Shell 极速入门教程

简介

- `Author` : *GoogTech*
- `Website` : *https://shell.googtech.io*
- `Email` : *GoogTech@qq.com*
- `Date` : *2021,6,26 ~ 2021,6,27*

以下 Shell 程序均在 Windows 平台上编写😁, 开发工具为 VSCode, 开发环境为 GitBash.

目录

Hello Shell

```
#!\Git\Git\bin\bash

echo "hello world"
```

运行结果如下所示：

```
hello world
```

- `echo`：此命令用于向窗口输出文本.
- `#!`：其后路径所指的程序即是解释此脚本文件的程序解释器.

注释

单行注释

```
#!\Git\Git\bin\bash

# echo "Hello World"
```

多行注释

```
#!\Git\Git\bin\bash

:<<EOF
echo "Hello World"
EOF
```

---

echo 命令

显示普通字符串

```
#!\Git\Git\bin\bash

echo "Hello World"
```

运行结果如下所示：

```
Hello World
```

显示转义字符串

```
#!\Git\Git\bin\bash

echo "\"Hello World"\"
```

运行结果如下所示：

```
"Hello World"
```

显示变量

```
#!\Git\Git\bin\bash

alias="GoogTech"

echo ${alias}
```

运行结果如下所示：

```
GoogTech
```

显示结果定向到指定文件

```
#!\Git\Git\bin\bash

echo "echo some informations into the specified file" > file.txt
```

运行上述程序后，该脚本首先会在当前目录下创建一个名为 `file.txt` 的文件，然后将字符串写入到该文件中，文件中的内容如下所示：

```
echo some informations into the specified file
```

显示命令执行的结果

```
#!\Git\Git\bin\bash

echo `pwd`
```

运行结果如下所示：

```
/f/Git/workbench/shell
```

- 注：程序中使用的是反引号 ``，而非单引号 ''.

变量

使用变量

```
#!\Git\Git\bin\bash

alias="GoogTech"

echo $alias
echo ${alias}
```

运行结果如下所示：

```
GoogTech
GoogTech
```

- `$`：用于引用已定义的变量.
- `{}`：变量名外的花括号是可选的, 加花括号是为了帮助解释器识别变量的边界, 故建议给所有变量加上花括号.

只读变量

```
#!\Git\Git\bin\bash

myUrl="https://googtech.io"

readonly myUrl

myUrl="https://github.com/googtech"
```

运行结果如下所示：

```
f:\Git\workbench\shell\readonlyTest.sh: line 7: myUrl: readonly variable
```

- `readonly`：此命令可将变量定义为只读变量, 进而其值不能被改变.

删除变量

```
#!\Git\Git\bin\bash

url="https://googtech.io"

unset url

echo $url
```

运行结果如下所示：

```
# 无任何输出哈~
```

- `unset`：用于删除变量, 但此命令不能删除只读变量.

字符串

单引号与双引号

```
#!\Git\Git\bin\bash

alias='GoogTech'

intro="Hey guy I'm \"$alias\" !"

echo ${intro}
```

运行结果如下所示：

```
Hey guy I'm "GoogTech" !
```

- `' '`：单引号字符串中的变量是无效的，而且无法使用转义符.
- `" "`：双引号中可以引用变量，而且可以使用转义符.

拼接字符串

```
#!\Git\Git\bin\bash

alias="GoogTech"

greeting1="hey "$alias" !"
greeting2="hello ${alias} !"

echo ${greeting1}
echo ${greeting2}
```

运行结果如下所示：

```
hey GoogTech !
hello GoogTech !
```

获取字符串长度

```
#!\Git\Git\bin\bash

alias="GoogTech"

echo "The length of my alias: "${#alias}
```

运行结果如下所示：

```
The length of my alias: 8
```

提取字串

```
#!\Git\Git\bin\bash

alias="GoogTech"

# 从字符串的第 1 个字符开始截取 4 个字符
echo "${alias:0:4}le"
```

运行结果如下所示：

```
Google
```

查找子串

```
#!\Git\Git\bin\bash

alias="GoogTech"

# 查找字符 T 或 h 的位置，即返回第一个出现字符的 index
echo `expr index "$alias" Tech`
```

运行结果如下所示：

```
5
```

- 注：脚本中使用的是`反引号`，而非`单引号`.

