## 一、 基础概念

### 1. shell 介绍

Shell就是一个命令行解释器，它的作用是解释执行用户的命令,用户输入一条命令,Shell就解释执行一条,这种方式称为交互式(Interactive)。

Shell还有一种执行命令的方式称为批处理(Batch),用户事先写一 个Shell脚本(Script),其中有很多条命令,让Shell一次把这些命令执行完,而不必一条一条地敲 命令。Shell脚本和编程语言很相似,也有变量和流程控制语句,包括循环和分支。但Shell脚本是解释执行的,不需 要编译,Shell程序从脚本中一行一行读取并执行这些命令,相当于一个用户把脚本中的命令一行一 行敲到Shell提示符下执行。作为程序设计语言，它虽然不是 Linux系统内核的一部分，但它调用了系统内核的大部分功能来执行程序、创建文档并以并行的方式协调各个程序的运行。

### 2. 常见的shell

Unix系统中常见的shell有：sh、csh、ksh、tcsh、bash

使用命令： vim /etc/shells 可以查看当前系统支持的bash

下面是对sh、bash的介绍，意思就是bash是Linux默认的shell，是对sh的增强扩展，与sh兼容。

① sh(全称 Bourne Shell): 是UNIX最初使用的 shell，而且在每种 UNIX 上都可以使用。

Bourne Shell 在 shell 编程方便相当优秀，但在处理与用户的交互方便作得不如其他几种 shell。

② bash（全称 Bourne Again Shell）: LinuxOS 默认的，它是 Bourne Shell 的扩展。

与 Bourne Shell 完全兼容，并且在 Bourne Shell 的基础上增加了很多特性。可以提供命令补全，命令编辑和命令历史等功能。它还包含了很多 C Shell 和 Korn Shell 中的优点，有灵活和强大的编辑接口，同时又很友好的用户界面。

在一般情况下，人们并不区分 Bourne Shell 和 Bourne Again Shell，所以，像 #!/bin/sh，它同样也可以改为#!/bin/bash。

#!告诉系统其后路径所指定的程序即是解释此脚本文件的Shell程序。

### 3. 举个例子

cat test.sh

#!/bin/bash

# comment row

echo "Hello World !"

shell脚本的第一行按惯例都写上 #!/bin/bash 以指定解释次脚本的shell程序；

shell中行首用#来注释一行，注释是对shell程序的描述，多行注释每行就都加#；

echo 是打印输出，字符串一般用引号引起来。

shell脚本文件后缀名一般是sh，不是必须，想洗头一些service脚本都没有.sh的后缀。

脚本文件要有执行权限才可以以绝对路径或在脚本目录下以./test.sh这种方式运行；

如果没有执行权限，需要用sh或是bash命令去执行：

sh test.sh

sh在系统中是bash的一个软连接，所以sh bash命令一样。

## 二、 shell 变量

### 1. 变量定义

定义变量的格式： 变量名=变量值

注意：变量名和等号之间不能有空格

如：

your\_name=jack

yourName="jack"

字符串如果是连续的可以不加引号，如果有空格或是其他特殊字符需加引号；

变量命名规范一般有下划线和驼峰式，根据个人习惯来用，

一般变量名建议用下划线，函数建议用驼峰式。

变量名的命名规则：

①首个字符必须为字母（a-z，A-Z）；

②中间不能有空格，可以使用下划线（\_）；

③不能使用标点符号；

④不能使用bash里的关键字（可用help命令查看保留关键字）。

### 2. 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

echo $your\_name

echo ${your\_name}

变量名外面的花括号是可选的，加不加都行，加花括号是为了帮助解释器识别变量的边界，比如下面这种情况：

echo "I am good at ${skill}Script"

### 3. 其他变量相关

1) 只读变量

用readonly修饰

readonly your\_name

2) 删除变量

用unset删除变量

unset your\_name

3) 变量分类

①局部变量

局部变量在脚本或命令中定义，仅在当前shell实例中有效，其他shell启动的程序不能访问局部变量；

②环境变量 所有的程序，包括shell启动的程序，都能访问环境变量，有些程序需要环境变量来保证其正常运行。必要的时候shell脚本也可以定义环境变量；

③shell变量 shell变量是由shell程序设置的特殊变量。shell变量中有一部分是环境变量，有一部分是局部变量，这些变量保证了shell的正常运行；

## 三、 shell字符串

### 1. 用单双引号的字符串

字符串可以用单引号、双引号来引用。

str='this is a string'

