

## 第三节 shell 常用语法

### 1. 概论

shell 是我们通过命令行与操作系统沟通的语言。

shell 脚本可以直接在命令行中执行，也可以将一套逻辑组织成一个文件，方便复用。

AC Terminal 中的命令行可以看成是一个 shell 脚本在逐行执行。

Linux 中常见的 shell 脚本有很多种，常见的有：

- Bourne shell(/usr/bin/sh 或 /bin/sh)
- Bourne Again Shell(bin/bash)
- C Shell(/usr/bin/csh)
- K Shell( /usr/bin/ksh)
- zsh
- ...

Linux 系统中一般默认使用 bash，所以接下来讲解 bash 中的语法

文件开头需要写 `#!/bin/bash`，指明 bash 为脚本解释器

### 脚本示例

新建一个 `test.sh` 文件，内容如下：

```
1  #! /bin/bash
2  echo "Hello world!"
```

### 运行方式

作为可执行文件

```
1  chmod +x test.sh #使脚本具有可执行权限
2  ./test.sh #当前路径下执行
3
4  /home/acs/test.sh # 绝对路径下执行
5
6  ~/test.sh #家目录路径下执行
7
8  #用解释器执行
9  bash test.sh
10
11 #1. 为test文件加执行权限，
12 acs@740d1bdabef2:~$ chmod +x test.sh      acs@740d1bdabef2:~$ ls
      homework  main.cpp  test.sh    #文件列表，此时test.sh 会改变颜色
13 #2. 直接执行当前目录的test.sh
14 acs@740d1bdabef2:~$ ./test.sh
15 Hello world! #脚本输出
16
17 #3. 绝对路径下执行文件
18 acs@740d1bdabef2:~$ /home/acs/test.sh
```

```
19 Hello world!      #脚本输出
20
21 #4.家目录下执行
22 acs@740d1bdabef2:~$ ~/test.sh
23 Hello world!      #脚本输出
24
```

```
Hello World!
acs@740d1bdabef2:~$ chmod +x test.sh
acs@740d1bdabef2:~$ ls
homework  main.cpp  test.sh
acs@740d1bdabef2:~$ ./test.sh
Hello World!
acs@740d1bdabef2:~$ /home/acs/test.sh
Hello World!
acs@740d1bdabef2:~$ ~/test.sh
Hello World!
acs@740d1bdabef2:~$ |
```

## 2.注释

### 单行注释

每行中 # 之后的内容为注释内容

```
1 # 这是一行注释
2
3 echo "Hello world" # 这也是注释
4
```

### 多行注释

格式:

```
1 :<<EOF
2 第一行注释
3 第二行注释
4 第三行注释
5 EOF
6
```

其中, EOF 可以换成其它任意字符。例如:

```
1 :<<ZS
2 注释改为ZS
3 第二行注释
4 第三行注释
5 ZS
6
```

### 3.变量

#### 定义变量

定义变量，不需要加 `$` 符号，例如：

```
1 name1 = 'jtx' #单引号定义字符串
2 name2 = "jtx" #双引号定义字符串
3 name3 = jtx #也可以不加引号，同样表示字符串
```

#### 使用变量


使用变量，需要加上 `$` 符号，或者 `${}` 符号。花括号是可选的，主要为了帮助解释器识别变量边界。

```
1 name=jtx
2 echo $name #输出jtx
3 echo ${name} #输出jtx
4 echo ${name}string #输出jtxstring
5
```

#### 只读变量

使用 `readonly` 或者 `declare` 可以将变量变为只读。

```
1 name=jtx
2 readonly name
3 declare -r name #两种写法都可以
4
5 name=aaa # 会报错，因为此时name只读
```



```
1 #!/bin/bash
2
3 name=jtx
4 readonly name
5
6 name = aaa
```

Terminal output:

```
acs@740d1bdabef2:~$ ls
homework main.cpp test.sh test_zs.sh testbl.sh
acs@740d1bdabef2:~$ ./testbl.sh
./testbl.sh: line 7: name: command not found
acs@740d1bdabef2:~$
```



```
1 #!/bin/bash
2
3 name=jtx
4
5 declare -r name
6
7 name = aaa
```

Terminal output:

```
acs@740d1bdabef2:~$ ls
homework main.cpp test.sh test_zs.sh testbl.sh
acs@740d1bdabef2:~$ ./testbl.sh
./testbl.sh: line 7: name: command not found
acs@740d1bdabef2:~$
```

#### 删除变量

`unset` 可以删除变量。

```
1 name=jtx
2 unset name
3 echo $name #输出空行，因为name变量已经被删除
```



```
1 #!/bin/bash
2
3 name=jtx
4 unset name
5 echo $name
```

Terminal output:

```
acs@740d1bdabef2:~$ ls
homework main.cpp test.sh test_zs.sh testbl.sh
acs@740d1bdabef2:~$ ./testbl.sh

```

## 变量类型

- 自定义变量(局部变量)
  - 子进程不能访问的变量
- 环境变量(全局变量)
  - 子进程可以访问的变量

### 1. 自定义变量改成环境变量:

```
1  #注意: 一个bash相当于一个子进程
2
3  acs@740d1bdabef2:~$ name=jtx  #定义name变量
4  acs@740d1bdabef2:~$ echo $name  #在当前bash下输出name的值
5  jtx                               #输出结果
6  acs@740d1bdabef2:~$ bash        #新建一个bash
7  acs@740d1bdabef2:~$ echo $name  #在新建的bash下输出name的值
8                                     #由于此时变量为局部变量, 故此时输出为空
9  acs@740d1bdabef2:~$ exit        #退出新建的bash
10 exit
11 acs@740d1bdabef2:~$ export name  #将name修改为环境变量
12 acs@740d1bdabef2:~$ bash        #新建一个bash
13 acs@740d1bdabef2:~$ echo $name  #在新建的bash中输出name的值
14 jtx                               #此时 name为环境变量, 故新的bash可以访问到
15 acs@740d1bdabef2:~$ exit
16 exit
```

### 2. 环境变量改为自定义变量:

```
1  #通过此例, 可以看到 declare 的用法
2  acs@740d1bdabef2:~$ export name=jtx  #定义全局变量
3  acs@740d1bdabef2:~$ bash            #新建一个子进程
4  acs@740d1bdabef2:~$ echo $name      #子进程可以访问到name
5  jtx
6  acs@740d1bdabef2:~$ exit            #退出当前子进程
7  exit
8  acs@740d1bdabef2:~$ declare +x name  #通过declare将name变为局部变量
9
9  acs@740d1bdabef2:~$ bash
10 acs@740d1bdabef2:~$ echo $name      #子进程无法访问到name
11
12 acs@740d1bdabef2:~$ exit
13 exit
14 acs@740d1bdabef2:~$ declare -x name  #通过declare 将name变为全局变量
15 acs@740d1bdabef2:~$ bash
16 acs@740d1bdabef2:~$ echo $name      #子进程又可以访问到name
17
17 jtx
18 acs@740d1bdabef2:~$ exit
19 exit
```

