



Shell 脚本编程

Shell 变量

1

主要内容和学习要求

- q shell 变量的设置、查看和清除
- q 局部变量与作用域
- q 环境变量及其设置
- q 内置命令 `declare` 和 `printf`
- q 变量测试与赋值
- q 位置变量与变量的间接引用
- q 命令替换的两种方式
- q 整型变量的算术运算和算术扩展
- q 数组变量及其引用方法

2

变量

q 变量命名

- | 变量名必须以字母或下划线开头，后面可以跟字母、数字或下划线。任何其它字符都标志变量名的结束。
- | 变量名关于大小写敏感。

q 变量类型:

- | 根据变量的作用域，变量可以分为局部变量和环境变量
- | 局部变量只在创建它们的 shell 中可用。而环境变量则在所有用户进程中可用，通常也称为全局变量。

q 变量赋值: `variable=value`

- | 等号两边不能有空格
- | 如果要给变量赋值，可以在等号后面跟一个换行符

3

变量

q 显示变量的值

```
echo $variable 或 echo ${variable}
```

q 清除变量

```
unset variable
```

q 显示所有变量

```
set
```

例:

```
myname=jypan
echo $myname
unset myname
echo $myname
```

4

变量举例

```
[jypan@qtm213 ~]$ round=world
[jypan@qtm213 ~]$ echo $round

[jypan@qtm213 ~]$ name=Peter Piper
[jypan@qtm213 ~]$ echo $name

[jypan@qtm213 ~]$ name="Peter Piper"
[jypan@qtm213 ~]$ echo $name

[jypan@qtm213 ~]$ x=
[jypan@qtm213 ~]$ echo $x

[jypan@qtm213 ~]$ file.bak="$HOME/junk"
```

5

局部变量和作用域

q 变量的作用域

是指变量在一个程序中那些地方可见。对于 shell 来说，局部变量的作用域限定在创建它们的 shell 中。

例:

```
[jypan@qtm213 ~]$ echo $$ → $ 是特殊变量，用来存储当前运行进程的PID
5817
[jypan@qtm213 ~]$ round=world
[jypan@qtm213 ~]$ echo $round
```

```
[jypan@qtm213 ~]# bash
[jypan@qtm213 ~]$ echo $$
5751
[jypan@qtm213 ~]$ echo $round

[jypan@qtm213 ~]$ exit
```

6

只读变量

q 只读变量

是指不能被清除或重新赋值的变量。

```
readonly variable
```

例:

```
[jypan@qtm213 ~]$ myname=jypan
[jypan@qtm213 ~]$ readonly myname
[jypan@qtm213 ~]$ unset myname
bash: unset: myname: cannot unset a readonly variable

[jypan@qtm213 ~]$ myname="jypan"
bash: myname: readonly variable
```

7

环境变量

q 环境变量

- | 作用域包含创建它们的 shell，以及从该 shell 产生的任意子 shell 或进程。
- | 按照惯例，环境变量通常使用大写。
- | 环境变量是已经用 `export` 内置命令输出的变量。

q 变量被创建时所处的 shell 称为父 shell。如果在父 shell 中启动一个新的 shell (或进程)，则该 shell (或进程) 被称为子 shell (或子进程)。

- | 环境变量就象 DNA，可以从父亲传递给儿子，再到孙子，但不能从子进程传递给父进程。

8

环境变量举例

q 设置环境变量

```
export variable=value
```

```
variable=value; export variable
```

```
例: [jypan@fish213 ~]$ echo $$
[jypan@fish213 ~]$ export round=world
[jypan@fish213 ~]$ bash
[jypan@fish213 ~]$ echo $$
15175
[jypan@fish213 ~]$ echo $round
```

```
export -n variable 将全局变量转换成局部变量
```

```
export -p 列出所有全局变量
```

9

内置命令 declare

q 内置命令 declare 可用来创建变量。

```
declare [选项] variable=value
```

declare 常用选项

选项	含义
-r	将变量设为只读 (<i>readonly</i>)
-x	将变量输出到子 shell 中 (<i>export</i> 为全局变量)
-i	将变量设为整型 (<i>integer</i>)
-a	将变量设置为一个数组 (<i>array</i>)
-f	列出函数的名字和定义 (<i>function</i>)
-F	只列出函数名

10

declare 举例

```
例: declare myname=jypan

declare -r myname=jypan
unset myname
declare myname="Jianyu Pan"

declare -x myname2=pjy

myname2=pjy
declare -x myname2

declare
```

