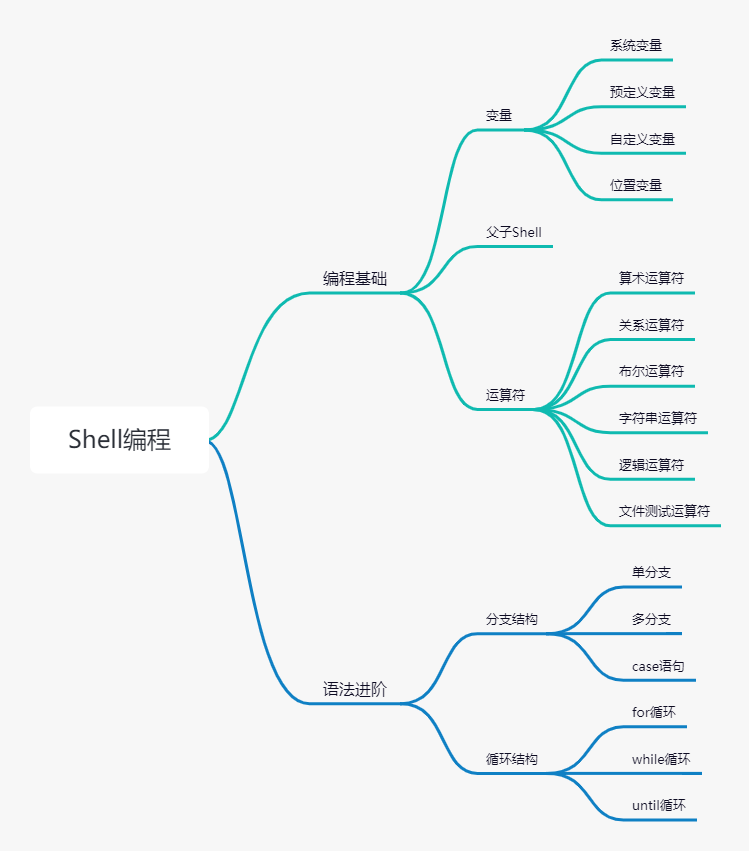
**项目七 Shell编程**

# 【项目介绍】

在统信UOS操作系统中，Shell不仅是常用的命令解程程序，还是高级编程语言。用户可以通过编写Shell程序来完成大量自动化的任务。Shell 可以互动地解释和执行用户输入的命令，也可以用来进行程序设计。它提供了定义变量和参数的手段以及丰富的程序控制结构。

# 【知识图谱】



# 任务 1 Shell编程基础

## 【任务描述】

本任务主要是带大家去掌握Shell编程基础，例如变量、父子Shell、运算符。通过本任务的学习，大家要掌握Shell脚本的基本知识。

## 【知识储备】

（1）Shell命令：例如export、echo、exit、pwd、return

（2）Linux命令：例如date、rm、mkdir

（3）文本处理功能：例如awk、cut、sed、grep

小贴士：

shell补充的快捷键

ESC+T：调换光标前的两个单词

## 7.1.1 Shell编程---变量

在统信UOS操作系统中，使用Shell Script来编写程序时，要掌握Shell变量、Shell运算符、Shell流程控制语句等相关变量、运算符、语法、语句。Shell变量是Shell传递数据的一种方式，用来代表每个取值的符号名，当Shell Script需要保存一些信息，如一个文件名或一个数字时，会将其存放在一个变量中。

Shell 变量的设置规则如下：

1. 变量名称可以由字母、数字和下画线组成，但是不能以数字开头，环境变量名称建议采用大写字母，用于区分。
2. 在bash中，变量的默认类型都是字符串型，如果要进行数值运算，则必须指定变量类型为数值型。
3. 变量用“=”连接值，等号两侧不能有空格。
4. 如果变量的值有空格，则需要使用单引号或者双引号将其括起来。

有效的Shell变量名示例如下:

1. USERNAME
2. LD\_LIBRARY\_PATH
3. \_var
4. var1

无效的变量名示例如下:

1. ?var=123
2. User\*name=yantaol

Shell 中的变量分为环境变量、位置参数变量、预定义变量和用户自定义变量，从变量的作用域角度分为全局变量和局部变量，可以通过 set命令查看系统中的所有变量。

（1）环境变量用于保存与系统操作环境相关的数据，如HOME、PWD、SHELL、 USER等。

（2）位置参数变量主要用于向脚本中传递参数或数据，变量名不能自定义，变量的作用固定。

（3）预定义变量是 Shell 中已经定义好的变量，变量名不能自定义，变量的作用也是固定的。

（4）用户自定义变量以字母或下画线开头，由字母、数字或下画线组成，大小写字母的含义不同，变量名长度没有限制。

1. **变量使用**

习惯上使用大写字母来命名变量，变量名以字母或下画线开头，不能以数字开头。在使用变量时，要在变量名前面加上“$”。

1. 变量赋值“=”



1. 使用单引号和双引号的区别



图1：双引号的作用



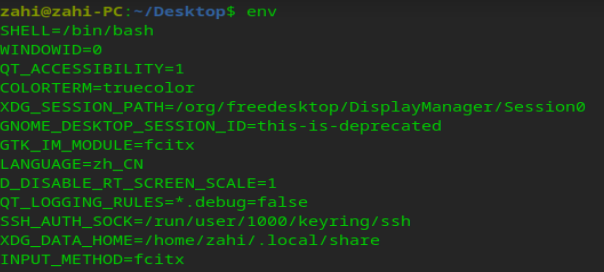
图2：单引号的作用

综合图1、2不难看出单引号的内容会被全部输出，但是双引号中的内容会有所变化，因为双引号会对其中的特殊符号进行转义。

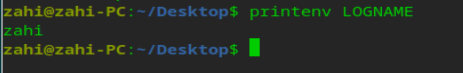
1. 列出本地定义的Shell变量“set”

C:\Users\Zahi\Documents\WeChat Files\zhangzhihao634584\FileStorage\Temp\1657688033798.png

1. 显示所有的环境变量env



1. 显示指定的环境变量printenv



其中printenv在不添加参数的条件下跟env功能上是一致的。

1. 撤销变量“unset”



注意：如果我们申明的变量是一个静态的变量(只读变量)，那么我们就不能用unset命令来进行撤销



1. **环境变量**

用户自定义变量只在当前的Shell中生效，而环境变量会在当前Shell及其所有的子Shell中生效。如果将环境变量写入相应的配置文件，则这个环境变量将会在所有的Shell中生效。

1. **位置参数变量**

$n：$0代表命令本身，$1-9代表接收的第 1-9个参数，10 及以上需要用{}括起来。

例如${10}代表接收的第10个参数。

$\*：代表接收所有参数，将所有的参数看作一个整体。

$@：代表接收所有参数，将每个参数都区别对待。

$#：代表接收的参数的个数。

1. **预定义变量**

预定义变量是在Shell中已经定义的变量，和默认环境变量有些类似，但不同的是预定义变量是不能重新定义的，用户只能根据Shell的定义来使用这些变量。预定义变量及其功能如下表7.1所示。

**表7.1 预定义变量及其功能说明**

|  |  |
| --- | --- |
| 预定义变量 | 功能说明 |
| $ | 最后一次执行的命令的返回状态。如果这个变量的值为0就证明上一条命令是执行正确的；如果这个变量的值不是0，则证明上一条执行的是错误的 |
| $$ | 当前进程的进程号 |
| $! | 后台运行的最后一个进行的进程号 |

## 7.1.2 Shell编程---父子Shell

在Shell环境内嵌套一个Shell，那么第一个Shell就是新开Shell的父Shell，而新开的Shell就是第一个Shell的子Shell。子Shell和父Shell最大的区别就是，环境变量的集成关系，如在子环境设置的当前变量，父环境变量是不可见的。

