

# 学习 Shell Script

教师: 李军辉

# 第一个Shell 程序

```
#!/bin/bash
# Program:
#           This program shows "Hello World!"
#           in your screen.
# 2017-11-04 by Li Junhui

echo -e "Hello World! \a \n"
exit 0
```

```
#!/bin/bash
# Program:
#           This program shows "Hello World!"
#           in your screen.
# 2017-11-04 by Li Junhui

echo -e "Hello World! \a \n"
exit 0
```

- 第一行 `#!/bin/bash` 在宣告这个 script 使用的 shell 名称

```
#!/bin/bash
# Program:
#           This program shows "Hello World!"
#           in your screen.
# 2017-11-04 by Li Junhui

echo -e "Hello World! \a \n"
exit 0
```

- 整个 script 当中，除了第一行的『#!』是用来宣告 shell 的之外，其他的 # 都是『注解』用途

```
#!/bin/bash
# Program:
#           This program shows "Hello World!"
#           in your screen.
# 2017-11-04 by Li Junhui

echo -e "Hello World! \a \n"
exit 0
```

- 一般来说，建议你一定要养成说明该 script 的：1. 内容与功能；2. 版本信息；3. 作者与联络方式；4. 建档日期；5. 历史纪录等等。

```
#!/bin/bash
# Program:
#           This program shows "Hello World!"
#           in your screen.
# 2017-11-04 by Li Junhui

echo -e "Hello World! \a \n"
exit 0
```

- 利用 exit 这个命令来让程序中断，并且回传一个数值给系统。

```
#!/bin/bash
# Program:
#           User inputs his first name and last name
#           Program shows his full name.
# 2017-11-04 by Li Junhui

read -p "Please input your first name: " firstname
read -p "Please input your last name:
" lastname
echo -e "\nYour full name is: $firstname $lastname"
```

# 利用 date 进行文件的创建

```
# 1. 让使用者输入文件名称，并取得 fileuser 这个变量；  
echo -e "I will use 'touch' command to create 3 files."  
read -p "Please input your filename: " fileuser # 提示使用者输入 #  
  
2. 为了避免使用者随意按 Enter，利用变量功能分析档名是否有  
配置？  
filename=${fileuser:-"filename"} # 开始判断有否配置档名
```

```
# 3. 开始利用 date 命令来取得所需要的档名了；  
date1=$(date -date='2 days ago' +%Y%m%d) # 前两天的日期  
date2=$(date -date='1 days ago' +%Y%m%d) # 前一天的日期  
date3=$(date +%Y%m%d) # 今天的日期  
file1=${filename}${date1} # 底下三行在配置档名  
file2=${filename}${date2}  
file3=${filename}${date3}
```

# 4. 将档名创建吧！

touch "\$file1" # 底下三行在创建文件

touch "\$file2"

touch "\$file3"

# 简单的加減乘除

```
echo -e "You SHOULD input 2 numbers, I will cross them! \n"  
read -p "first number: " firstnu  
read -p "second number: " secnu  
total=$(($firstnu*$secnu))  
echo -e "\n The result of $firstnu x $secnu is ==> $total"
```

script 的运行方式差异 (source, sh script, ./script)

## 数值运算及处理

# 数值运算及处理

- 整数运算操作
- 几个数值处理技巧
- 小数运算操作

# 整数运算操作

使用 expr 命令, 计算表达式

- 格式: expr 数值1 操作符 数值2

使用 \$[ ] 命令, 算式替换

- 格式: \$[数值1 操作符 数值2]

操作符: + - \* / %

# 整数运算操作 (示例1)

```
expr 10 + 15
```

```
expr 10 - 15
```

```
expr 10 * 2
```

```
X=10; Y=15; echo $X + $Y
```

## 整数运算操作 (示例2)

```
echo $[10 + 15]
```

```
echo $[10 - 15]
```

```
echo $[10 * 2]
```

```
X=10; Y=15; echo ${X + Y}
```