数组

读取数组

```
#!\Git\Git\bin\bash

# 定义数组
arr=(1 2 3 4 5 6)

# 添加数组元素
arr[6]=7

# 打印数组中下标值为 0 和 5 的数组元素
echo "${arr[0]}, ${arr[5]}"

# 打印数组中的所有的元素，或使用 echo ${arr[*]}
echo ${arr[@]}
```

运行结果如下所示：

```
1, 6
1 2 3 4 5 6 7
```

获取数组长度

```
#!\Git\Git\bin\bash

arr=(a abc abcd abcde abcdef abcdefg)

# 获取数组中元素的个数，或使用 #arr[*]
length1=${#arr[@]}
echo ${length1}

# 获取数组中指定下标元素的长度
length3=${#arr[5]}
echo ${length3}
```

运行结果如下所示：

```
6
7
```

传参

```
#!\Git\Git\bin\bash

echo "The program file name: $0"

echo "The first parm: $1"
echo "The second parm: $2"
echo "The third parm: $3"

echo "the amount of parm: $#"
```

运行结果如下所示：

```
$ bash parmTest.sh a ab abc
The program file name: parmTest.sh
The first parm: a
The second parm: ab
The third parm: abc
the amount of parm: 3
```

运算符

注意点：

1. 表达式和运算符之间需要有空格.
2. 完整的表达式要被 ``` 包含.

算术运算符

假定变量 a 为 10, 变量 b 为 20.

运算符	说明	举例
+	加法	`expr $a + $b` 结果为 30
-	减法	`expr $a - $b` 结果为 -10
*	乘法	`expr $a \* $b` 结果为 200
/	除法	`expr $b / $a` 结果为 2
%	取余	`expr $b % $a` 结果为 0
=	赋值	a=$b 将把变量 b 的值赋给 a
==	相等。用于比较两个数字，相同则返回 true	[$a == $b] 返回 false
!=	不相等。用于比较两个数字，不相同则返回 true	[$a != $b] 返回 true

- 乘号 `*` 前面必须加反斜杠 `\` 转义符才能实现乘法运算.

关系运算符

假定变量 a 为 10, 变量 b 为 20.

运算符	说明	举例
-eq	检测两个数是否相等, 相等返回 true	[$a -eq $b] 返回 false
-ne	检测两个数是否不相等, 不相等返回 true	[$a -ne $b] 返回 true
-gt	检测左边的数是否大于右边的, 如果是则返回 true	[$a -gt $b] 返回 false
-lt	检测左边的数是否小于右边的, 如果是则返回 true	[$a -lt $b] 返回 true
-ge	检测左边的数是否大于等于右边的, 如果是则返回 true	[$a -ge $b] 返回 false
-le	检测左边的数是否小于等于右边的, 如果是则返回 true	[$a -le $b] 返回 true

- 关系运算符只支持数字, 不支持字符串.

布尔运算符

假定变量 a 为 10, 变量 b 为 20.

运算符	说明	举例
!	非运算, 表达式为 true 则返回 false, 否则返回 true	[! false] 返回 true
-o	或运算, 有一个表达式为 true 则返回 true.	[$a -lt 20 -o $b -gt 100] 返回 true.
-a	与运算, 两个表达式都为 true 才返回 true.	[$a -lt 20 -a $b -gt 100] 返回 false.

逻辑运算符

假定变量 a 为 10, 变量 b 为 20.

运算符	说明	举例
&&	逻辑的 AND	[[$a -lt 100 && $b -gt 100]] 返回 false
||	逻辑的 OR	[[$a -lt 100 || $b -gt 100]] 返回 true

字符串运算符

假定变量 a 为 "abc", 变量 b 为 "efg".

