Linux 学习笔记

郑华

2018 年 1 月 17 日

目录

第一章　Linux 基本知识

1.1　基本技能

正如你所见，Linux 发行版并非 Linux，Linux 仅是指操作系统的内核。

1.1.1　推荐的发行版

- UBUNTU 适合纯菜鸟，追求稳定的官方支持，对系统稳定性要求较弱，喜欢最新应用，相对来说不太喜欢折腾的开发者。
- Debian，相对 UBUNTU 难很多的发行版，突出特点是稳定与容易使用的包管理系统，缺点是企业支持不足，为社区开发驱动。
- Arch，追逐时尚的开发者的首选，优点是包更新相当快，无缝升级，一次安装基本可以一直运作下去，没有如 UBUNTU 那样的版本概念，说的专业点叫滚动升级，保持你的系统一定是最新的。缺点显然易见，不稳定。同时安装配置相对 Debian 再麻烦点。
- Gentoo，相对 Arch 再难点，考验使用者的综合水平，从系统安装到微调，内核编译都亲历亲为，是高手及黑客显示自己技术手段，按需配置符合自己要求的系统的首选。
- Slackware 与 Gentoo 类似。
- CentOS，社区维护的 RedHat 的复刻版本，完全使用 RedHat 的源码重新编译生成，与 RedHat 的兼容性在理论上来说是最好的。如果你专注于 Linux 服务器，如网络管理，架站，那么 CentOS 是你的选择。
- LFS，终极黑客显摆工具，完全从源代码安装，编译系统。安装前你得到的只有一份文档，你要做的就是照文档你的说明，一步步，一条条命令，一个个软件包的去构建你的 Linux，完全由你自己控制，想要什么就是什么。如果你做出了 LFS，证明你的 Linux 功底已经相当不错，如果你能拿 LFS 文档活学活用，再将 Linux 从源代码开始移植到嵌入式系统，我敢说中国的企业你可以混的很好。

1.1.2 Linux 基础

你得挑一个适合你的系统，然后在虚拟机安装它，开始使用它。如果你想快速学会 Linux，我有一个建议就是**忘记图形界面**，不要想图形界面能不能提供你问题的答案，**而是满世界的去找，去问，如何用命令行解决你的问题**。

在这个过程中，你最好能将 Linux 的命令掌握的不错，起码常用的命令得知道，同时建立了自己的知识库，里面是你积累的各项知识。

1.1.3 Linux 平台 C++ 开发

你需要学习的是 Linux 平台的 C/C++ 开发，同时还有 Bash 脚本编程，如果你对 Java 兴趣很深还有 Java。同样，建议你抛弃掉图形界面的 IDE，从 VIM 开始，为什么是 VIM，而不是 Emacs，我无意挑起编辑器大战，但我觉得 VIM 适合初学者，适合手比较笨，脑袋比较慢的开发者。Emacs 的键位太多，太复杂，我很畏惧。然后是 GCC，Make，Eclipse (Java，C++ 或者)。

虽然将 C++ 列在了 Eclipse 中，但我并不推荐用 IDE 开发 C++，因为这不是 Linux 的文化，容易让你忽略一些你应该注意的问题。IDE 让你变懒，懒得跟猪一样。如果你对程序调试，测试工作很感兴趣，**GDB 也得学的很好**，如果不是 GDB 也是必修课。这是开发的第一步，注意我并没有提过一句 Linux 系统 API 的内容，这个阶段也不要关心这个。你要做的就是积累经验，在 Linux 平台的开发经验。

我推荐的书如下：C 语言程序设计。C 语言，白皮书当然更好。C++ 推荐 C++ Primer Plus，Java 我不喜欢，就不推荐了，附一个别人的书单：java 入门书籍。工具方面推荐 **VIM 的官方手册**，**GCC 中文文档**，**GDB 中文文档**，**GNU 开源软件开发指导** (电子书)，汇编语言程序设计 (让你对库，链接，内嵌汇编，编译器优化选项有初步了解，不必深度)。

如果你这个阶段过不了就不必往下做了，这是底线，最基础的基础，否则离开，不要霍霍 Linux 开发。不专业的 Linux 开发者作出的程序是与 Linux 文化或 UNIX 文化相背的，程序是走不远的，不可能像 Bash，VIM 这些神品一样。所以做不好干脆离开。

1.1.4 UNIX 环境高级编程

UNIX 环境高级编程堪称神作，经典中的经典。

接下来进入 Linux 系统编程，不二选择，**APUE，UNIX 环境高级编程**，一遍一遍的看，看 10 遍都嫌少，如果你可以在大学将这本书翻烂，里面的内容都实践过，有作品，你口头表达能力够强，你可以在面试时说服所有的考官。

（可能有点夸张，但 APUE 绝对是圣经一般的读物，即使是 Windows 程序员也从其中汲取养分，Google 创始人的案头书籍，扎尔伯克的床头读物。）

这本书看完后你会对 Linux 系统编程有相当的了解，知道 Linux 与 Windows 平台间开发的差异在哪？它们的优缺点在哪？我的总结如下：做 Windows 平台开发，很苦，微软的系统 API 总在扩容，想使用最新潮，最高效的功能，最适合当前流行系统的功能你必须时刻学习。Linux 不是，Linux 系统的核心 API 就 100 来个，记忆力好完全可以背下来。而且经久不变，为什么不变，因为要同 UNIX 兼容，符合 POSIX 标准。所以 Linux 平台的开发大多是专注于底层的或服务器编程。

这是其优点，当然图形是 Linux 的软肋，但我站在一个开发者的角度，我无所谓，因为命令行我也可以适应，如果有更好的图形界面我就当作恩赐吧。另外，Windows 闭源，系统做了什么你更本不知道，永远被微软牵着鼻子跑，想想如果微软说 Win8 不支持 QQ，那腾讯不得哭死。而 Linux 完全开源，你不喜欢，可以自己改，只要你技术够。

另外，Windows 虽然使用的人多，但使用场合单一，专注与桌面。而 Linux 在各个方面都有发展，尤其在云计算，服务器软件，嵌入式领域，企业级应用上有广大前景，而且兼容性一流，由于支持 POSIX 可以无缝的运行在 UNIX 系统之上，不管是苹果的 Mac 还是 IBM 的 AS400 系列，都是完全支持的。另外，Linux 的开发环境支持也绝对是一流的，不管是 C/C++，Java，Bash，Python，PHP，Javascript，就连 C# 也支持。而微软除 Visual Stdio 套件以外，都不怎么友好，不是吗？

如果你看完 APUE 的感触有很多，希望验证你的某些想法或经验，推荐 UNIX 程序设计艺术，世界顶级黑客将同你分享他的看法。

1.2 方向选择

1.2.1 网络方向

服务器软件编写及**高性能的并发程序**编写

现在是时候做分流了。大体上我分为四个方向：网络，图形，嵌入式，设备驱动。

如果选择网络，再细分，我对其他的不是他熟悉，只说服务器软件编写及高性能的并发程序编写吧。相对来说这是网络编程中技术含量最高的，也是底层的。需要很多的经验，看很多的书，做很多的项目。

我的看法是以下面的顺序来看书：

1. APUE 再深读 – 尤其是进程，线程，IPC，套接字

2. 多核程序设计 - Pthread 一定得吃透了，你很 NB

3. UNIX 网络编程 – 卷一，卷二

4. TCP/IP 网络详解 – 卷一再看上面两本书时就该看了

5. TCP/IP 网络详解 – 卷二我觉得看到卷二就差不多了，当然卷三看了更好，努力，争取看了

6. Lighttpd 源代码 - 这个服务器也很有名了

7. Nginx 源代码 – 相较于 Apache，Nginx 的源码较少，如果能看个大致，很 NB。看源代码主要是要学习里面的套接字编程及并发控制，想想都激动。如果你有这些本事，可以试着往暴雪投简历，为他们写服务器后台，想一想全球的魔兽都运行在你的服务器软件上。

8. Linux 内核 TCP/IP 协议栈 – 深入了解 TCP/IP 的实现

如果你还喜欢驱动程序设计，可以看看更底层的协议，如链路层的，写什么路由器，网卡，网络设备的驱动及嵌入式系统软件应该也不成问题了。

当然一般的网络公司，就算百度级别的也该毫不犹豫的雇用你。只是看后面这些书需要时间与经验，所以 35 岁以前办到吧！跳槽到给你未来的地方！

1.2.2 图形方向

我觉得图形方向也是很有前途的，以下几个方面。

1. Opengl 的工业及游戏开发，国外较成熟。

2. 影视动画特效，如皮克斯，也是国外较成熟。

3. GPU 计算技术，可以应用在浏览器网页渲染上，GPU 计算资源利用上，由于开源的原因，有很多的文档程序可以参考。如果能进火狐开发，或 google 做浏览器开发，应该会很好。

1.2.3 嵌入式方向

嵌入式方向没说的，Linux 很重要。

掌握多个架构，不仅 X86 的，ARM 的，单片机什么的也必须得懂。硬件不懂我预见你会死在半路上，我也想走嵌入式方向，但我觉得就学校教授嵌入式的方法，我连学电子的那帮学生都竞争不过。奉劝大家，一定得懂硬件再去做，如果走到嵌入式应用开发，只能祝你好运，不要碰上像 Nokia，Hp 这样的公司，否则你会很惨的。

1.2.4 驱动程序设计

软件开发周期是很长的，硬件不同，很快。每个月诞生那么多的新硬件，如何让他们在 Linux 上工作起来，这是你的工作。由于 Linux 的兼容性很好，如果不是太低层的驱动，基本 C 语言就可以搞定，系统架构的影响不大，因为有系统支持，你可能做些许更改就可以在 ARM 上使用 PC 的硬件了，所以做硬件驱动开发不像嵌入式，对硬件知识的要求很高。

可以从事的方向也很多，如家电啊，特别是如索尼，日立，希捷，富士康这样的厂子，很稀缺的。

1.3 硬件设备

在 linux 系统中，每个设备都被当成一个文件来对待，如图1.1。

设备	设备在 linux 中的文件名
SATA/USB 硬盘/U 盘	/dev/sd[a-p]
当前鼠标	/dev/mouse

表 1.1: 设备文件名

1.4 磁盘分区知识

磁盘分区是使用分区编辑器 (`partition editor`) **在磁盘上**划分几个**逻辑部分**。碟片一旦划分数个分区 (`Partition`)，不同类的目录与文件可以存储进不同的分区。

1.4.1 分区类型

主分区 主分区中不能再划分其他类型的分区，因此每个**主分区**都相当于**一个逻辑磁盘**（在这一点上主分区和逻辑分区很相似，但主分区是直接在硬盘上划分的，逻辑分区则必须建立于扩展分区中）

扩展分区 最多只有 1 个，在使用 MBR(主引导分区记录) 方式下主分区与扩展分区最多只能有 4 个，而且扩展分区不能写入数据，只能包含逻辑分区

1.4.2 GUID

1.5 裸机装机必备软件及技巧

vim 必备，要么你修改配置文档都是个问题。`sudo apt-get install vim`

网络配置 有网才能工作

1.ln -s 创建快捷命令

2.1 配置 IP(一般不动)

- `vim /etc/network/interfaces`
- 根据情况确定用动态 IP 还是静态 IP

```
#动态IP
    auto eth0
    iface eth0 inet dhcp

#静态IP
    auto eth0
    iface eth0 inet static
    address xx.xx.xx.xx
    netmask xx.xx.xx.xx
    gateway xx.xx.xx.xx
```

2.2 配置 DNS

- `sudo vi /etc/resolv.conf`
- 添加`nameserver 8.8.8.8` 或 `8.8.4.4`
- 重启网络：

```
service networking restart

sudo /etc/init.d/networking restart
```

更新源 使用国内阿里云镜像，更快速度。

输入法

1. `sudo apt-get update`
2. `sudo apt-get install fcitx-googlepinyin`
3. 注销或重启
4. 找标题栏中的小企鹅，点开找配置 fcitx
5. 点+, 取消选中 only. 查找 Google-pinyin

文本工具 Texlive 与 Texstudio

git

网易云音乐 有音乐工作才有动力

1. 官网下载 deb
2. `sudo dpkg -i ~/Downloads/xx.deb`
3. 出现错误时`sudo apt install -f`
4. 再次运行安装命令

第二章　命令相关

2.1　导航命令

pwd　查看当前工作目录

cd　改变当前工作目录

- 绝对路径：从根目录 / 开始
- 相对路径：从当前位置开始，. 表示当前目录，.. 表示上级目录

ls　列出目录内容

表 2.1: **ls** 参数说明

第一可选参数	第二可选参数	意义
-a	--all	列出所有文件，包括以点号开头的文件，这些文件通常是不列出来的 (比如隐藏的文件)
-d	--directory	查看目录的详细信息，而不是目录里的内容
-F	--classify	如果名字是目录，则会加上一个斜杠
-l		使用长格式显示结果
-i		-inode 印出每个文件的inode 号
-r	--reverse	以相反的顺序显示结果
-S		按照文件的大小对结果进行排序
-t		按照文件的修改时间排序
--color	={never, auto, always}	结果颜色选项
--full-time		完整呈现结果的最后修改时间

