Shell学习笔记

Shell定义

Shell是一个命令行解释器，它接收应用程序/用户命令，然后调用操作系统内核
Shell还是一个功能相当强大的编程语言，易编写、易调试、灵活性强

脚本格式

脚本以 #!/bin/bash 开头(指定解释器)

脚本常用执行方式

- 采用bash或sh + 脚本的相对路径或绝对路径(启用bash进程进行脚本解析，不调用脚本，不用赋予脚本+x权限)
    - sh或bash + 脚本相对路径
        - sh(bash) ./文件名.sh
    - sh或bash + 脚本绝对路径
        - sh(bash) /home/....(路径)/文件名.sh
- 采用输入脚本的绝对路径/相对路径执行脚本(直接调用脚本，在当前环境执行脚本，必须具有可执行权限+x)
    - 相对路径：./文件名.sh
    - 绝对路径：/home/...(路径)/文件名.sh
- [特殊]在脚本的相对路径/绝对路径前加上"."或source
    - source 文件.sh(/home/.../文件.sh)
    - . 文件.sh(/home/.../文件.sh)
      第一、二种的./指的是相对路径，而第三种的.是一个.命令。第一、二种的执行方式是创建一个子shell进行执行，当前shell环境不受影响，第三种执行方式是直接在当前shell环境下解析执行，不创建子shell，避免了父子shell嵌套的麻烦。子shell设置的变量对父shell是不可见的。

变量

变量分为系统预定义变量和用户自定义变量，也可分为全局变量和局部变量
全局变量：对其子shell有效，但对全局变量的更改只对当前shell有效，对其他shell无效
局部变量：只对当前shell有效，父、子shell都不可见
使用 ps -f 命令查看变量在哪一个shell里

系统预定义变量

常用系统变量

$HOME,$PWD,$USR,$SHELL,$PATH等
输出的时候echo $HOME定义的是一个变量，而printenv HOME只是打印内容
显示当前shell中所有的变量：set

用户自定义变量

基本语法

- 定义局部变量：变量名=变量值，注意的是=前后不能有空格，不然会把空格前面的当成一个命令而无法执行
- 定义全局变量：先定义局部变量，然后执行 export 变量名 就成了全局变量
- 定义只读变量：readonly 变量名=变量值，只读变量不能撤销unset
- 撤销变量：unset 变量名

变量定义规则

- 变量名称由数字、字母、下划线组成，不以数字开头。全局变量建议大写
- 等号两侧没有空格
- 在bash中，变量默认类型是字符串，无法进行数值计算
- 变量的值有空格需要用双引号或单引号引起来

特殊变量

输出$n这个字符不能用"",而用'',""脚本会认为$n是一个参数，而''则认为$n是字符

基本语法

- $n：n为数字，$0代表该脚本名称，$1-9代表第一到九个参数，十以上的参数用大括号包含${10}
- $#：获取所有输入参数的个数，常用于循环，判断参数个数是否正确、加强脚本健壮性
- $*：获取当前命令行里提供的所有参数，*$*把所有的参数看成一个整体
- $@：获取当前命令行里提供的所有参数，$@把每个参数区别对待，获取的是一个参数集合，可以遍历每个参数
- $?：最后一次命令的返回状态，0表示正确执行，非0表示上一个命令执行不正确

运算符

在linux里可以执行 expr 数字 运算符 数字 这样的形式得到一个计算结果，乘法运算符*需要用*来替换

基本语法

- $((运算式))
- $[运算式]

如果要把expr命令的执行结果复制到变量里，有两种方式

- a='expr 数字 运算符 数字'
- a=$(expr 数字 运算符 数字)

条件判断

基本语法

条件表达式判断符号前后要有空格，如[$a = hello]判断a的值是否是hello，[$a != hello]判断a的值是否是hello

- test 条件表达式：判断这个条件表达式是真还是假，该命令没有返回值，需要用$?来判断表达式的真假，0表示真，非0表示假
- [条件表达式]：表达式前后必须要有空格

常用判断条件

- 两个整数之间比较
  - -eq 等于(equal)
  - -ne 不等于(not equal)
  - -lt 小于(less than)
  - -le 小于等于(less equal)
  - -gt 大于(greater than)
  - -ge 大于等于(greater equal)
- 字符串之间比较
  - = 等于
  - != 不等于
- 按照文件权限进行判断
  - -r 有读权限(read)
  - -w 有写权限(write)
  - -x 有执行权限(execute)
- 按照文件类型判断
  - -e 文件存在(existence)
  - -f 文件存在并且是一个常规的文件(file)
  - -d 目录存在并且是一个目录(directory)

