

## 1、case...in分支语句

### 1.1 语法格式

### 1.2 练习题

## 2、select...in选择分支语句

### 2.1 语法格式

### 2.2 参考案例

## 3、for循环

### 3.1 for循环的语法格式

### 3.2 for循环的案例

## 4、while循环

### 4.1 while循环的语法格式

### 4.2 参考案例

## 5、break和continue

### 5.1 break关键字

### 5.2 continue关键字

## 6、shell函数

### 6.1 shell函数定义的格式

### 6.2 shell函数的调用

### 6.3 shell函数中获取函数传递的实参值

### 6.4 shell函数的返回值

### 6.5 shell函数的案例

### 6.6 main函数返回值的使用

# 1、case...in分支语句

## 1.1 语法格式

```
1  1. 语法格式
2      case $变量名/常量表达式 in      ---> 等价于: switch(常量表达式)
3          字符串常量1)                ---> 等价于 : case 常量:
4              shell语句1              ---> 等价于: C语句块
5              ;;                      ---> 等价于: break;
6          字符串常量2)
7              shell语句2
8              ;;
9          字符串常量3)
10             shell语句3
11             ;;
12             .....
13             *)                      ---> 等价于: default:
14             shell语句n
15             ;;
16      esac
17
18  2. 字符串常量表达式的写法:
19      a)    ---> 字符a
20      1)    ---> 数字1
21      yes|Yes|YES|y) ---> 字符串yes, Yes, YES, y中的任意一个
```

```
22 [0-9]) ---> 数组0-9中的任意一个
23 [a-zA-Z]) ---> a-z或者A-Z中的任意一个字符
24 NO) ---> 字符串NO
25 *) ---> 任意字符串
26 1|3|5|7|9|12) ---> 数字1, 3, 5, 7, 9, 12中的任意一个
27
28 3. 使用的注意事项
29 在case...in中除了最后一个分支的两个分号可以省略不写,
30 前边的所有的分支中的两个分号都不可以省略不写。
```

## 1.2 练习题

```
1 练习题1: 成绩的分类
2
3 #!/bin/bash
4 # your code
5
6 read -p "请输入成绩, 范围为0-100 > " score
7 if [ \(" $score" -lt 0 \) -o \(" $score" -gt 100 \) ]
8 then
9     echo "输入成绩不合理, 范围为0-100"
10    exit
11 fi
12
13 case $((score / 10)) in
14     9|10)
15         echo "$score 成绩为A类"
16         ;;
17     8)
18         echo "$score 成绩为B类"
19         ;;
20     7)
21         echo "$score 成绩为C类"
22         ;;
23     6)
24         echo "$score 成绩为D类"
25         ;;
26     *)
27         echo "$score 成绩为E类"
28         ;;
29 esac
```

```
1 练习题2: 从终端输入任意一个字符, 判断此字符为数字, 字母, 运算符, 还是标点符号
2 #!/bin/bash
3 # your code
4 read -p "请输入一个字符 > " ch
5
6 case $ch in
7     [0-9])
8         echo "$ch是一个数字"
9         ;;
10    [a-zA-Z])
11        echo "$ch是一个字母"
12        ;;
```

```

13     +|-|\*|/|%)
14         echo "$ch是一个运算符"
15         ;;
16     \,|\.|\;|\:|\'|\\"|\!)
17         echo "$ch是一个标点符号"
18         ;;
19     *)
20         echo "$ch 其他符号"
21         ;;
22 esac
23

```

```

1  练习题3: 从终端输入年份和月份, 判断月份对应的天数。
2  #!/bin/bash
3  # your code
4
5  read -p "请输入年份和月份 > " year month
6  if [ $year -lt 0 -o $month -lt 1 -o $month -gt 12 ]
7  then
8      echo "输入的年份或者月份不合理, 请重新执行"
9      exit
10 fi
11
12 case $month in
13     1|3|5|7|8|10|12)
14         echo "$year年的$month月份共计31天"
15         ;;
16     4|6|9|11)
17         echo "$year年的$month月份共计30天"
18         ;;
19     2)
20         if (( $year % 4 == 0 && $year % 100 != 0 || $year % 400 == 0 ))
21         then
22             echo "$year年的$month月份共计29天"
23         else
24             echo "$year年的$month月份共计28天"
25         fi
26         ;;
27 esac

