Shell脚本

#!/bin/bash

echo "Hello World !"

#!是一个约定的标记，它告诉系统这个脚本需要什么解释器来执行，即使用哪一种Shell

echo命令用于向窗口输出文本

运行Shell脚本

1.将代码保存为test.sh，并cd到相应目录

chmod +x ./test.sh #使脚本具有执行权限

./test.sh #执行脚本

//如果直接写test.sh，linux系统会去PATH里寻找有没有test.sh，而只有/bin，/sbin，/usr/sbin等在PATH中，当前目录通常不在PATH里。

2.作为解释器参数

直接运行解释器，参数就是shell脚本的文件名

/bin/sh test.sh

或

/bin/php test.php

这种方式运行的脚本，不需要再第一行指定解释器信息，写了也没用

Shell变量

定义变量时，变量名不加$（$，PHP语言中变量需要），如：

your\_name="runoob.com"

注：变量名和等号之间不能有空格

变量名命名规则：

1.只能用英文字母，数字和下划线，首个字母不能以数字开头

2.中间不能有空格，可以使用\_

3.不能使用标点符号

4.不能使用bash里的关键字

除了显示地直接赋值，还可以用语句给变量赋值

for file in 'ls /etc'

或

for file in $(ls /etc)

//将/etc下目录的文件名循环出来

使用一个定义过的变量，只要再变量名前面加$

your\_name="qinjx"

echo $your\_name

echo ${your\_name}

变量外面的花括号是为了帮助解释器识别变量的边界

for skill in Ada Coffe Action Java; do

echo "I am good at ${skill}Script"

done

如果不加，解释器会把$skillScript当成变量（其值为空）

对变量再次赋值时变量名前不加$

readonly命令可以将变量定义为只读变量，只读变量的值不能被改变

readonly variable\_name

unset命令删除变量

unset variable\_name

变量删除后不能再次使用。unset命令不能删除只读变量

变量类型：

1.局部变量。局部变量在脚本或命令中定义，仅在当前shell实例中有效，其他shell启动的程序不能访问局部变量

2.环境变量。所有的程序，包括shell启动的程序，都能访问环境变量，有些程序需要环境变量来保证其正常运行。必要的时候shell脚本也可以定义环境变量

3.shell变量。shell变量是由Shell程序设置的特殊变量。shell变量中有一部分是环境变量，有一部分是局部变量，这些变量保证了shell的正常运行

Shell字符串

字符串可以用单引号，双引号，也可以不用引号

1.单引号

str='this is a string'

单引号里任何字符原样输出，单引号字符串中的变量是无效的

单引号字串中不能出现单独一个单引号（对单引号使用转义符后也不行），但可成对出现，作为字符串拼接使用。

2.双引号

your\_name='runoob'

str="Hello, I know you are \"$your\_name\"! \n"

echo -e $str

双引号里可以有变量，可以出现转义字符 \"

3.拼接字符串

your\_name="runoob"

# 使用双引号拼接

greeting="hello, "$your\_name" !"

greeting\_1="hello, ${your\_name} !"

echo $greeting $greeting\_1

# 使用单引号拼接

greeting\_2='hello, '$your\_name' !'

greeting\_3='hello, ${your\_name} !'

echo $greeting\_2 $greeting\_3

输出结果为：

hello, runoob ! hello, runoob !

hello, runoob ! hello, ${your\_name} !

name="Shell"

url="http://c.biancheng.net/shell/"

str1=$name$url #中间不能有空格

str2="$name $url" #如果被双引号包围，那么中间可以有空格

str3=$name": "$url #中间可以出现别的字符串

str4="$name: $url" #这样写也可以

str5="${name}Script: ${url}index.html" #这个时候需要给变量名加上大括号

获取字符串长度

string="abcd"

echo ${#string}

提取子字符串

string="runoob is a great site"

echo ${string:1:4}#输出unoo

查找子字符

查找字符i或o是第几个字符（哪个先出现就计算哪个）

string="runoob is a great site"

echo `expr index "$string" io` #输出4

//`是反引号

Shell数组

bash支持一维数组（无多维数组），且没有限制数组的大小

数组元素下标由0开始编号。获取数组中的元素要用下标，下标可以是整数或者算术表达式，值应该>=0

定义数组

数组名=(值1 值2 ... 值n)