1. **父Shell**

父 Shell 是用于登录某个远程主机或虚拟控制器终端或在 GUI 中运行终端仿真器时所启动的默认的交互式 Shell 。

source script # 在当前 Shell 执行 script 文件

. script # 在当前 Shell 执行 script 文件

1. **子 Shell（subshell）**

子 Shell 是父 Shell 进程调用了 fork() 函数，在内存中复制出一个与父 Shell 进程几乎完全一样的子进程。

* 子 Shell 继承了父 Shell 的所有环境变量（包括全局和局部变量）
* 可以通过环境变量 BASH\_SUBSHELL（其值表明子 Shell 的嵌套深度）判断是第几层子 Shell（0 说明当前 Shell 不是子 Shell）

`command[;command...]` # command 在子 Shell 中执行

( command[;command...] ) # command 在子 Shell 中执行（可嵌套）

command1 | command2 # command1 和 command2 都在子 Shell 中运行

1. **子Shell从父Shell继承得来的属性如下：**

* 当前工作目录
* 环境变量
* 标准输入、标准输出和标准错误输出
* 所有已打开的文件标识符
* 忽略的信号

1. **子Shell不能从父Shell继承的属性，归纳如下：**

* 除环境变量和.bashrc文件中定义变量之外的Shell变量
* 未被忽略的信号处理

## 7.1.3 Shell编程---运算符

Shell支持很多的运算符，其中就包括算术运算符、关系运算符、布尔运算符、字符串运算符、逻辑运算符和文件测试运算符等等。

1. **算数运算符**

原生的bash是不支持间的数学运算的，但是可以通过其他的命令来完成，比如awk和expr，其中expr是比较常见的。expr是一个表达式计算命令，可以用它来完成表达式的求值操作。

列如，需要求两个数的求和，编写add.sh脚本：



add.sh脚本运算结果

小提示：脚本执行的方式

第一种：采用bash+脚本相对路径或绝对路径

sh test.sh

第二种：采用脚本绝对路径或相对路径直接运行(需要赋予文件执行权限）

./test.sh

第三种：采用source或. +脚本相对路径或绝对路径

source ./test.sh 或 . ./test

其中add.sh内容如下：



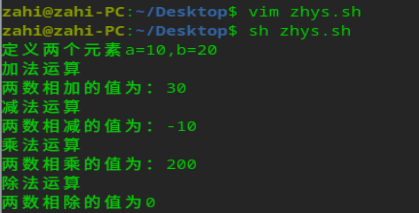
add.sh脚本内容

注意：add.sh的第一行#!是必须要填写的不能省略，表达式和运算符之间必须要有空格，脚本中“3 + 3” 不能写成“3+3”，这个跟大多数的编程有些不一样，同时完整的表达式是要加反引号（``）不是单引号（’’）。

算数运算符有以下几种：

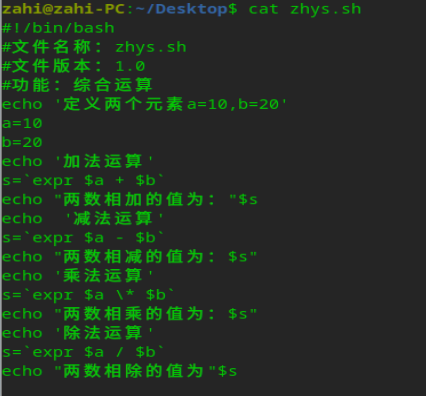
1. +（加），如`expr $A + $B`。
2. -（减），如`expr $A - $B`。
3. \*（乘），如`expr $A \* $B`。
4. /（除），如`expr $A / $B`。
5. =（赋值），如A=$B表示将变量B赋值给A。
6. ==（等于），用于比较两个数字，相等则返回true。
7. ！=（不等），用于比较两个数字，不相等则返回true。
8. %（取余），如`expr $A % $B`。

