Shell Notes

/sbin The system binary directory, where many GNU admin-level utilities are stored.

/bin The binary directory, where many GNU user-level utilities are stored.

/var The variable directory, for files that change frequently, such as log files.

CP:

-l Create a file link instead of copying the files. (创建的是一个硬链接)

-s 创建的是一个软链接

Stat 查看详细文件详细信息

Ps:

■ Unix-style parameters, which are preceded by a dash

■ BSD-style parameters, which are not preceded by a dash

■ GNU long parameters, which are preceded by a double dash

Display processes in a hierarchical format (showing parent processes).

ps -efH

ps –ef 显示所有进程

sort 对输入的文件内容排序。

**Shell 环境变量：**

printenv 可以打印一些环境(global)

export it to the global environment

unset varite

数组定义

mytest=(one two three four five)

数组使用：

${mytest[1]}

To display an entire array variable, you use the asterisk wildcard character as the index value:

${mytest[\*]}

/etc/grop 格式：

组名：密码：组ID：用户

权限：

RWX-RWX-RWX

Owner-group-others

111-111-111

Umask 以补码的形式存在，比如 002 就是others 默认不能写

Sticky Bit:  
**SUID:** 当一个设置了SUID 位的可执行文件被执行时，该文件将以所有者的身份运行，也就是说无论谁来执行这个文件，他都有文件所有者的特权。

**SGID:** 当一个设置了SGID 位的可执行文件运行时，该文件将具有所属组的特权， 任意存取整个组所能使用的系统资源

若一个目录设置了SGID，则所有被复制到这个目录下的文件， 其所属的组都会被重设为和这个目录一样，除非在复制文件时加上-p （preserve，保留文件属性）的参数，才能保留原来所属的群组设置。

**sticky-bit:**对一个文件设置了sticky-bit之后，尽管其他用户有写权限， 也必须由属主执行删除、移动等操作。

--------vim-----------

■ G to move to the last line in the buffer

■ *num* G to move to the line number *num* in the buffer.

■ gg to move to the first line in the buffer

yw 复制当前位置到后面的单词，

y$ /yy 复制当前行

同样d也是类似命令

J 删除换行符

a 从当前位置追加文字，A从行后面添加

r修改一个字符 R修改多个字符直到按esc

查找和替换：

:[% / #,#]s/old/new/g[c] %表示全文替换 / #,# 表示多少行到多少行，c表示是否每个提示

Shell Script:

#!/bin/bash

the first line of a shell script file is a special

case, and the pound sign followed by the exclamation point tells the shell what shell to run the

script under (yes, you can be using a bash shell and run your script using another shell).

变量：

No spaces can appear between the

variable, the equal sign, and the value (another trouble spot for novices).

定义不能有空格出现

Eg:

Var1=10 不能出现这种 var1 = 10

`

One of the most useful features of shell scripts is the lowly back quote character, usually called the

backtick (`) in the Linux world.

<< 输入

>> 输出

|

Instead of

redirecting the output of a command to a file, you can redirect the output to another command

**expr** 1 + 5 做算数运算

方便的方法： $[ express ]

浮点运算用 built-in bash calculator (called bc)

The bash calculator recognizes:

■ Numbers (both integer and floating point)

■ Variables (both simple variables and arrays)

■ Comments (lines starting with a pound sign or the C language /\* \*/ pair

■ Expressions

■ Programming statements (such as if-then statements)

■ Functions

var1=`echo " scale=4; 3.44 / 5" | bc` 这样就可以计算浮点型了

variable=`echo "options; expression" | bc`

the scale variable to four decimal places

variable=`bc *<<* EOF

options

statements

expressions

EOF

`

计算圆周率：

echo "scale=100; a(1)\*4" | bc –l

shell 退出状态：

**Code Description**

0 Successful completion of the command

1 General unknown error

2 Misuse of shell command

126 The command can’t execute

127 Command not found

128 Invalid exit argument

128+x Fatal error with Linux signal x

130 Command terminated with Ctl-C

255 Exit status out of range

**$?** 获取最后一条命令的执行状态

条件语句：

if *command*

then

*commands*

fi

或者：

if command; then

commands

fi

还有：

if *command*

then

*commands*

else

*commands*

fi

或者：

if *command1*

then

*commands*

elif *command2*

then

more *commands*

fi

if *test condition*

then

*commands*

fi

if [ *condition* ] ## have a space after the first bracket, and a space before the last bracket or you’ll get an error

message.

