# shell基础知识

## 1.1echo 和printf

### 1.1.1echo

**1)echo取消自动换行**

默认echo xxx 是会自动换行的,通过参数 -n 能取消换行

echo -n

**2)echo转义字符**

echo默认不认识转义字符,需要加上-e参数

echo -e “\t hahah”

echo -e ‘\t hahah’

**3)echo单双引号区别**

a=hello

echo ‘$a haha’ #结果:$a haha

echo “$a haha”#结果:hello haha

**4)echo 感叹号的输出**

echo hello! ok

echo ‘hello!’ ok

echo “hello!” error

所以对于各种特殊符号的输出用单引号将符号单独包围住

echo “hello”’!’

### 1.1.2printf

**1)printf换行符**

printf默认不会加换行符,要自己添加\n

**2)printf单双引号的区别和echo一样**

a=hello

print ‘$a’ #输出:$a

printf “$a”输出:hello

print $a 输出:hello 和双引号有点像

**3)printf %和参数个数不一致**

printf “hello%s” 1 2 3

#结果:

hello1hello2hello3

意思就是相当于来了个for循环,并且把输出的结果连起来

**4)printf格式化**

printf “%5s” a #5个字符的宽度,不足的左边补上空格

结果: a

printf “%-5s” a #5个字符的宽度,不足5个字符右变补空格

结果:a

printf “%6.3f” 4.3 #结果:4.300

分析:宽度为6位其中”.”也包含在宽度6中,3表示保留小数点3位,不足3为添0



## 1.2命令执行顺序

**1)分号”;”**

command1;command2

命令之间没有逻辑关系,命令从前往后执行,前面的命令执行错了,也不影响后面命令的执行

**2)&&**

command1&&command2

逻辑与,命令从前往后执行,command1执行成功后,才执行command2命令,command1执行失败,command2不会被执行

判断是否执行成功主要看回传码 echo $?

**3)||**

command1 || command2

逻辑或,命令总前往后执行,当command1执行失败后才会执行command2,当command1执行成功后,不会执行command2

**4)&**

command1 & command2

这个和”;”一样,没有逻辑关系,只是先执行command1后放入后台,再执行command2

## 1.3变量

1.变量的赋值str=hello 注意等号左右没有空格。如果有空格就是进行比较运算符的比较运算了

2.变量的引用:$str或者${str}

3.删除变量:unset str

4.查看所有变量set或declare不加参数

5.定义只读变量readonly str=hello,注意str将不能被修改或者unset删除了,只能退出脚本,或者退出shell

6.环境变量:export str可以临时将普通变量变为环境变量,永久的环境变量要在/etc/profile里定义

7.获取变量的长度str=hello;echo ${#str}

8.变量内容的删除和替换:

删除和替换,原来的str不会变化

**从左向右匹配,**

str=hello,world,hello,linux,linux

删除最短的: echo ${str#\*hello,} 格式 ${变量名称#被删除的字符串}

结果:~~hello~~,world,hello,linux,linux

删除最长的echo ${str#\*hello,}

结果:~~hello,world,hello,~~linux,linux

**从右往左匹配(实际上只是从右边看起,还是从左往右):**

删除最短的: echo ${str%,linux\*}

hello,world,hello,linux~~,linux~~

删除最长的:echo ${str%%,linux\*}

hello,world,hello~~,linux,linux~~

替换一个字符串:echo ${str/hello/lw} 格式: ${变量/旧字符串/新字符串}

lw,world,hello,linux,linux

替换所有字符串:echo ${str//hello/lw} 格式: ${变量//旧字符串/新字符串}

lw,world,lw,linux,linux

|  |  |
| --- | --- |
| ${变量#关键词} ${变量##关键词} | 若变量内容从头开始的数据符合『关键词』，则将符合的最短数据删除 若变量内容从头开始的数据符合『关键词』，则将符合的最长数据删除 |
| ${变量%关键词} ${变量%%关键词} | 若变量内容从尾向前的数据符合『关键词』，则将符合的最短数据删除 若变量内容从尾向前的数据符合『关键词』，则将符合的最长数据删除 |
| ${变量/旧字符串/新字符串} ${变量//旧字符串/新字符串} | 若变量内容符合『旧字符串』则『第一个旧字符串会被新字符串取代』 若变量内容符合『旧字符串』则『全部的旧字符串会被新字符串取代』 |

