

具体的 shell 命令用法可以通过 help 或 “man 命令”进入手册来查询其具体用法
终端本质上是对应着 Linux 上的 /dev/tty 设备，Linux 的多用户登陆就是通过不同的 /dev/tty 设备完成的，Linux 默认提供了 6 个纯命令行界面的 “terminal” (准确的说这里应该是 6 个 virtual consoles) 来让用户登录，在物理机系统上你可以通过使用 [Ctrl]+[Alt]+[F1]~[F6] 进行切换。当切换到其中一个终端后想要切换回图形界面，可以按下 [Ctrl]+[Alt]+[F7] 来完成。
命令行提示符 \$ 表示普通用户，# 表示 root 用户。

几个技巧：

Tab 键： 补全命令、参数

Ctrl+c： 强行终止当前程序

Ctrl+z： 将当前程序放到后台，拖回前台输入命令 fg

Ctrl+a： 将光标输入行头，相当于 home 键

Ctrl+e： 将光标输入行末，相当于 end 键

在终端切换以 root 账户登录：sudo su 退出输入： exit

通配符

Linux 中常用通配符

字符	含义
*	匹配 0 或多个字符
?	匹配任意一个字符
[list]	匹配 list 中的任意单一字符
[!list]	匹配 除list 中的任意单一字符以外的字符
[c1-c2]	匹配 c1-c2 中的任意单一字符 如：[0-9] [a-z]
{string1,string2,...}	匹配 string1 或 string2 (或更多)其一字符串
{c1..c2}	匹配 c1-c2 中全部字符 如{1..10}

如利用通配符批量创建 5 个文件

```
touch file{1..5}.txt
```

下列笔记是常用的 shell 命令

pwd (printing working directory)

显示当前目录

格式为： pwd [参数]

可选参数为

-P 显示物理地址，为默认值

-L 目录为连接路径时，显示连接路径

wc (word count)

wc 命令用来计算数字。利用 wc 指令我们可以计算文件的 Byte 数、字数或是行数，若不指定文件名称，或是所给予的文件名为“-”，则 wc 指令会从标准输入设备读取数据。

格式为： wc [选项] [文件]

选项有： -c (只显示 Bytes 数)

-l (只显示行数)

-w (只显示字数)

-m (统计字符数)

-L (打印

最长行行数)

如，统计/bin 目录下的命令个数

```
ls /bin | wc -l
```

cd (change directory)

切换当前目录至指定目录

格式为： cd [目录名]

有几个特殊参数

/ 系统根目录

. 当前系统目录

.. 当前目录的父目录
~ 当前用户主目录, 即 /home/maroon 目录
- 上次所在目录

ls (list)

列出对应目录清单

格式为: ls [选项] [目录名]

常用选型有 -a (列出目录下所有文件) -l (除文件名外, 打印包括文件权限、大小等详细信息)

-d (directory, 显示文件夹即可, 不显示其下具体文件) -h (human-readable, 以容易理解的方式列出文件大小, 1k=1048) -t (按时间顺序排列)

目录名缺省为当前目录

如, 列出/home 文件夹下的所有文件和目录的详细资料

```
ls -al /home
```

以容易理解的格式列出/home 目录中所有以"m"开头的文件目录的大小

```
cd /home
```

```
ls -lh m*
```

mkdir (make directory)

创建目录名

格式为: mkdir [选项] [目录名]

选项有 -m(mode, 设置权限) -p(parents, 创建多个目录) -v(verbose, 每次创建新目录都显示信息)

如, 一次创建多个目录, 并且显示具体创建信息

```
mkdir -vp temp/text
```

创建权限为 777 的目录

```
mkdir -m 777 temp
```

touch

touch 命令有两个功能: 一是用于把已存在文件的时间标签更新为系统当前的时间(默认方式), 它们的数据将原封不动地保留下来; 二是用来创建新的空文件。

格式为: touch [选项] [文件]

参数有: -a(只更改存取时间) -c(不创建文件) -d<时间日期>/-t<日期时间>(使用指定的时间)
-m(只更改变动时间)

如, 创建文件 ex2

```
touch ex2
```

rm (remove)

删除目录中的文件或目录, 对于链接文件, 只删除链接

格式为: rm [选项] [文件或目录名]

选项有 -f(force, 忽略不存在的文件, 不给提示) -i(interactive, 交互式删除) -v(verbose, 详细显示步骤) -r(递归)

如, 删除后缀名为.log 的所有文件, 删除前逐一询问

```
rm -i *.log
```

删除/var/log/httpd/access 目录以及其下所有文件、文件夹:

```
rm -rf /var/log/httpd/access
```

mv (move)

移动或更改文件名, 常用来备份文件或目录

格式为: mv [选项] [源文件或目录] [目标文件或目录]

选项有 -b(back, 覆盖前先备份) -f(force, 强行覆盖) -i(interactive, 询问是否覆盖)

-u(update, 目标文件已经存在, 源文件较新时更新) -t(target, 移动多个源文件到一个目录下, 此时目标目录在前, 源文件在后)

如, 将文件 a.txt 移动到 test1 目录下, 如果文件存在, 覆盖前会询问是否覆盖,

```
mv -i a.txt test1 (test1 目录已经存在, 否则会执行重命名)
```

```
sudo mv opencv-3.3.1/ opencv-3.3.1-dev
```

cp (copy)

复制文件或目录

格式为: cp [选项] [源文件] [目录]

选项有 -t(target, 指定目标目录, 此时目标目录在前, 源文件在后) -i -f -u -n (no-clobber, 不要覆盖已存在的文件) -s(symbolic-link, 建立源文件的符号链接, 而非复制文件) -r(复制文件夹)

如，将 test1 目录下的所有文件复制到 test2 目录下，覆盖前询问，可以使用如下命令：

```
cp -i text1/* text2
```

cat (concatenate)

将文件或标准输入组合输出到标准输出，常用来显示文件或连接文件，反向显示文件内容命令为 tac

格式为：cat [选项] [文件]

常用参数有：-A (show-all) -b(number-nonblank, 对非空输出行编号) -n(number, 对所有输出行进行编号) -s(squeeze-blank, 多个空白行转换为一个空白符)

如，把 a.log 的文件内容加上行号后输入 b.log 这个文件里，多行空行换成一行为一行输出

```
cat -ns a.log > b.log
```

nl (number of lines)

计算文件中的行数

格式为：nl [选项] [文件]

选项有 -b a (空行也列出行号，类似于 cat -n) -b t(空行不列行号，默认值) -w(行号栏位的位数)

-n ln(行号在自己栏位的最左方显示) -n rn(行号在自己栏位的最右方显示，不加 0) -n rz(行号在在自己栏位的最右方显示，加 0)

如，把 a.log 的文件内容加上行号后显示，行号在屏幕最右方加 0 显示，行号栏目占位数为 3

```
nl -n rz -w 3 a.log
```

more

功能类似于 cat，cat 将文件内容从上到下显示，more 命令一页页显示，方便阅读，按空格键往下翻，按 b (back)键显示上一页，=键输出当前行号，q 键退出 more。此外还可以搜索字符串

格式为：more [选项] [文件]

参数有：+n(从第 n 行开始显示) -n(定义屏幕大小为 n 行) +/pattern(在文件显示前搜寻字符串 pattern, 从该字符串前两行开始显示)

如，从 a.log 文件中查找第一个出现 "g" 字符串的行，并从该处前两行开始显示输出，规定每屏的行数为 5

```
more -5 +/g a.log
```

less

也是对文件及输出的显示工具，功能极其强大

格式为：less [选项] [文件]

常用参数为 -f(强迫打开) -i(忽略大小写) -N(显示每列行号) -s(显示连续空行为一行)

常用操作：/字符串 向下搜索字符串 ? 字符串 向上搜索字符串

如，显示 a.log 文件中的内容，搜索字符串 "hello"，可以使用如下命令

```
less a.log
```

```
/hello
```

less 与 cat 和 more 的区别：

cat 命令功能用于显示整个文件的内容，单独使用没有翻页功能。因此经常和 more 命令搭配使用，cat 命令还有就是可以将数个文件合并成一个文件的功能。more 命令功能：让画面在显示满一页时暂停，此时可按空格键继续显示下一个画面，或按 q 键停止显示。

less 命令功能：less 命令的用法与 more 命令类似，也可以用来浏览超过一页的文件。所不同的是 less 命令除了可以按空格键向下显示文件外，还可以利用上下键来卷动文件。当要结束浏览时，只要在 less 命令的提示符“:”下按 q 键即可。其实这三个命令除了 cat 命令有合并文件的功能，其余功能上相近，只是从浏览习惯和显示方式上有所不同。

head

显示文件开头，默认为 10 行，对应于 tail 命令（显示文件末尾内容）

格式为：head [选项] [文件]

选项有 -q(隐藏文件名) -v(显示文件名) -c<字节数>(显示字节数) -n<行数>(显示行数，参数为负时显示文件末尾行)

如，显示 a.log 和 b.log 文件中的前 5 行内容，可以使用如下命令：

```
head -n 5 a.log b.log
```

which

在 PATH 变量指定的路径中搜索可执行文件的所在位置，一般用来确认系统中是否正确安装了指定软件

如，确认是否正确安装了 gcc

```
which gcc
```

whereis

定位文件，还可搜索源代码，指定备用搜索路径和搜索不寻常项的能力。whereis 命令查找速度非常快，这是因为它根本不是在磁盘中漫无目的乱找，而是在一个数据库中（/var/lib/mlocate/）查询。这个数据库是 Linux 系统自动创建的，包含有本地所有文件的信息，并且每天通过自动执行 updatedb 命令更新一次。也正是因为这个数据库要每天才更新一次，就会使得 whereis 命令的搜索结果有时候会不准确，比如刚添加的文件可能搜不到

格式为：whereis [选项] [文件]

常用参数：-b (定位可执行文件) -m (定位帮助文件) -s (定位源代码文件)

如，搜索 gcc 帮助文件的路径

```
whereis -m gcc
```

locate

与 whereis 命令类似，且使用相同的数据库。但 whereis 命令只能搜索可执行文件、联机帮助文件和源代码文件，如果要获得更全面的搜索结果，可以使用 locate 命令。

格式：locate [选项] [搜索字符串]

参数 -q (quiet, 不显示出错信息) -n (至多显示 n 个输出) -r (使用正则表达式作搜索条件)

如，搜索 etc 目录下所有以 sh 开头的文件，可以使用如下命令：

```
locate /etc/sh
```

```
locate rosdep
```

find

沿文件层次结构向下遍历，匹配符号条件的文件，并执行相应操作，其功能非常强大

格式为：find [搜索路径] [表达式]

默认路径是当前目录，默认表达式为 -print

常用参数有 -print (输出到标准输出) -delete (删除搜索到的文件) -exec (对匹配的文件执行参数给出的 shell 命令) -name (按文件名查找文件) -type (按类型查找文件, 常用文件类型有 b (块设备文件)、c (字符设备文件)、d (目录)、f (普通文件)、l (符号链接)) -perm (根据文件权限查找文件)

-user (所有者选项) -mtime -n +n (按照文件更改时间查找文件, -n 表示更改时间小于 n 天, +n 表示更改时间大于 n 天) -size +10k/-10k/10k (搜索大于/小于/等于 10k 的文件)

此外还可使用逻辑操作符 -add -or -not () (圆括号字符在 shell 中有特殊含义，所以在命令中使用它们的时候需要引起来，通常使用 \ 转义字符)

如，打印当前目录下所有以 .txt 结尾的符号链接

```
find . -type l -name "*.txt" -print
```

打印当前目录下所有权限为 777 的 php 文件 (web 服务器上的 php 文件一般需要执行权限)

```
find . -type f -name "*.php" -perm 777
```

打印当前目录下 root 用户拥有的所有文件

```
find . -type f -user root
```

打印当前目录下权限不是 777 和 664 的所有文件

```
find . -type f \( ! -perm 777 -and ! -perm 644 \)
```

现在想要把所有 c 语言代码文件下载下来，如果一个一个的下载很麻烦，我们可以先查找到所有的 c 语言代码文件，然后将这些文件内容写入到一个文件中，下载该文件

```
find . -name "*.c" -exec cat {} \; > all.c
```

在这里说明一下 {} 和 \;，{} 其实它就是一个占位符，在 find 命令的执行过程中会不断地替换成当前找到的文件。而 \; 是 -exec 的命令结束标记，因为规定 -exec 后面的命令必须以 ; 结束，但在 shell 中有特殊含义，必须要转义，所以写成 \;。

xargs

xargs 命令可以从标准输入接收输入，并把输入转换为一个特定的参数列表

格式为：command | xargs [选项] [command]

常用参数 -n (指定每行最大的参数数量) -d (指定分隔符)

如，echo "nameXnameXnameXname" | xargs -dX -n2

查找当前目录下所有 c 代码文件，统计总行数，可以使用如下命令

```
find . -type f -name "*.c" | xargs wc -l
```

将 find 产生的输出 (test2 目录下的所有 tes 文件)，作为 rm 的参数，从而完全删除

```
find test2/ -name '*.tes' | xargs rm -rf
```

grep (global search regular expression(RE) and print out the line)

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。

格式为：grep [选项] pattern [file]

常用参数：-c (计算搜索到字符串即 pattern 的次数) -i (忽略大小写) -n (输出行号) -v (反向选

择, 打印不匹配的行) **-r (递归搜索)** **-E**(将范本样式为延伸的普通表示法来使用, 意味着使用扩展正则表达式) **-color=auto**(找到的关键字加颜色显示) **-o**(只打印匹配项, 一个个按列显示)
如, 将/etc/passwd 文件中出现 root 的行取出来, 关键词部分加上颜色显示:
`grep "root" /etc/passwd --color=auto` 或
`cat /etc/passwd | grep "root" --color=auto`
将/etc/passwd 文件中没有出现 root 和 nologin 的行取出来:
`grep -v "root" /etc/passwd | grep -v "nologin"`
在当前目录下递归搜索文件中包含 main()的文件(经常用于查找某些函数位于哪些源代码文件中)
`grep -r "main()" .`

正则表达式

正则表达式是一种符号表示法, 被用来识别文本模式。在某种程度上, 它们与匹配文件和路径名的 shell 通配符比较相似, 但其规模更大。形式和功能上正则表达式和通配符很像, 不过它们之间又有很大差别, 特别在于一些特殊的匹配字符的含义上, 比如*在通配符中表示匹配0或多个字符, 而在正则表达式中为匹配之前字符串的0次或多次。
正则表达式元字符由以下字符组成:

正则表达式	描述	示例
^	行起始标记	^shiyang 匹配以 shiyang 起始的行
\$	行尾标记	\$shiyang 匹配以 shiyang 结尾的行
.	匹配任意一个字符	a.c 匹配 abc、aac, 但不匹配 abbc
[]	匹配包含在[字符]之中的任意一个字符	ab[cd] 匹配 abc 或 abd
[^]	匹配除[字符]之外的任意一个字符	1[^01] 匹配 12、13, 但不匹配 10、11
[-]	匹配[]指定范围内的任意一个字符	[1-5] 匹配 1-5 的任意一个数字
{ n }	匹配之前的项 n 次	[0-9]{2} 匹配任意一个两位数, 相当于[0-9][0-9]
{ n, }	之前的项至少需要匹配 n 次	[0-9]{2,} 匹配任意一个两位或更多位的数字
{ n, m }	指定之前的项需要匹配的最小和最大次数	[0-9]{2, 5} 匹配从两位数到五位数之间的任意一个数字
?	匹配之前的项 1 次或者 0 次	shiy?an 匹配 shiyang 或 shian
*	匹配之前的项 0 次或者多次	shiy*an 匹配 shian、shiyang、shiyang
+	匹配之前的项 1 次或者多次	shiy+an 匹配 shiyang、shiyang
()	创建一个用于匹配的子串	ma(in)? 匹配 ma 或 main
	匹配 两边的任意一项	Dec (1st 2nd) 匹配 Dec 1st 或者 Dec 2nd
\	将上面的特殊字符进行转义	a+b 匹配 a+b

如, 利用 Linux 系统自带的字典查找一个五个字母的单词, 第三个字母为 j, 最后一个字母为 r
`grep '^..j.r$' /usr/share/dict/words`
验证固定电话, 打印符合条件的电话, 区号: 前面一个 0, 后面跟 2-3 位数字: 0[0-9]{2,3}; 电话号码: 7-8 位数字: [0-9]{7,8}; 分机号: 一般都是 3-4 位数字: [0-9]{3,4}
`grep -E "0[0-9]{2,3}-[0-9]{7,8}(-[0-9]{3,4})?$" telephone.txt`
不加 -E 选项时执行命令时没有任何匹配输出, 那上例没加为什么可以呢, 这是因为 上例中使用[基本正则表达式](#)即可正确匹配, 而此例中需要使用[拓展正则表达式](#)。

cut

cut 命令是一个将文本按列进行切分的小工具, 它可以指定分隔每列的定界符。如果一行数据包含多个字段(多列), 现在想要提取其中的一列或多列, 这是 cut 命令就可以大显身手了。
格式为: cut [选项] [文件名]
常用参数有: **-b** (以字节为单位分割) **-c** (以字符为单位分割, -c -5 前五个, -c 5- 前五个之后的, -c

5 第五个, -c 2-5 第二个到第五个) -d(自定义分隔符, 默认为制表符) -f(自定义字段, 如一系列和三列-f 1, 3、第一列-f 1、前三列-f 1-3) -complement (抽取除-c, -f 指定的文本外的整个文本行) 如, 取出 student.txt 文件中除第一列的其他列

```
cut -f 1 -d ' ' student.txt -complement
```

sed

与 vim 不同, sed 属于流编辑器, 即在编辑文件的时候不用把整个文件都读入内存, 可以一行行读入, 操作完成后再读入下一行, 可以占用较小的内存资源。

格式为: sed [选项] [操作] [文件名]

如将 /etc/passwd 的内容列出并且列印行号, 同时, 请将第 2~5 行删除:

与 grep 一样, sed 也支持特殊元字符, 来进行模式查找、替换。不同的是, sed 使用的正则表达式是括在斜杠"/"之间的模式。如果要把正则表达式分隔符"/"改为另一个字符, 比如 o, 只要在这个字符前加一个反斜线, 在字符后跟上正则表达式, 再跟上这个字符即可。例如: sed '\o^Myop' datafile

paste

paste 命令的功能正好与 cut 相反。它会添加一个或多个文本列到文件中, 而不是从文件中抽取文本列。

格式为: paste [选项] [文件名]