then

*commands*

fi

There are three classes of conditions the test command can evaluate:

■ Numeric comparisons

■ String comparisons

■ File comparisons

数字比较：

*n1* -eq *n2* Check if *n1* is equal to *n2*.

*n1* -ge *n2* Check if *n1* is greater than or equal to *n2*.

*n1* -gt *n2* Check if *n1* is greater than *n2*.

*n1* -le *n2* Check if *n1* is less than or equal to *n2*.

*n1* -lt *n2* Check if *n1* is less than *n2*.

*n1* -ne *n2* Check if *n1* is not equal to *n2*.

字串比较：

*str1* = *str2* Check if *str1* is the same as string *str2*.

*str1* != *str2* Check if *str1* is not the same as *str2*.

*str1 < str2* Check if *str1* is less than *str2*.

*str1 > str2* Check if *str1* is greater than *str2*.

-n *str1* Check if *str1* has a length greater than zero.

-z *str1* Check if *str1* has a length of zero.

文件比较：

-d *file* Check if *file* exists and is a directory.

-e *file* Checks if *file* exists.

-f *file* Checks if *file* exists and is a file.

-r *file* Checks if *file* exists and is readable.

-s *file* Checks if *file* exists and is not empty.

-w *file* Checks if *file* exists and is writable.

-x *file* Checks if *file* exists and is executable.

-O file Checks if *file* exists and is owned by the current user.

-G *file* Checks if *file* exists and the default group is the same as the current

user.

*file1* -nt *file2* Checks if *file1* is newer than *file2*.

*file1* -ot *file2* Checks if *file1* is older than *file2*.

条件的特别用法：

■Double parentheses for mathematical expressions

■ Double square brackets for advanced string handling functions

(( *expression* ))

(( val2 = $val1 \*\* 2 )) ：\*\* 表示乘方的意思

双括号可以用的运算符：

*val*++ post-increment

*val*-- post-decrement

++*val* pre-increment

--*val* pre-decrement

! logical negation

∼ bitwise negation

\*\* exponentiation

*<<* left bitwise shift

*>>* right bitwise shift

& bitwise Boolean AND

| bitwise Boolean OR

&& logical AND

|| logical OR

[[ *expression* ]]

[[ $USER == r\* ]] 双方括号里面可以用到正则表达式

Case 结构：

case *variable* in

*pattern1* | *pattern2*) *commands1*;;

*pattern3*) *commands2*;;

\*) *default commands*;;

esac

eg:

case $USER in

rich | barbara)

echo "Welcome, $USER"

echo "Please enjoy your visit";;

testing)

echo "Special testing account";;

jessica)

echo "Don’t forget to log off when you’re done";;

\*)

echo "Sorry, you’re not allowed here";;

easc

For 结构：

for var in *list*

do

*commands*

done

c-style:

for (( *variable assignment* ; *condition* ; *iteration process* ))

While 结构：

while test *command*

do

*other commands*

done

until 结构：

until test *commands*

do

*other commands*

done

break, continue 都可以在for里面使用

break n; continue n;

后面可以加上数字，跳出几次循环

IFS 变量为系统变量默认的 分割 字符：

IFS, called the *internal field*

*separator*. The IFS environment variable defines a list of characters the bash shell uses as field

separators. By default, the bash shell considers the following characters as field separators:

■ A space

■ A tab

■ A newline

获取参数的个数：$#

The $\* and $@ variables provide one-stop shopping for all of your parameters

shift command：

When you use the shift command, it ‘‘downgrades’’ each parameter variable one position by

default. Thus, the value for variable $3 is moved to $2, the value for variable $2 is moved to $1,

and the value for variable $1 is discarded (note that the value for variable $0, the program name,

remains unchanged).