# 几个数值处理技巧

## 变量的递更处理

- 格式: `let 变量名++`、`let 变量名-`

## 使用随机数

- `RANDOM` 变量

## 生成数字序列

- 格式: `seq 首数 末数`、`seq 首数 增量 末数`

## 变量的递更处理 (示例1)

X=10; Y=5

let X++; echo \$X

let Y-=; echo \$Y

let X+=2; echo \$X

# 几个数值处理技巧

## 使用随机数

- RANDOM 变量

```
echo $RANDOM
```

```
echo ${$RANDOM%100}
```

# 生成数字序列

- 格式: seq 首数 末数、 seq 首数 增量 末数

seq 3

seq 3 5

seq 3 3 20

# 小数运算操作

将表达式给bc 命令处理

```
echo "45.3 - 12.2" | bc
```

```
echo "scale=4; 10/3" | bc
```

# 数值运算及处理

- 整数运算操作
- 几个数值处理技巧
- 小数运算操作

## 字符串处理

# 字符串处理

- 字符串长度
- 字符索引
- 子串截取操作
- 字符串替换
- 字符串删除

## 字符串长度

```
string="hello,everyone my name is xiaoming"  
echo ${#string}  
expr length "$string"
```

# 字符索引

expr index \$string substring索引命令功能在字符串\$string上找出substring中字符第一次出现的位置，若找不到则expr index返回0。

```
string="hello, everyone my name is xiaoming"
```

```
expr index "$string" my
```

```
expr index "$string" nihao
```

# 子串截取操作

## 路径分割

- `dirname` 命令、`basename`命令

# 子串截取操作

## 路径分割

- `dirname` 命令、`basename`命令

## 使用 `expr` 命令

- 格式: `expr substr $Var1 起始位置 截取长度`

# 子串截取操作

## 路径分割

- `dirname` 命令、`basename`命令

## 使用`expr`命令

- 格式: `expr substr $Var1 起始位置 截取长度`

## 使用\${}表达式

- 格式:  `${Var1:起始位置:截取长度}`

## 路径分割 (示例1)

```
Var1="/etc/httpd/conf/httpd.conf"
```

```
dirname $Var1
```

```
basename $Var1
```

# 字符串截取 (示例1)

使用expr命令

- 格式: expr substr \$Var1 起始位置 截取长度

Var1=“Today is Tuesday!”

expr substr \$Var1 4 6 注意: 起始位置是从1开始

## 字符串截取 (示例2)

使用\${}表达式

- 格式: \${Var1:起始位置:截取长度}

Var1=“Today is Tuesday!”

```
echo ${Var1:4:6}
```

注意: 起始位置是从0开始

## 字符串截取 (示例2)

使用\${}表达式

- 格式: \${Var1:起始位置:截取长度}

Var1=“Today is Tuesday!”

```
echo ${Var1:4:6}
```

注意: 起始位置是从0开始

```
echo ${Var1::6}
```

# 字符串替换

使用\${}表达式:

- 格式1: \${Var1/old/new}
- 格式1: \${Var1//old/new}

Var1=“Today is Tuesday!”

```
echo ${Var1/T/MM}
```

```
echo ${Var1//T/MM}
```

## 字符串删除

删除字符串和抽取字符串相似\${string#substring}为删除string开头处与substring匹配的最短字符串，而\${string##substring}为删除string开头处与substring匹配的最长字符串。

```
string="20091111 readnow please"
echo ${string#2*1}
string="20091111 readnow please"
echo ${string##2*1}
```

## 条件测试

# 条件测试

- 测试操作规范
- 文件状态的检测
- 整数值比较、字符串匹配

# 测试操作规范

测试的本质：

- 是一条操作命令
- 根据 \$? 返回值判断条件是否成立

# 测试操作规范

测试的本质：

- 是一条操作命令
- 根据 \$? 返回值判断条件是否成立

操作规范：

- 格式1: test 条件表达式
- 格式2: [条件表达式]

# 测试操作规范

## 测试操作的练习方法

- 直接跟 `&& echo YES` 判断结果
- 用法: [条件表达式] `&& echo YES`

# 测试操作规范

## 测试操作的练习方法

- 直接跟 && echo YES 判断结果
- 用法: [条件表达式] && echo YES

示例: ls -dl /home/abc /home/lijunhui

[ -d “/home/abc” ] && echo YES

[ -d “/home/lijunhui” ] && echo YES

# 文件状态的检测

存在及识别:

- -e: 目标是否存在 (**E**xist)
- -d: 是否为目录 (**D**irectory)
- -f: 是否为文件 (**F**ile )

# 文件状态的检测

存在及识别:

- -e: 目标是否存在 (Exist)
- -d: 是否为目录 (Directory)
- -f: 是否为文件 (File )

示例:

```
[ -d /home/lijunhui] && echo YES
```

```
[ -f /home/lijunhui] && echo YES
```

