细节注意：

在Shell脚本中，使用`$(( ))`和`(( ))`的区别在于：

- `$(( ))`用于进行算术运算并返回结果，通常用于将算术表达式的结果赋值给变量或作为命令的参数。
- `(( ))`用于进行算术运算，但不返回结果，通常用于在条件表达式中进行数值比较或递增递减操作。

因此，当您需要获取算术表达式的结果时，应该使用`$(( ))`，而当您只需要进行算术运算而不需要获取结果时，可以使用`(( ))`。

---

a 减 1：

a=$((a-1)) 等价于 ((a--))

a是否大于0：

[$a -gt 0] 等价于 ((a > 0)) *注意空格*

比较运算符:记忆就是英文缩写
注意，用字母缩写就用中括号[ ] 用符号就用双括号(())
例子：*使用场景*

```
[ 变量 -gt 值 ]
(( 变量 > 值 ))
```

-eq : 等于 ==
-ne :不等于 ! =
-gt :大于 >
-lt :小于 <
-ge :大于等于 > =
-le :小于等于 < =

-1. 如果用echo打印的语句末尾为!,那么必须在!后面再加一个空格
如: echo "hello world! "

用 -n参数表示不换行
echo -n "hello world" 打印完后不换行

0.$表示取值，相当于c语言里面的* 解引用，不用的化相当于字符串

1.等号左右不能出现空格，不然会赋值为0

2.当进行算数运算时：
您应该使用`$(( ))`或者`let`来进行数学运算。因此，您可以将`c=$(a+b)`修改为`c=$((a+b))`或者`let c=a+b`，这样就可以正确地计算变量c的值了。

例如：

正确

```Shell
#!/bin/bash


 a=10
 b=20
 let c=a+b

 echo "$c"
```

正确：

```Shell
#!/bin/bash


 let a=10
 let b=20
 let c=a+b

 echo "$c"
```

正确：

```Shell
#!/bin/bash


  a=10
  b=20
  c=$((a+b))

 echo "$c"
```

3.read为输入（还有一种是命令行中输入数据）

```Shell
#!/bin/bash

echo "请输入一个数字："

read number

echo "你输入的数字为：$number"
```

结果为：

```
22448@LAPTOP-GN7534KO MINGW64 ~/Desktop (master)
$ sh sum.sh
请输入一个数字：
10
你输入的数字为：10
```

当然如果你想写成一行也可以：
在Linux Shell中，`-p`选项用于`read`命令，它允许您在接收用户输入时提供提示信息。

```Shell
#!/bin/bash

read -p "请输入一个数字: " number

echo "你输入的数字为: $number"
```

结果为:

```
22448@LAPTOP-GN7534KO MINGW64 ~/Desktop (master)
$ sh sum.sh
请输入一个数字: 10
你输入的数字为: 10
```

调用.sh时输入变量值

```Shell
#!/bin/bash

if [ $# -eq 0 ]; then
        echo "无参数"
else
        sum=0
        count=$#
        for num in "$@"; do
                sum=$((sum + num))
        done
        average=$(($sum / $count))
        echo "平均值为: $average"
fi
```

解释

```Plain Text
这个脚本用于计算传入参数的平均值。如果没有传入参数，则输出"无参数"；否则，计算参数的平均值并输出。

脚本中的`$#`用于获取传入参数的个数，`$@`用于获取所有的参数。在for循环中，对每个参数进行累加，然后使用bc命令计算平均值，并保留两位小数。
```

```
这个脚本看起来是正确的，可以用于计算传入参数的平均值。
```

表示参数的个@数表示参数列表

scale=2 表示保留精确度2位

| bc 表示用浮点数的计算，以提高精度。

使用:

```
22448@LAPTOP-GN7534KO MINGW64 ~/Desktop (master)
$ sh sum.sh 10 20 30
平均值为: 20
```

例2:

两种写法:

```Shell
#!/bin/bash

方法二：使用位置参数
average=$(( ($1+$2+$3)/3 ))
echo "平均值为: $average"
```

```Shell
#!/bin/bash

let average=($1+$2+$3)/3
echo "平均值为: $average"
```

```
22448@LAPTOP-GN7534KO MINGW64 ~/Desktop (master)
$ sh sum.sh 5 6 7
平均值为: 6
```

