Linux Ubuntu 学习笔记

一.命令行基本的知识

1.命令行结构

command [-options] [-argument]

Linux 中提供以-为开头的命令选项，这样的命令选项一般都是单个字符的，是典型的 Unix 风格的命令。Linux 同样提供以—开头的命令选项，这样的命令的选项通称为 GUN 选项，其含义一般而言就是字面理解的单个英文字符。如上面格式那样，options 和 argument 通常是可以省略的。例如 date，uname(列出与系统相关的一些参数)。命令选项用来限定命令的具体的功能，有些命令的选型本身就可能需要一些参数。

在控制台或者终端中，通常一次只能输入一个命令。使用；来分隔命令行可以输入多个命令，也可以使用()来将多个命令组合成一个命令。注意，；和()的具体的作用是不同的，虽然有的时候执行的效果是一样的。多命令的输入和组合命令的使用，使得 shell 的功能更加的强大。如果命令比较长的时候可以使用\ 来续行输入。

2.后台进程

Linux 中，shell 通常是以前台的形式解释执行用户输入的命令的。($一般用户，#root 用户)。前台执行的缺点显而易见，需要用户等待命令执行完成。为此，Shell 提供了后台执行的机制，其形式是在需要后台执行的命令之后加上一个&，系统会提示并给出一个作业号和进程号。可以通过 fg 命令将后台作业转为前台进程继续运行。可以通过 ps 获取进程信息，也可使用 kill 命令来杀死进程。

注意如果后台进程有输出数据，其输出信息会随时出现在用户的终端屏幕上，因而可能会造成屏幕的混乱。如果用户正在使用 vi/vim,那就会干扰编辑器的正常的工作。

3.标准输入输出和标准错误输出

Linux 系统中，任何命令，包括 shell 自身，通常总是读取来自终端键盘的输入数据，这个数据来源被称之为标准输入(stdin)，其文件文件描述符为 0。命令运行的结果通常被输出到用户终端屏幕上，这个输出目的被称之为标准输出(stdout),其文件描述符为 1。如果出现问题则相应的错误信息输出到用户终端屏幕上，这个输入目的通常被称之为标准错误输出(stderr)，其文件描述符为 2。

一旦注册到系统中，系统总是为用户打开 3 个默认的文件：标准输入(键盘),标准输出(终端屏幕)，标准错误输出(用于输出错误信息的终端屏幕)。

cat 命令会将会读取来自标准输入输出的数据，并逐字逐字的显示。

4.输入/输出重定向

为了分析命令的目的，有的时候需要将命令的标准输出保存到某个文件中去，这就需要使用重定向机制。符号'>'就是表示重定向的含义。例如 ls -l > fname,将命令输入到 fname 文件中，如果文件存在，则会被清除后在重新写入，如果文件不存在，则创建一个新的文件，然后将相应的输出数据保存到文件中去。如果不想将原来文件的中的数据删除掉，可以使用“>>”将数据附加到文件的后面。>>使用的策略和>相似。

任何命令(包括 shell 本身)的标准输入输出都可以重定向，使命令直接读取某个文件的输入而不是键盘的输入。例如 wc 程序，命令形如是 wc -l < xxx.xxx。

在 linux 系统中，标准输入，标准输出和标准错误输出这个 3 个文件通常总是打开的。这三个文件包括其他打开的文件均可以做 I/O 重定项处理。I/O 重定向就是由 shell 读取或者捕获来自文件，命令，程序或者脚本中的输出，将其作为输入传递给另外的一个文件，命令，程序或者脚本。

系统会为每个打开的文件分配一个文件描述符，每个文件都有一个与之关联的描述符。文件描述符是一个数字，便于 Linux 系统跟踪打开的文件(可以简单的认为文件描述符是一个简化的文件指针)。stdin，stdout，stderr 的文件描述符是 0，1，2。下表给出各种 I/O 重定向的规定和说明。

I/O 重定向	简单说明
<filename	使用指定的文件作为标准输入(文件描述符为 0)，以便从指定的文件中接受输出参数
>filename	使用指定的文件作为标准输出(文件描述符为 1)，如果文件不存在，则创建命名的文件，如果文件存在且 noclobber 标志已经设置，将会产生错误；否则，将会清除文件中的原来所有数据内容。
>|filename	除了忽略了 noclobber 标志之外，其功能和“>frame”相同
>>filename	将指定的文件作为标准输出，如果文件存在，则把输出内容附加到文件的后面；否则，创建指定的文件
<>filename	以读写的方式打开指定的文件，并使之作为标准输入

I/O 重定向	简单说明
<<[-] fstr	以指定的标志字符串 fstr 之后的文档(称为 Here 文档)作为标准输入，从 fstr 之后最行读取数据，直到遇到第二哥 fstr（或 EOF）标志。此时，第一个 fstr 是标准输入的起始标志，第二个 fstr（或 EOF）是标准输入的结束标志，附加'-'的时候，shell 会自动的忽略后随文本行前的指表符
<&digit	使用指定的文件描述符复制一个标准输入
>&digit	使用制定的文件描述符复制一个标准输出
<&-	关闭标准输入，而"n<&-"表示关闭输入文件描述符 n
>&-	关闭标准输出，而"n>&-"表示关闭输出文件描述符 n
<&j	把标准输入重定向到文件描述符 j 表示的文件中
>&j	把标准输出重定向到文件描述符 j 表示的文件中
&>filename	把标准输出和标准输入均重定向到制定的文件中

备注：如果'<'或者'>'前面有一个数字，则表示相应的文件描述符(也就是指代的相应的文件)

使用下列的形式，可以把多个 I/O 重定向组合到一个命令中。

Command < input-file >output-file 例如 command >command.log 2>&1-将命令的任何的错误信息写到 command.log 文件中，这里将标准输出重定向到文件中。注意 I/O 重定向的顺序是非常的重要的，shell 将会根据文件描述符(文件)出现的顺序决定 I/O 重定向的关联的关系。例如 command 2>&1 1>fname 和 command 1>fname 2>&1 之间的区别是很大的，前者执行的结果是命令的标准输出和错误输出的内容均被写到了文件 fname 中；后者执行的结果是命令的标准输出会被写到标准输出（stdout），而命令的标准输出被写道给定的文件 fname 中。

使用命令和重定定向可以创建一个新的空文件。命令>fname 表明如果文件存在则清空文件。

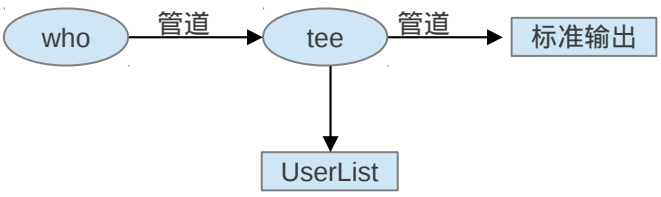
5.管道

在 linux 中管道是一种先进先出的单向数据通道。利用管道符'|'，将一个命令的标准输出连接到另一个命令的标准输入。例如，利用管道将 ls 和 wc 命令连接起来。使用管道的方式连接两个命令的时候，shell 将两个进程连接起来，将一个进程的标准输出传递到另一个进程的标准输入。shell 将会协调两个进程进行同步，使得两个程序能够并发的运行，这样可以省略存储中间处理结果的临时文件，实际上管道是一种特殊的 I/O 重定向。

管道的常见的用法是为 grep 程序提供原始数据，grep 程序来按照一定的检索的原则和模式读取来自标准输入的数据。例如：ps -ef | grep cron, who |sort.

若要依次加工处理多个命令，脚本和程序的输出数据，管道是非常的有用的。例如：ps -ef |grep cron |grep -v grep | gawk '{print $2}'

复制和备份一个完整的目录也可以利用管道，组合使用 find 和 cpio 命令。Linux 还提供一个相当于三个管道的使用程序 tee。实例：who | tee userlist



tee 命令的功能图解

6.元字符和文件名生成

Linux 中很多的命令都是以文件的作为命令的参数的。可以使用 ls 命令列出文件的各项的参数 atmmon.c 的访问权限，文件的大小及文件属主的等属性。

ls -l atomon.c

需要处理一组具有共同的属性的文件时，shell 提供了一种文件名生成机制，使用户利用元字符(或称通配符)实行模式匹配，最终生成一个具有同属性的文件的列表。文件命令生成机制在有的时候是非常的有效的，可以大量的减少文件名的输入。

元字符	简单说明
*	可以匹配任何数目的字符或字符串，包括空字符串。例如“boc*”表示任何一个以“boc”为起始的字符串，“*.c”表示任何一个以“.c”为文件名后缀的 c 程序文件
?	可以匹配相应位置的任何一个字符。例如，“file？”表示任何一个以“file”为起始字符，后面附加单个字符的字符串
[……]	由方括号定义的字符集或字符范围，可以使用其中任何一个字符匹配文件名相应位置的字符。方括号中字符集可以由任何字符组成，数量不限。字符可以一一列举，也可在两个字符之间加一个减号“-”，表示字符范围。例如，[a-z],[0-9]
[!.....]或者[^.....]	如果方括号中的第一个字符是“！”或者“^”,则其意义恰好相反，表示可以匹配任何一个不属于给定字符集范围的字符

实例：ls -l [a-z]*　　ls -l file?　　ls -l [pw]? 等价于 ls -l p* w*　　ls -l [A-Z]*
ls -l [!a-z]*or [^a-z]*　　文件检索的示例：echo /home/xxx/*/core

7.转义与引用

转义和引用是两个截然相反的概念。Shell 中的特殊含义的元字符有：‘<','>','*','?','|','&'等，使用转移符‘\‘使之作为普通的字符显示。转移字符的作用，普通字符前加上'\'后，成为具有特殊的含义的部分转移字符，而元字符加上‘\‘后，失去其特殊含义而转变为普通字符。

转义字符	简单说明	转义字符	简单说明
\a	生成声音提示	\\	反斜杠
\b	退格符	\'	单引号
\e	Esc 字符	\nnn	采用八进制数值表示等价的 ASCII 码
\f	换页符	\xHH	采用一位或者二位十六进制表示 ASCII 码
\n	换行符	\eX	Ctrl-X 字符
\r	回车符	\t	制表符

转义符号‘\’是一种引用单个特殊字符的最佳的方法，但是引用多个字符时使用‘\'就不太合适了。因此，shell 提供了第二种引用方式，采用单引号的方式引用元字符，单引号之间的所有的字符均是按普通文字本身进行的解释的。

Linux 中，某些命令的或者实用程序会重新解释或者扩展使用单引号传递的字符中的特殊字符。使用引号引用参数或者变量的一个重要的用途就是确保能把其中的特殊的字符传递给被调用的实用程序。

第三种方式是利用双引号引用字符串，防止部分(但并非全部)元字符提前解释。在此情况下，除了“！”，“$”,“‘”，“\”和“}”之外，其他元字符均按文字本身处理，如下所示。

元字符 / 引用方式	\	$	*	?	“	'	`
\	no	no	no	no	no	no	no
'	no	no	no	no	no		no
''	yes	yes	no	no		no	yes

8.命令历史

Bash 以及其他的大多数的 shell(Korn Shell，TC shell，C shell 和 Z shell 等)均支持命令历史机制，一般维护用户输入的文件。Shell 的命令历史机制和编辑功能可以让用户重复利用先前输入的命令，提高用户交互访问的能力。命令历史机制的实现主要是通过提供下面的内部命令和环境变量实现的。

a. fc:用于列出(-l 选项)，编辑(-e 选项)或者重新执行名利历史文件中记录的命令

b. history:用于列出命令历史缓冲去或者文件中记录的命令

c. HISTFILE: 用于指定命令历史文件。使 shell 能够在停止运行前把缓冲区中命令历史写入到指定的文件中，以便在下次启动时设立了能够读取其中保存的上次的执行的命令的历史记录。如果 HISTFILE 变量没有定义，或这定义的文件没有写的权限，默认的命令历史文件为$HOME/.bash_history

d. HISTSIZE:制定命令历史文件的大小，用于限定当前会话期间需要保存到命令历史文件中的命令数量。如果没有定义，文件容量的默认值为 500，及其保存最近执行的 500 条命令—注意：该数值比较的老，Ubuntu 12.10 支持的 HISTSIZE 默认为 1000，HISTSIZEFILE 默认为 2000

e. HISTFILESIZE:作用和上面的那个命令相同。

8.1 fc 命令

利用特殊的内置命令 fc，可以按照命令序号或者利用命令的起始字符(或者字符串)显示，编辑或运行先前的命令，在使用 fc 命令列举命令历史缓冲区或者文件中的命令时，可以指定当个命令也可制定一个命令范围。

实际上，shell 的命令历史机制主要由 fc 内置命令实现。Fc 的语法格式如下：

fc [-e ename] [-nlr] [first [last]]

fc -e [old = new] [command]

fc 的第一条语法格式表示用户输入的命令历史缓冲区或者文件中选项指定范围(从 first 到 last)的命令，范围的上界不能超过 HISTSIZE 变量指定的值。first 和 last 既可以用于匹配最近执行的，以给定值为起始字符串的命令，也可以是命令在命令历史缓冲区或者文件中的序号(如果在序号前加“-”,表示相对于当前命令的偏移值).如果 first 和 last 均未制定，则其默认的行为有两种：在编辑历史缓冲区或者文件的时仅选择一个命令，在显示命令缓冲区或者文件的时候（用了-l 命令），默认从前一个命令中开始回溯 16 个命令。‘-l’选项将会在标准输出中列出选择的命令；'-n'意味着在列出并执行选定的范围的命令时，省略命令历史文件记录中序号；‘-r’选项意味着由近致远的输出选定的范围的命令。执行果如下：

```
[redacted]:~$ fc -l
1       su root
2       fish
3       sudo apt-get install fish
4       fish
5       sudo nautilus
6       fish
7       fc --help
8       fc -help
9       fc -h
10      fish
11      fc -l
12      clear
13      fish
14      echo $HISTSIZE
15      echo $HISTFILE
16      echo $HISTFILESIZE
```

```
[redacted]:~$ fc -n
fc -l
2       fish
3       sudo apt-get install fish
4       fish
5       sudo nautilus
6       fish
7       fc --help
8       fc -help
9       fc -h
10      fish
11      fc -l
12      clear
13      fish
14      echo $HISTSIZE
15      echo $HISTFILE
16      echo $HISTFILESIZE
17      fc -l
xiajian@xiajian-pc:~$ echo xiajian
xiajian
xiajian@xiajian-pc:~$ fc -n
echo xiajian
xiajian
```

在第一种命令形式中，如果‘-nlr’3 个选项均未制定，则制定的或者默认的编辑器，编辑命令历史缓冲区或者文件中的命令。范围由 first 和 last 确定(如果命令没有指定，行为如上述介绍)。

'-e'选项用于定义在校正或者编辑先前的命令时使用的编辑器。如果没有指定编辑器的名字，则依次查看 FCEDIT 变量和 EDITOR 变量并将其中的值设为默认的编辑器。若 FCEDIT 和 EDITOR 都没有设置的话，使用 nano 作为默认的编辑器。在完成命令的编辑之后，退出编辑器时即可输出并重新执行刚才校正过的命令；如果编辑多个命令时，在推出编辑器时将会依次执行所选择的所有命令。

第二种命令形式表示跳过编辑阶段。如果存在‘old=new‘形式的字符串替换，由 shell 直接执行命令替换，然后重新执行编辑后的新命令。如果没有给出命令，表示执行之前刚执行的命令。

示例如下：

ls -l /etc/profile   fc -s(表示将刚才的命令再次执行一遍)

fc -s 115//表示重复执行先前的第 115 号命令

ls -l /home/gqxing/incl

fs -s incl= src //表示将 ls 命令中参数中的 incl 替换为 src 再执行一遍

8.2 history 命令

内置命令 history 是 fc 命令的一个特例（Korn Shell 中，history 只是利用了 fc 命令定义的一个命令的别名，即是 alias history= ‘fc -l’），用于读取，显示或者清除命令历史记录的命令。bash 中会显示历史缓冲区中的所有的命令，在 korn shell 中列出 16 条命令。要清除命令历史缓冲区中的命令，可以使用下列的命令。

Notice：命令历史机制仅适用于交互式 shell，不能在 shell 脚本中使用

8.3 重复执行先前的命令

在 bash 中，为了重复执行先前的命令，可以利用感叹号'!'引用机制。‘！’表示引用命令历史缓冲区或者文件中命令。若要不加修改的重复执行最近刚执行的命令，可以使用！！命令。

！string 命令表示重新执行最近执行的，以给定的 string 为起始字符串的命令。而!?string[?]则表示重新执行最近运行的，其中包含给定的字符串的命令。因此，键入！gcc 命令，将会再次执行最近输入以 gcc 为起始字符串的命令。

重复执行 uname 命令，可以使用下列的命令。

!n 表示重复执行命令历史缓冲区或者文件中的第 n 号命令，！-n 则表示重新执行最近执行的倒数第 n 个号命令。

!!:[g]s/old/new[/]命令，可以修正刚执行的名令，然后再执行。即在执行的之前，参数 g 表示替换所有配的字符。

命令	简单说明
!	表示引用命令历史缓冲区或者文件中的命令，除非后接空格，换行，等号以及‘（’
!!	重复执行先前刚执行的命令，相当于输入‘！-1’
!N	重复执行命令历史缓冲区或文件中序号为 N 的命令
!-N	重复执行重当前命令开始倒计数的第 N 个命令
!string	重复执行最近执行的，以给定的 string 为起始字符串的命令
!?string[?]	重复执行最近执行的，包含给定的 string 的命令
!!string	引用前一个命令，附加给定的字符串 string，然后重复执行组合后的命令
!Nstring	引用第 N 个命令，附加给定的字符串 string,然后重新执行的组合后的命令
!#	引用迄今为止已经输入的所有的字符
!$	引用前一个命令的最后的一个参数
!!:[g]s/old/new/	重复执行先前的命令，在执行之前，先以给定的字符串 new 替换命令中出现的第一个或者全部为 old 的字符(取决与前面是否有 g 选项)
^old^new^	重复执行先前的命令(快速替换形式)，执行之前，先以给定的字符串 new 替换，命令中出现第一个出现的字符串

8.4 编辑并执行校正后的命令

Bash 中，用户可以利用 shell 提供的命令历史机制和名令行编辑功能，使用熟悉的编辑器(vi[m]/emacs)先前输入的命令进行编辑，从而生成新的命令，然后提交给 shell 执行。调用 bash 时，如果使用了“--noediting”选项，将会关闭命令行编辑功能。

进入 Bash 之后，Shell 将按照 FCEDIT 和 EDITOR 变量，以及 emacs 的顺序默认的命令行命令行编辑器。如果要使用 vi(vim)编辑命令行，可以采用下面的方式来设置;

FCEDIT=vi[m];export FCEDIT

EDITOR=vi[m];export EDITOR

Vi(m)的模式：命令模式，普通模式，插入模式以及可视模式，其中个人觉得比较有用的是：命令模式，插入模式，普通模式。不是很清楚普通模式和插入模式的区别。就建绑定而言，有两种风格 vi 风格和 emacs 风格。

vi(命令模式)	emacs	简单说明
上箭头，减号“-”或者 k	上箭头或者 ctrl +P	获取前一个命令
下箭头，加号“+”或者 j	下箭头或者 ctrl+N	获取下一个命令
左键头，ctrl-H,或者 h	左箭头或者 ctrl+B	左移一个字符位置
右键头，空格键或者 l	右箭头或者 ctrl+F	右移一个字符位置
b	Esc+B	左移一个字
w	Esc+F	左移一个字
退格键	退格键	删除关标位置左边的一个字符
x	Delete 或者 Ctrl+D	删除关标所在位置的一个字符

8.5 命令行的补充

利用 Linux 中的 Readline 库，Bash 还支持命令行补充。使用 Tab 键进行命令的补全，这个机制可以用来查阅和检索记不清的命令。

1.命令的补充

当键入部分的命令名时，Bash 将会按照命令的检索路径，搜寻以给定的文字为起始字符串的命令。如果找不到匹配的命令，bash 将会发出鸣叫声。如果发现多个匹配的命令，在 vi(m)编辑模式里，Bash 不会输出任何的信息，而在 emacs 编辑模式中，Bash 将会发出鸣叫声。再按下 Tab 键之后，Bash 将会显示一系列其前缀与用户输入部分匹配的命令，然后允许用户采用同样的方式继续完成的命令。

2.文件名补充

文件名的补充和命令的补充的功能类似，输入起始部分后，接着按下 Tab 键，Bash 将会提供文件名的剩余部分。如果补全不唯一时，将会列出所有可能匹配的部分。当输入足够的部分时，此时按下 Tab 将会补全全部的部分。

3.变量名的补充

如同命令和文件名补充一样，Bash 也支持变量补全。命令也是相似的。

4.配置 Readline 库

使用 readline 库函数的 Bash 或者其他的程序需要读取 INPUTRC 变量所指定的文件，以获得初始化的信息。如果没有设置 INPUTRC 环境变量，这些程序将会读取$HOME/.bashrc 文件(书上说的.inputrc 没有找到,后来在/usr/share/readline 目录下看到了 inputrc 的文件目录的设置)。使用 set variable value 这样的语法格式可以设置 readline 库函数的控制变量。从而控制 readline 库函数的处理动作。

控制变量	简单说明
editing-mode(emacs)	确定使用哪个编辑器，vi or emacs,并确定 readline 的启动模式。设置这个变量的效果相当与执行 set -o [vi|emacs]
Horizontal -scroll-mode(off)	如果设置这个变量为 on，意味着超长的命令行将会延伸到屏幕显示去的右边的边界之外。默认设置为 off，在此情况下，超长的命令行将会依次延续到第二行。
completion-ignore-case(off)	补全命令和文件名时是否区分大小写，将其设置为 On 意味着，不用区分大小写。
disable-completion(off)	确定是否执行命令或者文件名称的补充，将其设置为 On 意味这禁止执行命令或者文件名的补充
expand-tilde(off)	设置为 on 的时候，shell 能够扩展‘ ~ ’
match-hidden-files(off)	是否匹配隐藏文件(以'.'为起始字符的隐藏文件)
mark-directories(on)	把该变量设置为 off 时，意味这在执行文件名补充的时候不会在目录之后附加‘/’

9.命令别名

为了照顾用户的使用的习惯和对不同操作系统的偏好，简化输入命令，提供默认的选项，Bash 以及大多数的 shell 提供了别名机制.

