LinuxShell 应用编程专题讲座
written by 王保明

第一部分 shell 编程基础

1shell 简介

什么是 shell
存取权限和安全
shell 简单脚本
shell 特性

1.1 什么是 shell

- shell 是核心程序 kernel 之外的指令解析器，是一个程序，同时是一种命令语言和程序设计语言。
  - ✧ shell 是命令解析器，用户输入命令，它去解析。
- shell 类型　　ash、bash、ksh、csh、tcsh
  - ✧ cat /etc/shells 看系统下的 shell
  - ✧ echo $SHELL 看当前用户运行的 shell
- 程序在 shell 中运行
  - ✧ ls 命令执行过程分析
- shell 中可以运行子 shell
  - ✧ /bin/csh 退出子 shell
- linux 下默认的 shell 是 bash
  - ✧ bash 特点，快速（上下键）；tab 键盘自动补齐；自动帮助功能 help

在 shell 下执行 help 命令，可以查看 shell 提供的命令
[test@localhost ~]$ help GNU bash, version 3.2.25(1)-release (x86_64-redhat-linux-gnu) These shell commands are defined internally. Type `help' to see this list. Type `help name' to find out more about the function `name'. Use `info bash' to find out more about the shell in general. Use `man -k' or `info' to find out more about commands not in this list. A star (*) next to a name means that the command is disabled. JOB_SPEC [&]　　((expression)) . filename [arguments]　　: [arg...]　　[[expression]] alias [-p] [name[=value] ...]　　bg [job_spec ...]

```
bind [-lpvsPVS] [-m keymap] [-f fi break [n]
builtin [shell-builtin [arg ...]]    caller [EXPR]
case WORD in [PATTERN [| PATTERN]. cd [-L|-P] [dir]
command [-pVv] command [arg ...]      compgen [-abcdefgjksuv] [-o option
complete [-abcdefgjksuv] [-pr] [-o continue [n]
declare [-afFirtx] [-p] [name[=val dirs [-clpv] [+N] [-N]
disown [-h] [-ar] [jobspec ...]       echo [-neE] [arg ...]
enable [-pnds] [-a] [-f filename]    eval [arg ...]
exec [-cl] [-a name] file [redirec exit [n]
export [-nf] [name[=value] ...] or false
fc [-e ename] [-nlr] [first] [last fg [job_spec]
for NAME [in WORDS ... ;] do COMMA for (( exp1; exp2; exp3 )); do COM
function NAME { COMMANDS ; } or NA getopts optstring name [arg]
hash [-lr] [-p pathname] [-dt] [na help [-s] [pattern ...]
history [-c] [-d offset] [n] or hi if COMMANDS; then COMMANDS; [ elif
jobs [-lnprs] [jobspec ...] or job kill [-s sigspec | -n signum | -si
let arg [arg ...]                          local name[=value] ...
logout                                          popd [+N | -N] [-n]
printf [-v var] format [arguments] pushd [dir | +N | -N] [-n]
pwd [-LP]                                      read [-ers] [-u fd] [-t timeout] [
readonly [-af] [name[=value] ...]    return [n]
select NAME [in WORDS ... ;] do CO set [--abefhkmnptuvxBCHP] [-o opti
shift [n]                                       shopt [-pqsu] [-o long-option] opt
source filename [arguments]               suspend [-f]
test [expr]                                     time [-p] PIPELINE
times                                           trap [-lp] [arg signal_spec ...]
true                                            type [-afptP] name [name ...]
typeset [-afFirtx] [-p] name[=valu ulimit [-SHacdfilmnpqstuvx] [limit
umask [-p] [-S] [mode]                    unalias [-a] name [name ...]
unset [-f] [-v] [name ...]                until COMMANDS; do COMMANDS; done
variables - Some variable names an wait [n]
while COMMANDS; do COMMANDS; done    { COMMANDS ; }
```

1.2 存取权限与安全

简介:　文件和目录的权限（-rwxr—r--）

setuid(suid/guid) (chmod u+s g+s file)

chown 和 chgrp(chown user file/chgrp group file)

umask (umask nnn) （文件创建时的缺省权限位）

- 文件和目录的权限(-rwxr--r--)
  - ✧ linux 下安全解决方案很多，现在讨论文件和目录的访问权限
  - ✧ 练习 ls –lh

文件的权限 硬链接数 用户名用户组 文件大小、最近修改时间 文件名称
drwxr-xr-x 2 test test 4096 Jun 23 08:07 cppsocket1 drwxr-xr-x 8 test test 4096 Jun 23 08:08 cppsocket2 drwxr-xr-x 9 test test 4096 Jun 18 12:32 gcc -rw-r--r-- 1 test test 59827 Jun 23 08:11 gcc_mk_gdb.tar.gz drwxr-xr-x 7 test test 4096 Jun 23 08:09 mk drwxr-xr-x 4 test test 4096 Jun 23 09:47 mygcc drwxr-xr-x 5 test test 4096 Jun 23 11:13 mymk -rw-r--r-- 1 root root 63 Nov 9 2013 oraInst.loc -rw-rw-r-- 1 oracle oinstall 732 Nov 9 2013 oratab drwxr-xr-x 2 root root 4096 Nov 9 2013 pam.d drwxr-xr-x 3 root root 4096 Nov 8 2013 racoon lrwxrwxrwx 1 root root 7 Nov 8 2013 rc -> rc.d/rc drwxr-xr-x 10 root root 4096 Nov 8 2013 rc.d
硬链接数？ d-目录 –一般文件 –l 快捷方式 –c 字符设备 –b 块设备 s-socket 设备文件 –p 管道文件
-rw-r--r-- 1 root root 5 Jun 23 12:39 scim-bridge-0.3.0.lockfile-0@localhost:0.0 -rw-r--r-- 1 oracle oinstall 5 Jun 18 16:13 scim-bridge-0.3.0.lockfile-501@localhost:0.0 srwxr-xr-x 1 root root 0 Jun 23 05:39 scim-bridge-0.3.0.socket-0@localhost:0.0 srwxr-xr-x 1 oracle oinstall 0 Jun 18 07:13 scim-bridge-0.3.0.socket-501@localhost:0.0 srw------- 1 oracle oinstall 0 Jun 18 07:13 scim-helper-manager-socket-oracle srw------- 1 root root 0 Jun 23 05:39 scim-helper-manager-socket-root srw------- 1 oracle oinstall 0 Jun 18 07:13 scim-panel-socket:0-oracle srw------- 1 root root 0 Jun 23 05:39 scim-panel-socket:0-root srw------- 1 oracle oinstall 0 Jun 18 07:13 scim-socket-frontend-oracle srw------- 1 root root 0 Jun 23 05:39 scim-socket-frontend-root drwx------ 2 root root 4.0K Jun 23 05:39 ssh-grWlnX5288 drwx------ 2 oracle oinstall 4.0K Jun 18 07:13 virtual-oracle.26qtL1 drwx------ 2 root root 4.0K Jun 23 05:39 virtual-root.selw4P [test@localhost ~]$ ls -l /tmp/.X11-unix/X0 srwxrwxrwx 1 root root 0 Jun 22 20:07 /tmp/.X11-unix/X0 这里 s 为 socket 文件 [test@localhost ~]$

- chmod 改变文件或目录的权限位

man chmod 或者 info chmod
NAME chmod – change file access permissions 改变文件或者目录的权限位 SYNOPSIS chmod [OPTION]… MODE[,MODE]… FILE… chmod [OPTION]… OCTAL-MODE FILE… chmod [OPTION]… –reference=RFILE FILE… DESCRIPTION
chmod 语法格式

chmod [who] operator [permission] filename

who(u , g, o, a)

operator(+, - =)

permission(r, w, x, s, t)

chmod 有两种用法 (The format of a symbolic mode)

1、 数字方式

用户 用户组 其他 全部 增加 去掉 = 读 写 执行 s（具有超级用户）t（执行文件在缓存）eg: chmod u=rwx, g+w, o+r myfile

注意：S 位存在的意义：写一个程序，被执行是，临时拥有超级用户权限，执行完毕以后，有恢复普通身份，例如，写一个数据库脚本，这个数据库脚本具有超级管理员身份运行，运行完毕以后，恢复普通身份，避免破坏系统。

查看带有 s 位的应用程序，命令 ls –l /bin | grep ‘^…s’

```
[root@localhost 01]#
[root@localhost 01]# ls -l /bin | grep '^...s'
-rwsr-xr-x 1 root root     61424 Jul    3    2009 mount
-rwsr-xr-x 1 root root     37312 Apr 24    2009 ping
-rwsr-xr-x 1 root root     32736 Apr 24    2009 ping6
-rwsr-xr-x 1 root root     28336 Jul 13    2009 su
-rwsr-xr-x 1 root root     41224 Jul    3    2009 umount
[root@localhost 01]#
[root@localhost 01]#
```

2、 chmod 数字方式 chmod mode file chmod 775 file

r w x 数字代表 4 2 1

3、chown root.test myfile 改变 myfile 的用户名、用户组

- setuid（suid/guid）（chmod u+s, g+s file）

- chown 和 chgrp（chown use file/chgrp group file）
  - ✧ chown 改变文件、目录所在的用户和用户组
  - ✧ chgrp 改变文件、目录所在的用户和用户组