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是无效的；
* 单引号字串中不能出现单引号（对单引号使用转义符后也不行）。

str="Hello, I know your are \"$your\_name\"! \n"

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

### 2. 字符串相关操作

1) 拼接字符串

a变量字符串和b变量字符串拼接，默认写在一起就是拼接；

如果两个字符串变量引用间有空格，打印是用1个空格拼接2个字符串的非空部分；

[root@localhost ~]# a='hello ' b='world'

[root@localhost ~]# echo $a$b

hello world

[root@localhost ~]# echo $a $b

hello world

[root@localhost ~]# a='hello' b='world'

[root@localhost ~]# echo $a$b

helloworld

[root@localhost ~]# echo $a $b

hello world

2) 获取字符串长度

[root@localhost ~]# a='hello'

[root@localhost ~]# echo ${#a}

5

3) 截取字符串

以下实例从字符串第 2 个字符开始截取 2 个字符：

[root@localhost ~]# a='hello'

[root@localhost ~]# echo ${a:1:2}

el

4) 查找子字符串出现的位置

查找hello中o出现的位置，

[root@localhost ~]# a='hello'

[root@localhost ~]# echo `expr index "$a" o`

## 四、 shell数组

### 1. shell数组介绍

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。类似与C语言，数组元素的下标由0开始编号。获取数组中的元素要利用下标，下标可以是整数或算术表达式，其值应大于或等于0。

### 2. 定义数组

定义格式：

数组名=(值1 值2 ... 值n)

array\_name=(value0 value1 value2 value3)

还可以单独定义数组的各个元素：

array\_name[0]=value0

array\_name[1]=value1

array\_name[n]=valuen

### 3. 数组操作

1) 读取数组

格式：

${数组名[下标]}

value1=${array\_name[1]}

2) 获取数组长度

获取数组长度的方法与获取字符串长度的方法相同，

格式：

length=${#array\_name[@]}

# 或者

length=${#array\_name[\*]}

例子：

[root@localhost ~]# array\_name[0]=value0

[root@localhost ~]# array\_name[1]=value1

[root@localhost ~]# array\_name[2]=value2

[root@localhost ~]# echo ${#array\_name[@]}

3

[root@localhost ~]# echo ${#array\_name[\*]}

3

获取数据单个元素的长度：

[root@localhost ~]# echo ${#array\_name[1]}

6

3) 获取数组所有元素

[root@localhost ~]# echo ${array\_name[@]}

value0 value1 value2

## 五、shell脚本传参

在执行 Shell 脚本时，向脚本传递参数，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，以此类推。

### 1. 实例

创建一个test.sh脚本，内容如下：

[root@localhost ~]# cat test.sh

#!/bin/bash

# author: yangruipeng

# date: 2017-3-28

# version: 1.0

# email: yrp696@163.com

echo "Shell 传递参数实例！";

echo "执行的文件名：$0";

echo "第一个参数为：$1";

echo "第二个参数为：$2";

echo "第三个参数为：$3";

说明：

#号开头的行表示注释；

$0表示当前执行的脚本文件名；

$1 $2 $3表示命令行传递给脚本的3个参数

执行结果如下：

[root@localhost ~]# chmod u+x test.sh

[root@localhost ~]# ./test.sh 1 2 3

Shell 传递参数实例！

执行的文件名：./test.sh

第一个参数为：1

第二个参数为：2

第三个参数为：3

### 2. 几个常用的处理参数的变量

$0 shell脚本本身文件名

$? 最后运行的命令的结束代码,即返回值；

$# 传递到脚本的参数个数；

$\* 所有参数列表，用双引号引用时以一个单字符串显示"$1 $2 … $n"的形式输出;

