# 变量

## 变量类型

根据变量类型确定的时间，可以将程序设计语言分为：静态类型语言和动态类型语言。

根据是否强制要求类型定义，可以将程序设计语言分为：强类型语言和弱类型语言。

Shell语言是一种动态类型语言和弱类型语言。即在shell中，变量的数据类型无需显式地声明，变量的数据类型会根据不同的操作有所变化。准确地讲，shell中的变量是不分数据类型的，统一地按照字符串存储。

## 普通变量

### 变量命名

定义变量时，变量名不加美元$，**变量名和等号之间不能有空格**（python中可以增加空格），这个和我们熟悉的编程语言都不一样。

注：在shell语言中，变量名的大小是敏感的，因此，大小写不同的两个变量名并不代表同一个变量。

### 变量定义

为了更好地控制变量的相关属性，bash提供了一个名称为declare的命令来声明变量，语法如下：

declare attribute variable

其中attribute表示变量的属性，如下：

-p:显示所有变量的值

-i:将变量定义为整数

-r:将变量声明为只读变量，只读变量不允许修改，也不允许删除

-a:将变量声明为数组变量

-f:显示所有自定义函数，包括名称和函数体

-x:将变量设置成环境变量，这样在随后的脚本和程序中可以使用

注：declare命令又写做typeset。

### 变量和引用

Shell语言中一共有3种引号，分别为单引号(‘’)、双引号(“”)和反引号(``)。这3个引号作用是不同的：单引号括起来的字符都作为普通字符出现；由双引号括起来的字符，除“$”、“\”、“’”、““”这几个字符仍是特殊字符并保留其特殊功能外，其余字符扔作为普通字符对待；由反引号括起来的字串被shell解释为命令，在执行时，shell首先执行该命令，并以它的标准输出结果取代整个反引号（包括两个反括号）部分。

### 变量作用域

#### 全局变量

#### 局部变量

在shell语言中，可以在函数内部通过local关键字定义局部变量，另外，函数的参数也是局部变量。

注：在出现同名的情况下，函数内部的局部变量会优先被使用。

### 变量赋值

在shell中，变量的赋值使用以下语法：

variable name=value

一般情况下，shell中将所有普通变量的值都看做字符串。如果value中包含空格、制表符和换行符，则必须用单引号或者双引号将其引起来。双引号内允许变量替换，则单引号则不可以。

注：中间的等于号“=”称为赋值符号，赋值符号的左右两边不能直接跟空格，否则shell将其视为命令。

### 使用变量

使用一个定义过的变量，只要在变量名前面加美元符号$即可（python不需要加$）,如$name/${name}，变量名外面的花括号是可选的，加花括号是为了帮助解释器识别变量的边界。

推荐给所有变量加上花括号，这是良好的编程习惯。

### 清除变量

清除变量使用unset语句：

unset variable name

## 系统变量

$- 显示shell使用的当前选项，与set命令功能相同

$n：表示参数的位置

$#：命令行参数的个数

$0：当前shell脚本的名称

$\*：以“参数1 参数2……”的形式将所有的参数通过一个字符串返回

$@：以“参数1”“参数2”……的形式返回每个参数

$?：前一个命令或者函数的返回状态码

$! 后台运行的最后一个进程的ID号

$$：返回本程序的进程ID

$\*与$@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）。

$@与$\*相同，但是使用时加引号，并在引号中返回每个参数。

## 环境变量

## 数组

### 定义

Bash支持一维数组（不支持多维数组），并且没有限定数组的大小。类似C语言，数组元素的下标由0开始，获取数组中元素要利用下标，下标可以是整数或算术表达式，其值应该大于或等于0。

#### 通过指定元素值定义数组

在shell中，用户可以通过直接指定数组中的元素值来定义一个新的数组变量，语法如下：

array[key]=value

#### 通过declare语句定义数组

可以使用declare来声明数组：

declare –a array

#### 通过元素值集合定义数组

在shell中，用括号来表示数组，数组元素用空格符号分来（不是逗号），定义数组的一般形式为：

数组名=(值1 值2… 值n)