    ✧ chown [-R] owner myfile
    ✧ chown owner.group myfile
    ✧ chown .group myfile //注意有个点
    ✧ chgrp [-R] group myfile

  - ✧

- umask（umask nnn）
  - ✧ 熟悉了修改文件的权限位和用户组后，思考：创建文件时，如何默认指定权限位那？
  - ✧ 先运行 umask 命令，看看结果如何？
  - ✧ umask 命令，用来指定，用户创建文件、目录，默认的权限位

    ✧ 问题抛出在 0022umsk 值下， 文件默认权限位是 644 目录默认权限位是 755

    ✧ [root@localhost mytest]# umask
    ✧ 0022

✧ [root@localhost mytest]# ls -lt
✧ total 4
✧ drwxr-xr-x 2 root root 4096 Jun 23 21:41 dir
✧ -rw-r--r-- 1 root root 0 Jun 23 21:41 myfile.c
✧ 实验：当 umask 为 022，创建文件，权限位默认是：644
当 umask 为 077，创建目录，权限位默认是 755
做实验，测试！

✧ umask 的秘密

umask	文件	目录
0	6	7
1	5	6
2	4	5
3	3	4
4	2	3
5	1	2
6	0	1
7	0	0

✧ 规律如果文件权限位=7-umask-1 目录权限位=7-umask

用户 用户组 其他用户
7 5 5
rwx r-x r-x
022====》7 55
8421
rwx r-x r-x

✧ ls –lda 命令只查看目录
✧ umask 的配置 /etc/profile($HOME/.profile $HONME/.bash_profile)

- 文件的符合链接（ln [-s] source_path target_path）
  ✧ 硬连接
  ✧ 软连接

```
[test@localhost shell]$ ls
my.tar.gz
[test@localhost shell]$ ln my.tar.gz my2.tar.gz   建立一个硬连接
[test@localhost shell]$ ln -s my.tar.gz   kjfs  建立一个软连接
[test@localhost shell]$ ls -lt
total 128
lrwxrwxrwx 1 test test         9 Jun 23 22:32 kjfs -> my.tar.gz
-rw-r--r-- 2 test test 59827 Jun 23 22:30 my.tar.gz
-rw-r--r-- 2 test test 59827 Jun 23 22:30 my2.tar.gz
[test@localhost shell]$
```

1.3shell 脚本

- 使用 shell 脚本的原因
  - ✧ 功能强大
  - ✧ 节约时间
- shell 脚本基本元素

```
#!/bin/bash

#shell 脚本
mytext="hello world"
echo $mytext;
```

注意初学者易犯错误：
1） 第一行#不是注释 其他#都是注释；
2） echo $mytextA;写错现象；
3） 写错现象#!bin/bash
4） 脚本如果没有执行权限 chmod 111 01hello.sh，如何办
5） chmod u+x 01hello.sh chmod u-x 01hello.sh

- shell 脚本运行方式
  ./01shell.sh $PATH

1.4shell 特性

简介：别名、管道、命令替换、重定向、后台处理、模式匹配、变量、特殊字符

- 别名
  - alias 查看本用户下的 alias 配置

```
[test@localhost ~]$ alias
alias l.='ls -d .* --color=tty'
alias ll='ls -l --color=tty'
alias ls='ls --color=tty'
alias vi='vim'
alias which='alias | /usr/bin/which --tty-only --read-alias --show-dot
--show-tilde'
[test@localhost ~]$
```

  - 自定义别名: alias ll = 'ls -alh'
  - alias ll = 'ls –l –color=tty'
  - cat $HOME/.bashrc 在这个用户下配置着 alias 命名的配置
- 命令替换
  - ls `cat myfile` –alh 通过单反引号，把'cat myfile'的内容输出 传给 ls 命名
  - 注意反单引号，在键盘的~符号的下方。也就是在键盘左上角 ESC 键的下边
- 后台处理
  - 什么是后台 nohup commond &

- 一个终端可以同时运行多个程序
  - nohup tar –cvf 111.tar.gz & 可以同时运行多个大文件压缩
  - jobs –l 可以查看后台正在运行的命令
- 管道
  - 把一个命令的输出作为另外一个命令的输入
    - ls -l| sort; ls | sort
- 重定向 < 输入 >输出
  - 与管道相关，可以改变程序运行的输入来源和输出地点
  - sort< myfile.txt
  - sort<myfile.txt >myfile+sort.txt
- 模式匹配
  - 显示以 txt 为扩展的文件或者显示以 a 开头的文件，这种能力叫做模式匹配
  - 正则表达式
- 特殊字符
  - 双引号(“”):用来使 shell 无法认出空格、制表符和其他大多数特殊字符，比如：建立一个带空格的文件 touch “wang bao ming”。
  - 单引号(‘):用来使 shell 无法认出所有特殊字符。
  - 反引号(`):用来替换命令
  - 反斜杠(\):用来使 shell 无法认出的特殊字符，使其后的字符失去了特殊的含义，转义字符。eg：创建带空格文件名 touch my\ file
  - 分号(;):允许在一行上放多个命令。mv 1.txt 2.txt; mv 2.txt 3.txt;
  - &：命令后台执行
  - 括号()：创建成组的命令
  - 大括号{}：创建命令块。
  - 竖杠(|):管道标示符
  - < > &：重定向表示符
  - * ? [ ] !：表示模式匹配
  - $:变量名的开头
  - #:表示注释（第一行除外）
  - 空格、制表符、换行符：当做空白

2 变量和运算符

本地变量 影响变量的命令
环境变量 引号
变量替换 运算符
位置变量 表达式变量
标准 shell 变量 运算符的优先级
特殊变量

2.1 变量

简介：什么是 shell 变量？本地变量，环境变量，变量替换（显示变量），位置变量，标

准变量，特殊变量，影响变量的命令。

- 什么是 shell 的变量？
  - 为使用 shell 编程更加有效,系统提供了一些 shell 变量。shell 变量可以保存注诸如路径名、文件名、或者数字这样的变量。从这一点上看，shell 编程中，变量至关重要。
  - eg：echo $myVar1
- 本地变量
  - 本地变量在用户现有的 shell 生命期的脚本中使用。
  - variablename=value
    - 定义本地变量：MYVAR=”test” 注意=左右不要有空格
    - 使用本地变量$MYVAR echo $MYVAR or echo ${MYVAR}
  - set 显示本地所有的变量
  - readonly variablename
    - readonly myvar1=”test2” ;myvar1=”test2”;bash 会报错误

```
[wbm@wmblinux64 ~]$ readonly
declare -r BASHOPTS="checkwinsize:cmdhist:expand_aliases:extquote:force_fignore:hostcomplete:interactive_comments:login_shell:progcomp:promptvars:sourcepath"
declare -ir BASHPID=""
declare -ar BASH_VERSINFO='([0]="4" [1]="1" [2]="2" [3]="1" [4]="release" [5]="x86_64-redhat-linux-gnu")'
declare -ir EUID="500"
declare -r MYYY="test2"
declare -ir PPID="7140"
declare -r SHELLOPTS="braceexpand:emacs:hashall:histexpand:history:interactive-comments:monitor"
declare -ir UID="500"
```

  - 实验
    - [wbm@wmblinux64 myshell]$ set |grep "MYV*" 查看我刚才定义的本地变量
    - 退出终端，重新登录，问，刚才登录的本地变量还存在吗？
    - 仔细思考本地变量背后的含义！ echo $MYV 每登录一个终端，都会运行一个 shell 程序，这个本地变量，就保存在这个 shell 程序中；如果再新登录，将启动新的 shell 程序，与原来的 shell 不同。建立起概念。
- 环境变量
  - 环境变量用于所有用户进程（经常称为子进程）。登录进程称为父进程。shell 中执行的用户进程均成为子进程。不像本地变量（只用于现在的 shell），环境变量可用于所有子进程，这包括编辑器、脚本和应用程序。
  - $HOME/.bash_profile (/etc/profile，所有用户都使用的 profile)
  - export 声明环境变量 export myvar=”tttt” 查看环境变量
  - env 查看新增加的环境变量，

- 实验：查看刚才增加的环境变量

- 变量替换（显示变量）
  - 用变量的值替换它的名字
  - echo
  - 在变量名前加$,使用 echo 命令可以显示单个变量取值。
  - echo ${MYVAR} 或者 echo $MYVAR

$(variablename)　显示实际值 $(variablename:+value)　若设置了 variablename,则显示：否则空 $(variablename:?value)　若未设置 variablename,则显示用户自定义信息 valude $(variablename:-value)　若未设置 variablename,则显示其值 $(variablename:=value)　若未设置了 variablename,则设置其值，并显示

```
var1=”wang
echo ${var1:+”bbb”}
”
```

- unset 清除变量
  - unset testvar　；测试 echo $testvar;
  - readonly 的变量，不能被清除
- 位置变量
  - 位置变量表示$0 $1 $2 $3 …$9

```
$0 代表 bash 文件名称 其他就是命令行参数
./02param a b c

#!/bin/bash
#param
echo "脚本的名字 $0"
echo "parm 1: $1
echo "parm 2: $2"
echo "parm 3: $3"
echo "parm 4: $4"
echo "parm 5: $5"
```

  - 向脚步中使用位置参数

```
#!/bin/bash
#param.sh
find /usr/lib/ -name $1 –print　查找目录下文件名是第一个参数的文件
```

