# shell脚本编写教程(概念+代码)

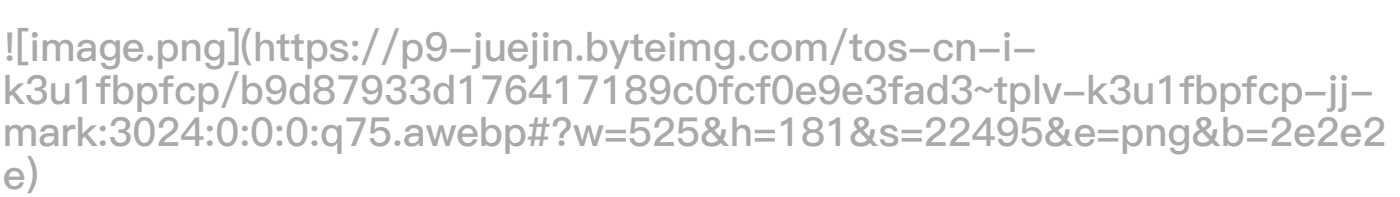
在一个合格的开发工程师的工具箱中，精通编写和理解 Shell 脚本是至关重要的。本文将从变量定义、条件判断和流程控制几个方面详细说明如何编写 Shell 脚本，并提供相应的代码示例。

## 系统变量和特殊变量

首先，让我们来了解 Linux 系统中的系统变量和用户变量。

* **系统变量** 存储在 `/etc/profile` 文件中，这些变量对所有用户都生效。
* **用户变量** 存储在 `~/.bashrc` 文件中，只对当前用户生效。

除了系统和用户变量外，我们还可以在 Shell 中自定义变量。通过简单的赋值操作即可定义变量，例如 `变量=值`。如果需要，可以使用 `unset` 命令来撤销已经定义的变量。




### 特殊变量

在Linux系统中是有一些特殊变量的。

* `$n` （功能描述：n为数字，`$0`代表该脚本名称，`$1`-`$9`代表第一到第九个参数，十以上的参数，十以 上的参数需要用大括号包含，如`${10}`）
* `$#` （功能描述：获取所有输入参数个数，常用于循环）
* `$*` （功能描述：这个变量代表命令行中所有的参数，`$*`把所有的参数看成一个整体）
* `$@` （功能描述：这个变量也代表命令行中所有的参数，不过`$@`把每个参数区分对待）
* `$? ` （功能描述：最后一次执行的命令的返回状态。如果这个变量的值为

0，证明上一个命令正确执 行；如果这个变量的值为非0（具体是哪个数，由命令自己来决定），则证明上一个命令执行不正确了。）

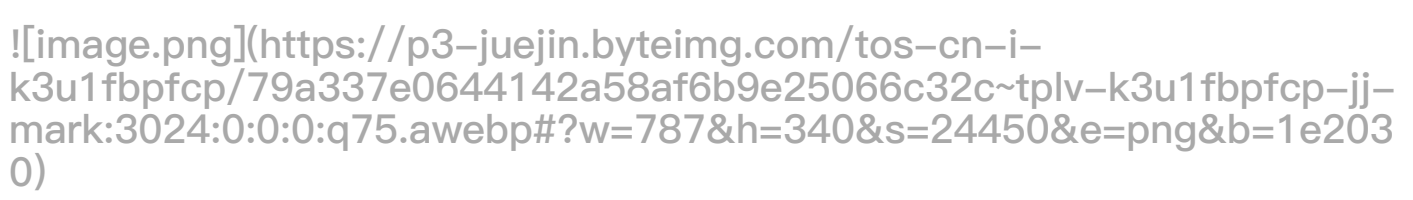
我们可以通过编写脚本来验证.

