一、case 语句

P1 case 语法格式

主要目的判断变量的取值，功能类似以 if 的多分支语句
case 范例，使用 case 语句编写脚本
当用户输入 redhat 参数，脚本返回 fedora
当用户输入 fedora 参数，脚本返回 redhat
当用户输入其他参数，则提示"必须输入 redhat 或者 fedora"

```
[root@svr7 ~]# mkdir /root/shell/day03
[root@svr7 ~]# cd /root/shell/day03
[root@svr7 day03]# vim case1.sh
#!/bin/bash
read -p "请输入 redhat|fedora： " key
case $key in
redhat)
        echo "fedora.";;
fedora)
        echo "redhat.";;
*)
        echo "必须输入 redhat 或 fedora."
esac
[root@svr7 day03]# chmod   +x case1.sh
[root@svr7 day03]# ./case1.sh
请输入 redhat|fedora：redhat
fedora.
```

判断用户的输入
当用户输入 y|Y|Yes|YES 参数，脚本返回 you enter 参数值
当用户输入 n|N|NO|no 参数，脚本返回 you enter 参数值
当用户输入其他参数，则提示"error"

```
[root@svr7 day03]# vim case2.sh
#!/bin/bash
read -p "Are you sure?[y/n]:" sure
case $sure in
y|Y|yes|YES)
        echo "you enter $sure,OK";;
n|N|NO|no)
        echo "you enter $sure,OVER";;
*)
        echo "error";;
esac
[root@svr7 day03]# chmod   +x case2.sh
```

```
[root@svr7 day03]# ./case2.sh
Are you sure?[y/n]:y
you enter ,OK
[root@svr7 day03]# ./case2.sh
Are you sure?[y/n]:n
you enter n,OVER
[root@svr7 day03]# ./case2.sh
Are you sure?[y/n]:
error
```

数组

数组也是一个变量，是一个有点特殊的变量。存储多个数据的集合就是数组。

```
[root@svr7 day03]# test=(aa bb cc)                    #定义数组
[root@svr7 day03]# echo ${test[0]}                    #调用数组的值，数组起始下标为 0
aa
[root@svr7 day03]# echo ${test[1]}
bb
[root@svr7 day03]# echo ${test[2]}
cc
```

编写一个石头剪刀布游戏的脚本

#通过随机数获取计算机的出拳

#出拳的可能性保存在一个数组中，game[0],game[1],game[2]分别是 3 中不同的可能

先把石头剪刀布，出拳的可能性编写出来，定义 game 变量，然后是计算机出拳，是石头剪刀布其中的一种可能，不能是其他的，拿 random 对 3 取余，结果肯定是 0-2 之间的数字

```
[root@svr7 day03]# vim stone.sh
#!/bin/bash
game=(石头 剪刀 布)
num=$[RANDOM%3]
computer=${game[$num]}
echo "请根据下列提示选择您的出拳手势"
echo "0.石头"
echo "1.剪刀"
echo "2.布"
read  -p  "请选择 1-3:"  person
case  $person  in
0)
        if [ $num -eq 0 ];then
                echo "平局"
        elif [ $num -eq 1 ];then
                echo "你赢"
        else
                echo "计算机赢"
        fi;;
```

```
1)
        if [ $num -eq 0 ];then
                echo "计算机赢"
        elif [ $num -eq 1 ];then
                echo "平局"
        else
                echo "你赢"
        fi;;
2)
        if [ $num -eq 0 ];then
                echo "你赢"
        elif [ $num -eq 1 ];then
                echo "计算机赢"
        else
                echo "平局"
        fi;;
*)
        echo "必须输入 0-2 的数字"
esac
[root@svr7 day03]# chmod +x stone.sh
[root@svr7 day03]# ./stone.sh
请根据下列提示选择您的出拳手势
1.石头
2.剪刀
3.布
请选择 0-2:1
你赢
```

二、shell 函数

P1 语法格式

在 Shell 脚本中，将一些需重复使用的操作，定义为公共的语句块，即可称为函数。通过使用函数，可以使脚本代码更加简洁，增强易读性，提高 Shell 脚本的执行效率。

