一 shell

1、什么是shell？

Shell是在Linux内核与用户之间的解释器程序，通常指的是bash，负责向内核翻译及传达用户/程序指令

shell的使用方式：

1.交互执行指令：人工干预，执行效率低。

2.非交互执行指令：安静地在后台执行，执行效率高，方便写脚本。

2、解释器

/bin/sh

/bin/bash #默认解释器

/sbin/nologin #禁止用户登陆

/usr/bin/sh

/usr/bin/bash

/usr/sbin/nologin #禁止用户登陆

/bin/tcsh

/bin/csh

/bin/ksh

/bin/rksh

修改默认解释器：创建(修改)用户指定默认解释器

1）Bash基本特性

快捷键，Tab补全

命令历史

命令别名

标准输入输出

重定向

管道操作

--历史命令

保存位置：～/.bash\_history

控制历史命令的数量：/etc/profile

[root@server0 ~]# grep ^HISTSIZE /etc/profile

HISTSIZE=1000 #默认记录1000条

[root@svr5 ~]# !1028 #重新执行历史命令列表中的第1028条操作

[root@svr5 ~]# !str #执行最近一次以str开头的历史命令

#history -c

#> ~/.bash\_history #组合清空历史

运算器 控制器 存储器 输入 输出

2)重定向标准输入/输出/错误输出

标准输入（stdin），描述号为0；

标准输出（stdout），描述号为1；

标准错误（stderr），描述号为2;

> :可重定向正确信息

>> :可实现追加正确信息

2> :可重定向错误信息

2>> :可实现追加错误信息

&> :可重定向错误和正确信息

>&2 :将输出信息转换为错误输出( $? != 0)

例子：

[root@server0 ~]# ls /etc/hosts nm.txt >1.txt 2>2.txt

[root@server0 ~]# cat 1.txt

/etc/hosts

[root@server0 ~]# cat 2.txt

ls: 无法访问nm.txt: 没有那个文件或目录

交互式发邮件

[root@server0 ~]# mail -s tile root

非交互式发邮件

[root@server0 ~]# mail -s title root < /etc/passwd

3)管道 |

借助于管道符“|”，可以将一条命令的标准输出交给另一条命令处理，在一条命令行内可依次使用多个管道

4 )shell脚本

脚本声明：解释器

注释：描述，版本

执行代码

5)执行注意事项

bash test.sh #打开子进程执行程序

./test.sh #打开子进程执行程序

source test.sh #不打开子进程执行程序

例子：

#vim test.sh

#!/bin/bash

sleep 1000

另起一个终端查看进程#pstree

source 配置文件 #刷新配置文件

3、变量

本案例要求熟悉Shell变量的使用，主要练习或验证下列内容：

定义/赋值/查看变量

环境/预定义/位置变量的应用

除了学会建立和引用变量以外，还要认识环境变量PWD、USER、HOME、SHELL，还有预定义变量$0、$$、$?、$#、$\*，以及位置变量$1、$2、$10、……的作用。

1）环境变量

[root@server0 ~]# env #查看当前系统自带的环境变量

[root@server0 ~]# set #查看当前系统定义的环境变量(系统自带，自己定义的)