Shift 是把所有的变量向$1移动，即左移到$1中 $0不变

Getopt 格式：

The getopt command is a great tool to have handy when processing command line options and

parameters. It reorganizes the command line parameters to make parsing them in your script

easier.

getopt *options* optstring *parameters*

The *optstring* is the key to the process. It defines the valid option letters used in the command

line. It also defines which option letters require a parameter value.

Eg: getopt ab:cd -a -b test1 -cd test2 test3

高级命令：

getopts optstring *variable*

The OPTARG environment

variable contains the value to be used if an option requires a parameter value. The OPTIND

environment variable contains the value of the current location within the parameter list where

getopts left off

获取用户输入：

Read 命令 –p 参数 输入提示

The REPLY environment variable will contain all of the data entered in the input, and it can be

used in the shell script as any other variable.

-t 参数 超时

-s 静默读取

-----------------数据输入输出--------------------

**STDIN：**The STDIN file descriptor references the standard input to the shell

0 STDIN Standard input

1 STDOUT Standard output

2 STDERR Standard error

所以有些命令就会出现这样的格式：

2> null 错误信息重定向到垃圾桶

1 > test2 标准输出重定向到 test2文件

如果有需求也可以同时将 stderr & stdout 重定向： &> null

You can use the STDOUT and STDERR file descriptors in your scripts to produce output in

multiple locations simply by redirecting the appropriate file descriptors. There are two methods

for redirecting output in the script:

■ Temporarily redirecting each line

■ Permanently redirecting all commands in the script

临时的：>&2

永久的： exec 1>testout exec 0< testfile

exec 3*>*&-

关闭文件描述符

lsof 列出所有打开的文件描述符

lsof -a -p $$ -d 0,1,2

you can use the special environment variable

$$, which the shell sets to the current PID.

The -a option is used to AND the results of the other

two options

禁止命令输出

ls -al *>* /dev/null

创建本地的临时文件：

Mktmp testing.XXXX 后面的X必须为大写且至少4位

-t 是创建在 /tmp目录下面

-d 是创建临时目录

Tee 同时输出到其他文件或终端

Script control:

**Signal Value Description**

1 SIGHUP Hang up the process.

Ctrl-C 2 SIGINT Interrupt the process.

3 SIGQUIT Stop the process.

Kill 9 SIGKILL Unconditionally terminate the process.

15 SIGTERM Terminate the process if possible.

17 SIGSTOP Unconditionally stop, but don’t terminate the process.

Ctrl-Z 18 SIGTSTP Stop or pause the process, but don’t terminate.

19 SIGCONT Continue a stopped process.

捕获信号：

trap *commands signals*

trap – EXIT 移除一个信号

后台运行： 用 & 符号

无终端运行程序：nohup 命令

Jobs 查看运行或者暂停的程序

Nice & renice

The scheduling priority is an integer value, from −20 (the highest priority) to +20 (the lowest

priority). By default, the bash shell starts all processes with a priority of 0.

定时命令：

when you’re not there. The Linux system provides three

ways of running a script at a preselected time:

■ The at command

■ The batch command

■ The cron table

Cron 格式:

*min hour dayofmonth month dayofweek command*

*eg:* 15 10 \* \* \* *command*

15 16 \* \* 1 *command*

00 12 \* \* \* if [ `date +%d -d tomorrow` = 01 ] ; then ; *command*

***Linux 运行级别：***

0 Halt

1 Single-user mode

2 Multi-user mode, usually without networking support

3 Full multi-user mode, with networking

4 Unused

5 Multi-user mode, with networking and a graphical X Window session

6 Reboot

Linux shell 高级脚本：

创建函数：

function *name* {

*commands*

}

也可以定义为这样：

*name()* {

*commands*

}

$? 用这个返回一个上条命令退出状态 如果成功默认为0 返回只能是0-255

Return 可以指定函数退出的状态

Shell 变量有全局和局部之分，局部变量必须加 local 不然就会默认定义为全局变量，这样会造成困扰

linux中shell变量$#,$@,$0,$1,$2的含义解释:   
变量说明:   
$$   
Shell本身的PID（ProcessID）   
$!   
Shell最后运行的后台Process的PID   
$?   
最后运行的命令的结束代码（返回值）   
$-   
使用Set命令设定的Flag一览   
$\*   
所有参数列表。如"$\*"用「"」括起来的情况、以"$1 $2 … $n"的形式输出所有参数。   
$@   
所有参数列表。如"$@"用「"」括起来的情况、以"$1" "$2" … "$n" 的形式输出所有参数。   
$#   
添加到Shell的参数个数   
$0   
Shell本身的文件名   
$1～$n   
添加到Shell的各参数值。$1是第1参数、$2是第2参数…。

插播一个操作：

挂在NFS 文件系统：

Sudo apt-get install nfs-kernel-server portmap nfs-common

配置 /etc/exports文件

追加如下： /home/zhaoyang \*(rw,sync,no\_root\_squash)

nfs允许挂载的目录及权限，在文件/etc/exports中进行定义，各字段含义如下：

/home/kevin：要共享的目录

\* ：允许所有的网段访问

rw ：读写权限

sync：资料同步写入内在和硬盘

no\_root\_squash：nfs客户端共享目录使用者权限

即便是$1的这样的参数也必须通过传入才能使用

数组不能直接将数组名传到函数里面必须 通过 吧所有的成员传过去 ${array[\*]}

接收时不能用$1来接收必须用

Shell 脚本也可以递归和c语言一样，也可以返回值

创建shell库：

The source command has a shortcut alias, called the *dot operator*. To source the myfuncs library

file in a shell script, all you need to do is add the following line:

. ./myfuncs

命令行可以直接定义函数：

$ function divem { echo $[ $1 / $2 ]; }

$ divem 100 5

20

When you define the function on the command line, you must remember to include a semicolon

at the end of each command。

在.bashrc中也可以定义函数

Select命令格式：

The select command allows you to create a menu from a single command line, then retrieve

the entered answer and automatically process it

select variable in list

do

commands

done

shell 调试 bashdb 可以但不调试shell:  
n 下一条语句

b number 在第几行打断点

R 重启程序

颜色控制：

ANSI escapse codes:

Most all terminal emulation software recognizes the ANSI escape codes for formatting display output.

The ANSI escape codes begin with a *control sequence indicator* (CSI), which tells the terminal

that the data represents an escape code, followed by data indicating the operation to perform on

the display.

To control the display format, you must use the Select Graphic Rendition (SGR) escape codes.

The format of an SGR escape code is:

CSI*n*[;*k*]m

The m in the code indicates the SGR escape code. The *n* and *k* parameters define which display

control is used. You can specify just one parameter or two at the same time, separating them

using the semicolon. There are three classes of display control parameters:

■ Effect control codes

■ Foreground color control codes

■ Background color control codes

0 Reset to normal mode.

1 Set to bold intensity.

2 Set to faint intensity.

3 Use italic font.

4 Use single underline.

5 Use slow blink.

6 Use fast blink.

7 Reverse foreground/background colors.

8 Set foreground color to background color (invisible text).

a complete ANSI escape control code looks like this:

^[[0m

实际上是 \033[0m

\033 就是 ^[

echo -e "\033[字背景颜色；文字颜色m字符串\033[0m"

\33[0m 关闭所有属性   
　　\33[1m 设置高亮度   
　　\33[4m 下划线   
　　\33[5m 闪烁   
　　\33[7m 反显   
　　\33[8m 消隐   
　　\33[30m — \33[37m 设置前景色   
　　\33[40m — \33[47m 设置背景色   
　　\33[nA 光标上移n行   
　　\33[nB 光标下移n行   
　　\33[nC 光标右移n行   
　　\33[nD 光标左移n行   
　　\33[y;xH设置光标位置   
　　\33[2J 清屏   
　　\33[K 清除从光标到行尾的内容   
　　\33[s 保存光标位置   
　　\33[u 恢复光标位置   
　　\33[?25l 隐藏光标   
　　\33[?25h 显示光标

