

## 42 变量在手，Shell不愁

更新时间：2019-08-09 10:20:02



“

学习这件事不在乎有没有人教你，最重要的是在于你自己有没有觉悟和恒心。

—— 法布尔

”

### 内容简介

1. 前言
2. 定义变量
3. echo：显示内容
4. read：请求输入
5. 数学运算
6. 环境变量
7. 参数变量
8. 数组
9. 总结

### 1. 前言

上一课 [带你玩转Linux和Shell编程 | 第五部分第三课：一入Shell深似海，酷炫外壳惹人爱](#)，我们初步学习了 Shell。

这一课我们正式深入 Shell 编程的学习。

跟几乎所有其它编程语言一样，Shell 语言中也有“变量”，英语是 `variable`。我们可以用变量在内存中暂时储存信息。

正如我们上一课所说，我们将用 Bash 这一种 Shell 来演示 Shell 编程。如果你用的是其它种类的 Shell，也没太大关系，毕竟语法是类似的。

之后的课程，Bash 和 Shell 这两个词会混用，表示以下意思：“Bash 这个 Shell”或者“Shell 的统称”。请就不同语境自行理解。不过基本上我们都会用比较准确的那一个词。

Bash 中的变量比较特殊。如果你已经用过 C 语言或其它编程语言，那么你会惊讶于 Bash 中变量的运作方式。

学好 Shell 编程是很有用的。好了，平复一下激动的心情，我们开始探索吧。

## 2. 定义变量

首先，我们来创建一个新的文件，起名叫 `variable.sh` (`variable` 是英语“变量”的意思)。

```
vim variable.sh
```

在这个文件的第一行，我们需要指明用哪一种 Shell 程序来解析我们的文件。因为我们是学习 Bash 这个 Shell，因此这样写：

```
#!/bin/bash
```

这就表明了我们要编写 Bash 程序。

现在，就让我们来定义一个变量吧。所有的变量都有一个名字和一个值：

```
message='Hello World'
```

‘Hello World’ 是英语“你好，世界”的意思。

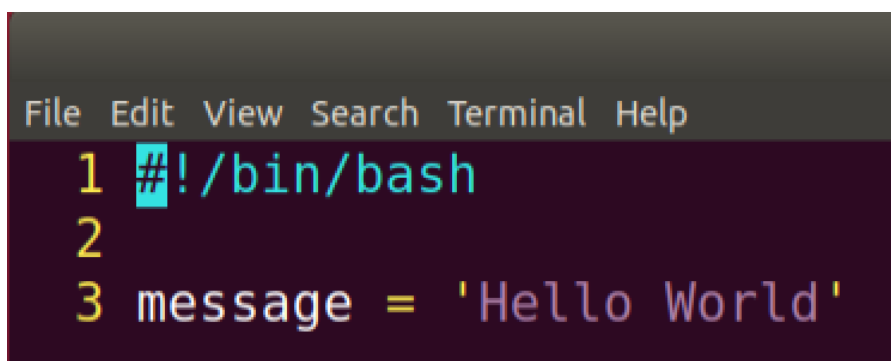
通过上面这句话，我们创建了一个变量 `message`（英语“信息，消息”的意思）：

- 变量名是 `message`
- 变量值是 `Hello World`

注意：在等号两边不要加空格。

Bash 的语法在不少地方是比较“吹毛求疵”的，因此最好不要“惹到”它。我们的课程中会不时提醒大家该注意的地方，因为有不少。

当然了，你可以尝试在等号两边加空格：



```
File Edit View Search Terminal Help
1 #!/bin/bash
2
3 message = 'Hello World'
```

但你会发现运行这个脚本之后将出错：

```
oscar@oscar-laptop: ~  
File Edit View Search Terminal Help  
oscar@oscar-laptop:~$ chmod +x variable.sh  
oscar@oscar-laptop:~$ ./variable.sh  
./variable.sh: line 3: message: command not found  
oscar@oscar-laptop:~$
```

错误信息是：

```
line 3 : message: command not found
```

意思是：“错误在第三行，没有找到 message 这个命令”。

可见，如果在等号两边加上空格，message 就被“孤立”起来了，那么 Bash 就会把 message 当成是一个命令，但这命令又不存在。

好了，我们回到主题。如果你要在变量的值中加入一个单引号（`'`），需要在前面加上反斜杠（`\`）。

事实上，单引号用于界定内容，所以要真的在内容里插入一个单引号，就须要在前面用一个特殊的符号来指明，这个特殊的符号就是反斜杠，也称为“转义字符”，或简称“转义符”。例如：

```
message='Hello, it\'s me'
```

好了，重新来看我们的 Shell 脚本文件，现在它的内容应该是这样的：

```
#!/bin/bash  
  
message='Hello World'
```

我们来运行这个脚本看看（当然了，先要给它运行的权限，应该知道怎么做吧。如果跟着我们的课程学到现在还不知道怎么做的话...“来人哪，拖出去...”）：

```
./variable.sh
```

```
oscar@oscar-laptop: ~  
File Edit View Search Terminal Help  
oscar@oscar-laptop:~$ chmod +x variable.sh  
oscar@oscar-laptop:~$ ./variable.sh  
oscar@oscar-laptop:~$
```

可以看到，回车将其运行起来之后，什么也没发生。

那么，这个脚本到底做了什么呢？

回答：“它把 `Hello World` 放入了内存中，这就是它所做的所有事情了。所以在屏幕上并没有显示任何东西。”