常用参数有 -s(将每个文件合并成行, 而不是单独粘贴) -d(自定义分隔符, 默认为制表符)

如, 将 student.txt 和 telephone.txt 文件中的内容按列拼接, 指定分隔符为':'

tr (translate)

tr 命令常被用来更改字符。我们可以把它看作是一种基于字符的查找和替换操作。换字是一种把字符从一个字母转换为另一个字母的过程。tr 从标准输入中替换、缩减和/或删除字符, 并将结果写到标准输出。

格式为: tr [选项] SET1 SET2

常用参数 -d 删除匹配 SET1 的内容, 不作替换

如, 将输入的字符大写转换为小写

将输入的字符中的数字删除

sort

对于文件及标准输入的文本进行从小到大排序

格式为: sort [选项] [文本名]

常用参数为 -n(基于字符串长短来排序) -k(指定排序关键词) -b(排序时忽略每行开头空格) -r(以相反顺序即降序排列) -t(自定义分隔符, 默认为制表符)

如, 列出/usr/bin/目录下使用空间最多的前 10 个目录文件

uniq (unique)

uniq 命令用于报告或忽略文件中的重复行, 只能用于排过序的数据, 常与 sort 命令结合使用。

格式为: uniq [选项] [文件名]

常用参数有 -c(在每行前加上出现次数的标号) -d(只输出重复出现的行) -u(只显示唯一的行)

如, 找出/bin 目录和/usr/bin 目录下所有相同的命令

join

join 命令用来将两个文件中, 制定栏位内容相同的行连接起来。找出两个文件中, 指定栏位内容相同的行, 并加以合并, 再输出到标准输出设备。与 uniq 命令相同, 常能用于排过序的数据。

格式为 join [选项] 文件1 文件2

常用参数 -j FIFLD (等同于 -1 FIFLD -2 FIFLD, 两个文件匹配字段相同)

如, 指定两个文件的第三个字段为匹配字段, 连接两个文件

common

逐行比较文本文件, 显示结果包括 3 列: 第 1 列为只在第一个文件中找到的行, 第 2 列为只在第二个文件中找到的行, 第 3 列为两个文件的共有行。与 uniq, join 相同, 只能用于排过序的数据

格式为 common [选项] 文件1 文件2

参数有 -1 (不能输出文件 1 特有的行) -2 (不能输出文件 2 特有的行) -3 (不输出两个文件共有的)

行)

如, 比较 file1.txt 和 file2.txt 两个文件的内容, 只显示两个文件共有的内容

```
common -12 file1.txt file2.txt
```

diff (differential)

diff 命令在最简单的情况下, 比较给定的两个文件的不同。如果使用“-”代替“文件”参数, 则要比对的内容将来自标准输入。diff 命令是以逐行的方式, 比较文本文件的异同处。如果该命令指定进行目录的比较, 则将会比较该目录中具有相同文件名的文件, 而不会对其子目录文件进行任何比较操作。

格式为: diff [选项] 文件

常用参数有 -c(上下文模式) -u(统一模式) -a(逐行比较) -r(递归比较目录下的文件)

patch

patch 命令被用来把更改应用到文本文件中。它接受从 diff 程序的输出, 并且通常被用来把较老的文件版本转变为较新的文件版本。让我们考虑一个著名的例子。Linux 内核是由一个大型的, 组织松散的贡献者团队开发而成, 这些贡献者会提交固定的少量更改到源码包中。这个 Linux 内核由几百万行代码组成, 虽然每个贡献者每次所做的修改相当少。对于一个贡献者来说, 每做一个修改就给每个开发者发送整个的内核源码树, 这是没有任何意义的。相反, 提交一个 diff 文件。一个 diff 文件包含先前的内核版本与带有贡献者修改的新版本之间的差异。然后一个接受者使用 patch 程序, 把这些更改应用到他自己的源码树中。

使用 diff/patch 组合提供了两个重大优点:

- 一个 diff 文件非常小, 与整个源码树的大小相比较而言。

- 一个 diff 文件简洁地显示了所做的修改, 从而允许程序补丁的审阅者能快速地评估它。

当然, diff/patch 能工作于任何文本文件, 不仅仅是源码文件。它同样适用于配置文件或任意其它文本。

准备一个 diff 文件供 patch 命令使用, GNU 文档建议这样使用 diff 命令:

```
diff -Naur old_file new_file > diff_file
```

old file 和 new file 部分不是单个文件就是包含文件的目录。这个 r 选项支持递归目录树。

格式为: patch [选项] 补丁文件

常用参数有 -p num(忽略几层文件夹) -E(发现空文件时删除) -R(取消打过的补丁)

如, 生成 file1.txt 和 file2.txt 的 diff 文件, 然后应用 patch 命令更新 file1.txt 文件

```
diff -Naur file1.txt file2.txt > patchdiff.txt
```

```
patch < patchdiff.txt
```

取消上边打的补丁

```
patch -R < patchdiff.txt
```

df (disk free)

estimate file space usage, 检查 linux 服务器的文件系统的磁盘空间占用情况

格式为: df [选项] 文件

参数 -a(全部文件系统列表) -h(方便阅读显示) -i(显示 inode 信息) -T(文件系统类型) -t<文件系统类型>(只显示选定文件系统的磁盘信息) -x<文件系统类型>(不显示选定文件系统的磁盘信息)

如, 指定类型磁盘

```
df -t ext4
```

du (disk usage)

estimate file space usage, 与 df 不同, du 是对文件和目录使用空间的查看

格式为: du [选项] 文件

参数为 -a(显示目录中个别文件的大小) -b(显示大小以 byte 为单位) -k(以 KB 为单位) -m(以 MB 为单位) -s(仅显示总计, 列出最后加总的值) -h(以 K, M, G 为单位, 提高可读性)

-c(除显示单独目录大小外, 显示所有目录与文件总和)

如, 显示几个文件或目录各自占用磁盘空间的大小, 并且统计总和

```
du -ch file1.txt file2.txt
```

按照空间大小逆序排序显示

```
du -h | sort -nr | head -10(等同于 head -n 10)
```

time

测量一个命令的运行时间

格式为: time 命令

如, 将命令 date 的运行时间保存到本地文件中

```
{ time date; } 2> 1.txt
```

clear

清屏

常用操作：

用户及文件权限管理

Linux 目录结构及文件基本操作

环境变量与文件查找

文件打包与压缩

用户及文件权限管理

文件系统操作与磁盘管理

Linux 下的帮助命令

Linux 任务计划 crontab

命令执行顺序控制与管道

简单的文本处理

数据流重定向

正则表达式基础

Linux 下软件安装

Linux 进程概念

Linux 进程管理

Linux 日志系统

用户及文件权限管理

一、用户

1. 查看用户：*whoami*

2. 创建用户：`sudo adduser <用户名>`

在 Linux 中，root 权限拥有至高无上的权利，一般登录系统时都是以普通账户的身份登录的，要创建用户需要 root 权限，这里就要用到 `sudo` 这个命令了。不过使用这个命令有两个大前提，一是你要知道当前登录用户的密码，二是当前用户必须在 `sudo` 用户组。

sudo adduser lilei

3. 切换用户：`su <user>` 或

`su -l <user>` 切换用户，环境变量也变为对应的用户的环境变量

如切换到 root 用户：

su root 或 *su* 或 *su -*

`su` 命令和 `su -` 命令区别是：`su` 只是切换了 root 身份，但 Shell 环境仍然是普通用户的 Shell；而 `su -` 连用户和 Shell 环境一起切换成 root 身份了。只有切换了合适的 Shell 环境才不会出现 PATH 环境变量错误，报 command not found 的错误。

ls /home 这个命令不但可以添加用户到系统，同时也会默认为新用户创建 home 目录

su -l lilei

4. 退出用户（与退出终端的操作相同）：*exit* 或 快捷键 `Ctrl+d`

用户组

在 Linux 里面每个用户都有一个归属（用户组），用户组简单地理解就是一组用户的集合，它们共享一些资源和权限，同时拥有私有资源，就跟家的形式差不多，你的兄弟姐妹（不同的用户）属于同一个家（用户组），你们可以共同拥有这个家（共享资源），爸妈对待你们都一样（共享权限），你偶尔写日记，其他人未经允许不能查看（私有资源和权限）。当然一个用户是可以属于多个用户组的，正如你既属于家庭，又属于学校或公司。

查看所属用户组：`groups <user>` 或

`cat /etc/group | sort` 查看所有用户组信息

`cat /etc/group | grep "user"` 查找具体用户信息

`etc/group` 文件格式说明

`/etc/group` 的内容包括用户组（Group）、用户组口令、GID 及该用户组所包含的用户（User），每个用户组一条记录。格式如下：

`group_name:password:GID:user_list`

有时输出的 password 字段为一个 'x' 并不是说密码就是它，只是表示密码不可见而已。

5. 将其它用户加入 sudo 用户组

默认情况下新创建的用户是不具有 root 权限的，也不在 sudo 用户组，可以让其加入 sudo 用户组从而获取 root 权限。

使用 usermod 命令可以为用户添加用户组，同样使用该命令你必需有 root 权限，你可以直接使用 root 用户为其它用户添加用户组，或者用其它已经在 sudo 用户组的用户使用 sudo 命令获取权限来执行该命令

这里用 maroon 用户执行 sudo 命令将 lilei 添加到 sudo 用户组，让它也可以使用 sudo 命令获得 root 权限

```
sudo usermod -G sudo lilei
```

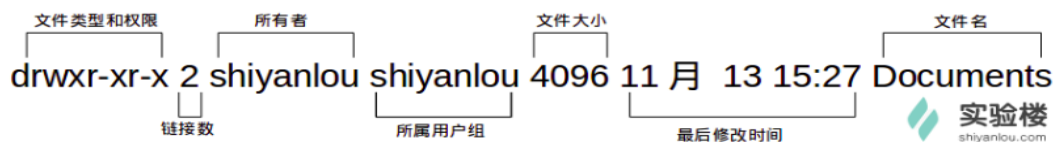
6. 删除用户

```
sudo deluser lilei --remove-home
```

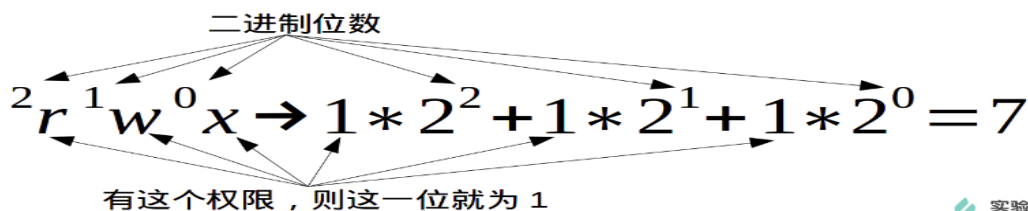
二、文件权限管理

```
ls -l
```

列出的详细信息包括



文件类型和权限具体内容
为



更改文件所有者

```
sudo chown <用户名> <文件名>
```

修改文件权限 chmod 命令

1. 二进制数字表示

```
chmod 700 <文件名>
```

2. 加减赋值操作

`chmod go-rw <文件名>`

'g' 'o' 还有 'u', 分别表示 group, others, user, '+', '-' 就分别表示增加和去掉相应的权限。上面命令使得 group 及 others 不能进行读写操作。

Linux 目录结构及文件基本操作

一、Linux 目录结构

1. FHS (Filesystem Hierarchy Standard, 文件系统层次结构标准)

FHS 定义了两层规范, 第一层是, / 下面的各个目录应该要放什么文件数据, 例如 /etc 应该要放置设置文件, /bin 与 /sbin 则应该要放置可执行文件等等。

第二层则是针对 /usr 及 /var 这两个目录的子目录来定义。例如 /var/log 放置系统登录文件、/usr/share 放置共享数据等等。

附件一详细列出了 / 下各目录存放的数据类型。

FHS 是根据以往无数 Linux 用户和开发者的经验总结出来的, 并且会维持更新, FHS 依据文件系统使用的频繁与否以及是否允许用户随意改动 (不是不能), 将目录定义为四种交互作用的形态, 如下表所示:

	可分享的(shareable)	不可分享的(unshareable)
不可变的(static)	/usr(软件放置处)	/etc(配置文件)
	/opt(第三方软件)	/boot(开机及内核文件)
可变动的(variable)	/var/mail(用户邮件信箱)	/var/run(程序相关)
	/var/news(新闻组)	/var/lock(文件锁相关)

2. Linux 目录路径

如果想进入某个具体的目录或者想获得某个目录的文件 (目录本身也是文件) 那就得用路径来找到了。使用 `cd` 命令可以切换目录, 在 Linux 里面使用 `.` 表示当前目录, `..` 表示上一级目录 (以 `.` 开头的文件都是隐藏文件, 所以这两个目录必然也是隐藏的, 可以使用 `ls -a` 命令查看隐藏文件), `-` 表示上一次所在目录, `~` 通常表示当前用户的 "home" 目录。使用 `pwd` 命令可以获取当前所在路径 (绝对路径)。

绝对路径

关于绝对路径, 简单地说就是以根 "/" 目录为起点的完整路径, 以你所要到的目录为终点, 表现形式如: /usr/local/bin, 表示根目录下的 usr 目录中的 local 目录中的 bin 目录。

相对路径

相对路径, 也就是相对于你当前的目录的路径, 相对路径是以当前目录 `.` 为起点, 以你所要到的目录为终点。表现形式如: usr/local/bin (这里假设你当前目录为根目录)。我们表示相对路径实际并没有加上表示当前目录的那个 `.`, 而是直接以目录名开头, 因为这个 usr 目录为 / 目录下的子目录, 是可以省略这个 `.` 的; 如果是当前目录的上一级目录, 则需要使用 `..`, 而 `../..` 表示上一级目录的上一级目录。如假设当前在 /usr/local/bin 目录, 进入上一级的 local 目录可以使用命令

`cd ..` 或

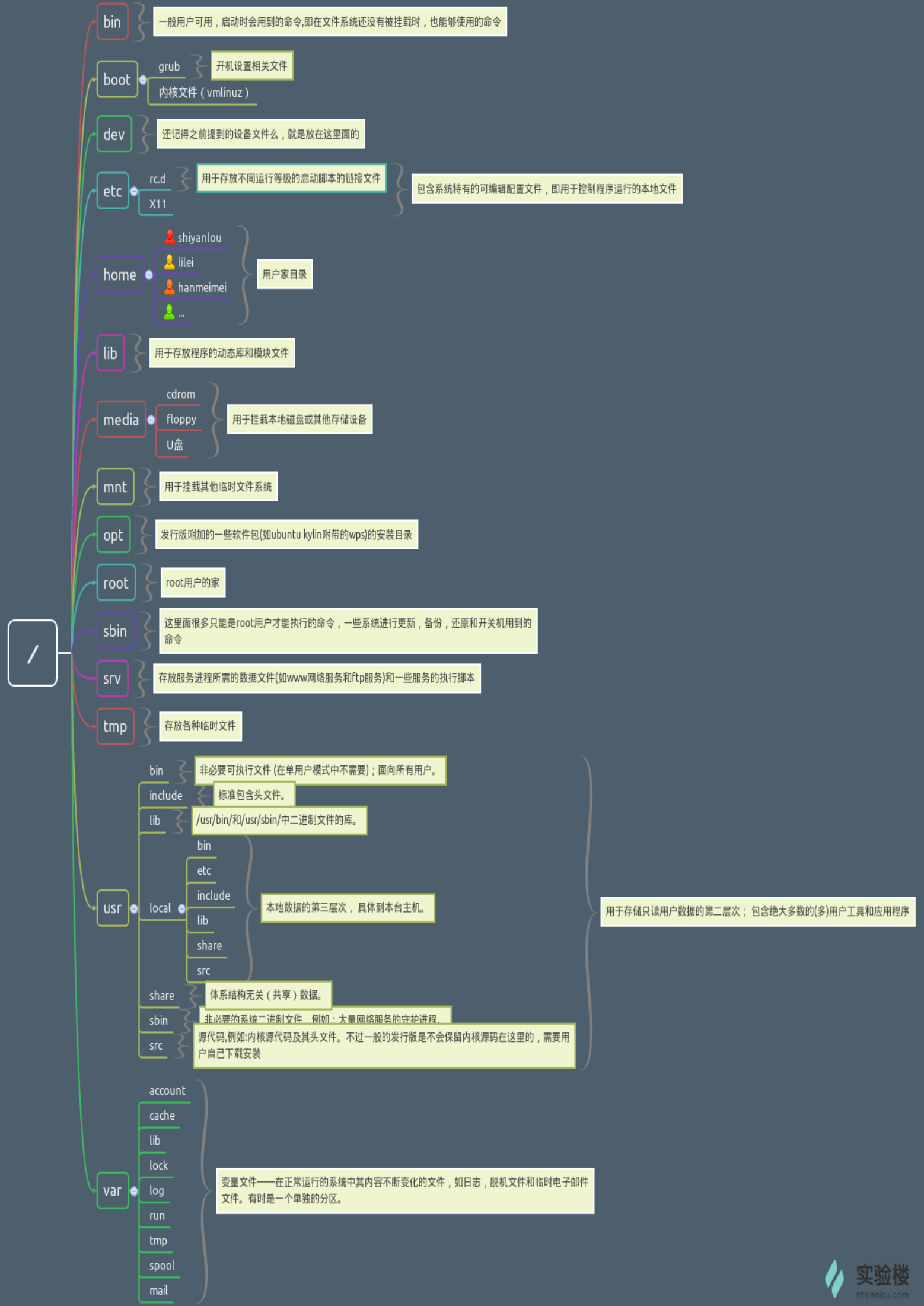
`cd /usr/local` 此时使用相对命令比较简单

如果要进入的是 usr 目录,

`cd /usr` 或

`cd ../../` 此时使用绝对命令比较简单

附件一 (见下页)



二、Linux 文件的基本操作

1. 新建

新建空白文件

使用 touch 命令创建空白文件, 关于 touch 命令, 其主要是来更改已有文件的时间戳的 (比如, 最近访问时间, 最近修改时间), 但其在不加任何参数的情况下, 只指定一个文件名, 则可以创建一个为指定文件名的空白文件 (不会覆盖已有同名文件) 在当前目录。