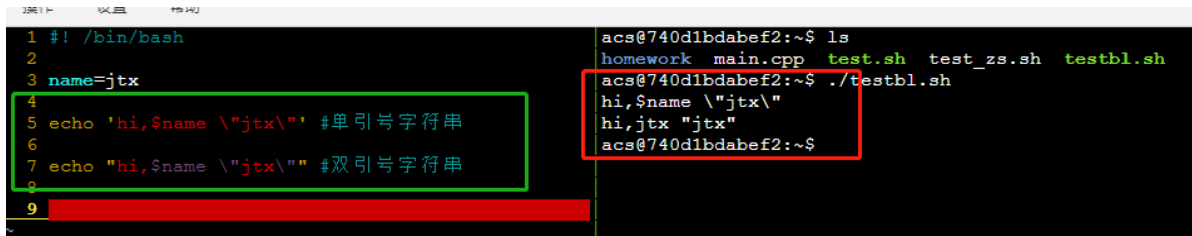
## 字符串

字符串可以使用单引号，也可以使用双引号，也可以不用引号。

单引号与双引号的区别：

- 单引号中的内容会原样输出，不会执行，不会获取变量的值
- 双引号中的内容可以执行、可以获取变量的值

```
1 name=jtx
2 echo 'hi, $name \'jtx\'' #单引号字符串，输出 hi,$name \'jtx\'
3 echo "hi, $name \'jtx\'" #双引号字符串，输出 hi,jtx "jtx"
```



```
1 #!/bin/bash
2
3 name=jtx
4
5 echo 'hi,$name \'jtx\'' #单引号字符串
6
7 echo "hi,$name \'jtx\'" #双引号字符串
8
9
```

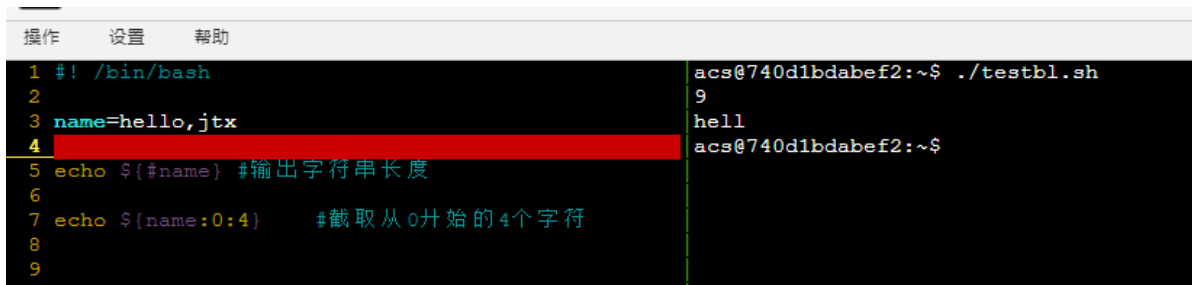
```
acs@740d1bdabef2:~$ ls
homework main.cpp test.sh test_zs.sh testbl.sh
acs@740d1bdabef2:~$ ./testbl.sh
hi,$name \'jtx\'
hi,jtx "jtx"
acs@740d1bdabef2:~$
```

获取字符串长度

```
1 name=jtx
2 echo ${#name} #输出3
```

截取子串

```
1 name="hello,jtx"
2 echo ${name:0:4} #截取从0开始的4个字符
```



```
1 #!/bin/bash
2
3 name=hello,jtx
4
5 echo ${#name} #输出字符串长度
6
7 echo ${name:0:4} #截取从0开始的4个字符
8
9
```

```
acs@740d1bdabef2:~$ ./testbl.sh
9
hell
acs@740d1bdabef2:~$
```

## 4.默认变量

### 文件参数变量

在执行 shell 脚本时，可以向脚本传递参数。

\$0 是文件名(包含路径)。例如：

```
1 | #!/bin/bash
2 |
3 | echo "文件名:"$0
4 | echo "第一个参数: "$1
5 | echo "第二个参数: "$2
6 | echo "第三个参数: "$3
7 | echo "第四个参数: "$4
8 |
```

然后执行该脚本：

```
1 | acs@740d1bdabef2:~$ ./test4.sh          #没有参数的情况下， $1,$2,$3,$4均为空
2 |
3 | ./test4.sh
4 |
5 |
6 |
7 | acs@740d1bdabef2:~$ ./test4.sh 1 2 3 4
8 | ./test4.sh
9 | 第二个参数: 1
10 | 第一个参数: 2
11 | 第一个参数: 3
12 | 第一个参数: 4
```

其它参数相关变量

参数	说明
<code>\$#</code>	代表w文件传入的参数个数，如上例中值为4
<code>\$*</code>	由所有参数构成的用空格隔开的字符串，如上例中值 <code>"\$1" "\$2" "\$3" "\$4"</code>
<code>@</code>	每个参数分别用双引号括起来的字符串，如上例中值为 <code>"\$1" "\$2" "\$3" "\$4"</code>
<code>\$\$_</code>	脚本当前运行的进程ID
<code>\$?</code>	上一条命令的退出状态(注意不是 <code>stdout</code> ，而是 <code>exit code</code> )。0表示正常退出，其他值表示错误
<code>\$(command)</code>	返回 <code>command</code> 这条命令的 <code>stdout</code> （可嵌套）
<code>command</code>	返回 <code>command</code> 这条命令的 <code>stdout</code> (不可嵌套)

5.数组

数组中可以存放多个不同类型的值，只支持一维数组，初始化不需要指明数组大小。

数组下标从0开始。

定义

数组用小括号表示，元素之间用空格隔开。例如：

```
1 | array=(1 abc "def" jtx)
```

也可以直接定义数组中某个元素的值：

```
1 array[0]=1
2 array[1]=abc
3 array[2]="def"
4 array[3]=jtx
```

### 读取数组中某个元素的值

格式：

```
1 ${array[index]}
```

例如：

```
1 array=(1 abc "def" jtx)
2 echo ${array[0]}
3 echo ${array[1]}
4 echo ${array[2]}
5 echo ${array[3]}
```

### 读取整个数组

格式：

```
1 ${array[@]} #第一种写法
2 ${array[*]} #第二种写法
```

例如：

```
1 array=(1 abc "def" jtx)
2
3 echo ${array[@]} #第一种读取整个数组的写法
4 echo ${array[*]} #第二种读取整个数组的写法
```

### 数组长度

类似于字符串

```
1 ${#array[@]} #第一种写法
2 ${#array[*]} #第二种写法
```

例如：

```
1 array=(1 abc "def" jtx)
2
3 echo ${#array[@]} #第一种写法
4 echo ${#array[*]} #第二种写法
```

```
操作 设置 帮助
acs@740d1bdabef2:~$ ls
EOF      test.sh  test6.sh
homework test4.sh test_zs.sh
main.cpp test5.sh testbl.sh
acs@740d1bdabef2:~$ ./test5.sh
读取数组中的每个元素：
1
abc
def
jtx
读取整个数组：
第一种方式:1 abc def jtx
第二种方式:1 abc def jtx
数组长度：
第一种方式，数组长度：4
第二种方式，数组长度：4
acs@740d1bdabef2:~$

1 #!/bin/bash
2
3 array=(1 abc "def" jtx)
4
5 echo "读取数组中的每个元素："
6 echo ${array[0]}
7 echo ${array[1]}
8 echo ${array[2]}
9 echo ${array[3]}
10
11
12 echo "读取整个数组："
13 echo "第一种方式:${array[*]}"
14 echo "第二种方式:${array[@]}"
15
16 echo "数组长度："
17 echo "第一种方式，数组长度:${#array[@]}"
18 echo "第二种方式，数组长度:${#array[*]}"
19
20
```

## 6. expr 命令

expr 命令用于求表达式的值，格式为：

```
1 | expr 表达式
```

表达式说明：

- 用空格隔开每一项
- 用反斜杠放在 shell 特定的字符前面(发现表达式运行错误时，可以试试转义)
- 对包含空格和其它特殊字符的字符串要用括号括起来
- expr 会在 stdout 中输出结果。如果为逻辑关系表达式，则结果为真，stdout 为1，否则为0。
- expr 的 exit code：如果为逻辑关系表达式，则结果为真，exit code 为0，否则为1。