11

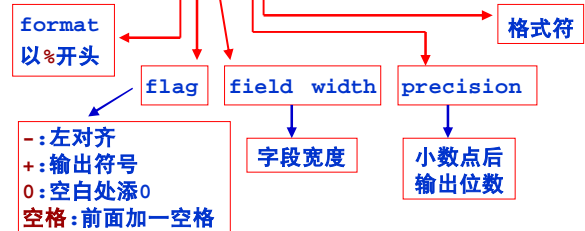
printf 命令

q printf 可用来按指定的格式输出变量

```
printf format 输出参数列表
```

printf 的打印格式与 C 语言中的 printf 相同

```
printf "%-12.5f\n" 123.456
```



12

printf 命令

printf 命令的格式说明符

c	字符型	g/G	浮点数 (自动)
d	十进制整数	o	八进制
e/E	浮点数 (科学计数法)	s	字符串
f	浮点数 (小数形式)	x/X	十六进制

format 中还可以使用

\a	警铃	\t	水平制表符
\b	退后一格	\v	垂直制表符
\n	换行	\\	反斜杠
\f	换页	\"	双引号
\r	回车	%%	百分号

13

printf 命令举例

```
例: printf "The number is: %.2f\n" 100

printf "%-20s|%.2f|\n" "Joy" 10

printf "%-10d%010o%+10x\n" 20 20 20

printf "%6d\t%6o\"%6x\" \n" 20 20 20
```

14

变量测试

q shell 提供一些专用的修饰符来检查某个变量是否已被设置, 然后根据测试结果指定变量的值, 也称变量置换

<code>\${var:-word}</code>	如果 <code>var</code> 存在且非空, 则表达式的值为 <code>\$var</code> ; 如果 <code>var</code> 未定义或为空值, 则表达式的值为 <code>word</code> , 但 <code>var</code> 的值不变。
<code>\${var:=word}</code>	如果 <code>var</code> 存在且非空, 则表达式的值为 <code>\$var</code> ; 如果 <code>var</code> 未定义或为空值, 则表达式的值为 <code>word</code> , 且 <code>var</code> 被赋值 <code>word</code> 。
<code>\${var:?word}</code>	如果 <code>var</code> 存在且非空, 则表达式的值为 <code>\$var</code> ; 如果 <code>var</code> 未定义或为空值, 则输出信息 <code>word</code> , 并终止脚本。
<code>\${var:+word}</code>	如果 <code>var</code> 存在且非空, 则表达式的值为 <code>word</code> ; 否则返回空值, 但 <code>var</code> 的值不变。

15

变量测试举例

```
例: color=blue
newcolor=${color:-grey}

unset color
echo "The sky is ${color:-grey} today"
echo $color

echo "The sky is ${color:=grey} today"
echo $color

echo "The sky is ${color:?error} today"
echo $color

echo "The sky is ${color:+blue} today"
echo $color
```

16

位置参数（命令行参数）

Q 位置参数是一组特殊的内置变量，通常被 shell 脚本用来从命令行接受参数，或被函数用来保存传递给它的参数。

Q 执行 shell 脚本时，用户可以通过命令行向脚本传递信息，跟在脚本名后面的用空格隔开的每个字符串都称为位置参数。

Q 在脚本中使用这些参数时，需通过位置参数来引用。例如：\$1 表示第一个参数，\$2 表示第二个参数，以此类推。\$9 以后需要用花括号把数字括起来，如第 10 个位置参数以 \${10} 的方式来访问。

17

位置参数列表

\$0	当前脚本的文件名
\$1-\$9	第 1 个到第 9 个位置参数
\${10}	第 10 个位置参数，类似地，有 \${11}, ...
\$#	位置参数的个数
\$*	以单字符串显示所有位置参数
@	未加双引号时与 \$* 含义相同，加双引号时有区别
\$\$	脚本运行的当前进程号
\$_	最后一个后台运行的进程的进程号
?	显示前面最后一个命令的退出状态。 0 表示没有错误，其他任何值表示有错误。
-	显示当前 shell 使用的选项

18

位置参数举例

例1: 显示位置参数的值。

```
(shprg1.sh)
```

例2: \$* 与 @\$ 的区别: 二者仅在被双引号括起来时有区别，此时前者将所有位置参数看成一个字符串，而后者将每个位置参数看成单独的字符串。

```
(shprg2.sh)
```

U 如果位置参数中含有空格，则需要使用双引号

```
args2.sh This is "Peter Piper"
```

19

basename

Q basename

返回不含路径的文件名或目录名

```
basename /home/jypan/linux
```

```
basename ~
```

```
shprg3.sh
```

20

变量的间接引用

```
str1="Hello World"
str2=str1
echo $str2
```

! 如何通过 str2 的值来引用 str1 的值? (间接引用)

```
echo $$str2 ?
echo ${$str2} ?
```

```
eval newstr=\$$str2
echo $newstr
```

```
newstr=${!str2} # bash.2.0以上才支持
echo $newstr # echo ${!str2}
```

```
脚本范例: args3.sh Hello world!
```

21

eval

```
eval arg1 [arg2] ... [argN]
```

! 将所有的参数连接成一个表达式，并计算或执行该表达式，参数中的任何变量都将被展开。

```
listpage="ls -l | more"
$listpage
```

```
listpage="ls -l | more"
eval $listpage
```

```
eval newstr=\$$str2
```

22

命令替换

Q 命令替换的用处是将命令的输出结果赋给一个变量，或者用命令的输出结果代入命令所处的位置。

Q 所有的 shell 都支持使用反引号来执行命令替换。

```
echo "The hostname is `hostname`"
```

Q Bash 除了使用反引号来执行命令替换外，还有另外一种替换方法：将命令放在前有美元符号的一对圆括号内。

```
echo "The hostname is $(hostname)"
```