条件语句

4.if语句

在Shell脚本中，`if`语句用于根据条件执行不同的代码块。`if`语句的基本语法如下：

1)如果then不跟条件在一行，条件[]后面就不要加 ";"

```Shell
if [ condition ]
then
   # 在条件成立时执行的代码块
else
   # 在条件不成立时执行的代码块
fi
```

例子：

有很多细节要注意：

I.条件里面 a == b *左右 要与[]之间有一个空格*
II.== 可以用 -eq 来替代
III. 由于这里的then并没有和条件写一块，所以]后面不用加分号";"

```Shell
#!/bin/bash

read -p "请输入a: " a
read -p "请输入b: " b


if [ $a == $b ]
then
    echo "相等"
else
    echo "不相等"
fi
```

```
22448@LAPTOP-GN7534KO MINGW64 ~/Desktop (master)
$ sh sum.sh
请输入a: 10
```

```
请输入b: 20
不相等
```

2)如果then与在条件在一行，那么条件[]后面就要加";"

```shell
if [ condition ]; then
   # 在条件成立时执行的代码块
else
   # 在条件不成立时执行的代码块
fi
```

例如:
注意细节:
I.条件里面 a == b *左右 要与[]之间有一个空格*
II.== 可以用 -eq 来替代
III.由于then和条件写在一行，所以]后面要加分号";"

```shell
#!/bin/bash

read -p "请输入a: " a
read -p "请输入b: " b


if [ $a == $b ]; then
    echo "相等"
else
    echo "不相等"
fi
```

```
22448@LAPTOP-GN7534KO MINGW64 ~/Desktop (master)
$ sh sum.sh
请输入a: 10
请输入b: 10
相等
```

3)elseif语句

其实跟java c语言的基本上没区别

注意细节:
I.then 分号问题 同上
II. else if 为 elif
III.最后别忘了 fi结尾
IV. else可以不写(如果情况允许)

语法格式:

```Shell

if [ condition ]; then
   # 在该条件成立时执行的代码块
elif [ condition ]; then
   # 在该条件成立时执行的代码块
elif [ condition ]; then
   # 在该条件成立时执行的代码块
   .
   .
   .
else
        #其它情况执行
fi
```

```Shell
#!/bin/bash

read -p "请输入成绩: " grade

if [ $grade -ge 90 ]; then
    echo "大于等于90"
elif [ $grade -ge 80 ]; then
    echo "大于等于80"
elif [ $grade -ge 70 ]; then
    echo "大于等于70"
elif [ $grade -ge 60 ]; then
    echo "大于等于60"
else
    echo "不及格"
fi
```

结果为：

```
22448@LAPTOP-GN7534KO MINGW64 ~/Desktop (master)
$ sh sum.sh
请输入成绩: 88
大于等于80
```

---

4)switch case:

Plain Text

在传统的Bourne shell（sh）中，并没有直接的switch-case语句。不过，您可以使用多个if-else语句来模拟switch-case的功能。另外，像Bash等现代的shell脚本语言提供了类似于其他编程语言中switch-case的功能，可以使用case语句来实现类似的逻辑。以下是一个简单的示例：

Shell

```
#!/bin/bash

fruit="apple"

case $fruit in
    "apple")
        echo "苹果"
        ;;
    "banana")
        echo "香蕉"
        ;;
    "orange")
        echo "橙子"
        ;;
    *)
        echo "未知水果"
        ;;
esac
```

循环语句

在Shell脚本中，常见的循环语句包括for循环和while循环。

for循环

在Shell脚本中，for循环的基本语法如下：*也是有两种*

如果do写在for那一行，那么列表后面要加";"

```Shell
for 变量名 in 列表; do
    # 循环体
done
```

例子：

```Shell
#!/bin/bash

# 使用空格分隔元素列表
for a in 1 2 3 4 5 6; do
    echo "$a"
done
```

```
22448@LAPTOP-GN7534KO MINGW64 ~/Desktop (master)
$ sh sum.sh
1
2
3
4
5
6
```