### 字符串表达式

- length string

返回 string 的长度

- index string charset

charset 中任意单个字符在 string 中最前面的字符位置，下标从1开始。如果在 string 中完全不存在 charset 的字符，则返回0。

- substr string position length

返回 string 字符串从 position 开始，长度最大为 length 的子串。如果 position 或 length 为负数，0或非数值，则返回空字符串。

例如：

```
1 | str="Hello world!"
2
3 | echo `expr length "$str"` # 表达式要放在 `` 里面，表示执行改命令，输出12
4 | echo `expr index "$str" awd` #输出7，下标从1开始
5 | echo `expr substr "$str" 3 3` #输出llo
```



## 整数表达式

`expr` 支持普通的算术操作，算术表达式优先级低于字符串表达式，高于逻辑关系表达式。

- `+` `-`

加减运算。两端参数会转换为整数，如果转换失败则报错。

- `*` `/` `%`

乘，除，取模。两端参数会转换为整数，如果转换失败则报错。

- `()` 括号里面的内容优先运算，但需要用反斜杠转义。

例如：

```
1 a=3
2 b=4
3
4 echo `expr $a + $b` #输出7
5 echo `expr $a - $b` #输出-1
6 echo `expr $a \* $b` #输出12， *需要转义
7 echo `expr $a / $b` #输出0，整除
8 echo `expr $a % $b` #输出3
9 echo `expr \($a + 1\) \* \($b + 1\)` #输出20， 值为(a+1) * (b+1)
```

## 逻辑关系表达式

- `|`

如果第一个参数非空且非0，则返回第一个参数的值。否则，当第二个参数非空且非零时，返回第二个参数；否则返回0。如果第一个参数非空或非零，不会计算第二个参数

- `&`

如果两个参数都非空且非零，则返回第一个参数，否则返回0。如果第一个参数为空或为零时，不会计算第二个参数。

- `<` `<=` `=` `==` `!=` `>=` `>`

- 比较两端的参数，如果为 `true`，则返回1，否则返回0。`==` 是 `=` 的同义词。`expr` 首先尝试将两端转为整数，并做算术比较，如果转换失败，则按字符集排序规则做字符比较。

- `()`

括号里面的内容优先运算，但需要用反斜杠转义。

例如：

```
1 a=3
2 b=4
3
4 echo `expr $a \> $b` #输出0，需要转义
5 echo `expr $a '<' $b` #输出1，用''也可以
6 echo `expr $a '>=' $b` #输出0
7 echo `expr $a \<\ $b` #输出1
8
9 c=0
10 d=5
11
12 echo `expr $c \& $d` #输出0
13 echo `expr $a \& $b` #输出3
14 echo `expr $c \|| $d` #输出5
```

```
15 | echo `expr $a \| $b` #输出3
```

```
acs@740d1bdabef2:~$ ls
EOF      main.cpp  test4.sh  test_zs.sh
homework  test.sh   test5.sh  testbl.sh
acs@740d1bdabef2:~$ ./test5.sh
test5.sh
7
-1
12
0
3
20
0
3
5
3
acs@740d1bdabef2:~$

1 #!/bin/bash
2
3 echo "test5.sh"
4
5 a=3
6 b=4
7
8 echo `expr $a + $b` #输出7
9 echo `expr $a - $b` #输出-1
10 echo `expr $a \* $b` #输出12, *要转义
11 echo `expr $a / $b` #输出0, 整除
12 echo `expr $a % $b` #输出3, 取模
13
14 echo `expr \( $a + 1 \| \) \* \( $b + 1 \| \)` #输出20, (a+1)*(b+1)
15
16 c=0
17 d=5
18
19 echo `expr $c \% $d` #输出0
20 echo `expr $a \% $b` #输出3
21 echo `expr $c \| $d` #输出5
22 echo `expr $a \| $b` #输出3
23
24
```

## 7. read 命令

read 命令用于从标准输入中读取单行数据。当读到文件结束符时，exit code 为1，否则为0。

### 参数说明

- -p: 后面可以接提示信息
- -t: 后面跟秒数，定义输入字符的等待时间，超过等待时间后会自动忽略此命令，但 -t 后面的命令继续执行

例如：

```
1 | acs@740d1bdabef2:~$ read name #读入name的值
2 | hello jtx #输入name的值
3 | acs@740d1bdabef2:~$ echo $name
4 | hello jtx #输出name的值
5 | acs@740d1bdabef2:~$ read -p "what's your name?" -t 10 name #读入name的值，等待时
   | 间为10s, -p提示语
6 | what's your name?my name is jtx #输入name的值
7 | acs@740d1bdabef2:~$ echo $name #输出name的值
8 | my name is jtx #标准输出
9 | acs@740d1bdabef2:~$
```

```
acs@740d1bdabef2:~$ read name
hello jtx
acs@740d1bdabef2:~$ echo $name
hello jtx
acs@740d1bdabef2:~$ read -p "what's your name?" -t 10 name
what's your name?my name is jtx
acs@740d1bdabef2:~$ echo $name
my name is jtx
acs@740d1bdabef2:~$ |
```

## 8. echo 命令

echo 用于输出字符串。命令格式：

```
1 | echo string
```

## 显示普通字符串

```
1 echo "Hello jtx"
2 echo Hello jtx #可以直接输出想表达的字符串
```

## 显示转义字符

```
1 echo "\"Hello jtx\"" # 转义双引号
2 echo \"Hello jtx\" #省略外层双引号
3
4 #示例
5 acs@740d1bdabef2:~$ echo "\"Hello jtx\""
6 "Hello jtx"
7 acs@740d1bdabef2:~$ echo \"Hello jtx\"
8 "Hello jtx"
9
```

## 显示变量

```
1 name=jtx
2 echo "wo shi $name" #输出 wo shi jtx
3
4 #示例
5 acs@740d1bdabef2:~$ name=jtx
6 acs@740d1bdabef2:~$ echo "wo shi $name"
7 wo shi jtx #输出结果
```

## 显示换行

```
1 echo -e "Hi\n" # -e 开启转义
2 echo "jwjtx"
3
4 #示例
5 acs@740d1bdabef2:~$ echo -e "Hi \n jwjtx"
6 Hi #输出，换行
7
8 jwjtx
9
10 acs@740d1bdabef2:~$ echo "Hi \n jwjtx"
11 Hi \n jwjtx #输出，不换行
```

## 显示不换行

```
1 echo -e "Hi \c" # -e 开启转义 \c 不换行
2 echo "jwjtx"
3
4 #示例
5 acs@740d1bdabef2:~$ echo -e "Hi \c"
6 Hi acs@740d1bdabef2:~$ #输出 Hi 后，没有换行
```

## 显示结果定向至文件

```
1 | echo "Hello world" > out.txt    #将内容输出到out.txt中
```

## 原样输出字符串，不进行转义或取变量(用单引号)

```
1 | name=jtx
2 | echo '$name\"'
3 |
4 | #输出结果
5 | acs@740d1bdabef2:~$ name=jtx
6 | acs@740d1bdabef2:~$ echo '$name\"'
7 | $name\"          #输出结果
```

## 显示命令的执行结果

```
1 | echo `date`
2 |
3 | #输出结果
4 | acs@740d1bdabef2:~$ echo `date`
5 | Tue Dec 21 16:02:24 CST 2021
6 |
```

