Part I
UNIX 简介

1 学习指导

引子

- UNIX 是什么？
- 这门课讲什么？
- 为什么学习 UNIX？
- 能够从这门课中得到什么？
- 如何学习 UNIX？
- 这门课如何安排？

UNIX

- UNIX 是一类操作系统
- 应用于所有类型的计算机
- 包含众多的变体
- 诞生于 1969 年，贝尔实验室

内容

- UNIX 的用户环境
  - 普通用户
  - 程序员
- UNIX 基础
  - 命令
  - 实用工具
- 文件系统
- shell

动机

- UNIX 是最经典的操作系统
- 广泛应用于各种领域
- 富有特色的命令和实用工具集合
- UNIX 是程序员的操作系统
- 拥有独特的技术魅力
- 深远的技术文化影响

目标

- 学习 UNIX 环境中的基本概念
- 使用 UNIX 的常用命令和使用工具
- 学习 shell（重定向，管道，文件名替换）
- 了解正则表达式
- 熟悉一种文本编辑工具

方法

- 注意总结规律
- 重视实验和练习
- 增加课外训练
- 勇于尝试

建立自己的练习环境

- 获得 Live CD 的镜像，并写到 CDROM 光盘上。
- 制作一只可以启动的 USB 盘。（或者移动硬盘）
- 在 Windows 操作系统上，安装虚拟机软件（vmware, VirtualPC, VirtualBox），在虚拟机上安装 Linux 系统。
- 在 Windows 分区上，安装一套独立的 Linux 系统。（整个 Linux 系统，作为一个 Windows 的文件。wubi）
- 调整硬盘的分区，在空闲的分区上，安装 Linux 操作系统。（双系统）
- 在整个硬盘上，安装 Linux 系统。（为了偶尔使用 Windows，可以在 Linux 系统上安装虚拟机，在虚拟机当中，安装 Windows 系统。）

发行版本的选择

Linux 就是一种 UNIX.
可以在 PC 机上运行，是最好的学习环境。
Ubuntu, Fedora, Debian, SuSe ……等等，分别是不同的 Linux 发行套件。
你还可以选择 FreeBSD 作为学习 UNIX 的环境。
参考：Linuxidc.com

计划

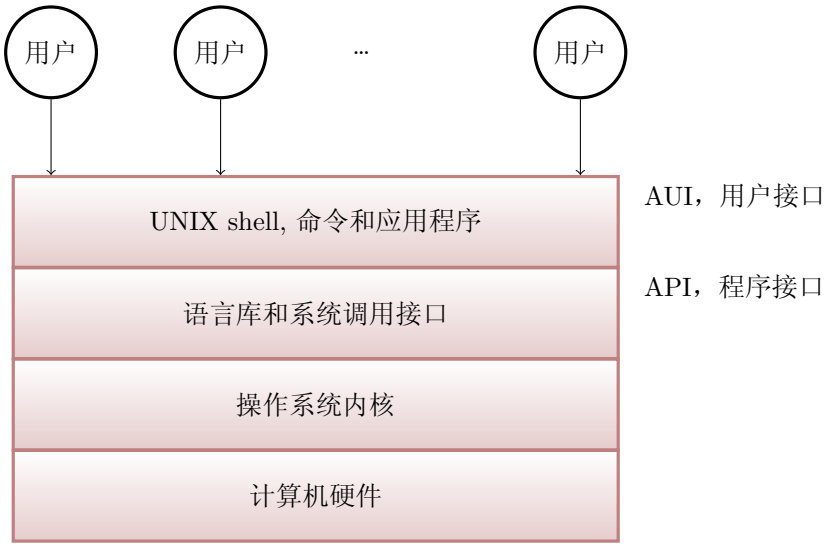
- 讲授：24 学时
- 实验：8 学时
- 课外练习：不少于 32 学时

2 操作系统

操作系统

操作系统是负责管理计算机系统软硬件资源，使用户简单、高效、公平、有序、安全地使用计算机系统的系统软件。

- 为用户和应用程序提供接口
- 管理并分配资源
- 加载并执行用户程序
- 两类主要的用户：一般用户，程序员
- 两个主要的接口：AUI，API
  - Application User's Interface
  - Application Programmer's Interface
- 四个层次
  - UNIX shell, 命令和应用程序
  - 语言库和系统调用接口
  - 操作系统内核
  - 计算机硬件



两种类型的用户界面：

- 字符用户界面，CUI
- 图形用户界面，GUI

CUI, Character User Interface
GUI, Graphical User Interface

Linux公社（LinuxIDC.com）于2006年9月25日注册并开通网站，Linux现在已经成为一种广受关注和支持的一种操作系统，IDC是互联网数据中心，LinuxIDC就是关于Linux的数据中心。

LinuxIDC.com提供包括Ubuntu，Fedora，SUSE技术，以及最新IT资讯等Linux专业类网站。

并被收录到Google 网页目录-计算机 > 软件 > 操作系统 > Linux 目录下。

Linux公社（LinuxIDC.com）设置了有一定影响力的Linux专题栏目。

包括:

Ubuntu专题

Fedora专题

RedHat专题

SUSE专题

红旗Linux专题

Android专题

UNIX 的特点

- 多用户
- 多任务
- 层次型文件系统
- I / O 操作与设备无关
- 独特的用户界面：shell
- 丰富而完善的系统工具
- 出色的服务功能
- 支持多种硬件平台（UNIX 是用 C 语言编写的）

UNIX 的基本结构

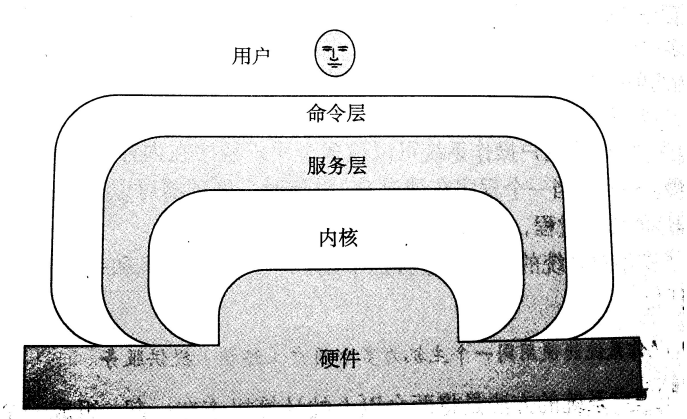


图 1.6 操作系统的分层结构

3 UNIX 的现状与历史

UNIX 家族

UNIX 是一类操作系统。有时，我们会把这类操作系统称为“类 UNIX 操作系统”。

- AIX (IBM)
- HP-UX (HP)
- Solaris (SUN, Oracle)
- FreeBSD, NetBSD, OpenBSD (Berkeley Software Distribution)
- GNU/Linux

UNIX 的简要历史

- 1969 年 诞生
- 1970 年 移植到 PDP-11/20 等系列的计算机上
- 1973 年 用高级语言 C 重写
- 1975 年 向教育机构提供
- 1978 年 功能趋于完善，出现两个重要分支
- 1983 system V，4.2 BSD
- 1988 标准化 POSIX.1
- 199x Linux

练习

在你自己的计算机上安装一套 Linux

Part II
UNIX 系统入门

4 会话

登录

UNIX 是一个多用户系统，使用 UNIX 系统必须首先登录。

- 系统在终端上显示登录提示：
  Login:
- 输入用户名，回车结束
- 输口令，回车结束
  （口令不会显示在屏幕上，连星号也不显示）
- 验证通过后，显示系统信息，并出现系统提示符。
  $ 或者 % 或者 #

退出

- 在系统提示符下，按 Ctrl-D 组合键（不用回车）
- 或者，输入 exit 命令（需要回车）

工作结束后及时退出系统的意义

- 防止他人冒用系统
- 停止计费
- 释放该用户占用的资源

会话（Session）

- 用户使用一次计算机的全过程
- 从登录开始
- 到退出结束

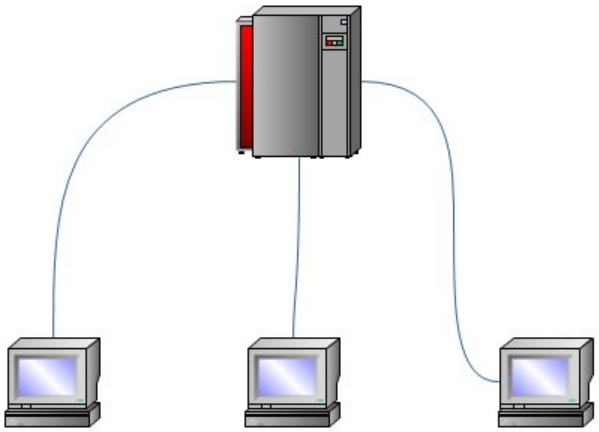
口令

- 修改口令的命令：
  passwd
- 普通用户：
  只能修改自己的口令
  口令太短或者太简单，系统可能拒绝修改
- 超级用户：
  有权修改任何人的口令
  对口令的指定，没有更多限制

用户

- 普通用户（一般用户）
  权限受限
- 超级用户（系统管理员，根用户，root 用户）
  权限不受限制
- 超级用户的提示符，与普通用户的提示符有区别
  #（井号）提示符，专门用于超级用户，提醒用户注意
- 切换用户身份的 UNIX 命令：
  su [< 用户名 >]
- 普通用户临时需要执行特权命令，可以使用：
  sudo < 命令 >

主机终端模式



Linux PC 上的虚拟终端

Linux 环境下的 12 个虚拟终端（控制台）：

- 字符终端
  Ctrl + Alt + F1
  Ctrl + Alt + F2
  ...
  Ctrl + Alt + F6
- 图形终端
  Ctrl + Alt + F7
  Ctrl + Alt + F8
  ...
  Ctrl + Alt + F12

终端和命令行

- UNIX 系统通过在终端上输入命令操作计算机。
- 所输入的命令，用回车符作为结束。这被称为命令行
- 命令行是逻辑上的行，不受屏幕宽度的限制
- 行尾反斜线符号“\”，表示命令行未完，后面还有续行

