简介：

Shell 是一个用C语言编写的程序，它是用户使用Linux的桥梁。Shell既是一种命令语言，又是一种程序设计语言。

Shell 是指一种应用程序，这个应用程序提供了一个界面，用户通过这个界面访问操作系统内核的服务。

Ken Thompson的sh是第一种Unix Shell，Windows Explorer是一个典型的图形界面Shell。

Shell自己总结

[]代表Java中的小括号，单一的中括号内的逻辑判断if[ a\>b ]-o[ c\>b ],-o代表或者，-a代表与即and，双括号的写法[[ a>b || c>b ]]

$\*代表输入的命令，$0 第一条命令，$1代表第二条命令

while[[ $a –lt 5 ]]/while(($a>5))

Shell教程入门

1、shell脚本

shell脚本shell script，是一种为shell而编写的脚本程序。然而通常所说的shell却是指shell脚本，而非shell本身。

Linux系统shell种类众多，常用的有sh和bash。

2、shell脚本实践

shell脚本用#!/bin/bash或#!/bin/sh之类的方式，制定运行脚本的shell,#!是识别符号。

1. #!/bin/sh
2. echo "Hello World"

运行shell脚本的两种方法：

* 作为可执行程序

将shell文本保存为.sh格式文件，赋予执行权限

* 1. chmod + x ./test.sh *# 赋予可执行权限*
  2. ./test.sh *#执行脚本，注意此处需要指明当前目录下的test.sh哦，不然会去path路径找的，呵呵。*
* 作为解释其参数

直接解释运行,如此就不用写#!/bin/sh之类的引用注释了。

* 1. /bin/sh test.sh
  3. *# 类似java、php、python脚本*
  5. /bin/php test.php

Shell变量

变量命名语法，不需要$符号，php需要。命名要求：

* 首字母必须字母a-z或A-Z
* 不能空格，可用\_
* 不能标点
* 不能使用bash关键字

1. *#注意，不同于其他编程语言，等号之间不要有空格。*
2. your\_variable="your,name"
3. *# 使用语句给变量赋值，如下循环显示出/etc目录下的文件名*
4. for file in 'ls /etc'

1、使用变量

使用已定义变量，只需在变量前加$符号即可：

1. your\_variable='yourname'
2. echo $your\_variable
3. echo $(your\_variable)

变量名可以加{}来标识变量名的范围，如：

1. for skill in Ada Coffe Action Java; do
2. echo "I am good at ${skill}Script"
3. done

如果不加{}，skill就可能被认为skillScript而导致变量找不到

已定义的变量可重新定义：

1. your\_name="tom"
2. echo $your\_name
3. your\_name="john"
4. echo $your\_name

2、只读变量

使用readonly命令将变量只读，则不可再改变，否则报错。

1. #!/bin/bash
2. myUrl="http://www.w3cschool.cc"
3. readonly myUrl
4. myUrl="new url" *#此处就会报错，因为变量只读了。*

3、删除变量

使用unset命令删除变量：

unset variable\_name

==删除变量后不能再用，而unset不能删除只读变量==

1. #!/bin/sh
2. myUrl="http://www.google.com"
3. unset myUrl
4. echo $myUrl *#此时输出就没接过了，因为变量被删除了。*

4、变量类型

运行shell时，会同时存在三种变量：

* 局部变量：脚本中定义的变量，仅作用于本shell脚本内。
* 环境变量：所有程序，包括shell启动程序都能访问的环境变量。
* shell变量：shell程序的特殊变量。

5、shell字符串

shell常用number和string，其中string可以单引号、双引号或者不用引号。但是略有区别：

* 单引号

str='string test'

单引号''之间的字符原样输出，里面的变量也会失效。其内部不能再有单引号，哪怕转义符号都失效。

* 双引号
  + your\_name='your name'
  + str=*"Hello ,world ,\"*$your\_name\*"! \n"*

双引号里面可以有变量，可以有转义符号。

* 字符串拼接
  + n1="abc"
  + test="hello, "$n1" !"
  + test1="hello, ${n1} !"
  + echo $test $test1
* 获取字符串长度

使用#标识变量长度

* + str="abcdef"
  + *#输出字符串长度*
  + echo ${#str}
* 获取子字符串
  + str="abcdef"
  + *#下标从左至右，0开始，*
  + echo ${str:1:4}
* 查找子字符串

使用`符号

* + str="hello world nihaome"
  + *#查找字符i或s的位置,反引号*
  + echo `expr index "$str" is`

