目录

[把简单的技术学到极致就是高手 2](#_Toc477280510)

[shell 基础入门 2](#_Toc477280511)

[bash 特性 2](#_Toc477280512)

[命令hash 2](#_Toc477280513)

[变量 2](#_Toc477280514)

[本地变量： 3](#_Toc477280515)

[环境变量： 3](#_Toc477280516)

[逻辑运算： 4](#_Toc477280517)

[短路法则： 4](#_Toc477280518)

[脚本编程 5](#_Toc477280519)

[文本编程器： 5](#_Toc477280520)

[执行脚本： 5](#_Toc477280521)

[练习1： 5](#_Toc477280522)

[登录类型： 6](#_Toc477280523)

[profile 类： 6](#_Toc477280524)

[bashrc类： 6](#_Toc477280525)

[条件测试 7](#_Toc477280526)

[整数测试 8](#_Toc477280527)

[字符串测试 8](#_Toc477280528)

[文件测试 8](#_Toc477280529)

[流程控制 9](#_Toc477280530)

[if语句 9](#_Toc477280531)

[IO重定向及管道 9](#_Toc477280532)

[程序的数据流： 9](#_Toc477280533)

[fd: file descriptor 文件描述符 9](#_Toc477280534)

[IO重定向 10](#_Toc477280535)

[输出重定向： 10](#_Toc477280536)

[错误输出流重定向:2>, 2>> 10](#_Toc477280537)

[合并输出流： &>, &>> 10](#_Toc477280538)

[输入重定向 10](#_Toc477280539)

[管道命令 11](#_Toc477280540)

[文本处理工具 11](#_Toc477280541)

[linux 文本处理三剑客： 11](#_Toc477280542)

[正则表达式： 12](#_Toc477280543)

[元字符： 12](#_Toc477280544)

[字符匹配： 12](#_Toc477280545)

[匹配次数 12](#_Toc477280546)

[位置锚定： 13](#_Toc477280547)

[练习： 13](#_Toc477280548)

[分组及引用 14](#_Toc477280549)

[后向引用 14](#_Toc477280550)

[扩展正则表达式 14](#_Toc477280551)

[练习： 15](#_Toc477280552)

[grep 15](#_Toc477280553)

[options: 15](#_Toc477280554)

[文本查看及处理 16](#_Toc477280555)

[练习题 17](#_Toc477280556)

[1. 请执行命令取出linux中eth0的IP地址(请用cut，有能力者也可分别用awk,sed命令答) 17](#_Toc477280557)

[2. 批量杀死含有关键词report的进程 17](#_Toc477280558)

# **把简单的技术学到极致就是高手**

作者：梁毅强

邮箱：398504533@qq.com

# shell 基础入门

## 软件工具原则

1.一次做好一件事

最重要的原则，若程序只做一件事，那么无论设计、编写、调度、维护都会变得非常方便

2.处理文本行，不要处理二进制数据

文本行是UNIX的通用格式

3.使用正则表达式

POSIX只有两种标准，一种是用于grep的基本型正则，一种是用于egrep的扩展型正则

4.默认使用标准输入、输出

5.“你叫它做什么，你就会得到什么”的设计哲学

6.输入输出格式一致

7.让工具去做困难的部分

8.构建工具前先思考

## 三种基本命令

### 内建命令

内置的，build-in 如cd 、dir 、test 、echo、printf

### 函数

由shell语言写成的程序代码

### 外部命令

由shell副本所执行的命令

## bash 特性

### 命令hash

缓存命令的查找结果:key-value

hash: 列出

hash –d :command :删除

hash –r :清空

### 变量

程序 = 指令 + 数据

指令：由程序文件提供

数据：IO设备、 文件、管理、变量

程序： 算法+数据结构

变量名+指向的内存空间

变量赋值：name=value

变量类型：存储格式、表示数据范围、参与的运算

bash：把所有变量视作字符型

变量无需事先声明(声明和赋值一同进行)

