# 变量

## 定义变量

|  |
| --- |
| variableName="value"  如：  myUrl="http://see.xidian.edu.cn/cpp/linux/"  myNum=100 |

命令规则：

* 首个字符必须为字母（a-z，A-Z）。
* 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）。

注意，变量名和等号之间不能有空格

## 使用变量

|  |
| --- |
| your\_name="mozhiyan"  echo $your\_name  echo ${your\_name} |

推荐给所有变量加上花括号，这是个好的编程习惯

## 删除变量

|  |
| --- |
| unset variable\_name |

## 特殊变量

|  |  |
| --- | --- |
| 特殊变量列表 | |
| **变量** | **含义** |
| $0 | 当前脚本的文件名 |
| $n | 传递给脚本或函数的参数。n 是一个数字，表示第几个参数。例如，第一个参数是$1，第二个参数是$2。 |
| $# | 传递给脚本或函数的参数个数。 |
| $\* | 传递给脚本或函数的所有参数。 |
| $@ | 传递给脚本或函数的所有参数。被双引号(" ")包含时，与 $\* 稍有不同，下面将会讲到。 |
| $? | 上个命令的退出状态，或函数的返回值。 |
| $$ | 当前Shell进程ID。对于 Shell 脚本，就是这些脚本所在的进程ID。 |

## 命令替换

|  |
| --- |
| `command`  如：  DATE=`date`  echo "Date is $DATE" |

注意是反引号，不是单引号，这个键位于 Esc 键下方

## 变量替换

|  |  |
| --- | --- |
| **形式** | **说明** |
| ${var} | 变量本来的值 |
| ${var:-word} | 如果变量 var 为空或已被删除(unset)，那么返回 word，但不改变 var 的值。 |
| ${var:=word} | 如果变量 var 为空或已被删除(unset)，那么返回 word，并将 var 的值设置为 word。 |
| ${var:?message} | 如果变量 var 为空或已被删除(unset)，那么将消息 message 送到标准错误输出，可以用来检测变量 var 是否可以被正常赋值。 若此替换出现在Shell脚本中，那么脚本将停止运行。 |
| ${var:+word} | 如果变量 var 被定义，那么返回 word，但不改变 var 的值。 |

# expr

|  |
| --- |
| val=`expr 2 + 2`  echo "Total value : $val" |

**两点注意：**

* 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；
* 完整的表达式要被 ` ` 包含，注意这个字符不是常用的单引号，在 Esc 键下边。

# 运算符

## 算术运算符列表

|  |  |  |
| --- | --- | --- |
| **算术运算符列表** | | |
| **运算符** | **说明** | **举例** |
| + | 加法 | `expr $a + $b` 结果为 30。 |
| - | 减法 | `expr $a - $b` 结果为 10。 |
| \* | 乘法 | `expr $a \\* $b` 结果为  200。 |
| / | 除法 | `expr $b / $a` 结果为 2。 |
| % | 取余 | `expr $b % $a` 结果为 0。 |
| = | 赋值 | a=$b 将把变量 b 的值赋给 a。 |
| == | 相等。用于比较两个数字，相同则返回 true。 | [ $a == $b ] 返回 false。 |
| != | 不相等。用于比较两个数字，不相同则返回 true。 | [ $a != $b ] 返回 true。 |

## 关系运算符列表

|  |  |  |
| --- | --- | --- |
| **关系运算符列表** | | |
| **运算符** | **说明** | **举例** |
| -eq | 检测两个数是否相等，相等返回 true。 | [ $a -eq $b ] 返回 true。 |
| -ne | 检测两个数是否相等，不相等返回 true。 | [ $a -ne $b ] 返回 true。 |
| -gt | 检测左边的数是否大于右边的，如果是，则返回 true。 | [ $a -gt $b ] 返回 false。 |
| -lt | 检测左边的数是否小于右边的，如果是，则返回 true。 | [ $a -lt $b ] 返回 true。 |
| -ge | 检测左边的数是否大等于右边的，如果是，则返回 true。 | [ $a -ge $b ] 返回 false。 |
| -le | 检测左边的数是否小于等于右边的，如果是，则返回 true。 | [ $a -le $b ] 返回 true。 |