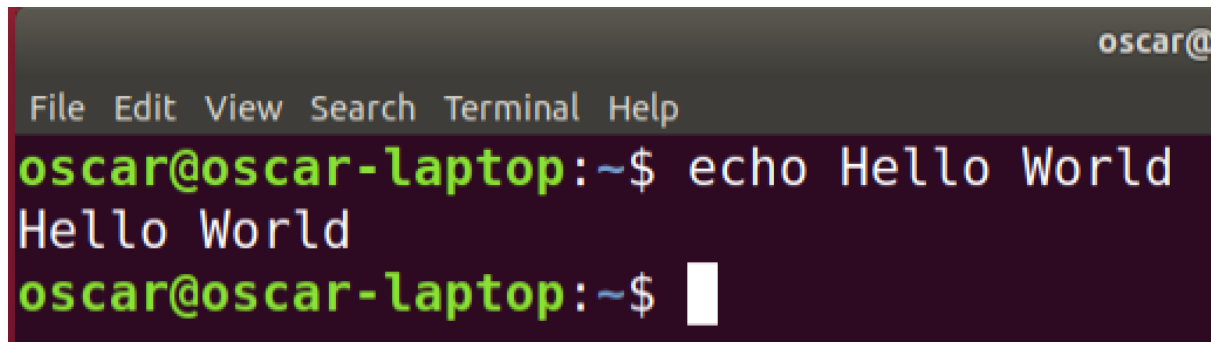
为了显示一个变量，我们需要来学习一个新命令。

### 3. echo : 显示内容

在我们继续讲解变量之前，我们先来介绍一个很重要的命令：echo。

echo 在英语中是“回声”的意思。它的作用是在终端上显示传入的信息。例如：

```
echo Hello World
```

A terminal window with a dark background and light green text. The prompt is 'oscar@oscar-laptop:~\$'. The command 'echo Hello World' is entered, and the output 'Hello World' is displayed on the next line. The prompt is then shown again with a cursor.

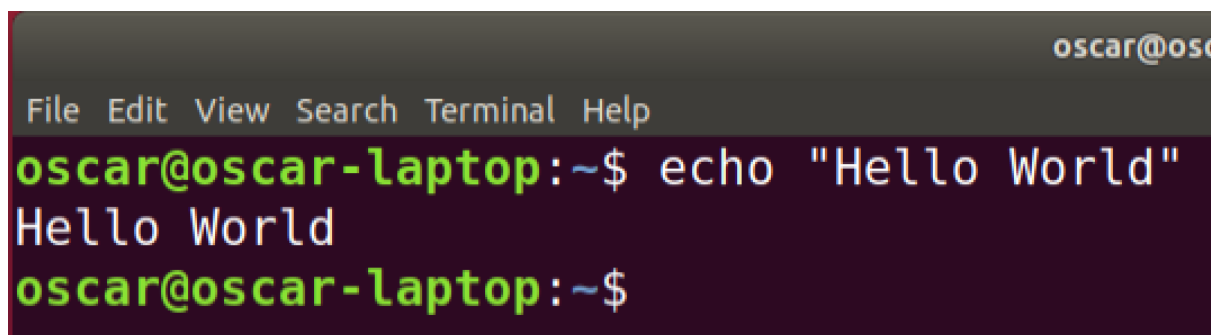
如你所见，使用起来很简单。这里引号却并不必要了。

但是，echo 的运行机制到底是什么呢？

实际上，上面的那句命令中，我们传给了 echo 两个参数：Hello 和 World。

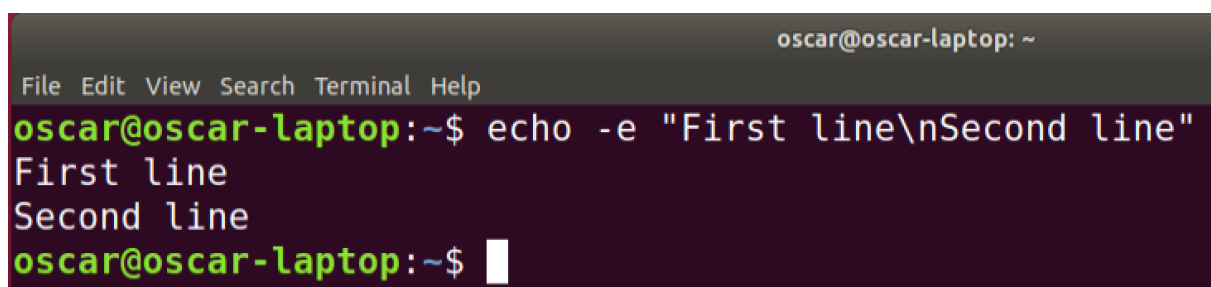
每个单词都被当做是 echo 命令的参数，被显示了出来。如果用引号把这几个单词都括起来，那么就会被当作一个参数了。

```
echo "Hello World"
```

A terminal window with a dark background and light green text. The prompt is 'oscar@oscar-laptop:~\$'. The command 'echo "Hello World"' is entered, and the output 'Hello World' is displayed on the next line. The prompt is then shown again with a cursor.

如果要插入换行符，那么需要用到 -e 参数（为了使“转义字符”发生作用），在句子中也要加入 \n，以表示换行：

```
echo -e "First line\nSecond line"
```

A terminal window with a dark background and light green text. The prompt is 'oscar@oscar-laptop:~\$'. The command 'echo -e "First line\nSecond line"' is entered, and the output is displayed on two lines: 'First line' and 'Second line'. The prompt is then shown again with a cursor.

### 显示变量

在我们的 Bash 脚本中，如果要显示一个变量，用 `echo` 后接变量名还不够，须要在变量名前加上美元符号（`$`）。

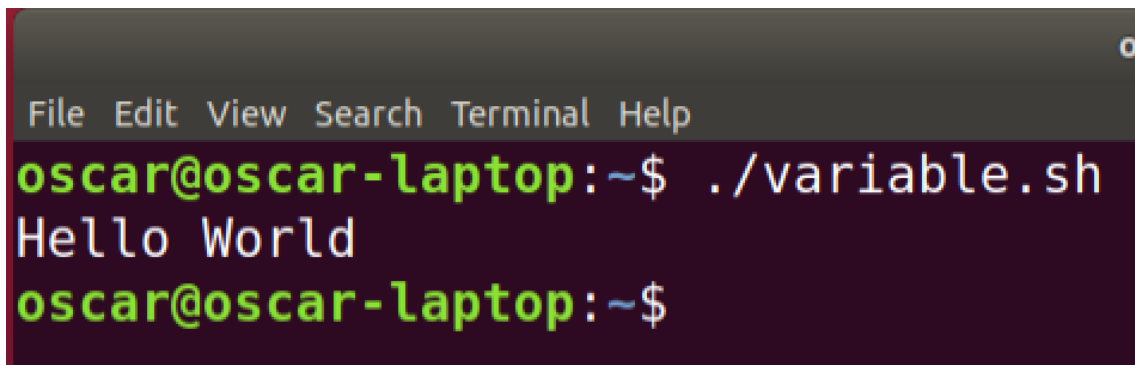
例如：

```
#!/bin/bash

message='Hello World'
echo $message
```

对比第 3 行和第 4 行：当我们定义一个变量（第 3 行）的时候，不需要在前面加美元符号（`$`）；但是要使用此变量（第 4 行）时，就要加上美元符号。

运行结果：



A terminal window with a dark background and light green text. The menu bar at the top includes 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The prompt is 'oscar@oscar-laptop:~\$'. The user enters './variable.sh', and the output is 'Hello World'. The prompt returns to 'oscar@oscar-laptop:~\$'.