我们可以运用上述算术运算符进行加减乘除综合运算，相关命令如下：



zhys.sh脚本运算结果

其中zhys.sh内容如下：

****

zhys.sh脚本

1. **关系运算符**

关系运算符只支持数字，不支持字符串，除非字符串的值是数字。

常用的关系运算符如表7.1所示：假设A=10，B=20

**表7.1：常用的关系运算符**

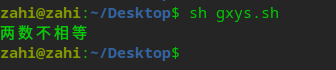
|  |  |  |
| --- | --- | --- |
| 运算符 | 功能说明 | 举例 |
| -eq | 检测两个数是否相等，相等则返回true | [ $A –eq $B] 返回true |
| -ne | 检测两个数是否不相等，不相等则返回true | [ $A –ne $B] 返回true |
| -gt | 检测运算符左边的数是否大于运算符右边的数，如果是则返回true | [ $A –gt $B] 返回true |
| -lt | 检测运算符左边的数是否小于运算符右边的数，如果是则返回true | [ $A –lt $B] 返回true |
| -ge | 检测运算符左边的数是否大于等于运算符右边的数，如果是则返回true | [ $A –ge $B] 返回true |
| -le | 检测运算符左边的数是否小于等于运算符右边的数，如果是则返回true | [ $A –le $B] 返回true |

我们可以运用关系运算符完成运算，相关命令如下：

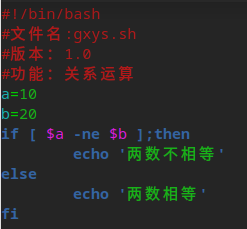
1. 等于运算脚本如下：



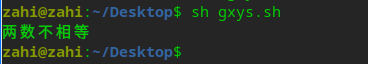
运算结果：



1. 不等于运算脚本如下



运算结果：



大于运算脚本如下：

小于运算脚本如下：

大于等于运算脚本如下：

小于等于运算脚本如下：

1. **布尔运算符**

常用的布尔运算符如表7.2所示：

**表7.2 常用的布尔运算符**

|  |  |  |
| --- | --- | --- |
| 运算符 | 功能说明 | 举例 |
| -a | 与运算，两个表达式都为true时，则返回true | [ $A –lt 20 -a $B –gt 10] 结果为:true |
| -o | 或运算，两个表达式只要其中一个为true时，则返回true | [ $A –lt 20 -o $B –gt 10] 结果为:true |
| ! | 非运算，表达式结果为true时，返回false | [ !true] 结果为：false |

1. **字符串运算符**

常用字符串运算符如表4.3所示

**表7.3 常用的字符串运算符**

|  |  |  |
| --- | --- | --- |
| 运算符 | 功能说明 | 举例 |
| = | 检测两个字符是否相等，相等则返回true | [ $A = $B ]返回结果false |
| != | 检测两个字符是否不相等，不相等则返回true | [ $A != $B ]返回结果true |
| -z | 检测字符串长度是否为0，为0则返回true | [ -z $B ]返回结果false |
| -n | 检测字符串长度是否不为0，不为0则返回true | [ -n “$B” ]返回结果false |
| $ | 检测字符串长度是否为空，不为空则返回true | [ $B ]返回结果true |

1. **逻辑运算符**

常用的逻辑运算符如表7.4所示

**7.4常用的逻辑运算符**

|  |  |  |
| --- | --- | --- |
| 运算符 | 功能说明 | 举例 |
| && | 逻辑与 | [ &A –lt 50 && $Y –gt 50]返回true |
| || | 逻辑或 | [$A –lt 50 || $B –gt 50] 返回true |