2.2 Linux 系统

表 2.3: Linux 目录详解

目录	存放的内容类别
/	文件系统的入口，**最高一级目录**
/bin	**基础系统所需要的命令位于此目录**，是最小系统所需要的命令，如：ls, cp, mkdir 等。这个目录中的文件都是可执行的，一般的用户都可以使用。
/boot	包含 **Linux 内核及系统引导程序所需要的文件**，比如 vmlinuz initrd.img 文件都位于这个目录中。
/dev	**设备文件存储目录**，是 Linux 文件系统的一个闪亮的特性 - 所有对象都是文件或目录。仔细观察这个目录你会发现 hda1, hda2 等, 它们代表系统主硬盘的不同分区。
/etc	存放系统程序或者一般工具的**配置文件**
/lib	库文件存放目录这里包含了**系统程序所需要的所有共享库文件**, 类似于 Windows 的共享库 DLL 文件。
/lost+found	在 ext2 或 ext3 文件系统中，当系统意外崩溃或机器意外关机，而**产生一些文件碎片放在这里**
/media	即插即用型**存储设备的挂载点**自动在这个目录下创建，比如 USB 盘系统自动挂载后，会在这个目录下产生一个目录；
/mnt	这个目录一般是用于**存放挂载储存设备的挂载目录**的, 比如有 cdrom 等目录。有时我们可以把让系统开机自动挂载文件系统，把挂载点放在这里也是可以的。比如光驱可以挂载到/mnt/cdrom
/opt	表示的是可选择的意思，**有些软件包也会被安装在这里，也就是自定义软件包**
/proc	/proc 目录是伪装的文件系统 proc 的挂载目录, proc 并不是真正的文件系统。这个目录本身是一个“虚拟文件系统（virtual filesystem）”喔！他放置的数据都是在内存当中，例如系统核心、行程信息（process）、周边设备的状态及网络状态等等。因为这个目录下的数据都是在内存当中，所以本身不占任何硬盘空间啊！
/root	Linux **超级权限用户** root 的家目录
/sbin	多是涉及**系统管理的命令的存放，是超级权限用户 root 的可执行命令存放地**，普通用户无权限执行这个目录下的命令；这个目录和/usr/sbin; /usr/X11R6/sbin 或/usr/local/sbin 目录是相似的；我们记住就行了，凡是目录 sbin 中包含的都是 root 权限才能执行的
/tmp	临时文件目录，有时用户运行程序的时候，会产生临时文件。
/usr	(Unix Software Resources) 这个是**系统存放程序的目录**，比如命令、帮助文件等。这个目录下有很多的文件和目录。当我们**安装一个 Linux 发行版官方提供的软件包时，大多安装在这里**。

$PATH 当执行一个指令 (ls) 的时候，系统会依照PATH 的设置去每个 PATH 定义的目录下搜寻文件名为 (ls) 的可执行文件，如果在 PATH 定义的目录中含有多个文件名为 (ls) 的可执行文件，那么先搜寻到的同名指令先被执行。

如果命令不在PATH 指定路径下时，单纯敲击指令名称是不会执行的，有以下两种方法可以执行该指令

1. 使用绝对路径或相对路径指定命令 `./ls` 或 `/root/ls`
2. 将目标目录添加到PATH 中 `PATH = "${PATH}:/root"`

用户配置

- `~/.bash_profile`

  每个用户都可使用该文件输入**专用于自己使用的 shell** 信息,当用户登录时, 该文件仅仅执行一次! 默认情况下, 他设置一些环境变量, 执行用户的.bashrc 文件. 此文件类似于/etc/profile, 也是需要需要重启才会生效，`~/.bash_profile` **只对当前用户生效**。

- `~/.bashrc`

  该文件包含**专用于你的 bash shell 的 bash 信息**, 当登录时以及每次打开新的 *shell* 时, 该文件被读取. (每个用户都有一个.bashrc 文件,在用户目录下) 此文件类似于/etc/bashrc, **不需要重启生效, 重新打开一个 bash 即可生效**, 但~/.bashrc 只对当前用户新打开的 bash 生效。

- `/etc/profile`

  `/etc/profile` 对所有用户生效

- `/etc/bashrc`

  `/etc/bashrc` 对所有用户新打开的 bash 都生效

交换区 在内存小于 2G 的情况下，交换分区**应为内存的 2 倍**，超过 2G 的话，交换分区为**物理内存加上 2G**

符号链接 符号链接又叫软链接, 是一类特殊的文件，这个文件包含了**另一个文件的路径名**

开关机

- `shutdown -r 20:30`: 在 20:30 重启

1. `-c`: 取消前一个关机命令
2. `-h`: 关机
3. `-r`: 重启

- `poweroff`
- `init 0`

2.3 操作文件和目录

2.3.1 文件系统

EXT2

- `superBlock`: 记录文件系统相关属性
  - dataBlock 与 Inode 大小熟悉
  - dataBlock 与 Inode 数量
- `bitMap`: 用于判断有哪些数据块可用
- `inodeMap`: 用于判断有哪些 Inode 块可用
- `inode(s)`: 存放文件的相关属性，和相关数据块指针 (涉及 2 级 3 级指针)。*每个文件仅会占用 1 个 Inode*
- `dataBlock(s)`: 存放数据的实际地方，*每个数据块只能存放一个文件的内容*, 所以目录文件一般独自占用一个 dataBlock。

-> 例子：当我们在 Linux 下的 ext2 **创建一个一般文件时，ext2 会分配一个 inode 与相对于该文件大小的 block 数量给该文件**。例如：假设我的一个 block 为 4 KBytes，而我要创建一个 100 KBytes 的文件，那么 linux 将分配一个 inode 与 25 个 block 来储存该文件！但同时请注意，由于 inode 仅有 12 个直接指向，因此还要多一个 block 来作为区块号码的记录喔！

dumpe2fs 列出相关设备的 superBlock 的相关信息

```
[root@study ~]# dumpe2fs /dev/vda5
```

df (disk free 可用磁盘) 列出文件系统的**整体磁盘使用量**；

- `-h` 选项，以人类易读的格式输出（例如，5K，500M 及 5G）
- `-a` 选项，显示所有文件系统的磁盘使用情况
- `-i` 选项，用于显示文件系统的 inode 信息

参考 https://linux.cn/article-6466-1.html

du du 可以**显示当前目录及子目录的磁盘占用情况**

- `-d` 选项可以指明递归目录的深度
- `-s` 等价于
- `-h` 表示以可读的形式显示，比如B，KB，GB 等

2.3.2 相关命令

通配符 `->`

表 2.5: 通配

形式	含义
*	匹配任意多字符-包括 0 个 1 个
?	匹配任一单个字符-不包括 0 个
[characters]	匹配任意一个属于字符集中的字符
[!characters]	匹配任意一个不属于字符集的字符
[[:class:]]	匹配任意一个属于指定类的字符，如[[:digit:]]
[[:alnum:]]	匹配任意一个字母或数字
[[:alpha:]]	匹配任意一个字母
[[:digit:]]	匹配任意一个数字
[[:lower:]]	匹配任意一个小写字母
[[:upper:]]	匹配任意一个大写字母

cd 改变目录简化操作有如下：

- `cd ~ || cd` ：进入当前用户家目录
- `cd -` ：进入上次目录
- `cd ..` ：进入上一级目录
- `cd .` ：进入当前目录

pwd Print Working Directory 打印当前工作目录，即当前所处位置的绝对路径

`-P`：显示出确实的路径，而非使用链接`link` 路径。

```
范例：显示出实际的工作目录，而非链接文件本身的目录名而已
[root@study ~]# cd /var/mail   <==注意，/var/mail是一个链接文件
[root@study mail]# pwd
/var/mail          <==列出目前的工作目录
[root@study mail]# pwd -P
/var/spool/mail    <==怎么回事？有没有加 -P 差很多～
[root@study mail]# ls -ld /var/mail
lrwxrwxrwx. 1 root root 10 May  4 17:51 /var/mail -> spool/mail
#  看到这里应该知道为啥了吧？因为 /var/mail 是链接文件，链接到 /var/spool/mail
#  所以，加上 pwd -P 的选项后，会不以链接文件的数据显示，而是显示正确的完整路径啊！
```

Fig 2.1: pwd -P 选项示例

cp 复制文件和目录，默认会改变时间戳

- `-r`: 复制目录
- `-p`: 连带文件属性复制
- `-d`: 若源文件是链接文件，则复制链接属性
- `-a`: 相当于`-pdr`
- `-i`: 若目标文件已经存在时，在覆盖时会询问动作是否进行。
- `-u`: 只有目标文件与原文件有差异时，才会复制

`cp fileName1  /dir/fileName2`: 改名复制

`cp fileName1  /dir/`: 原名复制

-> 复制总是希望复制到的数据最后是我们自己的，所以，在默认的条件中，`cp` 的**来源文件**与**目的文件**的**权限是不同的**，**目的文件**的**拥有者**通常会是**指令操作者本身**。

mv 移动或重命名文件和目录

当源文件和目标文件在同一目录就是改名

当源文件和目标文件不在同一目录就是剪切

dd 用指定大小的块拷贝一个文件，并在拷贝的同时进行指定的转换

- if= 文件名：输入文件名，缺省为标准输入。即指定源文件。`< if=input file >`
- of= 文件名：输出文件名，缺省为标准输出。即指定目的文件。`< of=output file >`
- http://www.cnblogs.com/ginvip/p/6370836.html

mkdir 创建目录

- `-m` ：设置文件的权限喔！直接设置，不需要看默认权限umask 的脸色
- `-p` ：递归创建 `mkdir -p /Hello/Practice/First`

rmdir Remove Empty Directory: 即只能删除空目录

rm 移除文件和目录，默认删除时会确认

- `rm -r`: 删除目录
- `rm -f`: 强制删除，不用确认是否删除

ln 创建硬链接和符号链接

- **硬链接**:`ln file file_hard`
  1. 拥有相同的 i 节点和存储块，可以看作是带有引用计数的shared_ptr，因为本质只有一个资源，而建立新的链接相当于添加新的代理类，但是访问资源都是一个资源。
  2. 可以通过 i 节点识别: 通过命令`ls -i` 查看
  3. 不能针对目录使用
- **软链接**:`ln -s file file_soft` 在软连接中，文件实际上是一个文本文件，其中包含的有另一文件的位置信息。

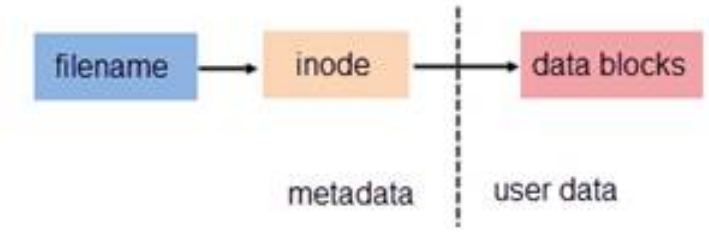


Fig 2.2: Linux Link Data Access Roud

1. 类似于 windows 的快捷方式
2. 依赖于硬链接
3. 有自己的存储块，存储对应的链接命令文件位置。
4. 访问时，先访问自己的存储块读取要读写的文件，然后再打开该文件，如果硬链接将该文件删除了，那么执行软链接就会失败。
5. 改变软链接文件同样对源文件进行操作。
6. 删除其对文件是否再存在没影响
7. Linux 使用时尽量使用**绝对路径**，两者都，相对可能会出错

-> **区别**：https://www.ibm.com/developerworks/cn/linux/l-cn-hardandsymb-links/

我们知道文件都有文件名与数据，这在 Linux 上被分成两个部分：**用户数据** (user data) 与**元数据** (metadata)。用户数据，即文件数据块 (data block)，数据块是记录文件真实内容的地方；而元数据则是文件的附加属性，如文件大小、创建时间、所有者等信息。

在 Linux 中，**元数据中的 inode 号** (inode 是文件元数据的一部分但其并不包含文件名，inode 号即索引节点号) **才是文件的唯一标识而非文件名**。文件名仅是为了方便人们的记忆和使用，系统或程序通过 inode 号寻找正确的文件数据块。图2.2展示了程序通过文件名获取文件内容的过程。

查看 inode 号可使用命令 `stat` 或 `ls -i`;

查找有相同 inode 号的文件 `find / -inum 1114`

查看路径 /home 有相同 inode 的所有硬链接 `find /home -samefile /home/old.file`

1. 若一个 inode 号对应多个文件名，则称这些文件为硬链接。换言之，硬链接就是同一个文件使用了多个别名
2. 软链接与硬链接不同，若文件用户数据块中存放的内容是另一文件的路径名的指向，则该文件就是软连接。软链接就是一个普通文件，只是数据块内容有点特殊。

查看文件

cat 由第一行开始显示文件内容

- -b：列出行号，仅针对非空白行，空白行不标行号。
- -n：打印出行号，包括空白行
- -v：列出一些看不出的特殊字符
- -E：将结尾的断行字符$ 显示出来
- -T：将tab 键以^I 的形式显示出来
- -A：-vET 可以列出特殊字符，而不是以空白代替

tac 由最后一行开始显示文件内容

nl 显示的时候，顺道输出行号

more 一页一页的显示文件内容

less 与more 类似，但是比more 更好的是，他可以往前翻页

head 只看头几行，head -n 5 ls-text.txt

-n：后面接数字，代表显示几行的意思

tail 只看尾部几行，tail -n 5 ls-text.txt

- -f 当文件增长时, 输出后续添加的数据
- -s 与-f 合用, 表示在每次反复的间隔休眠 S 秒

为了查看不断更新的日志文件，可以使用的指令tail -f

od 以二进制的方式读取文件内容

touch 修改文件时间或创建新文件

文件时间概念

- `modification Time(mtime)`：**当该文件的"内容数据"变更时，就会更新这个时间**！内容数据指的是**文件的内容**，而不是文件的属性或权限喔！
- `status Time(ctime)`：**当该文件的"状态 (status)"改变时，就会更新这个时间**，举例来说，像是**权限与属性**被更改了，都会更新这个时间啊。
- `access Time(atime)`：当"该文件的内容被取用"时，就会更新这个读取时间 (access)。举例来说，我们使用 `cat` 去读取 `/etc/man_db.conf` ，就会更新该文件的 `atime` 了。

touch 选项

- `-a`: 修改文件访问时间 atime
- `-c`: 修改文件的时间 ctime
- `-m`: 修改文件的 mtime

2.4 命令

type 说明如何解释命令名：`type command`

which 说明会执行在哪块的可执行程序：`which ls` –> /bin/ls

man 显示程序的手册页:`man command`，当有多个级别的命令时，可以使用-f 指定级别。

–help 显示程序的使用信息:`command --help`

apropos 显示适合的命令:`apropos keyWords`

whatis 显示命令的简要描述:`whatis ls`

info 显示程序的 info 条目:`info command`

alias 使用别名创建自己的命令:`alias name='command[s]'`

参考 http://man.linuxde.net

2.5 重定向

这个功能可以把命令行的输入重定向为从文件中获取内容，也可以把命令行的输出结果重定向到文件中。使用重定向符'>'，后接文件名，就可以把标准输出重定向到另一个文件中，而不是显示在屏幕上。

当使用重定向符'>' 来重定向标准输出时，目的文件通常会从文件开头部分重新改写。如果需要删除一个文件内容（或者创建一个新的空文件），可以采用这样的方式。`> fileName`