6、Shell数组

bash仅支持一维数组，可利用下标或表达式操作元素。

* 定义数组

shell中使用()表示数组，元素用空格来分割。

* 1. array=(1 2 3 4 5 6 7)
  3. *#或者*
  5. array=(
  6. a
  7. b
  8. c
  9. )
  11. *#也可以单独定义,下标可以不连续，也无范围限制。*
  13. array[0]=1
  14. array[1]=2
  15. array[3]=7
* 读取数组
  1. variable=${array[index]}
  3. *# @符号代替index表示获取所有元素*
  5. echo ${variable[@]}
* 获取数组长度

类似字符串的获取

* 2. *#获取元素个数*
  4. length=${*#array\_name[@]}*
  6. *#或者\*通配符*
  8. length=${*#array\_name[\*]}*
  10. *#获取数组单个元素的长度*
  12. length\_n=${*#array\_name[n]}*

7、Shell注释

使用#至于行首，表示该行注释，shell无多行注释，只能每行都#

1. *#-----------------------*
2. *#我是个多行注释*
3. *#但是只能这么写*
4. *#-----------------------*

要是多行代码需要注释，可以将定义为函数，加{}包裹起来，该函数不被调用，则类似于注释掉。

Shell传递参数

在执行shell脚本时，可以向脚本传递参数，脚本内获取参数的格式：$n，n代表数字编号，为脚本内需要获取的参数的编号。

1、实例

示例向脚本传递三个参数，并输出，$0为执行文件名：

1. #!/bin/bash
2. *#传参测试*
4. echo "Shell 传参测试";
5. echo "file name: $0";
6. echo "first variable: $1";
7. echo "second variable: $2"
8. echo "third variable: $3";

通过赋予权限，或者指定执行，可实现输出：

1. $ chmod +x test.sh
2. *# $符号在次表示root用户执行，下面是传入1，2，3，三个数到脚本*
3. $ ./test.sh 1 2 3
5. *#输出结果：*
6. Shell 传参测试
7. file name: ./test.sh
8. first variable: 1
9. second variable 2
10. third variable 3

另有几个特殊字符处理参数：

| **参数处理** | **说明** |
| --- | --- |
| $# | 传递到脚本的参数的个数 |
| $\* | 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 |
| $@ | 类似$\*，使用时许加引号，并在引号中返回每个参数。 |
| $- | 显示shell使用的当前选选项，类似set命令 |
| $? | 显示最后命令的退出状态。0 表示无错误。其他都是错。 |

1. #!/bin/sh
2. echo "Shell 传递参数实例！";
3. echo "第一个参数为：$1";
5. echo "参数个数为：$#";
6. echo "传递的参数作为一个字符串显示：$\*";

执行效果：

1. $ chmod +x test.sh
2. $ ./test.sh 1 2 3
3. Shell 传递参数实例！
4. 第一个参数为：1
5. 参数个数为：3
6. 传递的参数作为一个字符串显示：1 2 3

$\*与$@的异同：

* 都是应用所有参数
* 不同：只有在双引号中体现。假设在脚本运行时写了三个参数 1、2、3，，则 ” \* ” 等价于 “1 2 3”（传递了一个参数），而 “@” 等价于 “1” “2” “3”（传递了三个参数）。

  2. #!/bin/bash
  4. echo "-- \$\* demo ---"
  5. for i in "$\*";do
  6. echo $i
  7. done
  9. echo "-- \$@ demo ---"
  10. for i in "$@"; do
  11. echo $i
  12. done

执行效果：

* 1. $ chmod +x test.sh
  2. $ ./test.sh 1 2 3
  3. -- $\* 演示 ---
  4. 1 2 3
  5. -- $@ 演示 ---
  6. 1
  7. 2
  8. 3

Shell数组

Bash shell仅支持一维数组，不限定大小，初始化时候不需要指定大小。下标0开始，()包裹，空格分割元素。

array=(a b c d)

* 读取数组

格式${array[index]}

1. #!/bin/bash
2. my\_array=(a b "c" d)
4. echo "first: ${my\_array[0]}"
6. *#然后执行文件，获得输出结果*
7. *#可用@或\*来代替index获取所有元素*