格式 1：

```
function   函数名 {
    命令序列
    .. ..
}
```

格式 2：

```
函数名() {
    命令序列
    .. ..
```

```
}
```

函数的调用

直接使用“函数名”的形式调用，如果该函数能够处理位置参数，则可以使用“函数名 参数 1 参数 2 . . .”的形式调用。注意：函数的定义语句必须出现在调用之前，否则无法执行。

P2 函数范例

简单示例

```
[root@svr7 day03]# imsg(){          #定义函数
>    echo "hello world"
> echo "computer cloud"
> }
[root@svr7 day03]# imsg             #调用函数

[root@svr7 day03]# function msg {
> echo "jacob"
> echo "this is a test function"
> }
[root@svr7 day03]# msg
```

加法器

传递参数计算两个数字之和

```
[root@svr7 day03]# add(){
> echo $[$1+$2]
> }
[root@svr7 day03]# add 1 2
3
[root@svr7 day03]# add 10 8
18
```

运用函数编写打印不同颜色字体的脚本（输出颜色，每次定义颜色过于麻烦，可以写个函数提前定义）

```
[root@svr7 day03]# vim color.sh
#!/bin/bash
cecho(){
        echo -e "\033[$1m$2\033[0m"
}
cecho 31 OK
cecho 32 OK
cecho 33 OK
cecho 34 OK
cecho 35 Error
[root@svr7 day03]# chmod +x color.sh
[root@svr7 day03]# ./color.sh
```

多进程版 ping 测试的脚本（目的 ping 某个网段的主机，是否可以 ping 通）

```
[root@svr7 day03]# vim mutiping.sh
#!/bin/bash
myping(){
    ping -c1 -i0.2 -W1 $1    &>/dev/null
    if [ $? -eq 0 ];then
        echo "$1 is up"
    else
        echo "$1 is down"
    fi
}
for   i   in   {1..254}
do
     myping   192.168.4.$i   &
done
[root@svr7 day03]# chmod +x mutiping.sh
[root@svr7 day03]# ./mutiping.sh       #此时测试完成之后，会卡住不动，需要回车返回命令行，这是因为脚本在执行完之后已经结束返回命令行了，但是由于后台有的主机信息还没有返回结果，所以会出现这个情况
```

若要解决这个问题在意在脚本的最后加 wait 参数

```
[root@svr7 day03]# vim mutiping.sh
.....
wait
[root@svr7 day03]# ./mutiping.sh       #再次测试，成功
#使用&符号，将执行的函数放入后台执行,wait 等待所有后台进程结束后退出脚本
```

三、中断与退出

P1 基本语法

通过 break、continue、exit 在 Shell 脚本中实现中断与退出的功能。

break 可以结束整个循环；continue 结束本次循环，进入下一次循环；exit 结束整个脚本。

```
[root@svr7 day03]# vim tmp.sh
#!/bin/bash
for i in {1..5}
do
        [ $i -eq 3 ] && continue
        echo $i
done
echo over
[root@svr7 day03]# chmod +x tmp.sh
[root@svr7 day03]# ./tmp.sh
```

将上面的 continue 替换为 break，exit 分别测试脚本执行效果

机选双色球选号

先定义两个变量 red_ball 和 bule_ball 两个变量都为空，每选出一个号码通过+=的方式存储到变量中，通过 grep 判断新机选的红球号码是否已经存在，-w 过滤单词

```
[root@svr7 day03]# vim double_color.sh
#!/bin/bash
red_ball=""
blue_ball=""
while :
do
clear
echo "--机选双色球--"
tmp=$[RANDOM%33+1]
echo "$red_ball" | grep -q -w $tmp && continue
red_ball+=" $tmp"
echo -en "\033[91m$red_ball\033[0m"
word=$(echo "$red_ball" | wc -w)

if [ $word -eq 6 ];then
        blue_ball=$[RANDOM%16+1]
        echo -e "\033[34m $blue_ball\033[0m"
        break
fi
sleep 0.5
done
[root@svr7 day03]# chmod +x double_color.sh
[root@svr7 day03]# ./double_color.sh
--机选双色球--
 3 4 6 25 1 27 15
```

扩展：

```
grep   -q   # -q 无论找得到数据还是找不到数据，都不显示在屏幕上面
grep   -w   #看做的是一个独立的参数
[root@svr7 day03]# echo "abc" | grep -w a        #找不到
[root@svr7 day03]# echo "this is a apple" | grep -w a        #可以找到独立的 a
this is a apple
```