正则表达式则是字符串处理方式

11.2 基本正则表达式

11.2.1 注意语系的问题

- 语系对正则表达式有影响, 例如
 - `LANG=C` [A-Z] 为 A B C ... Z
 - `LANG=zh_TW` [A-Z] 则为 A b B c C d D ... z Z
- 一些预设定的特殊符号

特殊符号	代表意义
<code>[:alnum:]</code>	英文大小写字符及数字, 即 0-9, A-Z, a-z
<code>[:alpha:]</code>	任何英文大小写字符, 即 A-Z, a-z
<code>[:blank:]</code>	空格键与 [Tab] 按键
<code>[:cntrl:]</code>	控制按键, 包括 CR, LF, Tab, Del.. 等
<code>[:digit:]</code>	数字, 即 0-9
<code>[:graph:]</code>	空格符(空格键与 [Tab] 按键)外的其他所有按键
<code>[:lower:]</code>	小写字符, 即 a-z
<code>[:print:]</code>	任何可以被打印出来的字符
<code>[:punct:]</code>	标点符号(punctuation symbol), 即: " ? ! ; : # \$...
<code>[:upper:]</code>	大写字符, 即 A-Z
<code>[:space:]</code>	会产生空白的字符, 包括空格键, [Tab], CR 等
<code>[:xdigit:]</code>	16 进位的数字类型, 包括: 0-9, A-F, a-f

11.2.2 与 grep 结合使用

- `grep [-A] [-B] [--color=auto] '搜索字符串' filename`
- grep 所使用的搜索字符串, 可以使用正则表达式表示
- 选项与参数:
 - `-A` : 后面可加数字, after 的意思, 除了显示该行外, 后续的 n 行也显示出来
 - `-B` : 后面可加数字, before 的意思, 除了显示该行外, 前面的 n 行也显示出来
 - `--color=auto` 搜索匹配内容用特殊颜色显示
- 示例
 - `dmesg | grep 'qxl'` 基本搜索
 - `dmesg | grep -n --color=auto 'qxl'` 显示行号
 - `dmesg | grep -n -A3 -B2 --color=auto 'qxl'` 显示上下文

11.2.3 grep 搜索示例

- 数据文件内容

```
"Open Source" is a good mechanism to develop programs. apple is my favorite food.  
Football game is not use feet only.  
this dress doesn't fit me.
```

```

However, this dress is about $ 3183 dollars.^M
GNU is free air not free beer.^M
Her hair is very beauty.^M
I can't finish the test.^M
Oh! The soup taste good.^M
motorcycle is cheap than car.
This window is clear.
the symbol '*' is represented as start.
Oh! My god!
The gd software is a library for drafting programs.^M You are the best is mean you are the no. 1.
The world <Happy> is the same with "glad".
I like dog.
google is the best tools for search keyword.
gooooooole yes!
go! go! Let's go.
# I am VBird

```

- 搜索特定字符串

```

grep -n 'the' regular_expression.txt
grep -vn 'the' regular_expression.txt
grep -in 'the' regular_expression.txt

```

- 匹配某个集中中的任一字符

```

grep -n 't[ae]st' regular_expression.txt
grep -n 'oo' regular_expression.txt
grep -n '[^g]oo' regular_expression.txt
grep -n '[^a-z]oo' regular_expression.txt
grep -n '[0-9]' regular_expression.txt
grep -n '^[:lower:]']oo' regular_expression.txt
grep -n '[[[:digit:]]]' regular_expression.txt

```

- 匹配行首 (^) 和行尾 (\$)

```

grep -n '^the' regular_expression.txt
grep -n '^[a-z]' regular_expression.txt
grep -n '^[[[:lower:]]]' regular_expression.txt
grep -n '^[^a-zA-Z]' regular_expression.txt
grep -n '\.$' regular_expression.txt
cat -An regular_expression.txt | head -n 10 | tail -n 6
grep -n '^$' regular_expression.txt
grep -v '^$' /etc/rsyslog.conf | grep -v '^#'

```

- 匹配任意一个字符 (.) 或重复多个字符 (*)

```

grep -n 'g..d' regular_expression.txt
grep -n 'ooo*' regular_expression.txt
grep -n 'goo*g' regular_expression.txt
grep -n 'g*g' regular_expression.txt
grep -n 'g.*g' regular_expression.txt # .* 代表零个或多个任意字符
grep -n '[0-9][0-9]*' regular_expression.txt

```

- 限定连续字符范围 {}

```

grep -n 'o\{2\}' regular_expression.txt
grep -n 'go\{2,5\}g' regular_expression.txt
grep -n 'go\{2,\}g' regular_expression.txt

```

11.2.4 特殊字符汇总

RE 字符	含义
<code>^word</code>	匹配行首为 word 的行
<code>word\$</code>	匹配行尾为 word 的行
<code>.</code>	一个任意字符
<code>\</code>	转义字符，去除特殊符号的特殊意义
<code>\</code>	重复零个到任意多个的前一个 RE 字符
<code>[list]</code>	匹配 list 字符集合中的任一 RE 字符
<code>[n1-n2]</code>	匹配列出来的字符范围内的任一 RE 字符，如 [a-z]
<code>[^list]</code>	匹配非 list 列表中的任一 RE 字符
<code>{n}</code>	匹配连续 n 个前一个 RE 字符
<code>{n,m}</code>	匹配连续 n 到 m 个的前一个 RE 字符
<code>{n,}</code>	匹配连续 n 个或以上的前一个 RE 字符