$@ 所有参数列表，用双引号引用时以每个参数作为一个字符串显示，以"$1" "$2" … "$n" 的形式输出

## 六、 shell运算符

### 1. 运算符分类

Shell 和其他编程语言一样，支持多种运算符，包括：

* 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

### 2. 算数运算符

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，

注意数字和运算符之间有1个空格。

[root@localhost ~]# expr 2 + 2

4

下表列出了常用的算术运算符，假定变量 a 为 10，变量 b 为 20：

|  |  |  |
| --- | --- | --- |
| **运算符** | **说明** | **举例** |
| + | 加法 | `expr $a + $b` 结果为 30。 |
| - | 减法 | `expr $a - $b` 结果为 -10。 |
| \* | 乘法 | `expr $a \\* $b` 结果为  200。 |
| / | 除法 | `expr $b / $a` 结果为 2。 |
| % | 取余 | `expr $b % $a` 结果为 0。 |
| = | 赋值 | a=$b 将把变量 b 的值赋给 a。 |
| == | 相等。用于比较两个数字，相同则返回 true。 | [ $a == $b ] 返回 false。 |
| != | 不相等。用于比较两个数字，不相同则返回 true。 | [ $a != $b ] 返回 true。 |

### 3. 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。

|  |  |  |
| --- | --- | --- |
| **运算符** | **说明** | **举例** |
| -eq | 检测两个数是否相等，相等返回 true。 | [ $a -eq $b ] 返回 false。 |
| -ne | 检测两个数是否相等，不相等返回 true。 | [ $a -ne $b ] 返回 true。 |
| -gt | 检测左边的数是否大于右边的，如果是，则返回 true。 | [ $a -gt $b ] 返回 false。 |
| -lt | 检测左边的数是否小于右边的，如果是，则返回 true。 | [ $a -lt $b ] 返回 true。 |
| -ge | 检测左边的数是否大于等于右边的，如果是，则返回 true。 | [ $a -ge $b ] 返回 false。 |
| -le | 检测左边的数是否小于等于右边的，如果是，则返回 true。 | [ $a -le $b ] 返回 true。 |

下表列出了常用的关系运算符，假定变量 a 为 10，变量 b 为 20：

### 4. 布尔运算符

下表列出了常用的布尔运算符，假定变量 a 为 10，变量 b 为 20：

|  |  |  |
| --- | --- | --- |
| **运算符** | **说明** | **举例** |
| ! | 非运算，表达式为 true 则返回 false，否则返回 true。 | [ ! false ] 返回 true。 |
| -o | 或运算，有一个表达式为 true 则返回 true。 | [ $a -lt 20 -o $b -gt 100 ] 返回 true。 |
| -a | 与运算，两个表达式都为 true 才返回 true。 | [ $a -lt 20 -a $b -gt 100 ] 返回 false。 |

### 5. 逻辑运算符

以下介绍 Shell 的逻辑运算符，假定变量 a 为 10，变量 b 为 20:

|  |  |  |  |
| --- | --- | --- | --- |
| **运算符** | **说明** | **举例** | |
| && | 逻辑的 AND | | [[ $a -lt 100 && $b -gt 100 ]] 返回 false |
| || | 逻辑的 OR | | [[ $a -lt 100 || $b -gt 100 ]] 返回 true |

### 6. 字符串运算符

下表列出了常用的字符串运算符，假定变量 a 为 "abc"，变量 b 为 "efg"：

|  |  |  |
| --- | --- | --- |
| **运算符** | **说明** | **举例** |
| = | 检测两个字符串是否相等，相等返回 true。 | [ $a = $b ] 返回 false。 |
| != | 检测两个字符串是否相等，不相等返回 true。 | [ $a != $b ] 返回 true。 |
| -z | 检测字符串长度是否为0，为0返回 true。 | [ -z $a ] 返回 false。 |
| -n | 检测字符串长度是否为0，不为0返回 true。 | [ -n $a ] 返回 true。 |
| str | 检测字符串是否为空，不为空返回 true。 | [ $a ] 返回 true。 |

