Linux 系统编程-基础篇

第一章 Linux初步认识

1.1 Linux发展史

Linux是一套免费使用和自由传播的类Unix操作系统，那什么是Unix以及类Unix操作系统。

UNIX操作系统（尤尼斯），是一个强大的多用户、多任务操作系统，支持多种处理器架构，按照操作系统的分类，属于分时操作系统，最早由KenThompson、Dennis Ritchie和Douglas McIlroy于1969年在AT&T的贝尔实验室开发。

引起了学术界的广泛兴趣并对其源码索取，Unix第五版就以“仅用于教育目的”的协议，提供给各大学作为教学之用，成为当时操作系统课程中的范例教材。

后来AT&T意识到了Unix的商业价值，不再将Unix源码授权给学术机构使用，AT&T妄图私有化的Unix，为了私有化Unix，在1986年，IEEE指定了一个委员会制定了一个开放作业系统的标准，称为 POSIX （Portable Operating Systems Interface）。

AT&T的这种商业态度，让当时许多的Unix的爱好者和软件开发者们感到相当的痛心和忧虑，他们认为商业化的种种限制并不利于产品的发展，相反还能导制产品出现诸多的问题。

后来商业化Unix的版本确实出现了诸多问题，引起了大众的不满和反对。于是，大家开始有组织地结成“反叛联盟”以此对抗AT&T的商业化行为。

另一方面，关于“大教堂”（集权、封闭、受控、保密）和“集市”（分权、公开、精细的同僚复审）两种开发模式成为了新思潮的中心思想。这个新思潮对IT业产生了非常深远影响。为整个计算机世界带来了革命性的价值观。

此时，一个名叫Richard Stallman（理查德·斯托曼）的领袖出现了，他认为Unix是一个相当好的操作系统，如果大家都能够将自己所学贡献出来，那么这个系统将会更加的优异！他倡导的Open Source的概念，就是反对Unix实验室里的产品商业化私有化。

尽管Stallman既不是、也从来没有成为一个Unix程序员，但在后1980的大环境下，实现一个仿Unix操作系统成了他追求的明确战略目标。Richard Stallman早期的捐助者大都是新踏入Unix土地的ARPANET黑客，他们对代码共享的使命感甚至比那些有更多Unix背景的人强烈。

为了这个理想，Richard Stallman于1984年创业了GNU，又称革奴计划，计划开发一套与Unix相互兼容的软件。1985 年 Richard Stallman 又创立了自由软件基金会（Free Software Foundation）来为 GNU 计划提供技术、法律以及财政支持。尽管 GNU 计划大部分时候是由个人自愿无偿贡献，但 FSF 有时还是会聘请程序员帮助编写。当 GNU 计划开始逐渐获得成功时，一些商业公司开始介入开发和技术支持。当中最著名的就是之后被 Red Hat 兼并的 Cygnus Solutions。

自90年代发起这个计划以来，GNU 开始大量的产生或收集各种系统所必备的组件，像是——函数库、编译器gcc、调试工具gdb、文本编辑器vi、网站服务器，以及一个Unix的使用者接口（Unix shell）等等。但由于种种原因，GNU一直没有开发出一款开源的操作系统kernel。正当Richard Stallman在为操作系统内核伤脑筋的时候，Linux出现了。

1990年，Linus Torvalds（林纳斯·托瓦兹）还是芬兰赫尔辛基大学的一名学生，最初是用汇编语言写了一个在保护模式下处理多任务切换的程序，后来从Minix（Andy Tanenbaum教授所写的很小 的Unix操作系统，主要用于操作系统教学）得到灵感，进一步产生了自认为狂妄的想法——写一个比Minix更好的Linux，于是开始写了一些硬件的设备驱动程序，一个小的文件系统。这样0.0.1版本的Linux就出来了，但是它只具有操作系统内核的勉强的雏形，甚至不能运行，你必须在有Minix的机器上编译以后才能玩。这时候Linus已经完全着迷而不想停止，决定踢开Minix，于是在1991年10 月5号发布Linux 0.0.2版本，在这个版本中已经可以运行bash 和gcc。

从一开始，Linus就决定自由扩散Linux，包括原代码，随即Linux引起黑客们（hacker）的注意，通过计算机网络加入了Linux的内核开发。Linux倾向于成为一个黑客的系统——直到今天，在Linux社区里内核的开发被认为是真正的编程。由于一批高水平黑客的加入，使Linux 发展迅猛，几乎一两个礼拜就有新版或修正版的出现，到1993年底94年初，Linux 1.0终于诞生了！Linux 1.0已经是一个功能完备的操作系统，而且内核写得紧凑高效，可以充分发挥硬件的性能，在4M内存的80386机器上也表现得非常好，至今人们还在津津乐道。

Linux 的标志和吉祥物是一只名字叫做 Tux 的 企鹅，标志的由来是因为Linus在澳洲时曾被一只动物园里的企鹅咬了一口，便选择了企鹅作为Linux的标志。

Tux：Linux吉祥物/Logo

Linux 的历史是和GNU紧密联系在一起的。从1983年开始的GNU计划致力于开发一个自由并且完整的类Unix操作系统，包括软件开发工具和各种应用程序。到1991年 Linux 内核发布的时候，GNU已经几乎完成了除了系统内核之外的各种必备软件的开发。在 Linus Torvalds 和其它开发人员的努力下，GNU组件可以运行于Linux内核之上。整个内核是基于 GNU 通用公共许可，也就是GPL（GNU General Public License，GNU通用公共许可证）。

1.2 Linux发行版本

Linux存在着许多不同的Linux发行版本，比如Ubuntu、CentOS、Redhat、Debian、Fedora、Android等等。但它们都使用了Linux内核。Linux可安装在各种计算机硬件设备中，比如手机、平板电脑、路由器、视频游戏控制台、台式计算机、大型机和超级计算机。

Linux是一个基于POSIX (Portable Operating System Interface ，缩写为 POSIX ）的多用户、多任务、支持多线程和多CPU的操作系统。它能运行主要的UNIX工具软件、应用程序和网络协议。它支持32位和64位硬件。Linux继承了Unix以网络为核心的设计思想，是一个性能稳定的多用户网络操作系统。

严格来讲，Linux这个词本身只表示Linux内核，但实际上人们已经习惯了用Linux来形容整个基于Linux内核，并且使用GNU 工程各种工具和数据库的操作系统。

- CentOS系统

  很多网站站长一般都选择centOS系统，CentOS是从redhat源代码编译的重新发布版。CentOS去除很多与服务器功能无关的应用，系统简单但非常稳定，命令行操作可以方便管理系统和应用，并且有帮助文档和社区的支持。

- Ubuntu系统

  Ubuntu有亮丽的用户界面，完善的包管理系统，强大的软件源支持，丰富的技术社区，并且Ubuntu对计算机硬件的支持好于centos和debian，兼容性强，Ubuntu应用非常多，但对于服务器系统来说，需要的是稳定，操作方便，维护简单的系统。如果你需要在服务器端使用图形界面, ubuntu是一个不错的选择，你需要注意的是，图形界面占用的内存非常大，而内存越大的vps(virtual private server)价格也越高。

- Debian系统

  Debian也非常适合做服务器操作系统，与Ubuntu比较，它没有太多的花哨界面，稳定压倒一切。debian这个Linux系统，底层非常稳定，内核和内存的占用都非常小，比如128M的内存就能运行，比Centos占用的资源还要少，但debian的帮助文档和技术资料比较少。

第二章 shell

2.1 什么是shell

在计算机科学中，Shell俗称壳， 是指命令解析器。它类似于DOS下的command和后来的cmd.exe。它接收用户命令，然后调用相应的应用程序。

同时它又是一种程序设计语言。作为命令语言，它交互式解释和执行用户输入的命令或者自动地解释和执行预先设定好的一连串的命令；作为程序设计语言，它定义了各种变量和参数，并提供了许多在高级语言中才具有的控制结构，包括循环和分支。

- 参看系统有哪些shell：

```
cat /etc/shells
```

- 输出如图：

```
# /etc/shells: valid login shells
/bin/sh
/bin/dash
/bin/bash
/bin/rbash
```

- 查看当前正在使用的shell：

```
echo $SHELL
```

- 输出如图：

```
/bin/bash
```

Bash (GNU Bourne-Again Shell) 是许多Linux发行版的默认Shell。事实上，还有许多传统UNIX上用的Shell，例如tcsh、csh、ash、bsh、ksh等等，Shell Script大致都类同，当您学会一种Shell以后，其它的Shell会很快就上手，大多数的时候，一个Shell Script通常可以在很多种Shell上使用。

2.2 命令行补全功能

在使用bash命令行时，在提示符下，输入某个命令的前面几个字符，然后按TAB键，就会列出以这几个字符开头的命令供我们选择。不光如此，还可以进行参数补全，但只限于文件参数，当输入到参数部分时，按TAB键，就会列出以这个参数开头的文件路径供我们选择。

- 输入apt按TAB键

```
apt
```

- 输出

```
apt                 aptd                apt-key
apt-add-repository  aptdcon             apt-mark
apt-cache           apt-extracttemplates apt-mirror
apt-cdrom           apt-ftparchive      apt-sortpkgs
apt-config          apt-get
```

2.3 bash常用快捷键

- 在使用bash时候，经常会用到这些快捷方式，有助于提高输入指令的速度 。

功能	快捷键
查看上一条指令	ctrl+p (p代表previous) / up键
查看下一条指令	ctrl+n (n代表next) / down键
光标左移	ctrl+b (b代表backward) / Left键
光标右移	ctrl+f (f代表forward) / right键
光标移动到开头	ctrl+a / Home键
光标移动到末尾	ctrl+e (e代表end) / End键
清屏	ctrl+l
删除光标所在字符	ctrl+d
删除光标前面的字符	ctrl+h
删除光标后所有字符	ctrl+k
删除光标前所有字符	ctrl+u
重复执行当前指令	ctrl+o(需要先翻出一条已经执行过的指令)
中断当前指令	ctrl+c

第三章 分区与目录

3.1 分区

1. Linux硬盘的分区主要分为主分区（primary partion）和扩展分区(extension partion)两种，主分区和扩展分区的数目之和不能大于四个。
2. 主分区可以马上被使用但不能再分区。
3. 扩展分区必须再进行二次分区后才能使用，扩展分区进行二次分区后， 叫做逻辑分区（logical partion）， 逻

辑分区没有数量上限制。

3.2 磁盘文件

Linux中一个硬盘就是一个文件，存放/dev目录下面，IDE硬盘命名为hdx（x为从a—d， 一般电脑的IDE硬盘最多四个）。 SCSI，SATA，USB硬盘等，在/dev/目录下被命名为sdx（x为a—z）

比如SCSI硬盘，主分区从sda1开始到sda4，逻辑分区从sda5开始，（逻辑分区永远从sda5开始…）

- 设备名可以使用fdisk –l查看:

```
sudo fdisk -l
```

- 输出:

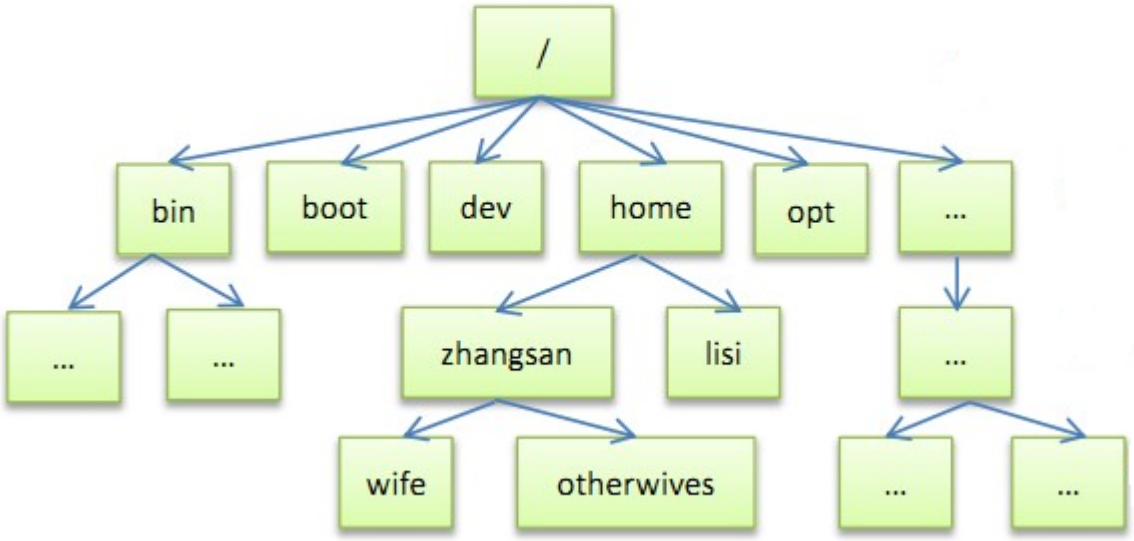
```
Disk /dev/sda: 42.9 GB, 42949672960 bytes
255 heads, 63 sectors/track, 5221 cylinders, total 83886080 sectors
Units = 扇区 of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x000e626e

   设备 启动      起点          终点      块数   Id  系统
/dev/sda1   *        2048    81788927    40893440   83  Linux
/dev/sda2        81790974    83884031     1046529    5  扩展
/dev/sda5        81790976    83884031     1046528   82  Linux 交换 /
```

可以看到这个硬盘文件/dev/sda大小为42.9G, 并且它被分成一个主分区/dev/sda1, 一个扩展分区/dev/sda2, 扩展分区又分成一个逻辑分区 /dev/sda5。

3.3 目录结构

Linux中采用的是树形结构目录。




```
/               #根目录
/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， cd等等。这个目录中的文件都是可执行的，一般的用户都可以使用。
/dev    #设备文件，比如声卡、磁盘、鼠标、键盘等。
/etc    #系统管理和配置文件
/etc/init.d     #启动配置文件和脚本,可在这里添加启动脚本。
/etc/rc.local   #用户添加启动项

/home   #用户主目录，比如用户user的主目录就是/home/user，可以用~user表示
/lib    #标准程序设计库存放路径，又叫动态链接共享库，作用类似windows里的.dll文件
/sbin   #超级管理命令，这里存放的是系统管理员使用的管理程序。
/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。 /tmp就用来存放临时文件的。
/root   #系统管理员的主目录
/mnt    #用来临时挂载其他的文件系统
/lost+found     #这个目录平时是空的，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。当系统启动的过程中fsck工具会检查这里，并修复已经损坏的文件系统。
/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
/proc   #虚拟文件目录，可直接访问这个目录来获取系统信息。
/var    #所有服务的登录文件或错误讯息档案（log files）都在 /var/log 里面
/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。在一般情况下，GRUB或LILO系统引导管理器也位于这个目录；

/usr    #最庞大的目录，要用到的应用程序和文件几乎都在这个目录，其中包含：
        /usr/bin                #众多的应用程序
        /usr/sbin               #超级用户的一些管理程序
        /usr/include    #Linux下开发和编译应用程序所需要的头文件
        /usr/lib                #常用的动态链接库
        /usr/share/man  #帮助文档
        /usr/src                #源代码，Linux内核的源代码就放在/usr/src/Linux里
        /usr/local/bin  #本地增加的命令
        /usr/local/lib  #本地增加的库根文件系统
```

- `.` 代表当前路径。

- `..`代表上一级目录。
- `~`代表用户目录路径。

3.4 常用命令详解

3.4.1 ls 命令

ls是英文单词list的缩写.用来查看文件目录的属性。

- 例如直接输入ls按回车，查看根目录的文件以及目录。

```
ls /
```

- 输入如图：

```
bin    dev   initrd.img  media  proc  sbin  tmp  vmlinuz
boot   etc   lib         mnt    root  srv   usr  workspace
cdrom  home  lost+found  opt    run   sys   var
```

- `ls -l` -l 参数代表以列表的方式显示。

```
where@ubuntu:~$ ls -l

总用量 31224
-rw------- 1 where where 31916032  7月  7 18:39 core
drwxr-xr-x 2 where where     4096  7月  8 12:05 Desktop
drwxr-xr-x 2 where where     4096  6月 25 16:53 Documents
drwxr-xr-x 2 where where     4096  6月 25 16:53 Downloads
-rw-r--r-- 1 where where     8980  7月  6 15:13 examples.desktop
-rw------- 1 where where       12  7月  8 18:08 helloworld.txt
drwxr-xr-x 2 where where     4096  6月 25 16:53 Music
drwxrwxr-x 2 where where     4096  7月 30 18:28 mydir
drwxrwxr-x 2 where where     4096  7月  8 18:18 nfs
drwxr-xr-x 2 where where     4096  6月 25 16:53 Pictures
drwxr-xr-x 2 where where     4096  6月 25 16:53 Public
drwxr-xr-x 2 where where     4096  6月 25 16:53 Templates
drwxr-xr-x 2 where where     4096  6月 25 16:53 Videos
```

```
drwxr-xr-x 2 where where     4096  6月 25 16:53 Videos
#第一个字符d 代表这是一个目录文件。
  # -  代表普通文件
  # c  字符设备文件
  # b  块设备文件
  # p  管道文件
  # l  链接文件
  # s  socket文件
#后面的rwxr-xr-x字符，代表user、group、other对文件所拥有的权限，rwx代表该用户拥有读写执行的权限。r-x
代表同一组的用户拥有的读和执行权限，后一个r-x代表其他用户拥有读和执行权限。
# 2代表文件硬链接的计数，表示该文件有两个硬链接。
# where 文件所属的用户名。
# where  文件所属的用户组。
# 4096 文件大小，单位字节。
# 6月 25 16:53  文件最后被修改的日期。
# Videos 文件名
```

- `ls -a` -a参数代表all的意思，表示把所有的文件都罗列出来，包括隐藏文件，点号开头的在Linux中都表示隐藏文件。

```
where@ubuntu:~$ ls -a
.              Desktop           .local            .thunderbird
..             .dmrc             .mozilla          Videos
.bash_history  Documents         Music             .viminfo
.bash_logout   Downloads         mydir             .vimrc
.bashrc        .emacs.d          nfs               .Xauthority
.cache         examples.desktop  .pam_environment  .xinputrc
.compiz        .gconf            Pictures          .xsession-errors
.config        .gvfs             .profile          .xsession-errors.old
core           helloworld.txt    Public
.dbus          .ICEauthority     Templates
```

- `ls -ld [filename]`代表只列出目录文件的属性。

3.4.2 stat命令

查看文件的访问时间，修改时间等。

```
where@ubuntu:~$ stat hello
  文件："hello"
  大小：7341        块：16         IO 块：4096   普通文件
设备：801h/2049d   Inode：1878587     硬链接：1
权限：(0775/-rwxrwxr-x)  Uid：( 1000/   where)   Gid：( 1000/   where)
最近访问：2016-08-12 22:38:25.272181218 +0800
最近更改：2016-08-12 22:38:25.272181218 +0800
最近改动：2016-08-12 22:38:25.272181218 +0800
创建时间：-
```

访问时间，是指通过指令如cat、vi等来查看的文件的最近一次时间。

更改时间，是指修改文件内容的最近一次时间。

改动时间，是指修改文件属性的最近一次时间。

注意：访问时间是内容更改后，第一次访问的时间，后面再次访问的时候访问时间不会改变。

3.4.3 cd命令

cd 是 change directory的缩写，表示改变当前所在路径。

命令	用途
cd -	回上一次所在的目录
cd ~或 cd	回当前用户的主目录
cd /	回到根目录
cd ..	回当前目录的上一级目录

3.4.4 pwd命令

pwd 是英文print working directory 显示当前所在路径。

```
where@ubuntu:~$ pwd
/home/where
```

3.4.5 which命令

寻找可执行文件 ， 并在PATH环境变量里面寻找。

```
where@ubuntu:~$ which ls
/bin/ls
where@ubuntu:~$ which reboot
/sbin/reboot
where@ubuntu:~$
```

3.4.6 touch命令

```
touch [OPTION]  [FILE]
```

- 将每个文件的访问及修改时间都更新为目前的时间。
- 如果文件不存在，则创建一个字节数为0的文件。

```
-a          #只更新访问时间，不改变修改时间
-c          #不创建不存在的文件
-m          #只更新修改时间，不改变访问时间
-r file     #使用文件file的时间更新文件的时间
-t          #将时间修改为参数指定的日期,如：07081556代表7月8号15点56分
```

例如：

```
touch -a file1  #更新file1文件的访问时间
touch -r file1 file2 #使用file1的时间来更新文件file2
```

3.4.7 mkdir 命令

mkdir 是make directory的英文缩写。

```
mkdir [OPTION] [DIRECTORY]
```

创建目录DIRECTORY，可以一次创建多个。OPTION如果是-p，表示可以连同父目录一起创建。

```
where@ubuntu:~$ mkdir -p dir/dir/dir/dir/dir
where@ubuntu:~$
```

3.4.8 rmdir 命令

rmdir是remove directory的英文缩写。

```
rmdir [OPTION]  [DIRECTORY]
```

删除空目录，可以一次删除多个。OPTION如果是-p，表示可以连同空的父目录一起删除。但是一旦父目录中还包含其他文件，则删除失败。

3.4.9 rm 命令

rm是remove的英文缩写。

```
rm [options] [file]
```

可以用来删除普通文件，也可以用来删除目录，特别用来删除目录中嵌套有子目录的目录文件。

常用参数：

```
-f  --force  #强制删除，不询问是否要删除。
-r  --recursive  #递归删除，包括文件夹中的内容。
```

3.4.10 mv命令

mv是英文单词move的缩写。可以用来移动文件夹或者文件，也可以用来更改文件名。

```
mv [srcfile] [destfile]
```

```
mv file /       #把文件file移动到根目录中。
```

```
mv file file_bak  #把文件file重命名为file_bak。
```

3.4.11 cp命令

cp是英文单词copy的缩写，表示拷贝文件。

```
cp [srcfile] [destfile]
```

- 可以用来拷贝普通文件

```
cp file file_bak     #拷贝一份file为file_bak
```

- 可以用来拷贝目录

```
cp dir dir_bak -r   #拷贝一个目录dir为dir_bak, -r参数代表递归拷贝，把dir目录中的文件也拷贝过去
```

3.4.12 cat命令

cat英文单词concatenate连锁的缩写，用来查看文件内容，以及将几个文件连成一个文件，

- 不填文件参数，默认的情况下是从标准输入中获取内容：

```
where@ubuntu:~$ cat  #cat从标准输入中获取内容，并且打印一次。
helloworld
helloworld
```

- 查看文件mydir/test.cpp文件

```
where@ubuntu:~$ cat mydir/test.cpp
#include <stdio.h>

int main()
{
    printf("%d\n", 3.0 );
    return 0;
}
```

