# Shell学习笔记

目录

[一、shell变量 1](#_Toc479951436)

[二、shell字符串 1](#_Toc479951437)

[三、shell数组 2](#_Toc479951438)

[四、shell注释 2](#_Toc479951439)

[五、shell传递参数 2](#_Toc479951440)

[六、shell运算符 3](#_Toc479951441)

[五、shell echo： 4](#_Toc479951442)

[六、shell之printf命令 5](#_Toc479951443)

[七、shell之test命令 5](#_Toc479951444)

[八、shell之流程控制 6](#_Toc479951445)

[九、shell之函数 9](#_Toc479951446)

[十、shell之输入输出重定向 9](#_Toc479951447)

[十一、shell之文件包含 11](#_Toc479951448)

## 一、shell变量

命名规则：

首个字母必须为字母;

中间不能有空格，可以使用下划线(\_);

不能使用标点符号；

不能使用bash里的关键字

赋值方式:

可以显示地赋值，还可以用语句变量赋值： for file in `ls /etc`；

赋值时，变量名和等号之间不可以有空格

使用变量：

使用时在变量名前加 $ 即可， 如 $var， 使用变量时可以加 ｛｝，也可以不加 ｛｝，但从编程习惯来讲，推荐加上｛｝；

已经定义的变量可以被重新定义；

只读变量：

先给变量赋值，然后使用 readonly 声明变量即可

删除变量：

使用unset删除变量，但是unset无法删除只读变量

Eg： unset 变量名

变量类型：

1. 局部变量：尽在当前shell中有效，其他shell不能访问
2. 环境变量：
3. Shell变量：

## 二、shell字符串

Shell可以 用单引号，也可以用双引号，也可以不用引号。

单引号：

单引号里的任何字符都会原样输出，单引号字符串中的变量是无效的；

单引号字符串中不能出现单引号（对单引号使用转义字符也不行）。

双引号：

双引号里可以有变量；

双引号里可以出现转义字符；

拼接字符串：

可以使用双引号做拼接，也可以使用‘+’号做拼接。

获取字符串长度：

echo ${#string}: 输出String的长度

提取字符串：

Echo ${string:1:4} : 输出string变量中从位置1开始的4个字符，位置从0开始计算；

查找子字符串：

echo `expr index “string” is` : 输出字符‘i’或‘s’的位置

## 三、shell数组

Shell只支持一维数组，不支持多维数组，并且没有限定数组的大小。且数据下标从0开始。

定义数组：

Shell中数组用“（）”表示，数组元素用“空格”分割开，一般形式为

数组名=（值1 值2 值3）；

可以单独定义数组的各个分量：arr[0] = val0，可以不使用连续的下标，且下标的范围没有限制；

读取数组元素：

${数组名[下标]}；

获取数组中的所有元素：

${array[@]}

获取数组的长度：

echo ${#arr[@]} 或 echo ${#arr[\*]}

获取数组单个元素的长度：

Echo ${#arr[n]}

## 四、shell注释

#用来注释；

没有多行注释，可以采用加｛ ｝的方式，将其视为一个函数，不调用即可实现多行注释的效果

## 五、shell传递参数

执行shell脚本时，可以传递参数，脚本内获取参数的格式为： $n. n代表一个数字，1为执行脚本的第一个参数， 2为执行脚本的第二个参数，依次类推。

（1）$0 代表的是脚本本身的名称；

（2）处理参数的几个特殊字符：

$#：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

$@: 以多个字符串显示所有向脚本传递的参数，每一个参数对应一个字符串

$$: 脚本运行的当前进程ID号

$?: 显示最后命令的退出状态。0表示没有错误，非零表示有错误。

$\*与$@的区别：

相同点：都是引用所有参数

不同点：只有在双引号中体现出来。假设脚本运行时写了3个参数1， 2， 3， \*等价于“1 2 3”，@ 等价于“1”，“2”，“3”。

## 六、shell运算符

Shell支持算术运算符，关系运算符，布尔运算符，字符串运算符，文件测试运算符

1. 算术运算符：

原生bash不支持简单的数学运算，但是可以通过其他命令来实现，例如 awk 和 expr, expr最常用。Expr是一款表达式计算工具，使用它能完成表达式的求值操作；

表达式和运算符之间要有空格 “2 + 2”，而不是“2+2”（错误）；

完整的表达式要被` `包含；

常用的运算符说明：

+ : 加法

-：减法

\*：乘法

/ ：除法

%：取余

= ：赋值

== : 用于比较两个数字，相同返回true

!= ：用于比较两个数字，不相同true

\*: 乘法乘号前必须加（\） 才能实现乘法；

在mac中shell的expr语法是：$((表达式)，表达式中的”\*” 不需要转义)；

Shell中只支持整数运算，不支持浮点运算.