## 布尔运算符列表

|  |  |  |
| --- | --- | --- |
| **布尔运算符列表** | | |
| **运算符** | **说明** | **举例** |
| ! | 非运算，表达式为 true 则返回 false，否则返回 true。 | [ ! false ] 返回 true。 |
| -o | 或运算，有一个表达式为 true 则返回 true。 | [ $a -lt 20 -o $b -gt 100 ] 返回 true。 |
| -a | 与运算，两个表达式都为 true 才返回 true。 | [ $a -lt 20 -a $b -gt 100 ] 返回 false。 |

## 字符串运算符列表

|  |  |  |
| --- | --- | --- |
| **字符串运算符列表** | | |
| **运算符** | **说明** | **举例** |
| = | 检测两个字符串是否相等，相等返回 true。 | [ $a = $b ] 返回 false。 |
| != | 检测两个字符串是否相等，不相等返回 true。 | [ $a != $b ] 返回 true。 |
| -z | 检测字符串长度是否为0，为0返回 true。 | [ -z $a ] 返回 false。 |
| -n | 检测字符串长度是否为0，不为0返回 true。 | [ -z $a ] 返回 true。 |
| str | 检测字符串是否为空，不为空返回 true。 | [ $a ] 返回 true。 |

## 文件测试运算符列表

|  |  |  |
| --- | --- | --- |
| **文件测试运算符列表** | | |
| **操作符** | **说明** | **举例** |
| **-b file** | **检测文件是否是块设备文件，如果是，则返回 true。** | **[ -b $file ] 返回 false。** |
| **-c file** | **检测文件是否是字符设备文件，如果是，则返回 true。** | **[ -b $file ] 返回 false。** |
| **-d file** | **检测文件是否是目录，如果是，则返回 true。** | **[ -d $file ] 返回 false。** |
| **-f file** | **检测文件是否是普通文件（既不是目录，也不是设备文件），如果是，则返回 true。** | **[ -f $file ] 返回 true。** |
| **-g file** | **检测文件是否设置了 SGID 位，如果是，则返回 true。** | **[ -g $file ] 返回 false。** |
| **-k file** | **检测文件是否设置了粘着位(Sticky Bit)，如果是，则返回 true。** | **[ -k $file ] 返回 false。** |
| **-p file** | **检测文件是否是具名管道，如果是，则返回 true。** | **[ -p $file ] 返回 false。** |
| **-u file** | **检测文件是否设置了 SUID 位，如果是，则返回 true。** | **[ -u $file ] 返回 false。** |
| **-r file** | **检测文件是否可读，如果是，则返回 true。** | **[ -r $file ] 返回 true。** |
| **-w file** | **检测文件是否可写，如果是，则返回 true。** | **[ -w $file ] 返回 true。** |
| **-x file** | **检测文件是否可执行，如果是，则返回 true。** | **[ -x $file ] 返回 true。** |
| **-s file** | **检测文件是否为空（文件大小是否大于0），不为空返回 true。** | **[ -s $file ] 返回 true。** |
| **-e file** | **检测文件（包括目录）是否存在，如果是，则返回 true。** | **[ -e $file ] 返回 true。** |

# 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` |

# 数组

## 定义数组

|  |
| --- |
| 方式一：  array\_name=(value0 value1 value2 value3)  方式二：  array\_name=(  value0  value1  value2  value3  )  方式三：  array\_name[0]=value0  array\_name[1]=value1  array\_name[2]=value2 |

## 读取数组

|  |
| --- |
| #读取一个  valuen=${array\_name[2]}  #读取所有  ${array\_name[\*]}  ${array\_name[@]} |

## 读取数组长度

|  |
| --- |
| # 取得数组元素的个数  length=${#array\_name[@]}  length=${#array\_name[\*]}  # 取得数组单个元素的长度  lengthn=${#array\_name[n]} |

# 分支

## if else

### if ... else 语句

|  |
| --- |
| if [ $a == $b ]  then  echo "a is equal to b"  fi |

### if ... else ... fi 语句

|  |
| --- |
| if [ $a == $b ]  then  echo "a is equal to b"  else  echo "a is not equal to b"  fi |