- 将文件file1 file2连成file3文件

```
cat file1 file2 > file3
```

3.4.13 more命令

more 是我们最常用的工具之一，最常用的就是显示输出的内容，然后根据窗口的大小进行分页显示，并且提示文件的百分比。

参数如下：

```
+num    #从第num行开始显示；
-num    #定义每屏显示num行；
```

例如：

```
more  +10 -5 file #从第10行开始，每一页5行
```

打开之后的动作：

快捷	功能
f（或空格键）	向下显示一屏
b	返回上一屏
n Enter	键可以向下滚动显示n行，默认为1行
q	退出

3.4.14 less命令

less工具也是对文件或其它输出进行分页显示的工具

参数如下:

```
-f    #强制打开文件，二进制文件显示时，不提示警告；

-N    #在每行前输出行号；
```

打开之后的动作：

快捷	动作
空格键	向下滚动一屏
j或方向键↓	向下滚动一行
k或方向键↑	向上滚动一行
n Enter	向下移动n行, 不填默认一行
b	向上滚动一屏
f	向下滚动一屏
d	向下滚动半屏
u	向上滚动半屏
q	退出
g	跳到第一行
G	跳到最后一行

3.4.15 locate 命令

全盘寻找文件，文件名部分匹配，只要有包含该字符串的都罗列出来，这个指令查找速度很快，它需要一个数据库，这个数据库由每天的例行工作（crontab）程序来更新。当我们建立好这个数据库后，就可以方便地来搜寻所需文件了。

```
where@ubuntu:~$ locate sources.list
/etc/apt/sources.list
/etc/apt/sources.list.d
/etc/apt/sources.list~
/usr/share/doc/apt/examples/sources.list
/usr/share/man/de/man5/sources.list.5.gz
/usr/share/man/es/man5/sources.list.5.gz
/usr/share/man/fr/man5/sources.list.5.gz
/usr/share/man/it/man5/sources.list.5.gz
/usr/share/man/ja/man5/sources.list.5.gz
/usr/share/man/man5/sources.list.5.gz
/usr/share/man/pl/man5/sources.list.5.gz
/usr/share/man/pt/man5/sources.list.5.gz
/var/lib/dpkg/info/python-pkg-resources.list
/var/lib/dpkg/info/python3-pkg-resources.list
```

- 马上创建的文件没办法使用locate查找到:

```
where@ubuntu:~$ touch newfile
where@ubuntu:~$ locate newfile
where@ubuntu:~$
```

- 如果想马上更新可以使用一下指令：

```
updatedb
```

3.4.16 find 命令

直接在全文件系统上搜寻，功能强大，速度慢。

```
find   [path]   [-option]   [   -print  -exec   -ok   command ]   {} \;
```

```
 path:   #要执行查找的目录。
-option: #查找的具体方法。
-print:  #find命令将匹配的文件输出到标准输出。
-exec:   #find命令对匹配的文件执行该参数所给出的shell命令。相应命令的形式为'command' {} \;，注意{}和\；之间的空格。
-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。
```

```
find / -name filename   #在根目录里面搜索文件名为filename的文件
find /etc -name *s*     #在目录里面搜索带有s的文件
find /etc -name *S      #在目录里面搜索以s结尾的文件
find /etc -name s*      #在目录里面搜索以s开头的文件

find / -amin -10        #在系统中搜索最后 1 0 分钟访问的文件
find / -atime -2        #查找在系统中最后 4 8 小时访问的文件
find / -mmin -5         #查找在系统中最后 5 分钟修改过的文件
find / -mtime -1        #查找在系统中最后 2 4 小时修改过的文件
find / -ctime -1        #查找在系统中最后 2 4 小时被改变状态的文件

find / -user username   #查找在系统中属于用户username的文件
find / -group groupname #groupname 查找在系统中属于groupname的文件


find / -empty           #查找在系统中为空的文件或者是文件夹
find / -inum 3          #查找inode号为3的文件
find / -type d          #查找为文件类型为文件夹的文件d为文件夹
                                f     #普通文件
                                d     #目录文件
                                l     #链接文件
                                b     #块设备文件
                                c     #字符设备文件
                                p     #管道文件
                                s     #socket文件
```

- 查找当前目录中所有c源文件的文件属性：

```
find ./ -name "*.c" -exec ls -l {} \;
```

3.4.17 grep 命令

搜索内容中是否包含指定的字符串，并打印出该行。

- 默认情况下，从标准输入中获取内容，例如

```
where@ubuntu:~$ grep helloworld
hasdlfj
helloworldjsldjf
helloworldjsldjf
```

标准输入中输入了包含helloworld的一行字符后，grep会重复打印一次，退出该条命令，使用ctrl+c.

- 查找当前路径中所有包含stdio.h的文件。

```
where@ubuntu:~/mydir$ grep stdio.h *
file:#include <stdio.h>
test1.c:#include <stdio.h>
test.cpp:#include <stdio.h>
```

- 常用参数有：

```
    -i    --ignore-case   #忽略字符大小写的差别。
    -v                    #输出没有指定字符串的文件
    -c                    #只输出匹配行的计数。
    -R                    #连同子目录中所有文件一起查找。
```

- 查找当前目录c源文件中，包含头文件stdio.h的所有文件：

```
find ./ -name "*.c" |xargs grep stdio.h
```

3.4.18 ln命令

ln是英文单词link的缩写，用来创建链接的命令。

Linux链接分两种，一种被称为硬链接（Hard Link），另一种被称为符号链接（Symbolic Link）。默认情况下，ln命令产生硬链接。

【硬链接】

硬链接指通过索引节点来进行链接。在Linux的文件系统中，保存在磁盘分区中的文件不管是什么类型都给它分配一个编号，称为索引节点号(Inode Index)。在Linux中，多个文件名指向同一索引节点，一般这种链接就是硬链接。

硬链接的作用是允许一个文件拥有多个有效路径名，这样用户就可以建立硬链接到重要文件，以防止“误删”。

如果有多个硬链接，只删除一个链接并不影响本身和其它的链接，只有当最后一个链接被删除后，文件的才会被正在删除。也就是说，文件真正删除的条件是与之相关的所有硬链接文件均被删除。

【软链接】

另外一种链接称之为符号链接（Symbolic Link），也叫软链接。软链接文件有类似于Windows的快捷方式。它实际上是一个特殊的文件。符号链接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。

- 给flie文件创建一个硬链接

```
touch file
ln file file_hard
```

注意：硬链接不能给目录文件创建链接。

- 给file文件创建一个软链接

```
touch file
ln -s file flie_soft
```

注意：软链接的时候尽量使用绝对路径，避免由于链接文件移动后，造成文件失效。

3.4.19 wc 命令

Linux系统中的wc为英文Word Count的缩写，命令的功能为统计指定文件中的字节数、字数、行数，并将统计结果显示输出。

- 命令格式：

```
wc [options]  [filename...]
```

- 命令功能：

  统计指定文件中的字节数、字数、行数，并将统计结果显示输出。如果没有给出文件名，则从标准输入读取。wc同时也给出所指定文件的总统计数。

- 命令参数：

```
-c  #统计字节数。
-l  #统计行数。
-m  #统计字符数。这个标志不能与 -c 标志一起使用。
-w  #统计字数。一个字被定义为由空白、跳格或换行字符分隔的字符串。
-L  #打印最长行的长度。
```

- 默认情况下统计文件的行数、字数、字节数

```
where@ubuntu:~/mydir$ wc test.cpp
 7 11 69 test.cpp
```

上图中的7表示行数，11表示单词数，69代表字节数。

- 可以同时统计多个文件

```
where@ubuntu:~$ wc  newfile  myfile
0 0 0 newfile
0 0 0 myfile
0 0 0 总用量
```

- 统计所有c源文件的行数:

```
find ./ -name "*.c" -exec wc -l {} \;
```

3.4.20 od命令

od是英文octal dump的缩写，功能是把文件用8进制或者其他的格式显示出来，通常用于查看特殊格式文件的内容，可以用来查看不可见字符。

```
od [option] [filename]
```

```
option:

-t      #指定数据的显示格式，主要的参数有：

c       #将ASCII字符显示出来

d[SIZE] #有符号十进制数,以SIZE字节对齐。

f[SIZE] #浮点数,以SIZE字节对齐。

o[SIZE] #八进制（系统默认值为02）,以SIZE字节对齐。

u[SIZE] #无符号十进制数,以SIZE字节对齐。

x[SIZE] #十六进制数,以SIZE字节对齐。
```

例如：

```
where@ubuntu:~/mydir$ od -tcx2 test.cpp
0000000    6923    636e    756c    6564    3c20    7473    6964    2e6f
          #   i   n   c   l   u   d   e       <   s   t   d   i   o   .
0000020    3e68    0a0a    6e69    2074    616d    6e69    2928    7b0a
          h   >  \n  \n   i   n   t       m   a   i   n   (   )  \n   {
0000040    090a    7270    6e69    6674    2228    6425    6e5c    2c22
         \n  \t   p   r   i   n   t   f   (   "   %   d   \   n   "   ,
0000060    3320    302e    2920    0a3b    7209    7465    7275    206e
            3   .   0       )   ;  \n  \t   r   e   t   u   r   n
0000100    3b30    7d0a    000a
          0   ;  \n   }  \n
0000105
```

3.4.21 du命令

du是英文Disk usage的缩写，表示计算某个目录在硬盘中所占的空间大小，默认情况下以kb为单位。通过递归统计每一个目录中所占用的空间大小。

常用参数如下:

参数	功能
-h	人性化显示方式，自动决定显示单位。
-k	使用Kb单位来显示
-m	使用Mb单位来显示

3.4.22 df命令

df是英文Disk free的缩写，用来统计磁盘是使用情况。

参数	功能
-a	全部文件系统列表， 包括大小只有0 blocks的文件系统
-h	方便阅读方式显示
-H	等于“-h”，但是计算式，1K=1000，而不是1K=1024
-k	使用Kb单位来显示
-m	使用Mb单位来显示
-l	只显示本地文件系统
-T	文件系统格式

3.4.23 gedit命令

gedit全称 GNU edit 是一个文本编辑器，类似windows里面的txt文本编辑器。

编辑file文本。

```
gedit file
```

3.4.24 管道命令

用法: command 1 | command 2 他的功能是把第一个命令command 1执行的结果作为command 2的输入。

管道命令操作符是："|"它只能处理经由前面一个指令传出的正确输出信息，对错误信息信息没有直接处理能力.

```
ls -l | grep hello
```

3.4.25 重定向

在shell中，标准输入是0，标准输出是1，标准错误是2.