变量引用：${var\_name}, $var\_name

变量类型：

本地变量: 作用域仅为当前shell进程

环境变量：作用域为当前shell进程及其子进程

局部变量：作用域仅为某代码片断(函数上下文)

位置参数变量：当执行脚本的Shell进程传递的参数

特殊变量：shell内置的变量

$?:0成功、1~255失败

变量名：见名知义，命名机制遵循某种规则

#### 本地变量：

变量赋值：

first\_name=jerry

[root@localhost ~]# first\_name=jerry

[root@localhost ~]# echo $first\_name

jerry

[root@localhost ~]# unset first\_name

[root@localhost ~]# echo $first\_name

[root@localhost ~]#

#### 环境变量：

变量赋值：

1. export name=value
2. name=value

export name

1. name=value

declare –x name

[root@localhost ~]# firstName=jerry

[root@localhost ~]# echo $firstName

jerry

[root@localhost ~]# declare -x firstName

[root@localhost ~]# bash

[root@localhost ~]# echo $firstName

jerry

[root@localhost ~]#

查看方式：export, env, declare –x, printenv

撤销环境变量：unset name

只读变量：

declare –r name

readonly name

无法重新赋值，并不支持撤销，存活时间为当前shell进程的生命周期，随shell进程终止而终止。

多命令执行

~]# command1;command2;command3;...

#### shell 变量

|  |  |
| --- | --- |
| **变量** | **含义** |
| $0 | 当前脚本的文件名 |
| $n | 传递给脚本或函数的参数。n 是一个数字，表示第几个参数。例如，第一个参数是$1，第二个参数是$2。 |
| $# | 传递给脚本或函数的参数个数。 |
| $\* | 传递给脚本或函数的所有参数。 |
| $@ | 传递给脚本或函数的所有参数。被双引号(" ")包含时，与 $\* 稍有不同，下面将会讲到。 |
| $? | 上个命令的退出状态，或函数的返回值。 |
| $$ | 当前Shell进程ID。对于 Shell 脚本，就是这些脚本所在的进程ID。 |

### 特殊文件:

#### /dev/null

/dev/null: 又叫bit bucket 传到此文件的数据都会被系统丢弃

实用例子，测试一个文件是否包含某个模式

不需要结果，只需要退出状态

if grep "dpass" pas.sh > /dev/null;then

echo "find pattern"

else

echo "can not find pattern"

fi

ps:读取/dev/null会立即返回文件结束符

#### /dev/tty

当程序打开此文件时，UNIX会自动将它重定向到一个终端，再与程序结合

读取密码操作

#!/bin/bash

printf "Enter new password: "

stty -echo #关闭自动打印每个输入字符的功能

read pass < /dev/tty #接收终端数据

printf "Enter again: "