### if ... elif ... fi 语句

|  |
| --- |
| if [ $a == $b ]  then  echo "a is equal to b"  elif [ $a -gt $b ]  then  echo "a is greater than b"  elif [ $a -lt $b ]  then  echo "a is less than b"  else  echo "None of the condition met"  fi |

## case

|  |
| --- |
| 1. echo 'Input a number between 1 to 4' 2. echo 'Your number is:\c' 3. **read** aNum 4. **case** $aNum **in** 5. 1) echo 'You select 1' 6. ;; 7. 2) echo 'You select 2' 8. ;; 9. 3) echo 'You select 3' 10. ;; 11. 4) echo 'You select 4' 12. ;; 13. \*) echo 'You do not select a number between 1 to 4' 14. ;; 15. **esac** |

# 循环

循环中也有break和continue两个关键字，含义和java一样。

## for

|  |
| --- |
| 1. 顺序输出当前列表中的数字 2. **for** loop **in** 1 2 3 4 5 3. **do** 4. echo "The value is: $loop" 5. **done** |

|  |
| --- |
| 1. 顺序输出字符串中的字符 2. **for** str **in** 'This is a string' 3. **do** 4. echo $str 5. **done** |

|  |
| --- |
| 1. 显示主目录下以 .bash 开头的文件 2. **for** FILE **in** $HOME/.bash\* 3. **do** 4. echo $FILE 5. **done** |

## while

### 普通循环

|  |
| --- |
| 1. COUNTER=0 2. **while** [ $COUNTER -lt 5 ] 3. **do** 4. COUNTER='expr $COUNTER+1' 5. echo $COUNTER 6. **done** |

### 读取键盘信息

|  |
| --- |
| 1. echo 'type <CTRL-D> to terminate' 2. echo -n 'enter your most liked film: ' 3. **while** **read** FILM 4. **do** 5. echo "Yeah! great film the $FILM" 6. **done** |

## until

|  |
| --- |
| 1. a=0 2. **until** [ ! $a -lt 10 ] 3. **do** 4. echo $a 5. a=`expr $a + 1` 6. **done** |

# 函数

## 定义与调用（无返回值）

|  |
| --- |
| funSayHello(){  echo "hello lzg!"  }  funSayHello |

## 定义与调用（有返回值）

|  |
| --- |
| funSayHello(){  echo "hello lzg!"  return 0;  }  funSayHello  ret=$?  echo $ret |

## 定义与调用（有参数）

|  |
| --- |
| 1. #!/bin/bash 2. funWithParam(){ 3. echo "The value of the first parameter is $1 !" 4. echo "The value of the second parameter is $2 !" 5. echo "The value of the tenth parameter is $10 !" 6. echo "The value of the tenth parameter is ${10} !" 7. echo "The value of the eleventh parameter is ${11} !" 8. echo "The amount of the parameters is $# !" # 参数个数 9. echo "The string of the parameters is $\* !" # 传递给函数的所有参数 10. } 11. funWithParam 1 2 3 4 5 6 7 8 9 34 73 |

**注意：**$10 不能获取第十个参数，获取第十个参数需要${10}。当n>=10时，需要使用${n}来获取参数。

# 重定向

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。

|  |  |
| --- | --- |
| **全部可用的重定向命令列表** | |
| **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。 |

# /dev/null

/dev/null 是一个特殊的文件，写入到它的内容都会被丢弃；如果尝试从该文件读取内容，那么什么也读不到。但是 /dev/null 文件非常有用，将命令的输出重定向到它，会起到”禁止输出“的效果。

* 屏蔽屏幕上输出结果（屏蔽stdout）：

|  |
| --- |
| command > /dev/null |

* 屏蔽 stdout 和 stderr

|  |
| --- |
| command > /dev/null 2>&1 |

# 引用其它脚本文件

使用 . filename 或 source filename

* test1.sh

|  |
| --- |
| var1='hello world' |

* test.sh(引入test1.sh，并使用其中的变量)

|  |
| --- |
| source ./test1.sh  funSayHello(){  echo $var1  echo "hello lzg!"  return 0;  }  funSayHello  ret=$?  echo $ret |

**注意：被包含脚本不需要有执行权限。**