使用>表示重定向。1>表示标准输出重定向，2>表示标准错误重定向。默认情况下>表示输出重定向。

例如：

```
ls > list.txt                        #ls的输出重定向到文件list.txt中。
find / -name "*.c" 2>/dev/null      #标准错误重定向到无底洞文件。
find / -name "*.c" 2>/dev/null      #标准错误重定向到无底洞文件。
find / -name "*.c" >/dev/null 2>&1  #标准输出、标准错误重定向到无底洞文件。
```

3.4.26 后台运行

Linux中可以使用&，让程序在后台运行。

例如：

```
cat &
```

3.4.27 awk命令

awk就是把文件逐行的读入，以空格为默认分隔符将每行切片，切开的部分再进行各种分析处理。

```
awk [-F  field-separator]  'commands'  [input-file(s)]
```

其中，commands 是真正awk命令，[-F 域分隔符]是可选的。 input-file(s) 是待处理的文件。

print 是awk打印指定内容的主要命令, $1 分割出来的第一段，$2分割出来的第二段，依次类推，$0代表所有字段例如：

```
ls -l | awk '{print $1 "\t" $2 "\t" $3 "\t" $4}'
```

- 将/proc/meminfo文件中的字段提取出来，并且在每个字段前面添加meminfo:

```
awk -F ':' '{print "meminfo:" $1}' /proc/meminfo
```

3.4.28 查看Ubuntu版本

```
where@ubuntu:~/ch6$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 16.04.1 LTS
Release:    16.04
Codename:   xenial
```

3.4.29 关闭桌面系统

- 使用一下命令进行关闭桌面系统

```
sudo systemctl set-default multi-user.target
sudo reboot
```

- 如果想切换回桌面系统

```
sudo systemctl set-default graphical.target
sudo reboot
```

第四章 用户管理

Linux的多用户概念是指多个用户同时可以使用这个系统。每个用户一般在/home下的都有自己的用户目录，root的家目录在/root。用户执行任务的时候一般在自己的家目录中执行，用户之间互相不影响。

4.1 查看当前登录用户

Linux中使用whoami来查看当前登录用户。

4.2 查看所有用户信息

```
cat /etc/passwd
```

4.3 查看用户在哪些组里面

```
groups [user] #如果不填user，代表当前用户所在的组。
```

4.4 查看组里面有哪些用户

```
cat /etc/group
```

4.5 su与sudo命令

su就是switch user的意思，用来切换不同的用户，sudo在ubuntu中表示以管理员权限运行某条指令，例如：

```
su root   #切换成root用户
```

```
sudo apt-get install samba  #以管理员权限安装samba，否则无法安装成功
```

4.6 创建用户

Linux中可以使用useradd来添加用户，主要参数如下：

```
-s #指定新用户登陆时shell类型
-g #指定所属组，该组必须已经存在，不指定的时候自动创建一个与用户名重名的用户组。
-d #指定用户目录，该目录必须存在
-m #用户目录不存在时，自动创建该目录
```

例如：

```
sudo useradd huang -s /bin/bash -g where -m
```

以上是添加了一个使用bash属于root组的用户huang，并且在/home/下创建了默认的用户目录huang

4.7 设置密码

Linux中使用passwd来给用户设置密码，或者更改密码。

```
passwd [user]
```

```
where@ubuntu:/home$ sudo passwd huang
输入新的 UNIX 密码：
重新输入新的 UNIX 密码：
passwd：已成功更新密码
```

4.8 创建用户组

Linux中使用groupadd来创建用户组。

```
groupadd [group]
```

4.9 组管理

- 添加用户的附属组

```
usermod -a -G [group1,group2,group3...] [user] #-G代表组，group是组名，user是用户，-a追加用户组否则直接替换掉所有的附属组。
或
gpasswd -a [user] [group] #-a代表添加，user是用户，group是组名
```

注意：如果修改的是当前正在使用的用户的附属组，那么需要重新登录该用户，附属组才会生效。

例如：

```
usermod -a -G huang,root where #给where用户添加huang以及root用户组
usermod -G huang,root where    #设置where用户的附加组为huang，root
```

- 移除用户的附属组，不能移除主用户组

```
gpasswd -d [user] [group] #-d代表删除，user是用户， group是组名
```

- 更改组名

```
groupmod -n [newgroup] [oldgroup] #-n代表new，newgroup是新组名，oldgroup是原组名
```

4.10 删除用户

```
deluser --remove-home [user]
```

Linux中使用deluser来删除用户，一般需要添加参数`--remove-home`也可以使用`-r`代表把它的主目录也删除。

```
where@ubuntu:/home$ sudo deluser --remove-home  huang
正在寻找要备份或删除的文件...
正在删除文件...
正在删除用户 'huang'...
警告：组""没有其他成员了。
完成。
```

注意：如果将要删除的用户，其同名用户组没有其他用户，那么这个同名用户组也会被删除。

4.11 删除用户组

```
groupdel [group]
```

第五章 文件属性

在Linux中，每个文件有三组权限，不同用户有不同的文件操作权限，用户被分为文件所属用户user，文件所属组group里面的用户， 以及其他用户others。

例如： `drwxr-xr-x 2 where where 4096 6月 25 16:53 Videos` user的权限是`rwx`，代表可读可写可执行， group的权限是`r-x`，代表可读可执行，others `r-x`代表可读可以执行。

5.1 chmod 更改文件权限

- 文字设定法

```
chmod [who] [ +|-|=] [mode] [filename]
```

  - 操作对象`who`可是下述字母中的任一个或者它们的组合：

```
u #表示“用户（user）”，即文件或目录的拥有者。
g #表示“同组（group）用户”，即与文件属主有相同组ID的所有用户。
o #表示“其他（others）用户”。
a #表示“所有（all）用户”。它是系统默认值。
```

  - 操作符号可以是：

```
+ #添加某个权限。
- #取消某个权限。
= #赋予给定权限并取消其他所有权限（如果有的话）
```

  - 设置`mode`所表示的权限可用下述字母的任意组合：

```
+ #添加某个权限。
- #取消某个权限。
= #赋予特定权限。
```

例如：

```
chmod u+x file               #user加上执行权限
chmod ugo-w file             #user group others去掉写权限
chmod a=rw file              #全部加上读写权限
chmod a= file                #全部去掉权限
chmod u=rwx,g=rw,o=r file    #user拥有读写执行权限，group拥有读写权限，others拥有读权限
```

- 数字设定法

  ```
  chmod [mode] [filename]
  ```

  数字的含义:

  ```
  0表示没有权限，
  1表示可执行权限，
  2表示可写权限，
  4表示可读权限，

  user    group      other
  r w x   r w x      r w x
  4 2 1   4 2 1      4 2 1
  ```

例如:

```
$ chmod 764 file   代表 user有rwx权限，group有rw权限， other有r权限
```

- 目录文件权限

  ```
  #如果没有读权限，无法查看目录内容（ls命令）；

  #如果没有写权限，无法在目录中创建文件，无法删除文件；

  #如果没有执行权限，无法切换到该目录为当前工作路径(cd命令)，不能查看目录完整内容，不能创建文件，不能删除文件，不能更改目录中文件的内容。
  ```

- 非目录文件权限

  ```
  #如果没有读权限，无法查看文件内容；

  #如果没有写权限，无法修改文件内容；

  #如果没有执行权限，无法执行可执行二进制文件。
  ```

- 如果要递归更改目录中的的权限可以加上-R

```
chmod 755 dir -R
```

5.2 chown更改文件拥有者

Linux下使用chown英文change owner的缩写，表示改变文件的用户。

```
chmown [user]:[group] [file]
```

例如：

```
where@ubuntu:~/mydir$ ls -l file
-rwxrw-r-- 1 where where 29078  7月 10 22:43 file
where@ubuntu:~/mydir$ sudo chown root:root file
where@ubuntu:~/mydir$ ls -l file
-rwxrw-r-- 1 root root 29078  7月 10 22:43 file
```

`chown root:root file`表示把file文件的用户组该为root，用户改为root

如果想要递归更改目录中的文件以及子目录，那么需要添加一个参数-R：

- 把目录中所有所有的文件的拥有者都改为某个用户：

```
sudo chown root:root pathname -R
```

第六章 压缩包管理

6.1 tar

tar 是英文Tape archive 磁带档案的缩写，可以用来创建档案，以及释放档案。

主要参数如下:

```
c #创建新的档案文件。
x #从档案文件中释放。

f #file的缩写，使用档案文件，这个选项通常是必选的。
v #报告tar处理的信息。

z #用gzip来压缩/解压缩文件。
j #用bzip2来压缩/解压缩文件。

-C #参数大写的C代表指定压缩包的位置。
```

- 不压缩打包以及解包

```
tar cvf mydir.tar mydir
tar xvf mydir.tar mydir
tar xvf mydir.tar mydir -C /tmp #指定解压到某个目录
```

- 打gz压缩包以及解包

```
tar zcvf mydir.tar.gz mydir
tar zxvf mydir.tar.gz
tar zxvf mydir.tar.gz -C /tmp #指定解压到某个目录
```

- 打bz2压缩包以及解包:

```
tar jcvf mydir.tar.bz2 mydir
tar jxvf mydir.tar.bz2
tar jxvf mudir.tar.bz2 -C /tmp  #指定加压缩到某个目录
```

6.2 rar

打包：把mydir压缩成myrar.rar `-r`代表递归打包。

```
rar a -r myrar.rar mydir
```

解包：把mydir.rar解压缩到当前目录

```
unrar x myrar.rar
```

注意:如果没有rar、unrar需要使用apt-get install rar unrar来安装。

6.3 zip

打包：`-r`递归打包。

```
zip  -r mydir.zip mydir
```

解包：

```
unzip mydir.zip
```

6.4 gzip

gzip只能压缩非目录文件，默认情况下，压缩完源文件删除。

```
-r或--recursive   #递归处理，将指定目录下的所有文件及子目录一并处理。
-d或--decompress  #解开压缩文件。
-v或--verbose     #显示指令执行过
-[num]             #-1表示最快压缩方法（低压缩比），-9表示最慢压缩方法（高压缩比）。
```

```
where@ubuntu:~/workspace$ ls
file
where@ubuntu:~/workspace$ gzip file
where@ubuntu:~/workspace$ ls -l
总用量 4
-rw-r--r-- 1 where wherenew 36 8月 17 22:30 file.gz
```

```
gzip -rv9 dir #递归压缩目录中的每一个文件，每个文件都打包成一个gz文件，并且显示细节
gzip -drv dir #递归解压目录中所有gz文件
```

第七章 安装软件

在ubuntu当中，安装应用程序知道的有三种方法，分别是apt-get、dpkg和安装源码包三种。下面针对每一种方法各举例来说明。

7.1 apt

apt 英文全拼为 Advanced Packaging Tool。apt-get获取软件包的时候是通过/etc/apt/sources.list中配置的源来查找，源其实就是一个个资源站，apt-get先从资源站中获取到软件包的列表，在需要安装的时候再下载对应的软件包，并进行安装。

7.1.1 使用apt安装软件

使用 `sudo apt-get install` 来安装应用程序算是最常见的一种安装方法了，比如我要安装samba这个软件，会帮我把所有的依赖包都一起安装了。

apt-get方法安装的软件，其实也是下载deb包，所有下载的deb包都缓存到了/var/cache/apt/archives目录下。

```
where@ubuntu:~$ sudo apt-get intall samba
正在读取软件包列表... 完成
正在分析软件包的依赖关系树
正在读取状态信息... 完成
下列软件包是自动安装的并且现在不需要了：
  attr libaio1 libhdb9-heimdal libkdc2-heimdal python-dnspython
  samba-dsdb-modules samba-vfs-modules tdb-tools
Use 'apt-get autoremove' to remove them.
下列软件包将被【卸载】：
  samba
升级了 0 个软件包，新安装了 0 个软件包，要卸载 1 个软件包，有 4 个软件包未被升级。
解压缩后将会空出 11.4 MB 的空间。
您希望继续执行吗？ [Y/n] y
```

以上询问是否要继续执行卸载动作，输入y回车即可安装。

7.1.2 使用apt卸载软件

```
apt-get remove  [package]                    #卸载软件
apt-get remove  [package] --purge             #卸载软件，包括配置文件等
apt-get auto-remove [package] --purge         #卸载软件，包括配置文件,依赖包等
```

使用`sudo apt-get remove`来卸载已经安装的软件, 比如卸载掉samba这个软件。

```
where@ubuntu:~$ sudo apt-get remove samba
正在读取软件包列表... 完成
正在分析软件包的依赖关系树
正在读取状态信息... 完成
下列软件包是自动安装的并且现在不需要了：
  attr libaio1 libhdb9-heimdal libkdc2-heimdal python-dnspython
  samba-dsdb-modules samba-vfs-modules tdb-tools
Use 'apt-get autoremove' to remove them.
下列软件包将被【卸载】：
  samba
升级了 0 个软件包，新安装了 0 个软件包，要卸载 1 个软件包，有 4 个软件包未被升级。
解压缩后将会空出 11.4 MB 的空间。
您希望继续执行吗？ [Y/n]
```

以上询问是否要继续执行卸载动作，输入y回车即可卸载。

7.1.3 apt-get 常用参数

```
apt-get update                                  #更新源
apt-cache search  [package]                     #搜索软件包
apt-get install [package] --reinstall           #重新安装包
apt-get upgrade                                 #更新已安装的包
apt-get source                                  #获取源码包
```

