1、case...in分支语句

1.1 语法格式

```
1. 语法格式
    case  $变量名/常量表达式  in     ---> 等价于：switch(常量表达式)
        字符串常量1)                 ---> 等价于 ：case 常量:
            shell语句1               ---> 等价于: C语句块
            ;;                     ---> 等价于: break;
        字符串常量2)
            shell语句2
            ;;
        字符串常量3)
            shell语句3
            ;;
        ......
        *)                         ---> 等价于：default:
            shell语句n
            ;;
    esac

2. 字符串常量表达式的写法：
    a)     ---> 字符a
    1)     ---> 数字1
    yes|Yes|YES|y)  ---> 字符串yes，Yes,YES,y中的任意一个
```

```
    [0-9])  ---> 数组0-9中的任意一个
    [a-zA-Z])  ---> a-z或者A-Z中的任意一个字符
    NO)  ---> 字符串NO
    *)   ---> 任意字符串
    1|3|5|7|9|12)  ---> 数字1，3，5，7，9，12中的任意一个

3. 使用的注意事项
    在case...in中除了最后一个分支的两个分号可以省略不写，
    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

```
练习题1：成绩的分类

#!/bin/bash
# your code

read -p "请输入成绩，范围为0-100 > " score
if [ \( "$score" -lt 0 \) -o \( "$score" -gt 100 \) ]
then
    echo "输入成绩不合理，范围为0-100"
    exit
fi

case $((score / 10)) in
    9|10)
        echo "$score 成绩为A类"
        ;;
    8)
        echo "$score 成绩为B类"
        ;;
    7)
        echo "$score 成绩为C类"
        ;;
    6)
        echo "$score 成绩为D类"
        ;;
    *)
        echo "$score 成绩为E类"
        ;;
esac
```

```
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
#!/bin/bash
# your code
read -p "请输入一个字符 > " ch

case $ch in
    [0-9])
        echo "$ch是一个数字"
        ;;
    [a-zA-Z])
        echo "$ch是一个字母"
        ;;
```

```
    +|-|\*|/|%)
        echo "$ch是一个运算符"
        ;;
    \,|\.|\;|\:|\'|\"|\!)
        echo "$ch是一个标点符号"
        ;;
    *)
        echo "$ch 其他符号"
        ;;
esac

```

练习题3：从终端输入年份和月份，判断月份对应的天数。

```
#!/bin/bash
# your code

read -p "请输入年份和月份 > " year mouth
if [ $year -lt 0 -o $mouth -lt 1 -o $mouth -gt 12 ]
then
    echo "输入的年份或者月份不合理，请重新执行"
    exit
fi

case $mouth in
    1|3|5|7|8|10|12)
        echo "$year年的$mouth月份共计31天"
        ;;
    4|6|9|11)
        echo "$year年的$mouth月份共计30天"
        ;;
    2)
        if (( $year % 4 == 0 && $year % 100 != 0 || $year % 400 == 0 ))
        then
            echo "$year年的$mouth月份共计29天"
        else
            echo "$year年的$mouth月份共计28天"
        fi
        ;;
esac
```

练习题4：使用case..in实现一个简单的计算器

```
read -p "请输入一个表达式"  lvalue operator rvalue

#!/bin/bash
# your code
read -p "请输入一个算数表达式" lvalue operator rvalue

case $operator in
    +)
        ((result = lvalue + rvalue))
```

```
        ;;
    -)
        result=$((lValue - rValue))
        ;;
    \*)
        result=$[lValue * rValue]
        ;;
    /)
        result=$(($lValue / $rValue))
        ;;
    %)
        result=$[$lValue % $rValue]
        ;;
    *)
        echo "输入的运算符不合理"
        ;;
esac

if [ "$operator" == '+' -o "$operator" == '-' -o "$operator" == '*' -o "$operator" == '\' -o "$operator" == '%' ]
then
    echo "$lValue $operator $rValue = $result"
fi
```

2、select...in选择分支语句

2.1 语法格式