  - 向系统命令传递参数
- 标准变量
  - bash 默认建立了一些标准环境变量，可在/etc/profile 中定义
  - EXINIT　定义一些 vim 的初始化参数
  - HOME　echo $HOME
  - IFS　在 linux 系统中字符之间的间隔
    - 测试 echo $IFS
  - LOGNAME
    - 测试 echo $LOGNAME set |grep “LOG”
  - MAIL

- 当前登录用户，其邮箱在什么地方存储
- MAILCHECK
- MAILPATH
- TERM　登录服务器，终端类型 vt100
- PATH 标准变量
  - 可以使用 set |grep “PATH”来查看 path 配置路径
  - 当我们在 shell 运行一个程序时，shell 会从 path 路径中查找程序。
- TZ　时区
- PS1 提示符
  - echo $PS1
  - PS1=’[\u@\h \w] \$’　u 代表用户名 h 主机名 w 代表当前目录
  - 可以自己更改 PS1 比如：　PS1=”wangbaoming”
- PS2 “> ”　大尖括号+空格 在一行上面运行多个命令
  - 一条命令没有写完，自动换行后，shell 提示符

```
[test@localhost ~]$ ls -lt | wc -w
227
[test@localhost ~]$ ls -lt | \
> wc -w
227
```

  -

```
[test@localhost ~]$ for loop in `cat myfile`
> do
> echo $loop
> done
```

- PWD
- SHELL 我当前运行的 shell 时那个 shell 解析器
- MANPATH
- TERMINFO

- 特殊变量
  - $# 传递到脚本的参数个数
  - $* 以一个单字串显示所有向脚步传递的参数，与位置变量不同，次选项参数可以超过 9 个。
  - $$ 脚步运行的当前进程 ID 号
  - $! 后台运行的最后一个进程的进程 ID
  - $@ 与$#相同，但是使用时加引号，并在引号中返回每个参数
  - $- 显示 shell 使用的当前选项，与 set 命令功能相同
  - $? 显示最后命令的退出状态。0 表示没有错误，其他任何值表明有错误

```
#!/bin/bash
#param

echo "脚本的名字 $0"
echo "parm 1: $1"
```

```
echo "parm 2: $2"
echo "parm 3: $3"
echo "parm 4: $4"
echo "parm 5: $5"
echo "显示参数的个数:$#"

echo "显示脚本全部参数:$*"
echo "显示前一命令运行后状态:$?"
```

- 影响变量的命令
  - declare 设置或显示变量
    - -f 只显示函数名
    - -r 创建只读变量（declare 和 typeset）
    - -x 创建转出变量
    - -l 创建整数变量
    - -使用+代替-，可以颠倒选项的含义（只读变量除外）
  - export 用户创建传给 shell 的变量，即：创建环境变量
    - -- 表明选项结束，所有后续参数都是实参
    - -f 表明在“名-值”对中的名字是函数名。
    - -n 把全局变量转换成局部变量。换句话说，命令的变量不在传给 shell
    - -p 显示全局变量列表
  - readonly 用于显示或者设置只读变量
    - -- 表明选项结束
    - -f 创建只读变量
  - set 使用 set 命令显示所有本地定义的 shell 变量。
    - 设置或者重设各种 shell
    - set -a
  - env 查看所有环境变量。
  - shift [n] 用于移动位置变量，调整位置变量，是$3 的值赋予$2,$2 的值赋予$1
  - typdeset 用于显示或设置变量
    - 是 declare 的同义词
  - unset 用于取消变量的定义
    - -- 表明选项结束
    - -f 删除只读变量，但不能取消从 shell 环境中删除指定的变量和函数。如：PATH，PS1，PS2，PPID，UID，EUID 等的设置。

```
#!/bin/bash
#param

echo "脚本的名字 $0"
echo "parm 1: $1"

echo "parm 2: $2"
echo "parm 3: $3"
echo "parm 4: $4"
```

```
echo "parm 5: $5"

echo "开始 shift 2"
shift 2

echo "parm 1: $1"
echo "parm 2: $2"
```

2.2 引号

简介：引号的必要性、双引号、单引号、反引号、反斜杠

- 引号的必要性
  - 变量和替换操作，在脚本中执行变量替换时，最容易犯的一个错误就是引号的错误。
  - 例如：echo aa.tar.gz * ;　　echo “aa.tar.gz *” 效果大不一样
  - 双引号　使用双引号可以引用除字符$，`反引号，\反斜杠外的任意字符或者字符串。
    - echo -e " hello world, $SHELL '\n* wangbaoming 'echo $myvar' "
  - 单引号　单引号与双引号类似，不同的是 shell 会忽略任何引用值。换句话说，如果屏蔽了其特殊含义，会将引号里的所有字符，包括引号都作为一个字符。
    - echo -e ' hello world, $SHELL '\n* wangbaoming '
    - 结论：单引号让特殊字符失去意义。

```
[test@localhost ~]$
[test@localhost ~]$ echo -e "hello world, $SHELL"
hello world, /bin/bash
[test@localhost ~]$ echo -e 'hello world, $SHELL'
hello world, $SHELL
[test@localhost ~]$
```

  - 反引号　反引号用于设置系统命令的输出到变量。shell 将反引号中的内容作为一个系统命令，并执行其内容。
    - eg：echo “* wangbaoming `echo $PATH`”
  - 反斜杠
    - 如果一个字符有特殊含义，反斜杠防止 shell 误解其含义，即：屏蔽其特殊含义。
    - 下属字符含有特殊含义：$ * + ^ ` “ | ?
    - eg:echo * echo *

2.3 运算符

运算符是对计算机发的指令

运算符对象

数字、字符（子面值）

变量

表达式

表达式　运算符和运输对象的组合体。

表达式类型

- 按位运算符
  - ~op1 反运算符
  - op1<<op2　　左移运算符
  - op1>>op2　　右移运算符
  - op1 & op2　　与比较运算符
  - op1 ^ op2　　异或运算符
  - op1 | op2　　或运算符
  - eg: echo $[2<<4]　echo $[2^4]
- $[ ] 表示形式告诉 shell 对方括号中的表达式求值
  eg: echo $[3+9]
- 逻辑运算符
  - && 逻辑与运算
  - || 逻辑或运算符 echo $[1||1]
- 赋值运算符
  - =，+=，-=，*=，/=，%=，&=，^=、|=，<<=，>>=
  - let count = $count + $change
  - let count += $change

```
[wbm@wmblinux64 myshell]$ var=10
[wbm@wmblinux64 myshell]$ let var+=5
[wbm@wmblinux64 myshell]$ echo $var
15
```

表达式替换

- $[] 和 $(( )) 习惯使用$[]，所有 shell 的求值都是用整数完成
- $[]可以接受不同基数的数字
  - [base#n] n 表示基数从 2 到 36 任意基数
    [wbm@wmblinux64 myshell]$ echo $[10#8+191]
    199

运算符的优先级（不确定的地方，多加括号）

03shell 输入与输出

提纲：　echo　　read　　cat　管道　　tee　文件重定向　标准输入输出和错误合并标准输出和标准错误 exec 和使用文件描述符

目标：终端用户登录 linux 服务器，为每一个终端用户启动一个 shell 程序。

- echo　　echo 命令可以显示文本行或变量，或者把字符串输入到文件。
- echo [option] string

- -e 解析转义字符
- -n 回车不换行，linux 系统默认回车换行 eg: echo –n “abc” ; echo “abc”
- -转义字符（\c（回车不换行） \f（禁止）\t(回车换行) \n（回车换行））

```
#!/bin/bash
#echo
echo -e "demo01.... \n\n\n"
echo "ok"
#echo 空，后面没有任何东西，将要有一个回车换行
echo
echo "demo02.....\n\n\n"
echo "demo03 line" >mylog.txt
```

- read read 语句可以从键盘或文件的某一行文本中读入信息，并将其制复制给一个变量。
  - read var1 var2 …. 若只指定了一个变量，那么 read 将会把所有的输入赋给该变量，直至遇上第一个文件结束符或者回车。如果给了多个变量，他们按照顺序分别赋予不同的变量。shell 将用空格作为变量之间的分隔符。

```
#!/bin/bash
#readme
#注意回车不换行的用法

echo -n "First Name:"
read firstname

echo -n "Last Name:"
read lastname lastname2

echo -e "FirstName: ${firstname}\n"
echo -e "LastName: ${lastname}\n"

echo -e "LastName2: ${lastname2}\n"
```

- cat 是一个简单而通用的命令，可用它显示文件内容、创建文件、还可以用它来显示控制字符。
- cat [options] filename1 … filename2 ….
  - -v 显示控制字符
  - 使用 cat 命令时注意，它不会在文件分页处停下来；它会一下显示完整个文件。如果希望每次显示一页，可以使用 more 命令 或把 cat 命令的输出通过管道传递到另外一个具有分页功能的命令（more、less）中。
  - man cat
  - cat myfile cat myfile1 myfile2 myfile3; cat myfile1 myfile2 > myfile12
  - cat –v dos.txt 显示 dos 文件的^M
  - eg: cat longfile |more ; cat longfile | less; less longfile
- 管道（1）