#### 通过键值对定义数组

为数组元素赋值时，一般都是连续赋值的。实际上，在shell中，数组元素的索引并不一定是连续的。

用户可以使用键值对的方式来定义数组，如下：

array=([0]=value0 [1]=value1 … [n]=valuen)

### 赋值

#### 按索引为元素赋值

#### 通过集合为数组赋值

#### 在数组末尾追加新元素

#### 通过循环为数组元素赋值

### 复制数组

复制数组是指创建一个已经存在的数组的副本，语法如下：

newarray=(“${array[@]}”)

### 连接数组

数组连接的语法如下：

(“${array1[@]}” “${array2[@]}”)

注：在执行数组连接时，参与连接的数组之间要保留1个空格。

### 加载文件到数组

在shell中，用户可以将普通的文本文件的内容直接加载到数组中，文件的每一行构成数组一个元素的内容。

示例：

#!/bin/bash

#加载文件内容

content=(`cat “demo.txt”`)

#通过循环输出数组内容

for s in “${content[@]}”

for

echo “$@”

done

### 访问数组

读取数组元素值的一般格式为：

${数组名[下标]}

注：用数组名[@]表示获取数组中的所有元素。

#### 访问第1个数组元素

#### 通过下标访问数组元素

#### 计算数组的长度

#### 通过循环遍历数组元素

#### 引用所有的数组元素

#### 以切片方式获取部分数组元素

#### 数组元素替换

### 删除数组

删除指定数组元素：

unset array[n]

删除整个数组：

unset array

### 获取数组长度

获取数组元素的个数：

${#array\_name[@]}/${#array\_name[\*]}

获取数组中单个元素的长度：

${#array\_name[n]}

## 字符串

### 定义

字符串可以用单引号、双引号，也可以不用引号。

单引号字符串的限制：

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时无效的

单引号字符串中不能出现单引号

双引号的优点：

双引号里可以有变量

双引号里可以出现转义字符

### 拼接

### 获取长度

${#stirng}

### 提取子字符串

${string m:m} #输出m到n的字符

### 查找

`expr index “string” is` #查找is的位置

### 匹配

${var##\*string} #对var从左到右截取最后一个string后的字符串

${var#\*string} #从左向右截取第一个string后的字符串

${var%%string\*} #从左向右截取最后一个string后的字符串

${var%string\*} #从左向右截取第一个string后的字符串

# 函数

## 函数

### 定义

### 调用

### 函数链接

函数链接并不仅仅局限于在某个函数中调用另外一个函数，还可以进行多层嵌套调用，或者在某个函数中调用多个函数。

注：在函数嵌套调用时，一定要注意函数定义的顺序和调用的顺序。必须按照县定义，后调用的原则。

### 返回值

在shell中，return语句只能返回某个0~255之间的整数值。

当一个shell脚本执行结束的时候，可以通过exit命令返回其退出状态码。与之相类似，函数中的return语句实际上是用来返回函数的退出状态码的。

### 别名

用户可以使用alias命令来设置别名，基本语法：

alias name=”command”

函数和别名的相似之处在于，它们都通过一个名称映射到一个或者一组命令。与函数相比，别名的功能相对比较 简单，其主要区别在于：

1. 用户不能为一组命令指定别名；
2. 别名中不能通过系统变量操作参数列表。

当用户不需要某个别名时，可以使用unalias命令将其删除。

## 函数参数

### 获取函数参数的个数

### 通过位置变量接收参数值

### 移动位置参数

在shell脚本中，用户可以使用shift命令来使得脚本的所有的位置参数向左移动一个位置，从而使得用户可以通过9以内的位置变量来获取超过9个的参数。

注：shift命令会影响到系统变量$#的值。

### 通过getopts接收函数参数

getopts是bash内置的一个命令，通过该命令，用户可以获取函数的选项以及参数值，或者是脚本的命令行选项以及参数值。语法如下：

getopts optstring [args]