9.变量的赋值

echo ${str-hello}:如果str没有被定义,输出hello,str不变

echo ${str=hello}:”=”和”-”很像,输出hello,str也变为hello了

echo ${str:-hello}:如果str没有被定义或者字符串为空,则输出hello,str不变

echo ${str:=hello}:”:=”和”:-”很像,输出hello,str也变为hello了

echo ${str+hello}:”+”和”-”相反,即如果字符串定义了,则输出hello,str不变

echo ${str:+hello}:”:+”和”:-”相反即如果字符串不为空,则输出hello,str不变

10.取子字符串

格式:${str:offset:length}

str=hello;

echo ${str:0} #hello

echo ${str:1} #ello

echo ${str:2:2} #ll

11.特殊变量

$$ 当前SHELL的PID

$# 参数的个数,如果是脚本传参,则不算脚本名称,如果是函数,则不算函数名称

$\* 参数的整体,你可以想象\*就代表所有,整体的意思比如你传入了 a b c d:在for循环时写

for i in “$\*”就是 for i in “a b c d”其实只循环了一次

$@ 所有单个参数,比如传入了a b c d四个参数,for i in “$@”会循环4次,其实就相当于

for i in “a” “b” “c” “d”

$0 脚本名,注意可能是函数名

$1..$n 参数位置

## 1.4算术运算

let i=i-1 其他:let i=$i-1, let i-=1,

i=$((i-1)) 其他:i=$(($i-1))

i=$[ i - 1 ] 其他:i=$[ $i - 1 ]

echo $((i=i-1)) 其他:echo $((i=$i-1))

bc命令:其实如果只是简单的加减乘除上面的方法够用了,如果要用到高级的运算,那只能用bc了,例如如何开平方等等

bc方法1:

echo "scale=3;r=3;5/r" | bc #注意scale只对除法、取余、乘幂有效,缺点,如果计算量大了,则写在一行太挤了

bc方法2:

[root@localhost ~]# bc<<EOF

> r=1.3

> scale=3

> r/2

> EOF

.650

[root@localhost ~]#

## 1.5重定向

Stdin:0

Stdout:1

Stderr:2

**1).将stdout和stderr都重定向同一个文件中**

a. find / -name ls > a.txt 2>&1

b. find / -name ls &> a.txt(重点,因为好理解,也简单) 注意&>>和&>很像,&>>是追加

cat 01.txt 是将文件的内容输出到标准输出(默认)

如何输出标准输入到屏幕中呢？

cat << EOF

hello,worl

EOF

这样就可以了

如何输出标准输入到01.txt中呢？

cat << EOF > 01.txt

hello,world

EOF

或者 cat >01.txt << EOF

haha

EOF

**2)双重定向tee**

tee:是可以将文件首先重定向到文件中,再将内容输出到标准输出

-:表示标准输入

参数: -a 是追加到文件中

cat a.txt | tee b.log|cat –

cat << EOF | tee 01.txt

hello,world

EOF

hello,world

## 1.6数组

定义数组:

1. arr=(1 2 3) 注意:不能加上逗号,加上”()”表示数组

echo ${arr[0]} #1

echo ${arr[1]} #2

echo ${arr[2]} #3

错误写法: arr=(1,2,3);这样也是个数组只是只有一个元素 echo ${arr[0]} #1,2,3

1. arr[0]=1;arr[1]=1;arr[2]=2;

数组的长度:

echo ${#arr[\*]}或者 ${#arr[@]}

关联数组

declare -A arr #先声明一个关联数组

arr=([name]=lw [age]=14)

echo ${arr[name]} #lw

arr[name]=bob #修改

echo ${arr[\*]}或者${arr[@]} 查看所有

echo ${!arr[@]}或者${!arr[\*]} 查看索引列表结果为lw 14

## 1.7shell命令的替换

Linux中使用反引号``(等价于$())来执行命令替换,在``或者$()里的命令提前于整行命令运行

,运行完将执行的结果插入在命令替换符号处,此时输出的结果含有>=1个空白字符,则压缩为1个空白字符,并且会忽略结尾的换行符\n

命令替换分为两个过程：(1)开启子shell执行其中的命令(2)将子shell中的输出结果打包插入在命令行中。但打包输出结果的过程是可以控制的(例如上面使用双引号)。