- 可以通过管道把一个命令的输出传递给另外一个命令做输入。管道用竖线表示
- 格式：命令 1 | 命令 2
- cat myfile | more
- ls –l | grep “myfile”
- df –k | awk ‘{print $1}’ | grep –v “Filesystem 文件系统”
  - df –k 看磁盘空间 查找第一列 去除 filesystem 字符排除掉

- tee 命令把结果输出到标准输出，另一个副本输出到相应文件
  - tee –a file -a: 表示追加 不加-a 表示覆盖
  - 该命令一般用于管道之后 （一般看到输出，并存文件）
  - eg who | tee –a who.out
    - df –k | awk ‘{print $1}’ | grep –v “Filesystem” | tee partation.txt
- 标准输入、输出和错误

  当我们在 shell 中执行命令的时候，每个进程都和三个打开的文件相联系，并使用文件描符来引用这些文件。由于文件描述符不容易记忆， shell 同时也给出了相应的文件名。

  下面就是这些文件描述符及它们通常所对应的文件名：

  输入文件—标准输入 0

  输出文件—标准输出 1

  错误输出文件—标准错误 2

  系统中实际上有 12 个文件描述符，但是正如我们在上表中所看到的，0、1、2 是标准输入、出和错误。可以任意使用文件描述符 3 到 9。

文件重定向

文件重定向：改变程序运行的输入来源和输出地点

command > filename	把标准输出重定向到一个新文件中
command >> filename	把标准输出重定向到一个文件中 (追加)
command 1 > fielname	把标准输出重定向到一个文件中
command > filename 2>&1	把标准输出和标准错误一起重定向到一个文件中
command 2 > filename	把标准错误重定向到一个文件中
command 2 >> filename	把标准输出重定向到一个文件中 (追加)
command >> filename 2>&1	把标准输出和标准错误一起重定向到一个文件中 (追加)
command < filename >filename2	command命令以 filename文件作为标准输入，以 filename2文件作为标准输出
command < filename	command命令以filename文件作为标准输入
command << delimiter	从标准输入中读入，直至遇到 delimiter分界符
command <&m	把文件描述符 m作为标准输入
command >&m	把标准输出重定向到文件描述符 m中
command <&-	关闭标准输入

重定向标准输出

eg：cat file | sort 1 > sort.out

cat file | sort >sort.out 这两个命令等同

pwd >> path.out

nullfile.txt 如果想创建一个长度为 0 的空文件，可以用'>filename'：

重定向标准输入

eg1： sort < file 将 file 的内容输入到 sort 进行排序，排序后的结果 sort 命令输出

sort < name.txt > name.out

eg2:分隔符用法

```
$ cat >> myfile <<MAYDAY
> Hello there I am using a $TERM terminal
> and my user name is $LOGNAME
> bye...
> MAYDAY

$ pg myfile
Hello there I am using a vt100 terminal
and my user name is dave
bye...
```

重定向标准错误

eg：$ grep "trident" missiles 2>/dev/null

在这个例子中，grep 命令在文件 missiles 中搜索 trident 字符串：

结合使用标准输出和标准错误

eg：$ cat account_qtr.doc account_end.doc 1>accounts.out 2>accounts.err

合并标准输出和标准错误

eg: grep “standard” standard.txt >grep.out 2>&1

在上面的例子中，我们将 cleanup 脚本的输出重定向到 cleanup .out 文件中，而且其错误也重定向到相同的文件中。

注意：shell 是从左至右分析相应的命令

```
/home/wbm/myshell
[wbm@wmblinux64 myshell]$ cat file1 file2 file3 > err.out 2<&1
[wbm@wmblinux64 myshell]$ cat err.out
111
1111
111111
222
22222
2222222
cat: file3: 没有那个文件或目录
[wbm@wmblinux64 myshell]$
```

exec 和文件描述符在一起

1、 exec 命令可以用来代替当前 shell；换句话说，并没有启动子 shell，使用这一命令时，任何现有环境都将被清除，并重新启动一个 shell。

exec command command 通常是一个 shell 脚本。

Eg：测试执行完该命令后，需要重新登录 shell，然后 export 声明的变量都失效了。exec ./hello.sh

2、对文件描述符进行操作时，也只有这个时，它不会覆盖你的当前的 shell。

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

```
#
exec 3<&0 0<name.txt
read line1
read line2
exec 0<&3

echo $line1
echo $line2
```

04 控制流结构

简介：控制结构　if then else 语句　case 语句　for 循环　until 循环
while 循环　break 控制　confinue 控制

4.0 流控制是什么

```
#!/bin/bash
#创建一个目录
    make /home/wbm/shell/txt
#复制所有 txt 文件到指定目录
    cp *.txt /home/wbm/shell/txt
    rm -f *txt
```

上述脚本会出现问题吗？
如果目录创建失败或成功如何处理
文件拷贝失败如何处理

4.1 条件测试

有时判断字符串是否相等或检查文件状态或是数字测试等。Test 命令用于测试字符串，文件状态和数字。

4.1.1 文件状态测试

格式　test condition　或 [condition]
使用方括号时，要注意在条件两边加上空格。

文件测试状态：测试的结果可以用$?的值来判断，0 表示成功，其他为失败

-d	目录	-s	文件长度大于 0、非空
-f	正规文件	-w	可写
-L	符号链接	-u	文件有 suid 位设置
-r	可读	-x	可执行

-z　　字符串为空

脚本测试

```
[wbm@wmblinux64 041]$ ls
01test.sh    1    iftest.sh
[wbm@wmblinux64 041]$ test -d dir
[wbm@wmblinux64 041]$ echo $?
```

脚本练习：

```
#!/bin/bash
echo "test use 1"
test -w tmp.txt
echo $?
echo "test use 2 [] begin"
[ -w tmp.txt ]
echo $?
```

对上面 bash 进行 if 控制

```
#!/bin/bash
echo "test use 1"
test -w tmp.txt
if [ $? -eq "0" ]; then
        echo "success \n";
else
        echo "net success";
fi
echo $?
echo "test use 2 [] begin"
[ -w tmp.txt ]
echo $?
```

要测试其他的类似.

测试时使用逻辑操作符

-a　　逻辑与，操作符两边均为真，结果为真，否则为假。

-o　　逻辑或，操作符两边一边为真，结果为真，否则为假。

!　　逻辑否，条件为假，结果为真。

下面的例子测试两个文件 xaa 和 xab 是否均可读.

```
[root@localhost test]# [ -w xab -a -w xac ]
[root@localhost test]# echo $?
0
```

测试 xac 是否可执行或者 xab 是否可写,使用逻辑或操作

```
[root@localhost test]# [ -w xab -o -x xac ]
[root@localhost test]# echo $?
0
```

测试 xac 是否可执行而且 xab 是否可写,使用逻辑或操作

```
[root@localhost test]# [ -w xab -a -x xac ]
[root@localhost test]# echo $?
1
```

4.1.2 字符串测试

格式

```
test "string"
test string_operator "string"
test "string" string_operator "string"
[ string_operator string ]
[ string string_operator string ]
```

使用方括号时，要注意在条件两边加上空格。

测试两个字符是否相等。退出状态变量 $?，0 表示成功，1 表示失败。

操作数 string_operator

= 两个字符串相等

!= 两个字符串不等

-z 空串

-n 非空串

练习 1：测试环节变量是否为空

```
[root@localhost test]# echo $EDITOR

[root@localhost test]# [ -z $EDITOR ]
[root@localhost test]# echo $?
0
```

练习 2：测试字符串是否相等

```
[root@localhost test]# AAA="aaa"
[root@localhost test]# BBB="bbb"
[root@localhost test]# AAA1="aaa"
[root@localhost test]# [ "$AAA" = "$BBB" ]
[root@localhost test]# echo $?
1
[root@localhost test]# [ "$AAA" = "$AAA" ]
[root@localhost test]# echo $?
0
```

4.1.3 测试数值

格式 测试两个数值大小

"number" numberic_operator "number"

或者

["number" numberic_operator "number"]

numberic_operator 可为:

-eq 数值相等。

-ne 数值不相等。

-gt 第一个数大于第二个数。

-lt 第一个数小于第二个数。
-le 第一个数小于等于第二个数。
-ge 第一个数大于等于第二个数。

```
[root@localhost test]# NUMBER=130
[root@localhost test]# [ "$NUMBER" -eq "130" ]
[root@localhost test]# echo $?
0
```

4.1.4 expr 数字运算

```
expr argument operator argument
```

加法运算：expr 10 + 10
减法运算：expr 20 - 10
加法运算：expr 10 / 5
乘法运算：expr 10 * 5

```
[root@localhost test]# expr 10 + 10
20
[root@localhost test]# expr 30/3
30/3
[root@localhost test]# expr 30 / 3
10
[root@localhost test]# expr 30 / 3 / 2
5
[root@localhost test]# expr 30 \* 3
90
```

如果是非数字参加运算会报错误，利用此点可以用来测试数字格式。
root@localhost opt]# if expr a + 100; then echo aaaa; else echo bbbb; fi;
expr: non-numeric argument
bbbb
没有达到预期效果:而且使用乘号时也要用反斜线屏蔽其特殊意义

4.1.5 理论提高 man test

4.2 if then else 语句

语法 1
if 条件
then 命令
fi
注意 if 语句必须以 fi 终止
练习：
#if test
if ["13" -lt "12"] # "13" 前一个空格，“13”后也有一个空格。

```
then
        echo "yes 13 is less then 12"
else
        echo "NO"
fi
```