运算符	说明	举例
=	检测两个字符串是否相等, 相等返回 true	[$a = $b] 返回 false
!=	检测两个字符串是否不相等, 不相等返回 true	[$a != $b] 返回 true
-z	检测字符串长度是否为0, 为 0 返回 true	[-z $a] 返回 false
-n	检测字符串长度是否不为 0, 不为 0 返回 true	[-n "$a"] 返回 true
$	检测字符串是否为空, 不为空返回 true	[$a] 返回 true

文件测试运算符

操作符	说明	举例
-b file	检测文件是否是块设备文件, 如果是, 则返回 true	[-b $file] 返回 false
-c file	检测文件是否是字符设备文件, 如果是, 则返回 true	[-c $file] 返回 false
-d file	检测文件是否是目录, 如果是, 则返回 true	[-d $file] 返回 false
-f file	检测文件是否是普通文件(既不是目录, 也不是设备文件), 如果是, 则返回 true	[-f $file] 返回 true
-g file	检测文件是否设置了 SGID 位, 如果是, 则返回 true	[-g $file] 返回 false
-k file	检测文件是否设置了粘着位(Sticky Bit), 如果是, 则返回 true	[-k $file] 返回 false
-p file	检测文件是否是有名管道, 如果是, 则返回 true.	[-p $file] 返回 false
-u file	检测文件是否设置了 SUID 位, 如果是, 则返回 true	[-u $file] 返回 false
-r file	检测文件是否可读, 如果是, 则返回 true	[-r $file] 返回 true
-w file	检测文件是否可写, 如果是, 则返回 true	[-w $file] 返回 true
-x file	检测文件是否可执行, 如果是, 则返回 true	[-x $file] 返回 true
-s file	检测文件是否为空(文件大小是否大于0), 不为空返回 true	[-s $file] 返回 true
-e file	检测文件(包括目录)是否存在, 如果是, 则返回 true	[-e $file] 返回 true

printf 命令

printf 由 POSIX 标准所定义, 因此使用 printf 的脚本比使用 echo 移植性好.

语法

```
printf format-string [arguments...]
```

- `format-string` : 为格式控制字符串
- `arguments` : 为参数列表

```
#!\Git\Git\bin\bash

printf "Hello World \n"
```

运行结果如下所示 :

```
Hello World
```

常用格式替代符

```
#!\Git\Git\bin\bash

printf "%-10s %-8s %-4s\n" Name Gender Weight
printf "%-10s %-8s %-4s\n" Goog male 65.123
printf "%-10s %-8s %-4s\n" Tech female 50.456
```

运行结果如下所示 :

```
Name       Gender   Weight
Goog       male     65.123
Tech       female   50.456
```

- `%s` : 输出字符串
- `%d` : 输出整型数字
- `%c` : 输出一个字符
- `%f` : 输出实数的小数形式
- `%-4.2f` : 指格式化为小数, 其中 `.2` 指保留 2 位小数
- `%-10s` : 指一个宽度为 10 个字符(`-` 表示左对齐, 没有则表示右对齐)

流程控制

if

```
#!\Git\Git\bin\bash

a=1
b=2

if [ $a == $b ]
then
    echo "a = b"
elif [ $a -gt $b ]
then
    echo "a > b"
elif [ $a -lt $b ]
then
    echo "a < b"
else
    echo "no result"
fi
```

运行结果如下所示：

```
a < b
```

for

```
#!\Git\Git\bin\bash

# 顺序打印当前列表中的数字
for num in 1 2 3 4 5 6
do
    echo "The number: $num"
done

# 顺序打印字符串中的字符
for str in This is a string
do
    echo "The string: $str"
done
```

运行结果如下所示：

```
The number: 1
The number: 2
The number: 3
The number: 4
The number: 5
The number: 6
The string: This
The string: is
The string: a
The string: string
```

while

```
#!\Git\Git\bin\bash

int=1

while(($int<=3))
do
    echo ${int}
    let "int++"
done
```