命令行格式

```
$  <命令名>  [-<选项>]  [<参数>]  <回车>
```

- 系统提示符　提示用户可以输入命令
- 命令名　　（大小写敏感）
- 选项　　（减号，或两个减号引导）
- 参数　　（多个参数用空格分隔）
- 回车符作为结束

命令行举例

```
$ ls <回车>
$ ls
  ls
```

```
  ls music
  ls -l music
```

```
  ls -l -a music
  ls -la music
```

```
  ls --format=long --all music
```

```
  ls --format=lo\
> ng --all music
```

5 简单的 UNIX 命令

[fragile] 几条简单的 UNIX 命令

- 时间和日期命令：

```
  date
```

- 用户信息显示：

```
  who
```

- 显示日历：

```
  cal
```

命令的变化：选项

```
  who -H
  who --heading
```

```
  cal 10 2012
  cal -m 2
  cal -3
  cal -3 -m 2
```

哪些选项？

6 帮助

[fragile] 联机手册

- UNIX 系统中一般都安装有用户手册的电子版。
  查看联机手册的命令：

```
man
```

- 用法

```
man  <命令名>
```

- 例子

```
man  cal
man  date
man  who
```

- 联机手册分为若干章节，可用以下命令进一步了解：

```
man  man
```

- 要退出联机帮助，按小写的“q”键。

UNIX 的联机手册分为若干“节”（section）：

```
1 Executable programs or shell commands
2 System calls (functions provided by the kernel)
3 Library calls (functions within program libraries)
4 Special files (usually found in /dev)
5 File formats and conventions eg /etc/passwd
6 Games
7 Miscellaneous (including macro  packages
     and  conventions), e.g. man(7), groff(7)
8 System administration commands (usually only for root)
9 Kernel routines [Non standard]
```

[fragile] 根据关键词查联机手册

- 不知道命令名，如何查找信息？
  可以使用关键字查找：

```
man  -k <keyword>
```

- 例子

```
man  -k calendar
man  -k line number
```

- 输出信息太多，难以辨认时，可采用管道和过滤命令筛选：

```
man  -k ling number  |  grep count
```

其他获得帮助的渠道

- 教学程序 learn（如果系统没有安装该程序，系统将会显示：`bash: learn: command not found` 之类的信息）
- 一般性帮助 help（系统之间不同）
- 联机文档：info
- apropos 命令
- whatis 命令
- 在命令之后使用帮助选项
  `--help`, `--usage`, `-?`, `-h` 等。

纠正错误：命令行编辑

- 删除光标左侧的一个字符
  [Backspace] 键，或者 [Ctrl − h] 组合键
- 删除一整行
  [Ctrl − u] 组合键
  （删除光标左侧的所有内容）
- 其他更多的编辑命令键
  [Ctrl − w]，[Ctrl − k]，…
- 提示：不要强记这些键的功能，注意掌握概念。

小结

- 登录与退出
- 命令行与命令行编辑
- 简单的 UNIX 命令
  who, date, cal
- 帮助

Part III
UNIX 文件系统基础

情境

Tom 与 David 合作完成一个项目。
Tom 写了一个便条，保存成文件，给 David 看。

1. 文件的名字
2. 保存在什么地方
3. 文件的格式
4. 如何查看
5. 如何修改、改名、移动、删除

7 文件和目录

文件和目录

- 文件的一般概念：
  文件是对外存储器上相关数据集合的抽象。
- UNIX 扩展了文件的概念：
  In UNIX Everything is a File
- 目录系统提供了组织文件的结构：
  目录也是一种文件

文件类型

- 普通文件　Regular Files
- 目录文件　Directory Fies
- 特殊文件　Special Files
  - 设备
  - 管道
  - 套接字

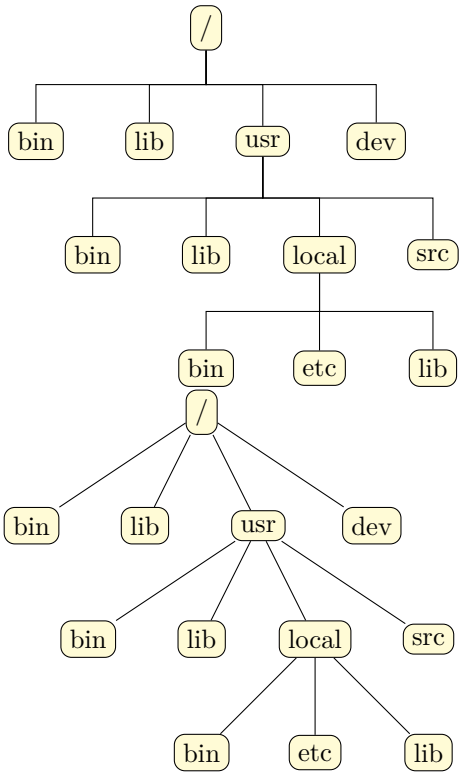
文件（目录）名

- 文件名可以由字母、数字和其他字符组成
- 某些字符有特别含义，尽量避免使用
- “/”字符不能用于一般文件名中
- 根目录的名字只能是“/”

层次目录

- 根目录
- 子目录
- 父目录
- 树型结构

例子：Linux 的目录结构



examtree 目录树举例

用户主目 The Home Directory

作为一个“多用户”的操作系统，每个用户拥有各自独立的“主目录”，如:

```
/home/david,
/home/tom,
/home/jack,
/home/mary
```

一般情况下，普通用户没有访问其他用户主目录的权限。

工作目录 The Working Directory

用户的很多操作，都与一个特定的目录相关联。

工作目录又称“当前目录”

可以用 `cd` 命令把工作目录调整到目录树上的任一位置。（需要对应的权限）

8 路径

路径和路径名

路径用来标识文件的逻辑位置。

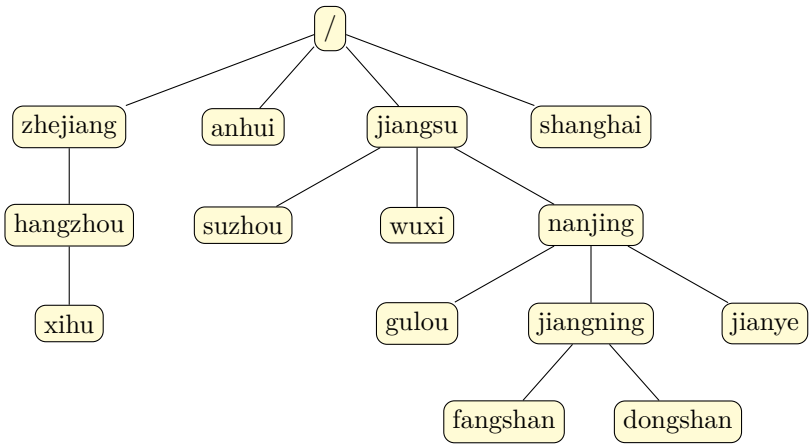
路径由一连串的目录名组成，中间由“/”分隔。

（与 Windows 下的分隔符号的方向相反）

路径的文件或目录名，称为文件或目录的“路径名”。

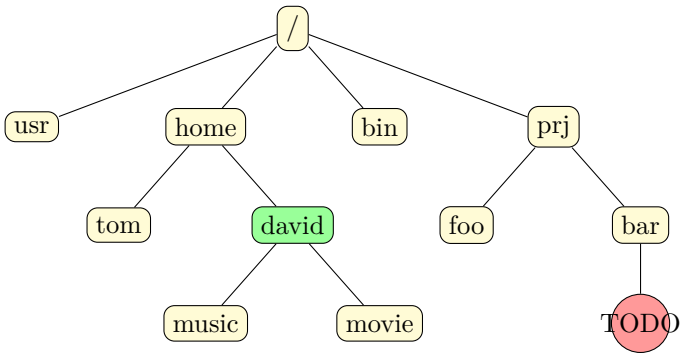
- 绝对路径名
- 相对路径名

目录树的例子



linuxtree<2> 实际的目录

一个假想的目录树



9 操作

针对目录的操作

- 显示工作目录的目录名：pwd
- 改变工作目录：cd
- 创建目录：mkdir
- 删除目录：rmdir
- 目录列表：ls

小结

- 目录和文件的概念
- 路径
- 针对目录的操作

Part IV
UNIX 文件系统基础（续）

10 ls 命令

ls 命令的各种形式

```
ls
```

```
ls  /prj/foo
ls  /prj/foo/TODO
ls  ../prj/foo
ls  bar
```

```
cd  /prj/foo
ls
```

```
ls  -l -a  music
ls  -la  music
```

隐藏文件

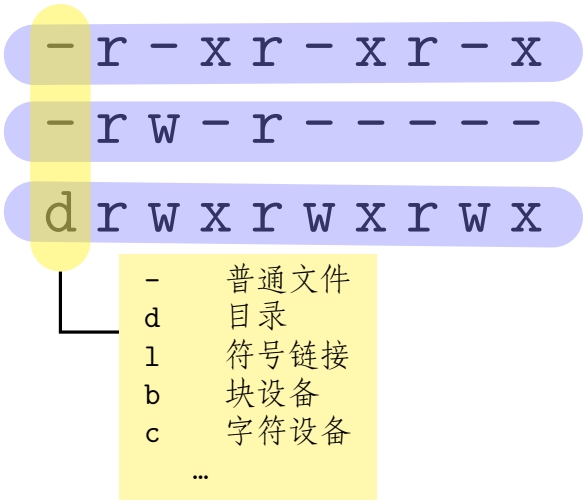
- 特殊的文件名：“.”开头的文件
- 每个目录中都有的两个特殊文件：. 和..
- 隐藏文件不是秘密
- 隐藏文件的应用场合
- 以上也适用于隐藏目录