### 间接参数传递

### 通过全局变量传递数据

### 传递数组参数

## 函数库文件

## 递归函数

# 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。

## 数值测试

一般是将参数列表与预期的进行比较，即$1、$#等。

-eq 等于则为真

-ne 不等于则为真

-gt 大于则为真

-ge 大于等于为真

-lt 小于则为真

-le 小于等于则为真

## 字符串测试

= 等于则为真

!= 不相等则为真

-z 字符串 字符串的长度为零则为真

-n 字符串 字符串的长度不为零则为真

## 文件测试

-e 文件名 如果文件存在则为真

-f 文件名 如果文件存在且可读则为真

-w 文件名 如果文件存在且可写则为真

-x 文件名 如果文件存在且可执行则为真

-s 文件名 如果文件存在且至少有一个字符则为真

-d 文件名 如果文件存在且为目录则为真

-f 文件名 如果文件存在且为普通文件则为真

-c 文件名 如果文件存在且为字符型特殊文件则为真

-b 文件名 如果文件存在且为块特殊文件则为真

# 流程控制

## if else

## for

for循环的一般格式为：for var in item1 item2 … itemN

## while

## 无限循环

## util

## case

## 跳出循环

### break

### continue

## exit

# echo命令

## 显示普通字符

## 显示转义字符

## 显示变量

## 显示换行

## 显示不换行

## 显示结果定向至文件

## 原样输出字符串

## 显示命令执行结果

## 操作命令执行结果

# print命令

# 重定向

## 定义

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到你的终端。一个命令通常从一个标准输入的地方读取输入，默认情况下，这恰是你的终端。同样，一个命令通常将其输出写入到标准输出，默认情况下，这也是的终端。

重定向命令如下：

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m合并

<<tag 将开始标记tag和结束标记tag之间的内容作为输入

## 输出重定向

重定向一般通过在命令间插入特定的符号来实现，语法如下：

command > file1

上面这个命令执行command然后将输出的内容存入file1。

注：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，轻使用>>操作符。

## 输入重定向

和输出重定向一样，Unix命令也可以从文件获取输入，语法为：

command < file1

这样，本来需要从键盘获取输入的命令会转移到文件读取内容。

注意：输出重定向是大于号（>），输入重定向是小于号（<）。

一般情况下，每个Unix/Linux命令运行时都会打开三个文件：

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file。

## /dev/null

如果希望执行某个命令，但又不希望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，写入到它的内容都会被丢弃；如果尝试从该文件读取内容，那么什么也读不到。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。

如果希望屏蔽stdout和strerr，可以这样写：

$command > /dev/null 2>&1

注：0是标准输入（STDIN），1是标准输出（STDOUT），3是标准错误输出（STDERR）。

# 正则表达式

# grep

# ----

# 基本文本处理

## echo

## 文本格式化输出

## sort

## 文本统计

## cut

## paste

## join

## tr

# 流编辑sed

## 简介

Linux sed命令是利用script来处理文本文件。Sed可依照script的指令，来处理、编辑文本文件。Sed主要用来自动编辑一个或多个文件，简化对文件的反复操作，编写转换程序等。

sed语法：

sed [-hnV] [-e<script>][-f<script文件>][文本文件]