# 文件状态的检测

权限的检测：

- -r: 是否有读取(Read)权限
- -w: 是否有写入(Write)权限
- -x: 是否有可执行(execute)权限

# 文件状态的检测

权限的检测:

- -r: 是否有读取(Read)权限
- -w: 是否有写入(Write)权限
- -x: 是否有可执行(eXcute)权限

示例:

```
[ -r /home/lijunhui ] && echo YES
```

```
[ -w /home/lijunhui ] && echo YES
```

# 整数值比较、字符串匹配

整数值比较:

- -eq: 等于 (Equal)
- -ne: 不等于 (Not Equal)
- -gt: 大于 (Greater Than)
- -lt: 小于 (Less Than)
- -ge: 大于或等于 (Greater or Equal)
- -le: 小于或等于 (Less or Equal)

## 整数值比较(示例)

如果文件a.txt多于5行, 打印YES

```
[ $(cat a.txt | wc -l) -gt 5 ] && echo YES
```

# 整数值比较、字符串匹配

## 字符串匹配

- `=`: 两个字符串相同
- `!=`: 两个字符串不相同

# 整数值比较、字符串匹配

## 字符串匹配

- `=`: 两个字符串相同
- `!=`: 两个字符串不相同

示例:

```
echo $USER
```

```
[ $USER = "root" ] && echo YES
```

```
[ $USER != "root" ] && echo YES
```

# 条件测试

- 测试操作规范
- 文件状态的检测
- 整数值比较、字符串匹配

使用if判断结构

# 使用if判断结构

- 单分支、双分支的if应用
- 多重分支的if应用

# 单分支、双分支的if应用

单分支if语句结构

if 条件测试

then 命令序列

fi

## 单分支 (示例1)

检查目录 /expriment 是否存在, 如果不存在, 创建此目录

mkdir.sh

# 单分支、双分支的if应用

双分支if语句结构

**if** 条件测试

**then** 命令序列1

**else** 命令序列2

**fi**

## 双分支 (示例1)

判断目标主机是否可访问，显示检测结果

chkhost.sh

# 多重分支的if应用

多分支if语句结构

**if** 条件测试1;

**then** 命令序列1

**elif** 条件测试2;

**then** 命令序列2

.....

**else** 命令序列n

**fi**

# 多重分支 (示例1)

判断成绩, 区分优秀/合格/不合格

chkgrade.sh

使用for 循环

# 使用for 循环

- for 语句的语法结构
- 基本用法示范

# for 语句的语法结构

for 变量名 in {取值列表}

do

命令序列

done

# for 语句的语法结构 (1)

```
#!/bin/bash

# 2016-11-22 by Li Junhui

for i in $(seq 10)
do
    echo $i
done
```

## for 语句的语法结构 (2)

```
#!/bin/bash

# 2016-11-22 by Li Junhui

for i in "1st" "2nd" "3rd"
do
    echo $i
done
```

## for 语句的语法结构 (3)

```
#!/bin/bash

# 2016-11-22 by Li Junhui

for (i = 0; i < 10; i++)
do
    echo $i
done
```

## for 语句的语法结构 (4)

```
#!/bin/bash

# 2016-11-22 by Li Junhui

for f in `ls ./`  
do  
    echo $f  
done
```

## for 语句的语法结构 (5)

```
#!/bin/bash

for i in $(cat $1)
do
    echo $i
done
```

## for 应用举例 (1): 统计单词/词形个数

统计某一文件 (nist02.en) 中单词总数, 以及词形总数.

## for 应用举例 (1): 统计单词/词形个数

统计某一文件 (nist02.en) 中单词总数, 以及词形总数.

单词总数: `cat nist02.en | wc -w`

`test.sh nist02.en | wc -l`

## for 应用举例 (1): 统计单词/词形个数

统计某一文件 (nist02.en) 中单词总数, 以及词形总数.