Q 命令替换可以嵌套使用。

如果使用反引号，则内部的反引号必须用反斜杠来转义。

```
echo `basename `pwd``
echo $(basename $(pwd)) #see shprg4.sh
```

23

算术运算

Q Bash 变量是没有严格的类型定义，本质上 Bash 变量都是字符串，但 Bash 也允许定义整型变量，可以参加运算与比较。

Q 可以用 declare 命令定义整型变量。

```
declare -i num
num=1; echo $num
num=$num+3; echo $num #num=num+3
num2=$num+3; echo $num2
```

```
declare -i #列出所有整型变量
```

```
num2=1; echo $num2
num2=$num2+1; echo $num2
```

未被定义为整型的变量不能直接参加算术运算!

24

整数运算

- q `declare` 定义的整型变量可以直接进行算术运算。
- q 未被定义为整型的变量，可用内置命令 `let` 进行算术运算。

```
num2=1; echo $num2
let num2=4+1; echo $num2
let num2=$num2+1; echo $num2
```

- ! 赋值符号和运算符两边不能留空格!
- ! 如果将字符串赋值给一个整型变量时，则变量的值为 0
- ! 如果变量的值是字符串，则进行算术运算时设为 0

```
let num2=4 + 1
let "num2=4 + 1" # 用引号忽略空格的特殊含义
```

用 `let` 命令进行算术运算时，最好加双引号。

25

let 命令运算操作符

let 命令操作符

<code>+</code> , <code>-</code> , <code>*</code> , <code>/</code>	(四则运算)
<code>**</code> , <code>%</code>	(幂运算和模运算, 取余数)
<code><<</code> , <code>>></code>	(按位左移和按位右移)
<code>&</code> , <code>^</code> , <code> </code>	(按位与、按位异或和按位或)
<code>=</code> , <code>+=</code> , <code>--</code> , <code>*=</code> , <code>/=</code> , <code>%=</code>	(赋值运算)
<code><<=</code> , <code>>>=</code> , <code>&=</code> , <code>^=</code> , <code> =</code>	(比较操作符)
<code><</code> , <code>></code> , <code><=</code> , <code>>=</code> , <code>==</code> , <code>!=</code>	(比较操作符)
<code>&&</code> , <code> </code>	(逻辑与和逻辑或)

注: 按位运算是以二进制形式进行的。

```
a=2; let "a<<=2" # 用引号忽略 << 的特殊含义
```

26

浮点数运算

- q `Bash` 只支持整数运算，但可以通过使用 `bc` 和 `awk` 工具来处理更复杂的运算。

```
n=$(echo "scale=3; 13/2" | bc )
echo $n
```

```
m=`awk 'BEGIN{x=2.45;y=3.123; \
printf "%.3f\n", x*y}'`
echo $m
```

27

算术扩展

- q 除了使用 `let` 命令外，`Shell` 可以通过下面两种方式对一个算术表达式进行求值。

```
$(expression)
$((expression))
```

例:

```
num2=$((4 + 1)); echo $num2
num2=$((($num2*2-3)); echo $num2
```

用 `let` 命令和 `$(...)`, `$((...))` 进行整数运算时，美元符号 `$` 可以省略，但最好写上。

注意 `$(...)`, `$((...))`, `$(...)`, `$((...))` 的不同作用

28

数组变量

- q `Bash 2.x` 以上支持一维数组，下标从 0 开始。
- q 数组可以用 `declare` 命令创建，或直接给变量名加下标来创建。

```
declare -a variable
variable=(item1 item2 item2 ... )
```

```
variable=(item1 item2 item2 ... )
variable[n]=value
```

- q 数组的引用

```
${variable[n]}
```

29

数组变量举例

```
declare -a stu
stu=(math1101 math1102 math1103)
echo ${stu[0]} # 列出stu的第一个元素
echo ${stu[*]} # 列出stu的所有元素
echo ${#stu[*]} # 给出数组stu中元素的个数
```

- q 数组与数组元素的删除

```
unset stu[1] # 删除stu的第二个元素
unset stu # 删除整个数组
```

- q 数组赋值时无须按顺序赋值

```
x[3]=100; echo ${x[*]}
state=(ME [3]=CA [2]=NT); echo ${state[*]}
```

30

相关命令小结

```
echo $variable 或 echo ${variable}
```

```
unset variable
```

```
set
```

```
readonly variable
```

```
export variable=value
export -n variable
export -p
```

```
declare [选项] variable=value
```

```
printf format 输出参数列表
```

31

相关命令小结

```
basename
```

```
let
```

```
$(expression) 、 $((expression))
```

```
eval newstr=\${str2} 、 newstr=${!str2}
```

```
`hostname` 、 $(hostname)
```

```
${var:-word}、${var:=word}、${var:?word}、${var:+word}
```

```
$0、$1-$9、${n}、$#、$*、$@、$$、$!、$?、$_
```

32