文件的详细信息

- 文件类型
- 文件访问模式（权限）
- 链接数
- 属主
- 组
- 大小
- 上次修改日期
- 文件名

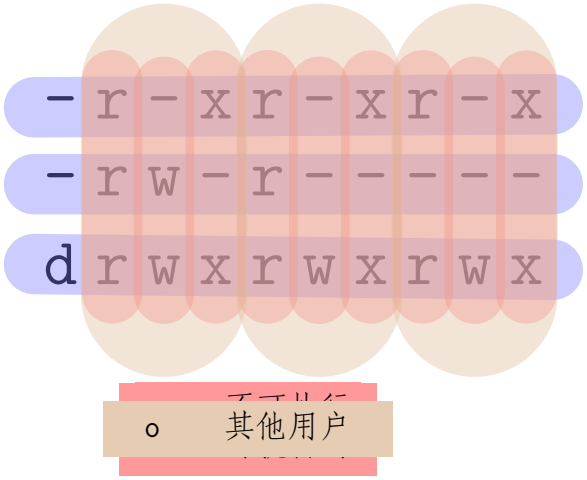
11 文件类型

文件类型



12 权限

文件访问模式（权限）



改变文件或目录的访问模式

```
chmod 777 foo
chmod 755 bar
chmod 600 blabla
```

```
chmod g+w foo
chmod go-x foo
```

```
chmod u=rw,go-x foo
chmod a=rwx foo
```

13 操作示例

针对文件的其他操作

- 显示文件内容：cat
  原意 concatenate，连接。
- 打印命令：lp, lpr
- 取消打印：cancel
- 打印机状态：lpstat
- 删除文件：rm
  注意：一定要了解删除文件的风险！

回顾假想情境

Tom 与 David 合作完成一个项目。
Tom 写了一个便条，保存成文件，给 David 看。

Tom 的操作：	David 的操作：
1. 登录	1. 登录
2. 新建目录	2. 找到文件
3. 新建文件	3. 查看文件内容
4. 撰写内容	4. 删除文件
5. 退出	5. 退出

操作示范

- 以超级用户身份建立两个普通用户
- 模拟 Tom 的操作过程
- 模拟 David 的操作过程
- 留意未曾学习过的命令
- 总结整个操作过程涉及的概念

Part V
深入 UNIX 文件系统

文件操作

- 创建文件
- 复制文件
- 移动文件位置
- 文件改名
- 删除文件

14 创建文件

[fragile] 创建文件：初识 vi 编辑器

- 用 vi 编辑器新建一个文件，命令如下

```
vi  <文件名>
```

- 注意编辑器的初始画面
- 按 i 键进入文本模式
- 输入文本
- 按 Esc 键回到命令模式
- 使用以下命令，保存文件并退出 vi 编辑器

```
:wq  <回车>
或者
ZZ
```

15 复制、移动、删除文件

[fragile] 复制文件

- cp 命令的一般形式

```
cp  originalfile  copiedfile
```

- cp 命令的风险 如已经存在同名的目标文件，将会被覆盖！

  使用 cp 命令的选项，可以降低风险。另外的选项，增加 cp 命令的灵活性。

  –b 备份目标文件

  –i 覆盖确认提示

–r 连同目录复制

- 问题：修改其中一个文件的内容，是否影响另外一个复本的内容？

[fragile] 移动文件

- mv 命令的一般形式

```
mv  /path/to/old_dir/filename  /path/to/new_dir
```

- mv 命令的风险和预防
- mv 命令的实际动作和复制不同
- mv 命令用于目录时，整个子目录连同下级目录一起移动

[fragile] 文件改名

- 文件改名（重命名）仍然使用 mv 命令

```
mv  oldname  newname
```

- 移动文件的位置，和对文件重新命名本质相同
- mv 并不复制文件内容，只修改指向文件的索引
- 猜想：对于比较大的文件，执行 cp 和 mv 命令的时间，哪个更长？

[fragile] 删除文件

- rm 命令的一般形式

```
rm  somefile
```

- 删除命令的风险 警告：没有所谓“恢复删除”的命令。所以使用删除命令要仔细！ 使用 rm -i 选项，可以减少发生灾难的机会。

```
rm  -i  somefile
```

[fragile] 删除目录

- 回顾：rmdir 命令只能删除空目录
- 一般的 rm 命令不能删除目录
- rm 加上 –r 选项，可以删除目录（空或非空）

```
rm  -r  somedir
```

- 术语：递归删除
- 一个强大而危险的命令：

```
rm  -rf  /some/important/dir
```

- 这个最危险的命令，也是被使用最为频繁的命令。

16 链接

[fragile] 链接

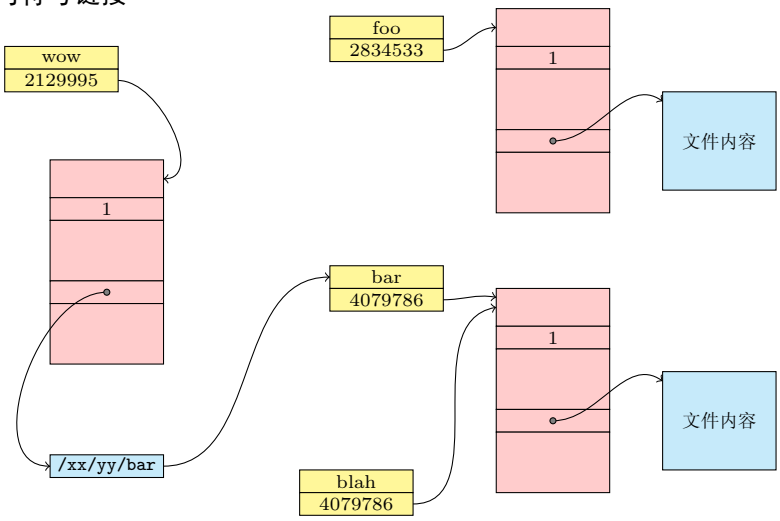
- 链接是什么？ 指向文件的一个名字。
- 链接有两类：
  - 链接（硬链接）
  - 符号链接
- 创建链接

```
ln  existed_file  new_name
```

- 创建符号链接

```
ln  -s  existed_file  another_name
```

链接与符号链接



链接操作演示

- 建立链接的操作
- 更改其中一个文件的内容，是否影响链接对应的文件？
- 文件信息中链接数的变化
- i-node 信息
- 符号链接与硬链接的区别

Part VI
文件内容操作

17 查看文件内容

查看文件内容

- 使用文本编辑器查看
- 使用 cat 命令查看
- 几种分页查看命令 pg, more, less 命令
- 查看文件的局部内容 head, tail, cut 命令

[fragile] 使用 vi 编辑器查看文件

- 以只读方式启动 vi 编辑器

```
vi  -R  filename
```

- 等效的命令

```
view   filename
```

- 使用 vi 编辑器的翻页命令前后查看
- 退出

```
:q   或者  :q!  或者   ZQ
```

cat 命令

cat 将文件内容发送至标准输出

```
cat  file
```

长文件的滚屏控制

Ctrl-S 暂停

Ctrl-Q 恢复

cat 查看多个文件内容

```
cat  fileA  fileB  fileC
```

分页查看工具

pg

```
pg  longfile
```

more

```
more  longfile
```

less

```
less  longfile
```

h 帮助
q 退出
回车 下一行
空格 下一屏
b 回滚一屏

查看文件局部内容

- 查看头部
- 查看尾部
- 查看中部
- 查看特定的列

[fragile]head 命令

- 缺省参数，显示文件的头 10 行

```
head  longfile
```

- 显示文件的前 26 行

```
head  -n 26  longfile
```

- 从头开始显示文件，除了尾部最后的 9 行

```
head  -n -9  longfile
```

-

[fragile]tail 命令

- 缺省参数，显示文件的末尾 10 行

```
tail  longfile
```

- 显示文件的末尾 26 行

```
tail  -n 26  longfile
```

- 跳过前部，从第 9 行开始显示文件

```
tail  -n +9  longfile
```

- [fragile] 显示文件中间的部分
- 没有一个专用的命令
- 联合使用 head 和 tail 命令
- 显示文件的第 24 行到第 26 行

```
head  -n 26  longfile  |  tail  -n 3
```

- 另一种方法

```
tail  -n +24  longfile  |  head  -n 3
```

- UNIX 的哲学思想之一：
  Small tools cooperate with standardized inputs and outputs.

管道：UNIX 系统中，一种进程通信的重要机制！

查看文件的某些列

cut 分离出文件中指定的列

```
cut  -f1,7  -d:  /etc/passwd
cut  -f2-5  /path/to/file
```

18 合并文件内容

合并文件内容

- 多个文件连结到一起
- 文件按行合并
- 输出重定向

[fragile]cat 命令

- 用 cat 命令连结多个文件 concatenate：连结
- 例子

```
cat  fileA  fileB  fileC  >  file_all_in_one
```

- 符号“>”表示输出重定向
- 符号“>>”表示输出重定向，追加方式

```
cat  fileA   >  file_all_in_one
cat  fileB  >>  file_all_in_one
cat  fileC  >>  file_all_in_one
```

按行合并文件

paste 命令

```
paste  one  two
```

将两个文件按行合并后显示。

paste 命令输出到文件

```
paste  one  two  >  one_and_two_side_by_side
```

将两个文件按行合并生成新的文件。

19 同样功能多种方法

复制文件的不同方法

方法一

```
cp  fileA  fileB
```

方法二

```
cat  fileA  >  fileB
```

文件改名的不同方法

方法一

```
mv  fileA  fileB
```

方法二

```
cp  fileA  fileB
rm  fileA
```

方法三

```
ln  fileA  fileB
rm  fileA
```

综合举例

合并两个文件

Part VII
vi 编辑器初步

20 启动 vi 的命令行

回顾：用 vi 创建新文件

- 用 vi 编辑器新建一个文件，命令如下

```
vi  <文件名>
```

- 按 i 键进入文本模式
- 输入文本
- 按 Esc 键回到命令模式
- 使用以下命令，保存文件并退出 vi 编辑器

```
:wq  <回车>
或者
ZZ
```

