# SHELL

Day1

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Shell与bash的区别 (包括与被包括)

cat /etc/shells //解释器环境

bash优点： 具有快捷键、历史命令、自动补齐、别名

脚本执行方法：

bash、sh、source（不开子进程）、. 、文件路径，来加载脚本文件

变量

//查看环境变量env

//查看全部变量set

//全局变量export

Day2

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

整数运算工具

Expr $[] $(()) Let i++

小数运算工具

bc (也可以做判断使用 echo “3 > 2”| bc）

条件测试

test 表达式

[ 表达式 ]

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 判断是否为空 [ -z $a ]

整数值比较 （比较非整数值时会出错）

-eq -ne -gt -ge -lt -le

字符串比较

= 等于,如:if [ "$a" = "$b" ]

== 等于,如:if [ "$a" == "$b" ], 与=等价

注意:==的功能在[[]]和[]中的行为是不同的,如下:

1 [[ $a == z\* ]] # 如果$a以"z"开头(模式匹配)那么将为true

2 [[ $a == "z\*" ]] # 如果$a等于z\*(字符匹配),那么结果为true

3

4 [ $a == z\* ] # File globbing 和word splitting将会发生

5 [ "$a" == "z\*" ] # 如果$a等于z\*(字符匹配),那么结果为true

!= 不等于,如:if [ "$a" != "$b" ]， 这个操作符将在[[]]结构中使用模式匹配.

< 小于,在ASCII字母顺序下.如:

if [[ "$a" < "$b" ]]

if [ "$a" \< "$b" ] 在[]结构中"<"需要被转义.

> 大于,在ASCII字母顺序下.如:

if [[ "$a" > "$b" ]]

if [ "$a" \> "$b" ] 在[]结构中">"需要被转义.

-z 字符串为"null".就是长度为0.

-n 字符串不为"null"

文件状态

-e -d -f -r -w -x //root对所有文件具有rw权限，没有x权限

if选择结构

单分支：

if 条件测试；then

命令序列

fi

双分支：

if 条件测试；then

命令序列1

else

命令序列2

fi

多分支：

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

// who 当前登陆用户

// ping 为了节省ping测试时间，可以只发送3个测试包（-c 3）、缩短发送测试包的间隔秒数（-i 0.2）、等待反馈的超时秒数（-W 3）

Day3

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

for循环

for 变量名 in 值列表

do

命令序列

done

for((i=0,i<10,i++))

do

命令序列

done

//造数工具 {1..10}（括号内不能有变量） $RANDOM seq 3 2 10(从3开始每次加2)

While循环

while 条件测试

do

命令序列

done

//let i++

Case循环

case 变量值 in

模式1)

命令序列1 ;;

模式2)

命令序列2 ;;

.. ..

\*)

默认命令序列

esac

Shell函数

Function 函数名{

命令序列

... ...

}

函数名（）{

... ...

}

//break结束并退出循环;

//continue 在循环中不执行continue下面的代码，转而进入下一轮循环;

//exit退出脚本，常带一个整数给系统，如exit 0;

//Return 在函数中将数据返回或返回一个结果给调用函数的脚本;

//echo -e “\33[35m OK \033[0m”

0x样式 3x字体颜色 4x 背景颜色

Day4

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

字符串处理

子串截取、子串替换、按条件掐头去尾、变量初始值处理

子串截取的三种用法：

${变量名:起始位置:长度} //起始位置0

expr substr "$变量名" 起始位置 长度 //起始位置1

echo $变量名 | cut -b 起始位置-结束位置

echo $var | cut -b 1,3,5 //逗号截取几个不连续的字符串

子串替换的两种用法：

只替换第一个匹配结果：${变量名/old/new}

替换全部匹配结果：${变量名//old/new}

字符串掐头去尾：

从左向右，最短匹配删除：${变量名#\*关键词}

从左向右，最长匹配删除：${变量名##\*关键词}

从右向左，最短匹配删除：${变量名%关键词\*}

从右向左，最长匹配删除：${变量名%%关键词\*}

//字符串连接 c=$a$b

//初值的检测与设置

${var:-word} //若var存在且非空则返回$var,否则返回word

${var:=word} //若var存在且非空则返回$var,否则返回word，并附值给变量var

expect预期交互

yum -y install expect //必须安装软件

#!/bin/bash

expect << EOF

spawn ssh 172.25.0.10                          #//创建交互式进程

expect "password:" { send "123456\r" }             #//自动发送密码

expect "#” { send "pwd > /tmp/$user.txt \r" } #//发送命令

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

EOF

Shell中通常将EOF（End Of File）与 << 结合使用，表示后续的输入作为子命令或子Shell的输入，直到遇到EOF为止，再返回到主调Shell。可以把EOF替换成其他东西，意思是把内容当作标准输入传给程序。

正则表达式

通配符是系统level的 而正则表达式需要相关工具的支持: egrep, awk, vi, perl。

在文本过滤工具里，都是用正则表达式，比如像awk，sed等，是针对文件的内容的。

通配符多用在文件名上，比如查找find，ls，cp，等等。

基础正则表达式  
grep -[acinv] '搜索内容串' filename  
-a 以文本文件方式搜索  
-c 计算找到的符合行的次数  
-i 忽略大小写  
-n 顺便输出行号  
-v 反向选择，即找没有搜索字符串的行