```

```

1  练习题4: 使用case..in实现一个简单的计算器
2  read -p "请输入一个表达式" lvalue operator rvalue
3
4  #!/bin/bash
5  # your code
6  read -p "请输入一个算数表达式" lvalue operator rvalue
7
8  case $operator in
9      +)
10         ((result = lvalue + rvalue))

```

```

11
12     ;;
13 -)
14     result=$((lvalue - rvalue))
15     ;;
16 \*)
17     result=$((lvalue * rvalue))
18     ;;
19 /)
20     result=$((lvalue / rvalue))
21     ;;
22 %)
23     result=$((lvalue % rvalue))
24     ;;
25 *)
26     echo "输入的运算符不合理"
27     ;;
28 esac
29
30 if [ "$operator" == '+' -o "$operator" == '-' -o "$operator" == '*' -o
"$operator" == '/' -o "$operator" == '%' ]
31 then
32     echo "$lvalue $operator $rvalue = $result"
33 fi

```

## 2、select...in选择分支语句

### 2.1 语法格式

```

1 1. 格式
2   select 变量名 in 单词列表
3   do
4       shell语句
5   done
6
7 2. 单词列表
8   多个以空格分隔的单词组成的单词列表
9
10 3. 执行过程
11   选择单词列表中的某个单词，赋值给“变量名”，然后选择分支中的“shell语句”
12   就会被执行一次。

```

### 2.2 参考案例

```

1 #!/bin/bash
2 # your code
3 select number in one two three four five
4 do
5     # 输出number变量的值
6     echo "number = $number"
7 done

```

```

8
9 linux@ubuntu:day04$ bash 08select.sh    ---> 执行select代码会出现一个菜单
10 1) one
11 2) two
12 3) three
13 4) four
14 5) five
15 #? 1          ---> 只能输入菜单对应的编号，菜单有单词列表构成，从1开始
16 number = one
17 #? 3
18 number = three
19 #? 6          ---> 输入大于菜单中的最大值无效
20 number =
21 #? four       ---> 输入单词列表中的单词也无效
22 number =
23 #? ^C         ---> 只能使用ctrl + c强制退出

```

```

1 select...in的使用场合?
2     select...in经常和case...in配合使用，提高代码的交互性。
3     #!/bin/bash
4     # your code
5
6 select os in macOS ubuntu windows redhat Android
7 do
8     case $os in
9         macOS)
10             echo "打开$os操作系统"
11             ;;
12         ubuntu)
13             echo "打开$os操作系统"
14             ;;
15         windows)
16             echo "打开$os操作系统"
17             ;;
18         redhat)
19             echo "打开$os操作系统"
20             ;;
21         Android)
22             echo "打开$os操作系统"
23             ;;
24         *)
25             echo "未知的操作系统"
26             ;;
27     esac
28 done

```

```

1 #!/bin/bash
2 # your code
3
4 select thing in score mouth cal
5 do
6     case $thing in
7         score)
8             source 04score.sh

```

```

9         ;;
10        mouth)
11            source 06year.sh
12            ;;
13        cal)
14            source 07cal.sh
15            ;;
16        *)
17            echo "未知的操作"
18            ;;
19    esac
20 done

```

## 3、for循环

### 3.1 for循环的语法格式

```

1  1. 格式1: 类似于C语言的风格
2      for ((表达式1; 表达式2; 表达式3))
3      do
4          shell语句4
5      done
6
7      执行过程: [1,2][4,3,2][4,3,2][4,3,2]....
8
9  2. 格式2: 类似pythone风格
10     for 变量名 in 单词列表
11     do
12         shell语句
13     done
14     执行过程: 每次循环, 依次将单词列表中的单词赋值给"变量名",
15               每赋值一次for循环语句中的"shell语句"执行一次,
16               直到单词列表中的所有的单词都依次赋值给"变量名"之后,
17               循环结束。