11.2.5 sed 工具

- Stream EDitor 流编辑器
- `sed [-nefr] [动作]`
- 选项与参数:
 - `-n` : silent 模式。默认 sed 会将数据输出到屏幕上。使用 -n 后，只有经过 sed 特殊处理的那一行 (或者动作) 才会被显示出来
 - `-e` : 在命令列模式上进行 sed 的动作编辑
 - `-f` : 将 sed 的动作写在一个文件内，-f filename 执行 filename 内的 sed 动作
 - `-r` : sed 的动作支持扩展型正规表示法的语法 (默认是基本正则表达式语法)
 - `-i` : 直接修改文件内容，而非处理完毕后由屏幕输出
- 动作说明: `[n1[,n2]]function`
 - `n1, n2` : 不是必须的，表示在 n1 到 n2 行执行设定的动作
 - function 说明
 - `a` : 追加，后面可以接字符串，字符串会添加在当前行的下一行
 - `c` : 取代，后面可以接字符串，这些字符串可以取代 n1,n2 之间的行
 - `d` : 删除，因为是删除啊，所以 d 后面通常不接任何咚咚；
 - `i` : 插入，i 的后面可以接字符串，而这些字符串会在新的一行出现 (目前的上一行)；
 - `p` : 打印，亦即将某个选择的数据印出。通常 p 会与参数 sed -n 一起运作 ~
 - `s` : 取代，可以直接进行取代的工作哩！通常这个 s 的动作可以搭配正规表示法！例如 1,20s/old/new/g 就是啦！
- 示例
 - 以行为单位的新增/删除功能


```
nl /etc/passwd | sed '2,5d'
nl /etc/passwd | sed '2a drink tea'
nl /etc/passwd | sed '2a Drink tea or .....\
> drink beer ?'
```
 - 以行为单位的取代与显示功能


```
nl /etc/passwd | sed '2,5c No 2-5 number'
nl /etc/passwd | sed -n '5,7p'
```
- 部分数据的搜索与替代
 - sed 's/要被取代的字符串/新的字符串/g'
 - 类似 vi 编辑器中的相关操作

```
/sbin/ifconfig eth0 | grep 'inet' | sed 's/^.*inet //g' \
> | sed 's/ *netmask.*$/g'
cat /etc/man_db.conf | grep 'MAN' | sed 's/#.*$/g' | sed '/^$/d'
```

- 直接修改文件内容，`-i` 选项

```
sed -i 's/\.*/!/_g' regular_express.txt
sed -i '$a # This is a test' regular_express.txt
```

11.3 扩展正则表达式

RE 字符	意义与范例
+	重复“一个或一个以上”的前一个 RE 字符 egrep -n ‘go+d’ regular_express.txt
?	“零个或一个”的前一个 RE 字符 egrep -n ‘go?d’ regular_express.txt
	用或 (or) 的方式找出数个字符串 egrep -n ‘gd’
()	找出“群组”字符串 egrep -n ‘g(la’
()+	多个重复群组的判别 echo ‘AxyzxyzxyzC’

11.4 文件的格式化与相关处理

11.4.1 格式化打印: printf

- printf ‘打印格式’ 实际内容
- 选项与参数:
- 关于格式方面的几个特殊样式:
 - `\a` : 警告声音输出
 - `\b` : 退格键 (backspace)
 - `\f` : 清除屏幕 (form feed)
 - `\n` : 输出新的一行
 - `\r` : 亦即 Enter 按键
 - `\t` : 水平的 [tab] 按键
 - `\v` : 垂直的 [tab] 按键
 - `\xNN` : NN 为两位数的数字, 可以转换数字成为字符。
- C 程序语言内, 常见格式设置
 - `%ns n` s 是数字, n 代表 string , 即多少个字符;
 - `%ni n` i 是数字, n 代表 integer , 亦即多少整数字数;
 - `%N.nf N` 表示整数部分长度, n 为小数部分长度, f 表示浮点数
- printf.txt 文件内容

Name	Chinese	English	Math	Average
DmTsai	80	60	92	77.33
VBird	75	55	80	70.00
Ken	60	90	70	73.33

- 示例

```
printf '%s\t %s\t %s\t %s\t %s\t \n' $(cat printf.txt)
printf '%10s %5i %5i %5i %8.2f \n' $(cat printf.txt | grep -v Name)
printf '\x45\n'
```

11.4.2 awk

- awk 适合处理格式化文本，将一行数据分成若干字段来处理，默认字段分隔符为空格或 TAB 键
- awk ‘条件类型 1{动作 1} 条件类型 2{动作 2} ...’ filename

```
last -n 5 | awk '{print $1 "\t" $3}'  
# $0 表示一整个全部文本
```

- awk 内建变量

变量名称	代表意义
NF	每行 (\$0) 字段总数
NR	当前 awk 处理文本行数
FS	字段分隔符，默认是空格键

```
last -n 5 | awk '{print $1 "\t lines: " NR "\t columns: " NF}'
```

- awk 逻辑运算符

运算单元	代表意义
>	大于
<	小于
>=	大于或等于
<=	小于或等于
==	等于
!=	不等于

```
cat /etc/passwd | awk '{FS=":"} $3 < 10 {print $1 "\t" $3}'  
cat /etc/passwd | awk 'BEGIN {FS=":"} $3 < 10 {print $1 "\t" $3}'
```

- pay.txt

```
Name      1st      2nd      3th  
VBird    23000    24000    25000  
DMTsai   21000    20000    23000  
Bird2    43000    42000    41000
```

```
cat pay.txt | \  
> awk 'NR==1{printf "%10s %10s %10s %10s %10s\n",$1,$2,$3,$4,"Total" }  
> NR>=2{total = $2 + $3 + $4  
> printf "%10s %10d %10d %10d %10.2f\n", $1, $2, $3, $4, total}'
```

或者

```
cat pay.txt | \  
> awk '{if(NR==1) printf "%10s %10s %10s %10s %10s\n",$1,$2,$3,$4,"Total"}  
> NR>=2{total = $2 + $3 + $4  
> printf "%10s %10d %10d %10d %10.2f\n", $1, $2, $3, $4, total}'
```

- 输出

```
Name      1st      2nd      3th      Total  
VBird    23000    24000    25000    72000.00  
DMTsai   21000    20000    23000    64000.00  
Bird2    43000    42000    41000    126000.00
```

11.4.3 文件对比工具

11.4.3.1 diff

- diff [-bBi] from-file to-file
- 以行为单位，比较文件
- 选项与参数：
 - from-file : 一个档名，作为原始比对文件的档名;
 - to-file : 一个档名，作为目的比对文件的档名;
 - from-file 或 to-file 都可以用 - 取代，表示 Standard input
 - -b : 忽略一行当中，仅有多个空白的差异 (例如 “about me” 与 “about me” 视为相同)
 - -B : 忽略空白行的差异。
 - -i : 忽略大小写的不同。
- 准备数据文件

```
mkdir -p /tmp/testpw
cd /tmp/testpw
cp /etc/passwd passwd.old
cat /etc/passwd | sed -e '4d' -e '6c no six line' > passwd.new
• diff passwd.old passwd.new
• diff /etc/rc0.d/ /etc/rc5.d/
```