read pass2 < /dev/tty

stty echo #关闭禁止显示

## 脚本编程

如何写shell脚本

脚本第一行，给出shebang,解释器路径，用于指明解释器执行当前脚本的程序文件

#!/bin/bash

#!/usr/bin/python

### 文本编程器：

行编辑器：sed, awk

全屏编辑器：vi,nano,vim

shell 脚本是什么：

命令的堆积

但很多命令不具有幂等性，需要用程序逻辑来判断运行条件是否满足，以避免

### 执行脚本：

(1):赋予执行权限，并直接运行程序文件

chmod +x filename

~]# myfirst.sh #系统当作是命令,会到环境变量下找

~]# ./myfirst.sh #这才是执行脚本

例：

#!/bin/bash

useradd user2

echo "user2" | passwd --stdin user2 #将user2的密码改为user2

(2)直接运行解释器

bash /path/to/script\_file

#### 练习1：

1. 创建临时文件/tmp/myfile.XXXX

mktemp /tmp/myfile.XXXX #目录的话加 –d

### 登录类型：

交互式登录shell进程：

直接通过某终端输入账号和密码后登录打开的Shell进程

使用su – username 执行的登录切换

非交互登录：

su username

图形界面下打开的终端

运行脚本

### profile 类：

全局：对所有用户都生效

/etc/profile

/etc/profile.d/\*.sh

用户个人：仅对当前用户有效

~/.bash\_profile

功用：

1. 用于定义环境变量
2. 运行命令或脚本

### bashrc类：

全局：

/etc/bashrc

用户个人：

~/.bashrc

功用：

1. 定义本地变量
2. 定义命令别名

读取顺序：

交互式：

/etc/profile -> /etc/profile.d/\* ->~/.bash\_profile -> ~/.bashrc ->/etc/bashrc

非交互式：

~/.bashrc -> /etc/bashrc ->/etc/profile.d/\*

命令行中定义的特性，例如变量和别名作用域为当前shell进程的生命周期

配置文件定义的特性，只对随后新启动的shell 进程生效

让配置文件立即生效：

1. 通过命令行重复定义一次
2. 让shell重读配置文件

. /path/to/config\_file

source /path/to/config\_file

### 接收命令行参数

]# cat > finduser

#!/bin/bash

who | grep $1

^D

保存文件后chmod +x finduser

]# ./finduser leon

leon pts/6 2017-03-17 22:32 (192.168.233.1)

显示结果

如果参数大于9，则要用大括号将数据括起来，如 ${10}

### 执行跟踪

-x 可以对每行调用进行跟踪

]# sh -x finduser leon

+ who

+ grep leon

leon pts/6 2017-03-17 22:32 (192.168.233.1)

### 文件包含

两种方式：

1. 点号

. filename

1. 用source

source filename

注：被包含的脚本不需要有执行权限

## 条件测试

test expression

[ expression ]

[[ expression ]]

[] 匹配指定范围内的任意单个字符

几种特殊的格式

[a-z], [A-Z], [0-9], [a-z0-9]

[[:upper:]] 所有大写字母

[[:lower:]] 所有小写字母

[[:alpha:]] 所有字母

[[:digit:]] 所有数字

[[:alnum:]] 所有的字母和数字

[[:space:]] 所有空白字符

[[:punct:]] 所有标点符号

[root@localhost test]# ll

total 0

-rw-r--r--. 1 root root 0 Mar 5 00:08 m33434a

-rw-r--r--. 1 root root 0 Mar 5 00:01 M54r3

-rw-r--r--. 1 root root 0 Mar 5 00:02 mjkd3

-rw-r--r--. 1 root root 0 Mar 5 00:05 p a

-rw-r--r--. 1 root root 0 Mar 5 00:09 p,a

[root@localhost test]# ls p[^[:alnum:]]a

p a p,a

练习1：显示/var 目录下所有以l开头，以一个小写字母结尾，且中间出现一位任意字符的文件或目录

[root@localhost ~]# ls -d /var/l?[[:lower:]]

/var/lib /var/log

-d 显示目录，如果有文件也会显示出文件

如果不加-d 则会将 /var/lib /var/log 这两个目录下的文件也显示出来

练习2：显示/etc 目录下，以任意一位数字开头，且以非数字结尾的文件或目录

[root@localhost ~]# touch /etc/3abc3h

[root@localhost ~]# ls -d /etc/[[:digit:]]\*[^[:digit:]]

/etc/3abc3h

练习3：显示/etc 目录下，以非字母开头，后面跟一个字母及其它任意长度作意字符的文件或目录

[root@localhost ~]# ls -d /etc/[^a-z][a-z]\*

/etc/2aabb

练习4：复制/etc 目录下，所有以m开头，以非数字结尾的文件或目录至/tmp/var 目录

[root@localhost ~]# cp -r /etc/m\*[^0-9] /tmp/var

练习5：复制/usr/share/man 目录下，所有以man开头，后跟一个数字结尾的文件或目录到/tmp/man目录下

[root@localhost var]# cp -rf /usr/share/man/man\*[0-9] /tmp/man

练习6：复制/etc 目录下，所有以.conf结尾，且以m,n,r,p开头的文件或目录至/tmp/conf.d 目录下

[root@localhost var]# cp -rf /etc/[m,n,r,p]\*\.conf /tmp/conf.d

### 整数测试

-eq

-gt

-lt

-ne

-le

-ge

### 字符串测试

==

!=

-z $VAR

-z $VAr

>, >=, <, <=

### 文件测试

-e /path/to/somewhere 是否存在

-f 普通文件

-d 目录

-l 链接

-r 读权限

-w 写权限

-x 执行权限

示例：

1. 检测文件是否是可执行文件

~] [ -x filename ]

~] echo $?