参数说明：

-e<script>/-expression=<script> 仅显示script处理后的结果

-f<script文件>/-file=<script文件> 以选项中指定的script文件来处理输入的文本文件

-h/--help 显示帮助

-n/-quiet/--silent 仅显示script处理后的结果

-V/--version 显示版本信息

-r sed动作支持的是延伸型正则表示法的语法（默认是基础正则表示法语）

-i 直接修改读取的文件内容，而不是输出到终端

动作说明：

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，这些字串可以取代n1，n2之间的行

d删除：因为是删除，所以d后面通常不接任何东西

i插入：i的后面可以接字串，而这些字串会在新的一行出现（目前的上一行）

p打印：将某个选择的数据印出，通常p会与参数sed –n一起运行

s取代：可以直接进行取代的工作，通常这个s的动作可以搭配正则表达式

## 查找

使用grep或者awk过滤

## 选择

在sed命令中，选择文本行主要通过位置参数来完成，基本语法如下：

[address1[,address2]] p

## 插入

追加文本是在位置参数指定的位置的后面插入文本，而插入文本则是在位置参数指定的位置的前面插入文本。

在sed命令中，子命令i用来实现文本的插入，其语法如下：

[address1] i string

## 追加

sed命令提供了a子命令来实现文本的追加(append)，其语法如下：

[address1] a string

## 取代

文本的替换要使用s子命令，其语法如下：

[address1] [,address2] s/pattern/replacemen/[flag]

在s命令中，位置参数通常会被省略，表示在所有的文本行中进行替换操作，其语法如下：

s/pattern/replacemen/[flag]

s命令表示执行替换(substitute)操作，pattern为使用正则表达式表示的匹配模式，replacement为用来替换的由一般字符组成的字符串。

flag为替换标志，会影响s命令的行为，通常选项：

g：全局匹配，会替换文本行中所有符合规则的字符串

十进制数字：如果flag为一个十进制数字n，则表示替换文本行中第n个符合规则的字符串

p：替换第1个符合规则的字符串，并且将缓冲区输出到标准输出

w：替换第1个符合规则的字符串，并且将受影响的行输出到磁盘文件中

空：如果没指定flag，则表示替换文本行中第1个符合规则的字符串

## 删除

sed提供了d子命令来实现文本行的删除(delete)，其语法如下：

[address1 [,address2]] d

命令d表示删除位置参数指定的行。如果省略位置参数，则表示删除文本文件中的所有的行。

在执行删除操作时，sed命令会首先读取一行文本到缓冲区，然后将符合位置参数的文本行删除，接着读取并处理下一行。

## 组合命令

### 使用-e选项执行多个子命令

sed命令的-e选项可以使sed将跟在其后面的字符串作为子命令执行。

### 使用分号执行多个子命令

用户还可以使用分号来将各个子命令隔开，其语法如下：

sed –e ‘command1;command2;…’ filename

# awk

## 简介

### 基本语法

awk命令的基本语法如下：

awk pattern {actions}

pattern表示匹配模式，actions表示要执行的操作。Pattern和actions都是可选的，但是二者必须保证有一个。

匹配模式包括：

正则表达式：需要用斜线将正则表达式包围起来

关系表达式：

模式1，模式2：指定一个行的范围，该语法不能包括BEGIN和END模式

BEGIN：让用户指定在第1行文本被处理之前所发生的操作，通常可在这里设置全局变量

END：让用户在最后1行文本被读取之后发生的操作

awk命令的操作由一个或者多个命令、函数或者表达式组成，它们之间由换行度或者分号隔开，并且位于大括号内。通常情况下，有以下4中操作：

1. 变量或者数组赋值
2. 输出命令，例如printf或者print
3. 内置函数
4. 流程控制语句，例如if、while、for

### 工作流程

### 执行方式

## 模式匹配

### 关系表达式

### 正则表达式

### 混合模式

### 区间模式

### BEGIN模式

### END模式

## 变量

### 定义

### 引用

### 系统内置变量

### 记录和字段

#### 分隔符

#### 引用

## 数组

### 定义

### 赋值

### 遍历

## 运算符和表达式

### 算术运算符

### 赋值运算符

### 条件运算符

### 逻辑运算符

### 关系运算符

### 其他运算符

## 函数

### 字符串函数

### 算术函数

## 流程控制

## 格式化输出

## awk与shell交互

# 文件操作

# 进程

# 调试