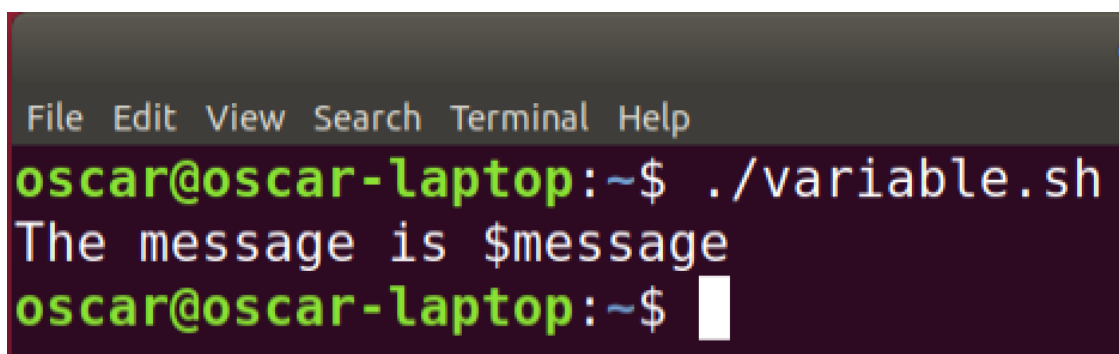
至此，我们已经学会在 Shell 脚本中显示变量了，不错不错。

现在，假设我们要用 `echo` 同时显示文字和变量值。也许你会认为这样行得通：

```
#!/bin/bash

message='Hello World'
echo 'The message is $message'
```

但是，运行起来后，结果却并不是我们所想的那样（“早知道伤心总是难免的，你又何苦一往情深...”）：



A terminal window with a dark background and light green text. The menu bar at the top includes 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The prompt is 'oscar@oscar-laptop:~\$'. The user enters './variable.sh', and the output is 'The message is \$message'. The prompt returns to 'oscar@oscar-laptop:~\$'.

可以看到，运行起来后，显示是这样的：

```
The message is $message
```

为了更好地理解到底是怎么回事，我们需要学习一下引号的作用。

## 引号

我们可以用引号来界定包含空格的字符串。

引号一共有三种：

类型	表示
单引号	'
双引号	"
反引号	`

根据引号类型不同，`Bash` 的处理方式也会不同。

## 单引号

我们从单引号开始学习吧。美式键盘中，单引号( `'` ) 位于 `Enter` 键（回车键）的左方。

之前的例子里，我们也已经使用过单引号了。

```
#!/bin/bash

message='Hello World'
echo 'The message is $message'
```

正如之前我们测试的一样，如果变量被包含在单引号里面，那么变量不会被解析，美元符号（`$`）保持原样输出。

显示：

```
The message is $message
```

因为：单引号忽略被它括起来的所有特殊字符。

## 双引号

一般来说，要输入双引号，需要用“`Shift`键 + 单引号的按键”。

不同于单引号忽略所有特殊字符，双引号忽略大多数特殊字符，但不包括：美元符号（`$`）、反引号（```）、反斜杠（`\`），这 3 种特殊字符将不被忽略。不忽略美元符号意味着 `Shell` 在双引号内部可进行变量名替换。

例如：

```
#!/bin/bash

message='Hello World'
echo "The message is $message"
```

执行以上脚本，显示：

```
The message is Hello World
```

```
File Edit View Search Terminal Help
oscar@oscar-laptop:~$ ./variable.sh
The message is Hello World
oscar@oscar-laptop:~$
```

可以看到用了双引号，我们预期的效果就实现了。这一次，变量 `message` 被解析，内容被显示出来了。

## 反引号

反引号 ( ``` ) 不太常用，位于键盘的 `Tab` 键的上方、数字键 `1` 的左方。

反引号要求 `Shell` 执行被它括起来的内容。什么意思呢？我们来看一个例子你就懂了：

```
#!/bin/bash

message=`pwd`
echo "You are in the directory $message"
```

运行这个脚本，显示：

```
You are in the directory /home/oscar
```

```
File Edit View Search Terminal Help
oscar@oscar-laptop:~$ ./variable.sh
You are in the directory /home/oscar
oscar@oscar-laptop:~$
```

当然了，`/home/oscar` 是我的情况，因为我的 `variable.sh` 这个 `Shell` 脚本文件是位于 `/home/oscar` 中。如果你运行上面的脚本，`pwd` 命令显示的目录应该和我不一样。