语法 2

```
if 条件 1
then
    命令 1
elif 条件 2
then
    命令 2
else
    命令 3
Fi
```

综合练习

```
#!/bin/bash
#if test
#this is a comment line
echo "Enter your filename:"
read myfile
if [ -e $myfile ]
then
  if [ -s $myfile ];then
    echo "$myfile exist and size greater than zero"
  else
    echo "$myfile exist but size is zero"
  fi
else
echo "file no exist"
fi
```

4.3case 多选择语句

case 多选择语句格式：

```
case 值 in
模式 1)
    命令 1
    ;;
模式 2)
    命令 2
```

```
    ;;
esac
```

1)case 取值后面必须为单词 in；每一模式必须以右括号结束。

2)取值可以为变量或常数。匹配发现取值符合某一模式后，其后的所有命令开始执行，直到;;

3)模式匹配*表示任意字符; ?表示任意单字符; [..]表示类或范围中任意字符

练习：

```
#!/bin/bash
#case select
echo -n "enter a number from 1 to 3:"
read num
case $num in
1)
        echo "you select 1"
        ;;
2)
        echo "you select 2"
        ;;
3)
        echo "you select 3"
        ;;
y|Y)
        echo “you select y”
        ;;
*)
        echo "`basename $0`:this is not between 1 and 3">&2
        exit;
        ;;
esac
```

4.4for 循环

语法格式：

```
for 循环一般格式
for  变量名  in  列表
do
    命令 1
    命令 2
done
```

1 当变量值在列表里，for 循环即执行依次所有命令，使用变量名访问列表中取值。

2 命令可为任何有效的 shell 命令和语句。变量名为任意单词。

3 in 列表用法是可选择，如果不用它，for 循环使用命令行的位置参数。

4 in 列表可以包含替换、字符串和文件名

练习：1

```
#!/bin/bash

for i in 1 2 3 4 5
do
        echo $i
done
```

练习：2

```
#!/bin/bash

for i    in "aaa bbb ccc "
do
            echo $i
done
echo "你看出来效果不一样了吗"
for i in aaa bbb ccc
do
            echo $i
done
```

练习 3 注意 in 后为 命令 反引号

```
#!/bin/bash
for loop in `cat myfile`
do
        echo $i
done
```

注意：打印文件内容按照 行+空格 为一行

练习 4：如果 in 后没有语句，此时语句等价于： in $@

```
#!/bin/bash
#for_noin03.sh
i=1
for param
do
            echo "param #$I is $param"
            i=$i+1
done
```

测试：[wbm@wmblinux64 041]$ ./for_noin03.sh aa bbb ccc

4.5until 循环

语法

until 循环格式

```
until  条件
do
        命令 1
        命令 2
done
```

条件可以为任意测试条件，测试发生在循环末尾，因此循环至少执行一次。

练习：

```
#!/bin/sh
#until_mom  监控分区
Part="/backup"
Look_Out=`df | grep "$Part" | awk '{printf $5}' | sed 's/%//g' `
echo $Look_Out;
until [ "$Look_Out" -gt "90" ]
do
        echo -e "Filesystem $Part is nearly full " | mail root
        Look_Out=`df | grep "$Part" | awk '{printf $5}' | sed 's/%//g' `
        sleep 3600
done
```

练习 2：

```
#!/bin/bash
#until_mom  枷锁文件

until [ ! -f a.lock ]
do
            echo "check a.lock exist"
            sleep 2
done
echo "you may start another application safely"
```

执行：nohup ./until02.sh &

4.6while 循环

语法

```
while  命令 （可以是一个命令也可以是多个，做条件测试）
do
```

```
        命令 1
        命令 2
        …
done
```

注：在 while 和 do 之间虽然通常是一个命令，但可以放几个命令。

命令通常用作测试条件。

练习 1：

```
[wbm@wmblinux64 041]$ vi while01.sh

#!/bin/bash
#while01
echo "ctrl+d quit"
while echo -n "please enter you name:"; read Name
do
        echo "Yeah, you name: $Name";
done
```

练习 2：从文件中按照行读数据

```
#!/bin/bash
#while02
while read LINE
do
        echo $LINE
done < names.txt
```

```
#!/bin/bash
#while02
#这样写是不对的 ， 需要把<name.txt 放在 while 循环后面
while read LINE < names.txt
do
        echo $LINE
done
```

备注：如果从文件中读入变量<filename 要放到 done 后

4.7break 和 continue

break [n]

退出循环

如果是在一个嵌入循环里，可以指定 n 来跳出循环个数

continue

跳出循环步

注意：continue 命令类似于 break 命令，只有一点重要差别，它不会跳出循环，只是跳出这

个循环步

总结：break 跳出 continue 跳过

```
#!/bin/bash
#breakout
while    :

do
        echo -n "Enter any num[1...5]:"
        read num
        case $num in
        1|2|3|4|5)
                echo "You enter a num between 1 and 5"
                ;;
        *)
                echo "Wrong num, bye"
                break;
                ;;
        esac
done
```

```
#!/bin/bash
#breakout
while    :

do
        echo -n "Enter any num[1...5]:"
        read num
        case $num in
        1|2|3|4|5)
                echo "You enter a num between 1 and 5"
                ;;
        *)
                echo -n "Wrong num, continue (y/n)?: "
                read IS_CONTINUE
                case $IS_CONTINUE in
                        y|yes|Yes)
                                continue
                                ;;
                        *)
                                break;
                                ;;
                esac
        esac
```

done

05 文本过滤

简介：正则表达式、find 介绍、grep 介绍、awk 介绍、sed 介绍、合并与分割（sort、uniq、join、cut、paste、split）。

5.1 正则表达式

概念：一种用来描述文本模式的特殊语法
由普通字符（例如：字符 a 到 z），以及特殊字符（元字符，如/ * ？等）组成匹配的字符串
文本过滤工具在某种模式之下，都支持正则表达式。

5.2 基本元字符集及其含义

^	只匹配行首
$	只匹配行尾
*	一个单字符后紧跟*，匹配0个或多个此单字符
[ ]	匹配[]内字符。可以是一个单字符，也可以是字符序列。可以使用 - 表示[]内字符序列范围，如用[1-5]代替[12345]
\	用来屏蔽一个元字符的特殊含义。因为有时在 shell中一些元字符有特殊含义。\可以使其失去应有意义
.	匹配任意单字符
pattern\{n\}	用来匹配前面 pattern出现次数。n为次数
pattern\{n，\}m	含义同上，但次数最少为 n
pattern\{n，m\}	含义同上，但 pattern出现次数在n与m之间

练习
A\{3\}B　　AAAB
A\{3,\}B AAAB AAAAB ...
A\{3,5\}B AAAB AAAAB AAAAAB

5.3 使用句点匹配单字符

句点“.”可以匹配任意单字符。例如，如果要匹配一个字符串，以 beg 开头，中间夹一个意字符，那么可以表示为 b e g . n，　“.”可以匹配字符串头，也可以是中间任意字符。

在 ls -l 命令中，可以匹配一定权限：

在ls -l命令中，可以匹配一定权限：

```
...x..x..x
```

此格式匹配用户本身，用户组及其他组成员的执行权限。

drwxrwxrw- – no match
-rw-rw-rw- – no match
-rwx-rwxr-x – match
-rwx-r-x-r-x – match

假定正在过滤一个文本文件，对于一个有 10个字符的脚本集，要求前 4个字符之后为XC，匹配操作如下：

```
....xc....
```

以上例子解释为前4个字符任意，5，6字符为XC，后4个字符也任意，按下例运行：

1234XC9088 – match
4523XX9001 – no match
0011XA9912 – no match
9931XC3445 – match

注意，“. ”允许匹配ASCII集中任意字符，或为字母，或为数字。

5.4 行首以^匹配字符串或字符序列

^只允许在一行的开始匹配字符或单词。例如，使用 ls -l 命令，并匹配目录。之所以可以样做是因为 ls -l 命令结果每行第一个字符是 d，即代表一个目录。

```
^d
```

drwxrwxrw- – match
-rw-rw-rw- – no match
drwx-rwxr-x – match
-rwx-r-x-r-x – no match

回到脚本（1），使用^001，结果将匹配每行开始为001的字符串或单词：

1234XC9088 – no match
4523XX9001 – no match
0011XA9912 – match
9931XC3445 – no match

可以将各种模式结合使用，例如：

```
^...4xc....
```

结果为：

1234XC9088 – match
4523XX9001 – no match
0011XA9912 – no match
9931XC3445 – no match
3224XC193 – no match

5.5 行尾以$匹配字符串或字符

可以说$与^正相反，它在行尾匹配字符串或字符， $符号放在匹配单词后。假定要匹配以词 t r o u b l e 结尾的所有行，操作为：trouble$

^$ 匹配空行

^.$ 匹配包含一个字符的行

将匹配字符u一次或多次：

computer
computing
compuuuuute

5.6 用*$匹配单字符串或其重复序列

将匹配字符u一次或多次：

computer
computing
compuuuuute

5.7 使用\屏蔽一个特殊字符的含义

$. ' " * [ ] ^ | ( ) \ + ?

假定要匹配包含字符“.”的各行而“,”代表匹配任意单字符的特殊字符，因此需要屏蔽其含义。操作如下：

\.