单词总数: `cat nist02.en | wc -w`

`test.sh nist02.en | wc -l`

词形总数: `test.sh nist02.en | sort | uniq | wc -l`

## for 应用举例 (2): 批量增加用户账号

- 给定一个用户文件user.txt, 每行一个用户名
- 将初始密码设置为123456, 首次登录后必须更改

## for 应用举例 (2): 批量增加用户账号

```
#!/bin/bash

for i in $(cat ./users.txt)
do
    useradd $i
    echo -e "123456\n123456" | passwd $i
    chage -d 0 $i
done
```

## for 应用举例 (3): 批量检查主机状态

检测一个IP范围内主机的状态

- 10.10.65.1 - - 10.10.65.10
- 根据是否ping 通来判断

## for 应用举例 (3): 批量检查主机状态

```
#!/bin/bash

IP_PRE="10.10.65."
for IP in $(seq 1 5)
do
    ping -c 3 -i 0.2 -W 3 ${IP_PRE}${IP} &> /dev/null
    if [ $? -eq 0 ]; then
        echo "${IP_PRE}${IP} is up"
    else
        echo "${IP_PRE}${IP} is down"
    fi
done
```

# case分支语句

- case 语句结构
- 基本用法

# case 语句结构

case 变量值 in

模式1)

命令序列1

;;

模式2)

命令序列2

;;

.....

\*)

默认命令序列

esac

## case 基本用法

```
#!/usr/bin/bash

echo "input: "
read num
echo "the input data is $num"

case $num in
1) echo "January";;
2) echo "Feburay";;
3) echo "Match";;;
*) echo "not correct input"
esac
```

## case 基本用法

```
#!/bin/bash
read -p "press some key ,then press return :" KEY
case $KEY in
[a-z]|[A-Z])
    echo "It 's a letter."
    ;;
[0-9])
    echo "It 's a digit."
    ;;
*)
    echo "It 's function keys , Spacebar or other keys"
esac
```

## while语句

**while** [ 条件测试 ]

**do**

命令序列

**done**

## while 基本用法

```
#!/bin/bash
# Program:
#           Repeat question until user input correct answer

while [ "$yn" != "yes" -a "$yn" != "YES" ]
do
    read -p "Please input yes/YES to stop this"
done
echo "OK! you input the correct answer."
```

用while实现计算  $1+2+3+\dots+100$

用while实现计算  $1+2+3+\dots+100$

```
#!/bin/bash
# Program:
#           Use loop to calculate "1+2+3+...+100" result

s=0
i=0
while [ "$i" != "100" ]
do
    i=$((i+1))
    s=$((s+i))
done
echo "The result of '1+2+3+...+100' is ==> $s"
```

# until语句

until [ 条件测试 ]

do

命令序列

done

# until 基本用法

```
#!/bin/bash
# Program:
#           Repeat question until user input correct answer

until [ "$yn" == "yes" -o "$yn" == "YES" ]
do
    read -p "Please input yes/YES to stop this"
done
echo "OK! you input the correct answer."
```

## shell脚本的中断和继续

- break
- continue
- exit

# break

```
#!/bin/bash

for i in `seq 1 5`
do
    echo $i
    if [ $i == 3 ]
    then
        break
    fi
    echo $i
done
echo aaaa
```

## continue

```
#!/bin/bash

for i in `seq 1 5`
do
    echo $i
    if [ $i == 3 ]
    then
        continue
    fi
    echo $i
done
echo $i
```

exit

```
#!/bin/bash

for i in `seq 1 5`
do
    echo $i
    if [ $i == 3 ]
    then
        exit
    fi
    echo $i
done
echo aaaa
```

# shell脚本中的函数

```
#!/bin/bash

function sum() {
    sum=$[ $1+$2 ]
    echo $sum
}

sum $1 $2
```

# shell script 的追踪与 debug

选项与参数:

- -n : 不要运行 script, 仅查询语法的问题;
- -v : 再运行 script 前, 先将 scripts 的内容输出到萤幕上;
- -x : 将使用到的 script 内容显示到萤幕上, 这是很有用的参数!

## 练习题

- 1 给定一下目录，求这个目录下普通文件的个数。
- 2 给定一个目录，删除该目录下大小为0的文件。
- 3 给定一个目录，将该目录下所有普通文件更改名字为1, 2, ...
- 4 设计一个Shell程序，在~/userdata目录下建立50个目录，即user1~user50，并设置每个目录的权限，其中其他用户的权限为：读；文件所有者的权限为：读、写、执行；文件所有者所在组的权限为：读、执行。
- 5 假设存在两个文本文件A和B，以行为单位，求文件A和文件B交集，并集，差集。

给定一下目录，求这个目录下普通文件的个数。

```
#!/bin/bash

path=$1
count=0

for file in $(ls $path)
do
    if [ -f $file ]
    then
        let count++
    fi
done

echo "count = $count"

exit 0
```