可以看到，`pwd` 命令被执行了，执行的结果（显示当前所在目录）被赋值给 `message` 变量了。

一开始可能不是很好理解，不过反引号是很有用的，我们之后的课程还会继续和它打交道。

### 4. `read`：请求输入

我们可以请求用户输入文本，这就要用到 `read` 命令了。

`read` 是英语“阅读，读取”的意思。`read` 命令读取到的文本会立即被储存在一个变量里。

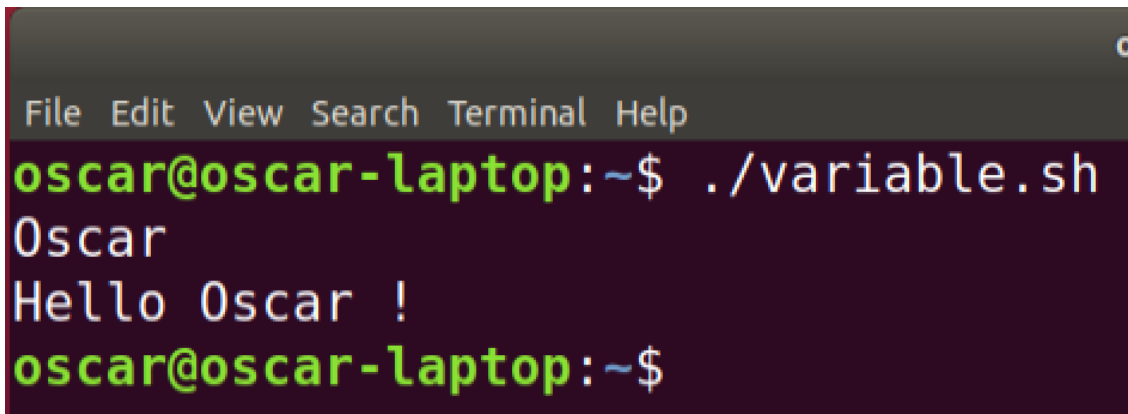
`read` 命令有好些有趣的用法。首先，最简单的当然是直接后接一个变量名，这样用户输入的文本就会被储存在这个变量中。

我们来写一个例子：

```
#!/bin/bash

read name
echo "Hello $name !"
```

我们运行上面的这个脚本，但什么都没显示。但是你可以输入文本，例如：

A terminal window with a dark background and light-colored text. The menu bar at the top shows 'File Edit View Search Terminal Help'. The prompt is 'oscar@oscar-laptop:~\$'. The user has run './variable.sh'. The script has printed 'Oscar' and 'Hello Oscar !'. The prompt is now 'oscar@oscar-laptop:~\$' again.

```
File Edit View Search Terminal Help
oscar@oscar-laptop:~$ ./variable.sh
Oscar
Hello Oscar !
oscar@oscar-laptop:~$
```

上面我输入了 Oscar，再按回车键，脚本就显示了 `Hello Oscar !`。

因为 read 命令读取了我在终端的输入（“Oscar”），将“Oscar”这个字符串赋值给 name 变量，所以才有了后面的输出。

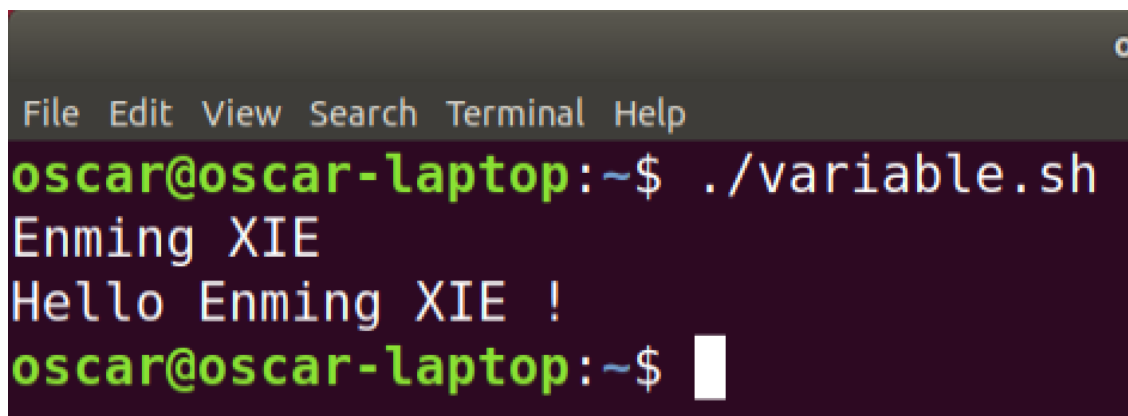
## 同时给几个变量赋值

我们可以用 read 命令一次性给多个变量赋值。如下：

```
#!/bin/bash

read firstname lastname
echo "Hello $firstname $lastname !"
```

我们运行以上脚本：

A terminal window with a dark background and light-colored text. The menu bar at the top shows 'File Edit View Search Terminal Help'. The prompt is 'oscar@oscar-laptop:~\$'. The user has run './variable.sh'. The script has printed 'Enming XIE' and 'Hello Enming XIE !'. The prompt is now 'oscar@oscar-laptop:~\$' again, followed by a cursor.

```
File Edit View Search Terminal Help
oscar@oscar-laptop:~$ ./variable.sh
Enming XIE
Hello Enming XIE !
oscar@oscar-laptop:~$
```

read 命令一个单词一个单词（单词是用空格分开的）地读取你输入的参数，并且把每个参数赋值给对应变量的。

这样我们输入的名和姓就分别被赋值给了 firstname（英语“名字”的意思）和 lastname（英语“姓氏”的意思）这两个变量。

如果你输入了比预期更多的参数，比如三个，四个，甚至更多，那么最后一个变量就会把多出来的参数全部拿走。



例如，我输入了 `Enming XIE`，按回车键。那么 `firstname` 变量就会取走 `Enming` 这个参数，而 `lastname` 变量则会取走 `XIE` 这个参数。