^ 匹配行首  
$ 匹配行尾   
. 匹配任意单个字符

\* 匹配零个或多个先前字符    
.\*一起用代表任意字符。   
[] 匹配一个指定范围内的字符，如'[Gg]rep'匹配Grep和grep。   
[^] 匹配一个不在指定范围内的字符  
\(..\) 标记匹配字符，如'\(love\)'，love被标记为1。   
\< 锚定单词的开始，  
\> 锚定单词的结束，如'grep\>'匹配包含以grep结尾的单词的行。   
x\{m\} 重复字符x，m次，如：'o\{5\}'匹配包含5个o的行。

x\{m,\} 重复字符x,至少m次，如：'o\{5,\}'匹配至少有5个o的行。   
x\{m,n\} 重复字符x，至少m次，不多于n次，如：'o\{5,10\}'匹配5--10个o的行。   
\w 匹配文字和数字字符，也就是[A-Za-z0-9\_]，如：'G\w\*p'匹配以G后跟零个或多个文字或数字字符，然后是p。   
\W \w的反置形式，匹配一个或多个非单词字符，如点号句号等。   
\b 单词锁定符，如: '\bgrep\b'只匹配grep。

扩展正则表达式

+ 最少匹配一次

？ 最多匹配一次

{n,m} 匹配n到m次

() 组合为整体，保留

| 或者

\b 单词边界

Day5

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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.. ..

选项

-n, --quiet, --silent

（屏蔽默认输出，默认sed会输出读取文档的全部内容）

-e 脚本, --expression=脚本

添加“脚本”到程序的运行列表

-f 脚本文件, --file=脚本文件

添加“脚本文件”到程序的运行列表

--follow-symlinks

直接修改文件时跟随软链接

-i[SUFFIX], --in-place[=SUFFIX]

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

-c, --copy

use copy instead of rename when shuffling files in -i mode

-b, --binary

does nothing; for compatibility with WIN32/CYGWIN/MSDOS/EMX (

open files in binary mode (CR+LFs are not treated specially))

-l N, --line-length=N

指定“l”命令的换行期望长度

--posix

关闭所有 GNU 扩展

-r, --regexp-extended

在脚本中使用扩展正则表达式

-s, --separate

将输入文件视为各个独立的文件而不是一个长的连续输入

-u, --unbuffered

从输入文件读取最少的数据，更频繁的刷新输出

-z, --null-data

separate lines by NUL characters

条件：

行号：

a,b（a到b行）

1p;3p(打印1行和3行)

a～b(打印a行，及以b为跳跃的行)

a ,+b（打印a行及后面的b行）

正则： /正则表达式/

！表示取反 如：sed '/xml/!d' a.txt #删除不包含xml的行

指令

P：打印输出

D：删除

S：替换（s/旧内容/新内容/选项）

sed 's/xml/XML/3' a.txt     //将每行中的第3个xml替换为XML

sed 's/xml/XML/g' a.txt     //将所有的xml都替换为XML

sed -r 's/^(.)(.\*)(.)$/\3\2\1/' nssw.txt //使用“（）”可实现保留功能

a 追加

sed '2a XX' a.txt //在第二行后面，追加XX

i 插入

sed '2i XX' a.txt //在第二行前面，插入XX

c 替换行

sed '2c XX' a.txt //将第二行替换为XX

Day5

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

格式1：awk [选项] '[条件]{指令}' 文件

格式2：前置命令 | awk [选项]‘[条件]{指令}’

选项命令：

-F 指定分隔符

指令：

Print 打印输出

输出字符串需要应用引号“”

awk if分支结构

awk -F: '{if($7~/bash$/){i++}}END{print i}' /etc/passwd

awk数组

awk’{ip[$1]}++’ //用于统计次数

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

awk 'BEGIN{a[0]=00;a[1]=11;a[2]=22; for(i in a){print i,a[i]}}'

条件

1. 使用正则设置条件

/正则表达式/

2）使用数值/字符串比较设置条件

==(等于) !=（不等于） >（大于）>=（大于等于） <（小于） <=（小于等于）

1. 逻辑测试条件

&&、||

4）数学运算

awk '$1%3==0{i++} END{print i}'

awk -F: '{if($7~/bash$/){i++}}END{print i}' /etc/passwd

//内置变量

FS 保存或设置字段分隔符

$n 指定分隔的第n个字段

$0 当前读入的整行文本内容

NF 记录当前处理行的字段个数（列数）

NR 记录当前已读入行的数量（行数）

格式3：awk [选项] ' BEGIN{指令} {指令} END{指令}' 文件

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

awk -F: '$1~/root/' /etc/passwd //第一列包含root，不写指令默认打印全部

awk -F: '/^(root|adm)/{print $1,$3}' /etc/passwd //用户名为root或adm

awk -F: '$7!~/nologin$/{print $1,$7}' /etc/passwd //登录Shell不以nologin结尾

Column 格式化对齐输出