也可以单独定义数组的各分量

array\_name[0]=value0

array\_name[1]=value1

可以不使用连续的下标，而且下标的范围没有限制

读取数组

${数组名[下标]}

如：valuen=${array\_name[n]}

使用@可以获取数组中所有元素

如：echo ${array\_name[@]}

获取数组的长度

length=${#array\_name[@]}

或

length=${#array\_name[\*]}

lengthn=${#array\_name[n]} #获取数组单个元素的长度

Shell注释

#开头的行就是注释

多行注释：

:<<EOF :<<!

......... .........

......... .........

......... .........

EOF !

Shell传递参数

可以在执行Shell脚本时向脚本传递参数。脚本内获取参数的格式为：$n。n代表一个数字，1为执行脚本的第一个参数，2为第二个参数.......

echo "Shell 传递参数实例！";

echo "执行的文件名：$0";

echo "第一个参数为：$1";

echo "第二个参数为：$2";

echo "第三个参数为：$3";

//$0为执行的文件名（包含文件路径）

在运行时：

chmod +x test.sh

./test.sh 1 2 3

处理参数的特殊字符

$# 参数个数

$\* 以一个单字符串显示所有向脚本传递的参数。（视为一个参数："$1 $2 ... $n"）

$$ 脚本运行的当前进程ID号

$! 后台运行的最后一个进程ID号

$@ 与$\*相同，但是视为n个参数，"$1""$2"..."$n"

$- 显示Shell使用的当前选项，与set命令功能相同

$? 显示最后命令的退出状态。0表示没有错误，其他任何值表示有错误

区分$\*和$@

脚本：

echo "-- \$\* 演示 ---"

for i in "$\*"; do

echo $i

done

echo "-- \$@ 演示 ---"

for i in "$@"; do

echo $i

done

输出：

-- $\* 演示 ---

1 2 3

-- $@ 演示 ---

1

2

3

Shell基本运算符

原生bash不支持简单的数学运算，但是可以使用其他命令来实现，如awk和expr，expr常用。

val=`expr 2 + 2`

注：

1.表达式和运算符中间要有空格

2.完整的表达式要被``包含

算数运算符

+ - \* / % :`expr $a + $b`//注意乘号前面必须加反斜杠\ ：val=`expr $a \\* $b`

= : a=$b

== != : [ $a == $b ] [ $a != $b ]

关系运算符

只支持数字，不支持字符串，除非字符串的值是数字

[ $a -eq $b ] 检查两个数是否相等

[ $a -ne $b ] 检查两个数是否不相等

[ $a -gt $b ] a>b?

[ $a -lt $b ] a<b?

[ $a -ge $b ] a>=b?

[ $a -le $b ] a<=b?

布尔运算符

!:非运算 [ ! false ] //返回true

-o:或运算 [ $a -lt 20 -o $b -gt 100 ]

-a:与运算 [ $a -lt 20 -a $b -gt 100 ]

逻辑运算符

&& 逻辑的AND [[ $a -lt 100 && $b -gt 100 ]]

|| 逻辑的OR [[ $a -lt 100 || $b -gt 100 ]]

字符串运算符

= 检测两个字符串是否相等 [ $a = $b ]

!= 检测两个字符串是否不相等 [ $a != $b ]

-z 检测字符串长度是否为0 [ -z $a ]

-n 检测字符串长度是否部位0 [-n "$a" ]

$ 检测字符串是否为空 [ $a ]

文件测试运算符

file="/var/www/runoob/test.sh"