Ex1: 包含多个空格:

|  |
| --- |
| root@localhost im]# echo 你好`echo -e "\thello"`  你好 hello  [root@localhost im]# |

Ex2:不包含空格

|  |
| --- |
| [root@localhost im]# echo 你好`echo "hello"`  你好hello  [root@localhost im]# |

Ex3:使用双引号不压缩空格

|  |
| --- |
| [root@localhost im]# echo 你好"`echo -e "\n\thello"`"  你好  Hello  [root@localhost im]# |

Ex4:忽略结尾的换行符(不管加不加””都会忽略的)

|  |
| --- |
| [root@localhost im]# echo $(echo -e "haha\n")你好  haha你好  [root@localhost im]# |

## 1.8$后面接单引号或者双引号

$”hah\n” 和 ”hah\n”没撒区别,但是$’hah\n’和’hah\n’是有区别的,对于单引号来讲,单引号里面的东西全都是字面值,但是在前面加上了$后就会将一些\n,\t等特殊字符以特殊字符对待,不会再视为字面值了

|  |
| --- |
| [root@localhost ~]# echo 'hah\n'  hah\n  [root@localhost ~]# echo $'hah\n'  hah  [root@localhost ~]# |

## 1.9test,[],[[]]区别

简单来讲test 和 []是一样的, [[]]和[]很像,他支持额外的正则表达式

Bash里面:true 就是0,false的就是1

|  |  |
| --- | --- |
| **1.文件类检测，例如[ -e /tmp/a.log ]。如非特别说明，则所有文件类操作都会追踪到软链接的源文件。** | |
| -e file | 文件是否存在(exist) |
| -f file | 文件是否存在且为普通文件(file) |
| -d file | 文件是否存在且为目录(directory) |
| -b file | 文件是否存在且为块设备block device |
| -c file | 文件是否存在且为字符设备character device |
| -S file | 文件是否存在且为套接字文件Socket |
| -p file | 文件是否存在且为命名管道文件FIFO(pipe) |
| -L file | 文件是否存在且是一个链接文件(Link) |

|  |  |
| --- | --- |
| **2. 文件属性检测，如test -r /tmp/a.log。如非特别说明，则所有文件类操作都会追踪到软链接的源文件。** | |
| -r file | 文件是否存在且当前用户可读 |
| -w file | 文件是否存在且当前用户可写 |
| -x file | 文件是否存在且当前用户可执行 |
| -u file | 文件是否存在且设置了SUID |
| -g file | 文件是否存在且设置了SGID |
| -k file | 文件是否存在且设置了sbit(sticky bit) |
| -s file | 文件是否存在且大小大于0字节，即用于检测文件是否为非空白文件 |
| -N file | 文件是否存在，且自上次read后是否被modify |
| **3. 两个文件之间的比较，如：test file1 -nt file2** | |
| file1 -nt file2 | (newer than)判断file1是否比file2新 |
| file1 -ot file2 | (older than)判断file1是否比file2旧 |
| file1 -ef file2 | (equal file)判断file2与file2是否为同一文件，可用在判断hard link的判定上。主要意义在判定，两个文件是否均指向同一个分区上的同一个inode |
| **4. 两个整数之间的判定，支持正负数，但不支持小数。例如test n1 -eq n2** | |
| int1 -eq int2 | 两数值相等(equal) |
| int1 -ne int2 | 两数值不等(not equal) |
| int1 -gt int2 | n1大于n2(greater than) |
| int1 -lt int2 | n1小于n2(less than) |
| int1 -ge int2 | n1大于等于n2(greater than or equal) |
| int1 -le int2 | n1小于等于n2(less than or equal) |
| **5. 判定字符串** | |
| -z string | (zero)判定字符串是否为空？若string为空字符串，则为true |
| string  -n string | 判定字符串是否非空？若string为空字符串，则false。注：-n可省略 |
| string1 = string2  string1 == string2 | string1和string2是否相同。相同则返回true。"=="和"="等价，但"="可移植性更好 |
| str1 != str2 | str1是否不等于str2，若不等，则返回true |
| str1 > str2 | str1字母顺序是否大于str2，若大于，则返回true |
| str1 < str2 | str1字母顺序是否小于str2，若小于，则返回true |
| **6.逻辑运算符，例如：test -r filename -a -x filename** | |
| -a或&& | (and)两表达式同时为true时才为true。"-a"只能在test或[]中使用，&&只能在[[]]中使用 |
| -o或|| | (or)两表达式任何一个true则为true。"-o"只能在test或[]中使用，||只能在[[]]中使用 |
| ! | 对表达式取反 |
| ( ) | 用于改变表达式的优先级，**为了防止被shell解析，应该加上反斜线转义\( \)** |