11.4.3.2 cmp

- 基于字节比较文件
- cmp [-l] file1 file2
- -l: 输出所有的不同， 默认只输出第一处不同

11.4.3.3 patch

- 给文件打补丁
- 补丁文件由 diff 命令生成
- patch [-pR] < patch_file
- -p : 后面可以接『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

```
diff -Naur passwd.old passwd.new > passwd.patch
cat passwd.patch
```

```
patch -pN < patch_file # <== 更新
patch -R -pN < patch_file # <== 还原
```

```
patch -p0 < passwd.patch # 将刚刚制作出来的 patch file 用来更新旧版数据
patch -R -p0 < passwd.patch # 恢复旧文件的内容
```

11.4.3.4 pr

- 打印文件
- pr /etc/man_db.conf
- 更多说明， man pr

12 学习 Shell Scripts

12.1 什么是 Shell scripts

- shell scripts 概述
 - shell scripts 是纯文本文件，可使用任意文本编辑器编写
 - shell scripts 是一种编程语言
 - shell scripts 与具体 shell 耦合很高，我们针对 bash 进行学习
 - 由于 shell scripts 中可以很方便的调用各种 linux 命令
- 一些注意事项
 - 命令的执行从上而下、从左而右
 - 命令下达时：命令、选项与参数间的多个空白都会被忽略掉
 - 空白行将被忽略，且 [tab] 键产生的空白同样视为空格键
 - 如果读取到一个 Enter 符号 (CR)，就尝试开始执行该行 (或该串) 命令
 - 如果一行的内容太多，则可以使用”[Enter]”进行续行
 - ”#” 开始的内容即为注释
- shell scripts 的执行
 - 直接运行脚本文件：`shell.sh` 必须要具备读与可执行 `rx` 权限
 - 绝对路径方式：输入 `/home/dmtsai/shell.sh` 来执行
 - 相对路径方式：假设工作目录在 `/home/dmtsai/`，使用 `./shell.sh` 来执行
 - 变量 PATH 功能：将 shell.sh 放在 PATH 指定的目录 内，然后直接 `shell.sh` 来执行
 - 以 bash 程序来执行：使用 `bash shell.sh` 或 `sh shell.sh` 来执行
- 一个简单的 shell scripts
 - hello.sh

```
#!/bin/bash
# Program:
# This program shows "Hello World!" in your screen.
# History:
# 2015/07/16 VBird First release
```

```
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH
echo -e "Hello World! \a \n"
exit 0
```

- 解读
 - `#!/bin/bash` 指示该文件需要使用 bash 来解释执行
 - 其他多行注释，一定要写，用来说明脚本的
 1. 内容与功能
 2. 版本信息
 3. 作者与联络方式
 4. 创建日期
 5. 历史纪录
 6. 版权遵循何种 license
 7. 程序正常执行的必要条件
 8. 其他必要或不必要的信息
 - 环境变量的设定
 - 主程序部分 `echo` 所在行
 - 设定程序返回值 `exit 0`，惯例：成功为 0，不成功为非 0

12.2 简单的 Shell scripts

12.2.1 几个简单脚本

- showname.sh

```
#!/bin/bash
# Program:
# User inputs his first name and last name. Program shows his full name. # History:
# 2015/07/16 VBird First release
```

```
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH
```

```
read -p "Please input your first name: " firstname # 提示使用者输入
read -p "Please input your last name: " lastname # 提示使用者输入
echo -e "\nYour full name is: ${firstname} ${lastname}" # 结果由屏幕输出
```

- create_3_filename.sh

```
#!/bin/bash
# Program:
# Program creates three files, which named by user's input and date command.
# History:
# 2015/07/16 VBird First release
```

```
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH
```

```
# 1. 让使用者输入文件名，并取得 fileuser 这个变量;
echo -e "I will use 'touch' command to create 3 files." # 纯粹显示信息
read -p "Please input your filename: " fileuser # 提示使用者输入
# 2. 为了避免使用者随意按 Enter，利用变量功能分析文件是否有设定？
filename=${fileuser:-"filename"} # 开始判断有否配置文件名
# 3. 开始利用 date 指令来取得所需要的文件了;
date1=$(date --date='2 days ago' +%Y%m%d) # 前两天的日期
date2=$(date --date='1 days ago' +%Y%m%d) # 前一天的日期
date3=$(date +%Y%m%d) # 今天的日期
file1=${filename}${date1} # 底下三行在配置文件名
file2=${filename}${date2}
file3=${filename}${date3}
# 4. 将文件建立吧!
touch "${file1}" # 底下三行在建立文件
touch "${file2}"
touch "${file3}"
```

- multiplying.sh

- 注意其中用到 `$((运算内容))` 来进行数学运算

```
#!/bin/bash
# Program:
# User inputs 2 integer numbers; program will cross these two numbers.
# History:
# 2015/07/16 VBird First release
```

```
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH
```

```
echo -e "You SHOULD input 2 numbers, I will multiplying them! \n"
read -p "first number: " firstnu
read -p "second number: " secnu
total=$((firstnu * secnu))
echo -e "\nThe result of ${firstnu} x ${secnu} is ==> ${total}"
```

- cal_pi.sh

```

#!/bin/bash
# Program:
# User input a scale number to calculate pi number.
# History:
# 2015/07/16 VBird First release

PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:/bin
export PATH

echo -e "This program will calculate pi value.\n"
echo -e "You should input a float number to calculate pi value.\n"
read -p "The scale number (10~10000) ? " checking
num=${checking:-"10"} # 开始判断有否有输入数值
echo -e "Starting calculate pi value. Be patient."
time echo "scale=${num}; 4*a(1)" | bc -lq

```

12.2.2 script 的执行方式

- 直接执行
 - 前一小节的执行方式，均为启动一个新的 bash 环境，来运行脚本
 - 这种情况下，脚本运行结束后，在其过程中设定的各种变量，不会对主 bash 有任何影响
- 使用 source 执行，这种方式，实在主 bash 中执行脚本，脚本做的各种动作，会直接影响主 bash 环境

12.3 判断式

- 利用 test 指令
 - 简单示例
 - `test -e /dmtsai && echo "exist" || echo "Not exist"`
- 系统内置测试标志