如果不需要从文件的首位置开始覆盖文件，而是从文件的**尾部开始添加内容**，我们可以使用重定向符'>>'来实现。

可以使用重定向符`&>` 来把标准输出和标准错误都重定向到同一文件中。

cat 除了查看文件，还可以将不同的文件合并到一个文件里，如 `cat movie*.mpeg >` movie movie.mpeg

管道 命令从标准输入得到数据，并将数据处理后发送到标准输出。使用管道操作符可以把一个命令的输出传送到另一个命令的标准输入中。'|'

sort 按照顺序排列输入表

uniq 可以接受来自于标准输入或者一个单个文件名对应的已经排好序的数据列表，默认情况下,该命令删除列表中的所有重复行,因此在管道中常与 sort 结合使用–> `ls /bin /usr/bin` sort | uniq | less|

wc 用来显示文件包含的-行数，字数，字节数`-l`，`-w`，`-m`，`-c`。

tee 从 stdin 读取数据，并同时输出到 stdout 和文件

/dev/null 文件 /dev/null 是一个特殊的文件，写入到它的内容都会被丢弃；如果尝试从该文件读取内容，那么什么也读不到。但是 /dev/null 文件非常有用，将命令的输出重定向到它，会起到”**禁止输出**”的效果。

2.6 shell 扩展

echo 把文本参数内容打印到标准输出:`echo *` –> 打印当前工作目录的所有文件名

波浪线扩展 如果没有指定用户名,则扩展为当前用户的主目录：`echo ~ --> /home/me  echo ~foo -`

算术扩展 调用方式：`echo $(( expression ))`

花括号扩展 `echo F-{A,B,C}-B` –> F-A-B F-B-B F-C-B ; `echo n{1..4}` –> n1 n2 n3; `echo n{z..a}` –> nz ny nx . na;

命令替换 可以把一个命令的输出作为一个扩展模式使用：`echo $(ls)` –>Desktop Documnet Music Pictures.. , 即在需要命令参数处调用‘`$(Command)`’

双引号扩展 如果把文本放到双引号里，那么所有特殊字符都将失去特殊含义除算数扩展和命令替换

单引号扩展 如果把文本放到单引号里，那么所有特殊字符都将失去特殊含义，没有除

2.7 权限

2.7.1 基础

```
Type Owner Group Others
[- rwx rwx ---]
[- 111 111 000]
[- 421 421 000]
[- 7 7 0 ]
```

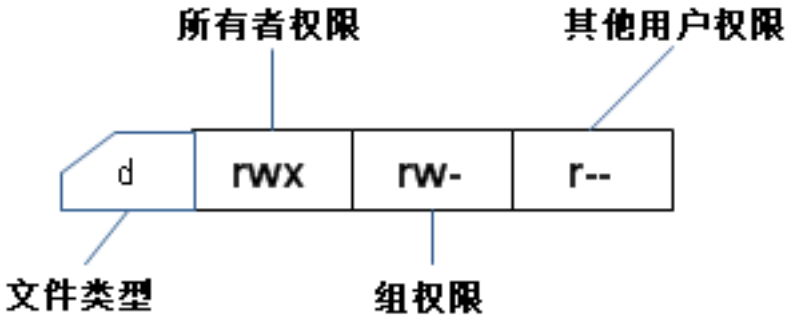


Fig 2.3: ls -l 文件权限

chmod 更改文件的模式

chmod 命令符号表示如下：

表 2.6: chmod 作用对象表示法

符号	含义
u	user 缩写，表示文件或者目录的所有者
g	文件所属群组
o	others 的缩写，表示其他用户
a	all 的缩写，是 u g o 三者的组合
+	添加权利
-	删除权利

如下示例：

```
$chmod u+x test.txt
--> 给test.txt 的user 添加可执行的权限
```

各种权限可以使用数字来表示，设置权限时可以直接使用数字进行设置:

```
chmod Owner+Group+Others file--> chmod 771 file
```

umask 设置文件的默认权限, 该值是需要减去的权限. 将给出的掩码与原文件的权限进行**异或运算**，即**如果掩码为 1，那么将该位置的权限擦除**

```
$umask 0002
--> 设置默认权限 0 000 000 010 ：即如果其他用户拥有写权限的话，将擦除该权限
```

如果 umask 为 022，所以 user 的权限并没有被拿掉什么权限，但是 group 和 others 的权限都被拿掉了 2，也就是写 (w) 权限。

```
$umask 022
 --> 创建文件时：( -rw-rw-rw- ) - ( -----w--w- ) ==> -rw-r--r--
 --> 创建目录时：( drwxrwxrwx ) - ( d----w--w- ) ==> drwxr-xr-x
```

chown 更改文件所有者, 所有者必须是已经存在系统中的帐号，也就是在/etc/passwd 这个文件中有纪录的使用者名称才能改变。

chgrp 更改文件所属群组, 不过，请记得，要被改变的群组名称必须要在/etc/group 文件内存在才行，否则就会显示错误！

passwd 更改用户密码

Set UID 普通用户执行特定用户的命令时，短暂时间内转化为特定用户，从而拥有一些只有特定用户拥有的权限，从而使用一些特定用户可以的动作。

只针对可执行权限 x. x->s, 即只有文件的 owner 位的权限如下形式才可以完成此操作。

`-rws r-x r-x`

添加该权限：chmod u=rwxs file 或 chmod u+s file

参考-> 鸟哥私房菜 6.4.3

2.7.2 UID EUID

http://keren.blog.51cto.com/720558/144908

2.8 进程

ps **显示当前**所有进程的运行情况

将程序运行至后台..xxCommand &

表 2.7: ps 参数含义

符号	含义
l	长格式输出
u	按用户名和启动时间的顺序来显示进程
j	用任务格式来显示进程
f	用树形格式来显示进程
a	显示所有用户的所有进程 (包括其它用户) 如：ps a 显示现行终端机下的所有程序
x	显示无控制终端的进程
r	显示运行中的进程

参考：http://www.cnblogs.com/wangkangluo1/archive/2011/09/23/2185938.html

后台执行 Command &

特点: 是执行后**命令行不会一直占用**，可以执行其他命令. 例如

```
$firefox &
...

$Available..
```

其他

ctrl+z 暂停当前运行进程

ctrl+c 结束当前运行进程

top **实时动态显示**当前所有任务的资源占用情况

jobs 列出所有活动作业的状态信息

bg 设置在后台中运行作业

fg 设置在前台中运行作业

kill 发送信号给某个进程，对应的 c 函数也叫 kill();

killall 杀死指定名字的进程

2.9 环境/etc

2.9.1 基本概念

login **shell**[需要输入用户名和密码的] 启动时会读取一个或多个启动文件以配置系统环境：

- `/etc/profile` 适用于所有用户的全局配置脚本
- `~/.bash_profile` 用户个人启动文件，可扩展或重写
- `~/.profile`

non-login **shell** 则会读取以下文件

- `/etc/bash.bashrc` 适用于所有用户的全局配置脚本
- `~/.bashrc` 用户个人启动文件，可扩展或重写

2.9.2 相关命令

printenv 打印部分或全部的环境信息

表 2.8: 一些必要的环境变量

符号	含义
DISPLAY	运行图形界面环境时界面的名称
EDITOR	用于文本编辑的程序名称
SHELL	本机 shell 名称
HOME	本机主目录的路径名
PWD	当前工作目录
USER	用户名
PATH	以冒号分割的一个目录列表，**当用户输入一个可执行程序的名称时，会查找该目录列表**

PATH 变量 通常时由启动文件 /etc/profile 中的一段代码设定，但并不如此，取决于系统的发行版本

```
// 这段代码将$HOME/bin 添加到PATH值的尾部
   PATH = $PATH:$HOME/bin
```

还有一个常见的命令 export, 该命令会告诉 shell, 将 shell 的子进程使用PATH 变量的内容

```
   export PATH
```

修改环境变量时应当把修改放入以下文件

- .bash_profile 或者 其他等效文件(如 Ubuntu中的 .profile)

其他的改变则应录入.bashrc, 如

```
// .bashrc 文件增加如下代码
   umask 0002
   export HISTSIZE = 1000
   alias ll = 'ls␣-l␣--color=auto'
```

但是只要启动 shell 时才会读取.bashrc, 所以对.bashrc 做出的修改只有在关闭 shell 终端回话并重启的时候才会生效。当然我们也可以使用以下命令强制命令 bash 重新读取.bashrc 文件

```
   $source .bashrc
```

set 设置 shell 选项

export 将环境导出到随后要运行的程序中

alias 为命令创建一个别名.

2.10 VI 编辑器

2.10.1 模式转换

模式如下，所有在**使用命令的时候一般需要按 Esc 按键返回到命令模式下才可以.**. 而并非在插入模式与 ex 转义方式下。通过输入 vi 的插入命令（i）、附加命令（a）、打开命令（o）、替换命令（s）、修改命令 (c) 或取代命令（r）可以从命令方式进入输入方式。

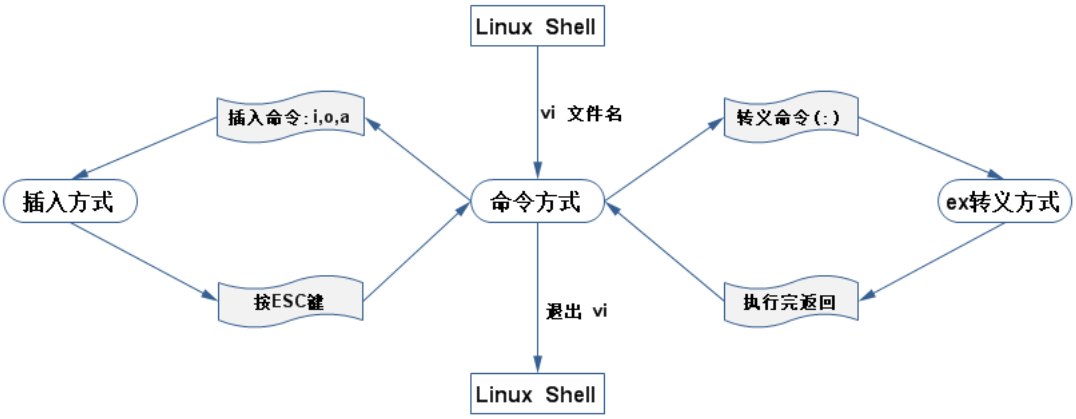


Fig 2.4: vi 模式转换示例

2.10.2 常用命令

http://blog.csdn.net/weixin_40910753/article/details/79077074

选中一块区域并缩进缩出

1. 按ESC 进入正常模式
2. 按shift+v 进入视觉模式
3. 2 不松手，移动上下键选择文本
4. gv> 缩进，命令可以重复执行，gv< 缩出

移动光标

数字 0 移动光标至本行开头，Home 也有此功能

$ 移动光标至本行末尾，End 也有此功能

num+G 移动光标到第 num 行

G 移动光标到文件最后一行

O 在当前行插入一行，小写则是后插入一行

删除操作

dd 删除当前行

num+dd 删除与本行在内的往后 num 行

dW 删除整个单词

d+num+G 删除当前行到第num 行

d+G 删除当前行到文件末尾

u 取消删除操作

复制粘贴

normal mode 命令在按返回后直接使用，不用使用:yy, 使用方式：

1. Esc
2. yy
3. p

yy 复制当前行

num+yy 复制连同当前行往后的 num 行

y+num+G 复制当前行到文件第 num 行

y+G 复制当前行到文件末尾

p 粘贴复制文本到当前光标前

J 合并后一行至当前行 https://stackoverflow.com/questions/2770739/vim-error-e492-not-

查找

'/keyword' 在当前文本向后查找 `keyword`

'?keyword' 在当前文本向前查找 `keyword`

n 寻找下一个

全局搜索和替换

表 2.9: ex 搜索替换使用规范

符号	含义
:	分号用于启动一条 ex 命令
%	确定作用范围，`%` 代表从文件的第一行到最后一行，`1,5` 代表第一行到第 5 行，`2,$` 代表从第二行到最后一行. 如果不指定范围的话，只会在当前行生效
s	指定了具体的操作- 本次操作为替换操作 (搜索和替换)
/find/replace	搜说 find 关机字，并将其替换为 replace
g	代指 global 全局的意思，也就是说对搜索到的每一行的每一个实例进行替换，是一个范围参数，如果 g 缺失，那么只替换每一行第一个符合条件的实例

http://blog.sina.com.cn/s/blog_736f1c59010136ry.html

查找删除注释

's/old/new/ ' 用 old 替换 new，替换当前行的第一个匹配

's/old/new/g ' 用 old 替换 new，替换当前行的所有匹配

'%s/old/new/' 用 old 替换 new，替换所有行的第一个匹配

'%s/old/new/g' 用 old 替换 new，替换整个文件的所有匹配

',5 s/^/#/g' 注释当前行到第 5 行

',+5 s/^/#/g' 注释当前行与之后的 5 行

'3,5 s/#//g' 解除 3-5 行的注释

编辑多文本

vi file1 file2

n 与 N 切换下一个文件

:buffers 显示当前打开的文件们

:buffer num 切换到文件 num

:e newfile [* 常用] 载入更多的文件

:tabe http://blog.csdn.net/achang21/article/details/44562253

文件间复制 跟普通复制粘贴一样，只是需要切换目标文件先

插入整个文件 `:r insertFile` 将 insertFile 的内容插入到光标位置之前

2.10.3 常用插件

ctags

http://blog.csdn.net/u013445530/article/details/46726109

1. 安装软件：`sudo apt-get install ctags`
2. 下载补全资源：`.h`
3. 生成 tags 文件：ctag 生成 `tags` 文件

4. 使用：ctrl+p 与 ctrl+n 使用

YouCompleteMe

2.11 存储介质挂载

2.11.1 挂载点的意义

每个 filesystem 都有独立的 inode、 block 、superblock 等信息，**这个文件系统**要能够**链接到目录树**才能 **被我们使用。将文件系统与目录树结合的动作我们称为"挂载"**。

重点是：**挂载点一定是目录**，该目录为进入该文件系统的入口。因此并不是你有任何文件系统都能使用，必须要"挂载"到目录树的某个目录后，才能够使用该文件系统的。

2.11.2 挂载

- windows -> 硬件设备映射为一个盘符
- linux -> 将硬件设备挂载成 (映射到) 一个目录

linux 的每个挂载了分区的目录就相当于 windows 系统中的盘符，比如/home/ftp 和 /oracle 目录我们就可以把她看做一个盘符和一个分区关联

http://blog.csdn.net/gongweijiao/article/details/8425629

mount 查询系统中已经挂载的设备

mount [-t 文件系统] [-o 特殊选项] 设备文件名 挂载目录

- -a 依据配置文件/etc/fstab 的内容，开机自动挂载
- -t 文件系统类型可以有 ext3, ext4, fat16, fat32 等
- -o 可以指定挂载的额外选项