回顾：用 vi 查看文件

- 以只读方式启动 vi 编辑器

```
vi  -R  filename
```

- 使用 vi 编辑器的翻页命令前后查看

```
即将学习
```

- 退出

```
:q   或者  :q!  或者   ZQ
```

21 文本编辑器

文本编辑器

编辑器是创建修改文本文件的基本工具。
编辑器所处理的目标，是纯文本文件（Plain Text File）。

两种编辑器的基本类型：

- 行编辑器（ed, ex）
- 全屏编辑器（vi, emacs）

注意区分普通编辑器，具有排版与格式控制功能的字处理器。

UNIX 系统上的各种编辑器

- ed
- ex
- vi
- emacs
- pico
- nano
- gedit
- kate
- kwrite
- leafpad

[fragile]vi 编辑器

- 每一种 UNIX 系统的缺省配置
- 支持几乎所有的操作系统（UNIX/Mac/Windows）
- 可以工作在字符方式下
- 对初学者来说，有点“不自然”
- 高效、强大
- 有多种变种程序
- vim 是一种改进的 vi

[fragile]vi 的启动与退出

- 启动 vi，命令如下

```
vi  <文件名>
```

- 各种编辑操作
- 保存文件并退出 vi 编辑器

```
:wq  <回车>
或者
ZZ
```

[fragile] 关于保存和退出的变化

- 编辑过程中，保存文件，但不退出 vi

```
:w  <回车>
```

- 未做改动，或者保存后未再改动，退出 vi

```
:q  <回车>
```

- 保存文件并退出 vi 编辑器

```
:wq  <回车>
```

- 有改动，但不想保存，放弃所做的修改，退出 vi

```
:q!  <回车>
```

编辑发生在缓冲区中，未经保存，文件内容不会变化

[fragile] 关于启动 vi 的变化

- 通常启动 vi 的命令如下

```
vi  <文件名>
```

- 启动 vi 时不带文件名参数之后...
- 在 vi 内打开原有文件

```
:e  <文件名>
```

- 或者在保存文件时，指定文件名

```
:w   <文件名>
:w!  <文件名>          强制覆盖已有的同名文件
```

- 也可以

```
:wq   <文件名>
:wq!  <文件名>
```

22 基本编辑命令

常用的编辑操作

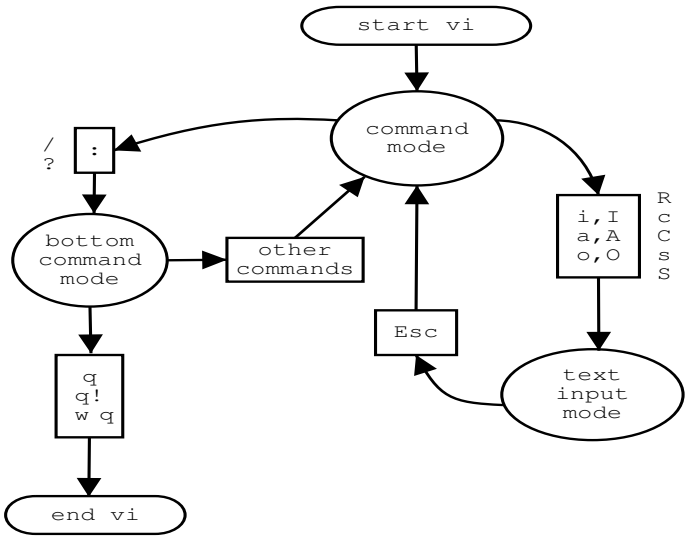
功能	命令
光标移动	h, j, k, l
增	i, I, a, A, o, O
删	x, X
改	r, R

vi 模式及切换

vi 编辑器是一种“有模式”的编辑器。

vi 的两种基本工作模式：

- 刚启动的时候，vi 自动处于“命令模式”。所有的按键均被理解成命令。
- i （或者 I a A R o O ... ）命令，可以令 vi 进入“文本模式”。
- Esc 键可以令 vi 由“文本模式”回到“命令模式”。
- 如果记不请当前模式，可以按两次 Esc 键，确保 vi 回到命令模式。



按键/操作	作用
Esc	回到命令模式
i, I, a, A, o, O, …	进入文本模式
另外一些命令	不影响模式

移动光标的更多手段

- 按单词移动光标: w, b, e, ge
- 移动到行首与行末：0（数字零）, ^, $
- 移动到上一段、下一段：{, }
- 本行内移动到特定字符：fx，tx，Fx，Tx （x 代表特定字符） 分号和逗号，用来重复这样的移动操作

编辑动作命令

编辑命令	功能
d	删除
y	复制（yank）
（小写）p	粘贴（到光标右/下）
（大写）P	粘贴（到光标左/上）
c	更改
s	替换

编辑范围指示

$	到行尾
0（数字零）	到行首
e	到词尾
w	到下一词首（包含词后的空）
b	（退）到词首
其他移动光标的命令	表示从当前位置到目标位置范围

动作 + 范围

- d$ 删除到行尾
- d0 删除到行首
- ce 改变到词尾（删除到词尾，并进入文本输入模式）
- yw 复制到下一词首（包含本词后的空）
- yb 向后退方向复制到词首

动作字符重复

- dd 删除一行
- yy 复制一行
- cc 改变一行

不指定域，而将动作命令字符重复一次，是代表对当前整行的操作。

数字 + 动作 + 范围

- 3dd 删除三行
- 5yw 复制五个词
- 4p 把临时寄存器中的内容在当前位置后，复制 4 次
- 12P 把临时寄存器中的内容在当前位置前，复制 12 次

数字用于光标移动命令之前

- 23l 向右移动 23 个字符
- 5j 向下移动 5 行
- 7w 向右移动 7 个单词
- 4fe 本行之内，向右移到第 4 个字母 e 上

动作 + 数字 + 范围

- c3w 更改 3 个单词
- d2d 删除两行（本行及下方一行）
- y2w 复制 2 个单词
- 5p 临时寄存器的内容在右方或本行下方复制 2 次
- 3d2d 删除 6 行

搜索与重复操作

- 搜索字符串

`/string`	向前搜索
`?string`	向后搜索
`*`	向前搜索光标处的单词
`#`	向后搜索光标处的单词
`n`	下一匹配处
`N`	上一匹配处

- 重复上一次的操作

  `.` （句点）

  （技巧：搜索操作与重复操作相结合的应用，可以提高编辑效果）

undo 机制

`u`	取消上一次的改动（回退一步）
`Ctrl-R`	撤销一次回退操作
`U`	一次性取消最近在本行上的全部改动

在线帮助

Linux 系统之下所配备的 vim 是 vi 的改进版本。可以使用命令 :help 获得在线帮助。

联机教程 vimtutor，声称在 25–30 分钟的时间内教会你初步应用。建议在实验课上练习一次。

直接输入

```
$ vimtutor [Return]
```

即可得到学习指导。该学习指导，可能是中文，也可能是英文，取决于当前的操作系统环境。

如果当前的环境是中文，而你又想得到英文的指导，请输入:

```
$ LANG=C vimtutor [Return]
```

Part VIII
深入 vi 编辑器

23 编辑命令的组合

编辑命令的组合

组合基本的编辑命令得到各种编辑动作

- 增加
- 删除
- 修改
- 移动
- 复制

举例

Backup files. CE just loves
files, and leaves them laying
everywhere. Keep in mind, th
of this editor is for beginne
these backup files may come i
some poor person who just era
entire document, and saved as
can turn off this behavior vi
personal defaults file.

CE requires a special keymap
terminal type that will be us
reason I chose to use my own
code, instead of using termin
simply, ...

编辑命令的组合

移动行

dd 删除一行
移动到目标处 p 贴上去

复制行

yy 复制一行到临时缓冲区
移动到目标处 p 贴上去

由 dd 或 yy 复制到临时缓冲区的内容可以多次被使用

编辑命令的组合

移动单词的位置

dw 删除一个词 ⇒ 光标移动到恰当位置 ⇒ p 或 P 贴上去

复制词

yw 复制一个词 ⇒ 光标移动到恰当位置 ⇒ p 或 P 贴上去

更大范围的光标移动

- 到达指定行：gg, 1G, G, 58G
- 滚屏：Ctrl-U Ctrl-D （半屏）Ctrl-F Ctrl-B （全屏）
- 移动到配对的括号: %
- 按句子移动：(,)
- 按段落移动：{, }
- 搜索：使用 / 或者 ? 搜索命令
- 搜索：使用 * 或者 # 搜索光标所在的单词

24 编号的缓冲区

vi 的缓冲区

- 无名缓冲区
  `d c s x y` 等命令删去或复制的内容自动放入此缓冲区
- 数字缓冲区
  `1 -- 9`，9 个数字编号的缓冲区形成 `FIFO` 缓冲队列
- 字母缓冲区
  `a -- z`，26 个字母编号的缓冲区内容不滚动
- 使用双引号作为引用缓冲区的标志
  `"1 "2 "3 …"9 ; "a "b "c …"z`

使用 vi 的缓冲区

- 相邻的两行复制到缓冲区 a 当中
  "a2yy 或"ay2y
- 把缓冲区 b 当中的内容，粘贴到光标之后
  "bp
- 把缓冲区 4 当中的内容粘贴到光标之前
  "4P
- 当前位置的 3 个单词删去，存放到缓冲区 c 之中，打算以后再用
  "cd3w

25 多重文件编辑

多重文件编辑

在启动 vi 时，如果以多个文件作为参数，就可以连续编辑多个文件，而不必重新启动 vi。

- :n 切换到下一个文件
- :wn 保存当前文件，并切换到下一个文件
- :n! 放弃当前文件的改变，并切换到下一个文件
- :N 切换到上一个文件
- :ar 列出在编辑队列中，有哪些文件
- :e 开始编辑一个新文件
- :sp 以多窗口的方式打开一个编辑窗
- :wq 保存并结束一个文件
- :wqa 保存并结束所有文件