## -p：显示提示信息

目前来说，我们的程序对用户来说比较难理解，因为没有任何提示信息，用户也许不知道该做什么。

幸好，`read` 命令提供了 `-p` 参数，`p` 是 `prompt` 的首字母，表示“提示”。

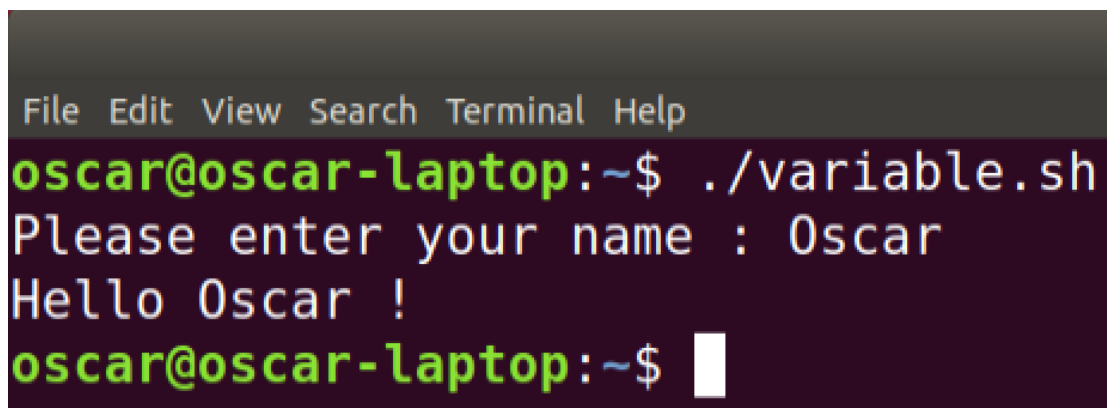
我们举个例子：

```
#!/bin/bash

read -p 'Please enter your name : ' name
echo "Hello $name !"
```

注意，`'Please enter your name : '` 是被引号括起来的，表示“请输入你的名字”。

运行以上脚本：



```
File Edit View Search Terminal Help
oscar@oscar-laptop:~$ ./variable.sh
Please enter your name : Oscar
Hello Oscar !
oscar@oscar-laptop:~$
```

这下我们的程序就比较友好了，因为用户在提示下就知道要做什么。

## -n：限制字符数目

用 `-n` 参数，我们可以限制用户输入的字符串的最大长度（字符数）。`n` 是 `number` 的首字母，是英语“数目”的意思。

例子：

```
#!/bin/bash

read -p 'Please enter your name (5 characters max) : ' -n 5 name
echo "Hello $name !"
```

“5 characters max” 是英语“最多 5 个字符”的意思。

运行这个脚本，我们发现一旦输入的字符数达到了我们限定的 5 个，那么 `Bash` 会立即显示“Hello + 我们输入的字符 !”，都不需要我们按下回车键。如下：

```
oscar@oscar-laptop: ~  
File Edit View Search Terminal Help  
oscar@oscar-laptop:~$ ./variable.sh  
Please enter your name (5 characters max) : OscarHello Oscar !  
oscar@oscar-laptop:~$
```

但是没有回车换行，带来的结果就是显示的字符串直接跟在我们输入的字符串后面了，不美观。我们可以加一个 `\n` 符号来换行：

```
#!/bin/bash  
  
read -p 'Please enter your name (5 characters max) : ' -n 5 name  
echo -e "\nHello $name !"
```

这次显示就美观了：

```
oscar@oscar-laptop: ~  
File Edit View Search Terminal Help  
oscar@oscar-laptop:~$ ./variable.sh  
Please enter your name (5 characters max) : Oscar  
Hello Oscar !  
oscar@oscar-laptop:~$
```

## **-t**：限制输入时间

用 `-t` 参数，我们可以限定用户的输入时间（以秒为单位），也就是说超过这个时间，就不读取输入了。t 是 time 的首字母，是英语“时间”的意思。

例子：

```
#!/bin/bash  
  
read -p 'Please enter the code to defuse the bomb (you have 5 seconds) : ' -t 5 code  
echo -e "\nBoom !"
```

‘Please enter the code to defuse the bomb (you have 5 seconds)’ 表示“请输入密码以拆除炸弹（你有 5 秒钟时间）”。

Boom 表示“爆炸声”。

运行这个脚本，如果你的输入时间超过了 5 秒钟，那么会显示 “Boom!”。

```
oscar@oscar-laptop: ~  
File Edit View Search Terminal Help  
oscar@oscar-laptop:~$ ./variable.sh  
Please enter the code to defuse the bomb (you have 5 seconds) : 1234  
Bomb!
```

## **-s**：隐藏输入内容

用 `-s` 参数，我们可以隐藏输入内容。一般用不到，但是如果你想要用户输入的是一个密码，那 `-s` 参数还是有用的。

`s` 是 `secret` 的首字母，表示“秘密”。

例子：

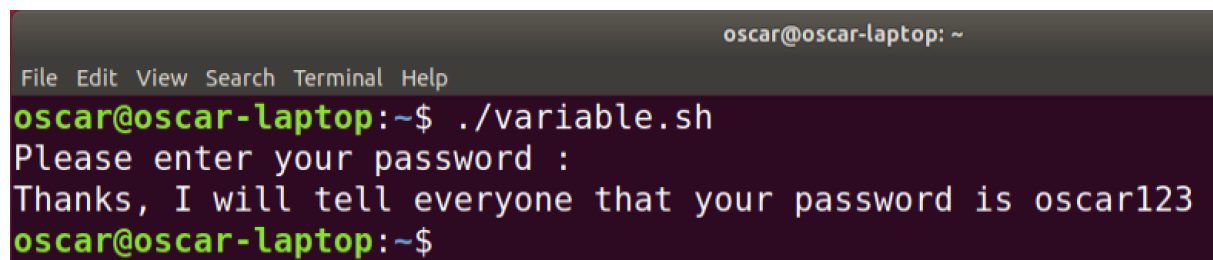
```
#!/bin/bash

read -p 'Please enter your password : ' -s password
echo -e "\nThanks, I will tell everyone that your password is $password"
```

‘Please enter your password’ 表示“请输入您的密码”。

‘Thanks, I will tell everyone that your password is’ 表示“谢谢，我会告诉每个人你的密码是”。哈哈，皮一下很开心~

运行：

A terminal window titled 'oscar@oscar-laptop: ~' with a menu bar (File, Edit, View, Search, Terminal, Help). The prompt is 'oscar@oscar-laptop:~\$'. The user enters './variable.sh'. The script prompts 'Please enter your password :', the user enters 'oscar123' (invisible). The script then outputs 'Thanks, I will tell everyone that your password is oscar123'. The prompt returns to 'oscar@oscar-laptop:~\$'.