-> 挂载U盘: 此盘在 windows 下格式化为 fat32 文件系统

1. 挂 u 盘之前，运行命令cat /proc/partitions, **看看现在系统中有哪些分区**。插上 u 盘以后，再次运行上述命令，看看多出来什么分区 (通常是 sda1 或者 sdb1)

2. 输入 `fdisk -l /dev/sda` 查看输出结果, **决定以什么文件系统挂载**。

```
#fdisk -l /dev/sda
 Disk /dev/sda: 131 MB, 131104768 bytes
 3 heads, 32 sectors/track, 2667 cylinders
 Units = cylinders of 96 * 512 = 49152 bytes
 Device Boot      Start         End      Blocks   Id System
/dev/sdb1   *           1        2668      128016    6 FAT16
```

3. 根据文件系统类型挂载, 如`mount -t msdos /dev/sdb1 /mnt/usb`

   可以使用`cat /proc/filesystems` 查看当前系统支持的文件系统格式

   - 如果是fat16 格式, 就用命令： `mount -t msdos /dev/sdb1 /mnt/usb`
   - 如果是fat32 格式,就用命令： `mount -t vfat /dev/sdb1 /mnt/usb -o iocharset=utf8`
   - 如果是ext2 格式, 就用命令： `mount -t ext2 /dev/sda1 /mnt/usb`
   - 如果是NTFS 格式, 就用命令： `mount -t ntfs /dev/sda1 /mnt/usb`

挂载实例：https://wenku.baidu.com/view/ef8aa226dd36a32d73758139.html

2.12 网络

ping **向网络主机发送特殊数据包** (ICMP ECHO_REQUEST). 多数网络设备收到该数据包后会做出回应, 通过此法可判断网络链接 (两者间) 是否正常. `ping baidu.com`

traceroute **跟踪网络数据包的传输路径**, 会显示文件通过网络从本地系统到指定主机过程中所有停靠点的列表。`traceroute slashdot.org`

netstat 检查网络设置以及相关统计数据, `-ie` 可以检查系统的网络接口信息, `-r` 可以显示内核的网络路由表 http://www.cnblogs.com/ggjucheng/archive/2012/01/08/2316661.html

netstat	options
netstat	-a (all) 显示所有选项，默认不显示LISTEN 相关 -t (tcp) 仅显示tcp 相关选项 -u (udp) 仅显示udp 相关选项 -n 拒绝显示别名，能显示数字的全部转化成数字 -l 仅列出有在 Listen (监听) 的服⊠状态 -p 显示建立相关链接的程序名 -r 显示路由信息，路由表 -e 显示扩展信息，例如uid 等 -s 按各个协议进行统计 -c 每隔一个固定时间，执行该netstat 命令

LISTEN和LISTENING 的状态只有用-a 或者-l 才能看到

- 列出所有 tcp 端口: netstat -at
- 列出所有 udp 端口: netstat -au
- 只列出所有处于监听状态 tcp 端口 : netstat -lt
- 显示tcp 和进程名称 netstat -pt

ftp:lftp:sftp File Transfer Protocol,ftp 服务器就是那些包含供网络上传、下载文件的机器

表 2.10: FTP 使用说明

命令	含义
ftp server	建立 FTP 连接，在命令上中先输入ftp 然后空格跟上 FTP 服务器的域名 'domain.com' 或者 IP 地址
anonymous	使用用户名密码登录, 绝大多数的 **FTP 服务器是使用密码保护的**，因此这些 FTP 服务器会询问'username' 和'password'. 如果你连接到被称作匿名 FTP 服务器，可以尝试anonymous 作为用户名以及使用空密码：Name: anonymous, Password:
目录操作	FTP 命令可以列出、移动和创建文件夹, 如同我们在本地使用我们的电脑一样。ls 可以打印目录列表，cd 可以改变目录，mkdir 可以创建文件夹
使用 FTP 下载文件	在下载一个文件之前，我们首先需要使用lcd 命令**设定本地接受目录位置**。lcd /home/user/yourdirectoryname 如果你不指定下载目录，文件将会下载到你登录 FTP 时候的工作目录。现在，我们可以使用命令 get **来下载文件**，比如：get file **文件会保存在**使用lcd 命令设置的目录位置
使用 FTP 上传文件	使用 put 命令上传文件：put file
关闭 FTP 链接	quit exit bye3 个命令都可以

wget 非交互式网络下载工具 wget http://www.linuxde.net/testfile.zip

*** 远程操作 ssh** 安全登录远程计算机:SSH 解决了与远程主机进行安全通信的两个基本问题：1、该协议可以验证远程主机的身份是否真实;2、该协议将本机与远程主机之间的通信内容全部加密

SSH 协议包括 2 部分：1、运行在远程主机上的 SSH 服务端，用来监听端口 **22** 上可能过来的连接请求;2、是本地系统上的 SSH 客户端，用来与远程服务器进行通信

关于 SSH 的相关命令有 scp，sftp, 分别表示远程拷贝和更简单高效快速的 ftp

例如：执行ssh happycast.net 命令，然后输入密码，接着就可以直接访问该服务器.. 但是每次连接都得输入密码我们可以使用以下步骤进行简化：

```
$ssh-keygen
$cd ~/.ssh
$ls
id_rsa id_rsa.pub

// 然后我们把公钥id_rsa.pub 上传至服务器/home/Usr1/.ssh/authorized_keys即可
```

```
$ssh-copy-id 服务器名(Usrl@happycasts.net)
```

*** 数据传输 rsync** 见归档备份

nc netcat 所做的就是在两台电脑之间建立链接并返回两个数据流，在这之后所能做的事就看你的想像力了。你能建立一个服务器，传输文件，与朋友聊天，传输流媒体或者用它作为其它协议的独立客户端。

http://blog.csdn.net/zhangxiao93/article/details/52705642

- 端口扫描
- ChatServer
- 文件传输

pv 是 Pipe Viewer 的简称，意思是通过管道显示数据处理进度的信息。这些信息包括已经耗费的时间，完成的百分比（通过进度条显示），当前的速度，全部传输的数据，以及估计剩余的时间。

http://www.jb51.net/LINUXjishu/409870.html

tcpdump http://www.cnblogs.com/ggjucheng/archive/2012/01/14/2322659.html

- 监视指定主机的数据包
  - 打印所有**进入或离开**sundown 的数据包:

    `tcpdump host sundown` 或 `tcpdump host 210.27.48.1`
  - 打印 helios **与** hot **或者与** ace 之间通信的数据包

    `tcpdump host helios and \( hot or ace \)` 或

    `tcpdump host 210.27.48.1 and \ (210.27.48.2 or 210.27.48.3 \)`
  - 打印 ace **与**任何不是helios 主机通信的 IP 数据包

    `tcpdump ip host ace and not helios` 或

    `tcpdump ip host 210.27.48.1 and ! 210.27.48.2`
  - 截获主机hostname **发送的**所有数据

    `tcpdump -i eth0 src host hostname`

- 监视所有**送到**主机hostname 的数据包

  ```
  tcpdump -i eth0 dst host hostname
  ```

- 获取主机10.2.4.1 在端口 23 上**接收或发出**的包：`tcpdump tcp port 23 and host 10.2.4.1`

- () 得使用转义字符进行，要么不识别

```
tcpdump \(src 172.17.14.98 and dst 172.17.15.112\) or \(src 172.17.15.112 and dst 172.17
```

iperf 可以测试TCP和UDP 带宽质量。iperf 可以报告带宽，延迟抖动和数据包丢失。

ifconfig

iptables

vmstate vmstat 命令是最常见的 **Linux/Unix 监控工具**，可以展现**给定时间间隔**的**服务器的状态值**, 包括服务器的 **CPU 使用率，内存使用，虚拟内存交换情况,IO 读写情况**

-> 使用：

```
vmstat [-a] [-n] [-S unit] [delay [ count]]

vmstat [-s] [-n] [-S unit]

vmstat [-m] [-n] [delay [ count]]

vmstat [-d] [-n] [delay [ count]]

vmstat [-p disk partition] [-n] [delay [ count]]

vmstat [-f]
```

参数	含义
delay	刷新时间间隔。如果不指定，只显示一条结果。
count	刷新次数。如果不指定刷新次数，但指定了刷新时间间隔，这时刷新次数为无穷。
-a	开启显示 active/inactive memory
-f	显示此系统启动以来的 forks 的总数，包括 fork、vfork 和 clone system calls
-m	显示 slabinfo 信息
-n	只显示头信息，不周期性显示. 也就是说开启这个参数，只显示头部信息一次。
-s	显示各种事件计数器表和内存统计信息，这显示不重复
-d	显示磁盘统计数据
-w	可以扩大字段长度，当内存较大时，默认长度不够完全展示内存。
-p	显示磁盘分区数据
-S	参数 S 控制输出性能指标的单位，k(1000) K(1024) 或 M(1048576) 默认单位为 K (1024 bytes)

http://www.cnblogs.com/tommyli/p/3746187.html

http://www.cnblogs.com/ggjucheng/archive/2012/01/05/2312625.html

2.13 正则表达式

$$\text{正则表达式} = \begin{cases} \text{字符类} \\ \text{数量限定类} \\ \text{位置限定类} \end{cases}$$

http://www.cnblogs.com/hanxiaoyu/p/5759477.html

2.13.1 字符类

字符	含义	例子
.	匹配任意一个字符	abc. 可以匹配 abcd、 abc9
[]	匹配括号中的任意一个字符	[abc]d 可以匹配 ad、 bd、cd
-	在[] 中使用，表示字符范围	[0-9 a-f A-F] 可以匹配一位 16 进制数字
^	位于括号内的开头, 匹配括号中的字符外的任意一个字符	[^xy] 可以匹配除x、y 之外的任一字符
[[:xx:]]	预定义的一些命名字符	[[:digit:]] 匹配一个数字

[\d] 等价于 [0-9]

/ 只是在某些语言中作为正则的**边界符**, 如 sed 在匹配正则时，不仅要指定 r, 正则还要以'/' 开头

2.13.2 数量限定类

字符	含义	例子
?	紧跟在他前面的单元应匹配 0 次或 1 次	[0-9]?\.[0-9] 匹配 0.0 2.3 .5 等, 特殊字符. 需要加转义字符
+	紧跟在他前面的单元应匹配 1 次或多次	[a-zA-Z0-9_-]+ @ [a-zA-Z0-9_.-]+ \.[a-zA-Z0-9_.-]+ 匹配 email 地址
*	紧跟在他前面的单元应匹配 0 次或多次	[0-9]* 匹配至少 1 位数字
{N}	紧跟在他前面的单元应精确匹配 N 次	[1-9][0-9]{2} 匹配 100 到 999 的整数
{N,}	紧跟在他前面的单元应匹配至少 N 次	[1-9][0-9]{2,} 匹配 3 位及 3 位以上的整数
{,M}	紧跟在他前面的单元应匹配最多 M 次	[0-9]{,1} 最多匹配一次数字
{N,M}	紧跟在他前面的单元应匹配至少 N 次, 最多 M 次	[0-9]{1,3} 表示 0-9 数字至少匹配 1 次，最多匹配 3 次

2.13.3 位置限定类

字符	含义	例子
^	匹配行首的位置	^content 匹配行首为 content 的行
$	匹配行末的位置	;$ 匹配行末尾为 ; 的行，^$ 匹配空行
\<	匹配单词开头的位置	\<th 匹配.. this.., 但不匹配 tenth 等
\>	匹配单词结尾的位置	p\> 匹配 leap, 但不匹配 parent
\b	匹配单词的开头\bx 或结尾x\b 的位置	\b at 匹配 at, 但不匹配 batch
\B	匹配非单词开头\Bx 或结尾的位置	\B at 匹配 battery, 但不匹配 attend、hat

2.13.4 特殊字符

字符	含义	例子
\	转义字符	([0-9]{1,3}\.){3}[0-9]{1,3} 匹配 IP 地址
()	将正则表达式的一部分括起来组成一个单元，可以对整个单元使用数量限定符	
|	链接两个子表达式，表示或的关系	

2.14 搜索命令

*** 文件搜索命令 locate** 通过 [部分]文件名 查找**文件**的具体位置，搜索速度比find 快。在后台数据库中按文件名搜索，搜索数据速度更快，但是后台数据库更新一般是 1 天更新一次，但是可以使用updatedb 强制更新后台数据库。

但是使用updatedb 时遵循/etc/updatedb.conf 配置文件的规则

- 开启搜索限制 PRUNE_BIND_MOUTS = "yes"
- 搜索时，不搜索的文件系统 PRUNEFS =
- 搜索时，不搜索的文件类型 PRUNENAMES =
- 搜索时，不搜索的路径 PRUNEPATHS =

当配置文件开启搜索规则时，update 命令执行后，locate 会在搜索时遵照 updatedb.conf 规则进行剔除搜索。

选项->

- -i：忽略大小写
- -c：不输出文件名，仅计算找到的文件数量
- -l：仅输出结果的几行，例如输出 5 行则是 -l 5

*** 文件搜索命令 find** 在文件系统目录框架中查找文件 (**完全匹配**)

```
find path -option [-print -delete -ls -quit] [[-exec -ok] command{}[; +]]
```

find	options	含义
find	-name filename	查找名为filename 的文件
	-perm	按执行权限来查找
	-user username	按文件属主来查找
	-group groupname	按组来查找
	-mtime -n +n	按文件更改时间来查找文件, -n 指n 天以内, +n 指n 天以前
	-atime -n +n	按文件访问时间来查
	-ctime -n +n	按文件创建时间来查找文件, -n 指n 天以内, +n 指n 天以前
	-newer f1 !f2	查更改时间比f1 新但比f2 旧的文件
	-type b/d/c/p/ll/f	查是块设备、目录、字符设备、管道、符号链接、普通文件
	-size n[c]	查长度为n 块 [或n 字节] 的文件,-小于size的, +大于size的, find /etc -size +20k -a -size -50k
	-inum fileNum	查找文件节点号为 fileNum 的文件
	-mount	查文件时不跨越文件系统mount 点
	-and -or	-and 条件与 -or 条件或

- -print 将查找到的文件输出到标准输出
- -exec command {} \; ——将查到的文件执行 command 操作,{} 为查找的结果, 也是新命令的参数。
- -ok 和-exec 的作用相同, 只不过以一种更为安全的模式来执行该参数所给出的 shell 命令, 在执行每一个命令之前，都会给出提示，让用户来确定是否执行。