```

### 3.2 for循环的案例

```

1  案例1: 定义一个数组, 数组有10个成员, 每个成员都是整数,
2          求数组中的所有的成员之和。
3
4  #!/bin/bash
5  # your code
6
7  arr=(10 20 30 40 50 60 70 80 90 100)
8  sum=0
9  # C风格的for循环实现数组中的成员求和
10 for ((i = 0; i < 10; i++))
11 do
12     sum=$((sum + ${arr[i]}))
13 done
14 echo "sum = $sum"
15
16

```

```

17 # python风格的for循环，实现数组中成员的求和
18 sum=0
19 for i in 0 1 2 3 4 5 6 7 8 9
20 do
21     sum=$((sum + ${arr[i]}))
22 done
23 echo "sum = $sum"
24
25 sum=0
26 # for val in 10 20 30 40 50 60 70 80 90 100
27 for val in ${arr[*]}
28 do
29     sum=$((sum + val))
30 done
31 echo "sum = $sum"
32
33 # seq命令：输出一个等差数列
34 # seq 起始值 结束值 ： 输出一个默认差值为1的等差数列
35 # seq 起始值 差值 结束值 ： 输出一个差值为“差值”的等差数列
36
37 sum=0
38 for i in `seq 0 9`
39 do
40     sum=$((sum + ${arr[i]}))
41 done
42 echo "sum = $sum"
43
44
45

```

```

1  练习题2：从终端输入一个行号，打印以西图形
2  *
3  **
4  ***
5  ****
6  *****
7  .....
8
9  #!/bin/bash
10 # your code
11
12 read -p "请输入行数 > " line
13 if [ $line -lt 1 ] ; then
14     echo "请输入大于1数字"
15     exit
16 fi
17
18 for row in `seq 1 $line`
19 do
20     for col in `seq 1 $row`
21     do
22         echo -n '*'
23     done
24     echo ''
25 done

```

```
1 练习题3:
2 使用for循环打印以下图形
3 *****
4      ****
5          ***
6              **
7                  *
```

```
1 练习题4:
2 使用for循环打印以下图形
3 #####*@@@
4 #####*@@@
5 #####*@@
6 #####*
7 #####
8
9 #!/bin/bash
10 # your code
11
12 read -p "请输入行数 > " line
13 if [ $line -lt 1 ] ; then
14     echo "请输入大于1数字"
15     exit
16 fi
17
18 for row in `seq 0 $((line - 1))`
19 do
20     for col in `seq 0 $((line - row - 2))`
21     do
22         echo -n '#'
23     done
24     for col in `seq 0 $((row * 2))`
25     do
26         echo -n '*'
27     done
28     for col in `seq 0 $((line - row - 2))`
29     do
30         echo -n '@'
31     done
32     echo ''
33 done
```

```
1 练习题5:
2     定义数组，数组长度为10，初始化为10个整数，
3     使用for循环完成数组的排序。
4
5 #!/bin/bash
6 # your code
7 array=(10 34 67 98 45 76 99 12 9 58)
8 # echo `seq 0 ${#array[*]} - 2)`
9 for i in `seq 0 ${#array[*]} - 2)`
```



```

10 do
11     # echo `seq 0 $(( ${#array[*]} - 2 - i ))`
12     for j in `seq 0 $(( ${#array[*]} - 2 - i ))`
13     do
14         if [ ${array[j]} -gt ${array[j+1]} ] ; then
15             tmp=${array[j]}
16             array[j]=${array[j+1]}
17             array[j+1]=$tmp
18         fi
19     done
20 done
21
22 echo "array=(${array[*]})"

```

## 4、while循环

### 4.1 while循环的语法格式

```

1 1. 格式1
2   while ((条件表达式1))
3   do
4       shell语句2
5   done
6
7 2. 格式2：和tset命令的结合
8   while [ 表达式1 ]
9   do
10      shell语句2
11  done
12
13 3. 执行过程
14  [1][2,1][2,1][2,1][2,1]