## 9.printf命令

printf 命令用于格式化输出，类似于 C++ 的 printf 函数。

默认不会在字符串末尾添加换行符。

命令格式：

```
1 | printf format-string [arguments...]
```

### 用法示例

脚本内容：

```
1 | printf "%10d.\n" 123    # 占10位，右对齐
2 | printf "%-10.2f.\n" 566.545532 #占10位，保留2位小数，左对齐
3 | printf "my name is %s\n" "jtx" #格式化输出字符串
4 | printf "%d * %d = %d\n" 2 3 `expr 2 \* 3` #表达式的值作为参数
5 |
6 | #输出样例
7 |      123.
8 | 566.55 .
9 | my name is jtx
10 | 2 \* 3 = 6
11 |
```

```
AC terminal
操作 设置 帮助

1 #!/bin/bash
2
3 echo "test8.sh"
4
5 printf "%10d.\n" 123      #占10位，右对齐
6 printf "%-10.2f, \n" 566.54532
7 printf "my name is %s \n" jtx      #格式化输出字符串
8 printf "%d \* %d = %d\n" 2 3 `expr 2 \* 3` #表达式的值作为参数
9
10

acs@740d1bdabef2:~$ ls
EOF homework main.cpp tes
acs@740d1bdabef2:~$ ./test8.sh
test8.sh
      123.
566.55
my name is jtx
2 \* 3 = 6
acs@740d1bdabef2:~$ |
```

## 10. test 命令与判断符号 []

### 逻辑运算符 && 和 ||

- && 表示与，|| 表示或
- 二者具有短路原则：
  - expr1 && expr2: 当 expr1 为假时，直接忽略 expr2
  - expr1 || expr2: 当 expr1 为真时，直接忽略 expr2
- 表达式的 exit code 为0，表示真；为非零，表示假。(与我们平常所见相反)

### test 命令

在命令行中输入 `man test`，可以查看 `test` 命令的用法。

`test` 命令用于判断文件类型，以及对变量做比较。

`test` 命令用 `exit code` 返回结果，而不是使用 `stdout`。0表示真，非0表示假。

例如：

```
1 test 2 -lt 3      # 2<3, 为真, 返回值为0
2 echo $?          #输出上个命令的返回值, 输出0
3
4 acs@740d1bdabef2:~$ test 2 -lt 3
5 acs@740d1bdabef2:~$ echo $?
6 0
```

```
1 acs@740d1bdabef2:~$ ls
2 EOF homework main.cpp test.sh test4.sh test5.sh test6.sh test8.sh
  test_zs.sh testbl.sh
3 acs@740d1bdabef2:~$ test -e test.sh && echo "exist" || echo "not exist" #判断
  test.sh是否存在, 存在输出"exist", 不存在输出"not exist"
4 exist
5 acs@740d1bdabef2:~$ test -e test1.sh && echo "exist" || echo "not exist" #同理
  判断test1.sh是否存在
6 not exist
```

### 文件类型判断

命令格式：

```
1 test -e filename      #判断文件是否存在
```

相关参数：

- `-e`：代表文件是否存在
- `-f`：判断是否为文件
- `-d`：判断是否为目录

## 文件权限判断

命令格式：

```
1 | test -r filename    #判断文件是否可读
```

相关参数：

- `-r`：文件是否可读
- `-w`：文件是否可写
- `-x`：文件是否可执行
- `-s`：是否为非空文件

## 整数间的比较

命令格式：

```
1 | test $a -eq $b    # a 是否等于 b
```

相关参数：

- `-eq`：两端是否相等
- `-ne`：两端是否不相等
- `-gt`：左边是否大于右边
- `-lt`：左边是否小于右边
- `-ge`：左边是否大于等于右边
- `-le`：左边是否小于等于右边

## 字符串比较

测试参数	代表意义
<code>test -z string</code>	判断string是否为空，如果为空，则返回 <code>true</code>
<code>test -n string</code>	判断string是否非空，如果非空，则返回 <code>true</code> (-n 可以省略)
<code>test str1 == str2</code>	判断 str1 是否等于 str2
<code>test str1 != str2</code>	判断 str1 是否不等于 str2

## 多重条件判定

命令格式：

```
1 | test -r filename -a -x filename
```

相关参数：

- `-a`：两条件是否同时成立
- `-o`：两条件是否至少成立一个

- `!`: 取反。

## 判断符号 `[]`

`[]` 和 `test` 用法几乎一模一样，更常用于 `if` 语句中。另外 `[]` 是 `[]` 的加强版，支持的特性更多。

例如：

```
1 [ 2 -lt 3 ] #2<3,为真, 返回值为0
2 echo $? #输出上个命令的返回值, 0
3
4 #样例
5 acs@740d1bdabef2:~$ [ 2 -lt 3 ]
6 acs@740d1bdabef2:~$ echo $?
7 0
```

```
1 acs@740d1bdabef2:~$ ls
2 EOF homework main.cpp test.sh test4.sh test5.sh test6.sh test8.sh
   test_zs.sh testbl.sh
3 acs@740d1bdabef2:~$ [ -e test.sh ] && echo "exist" || echo "not exist"
4 exist
5 acs@740d1bdabef2:~$ [ -e test1.sh ] && echo "exist" || echo "not exist"
6 not exist
```

注意：

- `[]` 内的每一项都要用空格隔开
- 中括号内的变量，最好用双引号括起来
- 中括号内的常数，最好用单或双引号括起来

例如：

```
1 name="hi jtx"
2 [ $name == "hi jtx" ] #错误, 等价于[ hi jtx == "hi jtx" ]
3 [ "$name" == "hi jtx" ] #正确
4
5 #示例
6 acs@740d1bdabef2:~$ name="hi jtx"
7 acs@740d1bdabef2:~$ [ $name == "hi jtx" ] #错误, 参数太多
8 -bash: [: too many arguments
9 acs@740d1bdabef2:~$ [ "$name" == "hi jtx" ] #正确, 0为真
10 acs@740d1bdabef2:~$ echo $?
11 0
```

```

acs@740d1bdabef2:~$ ls
EOF homework main.cpp test.sh test4.sh test5.sh test6.sh test8.sh test_zs.sh
acs@740d1bdabef2:~$ test -e test.sh && echo "exist" || echo "not exist"
exist
acs@740d1bdabef2:~$ test -e test1.sh && echo "exist" || echo "not exist"
not exist
acs@740d1bdabef2:~$ [ 2 -lt 3]
-bash: [: missing `]'
acs@740d1bdabef2:~$ [ 2 -lt 3 ]
acs@740d1bdabef2:~$ echo $?
0
acs@740d1bdabef2:~$ [ -e test.sh ] && echo "exist" || echo "not exist"
exist
acs@740d1bdabef2:~$ [ -e test1.sh ] && echo "exist" || echo "not exist"
not exist
acs@740d1bdabef2:~$
acs@740d1bdabef2:~$
acs@740d1bdabef2:~$ name="jtx"
acs@740d1bdabef2:~$ name="hi jtx"
acs@740d1bdabef2:~$ [ $name == "hi jtx" ]
-bash: [: too many arguments
acs@740d1bdabef2:~$ [ "$name" == "hi jtx" ]
acs@740d1bdabef2:~$ echo $?
0
acs@740d1bdabef2:~$ |

```

## 11.判断语句

### if...then 形式

类似于我们经常使用的 if-else 语句。

### 单层 if

命令格式:

```

1  if condition
2  then
3      语句1
4      语句2
5      ...
6  fi

```

示例:

```

1  a=3
2  b=4
3
4  if [ "$a" -lt "$b" ] && [ "$a" -gt 2 ]
5  then
6      echo ${a}in range
7  fi