测试的标志	含义
1. 文件类型相关	
-e	该“文件”是否存在?(常用)
-f	该“文件”是否存在且为文件 (file)?(常用)
-d	该“文件”是否存在且为目录 (directory)?(常用)
-b	该“文件”是否存在且为一个 block device 设备?
-c	该“文件”是否存在且为一个 character device 设备?
-S	该“文件”是否存在且为一个 Socket 文件?
-p	该“文件”是否存在且为一个 FIFO (pipe) 文件?
-L	该“文件”是否存在且为一个符号链接文件?
2. 文件权限相关	
-r	检测该文件是否存在且具有“可读”的权限?
-w	检测该文件是否存在且具有“可写”的权限?
-x	检测该文件是否存在且具有“可执行”的权限?
-u	检测该文件是否存在且具有“SUID”的属性?
-g	检测该文件是否存在且具有“SGID”的属性?
-k	检测该文件是否存在且具有“Sticky bit”的属性?
-s	检测该文件是否存在且为“非空白文件”?
3. 文件比较	
-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，是否均指向同一个 inode
4. 数值相关	
-eq	两数值相等 (equal)
-ne	两数值不等 (not equal)
-gt	n1 大于 n2 (greater than)

测试的标志	含义
-lt	n1 小于 n2 (less than)
-ge	n1 大于等于 n2 (greater than or equal)
-le	n1 小于等于 n2 (less than or equal)
5. 字符串相关	
test -z string	判定字符串是否为 0 ? 若 string 为空字符串, 则为 true
test -n string	判定字符串是否非为 0 ? 若 string 为空字符串, 则为 false。注: -n 亦可省略
test str1 == str2	判定 str1 是否等于 str2
test str1 != str2	判定 str1 是否不等于 str2
6. 条件组合	
-a	(and) 条件与
-o	(or) 条件或
!	反状态判定, 如 test ! -x file

- file_perm.sh

```
#!/bin/bash
# Program:
# User input a filename, program will check the flowing:
# 1.) exist? 2.) file/directory? 3.) file permissions
# History:
# 2015/07/16 VBird First release
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH
# 1. 让使用者输入档名, 并且判断使用者是否真的有输入字符串?
echo -e "Please input a filename, I will check the filename's type and permission. \n\n"
read -p "Input a filename : " filename
test -z ${filename} && echo "You MUST input a filename." && exit 0
# 2. 判断文件是否存在? 若不存在则显示讯息并结束脚本
test ! -e ${filename} && echo "The filename '${filename}' DO NOT exist" && exit 0
# 3. 开始判断文件类型与属性
test -f ${filename} && filetype="regular file"
test -d ${filename} && filetype="directory"
test -r ${filename} && perm="readable"
test -w ${filename} && perm="${perm} writable"
test -x ${filename} && perm="${perm} executable"
# 4. 开始输出信息!
echo "The filename: ${filename} is a ${filetype}"
echo "And the permissions for you are : ${perm}"
```

12.3.1 利用判断符号 []

- 使用 [], 等价于使用 test 指令, 更简洁, 如
 - [-z "\${HOME}"] ; echo \$?
- 注意事项
 - 在中括号 [] 内的每个部分都需要有空格键来分隔
 - 在中括号内的变量, 最好都以双引号括号起来
 - 在中括号内的常量, 最好都以单或双引号括号起来
- ans_yn.sh

```
#!/bin/bash
# Program:
# This program shows the user's choice
# History:
# 2015/07/16 VBird First release

PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
```

```

export PATH

read -p "Please input (Y/N): " yn
[ "${yn}" == "Y" -o "${yn}" == "y" ] && echo "OK, continue" && exit 0
[ "${yn}" == "N" -o "${yn}" == "n" ] && echo "Oh, interrupt!" && exit 0
echo "I don't know what your choice is" && exit 0

```

12.3.2 shell 脚本的命令行参数

- 使用 `/path/to/scriptname opt1 opt2 opt3 opt4` 类似方式执行命令, `opt1 ... opt4` 即为命令行参数
- shell 脚本可以读入这些参数, 然后根据参数做各种不同处理
- 对于上例
 - `$0` 为 `/path/to/scriptname opt1 opt2 opt3 opt4`
 - `$1` 为 `opt1`
 - `$2` 为 `opt2`
 - `$3` 为 `opt3`
 - `$4` 为 `opt4`
 - `$#` 为命令行上的参数个数, 这里为 4
 - `$*` 为 `$1c2c3c4`, `c` 为分隔符, 默认为空格, 也即 `$1 $2 $3 $4`, 是一个字符串 (拼接起来的)
 - `$@` 为 `"$1" "$2" "$3" "$4"`, 是一个字符串数组
- how_paras.sh

```

#!/bin/bash
# Program:
# Program shows the script name, parameters...
# History:
# 2015/07/16 VBird First release

PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

```

```

echo "The script name is ==> ${0}"
echo "Total parameter number is ==> ${#}"
[ "${#}" -lt 2 ] && echo "The number of parameter is less than 2. Stop here." && exit 0
echo "Your whole parameter is ==> '$@'"
echo "The 1st parameter ==> ${1}"
echo "The 2nd parameter ==> ${2}"

```

- 使用 `shift` 移动命令行参数
 - 由于 `$0, $1 ...` 最多只能到 `$9`, 当命令行参数超过 9 个时, 脚本无法一次读取到全部命令行参数
 - 使用 `shift` 可以将命令行所有参数整体进行左移
- shift_paras.sh

```

#!/bin/bash
# Program:
# Program shows the effect of shift function.
# History:
# 2009/02/17 VBird First release

```

```

PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

echo "Total parameter number is ==> ${#}"
echo "Your whole parameter is ==> '$@'"
shift # 进行第一次『一个变量的 shift』
echo "Total parameter number is ==> ${#}"

```

```

echo "Your whole parameter is ==> '$@'"
shift 3 # 进行第二次『三个变量的 shift 』
echo "Total parameter number is ==> $#"
echo "Your whole parameter is ==> '$@'"

```

- 运行命令行
 - sh shift_paras.sh one two three four five six
 - 观察并理解输出结果

12.4 条件判断式

12.4.1 if ... then

- 简单判断
 - 语法

```

if [ 判断表达式 ]; then
    当条件判断式成立时，可以运行的语句
fi

```

- ans_yn-2.sh

```

#!/bin/bash
# Program:
# This program shows the user's choice
# History:
# 2015/07/16 VBird First release

PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

read -p "Please input (Y/N): " yn
if [ "${yn}" == "Y" ] || [ "${yn}" == "y" ]; then
    echo "OK, continue"
    exit 0
fi
if [ "${yn}" == "N" ] || [ "${yn}" == "n" ]; then
    echo "Oh, interrupt!"
    exit 0
fi
echo "I don't know what your choice is" && exit 0

