

shell

1. 定义变量时，变量名不加美元符号\$：your\_name="blobbies"，注意，变量名和等号之间不能有空格。

2. 除了显示地直接赋值，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 下的目录文件名循环出来。

3. 使用变量时，只要在变量名前面加美元符号即可，如：

```
your_name="blobbies"
```

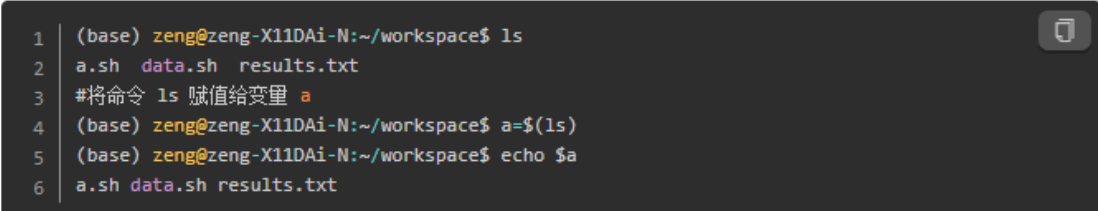
```
echo $your_name
```

```
echo ${your_name}
```

变量名外面的花括号是可选的，加花括号是为了帮助解释器识别变量的边界。

推荐给所有变量加上花括号，这是好的编程习惯。

在 bash shell 中，\$( )是将括号内命令的执行结果赋值给变量。



```
1 (base) zeng@zeng-X11DAI-N:~/workspace$ ls
2 a.sh data.sh results.txt
3 #将命令 ls 赋值给变量 a
4 (base) zeng@zeng-X11DAI-N:~/workspace$ a=$(ls)
5 (base) zeng@zeng-X11DAI-N:~/workspace$ echo $a
6 a.sh data.sh results.txt
```

#### 4. 局部变量、环境变量

5. 字符串推荐使用双引号

```
Str="hello, I know you are `${your_name}`"! \n"
```

```
Echo -e $str    #-e 开启转义
```

输出结果为：hello, I know you are "blobbies"!

6. 字符串拼接

```
Greeting="hello, ${your_name}"
```

```
Echo $greeting $greeting
```

输出结果为：hello,lobbies hello,lobbies

## 7. 获取字符串长度

```
Echo ${#your_name}
```

输出 8

## 8. 小段代码用#注释，大段代码可以

```
: <<EOF
```

```
EOF
```

## 9. 在执行脚本时，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，以此类推。

```
./test.sh 1 2 3
```

```
Echo “执行文件名：$0”
```

```
Echo “第一个参数：$1”
```

输出第一个参数： 1

参数处理	说明
\$#	传递到脚本的参数个数
\$*	以一个单字符串显示所有向脚本传递的参数。 如"\$*"用「」括起来的情况、以"\$1 \$2 ... \$n"的形式输出所有参数。
\$\$	脚本运行的当前进程ID号
\$_	后台运行的最后一个进程的ID号
\$@	与\$*相同，但是使用时加引号，并在引号中返回每个参数。 如"\$@"用「」括起来的情况、以"\$1" "\$2" ... "\$n" 的形式输出所有参数。
\$-	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，其他任何值表明有错误。

## 10. 显示结果定向至文件

```
Echo “it is a test” > myfile
```

## 11. Echo `date`

显示日期

## 12. 检查条件是否成立

### 数值测试:::

参数	说明
-eq	等于则为真
-ne	不等于则为真
-gt	大于则为真
-ge	大于等于则为真
-lt	小于则为真
-le	小于等于则为真

Num1=100, um2=100

If test \$[num1] -eq \$[num2]

Then

    Echo ‘两个数相等’

Else

    Echo “不等“

Fi

[] 执行基本算数运算, result=\$[1+2]

### 字符串测试:::

参数	说明
=	等于则为真
!=	不相等则为真
-z 字符串	字符串的长度为零则为真
-n 字符串	字符串的长度不为零则为真

### 文件测试:::

参数	说明
-e 文件名	如果文件存在则为真
-r 文件名	如果文件存在且可读则为真
-w 文件名	如果文件存在且可写则为真
-x 文件名	如果文件存在且可执行则为真
-s 文件名	如果文件存在且至少有一个字符则为真
-d 文件名	如果文件存在且为目录则为真
-f 文件名	如果文件存在且为普通文件则为真
-c 文件名	如果文件存在且为字符型特殊文件则为真
-b 文件名	如果文件存在且为块特殊文件则为真