USER ：代表当前用户

HOME：输出当前用户家目录

SHELL：代表当前用户所用的解释器

PWD：当前目录位置

UID：当前用户的uid

PATH：

PS1 ：设置提示

例子：

[root@server0 ~]# PS1=[强哥]#

[强哥]#exit

[root@server0 ~]# echo $PS1

[\u@\h \W]\$

[root@server0 ~]# echo $PS2

>

2）预定义变量

$$：当前进程的进程号

$?：输出上一个程序的返回状态码

$#：统计参数的个数

$\*：代表所有参数变量的值

$$

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…。

例子：

echo $0 //脚本的名称

echo $1 //第一个参数

echo $2 //第二个参数

echo $\* //所有参数

echo $# //所有的综合

echo $$ //当前进程的进程号

echo $? //上一个程序的返回状态码

3）位置变量

$n：表示第几个参数变量

提示：${10} 不能写成$10 $10=${1}0

4）变量的定义/赋值/查看

--新建/赋值变量

新建变量test，赋值“hello world”，通过set命令可以检查变量设置：

[root@svr5 ~]# test=11

--查看变量

通过echo $变量名 可输出变量值：

[root@svr5 ~]# echo $test

11

查看变量时，若变量名称与后面要输出的字符串连在一起，则应该以{}将变量名括起来以便区分：

[root@svr5 ~]# echo $testRMB //无法识别变量名test

[root@svr5 ~]# echo ${test}RMB //区分后可以识别

11RMB

--撤销自定义变量

若要撤销已有的变量，可使用unset命令：

[root@svr5 ~]# unset test //撤销变量test

[root@svr5 ~]# echo $test //查看时已无结果

5)三种引号对变量赋值的影响

--双引号的应用

使用双引号可以界定一个完整字符串。

--单引号的应用

界定一个完整的字符串，并且可以实现屏蔽特殊符号的功能。

--反撇号或$()的应用

使用反撇号或$()时，可以将命令执行的标准输出作为字符串存储，因此称为命令替换。

[root@svr5 ~]# tar -czf log-`date +%Y%m%d`.tar.gz /var/log

4、 read基本用法

执行后从会等待并接受用户输入（无任何提示的情况），并赋值给变量str

为了不至于使用户不知所措、莫名其妙，推荐的做法是结合-p选项给出友好提示：

[root@svr5 ~]# read -p "请输入一个整数：" i

请输入一个整数：240

[root@svr5 ~]# echo $i

240

5、stty终端显示控制

将回显功能关闭（stty -echo），

将回显功能恢复（stty echo）。

可参考下列操作创建一个测试脚本：

[root@svr5 ~]# vim user.sh //创建一个测试脚本

#!/bin/bash

read -p "请输入用户名:" username //读取用户名

stty -echo //关闭回显

read -p "请输入密码:" passwd //读取密码

stty echo //恢复回显

echo "" //恢复回显后补一个空行

useradd "$username"

echo "$passwd" | passwd --stdin "$username"

[root@svr5 ~]# chmod +x user.sh //添加执行权限

执行测试脚本user.sh，验证效果：

[root@svr5 ~]# ./user.sh

请输入用户名: root //输入root，回车

请输入密码: //输入1234567（不会显示），回车

6 export

功能：定义全局变量

二

1 Shell中的数值运算

使用expr、$[ ]、let等整数运算工具：定义变量X=1234，然后计算X与78的四则运算及求模结果

使用bc实现小数运算操作：以交互方式计算12.34与56.78的四则运算结果，另外再以非交互方式重复上述计算，最多显示4位小数

2 整数运算工具

1）使用expr命令

乘法操作应采用 \\* 转义，避免被作为Shell通配符；参与运算的整数值与运算操作符之间需要以空格分开，引用变量时必须加$符号。

首先定义变量X=1234，然后分别计算与78的加减乘除和求模运算结果：

[root@svr5 ~]# X=1234 //定义变量X

[root@svr5 ~]# expr $X + 78 //加法

1312

[root@svr5 ~]# expr $X - 78 //减法

1156

[root@svr5 ~]# expr $X \\* 78 //乘法，操作符应添加\转义

96252

[root@svr5 ~]# expr $X / 78 //除法，仅保留整除结果

15

[root@svr5 ~]# expr $X % 78 //求模

64

2）使用$[]或$(())表达式

乘法操作\*无需转义，运算符两侧可以无空格；引用变量可省略 $ 符号；计算结果替换表达式本身，可结合echo命令输出。

同样对于变量X=1234，分别计算与78的加减乘除和求模运算结果：

[root@svr5 ~]# X=1234

[root@svr5 ~]# echo $[X+78]

1312

[root@svr5 ~]# echo $[X-78]

1156

[root@svr5 ~]# echo $[X\*78]

96252

[root@svr5 ~]# echo $[X/78]