```

- 多重、复杂判断

- 语法

```

# 一个条件判断，分成功进行与失败进行 (else)
if [ 条件判断式 1 ]; then
    当条件判断式 1 成立时，可以运行的语句
elif [ 条件判断式 2 ]; then
    当条件判断式 2 成立时，可以运行的语句
else
    当上述条件判断式均不成立时，可以运行的语句
fi

```

- ans_yn-3.sh

```

#!/bin/bash
# Program:
# This program shows the user's choice
# History:

```

```

# 2015/07/16 VBird First release

PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

read -p "Please input (Y/N): " yn
if [ "${yn}" == "Y" ] || [ "${yn}" == "y" ]; then
    echo "OK, continue"
elif [ "${yn}" == "N" ] || [ "${yn}" == "n" ]; then
    echo "Oh, interrupt!"
else
    echo "I don't know what your choice is"
fi

• netstat.sh

#!/bin/bash
# Program:
# Using netstat and grep to detect WWW,SSH,FTP and Mail services.
# History:
# 2015/07/16 VBird First release

PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

# 1. 先作一些告知的动作而已 ~
echo "Now, I will detect your Linux server's services!"
echo -e "The www, ftp, ssh, and mail(smtp) will be detect! \n"

# 2. 开始进行一些测试的工作，并且也输出一些信息啰！
testfile=/dev/shm/netstat_checking.txt
netstat -tuln > ${testfile} # 先转存数据到内存当中！不用一直执行
netstat testing=$(grep ":80" ${testfile}) # 倾测看 port 80 在否？
if [ "${testing}" != "" ]; then
    echo "WWW is running in your system."
fi
testing=$(grep ":22" ${testfile}) # 倾测看 port 22 在否？
if [ "${testing}" != "" ]; then
    echo "SSH is running in your system."
fi
testing=$(grep ":21" ${testfile}) # 倾测看 port 21 在否？
if [ "${testing}" != "" ]; then
    echo "FTP is running in your system."
fi
testing=$(grep ":25" ${testfile}) # 倾测看 port 25 在否？
if [ "${testing}" != "" ]; then
    echo "Mail is running in your system."
fi

• cal_retired.sh

#!/bin/bash
# Program:
# You input your demobilization date, I calculate how many days before you demobilize.
# History:
# 2015/07/16 VBird First release

PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

```

```

# 1. 告知用户这支程序的用途，并且告知应该如何输入日期格式？
echo "This program will try to calculate :"
echo "How many days before your demobilization date..."
read -p "Please input your demobilization date (YYYYMMDD ex>20150716): " date2
# 2. 测试一下，这个输入的内容是否正确？利用正规表示法啰～
date_d=$(echo ${date2} | grep '[0-9]{8}') # 看看是否有八个数字
if [ "${date_d}" == "" ]; then
    echo "You input the wrong date format...."
    exit 1
fi

# 3. 开始计算日期啰～
declare -i date_dem=$((date --date="${date2}" +%s) # 退伍日期秒数
declare -i date_now=$((date +%s) # 现在日期秒数
declare -i date_total_s=$((date_dem-$date_now)) # 剩余秒数统计
declare -i date_d=$((date_total_s/60/60/24)) # 转为日数

if [ "${date_total_s}" -lt "0" ]; then # 判断是否已退伍
    echo "You had been demobilization before: " $((-1*${date_d})) " ago"
else
    declare -i date_h=$((($date_total_s-$date_d)*60*60*24)/60/60)
    echo "You will demobilize after ${date_d} days and ${date_h} hours."
fi

```

12.4.2 case ... esac

- 相当于其他编程语言中的 switch case 控制结构

- 语法

```

case $ 变量名称 in # <== 关键词 case , 变量前要有 $
    " 第一个变量内容") # <== 每个变量内容建议用双引号括起来, 小括号) 结束
        程序段
    ;;
    " 第二个变量内容")
        程序段
    ;;
*)
    # <== 最后一个变量内容用 * 来代表所有其他值
    不包含第一个变量内容与第二个变量内容的其他程序执行段
    exit 1
    ;;
esac # <== 最终的 case 结尾

```

- hello-3.sh

```

#!/bin/bash
# Program:
# Show "Hello" from $1.... by using case .... esac
# History:
# 2015/07/16 VBird First release

PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

case ${1} in
    "hello")
        echo "Hello, how are you ?"
        ;;
    "")

```

```

        echo "You MUST input parameters, ex> ${0} someword"
        ;;
*) # 其实相当于通配符, 0~ 无穷多个任意字符
    echo "Usage ${0} {hello}"
    ;;
esac

```

- show123.sh

```

#!/bin/bash
# Program:
# This script only accepts the flowing parameter: one, two or three.
# History:
# 2015/07/17 VBird First release

PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

echo "This program will print your selection !"
# read -p "Input your choice: " choice # 暂时禁用
# case ${choice} in # 暂时禁用
case ${1} in
    "one")
        echo "Your choice is ONE"
        ;;
    "two")
        echo "Your choice is TWO"
        ;;
    "three")
        echo "Your choice is THREE"
        ;;
*)
    echo "Usage ${0} {one|two|three}"
    ;;
esac

```

12.4.3 函数功能

- 语法

```

function fname(){
    代码段
}

```

- 使用时，函数的具体实现应该放在调用前，否则无法调用

- show123-2.sh

```

#!/bin/bash
# Program:
# Use function to repeat information.
# History:
# 2015/07/17 VBird First release

```

```

PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

```

```

function printit(){
    echo -n "Your choice is " # 加上 -n 可以继续在同一行显示
}

```

```

echo "This program will print your selection !"
case ${1} in
    "one")
        printit; echo ${1} | tr 'a-z' 'A-Z' # 将参数做大小写转换!
        ;;
    "two")
        printit; echo ${1} | tr 'a-z' 'A-Z'
        ;;
    "three")
        printit; echo ${1} | tr 'a-z' 'A-Z'
        ;;
    *)
        echo "Usage ${0} {one|two|three}"
        ;;
esac