除了”; & (| 《 = ”以及换行，空白符，元字符，引号，变量，命令替换之外，命令别名可以由任何数量的字符或者字符串组成。

别名机制主要是由 alias 和 unalias 命令实现的。其中 alias 命令用于定义和列出用户设置的（包括系统定义的）命令别名。其语法格式的简写如下

alias xxx='chmod 755'

要用一个简单的命令别名代替一个复杂的命令，确保使用的 ls 命令总是包含一组基本的选项，

alias ll =“ls -1”

在 linux 中，如果没有特意使用 set 命令设置 noclober 特性，当使用 cp，rm 或者 mv 命令复制，删除或者重新命令文件时，如果目标文件和现有的文件同名，可能会在有意无意之间覆盖或删除同名的文件。为此，可以定义下列命令别名，以便在遇到此类情况的时候能够提示用户作出选择。

alias cp= 'cp -i '
alias mv= 'mv -i'
alias rm= 'rm -i'（注意：rm 后接=不能有空格）

在定义命令别名时，等号右边可以包含多个命令，中间由分号分隔，也可以使用管道符号，将多个命令连接起来，构成一个组合命令。

alias cf='ls|wc -w' --然后执行那个 cf 就行了 ，注意，这样的别名是有其作用域范围的，仅仅在当前的终端模拟器中有效.

unalias 命令的格式： unalias [-a] [name] 注意：不要使用-a 选项，会删除系统的定义的别名的

10.作业控制

几乎所有的 shell 都支持作业控制功能。在 Bash 中，set 命令的‘-m’或者‘-o monitor’选项用于启动 shell 的

作业控制的功能。一般而言，作业控制功能总是被启动的。

处理进程 ID 之外，shell 还会为每个每个作业分配一个数字比较小的作业号。例如，利用&符号启动后台作业，并使之异步地运行的时，shell 将会输出作业号和进程号 ID。

shell 采用作业控制表记录和跟踪当前的作业。利用 jobs 内部命令，列出系统中的所有作业，其中包括作业号，作业命令以及正在运行或暂时停止运行等状态信息。

当运行一个较长的时间的才能完成的程序，为了能够在此期间继续执行其他的任务，可以使用的 Ctrl+Z 键以及 bg 命令，把程序放到后台运行。按下 Ctrl +Z 键时会向 shell 和当前程序发送一个 STOP 信号。收到此信号之后，shell 将会输出一个表示作业已经停止的信息，接着输出另一个命令提示符。bg 命令将作业放到后台运行，fg 命令可以将作业回到前台运行，kill %number 可以将一个后台作业停止，wait %number 等待一个当前正在运行的命令。

通常而言，后台作业可以输出数据，但是不允许从终端读取数据。如果后台作业需要从终端读取数据，作业将会停止运行。使用“stty tostop”命令设置终端选项，将会禁止后台作业输出数据，因此，当遇到数据输出或者需要从终端读取数据时，后台作业也会停止运行。

引用作业的方式：

%number 使用作业号引用后台作业
%string 使用给定的字符串作为起始字符串引用作业
%?string 引用命令行含有给定字符串的作业
%% 引用当前作业
%+ 等价于%%，表示当前作业
%- 引用前一个作业

在 ubuntu Linux 系统中，用户提交的后台作业后，如果使用 exit 或 Ctrl-D 键或关闭终端窗口退出系统，系统会不加警告地立即终止后台作业的运行。使用 nohup 命令保证提出系统后作业能够继续运行，直至正常结束。在 nohup 控制下，shell 会随时监控用户提交的后台的作业及其运行的状态。如果提交的后台作业处于停止状态时而准备退出系统时用户会收到一条警告信息：“There are stopped jobs.”此时，可以使用 jobs 命令查询后台作业，决定是否要等待作业完成。nobup 命令的功能让调用的进程忽略 SIGHUP 信号，同时把进程的输出重定向到主目录下的 nohup.out 文件中。当需要运行时间很长，用户不愿意等待时，可以借助 nohup 命令。

在推出 shell 时，系统会向用户注册 shell 的所有子进程发送一个 SIGHUP 信号。通常这个信号会引起用户所有的进程终止运行。如果使用 nohup 命令，有所正在运行的进程或者后台作业，包括已经屏蔽了 SIGHUP 信号的程序，将会忽略 nohup 继续执行。

11.会话记录和命令确认

11.1 保存会话记录

Linux 提供的工具 script 工具可以记录一个用户从注册到退出系统的整个或部分会话过程，包括用户输入和系统的相应信息。该工具可以字符终端设备以及 vim 会话过程。Vim 编辑过程中可能会使用控制字符，执行诸如光标移动等操作，因而如果使用 cat 命令显示捕获的 vim 过程，难以阅读，因此使用编辑器来查阅会话过程的记录。

通常，script 命令捕获的会话过程将会记录在当前目录下的 typescript 文件中。为了使用不同的文件存储会话过程，可以使用”-a”选项。否则，script 将会覆盖原有的文件内容，删除先前的会话记录。

Script XXXtype [……各种命令] exit[注意这里 exit 仅仅是推出 script 工具的记录,不是推出终端]

11.2 确保使用的命令是正确的

1.which ：当输入一个 Linux 命令，shell 将会按照 PATH 列出的目录顺序，也就是检索路径，历次检索与之匹配的命令。如果用户改变了 PATH 变量定义的检索路径的顺序，shell 到底执行的那个目录下的命令不得而知。因此，可以使用 which 命令予以验证。which 不考虑内部命令，仅仅检索并给出第一个外部命令。

2.whereis：用于检索与给定的命令相关的文件(根据标准目录位置，而不是检索路径)。例如:下列命令可用于寻找与给定的命令相关的文件。whereis tar ：将列出 tar 命令，tar 头文件，tar 的联机文档

3.which 与 whereis 比较：which 使用的 PATH 路径及其顺序。whereis 将会采取一种独立的检索路径的方式，按照 Linux 系统中的一系列标准目录进行检索，造出相应的二进制文件，联机帮助手册和程序的源代码，以及所有与之相关的文件。

4.apropos：需要执行某个特定的某个特定的处理任务，不知道具体的命令的时候可以考虑使用 apropos。可以模糊查询。

5.whatis：执行仅能检索与给定的关键字网全匹配的命令。

二.文件系统基础知识

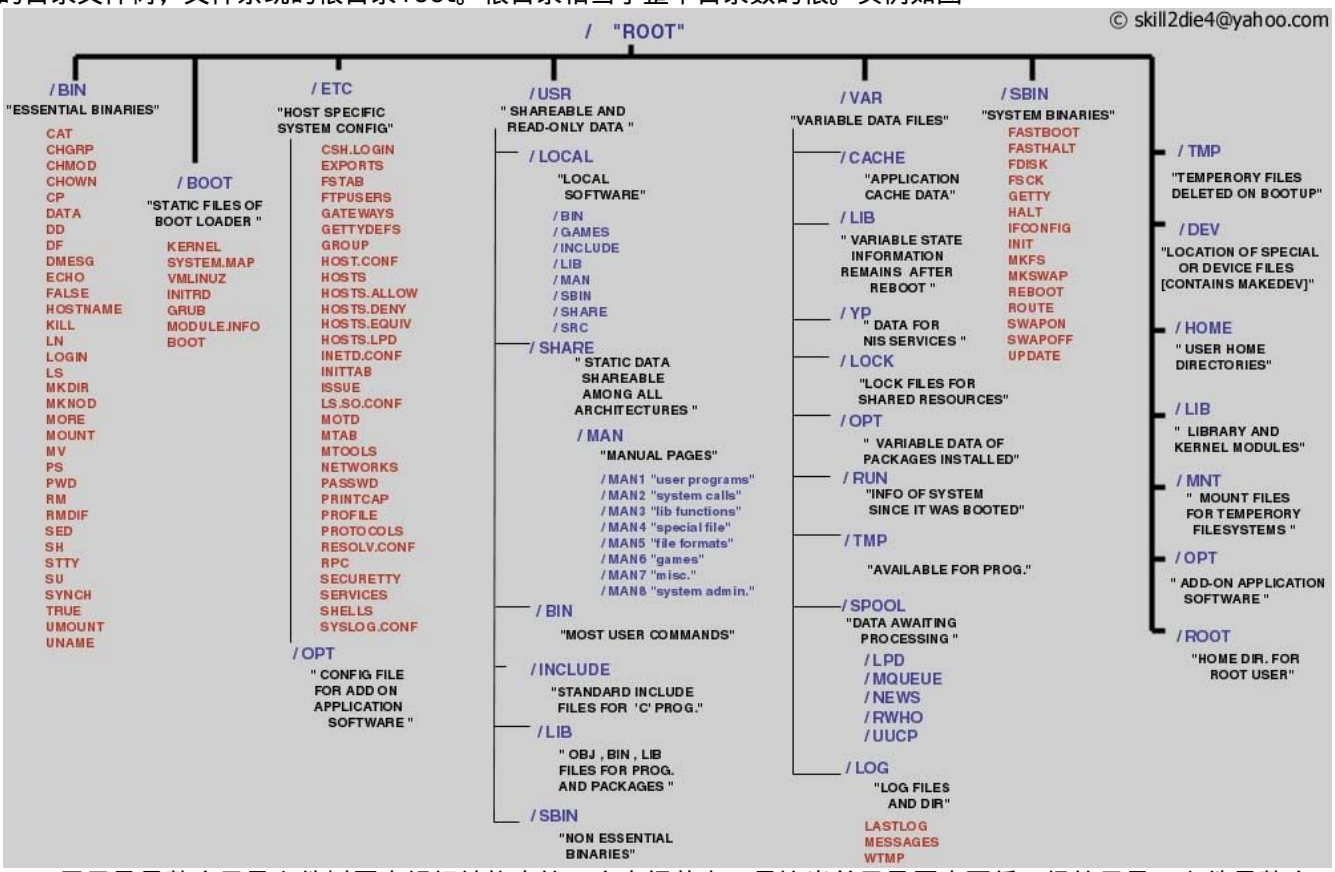
文件系统是 Linux 系统的重要的组成部分之一，承担信息的处理的组织，管理和维护等任务。文件是 Linux 系统中信息的存储，读写和执行的基本的单位。操作系统就是通过文件系统实现信息的存储，传输和加工等多种处理功能。

1.文件系统的层次结构

在 linux 中，信息的基本的组织单位称为文件。Linux 文件系统采用统一的逻辑的方法组织，存储，访问，操作，和管理信息，把文件组织在一个层次目录结构的文件系统中，每个目录包含一组相关的文件的组合。Linux 系统的一个重要的特性石提供了一种通用的文件处理的方式，简化了物理设备的访问，按文件方法处理物理设备。例如，在打印机上打印文件和在终端上显示文件的处理方式是类似的。

1.1 树形的层次结构

理论上而言，文件系统是一种逻辑组织结构。从用户的角度看，Linux 的文件系统只是一个树形层次组织结构的目录文件树，文件系统的根目录 root。根目录相当于整个目录数的根。实例如图



子目录是整个目录文件树层次组织结构中的一个中间节点，是比当前目录层次更低一级的目录。文件是整个目录树中的一个字节点。

文件系统中，若干个文件可以组成一个目录，而若干个不同的目录则可以构成一个目录的层次组织结构，而位于目录层次的结构的顶端的就是成为一个根目录的特殊目录。根目录包含了各种系统目录和文件。

操作系统中，文件的设计目的就是把文件有序的组织在一起。Linux 提供了一种便于从逻辑撒谎功能组织文件的文件系统。Linux 鼓励用户按一定的原则建立目录，源程序的目录，存储目标程序的目录以及存储文档的目录。文件系统的关键思想是其层次结构，无论系统中拥有多少个用户，每个用户都可以创建若干个目录层次以及子目录。

文件访问权限是多用户计算机文件系统的一个重要的组成部分，用于保护用户数据安全。

Linux 系统的一个重要的特性是，所有的 I/O 设备都与特殊文件联系在一起。用户可以向操作普通文件一样，操作特殊的文件，即可达访问 I/O 设备的目的。利用特殊文件，实现了用户与硬件设备的之间的通信，使用文件系统管理硬件设备的 I/O 处理。

1.2 路径名

当前工作目录中的所有文件都是可以直接存储的。但是如果要访问并非当前的文件，需要在文件名之前加上各级路径才能访问。文件访问的路径有两种：一个从当前的目录开始(相对路径)，一个是从根目录开始(绝对路径)。

每个目录中均包含”. “和”..“这两个特殊的目录文件，分别表示当前目录及其父目录。这两个特殊的目录将文件系统中的各级目录有机联结在一起。

简单的规则;

- 如果路径名以斜线字符开始，则说明路径名石从根目录开始的绝对路径，除此以外，其他的所有的路径都是相对当前目录的相对的路径名。
- 路径名要不是有斜线字符分隔的一系列名字，要不是单个名字，其中最后一个名字就是实际的文件。

其他的名字均是目录。文件可以任意类型的文件。

- 在任何一个目录中，“..“可以向上攀升文件系统的目录层次。在路径名中，除了”..“其他所有的名字均是降低目录层次。

2.文件系统的组织

Linux 文件系统中，目录和文件的组织结构是有一定规律可循的，这种结构有助于用户访问，管理和维护 Linux 系统。

/bin	系统，系统管理员，和不同用户常用的一些命令
/boot	引导程序，Linux 内核程序文件，磁盘内存映像 initrd 以及 GRUB
/dev	Linux 系统，任何一个设备对应一个或者多个特殊文件(或者称为设备文件)，这个目录下包含了系统支持的所有的设备文件。
/etc	整个系统 Linux 的核心，其中包含所用系统管理和维护方面的配置文件，例如各种***.conf。此外，还有大量的配置文件分别位于单独的子目录中，通常注意备份这个目录中的重要的配置文件以便在需要的时候恢复系统
/lib	该目录含有系统引导过程，以及运行系统命令所需要的内核模块和各种动态链接库(*.so 文件)，其中内核模块(驱动程序)位于/lib/modules/kernel-version 子目录
/lost+found	存放检测和修复损坏的文件，关联的命令有 fsck，debugs
/media	移动存储介质的安装点。一般的 GNOME 界面的安装移动存储介质时，都会自动安装在这个目录下
/opt	应用程序软件安装的目录
/proc	进程文件系统 proc 的根目录，其中的部分文件分别对应当前正在运行的进程，可用于访问当前进程的地址空间。/proc 是一个特殊的虚拟文件系统，其中不包含实际的文件，而是一些当前运行系统的系统信息，如 CPU，内存，运行时间，软件配置以及硬件配置等。读取和修改/proc/sys 目录中的适当的文件，可以显示和改变运行系统的可调参数，相当与执行的 sysctl 命令。/proc 目录中的绝大多数文件大小为 0，以数字命令的文件实际代表一个进程 ID，这些大小为 0 的文件爱你实际相当于指向内核空间数据的指针，可以用来查询相应的进程信息。因此可以将/proc 文件系统看作系统内核的一个信息控制中心
/root	超级用户 root 的主目录('/'是整个系统的根目录,不是非超级用户的主目录)
/sbin	其中包含与系统引导，管理和维护，以及与硬件配置等方面有关的命令或脚本文件，如 fdisk，init 和 ifconfig。主要供超级用户使用，普通用户通常无法使用
/srv	用于存储本地系统提供的服务所用的数据文件（现为空目录）
/sys	系统各种设备配置信息的根目录。
/temp	临时文件目录，用于存储系统运行过程中生成的临时文件，可以供用户存储自己的临时的文件。系统运行的时候，许多程序均使用这个目录存放临时数据文件，其中许多文件对于当前运行的进程是非常的重要的，删除这个文件会导致系统瘫痪，因此不要随便删除这个目录下的文件。
/usr	可以作为一个单独的文件系统，也可作为根目录下的一个子目录有这样的一些重要的子目录：bin-各种命令程序，include-各种 C 语言的头文件。lib-共享库函数，sbin-系统引导完成后的管理员常使用的命令，share-公文中文档，src-系统内核的源代码和文档等。
/var	用于存储各种可变长的数据文件（例如日志等），暂存文件或待处理的文件。核心的子目录：cache：APT 和 samba 等程序使用的工作目录。lib-软件包特定的动态链接库，配置文件，数据文件和状态信息，log:系统守护进程日志文件存放的目录 ，mail:存放用户电子邮件邮箱文件，run：系统运行信息的的根目录，其中各种.pid 文件存放的是进程号。tmp：各种临时文件

3.文件类型

逻辑上，文件是由系列连续的字节流组成，最后一个 EOF 字符结束；物理上，文件是磁盘或者其他的存储介质上的一系列的数据块组成。组成文件的数据不一定的是连续的。

Linux 可以支持不同类型的文件系统，以及不同类型的文件。这里描述，Linux 中的基本文件(驻留磁盘，用户交互)。文件是数据组织的基本单位，所有的输入输出操作都是通过文件实现的，系统处理任何设备和数据均可归结为对文件的操作。理论上讲，能够读写普通文件的程序都可以读写任何 I / O 设备。

从用途上来说，一个文件可以是如下这几种类型：

- 文档--普通的文本文件，包括脚本，程序源代码，配置数据和日志
- 命令-- 大多数的命令都是可执行的二进制的文件。
- 目录--包含文件命令列表的数据文件
- 设备—包括终端，打印机，磁盘和磁带机

Linux 中流行的 Ext2 / Ext3 文件系统，通常支持普通文件，目录文件，特殊文件，链接文件和符号链接文件以及管道文件，套接字文件。

3.1 普通文件

定义：命名的数据集合，一组信息的基本的存储的单位。Ext2 / Ext3 文件系统中，文件的命名的字符可达到 255 个字符。

普通文件可以存储任何的数据，ASCII 文本，源代码，Shell 脚本，配置数据以及各种文档，可以是二进制程序代码。普通文件自爱 Linux 系统中应用的最多。Linux 系统中文件没有记录，结构以及内容之分，所有的文件都是由一维字符流组成，并存储在磁盘等存储设备中。Linux 不刻意区分文件的类型，不以扩展命令来区分文件的类型。Linux 文件系统没有对文件的命名强加任何的约定，但是有一些约定俗称的东西。为了便于维护，Linux 也定义了若干标准的数据文件格式，二进制文件必须采用 a.out 文件格式等.