## 程序退出状态

0表示成功，其他值表示失败

内置变量?（以$?访问它）包括了Shell 最近一次所执行的程序状态

POSIX 的结束状态

|  |  |
| --- | --- |
| 值 | 意义 |
| 0 | 成功 |
| >0 | 在重定向或单词展开期间失败 |
| 1-125 | 自定义错误 |
| 126 | 命令找到了，但文件无法执行 |
| 127 | 命令找不到 |

编程时用 exit num 来设置退出状态

判断时通过 $? 获取退出状态

## 数组

mons=("Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov" "Dec")

获取数组元素个数

num=${#mons[@]}

## 字典

字典在使用前要先定义(declare)

declare -A dic

dic=([Jun]="1" [Feb]="2")

echo ${dic["Jun"]} #1

### 获取所有KEY

echo ${!dic[\*]} #

### 获取所有 value

echo ${dic[\*]}

遍历

## 函数

注意事项：

1. 调用函数只需要给出函数名，不需要在后面加括号
2. 函数返回值只能是整数，用来表示成功与否
3. 如果一定要让函数返回字符串，可以先定义一个变量，以接收函数计算结果，脚本在需要的时候访问这个变量即可

## 流程控制

### 逻辑运算：

运算数：真(true, yes, on, 1)

假(false, no, off, 0)

与:

1 &&1 = 1

1 && 0 = 0

0 && 1 = 0

0 && 0 = 0

或：

1 || 1 = 1

1 || 0 = 1

0 || 1 = 1

0 || 0 = 0

非：

!1 = 0

!0 = 1

异或：

#### 短路法则：

~]# command1 && command2

command1 为假，则command2不会再执行

否则，command1为真，则command2必须执行

~]# command1 || command2

command1为真，则command2不会再执行

否则，command1为假，则command2必须执行

### if语句

语法：

if pipeline

[ pipeline ... ]

then

statements-if-true-1

[ elif pipeline

[ pipeline ... ]

then

statements-if true-2

...]

[ else

statements-if-all-else-fails

]

fi

例子：

1.判断变量是否为空

a="d"

if [ -z "$a" ];then

echo "a is empty string"

else

echo "a is not empty"

fi

### 逻辑的NOT、AND、OR

有时，以否定表达测试操作会比较容易，“如果John不在家，则...”,

if ! grep pattern myfile > /dev/null;then

... 匹配不到模式

fi

POSIX 1992标准中引进标记方式

if grep pattern myfile > /dev/null

then

: #能匹配，但不做任何事，这个像Python里的pass

else

... 匹配不到模式

fi

### case 语句

语法：

case $1 in

-f)

...

;;

-d | --directory)

...

;;

\*)

echo $1: unknow option >&2

exit 1

esac

\*)里的;;可有可无

### for 循环

1. 双括号循环

#!/bin/sh

for ((i=1;i<=20;i+=1))

do

printf "%03d\n" $i

done

注：printf 可以输出像c语言那样的格式

1. 数字段形式

用{n..m}形式进行

for f in {1..10}

do

printf "%02d\n" $f

done

1. 详细列出

for d in 1 3 4 5

do

echo $d

done

也可通过seq 生成数字序列来进行循环遍历

1. 对存在的文件进行循环

for shname in `ls \*.sh`  
do   
          name=`echo "$shname" | awk -F. '{print $1}'`             
          echo $name  
done