1. **文件测试运算符**

常用的文件测试运算符如表7.5所示

**7.5常用的逻辑运算符**

|  |  |  |
| --- | --- | --- |
| 运算符 | 功能说明 | 举例 |
| -b file | 检测文件是否为块设备文件，如果是，返回true | [ -b $file ]返回 false |
| -c file | 检测文件是否为字符设备文件，如果是，返回true | [ -c $file ]返回 false |
| -d file | 检测文件是否为目录文件，如果是，返回true | [ -d $file ]返回 false |
| -f file | 检测文件是否为普通文件，如果是，返回true | [ -f $file ]返回 true |
| -g file | 检测文件是否设置了SGID位，如果是，返回true | [ -g $file ]返回 false |
| -k file | 检测文件是否设置粘滞位，如果是，返回true | [ -k $file ]返回 false |
| -p file | 检测文件是否为有名管道，如果是，返回true | [ -p $file ]返回 false |
| -u file | 检测文件是否为SUID位，如果是，返回true | [ -u $file ]返回 false |
| -r file | 检测文件是否可读，如果是，返回true | [ -r $file ]返回 true |
| -w file | 检测文件是否可写，如果是，返回true | [ -w $file ]返回 true |
| -x file | 检测文件是否执行，如果是，返回true | [ -x $file ]返回 true |
| -s file | 检测文件是否为空，如果是，返回false | [ -s $file ]返回 true |
| -e file | 检测文件是否存在，如果是，返回true | [ -e $file ]返回 true |

1. **$()和``**

在Shell中，$()和``是可以用于命令替换。

采用如上这两种方式都可以获得内核的版本号，但是也有其各自的优缺点。

1. $（）的优点和缺点

优点:输入直观，不容易输入错误

缺点:不是所有的Shell都支持$()

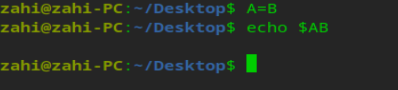
1. ``的优点和缺点

优点:``基本上是可以在所有的Shell中使用的。

缺点:``很容易输入错误

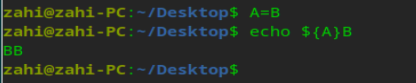
1. **${}**

${}可用于变量替换，一般情况下，$VAR与${VAR}没有什么不同，但是后者能准确的定位变量名称的范围。举例：



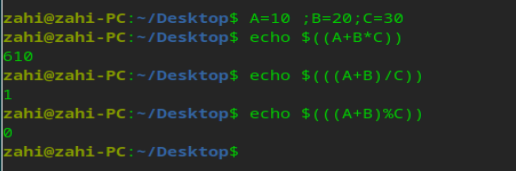
以上准本是将$A的结果替换出来的，之后在将B拼接在$A后面。但是结果如上图所示。

但是我们使用$A就不会出现上述的情况。



1. **$[]和$(())**

在$[]和$(())的作用的相似的，都可以用于数学的运算，支持加、减、乘、除、取余的运算，但是需要注意的是，bash只能进行整数的运算，浮点数是被当做字符串进行处理的。如下：



1. **[]**

[]为test命令的另一种形式，但使用时要注意以下几点。

1. 必须在其左括号的右侧和右括号的左侧各加一个空格，否则会报错。
2. test命令使用标准的数学比较符号来表示字符串的比较，而[]使用文体符号来表示数值的比较。
3. 大于符号或小于符号必须要进行转义，否则会被理解成重定向操作。
4. **(())和[[]]**

(())和[[]]分别是[]针对数学比较表达式和字符串表达式的加强版。

[[]]增加了模式匹配特效。（（））不需要再将表达式中的大于或小于符号转义，其除了可以使用标准的算术运算符外，还增加了一下运算符：a++（后增）、a--（后减）、++a（先增）、--a（先减）、！（逻辑反）、~(位求反)、\*\*（幂运算）、<<(左位移)、>>(右位移)、&(位布尔与)|(位布尔或)&&(逻辑与)||(逻辑或)