7.1.4 更新apt源

安装完ubuntu后，apt默认的源是 http://cn.archive.ubuntu.com/ubuntu/ ，这是官网的源，更新速度快，但是由于服务器在国外，所有下载速度非常慢。如果你要下载的软件包非常小，那么无所谓，如果很大，则非常有必要更换一个快点的源。

比如163源http://mirrors.163.com/ubuntu：

名称	作用
dists	main源存放地，一般使用这个
pools	security、backports等源的存放地
project	官方源的一个认证

系统	版本号	名称
Ubuntu	16.04	xenial
Ubuntu	15.10	willy
Ubuntu	14.04	trusty
Ubuntu	12.04	precise

名称	自由度
main	完全的自由软件
restricted	不完全的自由软件
universe	ubuntu官方不提供支持与补丁，全靠社区支持
muitiverse	非自由软件，完全不提供支持和补丁

例如:ubuntu 16.04版本163源地址

```
deb http://mirrors.163.com/ubuntu/ xenial main restricted universe multiverse
#64bit文件包列表对应地址  http://mirrors.163.com/ubuntu/dists/xenial/main/binary-amd64/Packages.gz
deb http://mirrors.163.com/ubuntu/ xenial-security main restricted universe multiverse
deb http://mirrors.163.com/ubuntu/ xenial-updates main restricted universe multiverse
deb http://mirrors.163.com/ubuntu/ xenial-proposed main restricted universe multiverse
deb http://mirrors.163.com/ubuntu/ xenial-backports main restricted universe multiverse

deb-src http://mirrors.163.com/ubuntu/ xenial main restricted universe multiverse
#源码包列表对应地址 http://mirrors.163.com/ubuntu/dists/xenial/main/source/Sources.gz
deb-src http://mirrors.163.com/ubuntu/ xenial-security main restricted universe multiverse
deb-src http://mirrors.163.com/ubuntu/ xenial-updates main restricted universe multiverse
deb-src http://mirrors.163.com/ubuntu/ xenial-proposed main restricted universe multiverse
deb-src http://mirrors.163.com/ubuntu/ xenial-backports main restricted universe multiverse
```

更新源步骤：

- `sudo cp /etc/apt/sources.list  /etc/apt/sources.list.bak` 拷贝一份。
- `sudo gedit  /etc/apt/sources.list` 打开源配置文件， 删除原来的源，将163的源粘贴进去，保存退出。
- `sudo apt-get update` 更新软件包列表。

7.2 deb包安装

Ubuntu软件包格式为deb，安装方法如下：

```
sudo  dpkg  -i  package.deb
```

dpkg的详细使用方法：

参数	含义
sudo dpkg -i package.deb	安装包
sudo dpkg -r package	删除包
sudo dpkg -P package	删除包（包括配置文件）
dpkg -L package	列出与已经安装软件关联的软件(可以用tab键补全包名)
dpkg -l	列出当前已安装的包

7.3 源码安装

如果要使用make安装的话，那么必须得安装build-essential这个依赖包，安装方法`sudo apt-get install build-essential`。在安装完毕以后，我们就可以进行源码安装。源码安装大致可以分为三步骤：

1. 配置：这是编译源代码的第一步，通过 `./configure` 命令完成。执行此步以便为编译源代码作准备生成makefile文件。

2. 编译：一旦配置通过，可即刻使用 `make` 指令来执行源代码的编译过程。视软件的具体情况而定，编译所需的时间也各有差异，我们所要做的就是耐心等候和静观其变。此步虽然仅下简单的指令，但有时候所遇到的问题却十分复杂。较常碰到的情形是程序编译到中途却无法圆满结束。此时，需要根据出错提示分析以便找到应对之策。

3. 安装：如果编译没有问题，那么执行 `sudo make install` 就可以将程序安装到系统中了。

下面以安装spawn-fcgi-1.6.4.tar.gz为例进行说明。

```
tar -zxf spawn-fcgi-1.6.4.tar.gz    #1.解压缩
cd spawn-fcgi-1.6.4                  #2.进入目录
./configure                          #3.配置
make                                 #4.编译
sudo make install                    #5.安装
```

7.4 搭建一个apt源服务器

- 第一步安装web服务器。

```
sudo apt-get install apache2 dpkg-dev
```

- 第二步将安装包链接到web服务器的目录下

```
cd  /var/www/html/
sudo ln -s /var/cache/apt/archives mypackages
```

`/var/cache/apt/archives` 平常我们apt安装下载到的deb包都在这个路径上
`/var/www/html/` 是我们主机web服务器的路径，在这个路径里面创建一套文件夹

- 第三步生成一个安装包列表

```
sudo mkdir -p /var/www/html/dists/xenial/main/binary-amd64
cd  /var/www/html/
sudo dpkg-scanpackages mypackages | gzip -9 > dists/xenial/main/binary-amd64/Packages.gz
```

- 第四步更新apt源,/etc/apt/sources.list填入

```
deb http://192.168.1.104 xenial main
```

- 第五步同步源

```
sudo apt-get update
```

第八章 网络管理

8.1 ifconfig命令

ifconfig 英文全拼network interfaces configuring，显示或配置网络设备。

- 查看网卡信息,ip、mac地址等等

  `ifconfig`
- 关闭网卡

  `sudo ifconfig ens33 down`
- 开启网卡

  `sudo ifconfig ens33 up`

- 配置临时IP
  `sudo ifconfig ens33 192.168.1.111`

8.2 设置静态IP

- 找到文件并作如下修改：

`sudo vi /etc/network/interfaces`

修改如下部分：

```
auto ens33
iface ens33 inet static
address 192.168.1.104
gateway 192.168.1.1
netmask 255.255.255.0
```

- 修改dns解析

因为以前是dhcp解析，所以会自动分配dns服务器地址而一旦设置为静态ip后就没有自动获取到的dns服务器了要自己设置一个

`sudo vi /etc/resolv.conf`

写上一个公网的DNS

```
nameserver 114.114.114.114
```

- 重启网卡：

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

- 如果还不生效重启系统

8.3 ping命令

ping脉冲，向远端IP地址发送ICMP（Internet Control Message Protocol）协议的数据包，检测网络状态，

```
where@ubuntu:~$ ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_seq=1 ttl=128 time=453 ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=128 time=5.86 ms
64 bytes from 192.168.1.1: icmp_seq=3 ttl=128 time=35.7 ms
64 bytes from 192.168.1.1: icmp_seq=4 ttl=128 time=5.33 ms
64 bytes from 192.168.1.1: icmp_seq=5 ttl=128 time=4.37 ms

64 bytes from 192.168.1.1: icmp_seq=6 ttl=128 time=3.54 ms
```

也可以向域名发送数据包。

```
where@ubuntu:~$ ping www.baidu.com
PING www.a.shifen.com (14.215.177.37) 56(84) bytes of data.
64 bytes from 14.215.177.37: icmp_seq=1 ttl=128 time=18.2 ms
64 bytes from 14.215.177.37: icmp_seq=2 ttl=128 time=14.4 ms
64 bytes from 14.215.177.37: icmp_seq=3 ttl=128 time=12.3 ms
64 bytes from 14.215.177.37: icmp_seq=4 ttl=128 time=23.0 ms
```

8.4 netstat命令

查看当前网络接口信息。

```
-a #显示所有socket，包括正在监听的。
-c #每隔1秒就重新显示一遍，直到用户中断它。
-n #以网络IP地址代替名称，显示出网络连接情形，显示端口。
-t #显示TCP协议的连接情况。
-u #显示UDP协议的连接情况。
-p #显示建立相关链接的程序名。
-l #查看正在处于监听状态的程序。
#LISTEN和LISTENING的状态只有用-a或者-l才能看到
```

常用组合：

```
sudo netstat -apn
```

第九章 进程管理

9.1 who命令

who功能是显示目前登入系统的用户信息，有多少的终端被使用。

终端是一种字符型设备，它有多种类型，通常使用tty来简称各种类型的终端设备。tty是Teletype的缩写。

pty伪终端（Pseudo Terminal），图形用户终端，远程登录终端。

ctrl+alt+ [F1~F7]打开不同的终端，F1~F6为字符终端，F7为图形终端。

who的主要参数如下:

```
-H #显示每列的标题。
-q #显示登录用户名和登录用户数量。
```

```
where@ubuntu:~$ who
huang    tty1         2016-11-19 21:19
where    tty7         2016-11-19 21:03 (:0)
where@ubuntu:~$ who -H
名称    线路        时间            备注
huang    tty1         2016-11-19 21:19
where    tty7         2016-11-19 21:03 (:0)
where@ubuntu:~$ who -q
huang where
# 用户数=2
```

9.2 ps命令

ps是英文process status的缩写，用来查看进程的状态，默认情况下，只显示当前终端的进程的PID、TTY、TIME、CMD。pstree树形结构显示进程。

```
-a #显示终端上的所有进程，包括其他用户的进程。
-x #显示没有控制终端的进程，并且多了stat字段
-e #显示所有进程。

-f #全格式，多了UID、ppid、STIME字段。
-u #把当前用户的进程全部显示出来，多了%CPU、%MEM、VSZ、RSS、STAT字段
```

例如最常用的参数组合`ps -aux` 或者`ps -ef`

```
where@ubuntu:~$ ps -aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.3   4576  3440 ?        Ss   05:34   0:05 /sbin/init
root         2  0.0  0.0      0     0 ?        S    05:34   0:00 [kthreadd]
root         3  0.2  0.0      0     0 ?        S    05:34   2:31 [ksoftirqd/0]
root         5  0.0  0.0      0     0 ?        S<   05:34   0:00 [kworker/0:0H]
root         7  0.0  0.0      0     0 ?        S    05:34   0:10 [rcu_sched]
root         8  0.0  0.0      0     0 ?        S    05:34   0:00 [rcu_bh]
root         9  0.0  0.0      0     0 ?        S    05:34   0:00 [migration/0]
root        10  0.0  0.0      0     0 ?        S    05:34   0:01 [watchdog/0]
root        11  0.0  0.0      0     0 ?        S<   05:34   0:00 [khelper]
root        12  0.0  0.0      0     0 ?        S    05:34   0:00 [kdevtmpfs]
root        13  0.0  0.0      0     0 ?        S<   05:34   0:00 [netns]
```

```
USER    #用户名
UID     #用户ID（User ID）
PID     #进程ID（Process ID）
PPID    #父进程的进程ID（Parent Process id）
SID     #会话ID（Session id）
%CPU    #进程的cpu占用率
%MEM    #进程的内存占用率
VSZ     #进程所使用的虚存的大小（Virtual Size）
RSS     #进程使用实际内存的大小.
TTY     #与进程关联的终端（tty）
STAT    #进程的状态：进程状态使用字符表示的（STAT的状态码）
    R   #运行Runnable (on run queue) 正在运行或在运行队列中等待。
    S   #睡眠Sleeping 休眠中,可以被中断。
    Z   #僵死Zombie（a defunct process) 进程已终止, 但进程描述符存在.
    D   #不可中断Uninterruptible sleep (ususally IO) cpu不能切到其他进程运行，必须要等到这个状态结
束。
    T   #task_stopped or task_traced 暂停状态或跟踪状态
    X   #死掉的进程

    <    #高优先级
    N    #低优先级
    L    #有些页被锁进内存
    s    #包含子进程
    +    #位于后台的进程组;
    l    #多线程

START   #进程启动的时间。
TIME    #进程消耗的cpu时间。
COMMAND #正在执行的命令行命令
```

9.3 jobs命令

jobs命令用于显示Linux中当前终端的任务列表及任务状态，包括后台运行的任务。该命令可以显示任务号及其对应的进程号。

```
-l #显示进程号,作业号等;
-p #显示进程号;
-r #仅输出运行状态（running）的任务;
-s #仅输出停止状态（stoped）的任务。
```