使用 ex 兼容命令

在全文范围内批量修改文本时，可以使用冒号开头的 s 命令。

- :%s/oldstr/newstr/g
  将所有的 oldstr 替换为 newstr。
- :%s/oldstr/newstr/gc
  将所有的 oldstr 替换为 newstr。交互确认。
- :s/oldstr/newstr/g
  将本行所有的 oldstr 替换为 newstr。
- :7,23s/oldstr/newstr/g
  将 7 到 23 行所有的 oldstr 替换为 newstr。
- :.,$s/oldstr/newstr/
  从当前行到最后一行，所有行的第一个 oldstr 替换为 newstr。

缩写和宏

```
:ab i18n internationalization
```

ab 意思是 abbreviate. 取消缩写的命令为：unab.

```
:unab i18n
```

```
:map Q kyypj
```

要取消指定的宏，使用 unmap 命令：

```
:unmap Q
```

vi 的外部命令使用

- 重新得到 shell 环境，而不退出 vi

  :shell 进入，exit 回到 vi 环境。

- 在 vi 内执行 UNIX 命令

  :!< 命令名 >

- 让命令的结果影响正在编辑的文件

  把命令的结果读到正在编辑的文件中

  例　　:r !cal 1 2005

  又例　:%!sort 对当前编辑缓冲区的内容按行排序。

定制 vi 编辑器

vi 有多种参数被改变，以此改变 vi 编辑器的工作方式。

例如：在使用 / 和 ? 命令搜索字符串时，对于大小写是否区分，可由 ignorecase 参数控制。

- :set ignorecase 不区分大小写
- :set noignorecase 区分大小写

更简洁的方式是　:set ic 和　:set noic

定制 vi 编辑器 1. 了解有哪些选项

- :set all
- :set
- :set < 选项名 >?

定制 vi 编辑器 2. 设定选项的方式

- 布尔选项　　:set xxx, :set noxxx

  例如，:set ignorecase, :set noignorecase

- 数字选项　　:set xxx=9

  例如，:set textwidth=78

- 字符串选项　:set xxx=str

  例如，:set encoding=utf8

定制 vi 编辑器 3. 常用选项的解释

ic	ignorecase	ignore upper/lower case when searching
is	incsearch	show partial matches for a search phrase
hls	hlsearch	highlight all matching phrases
nu	number	print the line number in front of each line

定制 vi 编辑器 4. 使用资源文件设定 vi 选项

vi 资源文件名 .exrc（一般放在用户主目录中）
（Linux 下的 vim 版本所使用的资源文件名为 .vimrc）
处理中文的用户需要设定:
set encoding=prc

26 程序员的编辑器

程序员的特殊需要

- 自动缩进
- 括号匹配
- 函数嵌套
- 变量定义

重点命令

- :set ai 设置自动缩进 autoindent
- Ctrl-T, Ctr-D 输入时调整缩进
- <, > 调整代码块的缩进
- % 查找匹配括号
- Ctrl-], Ctrl-T 追踪调用层次（要 ctags 程序配合）

几点忠告

- 没有可能，也没有必要把关于 vi 编辑器的一切细节全部掌握。
- 不是学会以后使用，而是使用当中学习。
- 只需极少的命令，即可开始使用。绝大多数命令是可以替代的，它们仅仅为了提高效率而存在。

阅读材料：
Seven habits of effective text editing，Written by Bram Moolenaar
网上资源:
The Vi Lovers Home Page, http://thomer.com/vi/vi.html

Part IX
UNIX shell

shell 是什么？

A Unix shell is a command-line interpreter and script host that provides a traditional user interface for the Unix operating system and for Unix-like systems.

关键的问题：shell 是不是操作系统的一部分？

- 本质上，shell 是 UNIX 系统上的一个应用程序
- 实际上，shell 已成为 UNIX 系统的一个组成部分

shell 种类

- Bourne Shell 家族
  - Bourne Shell (sh)
  - Korn Shell (ksh)
  - Bourne Again Shell (bash)
- C Shell 家族
  - C Shell (csh)
  - TC Shell (tcsh)

用户可以选择不同的 shell

- 登录时的缺省 shell
- 临时换用其他 shell：直接输入其他 shell 的程序名
- 改变缺省 shell：用 chsh 命令改变用户的缺省 shell
- 不同的 shell 基本功能相似，命令语法存在轻微的差异

shell 的功能

1. 命令行执行
2. 文件名替换
3. I/O 重定向
4. 管道
5. 环境控制
6. 后台处理
7. 脚本支持

27 重定向

标准输入，标准输出，错误输出

绝大多数 UNIX 程序，被设计成这样的工作方式：

- 缺省情况下，从标准输入设备那里读取输入数据
- 处理之后的结果，缺省情况下，发给标准输出设备
- 处理过程中的错误信息，发送到标准错误输出设备

- 标准输入（stdin）：缺省的程序输入文件
- 标准输出（stdout）：缺省的程序输出文件
- 标准错误（stderr）：接收程序出错信息的缺省文件

重定向

- > 输出重定向符号
- >> 输出重定向符号（添加方式）
- < 输入重定向
- << 输入重定向（脚本文件的嵌入文档）

再次考察不带任何参数的 cat 命令：

```
cat
```

我们已经知道，下面命令的作用：

```
cat one > two
```

猜一猜以下命令的作用：

```
cat > two
```

再猜一猜以下命令的作用：

```
cat < one
```

再看：`cat < one > two`

28 管道

管道

管道是进程之间的一种通信机制。

> UNIX 的设计思想之一是：复杂的任务，通过多个简单的功能模块合作解决。
>
> 所以，在 UNIX 当中，有许多命令，仅仅完成相当简单的工作。

字数统计命令 wc

计算指定文件中，文本的行数、单词数、字符数。例：

```
wc report
```

结果显示为：

```
4 30 155 report
```

如果在命令之后，不跟随指定的文件名，则默认为由标准输入设备获得输入。

管道操作的应用实例

想了解，目录/etc 下共有多少个文件，如何做？

```
ls -1 /etc
```

命令的输出有多少行，文件（连同子目录）就有多少个。

方法一

```
ls -1 /etc > temp
wc temp
rm temp
```

方法二

```
ls -1 /etc | wc -l
```

方法二，无需产生中间文件，且管道两侧的命令并发执行。

其他的例子：当前小时数？

```
date --rfc-3339=second | cut -c 12-13
```

```
date --rfc-3339=second | cut -c 12-13 > hournow
```

29 文件名替换

文件名替换

设想，当前工作目录下，有这些文件：

```
letter      article     report1     report2
reportm     report99    repeat      reprint
reply       meat        at
```

如果你想删除所有以“report”开头的文件，更简洁的方法是什么？

文件名替换元字符 通配符

许多 UNIX 命令用文件名作为其参数。当需要对一批文件进行同一类的操作时，逐个指明文件名是不方便的。

UNIX 允许使用通配符定义一种模式，用以指代所有与此模式匹配的文件。

- `?` 匹配任意单个字符
- `*` 匹配 0 到多个字符构成的串
- `[list]` 匹配所列出的字符
- `[!list]` 匹配未列出的字符

使用文件通配符的例子

- `a*`
- `*at`
- `report?`
- `report??`
- `rep??t`
- `rep*t`
- `[bdt]ark`
- `file[1-4]`
- `[!a-z]*`
- `*`

单独使用 `*` 并不能匹配那些由 `.` 开头的隐藏文件。

用 `.*` 匹配所有的隐藏文件。

显示信息的命令 echo

例：

```
echo hello there
```

将在屏幕上显示：

```
hello there
```

为了要知道 `rep*t` 将被 shell 使用文件名替换功能，展开成哪些文件，可以使用命令：

```
echo rep*t
```

Linux公社（LinuxIDC.com）于2006年9月25日注册并开通网站，Linux现在已经成为一种广受关注和支持的一种操作系统，IDC是互联网数据中心，LinuxIDC就是关于Linux的数据中心。

LinuxIDC.com提供包括Ubuntu，Fedora，SUSE技术，以及最新IT资讯等Linux专业类网站。

并被收录到Google 网页目录-计算机 > 软件 > 操作系统 > Linux 目录下。

Linux公社（LinuxIDC.com）设置了有一定影响力的Linux专题栏目。

包括:

Ubuntu专题

Fedora专题

RedHat专题

SUSE专题

红旗Linux专题

Android专题

警告

如果把 rm report* 误打成 rm report * 会有什么效果？

文件查找命令：find

find 命令在目录树中查找文件或子目录的位置。
可以使用文件通配符作为查找的依据。
find 命令的语法格式
`find 路径名 搜索选项 动作选项`
两个 find 命令的例子：

```
find . -name "*.c" -print
find . -name "*.bak" -exec rm {} \;
```

30 shell 元字符

shell 元字符

有一些字符，被 shell 赋予特殊的含义，从而失去它们原有的字面意义。
已经学习过的 shell 元字符：

```
*     ?      [      ]

>     <      |

空格符        回车符
```

转义

请看命令：

```
echo This is a test file > testfile
```

问题：上例中的 > 号被 shell 解释成重定向符号。若实际打算在屏幕上显示这个符号，如何实现？

```
echo This is a test file \> testfile
```