通配符*任意内容　?任意一个字符　[]任意一个括号内的字符

http://www.cnblogs.com/wanqieddy/archive/2011/06/09/2076785.html

*** 字符串搜索命令 grep**　grep 命令是一种强大的**文本**搜索工具，它能使用正则表达式搜索文本，并把匹配的行打印出来。grep 全称是Global Regular Expression Print，表示全局正则表达式版本，它的使用权限是所有用户 (**包含匹配**)

grep	options	regex pattern
	-c 只输出匹配行的计数 -I 不区分大小写 (只适用于单字符) -e 并列使用多个 -e 参数可以实现**或**条件 -h 查询多文件时不显示文件名 -l 查询多文件时只输出包含匹配字符的文件名 -n 显示匹配行及行号 -s 不显示不存在或无匹配文本的错误信息 -v 显示不包含匹配文本的所有行 -R 递归的对目录下的所有文件 (包括子目录) 进行 grep	\ 忽略正则表达式中特殊字符的原有含义 ^ 匹配正则表达式的开始行 netstat -an | grep -e EST -e WAIT $ 匹配正则表达式的结束行 \< 从匹配正则表达式的行开始 \> 到匹配正则表达式的行结束 [ ] 单个字符，如 [A] 即 A 符合要求 * 有字符，长度可以为 0 [-] 范围，如 [A-Z]，即 A、B、C 一直到 Z 都符合要求

命令搜索命令 whereis 与 which　which 指令会在环境变量$PATH 设置的目录里查找符合条件的文件。

whereis 指令会在特定目录中查找符合条件的文件。这些文件的属性应属于原始代码，二进制文件，或是帮助文件。

xargs　从标准输入中建立、执行命令行, 一般结合管道 | 在结果中使用。

```
find ~ -type f -name 'foo*' -print |xargs ls -l
```

stat　显示文件或文件系统的状态

2.15 文本相关

2.15.1 sed

sed 是一个很好的文件处理工具，本身是一个管道命令，主要是**以行为单位进行处理**，可以将数据行进行替换、删除、新增、选取等特定工作

```
sed [options] 'command' 输入文本
```

选项

- `-n` 使用安静 (silent) 模式。在一般 sed 的用法中，所有来自 STDIN 的资料一般都会被列出到萤幕上。但如果加上 -n 参数后，则只有经过 sed 特殊处理的那一行 (或者动作) 才会被列出来。
- `-e` 直接在指令列模式上进行 sed 的动作编辑；
- `-f` 直接将 sed 的动作写在一个档案内，-f filename 则可以执行 filename 内的 sed 动作；
- `-r` sed 的动作支援的是延伸型正规表示法的语法。(预设是基础正规表示法语法)
- `-i` 直接修改读取的档案内容，而不是由萤幕输出。

命令

- `a` **新增**，a 的后面可以接字串，而这些字串会在新的一行出现 (目前的下一行)
- `c` **取代**，c 的后面可以接字串，这些字串可以取代 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 的后面可以接字串，而这些字串会在新的一行出现 (目前的上一行)；
- `p` **列印**，亦即将某个选择的资料印出。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`

http://www.cnblogs.com/dong008259/archive/2011/12/07/2279897.html

2.15.2 awk

awk 是一个非常棒的数字处理工具。相比于 sed 常常作用于一整行的处理，awk 则比较**倾向于将一行分为数个"字段"来处理**。运行效率高，而且代码简单，对格式化的文本处理能力超强

http://blog.jobbole.com/109089/

2.15.3 cat

连接文件并打印到标准输出

- -A, –show-all 等价于 -vET
- -b, –number-nonblank 对非空输出行编号
- -e 等价于 -vE
- -E, –show-ends 在每行结束处显示 $
- -n, –number 对输出的所有行编号
- -s, –squeeze-blank 不输出多行空行
- -t 与 -vT 等价
- -T, –show-tabs 将跳字符显示为 ^I

2.15.4 sort

对**文本行**排序..

2.15.5 uniq

报告并省略重复行

2.15.6 cut

从每一行中移除文本区域

2.15.7 paste

合并文件文本行

2.15.8 join

基于某个共享字段来联合两个文件的文本行

2.15.9 comm

逐行比较两个已经排序好的文件

2.15.10 diff

逐行比较文件

2.15.11 patch

对原文件打补丁

2.15.12 tr

转换或删除字符

2.15.13 aspelx

交互式拼写检查器

2.16 归档备份

2.16.1 压缩与解压缩

常见格式有.zip .gz .bz2 .tar.gz .tar.bz2

- .zip
  - **压缩**
    1. zip 压缩文件名　源文件：压缩文件
    2. zip -r 压缩文件名 源目录：压缩目录
  - **解压缩** unzip 压缩文件
- .gz
  - **压缩**
    1. gzip 源文件：压缩为.gz 格式的压缩文件，源文件会消失
    2. gzip -c 源文件 > 压缩文件：压缩为.gz 格式，源文件保留
    3. gzip -r 目录：压缩目录下所有子文件，但是不能压缩目录
  - **解压缩**
    1. gzip -d 压缩文件：解压缩文件
    2. gunzip 压缩文件：解压缩文件
    3. 解压目录加-r 即可
- .bz2
  - **压缩**，不能压缩目录
    1. bzip2 源文件：压缩为.bz2 格式，不保留源文件
    2. bzip2 -k 源文件：压缩后保留原文件
  - **解压缩**，加关键字-k 保留压缩文件
    1. bzip2 -d 压缩文件
    2. bunzip2 压缩文件
- **打包** tar -cvf 打包文件名 源文件
  1. -c 打包
  2. -v 显示过程
  3. -f 指定打包后的文件名

  tar -cvf long.tar long

  将文件直接打包后压缩为.tar.gz 格式，-z 前缀
  1. -z **压缩**为.tar.gz:tar -zcvf 压缩文件名　原文件
  2. -x **解压缩**.tar.gz:tar -zxvf 压缩包名

将文件直接打包后压缩为.tar.bz2 格式，-j 前缀

1. -z **压缩**为.tar.bz2:tar -jcvf 压缩文件名　原文件
2. -x **解压缩**.tar.bz2:tar -jxvf 压缩包名
3. -C 解压到指定路径下tar -jxvf test.tat.bz2 -C /tmp/

2.16.2 同步文件和目录

rsync　在对 rsync 服务器配置结束以后，下一步就需要在客户端发出 rsync 命令来实现将服务器端的文件备份到客户端来，但是也可以同步本地的文件夹等，命令格式如下：

```
rsync [OPTION]... SRC DEST
rsync [OPTION]... SRC [USER@]HOST:DEST
rsync [OPTION]... [USER@]HOST:SRC DEST
rsync [OPTION]... [USER@]HOST::SRC DEST
rsync [OPTION]... SRC [USER@]HOST::DEST
rsync [OPTION]... rsync://[USER@]HOST[:PORT]/SRC [DEST]
```

对应于以上六种命令格式，rsync 有六种不同的工作模式：

- 拷贝本地文件。当SRC 和DES 路径信息都不包含有单个冒号":"分隔符时就启动这种工作模式。如：rsync -a /data /backup
- 使用一个远程 shell 程序 (如 rsh、ssh) 来实现将本地机器的内容拷贝到远程机器。当DST 路径地址包含单个冒号":"分隔符时启动该模式。如：rsync -avz *.c foo:src
- 使用一个远程 shell 程序 (如 rsh、ssh) 来实现将远程机器的内容拷贝到本地机器。当SRC 地址路径包含单个冒号":"分隔符时启动该模式。如：rsync -avz foo:src/bar /data
- 从远程 rsync 服务器中拷贝文件到本地机。当SRC 路径信息包含"::"分隔符时启动该模式。如：rsync -av root@172.16.78.192::www /databack
- 从本地机器拷贝文件到远程rsync 服务器中。当DST 路径信息包含"::"分隔符时启动该模式。如：rsync -av /databack root@172.16.78.192::www
- 列远程机的文件列表。这类似于rsync 传输，不过只要在命令中省略掉本地机信息即可。如：rsync -v rsync://172.16.78.192/www

http://www.cnblogs.com/subsir/articles/2565373.html

2.17 软件包管理

2.17.1 已编译文件

linux 的文件安装换句话说就是把相应的执行文件拷贝到系统执行命令的搜索目录中, 即$PATH 下, 或者在PATH 下创建可执行软件的符号链接

- 将文件拷贝到PATH 下任意目录

```
INSTALL_DIR=/usr/local/bin

all:
-@cd src && make

.PHONY: clean

clean:
-@cd src && make clean

install:
-cp -f bin/* $(INSTALL_DIR)
```

- ln -s 可执行软件位置 $PATH下任意目录/别名 : ln -s ~/.sublime/sublime_text ~/bin/subl

2.17.2 源码安装

固定 3 步 :

1. ./configure
2. make
3. sudo make install

2.17.3 deb 包安装

sudo dpkg -i xx.deb

Ubuntu

```
sudo apt-get install xx
```

2.18 参考

http://www.linuxcommand.org/index.php

linux 高级编程:http://guojing.me/linux-kernel-architecture/tags/#do_group_exit

第三章　Shell 脚本

练习题：https://wenku.baidu.com/view/e6664cafbd64783e08122b4f.html

https://zhangge.net/4023.html

参考：http://www.cnblogs.com/90zeng/p/shellNotes.html

在线解释王者：https://explainshell.com

3.1　Shell 基本知识

3.1.1　Shell

Shell 就是一个命令行解释器，它的作用是解释执行用户的命令，用户输入一条命令,Shell 就解释执行一条，这种方式称为交互式 (Interactive)

Shell 还有一种执行命令的方式称为**批处理 (Batch)**，*用户事先写一个 Shell 脚本 (Script)*，**其中有很多条命令，让 Shell 一次把这些命令执行完，而不必一条一条地敲命令**。Shell 脚本和编程语言很相似，也有变量和流程控制语句，包括循环和分支。但 **Shell 脚本是解释执行的，不需要编译,Shell 程序从脚本中一行一行读取并执行这些命令，相当于一个用户把脚本中的命令一行一行敲到 Shell 提示符下执行**。作为程序设计语言，它虽然不是 Linux 系统内核的一部分，但它调用了系统内核的大部分功能来执行程序、创建文档并以并行的方式协调各个程序的运行

3.1.2　Shell 执行脚本

shell 执行脚本是一门解释性语言、批量化处理语言，大大的节省了工作成本，shell **脚本第一行必须以#! 开头**，*它表示该脚本使用后面的解释器解释执行*。

Example:　//script.sh 注：这是一个文本文件

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

```
#注意 echo 默认为换行输出，如果不换行+\c
echo "this␣is␣a␣test"
ls
ls -l
echo "there␣are␣all␣files"
```

执行方式

```
//第一种执行方式:
[admin@localhost Shell]$ chmod +x script.sh
[admin@localhost Shell]$ ./script.sh

//第二种执行方式：
[admin@localhost Shell]$ /bin/bash script.sh
//等价于：
[admin@localhost Shell]$ shell script.sh
```

执行过程

Shel 会 fork 一个子进程并调用 exec 执行./script.sh 这个程序,exec 系统调用应该把子进程的代码段替换成./script.sh 程序的代码段, 并从它的 _start 始执行。然而 script.sh 是个文本文件, 根本没有代码段和 _start 函数, 怎么办呢? 其实 exec 还有另外一种机制, 如果要执行的是一个文本文件, 并且第一行指定了解释器, 则用解释器程序的代码段替换当前进程, 并且从解释器的 _start 开始执行, 而这个文本文件被当作命令行参数传给解释器。因此, 执行上述脚本相当于执行程序！

什么是“子程序”呢？就是说，在我目前这个 shell 的情况下，去启用另一个新的 shell，新的那个 shell 就是子程序

这个程序概念与变量有啥关系啊？关系可大了！因为子程序仅会继承父程序的环境变量，子程序不会继承父程序的自订变量

1. 交互式进程（父进程）创建一个子进程用于执行脚本，父进程等待子进程终止
2. 子进程程序替换 bash 解释器
3. 读取 shell 脚本的命令，将其以参数传递的方式传递给 bash 解释器
4. 子 bash 对 shell 脚本传入的参数进行读取，读一行识别到它是一个命令，则再创建一个子进程，子 bash 等待该新进程终止

5. 新进程执行该命令，执行完后将结果交给子进程

6. 子进程继续读取命令，创建新进程，新进程执行该命令，将结果返回给子进程，直到执行完最后一条命令

7. 子进程终止，将结果返回给交互式父进程

注意 注意：像 export、cd、env、set 这些内置命令，在键入命令行后，交互式进程不会创建子进程，而是调用 bash 内部的函数执行这些命令，改变的是交互式进程。

如果在命令行下，将多个命令用括号括起来，并用分号隔开来执行，交互式进程依然会创建一个子 shell 执行括号中的命令:

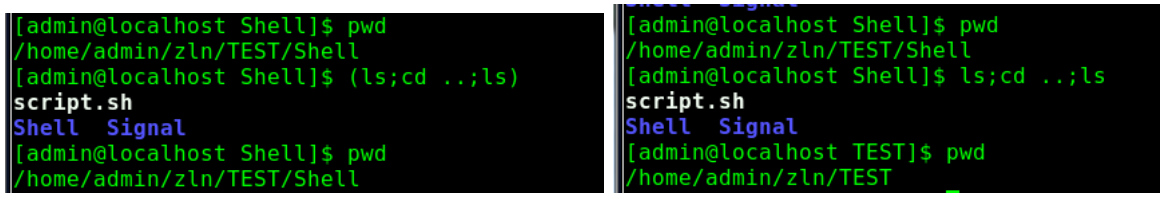


Fig 3.1: (comands) VS Comands

. 或者 source 这两个命令是 Shell 的内建命令，**这种方式不会创建子 Shell，而是直接在交互式 Shell 下逐行执行脚本中的命令**

Example

```
#!/bin/bash
ls
echo "#################"
cd ..
ls
```

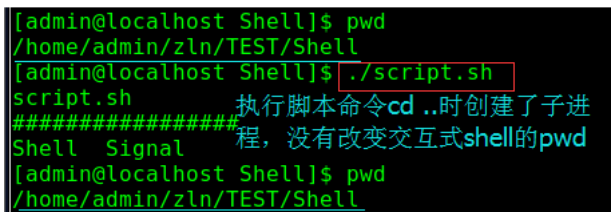


Fig 3.2: without use . Or source Command

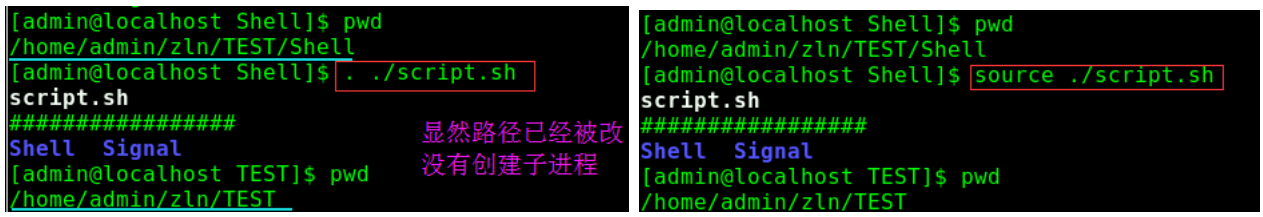


Fig 3.3: Use . Or source Command

3.1.3 Shell 变量

shell 变量不需要进行任何声明，直接定义即可，因为 shell 变量的值实际上都是字符串（对于没有定义的变量默认是一个空串）。定义的时候 shell 变量由大写字母加下划线组成，并且定义的时候等号两边不能存在空格，否则会被认为是命令！

shell 变量的种类

环境变量： shell 进程的环境变量可以从当前 shell 进程传给 fork 出来的子进程

本地变量： 只存在于当前 shell 进程

利用 printenv 可以显示当前 shell 进程的环境变量；利用 set 命令可以显示当前 shell 进程中的定义的所有变量（包括环境变量和本地变量）和函数。

一个 shell 变量定义后仅存在于当前 Shell 进程，是一个本地变量。用 export 命令可以把本地变量导出为环境变量。用 unset 命令可以删除已定义的环境变量或本地变量。

```
//分步 先定义后导出
COUNT=5
export COUNT

//一步完成定义和导出环境变量
export COUNT=5

//删除已经定义的环境变量
unset COUNT
```

变量引用： 引用 shell 变量要用到 $ 符号，加{} 可以防止歧义。

```
COUNT=5
echo $COUNT
echo ${COUNT}911
```

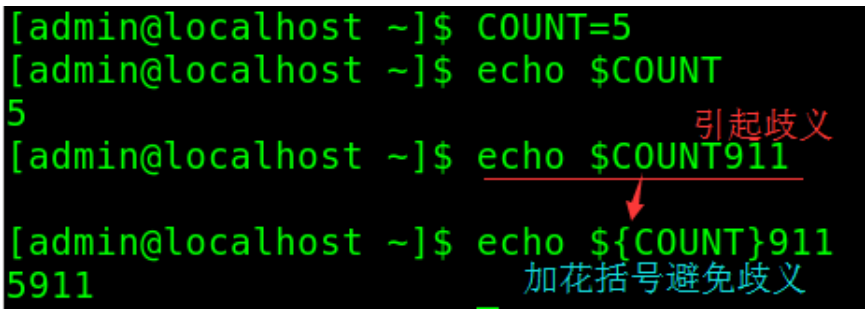


Fig 3.4: Shell Variable Use

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

下面的例子尝试更改只读变量，结果报错：

```
#!/bin/bash
myUrl="http://www.w3cschool.cc"
readonly myUrl
myUrl="http://www.runoob.com"

//运行脚本，结果如下：
/bin/sh: NAME: This variable is read only.
```

删除变量 使用 unset 命令可以删除变量。语法：unset variable_name, 变量被删除后不能再次使用。unset 命令不能删除只读变量。

```
#!/bin/sh
myUrl="http://www.runoob.com"
unset myUrl
echo $myUrl

// 以上没输出
```

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

- **单引号字符串的限制:**
  1. 单引号里的任何字符都会原样输出，单引号字符串中的变量是无效的；
  2. 单引号字串中不能出现单引号（对单引号使用转义符后也不行）。
- **双引号字符串的好处:**
  1. **双引号里可以有变量**

2. **双引号里可以出现转义字符**

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

// 拼接字符串
your_name="qinjx"
greeting="hello, "$your_name" !"
greeting_1="hello, ${your_name} !"
echo $greeting $greeting_1

// 获取字符串长度
string="abcd"
echo ${#string} #输出 4

// 提取子字符串：以下实例从字符串第 2 个字符开始截取 4 个字符
string="runoob is a great site"
echo ${string:1:4} # 输出 unoo

// 查找子字符串:查找字符 "i 或 s" 的位置：
string="runoob is a great company"
echo 'expr index "${string}" is' //# 输出 8
```

Shell 数组 bash **支持一维数组**（不支持多维数组），并且没有限定数组的大小。类似与 C 语言，数组元素的下标由 0 开始编号。获取数组中的元素要利用下标，下标可以是整数或算术表达式，其值应大于或等于 0

- **定义数组**：用括号来表示数组，数组元素用”空格”符号分割开。定义数组的一般形式为：数组名 =(值 1 值 2 ... 值 n)，还可以单独定义数组的各个分量
- **读取数组**：${数组名[下标]}
- **求数组长度**：获取数组长度的方法与获取字符串长度的方法相同,length=${#array_name[@]}

```
// 定义数组
array_name=(value0 value1 value2 value3)
array_name=(
    value0
    value1
    value2
    value3
```

```
)
// 单独定义数组的各个分量
array_name[0]=value0
array_name[1]=value1
array_name[n]=valuen

// 读取数组各元素和所有元素
valuei=${array_name[i]}
// 使用@符号可以获取数组中的所有元素
echo ${array_name[@]}

// 获取数组的长度
# 取得数组元素的个数
length=${#array_name[@]}
# 或者
length=${#array_name[*]}
# 取得数组单个元素的长度
lengthn=${#array_name[n]}
```

Shell 注释 以# 开头的行为注释行

3.1.4 Shell 通配符、命令代换、单引号、双引号

见基础相关拓展.

3.1.5 Shell 传递参数

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

- $#：参数个数
- $$：当前 shell 的 PID 进程 ID
- $@和$* 均表示所有参数，形式有所不同。$@："$1" "$2" ... "$n"；$*: "$1 $2 ... $n"。

```
//以下实例我们向脚本传递三个参数，并分别输出，其中 $0 为执行的文件名
#!/bin/bash
echo "Shell␣传递参数实例！";
echo "执行的文件名：$0";
```

```
echo "第一个参数为：$1";
echo "第二个参数为：$2";
echo "第三个参数为：$3";

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

3.1.6 Shell 各种括号

http://blog.csdn.net/taiyang1987912/article/details/39551385

( )-小括号

1. **命令组**: **括号中的命令将会新开一个子**shell 顺序执行，所以括号中的变量不能够被脚本余下的部分使用。括号中多个命令之间用分号隔开，最后一个命令可以没有分号，各命令和括号之间不必有空格

2. **命令替换**: 等同于`cmd`，shell 扫描一遍命令行，发现了$(cmd) 结构，便将$(cmd) 中的cmd 执行一次，得到其标准输出，再将此输出放到原来命令。有些shell 不支持，如 tcsh。

3. 用于初始化数组。如：array=(a b c d)

(( ))-双小括号

1. **整数扩展**。这种扩展计算是整数型的计算，不支持浮点型。((exp)) 结构扩展并**计算一个算术表达式的值**，如果表达式的结果为 0，那么返回的退出状态码为 1，或者是” 假”，而一个非零值的表达式所返回的退出状态码将为 0，或者是”true”。若是逻辑判断，表达式exp 为真则为 1，假则为 0

2. 只要括号中的运算符、表达式符合 C 语言运算规则，都可用在$((exp)) 中，甚至是三目运算符。作不同进位 (如二进制、八进制、十六进制) 运算时，**输出结果全都自动转化成了十进制**。如：echo $((16#5f)) 结果为95 (16 进位转十进制)

3. 单纯用 (( )) 也可**重定义变量值**，比如 a=5; ((a++)) 可将 $a 重定义为 6

4. **常用于算术运算比较，双括号中的变量可以不使用**$ 符号前缀。括号内支持多个表达式用逗号分开。只要括号中的表达式符合 C 语言运算规则，比如可以直接使用for((i=0;i<5;i++))，如果不使用双括号，则为for i in `seq 0 4`或者for i in {0..4}。再如可以直接使用if ((`$i<5`))，如果不使用双括号，则为if [$i -lt 5]

[ ]-中括号

1. bash 的内部命令，[和test 是等同的。如果我们不用绝对路径指明，通常我们用的都是 bash 自带的命令。if/test 结构中的**左中括号**是调用test 的命令标识，**右中括号**是关闭条件判断的。这个命令把它的参数作为比较表达式或者作为文件测试，并且根据比较的结果来返回一个退出状态码。if/test 结构中并不是必须右中括号，但是新版的 Bash 中要求必须这样

2. Test和[] 中可用的比较运算符**只有**==和!=，**两者都是用于字符串比较的**，不可用于整数比较，**整数比较只能**使用-eq，-gt 这种形式。*无论是字符串比较还是整数比较都不支持大于号小于号*。如果实在想用，对于字符串比较可以使用转义形式，如果比较"ab"和"bc"：[ab \< bc]，结果为真，也就是返回状态为 0。[ ] 中的逻辑与和逻辑或使用-a 和-o 表示。

3. **字符范围**。用作正则表达式的一部分，描述一个匹配的字符范围。作为test 用途的中括号内**不能使用正则**

4. 在一个array 结构的上下文中，中括号用来引用**数组**中每个元素的**编号**

[[ ]]-双中括号

1. [[是 bash 程序语言的关键字。并不是一个命令，[[ ]] 结构比[ ] 结构更加通用。在[[和]]之间所有的字符都不会发生文件名扩展或者单词分割，**但是会发生参数扩展和命令替换**

2. 支持**字符串的模式匹配**，使用=~ 操作符时甚至支持 shell 的正则表达式。字符串比较时可以把右边的作为一个模式，而不仅仅是一个字符串，比如[[hello == hell?]]，结果为真。[[ ]] 中匹配字符串或通配符，不需要引号。

3. 使用[[...]] **条件判断结构**，而不是[...]，**能够防止**脚本中的许多逻辑错误。比如，&&、|、< 和 >| 操作符能够正常存在于[[ ]] 条件判断结构中，但是如果出现在[ ] 结构中的话，会报错。比如可以直接使用if [[$a != 1 && $a != 2]]，如果不适用双括号，则为if [$a -ne 1] && [$a != 2] 或者if [$a -ne 1 -a $a != 2]

4. bash 把双中括号中的表达式看作一个单独的元素，并返回一个退出状态码

```
if ($i<5)
if [ $i -lt 5 ]
if [ $a -ne 1 -a $a != 2 ]
if [ $a -ne 1] && [ $a != 2 ]
if [[ $a != 1 && $a != 2 ]]

for i in $(seq 0 4);do echo $i;done
for i in `seq 0 4`;do echo $i;done
for ((i=0;i<5;i++));do echo $i;done
for i in {0..4};do echo $i;done
```

{ }-大括号

{} 常规用法

1.

2.

3.

{} 特殊的替换用法

1.

2.

3.

{} 模式匹配替换用法

1.

2.

3.

{} 字符串提取和替换

1.

2.

3.

3.1.7 参考

基本使用参考博客http://www.cnblogs.com/Lynn-Zhang/p/5758287.html。

使用教程：http://c.biancheng.net/cpp/view/6998.html

菜鸟教程-使用教程 2：http://www.runoob.com/linux/linux-shell.html

Bash 在线运行网址：http://www.runoob.com/try/runcode.php?filename=helloworld&type=bash

3.2 Shell 运算操作

3.2.1 算数运算

```
#!/bin/bash

a=10
b=20

val=`expr $a + $b`
echo "a␣+␣b␣:␣$val"

val=`expr $a - $b`
echo "a␣-␣b␣:␣$val"

//乘号(*)前边必须加反斜杠(\)才能实现乘法运算
val=`expr $a \* $b`
echo "a␣*␣b␣:␣$val"

val=`expr $b / $a`
echo "b␣/␣a␣:␣$val"

val=`expr $b % $a`
echo "b␣%␣a␣:␣$val"

if [ $a == $b ]
then
```

```
    echo "a 等于 b"
fi
if [ $a != $b ]
then
    echo "a 不等于 b"
fi

/////////////// Result -->//////////////////////
a + b : 30
a - b : -10
a * b : 200
b / a : 2
b % a : 0
a 不等于 b
```

3.2.2 关系运算

```
#!/bin/bash

a=10
b=20

if [ $a -eq $b ]
then
    echo "$a -eq $b : a 等于 b"
else
    echo "$a -eq $b: a 不等于 b"
fi
if [ $a -ne $b ]
then
    echo "$a -ne $b: a 不等于 b"
else
    echo "$a -ne $b : a 等于 b"
fi
if [ $a -gt $b ]
then
    echo "$a -gt $b: a 大于 b"
else
    echo "$a -gt $b: a 不大于 b"
fi
if [ $a -lt $b ]
then
    echo "$a -lt $b: a 小于 b"
```

```
else
    echo "$a -lt $b: a 不小于 b"
fi
if [ $a -ge $b ]
then
    echo "$a -ge $b: a 大于或等于 b"
else
    echo "$a -ge $b: a 小于 b"
fi
if [ $a -le $b ]
then
    echo "$a -le $b: a 小于或等于 b"
else
    echo "$a -le $b: a 大于 b"
fi

//////////////////////////Result ->////////////////////////////////
10 -eq 20: a 不等于 b
10 -ne 20: a 不等于 b
10 -gt 20: a 不大于 b
10 -lt 20: a 小于 b
10 -ge 20: a 小于 b
10 -le 20: a 小于或等于 b
```

3.2.3 布尔运算

```
#!/bin/bash

a=10
b=20

if [ $a != $b ]
then
    echo "$a != $b: a 不等于 b"
else
    echo "$a != $b: a 等于 b"
fi
if [ $a -lt 100 -a $b -gt 15 ]
then
    echo "$a -lt 100 -a $b -gt 15 : 返回 true"
else
    echo "$a -lt 100 -a $b -gt 15 : 返回 false"
fi
```

```
if [ $a -lt 100 -o $b -gt 100 ]
then
    echo "$a -lt 100 -o $b -gt 100 : 返回 true"
else
    echo "$a -lt 100 -o $b -gt 100 : 返回 false"
fi
if [ $a -lt 5 -o $b -gt 100 ]
then
    echo "$a -lt 100 -o $b -gt 100 : 返回 true"
else
    echo "$a -lt 100 -o $b -gt 100 : 返回 false"
fi
////////////////////Result ->/////////////////////////////////
10 != 20 : a 不等于 b
10 -lt 100 -a 20 -gt 15 : 返回 true
10 -lt 100 -o 20 -gt 100 : 返回 true
10 -lt 100 -o 20 -gt 100 : 返回 false
```