Style Foreground Background

1st Digit 2nd Digit 3rd Digit

0 - Reset 30 - Black 40 - Black

1 - FG Bright 31 - Red 41 - Red

2 - Unknown 32 - Green 42 - Green

3 - Unknown 33 - Yellow 43 - Yellow

4 - Underline 34 - Blue 44 - Blue

5 - BG Bright 35 - Magenta 45 - Magenta(洋红色)

6 - Unknown 36 - Cyan 46 - Cyan(蓝绿色)

7 - Reverse 37 - White 47 – White

echo -e "\e[Style;Foreground;BackgroundmTHINGSTOPRINT"中的Style、Foreground和 Background换成上面的数字就可以改变后面THINGSTOPRINT的显示颜色。比如，$ echo -e "\e[1;32;43mHello World! "

里之所以Hello World!最后的叹号后面加一个空格，是因为unix shell中叹号是保留字符，用来调用历史命令。比如!cmd就执行上一条执行过的cmd开头的命令，而!cmd:p则是只是将上一条cmd开头的命令唤出显示出来，而不执行。

Eg:

echo -e "\e[1;32;43mHello World! \e[0m"

echo -e "\e[0;32;43mHello World! \e[0m"

echo -e "\e[4;32;43mHello World! \e[0m"

echo -e "\e[5;32;43mHello World! \e[0m"

echo -e "\e[style;background;foreground mString \e[0m"

Dialog:

**Widget Description**

calendar Provides a calendar to select a date from

checklist Displays multiple entries where each entry can be turned on or off

form Allows you to build a form with labels and text fields to be filled out

fselect Provides a file selection window to browse for a file

gauge Displays a meter showing a percentage of completion

infobox Displays a message without waiting for a response

inputbox Displays a single text form box for text entry

inputmenu Provides an editable menu

menu Displays a list of selections to choose from

msgbox Displays a message and requires the user to select an OK button

pause Displays a meter showing the status of a specified pause period

passwordbox Displays a single textbox that hides entered text

passwordform Displays a form with labels and hidden text fields

radiolist Provides a group of menu items where only one item can be selected

tailbox Displays text from a file in a scroll window using the tail command

tailboxbg Same as tailbox, but operates in background mode

textbox Displays the contents of a file in a scroll window

timebox Provides a window to select an hour, minute, and second

yesno Provides a simple message with Yes and No buttons

eg:  
dialog --inputbox "Enter your age:" 10 20 2*>*age.txt

SED editor:

The sed editor is called a *stream editor*, as opposed to a normal interactive text editor. In an

interactive text editor, such as vim, you interactively use keyboard commands to insert, delete, or

replace text in the data. A stream editor edits a stream of data based on a set of rules you supply

ahead of time, before the editor processes the data.

The format for using the sed command is:

**sed *options script file***

-e *script* Add commands specified in the script to the commands run while

processing the input.

-f *file* Add the commands specified in the file to the commands run while

processing the input.

-n Don’t produce output for each command, but wait for the print command.

执行多个sed命令，用-e选项

Eg:  
sed -e ’s/brown/green/; s/dog/cat/’ data1

Sed –f 可以直接从文件中读取替换规则

Gawk: The gawk program is the GNU version of the original awk program in Unix.

Within the programming language you can:

■ Define variables to store data.

■ Use arithmetic and string operators to operate on data.

■ Use structured programming concepts, such as if-then statements and loops, to add

logic to your data processing.

■ Generate formatted reports by extracting data elements within the data file and repositioning

them in another order or format.

The basic format of the gawk program is:

**gawk *options program file***

-F *fs* Specify a file separator for delineating data fields in a line.

-f *file* Specify a filename to read the program from.

-v *var*=*value* Define a variable and default value used in the gawk program.

-mf *N* Specify the maximum number of fields to process in the data file.

-mr *N* Specify the maximum record size in the data file.

-W *keyword* Specify the compatibility mode or warning level for gawk.