## 【任务实施】

## 7.1.4 创建并管理变量

1. 定义变量

在我们定义变量的时候我们一般使用“=”来定义一个变量，通过如下的方式我们创建了一个变量，同时给它赋予了数字10为它的值。

C:\Users\Administrator\Documents\WeChat Files\zhangzhihao634584\FileStorage\Temp\1658478995637.png

1. 撤销变量

在我们定义变量的时候我们如果不需要这个变量那么我们就需要撤销这个变量，我们可以通过unset来对已经存在的变量进行撤销

C:\Users\Administrator\Documents\WeChat Files\zhangzhihao634584\FileStorage\Temp\1658479179310.png

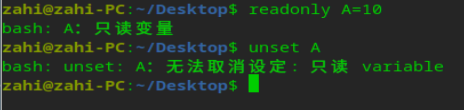
1. 定义静态变量

如果我们需要一个常量，也就是一个不能随时更改的变量，那么我们需要用到readonly，

下面我们就利用readonly来创建一个常量。

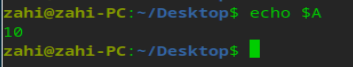


注意，readonly是不能采用unset来进行撤销的。



1. 输出变量

上面都是用来创建或撤销变量的，那我们如何来查看变量呢？这里需要用到echo这个基础指令。



上面A是我们创建的一个只读变量值为10。

## 【任务回顾】

【知识点总结】

1. Shell 中的变量分为环境变量、位置参数变量、预定义变量和用户自定义变量，从变量的作用域角度分为全局变量和局部变量。
2. Shell运算符，主要讲解了算术运算符、关系运算符、布尔运算符、字符串运算符、逻辑运算符、文件测试运算符、$()和``、$[]和$(())、${}、[]、（（））、[[]]。

【思考与练习】

1. 执行Shell脚本有哪几种方式？
2. Shell编程支持哪几种变量类型？
3. 可以使用（ ）命令对Shell变量进行算术运算。

A．Read B.expr C.export D.echo

1. Shell Script 通常使用（ ）符号作为脚本的开始。
2. # B.$ C.@ D.#!

# 任务 2 Shell语法进阶

## 【任务描述】

本任务主要是带大家去掌握Shell编程的分支结构、循环结构，这两种结构在编程中最为常见，同时也是必不可少的两个部分，通过本次任务学习，大家一定要去体会if的单双分支和循环的魅力。

## 【知识储备】

（1）Shell关键字：例如if...else、for do...done

（2）控制流语句：例如 if.….then.….else或执行重复操作的 Shell 循环。

## 7.2.1 Shell编程---分支结构

Shell流程分支控制语句可以使用单分支if条件语句、多分支if条件语句和case语句。下面进行分别举例。

1. **单分支if条件语句**

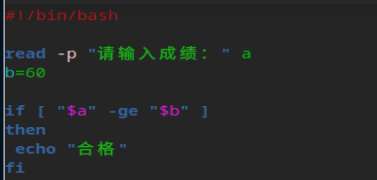
其语法格式如下：

if [ 条件语句 ];then

执行程序 #当条件语句成立时执行

fi #结束语句

例如：判断输入的成绩是否合格，大于等于60则判定成绩合格



执行结果如下：



注意：方括号跟条件语句之间要有空格，then可以换行写，这样就不需要再有方括号后面加“;”

1. **多分支if条件语句**

其语法格式如下：

If [ 条件语句1 ];then

执行程序

elif [ 条件语句2 ];then # elif 其实是 else if的缩写

执行程序

……

else

执行程序 #当所有条件都不成立的时候，最后执行的程序

Fi

例如：判断输入的成绩是否合格，大于等于60则判定成绩合格反之则为不合格



运行结果如下：



1. **case语句**

case语句相当于一个多分支的if条件语句，case变量的值用来匹配多个value值，等匹配到对应的value值时，则执行相对应的程序，直到遇到“；；”为止，case语句以esac作为结束符。

其语法格式如下：

case 值 in

value1)

执行程序1

；；

value2 )