如在主目录创建 test 空白文件

```
cd ~  
touch test
```

新建目录

使用 mkdir (make directories) 命令可以创建一个空目录, 也可同时指定创建目录的权限属性。使用 -p 参数, 同时创建父目录 (如果不存在该父目录), 这在有时候安装软件, 配置安装路径时非常有用:

```
mkdir -p father/son/grandson
```

2. 复制

复制文件

使用 cp (copy) 命令复制一个文件或目录到指定目录。将之前创建的 "test" 文件复制到 "/home/shiyanlou/father/son/grandson" 目录中:

```
cp test father/son/grandson
```

复制目录

要成功复制目录需要加上 -r 或者 -R 参数, 表示递归复制:

```
mkdir family  
cp -r father family
```

3. 删除

删除文件

使用 rm (remove files or directories) 命令, 删除一个文件或目录, 遇到想要删除一些为只读权限的文件, 直接使用 rm 删除会显示一个提示, 如果想忽略这提示, 直接删除文件, 可以使用 -f 参数强制删除:

```
rm -f test
```

删除目录

跟复制目录一样, 要删除一个目录, 也需要加上 -r 或 -R 参数:

```
rm -r family
```

4. 移动文件与文件重命名

移动文件

使用 mv (move or rename files) 命令, 移动文件 (剪切)。如将文件 "file1" 移到 "Documents" 目录, 格式为 mv 源目录文件 目的目录:

```
mkdir Documents  
mv file1 Documents
```

重命名文件

将文件 "file1" 重命名为 "myfile" mv 旧的文件名 新的文件名:

```
mv file1 myfile
```

5. 查看文件

使用 cat, tac 和 nl 命令查看文件

这两个命令都是用来打印文件内容到标准输出 (终端), 其中 cat 为正序显示, tac 倒序显示。

比如我们要查看 passwd 文件:

```
cat passwd
```

可以加上 -n 参数显示行号:

```
cat -n passwd
```

nl 命令, 添加行号并打印, 这是个比 cat -n 更专业的行号打印命令

```
nl passwd
```

使用这几个命令，默认的终端窗口大小，一屏显示不完文本的内容，得用鼠标拖动滚动条或者滑动滚轮才能继续往下翻页，要是可以直接使用键盘操作翻页就好了，那么就可以使用 more 和 less 命令分页查看文件。

如果说上面的 cat 是用来快速查看一个文件内容的，那么这个 more 和 less 就是天生用来"阅读"一个文件的内容的，比如说"man"手册内部就是使用的 less 来显示内容。其中 more 命令比较简单，只能向一个方向滚动，而"less"为基于"more"和"vi"(一个强大的编辑器)开发，功能更强大。less 的使用基本和 more 一致

使用 more 工具打开 passwd 文件：

```
more passwd
```

打开后默认只显示一屏内容，终端底部显示当前阅读的进度(百分比)。可以使用 Enter 键向下滚动一行，使用 Space 键向下滚动一屏，按下 h 显示帮助，q 退出。

使用 head 和 tail 命令查看文件，它们是只查看的头几行（默认为 10 行，不足 10 行则显示全部）和尾几行，改变查看的行数，加上 -n 参数，后面紧跟行数：

```
tail -n 1 /etc/passwd
```

6. 查看文件类型

在 Linux 下面文件的类型不是根据文件后缀来判断的，通常使用 file 命令可以查看文件的类型：

```
file /bin/ls
```

7. 编辑文件

在 Linux 下面编辑文件通常会直接使用专门的命令行编辑器比如 (emacs, vim, nano) , vim 编辑器有三种常用的模式

普通模式

在普通模式中，用的编辑器命令，比如移动光标，删除文本等等。这也是 Vim 启动后的默认模式。

插入模式

在这个模式中，大多数按键都会向文本缓冲中插入文本

命令行模式

在命令行模式中可以输入会被解释成并执行的文本。

三种常用模式的切换：vim 启动进入普通模式，处于插入模式或命令行模式时只需要按 Esc 进入普通模式，普通模式中按 i（插入）键都可以进入插入模式，普通模式中按 : 进入命令行模式。命令行模式中输入 wq 回车后保存并退出 vim，后边加感叹号 ! 强制保存退出。

环境变量与文件查找

一、环境变量

1. 变量

变量，准确的说应该是 Shell 变量，所谓变量就是计算机中用于记录一个值（不一定是数值，也可以是字符或字符串）的符号，而这些符号将用于不同的运算处理中。通常变量与值是一一对应的关系，可以通过表达式读取它的值赋值给其它变量，也可以直接指定数值赋值给任意变量。为了便于运算和处理，大部分的编程语言会区分变量的类型，用于分别记录数值、字符或者字符串等等数据类型。Shell 中的变量也基本如此，有不同类型（但不用专门指定类型名），可以参与运算，有作用域限定。

可以使用 declare 命令创建一个名为 tmp 的变量：

```
declare tmp
```

使用=进行赋值，

```
tmp=maroon
```

读取变量的值，使用 echo 命令和\$符号（\$符号用于表示引用一个变量的值）

```
echo $tmp
```

2. 环境变量

命令	说明
<code>set</code>	显示当前 Shell 所有环境变量，包括其内建环境变量（与 Shell 外观等相关），用户自定义变量及导出的环境变量
<code>env</code>	显示与当前用户相关的环境变量，还可以让命令在指定环境中运行
<code>export</code>	显示从 Shell 中导出成环境变量的变量，也能通过它将自定义变量导出为环境变量

环境变量就是作用域比自定义变量要大，如 Shell 的环境变量作用于自身和它的子进程。在所有的 UNIX 和类 UNIX 系统中，每个进程都有其各自的环境变量设置。关于环境变量，可以简单的理解成在当前进程的子进程是否有效，有效则为环境变量，否则不是（有些人也将所有变量统称为环境变量，只是以全局环境变量和局部环境变量进行区分，只要理解它们的实质区别即可

3. 命令的查找路径与顺序

在 Shell 中输入一个命令，Shell 是怎么知道在哪去找到这个命令然后执行的呢？这是通过环境变量 PATH 来进行搜索的，熟悉 Windows 的用户可能知道 Windows 中的也是有这么一个 PATH 环境变量。这个 PATH 里面就保存了 Shell 中执行的命令的搜索路径。

查看 PATH 环境变量的内容：

```
echo $PATH
```

通常这一类目录下放的都是可执行文件，当我们在 Shell 中执行一个命令时，系统就会按照 PATH 中设定的路径按照顺序依次到目录中去查找，如果存在同名的命令，则执行先找到的那个。

创建一个 Shell 脚本文件：

```
vim hello_shell.sh
```

在脚本文件中添加下面内容

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

```
for((i=0;i<10;i++)); do
    echo "hello shell"
done
```

```
exit 0
```

为文件添加执行权限：

```
chmod 755 hello_shell.sh
```

执行脚本：

```
./hello_shell.sh
```

创建一个 C 语言"hello world"程序：

```
vim hello_world.c
```

添加 C 文件

```
#include <stdio.h>
```

```
int main(void)
{
    printf("hello world!\n");
    return 0;
}
```

使用 gcc 生成可执行文件：

```
gcc -o hello_world hello_world.c
```

gcc 生成二进制文件默认具有可执行权限，无须修改

在 maroon 目录中创建 mybin 目录，将上述文件移动其中：

```
mkdir mybin
```

```
mv hello_shell.sh hello_world mybin/
```

此时，可在 mybin 目录中运行刚刚创建的程序：

```
cd mybin
```

```
./hello_shell.sh
```

```
./hello_world
```

4. 添加自定义路径到“PATH”环境变量

要在其他目录仍然运行创建的程序，需要添加自定义路径到“PATH”环境变量。PATH 里面的路径是以：

作为分割符，所以这样添加自定义路径：

```
PATH=$PATH:/home/maroon/mybin
```

此时可在其他目录执行创建的程序：

```
cd ~
```

```
hello_shell.sh
```

```
hello_world
```

然而这样还并没有很好的解决问题，因为给 PATH 环境变量追加了一个路径，它也只是在当前 Shell 有效，一旦退出终端，再打开就会发现又失效了。可以每次启动 Shell 时自动执行上面添加自定义路径到 PATH 的命令来解决这个问题。

在每个用户的 home 目录中有一个 Shell 每次启动时会默认执行一个配置脚本，以初始化环境，包括添加一些用户自定义环境变量等等。zsh 的配置文件是 .zshrc，相应 Bash 的配置文件为 .bashrc。它们在 etc 下还都有一个或多个全局的配置文件，不过一般只修改用户目录下的配置文件。

可以简单的使用下面命令直接添加内容到 .bashrc 中：

```
echo "PATH=$PATH:/home/maroon/mybin" >> .bashrc
```

上述命令中>>表示将标准输出以追加的方式重定向到一个文件中，注意前面用到的>是以覆盖的方式重定向到一个文件中，使用的时候一定要注意分辨。在指定文件不存在的情况下都会创建新的文件。

5. 删除已有变量

可以使用 unset 命令删除一个环境变量：

```
unset tmp
```

6. 如何让环境变量立即生效

在上面我们在 Shell 中修改了一个配置脚本文件之后（比如 bash 的配置文件 home 目录下的 .bashrc），每次都要退出终端重新打开甚至重启主机之后其才能生效，很是麻烦，我们可以使用 source 命令来让其立即生效，如：

```
source .bashrc
```

source 命令还有一个别名就是 .，注意与表示当前路径的那个点区分开，虽然形式一样，但作用和使用方式一样，上面的命令如果替换成 . 的方式就该是

```
././bashrc
```

二、搜索文件

与搜索相关的命令常用的有如下几个 whereis, which, find, locate。

whereis 简单快速

这个搜索很快，因为它并没有从硬盘中依次查找，而是直接从数据库中查询。whereis 只能搜索二进制文件 (-b)，man 帮助文件 (-m) 和源代码文件 (-s)。如果想要获得更全面的搜索结果可以使用 locate 命令。

```
whereis -m docker #查看 docker 帮助文件的地址
```

locate 快而全

通过 "/var/lib/mlocate/mlocate.db" 数据库查找，不过这个数据库也不是实时更新的，系统会使用定时任务每天自动执行 updatedb 命令更新一次，所以有时候刚添加的文件，它可能会找不到，需要手动执行一次 updatedb 命令。如果想只统计数目可以加上 -c 参数，-i 参数可以忽略大小写进行查找，whereis 的 -b, -m, -s 同样可以是使用。

```
locate docker #查询和 docker 有关的文件
```

which 小而精

which 本身是 Shell 内建的一个命令，我们通常使用 which 来确定是否安装了某个指定的软件，因为它只从 PATH 环境变量指定的路径中去搜索命令。

```
which docker #查看 docker 是否安装
```

find 精而细

find 应该是这几个命令中最强大的了，它不但可以通过文件类型、文件名进行查找而且可以根据文件的属性（如文件的时间戳，文件的权限等）进行搜索。与时间相关的命令参数：

参数	说明
<code>-atime</code>	最后访问时间
<code>-ctime</code>	创建时间
<code>-mtime</code>	最后修改时间

下面以 `-mtime` 参数举例：

- `-mtime n` : `n` 为数字，表示为在 `n` 天之前的“一天之内”修改过的文件
- `-mtime +n` : 列出在 `n` 天之前（不包含 `n` 天本身）被修改过的文件
- `-mtime -n` : 列出在 `n` 天之内（包含 `n` 天本身）被修改过的文件
- `newer file` : `file` 为一个已存在的文件，列出比 `file` 还要新的文件名

列出 `home` 目录中，当天（24 小时之内）有改动的文件：

```
find ~ -mtime 0
```

列出用户家目录下比 `Code` 文件夹新的文件：

```
find ~ -newer /home/shiyanlou/Code
```

文件打包与压缩

一、文件打包和解压缩

在 Windows 上最常见的不外乎这三种 `*.zip`, `*.rar`, `*.7z` 后缀的压缩文件，而在 Linux 上面常见的除了以上这三种外，还有 `*.gz`, `*.xz`, `*.bz2`, `*.tar`, `*.tar.gz`, `*.tar.xz`, `*.tar.bz2`。

1. zip 压缩打包程序

1.1 打包文件夹

```
zip -r -q -o maroon.zip /home/maroon
```

`-r` 参数表示递归打包包含子目录的全部内容，`-q` 参数表示为安静模式，即不向屏幕输出信息，`-o`，表示输出文件，需在其后紧跟打包输出文件名。

查看压缩文档信息：

```
du -h maroon.zip
file maroon.zip
```

1.2 设置压缩等级（9 最小，1 最大）

```
zip -r -9 -q -o maroon_9.zip /home/maroon -x ~/.zip
```

```
zip -r -1 -q -o maroon_1.zip /home/maroon -x ~/.zip
```

这里添加了一个参数用于设置压缩级别 - [1-9]，1 表示最快压缩但体积大，9 表示体积最小但耗时最久。最后那个 `-x` 是为了排除我们上一次创建的 `zip` 文件，否则又会被打包进这一次的压缩文件中。这里只能使用绝对路径。

再用 `du` 命令分别查看默认压缩级别、最低、最高压缩级别及未压缩的文件的大小：

```
du -h -d 0 *.zip ~ | sort
```

通过 `man` 手册可知：

`h`, `--human-readable`（顾名思义，你可以试试不加的情况）

`d`, `--max-depth`（所查看文件的深度）

1.3 创建加密 zip 包

使用 `-e` 参数可以创建加密压缩包：

```
zip -r -e -o maroon_encryption.zip /home/maroon
```

注意：关于 `zip` 命令，因为 Windows 系统与 Linux/Unix 在文本文件格式上的一些兼容问题，比如换行符（为不可见字符），在 Windows 为 `CR+LF`（Carriage-Return+Line-Feed：回车加换行），而在 Linux/Unix 上为 `LF`（换行），所以如果在不加处理的情况下，在 Linux 上编辑的文本，在 Windows 系统上打开可能看起来是没有换行的。如果想让在 Linux 创建的 `zip` 压缩文件在 Windows 上解压后没有任何问题，那么还需要对命令做一些修改：

```
zip -r -l -o maroon.zip /home/maroon
```

需要加上 -l 参数将 LF 转换为 CR+LF 来达到以上目的。

2. 使用 unzip 命令解压缩 zip 文件

将 maroon.zip 解压到前目录：

```
unzip maroon.zip
```

使用安静模式，将文件解压到指定目录，指定目录不存在会自动创建：

```
unzip -q maroon.zip -d ziptest
```

如果不想解压只想查看压缩包的内容可以使用 -l 参数：

```
unzip -l maroon.zip
```

注意：使用 unzip 解压文件时同样应该注意兼容问题，不过这里此时关心的不再是上面的问题，而是中文编码的问题，通常 Windows 系统上面创建的压缩文件，如果有包含中文的文档或以中文作为文件名的文件时默认会采用 GBK 或其它编码，而 Linux 上面默认使用的是 UTF-8 编码，如果不加任何处理，直接解压的话可能会出现中文乱码的问题（有时候它会自动帮你处理），为了解决这个问题，可以在解压时指定编码类型。

使用 -O（英文字母，大写 o）参数指定编码类型：

```
unzip -O GBK 中文压缩文件.zip
```

3. rar 打包压缩命令

rar 也是 Windows 上常用的一种压缩文件格式，在 Linux 上可以使用 rar 和 unrar 工具分别创建和解压 rar 压缩包。

首次使用需先安装 rar 和 unrar 工具：

```
sudo apt-get update
```

```
sudo apt-get install rar unrar
```

在使用 rar、unrar 命令时应注意命令参数前不加 -，否则会报错

```
rm *.zip
```

```
rar a maroon.rar .
```

上面的命令使用 a 参数添加一个目录~到一个归档文件中，如果该文件不存在就会自动创建。

同样，如果不解压只是查看文件，可以使用参数 l：

```
rar l maroon.rar
```

全路径解压：

```
unrar x maroon.rar
```

去掉路径解压：

```
mkdir tmp
```

```
unrar e maroon.rar tmp/
```

4. tar 打包工具

在 Linux 上面更常用的是 tar 工具，tar 原本只是一个打包工具，只是同时还是实现了对 7z, gzip, xz, bzip2 等工具的支持。

创建一个 tar 包：

```
tar -cf maroon.zip ~
```

上面命令中，-c 表示创建一个 tar 包文件，-f 用于指定创建的文件名，注意文件名必须紧跟在 -f 参数之后，比如不能写成 tar -fc maroon.tar，可以写成 tar -f maroon.tar -c ~。还可以加上 -v 参数以可视的方式输出打包的文件。上面会自动去掉表示绝对路径的 /，也可以使用 -P 保留绝对路径符。

解包一个文件（-x 参数）到指定路径的已存在目录（-C 参数）：

```
mkdir tardir
```

```
tar -xf maroon.tar -C tardir
```

只查看不解包文件 -t 参数：

```
tar -tf maroon.tar
```

对于创建不同的压缩格式的文件，对于 tar 来说是相当简单的，需要的只是换一个参数，这里以使用 gzip 工具创建 *.tar.gz 文件为例来说明。只需要在创建 tar 文件的基础上添加 -z 参数，使用 gzip 来压缩文件：

```
tar -czf maroon.tar.gz ~
```

解压 *.tar.gz 文件到当前文件夹：

```
tar -xzf maroon.tar.gz
```

现在要使用其他的压缩工具创建或解压相应文件只需要更改一个参数即可：

压缩文件格式	参数
*.tar.gz	-z
*.tar.xz	-J
*tar.bz2	-j

此外还有 gzip ↔ gunzip (相当于 gzip -d) 压缩和解压命令, 解压文件为 .gz 后缀。

文件系统操作与磁盘管理

一、简单文件系统操作

查看磁盘和目录的容量

使用 df 命令查看磁盘的容量, 加上 -h 参数, 以更易读的方式展示:

```
df -h
```

使用 du 命令查看目录的容量, -d 参数指定查看目录的深度:

```
du -h -d 0 ~ # 只查看 1 级目录的信息
```

二、简单的磁盘管理

1. 创建虚拟磁盘

dd 命令简介

dd 命令用于转换和复制文件, 不过它的复制不同于 cp。在 Linux 系统中, 一切即文件, 硬件的设备驱动 (如硬盘) 和特殊设备文件 (如 /dev/zero 和 /dev/random) 都像普通文件一样, 只要在各自的驱动程序中实现了对应的功能, dd 也可以读取自和/或写入到这些文件。这样, dd 也可以用在备份硬件的引导扇区、获取一定数量的随机数据或者空数据等任务中。dd 程序也可以在复制时处理数据, 例如转换字节序、或在 ASCII 与 EBCDIC 编码间互换。

dd 的命令行语句与其他的 Linux 程序不同, 因为它的命令行选项格式为选项=值, 而不是更标准的 --选项 值或 -选项=值。dd 默认从标准输入中读取, 并写入到标准输出中, 但可以用选项 if (input file, 输入文件) 和 of (output file, 输出文件) 改变。

```
dd of=test bs=10 count=1 或 dd if=/dev/stdin of=test bs=10 count=1
```