* 数组长度

类似字符串的长度获取

${*#array[index]}*

Shell运算符

shell支持多种运算符：

* 算数运算符
* 关系运算符
* 布尔运算符
* 字符串运算符
* 文件测试运算符

原生Bash不支持简单的数学运算，可以用awk和expr实现。

1. #!/bin/sh
2. val=`expr 2 + 2`
3. echo $val
5. *#注意，expr用反引号，表达式和运算符之间必须有空格，2+2就不行。*

1、算术运算符

若a = 10，b = 20

| **运算符** | **说明** | **举例** |
| --- | --- | --- |
| + | 加号 | `expr $a + $b`，result = 30 |
| - | 减号 | `expr $a - $b`，result = -10 |
| \* | 乘号 | `expr $a \* $b`，result = 200 |
| / | 除号 | `expr $b / $a`，result = 2 |
| % | 取余 | `expr $b % $a`，result=0 |
| = | 赋值 | a=$b，将b的值赋给a |
| == | 相等，比较数字，同则true。 | [$a == $b]返回false |
| != | 不等，比较数字，不同的true。 | [$a != $b]返回true。 |

1. #!/bin/bash
2. *# author:菜鸟教程*
3. *# url:www.runoob.com*
5. a=10
6. b=20
8. val=`expr $a + $b`
9. echo "a + b : $val"
11. val=`expr $a - $b`
12. echo "a - b : $val"
14. val=`expr $a \\* $b`
15. echo "a \* b : $val"
17. val=`expr $b / $a`
18. echo "b / a : $val"
20. val=`expr $b % $a`
21. echo "b % a : $val"
23. if [ $a == $b ]
24. then
25. echo "a 等于 b"
26. fi
27. if [ $a != $b ]
28. then
29. echo "a 不等于 b"
30. fi

执行结果：

1. a + b : 30
2. a - b : -10
3. a \* b : 200
4. b / a : 2
5. b % a : 0
6. a 不等于 b

注意：条件表达式必须在[]之间，且必须有空格，如[$a==$b]是错的！

\*乘号需要在expr表达式内用\转义

Mac电脑的shell的expr表达式：$((表达式))所以它的乘号\*不用转义

2、关系运算符

布尔类型，关系运算符仅支持数字，除非字符串的值也是数字。示例，若a = 10 ,b = 20:

| **运算符** | **说明** |
| --- | --- |
| -eq | equal |
| -ne | not equal |
| -gt | great than |
| -lt | less than |
| -ge | great equal |
| -le | less equal |

3、布尔运算符

| **运算符** | **说明** |
| --- | --- |
| ! | 非运算，表达式为true，则返回false。 |
| -o | 或运算，一个表达式为true，则返回true。 |
| -a | 与运算，两个都true，才返回true。 |

4、逻辑运算符

| **运算符** | **说明** |
| --- | --- |
| && | 逻辑and |
| || | 逻辑 or |

5、字符串运算符

| **运算符** | **说明** |
| --- | --- |
| = | 检测两字符串是否相等 |
| != | 检测两字符串是否不等 |
| -z | zero检测字符串长度是否为0，0则true |
| -n | not zero检测字符串长度是否非0。 |
| str | 检测字符串是否为空，不空则true。 |

6、文件测试运算符

用于检测类unix 文件的各种属性

| **操作符** | **说明** |
| --- | --- |
| -b file | 是否是块设备文件，是则true |
| -c file | 是否是字符设备文件，是则true |
| -d file | 是否是目录，是则true |
| -f file | 是否是普通文件(既非目录，亦非设备文件)，若是则true |
| -g file | 是否设置了SGID位，是则true |
| -k file | 是否设置粘着位，是则true |
| -p file | 是否有名管道，是则true |
| -u file | 是否设置SUID，是则true |
| -r file | 是否可读，是则true |
| -w file | 是否可写，是则true |
| -x file | 是否可执行，是则true |
| -s file | 是否为空或大小是否大于0，非空则true |
| -e file | 是否存在，是则true |

==注意shell脚本的表达式都要在[]内哦==

Shell echo命令

类似于php的echo，shell的echo用于输出字符串，格式echo string

* 显示普通字符串

1. echo "Hello World"
2. *#可以不带引号*
3. echo Hello World

* 显示转义字符
  1. echo "\"It is a test\""
  3. *#输出结果*
  5. "It is a test"
* 显示变量

read命令从标准输入中读取一行，并把输入行的每个字段指定给shell变量

* 2. #!/bin/sh
  3. read name
  4. echo "$name It is a test"