3.2.4 逻辑运算

```
#!/bin/bash

a=10
b=20

if [[ $a -lt 100 && $b -gt 100 ]]
then
    echo "返回 true"
else
    echo "返回 false"
fi

if [[ $a -lt 100 || $b -gt 100 ]]
then
    echo "返回 true"
else
    echo "返回 false"
fi

////////////////////////Result->////////////////////////////
返回 false
返回 true
```

3.2.5 字符串运算

```
#!/bin/bash
a="abc"
b="efg"

if [ $a = $b ]
then
   echo "$a␣=␣$b␣:␣a␣等于␣b"
else
   echo "$a␣=␣$b:␣a␣不等于␣b"
fi
if [ $a != $b ]
then
   echo "$a␣!=␣$b␣:␣a␣不等于␣b"
else
   echo "$a␣!=␣$b:␣a␣等于␣b"
fi
if [ -z $a ]
then
   echo "-z␣$a␣:␣字符串长度为␣0"
else
   echo "-z␣$a␣:␣字符串长度不为␣0"
fi
if [ -n $a ]
then
   echo "-n␣$a␣:␣字符串长度不为␣0"
else
   echo "-n␣$a␣:␣字符串长度为␣0"
fi
if [ $a ]
then
   echo "$a␣:␣字符串不为空"
else
   echo "$a␣:␣字符串为空"
fi
////////////////////////////Result-->////////////////////////////
abc = efg: a 不等于 b
abc != efg : a 不等于 b
-z abc : 字符串长度不为 0
-n abc : 字符串长度不为 0
abc : 字符串不为空
```

3.2.6 文件测试运算

```
#!/bin/bash

file="/var/www/runoob/test.sh"
if [ -r $file ]
then
    echo "文件可读"
else
    echo "文件不可读"
fi
if [ -w $file ]
then
    echo "文件可写"
else
    echo "文件不可写"
fi
if [ -x $file ]
then
    echo "文件可执行"
else
    echo "文件不可执行"
fi
if [ -f $file ]
then
    echo "文件为普通文件"
else
    echo "文件为特殊文件"
fi
if [ -d $file ]
then
    echo "文件是个目录"
else
    echo "文件不是个目录"
fi
if [ -s $file ]
then
    echo "文件不为空"
else
    echo "文件为空"
fi
if [ -e $file ]
then
    echo "文件存在"
```

```
else
    echo "文件不存在"
fi



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

3.3 Shell 脚本控制结构

3.3.1 if-else

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

3.3.2 while

let 命令 let 命令是 BASH 中用于计算的工具，用于执行一个或多个表达式，变量计算中不需要加上 $ 来表示变量。如果表达式中包含了空格或其他特殊字符，则必须引起来

```
#!/bin/sh
int=1
```

```
while(( $int<=5 ))
do
    echo $int
    let "int++"
done

# let 使用方法
let a=5+4
let b=9-3
let no--
let ++no
let no+=10
let no-=20
```

3.3.3 for

```
for loop in 1 2 3 4 5
do
    echo "The value is: $loop"
done
//////////////////////////Result is ....
The value is: 1
The value is: 2
The value is: 3
The value is: 4
The value is: 5

for str in 'This is a string'
do
    echo $str
done
//////////////////////////Result is ......
This is a string
```

3.3.4 until

```
until condition
do
    command
done
```

3.3.5 case

```
case 值 in
模式1)
    command1
    command2
    ...
    commandN
    ;;
模式2 )
    command1
    command2
    ...
    commandN
    ;;
esac

echo '输入␣1␣到␣4␣之间的数字:'
echo '你输入的数字为:'
read aNum

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

3.3.6 continue,break

3.4 Shell 函数

3.4.1 函数定义

shell 中函数的定义格式如下：

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

注意

1. 可以带 function fun() 定义，也可以直接 fun() 定义, 不带任何参数。
2. 参数返回，可以显示加：return 返回，如果不加，将以最后一条命令运行结果，作为返回值。return 后跟数值 n(0-255)
3. 函数返回值在调用该函数后通过 $? 来获得

```
#!/bin/bash
// Without Return
demoFun(){
    echo "这是我的第一个 shell 函数!"
}
echo "-----函数开始执行-----"
demoFun
echo "-----函数执行完毕-----"

---->Result:

-----函数开始执行-----
这是我的第一个 shell 函数!
-----函数执行完毕-----

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

```
----->Reuslt:

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

3.4.2 函数参数

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

```
#!/bin/bash

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

----------->Result:
第一个参数为 1 !
第二个参数为 2 !
第十个参数为 10 !
第十个参数为 34 !
第十一个参数为 73 !
参数总数有 11 个!
作为一个字符串输出所有参数 1 2 3 4 5 6 7 8 9 34 73 !
```

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

另外，还有几个特殊字符用来处理参数：

参数类型	说明
$#	传递到脚本的参数个数
$*	以一个单字符串显示所有向脚本传递的参数
$$	脚本运行的当前进程 ID 号
$!	后台运行的最后一个进程的 ID 号
$@	与$* 相同，但是使用时加引号，并在引号中返回每个参数。
$-	显示 Shell 使用的当前选项，与 set 命令功能相同
$?	显示最后命令的退出状态。0 表示没有错误，其他任何值表明有错误。

表 3.1: 参数说明

3.5 Shell 脚本调用已有脚本

和其他语言一样，**Shell 也可以包含外部脚本**。这样可以很方便的封装一些公用的代码作为一个独立的文件。

Shell 文件包含的语法格式如下：

```
. filename # 注意点号(.)和文件名中间有一空格
或
source filename
```

Example

```
//--->test1.sh
#!/bin/bash
url="http://www.runoob.com"

//--->test2.sh 引用test1.sh
#使用 . 号来引用test1.sh 文件
. ./test1.sh

# 或者使用以下包含文件代码
# source ./test1.sh

echo "菜鸟教程官网地址：$url"

--->Exec and Result:
$ chmod +x test2.sh
$ ./test2.sh
菜鸟教程官网地址：http://www.runoob.com
```

3.6 Shell 示例

How would you print just the 10th line of a file? From LeetCode

```
#!/bin/bash
count=0

while read line
do
    let ++count;
    if [ $count -eq 10 ] // if [[$count <= 10]] if((count <= 10))
        then echo $line
        break
    fi
done < file.txt
```

第四章　Linux 系统编程

4.1　线程

pthread_create　`pthread_create(pthread_t* id, pthread_attr_t* attr, void*(*start_rout`

- `void* (*start_routine) (void*)`: 这里是个陷阱，如果使用成员函数会存在一个参数不匹配问题，因为成员函数有一个隐含的第一参数 this, 而 this 为类指针类型，与 void* 不匹配，造成了 notMatch 错误。

4.2　进程

4.2.1　进程空间

http://www.cnblogs.com/xzzzh/p/6596982.html

http://www.cnblogs.com/smile267/archive/2012/10/21/2732099.html

4.2.2　uid、euid

4.2.3　fork

fork 之后子进程到底复制了父进程什么? http://blog.csdn.net/xy010902100449/article/details/44851453

这里就涉及到物理地址和逻辑地址（或称虚拟地址）的概念。

从逻辑地址到物理地址的映射称为地址重定向。分为：

静态重定向–在程序装入主存时已经完成了逻辑地址到物理地址和变换，在程序执行期间不会再发生改变。

动态重定向–程序执行期间完成，其实现依赖于硬件地址变换机构，如基址寄存器。

逻辑地址：CPU 所生成的地址。CPU 产生的逻辑地址被分为:p（页号）它包含每个页在物理内存中的基址，用来作为页表的索引；d（页偏移），同基址相结合，用来确定送入内存设备的物理内存地址。

物理地址：内存单元所看到的地址。用户程序看不见真正的物理地址。用户只生成逻辑地址，且认为进程的地址空间为 0 到 max。物理地址范围从 R+0 到 R+max，R 为基地址，地址映射 - 将程序地址空间中使用的逻辑地址变换成内存中的物理地址的过程。由内存管理单元（MMU）来完成。

fork () 会产生一个和父进程完全相同的子进程，但子进程在此后多会 exec 系统调用，出于效率考虑，linux 中引入了“写时复制“技术，也就是只有进程空间的各段的内容要发生变化时，才会将父进程的内容复制一份给子进程。在 fork 之后 exec 之前两个进程用的是相同的物理空间（内存区），子进程的代码段、数据段、堆栈都是指向父进程的物理空间，也就是说，**两者的虚拟空间不同，但其对应的物理空间是同一个。当父子进程中有更改相应段的行为发生时，再为子进程相应的段分配物理空间**，如果不是因为 exec，内核会给子进程的数据段、堆栈段分配相应的物理空间（至此两者有各自的进程空间，互不影响），**而代码段继续共享父进程的物理空间**（两者的代码完全相同）。而如果是因为 exec，由于两者执行的代码不同，子进程的代码段也会分配单独的物理空间。

fork 时子进程获得父进程数据空间、堆和栈的复制，所以变量的地址（当然是虚拟地址（相对位置））也是一样的。

每个进程都有自己的虚拟地址空间，不同进程的相同的虚拟地址显然可以对应不同的物理地址。因此地址相同（虚拟地址）而值不同没什么奇怪。具体过程是这样的：fork 子进程完全复制父进程的栈空间，也复制了页表，但没有复制物理页面，所以这时虚拟地址相同，物理地址也相同，但是会把父子共享的页面标记为“只读”（类似 mmap 的 private 的方式），如果父子进程一直对这个页面是同一个页面，知道其中任何一个进程要对共享的页面“写操作”，这时内核会复制一个物理页面给这个进程使用，同时修改页表。而把原来的只读页面标记为“可写”，留给另外一个进程使用。

这就是所谓的“写时复制”。正因为 fork 采用了这种写时复制的机制，所以 fork 出来子进程之后，父子进程哪个先调度呢？内核一般会先调度子进程，因为很多情况下子进程是要马上执行 exec，会清空栈、堆。。这些和父进程共享的空间，加载新的代码段。。。，这就避免了“写时复制”拷贝共享页面的机会。如果父进程先调度很可能写共享页面，会产生“写时复制”的无用功。所以，一般是子进程先调度滴。

假定父进程 malloc 的指针指向 0x12345678, fork 后,子进程中的指针也是指向 0x12345678，但是这两个地址都是虚拟内存地址（virtual memory），经过内存地址转换后所对应的物理地

址是不一样的。所以两个进程中的这两个地址相互之间没有任何关系。

（注 1：在理解时，你可以认为 fork 后，这两个相同的虚拟地址指向的是不同的物理地址，这样方便理解父子进程之间的独立性）（注 2：但实际上，Linux 为了提高 fork 的效率，采用了 copy-on-write 技术，fork 后，这两个虚拟地址实际上指向相同的物理地址（内存页），只有任何一个进程试图修改这个虚拟地址里的内容前，两个虚拟地址才会指向不同的物理地址（新的物理地址的内容从原物理地址中复制得到））

4.3 共享内存区

4.4 消息队列

4.5 信号量

4.6 库的使用

http://www.cnblogs.com/52php/p/5681711.html

4.6.1 介绍

使用 GNU 的工具我们如何在 Linux 下创建自己的程序函数库? 一个"程序函数库"简单的说就是一个文件包含了一些编译好的代码和数据，这些编译好的代码和数据可以在事后供其他的程序使用。程序函数库可以使整个程序更加模块化，更容易重新编译，而且更方便升级。

引用的那些**头文件中的函数**是怎么被执行的呢？这就要牵扯到**链接库**了！

库有两种，一种是 **静态链接库**，一种是 **动态链接库**，不管是哪一种库，*要使用它们，都要在程序中包含相应的 include 头文件*。编译过程如下：

`gcc -o helloWorld helloWorld.c` 生成一个 helloWorld 的执行文件，格式为 ELF(与 widows 的 exe 类似)。

4.6.2 静态函数库

是在程序执行前就加入到目标程序中去了；

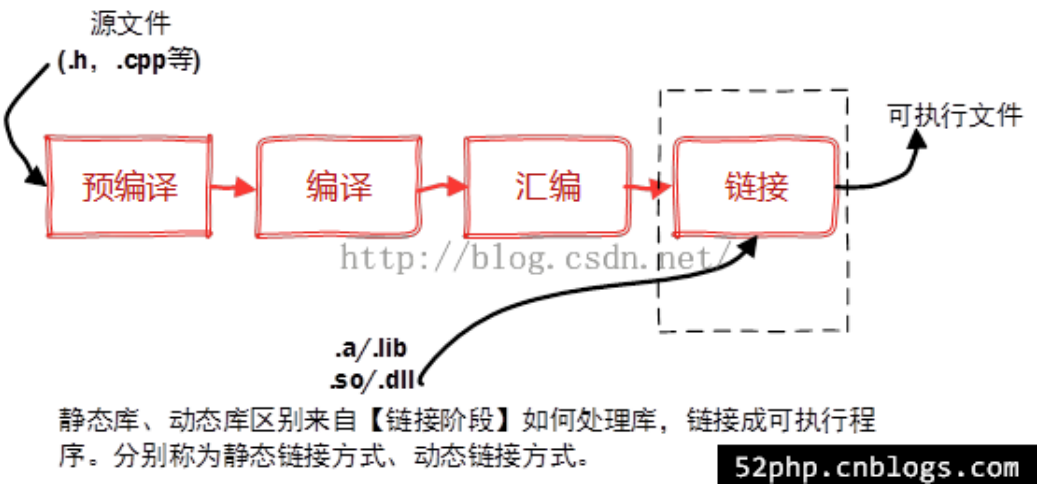


Fig 4.1: 编译过程

静态函数库实际上就是简单的一个普通的目标文件的集合，一般来说习惯用“.a”作为文件的**后缀**

静态库函数允许程序员把程序 link 起来而不用重新编译代码，节省了重新编译代码的时间。

如你想把自己提供的函数给别人使用，但是又想对函数的源代码进行保密，你就可以给别人提供一个静态函数库文件。理论上说，使用 ELF 格式的静态库函数生成的代码可以比使用共享函数库（或者动态函数库）的程序**运行速度上快一些，大概 1 - 5%，但是占用空间却大了很多**。