上述命令从标准输入设备读入用户输入 (缺省值, 所以可省略) 然后输出到 test 文件, bs (block size) 用于指定块大小 (缺省单位为 Byte, 也可为其指定如 'K', 'M', 'G' 等单位), count 用于指定块数量。

接着在标准输入中写入:

```
hello,Maroon!
```

指定只读取总共 10 个字节的数据, 当我输入了 "hello,Maroon!" 之后总共 13 个字节 (一个英文字符占一个字节) 内容, 显然超过了设定大小。使用和 du 和 cat 命令看到的写入完成文件实际内容确实只有 10 个字节 (那个黑底百分号表示这里没有换行符), 而其他的多余输入将被截取并保留在标准输入。

```
du -b test
```

```
cat test
```

dd 在拷贝的同时还可以实现数据转换, 那下面就举一个简单的例子: 将输出的英文字符转换为大写再写入文件:

```
dd if=/dev/stdin of=test bs=10 count=1 conv=ucase
```

更多转换参数可以通过 man 文档查看。

1.1 使用 dd 命令创建虚拟镜像文件:

```
dd if=/dev/zero of=virtual.img bs=1M count=256
```

```
du -h virtual.img
```

然后将这个文件格式化 (写入文件系统)

1.2 使用 mkfs 命令来格式化磁盘

可以简单的使用下面的命令来将创建的虚拟磁盘镜像格式化为 ext4 文件系统:

```
sudo mkfs.ext4 virtual.img
```

1.3 使用 mount 命令挂载磁盘到目录树

用户在 Linux/UNIX 的机器上打开一个文件以前, 包含该文件的文件系统必须先进行挂载的动作, 此

时用户要对该文件系统执行 `mount` 的指令以进行挂载。通常是使用在 USB 或其他可移除存储设备上，而根目录则需要始终保持挂载的状态。

使用 `mount` 来查看下主机已经挂载的文件系统：

```
sudo mount
```

`mount` 命令的一般格式如下：

```
mount [options] [source] [directory]
```

一些常用操作：

```
mount [-o [操作选项]] [-t 文件系统类型] [-w|--rw|--ro] [文件系统源] [挂载点]
```

挂载我们创建的虚拟磁盘镜像到 `/mnt` 目录：

```
mount -o loop --ro virtual.img /mnt 或 mount -o loop,ro virtual.img /mnt
```

在类 UNIX 系统中，`/dev/loop` 是一种伪设备，这种设备使得文件可以如同块设备一般被访问。

使用 `umount` 命令可以卸载已挂载磁盘：

```
sudo umount /mnt
```

1.4 使用 `fdisk` 为磁盘分区

查看硬盘分区表信息：

```
sudo fdisk -l
```

进入磁盘分区模式：

```
sudo fdisk virtual.img
```

然后根据自己规划好的分区方案利用提示信息分区，比如使用 128M（可用 127M 左右）的虚拟磁盘镜像创建一个 30M 的主分区剩余部分为扩展分区包含 2 个大约 45M 的逻辑分区。

操作完成后输入 `p` 查看结果，完成后输入 `w` 写入分区表。

1.5 使用 `losetup` 命令建立镜像与回环设备的关联

```
sudo losetup /dev/loop0 virtual.img
```

如果提示设备忙也可以使用其它的回环设备，"`ls /dev/loop*`"参看所有回环设备

解除设备关联：

```
sudo losetup -d /dev/loop0
```

1.6 使用 `mkfs` 格式化各分区

（前面是格式化整个虚拟磁盘镜像文件或磁盘），不过格式化之前，还要为各分区建立虚拟设备的映射，用到 `kpartx` 工具，需要先安装：

```
sudo apt-get install kpartx
```

```
sudo kpartx -av /dev/loop0
```

取消映射：

```
sudo kpartx -dv /dev/loop0
```

接着再是格式化，我们将其全部格式化为 `ext4`：

```
sudo mkfs.ext4 -q /dev/mapper/loop0p1
```

```
sudo mkfs.ext4 -q /dev/mapper/loop0p5
```

```
sudo mkfs.ext4 -q /dev/mapper/loop0p6
```

格式化完成后在 `/media` 目录下新建四个空目录用于挂载虚拟磁盘：

```
mkdir -p /media/virtualdisk_{1..3}
```

1.7 挂载磁盘分区

```
sudo mount /dev/mapper/loop0p1 /media/virtualdisk_1
```

```
sudo mount /dev/mapper/loop0p5 /media/virtualdisk_2
```

```
sudo mount /dev/mapper/loop0p6 /media/virtualdisk_3
```

上边过程完成后使用

```
df -h
```

能够找到上边创建磁盘分区。

Linux 下的帮助命令

一、内建命令与外部命令

内建命令实际上是 `shell` 程序的一部分，其中包含的是一些比较简单的 Linux 系统命令，这些命令是写在 `bash` 源码的 `builtins` 里面的，并由 `shell` 程序识别并在 `shell` 程序内部完成运行，通常在 Linux 系统加载运行时 `shell` 就被加载并驻留在系统内存中。而且解析内部命令 `shell` 不需要创建子进程，因此其执行速度比外部命令快。比如：`history`、`cd`、`exit` 等等。

外部命令是 Linux 系统中的实用程序部分，因为实用程序的功能通常都比较强大，所以其包含的程序量也会很大，在系统加载时并不随系统一起被加载到内存中，而是在需要时才将其调用内存。虽然其不包含在 `shell` 中，但是其命令执行过程是由 `shell` 程序控制的。外部命令是在 `Bash` 之外额外安装的，通常放在 `/bin`，`/usr/bin`，`/sbin`，`/usr/sbin` 等等。比如：`ls`、`vi` 等。

简单来说就是一个是天生自带的天赋技能，一个是后天得来附加技能。可以使用 `type` 命令来区分命令是内建的还是外部的。

```
type exit
type service
type ls
```

二、帮助命令的使用

1. help 命令

`help` 命令是用于显示 shell 内建命令的简要帮助信息。对于外部命令的话基本上都有一个参数 `-help`，这样就可以得到相应的帮助。即

内建命令： `help [命令]`

外部命令： `[命令] -help`

如，

```
help exit
ls -help
```

2. man 命令

`man` 没有内建与外部命令的区分，因为 `man` 工具是显示系统手册页中的内容，也就是一本电子版的字典，这些内容大多数都是对命令的解释信息。

```
man exit
man ls
```

3. info 命令

如果 `man` 显示的信息都还不够，可以使用 `info` 命令得到十分全面的信息。`info` 来自自由软件基金会的 GNU 项目，是 GNU 的超文本帮助系统，能够更完整的显示出 GNU 信息。所以得到的信息当然更多

```
info ls
```

简单比较这三个指令帮助命令，`help` 简答扼要，`man` 详细描述，`man` 则全面详尽。

Linux 任务计划 crontab

一、crontab 的使用

1. crontab 简介

`crontab` 命令常见于 Unix 和类 Unix 的操作系统之中，用于设置周期性被执行的指令。该命令从输入设备读取指令，并将其存放于 `crontab` 文件中，以供之后读取和执行。通常，`crontab` 储存的指令被守护进程激活，`crond` 为其守护进程，`crond` 常常在后台运行，每一分钟会检查一次是否有预定的作业需要执行。

通过 `crontab` 命令，可以在固定的间隔时间执行指定的系统指令或 shell script 脚本。时间间隔的单位可以是分钟、小时、日、月、周的任意组合。

2. crontab 使用

通过下面一个命令来添加一个计划任务：

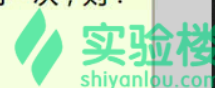
```
crontab -e
```

第一次启动会出现提示界面，需要选择编辑的工具，输入 `/usr/bin/vim/basic` 对应的数字，来选择比较熟悉的 `vim` 编辑器。出现的 `vim` 编辑器有 `crontab` 文档编辑的介绍，具体的格式与参数可见下图：

代表意义	分钟	小时	日期	月份	周	指令
数字范围	0-59	0-23	1-31	1-12	0-7	呀就指令啊

比较有趣的是那个『周』喔！周的数字为 0 或 7 时，都代表『星期天』的意思！另外，还有一些辅助的字符，大概有底下这些：

特殊字符	代表意义
*(星号)	代表任何时刻都接受的意思！举例来说，范例一内那个日、月、周都是 *，就代表着『不论何月、何日的礼拜几的 12:00 都执行后续指令』的意思！
,(逗号)	代表分隔时段的意思。举例来说，如果要下达的工作是 3:00 与 6:00 时，就会是： 0 3,6 * * * command 时间参数还是有五栏，不过第二栏是 3,6，代表 3 与 6 都适用！
-(减号)	代表一段时间范围内，举例来说，8 点到 12 点之间的每小时的 20 分都进行一项工作： 20 8-12 * * * command 仔细看到第二栏变成 8-12 喔！代表 8,9,10,11,12 都适用的意思！
/n(斜线)	那个 n 代表数字，亦即是『每隔 n 单位间隔』的意思，例如每五分钟进行一次，则： */5 * * * * command 很简单吧！用 * 与 /5 来搭配，也可以写成 0-59/5，相同意思！



点击 i 键，选择编辑器的插入模式，在最后一排输入下面命令，该任务是每分钟会在 /home/maroon 目录下创建一个以当前的年月日时分秒为名字的空白文件：

```
*/1 * * * * touch /home/maroon/${date +%Y%m%d%H%M%S}
```

注意 “ % ” 在 crontab 文件中，有结束命令行、换行、重定向的作用，前面加 “ \ ” 符号转意，否则，“ % ” 符号将执行其结束命令行或者换行的作用，并且其后的内容会被做为标准输入发送给前面的命令。

然后按 esc 键退出插入模式，按：进入命令行模式，输入 wq 回车，退出 vim 编辑器。此时可以通过 `crontab -l`

查看添加的任务。

3. 启动 cron 的守护进程

虽然添加了任务，但是如果 cron 的守护进程并没有启动，它根本都不会监测到有任务，当然也就不会帮我们执行，可以通过一下 2 种方式来确定我们的 cron 是否成功的在后台启动

```
ps aux | grep cron 或 pgrep cron
```

如果没有则需要手动启动，输入下列命令

```
sudo cron -f &
```

后台成功启动后，然后通过 `ls -l` 命令查看当前目录的文件，可以看到任务在创建之后便创建了一个当时时间的文件，后续在每分钟的 01 秒时执行一次我们的任务。

可以通过这样一个命令查看到执行任务命令之后在日志中的信息反馈：

```
sudo tail -f /var/log/syslog
```

当不需要这个任务的时候可以使用下面的命令去删除任务：

```
crontab -r
```

删除之后再查看任务列表，系统已经显示该用户并没有任务：

```
crontab -l
```

二、crontab 的深入

这个 `crontab -e` 是针对使用者的 cron 来设计的，也就是每个用户在添加任务，就会在 /var/spool/cron/crontabs 中添加一个该用户自己的任务文档，这样可以做到隔离，独立，不会

混乱。而对于系统的例行性任务，只要编辑 `/etc/crontab` 档案就可以了。

基本上，`cron` 这个服务的最低侦测限制是分钟，所以 `cron` 会每分钟去读取一次 `/etc/crontab` 与 `/var/spool/cron/crontabs` 里面的资料内容，因此，只要编辑完 `/etc/crontab` 这个文档，并且将其存储之后，那么 `cron` 的设定就自动的执行了。

输入下列命令：

```
ll /etc/ | grep cron
```

可以观察到关于 `cron` 的文件：

`/etc/cron.daily`，目录下的脚本会每天让执行一次，在每天的 6 点 25 分时运行；

`/etc/cron.hourly`，目录下的脚本会每小时让执行一次，在每小时的 17 分钟时运行；

`/etc/cron.monthly`，目录下的脚本会每月让执行一次，在每月 1 号的 6 点 52 分时运行；

`/etc/cron.weekly`，目录下的脚本会每周让执行一次，在每周第七天的 6 点 47 分时运行；

当然，以上的时间均是系统默认时间，可以根据自己的需求进行修改。

更多的相关知识可以用 `man` 命令来查看。

命令执行顺序控制与管道

一、命令执行顺序的控制

1. 顺序执行多条命令

通常情况下，每次只能在终端输入一条命令，按下回车执行，执行完成后，再输入第二条命令，然后再按回车执行。要想一次性输入完执行，可以使用简单的顺序执行 `' ; '` 来完成。如下

```
sudo apt-get update
sudo apt-get install cowsay
cowsay hello Maroon
```

可以一次性输入

```
sudo apt-get update; sudo apt-get install cowsay; cowsay hello Maroon
```

2. 有选择的执行命令

在顺序执行任务时，如果前面的命令执行不成功，而后面的命令又依赖与上一条命令的结果，那么就会造成花了时间，最终却得到一个错误的结果。那么需要能够有选择性的来执行命令，比如上一条命令执行成功才继续下一条，或者不成功又该做出其它什么处理，比如可以使用 `which` 来查找是否安装某个命令，如果找到就执行该命令，否则什么也不做

```
which cowsay>/dev/null && cowsay -f head-in ohch~
```

在 C 语言里面 `&&` 表示逻辑与，而且还有一个 `||` 表示逻辑或，同样 Shell 也有这两个符号，它们的区别就在于，shell 中的这两个符号除了也可用于表示逻辑与和或之外，就是可以实现命令执行顺序的简单控制。`&&` 表示如果前面的命令执行结果返回 0 则执行后面的，否则不执行，`||` 在这里就是与 `&&` 相反的控制效果，当上一条命令执行结果为 $\neq 0$ ($\$?\neq 0$) 时则执行它后面的命令。

可以从 `$?` 环境变量获取上一次命令的返回结果：

```
which cowsay
echo $? //cowsay 未安装，结果为 1
which cat
echo $? //结果为 0
```

除了基本的使用之外，还可以结合这 `&&` 和 `||` 来实现一些操作，比如：

```
which cowsay>/dev/null && echo "exist" || echo "not exist"
```

二、管道

管道是什么，管道是一种通信机制，通常用于进程间的通信（也可通过 `socket` 进行网络通信），它表现出来的形式就是将前面每一个进程的输出 (`stdout`) 直接作为下一个进程的输入 (`stdin`)。

管道又分为匿名管道和具名管道，在使用一些过滤程序时经常会用到的就是匿名管道，在命令行中由 `|` 分隔符表示。

1. 试用

先试用一下管道，比如查看 `/etc` 目录下有哪些文件和目录，使用 `ls` 命令来查看：

```
ls -al /etc
```

有太多内容，屏幕不能完全显示，这时候可以使用滚动条或快捷键滚动窗口来查看。不过这时候可以使用管道：

```
ls -al /etc | less
```

通过管道将前一个命令 (`ls`) 的输出作为下一个命令 (`less`) 的输入，然后就可以一行一行地看。

2.cut 命令,打印每一行的某一字段

打印/etc/passwd 文件中以:为分隔符的第 1 个字段和第 6 个字段分别表示用户名和其家目录:

```
cut /etc/passwd -d ':' -f 1,6
```

3.grep 命令,在文本中或 stdin 中查找匹配字符串

grep 命令是很强大的,也是相当常用的一个命令,它结合正则表达式可以实现很复杂却很高效的匹配和查找。

搜索/home/maroon 目录下所有包含"maroon"的所有文本文件,并显示出现在文本中的行号:

```
grep -rnl "maroon" ~
```

-r 参数表示递归搜索子目录中的文件,-n 表示打印匹配项行号,-l 表示忽略二进制文件。

也可以在匹配字段中使用正则表达式,查看环境变量中以"roon"结尾的字符串

```
export | grep ".*roon$"
```

其中\$就表示一行的末尾。

4. wc 命令,简单小巧的计数工具

wc 命令用于统计并输出一个文件中行、单词和字节的数目,参数可通过 wc -help 查看。输出/etc/passwd 文件的统计信息:

```
wc /etc/passwd
```

再来结合管道来操作一下,下面统计 /etc 下面所有目录数:

```
ls -dl /etc/* | wc -l
```

5.sort 排序命令

sort 功能很简单就是将输入按照一定方式排序,然后再输出,它支持的排序有按字典排序,数字排序,按月份排序,随机排序,反转排序,指定特定字段进行排序等等。

默认为字典排序:

```
cat /etc/passwd | sort
```

按特定字段排序:

```
cat /etc/passwd | sort -t':' -k 3 -n
```

上面的-t 参数用于指定字段的分隔符,这里是以":"作为分隔符;-k 字段号用于指定对哪一个字段进行排序。这里/etc/passwd 文件的第三个字段为数字,默认情况下是一字典序排序的,如果要按照数字排序就要加上-n 参数。

6. uniq 去重命令

uniq 命令可以用于过滤或者输出重复行。

使用 history 命令查看最近执行过的命令(实际为读取\${SHELL}_history 文件,如环境中的 ~/.zsh_history 文件):

```
history | cut -c 8- | cut -d ' ' -f 1 | uniq
```

然后经过层层过滤,可以发现的确只输出了执行的命令那一列,不过去重效果好像不明显,之所以不明显是因为 uniq 命令只能去连续重复的行,不是全文去重,所以要达到预期效果,需要先排序:

```
history | cut -c 8- | cut -d ' ' -f 1 | sort | uniq
```

输出重复行

输出重复过的行(重复的只输出一个)及重复次数

```
history | cut -c 8- | cut -d ' ' -f 1 | sort | uniq -dc
```

简单的文本处理

文本处理命令

1.tr 命令

tr 命令可以用来删除一段文本信息中的某些文字。或者将其进行转换。

```
tr [option]...SET1 [SET2]
```

常用的选项有:

选项 说明

-d 删除和 set1 匹配的字符

-s 去除 set1 指定的在输入文本中连续并重复的字符

如,删除 "hello maroon" 中所有的'o','l','h'

```
echo 'hello maroon' | tr -d 'olh'
```


将"hello" 中的ll,去重为一个l
`echo 'hello' | tr -s 'l'`
将输入文本，全部转换为大写或小写输出
`cat /etc/passwd | tr '[:lower:]' '[:upper:]'`
上面的'[:lower:]' '[:upper:]'有时也可以简单的写作'[a-z]' '[A-Z]'
关于 Linux 中的特殊符号，可见下表及说明

特殊符号	说明
<code>[:alnum:]</code>	代表英文大小写字节及数字，亦即 0-9, A-Z, a-z
<code>[:alpha:]</code>	代表任何英文大小写字节，亦即 A-Z, a-z
<code>[:blank:]</code>	代表空白键与 [Tab] 按键两者
<code>[:cntrl:]</code>	代表键盘上面的控制按键，亦即包括 CR, LF, Tab, Del.. 等等
<code>[:digit:]</code>	代表数字而已，亦即 0-9
<code>[:graph:]</code>	除了空白字节 (空白键与 [Tab] 按键) 外的其他所有按键
<code>[:lower:]</code>	代表小写字节，亦即 a-z
<code>[:print:]</code>	代表任何可以被列印出来的字节
<code>[:punct:]</code>	代表标点符号 (punctuation symbol), 亦即: "'?!;:#\$%&...'等
<code>[:upper:]</code>	代表大写字节，亦即 A-Z
<code>[:space:]</code>	任何会产生空白的字节，包括空白键, [Tab], CR 等等
<code>[:xdigit:]</code>	代表 16 进位的数字类型，因此包括： 0-9, A-F, a-f 的数字与字节

注意：之所以要使用特殊符号，是因为上面的[a-z]不是在所有情况下都管用，这还与主机当前的语系有关，即设置在 LANG 环境变量的值，zh_CN.UTF-8的话[a-z]，即为所有小写字母，其它语系可能是大小写交替的如，"a A b B...z Z"，[a-z]中就可能包含大写字母。所以在使用[a-z]时请确保当前语系的影响，使用[:lower:]则不会有这个问题。

2.col 命令
col 命令可以将 Tab 换成对等数量的空格键，或反转这个操作。
`col [option]`

常用的选项有：

选项 说明
-x 将 Tab 转换为空格
-h 将空格转换为 Tab (默认选项)

如，查看 /etc/protocols 中的不可见字符，可以看到很多 ^I，这其实就是 Tab 转义成可见字符的符号

`cat -A /etc/protocols`

使用 col -x 将 /etc/protocols 中的 Tab 转换为空格, 然后再使用 cat 查看，可以发现 ^I 不见了

`cat /etc/protocols | col -x | cat -A`

3.join 命令
这个命令用于将两个文件中包含相同内容的那一行合并在一起。
`join [option]... file1 file2`

常用的选项有：

选项 说明
-t 指定分隔符，默认为空格
-i 忽略大小写的差异
-1 指明第一个文件要用哪个字段来对比，，默认对比第一个字段
-2 指明第二个文件要用哪个字段来对比，，默认对比第一个字段

如，将/etc/passwd与/etc/group两个文件合并，指定以':'作为分隔符，分别比对第4和第3个

字段

```
sudo join -t':' -1 4 /etc/passwd -2 3 /etc/group
```

4. paste 命令

paste 命令与 join 命令类似，它是在不对比数据的情况下，简单地将多个文件合并一起，以 Tab 隔开。

paste [option] file...

常用的选项有：

选项 说明

-d 指定合并的分隔符，默认为 Tab

-s 不合并到一行，每个文件为一行

如，

```
echo hello > file1
```

```
echo maroon > file2
```

```
paste -d ':' file1 file2
```

```
paste -s file1 file2
```

数据流重定向

之前多次见过的>或>>操作，他们分别是将标准输出导向一个文件或追加到一个文件中。这其实就是重定向，将原本输出到标准输出的数据重定向到一个文件中，因为标准输出(/dev/stdout)本身也是一个文件，我们将命令输出导向另一个文件自然也是没有任何问题的。

一、数据流重定向

简单的回顾一下经常用到的两个重定向操作：

```
echo 'hello maroon' > redirect
```

```
echo 'hello Jing' >> redirect
```

```
cat redirect
```

当然前面没有用到的<和<<操作也是没有问题的，它们的区别在于重定向的方向不一致而已，>表示是从左到右，<右到左。

1. 简单的重定向

Linux 默认提供了三个特殊设备，用于终端的显示和输出，分别为 stdin（标准输入，对应于在终端的输入），stdout（标准输出，对应于终端的输出），stderr（标准错误输出，对应于终端的输出）。

文件描述符 设备文件 说明

0 /dev/stdin 标准输入

1 /dev/stdout 标准输出

2 /dev/stderr 标准错误

文件描述符：文件描述符在形式上是一个非负整数。实际上，它是一个索引值，指向内核为每一个进程所维护的该进程打开文件的记录表。当程序打开一个现有文件或者创建一个新文件时，内核向进程返回一个文件描述符。在程序设计中，一些涉及底层的程序编写往往会围绕着文件描述符展开。但是文件描述符这一概念往往只适用于 UNIX、Linux 这样的操作系统。

将 cat 的连续输出（here doc 方式，输入 EOF 结束）重定向到一个文件

```
mkdir Documents
```

```
cat > Documents/test.c <<EOF
```

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    printf("hello world\n");
```

```
    return 0;
```

```
}
```

```
EOF
```

将一个文件作为命令的输入，标准输出作为命令的输出

```
cat Documents/test.c
```

将 echo 命令通过管道传过来的数据作为 cat 命令的输入，将标准输出作为命令的输出

```
echo 'hi' | cat
```

将 echo 命令的输出从默认的标准输出重定向到一个普通文件

```
echo 'hello maroon' > redirect
```

```
cat redirect
```

2. 标准错误重定向

重定向标准输出到文件，这是一个很实用的操作，另一个很实用的操作是将标准错误重定向，标准输出和标准错误都被指向伪终端的屏幕显示，所以我们经常看到的一个命令的输出通常是同时包含了标准输出和标准错误的结果的。比如下面的操作：

使用 `cat` 命令同时读取两个文件，其中一个存在(`test.c` 之前已经创建，另一个不存在)

```
cat Documents/test.c hello.c
```

可以看到除了正确输出了前一个文件的内容，还在末尾出现了一条错误信息

下面命令将输出重定向到一个文件，这里将在看不到任何输出了

```
cat Documents/test.c hello.c > somefile
```

将标准错误重定向到标准输出，再将标准输出重定向到文件，注意要将重定向到文件写到前面

```
cat Documents/test.c hello.c >somefile 2>&1
```

或者只用 `bash` 提供的特殊的重定向符号"`&`"将标准错误和标准输出同时重定向到文件

```
cat Documents/test.c hello.c &>somefilehell
```

注意应该在输出重定向文件描述符前加上`&`，否则 `shell` 会当做重定向到一个文件名为 `1` 的文件中

3. 使用 `tee` 命令同时重定向到多个文件

经常可能还有这样的需求，除了需要将输出重定向到文件之外也需要将信息打印在终端，那么可以使用 `tee` 命令来实现：

```
echo 'hello shiyanlou' | tee hello
```

```
cat hello
```

4. 永久重定向

前面的重定向操作都只是临时性的，即只对当前命令有效，那如何做到“永久”有效呢，比如在一个脚本中，你需要某一部分的命令的输出全部进行重定向，难道要让在每个命令上面加上临时重定向的操作吗，当然不需要，我们可以使用 `exec` 命令实现“永久”重定向。`exec` 命令的作用是使用指定的命令替换当前的 `Shell`，及使用一个进程替换当前进程，或者指定新的重定向：

```
bash 先开启一个子 Shell
```

```
exec 1>somefile 使用 exec 替换当前进程的重定向，将标准输出重定向到一个文件
```

后面执行的命令的输出都将被重定向到文件中，直到退出当前子 `shell`，或取消 `exec` 的重定向

```
ls
```

```
exit
```

```
cat somefile
```

5. 创建输出文件描述符

默认在 `Shell` 中可以有 9 个打开的文件描述符，上面我们使用了也是它默认提供的 0, 1, 2 号文件描述符，另外我们还可以使用 3-8 的文件描述符，只是它们默认没有打开而已，可以使用下面命令查看当前 `Shell` 进程中打开的文件描述符：

```
cd /dev/fd/;ls -Al
```

同样使用 `exec` 命令可以创建新的文件描述符：

```
bash
```

```
exec 3>somefile
```

先进入目录，再查看，否则你可能不能得到正确的结果，然后再回到上一次的目录：

```
cd /dev/fd/;ls -Al;cd -
```

```
echo "this is test" >&3
```

```
cat somefile
```

```
exit
```

6. 关闭文件描述符

如上面我们打开的 3 号文件描述符，可以使用如下操作将它关闭：

```
exec 3>&-
```

```
cd /dev/fd/;ls -Al;cd -
```

7. 完全屏蔽命令的输出

在 `Linux` 中有一个被成为“黑洞”的设备文件，所以导入它的数据都将被“吞噬”。

在类 `UNIX` 系统中，`/dev/null`，或称空设备，是一个特殊的设备文件，它通常被用于丢弃不需要的输出流，或作为用于输入流的空文件，这些操作通常由重定向完成。读取它则会立即得到一个 `EOF`。

我们可以利用设个 `/dev/null` 屏蔽命令的输出：

```
cat Documents/test.c nefile 1>/dev/null 2>&1
```

向上面这样的操作将使你得不到任何输出结果。

8.使用 xargs 分割参数列表

xargs 是一条 UNIX 和类 UNIX 操作系统的常用命令。它的作用是将参数列表转换成小块分段传递给其他命令，以避免参数列表过长的问题。这个命令在有些时候十分有用，特别是当用来处理产生大量输出结果的命令如 find, locate 和 grep 的结果，详细用法请参看 man 文档。

```
cut -d : -f1 < /etc/passwd | sort | xargs echo
```

上面这个命令用于将/etc/passwd 文件按:分割取第一个字段排序后，使用 echo 命令生成一个列表。

Linux 下软件安装

一、Linux 上的软件安装

通常 Linux 上的软件安装主要有三种方式：

- 在线安装

- 从磁盘安装 deb 软件包

- 从二进制软件包安装

二、在线安装

在不同的 linux 发行版上面在线安装方式会有一些差异包括使用的命令及它们的包管理工具，因为我们的开发环境是基于 ubuntu 的，所以这里我们涉及的在线安装方式将只适用于 ubuntu 发行版，或其它基于 ubuntu 的发行版如国内的 ubuntukylin(优麒麟)，ubuntu 又是基于 debian 的发行版，它使用的是 debian 的包管理工具 dpkg，所以一些操作也适用于 debian。而在其它一些采用其它包管理工具的发行版如 redhat, centos, fedora 等将不适用(redhat 和 centos 使用 rpm)。

例如安装 w3m(一个命令行的简易网页浏览器)，输入如下命令

```
sudo apt-get install w3m
```

1. apt 包管理工具介绍

APT 是 Advanced Package Tool (高级包装工具) 的缩写，是 Debian 及其派生发行版的软件包管理器，APT 可以自动下载，配置，安装二进制或者源代码格式的软件包，因此简化了 Unix 系统上管理软件的过程。

当在执行安装操作时，首先 apt-get 工具会在本地的一个数据库中搜索关于 w3m 软件的相关信息，并根据这些信息在相关的服务器上下载软件安装，但既然是在线安装软件，为何会在本地的数据库中搜索？

我们需要定期从服务器上下载一个软件包列表，使用 `sudo apt-get update` 命令来保持本地的软件包列表是最新的，而这个表里会有软件依赖信息的记录，对于软件依赖，比如在安装 w3m 软件的时候，这个软件需要 libgc1c2 这个软件包才能正常工作，这个时候 apt-get 在安装软件的时候会一并替我们安装了，以保证 w3m 能正常的工作。

2. apt-get

apt-get 使用各用于处理 apt 包的公用程序集，可以用它来在线安装、卸载和升级软件包等，下面列出一些 apt-get 包含的常用的一些工具：

工具	说明
install	其后加上软件包名，用于安装一个软件包
update	从软件源镜像服务器上下载/更新用于更新本地软件源的软件包列表
upgrade	升级本地可更新的全部软件包，但存在依赖问题时将不会升级，通常会在更新之前执行一次 update
dist-upgrade	解决依赖关系并升级(存在一定危险性)
remove	移除已安装的软件包，包括与被移除软件包有依赖关系的软件包，但不包含软件包的配置文件
autoremove	移除之前被其他软件包依赖，但现在不再被使用的软件包
purge	与remove相同，但会完全移除软件包，包含其配置文件
clean	移除下载到本地的已经安装的软件包，默认保存在/var/cache/apt/archives/
autoclean	移除已安装的软件的旧版本软件包

下面是一些 apt-get 常用的参数：

参数	说明
-y	自动回应是否安装软件包的选项，在一些自动化安装脚本中使用这个参数将十分有用
-s	模拟安装
-q	静默安装方式，指定多个 q 或者 -q=#, #表示数字，用于设定静默级别，这在你不想要在安
-f	修复损坏的依赖关系
-d	只下载不安装
--reinstall	重新安装已经安装但可能存在问题的软件包
--install-suggests	同时安装APT给出的建议安装的软件包

3. 安装软件包

关于安装，如前面演示的一样只需要执行 apt-get install <软件包名>即可，除了这一点，还应该掌握的是如何重新安装软件包。很多时候我们需要重新安装一个软件包，比如系统被破坏，或者一些错误的配置导致软件无法正常工作。

可以使用如下方式重新安装：

```
sudo apt-get --reinstall install w3m
```

另外，在不知道软件包完整名的时候使用 Tab 键补全软件包名或采用后边搜索的方法进行安装。有时候需要同时安装多个软件包，还可以使用正则表达式匹配软件包名进行批量安装。

4. 软件升级

```
sudo apt-get update # 更新软件源
```

```
sudo apt-get upgrade # 升级没有依赖问题的软件包
```

```
sudo apt-get dist-upgrade # 升级并解决依赖关系
```

5. 卸载软件

如果现在需要卸载 w3m，同样只需一个命令加回车 `sudo apt-get remove w3m`，系统会有一个确认的操作，之后这个软件便被成功卸载了。

根据不同目的或可以执行：

```
sudo apt-get purge w3m 或 sudo apt-get --purge remove #不保留配置文件的移除 或
```

```
sudo apt-get autoremove #移除不再需要的被依赖的软件包
```

6. 软件搜索

当自己刚知道了一个软件，想下载使用，需要确认软件仓库里面有没有，就需要用到搜索功能了，命令如下：

```
sudo apt-cache search softname1 softname2 softname3.....
```

apt-cache 命令则是针对本地数据进行相关操作的工具，search 顾名思义在本地的数据库中寻找有关 softname1 softname2 相关软件的信息。如

```
sudo apt-cache search w3m
```

查找 qq：

```
sudo apt-cache search qq
```

从结果中可以看到有 qq 的对应 deb 文件：wine-qqintl，不过很可惜版本太老无法使用。

三、使用 dpkg 从本地磁盘安装 deb 软件包

1. dpkg 介绍

dpkg (Debian Package) 是 Debian 软件包管理器的基础。dpkg 与 RPM 十分相似，同样被用于安装、卸载和供给和 .deb 软件包相关的信息。经常可以在网络上见到以 deb 形式打包的软件包，就需要使用 dpkg 命令来安装。

dpkg 常用参数介绍：

参数	说明
<code>-i</code>	安装指定deb包
<code>-R</code>	后面加上目录名，用于安装该目录下的所有deb安装包
<code>-r</code>	remove，移除某个已安装的软件包
<code>-I</code>	显示 deb 包文件的信息
<code>-s</code>	显示已安装软件的信息
<code>-S</code>	搜索已安装的软件包
<code>-L</code>	显示已安装软件包的目录信息

2. 使用 dpkg 安装 deb 软件包

先使用 apt-get 加上 -d 参数只下载不安装，下载 emacs 编辑器的 deb 包：

```
sudo apt-get -d install emacs
```

下载完成后，可以查看 /var/cache/apt/archives/ 目录下的内容：

```
ls /var/cache/apt/archives/
```

然后将第一个 deb 拷贝到 home 目录下：

```
cp /var/cache/apt/archives/emacs24_24.3+1-4ubuntu1_amd64.deb ~
```

使用 dpkg 安装：

```
sudo dpkg -i emacs24_24.3+1-4ubuntu1_amd64.deb
```

安装失败，因为这个包还额外依赖了一些软件包，这意味着，如果主机目前没有这些被依赖的软件包，直接使用 dpkg 安装可能会存在一些问题，因为 dpkg 并不能解决依赖关系。

使用 apt-get 的 -f 参数修复依赖关系的安装：

```
sudo apt-get -f install
```

没有任何错误，这样就安装成功了，然后可以运行 emacs 程序：

```
emacs
```

3. 查看已安装软件包的安装目录

可以通过 dpkg 查看 linux 将软件安装的位置

使用 dpkg -L 查看 deb 包目录信息：