## IO重定向及管道

### 程序的数据流：

输入的数据流： 标准输入(stdin) 默认是键盘

输出的数据流： 标准输出(stdout) 显示器

错误输出流：错误输出(stderr) 显示器

### fd: file descriptor 文件描述符

标准输入：0

标准输出：1

错误输出：2

### IO重定向

#### 输出重定向：

>

特性：覆盖输出

禁止覆盖输出：set –C

关闭上述特性: set +C

仅对当前shell 有效

>>

特性：追加输出

cat /etc/issue > /tmp/issue.txt

#### 错误输出流重定向:2>, 2>>

[root@localhost tmp]# cat /etc/issue3 > /tmp/issue.txt 2>/tmp/err.txt

[root@localhost tmp]# cat /tmp/err.txt

cat: /etc/issue3: No such file or directory

#### 合并输出流： &>, &>>

[root@localhost tmp]# cat /etc/issse &> /tmp/issue.txt

[root@localhost tmp]# cat /tmp/issue.txt

cat: /etc/issse: No such file or directory

另一种合并用法：

[root@localhost tmp]# cat /etc/issse > /tmp/issue.txt 2>&1

#### 输入重定向

tr 命令用法

tr [OPTION]... SET1 [SET2]

把输入的数据当中的字符，凡是在SET1 定义范围内的，通通对位转换为SET2出现的字符

[root@localhost tmp]# tr [a-z] [A-Z] < /etc/issue

CENTOS RELEASE 6.4 (FINAL)

KERNEL \R ON AN \M

### 管道命令

连接程序，实现将一个命令的输出直接定向到后一个程序当作输入数据流

command1 | command2 | command3

tee命令

读取一个标准输入并写到一个标准输出他文件

command | tee /path/to/somefile

[root@localhost ~]# cat /etc/issue | tee /tmp/issue.tee|tr 'a-z' 'A-Z'

CENTOS RELEASE 6.4 (FINAL)

KERNEL \R ON AN \M

[root@localhost ~]# cat /tmp/issue.tee

CentOS release 6.4 (Final)

Kernel \r on an \m

练习1：将/etc/passwd 文件的前6行信息转换为大写字符后输出

[root@localhost ~]# head -6 /etc/passwd | tr 'a-z' 'A-Z'

ROOT:X:0:0:ROOT:/ROOT:/BIN/BASH

BIN:X:1:1:BIN:/BIN:/SBIN/NOLOGIN

DAEMON:X:2:2:DAEMON:/SBIN:/SBIN/NOLOGIN

ADM:X:3:4:ADM:/VAR/ADM:/SBIN/NOLOGIN

LP:X:4:7:LP:/VAR/SPOOL/LPD:/SBIN/NOLOGIN

SYNC:X:5:0:SYNC:/SBIN:/BIN/SYNC

## 文本处理工具

### linux 文本处理三剑客：

grep : 文本过滤工具(模式：pattern)工具

sed: stream editor,流编辑器， 文本编辑工具

awk:文本报告生成器，(格式化文本)

### 正则表达式：

1. 基本正则表达式: BRE
2. 扩展正则表达式:ERE

#### 元字符：

##### 字符匹配：

. 任意单个字符

[] 匹配指定范围内的作意单个字符

[^] 范围外

[:alpha:] [:digit:] [:upper:] [:lower:] [:punct:] [:space:] [:alnum:]

##### 匹配次数

限制前面的字符出现的次数

\*:匹配前面的字符任意次 0，1，多次

例：grep “x\*y”