问题：如果实际打算在屏幕上显示\> 这两个符号，又如何实现？

```
echo This is a test file \\\> testfile
```

问题：如果想实际显示\\\> 这样连续的四个符号呢？

引用

转义：

```
echo This is a test file \> testfile
```

引用：

```
echo "This is a test file > testfile"
```

或者：

```
echo 'This is a test file > testfile'
```

问题：如果想实际显示\\\> 这样连续的四个符号呢？

```
echo 'This is a test file \\\\> testfile'
```

“转义”的双重含义

1. 原来的符号（元字符）被 shell 解释成另外的含义，通过转义，恢复其字面含义。

   \ >，不再表示输出重定向的符号，就表示大于号本身。

   命令行很长，在一行当中写不下，需要换行。直接输入回车符 [Return]，shell 将会把它解释成一行命令的结束。

   可以在一行的末尾输入 \ 之后再回车开始一个新行，继续输入命令。

2. 原来的符号，仅仅是字面的含义，通过转义，代表另一种特殊的含义。

   如：“\a”代表响铃，“\n”代表换行，等等。

   ```
   echo -e Hi, "\n" this is a test. [Return]
   ```

引用的用法

对比：

```
echo 1                  2
```

和

```
echo "1                  2"
```

引号保留了中间的空格。

未用引号括住的空格，被认为是参数之间的分隔；使用引号之后，引号对之间的内容，被视为“引用”，shell 不再另行解释。

关于单引号和双引号之间的细微差别，请在后面留意。

对比：

```
echo "This is $HOME"
echo 'This is $HOME'
```

对比：

```
echo *
echo "*"
echo '*'
```

双引号对括住的 shell 变量引用，被 shell 展开。

单引号对所括住的一切内容，均严格保持原貌。

如何把引号本身显示出来？

```
echo '"UNIX" is a trademark'
echo "\"UNIX\"is a trademark"
```

小结

- shell 是操作系统中，负责用户接口的部分。
- 输入输出重定向，可以改变原始数据的来源，以及处理结果的输出目的地。
- 管道把进程之间的数据流连接起来。
- 文件名替换功能，提供通配符，方便操作。
- shell 使用转义和引用，处理具有特殊含义的元字符。

Part X
过滤程序和正则表达式

31 过滤程序

部分常用过滤命令

UNIX 中的某些命令，可以对其他程序的输出，做进一步的加工处理。这部分命令，可以被认为具有“过滤”功能。

已经学习过的，具有过滤作用的命令有：

```
head, tail, cut
wc, more, less
```

分离输出：tee 命令

重定向命令，可以将 UNIX 命令的输出，写入到一个文件当中。

例如：

```
ls -c /home > dir.list
```

希望又显示结果，又把结果存入文件，如何实现？

tee 命令可以解决：

```
ls -c /home | tee dir.list
```

排序命令：sort

sort 命令既可以对文件的内容排序，又可以对其他命令的输出排序。排序之后的内容，可以写入文件，也可送标准输出。

注意以下有用的排序选项：

-b 忽略前导空格
-d 用字典顺序排序
-f 忽略大小写的区别
-n 数字以数值大小排序
-r 逆序（缺省为升序）
-o 输出改为指定文件，而非标准输出设备

某些文件是“结构性的”。

每一行为一个记录，每个记录含若干字段，字段之间由空格或制表符（Tab）分隔。

请看电话号码文件的例子。

考虑如何按文件的大小，显示一个目录中的文件。怎样用 sort 命令作为过滤，来实现？

滤除重复行：uniq

经过 uniq 的过滤，相邻的重复行，仅仅显示一次。

uniq 还有滤掉空行的功能。

问题：如何滤掉所有的重复的行（包括那些不相邻的？）

搜索命令：grep

grep 命令可以从一个或一批文件之内容中，搜索含有特定字符串的行。

例：一大堆文件中，哪一个是我写给 Tom 的信？

例：在一份电话号码簿文件中查，38976542 是谁的电话？

更复杂的查找，需要使用“正则表达式”，来描述被搜索的字符串。

regex: Regular Expression (Global Regular Expression Print)

最简单的 grep 命令

grep ＜字符串模式＞ ＜接受搜索的文件名＞

以上命令执行后，如果文件中含有与“字符串模式”相匹配的行，该行就会被显示出来。

例：

grep monitor /etc/x11/XF86Config

其中的字符串模式，最好放到引号对中间。（为什么？）

grep 命令的常用选项

-c 显示每个文件中匹配的行数
-i 不分大小写
-l 显示具有相匹配行的文件名，而不是匹配行本身
-n 显示匹配行的行号
-v 显示不匹配的行（匹配的行不显示）
-r 搜索范围包含以下各级子目录中的文件

应用实例

1. 我写的一封信，只记得是写给 David 的，文件名忘记了，保存在什么地方也忘记了，但一定在我本人的主目录或者下级的某个子目录中，文件的内容里，一定有 David 这个词。如何找到那份文件？

先进入用户的主目录，再执行：

```
grep -ir david .
```

2. 我在某一个幻灯片文件中，谈到有关 more 命令的格式问题？

```
grep 'more' /mnt/sda1/unixabc/*.tex
```

输出的内容太多，可以再筛选一次：

```
grep 'more' /mnt/sda1/unixabc/*.tex | grep ' 格式'
```

3. 哪一个命令可以用来对软盘格式化？

```
man -k disk
```

，输出结果太多，不知所云！

```
man -k disk | grep '(1'
```

在 grep 命令中应用正则表达式

grep 命令不仅可以搜索一个固定的字符串，更重要的，还可以使用“正则表达式”搜索一种模式。

如：

```
grep "h[eu]llo" myfile
```

或者：

```
grep "colou?r" myfile
```

虽然正则表达式与文件名通配符的规则有相仿之处，其实两者很不相同。

使用正则表达式的命令或实用程序还有：

vi, awk, sed, perl ...

32 正则表达式

正则表达式 Regular Expression

正则表达式的起源

最早，两位神经生理学家 Warren McCulloch 和 Walter Pitts 研究出一种数学方式来描述神经网络。

1956 年, 一位叫 stephen Kleene 的数学家在 McCulloch 和 Pitts 早期工作的基础上，发表了一篇标题为“神经网事件的表示法”的论文，引入了正则表达式的概念。正则表达式就是用来描述他称为“正则集的代数”的表达式，因此采用“正则表达式”这个术语。

随后，Ken Thompson 将此方法用于搜索算法的一些早期研究，Ken Thompson 是 Unix 的主要发明人。正则表达式的第一个实用应用程序就是 Unix 中的 qed 编辑器。

正则表达式的定义

RE 描述了一种字符串匹配的模式（pattern），可以用来检查一个串是否含有某种子串、将匹配的子串做替换或者从某个串中取出符合某个条件的子串等。

shell 中的文件通配符也使用了 RE 使用的 *, ?, [ ] 等符号。但和正则表达式的意义不同。

（UNIX 当中存在的不一致性）

正则表达式的构成

RE 由以下主要元素构成：

1. 字符
2. 字符集
3. 限定符
4. 定位符
5. 选择符
6. 引用

字符

普通字符：A, B, ...a, b ...1, 2 ...
_（下划线）、␣（空格）...

特殊字符（不可打印字符）：换行符：\n，回车符：\r，制表符：\t，...

元字符：

. 任意单个字符（除换行符外）
^ 行的开头（或者表示逻辑非）
$ 行的结尾
\ 转义字符:
\$ 代表 $
\^ 代表 ^
\\ 代表 \ 自身。

举例：
`^once`　包含特殊字符 ^，只匹配那些以 once 开头的字符串。
与字符串 “once upon a time” 匹配
与 “There once was a man from New York” 不匹配

$ 符号匹配以给定模式结尾的字符串。
`bucket$`　与 “Who kept all of this cash in a bucket” 匹配
与 “buckets” 不匹配

字符 ^ 和 $ 同时使用时，表示精确匹配。例如：
`^bucket$`　匹配什么？

最普通的例子：
`once` 与以下的字符串如何匹配？

```
There once was a man from New York
Who kept all of his cash in a bucket.
```

字符集

字符集匹配集合中的任一字符。方括号 [ ] 是字符集的标识。
怎样表示方括号自身呢？
`[AaEeIiOoUu]`　匹配任何单个元音字符
`[tT]he`　匹配 “the” 和 “The”

^ 字符用在方括号内的首位时，表示 “NOT”，如：`[^Tt]` 代表除 T 或 t 之外的其他单个字符
可以用指定范围的方式，表示字符集：
`[a-e]` 等效于 `[abcde]`

`[a-z]`	所有的小写字母
`[A-Z]`	所有的大写字母
`[a-zA-Z]`	所有的字母
`[0-9]`	所有的数字
`[0-9\.\-]`	所有的数字，句号和减号
`[ \f\r\t\n]`	所有的白字符

字符集只匹配单个字符
`^[a-z][0-9]$` 匹配 “z2”, “t6”, “g7”，不匹配 “ab2”, “r2d3”。
`^[a-z][a-z][0-9]$` 可以匹配 “ab2”。

限定符

限定符指明在其前面的字符或子表达式出现的次数。

*　之前的字符或子表达式重复零次或多次
`zo*`　匹配　z, zo, zoo, zooo

\+　一次或多次
`zo+`　匹配　zo, zoo,zooo
不匹配　z

?　零次或一次
`do(es)?`　匹配　do 或　does
`does?`　匹配　doe 或　does

限定符（续）

{n}　匹配确定的 n 次
　　o{2}　不匹配 Bob 中的 o
　　匹配 food 中的两个 o
{n,}　至少 n 次
{,m}　至多 m 次
{n,m}　n 到 m 次

* 等价于 {0,} + 等价于 {1,} ? 等价于 {0,1}

定位符

定位符指明字符或子表达式所在的位置。

比如：
there were many orchards of fruit tree
in the dark it was like summer lightning
希望匹配单词 “the” 不希望匹配 “there” 当中的前三个字母。
可以使用模式 `the` （在单词的后面有一个空格）。
但这样仍可能误匹配单词 “breathe”（the 后面也有一个空格）

定位符（续）

`^`	:	行首
`$`	:	行尾
`\<`	:	单词的开始处
`\>`	:	单词的结尾处
`\b`	:	单词的边界（首或尾）
`\B`	:	不在单词的边界

例如：`\<the\>` 只匹配单词 “the”
`\<it\>` 只匹配单词 “it”，不匹配 “sitting” 当中的 “it”。

选择

圆括号（）表示一个组，某些应用场合中用 \(和 \) 代替
竖线 | 表示 “OR”，用来表示选择的含义。

例：

`ab+c`	表示 a 后面有一个 b 或多个 b 并跟有 c yabcw, abbc57 符合
`(ab)+c`	表示一个或多个字符串 ab 并跟有 c zabcd, ababc 符合
`industr(y|ies)`	相当于 `industry|industries` 匹配 industry, 也匹配 industries