```

输出结果:

```

1  3in range

```



```
acs@740d1bdabef2:~$ a=3
acs@740d1bdabef2:~$ b=4
acs@740d1bdabef2:~$ if [ "$a" -lt "$b" ] && [ "$a" -gt 2 ] ; then echo ${a}in range; fi
3in range
acs@740d1bdabef2:~$
```

## 单层 if-else

命令格式

```
1  if condition
2  then
3      语句1
4      语句2
5      ...
6  else
7      语句1
8      语句2
9      ...
10 fi
```

示例:

```
1  a=3
2  b=4
3
4  if ![ "$a" -lt "$b" ]
5  then
6      echo ${a} not lt ${b}
7  else
8      echo ${a} lt ${b}
9  fi
```

输出结果:

```
1  3 lt 4
```

```
AC Terminal
操作 设置 帮助

1 #!/bin/bash
2
3 #echo "testif.sh"
4
5 a=3
6 b=4
7
8 if ! [ "$a" -lt "$b" ]
9 then
10     echo ${a} not lt ${b}
11 else
12     echo ${a} lt ${b}
13 fi
14
15
16
~
~
~
~
```

```
acs@740d1bdabef2:~$ ls
EOF homework main.cpp test.sh tes
acs@740d1bdabef2:~$ ./testif.sh
3 lt 4
acs@740d1bdabef2:~$
```

## 多层 if-else-elif-else

命令格式:

```
1 if condition
2 then
3     语句1
4     语句2
5     ...
6 elif condition
7 then
8     语句1
9     语句2
10    ...
11 elif condition
12 then
13     语句1
14     语句2
15     ...
16 fi
```

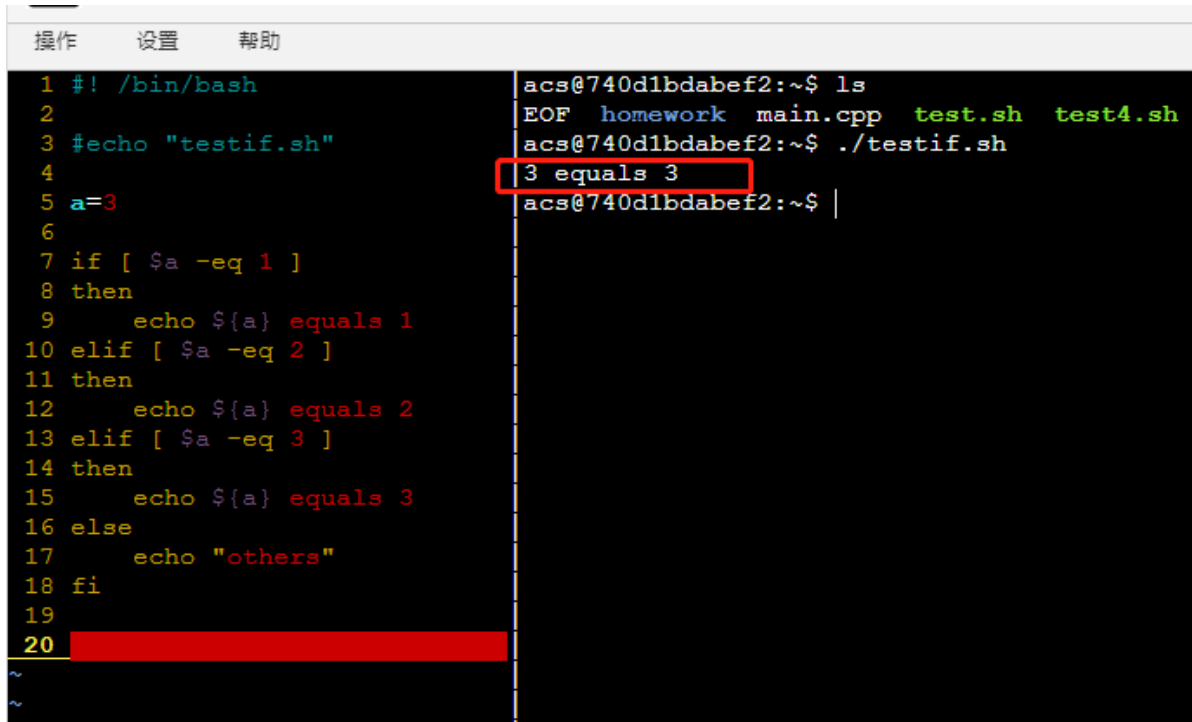
示例:

```
1 a=3
2
3 if [ $a -eq 1 ]
4 then
5     echo ${a} equals 1
6 elif [ $a -eq 2 ]
7 then
8     echo ${a} equals 2
9 elif [ $a -eq 3 ]
10 then
```

```
11     echo ${a} equals 3
12 else
13     echo others
14 fi
```

输出结果

```
1 3 equals 3
```



The image shows a terminal window with a dark background. The left pane displays the script content, and the right pane shows the execution output. The script defines a variable `a=3` and uses an `if-elif-then-fi` structure to print "3 equals 3". The output pane shows the command `./testif.sh` being executed, followed by the output `3 equals 3`, which is highlighted with a red rectangle.

```
操作 设置 帮助
1 #!/bin/bash
2 #echo "testif.sh"
3 a=3
4
5 if [ $a -eq 1 ]
6 then
7     echo ${a} equals 1
8 elif [ $a -eq 2 ]
9 then
10    echo ${a} equals 2
11 elif [ $a -eq 3 ]
12 then
13    echo ${a} equals 3
14 else
15    echo "others"
16 fi
17
18
19
20
```

```
acs@740d1bdabef2:~$ ls
EOF homework main.cpp test.sh test4.sh
acs@740d1bdabef2:~$ ./testif.sh
3 equals 3
acs@740d1bdabef2:~$ |
```

### case...esac 形式

类似于我们常用的 `switch` 语句。

命令格式：

```
1 case $变量名称 in
2     值1)
3         语句1
4         语句2
5         ...
6         ;; #类外于我们用的break语句
7     值2)
8         语句1
9         语句2
10        ...
11        ;;
12    *) #类似于我们用的 default
13        语句1
14        语句2
15        ...
16        ;;
17 esac
```

示例：

```

1  a=4
2
3  case $a in
4      1)
5          echo ${a} equals 1
6          ;;
7      2)
8          echo ${a} equals 2
9          ;;
10     3)
11         echo ${a} equals 3
12         ;;
13     *)
14         echo "others"
15 esac

```

输出结果

```
1 others
```

The screenshot shows a terminal window with a dark background. On the left, a script is being executed line by line, with line numbers 1 through 19. The script sets `a=5` and then enters a `case` statement. The right side of the terminal shows the command prompt `acs@740d1bdabef2:~$` followed by the execution of `ls`, `EOF homework main.cpp test`, and then `./testcas`. The output of the script, `others`, is displayed on the right and is highlighted with a red rectangular box.