多条件判断

- &&：表示前一条正确才会判断下一条
- ||：表示上一条执行失败才会判断下一条

流程控制

多个条件判断可以写在同一个条件判断语句中，但不能用&&或||表示而改成-a或-o
如[$a - gt18 - a$a -lt 35]、[$a - lt18 - o$a -gt 35]

if判断

基本语句

- 单分支
  - 第一种

```
if [ 条件判断式 ]; then
  程序
fi
```

- 第二种

```
if [ 条件判断式 ]
then
  程序
fi
```

- 多分支

```
if [ 条件判断式 ]
then
  程序
elif [ 条件判断式 ]
then
  程序
else
  程序
fi
```

注意事项:

- 中括号内的条件判断式前后要有空格
- if的后面也要有空格
- 多分支判断如果走到某个if变成了true则不再继续往下判断

case语句

基本语句

```
case $变量名 in
"值1")
  程序1 #如果变量的值等于值1,则执行程序1
;;
"值2")
  程序2 #如果变量的值等于值1,则执行程序2
;;
*)
  程序default #如果变量的值都不是以上的值,执行default
;;
esac
```

注意事项

- case的行尾单词必须是"in",每一个模式匹配必须以右括号")"结束
- 双分号";;"表示命令序列结束,也就是break
- *)表示默认模式,相当于default

for循环

基本语句

- 第一种

```
for (( 初始值; 循环控制条件; 变量变化 ))
do
  程序
done
```

- 第二种

```
for 变量 in 值1 值2 值3...
do
  程序
done
```

该语句中的变量取值范围就是in后面的值1、值2、值3...,相当于遍历序列的值

(())双小括号内的循环控制条件可以使用<=、<、>、>=等数学上的运算表达式

$*和$\@的区别

如果$*和$@在for循环中都不加""，效果是一样的，如果加上""，即"$"、"$@"，则$*是把输入的参数作为一个整体，集合大小为1遍历，而$@还是把集合内的每一个元素当作独立的一个元素来输出

while循环

基本语法

```
while [ 条件判断式 ]
do
  程序
done
```

read读取控制台输入

基本语法

```
read (选项) (参数)
```

- 选项
  - -p：指定读取值时的提示符
  - -t：指定读取值时等待的时间(秒)，如果-t不加则表示一直等待
- 参数
  - 变量：指定读取值的变量名

实操

提示7秒内，读取控制台输入的名称

```
#!/bin/bash

read -t 7 -p "Enter your name in 7 seconds: " name
echo $name
```

函数

系统函数

利用 $(系统函数) 来调用系统给定的函数

- basename
  - 基本语法

```
basename [string / pathname] [suffix]
```

basename命令会删掉所有的前缀包括最后一个('/')字符，然后将字符串显示出来
basename可以理解为取路径里的文件名称
suffix为后缀，如果suffix被指定了，basename会将pathname或string中的suffix去掉

- 实操
  返回文件的文件名并删除后缀
  绝对路径

```
basename /home/....../shicao.txt .txt
```

相对路径

```
basename ./shicao.txt .txt
```

- dirname
  - 基本语法

```
dirname 文件绝对路径
```

从给定的包含绝对路径的文件名中去除文件名(非目录部分)，然后返回剩下的路径(目录部分)
dirname可以理解为取文件路径的绝对路径名称

- 实操
  返回当前文件的目录路径

```
cd $(dirname $0) #切换到文件的目录下
echo $(pwd) #利用系统命令直接返回路径
#或者直接echo $(cd $(dirname $0); pwd) 利用;把两条命令隔开，表示先后执行相应命令
```

自定义函数

基本语法

```
[ function ] funname[()] {  #中括号可以省略
  Action;
  [return int;] #return的数字只能是0-255 return这一句可有可无
}
```

注意事项

- 必须在调用函数前，先声明函数，shell是逐行运行的
- 函数返回值只能通过$?系统变量获得，可以加return返回，如果没有return，则将最后一条命令运行结果作为返回值，return后面只能是数字，而且范围在0到255

实操

计算a和b的和

```
#!/bin/bash

function add() {
  s=$[$1 + $2]
  echo $s
}

read -p "输入a的值：" a
read -p "输入b的值：" b

sum = $(add $a $b)

echo "和为："$sum
```

归档文件

实际生产应用中，往往需要对重要数据进行归档备份
归档命令：

```
tar 选项 归档目的地 归档的目录名称
```

后面可以加上

- -c选项，表示归档
- -z选项表示同时进行压缩，得到的文件后缀名为.tar.gz

实操

实现一个每天对指定目录归档备份的脚本，输入一个目录名称(末尾不带/)，将目录下所有文件按天归档保存，并将归档日期附加在归档文件名上，放在/root/archive下

```
#!/bin/bash

#判断参数是否是一个
if [ $# -ne 1 ]
then
  echo "参数个数错误，只能输入一个归档的目录名"
  exit
fi

#获取目录名称
if [ -d $1 ]
then
  echo #什么也不输出表示空一行
else
  echo
  echo "目录不存在"
  echo
  exit
fi

DIR_NAME=$(basename $1)
DIR_PATH=$(cd $(dirname $1); pwd)
```

```
#获取日期
DATE=$(date +%y%m%d)

#定义生成的归档文件名称
FILE=archive_${DIR_NAME}_$DATE.tar.gz
DEST=/root/archive$FILE

#归档文件
tar -czf $DEST $DIR_PATH/$DIR_NAME    #c：归档，z：压缩，f：可视化

if [ $? -eq 0 ]
then
  echo
  echo "归档成功"
  echo
else
  echo "归档失败"
  echo
fi

exit
```