1. 创建`特殊变量.sh`这个脚本文件
2. 复制粘贴一下内容进`特殊变量.sh`脚本文件

```
#! /bin/bash

# 打印命令行参数
echo "\$1=\$2=\$3 输出："
echo "$1 $2 $3"

# 打印参数个数
echo "\$# 输出："
echo "参数个数: $#"

# 打印 $* 和 $@
echo "\$* 输出："
echo $*

echo "\$@ 输出："
echo $@

# 打印上一次命令的返回状态
echo "\$? 输出："
echo "上一次命令的返回状态: $?"

# 示例：触发一个执行失败的命令
# cat 命令试图读取一个不存在的文件
echo "触发一个执行失败的命令..."
cat ./error.txt

# 打印上一次命令的返回状态
echo "\$? 输出："
echo "上一次命令的返回状态: $?"
```

3. 打开一个终端窗口
4. 输入`chmod +x 特殊变量.sh`，确保`ll`命令查看到`特殊变量.sh`有可执行权限

5. `./特殊变量.sh a1 a2 a3`执行脚本,后面的参数可以自己定义(包括参数个数和参数名称)
6. 查看执行结果




条件判断
====

[ condition ]（注意condition前后要有空格）

1. 两个整数之间比较
1. = 字符串比较
2. -lt 小于（less than）
3. -le 小于等于（less equal）
4. -eq 等于（equal）
5. -gt 大于（greater than）
6. -ge 大于等于（greater equal）
7. -ne 不等于（Not equal）
2. 按照文件权限进行判断
1. -r 有读的权限（read）
2. -w 有写的权限（write）
3. -x 有执行的权限（execute）
3. 按照文件类型进行判断
1. -f 文件存在并且是一个常规的文件（file）
2. -e 文件存在（existence）
3. -d 文件存在并是一个目录（directory）

多条件判断使用`&&`和`||`来连接：

* `&&` 表示前一条命令执行成功时，才执行后一条命令
* `||` 表示上一条命令执行失败后，才执行下一条命令

接下来进行实操

1. 创建一个`条件判断.sh`,粘贴以下内容

```
#! /bin/bash

# 注意[]两边要加空格

# 判断输入参数是否大于2，如果大于就echo输出一句话
[ $# -gt 2 ] && echo "参数个数大于2" || echo 参数个数少于等于2个""

[ -e /home/meifannao/ping.txt ] && echo "文件存在" || echo "文件不存在"
```

2. 打开一个终端窗口
3. 输入`chmod +x 条件判断.sh`，确保`ll`命令查看到`条件判断.sh`有可执行权限
4. `./条件判断.sh a1 a2 a3`执行脚本,后面的参数可以自己定义(包括参数个数和参数名称)
5. 查看执行结果

if条件判断即案例
---

在bash中需要遵顼以下格式编写if语句

```
if [ 条件判断式 ];then
    程序
fi
```

或者

```
if [ 条件判断式 ]
    then
        程序
fi
```

接下来我们编写一个案例，

* 如果年龄小于18输出”未成年“
* 如果年龄大于18并且小于35，输出”青年”
* 如果年龄大于35输出”中老年“

可以先自己尝试编写shell脚本.(同时判断两个条件用`-a`连接)

参考脚本:

```
#! /bin/bash

if [ $1 -lt 18 ]; then
    echo ”未成年”
elif [ $1 -ge 18 -a $1 -lt 35 ]; then
    echo ”青年”
else
    echo ”中老年”
fi
```

流程控制(for,while,case)
==========

case
----

在 Bash 中，我们使用 `case` 语句来进行多条件匹配，其格式如下：

```
case $变量名 in
    ”值1”)
        如果变量的值等于值1，则执行程序1
    ;;
    ”值2”)
        如果变量的值等于值2，则执行程序2
    ;;
    …省略其他分支…
    *)
```