引用

&　:　整个正则表达式所匹配的内容
\n　:　第 n 个子表达式所匹配的内容
　　\1 第一个子表达式匹配的内容
　　\2 第二个子表达式匹配的内容

正则表达式的例子

用一个例子来帮助理解“引用”的概念。

例如，vim 正在编辑的文件中有两行：

```
025  nanjing
027  wuhan
```

尝试命令：

```
:%s/0[0-9]* \+[a-zA-Z]\+/X&Y/
:%s/\(0[0-9]*\) \+\([a-zA-Z]\+\)/\2  \1/
```

学习正则表达式的途径

最好的方法是在 vim (vi) 环境当中多加练习。

使用 `/pattern` 或者 `:%s/pattern/replacement/gc` 命令细心实验观察。

其他的应用

有些 Windows 操作系统上运行的编辑器，如 UltraEditor，允许使用正则表达式进行搜索。

PHP，JAVA，JAVAScript 和 VBScript 当中，都有正则表达式的应用。

在微软的 MS Word 当中，也有类似于正则表达式的搜索方式。

Part XI
UNIX shell（续）

33 shell 变量

shell 变量

为了具有一定的适应性，shell 需要一些信息构成运行环境。

例如：当前用户名（LOGIN 或 USER），

用户主目录（HOME）。

通常，为用户的应用程序，提供关于运行环境信息的那些变量，被成为环境变量。

shell 的常用环境变量

- HOME ..... 用户主目录
- DISPLAY ..... 当前显示器
- PWD ..... 当前目录
- SHELL ..... 用户缺省 shell
- TERM ..... 终端类型
- PATH ..... 搜索路径
- USER ..... 用户名
- ...

显示、设置、清除、引用 shell 变量

```
set          显示全部的 shell 变量
XYZ=10       设置 XYZ 变量的值为 10
unset XYZ    清除 XYZ 变量
echo $XYZ    引用 XYZ 变量的例子，变量之前的 $ 符号不能缺少
```

34 更多的 shell 元字符

更多的 shell 元字符

shell 还有一些元字符，具有特殊的意义。

前重音符号（‘），起命令替换的作用。

```
echo It is : `date`

echo -e The first line of password \
 file is "\n"`cat /etc/passwd | head -1`
```

```
echo -e The first line of password \
 file is "\n"`head -1 /etc/passwd`
```

“命令替换” 操作的另一种格式为：
$(命令)

以下是等效的两个命令：

```
echo It is : `date`
echo It is : $(date)
```

命令序列分隔 ;

```
date;sleep 5;cal
```

命令编组括号对 ()

```
( ls -c ; date ; pwd ) > outfile
```

对比：

```
ls -c ; date ; pwd > outfile
```

使用括号的另一层意义：

```
cd / ; date ; pwd
```

和

```
( cd / ; date ; pwd )
```

注意在命令执行之后，当前目录停留在哪里。

后台执行符号 &

例：

```
find . -name xyz.xyz -print > result &
```

思考：什么时候需要将作业放到后台执行？什么样的作业适合放到后台执行？

35 作业与进程控制

作业控制

命令行的末尾，使用 & 字符，可以把用户的作业放到后台执行。

适合放在后台执行的作业，通常在执行的过程中不需要交互。作业的输出可以使用重定向，改到文件中保存。

UNIX 对作业的控制操作

把正在前台运行的作业置于后台 [Ctrl-z]: 挂起作业，并置于后台。
了解当前作业情况 jobs
把后台作业调到前台执行 fg 作业号
让后台挂起的作业继续在后台执行 bg 作业号
终止正在运行的后台作用 kill 作业号
终止正在运行的前台作业 [Ctrl-c]
用户退出登录后，其作用仍继续运行 nohup 要执行的命令

进程控制

进程状态显示 ps
最常使用的选项：
ps -auxw
终止进程 kill 进程号
遇到顽固的进程，需要发出不同的信号给进程。
kill -信号值 进程号

Part XII
shell 编程

36 shell 脚本

shell 脚本

shell 脚本是一个文件，其中包含将由 shell 解释并执行的一系列命令。这些命令有的由 shell 直接执行，另一些由 shell 交给其他的实用程序执行。

UNIX 的 shell 脚本是一种将 UNIX 中的实用程序组合起来解决复杂问题的有效手段。

一个脚本的实例

有一个需要经常使用的命令，用来打开我讲课所使用的幻灯片文件，如下；

```
/usr/lib/Acrobat4/bin/acroread /mnt/sda1/step_unixabc.pdf
```

将这条命令写到一个文件中，文件的名字为：

```
myscript
```

以后在命令行上直接输入

```
$ ./myscript [RETURN]
```

就可以直接执行所需要的完整的命令。

如何执行脚本 1. 调用脚本

显式地将脚本文件提交给特定的 shell 执行。

```
$ sh myscript $ /bin/sh myscript $ bash myscript $ bash /home/david/myscript
$ /bin/csh /home/david/myscript
```

如何执行脚本 2. 使用具有执行属性的脚本

先为要执行的脚本文件加上可执行权限：

```
$ chmod u+x myscript
```

执行脚本时，通常脚本文件所在的目录，没有被包括在 PATH 环境变量之中，需要使用脚本文件的“路径名”。

```
$ ./myscript $ /home/david/myscript
```

如何执行脚本 3. 明确指明运行脚本的 shell

如果没有特殊声明，系统将会使用“缺省 shell”，来运行脚本程序。一般地，为了明确指出用哪个 shell 来运行脚本程序，需要在脚本的一开始，加上：

```
#!/bin/sh
```

或者

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

或者

```
#!/bin/csh
```

关于脚本的进一步说明

1. 脚本中的多个命令行将依次执行，除非遇到特殊的脚本控制命令。
2. 注释由 “#” 开始，到一行的末尾结束。
3. 脚本中，对于特殊字符的处理方式，跟命令行中处理特殊字符的方式相同。

```
$ echo -e "\07\07WARNING"
```

37 shell 脚本语言

shell 编程基础 变量

shell 脚本中的变量使用前不需要专门说明。如：

```
NAME=David
```

就定义了变量 NAME。

shell 变量只有一种类型：字符串

引用变量在变量名之前加 “$”，如：

```
echo $NAME
```

清除变量，用 unset 命令：

```
unset NAME
```

shell 编程基础 命令行参数

shell 脚本允许在命令行上输入选项和参数。

shell 脚本可以从命令行读入最多 10 个参数。它们被自动地置入 “位置变量” 当中。这些位置变量将在脚本中被使用。

$0	:	命令行上所键入的脚本文件名
$1,$2,...,$9	:	脚本文件名之后的第 1 到第 9 个参数
$#	:	命令行参数的个数
$@	:	所有命令行参数 （每个参数以引号形式保存）
$*	:	所有命令行参数 （保存在单一的字符串中）
$?	:	最后一个命令的退出状态（返回码）
$$	:	正在执行的进程的进程号

从一个最简单的脚本开始

考虑一种情况。用户 David 最近以来，一直在写一份报告，文件名为：report。他每天登录到系统之后，都先把昨天未完成的这份文件，复制到另外一个名为 keep 的子目录中，接着开始用 vi 命令继续编辑这份文件。

假设文件 report 的路径名为：/home/david/report keep 子目录的路径名为：/home/david/keep 请为他设计一个 shell 脚本。

没有脚本的情况下，David 所键入的命令如下：

```
$ cp report keep
$ vi report
```

脚本文件：svi

```
# This is shell script: svi
# Written for David by Tom
cp report keep
vi report
exit 0
```

当 David 的这份报告总算写好以后，他开始用同样的方式，编写另一份文件，名为 myplan。是否需要再写一个同样的脚本？

使用命令行参数

利用前面所介绍的脚本命令行参数，改造第一个脚本成为更有适应性的版本。

```
# This is shell script: svi
# Written for David by Tom
# Version 2: Modified base on original version
# usage: svi file_name
cp $1 keep
vi $1
exit 0
```

使用脚本：

```
$ ./svi report
$ ./svi myplan
$ ./svi other_file_name
```

使用 shell 变量 用户当前的工作目录，不是他的用户主目录的时候，脚本能正常工作吗？

局部的改动：

`cp $1 $HOME/keep` 或者： `cp $1 ~/keep`

改进后的第三版：

```
# This is shell script: svi
# Written for David by Tom
# Version 3: Modified base on version 2
# usage: svi file_name
cp $1 $HOME/keep
vi $1
exit 0
```

条件与测试 如果 David 使用 svi 脚本时，脚本文件名后面没有跟上参数，svi 脚本能够正常工作吗？

if-then 结构

```
if [ condition ]
then
    commands
    ...
    last-command
fi
```

用这一结构改造 svi 的上一个版本。

svi 脚本第四版：

```
...
DIR=$HOME/keep
if [ $# = 1 ]
then
    cp $1 $DIR
fi
vi $1
exit 0
```

修改之后，虽然脚本不带参数运行时，不会出错，但仍然存在其他问题。

if-then-else 结构

```
if [ condition ]
then
    true-commands
    ...
    last-true-command
else
    false-commands
    ...
    last-false-command
fi
```

svi 脚本第五版：

```
...
DIR=$HOME/keep
if [ $# = 1 ]
then
    cp $1 $DIR
    vi $1
else
    echo "You must specify a file name. Try again."
fi
exit 0
```

如果 David 第一次使用 svi 脚本，所处理的是一个尚未建立的文件，svi 脚本能够正常工作吗？

当脚本文件名之后带有参数，但该参数所对应的文件名实际不存在时，svi 脚本仍然会出错。

所以，我们需要能够判断“某个文件是否存在”的手段。

更加完善的 svi 脚本

```
DIR=$HOME/keep
if [ $# = 1 ]
then
```

```
    if [ -f $1 ]
    then
        cp $1 $DIR
        vi $1
    else
        echo "The file does not exist."
    fi
else
    echo "You must specify a file name. Try again."
fi
```

其中的“`if [ $# = 1 ]`”是什么意思？

test 命令详解

shell 脚本的控制结构，离不开条件测试，test 命令是最重要的测试手段。

前面 if-then 结构中的 [condition] 实际上就是 test 命令的另一种简写形式。

`if [ "$VAR" = value ]`

等同于：

`if test "$VAR" = value`

test 命令的各种判断形式：

1. 数值

-eq	-gt	-lt			
-ne	-ge	-le			

2. 字符串

=	!=	-n	-z		

3. 文件

-r	-w	-s	-f	-d	...