运行结果如下所示：

```
1
2
3
```

- `let`：此命令是 `bash` 中用于计算的工具，用于执行一个或多个表达式，变量计算中不需要加上 `$` 来表示变量. 如果表达式中包含了空格或其它特殊字符，则必须引起来.

until

util 循环与 while 循环在处理方式上刚好相反.

```
#!\Git\Git\bin\bash

num = 1

until [ $num -gt 3 ]
do
    echo ${num}
   ((num++))
done
```

运行结果如下所示：

```
1
2
3
```

case

```
#!\Git\Git\bin\bash

echo 'Please input a number: '
read num
case $num in
```

```
    1) echo 'The number you inputed: 1'
    ;;
    2) echo 'The number you inputed: 2'
    ;;
    3) echo 'The number you inputed: 3'
    ;;
    *) echo 'You inputed others number.'
    ;;
esac
```

运行结果如下所示：

```
$ bash caseTest.sh
Please input a number:
2
The number you inputed: 2

$ bash caseTest.sh
Please input a number:
4
You inputed others number.
```

break

```
#!\Git\Git\bin\bash

while :
do
    echo -n "please input a number between 1 and 3: "
    read num
    case $num in
        1|2|3) echo "the number you inputed: ${num}"
        ;;
        *) echo "you inputed others number and game over !"
            break
        ;;
    esac
done
```

运行结果如下所示：

```
$ bash breakTest.sh
please input a number between 1 and 5: 1
the number you inputed: 1
please input a number between 1 and 5: 2
the number you inputed: 2
please input a number between 1 and 5: 3
the number you inputed: 3
please input a number between 1 and 5: 4
you inputed others number and game over !
```

continue

```
#!\Git\Git\bin\bash

while :
do
    echo -n "please input a number between 1 and 3: "
    read num
    case $num in
        1|2|3) echo "the number you inputed: ${num}"
        ;;
        *) echo "you inputed others number and game over !"
            continue
            echo "This Line Never Be Executed !"
        ;;
    esac
done
```

运行结果如下所示：

```
$ bash continueTest.sh
please input a number between 1 and 3: 1
the number you inputed: 1
please input a number between 1 and 3: 2
the number you inputed: 2
please input a number between 1 and 3: 3
the number you inputed: 3
please input a number between 1 and 3: 4
you inputed others number and game over !
please input a number between 1 and 3:
```

函数

语法

```
[ function ] funname [()]
{
    action;
    [return int;]
}
```

- `function` 关键字可加可不加.

无参函数

```
#!\Git\Git\bin\bash

funcWithReturn() {
    echo "Please input the first number: "
    read num1
    echo "Please input the second number: "
    read num2
    return $(($num1+$num2))
}

funcWithReturn
echo "The sum: $?"
```

运行结果如下所示：

```
Please input the first number:
1
Please input the second number:
2
The sum: 3
```

- `$?`：用来获取调用函数的返回值.

函数参数

```
#!\Git\Git\bin\bash

funcWithParam() {
    echo "The first parm: $1"
    echo "The second parm: $2"
    echo "The thrid parm: $3"
    echo "the sixth parm: ${6}"
    echo "The amount of parm: $#"
    echo "All parms: $*"
}

funcWithParam 10 9 8 7 6 5
```

运行结果如下所示：

```
The first parm: 10
The second parm: 9
The thrid parm: 8
the sixth parm: 5
The amount of parm: 6
All parms: 10 9 8 7 6 5
```

- 当 n>=10 时, 需要使用 `${n}` 来获取参数.