退出脚本后在控制台输入

```
crontab -e
```

进行编辑定时任务

```
0 2 * * * /root/.../daily_archive.sh(执行哪个脚本) /root/...(脚本输入的参数，归档的哪个目录)
```

参数分别指的是：分钟数 小时数 天数 月数 星期几，*表示任何一天，0 2 * * *表示任何一天的凌晨两点钟进行后面的命令

```
crontab -l
```

查看当前系统里定义的定时任务

正则表达式

正则表达式使用单个字符串来描述、匹配一系列符合某个语法规则的字符串，也就是文本编辑器里面的用来检索、替换符合某个模式文本的操作，如Linux中的grep、sed、awk

常规匹配

匹配所有包含aa的行

```
cat 路径名/文件名 | grep aa
```

常用特殊字符

- ^
  匹配一行的开头，^a：匹配开头为a的所有行
- $
  匹配一行的结束，a$：匹配以a结尾的所有行
  ^$匹配的是所有空行，加-n可以显示空行的行号
- .
  匹配一个任意的字符，r..t：匹配所有r和t中间包含两个字符的所有字符串
- *
  不单独使用，它和上一个字符连用，表示匹配上一个字符0次或多次
  .表示任意字符出现任意多次，即模糊匹配
- []
  匹配某个范围内的一个字符
  - [6,8]：匹配6或8
  - [1-9]：匹配一个1到9的数字
  - [0-9]*：匹配任意长度数字字符串
  - [a-z]：匹配一个a到z的字符
  - [a-c,j-l]：匹配一个a-c或j-l的字符
- \
  \表示转义，不会单独使用，如找到一个文件里所有的$符号，需要输入 grep '$' 来找到所有的$符号，必须使用单引号引出来

实操

筛选手机号

```
grep ^1[34578][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]$
部分文本编辑器支持[0-9]{9}，grep需要加选项-E
grep -E ^1[34578][0-9]{9}$
```

文本处理工具

cut

从文件的每一行剪切字节、字符和字段并将这些字节、字符、字段输出

基本用法

```
cut [选项参数] filename
```

默认分隔符是制表符

选项参数

- -f：列号，提取第几列
- -d：分隔符，按照指定的分隔符分割列，默认是'\t'，第一个分隔符前面的就是第一列，第二个分隔符前面的就是第二列，以此类推
- -c：按字符进行切割，后面加-n表示切割字符的第几列

实操

截取文件名.txt的第一、二列，分隔符为一个空格，如果要截取多列的话可以加一个-，如6-表示第六列开始一直到最后，-4表示最开头到第四列

```
cut -d " " -f 1,2 文件名.txt
```

awk

把文件逐行读入，把空格作为分隔符将每行切片，切开的部分再进行分析处理

基本用法

```
awk [选项参数] '/pattern1/{action1} /pattern2/{action2}...' filename
```

- pattern：表示awk在数据中查找的内容，就是匹配模式，一个正则表达式
- action：在找到匹配内容时执行的一系列命令，只有匹配了pattern，action才会执行，pattern可以为空，action直接执行

选项参数

- -F：指定输入文件分隔符
- -v：赋值一个用户定义变量

awk内置变量

- FILENAME：文件名
- NR：已读的记录数(行号)
- NF：浏览记录的域的个数(切割后，列的个数)

实操

搜索passwd文件以root关键字开头的所有行，并输出该行的第七列

```
awk -F ":" '/^root/{print $7}' passwd
```

搜索passwd文件以root关键字开头的所有行，并输出该行的第一列和第七列，以逗号分隔

```
awk -F ":" '/^root/{print $1","$7}' passwd
```

输出passwd文件的第一列和第七列，以逗号分隔，且在所有行前面添加列名"user,shell"，在最后一行添加"root,/bin/fuck"

```
awk -F ":" 'BEGIN{print "user,shell"} {print $1","$7} END{print
"root,/bin/fuck"}' passwd
```

BEGIN在所有数据读取行之前执行，END在所有数据执行后执行

将passwd文件中用户id增加1并输出

```
awk -v i=1 -F ":" '{print $3+i}' passwd
```

统计passwd文件名，每行的行号，每行的列数

```
awk -F ":" '{print "文件名：" FILENAME ",行号：" NR ",列数：" NF}' passwd
```

查询ifconfig命令输出结果中的空行所在的行号

```
ifconfig | awk '/^$/{print NR}'
```

切割IP

```
ifconfig ens33 | awk '/netmask/{print $2}'
```

在awk中，如果一行的前面都是空格，默认不考虑，从非空格字符的后一个空格开始考虑列