15

[root@svr5 ~]# echo $[X%78]

64

[root@server0 ~]# echo $[x\*\*2] #x=20 ,20的2次幂

400

3）使用let命令

expr或$[]、$(())方式只进行运算，并不会改变变量的值；而let命令可以直接对变量值做运算再保存新的值。因此变量X=1234，在执行let运算后的值会变更；另外，let运算操作并不显示结果，但是可以结合echo命令来查看：

[root@svr5 ~]# X=1234

[root@svr5 ~]# let y=X+22

[root@svr5 ~]# echo $y

1256

[root@svr5 ~]# let X++; echo $X # X++(X=X+1)

[root@svr5 ~]# let X--; echo $X # X--(X=X-1)

[root@svr5 ~]# let X+=78 ; echo $X # X+=78(X=X+78)

[root@svr5 ~]# let X-=78 ; echo $X # X-=78(X=X-78)

[root@svr5 ~]# let X\*=78 ; echo $X # X\*=78(X=X\*78)

[root@svr5 ~]# let X/=78 ; echo $X # X/=78(X=X/78)

[root@svr5 ~]# let X%=78 ; echo $X # X%=78(X=X%78)

3 小数运算工具

1）bc交互式运算

先执行bc命令进入交互环境，然后再输入需要计算的表达式。以计算小数12.34与5.678的四则运算为例，相关操作如下：

[root@svr5 ~]# bc

bc 1.06.95

Copyright 1991-1994, 1997, 1998, 2000, 2004, 2006 Free Software Foundation, Inc.

This is free software with ABSOLUTELY NO WARRANTY.

For details type `warranty'.

12.34+56.78 //加法

69.12

12.34-56.78 //减法

-44.44

12.34\*56.78 //乘法

700.66

12.34/56.78 //除法

0

quit //退出交互计算器

[root@svr5 ~]#

2）bc非交互式运算

将需要运算的表达式通过管道操作交给bc运算。注意，小数位的长度可采用scale=N限制，除此以外也受参与运算的数值的小数位影响。以计算小数12.34与5.678的四则运算为例，相关操作如下：

[root@svr5 ~]# echo 'scale=4;12.34+5.678' | bc

18.018

[root@svr5 ~]# echo 'scale=4;12.34\*5.678' | bc

70.0665

[root@svr5 ~]# echo 'scale=4;12.34/5.678' | bc

2.1733

4、 条件测试

字符串匹配

比较整数值的大小

识别文件/目录的状态

多个条件/操作的逻辑组合

1）语法格式

使用“test 表达式”或者[ 表达式 ]都可以，表达式两边至少要留一个空格。

条件测试操作本身不显示出任何信息。测试的条件是否成立主要体现在命令执行后的返回状态（即 $?），所以可以在测试后查看变量$?的值来做出判断，或者结合&&、||等逻辑操作显示出结果（或作其他操作） 。

==： 比较两个字符串是否相同

!= ：比较两个字符串是否不相同

一行执行多条命令的情况

# A && B //仅当A命令执行成功，才执行B命令

# A || B //仅当A命令执行失败，才执行B命令

# A ; B //执行A命令后执行B命令，两者没有逻辑关系

-z 检查变量的值是否未设置（空值）

[root@svr5 ~]# var1="nb" ; var2=""

[root@svr5 ~]# [ -z "$var1" ] && echo "空值" || echo "非空值"

非空值

[root@svr5 ~]# [ -z $var2 ] && echo "空值" || echo "非空值"