要确定文件的内容,判断文件的类型时，可以考虑使用 file 和 ls 命令。从用户的角度来看，普通文件可以分为两种类型：文本文件(只包含可打印的字符)和二进制数据文件(每个字符允许有 256 种数值)。文件系统中提供如下操作：open，create，read，write。

对于文本文件而言,可以使用 cat 命令。对于二进制文件来说，其中内容无法直接显示，可以使用 od 命令以 ACSII 字符，八进制或者 16 进制数据等形式显示(敲入命令后发现，其实不太好阅读，文件太大了)。

3.2 目录文件

目录文件简称目录(directory)，是一种特殊的文件，其存储的内容是一系列的文件名及其信息节点号。除了存储的内容不同目录文件和普通文件在文件系统中存储方式是一样的。目录用于提供文件名，信息节点与文件数据之间的关联的关系。目录的结构决定了文件系统的组织结构。

目录文件是由一系列的目录项组成的，而每个目录项有主要是由两个不同字段组成，一个字段是信息的节点，用于引用信息节点，另一个字段是文件名。每个目录项他欧冠难过信息节点实现了文件名和文件数据的映射。Linux 文件系统中包含两个特殊的目录，'.'和'..'表示当前的目录及其目录。这里普通用户只能读取目录文件，而 OS 才能写目录文件。root 用户可以里利用文件系统调试器 debugfs 等实用程序来维护文件系统。目录文件始终是由操作系统负责维护的。操作提供不允许用户修改目录文件，但是可以创建文件。

任何目录中，可以创建普通文件，管道文件，特殊文件，符号链接文件。

Linux 中每个用户都有一个属于自己的主目录。一旦注册到 Linux 系统，系统会自动的将用户引导到自己的主目录中。用户可以在主目录中做所有允许做的事情。系统访问期间，用户所在的目录称为当前目录，工作目录或者当前工作目录。Pwd 命令输出表示用户当前工作的目录。

尽管用户不能直接写目录文件，但是任何用户进程均可利用相关的目录操作的系统调用访问目录，自由的读取目录文件，只要具有相应的权限。

3.3 特殊文件

特殊文件是 Linux 系统中最具特色的特征之一。特殊文件也称为设备文件，用作为用户和 I/O 设备的之间的接口，使得用户可以像处理普通文件一样，通过打开，读写特殊文件等操作，实现访问外部设备的 I / O 处理需求。

特殊文件不包含任何数据(仅仅包含一个目录项和一个信息节点)，仅仅是提供一种机制，在文件系统中建立一个物理设备和文件名之间的映射。系统支持的每一个设备，包括内存，都与至少一个特殊文件相关连。特殊文件是利用下列 mknod 命令或者系统调用创建的，而且必须提供相应的驱动程序，并集成到系统内核中。

mknod special type [major minor]

major 为主设备号，表示按设备类型组织的设备驱动程序指针数组的索引。minor 为次设备号，表示同类设备中某个设备的子设备，可以用作调用相应驱动程序的参数。special 为特殊文件名，type 表示特殊文件的类型。c 表示字符特殊设备文件，b 表示块特殊文件，p 表示管道文件。

当提交一个读写特殊设备文件的请求时，系统会直接调用相应的设备驱动程序，设备驱动程序负责在控制进程和相关物理设备之间的传输数据。设备文件提供一个访问设备驱动程序的访问点。系统中的设备要不是块设备，要不是字符设备。

1.块特殊文件:与采用数据块组织结构和处理方式的设备(如磁盘)相关连。数据块组织结构的设备实际上是指能以固定长度的数据块传输数据，也能随机访问任何数据快的存储设备，且随机访问时间相似。

Linux'系统中，典型的数据块是由 512，1024 或者 4096 个字符组成的。因为文件系统通常是驻留在块设备中，故许多文件系统的维护命令均要求使用块特殊文件作为参数。

通过块特殊文件接口传输数据时，系统通常会在其内存(或者高速缓冲区)中缓存数据。在指定的时间间隔内。系统会将数据写入到外部存储设备，从而更新存储设备中的数据。在交互访问的或者应用程序中，用户可以利用 sync 命令和 fsync()系统调用，强制把内存中的数据写入到存储设备中，实现存储设备与内存之间的数据同步。

使用这种方法设计的一个问题是，如果内存和存储设备之间数据没有同步之前，系统崩溃了，数据完整性就

会遭到破坏。如果内存中有文件系统超级块的最新信息，那文件系统可能无法修复。

2.字符特殊文件：和非数据块组织的任何设备关联的特殊文件。字符设备不使用定长的数据块，也不能随机访问，其最底层 I/O 接口一次只能处理一个字符。

控制台终端，printer，streaming tape 和其他的串行设备均属于字符设备。字符特殊文件接口传输的数据将会在设备驱动程序和控制进程之间直接传递，之间不通过系统数据缓冲区。

3.定义特殊文件

特殊文件只是一个接口，设备的 I / O 都是由设备的驱动程序完成的。系统中每增加一个新的设备，需要完成的任务：

(1) 获取和编写相应的设备驱动程序

(2) 更新系统配置，描述新添加的设备驱动程序

(3) 重新生成新的 Linux 内核，使其包含新的设备驱动程序

` (4) 利用 mknod 命令，创建新增设备的特殊文件

4.特殊文件的实例 –4 个字符特殊文件

Linux 系统利用同一个驱动程序和 4 个不同的次设备号，支持 4 个不同的字符特殊文件，作为用户与系统内存之间的设备接口。mem kmem(无) null zero

/dev/null 数据回收站或者漏斗文件，常常用来过滤程序的输出。读取这个文件时返回的数量为 0.最常用

/dev/zero 提供任意数量的 0，写入任何数据都会消失，

/dev/mem 提供计算机物理内存的接口，是 Linux 系统的内存的映像。

/dev/mem 提供系统内核的虚拟接口。许多系统程序利用这个文件访问系统内核，获取各种系统变量的的内核地址，维护系统数据。不存在的时候可以通过如下命令创建：

```
mknod  -m 640  /dev/kmem c 1 2
chown root:kmem /dev/kmem
```

3.4 链接文件

Linux 系统提供一种机制，使得不呕吐那个的文件名医用的同一个数据或者程序，也就是把同一个数据或者程序赋予不同的文件名，此文件称为链接文件[硬链接文件]。

采用链接文件的好处：文件系统中只需要保存一份数据或者程序副本。节省空间以及保持文件存储位置的一致性。

链接文件的的特征是其引用计数一般不为 1。Linux 系统中信息节点与数据是一一对应的，如果文件的信息节点相同，则其引用的就是同一个信息节点，因此拥有同一数据副本。链接文件可以通过 ln 命令或者 link()系统调用来创建。创建硬链接文件时，文件引用的是相同的信息节点和文件数据，只是在同一个目录或者不同目录中增加一个新的文件名而已。唯一限制就是两个文件必须位于同一个物理文件系统中。

Linux 系统中，对于同一个数据内容，处于应用的需要，经常会赋予不同的文件名。

3.5 符号链接文件

Linux 提供新的文件链接机制，以便用户能够跨越不同的物理文件系统建立链接文件。符号链接文件通过 ln -s 或者 symlink()系统调用实现。

和链接文件不同的是，符号链接文件本身也是一种数据文件，而且是单独的文件，其中的数据是指向目的文件(目录)的路径名。符号号目录块拥有自己的数据块，其中包含的就是其引用的目标文件(或目录)的完整路径名。

采用符号链接的最大好处是确保文件目录文件结构的兼容性 。利用符号链接文件，还可以照顾用户以往的上机习惯，把之前常用的命令名链接到新增的命名，实现命令名字的借用或者间接引用。vi 和 vim 命令引用的同一 vim 程序。

Linux 中存在 3 个与符号链接文件有关的系统调用，其目的是访问符号链接本身以其引用的目的文件或目标相关信息：

readlink()：读取符号链接文件引用的目的文件或者目录的路径名。

lstat(): 其功能类似于 stat 系统调用，除用于获取目标文件的属性信息外，还可用与获取符号链接文件本身的有关信息。

lchown(): 类似于 chown 系统调用，即可用于修改目标的文件的属主属性。也可用于修改符号链接文件本身的属主性。

3.6 管道文本

Linux 系统存在两种管道，即普通管道(管道)和管道文件。普通管道是一个可用文件描述符标识和存取的数据缓冲区，管道内的数据按先进先出的方式处理。普通管道通常是在程序中利用 pipe(2)系统调用创建的，程序执行结束后，管道也就自动的消失了。

管道文件和普通管道功能基本相似，只是创建方式不同。管道文件是通过 mknod 命令或者 mknod()系统调用创建的，而且作为一个特殊文件，存在于文件系统中，也称管道文件为命名管道(named pipe)。

普通管道是进程间通信的机制，管道文件用于缓存接受到的数据，同时让相应的进程以 FIFO 的方式读取数据。尽管 named pipe 有文件名和信息节点，但是他不拥有任何数据。(注:p 开头文件表示管道文件)

4.文件的安全保护机制

Linux 是一个多用户，多任务的操作系统，为了保证系统和信息的安全，Linux 从信息的基本组织单位—文件出发，把用户分为三类：文件属主，同组用户和其他用户(所有用户)。任何一个文件可以针对 3 类用户分别赋予一定的访问权限。这些权限分为读，写，和执行。

访问权限	解释
r	读取的权限是指用户可以显示文件内容，复制文件。允许用户进入某个目录(cd 命令)，即具有读取目录的权限。
w	如果文件具有可写许可，则相应的用户可读，可写文件。包括显示内容，复制，修改，移动删除。对目录而言就是，写的权限就是允许用户创建新文件和删除文件。
x	可以执行的权限：用户可以运行文件。对目录而言，可以访问其中的子目录。

ls -l 文件的访问权限及其属性。

chmod 命令可以修改文件或者目录的访问权限，前提是用户必须是文件或者目录的属主(具有修改文件访问权限的能力)或者 root 用户。Chmod 命令格式如下：

chmod permissons dir-or-file ----相对权限设置法：在文件现有的基础上是进行调整

使用一个字符表示用户的类系：

u—文件属主　　g-- 同组用户　　o—其他用户　　a—表示所有用户

使用“+” 或“-”表示增加或者撤销相应的访问权限。注意：符号后可接多个字符

r-读　　w-写　　x-执行

实例：chmod a+x

创建一个新文件时，系统会自动设置其访问权限为 rw-r—r-- (参考/etc/profile)

新建一个目录的访问权限是：rwxr-xr-x

注意：这里可以使用“ * ”通配符来修改多个文件的权限。

chmod numcode dir-or-file –绝对权限设置法

core element：

r 100 4　　w 010 2　　x 001 1

4.4 其他权限设置

umask [-S] [nnn] 注意：umask 命令中的数字代码和 chmod 命令中数字代码相反。无论何时创建一个文件，Linux 都会为文件设置一个默认的访问权限。一般而言，umask 022 就是默认文件的访问权限。--权限的问题是很多运行失败的源泉

三.文件和目录操作

1.创建文件

1.touch emptyfile　　2.>emptyfile　　3.echo “only one line in file”>newfile

2.显示文件列表

ls [options] [dir-or-file]

选项	**GNU** 选项	简单说明
-a	--all	列出指定或当前目录下所有的文件，包括以‘.’的隐藏文件
-b	--escape	文件名包含不可打印特殊字符时，以八进制数字形式列出文件的名字
-d	--directory	在使用 ls 命令列举文件时，如果指定的参数是一个目录，仅列出目录下的文件。‘-d’选项经常和‘-l’选项一起使用
-h	--human-readable	以 kb,mb 和 GB 的形式显示文件的大小(应该和-l，-s 等选项一同使用)
-i	--inode	在第一列列出其节点信息
-k	--block-size=1k	以 kb 为单位给出其文件的大小
-l	--format=long	以每行一个文件的长格式列出文件的类型，访问权限，链接数，用户属主，用户组，文件大小。以及最后修改时间和文件名等信息
-r	--reverse	以文件名反向字符排序的顺序显示文件列表
-R	--recursive	递归类出目录及其子目录下的所有文件
-s	--size	显示分配给文件的数据块的数量，文件占用的数据块的数量，而非文件的实际的大小。

常见文件的格式：

d 文件夹---file directory
-普通文件--common file
l 链接--link
b 块设备文件--block device file
c 字符设备文件--character device file
p 管道文件--pipe file
s 套接字文件—socket file

ls 命令中可以使用通配符，可以显示隐藏文件

3.显示文件内容

cat 命令:cat [options] [file]

more 命令分页显示文件：more [option] [file]

- ISPACE 显示下一页，或这是显示下‘i’行(如何指定了 i 后，按下空格健)
- iENTER 显示下一行，或者显示下‘i’行(如果指定行数 i 后，接着按下 Enter 键)
- ib(^B)回显前一页，或者回显前数第‘i’页(如果指定页数 i 后，接着输入字符‘b’或者按下 Ctrl +B)
- id(^D)会显下半页(初始值为 11)，回显下‘i’行(先 i 后 d)
- i/pattern: 从当前位置开始，检索下一个或者下面第‘i’个匹配给定模式的字符串
- !/command：调用 Shell 执行指定的命令
- if ： 下一页或者第‘i+1’行
- is：跳过指定行之后，接着显示下一页
- ^L :重新回显终端框口中原有的数据内容
- v 调用默认的 vim 编辑器，编辑当前文件，并把广光标定位在当前行。退出 vim 后再返回到 more 命令的原位置
- ： n 显示下一个文件(按命令行列举的文件名顺序)
- ： p 显示前一个文件(按命令行列举的文件名顺序)
- ： f 显示当前文件的名字与行号
- = 显示当前的行号(数)
- . 重复执行前一个命令
- h(?) 给出 more 命令的简要说明
- q(Q) 推出 more 命令

less 命令分页显示文件：语法格式 less [option] [file] 参数说明同上

head 命令显示文件的前几行内容

语法格式：head [-number|-n number] [file]

number 表示需要输出的行数。默认情况下，head 命令将显示给定文件的前十行内容。

tail 命令显示文件最后几行内容

tail [+|-number [-lbcf]] [file] +表示从文件的起始位置开始计算，-表示从文件的结束位置开始计算。number 是一个数字，表示需要输出的行数。-f 选项可以监控日志文件的最新的变化。

4.复制文件

cp 的语法格式：cp [-ir] source_file target_file

-i 命令表示交互复制方式，-r 表示递归复制方式 (即是递归复制目录及其子目录中的文件)。

5.移动文件

mv 命令(mv [-fi] source_file target_file)把文件从一个目录移动到另外一个目录中，或者重命名一个文件。-f 选项为强制移动或改名，-i 选项表示交互式处理。

6.删除文件

rm [-rfi] [file]

-r 表示递归地删除目录中的文件和目录本身；i 表示交互式的删除文件；-f 表示强制删除

备注：r-recursive| recursion,i -interaction|interactive,f-force

7.目录操作

显示目录：pwd 更改目录：cd ~表示的是$HOME 环境变量

8.几个比较的有用的工具

diff 工具，比较两个类似的文件，找出其中细微差别。diff 命令会分别读取两个输入文件，逐行分析其中的异同点。diff 命令最终会告诉用户，每个文件的从哪一行开始，有几个字符不相同，同时给出来两个文件中存在差别的行。Diff3 可以比较 3 个文件

find 工具，找出具有某个特征的文件，要了解系统中是否存在某个命令，不确定在那个目录下，可以使用

find 命令。

find 命令将按用户指定的条件，从指定的目录开始，找出满足匹配准则的所有文件。检索条件可以是文件名(包括通配符)，文件大小以及文件修改日期。

find 语法：find [-H] [-L] [-P] [-D debugopts] [-Olevel] [path...] [expression]

path....:检索的起始目录(如果不给定的话，使用当前的目录)，expression 是一种表达式选项，用于指定各种匹配准则和检索条件。-H,-L,-P 使用来控制符号链接(symbolic links)，在这些选项之后的命令行参数直到地一个以'-'，'('，'!'的开头的，都被当作是文件名或者目录名。

options	Simple Description
-name filename	检索匹配指定文件名的所有的文件，如果文件名中包含通配符，需要在文件名前后加单引号或者双引号
-user username	检索文件属主匹配指定用户的所有文件，其中，username 可以是任何合法的注册用户，即使用户 ID 号
-group groupname	检索用户组属性指定用户的所有文件，groupname 是用户组 ID
-nouser	检索文件属主不在/etc/passwd 中，即非本地系统用户
-nogroup	检索用户组非在/etc/group 中，即非本地系统用户组的文件
-atime [+|-]n	a -access，n 天前后访问的文件
-ctime [+|-]n	c-change，n 天前后状态改变的文件
-mtime [+|-]n	m-modify ，n 天前后内容修改的文件
-newer filename	修改日期比给定文件更近的文件
-depth	逐层深入各级子目录，采用先文件后目录的方式，自底向上依次检索所有文件和目录
-size [+|-]n [cwbkMG]	按照指定文件大小数据 n 检索符合条件的文件。n 表示一个 512 字节数据块为单位的文件大小。[cwbkMG]表示 n 的单位
-inum n	指定信息节点的文件
-type filetype	检索特定的文件类型。f-common file，d-directory，b-block special file，c-char special file，p-pipe file，l-link file，s-socket file
-perm [+|-]mode	匹配指定访问权限(八进制或者符号表示)的文件，-表示文件必须包含 mode 定义的所有权限，+表示至少包含其中一种访问权限，没有表示完全匹配
-perm /mode	类似与 -perm +mode
-links [+|-]n	检索链接数大于，等于或者小于指定数量 n 的文件
-exec cmd {}\ [;|+]	将 find 的结果作为参数提交给命令，有给定的命令作进一步的加工处理。花括号表示给定的命令参数由 find 命令的结果予以替换。命令后必须一转义分号'\;'或'\+'结束。以加号结束的时，意味着 find 命令结果汇总为一个参数集合，然后一次性提交给定的命令，使用+可以改善性能
-ok cmd;	功能类似'-exec'选项，差别是执行给定命令前输出请求信息，用户输入 Y 后才继续进行
-empty	文件为空且是普通文件或者目录
-ls	以'ls -dils'命令的输出格式输出匹配的文件，文件大小以 1k 字节的数据块为单位，
-print	打印检索结果，输出符合检索条件的文件名

备注：1.find 命令中使用括号需要转义，可以使用逻辑运算。

2.exec 方式执行时，一方面命令参数有限制，另一方面用户可创建进程数有限。可以使用管道加 xargs 命令协助执行。

grep 工具：功能强大的文本检索工具

语法格式：grep [-inv] string file

-i-表示比较时忽略大小写，-n-表示输出检索结果的行号，-v 表示检索不包含给定字符串或者模式的所有文本。string 表示检索模式，file 表示检索的文件。

Grep 的作用：

- 过滤其他命令的输出数据
- 检索多个文件，file 使用通配符

grep 中的正则表达式：

Meta character	Match Pattern
^	文本行的行首--检索模式的首字母
$	文本行的行尾--检索模式的尾字母
.	任何一个单字符
[....]	字符集或者字符范围中任何一个字符
[^.....]	不属于字符集或者字符范围中的任何一个字符
*	零个或者多个同一字符或正则表达式
+	一个或者多个同一字符的表达式
\	转移字符

备注：这些特殊字符在 Linux 系统中也是有特殊的意义的。因此在 grep 命令中使用正则表达式时，需要通过转义机制(引号)，使系统在解释命令行时期忽略这些元字符的特殊含义。在命令提示符输入带有正则表达式的 grep 命令时，需要引号括住正则表达式。在检索模式中，需要使用转义。

实例：

grep ‘^T’ stray.birds ： 匹配以 T 为开头的文本行

ls -l |grep '^d' ： 匹配以 d 为开头的文件

grep ‘$t’ stray.brids ： 匹配以 t 为结尾的文本行

grep '^b$' stray.birds ： 匹配仅有一个字符 b 的文本行

grep 'an.' stray.birds ： 匹配以 an 为开头的三个字符的文本行

ls -l | grep ‘^.....rw’ ： 列出当前目录中其他用户能够读写的文本

grep -i -n the stray.birds：找出文件中包含字符串‘the’的文本行，且忽略大小写区别，并在输出数据钱加上行号。

检索元字符本身时需要在元字符前面转义符号‘\’。命令行中使用引号来处理元字符的转义的问题。

Sort 工具

语法格式:sort [-bdfimnru] -k key -t sepchar -o output [file]

-n:按字符串的数值而不是按文字进行排序

-r：按照从大到小的顺序或者反向字符顺序排序

-k：关键字的字段位置，或者关键字的起止字符位置或者范围

-t：指定除空格以外的其他的字段分割符

-b：仅考虑字母数字和空格，按字典顺序排序。

四.编辑文件

Vim 是一个功能极其强大的带模式的文本编辑器。

一般仅仅用到 vim 的输入模式和命令模式，且使用 Esc 作为来切换模式。

输入模式：一般按插入命令 i 进入。

命令模式：各种 vim 命令完成各种编辑功能。Vim 命令一般是一个字符，两个字符或一个选用的数字加字符组成。通常，同一字母不同的大小写不同的字符命令，其意义是相同的，作用是不同的。