  7. *#运行效果：*
  9. [root@www ~]*# sh test.sh*
  10. OK *#标准输入*
  11. OK It is a test *#输出*
* 显示换行
  1. echo -e "Ok ! \n" *# -e 开启转义*
  2. echo "It is a test"
  4. *#输出结果：注意ok后面有换行*
  6. OK!
  8. It it a tes
* 显示不换行

  2. #!/bin/sh
  3. echo -e "OK! \c" *# -e 开启转义 \c 不换行*
  4. echo "It is a test"
  6. *#结果：*
  8. OK! It is a test
* 显示结果定向至文件

echo "It is a test" > test.txt

* 1. 1
* 原样输出字符，不转义不取变量，需要结合单引号
  1. echo '$name\"'
  3. *#输出结果*
  5. $name\"
* 显示命令执行结果

==命令用`来包裹==

* 1. echo `data`
  3. *#结果：*
  5. Thu Jul 24 10:08:46 CST 2014

Shell printf命令

类似C语言的printf()函数，shell使用printf会比echo更具有跨平台移植性。可以类似C的printf()函数使用一些复杂的表达式，printf不支持自动换行，需要借助\n

printf format-string [args...]

* 1

示例，模拟shell输出，

1. $ echo "hello ,shell"
2. hello ,shell
3. $ printf "hello ,shell \n"
4. hello ,shell
5. $

脚本化的printf命令使用：

1. #!/bin/bash
2. *# author:菜鸟教程*
3. *# url:www.runoob.com*
5. printf "%-10s %-8s %-4s\n" 姓名 性别 体重kg
6. printf "%-10s %-8s %-4.2f\n" 郭靖 男 66.1234
7. printf "%-10s %-8s %-4.2f\n" 杨过 男 48.6543
8. printf "%-10s %-8s %-4.2f\n" 郭芙 女 47.9876

执行脚本，显示结果：

1. 姓名 性别 体重kg
2. 郭靖 男 66.12
3. 杨过 男 48.65
4. 郭芙 女 47.99

%s,%c,%d,%f都是格式替换符，%-10s指一个宽度为10个字符（-表示左对齐，没有则右对齐），任何字符都会被显示在10个字符宽的字符内，如果不足则自动以空格填充，超过也会将内容全部显示出来。

%-4.2f 指格式化为小数，其中.2指保留2位小数。

1. *#!/bin/bash*
2. *# author:菜鸟教程*
3. *# url:www.runoob.com*
5. *# format-string为双引号*
6. printf "%d %s\n" 1 "abc"
8. *# 单引号与双引号效果一样*
9. printf '%d %s\n' 1 "abc"
11. *# 没有引号也可以输出*
12. printf %s abcdef
14. *# 格式只指定了一个参数，但多出的参数仍然会按照该格式输出，format-string 被重用*
15. printf %s abc def
17. printf "%s\n" abc def
19. printf "%s %s %s\n" a b c d e f g h i j
21. *# 如果没有 arguments，那么 %s 用NULL代替，%d 用 0 代替*
22. printf "%s and %d \n"

* 2

Printf的转移序列

| **序列** | **说明** |
| --- | --- |
| \a | 警告字符，通常为ASCII的BEL字符 |
| \b | 后退 |
| \c | 抑制不显示输出结果中任何姐wide换行字符（只在%b格式指示控制符下的参数字符串中有效），而且任何留在参数里的字符，任何接下来的采纳书以及任何留在格式字符串中的字符，都被忽略。 |
| \f | 换页 |
| \n | 换行 |
| \r | 回车 |
| \t | 水平tab |
| \v | 竖直tab |
| \\ | 转义后输出一个\符号 |
| \d dd | 表示1–3位的八进制字符，仅在格式字符串中有效。 |
| \0 dd d | 表示1–3位的八进制字符 |

Shell text命令

shell中test命令用于测试条件是否成立，可进行数字、字符和文件的条件测试

1. #!/bin/bash
2. *#用之前的各种运算符，测试test*
3. num1=100
4. num2=200
5. if test $[num1] -eq $[num2]
6. then
7. echo 'true'
8. else
9. echo 'false'
10. fi

Shell 流程控制

区别于其他编程语言，shell的流程控制不可为空，如

1. if(a>b){
2. System.out.println("ok");
3. }else{
4. *//此处不做任何事*
5. }

但是在shell中不能出现不做任何事的分支语句

1、if else

* if语句格式
  1. if condition
  2. then
  3. command1
  4. command2
  5. ...
  6. fi *# 是if的倒写*