空值 //变量var2已设置，但无任何值，视为空

2)整数值比较

-eq ：比较两个数是否相等

-ne ：比较两个数是否不相等

-gt ：比较前面的整数是否大于后面的整数

-ge ：比较前面的整数是否大于或等于后面的整数

-lt ：比较前面的整数是否小于后面的整数

-le ：比较前面的整数是否小于或等于后面的整数

3）识别文件/目录的状态

-e ：判断对象是否存在（不管是目录还是文件）

-d ：判断对象是否为目录（存在且是目录）

-f ：判断对象是否为文件（存在且是文件）

-r ：判断对象是否可读

-w ：判断对象是否可写

-x ：判断对象是否具有可执行权限

3）多个条件/操作的逻辑组合

--&&，逻辑与

给定条件必须都成立，整个测试结果才为真。

--||，逻辑或

只要其中一个条件成立，则整个测试结果为真。

三 条件，循环

1 if

if 条件测试|命令

fi

if 条件测试

then

命令序列1

else

命令序列2

fi

if 条件测试1 ;then

命令序列1

elif 条件测试2 ;then

命令序列2

else

命令序列n

fi

2 for

RANDOM ：产生随机数

random

for 随便一个未定义变量 in 参数，列表，命令

do

done

1）{a..n} ,seq n

{1..n}=`seq n`

区别：

{}里面不能有变量，seq 后可以是变量

for num in {a..n} #seq n

do

done

例子：

#!/bin/bash

for i in {1..5}

do

for j in {1..8}

do

echo -n "\* " #-n不换行

done

echo #默认换行

done

echo -e ‘ \033[32mOK\033[0m’

2）命令执行结果(值列表)

for num in 命令

do

done

for num in `cat /etc/passwd`

do

echo $num

done

3）（C语言风格的for循环语法格式）

[root@svr5 ~]# vim cfor.sh

#!/bin/bash

for ((i=1;i<=5;i++))

do

echo $i

done

4)break,continue,exit

3 while

while 条件测试

do

命令

done

常用死循环格式

while :

do

命令

done

4 case

格式:

case 变量值 in

模式1|模式5|模式6)

命令1;;

模式2)

命令2;;

\*)

命令21

esac

5 函数

unset 函数名 #取消函数名

wait :等待该脚本的程序执行完结束后回到命令行

echo -e “\033[32mOK\033[0m”

0x： 样式

3x：字体色 #31红色，32绿色

4x：背景色

10x：高亮色

1）

function 函数名(){

n命令序列

}

2）常用

函数名(){

n命令序列

}

四 字符串，正则表达式

字符串初值的处理

${var:-word}

若变量var已存在且非Null，则返回 $var 的值；否则返回字串“word”，原变量var的值不受影响

1. 字符串统计个数

[root@room9pc01 ~]# p=123456

[root@room9pc01 ~]# echo ${#p} #统计变量长度

6

2 子串截取的三种用法：

1）${变量名:起始位置:长度}

提示：使用${}方式截取字符串时，起始位置是从0开始的

例子：

[root@room9pc01 ~]# p=123456

[root@room9pc01 ~]# echo ${#p}

6

[root@room9pc01 ~]# echo ${p:1:3}

234

2）expr substr "$变量名" 起始位置 长度

提示：使用expr substr截取字符串时，起始编号从1开始，这个要注意与${}相区分

例子：

[root@room9pc01 ~]# p=123456789

[root@room9pc01 ~]# expr substr "$p" 1 5

12345

3）echo $变量名 | cut -b 起始位置-结束位置

提示：选项 -b 表示按字节截取字符，其中起始位置、结束位置都可以省略。当省略起始位置时，视为从第1个字符开始（编号也是从1开始，与expr类似），当省略结束位置时，视为截取到最后。