Example ->

定义一个加法函数，做成静态库，首先需要将声明放到头文件以便引用。

```
// add.h
#ifndef _ADD_
#define _ADD_
#include <iostream>

int add(int a, int b);
#endif
```

实现该函数

```
#include "add.h"

int add(int a, int b)
{
    return a+b;
}
```

生成静态库：

1. g++ -c add.cc 生成.o 文件
2. ar -crv libadd.a add.o 生成一个静态库

测试：

```
//目录结构
project
    |
    +---Main.cc
    |
    +---addlib
           |
           +----add.h
           |
           +----add.cc

#include <iostream>
#include "./addlib/add.h"

using namespace std;
int main()
{
    int num1 = 10;
    int num2 = 90;
    cout <<"the result is"<< add(num1,num2)<<endl;
    return 0;
}
```

不管是哪一种库，要使用它们，都要在程序中包含相应的 *include* 头文件，所以 include 使用相对路径找到头文件add.h

然后我们使用g++ -o Test Main.cc -L ./addlib/ -ladd 进行编译

- L 是指定加载库文件的路径
- l 是指定加载的库文件

静态库搜索路径 ->

1. 编译目标代码时指定的静态库搜索路径。-L ./src/ -lpthread

2. 环境变量LIBRARY_PATH 指定的动态库搜索路径。

```
export LIBRARY_PATH=/opt/lib:$LIBRARY_PATH
```

3. 配置文件/etc/ld.so.conf 中指定的动态库搜索路径，然后运行/sbin/ldconfig 刷新缓存

4. 默认的动态库搜索路径/lib，/usr/lib

4.6.3 动态函数库

静态链接的话，文件会很大，往往实现很小的一个功能就需要占用很大的空间，而且每次库文件升级的话，都要重新编译源文件，很不方便。如下：

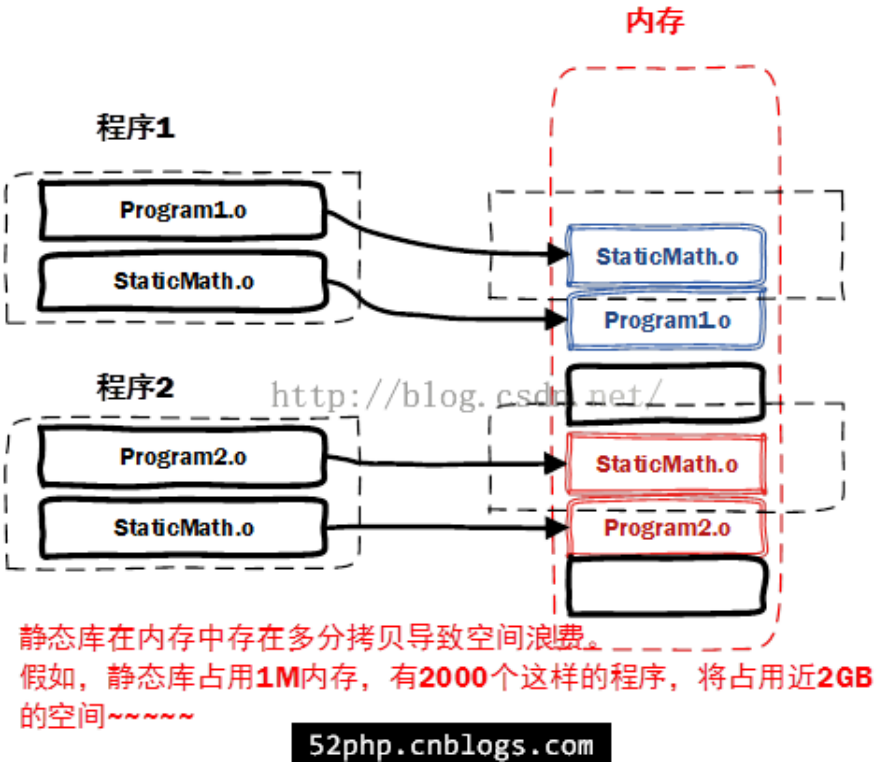


Fig 4.2: 静态库冗余空间

对于静态编译的程序 1 和程序 2，都应用库 staticMath。在内存中就又**两份相同的 static Math 目标文件，很浪费空间**，一旦程序数量过多就很可能会内存不足。

这么大的内存才只能运行这几个程序，实在不甘心。这样就又了动态库发挥威力的地方了。我们来看看动态链接的结果：

在这种模型中，两个程序只应用一个库，这个目标文件在内存中只有一份，供所有程序使用。并且在程序运行过程中动态调用库文件，**很方便，又不占空间**，但是动态链接有一个**缺点就是可移植性太差**，如果两台电脑运行环境不同，动态库存放的位置不一样，很可能导致程序运行失败。

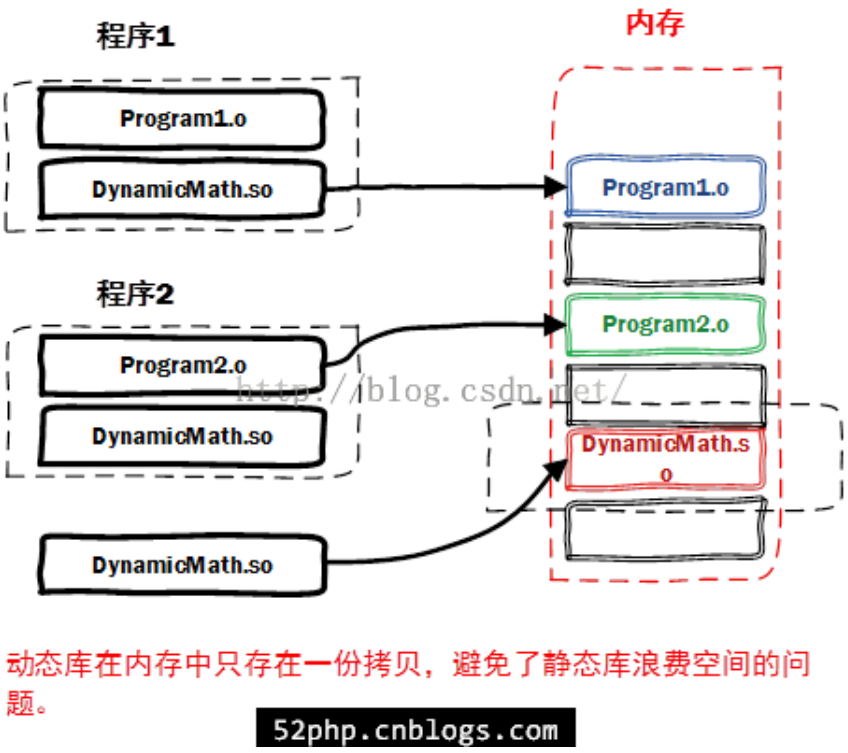


Fig 4.3: 动态库解决静态库冗余空间问题

Example 使用静态库代码结构与代码

生成一个libadd.so 的动态库 g++ -fPIC -shared -o libadd.so add.cpp。这样就生成一个了一个 libadd.so 的动态库。

使用如下命令进行编译：g++ -o Test Main.cc -L ./addlib/ -ladd -Wl,-rpath=./addlib/

但是如果不指定运行时搜索路径的话，会出现 cannot open shared obj 错误。如下编译：

g++ g++ -o Test Main.cc -L ./addlib/ -ladd

此时可以通过ldd ./Test 进行查看调用的动态库情况。具体的设置可参考动态库搜索路径的 4 种方法。

动态库搜索路径 ->

http://blog.csdn.net/weicao1990/article/details/51028335 https://www.cnblogs.com/cute/archive/2011/02/24/1963957.html

1. 编译目标代码时指定的动态库搜索路径。-L./src/ -ladd -Wl,-rpath=./src/

   前面两个-L -l 分别指明了动态库的位置和库的名字用于编译，只有引用，后面的-Wl,-rpath 则指明运行时到哪里运行，有内容。

2. 环境变量LD_LIBRARY_PATH 指定的动态库搜索路径。

```
export LD_LIBRARY_PATH=/opt/lib:$LD_LIBRARY_PATH
```

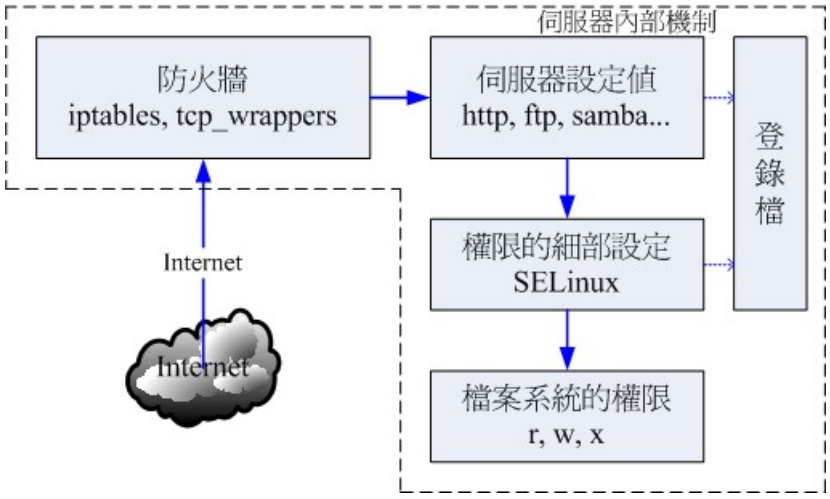
3. 配置文件/etc/ld.so.conf 中指定的动态库搜索路径, 然后运行/sbin/ldconfig 刷新缓存

4. 默认的动态库搜索路径/lib，/usr/lib, mv ./src/xx.lib /lib

第五章　Linux 服务器搭建

5.1　基本技能需求

首先，到底我们是如何连线到伺服器的？连线到伺服器又取得啥咚咚？我们先以底下这张图示来作个简单的说明好了：



看到没有，你连线到伺服器，重点在取得对方的资料，而一般资料的存在就是使用档案⊠！那你有没有权限取得？最终与该档案系统的设定有关啦！

上图显示的是：**首先**，用户端到伺服器的网路要能够通，等到用户端到达伺服器后，**会先由**伺服器的防火墙判断该连线能否放行，等到**放行之后**才能使用到伺服器软体的功能。而该功能又得要通过 SELinux 这个**细部权限**设定的项目后，才能够读取到档案系统。但能不能读到档案系统呢？这又跟档案系统的权限 (rwx) 有关啦！上述的每个部分都要能够成功，否则就无法顺利读取资料⊠。

所以，根据上面的流程我们大概可以将整个连线分为几个部分，包括：**网路、伺服器本身、内部防火墙软体设定、各项服务设定档、细部权限的 SELinux 以及最终最重要的档案权限**。

- **网络的基本概念**：以方便进行联网与配置及排错
- **熟悉操作系统的基本操作**：包括登录控制、账号管理、文本编辑器的使用等技巧

- **信息安全方面**：包括防火墙与软件更新方面的相关知识等
- 该服务器**协议所需要软件**的基本安装、配置、排错等

5.1.1 网路：了解网路基础知识与所需服务之通讯协定

- 基本的网路基础知识：包括乙太网路硬体与协定、TCP/IP、网路连线所需参数等；
- 各网路服务所对应的通讯协定原理，以及各通讯协定所需对应的软体。

5.1.2 伺服器本身：了解架网路伺服器之目的以配合主机的安装规划

想要架设伺服器吗？那... 架什么伺服器？这个伺服器要不要对 Internet 开放？这个服务要不要针对客户提供相关帐号？要不要针对不同的客户帐号进行例如**磁碟容量**、**可活动空间**与可用系统资源**进行限制**？如果要进行各项资源的限制，那伺服器作业系统应该要如何安装与设定？问题很多吧！所以，先了解你要的伺服器服务目的之后，后续的规划才能陆续出炉。不过，如果架站只是为了『 练功 』而已，呵呵！那就不需要考虑太多了

5.1.3 伺服器本身：了解作业系统的基本操作

网路服务软体是需要建置在作业系统上面的，所以基本的作业系统操作就得要了解才行啊！包括软体如何安装与移除？如何让系统进行例行的工作管理？如何依据伺服器服务之目的规划档案系统？如何让档案系统具有未来扩充性 (LVM 之类)？系统如何管理各项服务之启动？系统的开机流程为何？系统出错时，该如何进行快速复原等等，这都需要了解的呢！

5.1.4 内部防火墙设定：管理系统的可分享资源

一部主机可以拥有多种伺服器软体的运作，而很多 Linux distributions 出厂的预设值就已经开放很多服务给 Internet 使用了，不过这些服务可能并不是你想要开放的呢。我们在了解网路基础与所需服务的目的之后，接下来就是透过防火墙来规范可以使用本伺服器服务的用户，以让系统在使用上拥有较佳的控管情况。此外，**不管你的防火墙系统设定的再怎么严格，只要是你要开放的服务，那防火墙对于该服务就没有保护的效果**。因此，那个重要的线上更新软体机制就一定要定期进行！否则你的系统将会非常非常的不安全！

5.1.5 伺服器软体设定：学习设定技巧与开机是否自动执行

刚刚第一点就提到我们得要知道每种服务所能达成的功能，如此一来才能够架设你所需要的服务的网站。那你所需要的服务是由哪个软体达成的？同一个服务可否有不同的软体？每种软体可以达成的目的是否相同？依据所需要的功能如何设定你的伺服器软体？架设过程中如果出现错误，你该如何观察与除错？可否定期的分析伺服器相关的登录资讯，以方便了解该伺服器的使用情况与错误发生的原因？能否通知多个用户进行连线测试，以取得较佳的伺服器设定值？所以这里你可能就得要知道：

- 软体如何安装、如何查询相关设定档所在位置；
- 伺服器软体如何设定？
- 伺服器软体如何启动？如何设定自动开机启动？如何观察启动的埠口？
- 伺服器软体启动失败如何除错？如何观察登录档？如何透过登录档进行除错？
- 透过用户端进行连线测试，如果失败该如何处理？连线失败的原因是伺服器还是防火墙？
- 伺服器的设定修改是否有建立日志？登录档是否有定期分析？
- 伺服器所提供或分享的资料有无定期备份？如何定期自动备份或异地备份？

5.1.6 细部权限设定：包括 SELinux 与档案权限