## 12.循环语句

`for...in...do...done`

命令格式:

```

1  for var in var1 var2 var3
2  do
3      语句1
4      语句2
5      语句3
6      ...
7  done

```

示例1, 输出 a 2 cc, 每个元素一行:

```
1 for i in a 2 cc
2 do
3     echo $i
4 done
```

操作	设置	帮助
1	#!/bin/bash	acs@740d1bdabef2:~\$ ls
2		EOF homework main.cpp test.sh test4
3	for i in a 2 cc	acs@740d1bdabef2:~\$ ./testfor.sh
4	do	a
5	echo \$i	2
6	done	cc
7		acs@740d1bdabef2:~\$

示例2, 输出当前路径下的所有文件名, 每个文件名一行:

```
1 for file in `ls`
2 do
3     echo $file
4 done
```

1	#!/bin/bash	acs@740d1bdabef2:~\$ ls
2		EOF homework main.cpp test.sh test4.sh tes
3	:<<EOF	acs@740d1bdabef2:~\$ ./testfor.sh
4	for i in a 2 cc	EOF
5	do	homework
6	echo \$i	main.cpp
7	done	test.sh
8	EOF	test4.sh
9		test5.sh
10	for file in `ls`	test6.sh
11	do	test8.sh
12	echo \$file	test_zs.sh
13	done	testbl.sh
14		testcase.sh
		testfor.sh
		testif.sh
		acs@740d1bdabef2:~\$

示例3, 输出 1-10

```
1 for i in $(seq 1 10)
2 do
3     echo $i
4 done
```

操作	设置	帮助
1	#!/bin/bash	acs@740d1bdabef2:~\$ ls
2		EOF homework main.cpp test.s
3	for i in \$(seq 1 10)	acs@740d1bdabef2:~\$ ./testfor.s
4	do	1
5	echo \$i	2
6	done	3
7		4
8		5
~		6
~		7
~		8
~		9
~		10
~		acs@740d1bdabef2:~\$

示例4, 使用 {1..10} 或 {a..z}

```
1 for i in {1..10}
2 do
3     echo $i
4 done
```

操作	设置	帮助
1	#!/bin/bash	acs@740d1bdabef2:~\$ ls
2		EOF homework main.cpp test.sh test
3	for i in {a..z}	acs@740d1bdabef2:~\$ ./testfor.sh
4	do	a
5	echo \$i	b
6	done	c
7		d
8		e
~		f
~		g
~		h
~		i
~		j
~		k
~		l
~		m
~		n
~		o
~		p
~		q
~		r
~		s
~		t
~		u
~		v
~		w
~		x
~		y
~		z
~		acs@740d1bdabef2:~\$

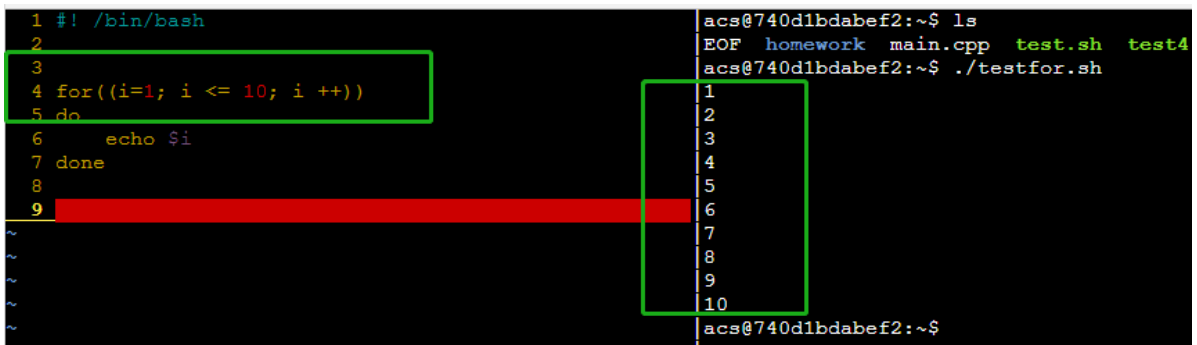
`for((...;...;...)) do...done`

命令格式:

```
1  for((表达式; 条件; 表达式))
2  do
3      语句1
4      语句2
5      ...
6  done
```

示例: 输出1-10, 每个数占一行

```
1  for ((i = 1; i <= 10; i ++))
2  do
3      echo $i
4  done
```



The screenshot shows a terminal window with a script being executed. The script content is highlighted with a green box on the left: `#!/bin/bash`, `for ((i=1; i <= 10; i ++))`, `do`, `echo $i`, `done`. The output on the right shows the numbers 1 through 10, each on a new line, also highlighted with a green box. The prompt indicates the user is at `acs@740d1bdabef2:~$`.

`while...do...done` 循环

命令格式:

```
1  while condition
2  do
3      语句1
4      语句2
5      ...
6  done
```

示例, 文件结束符为 `Ctrl+d`, 输入文件结束符后 `read` 指令返回 `false`。

```
1  while read name
2  do
3      echo $name
4  done
```

```
操作 设置 帮助
1 #!/bin/bash
2
3
4 while read name
5 do
6     echo $name
7 done
8
9
~
~
~
```

```
acs@740d1bdabef2:~$ ls
EOF homework main.cpp test.sh test4
acs@740d1bdabef2:~$ ./testfor.sh
hi
hi
jtx
jtx
^C
acs@740d1bdabef2:~$
```

## until...do...done 循环

当条件为真时结束。

命令格式：

```
1 until condition
2 do
3     语句1
4     语句2
5     ...
6 done
```

示例，当用户输入 `yes` 或者 `YES` 时结束，否则一直等待读入。

```
1 until [ "${word}" == "yes" ] || [ "${word}" == "YES" ]
2 do
3     read -p "input yes/YES to stop:" word
4 done
```

```
操作 设置 帮助
1 #!/bin/bash
2
3
4 until [ "${word}" == "yes" ] || [ "${word}" == "YES" ]
5 do
6     read -p "input yes/YES to stop: " word
7 done
8
9
10
~
~
~
```

```
acs@740d1bdabef2:~$ ls
EOF homework main.cpp test.sh test4.s
acs@740d1bdabef2:~$ ./testfor.sh
input yes/YES to stop: hi
input yes/YES to stop: jtx
input yes/YES to stop: nihaoa
input yes/YES to stop: yes
acs@740d1bdabef2:~$
```

## break 命令

跳出当前一层循环，`shell` 里面的 `break` 不能跳出 `case` 语句。

示例

```
1 while read name
2 do
3     for ((i=1;i<=10;i++))
4     do
5         case $i in
6             8)
7                 break
8                 ;;
9             *)
10                echo $i
11                ;;
12        esac
```



```
13     done
14 done
```

该示例每读入非 EOF 的字符串，会输出 1-7。（因为在8的时候，break就跳出了循环）

该程序可以输入 Ctrl+d 文件结束符来结束，也可以直接用 Ctrl+c 杀掉该进程。

### continue 命令

跳出当前循环。

示例：

```
1  #输出1-10中的奇数
2
3  for ((i=1;i<=10;i++))
4  do
5      if [ `expr $i % 2` -eq 0 ]
6      then
7          continue
8      fi
9          echo $i
10 done
```



```
AC Terminal
操作 设置 帮助
1 #!/bin/bash
2
3
4 for ((i = 1; i <= 10; i ++))
5 do
6     if [ `expr $i % 2` -eq 0 ]
7     then
8         continue
9     fi
10    echo $i
11 done
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588

```

```

1 [function] func_name() {    # function关键字可以省略
2     语句1
3     语句2
4     ...
5 }

```

## 不获取 return 值和 stdout 值

示例

```

1 func() {
2     name=jtx
3     echo "hi, $name"
4 }
5
6 #调用func
7 func

```

输出结果:

```

[?] /bin/bash
echo "funtest"

func(){
    name=jtx
    echo "hi, $name"
}

func
~
~
~
~

[root@VM-16-17-centos learn]# ls
funtest.sh
[root@VM-16-17-centos learn]# ./funtest.sh
funtest
hi, jtx
[root@VM-16-17-centos learn]#