4. 逻辑

-a	-o	!

svi 脚本第六版：

```
DIR=$HOME/keep
if [ $# = 1 ]
then
    if test -f $1
    then
        cp $1 $DIR
        vi $1
    else
        echo "The file you specified does not found !"
    fi
else
    echo "You must specify a file name. Try again."
fi
exit 0
```

注意：

`if test -f $1`

也可写成：

```
if [ -f $1 ]
```

结合上面第六版的 svi 脚本，考虑 David 先生有另外一种需要：他在工作的间隙写小说。这可以用 svi novel 命令来启动这项工作。更加方便的处理是：当使用 svi 脚本，不带任何参数时，缺省的情况就是对 novel 文件进行处理。

参数替换 1. 参数替换的格式

```
${parameter:option_character word}
```

例：

```
${CLASSNAME:-foobar}
```

以下是参数替换的不同形式：

```
$var
${var}
${var:-string}
${var:+string}
${var:=string}
${var:?string}
```

svi 脚本第七版：

```
DIR=$HOME/keep
FILENAME=${1:="novel"}
if [ -f $FILENAME ]
then
    cp $FILENAME $DIR
    vi $FILENAME
else
    echo "The file you specified does not found !"
    echo "Creat a new file named $FILENAME ..."
    sleep 2
    vi $FILENAME
fi
exit 0
```

shell 编程的更多内容 表达式运算命令 expr

shell 变量只能存放字符串类型的数据。

考虑：

```
x=1
y=2
echo $x, $y, $x + $y
```

会得到怎样的结果？

算术运算需要专门的命令：expr。应用的例子为：

```
expr 1 + 2
expr $1 + $2
echo `expr $x + $y`
```

expr 命令支持算术运算和关系运算。
1. 算术运算
expr 只能进行整数运算，五种运算为：
加：+，减：-，乘：*，除：/，取余：\%
2. 关系运算
关系运算的结果：0 表示 “假”；1 表示 “真”。
关系运算符号，是一目了然的。

```
    =    !=    \<    \<=    \>    \>=
```

赋值命令 let
举例说明：

```
x=100
let x=x+1
echo $x       <-- 101
let y=x*2
echo $y       <-- 202
```

注意空格和转义符的使用问题

循环结构 for 循环：for-in-done 结构

```
for variable in list-of-values
do
    commands
    ...
    last-command
done
```

假设 David 先生，要把当前工作目录下的所有文件，都改一个名字：

report $\Longrightarrow$ David_report
myplay $\Longrightarrow$ David_myplan
novel $\Longrightarrow$ David_novel

使用 `mv * David_*` 这样的命令，可以实现吗？
正确的方式，是使用一个脚本：

```
for i in *
do
  mv -v $i David_$i
done
```

这个脚本的缺陷是：执行之后，会把自己的文件名也改掉。
改进之后的脚本：

```
for i in *
do
  if [ ./$i != $0 ]
  then
      mv -v $i David_$i
  fi
done
```

以上脚本，每执行一次，当前目录之下的文件名之前就多处一个前缀。如果要想使得已有前缀的文件，不再被改名，应该怎样做？

```
if [ ./$i != $0 -a `echo $i | sed -e 's/David_//'` = $i ]
```

while 循环：while-do-done 结构

```
while [ condition ]
do
    commands
    ...
    last-command
done
```

until 循环：until-do-done 结构

```
until [ condition ]
do
    commands
    ...
    last-command
done
```

shell 脚本的调试

长的，或控制结构复杂的脚本，很容易包含错误。需要有可行的排错手段，用来调试脚本。

脚本文件中，用“#”开始的行，都被视为注释行。但第一行的头两个字符为“#!”时，另有含义。脚本文件的第一行为：

```
#!/bin/sh
```

表示该脚本由 /bin/sh 解释执行。

shell 的调试选项

shell 有三个常用选项，可以帮助 shell 脚本的调试。

名称	选项	功能
不执行	−n	读入命令行但不执行
冗余	−v	显示读入的行
执行跟踪	−x	执行时显示命令和参数

四. 启用调试选项的不同方法

1. 调用时启用

   ```
   sh -x myscript
   ```

2. 在脚本的首行启用

   ```
   #!/bin/sh -x
   ```

3. 在脚本中间用 set 命令启用

   ```
   set -x    (启用)
   set +x    (关闭)
   set -     (关闭所有已打开的调试选项)
   ```

Part XIII
流编辑器 sed

流编辑器 sed 的简单介绍

通过 shell 编程，可以将 UNIX 的各种命令组织起来，完成复杂的任务。

必须有更加强大的工具，提供 shell 编程的基础。

流编辑器（Stream EDitor）是一种基于正则表达式的，非交互式的编辑器。

sed 基础

一. sed 的基本工作方式

对于输入文件的每一行，执行一系列的 sed 命令。这些 sed 命令被称为 sed 脚本。

sed 脚本针对输入文件，或者输入流的每一个文本行进行处理。

二. sed 命令的语法 `sed 'script' files`

缺省情况下，处理之后的结果提供标准设备输出，也就是显示在屏幕上。

当命令后面没有跟上文件名时，处理的对象就是标准输入。

sed 也可以接受通过管道送来的字符流。

第一个例子

假设有一个文件，名字为： tel 其中当内容，是一些城市的长途电话直拨号码前缀。内容大体为：

```
1 nj 025
2 gz 020
3 sh 021
4 bj 010
5 wh 027
6 cd 028
```

用 sed 命令，把区号之前的 “0” 换成 “+86”。

形式一：命令行

```
sed 's/\<0/+86/' tel
```

形式二：sed 脚本文件

把 `s/\<0/+86/` 写到一个文件之中，比如文件名为：inter

在命令行上输入命令：

```
sed -f inter tel
```

形式三：使用管道

```
cat tel | sed 's/\<0/+86/'
```

或者：

```
cat tel | sed -f inter
```

把区号字段，和城市名字段对调，应当怎样实现？

Linux公社（LinuxIDC.com）于2006年9月25日注册并开通网站，Linux现在已经成为一种广受关注和支持的一种操作系统，IDC是互联网数据中心，LinuxIDC就是关于Linux的数据中心。

LinuxIDC.com提供包括Ubuntu，Fedora，SUSE技术，以及最新IT资讯等Linux专业类网站。

并被收录到Google 网页目录-计算机 > 软件 > 操作系统 > Linux 目录下。

Linux公社（LinuxIDC.com）设置了有一定影响力的Linux专题栏目。

包括:

Ubuntu专题

Fedora专题

RedHat专题

SUSE专题

红旗Linux专题

Android专题

sed 脚本的执行与模式空间

1. 从输入文件中读取一行
2. 把这一行复制出一个副本，放在缓冲区中
3. 对于缓冲区中的副本，应用 sed 脚本。
4. 如果 sed 脚本有多行，这些脚本行将依次被执行
5. 输入文件的一行处理完毕，再取输入文件的下一行，做同样处理。

上面提到的存放文本行副本的缓冲区，就是“模式空间”。
sed 的一切操作，都是在模式空间中进行的。

sed 脚本

sed 脚本，是由一行行的 sed 命令组成的。
只有一个单行的 sed 脚本，是常见的情况。
语法
最一般的 sed 脚本命令形式：

```
[address] command
```

方括号之内的行地址是可选的。
个别只能用于单行地址的 sed 脚本命令，表示成：

```
[line-address] command
```

command 部分可以有 25 种不同的命令。只需要掌握其中很少的一部分，就可以处理大多数的情况。

常用的 sed 脚本命令

1. 替换命令 s

用法：

```
[address]s/pattern/replacement/flags
```

我们已经用过这个命令，例如：

```
s/oldstring/newstring/
```

这条命令的后面有几种可以选择的标志，用来修饰替换方式。

1. n：1 到 512 之间的一个数字，表示对指定模式中第 n 次匹配处进行替换。
2. g：对模式空间中的所有符合匹配模式的内容进行替换。
3. p：显示模式空间内容
4. w file：将模式空间的内容写到文件 file 中。

2. 显示命令 p 假设有一个记录水果价格的文件，名为：fruite_prices.txt ，内容为：

```
Fruit           Price/lbs
Banana          0.89
Paech           0.79
Kiwi            1.50
Pineapple       1.29
Apple           0.99
Mango           2.20
```

要求列出价格低于 1 美元/磅的水果列表。
观察正则表达式：`/ 0\.[0-9][0-9]$/`
使用命令：
`sed '/ 0\.[0-9][0-9]$/p' fruite_prices.txt`
结果怎样？

再使用命令：`sed -n '/ 0\.[0-9][0-9]$/p' fruite_prices.txt`
结果怎样？
3. 删除命令 d
芒果已售完，希望从列表中删去， sed 脚本怎样写？

```
sed '/^[Mm]ango/d' fruite_pricea.txt
```

删除的动作，只在模式空间之内发生，不会影响原有的文件。
4. 追加、插入、更改（a, i, c）

```
[line-address] a \
    text

[line-address] i \
    text

[line-address] c \
    text
```

这些命令分别把所提供的文本，放置于模式空间的文本行之后、之前，或者取代它。
5. 列表命令 l
将模式空间内的字符列出。
其他的 sed 命令：

y	转换命令
=	打印行号
n	下一行
r	读文件
w	写文件
q	退出

Part XIV
附录

实验指导书

实验指导书的电子稿
`http://www.box.net/shared/mulga40cvv`

或者，
`ftp://202.119.167.222/`
在此寻找文件 `unixguide.pdf`