Bash learning notes

"#!" 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。
注意，变量名和等号之间不能有空格
定义变量时
your_name="runoob.com"

printf <FORMAT> <ARGUMENTS...>
printf "Surname: %s\nName: %s\n" "$SURNAME" "$FIRSTNAME"

http://www.runoob.com/linux/linux-shell-printf.html

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。

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。

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
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引号（对单引号使用转义符后也不行）。
双引号

```
your_name='qinjx'
str="Hello, I know your are \"$your_name\"! \n"
```

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="qinjx"
greeting="hello, "$your_name" !"
greeting_1="hello, ${your_name} !"
echo $greeting $greeting_1
```

获取字符串长度

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

提取子字符串

```
string="alibaba is a great company"
echo ${string:1:4} #输出 liba
```

查找子字符串

```
string="alibaba is a great company"
echo `expr index "$string" is`
```

Shell 注释

以"#"开头的行就是注释，会被解释器忽略。

sh 里没有多行注释，只能每一行加一个#号。

for 循环
for 循环一般格式为：

```
for var in item1 item2 ... itemN
do
    command1
    command2
    ...
    commandN
done
```

```
for var in item1 item2 ... itemN; do command1; command2… done;
for str in 'This is a string'
do
    echo $str
done
```

while 语句

while 循环用于不断执行一系列命令，也用于从输入文件中读取数据；命令通常为测试条件。其格式为：

```
while condition
do
    command
done
```

```
#!/bin/sh
int=1
while(( $int<=5 ))
do
        echo $int
        let "int++"
done
```

Shell 基本运算符
原生 bash 不支持简单的数学运算，但是可以通过其他命令来实现，例如 awk 和 expr，expr 最常用。

expr 是一款表达式计算工具，使用它能完成表达式的求值操作。

例如，两个数相加：

```
#!/bin/bash

val=`expr 2 + 2`
echo "两数之和为 : $val"
```

表达式和运算符之间要有空格，例如 **2+2** 是不对的，必须写成 **2 + 2**，这与我们熟悉的大多数编程语言不一样。

乘号(*)前边必须加反斜杠(\)才能实现乘法运算；

运行 Shell 脚本

将上面的代码保存为 **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

echo "Greetings"

echo "Greetings $USER, your current working directory is $PWD"
Accepting Inputs

read number
echo "The number you have entered is $number"
if..then..fi statement (Simple If)
if..then..else..fi statement (If-Else)
if..elif..else..fi statement (Else If ladder)
if..then..else..if..then..fi..fi..(Nested if)

if [ conditional expression ]
then
        statement1
        statement2
        .
fi

#!/bin/bash
```

```
count=100
if [ $count -eq 100 ]
then
   echo "Count is 100"
fi

if [ $count -eq 100 ]
then
   echo "Count is 100"
else
   echo "Count is not 100"
fi

count = 99
if [ $count -eq 100 ]
then
   echo "Count is 100"
elif [ $count -gt 100 ]
then
   echo "Count is greater than 100"
else
   echo "Count is less than 100"
fi

#!/bin/bash
count = 99
if [ $count -eq 100 ]
then
   echo "Count is 100"
else
   if [ $count -gt 100 ]
   then
      echo "Count is greater than 100"
   else
   echo "Count is less than 100"
   fi
fi


~$ echo "5+5"
5+5
~$ echo "5+5"| bc
10
~$ echo "5+5"| bc -l
```

```
10
~$ echo "5+5.2"| bc -l
10.2
~$ echo "5+5.2"| bc
10.2
~$ echo "3/4"| bc
0
~$ echo "3/4"| bc -l
.75000000000000000000
~`$ echo $`((3+3))
6

~$ echo "scale = 2; 10 * 100 / 30" | bc
33.33
~$ echo "scale = 2; 10 / 30 * 100" | bc
33.00
~$ echo "scale = 2; (10 / 30) * 100" | bc
33.00
```

Be careful with spacing in such expressions! Bash is very sensitive to them.

1. Declaring an Array and Assigning values
In bash, array is created automatically when a variable is used in the format like,
数组名=(值 1 值 2 ... 值 n)
array_name=(value0 value1 value2 value3)
name[index]=value
name is any name for an array
index could be any number or expression that must evaluate to a number greater than or equal to zero.You can declare an explicit array using declare -a arrayname.

```
$ cat arraymanip.sh
#! /bin/bash
Unix[0]='Debian'
Unix[1]='Red hat'
Unix[2]='Ubuntu'
Unix[3]='Suse'

echo ${Unix[1]}

$./arraymanip.sh
Red hat
```

Print the Whole Bash Array
echo ${Unix[@]}

Length of the Bash Array
$ cat arraymanip.sh
declare -a Unix = ('Debian' 'Red hat' 'Suse' 'Fedora');

echo ${#Unix[@]} #Number of elements in the array
echo ${#Unix} #Number of characters in the first element of the array.i.e Debian
$./arraymanip.sh