上述模式不认为反斜杠后面的字符是特殊字符，而是一个普通字符，即句点。

假定要匹配包含^的各行，将反斜杠放在它前面就可以屏蔽其特殊含义。如下：

\^

如果要在正则表达式中匹配以*.pas结尾的所有文件，可做如下操作：

*\.pas

即可屏蔽字符*的特定含义。

5.8 使用[]匹配一个范围或集合

假定要匹配任意一个数字，可以使用：

[0123456789]

然而，通过使用“-”符号可以简化操作：

[0 - 9]

或任意小写字母

[a - z]

要匹配任意字母，则使用：

[A - Z a - z]

表明从 A-Z、a-z 的字母范围。

如要匹配任意字母或数字，模式如下：

[A - Z a - z 0 - 9]

在字符序列结合使用中，可以用 []指出字符范围。假定要匹配一单词，以 s 开头，中间有任意字母，以 t 结尾，那么操作如下：

s[a-z A-Z]t

上述过程返回大写或小写字母混合的单词，如仅匹配小写字母，可使用：

s [a - z] t

如要匹配 Computer 或 computer 两个单词，可做如下操作：

[C c] o m p u t e r

为抽取诸如 Scout、shout、bought 等单词，使用下列表达式：

[ou] .*t

匹配以字母 o 或 u 开头，后跟任意一个字符任意次，并以 t 结尾的任意字母。

也许要匹配所有包含 system 后跟句点的所有单词，这里 S 可大写或小写。使用如下操作：

[S,s] y s t e m \ .

[]在指定模式匹配的范围或限制方面很有用。结合使用 *与[]更是有益，例如[A-Za-Z]*将匹配所有单词。

[A - Z a - z] *

注意^符号的使用，当直接用在第一个括号里，意指否定或不匹配括号里内容。

[^a-zA-Z]

匹配任一非字母型字符，而

[^ 0 - 9]

匹配任一非数字型字符。

通过最后一个例子，应可猜知除了使用^，还有一些方法用来搜索任意一个特殊字符。

5.8 使用\{\}匹配模式结果出现的次数

使用*可匹配所有匹配结果任意次，但如果只要指定次数，就应使用 \{\}，此模式有三种形式，即：

pattern\{n\} 匹配模式出现 n 次。

pattern\{n,\} 匹配模式出现最少 n 次。

pattern\{n,m} 匹配模式出现 n 到 m 次之间，n , m 为 0-255 中任意整数。

请看第一个例子，匹配字母 A 出现两次，并以 B 结尾，操作如下：

A \ { 2 \ } B ；A 出现 2 次匹配值为 AAB

A \ { 4 , \ } B；匹配 A 至少 4 次

可以得结果 AAAAB 或 AAAAAAAB，但不能为 AAAB。

如给出出现次数范围，例如 A 出现 2 次到 4 次之间：

A \ { 2 , 4 \ } B

则结果为 AAB、AAAB、AAAAB，而不是 AB 或 AAAAAB 等。

[0-9]\{3\}\.[0-9]\{3\}\.[0-9]\{3\}\.[0-9]\{3\}; 匹配 IP 地址

5.2find 和 Xargs

find 的常用功能

1、find 功能，查找目录或文件
查找具有某些特征文件的命令。
可遍历当前目录甚至于整个文件系统来查找某些文件或目录。
遍历大文件系统时，要放在后台执行

2、find 命令格式
find pathname -options [-print -exec -ok ...]

3、find 命令参数
pathname: find 命令所查找的目录路径。例如用.来表示当前目录，用/来表示系统根目录。
-print： find 命令将匹配的文件输出到标准输出。
-exec： find 命令对匹配的文件执行该参数所给出的 shell 命令。相应命令的形式为'command' { } \;，注意{ }和\;之间的空格。
-ok： 和-exec 的作用相同，只不过以一种更为安全的模式来执行该参数所给出的 shell 命令，在执行每一个命令之前，都会给出提示，让用户来确定是否执行。

4、find 命令选项
-name 按照文件名查找文件。
-perm 按照文件权限来查找文件。
-prune 使用这一选项可以使 find 命令不在当前指定的目录中查找,如果同时使用-depth 选项,那么-prune 将被 find 命令忽略。
-user 按照文件属主来查找文件。

-group 按照文件所属的组来查找文件。
-mtime -n +n
按照文件的更改时间来查找文件， - n 表示文件更改时间距现在 n 天以内，+ n 表示文件更改时间距现在 n 天以前。find 命令还有-atime 和-ctime 选项，但它们都和-m time 选项。
-nogroup 查找无有效所属组的文件，即该文件所属的组在/etc/groups 中不存在。
-nouser 查找无有效属主的文件，即该文件的属主在/etc/passwd 中不存在。
-newer file1 ! file2
查找更改时间比文件 file1 新但比文件 file2 旧的文件。
-type
查找某一类型的文件，诸如：
b - 块设备文件。
d - 目录。
c - 字符设备文件。
p - 管道文件。
l - 符号链接文件。
f - 普通文件。
-size n：[c] 查找文件长度为 n 块的文件，带有 c 时表示文件长度以字节计。
-depth：在查找文件时，首先查找当前目录中的文件，然后再在其子目录中查找。

-fstype：查找位于某一类型文件系统中的文件，这些文件系统类型通常可以在配置文件/etc/fstab 中找到，该配置文件中包含了本系统中有关文件系统的信息。
-mount：在查找文件时不跨越文件系统 mount 点。
-follow：如果 find 命令遇到符号链接文件，就跟踪至链接所指向的文件。
-cpio：对匹配的文件使用 cpio 命令，将这些文件备份到磁带设备中。

另外,下面三个的区别:

-amin n
查找系统中最后 N 分钟访问的文件
-atime n
查找系统中最后 n*24 小时访问的文件
-cmin n
查找系统中最后 N 分钟被改变文件状态的文件
-ctime n
查找系统中最后 n*24 小时被改变文件状态的文件
-mmin n
查找系统中最后 N 分钟被改变文件数据的文件
-mtime n
查找系统中最后 n*24 小时被改变文件数据的文件

重点训练
文件名选项是 find 命令最常用的选项，要么单独使用该选项，要么和其他选项一起使用。
以使用某种文件名模式来匹配文件，记住要用引号将文件名模式引起来。
不管当前路径是什么，如果想要在自己的根目录 $HOME 中查找文件名符合*.txt 的文件，
用~作为'pathname 参数，波浪号~代表了你的$HOME 目录。
$ find ~ -name "*.txt" -print
想要在当前目录及子目录中查找所有的‘*.txt’文件，可以用：
$ find . -name "*.txt" -print
想要的当前目录及子目录中查找文件名以一个大写字母开头的文件，可以用：
$ find . -name "[A-Z]*" -print
想要在/etc 目录中查找文件名以 host 开头的文件，可以用：
$ find /etc -name "host*" -print
想要查找$HOME 目录中的文件，可以用：
$ find ~ -name "*" -print 或 find . -print
要想让系统高负荷运行，就从根目录开始查找所有的文件。如果希望在系统管理员那里
留一个好印象的话，最好在这么做之前考虑清楚！
$ find / -name "*" -print
如果想在当前目录查找文件名以两个小写字母开头，跟着是两个数字，最后是 *.txt 的文
，下面的命令就能够返回名为 ax37.txt 的文件：
$ find . -name "[a-z][a-z][0--9][0--9].txt" –print

希望在系统根目录下查找更改时间在 5 日以内的文件，可以用：
$ find / -mtime -5 -print
为了在/var/adm 目录下查找更改时间在 3 日以前的文件，可以用：

$ find /var/adm -mtime +3 –print
次重点训练： 使用 exec 或 ok 来执行 shell 命令 find ./ -name "*.sh" -exec ls -l {} \;

find 一般练习

二、find 命令的例子；
1、查找当前用户主目录下的所有文件：
下面两种方法都可以使用
$ find $HOME -print
$ find ~ -print
2、让当前目录中文件属主具有读、写权限，并且文件所属组的用户和其他用户具有读权限的文件；
$ find . -type f -perm 644 -exec ls -l { } \;
3、为了查找系统中所有文件长度为 0 的普通文件，并列出它们的完整路径；
$ find / -type f -size 0 -exec ls -l { } \;
4、查找/var/logs 目录中更改时间在 7 日以前的普通文件，并在删除之前询问它们；
$ find /var/logs -type f -mtime +7 -ok rm { } \;
5、为了查找系统中所有属于 root 组的文件；
$find . -group root -exec ls -l { } \;
-rw-r--r-- 1 root root 595 10 月 31 01:09 ./fie1
6、find 命令将删除当目录中访问时间在 7 日以来、含有数字后缀的 admin.log 文件。
该命令只检查三位数字，所以相应文件的后缀不要超过 999。先建几个 admin.log*的文件 ，才能使用下面这个命令
$ find . -name "admin.log[0-9][0-9][0-9]" -atime -7 -ok
rm { } \;
< rm ... ./admin.log001 > ? n
< rm ... ./admin.log002 > ? n
< rm ... ./admin.log042 > ? n
< rm ... ./admin.log942 > ? n
7、为了查找当前文件系统中的所有目录并排序；
$ find . -type d | sort
8、为了查找系统中所有的 rmt 磁带设备；
$ find /dev/rmt -print

find 和 xargs 在一起

xargs - build and execute command lines from standard input 在使用 find 命令的-exec 选项处理匹配到的文件时， find 命令将所有匹配到的文件一起传递