[ -b $file ] 是否是块设备文件

[ -c $file ] 是否是字符设备文件

[ -d $file ] 是否是目录

[ -f $file ] 是否是普通文件（既不是目录，也不是设备文件）

[ -g $file ] 文件是否设置了SGID位

[ -k $file ] 文件是否设置了粘着位（Sticky Bit）

[ -p $file ] 文件是否是有名管道

[ -u $file ] 文件是否设置了SUID位

[ -r $file ] 文件是否可读

[-w $file ] 文件是否可写

[-x $file ] 文件是否可执行

[-s $file ] 文件是否为空（大小是否>0）不为空返回true

[-e $file ] 检测文件（包括目录）是否存在

-S 判断某文件是否socket

-L 检测文件是否存在并且是一个符号链接

Shell echo命令

Shell的echo和PHP的echo类似，都是用于字符串的输出

1.显示普通字符串

echo "It is a test"//输出It is a test ""可以省略

2.显示转义字符

echo "\"It is a test\""//输出"It is a test" ""可以省略

3.显示变量

read name//read命令从标准输入中读取一行，并把输入行的每个字段的值指定给shell变量

#!/bin/sh

read name

echo "$name It is a test"

4.显示换行

echo -e ""OK! \n" # -e 开启转义

echo "It is a test"

输出结果：

OK!

It is a test

5.显示不换行

echo -e "OK! \c" # -e 开启转义 \c 不换行

echo "It is a test"

输出结果：

OK! It is a test

6.显示结果定向至文件

echo "It is a test" > myfile

7.原样输出字符串，不进行转义或去变量（用单引号）

echo '$name\"'

输出结果：

$name\"

8.显示命令执行结果

echo `date` #结果将显示当前日期

Shell printf命令

默认printf不会像echo那样自动添加换行符，可以手动添加\n

语法：

printf format-string [arguments...]

//format-string 格式控制字符 arguments 参数列表

printf "Hello,Shell\n"

printf "%-10s %-8s %-4.2f\n" 郭靖 男 66.1234

输出结果为:

郭靖 男 66.12

# 格式只指定了一个参数，但多出的参数仍然会按照该格式输出，format-string 被重用

printf %s abc def

printf "%s\n" abc def

printf "%s %s %s\n" a b c d e f g h i j

运行结果：

abcdefabc

def

a b c

d e f

g h i

j

# 如果没有 arguments，那么 %s 用NULL代替，%d 用 0 代替

printf "%s and %d \n"

运行结果：

0

printf的转义序列

\a 警告字符，通常为ASCII的BEL字符

\b 后退

\c 抑制（不显示）输出结果中任何结尾的换行字符（只在%b格式指示符控制下的参数字符串中有效），而且，任何留在参数里的字符、任何接下来的参数以及任何留在格式字符串中的字符，都被忽略

\f 换页

\n 换行

\r 回车

\t 水平制表符

\v 垂直制表符

\\ \

\ddd 1到3位的八进制值的字符。仅在格式字符中有效

\0ddd 1到3位的八进制值字符

实例：

$ printf "a string, no processing:<%s>\n" "A\nB"

a string, no processing:<A\nB>

$ printf "a string, no processing:<%b>\n" "A\nB"

a string, no processing:<A

B>

$ printf "www.runoob.com \a"

www.runoob.com $ #不换行

Shell test命令

用于检查某个条件是否成立，它可以进行数值、字符和文件三个方面的测试

1.数值

-eq:等于

-ne:不等于

-gt:>

-ge:>=

-lt:<

-le:<=

if test $[num1] -eq $[num2]

then

echo '两个数相等！'

else

echo '两个数不相等！'

fi

代码中[]执行基本的算术运算

result=$[a+b] #注意等号两边不能有空格

2.字符串

=

!=

-z 字符串:长度为0

-n 字符串:长度不为0

if test $num1 = $num2

3.文件

-e 文件名:文件存在

-r 文件名:文件存在且可读

-w 文件名:文件存在且可写

-x 文件名:文件存在且可执行

-s 文件名:文件存在且至少有一个字符

-d 文件名:文件存在且为目录

-f 文件名:文件存在且为普通文件

-c 文件名:文件存在且为字符型特殊文件

-b 文件名:文件存在且为快=块特殊文件

if test -e ./bash

Shell还提供了与或非：-a -o !