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：

-eq: 检测两个数是否相等，相等返回true

-ne: 检测两个数是否不相等，不想等返回true

-gt：检测左边的数是否大于右边的，如果是返回true

-lt：检测左边的数是否小于右边的，如果是返回true

-ge：检测左边的数是否大于等于右边的，如果是，返回true

-le：检测左边的数是否小于等于右边的，如果是，返回true

1. 布尔运算符

常用的关系运算符:

! ：非运算

-o : 或运算

两个表达式有一个为true，则返回true

-a : 与运算

两个表达式都为true时，则返回true

1. 逻辑运算符

&&: 逻辑与 AND

||：逻辑或 OR

1. 字符串运算符

= ： 检测两个字符串是否相等，相等返回true

!= ：检测两个字符串是否不想等，不相等返回true

-z ：检测字符串长度是否为0，为0返回true

-n ：检测字符串长度是否不为0，不为0返回true

Str(字符串本身): 检测字符串是否为空，不空返回true

1. 文件测试运算符

-b file: 检测文件是否是块设备文件，如果是，则返回true

-c file：检测文件是否是字符设备文件，如果是，则返回true

-d file：检测文件是否是目录，如果是，则返回true

-f file：检测文件是否是普通文件（不是目录，也不是设备文件），如果是，则返回true

-g file：检测文件是否设置了SGID位，如果是，则返回true

-k file：检测文件是否射着了粘着位，如果是，则返回true

-p file：检测文件是否是有名管道，如果是，则返回true

-u file：检测文件是否设置了SUID位，如果是，则返回true

-r file: 检测文件是否可读，如果是，则返回true

-w file: 检测文件是否可写，如果是，则返回true

-x file: 检测文件是否可执行，如果是，则返回true

-s file: 检测文件是否为空（文件大小是否大于0），不空返回true

-e file: 检测文件（包括目录）是否存在，如果存在，则返回true

## 五、shell echo：

命令格式： echo 字符串

1. 显示普通字符串

Echo “….” (双引号可以省略)

1. 显示转义字符

Echo “ \” ” (双引号可省略，使用\进行转义)

1. 显示变量

Echo $var

1. 显示换行

Echo -e “ \n” ：使用-e开启转义， \n代表换行

1. 显示不换行

Echo –e “\c” : 使用-e开启转义， \n代表不换行

1. 显示结果定向至文件

Echo $var > file ： 使用 > 重定向至文件

1. 原样输出字符串，不进行转义或取变量（用单引号）

Echo ‘ $name \”’

1. 显示命令执行结果

Echo `date`

## 六、shell之printf命令

Printf使用引用文本或空格分隔的参数，可以在printf中使用格式化字符串，还可以指定字符串的宽度、左右对齐方式等，printf默认不会像echo自动添加换行符，可以手动添加“\n”; printf命令的语法如下：

Printf format-string [arguments…]

参数说明：

Format-string: 格式控制字符串

Arguments： 参数列表

Printf的转义序列说明：

\a：警告字符，通常为ASCII的BEL字符

\b: 后退

\f: 换页

\n: 换行

\r：回车

\t: 水平制表符

\v： 垂直制表符

[\\：一个字面上的反斜杠自负](file:///\\：一个字面上的反斜杠自负)

\ddd：表示1到3位八进制的字符。仅在格式字符串中有效

\0ddd：表示1到3位的八进制字符

## 七、shell之test命令

Shell中的test命令用于检查某个条件是否成立，它可以进行数值、字符和文件三个方面的测试。

1. 数值测试  
    -eq : 等于则为真

-ne： 不等于则为真

-gt: 大于则为真

-ge: 大于等于则为真

-lt： 小于则为真

-le： 小于等于则为真

[] : []中的代码执行基本的算术运算

1. 字符串测试：

= ： 相等则为真

!=: 不相等则为真

-z: 字符串的长度为零则为真

-n: 字符串的长度不为零则为真

（3） 文件测试：

-e: 文件存在为真

-r: 如果文件存在且可读为真

-w: 如果文件存在且可写为真

-x： 如果文件存在且可执行为真

-s: 如果文件存在且至少有一个字符为真

-d: 如果文件存在且为目录为真

-f: 如果文件存在且为普通文件则为真

-c: 如果文件存在且为字符型特殊文件则为真

-b: 如果文件存在且为特殊文件则为真

注： shell还提供了与（-a）、或（-o）、非（！）三个逻辑操作符，用于将测试条件连接起来，其优先级顺序排列为：！ --🡪 -a ---🡪

## 八、shell之流程控制

和java、PHP、c/c++等编程语言不一样，sh的控制流程不可为空，如else后边必须有执行语句，不能为空，如果else没有操作，就不要写else分支。

1. If

格式：

If 条件

Then

执行语句1

执行语句2

执行语句3

执行语句….

fi

1. If else

If 条件

Then 执行满足条件的语句

Else

执行不满足条件的语句

fi

1. If else-if else

If 条件1

Then 执行满足条件1的语句

Elif 条件2

Then 执行满足条件2的语句

Else

执行不满足条件1 和条件2 的语句

Fi

说明： if 语句经常与test命令结合在一起使用

1. for循环

shell支持for循环，for循环一般格式为

for var in item1 item2 item3 item4 …. itemN

do

command1

command2

…

commandN

done

也可写成一行

For var in item1 item2 … itemN；do command1 command 2 …. Done;