给 exec 执行。但有些系统对能够传递给 exec 的命令长度有限制，这样在 find 命令运行几分钟之后，就会出现 溢出错误。错误信息通常是“参数列太长”或“参数列溢出”。这就是 xargs 命令的用处所在，特别是与 find 命令一起使用。

find 命令把匹配到的文件传递给 xargs 命令，而 xargs 命令每次只获取一部分文件而不是全部，不像-exec 选项那样。这样它可以先处理最先获取的一部分文件，然后是下一批，并如此继续下去。

在有些系统中，使用-exec 选项会为处理每一个匹配到的文件而发起一个相应的进程，并非将匹配到的文件全部作为参数一次执行；这样在有些情况下就会出现进程过多，系统性能下降的问题，因而效率不高；

而使用 xargs 命令则只有一个进程。另外，在使用 xargs 命令时，究竟是一次获取所有的参数，还是分批取得参数，以及每一次获取参数的数目都会根据该命令的选项及系统内核中相应的可调参数来确定。
来看看 xargs 命令是如何同 find 命令一起使用的，并给出一些例子。

在当前目录下查找所有用户具有读、写和执行权限的文件，并收回相应的写权限：

```
# ls -l
drwxrwxrwx      2 sam       adm             4096 10 月  30 20:14 file6
-rwxrwxrwx      2 sam       adm                0 10 月  31 01:01 http3.conf
-rwxrwxrwx      2 sam       adm                0 10 月  31 01:01 httpd.conf

# find . -perm -7 -print | xargs chmod o-w
# ls -l
drwxrwxr-x      2 sam       adm             4096 10 月  30 20:14 file6
-rwxrwxr-x      2 sam       adm                0 10 月  31 01:01 http3.conf
-rwxrwxr-x      2 sam       adm                0 10 月  31 01:01 httpd.conf
```

用 grep 命令在所有的普通文件中搜索 hostname 这个词：

```
# find . -type f -print | xargs grep "hostname"
./httpd1.conf:#         different IP addresses or hostnames and have them handled by the
./httpd1.conf:# VirtualHost: If you want to maintain multiple domains/hostnames
on your
```

用 grep 命令在当前目录下的所有普通文件中搜索 hostnames 这个词：

```
# find . -name \* -type f -print | xargs grep "hostnames"
./httpd1.conf:#         different IP addresses or hostnames and have them handled by the
./httpd1.conf:# VirtualHost: If you want to maintain multiple domains/hostnames
on your
```

注意，在上面的例子中， \用来取消 find 命令中的*在 shell 中的特殊含义。

find 命令配合使用 exec 和 xargs 可以使用户对所匹配到的文件执行几乎所有的命令。

5.3grep 命令

Grep 命令功能：按照行方式处理文本。。。
grep 一般格式为：
grep [选项]基本正则表达式[文件]

Grep 参数
-c 只输出匹配行的计数。
-I 不区分大小写（只适用于单字符）。
-h 查询多文件时不显示文件名。
-l 查询多文件时只输出包含匹配字符的文件名。
-n 显示匹配行及行号。
-s 不显示不存在或无匹配文本的错误信息。
-v 显示不包含匹配文本的所有行。

多文件查找
$ grep "sort"*.doc
或在所有文件中查询单词“ sort it”
$ grep "sort it" *

$ grep -c "48"data.f
$ 4
grep 返回数字 4，意义是有 4 行包含字符串“48”

显示满足匹配模式的所有行行数
[root@localhost /]# grep -n "48" data.f
1:48 Dec 3BC1997 LPSX 68.00 LVX2A 138
2:483 Sept 5AP1996 USP 65.00 LVX2C 189
5:484 nov 7PL1996 CAD 49.00 PLV2C 234
6:483 may 5PA1998 USP 37.00 KVM9D 644

显示不匹配的行
[root@localhost /]# grep -v "48" data.f
47 Oct 3ZL1998 LPSX 43.00 KVM9D 512
219 dec 2CC1999 CAD 23.00 PLV2C 68
216 sept 3ZL1998 USP 86.00 KVM9E 234

大小写敏感
[root@localhost /]# grep -i "sept" data.f
483 Sept 5AP1996 USP 65.00 LVX2C 189
216 sept 3ZL1998 USP 86.00 KVM9E 234
正则表达式模式匹配

不匹配行首 如果要抽出记录，使其行首不是 48，可以在方括号中使用^记号，表明查询在行首开始。

```
[root@localhost /]# grep '^[^48]' data.f
219     dec     2CC1999        CAD     23.00    PLV2C    68
216     sept    3ZL1998        USP     86.00    KVM9E    234
```

如果抽取以 K 开头，以 D 结尾的所有代码，可使用下述方法，因为已知代码长度为 5 个字符：

```
[root@localhost /]# grep 'K...D' data.f
47          Oct         3ZL1998              LPSX       43.00      KVM9D      512
483         may         5PA1998               USP         37.00      KVM9D      644
```

使用 grep 匹配“与”或者“或”模式

```
[root@localhost /]# grep -E '216|219' data.f
219         dec         2CC1999              CAD         23.00      PLV2C      68
216         sept        3ZL1998              USP         86.00      KVM9E      234
```

Grep 命令原理

GREP 是 Global Regular Expression Print 的缩写

对于标准输入的每一行,grep 执行以下的操作:

(1) 把下一输入行复制到模式空间中. 模式空间是只可保存一个文本行的缓冲区.

(2) 对模式空间应用正则表达式.

(3) 如果有匹配存在,该行从模式空间中被复制到标准输出.

grep 实用程序对输入的每行重复这三个操作步骤.

注意 grep 处理下列情况的方式:

1.grep 是一个搜索程序,它只能搜索匹配一个正则表达式的一行的存在性.

2.grep 可以对一行采取唯一的动作是把它发送到标准输出. 如果该行不匹配正则表达式,则其不被打印.

3.行的选择只基于正则表达式. 行编号或其他准则不能用于选择行.

4.grep 是一个过滤器. 它可用在管道的左边或右边.

5.grep 不能用于增加,删除或修改行.

6.grep 不能用于只打印行的一部分.

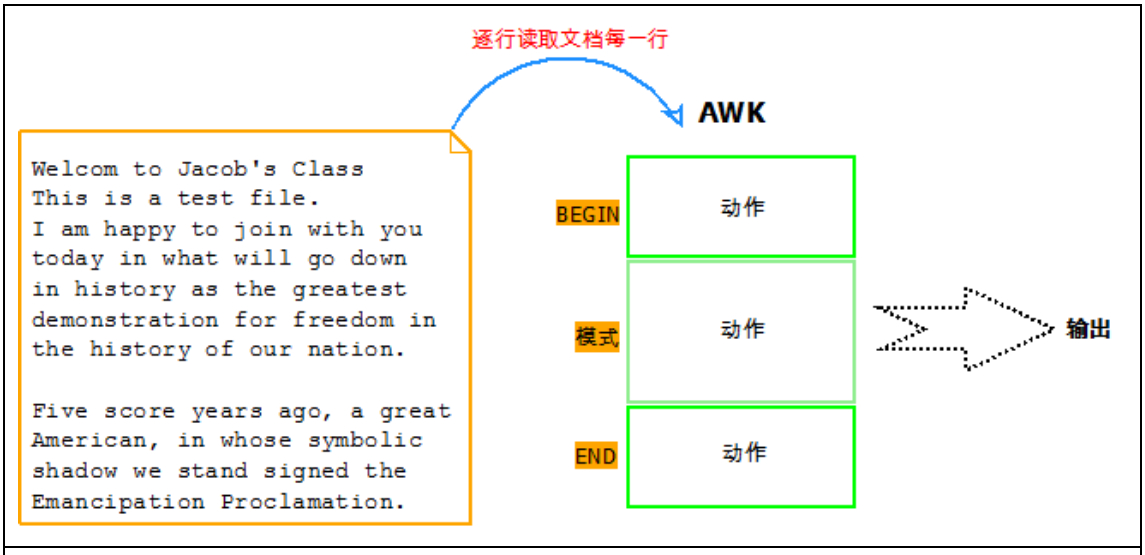
7.grep 不能只读取文件的一部分.

8.grep 不能基于前面的内容或下一行来选择一行.只有一个缓冲区,它只保存当前行.