处理参数的特殊字符

参数处理	说明
$#	传递到脚本或函数的参数个数
$*	以一个单字符串显示所有向脚本传递的参数
$$	脚本运行的当前进程ID号
$!	后台运行的最后一个进程的ID号
$@	与$*相同, 但是使用时加引号, 并在引号中返回每个参数
$-	显示Shell使用的当前选项, 与set命令功能相同
$?	显示最后命令的退出状态, 0表示没有错误, 其他任何值表明有错误

输入/输出重定向

输出重定向

```
#!\Git\Git\bin\bash

echo "echo string into user.txt file" > echo.txt
```

运行上述程序后, 该脚本首先会在当前目录下创建一个名为 `echo.txt` 的文件, 然后将字符串写入到该文件中, 文件中的内容如下所示 :

```
echo string into user.txt file
```

- `>` : 该输入重定向将覆盖原文件中的内容, 若需要将新内容添加在文件的末尾, 则可以使用 `>>` 操作符, 实例如下所示.

```
#!\Git\Git\bin\bash

echo "echo new string into user.txt file" >> echo.txt
```

运行结果如下所示(echo.txt) :

```
echo string into user.txt file
echo new string into user.txt file
```

输入重定向

```
#!\Git\Git\bin\bash

# 统计 echo.txt 文件中的行数
wc -l < echo.txt
```

运行结果如下所示：

```
2
```

/dev/null 文件

如果希望执行某个命令, 但不希望在屏幕上显示输出结果, 则可以将输出重定向到 `/dev/null`：

```
$ command > /dev/null

# 下述写法可以屏蔽 stdout 和 stderr
$ command > /dev/null 2>&1
```

- `/dev/null`：它是一个特殊的文件, 写入到它的数据都会被丢弃, 故将命令重定向到它, 会起到 "禁止输出" 的效果哟.

重定向命令列表

命令	说明
command > file	将输出重定向到 file
command < file	将输入重定向到 file
command >> file	将输出以追加的方式重定向到 file
n > file	将文件描述符为 n 的文件重定向到 file
n >> file	将文件描述符为 n 的文件以追加的方式重定向到 file
n >& m	将输出文件 m 和 n 合并
n <& m	将输入文件 m 和 n 合并
<< tag	将开始标记 tag 和结束标记 tag 之间的内容作为输入

- `标准输入文件(stdin)`：stdin的文件描述符为0, Unix程序默认从stdin读取数据.
- `标准输出文件(stdout)`：stdout的文件描述符为1, Unix程序默认向stdout输出数据.
- `标准错误文件(stderr)`：stderr的文件描述符为2, Unix程序会向stderr流中写入错误信息.

文件包含

Shell 可以包含外部脚本, 进而可以很方便地封装一些公用的代码作为一个独立的文件.

1. util.sh

```
#!\Git\Git\bin\bash

funcWithReturn() {
    return $(($1+$2))
}
```

2. test.sh

```
#!\Git\Git\bin\bash

source ./util.sh

funcWithReturn 1 2

echo "The sum: $?"
```

运行结果如下所示 :

```
The sum: 3
```

- 注 : 被包含文件不需要可执行权限.

基础练习题

1. 编写 shell 脚本, 用户输入整数, 如果大于等于0, 屏幕打印 "the test value is greater than 0", 如果小于0, 屏幕打印 "the test value is less than 0".

```
#!\Git\Git\bin\bash

###
 # @Author: GoogTech
 # @Email: googtech@qq.com
 # @Date: 2021-06-27
 # @Site: https://shell.googtech.io
 #
 # 1. 编写 shell 脚本, 用户输入整数, 如果大于等于0,
 #    屏幕打印 "the test value is greater than 0",
 #    如果小于0, 屏幕打印 "the test value is less than 0".
###

read -r -p "Please input a number : " num
if ((num>0))
then
    echo "The test value is greater than 0."
```

```
else
    echo "The test value is less than 0."
fi
```

运行结果如下所示：

```
$ bash Test.sh
Please input a number : 0
The test value is less than 0.

$ bash Test.sh
Please input a number : 1
The test value is greater than 0.
```