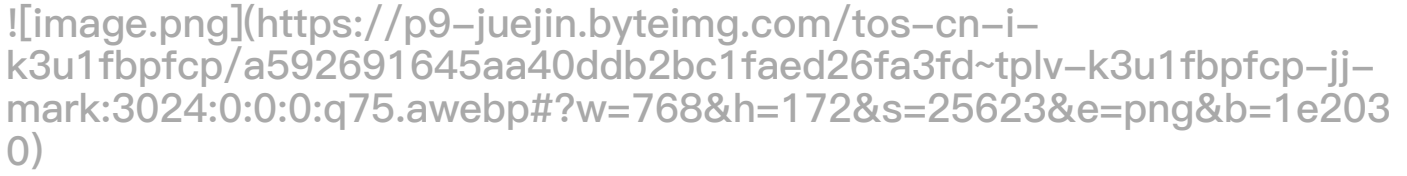
```
        如果变量的值都不是以上的值，则执行此程序
    ;;
esac

```

* case行尾必须为单词`“in”`，每一个模式匹配必须以右括号`“）”`结束。
* 双分号`“;;”`表示命令序列结束，相当于C++中的break。
* 最后的`“*）”`表示默认模式，相当于C++中的default。

当我们需要匹配输入参数是否符合预期时，可以使用 `case` 语句。以下是示例代码：

```
#! /bin/bash

case $1 in
    ”start”)
        echo ”输入了start”
    ;;
    ”end”)
        echo ”输入了end”
    ;;
    *)
        echo ”输入了未知”
    ;;
esac

```

这个脚本根据输入参数的值进行不同的操作，如果输入参数为 `”start”`，则输出 `”输入了start”`；如果输入参数为 `”end”`，则输出 `”输入了end”`；如果输入参数为其他值，则输出 `”输入了未知”`。




for循环
---

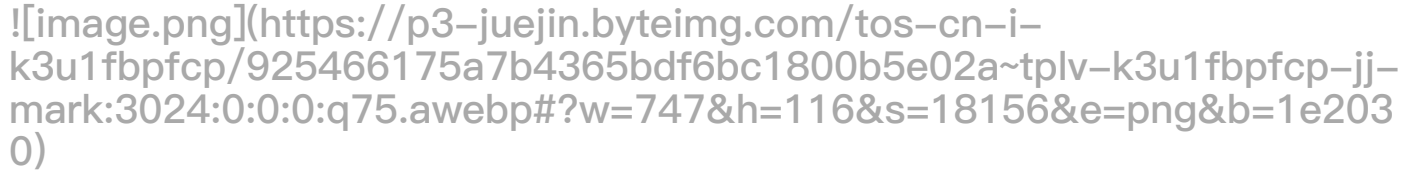
for循环的语法非常简单和c语言的for循环差不多，如下语法

```
for (( 初始值;循环控制条件;变量变化 ))
    do
        程序
    done
```

接下来，你可以尝试编写一个从1加到100的程序，参考如下：

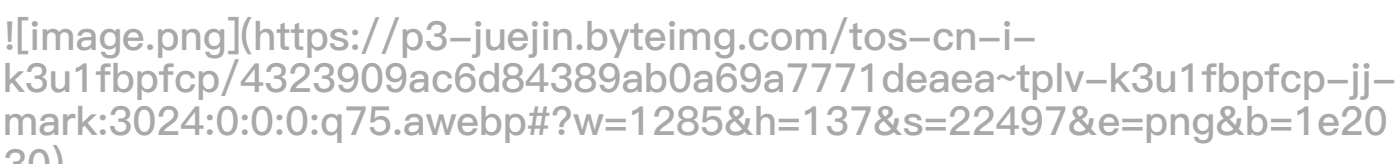
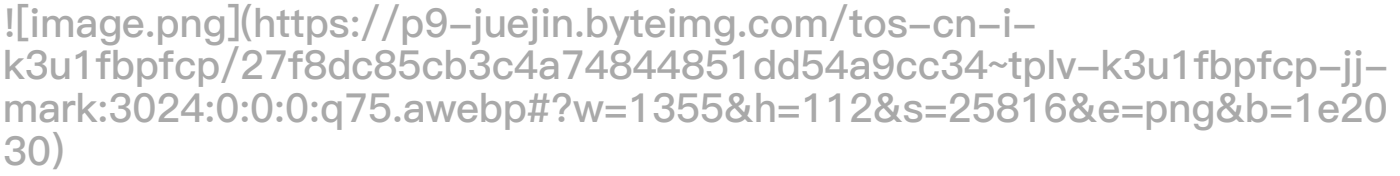
```
#! /bin/bash

# 计算从1到100的和

for ((i=0;i<=100;i++)); do
    s=$[$s+$i]
done

echo "从1到100的和是$s"
```