来个不换行的：

```Shell
#!/bin/bash
```

```
for a in 1 2 3 4 5 6; do
    echo -n "$a "
done
```

```
22448@LAPTOP-GN7534KO MINGW64 ~/Desktop (master)
$ sh sum.sh
123456
```

如果**do**不写在**for**那一行，那么列表后面不要加**";"**

Shell

```
for 变量名 in 列表
do
  # 在这里执行循环体的代码
done
```

其中，`变量名`是用来依次存储列表中的每个元素的变量名，`列表`是一个由空格分隔的元素列表。在每次循环中，`变量名`会依次取到`列表`中的每个元素，并执行循环体中的代码。

while循环

在**Shell**脚本中，**while**循环的基本语法如下：也是有两种

do写在**condition**的同一行，**]**后要加**";"**

Shell

```
while [ condition ]; do
  # 在这里执行循环体的代码
done
```

例**1:** *使用运算符的缩写*

Shell

```
#!/bin/bash
a=3
while [ $a -gt 0 ]; do
        echo -n "$a"
        a=$((a-1))
done
```

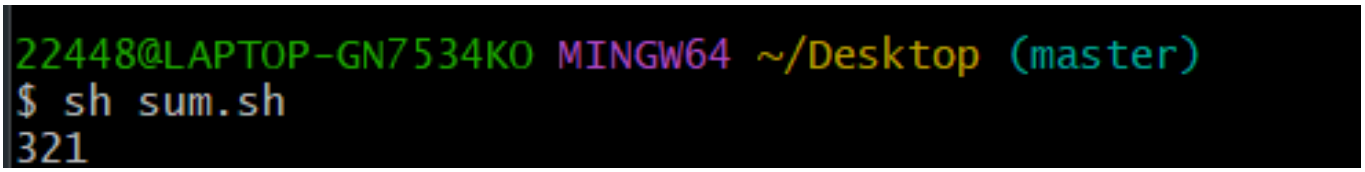
```
22448@LAPTOP-GN7534KO MINGW64 ~/Desktop (master)
```

```
22448@LAPTOP-GN7534KO MINGW64 ~/Desktop (master)
$ sh sum.sh
321
```

例子2:*使用运算符* 还有((a--))

```shell
#!/bin/bash

a=3
while (( a > 0 )); do
    echo -n "$a "
    ((a--))
done
```



do不与condition写在同一行，]后不用加";"

```shell
while [ condition ]
do
  # 在这里执行循环体的代码
done
```

用while实现dowhile

```shell
# 使用while循环实现类似do-while的功能
condition=false
while true; do
  # 在这里执行循环体的代码
  # 检查条件
  if $condition; then
    break
  fi
  # 更新条件
  condition=true
done
```

在这个例子中，我们使用了一个**while**循环来模拟**do-while**的功能。首先执行一次循环体的代码，然后检查条件，如果条件满足，则继续执行循环，否则跳出循环。

总结

对于赋值操作和运算操作：

在Shell脚本中，使用`$(( ))`和`(( ))`的区别在于：

- `$(( ))`*用于进行算术运算并返回结果*，通常用于将算术表达式的结果赋值给变量或作为命令的参数。
- `(( ))`*用于进行算术运算，但不返回结果*，通常用于在条件表达式中进行数值比较或递增递减操作。

因此，当您需要获取算术表达式的结果时，应该使用`$(( ))`，而当您只需要进行算术运算而不需要获取结果时，可以使用`(( ))`。

c=$((a+b))：这种将a与b相加赋值给c
let c=a+b：这种将a与b相加赋值给c
((a--))：a自减1

注意：=左右千万不能加空格

条件判断中运算符的使用

比较运算符：记忆就是英文缩写
注意，用字母缩写就用中括号[ ] 用符号就用双括号(())

例子：*使用场景*
[变量 -gt 值]
((变量 > 值))

-eq :等于 ==
-ne :不等于 ! =
-gt :大于 >
-lt :小于 <
-ge :大于等于 > =
-le :小于等于 < =