```
1. 格式
    select 变量名 in  单词列表
    do
        shell语句
    done

2. 单词列表
    多个以空格分隔的单词组成的单词列表

3. 执行过程
    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”
    就会被执行一次。
```

2.2 参考案例

```
#!/bin/bash
# your code
select number in one two three four five
do
    # 输出number变量的值
    echo "number = $number"
done
```

```
linux@ubuntu:day04$ bash 08select.sh    ---> 执行select代码会出现一个菜单
1) one
2) two
3) three
4) four
5) five
#? 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
number = one
#? 3
number = three
#? 6        ---> 输入大于菜单中的最大值无效
number =
#? four     ---> 输入单词列表中的单词也无效
number =
#? ^C       ---> 只能使用ctrl + c强制退出
```

```
select...in的使用场合？
    select...in经常和case...in配合使用，提高代码的交互性。
    #!/bin/bash
# your code

select os in macOS ubuntu windows redhat Android
do
    case $os in
        macOS)
            echo "打开$os操作系统"
            ;;
        ubuntu)
            echo "打开$os操作系统"
            ;;
        windows)
            echo "打开$os操作系统"
            ;;
        redhat)
            echo "打开$os操作系统"
            ;;
        Android)
            echo "打开$os操作系统"
            ;;
        *)
            echo "未知的操作系统"
            ;;
    esac
done
```

```
#!/bin/bash
# your code

select thing in score mouth cal
do
    case $thing in
        score)
            source 04score.sh
```

```
            ;;
        mouth)
            source 06year.sh
            ;;
        cal)
            source 07cal.sh
            ;;
        *)
            echo "未知的操作"
            ;;
    esac
done
```

3、for循环

3.1 for循环的语法格式

```
1. 格式1: 类似于C语言的风格
    for ((表达式1; 表达式2; 表达式3))
    do
        shell语句4
    done

    执行过程：[1,2][4,3,2][4,3,2][4,3,2]....

2. 格式2：类似pythone风格
    for 变量名  in  单词列表
    do
        shell语句
    done
    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",
        每赋值一次for循环语句中的"shell语句"执行一次，
        直到单词列表中的所有的单词都依次赋值给“变量名”之后，
        循环结束。
```

3.2 for循环的案例

```
案例1：定义一个数组，数组有10个成员，每个成员都是整数，
    求数组中的所有的成员之和。

#!/bin/bash
# your code

arr=(10 20 30 40 50 60 70 80 90 100)
sum=0
# C风格的for循环实现数组中的成员求和
for ((i = 0; i < 10; i++))
do
    sum=$((sum + ${arr[i]}))
done
echo "sum = $sum"


```

```shell
# python风格的for循环，实现数组中成员的求和
sum=0
for i in 0 1 2 3 4 5 6 7 8 9
do
    sum=$[sum + ${arr[i]}]
done
echo "sum = $sum"

sum=0
# for val in 10 20 30 40 50 60 70 80 90 100
for val in ${arr[*]}
do
    sum=$[sum + val]
done
echo "sum = $sum"

# seq命令：输出一个等差数列
# seq  起始值 结束值 ：  输出一个默认差值为1的等差数列
# seq  起始值 差值 结束值 ：  输出一个差值为“差值”的等差数列

sum=0
for i in `seq 0 9`
do
    sum=$[sum + ${arr[i]}]
done
echo "sum = $sum"



```

练习题2：从终端输入一个行号，打印以西图形

```
*
**
***
****
*****
........
```

```shell
#!/bin/bash
# your code

read -p "请输入行数 > " line
if [ $line -lt 1 ] ; then
    echo "请输入大于1数字"
    exit
fi

for row in `seq 1 $line`
do
    for col in `seq 1 $row`
    do
        echo -n '*'
    done
    echo ''
done
```

```
练习题3：
使用for循环打印以下图形
*****
 ****
  ***
   **
    *
```

```
练习题4：
使用for循环打印以下图形
####***@@@@
###*****@@@
##*******@@
#*********@
**********