```

- 参数传递
 - 函数获取参数的方式，跟主程序读取命令行参数方式类似

- show123-3.sh

```

#!/bin/bash
# Program:
# Use function to repeat information.
# History:
# 2015/07/17 VBird First release

PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

function printit(){
    echo "Your choice is ${1}" # $1 即为函数参数
}
echo "This program will print your selection !"
case ${1} in
    "one")
        printit 1 # 请注意， printit 指令后面还有接参数！
        ;;
    "two")
        printit 2
        ;;
    "three")
        printit 3
        ;;
    *)
        echo "Usage ${0} {one|two|three}"
        ;;
esac

```

- 到 /etc/init.d 目录查看相关代码，了解启动时的脚本执行机制及脚本设计方法

12.5 循环

12.5.1 while 和 until 循环

- while 循环语法

```

while [ condition ] # 中括号内的状态就是判断式
do # 循环开始

```

```

程序段落
done # 循环结束

• until 循环语法

until [ condition ]
do
    程序段落
done

• yes_to_stop.sh

#!/bin/bash
# Program:
# Repeat question until user input correct answer.
# History:
# 2015/07/17 VBird First release

PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

while [ "${yn}" != "yes" -a "${yn}" != "YES" ]
do
    read -p "Please input yes/YES to stop this program: " yn
done
echo "OK! you input the correct answer."

```

• cal_1_to_100.sh

```

#!/bin/bash
# Program:
# Use loop to calculate "1+2+3+...+100" result.
# History:
# 2015/07/17 VBird First release

PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

s=0 # 这是加总的数值变数
i=0 # 这是累计的数值，亦即是 1, 2, 3....
while [ "${i}" != "100" ]
do
    i=$((i+1)) # 每次 i 都会增加 1
    s=$((s+i)) # 每次都会加总一次!
done
echo "The result of '1+2+3+...+100' is ==> $s"

```

12.5.2 for 循环

- 语法

```

for var in con1 con2 con3 ...
do
    程序段
done

```

- 第一次循环时, \$var = con1
- 第二次循环时, \$var = con2
- 第三次循环时, \$var = con3
- ...

- userid.sh

```
#!/bin/bash
# Program
# Use id, finger command to check system account's information.
# History
# 2015/07/17 VBird first release

PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH
```

```
users=$(cut -d ':' -f1 /etc/passwd)
for username in ${users}
do
    id ${username}
done
```

- ping_ip.sh

```
#!/bin/bash
# Program
# Use ping command to check the network's PC state.
# History
# 2015/07/17 VBird first release
```

```
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH
```

```
network="192.168.1" # 先定义一个网域的前面部分!
for sitenu in $(seq 1 100) # seq 为 sequence(连续) 的缩写
do
    # 底下的程序在取得 ping 的回传值是正确的还是失败的!
    ping -c 1 -w 1 ${network}.${sitenu} &> /dev/null && result=0 || result=1
    # 开始显示结果是正确的启动 (UP) 还是错误的没有连通 (DOWN)
    if [ "${result}" == 0 ]; then
        echo "Server ${network}.${sitenu} is UP."
    else
        echo "Server ${network}.${sitenu} is DOWN."
    fi
done
```

- dir_perm.sh

```
#!/bin/bash
# Program:
# User input dir name, I find the permission of files.
# History:
# 2015/07/17 VBird First release
```

```
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH
```

```
# 1. 先看看这个目录是否存在啊?
```

```
read -p "Please input a directory: " dir
if [ "${dir}" == "" -o ! -d "${dir}" ]; then
    echo "The ${dir} is NOT exist in your system."
    exit 1
fi
# 2. 开始测试文件啰 ~
filelist=$(ls ${dir}) # 列出所有在该目录下的文件名
```

```

for filename in ${filelist}
do
    perm=""
    test -r "${dir}/${filename}" && perm="${perm} readable"
    test -w "${dir}/${filename}" && perm="${perm} writable"
    test -x "${dir}/${filename}" && perm="${perm} executable"
    echo "The file ${dir}/${filename}'s permission is ${perm}"
done

```

12.5.3 传统 for 循环

- 语法

```

for (( 初始化; 限制条件; 每次循环后动作 ))
do
    程序段
done

```

- cal_1_to_100-2.sh

```

#!/bin/bash
# Program:
# Try do calculate 1+2+....+${your_input}
# History:
# 2015/07/17 VBird First release

PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

read -p "Please input a number, I will count for 1+2+...+your_input: " nu
s=0
for (( i=1; i<=${nu}; i=i+1 ))
do
    s=$((s+i))
done

echo "The result of '1+2+3+...+${nu}' is ==> ${s}"

```

12.5.4 使用数组及随机数

- what_to_eat.sh

```

#!/bin/bash
# Program:
# Try do tell you what you may eat. # History:
# 2015/07/17 VBird First release

```

```

PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

eat[1]="" 卖当当汉堡" # 写下你所收集到的店家!
eat[2]="" 肯爷爷炸鸡"
eat[3]="" 彩虹日式便当"
eat[4]="" 越油越好吃大雅"
eat[5]="" 想不出吃啥学餐"
eat[6]="" 太师父便当"
eat[7]="" 池上便当"
eat[8]="" 怀念火车便当"

```

```

eat[9]="一起吃泡面"
eatnum=9
# 需要输入有几个可用的餐厅数!
check=$(( ${RANDOM} * ${eatnum} / 32767 + 1 ))
echo "your may eat ${eat[$check]}"

• what_to_eat-2.sh

#!/bin/bash
# Program:
# Try do tell you what you may eat.
# History:
# 2015/07/17 VBird First release

PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:/bin
export PATH

eat[1]="卖当当汉堡"
eat[2]="肯爷爷炸鸡"
eat[3]="彩虹日式便当"
eat[4]="越油越好吃大雅"
eat[5]="想不出吃啥学餐"
eat[6]="太师父便当"
eat[7]="池上便当"
eat[8]="怀念火车便当"
eat[9]="一起吃泡面"
eatnum=9
eated=0
while [ "${eated}" -lt 3 ]; do
    check=$(( ${RANDOM} * ${eatnum} / 32767 + 1 ))
    mycheck=0
    if [ "${eated}" -ge 1 ]; then
        for i in $(seq 1 ${eated})
        do
            if [ ${eatedcon[$i]} == $check ]; then
                mycheck=1
            fi
        done
    fi
    if [ ${mycheck} == 0 ]; then
        echo "your may eat ${eat[$check]}"
        eated=$(( ${eated} + 1 ))
        eatedcon[$eated]=${check}
    fi
done