可以看到，`read` 命令在读取输入时就不显示输入内容了。

## 5. 数学运算

请大家牢记：

在 Bash 中，所有的变量都是字符串！

Bash 本身不会操纵数字，因此它也不会做运算，就是这么“呆萌”。这是与其它编程语言不一样的地方。

幸好，我们可以用命令来达到目的。我们需要用到 `let` 命令。

`let` 是英语“让，使之...”的意思。因此 `let` 命令可以用于赋值。

例如：

```
let "a = 5"
let "b = 2"
let "c = a + b"
```

在上面这三句命令结束之后，`c` 的值将是 7。我们可以测试一下：

```
#!/bin/bash

let "a = 5"
let "b = 2"
let "c = a + b"

echo "c = $c"
```

```
File Edit View Search Terminal Help
oscar@oscar-laptop:~$ ./variable.sh
c = 7
oscar@oscar-laptop:~$
```

可用的运算符是以下几种：

运算	符号
加法	+
减法	-
乘法	*
除法	/
幂（乘方）	**
余（整数除法的余数）	%

除了 幂（乘方）有点特殊（使用两个星号 `**`），其它都和一般的编程语言类似。

来看一些例子：

```
let "a = 5 * 3" # $a = 15
let "a = 4 ** 2" # $a = 16 (4 的平方)
let "a = 8 / 2" # $a = 4
let "a = 10 / 3" # $a = 3
let "a = 10 % 3" # $a = 1
```

最后这两个例子我们做一下解释：

- `10 / 3 = 3` 因为是整数除法，所以没有小数。
- `10 % 3 = 1` 因为 10 除以 3，商是 3，余数是 1。

和其他大多数主流编程语言一样，Bash 也支持运算的连写，和 C 语言一样：

```
let "a = a * 3"
```

和：

```
let "a *= 3"
```

效果是一样的。

暂时，我们只学习了整数的运算，如果你要做带小数的运算，那么需要用到 `bc` 命令，可以自己去查阅，可以用 `man bc`。

## 6. 环境变量

到目前为止，我们在脚本文件中创建的变量只存在于脚本中。换言之，在 A 脚本程序中定义的变量不能被 B 脚本程序使用。

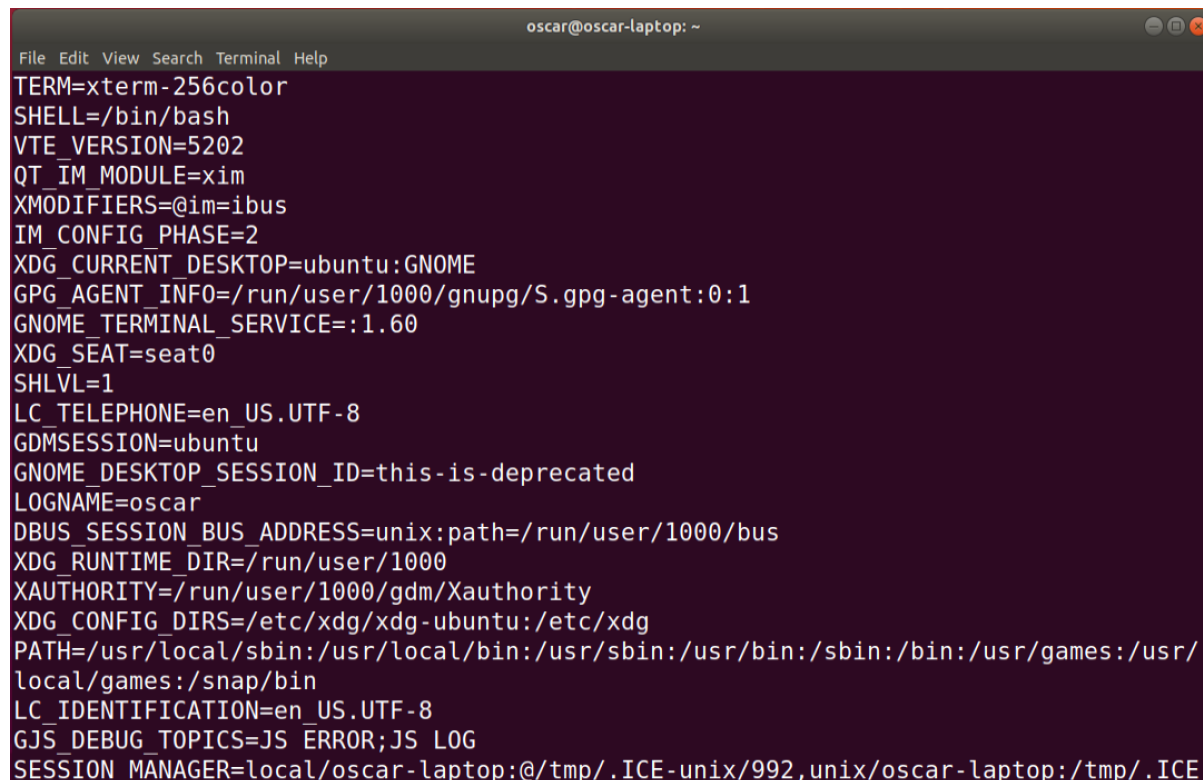
我们来学习一个被称为“环境变量”的特殊变量。顾名思义，Shell 的环境变量可以被此种 Shell 的任意脚本程序使用。我们有时也把环境变量称为“全局变量”。

我们可以用 `env` 命令来显示你目前所有的环境变量：

```
env
```

`env` 是 `environment` 的缩写，是英语“环境”的意思。

显示如下：



```
oscar@oscar-laptop: ~  
File Edit View Search Terminal Help  
TERM=xterm-256color  
SHELL=/bin/bash  
VTE_VERSION=5202  
QT_IM_MODULE=xim  
XMODIFIERS=@im=ibus  
IM_CONFIG_PHASE=2  
XDG_CURRENT_DESKTOP=ubuntu:GNOME  
GPG_AGENT_INFO=/run/user/1000/gnupg/S.gpg-agent:0:1  
GNOME_TERMINAL_SERVICE=:1.60  
XDG_SEAT=seat0  
SHLVL=1  
LC_TELEPHONE=en_US.UTF-8  
GDMSESSION=ubuntu  
GNOME_DESKTOP_SESSION_ID=this-is-deprecated  
LOGNAME=oscar  
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus  
XDG_RUNTIME_DIR=/run/user/1000  
XAUTHORITY=/run/user/1000/gdm/Xauthority  
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg  
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin  
LC_IDENTIFICATION=en_US.UTF-8  
GJS_DEBUG_TOPICS=JS ERROR;JS LOG  
SESSION_MANAGER=local/oscar-laptop:@/tmp/.ICE-unix/992,unix/oscar-laptop:/tmp/.ICE
```