执行程序2

；；

……

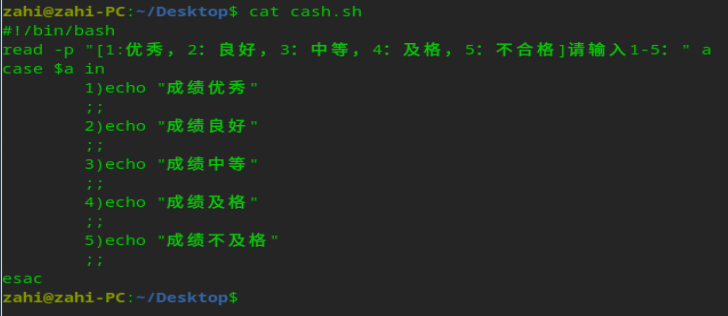
valuen )

执行程序n

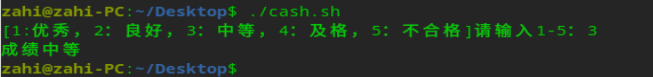
；；

Esac

如下面例子：



执行结果如下：



## 7.2.2 Shell编程---循环结构

Shell提供的循环结构有3中，分别为：for、while、util。

1. **for循环语句**

for循环语句用于在一个列表中执行有限次数的命令。for命令后跟一个自定义的变量、一个关键字in和一个字符串列表。第一次执行for循环语句的时候，字符串列表中的第一个字符会赋值给自定义变量，同时执行循环体，知道遇到done语句；第二次执行for循环语句的时候，会将字符串列表中的第二个字符赋值给自定义的变量，以此类推，知道字符串列表遍历完毕。

其语法格式如下：

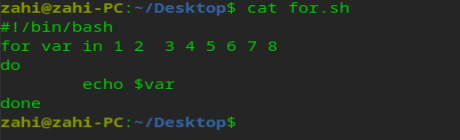
for 变量 [ in 列表]

do

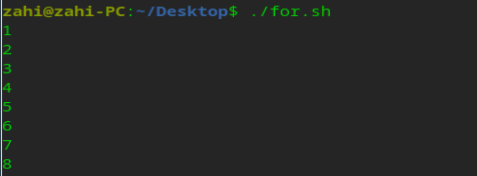
执行语句

done

例如，按照顺序输出列表中的数字。



执行结果如下：



1. **while循环语句**

while循环用于不断地执行一系列命令，一直到测试条件为false。

其语法格式为：

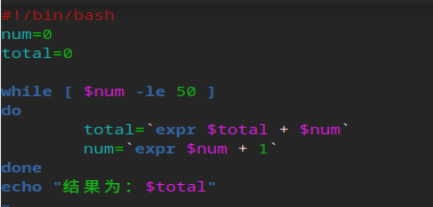
whiel 条件语句

do

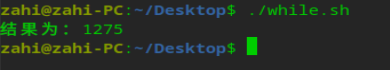
执行语句

done

例如，利用while循环求1到50的和：



执行结果如下：



1. **until循环语句**

until循环语句和while循环语句类似，区别在于util循环语句在条件为true的时候退出循环，反之则一直在循环体里面。我们while循环语句是当条件为false的时候推出循环体，反之一直在执行循环。

其语法格式如下：

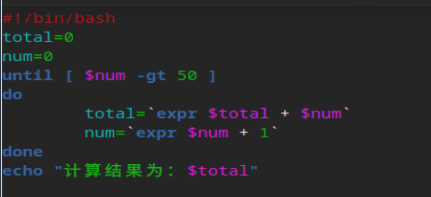
util 条件语句

do

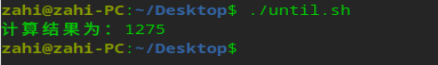
执行语句

done

例如，利用until循环求的1-50的总和：



执行结果如下：



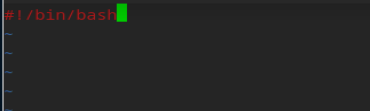
## 【任务实施】

## 7.2.4 项目实训：使用VM工具快速完成Shell脚本编程

第一步：利用vim命令打开一个 Shell脚本,如果脚本不存在则创建一个新的脚本。



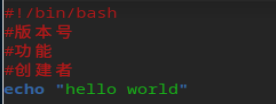
第二步：回车之后会进入脚本编辑界面，我们需要在文本编辑器里第一行输入#！/bin/bash，在输入之前编辑器无法输入任何信息需要我们按“i”在光标处插入才可以进行输入。