[[]]的用法:

1)当表达式为==或者!=时,不支持正则表达式,但能使用通配符"\*"、"?"和"[...]"

[[ abc == a\* ]];echo $? #0

2)当表达式为=~时可以使用正则表达式

[[ abcd =~ a..d$ ]];echo $? #0

注意事项:

1. 无论使用[]还是[[]]时,对于变量的引用最好加上双引号

name=”Ma long”;[ $name = “Ma long” ]会报错,因为会被解读成

[ Ma long = "Ma long" ],这时要使用双引号包围

[ “$name” = “Ma long” ]

1. 数值比较-eq时,bash会将-eq两边非数值的字符串转换为数值

[ “1” -eq 1 ] #ok, “1” 转换为了1

[ “1a” -eq 1 ] #这里会报错,不时返回1, “1a” 不能转换为1

空字符串和引用为定义的变量都时会报错的

为了防止报错的发生,还是采用 = 比较字符串,而不是-eq比较数值

1. []和[[]]中都会有空格的

## 2.0 IFS

IFS时bash 内部字段分隔符的环境变量

IFS默认会忽略字符串开头和结尾的空格,并且对中间的空格,制表符\t,换行符\n进行分割,多个空格合并为一个空格

有时候修改IFS会很有作用:

|  |
| --- |
| names='1,2,3,4'  oldIFS=$IFS  IFS=$','  for name in $names;  do  echo $name  done  IFS=$oldIFS |

## 2.1函数和循环

1. 函数:相当于命令

优先级:别名 > 函数 > 命令

移除函数:unset -f 函数名

函数只在当前shell中生效,如果要想子进程也继承,则export -f 函数名

函数可以递归,无限层级

函数返回的状态码就时函数体内最后一条命令产生的状态码

根据函数名称查看该函数的定义:typeset -f 函数名称

定义函数:

function f()

{

echo haha;

}

调用函数:f arg1 arg2

1. If结构

a=$1

if [ $a = 1 ];then

echo 1;

elif [ $a = 2 ];then

echo 2;

else

echo other;

fi

1. for循环

结构1:

for i in 1 2 3 4;

do

echo $i;

done

结构2:

for((i=0;i<=3;i++));

do

echo $i;

done

1. while循环

条件返回0执行循环体,否则退出循环

while 条件;

do

done

死循环:

while true;

do

done

或者

while :;

do

done

特殊的写法:

1)使用管道传递内容

echo “a b” |while read x1 x2;

do

echo $x1 $x2

done

2)<<<传递内容

while read line

do

echo $line

done <<< "abc xyz" #<<< 表示后面的字符串作为输入的数据

3)从文件读取内容

while read line

do

echo $line

done </etc/passwd

1. case选择

case "$1" in

start)

echo start;;

stop)

echo stop;;

\*)

echo other;;

esac

注意事项:

1)在管道的写法中在循环体定义的变量的循环体外时不生效的

echo “a b” |while read x1 x2;

do

a=100;

done

echo $a #a是未定义的,其他任何方式,a都是定义了的

2) exit、break、continue和return

Exit n:退出shell,退出状态码

Break n:退出循环包括for while until select n表示退出的循环层次

Continue n:退出当前循环,进入下一个循环

Return n:退出整个函数,n表示函数的退出状态码