A gawk program script is defined by opening and closing braces. You must place script commands

between the two braces

eg: gawk ’{print "Hello John!"}’

**Using data field variables**

As I mentioned, one of the primary features of gawk is its ability to manipulate data in the text

file. It does this by automatically assigning a variable to each data element in a line. By default,

gawk assigns the following variables to each data field it detects in the line of text:

■ $0 represents the entire line of text.

■ $1 represents the first data field in the line of text.

■ $2 represents the second data field in the line of text.

■ $n represents the *n*th data field in the line of text.

Eg:

gawk -F: ’{print $1}’ /etc/passwd

echo "My name is Rich" | gawk '{$4="Dave"; print $0}'

同样 gawk 也可以从脚本中读取规则：gawk -F: -f script2 /etc/passwd

To do that you use the BEGIN keyword. This forces gawk to execute the

program script specified after the BEGIN keyword before reading the data:

$ gawk ’BEGIN {print "Hello World!"}’

Sed基本用法：

s/*pattern*/*replacement*/*flags*

There are four types of substitution flags available:

■ A number, indicating the pattern occurrence for which new text should be substituted.

■ g — Indicates that new text should be substituted for all occurrences of the existing text.

■ p — Indicates that the contents of the original line should be printed.

■ w *file* — Write the results of the substitution to a file.

the commands you use in the sed editor apply to all lines of the text data. If you only

want to apply a command to a specific line, or a group of lines, you must use *line addressing*.

There are two forms of line addressing in the sed editor:

■ A numeric range of lines

■ A text pattern that filters out a line

Both forms use the same format for specifying the address:

[address]command

You can also group more than one command together for a specific address:

address {

command1

command2

command3

}

Eg:

sed ’2s/dog/cat/’ data1 替换第二行

sed ’2,3s/dog/cat/’ data1 替换2到3行

sed ’2,$s/dog/cat/’ data1 替换第二行之后的所有 $表示最后一行

/pattern/command 根据匹配来替换

即： /pattern1/s/pattern/replacement/flags

Sed 可以执行sed命令：sed ’3,$d’ data6

/pattern/command like d 删除

You can also delete a range of lines using two text patterns, but be careful if you do this. The first

pattern you specify ‘‘turns on’’ the line deletion, and the second pattern ‘‘turns off’’

the line deletion

I mentioned that you need to be careful with this, as the delete feature will ‘‘turn on’’ whenever

the sed editor detects the start pattern in the data stream

sed 同样还有 i a c命令：分别表示insert append 可以插入新行 change 修改某一行的数据

eg:

sed ’3i\

*>* This is an inserted line.’ data6

这是将这个数据插入到第三行的前面

The transform command (y) is the only sed editor command that operates on a single character.

The transform command uses the format:

[*address*]*y*/*inchars*/*outchars*/

Eg:

sed ’y/123/789/’ data7 将 1 2 3 分别替换成7 8 9

echo "This 1 is a test of 3 try." | sed 'y/123/456/'

$ This 4 is a test of 6 try.

The ‘‘More Substitution Options’’ section showed how to use the p flag with the substitution

command to display lines that the sed editor changed. There are three commands that also can

be used to print information from the data stream:

■ The lowercase p command to print a text line

■ The equal sign (=) command to print line numbers

■ The l (lowercase L) command to list a line

将行写入文件

[*address*]w *filename*

*读取*

[*address*]r *filename*

Regular Expression

In the Linux world, there are two popular regular expression engines:

■ The POSIX Basic Regular Expression (BRE) engine

■ The POSIX Extended Regular Expression (ERE) engine

单个字符匹配：

echo "This is a test" | sed -n ’/This/p’

常规字符 空格都能匹配。

特殊字符匹配：

The special characters recognized by regular expressions are:

.\*[]^${}\+?|()

Eg:

echo "\ is a special character" | sed -n ’/\\/p’

$ cat data2

The cost is $4.00

$ sed -n ’/\$/p’ data2

**Anchor characters**

The caret character (^) defines a pattern that starts at the beginning of a line of text in the data