[root@localhost scripts]# grep "x\*y" grep.txt

abxy

aby

xxxxxxy

yab

**.\*:匹配任意长度的任意字符**

\?:匹配其前面的字符0次或1次，即其前面的字符是可有可无

[root@localhost scripts]# grep "x\?y" grep.txt

abxy

aby

xxxxxxy

yab

\+:至少1次

[root@localhost scripts]# grep "x\+y" grep.txt

abxy

xxxxxxy

\{m\}:精确次数匹配

[root@localhost scripts]# grep "x\{6\}" grep.txt

xxxxxxy

axxxxxxxxxy

axxxxxxy

\{m,n\}:至少m次，至多n次

[root@localhost scripts]# grep "x\{4,6\}y" grep.txt

xxxxxxy

axxxxxxxxxy

axxxxxxy

特例：

\{0, n\} 至多n次，可以没有

\{m,\} :至少m次，最多不限

##### 位置锚定：

^：行首锚定，用于模式的最左侧，或者行的开头

[root@localhost scripts]# grep "root" /etc/passwd

root:x:0:0:root:/root:/bin/bash

operator:x:11:0:operator:/root:/sbin/nologin

chroot:x:505:505::/home/chroot:/bin/bash

$: 行尾锚定，用于模式的最右侧

^PATTERN$: 用于PATTERN来匹配整行

^$: 空白行

^[[:space:]]$:空行或包含空白字符的行

^leon$ : 这一行只有 leon 这个词

单词：非特殊字符组成的连续字符都称为单词

\< 或\b:词首锚定

\> 或\b:词尾锚定

\<PATTERN\>:匹配完整单词

注意： ^只有在开头，$只有在结尾时才有特殊意义

如, abc^cddf 则是匹配^字符自身

###### 练习：

1. 显示/etc/passwd 文件中不以/bin/bash结尾的行

[root@localhost scripts]# grep -v "/bin/bash$" /etc/passwd

bin:x:1:1:bin:/bin:/sbin/nologin

daemon:x:2:2:daemon:/sbin:/sbin/nologin

adm:x:3:4:adm:/var/adm:/sbin/nologin

lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin

sync:x:5:0:sync:/sbin:/bin/sync

1. 找出/etc/passwd 文件中两位数或三位数

[root@localhost scripts]# grep "[0-9]\{2,3\}" /etc/passwd

mail:x:8:12:mail:/var/spool/mail:/sbin/nologin

uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin

operator:x:11:0:operator:/root:/sbin/nologin

games:x:12:100:games:/usr/games:/sbin/nologin

gopher:x:13:30:gopher:/var/gopher:/sbin/nologin

ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin

1. 找出/etc/rc.d/rc.sysinit文件或etc/grub.cfg中，以至少一个空白字符开头，且后面非空白字符的行

[root@localhost scripts]# grep "^[[:space:]]\+[^[:space:]]" /etc/grub.conf

root (hd0,0)

kernel /vmlinuz-2.6.32-358.el6.x86\_64 ro root=/dev/mapper/VolGroup-lv\_root nomodeset rd\_NO\_LUKS LANG=en\_US.UTF-8 rd\_NO\_MD rd\_LVM\_LV=VolGroup/lv\_swap SYSFONT=latarcyrheb-sun16 crashkernel=auto rd\_LVM\_LV=VolGroup/lv\_root KEYBOARDTYPE=pc KEYTABLE=us rd\_NO\_DM rhgb quiet

initrd /initramfs-2.6.32-358.el6.x86\_64.img

1. 找出”netstat -tan” 命令中结果中以 ‘LISTEN’后跟0、1或多个空白字符结尾的行

~]# netstat -tan | grep "LISTEN[[:space:]]\*$"

##### 分组及引用

分组符号：

\(\) :将一个或多个字符捆绑在一起，当作一个整体进行处理

\(xy)\*ab

Note: 分组括号中的模式匹配到的内容会被正则表达式引擎自动记录

于内部的变量中，这些变量为：

\1:

\2:

....

###### 后向引用

引用前面的分组括号中的模式所匹配到的字符

[root@localhost scripts]# grep "\(l..e\).\*\1" lover.txt

He loves his lover.

She likes her liker.

非后向引用：

[root@localhost scripts]# grep "l..e" lover.txt

He loves his lover.

He likes his lover.

She likes her liker.

She loves her liker.