大多数的 vim 命令不需要按 Enter 命令，但是以‘:’为开头的命令需要在输入命令后再按下 Enter 键。以‘：’开始的命令实际上是 ex 命令，ex 命令其实 vim 提供的另一个用户界面-命令行界面。

1.保存编辑的文件并退出 vim：

使用 vim 编辑文件期间，用户所做的任何编辑处理并没有直接反映到实际的文件中，而是保存在 vim 在内存中创建的一个文件副本中。仅在发出‘w’命令的时候，内存缓冲区的内容才能永久性的保存到磁盘上的文件中。

Command	Simple Description
:w	保存编辑后的文件内容，但不退出 vim 编辑器。这个命令的作用是把内存缓存区中的数据写到特定的文件中
:w!	强制写文件，即强制覆盖原有的文件。如果原有文件的访问权限不允许，例如原有文件是只读文件，可以使用这个命令强制写入。前提是用户是文件属主。

:wq	保存然后退出 vim 编辑器。将内存缓冲区中的数据强制写道启动 vim 时制定的文件中，然后退出 vim 编辑器。替代命令是 ZZ
:wq!	强制保存然后退出 vim 编辑器。
ZZ	使用 ZZ 命令如果文件做个编辑，则吧内存缓冲区的数据写到启动 vim 时指定的文件，否则只是退出 vim 而已。注意是大写的 Z(shift+z)
:q	在没有做编辑处理，推出时可以使用这个命令。做过编辑后不允许使用这个命令并且出现警告
:q!	强制退出 vim，放弃编辑处理的结果。
:w filename	将编辑处理的结果保存到指定的文件中保存
:w! filename	将编辑处理的结果强制保存到指定的文件中，如果文件已存在，覆盖现有的文件
:wq! finename	将编辑处理的结果强制保存到指定的文件中，如果文件存在，覆盖现有的文件，退出 vim

2.vim 编辑器的基本命令

以下是一些在命令模式下常用的一些命令。

2.1 移动光标位置(命令模式)

command	Simple Description
hkjl	功能和箭头建完全相同
-	将光标移动到上一行第一个起始字符的位置(第一个非空白字符位置)
Enter	把光标移至下一行的第一个起始字符位置(第一个非空白字符位置)
退格键	光标左移
空格键	光标右移
Ctrl+F	向下滚动一页
Ctrl+B	向上滚动一页
Ctrl+U	向下滚动半屏
Ctrl+D	向上滚动半屏
Ctrl+E	文件上移一行
Ctrl+Y	文件下移一行
H	光标移到编辑窗体顶部第一行起始字符
M	光标移到编辑窗体中间第一行起始字符
L	光标移到编辑窗体最后第一行起始字符
w|W	光标右移一个字，关键差别在于处理标点符号上
b|B	光标左移一个字，关键差别在于处理标点符号上
e|E	光标当前字的最后一个字符，差别在于分隔符上
^|0	光标移到当前行的起始位置，即是第一个非空白的字符位置
$	把光标移植当前行的行尾，既当前行的最后一个字符位置
[n]G	将光标移动到给定行的行首
(|)	句首(句尾)位置
{ | }	段首位置或者段尾位置

2.2 输入文本

i 在光标前插入　　I 在行首插入

a 在光标后插入　　A 在行末插入

o 在当前行之下新建行　　O 在当前行之上新建行

2.3 修改和替换文本

Command	Simple Description
C	替换从光标开始直到行尾的所有的数据内容，以 Ecs 键结束
cw	替换单个字。命令模式下，光标移动到相应的位置，输入 cw 输入替换后的字符，输入后按 Esc 返回命令模式
[n]cc	替换行。命令模式，光标移动到目标行的任何字符位置，输入 cc 命令。ncc 可以替换多行文本，移动到目标行的第一行
[n]s	替换字符。ns 可以替换多个字符
S	替换当前行
r	替换单个字符
R	替换多个字符。数量不限，知道按下 Esc 键结束
[n]~	转换当前位置所在的字母的大小写。一直按~会可以转换多个字母的大小写

2.4 撤销先前的修改

u	用于撤销先前执行的编辑命令，输入 u 不要按 Ecs 键
U	撤销或恢复对当前文本的所做的全部的编辑处理

2.5 删除文本

[n]x	删除字符
[n]X	删除字符，删除光标前的字符
dw	删除单个子或部分字
[n]dd	删除文本行。光标可以在文本行的任意位置处,ndd 将光标所在的那个行认定为首行
D	删除文本行的行尾部分。

2.6 复制，删除和粘贴文本

vim 中‘复制-粘贴’等价的处理过程是先 yy 再 p。yy 是复制本行，p 实施实际的复制。

‘剪切-粘贴’的等价处理先 dd 再 p。如果在 yy 或者 dd 之前加上数字可以实现整块文本行的移动和处理。

command	Simple Description
[n]yy	复制文本行，注意复制动作必须和 p(P)连用才可以
[n]Y	功能同‘yy’命令
[n]dd	删除文本行，或者将文本行放到剪切板中
p	把剪切板中的数据复制到光标所在行的下面
P	把剪切板中的数据复制到光标所在行的上面

2.7 指定数量的重复执行的命令

很多命令前加一个计数值，表明相应的命令需要重复执行的次数。例如：3dd，2dw，4x

使用句号‘.’可以重复先前执行的文本编辑的命令。这一点对于复杂的编辑命令尤其方便。

3.使用 **ex** 命令

实际上，在需要处理大块的文本的时候，相比上述的复制-粘贴与剪切-粘贴处理方式相比，利用 ex 命令可以实现更加精确的处理，更加方便的编辑。

3.1 显示行号

:set nu --显示行号

:set nonu --关闭行号的显示

3.2 多行复制

ex 的复制命令的语法格式 - :line#1,line#2 **co** line#3

line#1 和 line#2 指定需要复制的文本行的范围,line#3 插入点的行号-执行是从插入点下一行开始的复制的

句号‘.’ 表示当前行，意味着从当前哈给你开始

‘$’ 表示文件的结尾，即文件的最后一行

3.3 移动文本行

语法格式 – :line#1,line#2 **m** line#3
line#的含义和复制相同,此时也可以使用‘.’和‘$’,且含义相同

3.4 删除和替换

语法格式- :line#1,line#2 **d**
line#的含义和复制相同,此时也可以使用‘.’和‘$’,且含义相同

4.检索和替换

vim 提供了强有力的检索和替换的功能。

4.1 检索字符串

Command	**Description**
:/string	检索给定的字符串，vim 将会从当前光标位置开始检索，当找到指定的字符串时，光标移动到第一个字符串出现的位置
:?string	从当前光标开始，反向检索给定的字符串。
n	从当前位置开始，继续检索下一个匹配的字符串
N	从当前位置开始，反向检索下一个匹配的字符串
/	同 n 命令，但是需要输入/后在按一下 Enter
?	同 N 命令，但是需要输入/后在按一下 Enter
:/pattern/+n	将光标移至匹配字符串 pattern 之后的第 n 行
:?pattern?+n	将光标移至匹配字符串 pattern 之前的第 n 行

备注：特殊字符(/,&,!,.^,*,$,\,?)具有一定的意义，使用的时候必须要加'\'转义

4.2 模式匹配(:/)

模式匹配使检索更加的精确，有效。

Command	Description
:/^search	检索在仅仅出现在行首的字符串，在字符串前加上一个‘^’。
:/search$	检索在仅仅出现在行谓的字符串，在字符串后加上一个‘$’。
:/\<search\>	匹配单个字起始部分的字符串.
使用'.','*'和'[…]'	通配符匹配任意的字符。

4.3 替换字符串

字符串替换是在字符串检索的基础上实现的。因此都可以使用通配符。

语法格式

：[g]/search-string/s//replace-string/[g][c]

其中，第一个 g 表示全文检索，s 表示替换，第二 g 表示替换所有的字符串，c 表示替换前要用户确认。

Ctrl +c 可以终止这种交互式的行为。

5.编辑多个文件

5.1 编辑多个文件

vim file1 file2

编辑 file1 结束后，输入：w。此时要编辑 file2 可以使用：n or ： n file2 退出编辑器的命令同上。

在编辑多个文件期间，可以随机使用“:e filename” or “:n filename”命令，直接跳转，也可以使用“:n”命令来跳转到下一个文件，使用“:#”命令交替编辑最近处理过的两个文件。

5.2 文件合并和文本行合并

vim 的 r 命令方便把指定的文件读入当前光标所在的位置，语法格式 – :line# r filename

6.定制 vim 编辑器运行时

6.1 临时设定 vim 运行环境

通常 vim 编辑器采用一系列的默认的选项定义作为自己的运行环境。特定的需求需要改变部分选项的默认值。

Vim 编辑器支持的部分选项

选项	缩写	默认值	简单说明
all	无	无	在编辑窗口中列出编辑器支持的所有的选项
magic	无	magic	在检索字符串时，下列字符默认情况下具有特殊的意义

			• ‘.’表示匹配任何一个字符 • ‘[....]’ 表示匹配指定字符集合的或者字符范围中的任何一个字符 • ‘*’表示匹配任何字符或者字符串 在设置了 nomagic 之后，上述这这些字符将失去特殊的意义。此时可以使用 magic 恢复。注意，‘^’和‘$’不受影响
autoindent	ai	noai	这个选项应和 shiftwidth 选项一起发挥作用，使新输入的文本与上一行起始位置自动对其。Ctrl+T 前进光标，Ctrl+D 反向移动光标
autowrite	aw	noaw	编辑多个文件时，如果设置了 autowrite，当使用:e 或者:n 命令在文件间切换，vim 会自动的保存编辑的内容。否则会提示用户
ignorecase	ic	noic	在 vim 编辑器中，字符或者字符串的匹配检索通常严格区分大小写的。设置了这个选项，可以使 vim 在匹配检索过程中忽略大小写。
list	无	nolist	同常，vim 将会按正常的方式显示文件文件，如将制表符解释为适当的空格，不会显回车字符等。如果设置了这选项，vim 将制表显示为'^I',在每一个文本行后部后部附加一个‘$’
laststatus	ls	ls=1	这个选项用于确定是否在编辑窗口中显示状态行。状态行包括当前文件名，'+'标识，和光标位置等，选项值有 0，1，2.0 表示关闭，1 表示存在两个以上编辑窗口时显示，2 总是显示
number	nu	nonu	编辑文本显示行号。
readonly	无	noreadonly	对正在编辑的文件启用写保护机制，当编辑文件，vim 会向用户发出警告，以避免修改或损害重要的文件
report	无	report=2	确定在删除或者复制多少文件行是需要在状态行中显示影响的文件信息。当删除或者复制较少的文本行，少于指定值时，不会显示任何信息。默认值为 2
scroll	scr	scr=nn	这个选项用于控制 Ctrl+D 或者 Ctrl+U 健，前滚或者后滚多少文本行，默认值设置为窗口行数的一半。
shell	sh	sh=path	在使用 vim 编辑时，用户可以临时或长久地调用一个 shell，运行单个 Linux 命令，这个选项用于确定调用哪个 shell，默认情况下设置为用户注册 shell
shiftwith	sw	sw=8	设置制表符的跳转位置，以便在输入模式下，当按下制表符 Tab，Ctrl-T 或者 Ctrl+D 健时，光标能够自动地跳转下一个或者前一个指标符
showmatch	sm	nosm	输入右圆括号，花括号或者方括号时提示相应的左圆括号，花括号或方括号。
showmode	smd	smd	根据 vim 编辑器当前所处的工作模式，在编辑窗口左下角显示输入模式“--INPUT--”以及替换模式“--REPLACE--”
tabstop=8	ts	ts=8	设置制表键的右移距离，默认值为 8 个空格
wrap	无	wrap	控制 vim 编辑器如何显示比较长的文本行。为了使 vim 能够吧较长的文本行延续到下一行，可以利用这个选项实现自动折行。
wrapmargin	wm	wm=0	指定编辑窗口右边距。
wrapscan	ws	wrapscan	这个选项影响字符串检索的方式。如设置了这个选项，从当前检索到文件尾部后，会从头开始继续检索。否则将会停止检索
compatible	cp	cp	除非.vimrc 文件存在，默认情况下，vim 会尝试采用和 vi 兼容

			的工作方式。

为了加快数据输入的速度，还可以使用下列命令形式，定义用户自己常用的缩写形式。

ab abbr string

6.2 永久定制 vim 的运行环境

使用 bash 可以在.bash_profile 中增加下列形式的变量设置：

export VIMINIT ='set para1 para2'

或者将常用的 vim 的设置及其定义加到系统范围的初始化文件：/etc/vim/vimrc 文件,每次调用 vim 的时候都会调用这个文件，从而使得定制的选项成为 vim 永久性的运行环境。

7.其他特殊说明

7.1 删除和替换特殊字符

在编辑文件时，有时会遇到文件中含有不可打印的特殊字符。例如:windows 和 Linux 系统使用的行终止符不同，对于 Linux 系统而言，取自 DOS 或者 windows 系统中的文本文件会包含多余的回车字符。

为了替换和删除这种特殊的字符，可以在 vim 的删除或者替换命令中，使用 Ctrl+M 输入特殊字符的 ASCII 编码，以删除或替换特殊字符。命令如下：

1,$ s/^M// --^M 是通过先输入 Ctrl+V 然后 Ctrl+M

7.2 编辑时运行 Linux 命令

有时候，需要在编辑过程中运行 Linux 命令。例如，如果需要把某个命令的输出作为文件的一部分。利用如下的命令，临时或者长时间地运行其他的 Linux 命令，或者将 Linux 命令的运行结果引入到正在编辑的文件中。

辅助编辑命令

命令	命 令 描 述
:sh	在编辑文件期间，如果想长时间的运行 shell 命令，然后再返回 vim 编辑器，可输入":sh"命令。此时，用户将处于正常的 shell 命令工作方式，当使用 Ctrl+D 键或者 exit 命令退出 shell 时，即可返回原来的 vim 编辑器，继续进行其他编辑处理
:!command	在编辑文件期间，如果想临时运行某个 shell 命令，然后继续进行编辑处理，可输入"!command"命令。此时，vim 将会在不退出编辑器的情况下，执行指定的 shell 命令。在命令运行结束之后，按下 Enter 键，即可回复原来的编辑处理状态
:!!command	在编辑期间，如果想把某个命令直接加到当前的编辑文本中，使用命令的输出代替当前行，然后继续进行编辑处理，可输入":!!command"

五.shell 基础知识

shell 编程涉及内容广泛，若要深入，需要对 Linux 系统深入理解。学习 shell 编程时，重原理，重实用。

1.shell 与 shell 编程

在 linux 系统中，shell 是用户和 Linux 系统进行联系的桥梁。根据 shell 的调用方式，可分为交互式注册 shell 和交互式非注册 shell。这两种的 shell 的初始化的方法不相同。交互式 shell 根据/etc/profile 和/etc/bash.bashrc 文件以及用户主目录下的~/.profile。交互式非注册的 shell 主要用于运行 shell 脚本，继承注册 shell 设置的运行环境，检查并使用 BASH_ENV 变量值作为初始化文件予以执行。

1.1shell 编程的必要性

Linux 提供了丰富的系统命令和各种各样的使用程序，适当组合即可满足大部分的应用需求，而不必重新编写新的程序，这就是 shell 编程最大的功用所在。Linux 中很多的系统配置，启动和管理任务都是通过 shell 脚本实现的。

shell 既是一个命令行解释器，也是一种强有力的语言。作为命令行解释器，shell 可以解释执行用户输入的 Linux 命令，实现 I / O 重定向，提供管道，元字符以及文件名生成等功能。作为语言，shell 可以执行简单的批处理，也可以使用其中的编程机制，如测试语句，条件转移和循环控制结构，执行复杂的操作，从而增加 shell 脚本的功能和灵活性。

执行 shell 程序之前，首先利用 chomd 命令修改权限。

shell 的发展：Bourne shell ,C shell -->Korn shell → Bash shell

成为专业的 Linux 系统管理人员，熟练的掌握 shell 编程的技巧是基本技能。而 Linux 系统启动的时候，就执行的/etc/init.d 目录下的一系列 shell 脚本，用以完成系统的各种配置，设置，以及启动各种系统服务。灵活地理解和运用这些 shell 脚本，对于分析系统行为是非常的重要的一项技能。

shell 是一种简单直观，可以快速实现复杂功能需求的解决方法。shell 编程的核心在于熟悉 Linux 命令，了解 shell 基本结构和规则。Shell 是一种结合剂类的语言。

Shell 编程的思想是将复杂任务分解成简单的子任务。

1.2shell 脚本

定义：利用文件编辑器，事先将一系列 Linux 命令或者可执行程序放到文件中，然后，修改文件的访问权限，使之可以像系统命令和实用程序一样执行。

shell 是一种面向过程的语言，有控制结构和变量，也有参数，使之完成一定的功能需求。简单的批处理文件加上变量，控制结构和参数传递等编程机制，shell 可以提供的功能将会异常的强大。

1.3 运行 shell 脚本

方法一：利用 sh 或者 bash 命令 + 相应的文件。这个方法貌似在我的机器下行不通。

方法二：利用 chmod 命令设置 shell 脚本文件，使得 shell 脚本具有”可执行”的访问权限。然后在命令提示符下直接输入 echoshell 脚本的文件名。

chmod 使用命令：

- chmod 755 scriptname(文件属主可以读，写和执行，其他用户只能读和执行)
- chmod a+rx scriptname(任何用户可以读和执行的权限)
- chmod u+rx scriptname(仅仅增加文件属主读和和执行的权限)
- scriptname (如果命令检索路径包含当前目录)
- path/scriptname 或./scriptname(如果命令检索路径不包含当前目录)

在测试和调试完成后，可以考虑将脚本作为一个工具。把 shell 脚本移至某个公共的用户命令目录,例如在/usr/local/bin 目录中。

1.4 退出与出口状态

当一个命令或者进程终止时，将会自动的向父进程或者 shell 返回一个出口状态。命令或者进程成功执行完毕，将会返回一个数值为 0 的出口状态。如果命令或进程在执行过程中出现异常而未正常结束，将会返回一个非零的出错代码。同样，脚本在结束运行的是也会返回一个出口状态。最后执行的一个命令决定 shell 脚本的出口的状态。可以使用 exit 命令终止 shell 脚本的运行，并返回 shell 脚本的出口状态。

在脚本中，可以利用 exit[n]命令结束脚本运行，并将向调用 shell 脚本的父进程返回一个数值为 n 的出口状态。注意 n 必须是位于 0～255 范围内的数值。0 是正常返回状态，其他的都是错误信息。依据这个返回值可以判断脚本是否正常结束。

```
#!/bin/bash
command_1
……
command_last
exit 或者 exit [$?] 或者不要这个语句
```

对于顺序执行的 shell 脚本，这种做法没有问题。对于带有控制转移恶化结构语句的 shell 脚本，终止的位置就不固定了，因此，尽量使用能够返回不同出口状态的 exit[n]语句，以便了解 shell 脚本的实际运行的结果。

任何时候任何位置(包括在命令提示符下)，都可以使用 shell 的内部变量$?给出执行前最后一条命令的出口。

1.5 调用适当的 shell 解释程序

shell 脚本起始处以'#!'开头的文本行表示文件包含一组命令，需要提交给指定的 shell 解释执行。'#!'其实就是两个字节的文件类型 magic code，表示当前表示当前文件是一个可执行的 shell 脚本。'#!'之后是命令解释程序的绝对路径名指向用于执行当前 shell 脚本的命令解释程序或者普通程序。在发现'#!'之后，系统将会采用指定的命令解释程序代替当前的 shell。bash 是 Linux 环境通用的 shell，也是默认的 shell。

如果特殊的标识行有多个，只有第一个标志行起作用。

下面演示一个特殊的 shell 脚本：

```
#!/bin/rm

echo "This line will never print on screen."

echo "And the current file will be deleted,because it is intepert by rm."

exit 1

#!/bin/more

This is a script to display this file self.

Contents of this file will be displayed

when you execute this script file.
```