Stream

表示以什么字符开始的行

echo "This ^ is a test" | sed -n ’/s ^/p’ ^ 这个不是特殊字符不需要转移

查找以book结尾的行

echo "This is a good book" | sed -n ’/book$/p’

sed -n ’/^this is a test$/p’ data4

**sed ’/^$/d’ data5 删除空行**

The dot special character is used to match any single character except a newline character

. 表示任意字符

**Character classes 字符集**

sed -n ’/[ch]at/p’ data6

只匹配 cat hat的行

**Negating character classes**

用^ 表示排除这些字符

sed -n ’/[^ch]at/p’ data6

使用范围：

**Using ranges**

**-**

sed -n ’/^[0-9][0-9][0-9][0-9][0-9]$/p’ data8

sed -n ’/[a-ch-m]at/p’ data6

**Special character classes**

**特殊字符集：**

[[:alpha:]] Match any alphabetical character, either upper or lower case.

[[:alnum:]] Match any alphanumeric character 0–9, A–Z, or a–z.

[[:blank:]] Match a space or Tab character.

[[:digit:]] Match a numerical digit from 0 through 9.

[[:lower:]] Match any lower-case alphabetical character a–z.

[[:print:]] Match any printable character.

[[:punct:]] Match a punctuation character.

[[:space:]] Match any whitespace character: space, Tab, NL, FF, VT, CR.

[[:upper:]] Match any upper-case alphabetical character A–Z.

**The asterisk**

Placing an asterisk after a character signifies that the character must appear zero or more times in

the text to match the pattern:

$ echo "ik" | sed -n ’/ie\*k/p’

ik

$ echo "iek" | sed -n ’/ie\*k/p’

Iek

**The question mark**

The question mark is similar to the asterisk, but with a slight twist. The question mark indicates

that the preceding character can appear zero or one time, but that’s all

**The plus sign**

The plus sign indicates that the preceding character can appear one or

more times, but must be present at least once.

$ echo "beeet" | gawk ’/be+t/{print $0}’

Beeet

**Using braces**

Curly braces are available in ERE to allow you to specify a limit on a repeatable regular expression.

This is often referred to as an *interval*. You can express the interval in two formats:

■ *m*: The regular expression appears exactly *m* times.

■ *m*,*n*: The regular expression appears at least *m* times, but no more than *n* times.

Eg:

$ echo "bt" | gawk **--re-interval** ’/be{1}t/{print $0}’

默认情况下 gawk不能识别 正则表达是的间隔 需要加**--re-interval**

**e 最多出现2次最少出现一次**

echo "bet" | gawk --re-interval ’/be{1,2}t/{print $0}’

bet

**The pipe symbol**

The pipe symbol allows to you to specify two or more patterns that the regular expression engine

uses in a logical OR formula when examining the data stream

eg:

$ echo "The cat is asleep" | gawk ’/cat**|**dog/{print $0}’

The cat is asleep

**Grouping expressions**

When you group a regular expression pattern, the group is treated like a standard character. You can apply a special

character to the group just as you would to a regular character. For example:

$ echo "Sat" | gawk ’/Sat(urday)?/{print $0}’

Sat

$ echo "Saturday" | gawk ’/Sat(urday)?/{print $0}’

Saturday

Sed的高级用法：

The sed editor includes three special commands that you can use to process multiline text:

■ N: Add the next line in the data stream to create a multiline group for processing.

■ D: Delete a single line in a multiline group.

■ P: Print a single line in a multiline group.

操作多行：

$ cat data1

This is the header line.

This is a data line.

This is the last line.

$ sed ’/header/{

*>* n

*>* d

*>* }’ data1

N 命令：  
Here’s a demonstration of how the N command operates:

$ cat data2

This is the header line.

This is the first data line.

This is the second data line.

This is the last line.

$ sed ’/first/{

*>* N

*>* s/\n/ /

*>* }’ data2

This is the header line.

This is the first data line. This is the second data line.

This is the last line.

The N command helps solve this problem:

$ sed ’

*>* N

*>* s/System.Administrator/Desktop User/

*>* ’ data3

The first meeting of the Linux Desktop User’s group will be held

on Tuesday.

All Desktop Users should attend this meeting.