输出如下：




while循环
---

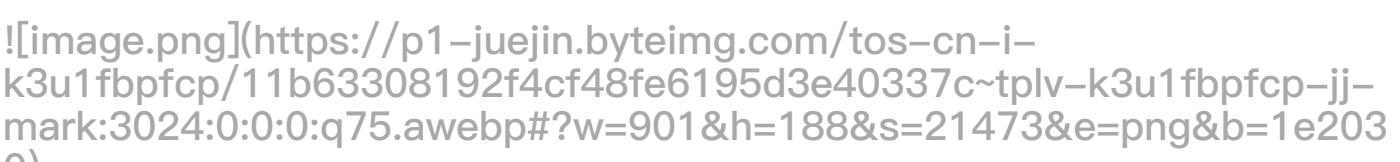
```
while [ 条件判断式 ]
    do
        程序
    done
```

```

可以通过while的语法编写从1加到100的程序,参考如下

```
#!/bin/bash
s=0
i=1
while [ $i -le 100 ]
    do
        s=$[$s+$i]
        i=$[$i+1]
done
echo $s
```

bash函数
======

bash是有一些系统函数的，比如`dirname`可以获得文件的绝对路径




`basename`命令会删掉所有的前缀包括最后一个 （‘/’）字符，然后将字符串显示出来。




现在我们可以通过shell编程，将一个路径下所有txt文件后缀改为.sh文件。比如在我的`/home/meifannao/data/test`目录下进行操作

```

mark:3024:0:0:0:q75.awebp#?w=595&h=112&s=8860&e=png&b=1e2030
)

参考代码:

```
#! /bin/bash

# 把/home/meifannao/data/test目录下所有后缀为.txt文件改为.sh

dir="/home/meifannao/data/test/"

for f in `ls /home/meifannao/data/test/*.txt`; do
    # f 就是其中的一个txt文件
    filename=`basename $f .txt`
    dist_filename=$filename".sh"

    mv $f $dir$dist_filename
done
```




自定义函数
---

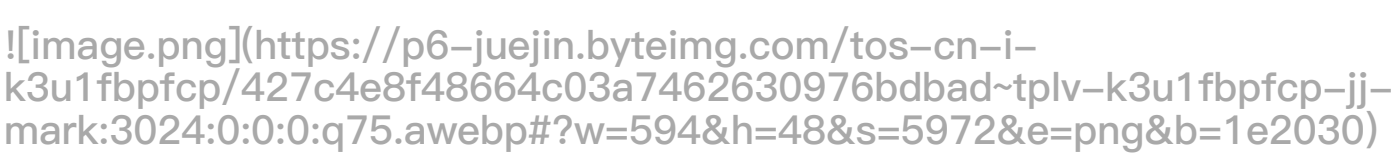
```
[ function ] funname[()]
{
    Action;
    [return int;]
}
funname
```

现在我们的任务是编写一个函数计算两个数之和，参考代码：

```
#! /bin/bash

# 自定义的函数
function sum() {
    let s=$1+$2
    # 只能返回0-255之间的值
    return $s
}

sum 10 20
echo $?