`ctrl+z` 发送一个`SIGSTOP`的信号，使进程进入暂停状态。

例如：

```
where@ubuntu:~$ cat
^Z
[3]+  已停止               cat
```

`[3]`代表作业号3

```
where@ubuntu:~$ jobs
[1]-  已停止               cat
[2]   已停止               cat
[3]+  已停止               cat
```

9.4 fg命令

fg是英文Foreground缩写， 把指定的后台作业或挂起作业移到前台运行。

```
where@ubuntu:~$ jobs
[1]-  已停止               cat
[2]   已停止               cat
[3]+  已停止               cat
where@ubuntu:~$ fg 1
cat
```

例如，使用vi的时候，不小心按了ctrl+z把vi挂到后台去了，这个时候可以使用fg将该任务提到前台。

9.5 bg命令

bg是英文background缩写， 把被挂起的进程提到后台执行。

```
where@ubuntu:~$ jobs
[1]+  已停止               cat
[2]   已停止               cat
[3]   已停止               cat
where@ubuntu:~$ bg 1
[1]+ cat &
```

后台运行的任务源码helloworld.c：

```
#include <stdio.h>
int main(void)
{
    int i = 0;
    while(i++ < 10)
    {
        sleep(1);
        puts("helloworld");
    }
    return 0;
}
```

```
gcc helloworld.c
./a.out &
```

9.6 kill命令

kill向指定进程发送信号。

```
kill [-option] [PID]
```

`kill -l`查看所有信号，可以指定使用启动的某个信号发送给进程例如：`kill -9 19204`

```
where@ubuntu:~$ kill -l
 1) SIGHUP   2) SIGINT   3) SIGQUIT  4) SIGILL   5) SIGTRAP
 6) SIGABRT  7) SIGBUS   8) SIGFPE   9) SIGKILL 10) SIGUSR1
11) SIGSEGV 12) SIGUSR2 13) SIGPIPE 14) SIGALRM 15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD 18) SIGCONT 19) SIGSTOP 20) SIGTSTP
21) SIGTTIN 22) SIGTTOU 23) SIGURG  24) SIGXCPU 25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF 28) SIGWINCH    29) SIGIO   30) SIGPWR
31) SIGSYS  34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

- 2) SIGINT

程序终止(interrupt)信号, 在用户键入Ctrl-C时发出，用于通知前台进程组终止进程。

- 3) SIGQUIT

和SIGINT类似, 但由QUIT字符(通常是Ctrl-\)来控制. 进程在因收到SIGQUIT退出时会产生core文件, 在这个意义上类似于一个程序错误信号。

- 9) SIGKILL

用来立即结束程序的运行. 本信号不能被阻塞、处理和忽略。如果管理员发现某个进程终止不了，可尝试发送这个信号。

- 11) SIGSEGV

试图访问未分配给自己的内存, 或试图往没有写权限的内存地址写数据段错误。

- 19) SIGSTOP

停止(stopped)进程的执行。注意它和terminate以及interrupt的区别：该进程还未结束, 只是暂停执行。本信号不能被阻塞处理或忽略。（Ctrl+z）

也可以用来给当前终端的任务发送信号，例如：`kill -9 %1`给当前终端任务号1的任务发送9号信号。

9.7 top命令

top是Linux下常用的性能分析工具，能够实时显示系统中各个进程的资源占用状况，类似于Windows的任务管理器。

`top`

```
top - 15:11:31 up 21:29,  4 users,  load average: 0.00, 0.01, 0.05
Tasks: 187 total,   1 running, 185 sleeping,   1 stopped,   0 zombie
%Cpu(s):  0.7 us,  0.4 sy,  0.0 ni, 98.9 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem:   1024236 total,   984040 used,    40196 free,    73540 buffers
KiB Swap:  1046524 total,    26120 used,  1020404 free.   542608 cached Mem

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
20405 root      20   0  212424  53188  25020 S  1.7  5.2   1:55.44 Xorg
21145 where     20   0  135372  34988  27440 S  1.0  3.4   0:39.14 gnome-termin+
 1314 root      20   0   27340   4740   4360 S  0.3  0.5   2:54.32 vmtoolsd
20972 where     20   0   86120  31236  24368 S  0.3  3.0   0:58.62 vmtoolsd
29747 root      20   0       0      0      0 S  0.3  0.0   0:00.92 kworker/u16:0
    1 root      20   0    4576   3448   2508 S  0.0  0.3   0:06.30 init
    2 root      20   0       0      0      0 S  0.0  0.0   0:00.04 kthreadd
    3 root      20   0       0      0      0 S  0.0  0.0   2:36.11 ksoftirqd/0
    5 root       0 -20       0      0      0 S  0.0  0.0   0:00.00 kworker/0:0H
    7 root      20   0       0      0      0 S  0.0  0.0   0:14.33 rcu_sched
    8 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcu_bh
    9 root      rt   0       0      0      0 S  0.0  0.0   0:00.00 migration/0
   10 root      rt   0       0      0      0 S  0.0  0.0   0:01.67 watchdog/0
   11 root       0 -20       0      0      0 S  0.0  0.0   0:00.00 khelper
   12 root      20   0       0      0      0 S  0.0  0.0   0:00.00 kdevtmpfs
   13 root       0 -20       0      0      0 S  0.0  0.0   0:00.00 netns
   14 root       0 -20       0      0      0 S  0.0  0.0   0:00.00 perf
```

- 各列的含义:

```
列名    含义
PID     #进程id
USER    #进程所有者的用户名
PR      #优先级
NI      #nice值。负值表示高优先级，正值表示低优先级
%CPU    #上次更新到现在的CPU时间占用百分比
TIME+   #进程使用的CPU时间总计，单位1/100秒
%MEM    #进程使用的物理内存百分比
VIRT    #进程使用的虚拟内存总量，单位kb。VIRT=SWAP+RES
SWAP    #进程使用的虚拟内存中，被换出的大小，单位kb。
RES     #进程使用的、未被换出的物理内存大小，单位kb。RES=CODE+DATA
SHR     #共享内存大小，单位kb
S       #进程状态(D=不可中断的睡眠状态,R=运行,S=睡眠,T=跟踪/停止,Z=僵尸进程)
COMMAND #命令名/命令行
```

按`q`就可退出。

第十章 挂载与卸载

Linux采用树形的文件管理系统，想要访问其他的块设备文件时，需要以挂载的方式，添加到树形目录中，这样才能访问块设备，无法直接访问/dev下的设备文件。

10.1 mount挂载命令

命令格式：

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

其中：

```
-t #vfstype 指定文件系统的类型，通常不必指定。mount 会自动选择正确的类型。
    #常用类型有：
    iso9660     #光盘或光盘镜像
    vfat        #Windows 9x fat32文件系统
    ntfs        #Windows NT ntfs文件系统
    smbfs       #Mount Windows文件网络共享
    nfs         #UNIX(LINUX) 文件网络共享

-o #options 主要用来描述设备或档案的挂接方式。
    #常用的参数有：
    loop:        #用来把一个文件当成硬盘分区挂接上系统
    ro:          #采用只读方式挂接设备
    rw:          #采用读写方式挂接设备
    iocharset:   #指定访问文件系统所用字符集,iocharset=utf8,iocharset=gb2312
    remount:     #重新挂载

device  #要挂接(mount)的设备。

dir     #需要挂载到那个目录。
```

例如挂载u盘：

```
sudo mount /dev/sdb1 /media/where/
```

我的U盘被ubuntu识别成sdb1，所以我将`/dev/sdb1`,挂载到`/media/where`目录，这样就能访问u盘里面的内容了。

10.2 查看挂载情况

可以使用df 以及sudo fdisk -l查看挂载情况。

```
where@ubuntu:/media/where$ df
文件系统           1K-块      已用      可用 已用% 挂载点
udev             502120         4    502116    1% /dev
tmpfs            102424      1596    100828    2% /run
/dev/sda1      40120704  11703036  26356612   31% /
none                 4         0         4    0% /sys/fs/cgroup
none              5120         0      5120    0% /run/lock
none            512116       152    511964    1% /run/shm
none            102400        40    102360    1% /run/user
/dev/sdb1      7176320     91849   7084471    2% /media/where
```

10.3 umount卸载命令

umount 参数可以是被挂载的目录或者设备。

例如:

```
umount /dev/sdb1
```

```
umount /media/where
```

效果是一样的。

注意：不要在挂载的目录中，卸载当前的挂载目录。

第十一章 ftp服务器的搭建

FTP（File Transfer Protocol，文件传输协议） 是 TCP/IP 协议组中的协议之一。

FTP协议包括两个组成部分，其一为FTP服务器，其二为FTP客户端。

默认情况下FTP协议使用TCP端口中的 20和21这两个端口，其中20用于传输数据，21用于传输控制信息。

11.1 安装vsftpd

- 使用apt-get 安装

```
sudo apt-get install vsftpd
```

- 配置vsftpd.conf文件

```
sudo cp /etc/vsftpd.conf /etc/vsftpd.conf.bak
sudo vi /etc/vsftpd.conf
```

配置文件详细介绍

```
#接受本地用户
local_enable=YES
#本地用户上传文件的umask。
local_umask=022

#可以上传(全局控制).
write_enable=YES
```

- 重启ftp服务器

```
sudo service vsftpd restart
```

注意：client端get文件出现226错误，代表client端没有本地写权限，无法创建文件，可能是c:\盘

11.2 ftp客户端登录

- 登录ftp服务器后，可以使用`get`获取服务器上的文件，`mget`是获取多个文件。
- 使用`put`可以上传文件到服务器，使用`mput`上传多个文件。
- `ls` 查看文件。
- `mkdir`创建文件。
- `rmdir`删除文件夹。
- `pwd`查看当前工作路径。
- `cd`切换工作路径
- `delete`删除文件。

第十二章 文件共享

nfs服务器

NFS（Network File System）即网络文件系统， 在NFS的应用中，本地NFS的客户端应用可以透明地读写位于远端NFS服务器上的文件，就像访问本地文件一样。nfs是通过挂载的方式来访问。

- 安装nfs服务器

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

- 设置`/etc/exports`配置文件

```
sudo vi /etc/exports

/home/where/nfs *(rw,sync,no_root_squash)
```

添加这行配置，rw读写权限，sync同步，no_root_squash来访的root用户保持root帐号权限。

客户端的指定方式:

指定ip地址的主机：192.168.1.100

指定子网中的所有：ip:192.168.1.1/255.255.255.0

所有主机：*

权限指定方式:

rw读写

ro只读

- 在用户目录下创建nfs目录

```
mkdir /home/where/nfs
```

- 重启服务器，重新加载配置文件

```
sudo service nfs-kernel-server restart
```

- 挂载

```
sudo mount -t nfs -o nolock -o tcp 192.168.1.111:/home/where/nfs /mnt
```

```
-t nfs                              #挂载类型
-o nolock                           #读写的时候不锁定
-o tcp                              #tcp模式，
192.168.1.111:/home/where/nfs    #服务器地址，

/mnt                                #挂载到本地mnt目录
```

samba服务器

Samba是在Linux和UNIX系统上实现SMB协议的一个免费软件，由服务器及客户端程序构成。SMB（Server Messages Block，信息服务块）是一种在局域网上共享文件和打印机的一种通信协议，它为局域网内的不同计算机之间提供文件及打印机等资源的共享服务。

我们可以使用samba来实现我们Linux虚拟机与windows主机文件共享，便于开发。

- 第一步安装samba服务器以及客户端

```
sudo apt-get install samba samba-common
```

- 第二步修改配置文件

```
sudo vi /etc/samba/smb.conf
```

```
[share]
    comment=my samba path  #注释
    path=/home/myhome/share #共享的路径
    browseable = yes #是否可以浏览
    read only = no #是否只读
    writable=yes#是否可写
    guest ok=yes#是否支持来宾用户
```

- 第三步重启服务器