Thank you for your attendance

The Hold space:

The *pattern space* is an active buffer area that holds the text examined by the sed editor while it

processes commands. However, it isn’t the only space available in the sed editor for storing text.

The sed editor utilizes another buffer area called the *hold space*.

h Copy pattern space to hold space

H Append pattern space to hold space

g Copy hold space to pattern space

G Append hold space to pattern space

x Exchange contents of pattern and hold spaces

eg:

cat data2

This is the header line.

This is the first data line.

This is the second data line.

This is the last line.

$ sed -n ’/first/{

*>* h

*>* p

*>* n

*>* p

*>* g

*>* p

*>* }’ data2

This is the first data line.

This is the second data line.

This is the first data line.

h类似于把匹配项存起来，用于后面使用

**Negating a Command**

The exclamation mark command (!) is used to negate a command

意思是匹配到的行执行，而执行其他行

Eg:

$ sed -n ’/header/!p’ data2

This is the first data line.

This is the second data line.

This is the last line.

Changing the Flow:

the exception is the D command, which forces the sed editor to return to the top

of the script without reading a new line of text

The sed editor provides a way to negate an entire section of

commands, based on an address, an address pattern, or an address range

The format of the branch command is:

[*address*]b [*label*]

**The ampersand**

The sed editor has a solution for you. The ampersand symbol (&) is used to represent the

matching pattern in the substitution command. Whatever text matches the pattern defined, you

can use the ampersand symbol to recall it in the replacement pattern. This lets you manipulate

whatever word matches the pattern defined:

$ echo "The cat sleeps in his hat." | sed ’s/.at/"&"/g’

The "cat" sleeps in his "hat".

增加行号：

sed '=' data1 | sed 'N; s/\n/ /'

gawk 高级用法：

One important feature of any programming language is the ability to store

and recall values using variables. The gawk programming language supports

two different types of variables:

■ Built-in variables

■ User-defined variables

FIELDWIDTHS A space separated list of numbers defining the exact width (inspaces) of each data field

FS Input field separator character.

RS Input record separator character.

OFS Output field separator character.

ORS Output record separator character.

**Eg:**gawk 'BEGIN{OFS="\t"}{print $1,$2,$3}' data1

gawk 'BEGIN{FIELDWIDTHS="3 10 2 5"}{print $1,$2,$3,$4}' data1

更多的内建变量：

ARGC The number of command line parameters present.

ARGIND The index in ARGV of the current file being processed.

ARGV An array of command line parameters.

CONVFMT The conversion format for numbers (see the printf statement). The default value is %.6 g.

ENVIRON An associative array of the current shell environment variables and theirvalues.

ERRNO The system error if an error occurs when reading or closing input files.

FILENAME The file name of the data file used for input to the gawk program.

FNR The current record number in the data file.

IGNORECASE If set to a non-zero value, ignore the case of characters in strings used in the gawk command.

NF The total number of data fields in the data file.

NR The number of input records processed.

OFMT The output format for displaying numbers. The default is %.6 g.

RLENGTH The length of the substring matched in the match function.

RSTART The start index of the substring matched in the match function.

用户定义的变量：

gawk ‘BEGIN{x=4; x= x \* 2 + 3; print x}’

gawk 还可以用if-then while 等等

gawk ’{if ($1 › 20) print $1 \* 2; else print $1 / 2}’ data4

**Built-in Functions**

atan2(*x, y*) The arctangent of *x* / *y*, with *x* and *y* specified in radians.

cos(*x*) The cosine of *x*, with *x* specified in radians.

exp(*x*) The exponential of *x*.

int(*x*) The integer part of *x*, truncated toward 0.

log(*x*) The natural logarithm of *x*.

rand() A random floating point value larger than 0 and less than 1.

sin(*x*) The sine of *x*, with *x* specified in radians.

sqrt(*x*) The square root of *x*.

srand(*x*) Specify a seed value for calculating random numbers

The **ash** shell is a low-budget shell that offers

basic features with a small footprint. This is perfect for low-memory applications,

such as embedded Linux systems