```
sudo dpkg -L emacs24 （为什么软件名称后面有 24？）
```

查看电脑 bios 模式（Legacy or UEFI）：

```
sudo dpkg -l | grep grub-efi
```

四、从二进制包安装

二进制包的安装比较简单，只需将从网络上下载的二进制包解压后放到合适的目录，然后将包含可执行的主程序文件的目录添加进 PATH 环境变量即可。

Linux 进程之初步了解

一、概念的理解

引入进程是因为传统意义上的程序已经不足以描述 OS 中各种活动之间的动态性、并发性、独立性还有相互制约性。就比如程序就像一个公司，只是一些证书，文件的堆积（静态实体）。而当公司运作起来就有各个部门的区分，财务部，技术部，销售部等等，就像各个进程，各个部门之间可以独立运做，也可以有交互（独立性、并发性）。

而随着程序的发展越做越大，又会继续细分，从而引入了线程的概念，当代多数操作系统、Linux 2.6 及更新的版本中，进程本身不是基本运行单位，而是线程的容器。就像上述所说的，每个部门又会细分为各个工作小组（线程），而工作小组需要的资源需要向上级（进程）申请。

线程（thread）是操作系统能够进行运算调度的最小单位。它被包含在进程之中，是进程中的实际运作单位。一条线程指的是进程中一个单一顺序的控制流，一个进程中可以并发多个线程，每条线程并行执行不同的任务。因为线程中几乎不包含系统资源，所以执行更快、更有效率。

二、进程的属性

2.1 进程的分类

大概明白进程是个什么样的存在，进一步对于进程的一个了解就是进程分类，可以从两个角度来分，第一个角度以进程的功能与服务的对象来分；第二个角度以应用程序的服务类型来分

第一个角度来看，我们可以分为用户进程与系统进程

用户进程：通过执行用户程序、应用程序或称之为内核之外的系统程序而产生的进程，此类进程可以在用户的控制下运行或关闭。

系统进程：通过执行系统内核程序而产生的进程，比如可以执行内存资源分配和进程切换等相对底层的工作；而且，该进程的运行不受用户的干预，即使是 root 用户也不能干预系统进程的运行。

第二角度来看，我们可以将进程分为交互进程、批处理进程、守护进程

交互进程：由一个 shell 终端启动的进程，在执行过程中，需要与用户进行交互操作，可以运行于前台，也可以运行在后台。

批处理进程：该进程是一个进程集合，负责按顺序启动其他的进程。

守护进程：守护进程是一直运行的一种进程，经常在 Linux 系统启动时启动，在系统关闭时终止。它们独立于控制终端并且周期性的执行某种任务或等待处理某些发生的事件。例如 httpd 进程，一直处于运行状态，等待用户的访问。还有经常用的 cron（在 CentOS 系列为 crond）进程，这个进程为 crontab 的守护进程，可以周期性的执行用户设定的某些任务。

2.2 进程的衍生

进程有这么多的种类，那么进程之间定是有相关性的，而这些有关联性的进程又是如何产生的，如何衍生的？

比如我们启动了终端，就是启动了一个 bash 进程，我们可以在 bash 中再输入 bash 则会再启动一个 bash 的进程，此时第二个 bash 进程就是由第一个 bash 进程创建出来的，他们直接又是个什么关系？

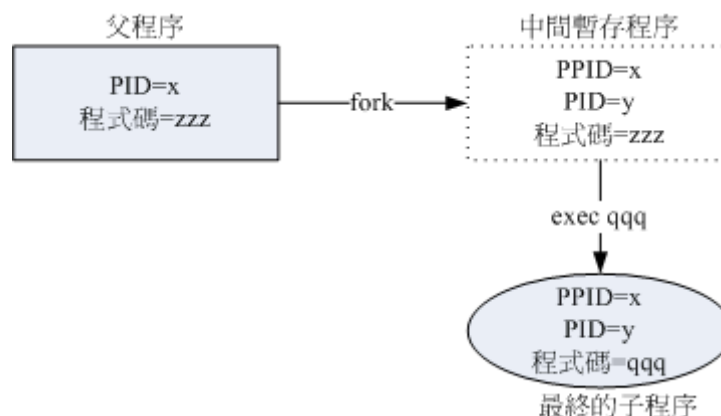
我们一般称呼第一个 bash 进程是第二 bash 进程的父进程，第二 bash 进程是第一个 bash 进程的子进程，这层关系是如何得来的呢？

关于父进程与子进程便会提及这两个系统调用 fork() 与 exec()

fork() 是一个系统调用（system call），它的主要作用就是为当前的进程创建一个新的进程，这个新的进程就是它的子进程，这个子进程除了父进程的返回值和 PID 以外其他的都一模一样，如进程的执行代码段，内存信息，文件描述，寄存器状态等等

exec() 也是系统调用，作用是切换子进程中的执行程序也就是替换其从父进程复制过来的代码段与数据段。

如右图：



子进程的退出与资源的回收与父进程有很大的相关性。当一个子进程要正常的终止运行时，或者该进程结束时它的主函数 main() 会执行 exit(n)；或者 return n，这里的返回值 n 是一个信号，系统会把这个 SIGCHLD 信号传给其父进程，当然若是异常终止也往往是因为这个信号。而这个时候的子进程代码执行部分其实已经结束执行了，系统的资源也基本归还给系统了，但是其进程的进程控制块

(PCB) 仍驻留在内存中，而它的 PCB 还在，代表这个进程还存在（因为 PCB 就是进程存在的唯一标志，里面有 PID 等消息），并没有消亡，这样的进程称之为僵尸进程（Zombie）。

正常情况下，父进程会收到两个返回值一个是 exit code 也是 SIGCHLD 信号与 reason for termination 之后，父进程会使用 wait(&status) 系统调用以获取子进程的退出状态，然后内核就可以从内存中释放已结束的子进程的 PCB；而如若父进程没有这么做的话，子进程的 PCB 就会一直驻留在内存中，一直留在系统中做为僵尸进程（Zombie）。Linux 系统中能使用的 PID 是有限的，如果系统中存在有大量的僵尸进程，系统将会因为没有可用的 PID 从而导致不能产生新的进程。

而另外如果父进程结束（非正常的结束），未能及时收回子进程，子进程仍在运行，这样的子进程称之

为孤儿进程。在 Linux 系统中，孤儿进程一般会被 init 进程所“收养”，成为 init 的子进程。由 init 来做善后处理，所以它并不至于像僵尸进程那样无人问津，不管不顾，大量存在会有危害。

进程 0 是系统引导时创建的一个特殊进程，也称之为内核初始化，其最后一个动作就是调用 fork() 创建出一个子进程运行 /sbin/init 可执行文件，而该进程就是 PID=1 的进程 1，也就是 init 进程，而进程 0 就转为交换进程（也被称为空闲进程），而进程 1（init 进程）是第一个普通用户态的进程，再由它不断调用 fork() 来创建系统里其他的进程，所以它是所有进程的父进程或者祖先进程。同时它是一个守护程序，直到计算机关机才会停止。

通过以下的命令我们可以很明显的看到这样的结构：

`pstree`

通过以上的显示结果可以看的很清楚，init 为所有进程的父进程或者说是祖先进程。

我们还可以使用这样一个命令来看，其中 pid 就是该进程的一个唯一编号，ppid 就是该进程的父进程的 pid，command 就是该进程是执行什么样的程序或者脚本而产生的

`ps -fxo user,ppid,pid,pgid,command`

可以在图中看见我们执行的 ps 就是由 bash 通过 fork-exec 创建的子进程而执行的。使用这样的命令我们也能清楚的看见 init 如上文所说是由进程 0 这个初始化进程来创建而出的子进程，而其他的进程基本是由 init 创建的子进程，或者是由它的子进程创建出来的子进程。所以 init 是用户进程的第一个进程也是所有用户进程的父进程或者祖先进程。就像一个树状图，而 init 进程就是这棵树的根，其他进程由根不断的发散，开枝散叶。

2.3 进程组与 Sessions

每一个进程都会是一个进程组的成员，而且这个进程组是唯一存在的，他们是依靠 PGID (process group ID) 来区别的，而每当一个进程被创建的时候，它便会成为其父进程所在组中的一员。

一般情况，进程组的 PGID 等同于进程组的第一个成员的 PID，并且这样的进程称为该进程组的领导者，也就是领导进程，进程一般通过使用 getpgrp() 系统调用来寻找其所在组的 PGID，领导进程可以先终结，此时进程组依然存在，并持有相同的 PGID，直到进程组中最后一个进程终结。

与进程组类似，每当一个进程被创建的时候，它便会成为其父进程所在 Session 中的一员，每一个进程组都会在一个 Session 中，并且这个 Session 是唯一存在的，Session 中的每个进程都称为一个工作(job)。每个会话可以连接一个终端(control terminal)。当控制终端有输入输出时，都传递该会话的前台进程组。Session 意义在于将多个 jobs 囊括在一个终端，并取其中的一个 job 作为前台，来直接接收该终端的输入输出以及终端信号。其他 jobs 在后台运行。

前台 (foreground) 就是在终端中运行，与用户能有交互的

后台 (background) 就是在终端中运行，但是用户并不能与其任何的交互，也不会显示其执行的过程。

2.4 工作管理

bash(Bourne-Again shell)支持工作控制(job control),而 sh(Bourne shell)并不支持。

并且每个终端或者说 bash 只能管理当前终端的中的 job，不能管理其他终端中的 job。

当一个进程在前台运作时可以用 ctrl + c 来终止它，但是若是在后台的话就不行了，并且在一个终端 bash 中只能管理当前 bash 里的 job。

可以通过 & 这个符号，让命令在后台中运行，如下：

`ls &`

结果所显示的 [1] 236 分别是该工作的 job number 与该进程的 PID，而最后一行的 Done 表示该命令已经在后台执行完毕。

可以通过 ctrl + z 使我们的当前工作停止并丢到后台中去：

`tail -f /var/log/dpkg.log`

`ctrl + z`

被停止并放置在后台的工作可以使用这个命令来查看：

`jobs`

结果的第一列显示的为被放置后台的工作的编号(jobnumber)，而第二列的 + 表示最近被放置后台的工作，同时也表示预设的工作，也就是若是有什么针对后台的工作的操作，首先对预设的工作，- 表示倒数第二被放置后台的工作，倒数第三个以后都不会有这样的符号修饰，第三列表示它们的状态，而最后一列表示该进程执行的命令。

可以通过这样的 fg (foreground) 命令将后台的工作拿到前台来，格式为 fg [%jobnumber]，如

`fg %3`

若是想让其后台运作就使用 bg (background) 命令，格式为 bg [%jobnumber]

`bg %3`

删除一个工作，或者重启等等：

`kill signal %jobnumber`

signal 从 1-64 个信号值可以选择，可以这样查看

`kill -l`

常使用下面的值：

信号值	作用
-1	重新读取参数运行，类似与restart
-2	如同 ctrl+c 的操作退出
-9	强制终止该任务
-15	正常的方式终止该任务

注意

若是在使用 kill+信号值然后直接加数字的话，这个数字代表的是 pid，你将会对 pid 对应的进程做操作，若是在使用 kill+信号值然后 %jobnumber，这时所操作的对象才是 job，这个数字就是就当前 bash 中后台的运行的 job 的 ID。如：

`kill -9 %1` 或

`kill -9 102`

Linux 进程之管理控制

一、进程的查看

不管在测试的时候还是在实际的生产环境中或者自己的使用过程中，难免遇到进程的一些异常，所以 Linux 为我们提供了一些工具可以查看进程的一些状态信息，我们可以通过 top 动态实时的查看进程的状态的以及系统的一些信息如 CPU、内存信息等等，同样可以通过 ps 来静态查看当前的进程信息，同时还可以使用 pstree 来查看当前活跃进程的树形结构。

1.1 top 工具的使用

top 工具是常用的一个查看工具，能实时的查看系统的一些关键信息的变化以及在进程中的实时变化：

`top`

在显示结果第一行的 load average (the system load is a measure of the amount of work that a computer system is doing) 是对当前 CPU 工作量的度量，具体来说也就是指运行队列的平均长度，后边的三个参数分别对应 1、5、15 分钟内 cpu 的平均负载。

就单个 CPU 单核的情况，load=0 表示当前没有任务，load<1 表示当前任务不多，资源还充足，load=1，资源正好用完，load>1 资源已经用完，大量进程还在请求，load>5 时，说明系统在超负荷运行。

对于多核系统需要将得到的这个值除以核数来看。可以通过以下的命令来查看 CPU 的个数与核数：

`cat /proc/cpuinfo |grep "physical id"|sort |uniq|wc -l` #查看物理 CPU 的个数

`cat /proc/cpuinfo |grep "physical id"|grep "0"|wc -l` #每个 cpu 的核心数

通常我们会先看 15 分钟的值来看这个大体的趋势，然后再看 5 分钟的值对比来看是否有下降的趋势。其他 top 命令显示的参数可以通过 [top 解释](#) 来查看。

注意第五行的 PR 是该进程执行的优先级 priority 值，NI 是该进程的 nice 值。NI 值叫做静态优先级，是用户空间的一个优先级值，其取值范围是 -20 至 19。这个值越小，表示进程“优先级”越高，即 -20 优先级最高，0 是默认的值，而 19 优先级最低。PR 值表示 Priority 值叫动态优先级，是进程在内核中实际的优先级值，Linux 实际上实现了 140 个优先级范围，取值范围是从 0-139，这个值越小，优先级越高。而这其中的 0 - 99 是实时的值，而 100 - 139 是给用户的。

同时，top 是一个前台程序，是可交互的，如下表

常用交互命令	解释
q	退出程序
l	切换显示平均负载和启动时间的信息
P	根据CPU使用百分比大小进行排序
M	根据驻留内存大小进行排序
i	忽略闲置和僵死的进程，这是一个开关式命令
k	终止一个进程，系统提示输入 PID 及发送的信号值。一般终止进程用15信号，不能正常结束则使用9信号。安全模式下该命令被屏蔽

1.2 ps 工具的使用

ps 也是我们最常用的查看进程的工具之一。

使用 -l 参数可以显示自己这次登陆的 bash 相关的进程信息罗列出来：

```
ps -l
```

相对来说我们更加常用下面这个命令，它将会罗列出所有的进程信息：

```
ps aux
```

若是查找其中的某个进程的话，我们还可以配合着 grep 和正则表达式一起使用，如下：

```
ps aux | grep bash
```

此外我们还可以在查看时，将连同部分的进程呈树状显示出来：

```
ps axjf
```

结果中 TPGID 栏写着 -1 的都是没有控制终端的进程，也就是守护进程。

也可以是用这样的命令，来自定义我们所需要的参数显示：

```
ps -afxo user,ppid,pid,pgid,command
```

1.3 pstree 工具的使用

通过 pstree 可以很直接的看到相同的进程数量，最主要是可以看到所有进程的之间的相关性：

```
pstree
```

参数选择：

#-A : 各程序树之间以 ASCII 字元來連接；

#-p : 同时列出每个 process 的 PID；

#-u : 同时列出每个 process 的所屬账户名称。

如：

```
pstree -up # 同时列出每个 process 的 PID 及所屬账户名称。
```

二、进程的管理

2.1 kill 命令的掌握

打开 gedit，查看 gedit 的 pid：

```
ps aux | grep gedit
```

查看到为 2928，使用 kill 来结束该进程：

```
kill -9 2928
```

再次查找，发现已经查找不到该进程：

```
ps aux | grep gedit
```

2.2 进程的执行顺序

可以通过 ps 命令看到大部分的进程都是处于休眠的状态，如果这些进程都被唤醒，那么该谁最先享受 CPU 的服务，后面的进程又该是一个什么样的顺序呢？进程调度的队列又该如何去排列呢？

当然就是靠该进程的优先级值来判定进程调度的优先级，而优先级的值就是上文所提到的 PR 与 nice 来控制与体现了。而 nice 的值是可以通过 nice 命令来修改的，而需要注意的是 nice 值可以调整的范围是 -20 ~ 19，其中 root 有着至高无上的权力，既可以调整自己的进程也可以调整其他用户的程序，并且是所有的值都可以用，而普通用户只可以调制属于自己的进程，并且其使用的范围只能是 0 ~ 19，因为系统为了避免一般用户抢占系统资源而设置的一个限制。

如打开 vim 编辑器放在后台（或直接打开图形界面），指定进程的优先级为 5（若优先级为 -5，则命令应为 -n -5）：

```
nice -n -5 vim &
```

再用 ps 查看其优先级：


```
ps -afxo user,ppid,pid,stat,pri,ni,time,command | grep vim
还可以用 renice 来修改已经存在的进程的优先级：
renice -5 pid
```

Linux 之日志系统

一、常见的日志

日志是一个系统管理员，一个运维人员，甚至是开发人员不可或缺的东西，系统用久了偶尔也会出现一些错误，我们需要日志来给系统排错，在一些网络应用服务不能正常工作的时候，我们需要用日志来做问题定位，日志还是过往时间的记录本，我们可以通过它知道我们是否被不明用户登陆过等等。在 Linux 中大部分的发行版都内置使用 syslog 系统日志，常见的日志一般存放在 /var/log 中，可以通过下面的命令查看：

```
ll /var/log
```

常见的系统日志及其记录的信息如下：

日志名称	记录信息
alternatives.log	系统的一些更新替代信息记录
apport.log	应用程序崩溃信息记录
apt/history.log	使用apt-get安装卸载软件的信息记录
apt/term.log	使用apt-get时的具体操作，如 package 的下载打开等
auth.log	登录认证的信息记录
boot.log	系统启动时的程序服务的日志信息
btmtp	错误登陆的信息记录
Consolekit/history	控制台的信息记录
dist-upgrade	dist-upgrade这种更新方式的信息记录
dmesg	启动时，显示屏幕上内核缓冲信息,与硬件有关的信息
dpkg.log	dpkg命令管理包的日志。
faillog	用户登录失败详细信息记录
fontconfig.log	与字体配置有关的信息记录
kern.log	内核产生的信息记录，在自己修改内核时有很大帮助
lastlog	用户的最近信息记录
wtmp	登录信息的记录。wtmp可以找出谁正在登陆进入系统，谁使用命令显示这个文件或信息等
syslog	系统信息记录

用这样的命令来看看 auth.log 中的信息：

```
cd /var/log
less auth.log
```

可以从中得到的信息有日期与 ip 地址的来源以及登陆的用户与工具。在 apt 文件夹中的日志信息，其中有两个日志文件 history.log 与 term.log，两个日志文件的区别在于 history.log 主要记录了进行了那个操作，相关的依赖有哪些，而 term.log 则是较为具体的一些操作，主要就是下载包，打开包，安装包等等的细节操作。可以通过下列命令查看：

```
less /var/log/apt/history.log
less /var/log/apt/term.log
```

其他的日志格式也都类似与之前我们所查看的日志，主要便是时间，操作。而这其中有两个比较特殊的日志，其查看的方式比较与众不同，因为这两个日志并不是 ASCII 文件而是被编码成了二进制文件，所以我们并不能直接使用 less、cat、more 这样的工具来查看，这两个日志文件是 wtmp，lastlog，其查看方法可以用 last 工具：