```
sudo service smbd restart
```

- 如果guest ok设置为no，可以通过smbpasswd来设置一个smb用户

```
sudo smbpasswd -a [user] #user必须是本地已经存在的用户
```

- 删除用户

```
sudo smbpasswd -x [user]
```

- windows系统想要访问时，直接按快捷键`win键+R`，然后输入对应的ip地址，例如“\\192.168.1.102”

如果想删除用户登录信息，可以使用cmd中执行`net use * /d /y`

第十四章 telnet服务器

Telnet是Internet远程登录服务的标准协议和主要方式，最初由ARPANET开发，现在主要用于Internet会话，它的基本功能是允许用户登录进入远程主机系统。

- 安装openbsd-inetd

```
sudo apt-get install openbsd-inetd
```

- 安装telnetd服务器

```
sudo apt-get install telnetd
```

- 重启openbsd-inetd

```
sudo service openbsd-inetd restart
```

- 使用telnet客户端登录，比如windows可以使用secureCRT，或者直接使用Linux自带telnet客户端，退出使用`exit`

```
telnet 127.0.0.1
```

第十五章 ssh服务

SSH 为 Secure Shell 的缩写，由 IETF 的网络小组（Network Working Group）所制定；SSH 为建立在应用层和传输层基础上的安全协议。

SSH 是目前较可靠，专为远程登录会话和其他网络服务提供安全性的协议。利用 SSH 协议可以有效防止远程管理过程中的信息泄露问题。

- ssh服务安装

```
sudo apt-get install openssh-server
```

- 登录ssh，在Linux端可以直接使用ssh命令登录，windows可以使用secureCRT登录,退出也是`exit`

```
ssh where@127.0.0.1 #where是登录用户名，127.0.0.1为ssh服务ip地址
```

第十六章 环境变量

环境变量（environment variables）一般是指在操作系统中用来指定操作系统运行环境的一些参数。

15.1 查看环境变量

- env命令，查看所有系统变量。
- echo命令，查看某个变量

```
where@ubuntu:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games
```

- 控制台设置环境变量

```
export PATH=$PATH:/home/where/mydir
```

- 在~/.bashrc中设置环境变量

`vi ~/.bashrc` 在末尾添加 `export PATH=/home/where/mydir:$PATH`

修改完~/.bashrc后需要，使用source来使得环境变量生效:

```
where@ubuntu:~$ source  .bashrc
where@ubuntu:~$ echo $PATH
/home/where/mydir:/home/where:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/g
ames:/usr/local/games
```

~/.bashrc的加载时机是新开一个终端就加载。

- 在/etc/profile中设置环境变量

`vi /etc/profile` 在末尾添加 `export PATH=/home/where/mydir:$PATH`

修改完/etc/profile后需要，使用source来使得环境变量生效:

```
where@ubuntu:~$ source  /etc/profile
where@ubuntu:~$ echo $PATH
/home/where/mydir:/home/where:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/g
ames:/usr/local/games
```

/etc/profile加载的时机时开机加载。

- 常见环境变量

LIBRARY_PATH：用于在程序编译期间查找动态链接库时指定查找共享库的路径。

LD_LIBRARY_PATH：用于在程序加载运行期间查找动态链接库时指定除了系统默认路径之外的其他路径。

C_INCLUDE_PATH：指明c头文件的搜索路径。

CPLUS_INCLUDE_PATH:指明c++头文件的搜索路径。

第十六章 VI（VIM）的使用

打开单个文件

`vim file`

同时打开多个文件

`vim file1 file2 file3 ...`

`:bn`下一个文件
`:bp`上一个文件

在vim窗口中打开一个新文件

`:open file`

在新窗口中打开文件

`:split file`

vim的模式

正常模式：（按Esc或Ctrl+[进入） 左下角显示文件名或为空
插入模式：（按i键进入） 左下角显示--INSERT--
可视模式：（v或者V） 左下角显示--VISUAL--
末行模式：　　(:、/、?等)

导航命令

% 括号匹配

插入命令

```
i #在当前位置生前插入
I #在当前行首插入
a #在当前位置后插入
A #在当前行尾插入
o #在当前行之后插入一行
O #在当前行之前插入一行
```

查找命令

```
/text    #查找text，按n键查找下一个，按N键查找前一个。
?text    #查找text，反向查找，按n键查找下一个，按N键查找前一个。
```

替换命令

```
: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

s代表替换，g表示一整行，%表示所有行的第一个
```

移动命令

```
h                #左移一个字符
l                #右移一个字符，这个命令很少用，一般用w代替。
k                #上移一个字符
j                #下移一个字符
w                #向前移动一个单词（光标停在单词首部）
b                #向后移动一个单词
gg               #移动到文件头。
G(shift + g)     #移动到文件尾。
num +g+g         #跳到某行
Ctrl + d         #向下滚动半屏
Ctrl + u         #向上滚动半屏
Ctrl + f         #向下滚动一屏
Ctrl + b         #向上滚动一屏
f(find)          #命令也可以用于移动，fx将找到光标后第一个为x的字符，3fd将找到第三个为d的字符。
F                #同f，反向查找。
```

撤销和重做

```
u            #撤销（Undo）
Ctrl + r     #重做（Redo），即撤销的撤销。
```

删除命令

```
x        #删除当前字符
dw       #删除一个单词
d$       #删除当前字符之后的所有字符（本行）
dd       #删除当前行
10d      #删除当前行开始的10行。
```

拷贝和粘贴

```
yy  #拷贝当前行
nyy #拷贝当前后开始的n行，比如2yy拷贝当前行及其下一行。
p   #在当前光标后粘贴,如果之前使用了yy命令来复制一行，那么就在当前行的下一行粘贴。
```

剪切命令

正常模式下按v（逐字）或V（逐行）进入可视模式，然后用jklh命令移动即可选择某些行或字符，再按d即可剪切

ndd 剪切当前行之后的n行。利用p命令可以对剪切的内容进行粘贴

退出命令

```
:wq #保存并退出
ZZ  #保存并退出
:q! #强制退出并忽略所有更改
```

窗口命令

```
:split file     #用新窗口打开文件split打开的窗口都是横向的，使用vsplit可以纵向打开窗口
Ctrl+w s        #水平分屏
Ctrl+w v        #垂直分屏
Ctrl+w w        #移动到下一个窗口
Ctrl+w j        #移动到下方的窗口
Ctrl+w k        #移动到上方的窗口
Ctrl+w h        #移动到左边的窗口
Ctrl+w l        #移动到右边的窗口
:close          #最后一个窗口不能使用此命令，可以防止意外退出vim
:q              #如果是最后一个被关闭的窗口，那么将退出vim
ZZ              #保存并退出
:only           #关闭所有窗口，只保留当前窗口
:wqa            #全部关闭窗口，并且写入文件
```

编程辅助

```
shift+k      #跳转到man帮助
num+shift+k #跳转到num页的man帮助

ctrl+p       #补全函数，需要包含头文件的情况下。使用ctrl+p选择上一条，ctrl+n选择下一条。
[+ctrl+i     #跳转到函数声明、变量和#define，包括系统函数，同文件的声明函数变量等。

ctrl+i       #向前跳到前几次光标位置
ctrl+o       #向后跳到后几次光标位置

ctrl+]       #跳转到函数定义、变量和#define，必须使用ctags生成分析文件tags才有有效
```

第十七章 编译基础

17.1 gcc

GCC（GNU Compiler Collection，GNU编译器套件），是由 GNU 开发的编程语言编译器。它是以GPL许可证所发行的自由软件，也是 GNU计划的关键部分。

GCC 原名为 GNU C 语言编译器（GNU C Compiler），因为它原本只能处理 C语言。GCC 很快地扩展，变得可处理 C++。后来又扩展能够支持更多编程语言，如Fortran、Pascal、Objective-C、Java、Ada、Go以及各类处理器架构上的汇编语言等，所以改名GNU编译器套件（GNU Compiler Collection）。

17.1.1 常用参数

```
-v / --v /--version     #查看gcc版本号
-E      #生成预处理文件
-S      #表示在程序编译期间，在生成汇编代码后，停止。
-c      #只生成.o文件，不链接成为可执行文件
-o      #确定输出文件的名称
-g      #包含调试信息，要想对源代码进行调试，我们就必须加入这个选项
-On     #n=0~3 编译优化,n越大优化得越多
-Wall   #提示更多警告信息，比如未使用的变量等
-D      #编译时定义,注意-D之后没有空格,如-DDEBUG,定义DEBUG这个宏
-I      #指定头文件目录
-L      #指定库路径
-l      #指定链接某个库，例如lpthread链接libpthread.so库
-static #默认情况下， GCC在链接时优先使用动态链接库，只有当动态链接库不存在时才考虑使用静态链接库，如果需要的话可以在编译时加上-static选项，强制使用静态链接库。
```

17.1.2 编译过程

编译过程是分为四个阶段进行的，即预处理(也称预编译，**Preprocessing**)、编译(**Compilation**)、汇编(**Assembly**)和连接(**Linking**)。

源码: test.c

```c
#include <stdio.h>
#define MAX_LINE 1024
int hello()
{
    int a = 1;
    return a;
}
int main()
{
    int a = 0;
    a = hello();
    return 0;
}
```

- 预处理

  ```
  gcc -E test.c -o test.i
  ```

- 编译成汇编代码

  ```
  gcc -S test.i -o test.s
  ```

- 汇编代表编译成目标文件

  ```
  gcc -c test.s -o test.o
  ```

- 连接成可执行程序

```
gcc test.o -o test
```

- 使用优化以及调试信息

```
gcc -g -O0 test.c -o test0
gcc -g -O1 test.c -o test1
gcc -g -O2 test.c -o test2
gcc -g -O3 test.c -o test3
```

- 更多警告信息

```
gcc -Wall test.c -o test
```

17.2 动态库与静态库

现实中每个程序都要依赖很多基础的底层库，不可能每个人的代码都从零开始。尽量不重复做别人已经做过的事，“站在巨人的肩膀上”做事情。

根据链接时期的不同，库又有：静态库和共享库（动态库）。

二者的不同点在于代码被载入的时刻不同， 静态库的代码在编译过程中已经被载入可执行程序，因此体积较大。共享库的代码是在可执行程序运行时才载入内存的，在编译过程中仅简单的引用，因此代码体积较小。

一般gcc在编译的时候默认使用动态库，可以直接添加-static强制使用静态库。

静态库的名字一般是libxxx.a（Linux）.

动态库的名字一般是libxxx.so（Linux），有时候也是 libxxx.so.major.minor，xxxx是该lib的名称，major是主版本号， minor是副版本号。

17.2.1 file命令

file程序是用来判断文件类型的，并且可以查看文件是否使用了动态库。

```
where@ubuntu:~$ file /bin/ls
/bin/ls: ELF 32-bit LSB  executable, Intel 80386, version 1 (SYSV), dynamically linked (uses
shared libs), for GNU/Linux 2.6.24, BuildID[sha1]=de9c5af34773e23cf554e0da78320e76a3b85120,
stripped
```

17.2.2 ldd命令

ldd是英文List dynamic dependencies的缩写，用来查看动态库，如果目标程序没有链接动态库，则打印“not a dynamic executable”（不是动态可执行文件）