```

**return返回的话，是$?中的**，执行会输出30，但是如果我们写的是`sum 100 200`的话，猜一下结果会是多少?

没错100+200=44



这是为什么呢？因为return返回的值只能在0-255之间。那我们怎么返回比255大的数呢?

```
#! /bin/bash

# 自定义的函数
function sum() {
    let s=$1+$2
    # 只能返回0-255之间的值
    echo $s
}

res=`sum 100 200`
echo $res
```

```

如上所示，将计算结果作为输出，而不是返回值，通过res变量进行接收即可.

shell工具
=======

cut命令
-----

cut的工作就是“剪”，具体的说就是在文件中负责剪切数据用的。cut 命令从文件的每一行剪切字节、字符和字段并将这些字节、字符和字段输出

基本用法: cut [选项参数] filename 说明：默认分隔符是制表符

选项参数

* -f 列号，提取第几列
* -d 分隔符，按照指定分隔符分割列

首先创建文件并添加内容.

```
touch wenben.txt
vim wenben.txt

ping pong kk
pa po ff
xm hw pg
xiaomi huawei apple
zijie baidu tencent

```

执行:`cut -d " " -f 1 wenben.txt`


```

k3u1fbpfcp/6a6f0d991f8d47129812402e92644e52~tplv-k3u1fbpfcp-jj-mark:3024:0:0:0:q75.awebp#?w=837&h=161&s=11451&e=png&b=1e2030)

执行`cut -d " " -f1,3 wenben.txt`

k3u1fbpfcp/04a3b649a8634260bd3963552027c6d3~tplv-k3u1fbpfcp-jj-mark:3024:0:0:0:q75.awebp#?w=713&h=166&s=13455&e=png&b=1e2030)

sed
---

sed是一种流编辑器，它一次处理一行内容。处理时，把当前处理的行存储在临时缓冲区中，称为“模式 空间”，接着用sed命令处理缓冲区中的内容，处理完成后，把缓冲区的内容送往屏幕。接着处理下一 行，这样不断重复，直到文件末尾。文件内容并没有改变，除非你使用重定向存储输出。

使用语法

```
sed [选项参数] ‘command’ filename
```

常用参数

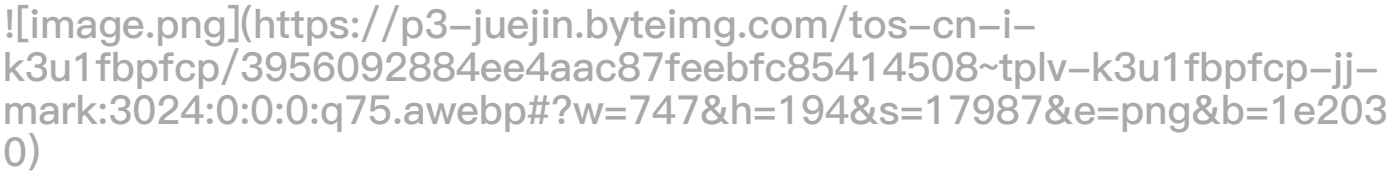
* a： 新增， a 的后面可以接字串，而这些字串会在新的一行出现(目前的下一行)
* c： 取代， c 的后面可以接字串
* d： 删除，因为是删除啊，所以 d 后面通常不接任何东西；