#!/bin/bash
# your code

read -p "请输入行数 > " line
if [ $line -lt 1 ] ; then
    echo "请输入大于1数字"
    exit
fi

for row in `seq 0 $((line - 1))`
do
    for col in `seq 0 $((line - row - 2))`
    do
        echo -n '#'
    done
    for col in  `seq 0 $((row * 2))`
    do
        echo -n '*'
    done
    for col in `seq 0 $((line - row - 2))`
    do
        echo -n '@'
    done
    echo ''
done
```

```
练习题5：
    定义数组，数组长度为10，初始化为10个整数，
    使用for循环完成数组的排序。

#!/bin/bash
# your code
array=(10 34 67 98 45 76 99 12 9 58)
# echo `seq 0 $((${#array[*]} - 2))`
for i in `seq 0 $((${#array[*]} - 2))`
```

```
do
    # echo `seq 0 $((${#array[*]} - 2 - i))`
    for j in `seq 0 $((${#array[*]} - 2 - i))`
    do
        if [ ${array[j]} -gt ${array[j+1]} ] ; then
            tmp=${array[j]}
            array[j]=${array[j+1]}
            array[j+1]=$tmp
        fi
    done
done

echo "array=(${array[*]})"
```

4、 while循环

4.1 while循环的语法格式

```
1. 格式1
    while ((条件表达式1))
    do
        shell语句2
    done

2. 格式2 ：和tset命令的结合
    while [ 表达式1 ]
    do
        shell语句2
    done

3. 执行过程
    [1][2,1][2,1][2,1][2,1]
```

4.2 参考案例

```
使用while循环实现0-100之间的所有的数据的求和。
#!/bin/bash
# your code
sum=0
i=0
while ((i<=100))
do
    sum=$((sum + i))
    ((i++))
done
echo "sum = $sum"

sum=0
i=0
while [ $i -le 100 ]
do
    sum=$[sum + i]
    i=$[i + 1]
```

```
done

echo "sum = $sum"
```

5、break和continue

5.1 break关键字

```
格式：
    break n     ---> 退出最近的n层循环体

注：
    1> n是一个大于等于1的整数，
        当n等于1时，可以省略不写

案例
    break 2   ---> 退出2层循环体
    break 1 / break   ---> 退出1层循环体
```

5.2 continue关键字

```
格式：
    continue n      ---> 退出最近的n层本次循环，执行下一次循环

注：
    1> n是一个大于等于1的整数，
        当n等于1时，可以省略不写

案例
    continue 2   ---> 退出2层本次循环，执行下一次循环
    continue 1 / continue   ---> 退出1层本次循环，执行下一次的循环
```

```
#!/bin/bash
# your code

for i in `seq 1 5`
do
    echo -n '#'
    for j in `seq 5 -1 1`
    do
        echo -n '@'
        if [ $i -eq $j ]
        then
            break 2
        fi
    done
done

echo ''

```

```
# 输出结果：#@@@@@
```

```
#!/bin/bash
# your code

for i in `seq 1 5`
do
    echo -n '#'
    for j in `seq 5 -1 1`
    do
        echo -n '@'
        if [ $i -eq $j ]
        then
            break 1
        fi
    done
done

echo ''

# 输出结果：#@@@@@#@@@@#@@@#@@#@
```

```
#!/bin/bash
# your code

for i in `seq 1 5`
do
    for j in `seq 5 -1 1`
    do
        if [ $i -eq $j ]
        then
            continue 1
        fi
        echo -n '@'
    done
    echo -n '#'
done

echo ''

# 输出结果：@@@@#@@@@#@@@@#@@@@#@@@@#
```

```
#!/bin/bash
# your code

for i in `seq 1 5`
do
    for j in `seq 5 -1 1`
    do
        if [ $i -eq $j ]
        then
            continue 2
        fi
```

```
        echo -n '@'
    done
    echo -n '#'
done

echo ''