我的环境变量，一个屏幕还没显示完，因此我只截取了一部分。

其中比较重要的几个环境变量是：

**SHELL**：指明目前你使用的是哪种 Shell。我目前用的是 Bash（因为 `SHELL=/bin/bash`）。

**PATH**：是一系列路径的集合。只要有可执行程序位于任意一个存在于 `PATH` 中的路径，那我们就可以直接输入可执行程序的名字来执行，而不需要加上所在路径前缀或进入到可执行程序所在目录去执行。上一课我们已经学习过 `PATH` 了。

**HOME**：你的家目录所在的路径。

**PWD**：目前所在的目录。

可以看到，这些环境变量的名字都约定俗成是大写的。

如何使用这些环境变量呢？很简单，就和平时使用变量一样：

```
#!/bin/bash  
  
echo "Your default Shell is $SHELL"
```

“Your default Shell is” 表示“你的默认的 Shell 是”。

```
File Edit View Search Terminal Help
oscar@oscar-laptop:~$ ./variable.sh
Your default Shell is /bin/bash
oscar@oscar-laptop:~$
```

有时，我们需要自己定义环境变量。你可以用 `export` 命令来完成。

`export` 是英语“输出，出口”的意思。用 `man export` 来查看 `export` 的命令手册：

```
oscar@oscar-laptop: ~
File Edit View Search Terminal Help
EXPORT(1POSIX)          POSIX Programmer's Manual          EXPORT(1POSIX)

PROLOG
This manual page is part of the POSIX Programmer's Manual. The Linux
implementation of this interface may differ (consult the corresponding
Linux manual page for details of Linux behavior), or the interface may
not be implemented on Linux.

NAME
export - set the export attribute for variables

SYNOPSIS
export name[=word]...

export -p

DESCRIPTION
The shell shall give the export attribute to the variables corresponding
to the specified names, which shall cause them to be in the environment
of subsequently executed commands. If the name of a variable is followed
by =word, then the value of that variable shall be set to word.

The export special built-in shall support the Base Definitions volume of
Manual page export(1posix) line 1 (press h for help or q to quit)
```

可以看到简单描述是“set the export attribute for variables”，表示“将属性值赋值给变量”。

可以看到用法之一是：

```
export name[=word]...
```

在 `.bashrc` 或 `.zshrc` 这样的 Shell 配置文件里可以找到这样的命令。比如我们之前在 `.bashrc` 文件的最后加过一句命令：

```
export EDITOR=nano
```

```
export EDITOR=nano
```

如上图，我们就定义了一个环境变量 EDITOR，它的值是 nano。editor 是英语“编辑器”的意思，所以当前 Shell 默认的编辑器是 Nano。

export 命令应该也算是经常会被用到的命令，特别是做软件开发的时候，经常需要用这个命令来设置一些环境变量，而且最好是永久性地设置在 .bashrc 等 Shell 的配置文件里。

## 7. 参数变量

就跟我们之前学过的各种 Linux 命令一样，你的 Shell 脚本也可以接收参数。

假设，我们可以这样调用我们的脚本文件：

```
./variable.sh 参数1 参数2 参数3 ...
```

这些个 参数1，参数2，参数3 ... 被称为“参数变量”。

但问题是我们还不知道如何接收这些参数到我们的脚本中。

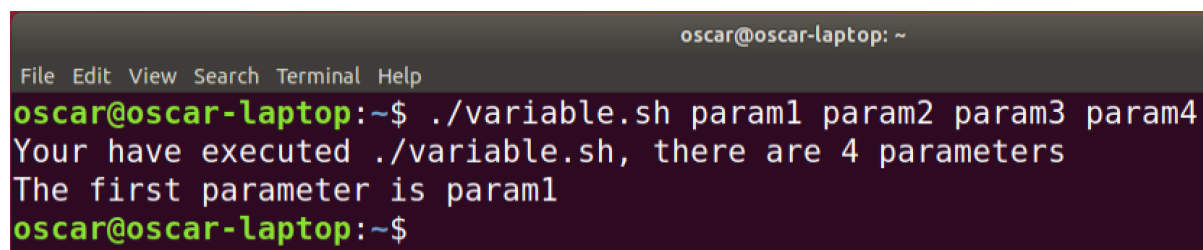
其实不难，因为这些变量是被自动创建的。

- `$#`：包含参数的数目。
  - `$0`：包含被运行的脚本的名称（我们的示例中就是 variable.sh）。
  - `$1`：包含第一个参数。
  - `$2`：包含第二个参数。
  - ...
  - `$8`：包含第八个参数。
  - ...
- 以此类推。

试一下：

```
#!/bin/bash

echo "You have executed $0, there are $# parameters"
echo "The first parameter is $1"
```



A terminal window titled 'oscar@oscar-laptop: ~' showing the execution of the script. The prompt is 'oscar@oscar-laptop:~\$' and the command entered is './variable.sh param1 param2 param3 param4'. The output shows 'Your have executed ./variable.sh, there are 4 parameters' and 'The first parameter is param1'. The prompt returns to 'oscar@oscar-laptop:~\$'.