- read without -r will mangle backslashes

2. 编写脚本, 检查密码, 如果用户三次输入密码均错误, 则退出脚本.

```
#!\Git\Git\bin\bash

###
 # @Author: GoogTech
 # @Email: googtech@qq.com
 # @Date: 2021-06-27
 # @Site: https://shell.googtech.io
 #
 # 2. 编写脚本, 检查密码, 如果用户三次输入密码均错误, 则退出脚本.
###

COUNT=0
NUMBER=3
PASSWORD=GoogTech.IO

while [ $COUNT -lt $NUMBER ]
do
read -r -p "Please input your pwd: " PWD
COUNT=$((COUNT+1))
if [ "$PWD" == $PASSWORD ]
then
    echo "The password is corret."
    exit
else
    echo "The password is incorret."
    continue
fi
done
```

运行结果如下所示：

```
$ bash Test.sh
Please input your pwd: GoogTech.io
The password is incorret.
Please input your pwd: GoogTech.Io
The password is incorret.
Please input your pwd: GoogTech.iO
The password is incorret.

$ bash Test.sh
Please input your pwd: GoogTech.IO
The password is corret.
```

3. 备份 /etc 目录, 备份的压缩包有时间标志.

```
#!\Git\Git\bin\bash

###
 # @Author: GoogTech
 # @Date: 2021-06-27
 # @Email: googtech@qq.com
 # @Site: https://shell.googtech.io
 #
 # 3. 备份/etc目录, 备份的压缩包有时间标志.
###

# 在根目录创建备份目录
mkdir -p /backup

# 备份 etc/ 目录, 并在压缩包文件名中添加时间标志
cd /
tar czf /backup/etc_"$(date +%F_%M)".tar.gz etc/
```

运行结果如下所示：

```
$ ls -l /backup/
total 508
-rw-r--r-- 1 GoogTech 197121 519700 Jun 27 10:20 etc_2021-06-27_20.tar.gz
```

- Quote this to prevent word splitting

4. 尝试从文件读取数据前, 测试下文件 /etc/shadow 是否可读, 首先判断是否是存在, 且是一个文件, 如果不存在输出 "sorry, the file /etc/shadow does not exist". 如果存在判断是否可读, 如果可读, 打印文件的最后一行, 如果不可读, 输出 "sorry, I am unable to read the /etc/shadow file".

```
#!\Git\Git\bin\bash

###
 # @Author: GoogTech
 # @Date: 2021-06-27
 # @Email: googtech@qq.com
 # @Site: https://shell.googtech.io
 #
```

```
 # 4. 尝试从文件读取数据前，测试下文件 /etc/shadow 是否可读，
 #    首先判断是否是存在，且是一个文件，
 #    如果不存在输出 "sorry, the file /etc/shadow  does not exist".
 #    如果存在判断是否可读，如果可读，打印文件的最后一行，
 #    如果不可读，输出 "sorry, I am unable to read the /etc/shadow file".
###

DIRECTORY_PATH="/etc"
FILE_PATH="/etc/shadow"

if [ ! -d "$DIRECTORY_PATH" ]
then
    echo "sorry, the file /etc/shadow does not exist."
elif [ ! -r "$FILE_PATH" ]
then
    echo "sorry, I am unable to read the /etc/shadow file."
else
    tail -n 1 $FILE_PATH
fi
```

运行结果如下所示：

```
$ echo "echo string into the /etc/shadow file" >> /etc/shadow

$ cat /etc/shadow
echo string into the /etc/shadow file

$ bash Test.sh
echo string into the /etc/shadow file
```

5. 创建一个 Shell 脚本, 它从用户那里接收 10 个数, 然后求出其总和、最大值及最小值.

```
#!\Git\Git\bin\bash

###
 # @Author: GoogTech
 # @Date: 2021-06-27
 # @Email: googtech@qq.com
 # @Site: https://shell.googtech.io
 #
 # 5. 创建一个 Shell 脚本，它从用户那里接收 10 个数，
 #    然后求出其总和、最大值及最小值.
###

SUM=0
COUNT=1
NUMBER=10

read -r -p "Please input a number: " NUM
MIN=${NUM}
MAX=${NUM}
SUM=$((SUM+NUM))

while [ $COUNT -lt $NUMBER ]
do
```

```
    read -r -p "Please input a number: " NUM
    if((MAX < NUM))
    then
        MAX=${NUM}
    fi
    if((MIN > NUM))
    then
        MIN=${NUM}
    fi
    SUM=$((SUM+NUM))
    COUNT=$((COUNT+1))
done

echo "MIN=$MIN, MAX=$MAX, SUM=$SUM"
```