每个句子可以用;分号结束，可以写在一行好几个语句。

* if else
  1. if condition
  2. then
  3. command1
  4. command2
  5. ...
  6. else
  7. command
  8. fi
* if else-if else
  1. if condition1
  2. then
  3. command1
  4. elif condition2 *#类似python中，else if是写作elif*
  5. then
  6. command2
  7. else
  8. command
  9. fi
  10. 9

2、for循环

shell的for循环格式：

1. for var in item1 item2 ... itemN
2. do
3. command1
4. command2
5. ...
7. done *#for循环结束的标识*
8. *#写成一行*
9. for var in item1 item2 ... itemN;do command1; command2;...;done;

* 9

3、while语句

格式：

1. while condition
2. do
3. command
4. done
5. *#示例*
7. #!/bin/sh
8. int=1
9. while(( $int<=5 ))
10. do
11. echo $int
12. let "int++" *# let是个关键命令*
13. done

* 3

可以结合read命令接收输入信息

4、无限循环

1. while :
2. do
3. command
4. done
6. *#或者*
7. while true
8. do
9. command
10. done
12. *#或者*
13. for (( ; ; ))

5、until循环

shell所有的until循环类似于一个特殊的for循环，知道满足条件时候才停止。一般还是while

1. until condition
2. do
3. command
4. done

条件可为任意测试条件，测试发生在循环末尾，因此循环至少执行一次—请注意这一点。

6、case

类似其他语言的switch…case语句

1. case value in
2. mode1)
3. command1
4. ...
5. ;; *#case的结束标志*
6. mode2)
7. command2
8. ...
9. ;;
10. esac *#case的反写*

value只会匹配一个case，或者不匹配，则mode可用\*号通配

1. echo '输入 1 到 4 之间的数字:'
2. echo '你输入的数字为:'
3. read aNum
4. case $aNum in
5. 1) echo '你选择了 1'
6. ;;
7. 2) echo '你选择了 2'
8. ;;
9. 3) echo '你选择了 3'
10. ;;
11. 4) echo '你选择了 4'
12. ;;
13. \*) echo '你没有输入 1 到 4 之间的数字'
14. ;;
15. esac

7、跳出循环

shell也使用break和continue来跳出循环。

* break

跳出所有循环，终止后面的执行。

* 2. #!/bin/bash
  3. while :
  4. do
  5. echo -n "输入 1 到 5 之间的数字:"
  6. read aNum
  7. case $aNum in
  8. 1|2|3|4|5) echo "你输入的数字为 $aNum!"
  9. ;;
  10. \*) echo "你输入的数字不是 1 到 5 之间的! 游戏结束"
  11. break
  12. ;;
  13. esac
  14. done
* continue

跳出本次循环，执行下一轮循环

* 2. #!/bin/bash
  3. while :
  4. do
  5. echo -n "输入 1 到 5 之间的数字: "
  6. read aNum
  7. case $aNum in
  8. 1|2|3|4|5) echo "你输入的数字为 $aNum!"
  9. ;;
  10. \*) echo "你输入的数字不是 1 到 5 之间的!"
  11. continue
  12. echo "游戏结束"
  13. ;;
  14. esac
  15. done

8、esac

case语句区别于C语言，用esac作为结束标志，)和;;标识每个case。

Shell函数

shell可以自定义函数，然后自己调用。函数格式：

1. *#function 关键字为可选项，参数也是可选*
2. [ function ] funname [()]
4. {
6. action;
8. [return int;]
10. }

说明：

1. 可以带function fun()定义，也可以fun()定义，不带参数。
2. 参数返回，可以显示加: return 返回，若不加，则返回最后一条指令结果。
3. #!/bin/bash
4. *# author:菜鸟教程*
5. *# url:www.runoob.com*
7. demoFun(){
8. echo "这是我的第一个 shell 函数!"
9. }
10. echo "-----函数开始执行-----"
11. demoFun
12. echo "-----函数执行完毕-----"

* 0

含有返回值的函数：

1. #!/bin/bash
2. *# author:菜鸟教程*
3. *# url:www.runoob.com*
5. funWithReturn(){
6. echo "这个函数会对输入的两个数字进行相加运算..."
7. echo "输入第一个数字: "
8. read aNum
9. echo "输入第二个数字: "
10. read anotherNum
11. echo "两个数字分别为 $aNum 和 $anotherNum !"
12. return $(($aNum+$anotherNum))
13. }
14. funWithReturn
15. echo "输入的两个数字之和为 $? !"