```

### 4.2 参考案例

```

1 使用while循环实现0-100之间的所有的数据的求和。
2  #!/bin/bash
3  # your code
4  sum=0
5  i=0
6  while ((i<=100))
7  do
8      sum=$((sum + i))
9      ((i++))
10 done
11 echo "sum = $sum"
12
13 sum=0
14 i=0
15 while [ $i -le 100 ]
16 do
17     sum=$((sum + i))
18     i=$((i + 1))

```

```
19 done
20
21 echo "sum = $sum"
```

## 5、break和continue

### 5.1 break关键字

```
1 格式：
2     break n      ---> 退出最近的n层循环体
3
4 注：
5     1> n是一个大于等于1的整数，
6         当n等于1时，可以省略不写
7
8 案例
9     break 2      ---> 退出2层循环体
10    break 1 / break ---> 退出1层循环体
```

### 5.2 continue关键字

```
1 格式：
2     continue n    ---> 退出最近的n层本次循环，执行下一次循环
3
4 注：
5     1> n是一个大于等于1的整数，
6         当n等于1时，可以省略不写
7
8 案例
9     continue 2    ---> 退出2层本次循环，执行下一次循环
10    continue 1 / continue ---> 退出1层本次循环，执行下一次的循环
```

```
1 #!/bin/bash
2 # your code
3
4 for i in `seq 1 5`
5 do
6     echo -n '#'
7     for j in `seq 5 -1 1`
8     do
9         echo -n '@'
10        if [ $i -eq $j ]
11        then
12            break 2
13        fi
14    done
15 done
16
17 echo ''
18
```

19 # 输出结果: #####

```
1 #!/bin/bash
2 # your code
3
4 for i in `seq 1 5`
5 do
6     echo -n '#'
7     for j in `seq 5 -1 1`
8     do
9         echo -n '@'
10        if [ $i -eq $j ]
11        then
12            break 1
13        fi
14    done
15 done
16
17 echo ''
18
19 # 输出结果: #####
```

```
1 #!/bin/bash
2 # your code
3
4 for i in `seq 1 5`
5 do
6     for j in `seq 5 -1 1`
7     do
8         if [ $i -eq $j ]
9         then
10            continue 1
11        fi
12        echo -n '@'
13    done
14    echo -n '#'
15 done
16
17 echo ''
18
19 # 输出结果: #####
```

```
1 #!/bin/bash
2 # your code
3
4 for i in `seq 1 5`
5 do
6     for j in `seq 5 -1 1`
7     do
8         if [ $i -eq $j ]
9         then
10            continue 2
11        fi
```

```
12     echo -n '@'
13     done
14     echo -n '#'
15 done
16
17 echo ''
18
19 # 输出结果: @@@@@@@@@@
```

## 6、shell函数

### 6.1 shell函数定义的格式

```
1 1. shell函数定义的格式
2   function 函数名()
3   {
4       函数体
5   }
6 2. 注意事项
7     1> shell中的函数依然遵循先定义后使用的原则
8     2> shell中定义的函数需要使用function修饰，可以省略function，一般不省略
9     3> 函数的返回值，函数是否有返回值由定义的函数决定
10    4> 函数的参数，函数是否有形参由定义函数时决定，
11        即使函数有参数，()中不需要书写任何东西。
```

### 6.2 shell函数的调用

```
1 函数没有参数的调用：
2     函数名
3
4 函数由参数的调用
5     函数名 实参的列表：每个参数使用空格隔开
```

### 6.3 shell函数中获取函数传递的实参值

```
1 在函数中获取函数传递的实参值，通过位置变量获取，$n，n是一个大于等于1的整数。
2 $n位置变量是局部变量。
3
4 如果在函数中使用$n时，
5 $0      ---> 表示的是脚本文件名
6 $1      ---> 调用函数时，传递的第1个参数
7 $2      ---> 调用函数时，传递的第2个参数
8 $3      ---> 调用函数时，传递的第3个参数
9 ...
10 $9      ---> 调用函数时，传递的第9个参数
11 $10     ---> 调用函数时，传递的第10个参数，和字符串0拼接
```

```
12  ${10}    ---> 调用函数时, 传递的第10个参数
13  ...
14  ${n}     ---> 调用函数时, 传递的第n个参数
```