```
last wtmp
last lastlog
以及 lastlog 工具：
lastlog -help
lastlog -u maroon
```

二、配置的日志

这些日志是如何产生的，并且通过上面的例子我们可以看出大部分的日志信息似乎格式都都很类似，并且为什么都会出现在这个文件夹中。

这样的实现可以通过两种方式，一种是由软件开发商来自定义日志格式然后指定输出日志位置，还有一种方式就是 Linux 提供的日志服务程序，而我们这里系统日志是通过 syslog 来实现，提供日志管理服务。

rsyslog 的全称是 rocket-fast system for log，它提供了高性能，高安全功能和模块化设计。rsyslog 能够接受从各种各样的来源，将其输入，输出的结果到不同的目的地。rsyslog 可以提供超过每秒一百万条消息给目标文件，这样能实时收集日志信息的程序都会有其守护进程如 rsyslog 的守护进程便是 rsyslogd。

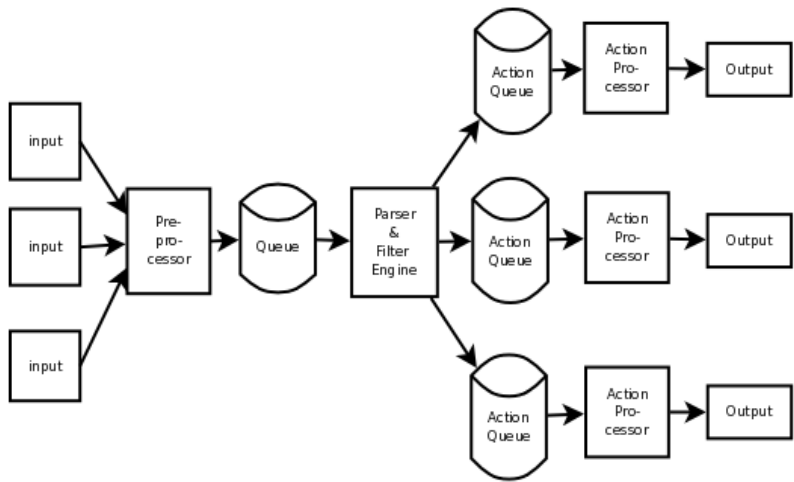
既然它是一个服务，那么它便是可以配置，为我们提供一些我们自定义的服务

首先我们来看 rsyslog 的配置文件是什么样子的，而 rsyslog 的配置文件有两个，一个是 /etc/rsyslog.conf 一个是 /etc/rsyslog.d/50-default.conf。第一个主要是配置的环境，也就是 rsyslog 的加载什么模块，文件的所有者等；而第二个主要是配置的 Filter Conditions，可以通过下列命令查看：

```
vim /etc/rsyslog.conf
```

```
vim /etc/rsyslog.d/50-default.conf
```

Rsyslog 架构如下图所示，从图中我们可以很清楚的看见，rsyslog 还有一个核心的功能模块便是 Queue，也正是因为它才能做到如此高的并发。



再来看 /etc/rsyslog.d/50-default.conf 这个配置文件，这个文件中主要是配置的 Filter Conditions，也就在流程图中所看见的 Parser & Filter Engine,它的名字叫 Selectors 是过滤 syslog 的传统方法，主要由两部分组成，facility 与 priority，其配置格式如下：

facility.priority log_location

其中一个 priority 可以指定多个 facility，多个 facility 之间使用逗号，分割开。

rsyslog 通过 Facility 的概念来定义日志消息的来源，以便对日志进行分类，Facility 的种类有：

类别	解释
kern	内核消息
user	用户信息
mail	邮件系统消息
daemon	系统服务消息
auth	认证系统
authpriv	权限系统
syslog	日志系统自身消息
cron	计划安排
news	新闻信息
local0~7	由自定义程序使用

如将auth 与 authpriv 的所有优先级的信息全都输出于 /var/log/auth.log 日志中：

```
auth,authpriv.* /var/log/auth.log
```

而其中有类似于这样的配置信息意思有细微的差别，如下：

```
kern.* -/var/log/kern.log
```

- 代表异步写入，也就是日志写入时不需要等待系统缓存的同步，也就是日志还在内存中缓存也可以继续写入无需等待完全写入硬盘后再写入。通常用于写入数据比较大时使用。

与日志相关的还有一个还有常用的命令 logger,logger 是一个 shell 命令接口，可以通过该接口使用 Syslog 的系统日志模块，还可以从命令行直接向系统日志文件写入信息，如下：

```
ping 127.0.0.1 | logger -it logger_test -p local3.notice & #向 syslog 写入数据
```

```
tail -f /var/log/syslog #查看是否有数据写入
```

三、转储的日志

在本地的机器中每天都有成百上千条日志被写入文件中，更别说是服务器，每天都会有数十兆甚至更多的日志信息被写入文件中，如果是这样的话，每天的日志文件不断的膨胀，那岂不是要占用许多的空间，所以有个叫 logrotate 的东西诞生了。

logrotate 程序是一个日志文件管理工具。用来把旧的日志文件删除，并创建新的日志文件。我们可以根据日志文件的大小，也可以根据其天数来切割日志，来管理日志。这个过程又叫做“转储”

大多数 Linux 发行版使用 logrotate 或 newsyslog 对日志进行管理。logrotate 程序不但可以压缩日志文件，减少存储空间，还可以将日志发送到指定 E-mail，方便管理员及时查看日志。

显而易见，logrotate 是基于 CRON 来运行的，其脚本是 /etc/cron.daily/logrotate；同时我们可以在 /etc/logrotate 中找到其配置文件：

```
cat /etc/logrotate.conf
```

```
# see "man logrotate" for details //可以查看帮助文档
# rotate log files weekly
weekly //设置每周转储一次(daily、weekly、monthly当然可以使用这些参数每天、星期、月)
# keep 4 weeks worth of backlogs
rotate 4 //最多转储4次
# create new (empty) log files after rotating old ones
create //当转储后文件不存在时创建它
# uncomment this if you want your log files compressed
compress //通过gzip压缩方式转储(nocompress可以不压缩)
# RPM packages drop log rotation information into this directory
include /etc/logrotate.d //其他日志文件的转储方式配置文件，包含在该目录下
# no packages own wtmp -- we'll rotate them here
/var/log/wtmp { //设置/var/log/wtmp日志文件的转储参数
    monthly //每月转储
    create 0664 root utmp //转储后文件不存在时创建它，文件所有者为root，所属组为utmp，对应的权限为0664
    rotate 1 //最多转储一次
}
```

当然在 /etc/logrotate.d/ 中有各项应用的 logrotate 配置，还有更多的配置参数，可以使用 man 查看，如按文件大小转储，按当前时间格式命名等等参数配置。

参考网站：

[Linux 命令实例练习](#)

[Linux 基础入门 \(新版\)](#)

[Linux 命令大全](#)

另，一个很不错的 linux 命令行的入门教学网站：

[UNIX Tutorial for Beginners](#)

git 远程操作

git 是目前最流行的版本管理系统，Git 有很多优势，其中之一就是远程操作非常简便。

git 命令之 git clone 用法：

在使用 git 来进行版本控制时，为了得一个项目的拷贝(copy)，我们需要知道这个项目仓库的地址(Git URL)。Git 能在许多协议下使用，所以 Git URL 可能以 ssh://, http(s)://, git://, 或是只是以一个用户名 (git 会认为这是一个 ssh 地址) 为前缀。

有些仓库可以通过不只一种协议来访问，例如，Git 本身的源代码你既可以用 git:// 协议来访问：

```
git clone git://git.kernel.org/pub/scm/git/git.git
```

也可以通过 http 协议来访问：

```
git clone http://www.kernel.org/pub/scm/git/git.git
```

git:// 协议较为快速和有效，但是有时必须使用 http 协议，比如你公司的防火墙阻止了你的非 http 访问请求。如果你执行了上面两行命令中的任意一个，你会看到一个新目录：'git'，它包含有所的 Git 源代码和历史记录。

在默认情况下，Git 会把"Git URL"里最后一级目录名的'.git'的后缀去掉，做为新克隆(clone)项目的目录名，如下列命令会建立一个目录叫'linux-2.6'：

```
git clone http://git.kernel.org/linux/kernel/git/torvalds/linux-2.6.git
```

另外，如果访问一个 Git URL 需要用法名和密码，可以在 Git URL 前加上用户名，并在它们之间加上@ 符合以表示分割，然后执行 git clone 命令，git 会提示你输入密码。如：

```
git clone robin.hu@http://www.kernel.org/pub/scm/git/git.git
```

这样将以作为 robin.hu 用户名访问 http://www.kernel.org/pub/scm/git/git.git，然后按回车键执行 git clone 命令，git 会提示你输入密码。

另外，我们可以通过 -b <name> 来指定要克隆的分支名，比如

```
git clone -b master2 ../server .
```

表示克隆名为 master2 的这个分支，如果省略 -b <name> 表示克隆 master 分支。

关于 git 的完整教程，可以参廖雪峰老师的教程：[廖雪峰 Git 教程](#)

终端下输入命令：

```
alsamixer
```

这是一个控制台下调节音量的命令，类似于 windows 下的音量控制窗口。

禁止使用触摸屏：

```
sudo modprobe -r psmouse
```

如果打开触摸板就是：

```
sudo modprobe psmouse
```

查看 Linux 版本的位数：

```
uname -a
```

如果显示的系统版本中出现 x86_64 字样，则说明系统是 64 位的。若出现 i386 或 i686 字样，则说明系统是 32 位的。

查看 CPU 配置：

```
lscpu
```

打开 pdf 文件：

```
evince file_name.pdf
```

打开图形文件：

```
eog image_name.pgm
```

包管理工具

主要的包管理工具

dpkg —Debian 包安装工具

apt-get —APT 的命令行前端

aptitude —APT 的高级字符和命令行前端

synaptic —图形界面的 APT 前端

dselect —使用菜单界面的包管理工具

tasksel —Task 安装工具

这些工具不是来取代对方的，比如 dselect 同时使用 APT 和 dpkg。

APT 使用 /var/lib/apt/lists/* 来跟踪可用的软件包，而 dpkg 使用的是 /var/lib/dpkg/available
aptitude

aptitude 是全新的可菜单操作的包安装工具，是针对 APT 设计的。从大多数参数来讲，aptitude 可以作为兼容 apt-get 的替代品。

PPA(Personal Package Archives)软件源

虽然 Ubuntu 官方软件仓库尽可能囊括所有的开源软件，但仍有很多软件包由于各种原因不能进入官方软件仓库。为了方便 Ubuntu 用户使用，launchpad.net 提供了个人软件包集，即 PPA，允许用户建立自己的软件仓库，通过 Launchpad 进行编译并发布为 2 进制软件包，作为 apt/新立得源供其他用户下载和更新。PPA 也被用来对一些打算进入 Ubuntu 官方仓库的软件，或者某些软件的新版本进行测试。

软件安装：
可以在 [launchpad 平台](#) 上直接搜索相关的软件名称以便获得相关源地址。如搜索为知笔记(wiznote) 然后添加源、更新源、安装：

```
sudo add-apt-repository ppa:wiznote-team
sudo apt-get update
sudo apt-get install wiznote
```

软件删除：

```
sudo add-apt-repository -r ppa:user/ppa-name
```

进入 /etc/apt/sources.list.d 文件夹，删除对应的源文件。

wget

Linux 系统中 wget 是一个下载文件的工具，它用在命令行下。对于 Linux 用户是必不可少的工具，我们经常要下载一些软件或从远程服务器恢复备份到本地服务器。wget 支持 HTTP，HTTPS 和 FTP 协议，可以使用 HTTP 代理。所谓的自动下载是指，wget 可以在用户退出系统之后在后台执行。这意味这你可以登录系统，启动一个 wget 下载任务，然后退出系统，wget 将在后台执行直到任务完成。

格式：wget [参数] [URL 地址]

参数：-r(递归下载) -c(断点续传) ..

如，下载单个网页或文件不需要加参数：

```
wget http://www.cnblogs.com/dingn/p/5658442.html
```

批量下载：

如果有多个文件需要下载，那么可以生成一个文件，把每个文件的 URL 写一行，例如生成文件

download.txt，然后用命令：

```
wget -i download.txt
```

这样就会把 download.txt 里面列出的每个 URL 都下载下来。（如果列的是文件就下载文件，如果列的是网站，那么下载首页）

pip

pip 是一个安装和管理 Python 包的工具

搜索：pip search "django"

安装：pip install django[>=2.0] (指点版本)

升级：pip install-U django

列出已安装的包：pip freeze

卸载包：pip uninstall django

如安装 Tensorflow:

```
pip install tensorflow
```

可以将需要安装的依赖库写为 .txt 文件，然后直接执行：

```
pip install -r requirements.txt
```


.run 文件安装

run 文件的安装很简单，只需要为该文件增加可执行属性，即可执行安装。比如安装 QT 编程软件，安装方法如下：

命令第一步：`chmod +x qt-opensource-linux-x64-5.7.0.run`

chmod 实际上是加权限命令，+x 表示可以执行

命令第二步：`./qt-opensource-linux-x64-5.7.0.run`

之后就出现 QT 安装的可视化界面

高翔的 C++ 编译讲解

C++ 程序编译

slambook/ch2/helloSLAM.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 int main( int argc, char** argv )
5 {
6     cout<<"Hello SLAM!"<<endl;
7     return 0;
8 }
```

`g++ helloSLAM.cpp`

`./a.out`

使用 cmake

理论上说，任意一个 C++ 程序都可以用 g++ 来编译。但当程序规模越来越大时，一个工程可能有多个文件夹和源文件，这时输入的编译命令将越来越长。通常一个小型 C++ 项目可能含有十几个类，各类间还存在着复杂的依赖关系。其中一部分要编译成可执行文件，另一部分编译成库文件。如果仅靠 g++ 命令，我们需要输入大量的编译指令，整个编译过程会变得异常烦琐。因此，对于 C++ 项目，使用一些工程管理工具会更加高效。在历史上工程师们曾使用 makefile 进行自动编译，但下面要谈的 cmake 比它更加方便。并且，我们会看到后面提到的大多数库都使用 cmake 来管理源代码。在一个 cmake 工程中，我们会用 cmake 命令生成一个 makefile 文件，然后，用 make 命令根据这个 makefile 文件的内容编译整个工程。仍然以上面的 helloSLAM.cpp 为例，这次我们不是直接使用 g++，而是用 cmake 来制作一个工程，然后再编译它。

在原文件夹中新建一个 CMakeLists.txt 文件，内容如下：

#申明要求的 cmake 最低版本

`cmake_minimum_required(VERSION 2.8)`

#申明一个 cmake 工程

`project(HelloSLAM)`

#添加可执行文件

`add_executable(helloSLAM helloSLAM.cpp)`

CMakeLists.txt 文件用于告诉 cmake 我们要对这个目录下的文件做什么事情。CMake-Lists.txt 文件的内容需要遵守 cmake 的语法。这个示例中，我们演示了最基本的工程：指定一个工程名和一个可执行程序。

现在，在当前目录下调用 cmake 对该工程进行分析：

`cmake .` #生成 makefile

用 make 命令对工程进行编译：

`make` #编译

现在这个过程中唯一让我们不满的是，cmake 生成的中间文件还留在我们的代码文件当中。当想要发布代码时，我们并不希望把这些中间文件一同发布出去。这时我们还需要把它们一个个删除，十分不便。一种更好的做法是让这些中间文件都放在一个中间目录中，在编译成功后，把这个中间目录删除即可。所以，更常见的编译 cmake 工程的做法如下：

`mkdir build`

`cd build`

`cmake ..`


`make`

我们新建了一个中间文件夹“build”，然后进入 build 文件夹，通过 `cmake ..` 命令对上一层文件夹，也就是代码所在的文件夹进行编译。这样，cmake 产生的中间文件就会生成在 build 文件夹中，与源代码分

开。当发布源代码时，只要把 build 文件夹删掉即可。

使用库

在一个 C++ 工程中，并不是所有代码都会编译成可执行文件。只有带有 main 函数的文件才会生成可执行程序。而另一些代码，我们只想把它们打包成一个东西，供其他程序调用。这个东西叫作库。一个库往往是许多算法、程序的集合，我们会在之后的练习中接触到许多库。例如，OpenCV 库提供了许多计算机视觉相关的算法，而 Eigen 库提供了矩阵代数的计算。因此，我们要学习如何用 cmake 生成库，并且使用库中的函数。如将 libhelloSLAM.cpp 生成库。

 slambook/ch2/libHelloSLAM.cpp

```
1 //这是一个库文件
2 #include <iostream>
3 using namespace std;
4 void printHello()
5 {
6     cout<<"Hello SLAM"<<endl;
7 }
```

只需在 CMakeLists.txt 中添加如下内容：

`add_library(hello libhelloSLAM.cpp)`


这条命令告诉 cmake，我们想把这个文件编译成一个叫作“hello”的库。然后，和上面一样，使用 cmake 编译整个工程后在 build 文件夹中就会生成一个 hello.a 文件，这就是我们得到的库。

在 Linux 中，库文件分成静态库和共享库两种。静态库以 .a 作为后缀名，共享库以 .so 结尾。所有库都是一些函数打包后的集合，差别在于静态库每次被调用都会生成一个副本，而共享库则只有一个副本，更省空间。如果想生成共享库而不是静态库，只需使用以下语句即可。

`add_library(hello_shared SHARED libhelloSLAM.cpp)`


此时得到的文件就是 hello_shared.so 了。

库文件是一个压缩包，里面有编译好的二进制函数。不过，如果仅有 .a 或 .so 库文件，那么我们并不知道里面的函数到底是什么，调用的形式又是什么样。为了让别人（或者自己）使用这个库，我们需要提供一个头文件，说明这些库里都有些什么。因此，对于库的使用者，只要拿到了头文件和库文件，就可以调用这个库了。下面编写 libhello 的头文件。

 slambook/ch2/libHelloSLAM.h

```
1 #ifndef LIBHELLOSLAN_H_
2 #define LIBHELLOSLAN_H_
3 void printHello();
4 #endif
```

此时在 helloSLAM.cpp 中添加头文件就可以在文件中使用库中定义的 printHello 函数了。

 slambook/ch2/useHello.cpp

```
1 #include "libHelloSLAM.h"
2 int main( int argc, char** argv )
3 {
4     printHello();
5     return 0;
6 }
```

然后，在 CMakeLists.txt 中添加一个可执行程序的生成命令，链接到刚才使用的库上：

`add_executable(useHello useHello.cpp)`

`target_link_libraries(useHello hello_shared)`

通过这两行语句，useHello 程序就能顺利使用 hello_shared 库中的代码了。

参考：<http://book.51cto.com/art/201705/540451.htm>

ccmake

ccmake 执行 CMake curses 交互界面进行交互配置，然后生成 makefile。第一次进入，会显示 “empty cache”，按 ‘c’ 即可，可以看到可配置的参数，需要修改默认参数时，上下键移动到对应位置按 “enter” 直接修改。全部修改完后按 ‘c’ 进行配置，配置完成按 ‘g’ 生成 makefile，按 ‘q’ 退出。