其中优先级：!>-a>-o

Shell流程控制

流程控制不可为空。即else分支中如果没有语句执行，就不要写这个else

语法格式

if condition

then

...

fi

//写成1行(适用于终端命令):if [ $(ps -ef | grep -c "ssh") -gt 1 ]; then echo "true"; fi

if condition

then

...

else

...

fi

if condition1

then

...

elif condition2

then

...

else

...

fi

for循环

for var in item1 item2 ... itemN

do

...

done

//写成一行:for var in item1 item2 ... item N; do command1; ... done;

for loop in 1 2 3 4 5

do

ehco $loop

done

输出结果

1

2

3

4

5

for str in This is a string

do

echo $str

done

输出结果

This

is

a

string

while语句

while condition

do

command

done

int=1

while(( $int<=5 )) #注意双括号

do

echo $int

let "int++"

done

//let用于执行一个或多个表达式，表达式见用,隔开，变量计算中不需要加$。如果表达式中包含空格或其他特殊字符，引起来。

无限循环

while :

do

...

done

或

while true

do

...

done

或

for (( ; ; ))

until循环

执行一系列命令直到条件为true。极少数情况下until优于while。

until condition

do

...

done

until [ ! $a -lt 10 ]

do

echo $a

a=`expr $a + 1`

done

case...esac

与switch...case类似。每个case分支用右圆括号开始，用两个分号表示break，跳出整个case...esac

case $aNum in

1) echo '你选择了1'

;;

2) echo '你选择了2'

;;

\*) echo '你没有输入1或2'

;;

esac

//右圆括号前面也可以是""字符串来完成字符串匹配

跳出循环

break命令

跳出所有循环（终止执行后面的所有循环）

continue命令

跳出当前循环

Shell函数

函数定义格式：//[]表示内容可省略

[ functuon ] funname [()]

{

action;

[return int;]

}

注：

1.可以带function fun()定义，也可以直接fun()定义，不带任何参数。

2.参数返回，可以显示加：return返回，如果不加，将最后一条命令运行结果，作为返回值。return后跟数值n(0-255

函数返回值在调用该函数后通过$?来获得。

funWithReturn

echo "输入两个数字之和为 $? !"

所有函数在使用前必须定义。这意味着必须将函数放在脚本开始的位置，直至shell解释器首次发现它时，才可以使用。调用函数仅使用其函数名即可。

函数参数

函数体内部通过$n(n为1 2 3 4...)来获取参数。

调用函数时

funWithParam 1 2 3 4 5 6 7

当n>=10时，需要用${n}

Shell文件包含

. filename

或

source filename

//可以看作直接将filename copy到当前脚本中

Shell输入/输出重定向

输出重定向

command >file 覆盖的方式把正确结果输出到file

command >>file 追加的方式把正确结果输出到file

command 2>file 覆盖的方式把错误信息输出到file

command 2>>file 追加的方式把错误信息输出到file

command >file 2>&1 覆盖的方式把正确结果和错误信息输出到file中

command >>file 2>&1 追加的方式把正确结果和错误信息输出到file中

command >file1 2>file2 覆盖的方式，正确结果输出到file1中，错误信息输出到file2中

command >>file1 2>>file2 追加的方式，正确结果输出到file1中，错误信息输出到file2中

输入重定向

command <file 将file内容作为command的输入

command <<END 从标准输入中读取数据，直至遇见分界符END才停止（分界符可以是任意的字符串）

command <file1 >file2 将file1作为command的输入，并将command的处理结果输出到file2

wc 选项 文件名

其中 -c统计字节数 -w统计单词数 -l统计行数

wc -l < users #输入重定向到users文件中

wc -l <<END #直到读取到END结束，而不是使用Ctrl+D

/dev/null文件

如果既不想把命令输出保存到文件中，也不想显示到屏幕上。可以将结果重定向到/dev/null中（可以当作Linux系统的垃圾箱，任何被放入垃圾箱的数据都会被丢弃，无法恢复）

ls -l &>/dev/null