Shell 教程 Shell echo命令

Shell 变量

定义变量时，变量名不加美元符号（$，PHP语言中变量需要），如：

```
your_name="runoob.com"
```

注意，变量名和等号之间不能有空格，这可能和你熟悉的所有编程语言都不一样。同时，变量名的命名须遵循如下规则：

- 命名只能使用英文字母，数字和下划线，首个字符不能以数字开头。
- 中间不能有空格，可以使用下划线（_）。
- 不能使用标点符号。
- 不能使用bash里的关键字（可用help命令查看保留关键字）。

有效的 Shell 变量名示例如下：

```
RUNOOB
LD_LIBRARY_PATH
_var
var2
```

无效的变量命名：

```
?var=123
user*name=runoob
```

除了显式地直接赋值，还可以用语句给变量赋值，如：

```
for file in `ls /etc`
或
for file in $(ls /etc)
```

以上语句将 /etc 下目录的文件名循环出来。

使用变量

使用一个定义过的变量，只要在变量名前面加美元符号即可，如：

```
your_name="qinjx"
echo $your_name
echo ${your_name}
```

变量名外面的花括号是可选的，加不加都行，加花括号是为了帮助解释器识别变量的边界，比如下面这种情况：

```
for skill in Ada Coffe Action Java; do
    echo "I am good at ${skill}Script"
done
```

如果不给skill变量加花括号，写成echo "I am good at $skillScript"，解释器就会把$skillScript当成一个变量（其值为空），代码执行结果就不是我们期望的样子了。

推荐给所有变量加上花括号，这是个好的编程习惯。

已定义的变量，可以被重新定义，如：

```
your_name="tom"
echo $your_name
your_name="alibaba"
echo $your_name
```

这样写是合法的，但注意，第二次赋值的时候不能写$your_name="alibaba"，使用变量的时候才加美元符（$）。

只读变量

使用 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。

下面的例子尝试更改只读变量，结果报错：

```
#!/bin/bash
myUrl="http://www.google.com"
readonly myUrl
myUrl="http://www.runoob.com"
```

运行脚本，结果如下：

```
/bin/sh: NAME: This variable is read only.
```

删除变量

使用 unset 命令可以删除变量。语法：

```
unset variable_name
```

变量被删除后不能再次使用。unset 命令不能删除只读变量。

实例

```
#!/bin/sh
myUrl="http://www.runoob.com"
```

```
unset myUrl
echo $myUrl
```

以上实例执行将没有任何输出。

变量类型

运行shell时，会同时存在三种变量：

- **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

字符串是shell编程中最常用最有用的数据类型（除了数字和字符串，也没啥其它类型好用了），字符串可以用单引号，也可以用双引号，也可以不用引号。单双引号的区别跟PHP类似。

单引号

```
str='this is a string'
```

单引号字符串的限制：

- 单引号里的任何字符都会原样输出，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的单引号（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。

双引号

```
your_name='runoob'
str="Hello, I know you are \"$your_name\"! \n"
echo -e $str
```

输出结果为：

```
Hello, I know you are "runoob"!
```

双引号的优点：

- 双引号里可以有变量
- 双引号里可以出现转义字符

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} !
```

获取字符串长度

```
string="abcd"
echo ${#string} #输出 4
```

提取子字符串

以下实例从字符串第 **2** 个字符开始截取 **4** 个字符：

```
string="runoob is a great site"
echo ${string:1:4} # 输出 unoo
```

查找子字符串

查找字符 **i** 或 **o** 的位置(哪个字母先出现就计算哪个)：

```
string="runoob is a great site"
echo `expr index "$string" io`  # 输出 4
```

注意： 以上脚本中 `` ` `` 是反引号，而不是单引号 `'`，不要看错了哦。

Shell 数组

bash支持一维数组（不支持多维数组），并且没有限定数组的大小。

类似于 C 语言，数组元素的下标由 0 开始编号。获取数组中的元素要利用下标，下标可以是整数或算术表达式，其值应大于或等于 0。

定义数组

在 Shell 中，用括号来表示数组，数组元素用"空格"符号分割开。定义数组的一般形式为：

```
数组名=(值1 值2 ... 值n)
```

例如：

```
array_name=(value0 value1 value2 value3)
```

或者

```
array_name=(
value0
value1
value2
value3
)
```

还可以单独定义数组的各个分量：

```
array_name[0]=value0
array_name[1]=value1
array_name[n]=valuen
```

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

读取数组元素值的一般格式是：

```
${数组名[下标]}
```

例如：

```
valuen=${array_name[n]}
```

使用 `@` 符号可以获取数组中的所有元素，例如：

```
echo ${array_name[@]}
```

获取数组的长度

获取数组长度的方法与获取字符串长度的方法相同，例如：

```
# 取得数组元素的个数
length=${#array_name[@]}
```

```
# 或者
length=${#array_name[*]}
# 取得数组单个元素的长度
lengthn=${#array_name[n]}
```

Shell 注释

以 `#` 开头的行就是注释，会被解释器忽略。

通过每一行加一个 **#** 号设置多行注释，像这样：

```
#--------------------------------------------
# 这是一个注释
# author：菜鸟教程
# site：www.runoob.com
# slogan：学的不仅是技术，更是梦想！
#--------------------------------------------
##### 用户配置区 开始 #####
#
#
# 这里可以添加脚本描述信息
#
#
##### 用户配置区 结束  #####
```

如果在开发过程中，遇到大段的代码需要临时注释起来，过一会儿又取消注释，怎么办呢？

每一行加个#符号太费力了，可以把这一段要注释的代码用一对花括号括起来，定义成一个函数，没有地方调用这个函数，这块代码就不会执行，达到了和注释一样的效果。

多行注释

多行注释还可以使用以下格式：

```
:<<EOF
注释内容...
注释内容...
注释内容...
EOF
```

EOF 也可以使用其他符号:

```
:<<'
注释内容...
注释内容...
注释内容...
'
```

```
:<<!
注释内容...
注释内容...
注释内容...
!
```

Shell 教程 Shell echo命令