1.6 位置参数

从命令行上传递给 shell 的参数，传递给函数的参数或者通过 set 命令得到的参数通常被被称作为位置参数。位置参数可以依出现的序号引用($0,$1,$2,....),故称之为位置参数。按惯例，$0 是 shell 脚本文件的名字，由调用 shell 脚本的进程设置。$1 是第一个实际参数，$2 是第二个实际参数，如此类推。注意从第 10 个位置参数开始，必须加上花括号，如${10},${11}等。特殊变量$*和$@表示所有位置参数，$#表示位置参数的总和。

利用特殊变量$#,{},!,可以容易的引用传递给 shell 脚本的最后一个参数。

```
args=$#
lastarg=${!args}
```

为了改变参数的内容，可以使用 shift 命令。shift 重新分配传递给 shell 脚本或者函数的位置参数：把参数的位置从右向左依次移动一位。整个位置参数也随之减 1，原有的$1 和$N 不复存在。注意，$0 始终不变。

利用 shift 命令可以依次处理每个位置参数。如果采用适当的循环结构和 test 语句，即可以处理任意数量的位置参数。尽管使用花括号也可以做到这一点，但是，shift 命令处理更加的灵活，方便。

编写 shell 脚本时，应该注意使用模块化的方式组织 shell 脚本。在学习 shell 时要注意多积累，多收集一些“模型”代码片段，这对将来在编写 shell 脚本是非常有用的，可以考虑建一个代码片段的储备库。例如下面展示了一个测试 shell 脚本是否提供了正确的参数。

```
if [ $# -ne $Number_of_expected_args ]
then
    echo “Usage：`basename $0` script_arguments”
    exit 1
fi
```

2.变量和变量替换

变量是每一种变成语言和脚本语言都必不可少的重要的组成部分之一。shell 中，变量可以用于字符串操作，算术操作，参数传递以及其他的运算。变量就是系统分配的一个或者一组内存位置，用以存储各种数据。

shell 变量可以由字母，数字和下划线等字符组成，但是地一个字符必须是字母或者是下划线。长度无限制。shell 不区分变量的类型，实际上，shell 仅仅支持一种类型的变量，即字符串变量。但 shell 对字符串的处理不仅仅限于字符串操作。shell 处理过程中，取决与上下文以及采用的运算符和命令工具。通常变量的首字符决定了运算的类型。

2.1 变量的分类

按用途：内部变量，本地变量，环境变量，参数变量和用户自定义变量

内部变量：为了便于 shell 编程而由 shell 设定的变量。例如表示错误类型 ERRNO 变量等

本地变量：在代码块或函数中定义，且仅在定义范围内有效的变量

环境变量：系统内核，系统命令和应用程序提供运行环境而设定的变量，常见的有 PATH，HOME，LANG 等变量。

参数变量：调用 shell 脚本或者函数是传递的变量，通常也称之为参数替换

用户定义变量：为了运行用户程序或者完成某个特定的任务而设定的普通变量或者临时变量

广义的讲,每个进程都有一个运行“环境”，这个运行环境将影响进程的执行行为。shell 也存在一组可以引用的环境变量，其中每个变量都被赋予一定的信息。就这一点而言，shell 同任何进程是一样的。

每当运行时，shell 都会创建一组自己的环境变量。更新或者添加新的环境变量将会引起 shell 更新自己的运行环境。如果在 shell 脚本中设置了环境变量，通常需要使用 export 命令予以公布，以便子进程可以继承。

Notice：环境变量的空间分配是有一定的限制的。建立太多的环境变量将会占用额外的空间，因而可能会引起问题。

2.2 变量的赋值

变量的赋值可以采用赋值运算符“=”实现，其语法格式如下。

```
variable =value
```

在 shell 中，“=”是一种通用的赋值运算符，可用与数字和字符串。变量赋值可以用于声明变量，对变量进行初始化。另外，为了是其他的 shell 脚本或者程序定义的变量，需要利用“export variable”命令，把变量导至 shell 运行环境。Notice：“=”前后不能有空格。

赋值的过程中，如果值是由多个单词组成的字符串，则需要使用单双引号，以便 shell 可以将其作为一个值处理。“variable= ”可以申明一个未初始化的变量。

另一种赋值方法是通过 read 语句，在执行过程中为变量赋值。“for variable in word-list”的形式为变量赋值。

在 shell 脚本中，适当地使用变量能够提高脚本的可读性和灵活性，确保数据的一致性，而且便于维护。

2.3 内部变量

shell 自动设置的内部变量

Variable	**Description**

#	$#是命令行参数或者位置参数的数量
-	$- 变量是传递给 shell 脚本的执行标识
?	$?变量是最近一次执行的命令或者 shell 脚本的出口状态
$	$$变量是 shell 脚本的进程 ID，shell 脚本经常使用$$变量组织临时文件名，确保文件名的一致性
_	$_是一个特殊变量，在 shell 开始运行时，变量的初始值为 shell 或其 shell 脚本的绝对路径名，之后就是最近执行的命令的最后一个选项或者参数。
!	$!变量的值是最近运行的一个后台的 PID
*	$*表示所有的位置的参数，其值是所有位置的参数的值，每个参数均可看作一个单独的字。应用的时候，$*相当与$1，$2...表示多个参数;而“$*”相当与“$1$2.....”，表示多个参数。
@	$@类似$*,唯一不同的是，“$@”相当于“$1”,”$2”.....,表示多个参数
LINENO	shell 脚本当前执行的命令的行号。仅仅在调试 shell 脚本时，这个变量才有意义
OLDPWD	使用 cd 命令切换到新目录之前所在的工作目录
OPTARG	getopts 命令已经处理的前一个选项参数
OPTIND	getopts 命令以及处理的前一个选项参数的索引
PPID	当前进程的父进程的 PID
PWD	表示当前工作目录，其变量之等同 pwd 命令的输出
RANDOM	每次调用这个变量时，即可生成一个均与分布于 0~32767 范围的内的随机的整数，给 RANDOM 赋值相当于初始化随机数序列的目的
REPLY	使用 read 命令的时候，如果没有指定变量参数，可以把 REPLY 变量作为 read 命令默认变量，而把 read 命令读取的输入数据赋予 REPLY 变量
SECONDS	$SECONDS 变量是脚本已经运行的时间(秒数)

Shell 使用环境变量

Variable	**Description**
BASH_ENV	使用非交互式(?)的 bash 运行 shell 脚本，如果这个变量已经设置了。shell 将会使用这个变量值指定的绝对路径文件名，作为当前的环境中执行的初始化的
CDPATH	定义的 cd 命令的检索路径。
COLUMNS	用于定义终端窗口的列宽。select 命令使用这个变量值确定数据显示宽度。也用这个值来确定 shell 编辑窗口的列数
EDITOR	用于确定命令行编辑使用的编辑的程序。vi 或着 emacs
FCEDIT	fc 命令使用的默认的编辑器
HISTFILE	用于制定的命令历史记录的文件。默认文件为： ~/.bash_history
HISTFILESIZE	用于设定命令历史文件保存的最大的命令记录数量，默认为 500，且机器不同数据不同
HISTSIZE	用于设定命令历史缓冲区保存的最大的命令记录数量，默认为 500，且机器不同数据不同
HOME	当前用户目录的路径名。
IFS	字段分隔符(默认为空格，制表符和换行符)。通常用来解析命令行或者字符串的基本构成元素
INPUTRC	readline 启动文件的名字，默认值为~/.inputrc (各个机器是不相同的，我的是.bashrc)
LANG	用于语言的设置
LC_ALL	用于统一设置 LC_*系列变量的值
LC_CTYPE	用于确定系统如何处理各种语言环境的字符集，包括字符的分类，字符的大小，

LC_MESSAGES	用何种语言输出系统提示信息
LC_NUMBERIC	用于确定本地化千分数值的显示格式
LINES	用于定义终端窗口的行宽
MAIL	定义用户邮箱路径的路径文件名
MAILCHECK	指定 shell 检查邮件的频度，默认为 60 秒。如果未设置或者小于 0，shell 将停止检查邮件
MAILPATH	定义系统检查是否有新的邮件的到来的文件名
PATH	指定命令检索路径及顺序。路径之间可以使用：以及空格进行区分。Path 通常由/etc/profile 以及$HOME/.profile
PPID	父进程的 ID
PS1	第一级 shell 命令的提示符，主提示符。变量的默认的值为”[\u@\h \w]\$”,其中”\$”表示普通用户为“$”,超级用户提示符为“#”
PS2	第二级命令符，默认值为“>”.
PS3	第 3 级命令提示符。默认值为“#？”。这个变量主要用作 select 循环控制结构使用的菜单选择提示符。
PS4	第 4 级提示符。默认为“+”。这个变量主要用于 shell 脚本的调试标志。在跟踪脚本执行的过程中，shell 将会自爱显示其执行的每一个命令前，首先输出这个变量的值。默认值为“+”
SHELL	shell 命令文件的完整的路径，vi 等工具使用这个变量值作为默认的 shell。
TMOUT	用于定义用户与系统会话过程的超时值。这个值对 read 和 select 会起作用。

2.4 变量的引用与替换

使用变量的主要目的是存储或者引用其中的数据值。在大型的程序，shell 或者 shell 脚本中，经常使用变量，引用变量中的数值。引用变量中的数值称作为变量替换。在变量名字前加一个美元符“$”前缀即可引用变量的数值。引用变量的形式：

$variable

${variable}　　---这种方式是为了避免第一种方式引起的不可读性

“$variable” or “${variable}”---最保险的引用方法，可以避免意象不到的错误(尤其是变量值中包含分割符时)

当一个变量值中是由一个不包含任何字符的字段分割符的字符串或者数值组成时，上述三种引用方式完全一致，但是如果变量值使用了制表符，换行符，以及由内部变量 IFS 定义的分割字符时，使用第三种变量是应用方式才是最保险的。使用双引号引用变量时可以进行替换。单引号具有转义的作用。

在如下的情况，引用变量时下不需要在变量名前加“$”前缀:

- 申明语句
- 赋值语句
- unset，export 命令
- 引用系统定义的信号

注意变量名及其引用的数值的区别。

2.5 变量的间接引用

情况是一个变量是另一个变量的名字，利用第一个变量引用第二个变量。这就是间接引用。

语法：eval var1=\$$var2

2.6 特殊变量替换

确保 shell，shell 脚本或者应用程序能够正常运行，shell 提供了一种特殊的变量替换机制。主要的功能如下：

- 对于未设置的变量，采用特殊替换表达式赋予默认值
- 采用特殊替换表达式替换或设置默认值
- 采用特殊替换表达式给出错误提示信息

特殊变量替换	简单说明
${var}	其作用同$var,表示变量 var 的值。但是这种形式表达意义更加的明确
${var:-value}	如果变量 var 没有设置值或者值为 null，使用 value 作为变量 var 的值进行变量替换。否则，使用变量 var 的值进行变量替换，但是变量 var 的值保持不变

${var:=value}	如果变量 var 未被设置或值为 null，把 value 赋予变量 var，同时进行变量替换
${var:+value}	如果变量 var 没有设置值或者值为 null，使用 null 进行变量替换。否则，使用变量 value 值进行变量替换，但是变量 var 的值保持不变
${var:?value}	如果变量已设置，使用 var 的值进行变量替换。如果变量 var 没有设置值或者值为 null，使用 value 作为错误提示信息。如果省略了 value，则使用默认的错误输出信息，表示变量 var 没有被设置，然后终止 shell 脚本的运行，并反会一个非 0 的出口的状态。变量 var 的值保持不变。

上述表达式中，冒号意味着需要测试给定的变量 var 是否已经设置。如果变量已设置在检查是否为 null。如果省略':',则意味着只需检查 var 是否尚未被设置。

比较	var 已设置且值为非 null	var 也设置且值为 null	var 未设置
${var:-value}	var 替换	value 替换	value 替换
${var-value}	var 替换	null	value 替换
${var:=value}	var 替换	value 赋值并替换	value 赋值并替换
${var=value}	var 替换	var 替换	value 赋值
${var:?value}	var 替换	错误，退出，返回 1	错误，退出，返回 1
${var?value}	var 替换	null	错误，退出，返回 1
${var:+value}	value 替换	null	null
${var+value}	value 替换	value 替换	null

在调用 shell 脚本时，如果提供的命令行参数不足，采用上述的默认的变量或者参数替换方法很有用。

其他特殊变量	说　明
${#var}	字符串的长度(即字符串变量 var 中的字符数量)。对于数组而言，${#array}是数组中的第一个元素的长度。注意：${#*},${#@}给出位置参数的个数，${#array[*]}和${#array[@]}给出的数组元素的个数
${var#pattern} ${var##pattern}	从$var 变量值的前部删除与给定模式匹配的最短或最长部分子串。如果引用在文件路径名，#{var##pattern}相当于 basename 命令
${var%pattern} ${var%%pattern}	从$var 变量值的后部删除与给定的模式匹配的最短或最长部分的子串。其中，${var%pattern}可用于抽取路径名的目录部分

2.7 变量声明和类型定义

shell 根据变量存储的内容来确定变量的类型，但是为了加快运算，提高 shell 编程的严谨性，在 bash 中可以使用 typeset 或者 declare 命令定义变量的类型。

记住变量类型是 shell 编程人员的责任。

typeset -i 选项可以将变量声明为整型。事先声明为整数的变量可以直接的执行算术运算。而不需要使用 expr 和 let 命令。-r 可以将变量声明为只读变量。

3.命令和命令替换

3.1shell 内部命令

Linux 系统提供了大量的命令供用户使用。出于性能方面的考虑，shell 也提供了若干具有同样功能的内部命令。内部命令比外部命令快的多，因为内部命令需要创建进程来执行。主要的内部命令如下：

内部命令	简单解释
.	用于从命令行中读取 shell 脚本，并在当前的 shell 环境中执行。在 shell 脚本中使用'.'命令，可以把指定的源文件读入到当前的脚本中去，并从当前位置开始执行。当多个 shell 脚本共用同一数据文件(?)的时候，点命令非常的有效的。点命令的可以执行环境变量脚本。
:	null 语句。自身不做任何的动作，总是返回一个值为 0 的出口状态。其功能是用作无限循环语句的测试条件，二是作为一种特殊的命令形式，引起 shell 处理紧随其后的变量替换和命令替换

alias	设置或者显示系统或者是用户定义的命令别名
bg	把指定的作业置入后台运行模式。如果没有给定作业号，将当前作业后台运行.
break	用于退出 for，while，until 等循环
cd	改变目录。用于转换到指定的目录
continue	用于结束 for，while 和 until 等循环语句中的当前循环，开始下一次循环
declare	用于申明变量或定义变量属性。如果未指定变量名，declare 命令将会给出所有变量的定义。-p 选项用于显示给定变量的值和属性。下列选项用于生命变量. • -a --数组变量 • -f – 声明一个函数或显示函数定义 • -i – 把指定的变量定义为整数变量，当赋予变量值时，按算术的要求执行算术运算 • -r – 把指定的变量定义为只读变量，在随后的处理中，不能为只读变量赋值。 • -x – 公布制定的变量，以便其他命令或 shell 脚本能够继续使用
echo	用于输出字符串，变量值，表达式的计算结果等参数，通常会在输出信息之后加上一个\n
enable	启用和禁用 shell 内部命令。这个因为内部命令高于 PATH 指定的命令。
eval	用于读取并组合随后的命令参数，然后提交 shell 执行。
exec	如果命令带有参数，则使用命令参数代替当前的 shell 进程，而不是创建一个新的进程。如果没有参数仅仅是作为 I / O 重定向。注意：exec 执行会引入命令终止运行时，因而会强制 shell 脚本随之结束执行，因此一般作为最后一个命令
exit	无条件停止 shell 脚本的执行。exit 可以带一个参数，也可以不带。好的 shell 编程习惯是在脚本执行结束后加一个 exit N。EOF 也会结束脚本的执行
export	用于在导出设置的变量，使变量的设置能够用于正在运行的脚本或者 shell 的所有的子进程
fc	用于列出，编辑或者重复执行命令历史缓冲去或文件中的记录的命令
fg	把指定的作业置入前台运行模式
getopts	用于解析传递给 shell 脚本的命令行参数。使用两个变量，OPTIND 下一命令行选项指针，OPTARG 是选项有关的参数。getopts 命令通常用于 while 循环处理所有的选项和参数
hash	用于记录给定命令的路径名，以便 shell 或者 shell 脚本在调用 hash 名利命令时不必按 PATH 变量指定的目录检索命令
jobs	显示指定或者全部活动的作业，-l 显示进程 ID 号，-n 显示已停止或终止的作业
kill	用于向指定的进程或作业发送信号，从而终止相应的进程或作业。信号可以是数字代码或者信号名字
let	实现算术运算
logout	退出注册 shell
printf	按给定的控制格式输出参数的值。输出格式是一个字符串。其中包含 3 个字符对象：普通字符序列，转义字符序列，格式规范
pwd	显示当前工作目录，作用等同于读取 PWD 变量的值。`pwd`等价与 PWD 的值
read	用于从标准输入读取数据，并把数据赋值到给定的变量中。也就是说 read 命令能以交互的方式读取来自键盘的输入数据。‘-r’表示忽略转义符号‘\’
readonly	把指定变量设置为只读变量。修改只读变量会引发错误
return	引起 shell 函数返回主程序的调用点。在非 shell 函数的情况下，return 命令相当于 exit
	命令用于改变内部变量或者脚本变量的值。set 命令的一个重要的用途就是重置位置参数。如果不带参数，set 命令将会输出所有环境变量的当前设置，这是 set 的另一个重要用途。 • -a 使定义所有的变量和函数设置能自动导入 shell 运行环境

set	• -e 如果任何一个命令结束后返回非零值的出口状态，立即终止 shell 脚本执行 • -f 禁用文件名生成机制 • -m 监控模式(启动作业控制功能)，交互式 shell 中默认打开的 • -n 读取脚本命令，检查是否存在语法错误，但不执行。 • -o 使用下列参数设置各种标志 1. emacs 使用 emacs 作为命令行编辑器 2. noclobber 防止标准输出>重定向覆盖现有的同名文件。一旦打开次标志，仅当‘>|’时才能清除现有文件 3. nolog 不允许将函数定义写入命令历史文件中 4. verbose 同-v 选项 5. vi 设置 vi 作为命令行的编辑器 • -t 读入并执行任何一条命令之后立即终止 shell 脚本的运行 • -u 执行变量替换时，事先未设置的变量按错误处理 • -v 显示 shell 读入的命令语句 • -x 显示执行的命令及其参数 • - 关闭‘-x’和‘-v’标志 • – 重新设置位置参数，或清除位置参数的设置
shift	把位置参数$2,$3,...$N 依次重命名为$1,$2,...$N-1. $1 和$N 不再存在，位置参数总体相应的减 1，故$#变量值也比原来的$#变量值少 1.
shopt	用于启用和控制 shell 控制选项的开关状态，以规范 shell 的行为。如果未带任何的命令选项，或者使用了-p 选项，将显示用户能够设置所有的看看控制选项及其开关状态。Shopt 支持的命令如下： • -s 启用相相应的控制选项 • -u 禁用相应的控制选项 部分重要的 shell 控制选项： • cmdhist 设置为 on 时，bash 会尝试把多个命令作为一个命令行保存到命令历史文件中，以便用户能够容易的重新编辑多行命令，默认为 on • histappend 设置这个选项时，当退出 shell 时，bash 将会把命令历史缓冲区中命令历史记录附加到 HISTFILE 变量指定的文件中，而不是覆盖文件，默认为 off • huponexit 设置这个选项时，退出交互式 shell 时，bash 将会向所有作业发送一个 SIGHUP 信号。默认为 off • sourcepath 设置这个 shell 控制选项，‘.’或者 source 内置命令将会使用 PATH 变量指定的目录检索指定的脚本文件。默认为 on
source	类似于‘.’命令，用于读取指定 shell 脚本文件，在当前 shell 环境中运行，运行结束后返回最后执行的一个命令的出口状态。
test	计算条件表达式
tiems	显示 shell 或进程占用用户和系统的时间
trap	shell 脚本信号捕获和错误处理
type	说明 shell 如何解释制定的命令
typeset	声明 shell 变量和函数的类型属性。功能及支持的选项完全等同与 declare
ulimit	用于设置和显示用户资源的限制。资源限制的值可以是一个数字，也可以是文字值 unlimited。‘-H’和‘-S’分别表示硬限制和软限制。硬限制不可变，软限制可增加，直到达到硬限制值。 • -a 列出当前所有的资源的限制 • -c 列出内存映像转储文件(core)的最大容量限制(以 kb 为单位) • -d 列出进程可用的数据区(段)的最大容量限制(以 kb 为单位) • -e 列出可用的 nice 优先级调整值的最大值 • -f 列出可以创建的最大文件的容量限制(以 1kb 的数据块为单位)

	• -n 列出可用的文件描述符的最大数量限制 • -s 列出每个进程可用的最大栈区的容量限制(kb 为单位) • -t 列出每个进程可用的最大 CPU 时间限制(以秒为单位) • -u 列出单个用户可用的最大进程数量的限制 • -v 列出可用的虚拟内存的最大数量限制(以 1kb 为单位)
umask	指定创建文件时应设定的默认访问权限
unalias	撤销已经设定的命令别名
unset	清楚 shell 变量，把变量设置为 null。注意：这个命令不影响位置参数
wait	等待指定的作业结束，显示作业结束的出口状态。没有指定作业号，意味着等待当前作业及其所有的子进程的结束。wait %1，wait 6618

3.2 部分命令的介绍

3.2.1“.”，source 与 exec 命令

当 shell 运行一个 shell 脚本时，通常需要创建一个新的 shell 进程。新的 shell 进程将会继承其父 shell 进程的环境变量(包括全局变量和已 export 的变量)，但是不会继承未公布的本地变量。在 shell 脚本中设置的变量，无论公布与否，退出 shell 脚本，其中设置的变量也随之消失。为了利用 shell 脚本设置的变量，通常需要使用“.”或者 source 内部命令，在当前的 shell 环境中解释执行的 shell 脚本。

与‘.’或者 source 命令类型，exec 也可在当前的进程空间中执行指定命令。但是 exec 的能力更加强大，不仅仅可以运行脚本，也可以运行编译的程序和命令。而且，‘.’和 source 命令调用的 shell 脚本运行结束之后，将会返回调用点，但 exec 不会返回。此外，‘.’或 source 命令运行的 shell 脚本能够访问当前进程空间中的本地变量。

exec 的两个作用：1.运行指定的命令时无需创建新的进程，2.重定向 shell 脚本中的文件描述符。由于不创建新的进程，命令的执行的速度更快。但是，exec 不返回调用点，因此只能作为 shell 脚本中的最后执行的一个命令。

exec 可以重定向 shell 脚本中的文本描述符，包括标准输入，标准输出和标准输出。< 标准输入，>标准输出，2> 标准错误输出。

实例：exec < infile ,exec > outfile 2>errfile

上述实例不会代替当前进程，exec 命令之后依然可以附加其他的命令。

从定向文件后，为了看到 shell 脚本中输出，可以使用/dev/tty 设备文件。/dev/tty 是一个伪设备文件，表示当前用户使用的终端窗口(和键盘)。无论何时，都可以使用这个设备文件引用用户的终端窗口，而不必知道当前用户的终端的实际设备文件的命令是什么(如果需要知道的话，可以通过 tty 命令获取).