例子：

[root@room9pc01 ~]# echo $p

123456789

[root@room9pc01 ~]# echo $p | cut -b 1-5

12345

[root@room9pc01 ~]# echo $p | cut -b 3-5

345

[root@room9pc01 ~]# echo $p | cut -b 3,5-6,8 #位置可连续，也可间断

3568

3 子串替换的两种用法：

1)只替换第一个匹配结果：${变量名/old/new}

2)替换全部匹配结果：${变量名//old/new}

4 字符串掐头去尾：

1)从左向右，最短匹配删除：${变量名#\*关键词}

[root@room9pc01 ~]# echo $A

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

[root@room9pc01 ~]# echo ${A#\*:}

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

2)从左向右，最长匹配删除：${变量名##\*关键词}

[root@room9pc01 ~]# echo $A

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

[root@room9pc01 ~]# echo ${A##\*:}

/bin/bash

3)从右向左，最短匹配删除：${变量名%关键词\*}

[root@room9pc01 ~]# echo $A

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

[root@room9pc01 ~]# echo ${A%:\*}

root:x:0:0:root:/root

4)从右向左，最长匹配删除：${变量名%%关键词\*}

[root@room9pc01 ~]# echo $A

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

[root@room9pc01 ~]# echo ${A%%:\*}

root

5）变量默认值

${min:-123} #min有值用他的值，没有值默认值为123

6)expect预期交互

实现SSH登录的自动交互

expect可以为交互式过程（比如FTP、SSH等登录过程）自动输送预先准备的文本或指令，而无需人工干预。触发的依据是预期会出现的特征提示文本

#yum -y install expect

1）<<

[root@room9pc01 下载]# mail -s jluo root << EOF

> dfsg

> dgsd

> dgs

> EOF

2）非交互命令

ip=192.168.4.10

expect << EOF

spawn ssh "$ip"

expect "password" { send "123456\n" } #必须用双引号

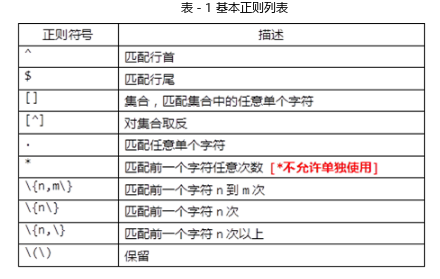
expect "#" { send "touch /test.txt\n"}

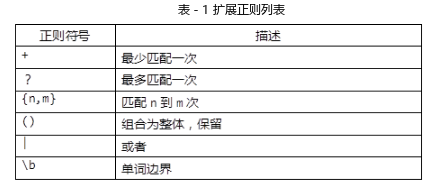
expect "#" { send "exit \n"}

EOF

提示：必须要用双引号，单引号会出错

7)正则表达式





基本正则兼容性强，几乎所有的软件都支持，书写麻烦

扩展正则兼容性差

例子：

[root@room9pc01 shell-test]# egrep "the\b" test/a.txt

the sdfsdfsdfsf

dfsfsdf dsfsthe dfsdfdffg

gfgdg gfdfg the fgfgdf

[root@room9pc01 shell-test]# egrep "\bthe\b" test/a.txt

the sdfsdfsdfsf

gfgdg gfdfg the fgfgdf

8）sed

Stream Editor 流式编辑器

特点：

非交互式，基于模式匹配过滤及修改文本

逐行处理，并将结果输出到屏幕

可实现对文本的输出，删除，替换，复制，剪切，导入，导出等各种操作

--用法

用法1：前置命令 | sed [选项] '条件指令'

用法2：sed [选项] '条件指令' 文件.. ..

条件可以是行号或者/正则/

没有条件时，默认为所有条件

指令可以是增、删、改、查等指令

默认sed会将所有输出的内容都打印出来，可以使用-n屏蔽默认输出

选项中可以使用-r选项，让sed支持扩展正则

--认识sed工具的基本选项

sed命令的常用选项如下：

-n（屏蔽默认输出，默认sed会输出读取文档的全部内容）

-r（让sed支持扩展正则）

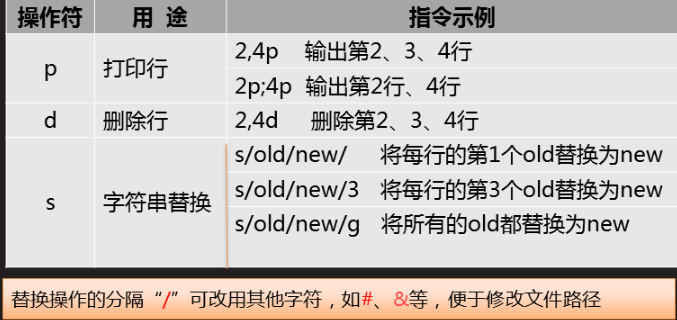
-i（sed直接修改源文件，默认sed只是通过内存临时修改文件，源文件无影响）

--常用的动作指令(条件指令)