```

## 获取 return 值和 stdout 值

不写 return 时, 默认 return 0。

示例

```

1 func(){
2     name=jtx
3     echo "hi,$name"
4
5     return 123
6 }
7
8 output=$(func)
9 ret=$?
10
11 echo "output = $output"
12 echo "return = $ret"
13

```

```
#!/bin/bash

func(){
    name=jtx
    echo "hi,$name"

    return 123
}

output=$(func)
ret=$?

echo "output = $output"
echo "return = $ret"
}
```

```
[root@VM-16-17-centos learn]# ls
funtest.sh
[root@VM-16-17-centos learn]# ./funtest.sh
output = hi,jtx
return = 123
[root@VM-16-17-centos learn]#
```

## 函数的输入参数

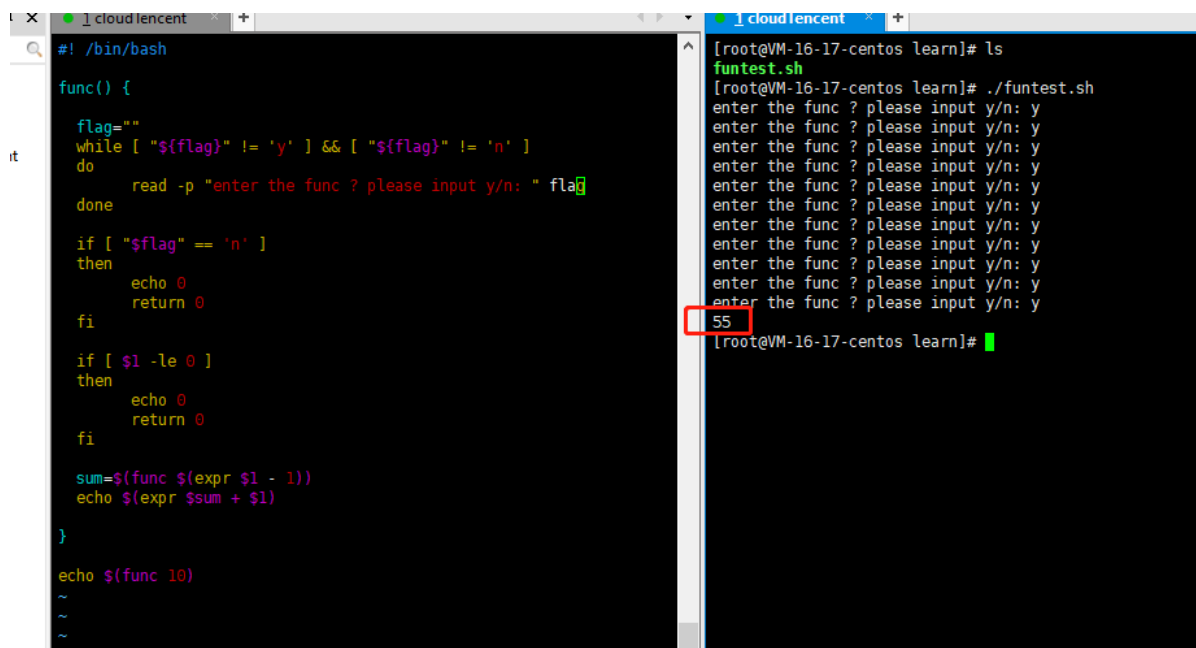
在函数内，`$1` 表示第一个输入参数，`$2` 表示第二个输入参数，依次类推。

函数内的 `$0` 仍然是文件名，而不是函数名。

示例：

```
1 func() {      #递归计算 $1 + ($1 - 1) + ($2 - 2) + ... + 0
2     flag=""
3     while [ "${flag}" != "y" ] && [ "${flag}" != "n" ]
4     do
5         read -p "enter the func($1) ? please input y/n: " word
6     done
7
8     if [ "${flag}" == 'n' ]
9     then
10        echo 0
11        return 0
12    fi
13
14    if [ $1 -le 0 ]
15    then
16        echo 0
17        return 0
18    fi
19
20    sum=$(func $(expr $1 - 1))
21    echo $(expr $sum + $1)
22
23 }
24
25 echo $(func 10)
```

输出结果：55



```
#!/bin/bash
func() {
    flag=""
    while [ "${flag}" != 'y' ] && [ "${flag}" != 'n' ]
    do
        read -p "enter the func ? please input y/n: " flag
    done
    if [ "$flag" == 'n' ]
    then
        echo 0
        return 0
    fi
    if [ $1 -le 0 ]
    then
        echo 0
        return 0
    fi
    sum=$(func $(expr $1 - 1))
    echo $(expr $sum + $1)
}
echo $(func 10)
```

```
[root@VM-16-17-centos learn]# ls
funtest.sh
[root@VM-16-17-centos learn]# ./funtest.sh
enter the func ? please input y/n: y
enter the func ? please input y/n: y
enter the func ? please input y/n: y
enter the func ? please input y/n: y
enter the func ? please input y/n: y
enter the func ? please input y/n: y
enter the func ? please input y/n: y
enter the func ? please input y/n: y
enter the func ? please input y/n: y
enter the func ? please input y/n: y
55
[root@VM-16-17-centos learn]#
```

## 函数内的局部变量

可以在函数内定义局部变量，作用范围仅在当前函数内。

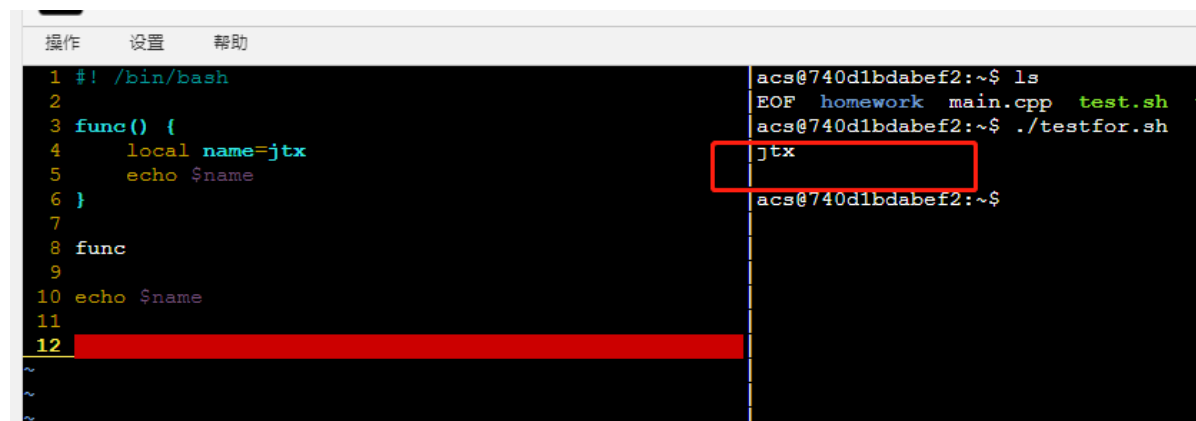
可以在递归函数中定义局部变量。

命令格式：

```
1 local 变量名=变量值
```

例如：

```
1  #!/bin/bash
2
3  func() {
4      local name=jtx
5      echo $name
6  }
7
8  func
9  echo $name
10
11 #输出结果中，第一行的jtx为函数内输出，第二行空位函数外调用找不到此值
```



```
操作 设置 帮助
1  #!/bin/bash
2
3  func() {
4      local name=jtx
5      echo $name
6  }
7
8  func
9
10 echo $name
11
12
```

```
acs@740d1bdabef2:~$ ls
EOF homework main.cpp test.sh
acs@740d1bdabef2:~$ ./testfor.sh
jtx
acs@740d1bdabef2:~$
```