### 扩展正则表达式

字符匹配

次数匹配

位置锚定

分组及引用

#### 练习：

1. 找出/proc/meminfo文件中，所有以大写或小写s开头的行（至少三种实现方式）

1). ~]# grep "^[sS]" /proc/meminfo

2).~ ]# grep -i "^s" /proc/meminfo

3).~ ]# grep -E "^(s|S)" /proc/meminfo

1. 显示当前系统上root、leon或user1用户的相关信息

~]# grep -E "^(root|user1|leon)\>" /etc/passwd

note:要用锚定词尾

1. 找出/etc/rc.d/init.d/functions 文件中某单词后面跟一个小括号的行

~]# grep -E "[[:alnum:]]+\(\)" /etc/rc.d/init.d/functions

1. 使用echo命令输出一绝对路径，使用egrep取出基名

~]# echo /etc/sysconfig/ |grep -E -o "[^/]+/?$"

1. 找出ifconfig命令结果中1~255之间的数值
2. 找出ifconfig命令结果中的IP地址

### 文本查找及处理

wc：word count 统计行数、单词数、字节数

cut：文本截取工具

[root@localhost ~]# cut -d: -f 1,7 /etc/passwd

root:/bin/bash

bin:/sbin/nologin

daemon:/sbin/nologin

-f 分隔符

#,# 离散的字段

#-# 连续的字段

# 单个字段

sort：排序文本（速度非常快）

[root@localhost ~]# cut -d: -f 3 /etc/passwd | sort

0

1

10

11

12

13

-t CHAR 指定分隔符

-k # 用于排序比较的字段

-r: 逆序排序

-n: 基于数值大小而非字符

-u: 重复的行只显示一行

[root@localhost ~]# cut -d: -f 7 /etc/passwd | sort -u

/bin/bash

/bin/sync

/sbin/halt

/sbin/nologin

/sbin/shutdown

上述命令与~]# cut -d: -f 7 /etc/passwd | sort |uniq 相等

uniq：移除重复的行

-c 次数统计

-d 仅显示至少出现两次重复的行

-u 仅显示单一的行

diff： 用于比较文本的不同之处

生成补丁： diff /path/to/oldfile /path/to/newfile > /path/to/patch\_file

patch：向文件打补丁

patch /path/to/oldfile < /path/to/patch\_file

## 练习题

### 1. ****请执行命令取出linux中eth0的IP地址(请用cut，有能力者也可分别用awk,sed命令答)****

ifconfig eth0|awk -F '[ :]+' 'NR==2 {print $4}'

解释：

1. NR==2 指定行数
2. –F ‘[ :]+’ 中括号里是一个空格后跟一个冒号，意思是匹配”格或冒号”，后再接一个+意思是可以有多个空格或冒号

整行就被切分为下面的数组

1 2 3 4 5 7 7 8

inet addr:192.168.72.130 Bcast:192.168.72.255 Mask:255.255.255.0

### 2. 批量杀死含有关键词report的进程

ps -ef|grep report|grep -v grep|cut -c 9-15|xargs kill -9

### 3.使用for 循环在/oldboy目录下批量创建10个文件，名称依次为：

oldboy-1

oldboy-2

....

oldboy-10

解法：

#!/bin/sh

for f in `seq 10` #注 seq 10 要用`包住，否则会被shell当成字符串” seq 10 “

do

[ ! -d /oldboy ] && mkdir -p /oldboy

touch /oldboy/oldboy-$f

done

### 4.将上一题文件名中的oldboy全部改成linux(用for循环)

#!/bin/sh

for f in `seq 10`

do

[ ! -d /oldboy ] && mkdir -p /oldboy

mv /oldboy/oldboy-$f /oldboy/linux-$f

done

以后要考虑文件前缀不同的问题

### 5.将/oldboy 目录下以linux为前缀的文件全部改为以oldboy为前缀

解法1：

#!/bin/sh

[ -d /oldboy ]&& cd /oldboy

echo `ls -al`

for f in `ls linux\*`

do

mv $f `echo $f|sed '#linux#oldboy#g'`

done

解法2：

rename oldboy linux \*oldboy\*

### 6.检测10.0.0.0/24 网段内的机器哪些能ping通哪些不能

#!/bin/sh

for n in `seq 254`

do

ping -c2 10.0.0.$n >/dev/null 2>&1

if [ $? -eq 0 ];then

echo "10.0.0.$n is up" >> /tmp/uplist.log

else

echo "10.0.0.$n is down" >> /tmp/downlist.log

fi

done

解释： >/dev/null 2>&1 不需要处理ping结果

$? ping的结果，如果是0则能ping通