5.4awk 命令

1 awk 语法及工作原理

awk 可从文件或字符串值基于指定规则浏览和抽取信息 有三种方式调用 a w k， 第一种是命令行方式，如： 这里，c o m m a n d s 是真正的 a w k 命令。本章将经常使用这种方法。 上面例子中，[- F 域分隔符]是可选的，因为 a w k 使用空格作为缺省的域分隔符，因此如果浏览域间有空格的文本，不必指定这个选项，但如果要浏览诸如 p a s s w d 文件，此文件各域 冒号作为分隔符，则必须指明- F 选项，如： 第二种方法是将所有 a w k 命令插入一个文件，并使 a w k 程序可执行，然后用 a w k 命令解释 作为脚本的首行，以便通过键入脚本名称来调用它。 第三种方式是将所有的 a w k 命令插入一个单独文件，然后调用：
awk 是一种编程语言，用于在 linux/unix 下对文本和数据进行扫描与处理。数据可以来自标准输入、文件、管道。 awk 分别代表其作者姓氏的第一个字母。因为它的作者是三个人，分别是 Alfred Aho、Peter Weinberger、Brian Kernighan。 实际上 awk 有很多种版本，如：awk、nawk、mawk、gawk、MKS awk、tawk... 这其中有开源产品也有商业产品。 目前在 Linux 中常用的 awk 编译器版本有 mawk,gawk，其中以 RedHat 为代表使用的是 gawk，以 Ubuntu 为代表使用的是 mawk。 gawk 是 GNU Project 的 awk 解释器的开源代码实现。
2 原理 1). awk 逐行扫描文件，从第一行到最后一行，寻找匹配特定模式的行，并在这些行上进行你想要的操作。 2). awk 基本结构包括模式匹配(用于找到要处理的行)和处理过程(即处理动作)。 pattern {action} 3). awk 有两个特殊的模式：BEGIN 和 END，他们被放置在没有读取任何数据之前以及在所有数据读取完成以后执行。



2 awk 的 3 种调用方式

3 awk 模式动作、域和记录、字段分隔符

有三种方式调用awk，第一种是命令行方式，如：

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

这里，commands是真正的awk命令。本章将经常使用这种方法。

上面例子中，[-F域分隔符]是可选的，因为awk使用空格作为缺省的域分隔符，因此如果要浏览域间有空格的文本，不必指定这个选项，但如果要浏览诸如 passwd文件，此文件各域以冒号作为分隔符，则必须指明 -F选项，如：

awk -F: 'commands' input-file

第二种方法是将所有 awk命令插入一个文件，并使 awk程序可执行，然后用 awk命令解释器作为脚本的首行，以便通过键入脚本名称来调用它。

第三种方式是将所有的awk命令插入一个单独文件，然后调用：

awk -f awk-script-file input-files(s)

-f选项指明在文件 awk_script_file中的awk脚本，input_file(s)是使用awk进行浏览的文件名。

4awk 应用举例

```
$ awk '{ print $0 }' /etc/passwd
```

$ awk -F":" '{ print $1 }' /etc/passwd
$ awk -F":" '{ print $1 " " $3 }' /etc/passwd
$ awk -F":" '{ print "username: " $1 "/t/tuid:" $3" }' /etc/passwd
[wbm@localhost ~]# awk -F: '$3>500 {print $1}' /etc/passwd 备注：列出计算机中 ID 号大于 500 的用户名

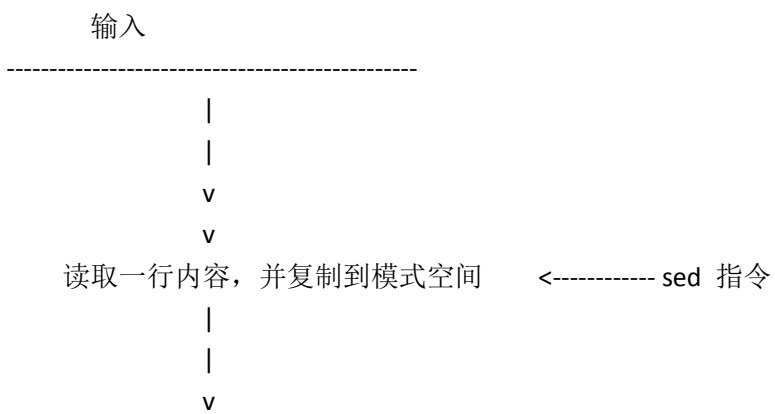
awk 条件及循环语句
例子：从*.sh 中查找 case 出现的文件及行号，分开打印。
Awk 内置函数

gsub(r,s)	在整个$0中用s替代r
gsub(r,s,t)	在整个t中用s替代r
index(s,t)	返回s中字符串t的第一位置
length(s)	返回s长度
match(s,r)	测试s是否包含匹配r的字符串
split(s,a,fs)	在fs上将s分成序列a
sprint(fmt,exp)	返回经fmt格式化后的exp
sub(r,s)	用$0中最左边最长的子串代替s
substr(s,p)	返回字符串s中从p开始的后缀部分
substr(s,p,n)	返回字符串s中从p开始长度为n的后缀部分

5.5sed 命令

sed 命令语法

1.sed 是一款流编辑工具，用来对文本进行过滤与替换工作，特别是当你想要对几十个配置文件做统计修改时，你会感受到 sed 的魅力！
sed 通过输入读取文件内容，但一次仅读取一行内容进行某些指令处理后输出，所以 sed 更适合于处理大数据文件。
.sed 流程：
* 通过文件或管道读取文件内容。
* sed 并不直接修改源文件，而是将读入的内容复制到缓冲区中，我们称之为模式空间（pattern space）。
* 根据 sed 的指令对模式空间中的内容进行处理并输出结果，默认输出至标准输出即屏幕上。



v

输出经过处理后的内容

二、sed 基本语法结构

sed Options... [script] [inputfile...]

sed 选项... [脚本指令] [输入文件]

如果没有输入文件，则 sed 默认对标准输入进行处理（即键盘输入）。脚本指令是第一个不以“-”开始的参数。

1.选项含义：

--version 显示 sed 版本。

--help 显示帮助文档。

-n,--quiet,--silent 静默输出，默认情况下，sed 程序在所有的脚本指令执行完毕后，将自动打印模式空间中的内容，这些选项可以屏蔽自动打印。

-e script 允许多个脚本指令被执行。

-f script-file,

--file=script-file 从文件中读取脚本指令，对编写自动脚本程序来说很棒！

-i,--in-place 直接修改源文件，经过脚本指令处理后的内容将被输出至源文件（源文件被修改）慎用！

-l N, --line-length=N 该选项指定 l 指令可以输出的行长度，l 指令用于输出非打印字符。

--posix 禁用 GNU sed 扩展功能。

-r, --regexp-extended 在脚本指令中使用扩展正则表达式

-s, --separate 默认情况下，sed 将把命令行指定的多个文件名作为一个长的连续的输入流。GNU sed 则允许把他们当作单独的文件， 这样如正则表达式则不进行跨文件匹配。

-u, --unbuffered 最低限度的缓存输入与输出。

sed 应用举例

[wbm@wmblinux64 05sed]$ sed -n 'p' test.txt
aaaa=111
bbbb=222

cccc=3333

dddd=44444
[wbm@wmblinux64 05sed]$
包括空行

[wbm@wmblinux64 05sed]$ sed -n '3,/ddd/p' testfile.txt 从第三行开始匹配，打印到含有 ddd

的行

[wbm@wmblinux64 05sed]$ sed -n '/^$/=' testfile.txt　打印空行
3
5
6

[wbm@wmblinux64 05sed]$ sed -n -e '/^$/p' -e '/^$/=' testfile.txt 即打印空行又打印行号

3

5

6

以上仅是 sed 程序本身的选项功能说明
这里就简单介绍几个脚本指令操作作为 sed 程序的例子。
a,append　追加
i,insert　插入
d,delete　删除
s,substitution　替换

查找替换应用举例
sed 2a TYPE=Ethernet' test.txt　第二行后添加 TYPE=Ethernet
sed '3i TYPE=Ethernet' test.txt　第三行前添加 TYPE=Ethernet
sed 's/yes/no/g' test.txt　将样本文件中的所有 yes 替换为 no
sed '3,4d' test.txt　删除第 3 至 4 行的内容
总结：以上大多数操作指令，都依据行号定位操作对象（地址），如：2a 即第二行后添加。但实际情况可能大多数情况你并不确定你要操作对象（地址）的行号，这时更多的我们会使用正则表达式确定操作对象（地址）。

下面是使用正则表达式定位操作行的示例：
sed '/222/a iiii=1111 ' testfile.txt
匹配到包含 222 的行，并在其后添加 iiii=1111
sed '/^aaa/d' testfile.txt
匹配以 aaaa 开始的行，并删除改行

需要执行多个指令时，可以使用以下三种方法：
#sed 's/yes/no/;s/static/dhcp/' test.txt　注：使用分号隔开指令。

#sed -e 's/yes/no/' -e 's/static/dhcp/' test.txt　注：使用-e 选项。

#sed '
>s/yes/no/
>s/static/dhcp/' test.txt　注：利用分行指令。
然而在命令行上输入过长的指令是愚蠢的，这时就需要-f 选项指定 sed 脚本文件，在脚本文件中可以包含多行指令，而且便于修改！

```
sed   -i   's/^M//g'   a.txt > new.out
cat gpdata_wbm10.bak | sed 's/^M//g' | awk '$0' > 2.log
```

sed 命令 DOS2UnixFile

使用技巧

vim 中，打开 vim 编译器执行 set nu 命令。

经验话语：

shell 脚本参数意义

在 shell 中，表示值是用$,相当于 DOS 中的%。

1.位置参数

一般是系统或用户提供的参数。

$[0-n],$0,表示指令本身，$1 表示第一个参数，一次类推。

$0 是内部参数，必须要有的，其后的就可有可无了

2.内部参数

$# ----参数数目

$? ----上一个代码或者 shell 程序在 shell 中退出的情况，如果正常退出则返回 0，反之为非 0 值。

$* ----所有参数的字符串

5.6 合并于分割