* 5

==调用函数返回值，用$?符号==，函数必须在被调用前定义。

* 函数参数

在Shell中，调用函数时可以向其传递参数。在函数体内部，通过 1表示第一个参数，$2表示第二个参数…

* 2. #!/bin/bash
  4. *# author:菜鸟教程*

  7. *# url:www.runoob.com*

  10. funWithParam(){
  11. echo "第一个参数为 $1 !"
  12. echo "第二个参数为 $2 !"
  13. echo "第十个参数为 $10 !"
  14. echo "第十个参数为 ${10} !"
  15. echo "第十一个参数为 ${11} !"
  16. echo "参数总数有 $# 个!"
  17. echo "作为一个字符串输出所有参数 $\* !"
  18. }
  19. funWithParam 1 2 3 4 5 6 7 8 9 34 73
  21. *#输出结果：*
  23. 第一个参数为 1 !
  24. 第二个参数为 2 !
  25. 第十个参数为 10 !
  26. 第十个参数为 34 !
  27. 第十一个参数为 73 !
  28. 参数总数有 11 个!
  29. 作为一个字符串输出所有参数 1 2 3 4 5 6 7 8 9 34 73 !

注意：$10不能获取第10个参数，因为当n>=10时候，要用$(n)来获取参数。

Shell 输入/输出重定向

类Uinx系统中，默认标准输入输出设备就是shell终端。重定向命令列表：

| **命令** | **说明** |
| --- | --- |
| command > file | 输出重定向到file |
| command < file | 输出重定向到file |
| command >> file | 输出追加到file |
| n > file | 文件描述符为n的文件重定向到file |
| n >> file | 文件描述符为n的文件追加到file |
| n >& m | 输出文件m和n合并 |
| n <& m | 输入文件m和n合并 |
| << tag | 开始标记tag和结束标记tag之间的内容作为输入 |

注意：需要注意的是文件描述符 0 通常是标准输入（STDIN），1 是标准输出（STDOUT），2 是标准错误输出（STDERR）。

1、输出重定向

command1 > file1

* 1

若file1存在，则被替代。可以用>>追加符号，则不替代。

2、输入重定向

command1 < file1

* 1

command1 < infile > outfile

* 1

同时替换输入和输出，执行command1，从文件infile读取内容，然后将输出写入到outfile中。

3、重定向深入讲解

一般类unix系统命令运行会同时打开三个文件：

* 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

默认情况下，command > file 将 stdout 重定向到 file，command < file 将stdin 重定向到 file。

如果希望 stderr 重定向到 file，可以这样写：

1. *#模拟终端*
2. $ command 2 > file

若要追加>>符号，其中2代表标准错误文件的代号。

如果希望将stdout和stderr合并后重定向到file

1. command > file 2>&1
2. *#或者*
3. command >> file 2>&1
4. *#如果输入、输出都重定向*
5. command <file1 >file2

4、Here Document

Here Document是shell的一种特殊重定向方式，用于将输入重定向到一个交互式shell脚本或程序：

1. command << delimiter
2. document
3. delimiter

作用将delimiter之间的document作为输入传递给command

注意：结尾的delimiter前后不得有任何符号，包括tab和空格。

1. wc -l << EOF
2. nihao
3. zhendehenhao
4. shime
5. ok
6. EOF
7. *#输出结果，获得的EOF之间的内容的行数。*
8. 4

5、/dev/null文件

若希望执行命令不在屏幕输出，可重定向到/dev/null

command > /dev/null

* 1

/dev/null为特殊文件，写入的内容立即不见，不可读出。

如果屏蔽stdout和stderr

command > /dev/null 2>&1

* 1

Shell文件包含

shell也可以使用外部脚本，便于封装：

1. . filename *#注意点号(.)与文件名之间有空格*
2. 或
3. source filename

示例：test1.sh

1. #!/bin/bash
2. *# author:菜鸟教程*
3. *# url:www.runoob.com*
5. url="http://www.runoob.com"

test2.sh

1. #!/bin/bash
2. *# author:菜鸟教程*
3. *# url:www.runoob.com*
5. *#使用 . 号来引用test1.sh 文件*
6. . ./test1.sh
8. *# 或者使用以下包含文件代码*
9. *# source ./test1.sh*
11. echo "菜鸟教程官网地址：$url"

执行时候test2.sh只需要test2.sh有执行权限即可，test1.sh不一定需要。