## 14. exit 命令

exit 命令用来退出当前 shell 进程，并返回一个退出状态；使用 \$? 可以接收这个退出状态。

exit 命令可以接受一个整数值作为参数，代表退出状态。如果不指定，默认状态值是0。

exit 退出状态只能是一个介于 0~255 之间的整数，其中只有 0 表示成功，其他值都表示失败。

示例：

```
1  #! /bin/bash
2
3  if [ $# -ne 1 ] #如果传入参数个数等于1，则正常退出
4  then
5      echo "exit normal"
6      exit 1
7  else
8      echo "exit not normal"
9      exit 0
10 fi
11
```



```
操作  设置  帮助
1 #! /bin/bash
2
3 if [ $# -ne 1 ] #如果传入参数个数等于1，则正常退出
4 then
5     echo "exit not normal"
6     exit 1
7 else
8     echo "exit normal"
9     exit 0
10 fi
11
12
acs@740d1bdabef2:~$ ls
EOF      main.cpp  test4.sh  test6.sh  test8.sh
homework test.sh  test5.sh  test8.sh  test
acs@740d1bdabef2:~$ ./testExit.sh test
exit normal
acs@740d1bdabef2:~$ echo $?
0
acs@740d1bdabef2:~$ ./testExit.sh
exit not normal
acs@740d1bdabef2:~$ echo $?
1
acs@740d1bdabef2:~$
```

## 15. 文件重定向

每个进程默认打开3个文件描述符：

- `stdin` 标准输入，从命令行读取数据，文件描述符为0
- `stdout` 标准输出，向命令行输出数据，文件描述符为1
- `stderr` 标准错误输出，向命令行输出数据，文件描述符为2

可以用文件重定向将这三个文件重定向到其他文件中。

---

### 重定向命令表

命令	说明
<code>command &gt; file</code>	将 <code>stdout</code> 重定向到 <code>file</code> 中
<code>command &lt; file</code>	将 <code>stdin</code> 重定向到 <code>file</code> 中
<code>command &gt;&gt; file</code>	将 <code>stdout</code> 以追加方式重定向到 <code>file</code> 中
<code>command n&gt; file</code>	将文件描述符 <code>n</code> 重定向到 <code>file</code> 中
<code>command n&gt;&gt; file</code>	将文件描述符 <code>n</code> 以追加的方式重定向到 <code>file</code> 中

## 输入和输出重定向

```

1 echo -e "hi,\c" > out.txt #将stdout重定向到 out.txt中
2 echo "hi,my name is jtx" >> out.txt #将字符串追加到out.txt中
3
4 read str < out.txt #从out.txt中读取字符串
5
6 echo $str

```

```

acs@740d1bdabef2:~/redirection$ echo -e "hi,\c" > out.txt
acs@740d1bdabef2:~/redirection$ is
out.txt test.sh
acs@740d1bdabef2:~/redirection$ cat out.txt
hi,acs@740d1bdabef2:~/redirection$ echo "my name is jtx" >> out.txt
acs@740d1bdabef2:~/redirection$ cat out.txt
hi,my name is jtx
acs@740d1bdabef2:~/redirection$
acs@740d1bdabef2:~/redirection$ read str < out.txt
acs@740d1bdabef2:~/redirection$ echo $str
hi,my name is jtx
acs@740d1bdabef2:~/redirection$

```

## 同时重定向 `stdin` 和 `stdout`

```

1 #! /bin/bash
2
3 read a
4 read b
5
6 echo $(expr $a + $b)

```

`touch` 一个 `input.txt`，里面的内容为：

```

1 20
2 30

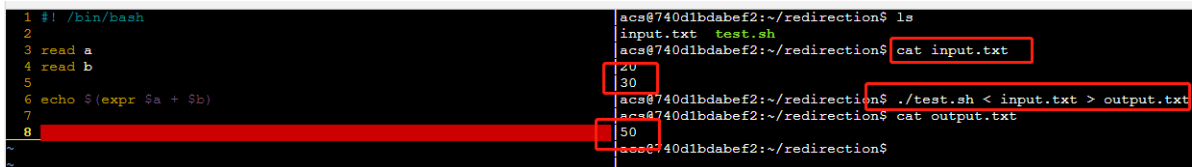
```

执行命令：

```

1 acs@740d1bdabef2:~/redirection$ ls
2 input.txt test.sh
3 acs@740d1bdabef2:~/redirection$ cat input.txt #查看input.txt内容
4 20
5 30
6 acs@740d1bdabef2:~/redirection$ ./test.sh < input.txt > output.txt
#input.txt内容作为输入，执行结果输出到output.txt中
7 acs@740d1bdabef2:~/redirection$ cat output.txt #查看output.txt
8 50

```



```

1 #!/bin/bash
2
3 read a
4 read b
5
6 echo $(expr $a + $b)
7
8

```

```

acs@740d1bdabef2:~/redirection$ ls
input.txt test.sh
acs@740d1bdabef2:~/redirection$ cat input.txt
20
30
acs@740d1bdabef2:~/redirection$ ./test.sh < input.txt > output.txt
acs@740d1bdabef2:~/redirection$ cat output.txt
50
acs@740d1bdabef2:~/redirection$

```

## 16.引入外部脚本

引入外部脚本类似我们 c/c++ 中的 `include`，类似 java 中的 `import`，bash 也可以引入其他文件中的代码。

语法格式：

```

1 . filename #一个点 空格 文件名
2
3 source filename #第二种方法

```

示例：

touch 第一个文件 `file1.sh`，内容：

```

1 #!/bin/bash
2
3 firstFile=firstName #创建变量firstFile

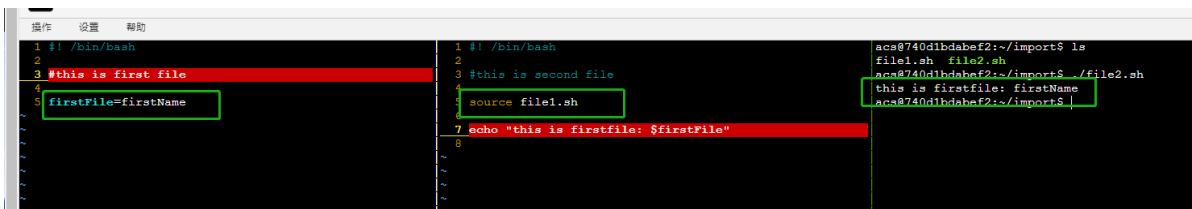
```

touch 第二个文件 `file2.sh`，内容：

```

1 #!/bin/bash
2
3 source file1.sh # 或者 . file1.sh
4
5 echo "this is firstFile: $name"

```



```

1 #!/bin/bash
2
3 #this is first file
4
5 firstFile=firstName

```

```

1 #!/bin/bash
2
3 #this is second file
4
5 source file1.sh
6
7 echo "this is firstfile: $firstFile"
8

```

```

acs@740d1bdabef2:~/import$ ls
file1.sh file2.sh
acs@740d1bdabef2:~/import$ ./file2.sh
this is firstfile: firstName
acs@740d1bdabef2:~/import$

```

## Shell 常用语法完结

到这里，关于 Linux 基础中篇幅最大的 Shell 常用语法就结束了。内容相对来说不算太多，也不算太少，用心即可学好。理论是一方面，最重要的还是实践，关于命令这个东西，我个人觉得不需要背，用的次数多了便记住了，对于这部分也是这样，我们通过学习理论知识，对整个知识框架有了了解，那么接下来就是把理论运用于实践当中，熟能生巧！