注意：第一行的内容指定了shell脚本解释器的路径，而且这个指定路径只能放在文件的第一行。第二行到第四行都是起到一个介绍作用，作用不如第一行。



之后我们就可以在编辑器里面写我们的第一个脚本了！



脚本编写完成我们只需要按“esc”进入命令模式之后再 shift+“:”进入底行编辑模式输入 wq保存退出！这样我们第一个脚本文件就编写完成了。

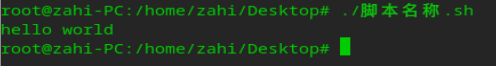
最后就是我们执行我们的脚本文件，脚本文件的执行有多种，这个在这个任务里面有提到过，不清楚的可以在前面进行翻阅。

C:\Users\Administrator\Documents\WeChat Files\zhangzhihao634584\FileStorage\Temp\1658585140331.png

出现这种情况是因为你没有执行权限所以在执行之前我们需要添加执行权限x，添加权限需要我们 用到chmod指令。

C:\Users\Administrator\Documents\WeChat Files\zhangzhihao634584\FileStorage\Temp\1658585235817.png

添加完毕之后我们执行脚本。



至此我们的脚本编写流程就全部完毕。

## 【任务回顾】

【知识点总结】

1. Shell流程分支控制语句，主要讲解了单分支if条件语句、多分支if条件语句、case语句。
2. Shell流程循环控制语句，主要讲解了for循环语句、while循环语句、until循环语句。

【思考与练习】

1. ( )不是Shell的循环控制结构。

A．for B.while C. switch D.until

1. 简述条件语句if和case之间的区别。
2. 编写Shell程序，分别利用for、while、until实现从整数1到100的和。

# 项目总结

本项目包含了4个部分

1. Shell变量，主要讲解了变量的使用、环境的变量、位置参数变量、预定义变量。
2. Shell运算符，主要讲解了算术运算符、关系运算符、布尔运算符、字符串运算符、逻辑运算符、文件测试运算符、$()和``、${}、$[]和$(())、[]、(())和[[]]。
3. Shell流程分支控制语句，主要讲解单分支和多分支if条件语句、case语句。
4. Shell流程循环控制语句，主要讲解了三种循环语句，for循环语句、while循环语句、until循环语句。

**项目十二 网络管理**

# 【项目介绍】

# 【项目框架】

# 任务 1 网络基础设置

## 【任务描述】

## 【知识储备】

## 12.1.1 计算机网络的分类

## 12.1.2 IP地址、域名

## 12.1.3 子网、子网掩码、网关

## 12.1.4 了解IPV4、IPV6

## 【任务实施】

## 12.1.5 图形化设置网络

## 【任务回顾】

【知识点总结】

【思考与练习】

# 任务 2 网络管理基础

## 【任务描述】

## 【知识储备】

## 12.2.1 connection修改

## 12.2.2 device管理

## 【任务实施】

## 12.2.3 命令行界面设置网络

## 【任务回顾】

【知识点总结】

【思考与练习】

# 任务 3 网络管理进阶

## 【任务描述】

## 【知识储备】

## 12.3.1 nmuti工具

## 12.3.2网络诊断

## 12.3.3网络下载

## 【任务实施】

## 12.3.4 设置并管理计算机系统网络

## 【任务回顾】

【知识点总结】

【思考与练习】

# 项目总结