给定一个目录，删除该目录下大小为0的文件。

```
#/bin/bash

for filename in `ls $1`
do
    if [ -f $filename ]
    then
        a=$(ls -l $filename | awk '{ print $5 }')
        if test $a -eq 0
        then rm $filename
        fi
    fi
done
```

给定一个目录，将该目录下所有普通文件更改名字为1, 2, ...

```
#!/bin/bash

path=$1
index=1

for file in $(ls $path)
do
    if [ -f $file ] && [ $file != $index ]
    then
        $(mv $file $index)
        let index++
    fi
done

exit 0
```

假设当前目录下有文件:

- image0001.png
- image0002.png
- image0003.png
- ...

现在需要将它们重命名为:

- 0001.png
- 0002.png
- 0003.png
- ...

假设当前目录下有文件:

- image0001.png
- image0002.png
- image0003.png
- ...

现在需要将它们重命名为:

- 0001.png
- 0002.png
- 0003.png
- ...

```
for f in *.png; do mv $f ${f#image}; done
```

设计一个Shell程序，在~/userdata目录下建立50个目录，即user1~user50，并设置每个目录的权限，其中其他用户的权限为：读；文件所有者的权限为：读、写、执行；文件所有者所在组的权限为：读、执行。

```
#!/bin/bash
for (( i = 1; i <= 50; i ++))
do
    mkdir -p ~/userdata/user$i
    cd ~/userdata
    chmod 754 user$i
done
```

假设存在两个文本文件A和B，以行为单位，求文件A和文件B交集， 并集， 差集。

- 交集: `cat a.txt b.txt | sort | uniq -d`

假设存在两个文本文件A和B，以行为单位，求文件A和文件B交集， 并集， 差集。

- 交集: `cat a.txt b.txt | sort | uniq -d`
- 并集: `sort a.txt | uniq > aa.txt; sort b.txt | uniq > bb.txt; cat aa.txt bb.txt | sort | uniq -d`

假设存在两个文本文件A和B，以行为单位，求文件A和文件B交集， 并集， 差集。

- 交集: cat a.txt b.txt | sort | uniq -d
- 交集: sort a.txt | uniq > aa.txt; sort b.txt | uniq > bb.txt; cat aa.txt bb.txt | sort | uniq -d
- 并集: cat a.txt b.txt | sort | uniq

假设存在两个文本文件A和B，以行为单位，求文件A和文件B交集， 并集， 差集。

- 交集: cat a.txt b.txt | sort | uniq -d
- 交集: sort a.txt | uniq > aa.txt; sort b.txt | uniq > bb.txt; cat aa.txt bb.txt | sort | uniq -d
- 并集: cat a.txt b.txt | sort | uniq
- 差集: cat a.txt b.txt b.txt | sort | uniq -u

假设存在两个文本文件A和B，以行为单位，求文件A和文件B交集， 并集， 差集。

- 交集: cat a.txt b.txt | sort | uniq -d
- 交集: sort a.txt | uniq > aa.txt; sort b.txt | uniq > bb.txt; cat aa.txt bb.txt | sort | uniq -d
- 并集: cat a.txt b.txt | sort | uniq
- 差集: cat a.txt b.txt b.txt | sort | uniq -u
- 差集: sort a.txt | uniq > aa.txt; sort b.txt | uniq > bb.txt; cat aa.txt bb.txt bb.txt | sort | uniq -u

## 练习题

- 1 编写shell脚本，计算 $1 + 2 + \dots + 100$ 之和。
- 2 编写shell脚本，输入一个数字n并计算 $1 + 2 + \dots + n$ 的和。要求：如果输入的数字小于1，则重新输入，直到输入正确的数字为止。
- 3 编写shell脚本，把/root/目录下的的呢目录（只需要一级）复制到/tmp/目录下。
- 4 编写shell脚本，批量建立用户user\_00、user\_01…user\_99。要求：所有用户同属于users组。
- 5 编写shell脚本，截取文件test.log中包含关键词abc的行中的第一列（假设分隔符为：），然后把截取的数字排序（假设第1列为数字），最后打印出重复超过10次的列。
- 6 编写shell脚本，判断输入的IP是否正确。要求：IP的规则是n1.n2.n3.n4，其中 $1 < n1 < 255$ ,  $0 < n2 < 255$ ,  $0 < n3 < 255$ ,  $0 < n4 < 255$ ）。