运行结果如下所示：

```
Please input a number: 1
Please input a number: 2
Please input a number: 3
Please input a number: 4
Please input a number: 5
Please input a number: 6
Please input a number: 7
Please input a number: 8
Please input a number: 9
Please input a number: 10
MIN=1, MAX=10, SUM=55
```

6. 写一个 Shell 脚本, 检查给出的串是否为回文.

```
#!\Git\Git\bin\bash

###
 # @Author: GoogTech
 # @Date: 2021-06-27
 # @Email: googtech@qq.com
 # @Site: https://shell.googtech.io
 #
 # 6. 写一个 Shell 脚本, 检查给出的串是否为回文.
###

read -r -p "Please input a string: " INPUT

len=${#INPUT}
count=$((len/2))

# 从左到中遍历字符串
for i in $(seq "$count")
do
    # 从右到中逐个获取字符串中的字符
    lasti=$((len-i+1))
    # 从左到右, 从右到左依次比较字符串中的字符
    first=$(echo "$INPUT"|cut -c "$i")
    second=$(echo "$INPUT"|cut -c $lasti)
```

```
    # 判断字符是否相等
    if [ "$first" != "$second" ]
    then
        echo "no, it's not a palindrome."
        exit
    fi
done

echo "yes, it's a palindrome."
```

运行结果如下所示：

```
Please input a string: GoogTech
no, it's not a palindrome.

Please input a string: GooG
yes, it's a palindrome.
```

- `seq`：此命令用于以指定增量从首数开始打印数字到尾数.
- `cut`：此命令用于在文件当中提取符合条件的列，`-c` 指定具体的字符.

7. 求阶乘

```
#!\Git\Git\bin\bash

###
 # @Author: GoogTech
 # @Date: 2021-06-27
 # @Email: googtech@qq.com
 # @Site: https://shell.googtech.io
 #
 # 7. 求阶乘
###

SUM=1

read -r -p "Please input a bigger number than 0 bro: " NUM

for i in $(seq 1 "$NUM")
do
    SUM=$((SUM*i))
done

echo "The Sum: ${SUM}"
```

运行结果如下所示：

```
Please input a bigger number than 0 bro: 5
The Sum: 120
```

8. 打印九九乘法表

```
#!\Git\Git\bin\bash

###
 # @Author: GoogTech
 # @Date: 2021-06-27
 # @Email: googtech@qq.com
 # @Site: https://shell.googtech.io
 #
 # 8. 打印九九乘法表
###

for ((i=1;i<=9;i++))
do
    for ((j=1;j<=9;j++))
    do
        # 当 j 小于或等于 i 才输出哈
        [ "${j}" -le "${i}" ] && printf "%s\t" "${j}*${i}=$((i*j))"
    done
    echo ""
done
```

运行结果如下所示：

```
1*1=1
1*2=2   2*2=4
1*3=3   2*3=6   3*3=9
1*4=4   2*4=8   3*4=12  4*4=16
1*5=5   2*5=10  3*5=15  4*5=20  5*5=25
1*6=6   2*6=12  3*6=18  4*6=24  5*6=30  6*6=36
1*7=7   2*7=14  3*7=21  4*7=28  5*7=35  6*7=42  7*7=49
1*8=8   2*8=16  3*8=24  4*8=32  5*8=40  6*8=48  7*8=56  8*8=64
1*9=9   2*9=18  3*9=27  4*9=36  5*9=45  6*9=54  7*9=63  8*9=72  9*9=81
```

致谢

> runoob : *https://www.runoob.com/linux/linux-shell.html*

总结

求知若饥, 虚心若愚, 愿你像一个向日葵, 充满阳光, 积极向上！