把 shell 脚本的标准输出重定向到/dev/tty,可以保证任何提示信息多可以送达用户终端窗口。shell file 中的 exec_demo2. sh 执行./exec_demo2.sh >outfile 2>errfile

使用 exec 重定向的一大优点是，不必在随后的在随后需要重定向的每个命令一一进行 I/O 重定向。

3.2.2. “:”与 true 命令

“:”与 true 命令不执行任何处理动作，作用仅仅是返回一个出口状态的为 0 的测试条件。True 是由 Linux 系统提供的,”:”是 shell 内部提供的。作用是通常是用于实现不定循环。实例如下：

```
while true
do
ls -l xxx.sh
done
```

3.2.3 echo 命令

echo 命令也许是 Linux 系统中应用最广泛的命令之一。echo 中显示各种信息，也可用于显示文件列表。作为显示信息的字符串，可以直接的写在 echo 语句后面，也可以在字符串前加引号。

为了组织信息的显示的格式，利用“-e”选项，echo 语句还支持少量的特殊字符：

- \n :在相应的位置插入一个换行符号
- \t :在相应的位置插入一个制表符
- \b :在相应的位置插入一个退格符
- \c :取消输出一个换行符。此外，利用-n 选项也能够达到同样的目的

3.2.4 read 命令

read 命令主要读取来自标准输入的数据，然后将其存储到指定的变量参数中去，实现交互式变量赋值。read 命令语法格式如下：read [options] vars

选 项	选 项 说 明

-a aname	读取每一个字段(字或字符串)，并依次赋予指定数组 aname 中的每一个元素
-d delim	使用指定的分隔符(而不是默认的换行副)作为输入数据的终止符
-e	如果输入来自键盘，在使用 Readline 库读取输入数据
-n nchars	读取定量的字符后立即返回.只要用户输入了 nchars 个字符，无需按下 Enter 建，read 命令将会立即返回。
-p prompt	首先在标准错误输出中显示一个提示字符串，然后等待用户输入，这个选项适合与从键盘中读取用户数据
-s	禁止显示输入的任何字符，可用于输入密码数据
-t timeout	用于控制输入超时。在给定的时间内没有输入数据，就会引起 read 命令超时，并返回一个错误状态信息。如果不是从用户终端或者管道读取信息，这个命令不起任何的作用。
-u fd	使用指定的整数 fd 作为文件描述符，以便 read 命令能够从指定的文件中读取数据。

3.2.5set 和 unset 命令

set 命令的一个重要的用途就是设置和重新设置位置参数的值。shell 规定，用户不能直接给位置参数赋值，但是，利用 set 命令及其参数变量，则可以修改或者重新设置位置参数的值。例子：见 retain.sh

使用 set—命令，也可以把参数变量设置为位置参数。如果“set--”命令后面未给定参数变量，意味着清除所有的位置参数。

```
var=“one two three four five”
set – $var
echo $1 $2 $3 $4 $5
echo “$@”
set  --
echo “$1”
```

如果不带任何的选项或者参数，set 命令会列出所有的环境变量，已经声明的或者设置的其他变量以及函数定义等。这个 set 的另一个重要的用途.

unset 命令用于清楚 shell 变量，把变量的值设置为 null。注意，这个命令并不影响位置参数。

3.2.6 set 命令和 shopt 命令

shell 既是一个解释程序，又是一个普通的系统命令。set 和 shopt 可以修改 shell 的默认的处理行为，从而定制自己的运行的环境。

3.2.7 expr 命令

expr 命令用于计算表达式的值，然后把计算的结果送到标准输出。其中的表达式可以是字符串比较表达式，整数算术表达式或者模式匹配表达式。语法输出格式：

```
expr  expression
```

现在 expr 基本被 test 命令或者”[[.....]]“结构取代，而且在“[[.....]]”结构中，'<'和‘>’符号之前是无需加转义符号的，所以现在很少使用 expr 命令做字符串比较(test 和“[[.....]]”不支持'>='和‘<=’)

表达式	计算结果
str1=str2	相等，计算结果为真，输出 1 返回 0，反之，结果为假，输出为 0，返回为 1
str1\>str2	大于，计算结果为真，输出 1 返回 0，反之，结果为假，输出为 0，返回为 1
str1\<str2	小于，计算结果为真，输出 1 返回 0，反之，结果为假，输出为 0，返回为 1
str1\>=str2	大于等于，计算结果为真，输出 1 返回 0，反之，结果为假，输出为 0，返回为 1
str1\<=str2	小于等于，计算结果为真，输出 1 返回 0，反之，结果为假，输出为 0，返回为 1
str1!=str2	不等于，计算结果为真，输出 1 返回 0，反之，结果为假，输出为 0，返回为 1

expr 命令用于计算整数表达式的值，将计算结果发送到标准输出。如果想要把计算结果值保存到一个变量中，需要采用命令替换的方式实现。Expr 支持：+,-,*(注意使用的时候需要加上转义符‘\’-->*),/,%.

3.2.8 let 命令和”((.....))”结构

let 命令和”((.....))”结构可来计算和测试整数算数运算表达式，执行整数算术运算。实际上，let 命令和”((.....))”结构是对 expr 命令的简化，取代并扩展了 expr 命令的整数算术元算功能。

除了上述五种基本运算，let 命令和”((.....))”结构还支持+=，-=，*=，/=，%=等运算符。

let 命令和”((.....))”结构也可以返回算术表达式计算结果的出口状态。如果算术表达式的计算结果为 0，出口状态为 1；反之，不为 0，出口状态($?)为 0。这一点和 test 语句，‘[....]’以及‘[[.....]]’恰好相反。

3.2.9 常数数值

shell 脚本一般按 10 进制数值解释字符中的数字字符，除非数字前有前缀。0-八进制，0x-16 进制。

3.3 命令替换

命令替换目的获取命令的输出，为变量赋值，或者对命令的输出做进一步的处理。实现方法：一是使用反向引号(`)引用命令，二是采用“$(....)”形式引用命令。其语法形式：

`command` or $(.....)

原则上，命令替换中的命令可以是任何命令(外部命令，shell 脚本，甚至是脚本中定义的函数).实际上，命令替换蕴含这两个过程：执行相应的命令，获取命令标准输出中的数据。在获取命令数据之后，可以使用赋值运算符(=)把数据赋予某个变量，或者做进一步处理。

命令的输出可以作为另一个命令的输入参数，甚至可以用于生成 for 循环中的参数列表。

在使用 echo 命令输出一个未加引号引用的变量，变量使用了命令替换的形式，将会抛弃命令输出数据后面的换行符，从而有可能导致不期望的后果。此时需要使用“”将变量引起来。

在命令替换过程中，使用 I / O 重定向或者 cat 命令，可以把文件全部内容赋予一个变量。实际上这个非必要的。'<'可以看作是 cat 命令的特例。

4.test 语句

编程语言都有条件测试功能，条件测试的结果将决定程序的控制流向和下一步的处理动作。Shell 通过 test 及其简化形式支持这个功能。

test 及其简化形式主要的功能是计算随后的表达式：检查文件属性，比较字符串，比较字符串表示的整数值。

语法形式：

test expression;

[expression] ; #简化但效率更高的 test 命令

[[expression]]. #比 [ ] 更通用的测试条件，扩展的 test 命令。

注意：方括号两边必须各加一个空格

在 shell 脚本中，使用“[[.....]]”测试结构，而不用“[.....]”结构，又是可以避免出现逻辑错误。在“[[...]]”测试结构中可以使用“&&”,”||”,”<”和“>”等运算符，而在“[...]”中使用就会出错。

“[[.....]]”中不允许执行文件名生成和单字解析操作。

4.1 文件测试运算符

文件测试主要是指文件状态的测试和属性的执行，如文件是否存在，文件的类型，文件的访问权限以及其他的属性。

文件的访问权限分 3 种类型：文件属主，同组用户和其他用户。这个就是使用 ll 命令显示的那一串字符串中的数据。

在 test 语句中，文件命令参数必须显式的指定，不能为空。文件名参数可以实际的文件名，也可是变量替换，命令替换或者文件名生成机制(通配符等)。如果是变量替换方式生成的，则必须使用双引号括起来。

表达式	简 单 说 明
-e file	文件存在，测试结果为真
-r file	文件存在且当前用户可读，测试为真
-w file	文件存在且当且用户可写，测试为真
-x file	文件存在且当前用户可执行，测试为真
-s file	文件存在且其大小>0,测试为真
-f file	文件存在且是个普通文件，测试为真
-d file	文件存在且是个目录，测试为真
-L file	文件存在且是符号链接，测试为真
-c file	文件存在且是字符特殊文件(比如终端)，测试为真
-b file	文件存在且是块特殊文件(比如磁盘)，测试为真
-p file	文件存在且是命名管道文件，则测试为真
-u file	文件存在且其 setuid 位置位已设置，则测试为真。setuid 为 root 用户。检查文件的 setuid 标志位通过查看访问权限是否有字符 s，setuid 会有安全隐患，对 shell 脚本而言，没有影响。
-g file	文件存在且其 setgid 位置位已设置，则测试为真。目录设置为 setgid 标志，该目录中创建的文件属于同组成员。这对于工作组共享同一目录非常有用。

表达式	简 单 说 明
-k file	文件存在且粘性标志为已经设置，则条件测试结果为真。粘性标志位是一种特殊文件的访问权限。如果某个可执行的文件设置了这个标志位，在运行的过程中，相应的程序将始终保持在高速缓存中，其执行和访问的速度更快。如果文件设置了这个标志位，则限制了用户的访问权限。查看访问权限是否有字符 t
f1 -nt f2	文件 f1 存在且修改日期比文件 f2 新，则条件测试结果为真
f1 -ot f2	文件 f1 存在且修改日期比文件 f2 早，则条件测试结果为真
f1 -ef f2	如果给定的文件 f1 和 f2 存在且指向同一个物理文件，则测试结果为真
!	逻辑非运算，当与上述的表达式一起使用的时候，测试结果与其本意相反

4.2 字符串测试运算符

表 达 式	简 单 说 明
-z str	如果给定的字符串长度为 0，则条件测试的结果为 true。‘! -z’时，字符串不加引号，结果是不可靠的
-n str	如果给定的字符串长度大于 0，则条件测试的结果为 true
s1 =s2	字符串相等为真
s1!=s2	字符串不想等为真
s1<s2	基于字符的 ASCII 编码值，如果给定的字符串 s1 小于 s2，条件测试为真，用法如下; • test s1 < s2 • [s1 \< s2] • [[s1<s2]]
s1>s2	同上。

4.2.1 等同性测试

test 语句支持两种字符串同等性测试，等于测试和不等于测试。注意，比较运算和赋值运算的区别的，用作比较时候，比较运算符两端必须各有一个空格。如果漏掉空格，test 语句就会把比较运算看作是赋值运算或嗲测试字符串的一部分，如下所示：

```
name= “John”
test “$name” = John
```

注意：在 test 语句的字符串比较表达式中，引用的变量和字符串前后一定要加双引号。

shell 按 IFS 处理机制从命令行中删除值为 null 的字符串参数。

4.2.2 长度测试：-z 和-n 选项用于测试字符串表达式的长度是否为 0 或非 0.

4.3 整数数值测试运算符

test 语句可以用于测试比较字符串表达式中包含整数值。Test 语句整数值的比较是通过 C 中函数 atoi()实现的，将其转换为 ASCII 整数值。

测试语法： expr1 -options expr2

options 取值：

-eq ： equal -ne ： not equal -gt ： greater than -lt ： less than

-ge ： greater and equal -le ： less and equal

4.4 逻辑运算

表 达 式	说 明
(exp)	用于计算括号中的组合表达式。如果整个表达式的计算结果为真，则测试为真。
! expr	逻辑非运算
exp1 -a exp2 exp1 && exp2	表达式进行逻辑与运算。注意'[...]'不允许使用 & &运算符。 • test exp1 -a exp2; • test \(exp1 -a exp2\) • [exp1] && [exp 2] • [[exp1 && exp2]]

exp1 -o exp2 exp1 || exp2	逻辑或，注意‘[...]’不允许使用||运算。 • test exp1 -o exp2 • test \\(exp1 -o exp2 \\) • [exp1 -o exp2] • [exp1] ||[exp2] • [[exp1 || exp2]]

比较两个文件的判断实例：

```
if  [ -f file1 -a -f file2 ]
then
   diff  file2 file2
fi
```

5. 命令行解释执行过程

shell 命令行解释执行是一个复杂的处理过程。了解命令行解释执行的过程有助于用户正确的访问系统，准确的输入命令，从而避免误用和出现意外的结果。

Shell 的执行过程主要是 16 个步骤：

1.读取命令行	2.命令历史替换	3.命令别名替换	4.花扩号替换
5.波浪号替换	6.I/O 重定向	7.变量替换	8.算术运算结果替换
9.命令替换	10.单词解析	11.文件名生成	12.引用字符处理
13.进程替换	14.环境处理	15.执行命令	16.跟踪执行过程

5.1 读取命令行

shell 解释程序执行的第一步是读取命令行(终端或者脚本)。在用户终端上访问 shell 时，还需要在用户终端上回显读取的每一个字符，然后对读取的命令进行解析和处理。需要跨越多个物理行的命令语句，在进一步解析处理之前，shell 将会连续地读取整个命令语句，直到 shell 自己认为输入已经结束。

在读取命令行时，shell 将会逐个判读读取的每个字符，直到遇到一个分号‘;’,'&', '&&'以及‘||’。

在读取完整的命令或者结构语句之后，shell 开始对命令语句进行自左向右的语法分析，把命令语句分解成一系列的单词或关键字(词法分析阶段)。依次剥离出一系列的单词，包括‘<’,'>','|',‘^’等特殊字符组成的单字。

5.2 命令历史替换

读取一个完整的命令行之后，检查是否需要使用命令历史缓冲区中记录的命令替换读取的命令，是否需要编辑命令。如果需要，则取出相应的命令，在经过适当的校正。例如：‘!!’.

如果想要提高 shell 运行的效率，省略这一步骤，使用：set +o histexpand. 历史替换不适用于 shell 脚本。

5.3 别名替换

shell 检查是读取的命令是否为命令别名，如果确认为命令别名，根据定义，使用原始的命令予以替换。

禁止命令：set -u expand_aliases

5.4 花括号扩展

花括号扩展提供了简单的文件名生产的方法。示例如下：mkdir ~/script/{old,new,tools,admin}

文件路径名较长的时，花括号扩展机制非常的有用。

5.5 波浪号替换

指定用户主目录的方法，$HOME,~.~+表示 PWD 变量的值， ~-表示 OLDPWD。

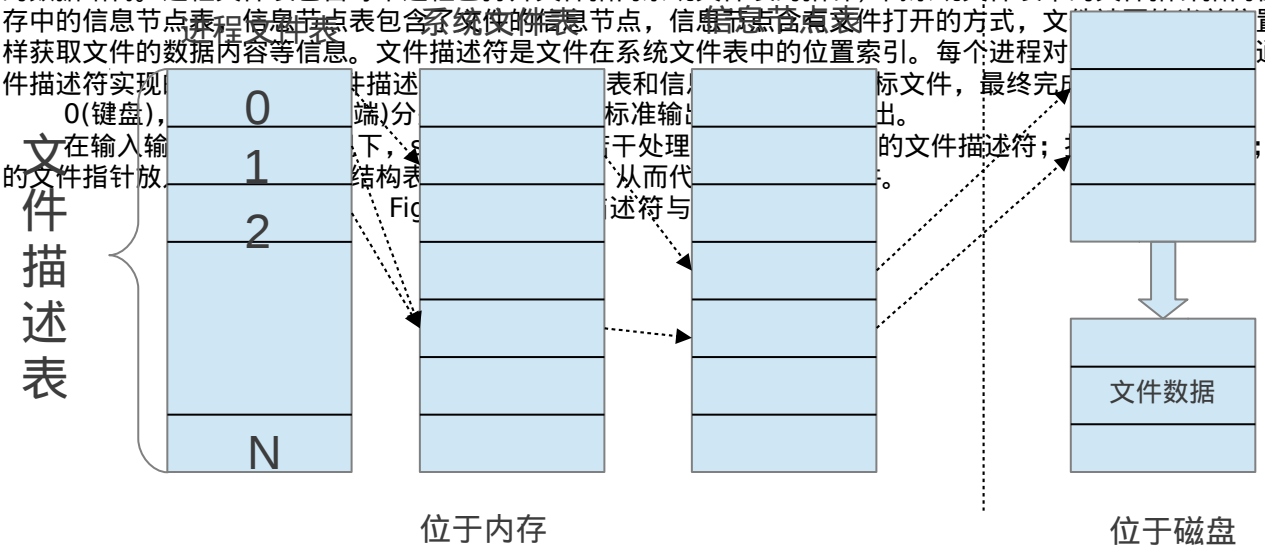
5.6 I/O 重定向

不了解文件描述副，进程文件表，系统文件表，和信息节点表，很难理解 I/O 重定向。这些都是 Linux 系维护的数据结构。进程文件表包含每个进程已打开文件指向系统文件表的指针，而系统文件表中的文件指针指向位于内存中的信息节点表。信息节点表包含系统的信息节点，信息节点含有文件打开的方式，文[illegible]置，怎样获取文件的数据内容等信息。文件描述符是文件在系统文件表中的位置索引。每个进程对[illegible]通过文件描述符实现[illegible]表和信[illegible]标文件，最终完[illegible]

0(键盘),[illegible]端)分[illegible]标准输[illegible]出。

在输入输[illegible]下，s[illegible]干处理[illegible]的文件描述符；[illegible]；把新的文件指针放[illegible]结构表[illegible]从而代[illegible]。

Fig[illegible]述符与



5.7 变量替换

变量替换涉及 shell 内部变量，用户自定义变量，命令行参数变量或者位置参数等变量替换。

变量替换由$符号与随后的变量名构成。注意变量名和参数名可以使用花扩号和单双引号。

5.8 算术结果替换：$((expression))