```

12.6 shell scripts 的调试

- sh [-nvx] script.sh
 - 选项与参数:
 - **-n** : 不执行 script.sh, 仅查询语法的问题
 - **-v** : 执行 script.sh 前, 先将其内容输出到屏幕上
 - **-x** : 先显示命令行, 再执行并显示输出, 调试时必用

13 第 13 章 Linux 账号管理与 ACL 权限设定

本章开始，只记录重要内容的摘要

13.1 Linux 账号与群组

13.1.1 使用者标识符

- uid, gid 的概念
- id 命令

13.1.2 使用者账号

- /etc/passwd 文件结构，7 个字段
- /etc/shadow 文件结构，9 个字段
 - 账号名
 - 密码（加密后的）
 - 最近修改密码的日期，1970/1/1 起流逝的天数
 - `echo $((($date --date="2015/05/04" +%s)/86400+1))`
 - 密码不可被修改的天数，0 表示随时可以被修改
 - 密码需要重新修改的天数，强制用户要在指定天数内修改密码
 - 密码需要修改前的提示天数
 - 密码过期后的宽限时间，过期后，密码失效，导致用户无法登陆
 - 账号失效日期，表示规定日期后，账号无法使用
 - 保留

13.1.3 关于群组：有效与初始群组、groups、newgrp

- /etc/group 结构
 - 组名
 - 组密码
 - gid
 - 此群用户列表（逗号分隔，注意不要有空格）
- 有效组 (effective group) 与初始组 (initial group)
 - 初始组，由 passwd 文件中，该用户所在行的 gid 确认
 - 用户可能属于多个组，但是创建文件时，使用有效组
- 几个相关命令及文件
- groups 命令，显示所有用户属于的组，第一个即为有效组
- newgrp 命令，切换有效组（必须是已经属于的）
- usermod，修改用户信息
- gpasswd，创建群管理员
 - /etc/gshadow 文件

13.2 账号管理

13.2.1 添加与删除账号

- useradd 命令，使用如下文件内容
 - /etc/default/useradd
 - /etc/login.defs
 - /etc/skel/*
- passwd，更改用户密码
- chage，更改用户密码过期信息
- usermod，修改用户信息

- userdel, 删除用户

13.2.2 用户功能

- id, 显示用户 id
- finger, 查询用户信息, 现默认不再安装 fingerd 服务
- chfn, change finger
- chsh, change shell

13.2.3 添加或删除组

- groupadd, 添加组
- groupmod, 修改组信息
- groupdel, 删除组
- gpasswd, 群组管理员功能

13.3 主机的权限规划

13.3.1 什么是 ACL 及如何支持启动 ACL

- ACL = Access Control List
- 更细的权限设定
 - user: 可以针对某个用户设定权限
 - group: 可以针对某个组设定权限
 - mask: 可以针对在该目录下建立新文件/目录时, 设定默认权限
- linux 默认支持
- 若确认, 可以输入
 - dmesg | grep -i acl
 - 应能看到输出中有 +ACL

13.3.2 ACL 设定: getfacl, setfacl

13.3.3 setfacl 设定某个文件的 ACL 权限

- setfacl 设定某个文件的 ACL 规范
- **setfacl [-bkRd] [{-m|-x} acl 参数] 目标文件名**
- 选项与参数:
 - **-m** : 设定后续的 acl 参数给文件使用, 不可与 -x 合用;
 - **-x** : 删除后续的 acl 参数, 不可与 -m 合用;
 - **-b** : 移除『所有的』ACL 设定参数;
 - **-k** : 移除『预设的』ACL 参数
 - **-R** : 递归设定 acl , 亦即包括次目录都会被设定起来;
 - **-d** : 设定『预设 acl 参数』的意思! 只对目录有效, 在该目录新建的数据会引用此默认值
- 示例
 - setfacl -m u:username:rx acl_test1
 - ls -l acl_test1 # 可发现权限最后多了一个 + 号
 - getfacl acl_test1 # 查看具体设定

13.3.4 getfacl 取得某个文件的 ACL 设定

- getfacl filename
- 更多示例
 - setfacl -m g:mygroup1:rx acl_test1

- setfacl -m m:r acl_test1 # 用户或者组权限，必须存在于 mask 的设定范围内，才会生效，有效权限 = 用户权限 & mask，mask 通常设定为 rwx
- 另外一个
 - cd /srv/projecta # permission denied
 - setfacl -m u:myuser1:rx /srv/projecta # 使用 root 设定
 - cd /srv/projecta # 再次使用 myuser1 试图进入，可以成功
 - 注意，此种权限设定，不会被 projecta 的子目录继承
- setfacl -m d:u:myuser1:rx /srv/projecta # 此种设定，权限可以被继承，在该目录下新建文件或目录时，权限能够继承到
- setfacl -b filename # 清除所有 ACL 设定
- setfacl -x u:myuser1 /srv/projecta # 清除某一设定
- setfacl -x d:u:myuser1 /srv/projecta # 清除某一设定

13.4 使用者身份切换

13.4.1 su 身份切换

- su [-lm] [-c 指令] [username]
- 选项与参数：
 - : 单纯使用 - 如『su -』代表使用 login-shell 的变量文件读取方式来登入系统；若使用者名称没有加上去，则代表切换为 root 的身份。
 - -l : 与 - 类似，但后面需要加欲切换的使用者账号！也是 login-shell 的方式。
 - -m :-m 与 -p 是一样的，表示『使用目前的环境设定，而不读取新使用者的配置文件』
 - -c : 仅进行一次指令，所以 -c 后面可以加上指令喔！

13.4.2 sudo

- sudo [-b] [-u 新使用者账号]
- 选项与参数：
 - -b : 将后续的指令放到背景中让系统自行执行，而不与目前的 shell 产生影响
 - -u : 后面可以接欲切换的使用者，若无此项则代表切换身份为 root 。
- sudo 执行流程
 1. 当用户执行 sudo 时，系统于 /etc/sudoers 文件中搜寻该使用者是否有执行 sudo 的权限；
 2. 若使用者具有可执行 sudo 的权限后，便让使用者『输入用户的密码』来确认；
 3. 若密码输入成功，便开始进行 sudo 后续接的指令（但 root 执行 sudo 时，不需要输入密码）；
 4. 若欲切换的身份与执行者身份相同，那也不需要输入密码。
- visudo 与 /etc/sudoers