如果我们有很多很多参数怎么办呢？可以用 shift 命令来“挪移”参数，以便依次处理。

shift 是英语“移动，移位”的意思。

什么意思呢？我们来看例子：

```
#!/bin/bash

echo "The first parameter is $1"
shift
echo "The first parameter is now $1"
```

```
oscar@oscar-laptop: ~
File Edit View Search Terminal Help
oscar@oscar-laptop:~$ ./variable.sh param1 param2 param3 param4
The first parameter is param1
The first parameter is now param2
oscar@oscar-laptop:~$
```

可以看到，在调用 shift 命令后，\$1 对应了第二个参数，\$2 对应了第三个参数，以此类推。

因此，shift 命令常被用在循环中，使得参数一个接一个地被处理。之后的课程我们会学习“循环”。

## 8. 数组

Bash 也支持数组类型。

数组是这样一种变量，它可以包含多个“格子”（被称为“数组的元素”），就好像一个表格一样。

人員	第一季	第二季	第三季	第四季	小計
甲	26	63	64	35	188
乙	76	8	52	38	174
丙	41	82	18	1	142
丁	64	74	37	26	201
戊	39	99	28	2	168
己	28	31	67	43	169
小計	274	357	266	145	1042

例子：

```
array=('value0' 'value1' 'value2')
```

上面的语句定义了一个数组变量，名叫 array（array 是英语“数组”的意思），其中包含三个值：value0，value1，value2。

如果要访问其中一个格子的内容，要用到这样的语法：

```
${array[2]}
```

以上语句表示数组中编号为 2 的元素（在我们的情况就是 value2）。

注意：和大多数编程语言一样，Shell 中的数组的下标（index）也是从 0 开始的，而不是从 1 开始。因此，第一个元素的编号（下标）就是 0，第二个元素的下标就是 1，以此类推。

我们也可以单独给数组的元素赋值，例如：

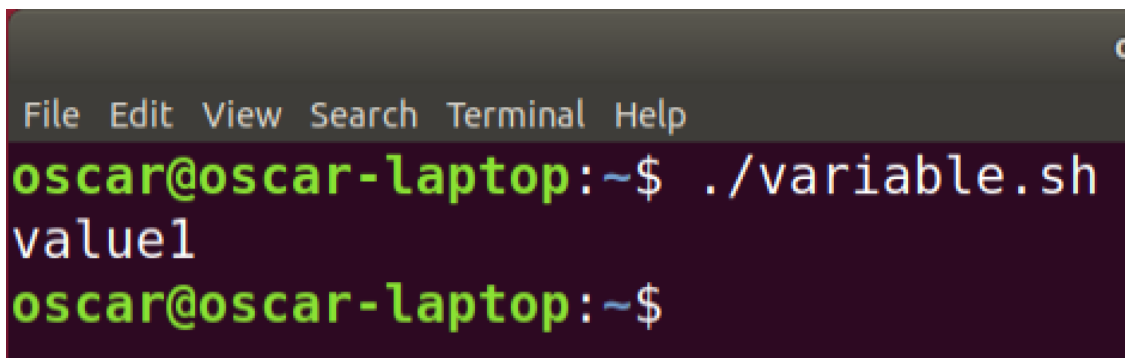
```
array[3]='value3'
```



来看一个例子：

```
#!/bin/bash

array=('value0' 'value1' 'value2')
array[5]='value5'
echo ${array[1]}
```

A terminal window with a dark background and light green text. The menu bar at the top shows 'File Edit View Search Terminal Help'. The prompt is 'oscar@oscar-laptop:~\$'. The user enters './variable.sh' and the output is 'value1'. The prompt returns to 'oscar@oscar-laptop:~\$'.

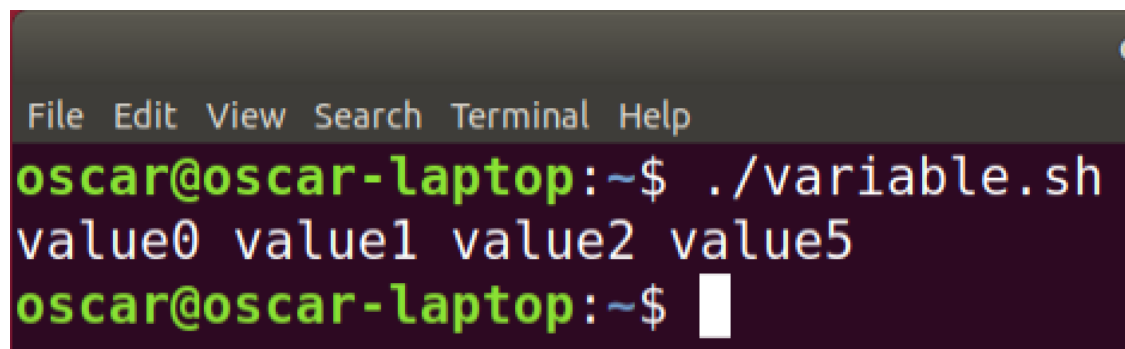
如你所见，数组可以包含任意大小的元素数目。而且数组的元素编号不需要是连续的，我们可以跳过一些“格子”。上面的程序中，我们就没有给数组的 3 号和 4 号“格子”赋值。

我们也可以一次性显示数组中所有的元素值，需要用到通配符 `*`（星号）。

例子：

```
#!/bin/bash

array=('value0' 'value1' 'value2')
array[5]='value5'
echo ${array[*]}
```

A terminal window with a dark background and light green text. The menu bar at the top shows 'File Edit View Search Terminal Help'. The prompt is 'oscar@oscar-laptop:~\$'. The user enters './variable.sh' and the output is 'value0 value1 value2 value5'. The prompt returns to 'oscar@oscar-laptop:~\$'.

这一课知识点还是很多的。大家需要多多练习，才能将这些内容记得更牢固。

## 9. 总结

和大多数编程语言一样，在 Shell 语言中我们也可以创建变量，变量用于在内存中暂存某些值。一个被起名叫 `variable` 的变量可以这样被访问：`$variable`；

`echo` 命令在终端显示文本或变量的值；

`read` 命令等待用户从键盘输入内容，并把输入的内容储存到变量里；

我们可以用 `let` 命令来实现算数运算；

环境变量可以被所有的脚本文件所使用。我们可以用 `env` 命令来列出所有的环境变量；

传递给脚本的参数会被转移到参数变量 `$1` , `$2` , `$3` ... 等等中。`$0` 则包含脚本的名称。`$#` 包含参数的数目。

今天的课就到这里，一起加油吧！

← 41 一入Shell深似海，酷炫外壳惹人爱

43 条件一出，Shell不服 →

————— 千学不如一看，千看不如一练 —————