```
where@ubuntu:~$ ldd /bin/ls
Linux-gate.so.1 =>  (0xb7765000)
libseLinux.so.1 => /lib/i386-Linux-gnu/libseLinux.so.1 (0xb772a000)
libacl.so.1 => /lib/i386-Linux-gnu/libacl.so.1 (0xb7721000)
libc.so.6 => /lib/i386-Linux-gnu/libc.so.6 (0xb7572000)
libpcre.so.3 => /lib/i386-Linux-gnu/libpcre.so.3 (0xb7534000)
libdl.so.2 => /lib/i386-Linux-gnu/libdl.so.2 (0xb752f000)
/lib/ld-Linux.so.2 (0x800d3000)
libattr.so.1 => /lib/i386-Linux-gnu/libattr.so.1 (0xb7529000)
```

17.2.3 编译动态库

- 编写mylib.h

```
#ifndef __MYLIB_H__
#define __MYLIB_H__
int test();
#endif
```

- 编写mylib.c

```
#include <stdio.h>
#include "mylib.h"
int test()
{
        printf("mytest\n");
}
```

- 编译成mylib.o文件

```
gcc -fPIC -c mylib.c
```

-fPIC 作用于编译阶段，告诉编译器产生与位置无关代码(Position-Independent Code)， 则产生的代码中，没有绝对地址，全部使用相对地址，故而代码可以被加载器加载到内存的任意 位置，都可以正确的执行。这正是共享库所要求的，共享库被加载时，在内存的位置不是固定的。

- 链接成动态库libmy.so

```
gcc  -shared  mylib.o -o libmy.so
```

- 编译跟链接一起执行

```
gcc -shared -fPIC mylib.c -o libmy.so
```

17.2.4 调用动态库

- 调用动态库，编写调用代码test.c

```
#include "mylib.h"
int main()
{
        test();
        return 0;
}
```

- 编译时带上库路径

```
gcc -o test test.c libmy.so
```

- 或者指定链接路径

```
gcc -o test test.c -L./ -lmy
```

- 设置环境变量LIBRARY_PATH

```
export LIBRARY_PATH=$LIBRARY_PATH/homw/where/lib   #/home/where为我当前库路径
gcc -o test test.c -lmy
```

- 或者把库拷贝到系统库路径中

```
mv libmy.so /usr/lib #或者 /lib /usr/local/lib
gcc -o test test.c -lmy
```

gcc默认查找动态的顺序：

```
1、-L参数指定的路径

2、LIBRARY_PATH环境变量指明库搜索路径

3、gcc内定库/lib、/usr/lib、/usr/local/lib
```

17.2.5 头文件引用

```
gcc test.c -o test -lmy
```

- 指定包含的文件路径

```
gcc test.c -o test -I./
```

- 修改C_INCLUDE_PATH

```
export C_INCLUDE_PATH=$C_INCLUDE_PATH:/home/where/lib
gcc test.c -o test
```

- gcc默认查找头文件的顺序

```
1、当前编译路径
2、-I 指定的路径
3、C_INCLUDE_PATH，CPLUS_INCLUDE_PATH环境变量等路径
4、再找系统内定目录如/usr/include、/usr/local/include
```

17.2.6运行时库查找

1、在配置文件/etc/ld.so.conf中指定动态库搜索路径

```
sudo vi /etc/ld.so.conf.d/mylib.conf #新建mylib.conf写入你自己的库路径，如/home/where/lib
sudo ldconfig                        #更新一下缓存才能生效
```

2、通过环境变量LD_LIBRARY_PATH指定动态库搜索路径

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/where/lib
```

3、默认的动态库搜索路径/lib、/usr/lib

17.2.7 编译静态库

- 编写mylib.h

```
#ifndef __MYLIB_H__
#define __MYLIB_H__
int test();
#endif
```

- 编写mylib.c

```
#include <stdio.h>
#include "test.h"
int test()
{
        printf("mytest\n");
}
```

- 编译成mylib.o文件

```
gcc -c mylib.c
```

- 使用ar生成静态库

```
ar rcs libmy.a mylib.o
```

```
参数r：#在库中插入模块(替换)。当插入的模块名已经在库中存在，则替换同名的模块。如果若干模块中有一个模块在库中不存在，ar显示一个错误消息，并不替换其他同名模块。默认的情况下，新的成员增加在库的结尾处，可以使用其他任选项来改变增加的位置。
参数c：#创建一个库。不管库是否存在，都将创建。
参数s：#创建目标文件索引，这在创建较大的库时能加快时间。
```

17.2.8 调用静态库

- 使用静态库编译

```
 gcc -o test test.c libmy.a
```

- 或者

```
gcc -o test test.c -L. -lmy
```

- 在静态动态库都存在的情况，强制使用静态库

```
gcc -o test test.c -L. -lmy -static
```

17.3 gdb的使用

gdb（GNU symbolic debugger）是一个由GNU开源组织发布的、UNIX/LINUX操作系统下的、基于命令行的、功能强大的程序调试工具。

```
start            #开始调试,停在第一行代码处,(gdb)start
l                #list的缩写查看源代码,(gdb)l
b <lines>        #b: Breakpoint的简写，设置断点。(gdb) b 8
b <func>         #b: Breakpoint的简写，设置断点。(gdb) b main
i breakpoints    #i:info 的简写。(gdb)i breakpoints
d [bpNO]         #d: Delete breakpoint的简写，删除指定编号的某个断点，或删除所有断点。断点编号从1开始递增。(gdb)d 1

s                #s: step执行一行源程序代码，如果此行代码中有函数调用，则进入该函数；(gdb) s
n                #n: next执行一行源程序代码，此行代码中的函数调用也一并执行。(gdb) n

r                #Run的简写，运行被调试的程序。如果此前没有下过断点，则执行完整个程序；如果有断点，则程序暂停在第一个可用断点处。(gdb) r
c                #Continue的简写，继续执行被调试程序，直至下一个断点或程序结束。(gdb) c
finish           #函数结束

p [var]              #Print的简写，显示指定变量（临时变量或全局变量 例如 int a）的值。(gdb) p a
display [var]        #display，设置想要跟踪的变量(例如 int a)。(gdb) display a
undisplay [varnum]   #undisplay取消对变量的跟踪，被跟踪变量用整型数标识。(gdb) undisplay 1
set args             #可指定运行时参数。(gdb)set args 10 20
show args            #查看运行时参数。
q                    #Quit的简写，退出GDB调试环境。(gdb) q
help [cmd]           #GDB帮助命令，提供对GDB名种命令的解释说明。如果指定了“命令名称”参数，则显示该命令的详细说明；如果没有指定参数，则分类显示所有GDB命令，供用户进一步浏览和查询。(gdb)help
回车                 #重复前面的命令，(gdb)回车
```

test.c

```c
#include <stdio.h>
int add(int a, int b)
{
    return a + b;
}
int sub(int a, int b)
{
    return a - b;
}
int mul(int a, int b)
{
    return a * b;
}
int main(void)
{
    char buf[] = "helloworld";
    int a = 3;
    int b = 5;
    int c = 0;
    c = add(a, b);
    printf("%s a+b=%d\n", buf, c);
    c = sub(a, b);
    printf("%s a-b=%d\n", buf, c);
    c = mul(a, b);
    printf("%s a*b=%d\n", buf, c);
    return 0;
}
```

- 编译的时候需要加上-g参数

```
gcc -o test -g test.c
gdb test
```

17.4 反汇编

objdump 用于生成反汇编文件,主要依赖objcopy编译时需要-g。

例如：

```
gcc hello.c -g
objdump -dSsx a.out > file
```

```
d  #反汇编需要执行指令的那些section
S  #输出C源代码和反汇编出来的指令对照的格式
s  #显示十六进制文件代码
x  #全部Header信息
```

17.5 检查内存泄漏

最令Linux程序员头疼的莫过于内存泄露了，即使你是在优秀的程序员，你也不能保证所以的malloc操作都有对应的free，那必要的工具就是必不可少的了。在一般的Linux发行版中，有一个自带的工具可以很方便的替你完成这些事，这个工具就是mtrace。

源代码*test.c:*

```
#include <stdio.h>
#include <mcheck.h>
int main()
{
    setenv("MALLOC_TRACE", "output", 1);
    mtrace();
    char * text = ( char * ) malloc (sizeof(char) * 100);
    return 0;
}
```

- 可以看出，只需要在你的程序中插入三行代码，就行

```
第一句，#include <mcheck.h>，包含头文件

第二句，setenv("MALLOC_TRACE", "output", 1); 设置一个环境变量 MALLOC_TRACE=output

第三句，mtrace()，调用mtrace.
```

- 将这个文件编译，注意，编译的时候一地要加上gcc的-g选项。

```
gcc test.c -g -o test
```

- 接着执行可执行文件，然后你会发现当前目录下多了一个output的文件。

```
./test
```

- 这个文件自然不是文本文件，所以需要工具来查看。

```
where@ubuntu:~/workspace$ mtrace test output
- 0x000000000090c010 Free 3 was never alloc'd 0x7f642f397e8d
- 0x000000000090c270 Free 4 was never alloc'd 0x7f642f46205f
- 0x000000000090c290 Free 5 was never alloc'd 0x7f642f4d1a6c

Memory not freed:
-----------------
           Address     Size     Caller
0x000000000090c700     0x64   at /home/where/workspace/test.c:8
```

Memory not freed:代表内存未被释放。

17.6 查看进程打开的文件

```
lsof [options] [name]
```

例如：源码test.c

```
#include <stdio.h>
#include <unistd.h>
int main(void)
{
    FILE* fp = fopen("file", "w");
    if(fp == NULL)
    {
        return 1;
    }
    while(1)
        sleep((unsigned int)1);
    return 0;
}
```

```
gcc test.c
./a.out
```

- 查看哪些进程打开了file文件

```
where@ubuntu:~/workspace$ lsof file
lsof: WARNING: can't stat() tracefs file system /sys/kernel/debug/tracing
      Output information may be incomplete.
COMMAND  PID  USER   FD   TYPE DEVICE SIZE/OFF   NODE NAME
a.out   6065 where    3w   REG    8,1        0 429381 file
a.out   6070 where    3w   REG    8,1        0 429381 file
```

- 查看进程pid为6065的进程所开发的文件

```
where@ubuntu:~/workspace$ lsof -p 6065
lsof: WARNING: can't stat() tracefs file system /sys/kernel/debug/tracing
      Output information may be incomplete.
COMMAND  PID  USER   FD   TYPE DEVICE SIZE/OFF    NODE NAME
a.out   6065 where  cwd    DIR    8,1     4096  395339 /home/where/workspace
a.out   6065 where  rtd    DIR    8,1     4096       2 /
a.out   6065 where  txt    REG    8,1     8656  429363 /home/where/workspace/a.out
a.out   6065 where  mem    REG    8,1  1864888 1053934 /lib/x86_64-Linux-gnu/libc-2.23.so
a.out   6065 where  mem    REG    8,1   162632 1053906 /lib/x86_64-Linux-gnu/ld-2.23.so
a.out   6065 where    0u   CHR  136,4      0t0       7 /dev/pts/4
a.out   6065 where    1u   CHR  136,4      0t0       7 /dev/pts/4
a.out   6065 where    2u   CHR  136,4      0t0       7 /dev/pts/4
a.out   6065 where    3w   REG    8,1        0  429381 /home/where/workspace/file
```

这边把这个进程所使用的动态库，以及占用的目录，打开的文件都显示出了。

第十八章 其他软件包的安装

18.1 man帮助文档

man系统有自带的libc的接口帮助文件，但是其他的man文档并没有安装，如有需要可以进行安装:

```
sudo apt-get install manpages-de  manpages-de-dev  manpages-dev glibc-doc manpages-posix-dev
manpages-posix
```

18.2 ctags安装

在程序的根目录下运行`ctags -R`，生成tags文件，然后在vi编辑程序时按Ctrl+]就会跳转到当前光标所在东西的定义处。若有多个tag，执行:ts，进行选择。按Ctrl+o即可跳回。不过，当修改过代码后，需要重新生成tags。

```
sudo apt-get install ctags
```