### 7. 文件测试运算符

文件测试运算符用于检测 Unix 文件的各种属性。

属性检测描述如下：

|  |  |  |
| --- | --- | --- |
| **操作符** | **说明** | **举例** |
| -b file | 检测文件是否是块设备文件，如果是，则返回 true。 | [ -b $file ] 返回 false。 |
| -c file | 检测文件是否是字符设备文件，如果是，则返回 true。 | [ -c $file ] 返回 false。 |
| -d file | 检测文件是否是目录，如果是，则返回 true。 | [ -d $file ] 返回 false。 |
| -f file | 检测文件是否是普通文件（既不是目录，也不是设备文件），如果是，则返回 true。 | [ -f $file ] 返回 true。 |
| -g file | 检测文件是否设置了 SGID 位，如果是，则返回 true。 | [ -g $file ] 返回 false。 |
| -k file | 检测文件是否设置了粘着位(Sticky Bit)，如果是，则返回 true。 | [ -k $file ] 返回 false。 |
| -p file | 检测文件是否是有名管道，如果是，则返回 true。 | [ -p $file ] 返回 false。 |
| -u file | 检测文件是否设置了 SUID 位，如果是，则返回 true。 | [ -u $file ] 返回 false。 |
| -r file | 检测文件是否可读，如果是，则返回 true。 | [ -r $file ] 返回 true。 |
| -w file | 检测文件是否可写，如果是，则返回 true。 | [ -w $file ] 返回 true。 |
| -x file | 检测文件是否可执行，如果是，则返回 true。 | [ -x $file ] 返回 true。 |
| -s file | 检测文件是否为空（文件大小是否大于0），不为空返回 true。 | [ -s $file ] 返回 true。 |
| -e file | 检测文件（包括目录）是否存在，如果是，则返回 true。 | [ -e $file ] 返回 true。 |

## 七、 echo printf test命令

### 1. echo 命令

显示普通字符串：

[root@localhost ~]# echo "It is a test"

It is a test

显示转义字符串：

[root@localhost ~]# echo "\"It is a test\""

"It is a test"

显示变量：

[root@localhost ~]# name=jack

[root@localhost ~]# echo "$name It is a test"

jack It is a test

显示换行和不换行：

-e是开启转义， \n表示换行， \c表示不换行

[root@localhost ~]# echo -e "OK! \n"

OK!

[root@localhost ~]# echo -e "OK! \c"

OK! [root@localhost ~]#

将输出重定向至文件

[root@localhost ~]# echo "hello world" > test.txt

[root@localhost ~]# cat test.txt

hello world

显示命令执行结果：

[root@localhost ~]# echo $(date)

Tue Mar 28 14:52:11 CST 2017

[root@localhost ~]# echo `date`

Tue Mar 28 14:52:16 CST 2017

以上表示执行命令有两种方式：$()和反引号

### 2. printf命令

printf 命令模仿 C 程序库（library）里的 printf() 程序标准所定义，因此使用printf的脚本比使用echo移植性好。printf 使用引用文本或空格分隔的参数，外面可以在printf中使用格式化字符串，还可以制定字符串的宽度、左右对齐方式等。默认printf不会像 echo 自动添加换行符，我们可以手动添加 \n。

printf 命令的语法：

printf format-string [arguments…]

[root@localhost ~]# printf hello

hello[root@localhost ~]# printf "hello\n"

hello

示例：

[root@localhost ~]# cat test.sh

#!/bin/bash

printf "%-10s %-8s %-4s\n" 姓名 性别 体重kg

printf "%-10s %-8s %-4.2f\n" 郭靖 男 66.1234

printf "%-10s %-8s %-4.2f\n" 杨过 男 48.6543

printf "%-10s %-8s %-4.2f\n" 郭芙 女 47.9876

[root@localhost ~]# ./test.sh

姓名 性别 体重kg

郭靖 男 66.12

杨过 男 48.65

郭芙 女 47.99

%s %c %d %f都是格式替代符

%-10s 指一个宽度为10个字符（-表示左对齐，没有则表示右对齐），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。

下表是一些可能会用到的printf的转义序列

|  |  |
| --- | --- |
| **序列** | **说明** |
| \b | 后退 |
| \c | 抑制（不显示）输出结果中任何结尾的换行字符（只在%b格式指示符控制下的参数字符串中有效），而且，任何留在参数里的字符、任何接下来的参数以及任何留在格式字符串中的字符，都被忽略 |
| \f | 换页（formfeed） |
| \n | 换行 |
| \r | 回车（Carriage return） |
| \t | 水平制表符 |
| \v | 垂直制表符 |
| \\ | 一个字面上的反斜杠字符 |

### 3. test 命令

1) 数值测试

|  |  |
| --- | --- |
| **参数** | **说明** |
| -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 |

[root@localhost ~]# num1=100 num2=100

[root@localhost ~]# test $num1 -eq $num2

[root@localhost ~]# echo $?