# 输出结果：@@@@@@@@@@
```

6、shell函数

6.1 shell函数定义的格式

```
1. shell函数定义的格式
    function 函数名()
    {
        函数体
    }
2. 注意事项
    1> shell中的函数依然遵循先定义后使用的原则
    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
    3> 函数的返回值，函数是否有返回值由定义的函数决定
    4> 函数的参数，函数是否有形参由定义函数时决定，
        即使函数由参数，()中不需要书写任何的东西。
```

6.2 shell函数的调用

```
函数没有参数的调用：
    函数名

函数由参数的调用
    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

```
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。
$n位置变量是局部变量。

如果在函数中使用$n时，
$0      ---> 表示的是脚本文件名
$1      ---> 调用函数时，传递的第1个参数
$2      ---> 调用函数时，传递的第2个参数
$3      ---> 调用函数时，传递的第3个参数
...
$9      ---> 调用函数时，传递的第9个参数
$10     ---> 调用函数时，传递的第1个参数，和字符串0拼接
```

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

6.4 shell函数的返回值

```
1. 通过shell函数中定义全局变量的方式返回。
    shell函数中定义的变量默认也属于全局变量，
    如果在shell函数中定义全局变量，则在shell函数外，
    可以之间使用shell函数中定义的全局变量

2. 通过return返回结果
    使用return返回值时，只能返回0-255之间的数。
    使用$?接收函数的返回值。

3. 通过echo命令返回结果，可以返回任何一个数或者字符串。
    接收函数的返回值通过命令置换符实现。
    retVal=`函数名`
    或者
    retVal=$(函数名)
```

6.5 shell函数的案例

```shell
#!/bin/bash
# your code

# 1. 定义函数没有参数，没有返回值
function print()
{
    echo "hello world"
}

# 2. 定义函数，函数有参数，没有返回值
function set_init_array()
{
    # arr[0]=$1
    # arr[1]=$2
    # arr[2]=$3
    # arr[3]=$4
    # arr[4]=$5

    # arr=($*)
    # arr=($@)

    echo $#   # 函数的参数的个数
    arr=($1 $2 $3 $4 $5)
}

# 3. 定义函数，有参数，有返回值，通过全局变量返回
function  add_func()
```

```
{
    sum=$(($1 + $2))
}

#4. 定义函数，有参数，有返回值，通过return返回
#       只能返回0-255之间的数，通过$?接收return的返回值
#        $? 作用：接收上一个函数或者命令的返回结果
function sub_func()
{
    local sub=0;
    sub=$(($1 - $2))
    return $sub
}

# 5. 定义函数，有参数，有返回值，通过echo返回结果
function mul_func()
{
    local mul=0;
    mul=$(($1 * $2))
    echo $mul
}

# 1.调用函数，没有参数，没有返回值
print

#2. 调用函数，函数有参数，没有返回值
set_init_array 111 222 333 444 555
echo "arr=(${arr[*]})"

#3. 调用函数，函数有参数，有返回值，通过全局变量返回
add_func  100 200
echo  "100 + 200 = $sum"

#4. 调用函数，有参数，有返回值，通过return返回
# 只能返回0-255之间的数
sub_func 200 100
echo "200 - 100 = $?"

sub_func 400 100
echo "400 - 100 = $?"

sub_func 100 200
echo "100 - 200 = $?"


#5. 函数调用，有参数，有返回值，通过echo返回
mul=`mul_func 100 200`
echo "100 * 200 = $mul"
```

6.6 main函数返回值的使用

01main.c

```
#include <stdio.h>

```

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

01main.sh

```sh
#!/bin/bash
# your code
# 调用C程序
#./main  hello
./main  hello world

# 在shell脚本文件中调用c程序时，
# c的程序可能执行成功，可能执行失败，
# 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，
# 以及判断具体失败的原因。
if [ $? -eq 0 ] ; then
    echo "main程序正常退出"
elif [ $? -eq 255 ] ; then
    echo "执行main程序时传递的参数个数错误"
fi
```