p：打印行

d：删除行

s：字符替换



注意：替换操作的分隔“/”可改用其他字符，如#、&等，便于修改文件路径

--sed工具的多行文本处理操作：

i： 在指定的行之前插入文本 insert

a：在指定的行之后追加文本 append

c：替换指定的行

r:读取

w:另存

例子：

##

[root@room9pc01 shell-test]# sed '2r user.txt' test.txt

hello

2017 2011 2018

aa

bb

2017 2017 2024

2017 2017 2017

2245fdsfdf

##

[root@room9pc01 shell-test]# sed 'w user' test.txt

hello

2017 2011 2018

2017 2017 2024

2017 2017 2017

2245fdsfdf

[root@room9pc01 shell-test]# cat user

hello

2017 2011 2018

2017 2017 2024

2017 2017 2017

2245fdsfdf

##

[root@room9pc01 shell-test]# sed '2w user' test.txt

hello

2017 2011 2018

2017 2017 2024

2017 2017 2017

2245fdsfdf

[root@room9pc01 shell-test]# cat user

2017 2011 2018

##

[root@svr5 ~]# sed '3,5d' a.txt //删除第3~5行

[root@svr5 ~]# sed '/xml/d' a.txt //删除所有包含xml的行

[root@svr5 ~]# sed '/xml/!d' a.txt //删除不包含xml的行，!符号表示取反

[root@svr5 ~]# sed '/^install/d' a.txt //删除以install开头的行

[root@svr5 ~]# sed '$d' a.txt //删除文件的最后一行

[root@svr5 ~]# sed '/^$/d' a.txt //删除所有空行

##

1~2p：1 1+2=3 3+3=5 …..

2~2p：2 2+2=4 4+2=6 …..

打印奇数行：

[root@svr5 ~]# sed -n '1~2p' /etc/passwd

打印偶数行：

[root@svr5 ~]# sed -n '2~2p' /etc/passwd

##

[root@room9pc01 shell-test]# sed -n 's#/bin/bash#/sbin/sh#gp' passwd

root:x:0:0:root:/root:/sbin/sh

Student:x:1000:1000::/home/Student:/sbin/sh

xiaohei:x:1002:1001::/sbin/nologin:/sbin/sh

注意：替换操作的分隔“/”可改用其他字符，如#、&等，便于修改文件路径

##

[root@svr5 ~]# vim anonftp.sh

#!/bin/bash

yum -y install vsftpd //安装vsftpd软件

cp /etc/vsftpd/vsftpd.conf{,.bak} //备份默认的配置文件

sed -i "s/^#anon/anon/" /etc/vsftpd/vsftpd.conf //修改服务配置

chmod 777 /var/ftp/pub //调整目录权限

systemctl start vsftpd //启动服务

systemctl enable vsftpd //设为自动运行

[root@svr5 ~]# chmod +x anonftp.sh

[root@svr5 ~]# ./anonftp.sh

--sed复制剪切

H：模式空间 –[追加] -->保持空间

h：模式空间 –[覆盖] -->保持空间

G：保持空间 –[追加] -->模式空间

g：保持空间 –[覆盖] -->模式空间

例子：

##

[root@room9pc01 shell-test]# sed '2h;5g' 10.txt

1

2

3

4

2

6

7

8

9

10

##

[root@room9pc01 shell-test]# sed '2H;5G' 10.txt

1

2

3

4

5

2

6

7

8

9

10

##

[root@room9pc01 shell-test]# sed '2h;5G' 10.txt

1

2

3

4

5

2

6

7

8

9

10

9）awk

awk编程语言/数据处理引擎

Aho Weinberger Kernighan

特点：

通常用在shell脚本中，获取指定的数据

单独使用时，可对文本数据做统计

用于过滤数据

逐行处理

--用法

格式1：前置命令 | awk [选项] ‘[条件]{指令}’

格式2：awk [选项] ‘[条件]{指令}’ 文件….