13.5 用户的特殊 shell 与 PAM 模块

13.6 Linux 上的用户通信

14 附录

14.1 使用到的脚本

14.1.1 md2pdf.sh

- 功能
 - 将 md 文件转化为 pdf 文件
 - 需要安装 pandoc 工具
- 脚本内容

```
#!/usr/bin/env bash

#FILENAME="${1%.}"
#FONTNAME="WenQuanYi Zen Hei"
FONTNAME="SimHei"
FLAGS="--pdf-engine=xelatex "
FLAGS+=" -H head.tex "

while getopts f:tnhi:c opt
do
    case "$opt" in
        h) echo
            echo "Usage:"
            echo "$(basename $0) [-t] [-n] [-f fontname] filename"
            echo "-t: 添加目录 TOC"
            echo "-n: 为不同层级添加数字编号"
            echo "-c: 为 URL 链接添加颜色"
            echo "-f fontname: 指定中文字体, 名字中不要带空格"
            echo
            exit;;
        i) FILENAME=${OPTARG}
           FILENAMENOEXT=${OPTARG%.*};;
        t) FLAGS+=" --toc ";;
        n) FLAGS+=" -N ";;
        f) FONTNAME=$OPTARG;;
        c) FLAGS+=" -V colorlinks";;
        *) echo "unknown option $opt";
           break;;
    esac
done

set -f
FLAGS+=" -V CJKmainfont=$FONTNAME "
pandoc $FLAGS $FILENAME -o ${FILENAMENOEXT}.pdf
```

14.1.2 head.tex

- 说明
 - md2pdf.sh 使用
 - 解决 latex 页面大小、首行缩进、中文正确断行、链接加颜色问题
 - 解决 md 文件多级列表、多级编号问题
- 文件内容

```
\usepackage{fancyvrb,newverbs}
\usepackage[top=2cm, bottom=1.5cm, left=2cm, right=2cm]{geometry}
\usepackage{indentfirst}
```

```

\XeTeXlinebreaklocale "zh"
\XeTeXlinebreakskip 0pt plus 1pt

% change background color for inline code in
% markdown files. The following code does not work well for
% long text as the text will exceed the page boundary
\definecolor{bgcolor}{HTML}{E0E0E0}
\let\oldtexttt\texttt

\renewcommand{\texttt}[1]{
  \colorbox{bgcolor}{\oldtexttt{#1}}
}

%% color and other settings for hyperref package
\hypersetup{
  bookmarksopen=true,
  linkcolor=blue,
  filecolor=magenta,
  urlcolor=RoyalBlue,
}

\usepackage{enumitem}
\setlistdepth{9}

\setlist[itemize,1]{label=$\bullet$}
\setlist[itemize,2]{label=$\bullet$}
\setlist[itemize,3]{label=$\bullet$}
\setlist[itemize,4]{label=$\bullet$}
\setlist[itemize,5]{label=$\bullet$}
\setlist[itemize,6]{label=$\bullet$}
\setlist[itemize,7]{label=$\bullet$}
\setlist[itemize,8]{label=$\bullet$}
\setlist[itemize,9]{label=$\bullet$}
\renewlist[itemize]{itemize}{9}

\setlist[enumerate,1]{label=$\arabic*.$}
\setlist[enumerate,2]{label=$\alpha*.$}
\setlist[enumerate,3]{label=$\roman*.$}
\setlist[enumerate,4]{label=$\arabic*.$}
\setlist[enumerate,5]{label=$\alpha*.$}
\setlist[enumerate,6]{label=$\roman*.$}
\setlist[enumerate,7]{label=$\arabic*.$}
\setlist[enumerate,8]{label=$\alpha*.$}
\setlist[enumerate,9]{label=$\roman*.$}
\renewlist[enumerate]{enumerate}{9}

```

14.1.3 批量 md 文件处理脚本

- 脚本功能说明
 - 将所有 chapterxx.md 拼成一个大文件
 - 将该文件转换为 pdf 文件
 - 删除该文件
- 脚本内容

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

```
FILENAME=$(basename $(pwd))_`date +%Y%m%d%H%M`
```

```

cat /dev/null > $FILENAME.md
for i in `ls chap*.md`;
do
    cat $i >> $FILENAME.md
    echo >> $FILENAME.md
    echo "\clearpage" >> $FILENAME.md
    echo >> $FILENAME.md
done

md2pdf.sh -t -n -c -i $FILENAME.md
rm $FILENAME.md

COVERFILE=$(pwd)/cover.pdf
#echo $COVERFILE
if [ -e $COVERFILE ]; then
    echo "cover.pdf found. adding it..."
    pdftk cover.pdf $FILENAME.pdf cat output $FILENAME.cover.pdf
    echo "done."
fi

```

14.1.4 生成的 pdf 添加水印

- 使用工具 pdftk为欲处理 pdf 文件添加背景
- 对于 macOS, 使用[这个版本](#)
- 脚本内容

```

#!/usr/bin/env bash

CNAME="gray"

while getopts c:hi: opt
do
    case "$opt" in
        h) echo
            echo "Usage:"
            echo "$(basename $0) [-c color] -i filename"
            echo "-c: 设定水印颜色 (默认为灰色)"
            echo "\t b 蓝色"
            echo "\t G 灰色"
            echo "\t g 绿色"
            echo "\t o 橙色"
            echo "\t y 黄色"
            echo "-i: 设定输入文件名"
            echo
            exit;;
        i) FILENAME=${OPTARG}
            FILENAMENOEXT=${OPTARG%.*};;
        c) COLOR=${OPTARG};;
        *) echo "未知选项 - $opt";
            break;;
    esac
done

case $COLOR in
    b) CNAME="blue";;
    G) CNAME="gray";;

```

```
g) CNAME="green";;
o) CNAME="orange";;
y) CNAME="yellow";;
esac

pdftk $FILENAME background $(dirname $0)/watermark_${CNAME}.pdf output ${FILENAME%.*}.wm.pdf
```

YOUNG@BUPT