远程传文件

安装 openssh:

```
sudo apt-get install openssh-client
```

复制文件到远程电脑:

```
scp [文件名] [用户名]@[对方 ip 地址]
```

如,

```
scp loitor.bag myfriend@192.168.10.14
```

```
[command] | while read line
do
```

```
...
```

```
done
```

command 命令的输出作为 read 循环的输入，通过输入重定向，把第一行的所有内容复制给变量 line，循环体内的命令一般包含对变量 line 的处理；然后循环处理第二行、第三行...直到最后一行。

Linux 命令行代码较多时，可以写成脚本文件，方便移植和管理。比如建立脚本文件 test.sh，从当前文件夹下的诸多文件中利用正则表达式找出 ROS 相关的论文：

```
grep -r " ROS" .
```

输出为：

```
匹配到二进制文件 ./2452.pdf
```

```
匹配到二进制文件 ./2460.pdf
```

```
匹配到二进制文件 ./1164.pdf
```

```
匹配到二进制文件 ./1233.pdf
```

```
匹配到二进制文件 ./2527.pdf
```

通过 cut 命令只保留匹配文件的路径：

```
grep -r " ROS" . | cut -f 2 -d " "
```

这时可以通过 while read line 命令将匹配到的文件拷贝到目标文件夹下：

```
grep -r " ROS" . | cut -f 2 -d " " | while read line
do
```

```
    cp $line /home/maroon/chosen_papers/
```

```
done
```

可以写成脚本文件 search.sh:

```
#!/bin/bash    表示本脚本由/bin/路径下的 bash 程序来执行
```

```
grep -r " ROS" . | cut -f 2 -d " " > file
```

```
cat file | while read line
```

```
do
```

```
    cp $line /home/maroon/chosen_papers/
```

```
done
```

运行脚本：

```
sudo chmod +x search.sh
```

```
./search.sh
```

统计文件夹下文件中数字的个数：

```
grep -ro "[0-9]" . | cut -d ':' -f 1 | uniq -dc > file
```

Docker

Docker 背后的想法是创建软件程序可移植的轻量容器，让其可以在任何安装了 Docker 的机器上运行，而不用担心底层操作系统。

Docker 的三个概念

1. 镜像 (Image)：类似于虚拟机的镜像，是一个包含文件系统的面向 Docker 引擎的只读模板。为程序的运行提供环境，如 Ubuntu 镜像就是包含 Ubuntu 操作系统环境的模板。

镜像基本操作：

`docker search centos` #查看 centos 镜像是否存在

`docker pull centos` #利用 pull 命令获取镜像

`docker images` #查看当前系统中的 images 信息

2. 容器 (Container)：容器是镜像创建的应用实例，可以创建、启动、停止、删除容器，各个容器之间相互隔离，互不影响。Docker 引擎利用容器来运行、隔离各个应用。镜像本身是只读的，容器从镜像启动时，Docker 在镜像上层创建了一个可写层，镜像本身不变。

容器基本操作：

`docker run -it centos:latest /bin/bash` #启动，这里-it 是两个参数：-i 和-t。前者表示打开并保持 stdout，后者表示分配一个终端 (pseudo-tty)。此时如果使用 exit 退出，则容器的状态处于 Exit，而不是后台运行。如果想让容器在后台一直运行，而不是停止，可以使用快捷键 ctrl+p ctrl+q 退出，此时容器的状态为 Up。除了这两个参数之外，run 命令还有很多其他参数。其中比较有用的是-d 后台运行。

`docker attach container_name[or container_id]` #进入后台的容器

`docker stop container_name[or container_id]` #停止正在运行的容器

`docker rm container_name[or container_id]` #删除容器 (需先停止运行)

`docker rmi [image_id]` #删除镜像 (需先删除容器)

`docker ps` #查看当前运行的所有容器

`docker ps -a` #查看容器的运行记录

`docker inspect container_name` #查看容器信息

`docker commit -m "introduction" -a "user_information" 72f1a8a0e394`

`user_name/image_name:tag` #将容器转化为镜像，-m 指定说明信息，-a 指定用户信息。72f1a8a0e394 代表容器的 id；xianhu/centos:git 指定目标镜像的用户名、仓库名和 tag 信息。

3. 仓库 (Repository)：类似于代码仓库，是 Docker 用来集中存放镜像文件的地方。与注册服务器 (Registry) 的区别：注册服务器是存放仓库的地方，而仓库是存放镜像的地方，一般每个仓库存放一个镜像，每个镜像利用 tag 进行区分，比如 Ubuntu 仓库存放多个版本 (如 14.04、16.04) 的 Ubuntu 镜像。

仓库基本操作：

`docker login` #登录 DockerHub，输入用户名、密码

`docker push user_name/iamge_name:tag_name` #上传镜像

`docker pull user_name/iamge_name:tag_name` #获取上传的镜像

再一次回顾一下三个重要的概念：镜像、容器、仓库：

从仓库 (一般为 DockerHub) 下载 (pull) 一个镜像，Docker 执行 run 方法得到一个容器，用户在容器里执行各种操作。Docker 执行 commit 方法将一个容器转化为镜像。Docker 利用 login、push 等命令将本地镜像推送 (push) 到仓库。其他机器或服务上就可以使用该镜像去生成容器，进而运行相应的应用程序了。

容器中的数据管理：可以解决运行在计算机上容器的命令历史、不同容器间的数据共享等问题。

1. 数据卷 (Data Volumes)

类似于 Linux 中的 mount 概念，建立容器时可以一并建立数据卷，并且能够挂载一个主机目录为数据卷。如下命令创建名为 mysql 的容器，并添加自定义的数据卷：

`dcoker pull mysql`

`docker run -d -p 3307:3306 --name mysql01 -e MYSQL_ROOT_PASSWORD=123456 -v /root/mysqldata:/var/lib/mysql mysql`

-p 3306:3306 是将本机的 3307 端口映射到容器的 3306 端口。-v 参数添加自定义数据卷"/root/mysqldata"，并将本机的该目录挂载到由镜像自建的用于存放 mysql 数据的数据卷"/var/lib/mysql"。

除了可以挂载本机目录外，也可以挂载本机文件，如：

`docker run --rm -it -v ~/.bash_history:/root/.bash_history ubuntu /bin/bash`

这里将本机的.bash_history 文件挂载到容器的.bash_history 文件上。根据数据卷的特性，这样在本机就能看到容器中的操作历史了。

`docker rm -v mysql01` #删除容器时连带删除数据卷

2. 数据卷容器 (Data Volumes Container)

数据卷容器是一个专门提供数据卷以供其他容器挂载的容器，可以实现不同容器间的数据共享。

`docker run -d -p 3306:3306 --name store -v /data/ -e MYSQL_ROOT_PASSWORD=123456 mysql` #建立数据卷容器，用来存放数据，建立数据卷容器后，可以使用`--volumes-from`选项将其他容器挂载到`/data/`目录下，建立这样的两个容器，命名为`u1`、`u2`：

`docker run -d -p 3307:3306 --name u1 --volumes-from store -e MYSQL_ROOT_PASSWORD=123456 mysql`

`docker run -d -p 3308:3306 --name u2 --volumes-from store -e MYSQL_ROOT_PASSWORD=123456 mysql`

此时容器`u1`和`u2`中都有`/data/`目录，并且和容器`store`中的该目录共用。即当我们在容器`u1`中新建、更改、删除文件时，容器`u2`同样能得到对应的操作。同时这里的数据卷容器`store`并不需要一直处于运行的状态。

`docker rm -v store` #删除数据卷容器`store`

参考：

<https://zhuanlan.zhihu.com/p/23599229>

<https://zhuanlan.zhihu.com/p/23630443>

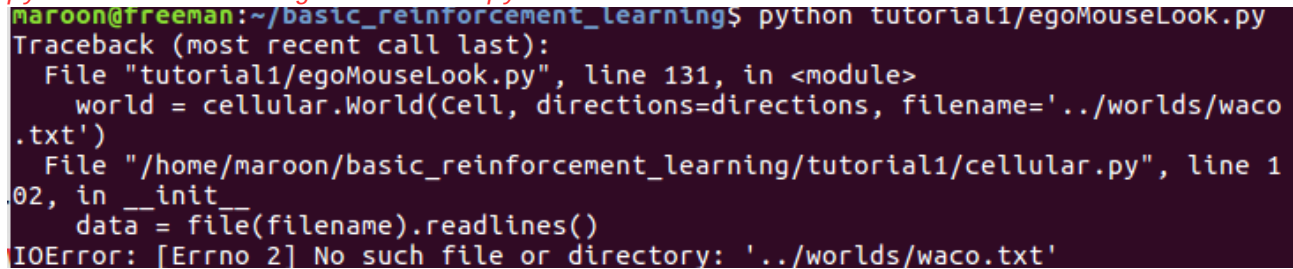
<https://docker-curriculum.com/>

Linux 终端执行 python 文件，文件中的相对位置是相对于当前位置。如：

`git clone https://github.com/vmayoral/basic_reinforcement_learning`

`cd basic_reinforcement_learning`

`python tutorial1/egoMouseLook.py`



```
maroon@freeman:~/basic_reinforcement_learning$ python tutorial1/egoMouseLook.py
Traceback (most recent call last):
  File "tutorial1/egoMouseLook.py", line 131, in <module>
    world = cellular.World(Cell, directions=directions, filename='../worlds/waco
.txt')
  File "/home/maroon/basic_reinforcement_learning/tutorial1/cellular.py", line 1
02, in __init__
    data = file(filename).readlines()
IOError: [Errno 2] No such file or directory: '../worlds/waco.txt'
```

报错提示无法找到位于`basic_reinforcement_learning/worlds/waco.txt`的文件，因为当前位置为`basic_reinforcement_learning`，执行下面命令则没问题：

`cd ~/basic_reinforcement_learning/tutorials`

`python egoMouseLook.py`

Linux Python 编程 IDE: pycharm

下载后解压到`/opt/`文件夹：

`sudo tar xzf pycharm-professional-2017.3.tar.gz -C /opt/`

`cd /opt/pycharm-2017.3/bin`

运行：

`./pycharm.sh`

添加到环境变量：

`echo "PATH=$PATH:/opt/pycharm-2017.3/bin" >> ~/.bashrc`

打开新的终端，直接运行：

`pycharm.sh`

同样，对于 C/C++ 编程 IDE: CLion

解压、添加环境变量后直接：

`clion.sh`

Windows 和 Linux 时间紊乱：

`sudo timedatectl set-local-rtc 1`

python2 和 python3 切换:

```
alias python=python3
python 进入 python3
alias python=python2
python 进入 python2
```

录屏软件:

安装:

```
sudo add-apt-repository ppa:peek-developers/stable
sudo apt update
sudo apt install peek
```

安装好后直接输入下面的命令运行程序:

```
peek
```

命令行处理命名有空格的文件, 如解压 Matlab 2016b Linux64 Crack.rar

```
rar x Matlab\ 2016b\ Linux64\ Crack.rar
```

设置安装包的环境变量

set the environmental variable PKG_CONFIG_PATH as:

```
export PKG_CONFIG_PATH=$PKG_CONFIG_PATH:$CODYCO_SUPERBUILD_ROOT/build/install/lib/pkgconfig/
```

ln 命令用来为文件创件连接, 连接类型分为硬连接和符号连接两种, 默认的连接类型是硬连接。如果要创建符号连接必须使用 "-s" 选项。

硬连接和软连接的异同: 从使用的角度讲, 两者没有任何区别, 都与正常的文件访问方式一样, 支持读写, 如果是可执行文件的话也可以直接执行。不同的是, 硬链接: 与普通文件没什么不同, inode 都指向同一个文件在硬盘中的区块。软链接: 保存了其代表的文件的绝对路径, 是另外一种文件, 在硬盘上有独立的区块, 访问时替换自身路径。例:

创建原始文件:

```
touch myfile && echo "This is a plain text file." > myfile
```

创建硬连接:

```
ln myfile hard
```

创建软连接:

```
ln -s myfile soft
```

此时改变 myfile hard soft 中的任一文件, 访问其他文件也会跟着变化。区别在于当删除原文件 myfile, 由于 hard 是硬连接文件, 其内容仍在, 而 soft 是通过 myfile 的绝对路径来访问文件内容的, 因此当访问内容时会失败:

```
rm myfile
```

```
cat hard
```

```
cat soft
```

上面第二个命令输出: This is a plain text file.

第三个命令输出: cat: soft: 没有那个文件或目录

切换 root 用户后执行安装脚本仍然提示权限不够:

切换 root 用户, 环境不变:

```
su
```

执行脚本文件:

```
installSDK64.sh
```

提示权限不够。

原因是脚本文件不具有可执行权限, 即先需给脚本文件可执行权限:

```
chmod +x installSDK64.sh
```

然后执行脚本文件, 成功

文本替换

```
sed -i s/"str1"/"str2"/g `grep "str1" -rl --include="*.[ch]" ./`
```

将当前目录下的所有 .c、.h 文件中的 str1 字符串替换为 str2 字符串。

参数解释:

sed:

-i 表示操作的是文件, ``括起来的 grep 命令, 表示将 grep 命令的结果作为操作文件

s/"str1"/"str2"/表示查找 str1 并替换为 str2, 后面跟 g 表示一行中有多个 str1 的时候, 都替换, 而不是仅替换第一个

grep:

-r 表示查找当前目录以及所有子目录

-l 表示仅列出符合条件的文件名, 传给 sed 命令做替换操作

--include="*.[ch]" 表示仅查找 .c、.h 文件

文件查找

将当前目录子目录下的 .STL 文件全部复制到当前文件夹下

首先查找文件：

```
find . -type f -name "*.STL"
```

然后通过管道复制到当前文件夹：

```
find . -type f -name "*.STL" | xargs -i cp {} .  
-i 参数是 xargs 命令的“替换字符串”选项，大括号的地方就是替换点
```

Ubuntu 内核自动升级

默认情况下，Ubuntu 的内核会自动升级，当升级到较高内核时，会出现进不去系统的情况。这时候应当在系统进入界面选择“Ubuntu 高级选项”进入低内核的 Ubuntu。那么如何删除已经升级的较高内核呢？

列出当前系统安装内核的 image 和 extra 文件：

```
dpkg --get-selections | grep linux-image
```

还可以查看安装内核的 headers 文件：

```
dpkg --get-selections | grep linux-headers
```

选择要删除版本的内核名，如 linux-image-4.4.0-97：

```
sudo apt-get purge linux-image-4.4.0-97
```

```
sudo autoremove
```

Anaconda – The most Popular Python Data Science Platform

Anaconda 附带了一大批常用数据科学包，它附带了 conda、Python 和 150 多个科学包及其依赖项。因此可以立即开始处理数据。Anaconda 是在 conda（一个包管理器和环境管理器）上发展出来的。在数据分析中，会用到很多第三方的包，而 conda（包管理器）可以很好的帮助你在计算机上安装和管理这些包，包括安装、卸载和更新包。同时还有管理环境的作用。

1、包管理

安装包，如 pandas：

```
conda install pandas
```

指定安装版本：

```
conda install pandas=1.10
```

搜索包：

```
conda search pandas
```

卸载包：

```
conda remove pandas
```

更新包：

```
conda update pandas
```

更新所有包：

```
conda update --all
```

列出已安装包：

```
conda list
```

列出已建立的环境：

```
conda env list
```

列出的当前环境旁会标有*号，默认的环境为 root

2、环境管理

安装 nb_conda 用于 notebook 自动关联 nb_conda 环境：

```
conda install nb_conda
```

创建环境，命名为 py3，指定安装 python3，同时预安装 pandas：

```
conda create -n py3 python=3 pandas
```

进入环境：

```
source activate py3
```

进入环境后可以使用之前的包管理命令查看、安装、更新、卸载包。

离开环境：

```
source deactivate
```

共享环境：

```
conda env export > environment.yaml OR
```

```
pip freeze > environment.txt
```

会将环境中所有包的名称、版本列出，然后其他人直接可以使用该文件更新自己的环境：

```
conda env update -f=/path/to/environment.yaml OR
```

```
pip install -r /path/to/environment.txt
```

删除环境：

```
conda env remove -n py3
```

安装 opencv python 版本，在 python 和 python3 下都可使用

下载源码：

```
cd ~
```

```
git clone https://github.com/Itseez/opencv.git
```

```
cd opencv
```

```
git checkout 3.0.0
```

```
cd ~
```

```
git clone https://github.com/Itseez/opencv_contrib.git
```

```
cd opencv_contrib
git checkout 3.0.0
```

编译:

```
cd ~/opencv
mkdir build
cd build
cmake -D CMAKE_BUILD_TYPE=RELEASE \
      -D CMAKE_INSTALL_PREFIX=/usr/local \
      -D INSTALL_C_EXAMPLES=ON \
      -D INSTALL_PYTHON_EXAMPLES=ON \
      -D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib/modules \
      -D BUILD_EXAMPLES=ON ..
```

```
make -j4
```

安装:

```
sudo make install
sudo ldconfig
```

Sym-link OpenCV 3: change the name from cv2.cpython-34m.so to cv2.so – this is so Python can import our OpenCV bindings using the name cv2

```
sudo ln -s /usr/local/lib/python3.5/dist-packages/cv2.cpython-35m-x86_64-linux-gnu.so
cv2.so
sudo ln -s cv2.cpython-35m-x86_64-linux-gnu.so /opt/ros/kinetic/lib/python2.7/dist-
packages/cv2.so
```

利用 pip 安装 github 上的库, 如 imgaug:

```
pip3 install git+https://github.com/aleju/imgaug
```

安装 pycocotools:

```
git clone https://github.com/pdollar/coco.git
cd coco/PythonAPI
```

修改 Makefile 文件中的 python 为 python3, 然后编译:

```
make
```

编译成功后, import pycocotools 可用, 但换个位置就会导入失败, 这时候需要设置环境变量:

```
gedit ~/.bashrc
export PYTHONPATH=$PYTHONPATH:/home/maroon/coco/PythonAPI
```

方法二: 在文件中导入

```
import sys
sys.path.append('/home/maroon/coco/PythonAPI')
```

这样就可以在该文件中直接导入了

Ubuntu16.04 上安装 Python3.6, 并且实现 Python3.5 与 3.6 的版本切换:

[How to Install Python 3.6.1 in Ubuntu 16.04 LTS](#)