If test -e ./bash

Then

Echo “件存在”

Else

Echo ”文件存在”

Fi

### 13. 流程控制

#### 实例

```
a=10
b=20
if [ $a == $b ]
then
    echo "a 等于 b"
elif [ $a -gt $b ]
then
    echo "a 大于 b"
elif [ $a -lt $b ]
then
    echo "a 小于 b"
else
    echo "没有符合的条件"
fi
```

#### 实例

```
num1=$((2*3))
num2=$((1+5))
if test ${num1} -eq ${num2}
then
    echo '两个数字相等!'
else
    echo '两个数字不相等!'
fi
```

For 循环：

```
For var in item1 item2 item3
Do
    Command1
    Command2
Done
```

While 循环:

```
While condition
Do
Done
```

多选择:

Case 值 in

模式 1)

```
    Command
;;
```

模式 2)

```
    Command
;;
```

Esac

## 14. 函数

```
demoFun(){
    echo "这是我的第一个 shell 函数!"
}
echo "-----函数开始执行-----"
demoFun
echo "-----函数执行完毕-----"
```

```
#!/bin/bash
# author: 菜鸟教程
# url: www.runoob.com

funWithReturn(){
    echo "这个函数会对输入的两个数字进行相加运算..."
    echo "输入第一个数字: "
    read aNum
    echo "输入第二个数字: "
    read anotherNum
    echo "两个数字分别为 $aNum 和 $anotherNum !"
    return $((aNum+anotherNum))
}
funWithReturn
echo "输入的两个数字之和为 $? !"
```

## 15. 输入输出定向

大多数 UNIX 系统命令从你的终端接受输入并将所产生的输出发送回到您的终端。

命令	说明
command > file	将输出重定向到 file。
command < file	将输入重定向到 file。
command >> file	将输出以追加的方式重定向到 file。
n > file	将文件描述符为 n 的文件重定向到 file。
n >> file	将文件描述符为 n 的文件以追加的方式重定向到 file。
n >& m	将输出文件 m 和 n 合并。
n <& m	将输入文件 m 和 n 合并。
<< tag	将开始标记 tag 和结束标记 tag 之间的内容作为输入。

需要注意的是文件描述符 0 通常是标准输入 (STDIN)，1 是标准输出 (STDOUT)，2 是标准错误输出 (STDERR)。

输出重定向:::

Command1 > file1 执行 command1 后将输出存入 file1

任何 file1 内的已经存在的内容将被新内容替代。如果要将新内容添加在文件

末尾，请使用>>操作符。

输入重定向:::

Command < file1 从文件获取输入，这样，本来需要从键盘获取输入的命令会转移到文件读取内容。

## 16. 重定向深入讲解

一般情况下，每个 Unix/Linux 命令运行时都会打开三个文件：

- 标准输入文件(stdin)：stdin的文件描述符为0，Unix程序默认从stdin读取数据。
- 标准输出文件(stdout)：stdout 的文件描述符为1，Unix程序默认向stdout输出数据。
- 标准错误文件(stderr)：stderr的文件描述符为2，Unix程序会向stderr流中写入错误信息。

默认情况下，command > file 将 stdout 重定向到 file，command < file 将stdin 重定向到 file。

如果希望 stderr 重定向到 file，可以这样写：

```
$ command 2>file
```

如果希望 stderr 追加到 file 文件末尾，可以这样写：

```
$ command 2>>file
```

2 表示标准错误文件(stderr)。

如果希望将 stdout 和 stderr 合并后重定向到 file，可以这样写：

```
$ command > file 2>&1
```

或者

```
$ command >> file 2>&1
```

如果希望对 stdin 和 stdout 都重定向，可以这样写：

```
$ command < file1 >file2
```

command 命令将 stdin 重定向到 file1，将 stdout 重定向到 file2。

## 17. &&、()、|| 决定 Linux 命令的执行顺序

Command1 && command2

：左边的 command1 执行成功后，command2 才能被执行。

`Command1 || command2`

: 左边的 `command1` 执行失败, 就执行 `command2`

`(command1;command2;command3)`

: 顺序执行多个命令

`Command1 | command2`

| 表示管道, 上一条命令的输出, 作为下一条命令参数。