## 6.4 shell函数的返回值

```
1  1. 通过shell函数中定义全局变量的方式返回。
2      shell函数中定义的变量默认也属于全局变量,
3      如果在shell函数中定义全局变量, 则在shell函数外,
4      可以之间使用shell函数中定义的全局变量
5
6  2. 通过return返回结果
7      使用return返回值时, 只能返回0-255之间的数。
8      使用$?接收函数的返回值。
9
10 3. 通过echo命令返回结果, 可以返回任何一个数或者字符串。
11     接收函数的返回值通过命令置换符实现。
12     retVal=`函数名`
13     或者
14     retVal=$(函数名)
```

## 6.5 shell函数的案例

```
1  #!/bin/bash
2  # your code
3
4  # 1. 定义函数没有参数, 没有返回值
5  function print()
6  {
7      echo "hello world"
8  }
9
10 # 2. 定义函数, 函数有参数, 没有返回值
11 function set_init_array()
12 {
13     # arr[0]=$1
14     # arr[1]=$2
15     # arr[2]=$3
16     # arr[3]=$4
17     # arr[4]=$5
18
19     # arr=($*)
20     # arr=($@)
21
22     echo $#    # 函数的参数的个数
23     arr=($1 $2 $3 $4 $5)
24 }
25
26 # 3. 定义函数, 有参数, 有返回值, 通过全局变量返回
27 function add_func()
```

```

28 {
29     sum=$(( $1 + $2 ))
30 }
31
32 #4. 定义函数，有参数，有返回值，通过return返回
33 #     只能返回0-255之间的数，通过$?接收return的返回值
34 #     $? 作用：接收上一个函数或者命令的返回结果
35 function sub_func()
36 {
37     local sub=0;
38     sub=$(( $1 - $2 ))
39     return $sub
40 }
41
42 # 5. 定义函数，有参数，有返回值，通过echo返回结果
43 function mul_func()
44 {
45     local mul=0;
46     mul=$(( $1 * $2 ))
47     echo $mul
48 }
49
50 # 1. 调用函数，没有参数，没有返回值
51 print
52
53 #2. 调用函数，函数有参数，没有返回值
54 set_init_array 111 222 333 444 555
55 echo "arr=({arr[*]})"
56
57 #3. 调用函数，函数有参数，有返回值，通过全局变量返回
58 add_func 100 200
59 echo "100 + 200 = $sum"
60
61 #4. 调用函数，有参数，有返回值，通过return返回
62 # 只能返回0-255之间的数
63 sub_func 200 100
64 echo "200 - 100 = $?"
65
66 sub_func 400 100
67 echo "400 - 100 = $?"
68
69 sub_func 100 200
70 echo "100 - 200 = $?"
71
72
73 #5. 函数调用，有参数，有返回值，通过echo返回
74 mul=`mul_func 100 200`
75 echo "100 * 200 = $mul"

```

## 6.6 main函数返回值的使用

01main.c

```

1 #include <stdio.h>
2

```

```

3  int main(int argc, const char *argv[])
4  {
5      /*your code*/
6      // 执行程序时，给程序传递两个参数
7      if (argc != 3)
8      {
9          printf("执行程序时，传递的参数错误\n");
10         printf("%s string1 string2\n", argv[0]);
11         return -1;
12     }
13     // 程序正常退出返回0，异常退出返回-1
14     // 将01main.c编译生成main的可执行程序 gcc 01main.c -o main
15     // 在脚本文件中调用main程序
16     return 0;
17 }

```

01main.sh

```

1  #!/bin/bash
2  # your code
3  # 调用C程序
4  #./main hello
5  ./main hello world
6
7  # 在shell脚本文件中调用c程序时，
8  # c的程序可能执行成功，可能执行失败，
9  # 此时可以判断程序退出时的返回值，判断C程序执行的成功于失败，
10 # 以及判断具体失败的原因。
11 if [ $? -eq 0 ] ; then
12     echo "main程序正常退出"
13 elif [ $? -eq 255 ] ; then
14     echo "执行main程序时传递的参数个数错误"
15 fi

```