5.9 命令替换

命令替换的表达式是：‘`’或$(...)形式。命令的范围很广。

在‘`’单一号之前增加转义符号‘\’,可以实现命令替换比表达式的嵌套。

5.10 单词解析

完成上述替换之后，shell 利用 IFS 变量设定的字符作为分隔符，对经过各种替换后生成的命令行再次进行解析，以分离出最基本的命令行元素或者单词。否则，shell 会删除额外的空格，制表符和换行，以及隐含的 null 参数。

IFS 变量默认为空格，制表符和换行符。

5.11 文件名生成：扩展元字符，生成文件名

5.12 引用字符处理：删除引用符号，删除转义符号‘\’和单双引号等引用字符，展开引号中的表达式，完成命令执行前所有的预备工作。

5.13 进程替换

相对于命令替换而言，命令替换是把命令的输出数据赋予给某个变量，以便进一步的处理。进程替换利用/dev/fd/<m>特殊文件或管道文件，把一个进程的输出结果，作为输入数据提交给另一个进程处理。进程替换是利用/dev/fd/<m>特殊文件或管道文件实现进程间标准输入与输出的交互通信的。

实现形式‘<(command)’或者‘>(command)’,注意，>和(之间没有空格。

功能特性：使用进程替换命令行中的文件名参数。

5.14 环境处理：变量赋值，检索 PATH 变量指定的目录，找出命令文件在的位置。

5.15 执行命令：外部命令启动单独子进程执行相应的命令，内部命令 shell 自己执行。

如果命令是一个编译后的程序，shell 启动的子进程使用 exec 系统调用执行应用程序。为了确定是否是可以直接运行的，子进程先尝试把命令文件加载到内存中，然后在执行，如果不能加载到内存中，则假定命令文件为 shell 脚本。前台会等待，后台仅仅输出作业号和进程号

5.16 跟踪执行过程

使用 set 命令的“-v”或“-x”等选项设定了跟踪命令执行的过程，shell 将会在这个处理步骤中输出当前正在执行的命令语句，同时在每个命令语句之前插入一个‘+’前缀，然后再输出命令的运行结果。

包含管道形式的并列的命令，并列命令的执行顺序没有严格的定义，命令的执行是随机的。

六.shell 高级编程

除了顺序执行命令之外，shell 还提供丰富的控制结构。

1.if-then 语法：

```
if command
then
   command-list
fi
```

或者：

```
if command
then
   command-list
else
```

```
if  [ condition ] ; then
        command-list
else
        command-list
fi
```

```
        command-list
    fi
    nested if -then condition test:
    if  [ condition ]
    then
        if  [ condition ]
        then
            command-list
        fi
    fi
```

```
if  [ condition1 ]  &&  [ condition2 ]
then
        command-list
fi
```

```
command-list
2.case statement
case  “$variable” in
    pattern1)
        command-list  ;;
    pattern2)
        command-list ;;
    pattern3)
        command-list ;;
esac
```

模式 pattern 中可以使用元字符，也可以使用运算符‘|’，表示多个模式的逻辑或关系。‘*’可以匹配任何数值，字符或者字符串，因此可以作为默认模式

- case 语句中的测试条件通常是单个数值或者单词，不会出现包含分隔符的字符串，测试变量前后不加双引号
- 匹配检查的模式必须是附加一个右圆括号后最
- 除了最后一组命令或代码块，其他模式之后据要附加一个双分号
- case 语句以 esac(case 的反序拼写)结束。

```
3.for loop statement
    for  var [in  word-list]
    do
        command-list
    done
    或者:
    for var in  [ word-list ] ; do
        command-list
    done
4.while loop statement
    while  [ comdition-is-true ]
    do
        command-list
    done
5.until loop statement
until [ condition-is-true  ]
  do
        command-list
  done
    或者:
 until  [ condition-is-true ] ; do
        command-list
 done
6.select loop statement
  select variable  [ in list ]
  do
        command-list
       break
   done
```

7.嵌套循环
8.循环控制和辅助编程命令
8.1 break 和 continue 命令
break 和 continue 循环控制命令和 C 或者其他编程中的 break 和 continue 语句功能相似。break 命令用于跳出相应的循环体，终止循环体的继续执行。而 continue 命令则用于越过循环体中尚未执行的命令的语句，结束本次循环，直接跳转到下一次循环迭代外。
break [n]
break 命令后附加一个数字参数。一个简单的 break 命令只是终止最内层的循环体。break n 可以跳过 n 层循环体，将控制转移到外部第 n 个关键字 done 后面的命令的语句处开始继续执行。
continue [n]
continue 类似 break 命令。
break 和 continue 命令只能用于 for，while，until 等循环语句的
8.2 true 命令
true 命令用于返回一个表示测试成功的返回值 0 之后，不执行其他的任何动作。
true 命令和‘：’命令可以互相的替换
8.3 sleep 命令
sleep 命令可使 shell 暂时休眠一定的时间，然后在执行的下一个命令。语法如下：
sleep [n]
参数 n 以秒为计数。
8.4 shift 命令：用于修改位置参数的值。位置参数左移
8.5 getopt 命令
getopt 命令主要用于解析命令行的选项，检查选项的合法性。为 shell 编程提供方便，语法格式如下;
set -- `getopt opstring $*`
其中，opstring 是一个包含所有的合法的命令的选项的字符串。如果选项字母之后负有一个冒号“:“,则意味着相应的选项需要一个选项参数(选项字母与选项参数之间既可以存在空格分隔符,也可以连接在一起)
8.6 getopts 命令
getopts 是对 getopt 命令的更新和替代品。语法格式如下：
getopts opts string [arg....]
用于获取以减号或加号为起始字符的命令选项，选项参数和命令参数。
9. 循环语句的 I/O 重定向
对于循环体语句而言,所谓的 I/O 重定向主要是重定向循环提中的标准输入。当然，也可以利用 I/O 重定向和管道的方法，把循环中的输出结果保存到文件中，以备查阅。
9.1 while 循环的 I/O 重定向
9.2until 循环的 I/O 重定向
9.3for 循环的 I/O 重定向
10.Here 文档
here 文档是一种具有特殊用途的代码块，是 I/O 重定向的一种特例。Here 文档采用 I/O 重定向的方法，把一系列的需要从键盘上输入的命令，模拟人工的输入方式，一行一行地提交给交互式应答程序或者命令。如 ftp 和 MySql 数据库等。Here 文档的语法格式如下：

```
program << LimitString
    command1
    command2
    ....
    command3
LimitString
```

其中，特殊的 I/O 重定向符‘<<’与‘LimitString’表示 here 文档的开始，单独另一起一行的 LimitString 表示 here 文档的结束。‘LimitString’是启动指定的程序后中间一系列交互命令的风格符。I/O 重定向的效果是把 here 文档列出的命令提交给交互式应用程序或命令的标准输入。Here 文档相当与执行下列命令：

```
interactive-program < command-line
```

可以利用 here 文档来维护数据库。实例如下：

```
#!/bin/bash
#implement sql database maintance and backup
mysql -u root -psqladmin << EOF
USE mysql;
GRANT ALL PRIVILEGES ON books.* TO xxx@"localhost" IDENTIFIED BY 'mima'
FLUSH PRIVILLEGES;
```

quit
EOF
mysql -u xxx -pmima books < /tmp/bookdump.sql
注意 LimitString 确保其在 shell 脚本中是唯一的。
对于非交互式的实用程序或命令，有效的利用 here 文档，有时会取得非常的好的效果。

11.shell 函数

Bash 等 shell 也支持函数，尽管实现的功能有一定的限度。一个函数是一个子程序，其中利用一组代码块实现特定的处理功能。与 shell 脚本相比，函数被直接存储在内存中，执行的速度要快于 shell 脚本。函数语法：

function_name() {
}
或者
function function_name {
}
注意:函数名不能同现有的变量同名，函数和右括号之间必须连接一起，中间不能有空格。
注意定义函数的时候，不能在函数体内使用 exit 命令，因为函数执行和 shell 脚本同属同一个进程。表示函数的结束使用的是 return 命令。当仅函数需要返回一个参数的时候，需要使用 return，其他的都不是必须的。return 的参数需要显示的指定返回值，否则将会返回最后一条命令的出口状态。Shell 函数必须在使用前定义。w 对于经常使用的函数代码，最好在用户自己的.profile 文件中定义，这样可以保证，只要 shell 注册运行，随时都可以调用函数。
调用 shell 函数也可以提供参数，语法格式如下:function_name $arg1 $arg2 $arg3 ...$argN
以下是一个求素数的程序：

```
#!/bin/bash
#this example show how to use shell function
prime() {
   declare -i num lim rem
  num=3
 lim=`echo $1 | awk '{print sqrt($1)}'`
  while [ $num -le $lim ]
 do
    rem=`expr $1 % $num`
  if [ $rem -eq 0 ]
  then
    return 0
  fi
  num=`expr $num + 2`
 done
return 1
}
declare -i number
while true
do
 echo -e "\nEnter a number: \c"
 read number
 if [ $number -lt 2 ]
 then
    exit 1
 fi
 echo "The prime number list within $number"
 echo -e "2  \c"
 if [ $number -eq 2 ]
 then
    echo
    exit 2
 fi

 i=3
```

```
   while [ $i -le $number ]
   do
      prime $1
      ret=$?
      if [ $ret -eq 1 ]
      then
         echo -e “$i  \c”
      fi
      i=`expr $i + 2`
  done
done
exit 0
```

无论何时调用函数，都会重新设置 shell 进程当前的位置参数，使调用函数提供的参数成为新的位置参数。函数的位置参数同命令的位置参数及其相似，也是按$1,$2,.....的顺序排列的。注意：shell 函数仅仅接受值参数。如果把变量名作为参数传递给函数，参数将会把变量名作为字符串常量处理。也就是说，函数将按字符常量解释变量参数。

```
#!/bin/bash
echo_var() {
   echo "$1"
}
Message=Hello
Hello="Good Morning"
echo_var Message          #显示的是 Message
echo_var Hello            #显示的是 Hello
echo_var "${Message}"  #显示的是 Hello
echo_var "${Hello}"       #显示的是 Good Morning
echo_var ${Hello}         #显示的是 Good
exit 0
```

实际上，函数也是一个代码块，意味着也可以重定向函数的标准输入，标准输出和标准错误输出。例如可以将函数的标准输入重定向到$file 变量指定的文件中，当调用 function_name 函数时，函数的 read 语句不再读取来自键盘的输入数据，而是直接的读取来自$file 变量定义的文件。

```
#!/bin/sh
function_name() {
   ....
   read lines
} < $file  #重定向函数的标准输入
```

当然是也可以在普通调用 function_name 函数重定向一个文件中：function_name < $file

可以利用 shell 写一些简单的

12.逻辑与和逻辑或并列结构

命令的逻辑与(&&)和逻辑或(||)并列结构提供了一种依次执行一系列命令的手段。并且可以有效的替代复杂的，嵌套的 if-then 语句结构。

逻辑与命令并列结构:

```
command-1 && command-2 && command-3 &&....&& command-n
```

命令的逻辑与并列表示从第一个命令开始，依次执行每一个命令，如果当前命令的出口状态为 0，则继续执行下一个命令。如果中间的某个命令执行返回值大于 0，则并列的命令链的执行结束。第一个返回为非 0 的命令即为逻辑与并列结构中最后执行的一个命令。

逻辑或的并列的结构：

```
command-1 || command-2 || command-3 ||....|| command-n
```

和逻辑与命令相反，如果某个命令的出口状态不为 0，则继续执行下一个语句。如果如果中间某个出口状态返回为 0，则终止执行。

13.shell 数组

Bash 仅仅支持一维数组，当对数组大小没有影响，采用整数索引，数据元素从 0 开始，语法形式：

```
[declare -a] array-name={element1 element2 ....elementn}
array-name[index]=value
```

注意：使用 declare 命令声明的数组可以加速数组的处理，提高 shell 脚本的性能。

数组的部分赋值： array=([0]=”first” [2]=”second” [3]=”fourth”)

${array_name[@]}或者${array_name[*]}表示所有的数组元素

${#array_name[@]}或者${#array_name[*]}表示数组 array_name 的长度，即所有数组元素的数量。

${#array_name[index]}表示具体的数组元素的长度。${#array_name}is equal to ${#array_name}。可以使用 unset 命令删除数组元素，甚至整个数组。

灵活地组合使用数组初始化语句“array=(element1 element2 .....elementN)”与命令替换，能够把一个文本文件的内容加载到数组中。因而，定义初始化的第二种方法是命令替换法。命令替换的形式初始化数组，shell 使用空格作为数组的元素的分隔符，把每个字符串分配到一个数组元素中，最终构成一个字符串数组。在遇到制表符，shell 首先把它们转换成空格，并用作元素的分隔符。命令替换形式的数组定义语句实例：

“array=(`cat dict`)”

14.信号的捕获与处理

命令是一种能够影响进程运行状态的外部事件，是由用户终端，操作系统内核以及其他进程发送给当前或者指定进程的消息，用来通知进程某件事情已经发生了。进程可根据预订的方案，采取一定的处理措施或终止处理进程的执行，如果事先没有捕获信号，默认的处理动作是停止进程的执行。

在 shell 脚本中，trap 命令用于指定需要捕获的信号，以及应采取的处理动作。

trap [“command-list”] [signal ...]

或者

trap [“command-list”] [signal ...]

command-list 是无论何时收到指定的信号都会立即执行的命令或者一组命令。一旦命令执行完成，程序的控制逻辑就会恢复到引收到信号而中断的位置开始的位置继续执行，除非命令中包含 exit 命令。

signal 表示准备捕获的信号，信号可以是一个标准的信号名(推荐-可读性)或者一个数值。针对 shell 编程而言，比较重要，能够捕获的信号是 0，1，2，3，15，20 等。EXIT(0)可以用来清除 shell 脚本运行过程中创建的临时文件。此外，DEBUG，ERR，RETURN，主要用于调试和控制 shell 脚本的执行。

信号	信号名	简 单 说 明
0	EXIT	进程结束信号。shell 脚本运行正常结束时，即使没有显示执行 exit 语句，也可产生这个信号
1	SIGHUP｜HUP	挂断。终端通信连接断或者控制台进程终止时产生的信号。
2	SIGINT｜INT	中断。中断键(Ctrl+C)产生的信号
3	SIGQUIT｜QUIT	推出。Ctrl+\或者 Ctrl+Shift+｜键时产生的信号
15	SIGTERM｜TERM	终止信号，kill 命令产生的终止信号
20	SIGTSTP｜TSTP	键盘终止的信号。交互的时候,Ctrl+Z 键产生的信号。
	DEBUG	在执行 shell 的脚本之前，包括 for,case,select 命令语句，以及 shell 函数中第一个命令，都要运行的指定的命令。
	ERR	shell 脚本中任何一个命令，一旦运行结束返回非 0 的出口状态，都要运行 trap 语句中的指定的命令，当 while，until，if 等结构语句除外。
	RETURN	每当调用一个 shell 函数，或者利用“.”或者 source 命令执行一个 shell 脚本结束之后，都要调用指定的命令

注意，shell 脚本中有 5 个信号不应捕获，其实 9，19 是不能捕获的，信号 17，30 则不应捕获。信号的详细定义见进程管理的中信号表。捕获 10，12 或引起管道通信的问题。

使用 trap 命令，程序员可以采取下列措施：

- 捕获指定的信号，然后执行必要的命令或者处理动作
- 忽略收到的信号
- 清除先前的信号捕获设置

有时，为了防止任何人使用 Ctrl-C 键打断 shell 脚本的正常运行，可以使用下列形式的 trap 语句，屏蔽中断信号。

trap ‘’ INT|2|SIGINT

如果 trap 命令命令部分没有指定任何处理动作，或者明显地指定一个 null 值，表示忽略指定的信号。

trap ‘’ signal 或者 trap ‘:‘ signal

上述的两个命令的区别在于第一个 trap 语句表示完全忽略指定的信号，第二个意味着在收到指定的信号不做任何的处理。这两者在同一进程中差别并不明显。

第一种情况下，如果父进程忽略指定的信号，则子进程也将忽略相应的信号。

第二种情况下，如果父进程捕获指定的信号，还将通知子进程清除相应的信号的设置，包括继承子父进程的

信号的设置。但是父进程不会影响进程对相应的信号默认的处理。

使用 trap 语句时，如果仅仅定义了准备捕获的信号而未定义相应的处理动作，shell 将会删除先前为相应的信号定义的处理动作，此时 shell 将会默认动作处理这些捕获到的动作。实例如下:

```
trap 1 2 3
```

通常，trap 语句应该是 shell 脚本中第一个执行的语句，也可以根据具体的情况进程调整。在捕获到指定的信号之前，Shell 只是把 trap 语句读入内存，实际上并不执行的。

在数据库的更新脚本中，进行数据库的更新的时候，通常需要防止用户的中断信号打断脚本的执行，以免破坏数据的完整性。实例如下：

```
#!/bin/bash
while true
do
   echo "Please enter name,phone number,and email address"
   echo "separated by blank character (or enter quit to end):"
   read name phone email
   if test "${name}" = "quit"
   then
          break
   fi
   trap ""  SIGINT SIGQUIT                #屏蔽信号
   #Critical code segment to update database
   update.sql $name $phone $email
   trap SIGINT SIGQUIT                    #恢复信号的默认处理
done
```

15.其他 shell 课题

15.1 子 shell

在键盘输入命令执行单个命令，以交互的方式解释执行用户提交的命令，然后创建一个新的进程，执行用户提交的命令。在执行 shell 脚本时，当前 shell 会创建一个新的 shell 进程，以批处理的方式处理用户提交的 shell 脚本，解释执行的脚本文件中一些列的命令。每个运行的 shell 脚本都是当前 shell 的子进程。

shell 脚本本身也可以创建子进程。这些子 shell 采用并行的方式，能够同时执行多个处理任务。脚本中被一个外部命令都会使 shell 创建相应的子进程，但是内部命令不然。如下所示，位于()中的命令将以子 shell 的形式运行。

```
( command1; command2; command3; ...)
```

子 shell 中变量仅在子 shell 代码中有效，在调用子 shell 的父 shell 进程中是不可见的。子 shell 中的变量属于本地变量。

15.2Shell 脚本的调试

shell 不提供 shell 脚本的调试程序，甚至不提供任何有关调试的调试命令或结构语句。

以下情况会引起 shell 终止运行：

- 控制结构语句(循环结构和条件转移结构)出现语法错误
- 接受到某种信号，外部信号或者内部信号
- shell 内置语句的执行失败(read,test)
- 特殊替换中出现语法错误(引号不匹配或不匹配)
- 赋值语句出现故障(对 readonly 命令的变量赋值)

以下不会引起 shell 的停止运行：

- 试图执行一个不存在的命令
- I/O 的重定向的错误
- 命令异常中止
- 命令终止运行时返回非 0 值出口状态(除非使用 trap 语句跟踪此种情况)

shell 脚本的错误类型：

- 运行出现“syntax error”信息
- 可以运行，结果非预期
- 可以运行，结果无问题，存在边界效应

调试工具和方式：

- 使用 echo 在关键位置显示关键变量的值
- 在关键位置使用 tee 命令检查进程或数据流

- 使用 trap 命令捕获 shell 脚本终止时的位置。捕获 EXIT 信号，输出运行结果时的变量的值通常是一种很有用的方法。
- 在 trap 命令中，DEBUG 信号使 shell 能够在执行的脚本中的每个命令之前立即执行指定的处理动作
- 在 trap 命令，ERR 命令使 shell 能够执行脚本中的每一个命令时，跟踪运行过程中出现的异常的命令。trap ‘command’ ERR 设置有助于找出 shell 脚本中出错的命令语句，从而找出 shell 脚本错误的原因。
- 在 trap 命令中，RETURN 信号使得 shell 能够在调用脚本中的 shell 函数，或者使用‘.’或者 source 命令执行的其他的 shell 脚本之后，立即执行指定的处理动作。trap ‘command’ RETURN 设置能够 shell 函数或其他的脚本的执行的情况
- 设置命令执行过程的跟踪标志(增加-n，-v 或者-x 选项)
  a) “sh -n scriptname” 命令用于检查 shell 脚本的语法错误，实际上并不运行脚本。这一命令形式等价与在脚本中插入“set -n”或“set -o noexec”语句。注意，这种检查并不能找出所有的语法错误。
  b) “sh -v scriptname”命令的作用是在执行之前显示每一个命令，然后显示命令的执行结束。这一命令形式的等价于在脚本中插入”set -v”或“set -o verbose”语句。
  c) “-n”和“-v”命令可以一起工作。“sh -nv scriptname”命令意味着出详细的语法检查
  d) “sh -x scriptname” 命令的作用是以简化的方式显示一条命令语句的执行结果，等价于在 shell 脚本中插入“set -x”或者“set -o xtrace”语句
  e) 在脚本中插入并执行”set -u or set -o nounset”语句，使 shel 在遇到使用的未声明的 9 变量时给出错误信息。

信号捕捉机制对调试 shell 脚本是很有用的。

‘set -u’命令可以禁止 shell 使用事先没有声明的变量。使用这个特性可以发现变量名的拼写错误。

15.3 系统性能考虑

15.3.1PATH 变量的组织

PATH 变量设置的目录直接决定的了命令检索的速度。切忌不要将当前目录放到 PATH 变量前面。

15.3.2 文件访问的方式:当需要访问同一目录下的大量的文件时，最好利用 cd 命令进入该目录。否则，操作系统需要从路径名的第一个目录开始，逐层检索各个子目录，直到找到最终的文件名，这个过程很费时间。

15.3.3 内部命令和外部命令

内置命令语句的执行的速度快于外部系统命令,以下是原因:

- 内置命令不需要创建新的进程。创建新的进程耗费的时间比较的多，分配内存空间，填写进程控制表，从磁盘中将程序调入内存等。
- 内部命令需要按 PATH 变量检索指定的目录。这个也耗时。
- 内部命令的执行在当前进程中执行。

15.3.4 管道中的命令的顺序

使用管道时，应当考虑过滤程序的位置顺序，逐步减少数据处理的信息量，已改善整个管道命令的运行的效率。尽可能的把过滤程序(例如 grep)安排在最前面，把非过滤程序(sort)安排在后面。

七.进程管理

进程指处于运行状态的程序，从程序开始调度运行直至终止执行整个生命周期的全过程。进程管理是 Linux 系统中一种重要的组成部分，负责管理和运行所有的动态过程和资源(文件系统负责所有的静态信息和资源).