使用示例

1. 将“mei nv”这个单词插入到wenben.txt第二行下，打印。

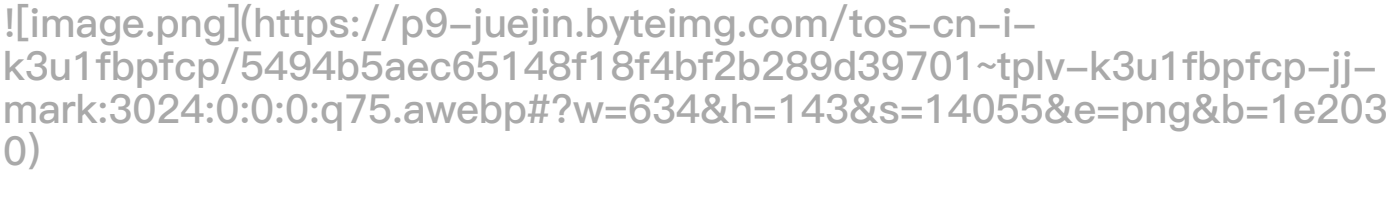
```
sed '2a mei nv' wenben.txt
```

2代表加入到第二行的结尾(也就是第三行的开头),a代表新增



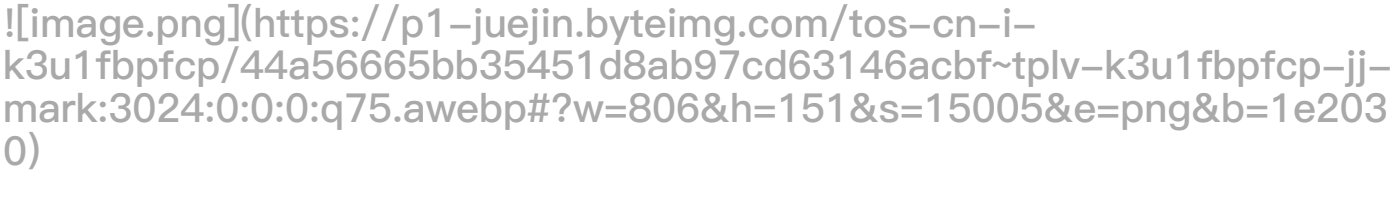
2. 删除wenben.txt文件所有包含xm的行

```
sed '/xm/d' wenben.txt
```



3. 将wenben.txt文件中的所有ping换成pang

```
sed 's/ping/pong/g' wenben.txt
```



awk
---

AWK的工作方式是逐行扫描输入文本文件，将每行分割成字段，并执行用户定义的操作。通常情况下，AWK的操作可以由用户通过命令行参数或者在一个单独的脚本文件中指定。

使用语法

```
awk options 'pattern {action}' file
```

* `options`：是一些选项，用于控制 `awk` 的行为。
* `pattern`：是用于匹配输入数据的模式。如果省略，则 `awk` 将对所有行进行操作。
* `{action}`：是在匹配到模式的行上执行的动作。如果省略，则默认动作是打印整行。

内建变量

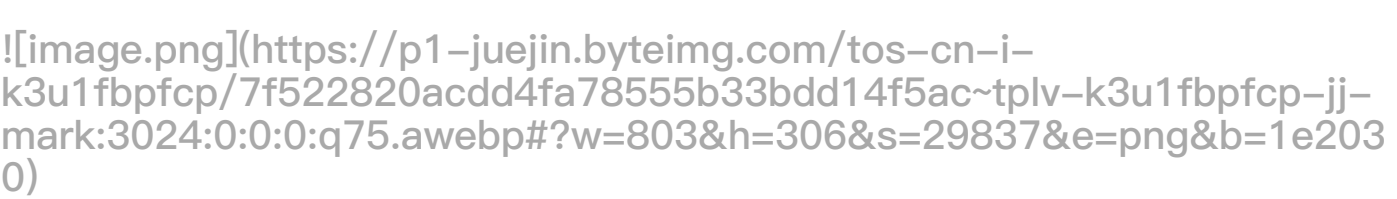
1. FS：列分割符。指定每行文本的字段分隔符，默认为空格或制表位。与"-F"作用相同
2. NF：当前处理的行的字段个数。
3. NR：当前处理的行的行号（序数）。
4. $0：当前处理的行的整行内容。
5. $n：当前处理行的第n个字段（第n列）。
6. FILENAME：被处理的文件名。

以下是AWK的命令示例

```
awk '{print}' wenben.txt#输出所有内容
awk '{print $0}' wenben.txt#输出所有内容
```

```
awk 'NR==1,NR==3 {print}' wenben.txt#输出第 1~3 行内容
awk 'NR==1;NR==3 {print}' wenben.txt#输出第 1和第3 行内容
awk '(NR>=1)&&(NR<=3) {print}' wenben.txt#输出第 1~3 行内容
```

```

```
awk '(NR%2)==1{print}' wenben.txt #输出所有奇数行的内容
awk '(NR%2)==0{print}' wenben.txt#输出所有偶数行的内容
```

```



原文链接: https://juejin.cn/post/7359541702049251343