当变量值在列表里，for循环即执行一次所有命令，使用变量名获取列表中的当前取值，命令可为任何有效的shell命令和语句；

In列表可以包含替换、字符串和文件名；

In列表是可选的，如果不用它，for循环使用命令行的位置参数。

1. while循环

while循环用于不断执行一系列命令，也用于从输入文件中读取数据；命令通常为测试条件。其格式为：

while 条件

do

执行命令

done

1. 无限循环

While ：

Do

Command

Done

Wjhile true

Do

Command

Done

For ((;;))

command

done

(7) until循环

Until循环执行一系列命令直至条件为真时停止；

Until与while循环在处理方式上刚好相反；

一般while循环优先于until循环，但在某些时候也只是极少数情况下，until循环更加有用

Until condition

Do

Command

Done

说明：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。

1. case

case为多选择语句。可以用case语句匹配一个值与一个模式，如果匹配成功，执行相匹配的命令。

Case语句的格式如下：

Case 值 in

模式1）

Command1 ;;;

模式2）

Command2;;;

Esac

Case取值后面必须为单词in,每一模式必须以右括号结束；取值可以为变量或常数；匹配发现值符合某一模式后，期间所有命令执行至；；

取值将检测匹配的每一个模式。一旦模式匹配，则执行完匹配模式相应命令后不再继续其他模式；如果无一模式匹配，使用 \* 号捕获该值，再执行后面的命令；

1. 跳出循环（break continue）

在循环过程中，有时候需要在未达到循环结束条件时强制跳出循环，shell用两个命令来实现该功能: break和continue.

Break:

Break命令跳出所有循环，终止后面的所有循环；

Continue:

Continue命令与break命令类似，区别在于它仅会跳出当前循环，而不会跳出所有循环。

## 九、shell之函数

Linux shell可以用户定义函数，然后在shell脚本中可以随便调用。

Shell中函数的定义格式如下：

[ function ] funname[()]

{

Action;

[return int;]

}

说明：

1. 可以带function fun() 定义，也可以直接fun()定义， 不带任何参数
2. 参数返回，可以显示加：return 返回，如果不加，将以最后一条命令运行结果，作为返回值。Return后跟数值(0-255)
3. 所有函数在使用前必须定义

1. 函数参数

Shell，调用函数时可以向其传递参数。在函数体内部，通过$n的形式来获取参数的的值， 例如$1表示第一个参数

参数说明：

$10不能获取第十个参数，获取第十个参数需要 ${10}.当 n >= 10时，需要使用${n}来获取参数；

$# : 传递到脚本的参数个数

$\*: 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

$!: 后台运行的最后一个京城ID

$@： 与$\*相同，但是同时加引号，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

$?: 显示命令的退出状态。0表示没有错误，其他任何值都表示有错误

## 十、shell之输入输出重定向

大多数UNIX系统命令从你的终端接受输入并将产生的输出发送回您的终端。一个命令通常从一个叫标准输入的地方读取输入，默认情况下，这恰好是你的终端。同样，一个命令通常将其输出写入到标准输入，默认情况下，也是你的终端。

1. 重定向命令列表

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的文件一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

注： 文件描述符 0 ： STDIN

1. ： STDOUT
2. ： STDERR
3. 输出重定向

语法： command > file

说明： 执行command1然后将输出的内容存入file1;任何file1内的已经存在的内容将被新内容替代，如果要将新内容添加在文件末尾，请使用>>操作符；

1. 输入重定向：

语法：command1 < file1

说明：本来需要从键盘获取输入的命令会转移到文件读取内容

同时替换：

Command < infile1 > outfile

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

1. 重定向深入讲解

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程序默认人向stdout输出数据

标准错误文件： stderr的文件描述符为2，unix程序会向stderr流中写入错误信息

注：

将stderr重定向到file: command 2 > file

将stderr追加到file文件末尾： command 2 >> file

将stdout和stderr合并后重定向到file：command > file 2>&1 或

Command >> file 2>&1

将stdin和stdout都重定向：command < file1 > file2

1. Here Document

Here Document是shell中的一种特殊的重定向方式，用来将输入重定向到一个交互式shell脚本或程序

基本形式：

Command << delimiter

Document

Delimiter

作用是将两个demiliter之间的内容(document)作为输入传递给command

注意：

结尾的delimiter一定要顶格写，前面不嫩各有任何字符，后面也不能有任何字符，包括空格和tab缩进；

开始的delimiter前后的空格会被忽略掉；

1. 如果希望执行某个命令，但又不在屏幕上显示输出结果，那么可以将输出重定向到/dev/null

Command > /dev/null

/dev/null是一个特殊的文件，导入到它的内容都会被丢弃；如果尝试从该文件读取内容，那什么也读不到。但是/dev/null文件非常有用，将命令的输出重定向到它，会起到“禁止输出”的效果。

屏蔽stdout 和stderr :

Command > /dev/null 2>&1

## 十一、shell之文件包含

Shell可以包含外部脚本。这样可以很方便的封装一些公用的代码作为一个独立文件。

Shell文件包含的语法格式如下：

.filename 或 source filename