提示：引号只能是单引号

--基本操作方法

格式：awk [选项] '[条件]{指令}' 文件

其中，print 是最常用的编辑指令；若有多条编辑指令，可用分号分隔。

Awk过滤数据时支持仅打印某一列，如第2列、第5列等。

处理文本时，若未指定分隔符，则默认将空格、制表符等作为分隔符。

[root@svr5 ~]# df -h | awk '{print $4}' //打印磁盘的剩余空间

--选项 -F 可指定分隔符

[root@svr5 ~]# awk -F: '{print $1,$7}' /etc/passwd

root /bin/bash

bin /sbin/nologin

daemon /sbin/nologin

awk还识别多种单个的字符，比如以“:”或“/”分隔，输出第1、10个字段

[root@svr5 ~]# awk -F [:/] '{print $1,$10}' /etc/passwd

root bash

bin nologin

daemon nologin

adm sbin

… …

--awk常用内置变量：

$0 文本当前行的全部内容

$1 文本的第1列

$2 文件的第2列

$3 文件的第3列，依此类推

NR 文件当前行的行号

NF 文件当前行的列数（有几列）

--awk处理的时机

BEGIN{ } 行前处理，读取文件内容前执行，指令执行1次

{ } 逐行处理，读取文件过程中执行，指令执行n次

END{ } 行后处理，读取文件结束后执行，指令执行1次

格式：awk ‘BEGIN{}条件{}END{}’

提示：字符串里面要用双引号

输出信息时，可以使用“\t”显示Tab制表位：

##

[root@room9pc01 shell-test]# awk -F: 'BEGIN{print "User","UID","Home"}{print $1,$3,$6}END{print "Total",NR,"lines"}' /etc/passwd | column -t

User UID Home

root 0 /root

bin 1 /bin

daemon 2 /sbin

column -t ：把数据对齐显示

##

[root@room9pc01 shell-test]# awk 'BEGIN{print "begin"}{}END{print "end"}' test.txt

begin

end

##

[root@room9pc01 shell-test]# awk "BEGIN{x=0}/bash$/{x++}END{print x}" /etc/passwd

3

--awk处理条件

##正则设置条件

输出账户名称包含root的基本信息（第1列包含root）：

[root@svr5 ~]# awk -F: '$1~/root/' /etc/passwd

输出其中登录Shell不以nologin结尾（对第7个字段做!~反向匹配）的用户名、登录Shell信息：

[root@svr5 ~]# awk -F: '$7!~/nologin$/{print $1,$7}' /etc/passwd

root /bin/bash

sync /bin/sync

shutdown /sbin/shutdown

##数值/字符串比较设置条件

比较符号：==(等于) !=（不等于） >（大于）

>=（大于等于） <（小于） <=（小于等于）

&& ||

--awk流程控制

if分支结构（单分支、双分支、多分支）

练习awk数组的使用

##awk过滤中的if分支结构

if (条件){

}

else if(条件){

}

else{

}

##awk数组

数组是一个可以存储多个值的变量，具体使用的格式如下：

定义数组的格式：数组名[下标]=元素值

调用数组的格式：数组名[下标]

遍历数组的用法：for(变量 in 数组名){print 数组名[变量]}。

[root@svr5 ~]# awk 'BEGIN{a[0]=11;a[1]=88;print a[1],a[0]}'

88 11

[root@svr5 ~]# awk 'BEGIN{a++;print a}'

1

[root@svr5 ~]# awk 'BEGIN{a0++;print a0}'

1

[root@svr5 ~]# awk 'BEGIN{a[0]++;print a[0]}'

1

[root@svr5 ~]# awk 'BEGIN{a[0]=0;a[1]=11;a[2]=22; for(i in a){print i,a[i]}}'

0 0

1 11

2 22

注意，awk数组的下标除了可以使用数字，也可以使用字符串，字符串需要使用双引号：

[root@svr5 ~]# awk 'BEGIN{a["hehe"]=11;print a["hehe"]}'

11