Linux 的进程的分类：系统进程和用户进程。系统进程主要负责 Linux 系统的生成，管理，维护和控制，包括 init 进程等。用户进程指得是用户通过 shell 命令界面(或 GUI 桌面)提交系统运行的命令和应用程序。除了少数进程外，系统所有的进程都是由 init 进程直接或者间接启动的，init 进程几乎是所有的进程的直接或者间接父进程。

每个进程都有一个系统赋予的进程 ID，并且与启动进程的用户 ID 等相关联。所有的进程由进程子系统负责管理。用户可以查看进程的状态信息，但只能控制自己的进程(向进程发送控制信号)。

Linux 的系统一个多用户，多任务的进程管理机制，确保所有的处于竞争状态的进程都能公平的合理的分享系统资源，保证重要的进程可以优先分配到资源，所有用户均可以在不同程度上控制自己的所控制的进程。

1.ps 命令概述

ps 命令的语法格式：

ps [-aAcefFHLW] [-g grplist] [-p proclist] [-t term] [-u usrlist]

选 项	GNU 选项	描 述
-a		显示系统中的所有的活动进程的当前状态信息(和终端无关的进程除外)
-A		显示系统中当前所有进程的状态信息，其作用等同于‘-e’

-c		和‘-l’选项一起使用的时候可以显示进程的调度的信息，包括进程的调度类别与优先级等。
-e		显示当前系统所有的进程的状态信息
-f		显示进程的重要的状态信息，尤其是起止运行时间和进程的占用 CPU 的时间等
-F		与‘-f’选项相比，可以显示更多的信息
-H		表示进程调用层次关系的缩进形式显示所有的进程的状态信息
	--forest	表示进程调用层次关系的树形结构显示所有的进程的状态信息
-l		显示进程详细状态信息
-w		显示完整的进程信息。信息过长的部分通常仅通常会被截断
-g group		显示指定的有效用户组 ID 或用户组有关的进程状态信息
-p pid		显示与指定终端设备有关的进程状态信息
-t term		显示与指定终端设备的有关的进程状态的信息
-u user		显示与指定的有效的 ID 或用户名有关的进程的状态信息

ps 命令给出的信息：

- S：　进程当前的工作状态
- PID：　进程标识
- PPID:　父进程的标识
- UID：　用户标识
- CLS：　进程所处的调度级别
- PRI：　进程优先权
- ADDR：　进程地址空间
- RSS　占用的内存
- TIME：　占用的 CPU 时间

进程的状态：R(run),S(sleep),D(等待 I/O),Z(僵尸),T(跟踪或停止),W(页面调度)

pstree 命令：

语法形式：pstree [-achlnpuH] [pid | user]

选项	简单描述
-a	显示进程的命令行参数。如果进程已经存储到交换区，则在进程之后加一对圆括号
-c	禁止压缩同等的进程子树
-h	高亮显示当前进程机器及其父进程
-H [pid]	高亮显示指定的进程
-l	显示完整的进程的信息，过长的部分会被剪切掉
-n	显示是进程 ID 而不是进程的名字来排序父进程
-p	在进程名之后，加圆括号输出进程 ID
-u	如果进程的 UID 与其父进程的 UID 不同,在进程名之后以添加园括号的形式输入用户 ID

2.监控进程及系统资源

top [-hv|-bcisS] [-d delay] [-n iterations] [-p pids]

3.终止进程的运行

Linux 系统既支持 POSIX 标准信号，也支持 POSIX 实时信号。收到信号之后的进程处理动作：

- 终止进程的运行
- 忽略收到的信号，进程继续执行
- 终止进程的执行，并把进程的内存映像转储到 core 文件中
- 暂时停止进程的执行

要强行终止一个进程可以使用 bash 的内部命令 kill 或者/bin/kill，kill 语法：

kill [-s sigspec | -n signum | -n sigspec] pid .....

kill -l [sigspec]

其中，sigspec 即可以是一个规范的信号名(SIGKILL),或者缩写的信号名 KILL,也可以是一个信号的编码。signum 是一个信号的编码，表示 kill 命令发送给指定的进程的的信号。

信号	信号名	默认动作	简单说明
1	SIGHUP	HUP	终止进程	挂断。终端通信连接或者控制进程终止时产生的信号
2	SIGINT	INT	终止进程	中断。按下 Ctrl-C 时发出的信号
3	SIGQUIT	QUIT	终止进程，生成内存映像文件(core)	推出。按 quit 键(Ctrl-| Ctrl+ shift+\)时产生的信号
4	SIGLL	ILL	终止进程，生成内存映像文件(core)	非法指令。执行非法的机器指令时产生的信号
5	SIGTRAP	TRAP	终止进程，生成内存映像文件(core)	硬件故障或断点跟踪等情况产生的信号
6	SIGABRT	ABRT	终止进程，生成内存映像文件(core)	异常终止信号(abort()函数产生的异常终止的信号)
7	SIGBUS	BUS	终止进程，生成内存映像文件(core)	总线故障
8	SIGFPE	FPE	终止进程，生成内存映像文件(core)	浮点运算异常。
9	SIGKILL	KILL	终止进程	无条件的终止进程信号
10	SIGUSR1	USR1	终止进程	用户定义信号，编程使用。
11	SIGSEGV	SEGV	终止进程，生成内存映像文件(core)	内存地址越界或者访问权限不足时产生的信号(当访问地址超出进程地址空间)
12	SIGUSR2	USR2	终止进程	用户定义信号，编程使用。
13	SIGPIPE	PIPE	终止进程	管道断开信号
14	SIGALRM	ALRM	终止进程	alarm()系统调用产生的时钟超时信号
15	SIGTERM	TERM	终止进程	进程终止信号,kill 命令的默认信号
16	SIGSTKFLT	STKFLT	终止进程	栈故障信号
17	SIGCHLD	CHLD	忽略	子进程状态发生变动信号
18	SIGCONT	CONT	继续(或忽略)	令进程继续运行的信号，作业控制使用。
19	SIGSTOP	STOP	终止进程	停止进程运行信号，用于作业控制。
20	SIGTSTP	TSTP	终止进程	键盘停止信号 Ctrl-Z
21	SIGTTIN	TTIN	终止进程	后台进程试图从控制终端读取数据时产生的信号
22	SIGTTOU	TTOU	终止进程	后台进程试图向控制终端输出数据时产生的信号
23	SIGURG	URG	0	当网络链接中收到数据的数据发生错误，通知进程出现紧急情况是发送的信号
24	SIGXCPU	XCPU	终止进程，生成内存映像文件(core)	进程超过最大软性 CPU 时间限制是产生的信号
25	SIGXFSZ	XFSZ	终止进程，生成内存映像文件(core)	当进程创建文件超过其能创建的软性最大文件容量限制是产生的信号
26	SIGVTALRM | VTALRM	终止进程	settimer 系统调用设置的虚拟间隔时钟超时信号
27	SIGPROF	PROF		settimer 系统调用设置的内核抽样间隔时钟超时信号
28	SIGWINCH	WINCH	0	窗口大小发生变动是产生的消息
29	SIGIO	IO	0	异步 I/O 事件出现后的信号
30	SIGPWR	PWR	0	电源故障信号，改由 UPS 提供系统电源供电时的信号
31	SIGSYS	SYS	终止进程，生成内存映像文件(core)	系统调用有误。非法系统调用的错误

4.调整分时进程的优先级

4.1 nice 命令： nice [-n number] [command [arguments]]

4.2 renice 命令: renice priority [[-p] pids] [[-u] users]

八.proc 文件系统

proc 是一个虚拟的文件系统，不占任何磁盘空间。proc 文件系统是系统运行状态的一个动态反映，可以用作是系统内核数据结构的接口，以便用户可以获取进程运行的状态以及可调用的参数等信息，而不必直接读取解释/dev/kmem 核心内存文件。Proc 文件系统通常安装在/proc 目录下，其中大多数文件只能读取，部分可修改。

proc 文件系统中的目录和文件分类：系统当前所有进程的内存映像，系统当前配置与运行状态信息以及系统内核可调参数。

1.进程内存映像文件

/proc 目录下的以数字形式命令的目录均为进程的内存映像。对于当前正在运行的没一个进程，均存在一个对应的目录，目录以相应的进程的 ID 命令。其中目录 1 对应于著名的 init 进程。

exe 是一个符号链接文件，其中包含了进程命令文件的实际的路径命令，引用这个文件相当与执行的执行的相应的命令。对于进程 1 而言，则相当与执行的/sbin/init 命令。

fdinfo 是从 Linux 内核 2.6.22 开始新增的一个目录，其中包含进程打开的每一个文件。文件以文件描述符的形式存在。显示的命令通过 ls -l fdinfo。

limits 文件中包含了进程的各种资源的软性限制，硬性限制，以及度量单位等。

maps 文件中包含映射的内存去及其访问权限，实例：

```
    sudo cat  /proc/1/maps
b7776000-b7778000  rw-p  00000000 00:00  0
b7778000-b7779000  r-xp   00000000 00:00  0         [vdso]
b7779000-b7799000  r-xp   00000000 fc:00   16777888   /lib/i386-linux-gnu/ld-2.15.so
b7799000-b779a000  r--p   0001f000  fc:00   16777888   /lib/i386-linux-gnu/ld-2.15.so
b779a000-b779b000  rw-p  00020000 fc:00    16777888   /lib/i386-linux-gnu/ld-2.15.so
b779b000-b77bf000   r-xp  00000000 fc:00    14942269   /sbin/init
b77bf000-b77c0000   r--p   00023000 fc:00    14942269   /sbin/init
b77c0000-b77c1000   rw-p 00024000 fc:00    14942269   /sbin/init
b8b07000-b8bac000   rw-p 00000000 00:00  0         [heap]
bfe28000-bfe49000    rw-p 00000000  00:00  0         [stack]
```

其中，第 1 个字段是进程占用的地址空间；第二个字段是由 r，w，x，s(share)，p(private)；第三个字段是相当于文件的起始位置的读写偏移的值；第四字段是表示主次设备号；第五个子段是文件所在文件系统的中信息节点号(0 表示不存在与内存区相关连的信息节点);第六个字段表示进程的路径文件名

mem 文件反映的是进程的内存页，可以利用 open(),read(),fseek()等系统调用访问 mem 文件，从而访问内存页面。

root 也是一个符号链接文件，指向进程的虚拟根目录。虚拟根目录是 Linux 系统的重要功能特性，利用 chroot 命令或者 chroot(2)系统调用，可以针对每个进程设置文件系统的虚拟根目录。

smaps 文件反映了每个进程的内存映射的使用情况。

```
b7779000-b7799000 r-xp 00000000 fc:00 16777888   /lib/i386-linux-gnu/ld-2.15.so
Size:                128 kB
Rss:                 108 kB
Pss:                   0 kB
Shared_Clean:        108 kB
Shared_Dirty:          0 kB
Private_Clean:         0 kB
Private_Dirty:         0 kB
Referenced:          108 kB
Anonymous:             0 kB
AnonHugePages:         0 kB
Swap:                  0 kB
KernelPageSize:        4 kB
MMUPageSize:           4 kB
Locked:                0 kB
```

stat 文件包含进程的运行的状态。实际上，ps 命令也是利用这个文件输出进程状态信息的。

statm 文件以页面的为单位，提供内存的状态信息—从左边第一个字段开始，其输出信息分别是：整个程序的大小，程序内存的驻留部分的大小，共享页数数量，代码部分的大小，库函数部分的大小，数据或栈段部分的大小。

Status 文件提供的信息主要来自/proc/pid/stat 和/proc/pid/statm 文件，只是更容易阅读和理解的形式给出，其中包含进程的名字，状态(R-run,S-sleep),PID,有效用户 ID，用户组 ID，当前分配的文件的描述符的数量

(FDSize),以及各种内存使用的情况，各种内存使用情况的数据—虚存大小，内存驻留部分的大小，共享代码库部分的大小(Vmlib),代码段(VmExe),数据段(vmData)以及栈段(VmStk)的大小等。实例如下：

```
xxx@xxx-pc /p/1> cat status
Name:   init
State:  S (sleeping)
Tgid:   1
Pid:    1
PPid:   0
TracerPid:      0
Uid:    0       0       0       0
Gid:    0       0       0       0
FDSize: 256
Groups:
VmPeak:     3632 kB
VmSize:     3632 kB
VmLck:         0 kB
VmPin:         0 kB
VmHWM:      2024 kB
VmRSS:      2024 kB
VmData:      712 kB
VmStk:       136 kB
VmExe:       144 kB
VmLib:      2536 kB
VmPTE:        16 kB
VmSwap:        0 kB
Threads: 1
SigQ:   0/31066
SigPnd: 0000000000000000
ShdPnd: 0000000000000000
SigBlk: 0000000000000000
SigIgn: 0000000000001000
SigCgt: 00000001a0016623
CapInh: 0000000000000000
CapPrm: ffffffffffffffff
CapEff: ffffffffffffffff
CapBnd: ffffffffffffffff
Cpus_allowed:   ff
Cpus_allowed_list:      0-7
Mems_allowed:  1
Mems_allowed_list:      0
voluntary_ctxt_switches:        1388
nonvoluntary_ctxt_switches:     600
```

2.系统配置信息

通过 proc 文件系统，可以了解当前系统的硬件设备的配置信息，如 CPU，内存，系统总线，I/O 设备，IDE 和 SCSI 磁盘设备，磁盘分区，硬件中断，I/O 端口的内存映射，硬件设备的主次设备号。

/proc/cpuinfo--CPU 及其其他的系统结构的信息。

```
Processor       : 0 --cpu 标识
vendor_id       : GenuineIntel
cpu family      : 6
model           : 37
model name      : Intel(R) Core(TM) i3 CPU       M 350  @ 2.27GHz
stepping        : 2
microcode       : 0x9
cpu MHz         : 933.000
cache size      : 3072 KB --二级缓存
physical id     : 0
```

siblings : 4
core id : 0
cpu cores : 2
apicid : 0
initial apicid : 0
fdiv_bug : no
hlt_bug : no
f00f_bug : no
coma_bug : no
fpu : yes
fpu_exception : yes
cpuid level : 11
wp : yes
flags : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe nx rdtscp lm constant_tsc arch_perfmon pebs bts xtopology nonstop_tsc aperfmperf pni dtes64 monitor ds_cpl vmx est tm2 ssse3 cx16 xtpr pdcm sse4_1 sse4_2 popcnt lahf_lm arat dtherm tpr_shadow vnmi flexpriority ept vpid
bogomips : 4521.76
clflush size : 64
cache_alignment : 64
address sizes : 36 bits physical, 48 bits virtual
power management:

/proc/filesystems 文件：

在 filesystems 文件中包含系统内核当前支持的文件的类表，表示相应的文件系统支持的模块已被编译进系统内核。在安装文件系统时，如果未指定文件系统的类型，mount 命令将会依次使用这个文件中的信息确定文件系统的类型，尝试安装文件系统。实例如下：

```
XXX@XXX-pc /proc> cat filesystems
nodev   sysfs
nodev   rootfs
nodev   bdev
nodev   proc
nodev   cgroup
nodev   cpuset
nodev   tmpfs
nodev   devtmpfs
nodev   debugfs
nodev   securityfs
nodev   sockfs
nodev   pipefs
nodev   anon_inodefs
nodev   devpts
        ext3
        ext4
nodev   ramfs
nodev   hugetlbfs
        vfat
nodev   ecryptfs
        fuseblk
nodev   fuse
nodev   fusectl
nodev   pstore
nodev   mqueue
        ext2
nodev   binfmt_misc
```

/proc/partitions 文件：包含磁盘与磁盘分区的主次设备号，容量(数据块的数量)以及设备文件的名，其内

容类似于“fdisk -l” ,实例是如下：

```
   8        0    312571224       sda
   8        1    248832          sda1
   8        2         1          sda2
   8        5    312320000       sda5
   11       0    1048575         sr0
   252      0    308248576       dm-0
   252      1     4050944        dm-1
```

/proc/mounts 文件:提供系统当前安装的所有的文件系统的信息。

/proc/version 文件：当前运行的 Linux 系统内核以及 gcc 的版本信息，也提供操作系统的信息(proc/sys/kernel/ostype 文件)以及版本号(proc/sys/kernel/ { osrelease,version}).实例：

Linux version 3.5.0-31-generic (buildd@aatxe) (gcc version 4.7.2 (Ubuntu/Linaro 4.7.2-2ubuntu1)) #52-Ubuntu SMP Thu May 16 16:30:01 UTC 2013

3.系统运行的状态信息

3.1/proc/cmdline 文件：启动系统时传递给 Linux 内核的引导参数。

3.2/proc/kcore 文件：系统物理内存的映像，采用的 ELF 文件格式存储

3.3/proc/kmgs 文件：存储系统内核的生成的各种信息

3.4/proc/loadavg 文件：系统的负载(位于运行队列的进程数量).

3.5/proc/meminfo 文件：提供了整个内存，空闲内存和已用内存的数量。

3.6/proc/stat 文件：自启动以后，系统内核中的各种的统计信息。其输出内容可以使用 vmstat 和 iostat 等命令获得。

3.7/proc/swaps 文件:交换区存储空间的配置及其使用的情况。

3.8/proc/uptime and /proc/vmstat

4.系统可调参数

/proc/sys 是一个非常重要的目录，其中的大量的文件分别位于 fs，net，kernel 等子目录下，每个文件对应于系统内核的一个或者多个可调用参数参数。

```
dr-xr-xr-x 1 root root 0 Jun  5 17:50 debug/
dr-xr-xr-x 1 root root 0 Jun  5 16:31 dev/
dr-xr-xr-x 1 root root 0 Jun  5 16:31 fs/
dr-xr-xr-x 1 root root 0 Jun  6  2013 kernel/
dr-xr-xr-x 1 root root 0 Jun  5 16:31 net/
dr-xr-xr-x 1 root root 0 Jun  5 16:31 vm/
```

九.磁盘管理

常用磁盘管理工具

命令	简单描述
df	查询文件系统的中可用的或已用的存储的空间的及文件的节点的数量
du	查询指定目录中的每个文件的或者目录占用的磁盘空间
find -size	检索目录中指定大小的文件
ls -s	以 1kb 的数据块显示文件的大小
cpio	创建，转储或者恢复 cpio 档案文件，实现文件或者数据的备份与恢复，也可用于整体目录层次的复制
tar	创建，转存或者恢复 tar 档案文件，实现文件或者文件的备份和恢复
dd	实现原始数据的复制，可以复制整个文件系统