0

[root@localhost ~]# test $num1 -gt $num2

[root@localhost ~]# echo $?

1

$?返回值为0说明上一条执行的结果为真，即$num1 等于 $num2是真的；

$?返回值为1说明上一条执行的结果为假，即$num1 大于 $num2是假的；

2) 字符串测试

|  |  |
| --- | --- |
| **参数** | **说明** |
| = | 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 |

[root@localhost ~]# str1='hello' str2='world'

[root@localhost ~]# test $str1 != $str2

[root@localhost ~]# echo $?

0

[root@localhost ~]# test $str1 = $str2

[root@localhost ~]# echo $?

1

返回值为0表示上一条执行结果为真，即 $str1 != $str2为真；

返回值为1表示上一条执行结果为假，即 $str1 = $str2为假；

3) 文件测试

|  |  |
| --- | --- |
| **参数** | **说明** |
| -e 文件名 | 如果文件存在则为真 |
| -r 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 |

[root@localhost ~]# test -e /etc/hosts

[root@localhost ~]# echo $?

0

[root@localhost ~]# test -e /etc/host

[root@localhost ~]# echo $?

1

/etc/hosts文件存在，所以$?返回值为0，即为真；

/etc/host文件不存在，所以$?返回值不为0，即为假；

### 4. read 命令

read命令接收标准输入（键盘）的输入，或其他文件描述符的输入。得到输入后，read命令将数据放入一个标准变量中。

[root@localhost ~]# read -p "Please input a str: " str1

Please input a str: hello

[root@localhost ~]# echo $str1

hello

上面的脚本读取命令行输入字符到变量str1，然后打印变量str1

常用选项：

-a 将输入读入一个数组

**-**d ：表示delimiter，即定界符

-p 指定提示语句

-n 指定字符个数

-t 指定等待时间

-s 输入不在终端显示（用于输入密码）

-r 指定读取命令把一个 \ (反斜杠) 处理为输入行的一部分，而不把它作为一个控制字符

## 八、流程控制

### 1. if 判断语句

if elif else语句格式：

if condition

then

command1

elif

command2

…

else

command3

fi

其中elif可以有0个或多个；

else可以有0个或1个；

没有elif和else的格式如下：

if condition

then

command1

fi

例子：

[root@localhost ~]# ps -ef|grep -c "ssh"

4

[root@localhost ~]# if [ $(ps -ef | grep -c "ssh") -gt 1 ]; then echo "true"; fi

true

[root@localhost ~]# if [ $(ps -ef | grep -c "ssh") -gt 1 ]; then echo "true"; else echo "false"; fi

true

[root@localhost ~]# if [ $(ps -ef | grep -c "ssh") -gt 5 ]; then echo "true"; else echo "false"; fi

false

### 2. case选择语句

Shell case语句为多选择语句。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语句格式如下：

case 值 in

模式1)

command1

command2

...

commandN

;;

模式2）

command1

command2

...

commandN

;;

Esac

case工作方式如上所示。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。匹配发现取值符合某一模式后，其间所有命令开始执行直至 ;;。

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。如果无一匹配模式，使用星号 \* 捕获该值，再执行后面的命令。

例子：

[root@localhost ~]# cat test.sh

#!/bin/bash

echo '输入 1 到 4 之间的数字:'

echo '你输入的数字为:'

read aNum

case $aNum in

1) echo '你选择了 1'

;;

2) echo '你选择了 2'

;;

3) echo '你选择了 3'

;;

4) echo '你选择了 4'

;;

\*) echo '你没有输入 1 到 4 之间的数字'

;;

esac

[root@localhost ~]# ./test.sh

输入 1 到 4 之间的数字:

你输入的数字为:

1

你选择了 1

[root@localhost ~]# ./test.sh

输入 1 到 4 之间的数字:

你输入的数字为:

5

你没有输入 1 到 4 之间的数字

### 3. for循环

Shell中最常用的循环就是for循环，

一般格式如下：

for var in item1 item2 ... itemN

do

command1

command2

...

commandN

done

写成一行

for var in item1 item2 ... itemN; do command1; command2… done;

例子：

[root@localhost ~]# for i in 1 2 3 4 5

> do

> echo $i

> done

1

2

3

4

5

### 4. while循环

while循环和其他编程语言的类似

一般格式如下：

while condition

do

command

done

例子：

[root@localhost ~]# while (($i<=5))

> do

> echo $i

> let "i++"

> done

1

2

3

4

5

使用中使用了 Bash let 命令，它用于执行一个或多个表达式，变量计算中不需要加上 $ 来表示变量。

无限循环语法格式：

while :

do

command

done

或者

while true

do

command

done

### 5. break 和 continue

和其他编程语言一样，表示跳出循环和跳出当前循环。

## 九、shell函数

### 1. Shell函数定义

一般格式：

[ function ] funname [()]

{

action;

[return int;]

}

中括号里的都是可选的，即可不写。

说明：

可以带function fun() 定义，也可以直接fun() 定义,不带任何参数；

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。

例子：

[root@localhost ~]# cat test.sh

#!/bin/bash

demoFun(){

echo "这是我的第一个 shell 函数!"

}

echo "-----函数开始执行-----"

demoFun

echo "-----函数执行完毕-----"

[root@localhost ~]# ./test.sh

-----函数开始执行-----

这是我的第一个 shell 函数!

-----函数执行完毕-----

### 2. shell函数参数

Shell函数的参数和shell脚本传参一样，通过 $n 的形式来获取参数的值，例如，$1表示第一个参数，$2表示第二个参数…

当n>=10时，需要使用${n}来获取参数。

## 十、shell重定向

### 1. 重定向介绍

大多数 UNIX 系统命令从你的终端接受输入并将所产生的输出发送回​​到您的终端。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出，默认情况下，这也是你的终端。

重定向命令列表如下：

|  |  |
| --- | --- |
| **命令** | **说明** |
| 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 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。 |

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。

### 2. 输出重定向

> 为新建文件，若目标文件已经存在会覆盖；

>> 为追加

例子：

[root@localhost ~]# who > users

[root@localhost ~]# cat users

root pts/0 2017-03-28 09:22 (10.65.2.243)

root pts/1 2017-03-28 10:14 (10.65.2.243)

[root@localhost ~]# who >> users

[root@localhost ~]# cat users

root pts/0 2017-03-28 09:22 (10.65.2.243)

root pts/1 2017-03-28 10:14 (10.65.2.243)

root pts/0 2017-03-28 09:22 (10.65.2.243)

root pts/1 2017-03-28 10:14 (10.65.2.243)

### 3. 输入重定向

例1：

[root@localhost ~]# wc -l < /etc/hosts

2

例2：

[root@localhost ~]# cat <<EOF > test.txt

Hello World!

> EOF

[root@localhost ~]# cat test.txt

Hello World!

### 4. /dev/null 文件

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 stderr，可以这样写：

command > /dev/null 2>&1

例子：

[root@localhost ~]# cat /etc/hosts /etc/host

127.0.0.1 localhost localhost.localdomain localhost4 localhost4.localdomain4

::1 localhost localhost.localdomain localhost6 localhost6.localdomain6

cat: /etc/host: No such file or directory

[root@localhost ~]# cat /etc/hosts /etc/host > /dev/null 2>&1

[root@localhost ~]#

## 十一、 文件包含

和其他语言一样，Shell 也可以包含外部脚本。这样可以很方便的封装一些公用的代码作为一个独立的文件。

Shell 文件包含的语法格式如下：

. filename或source filename

注意：点号(.)和文件名中间有一空格。

例子：

[root@localhost ~]# cat test1.sh

#!/bin/bash

your\_name="你的名字"

my\_name="我的名字"

echo $your\_name

[root@localhost ~]# cat test2.sh

#!/bin/bash

source test1.sh

echo $my\_name

[root@localhost ~]# ./test2.sh

你的名字

我的名字

## 十二、shell编程三剑客

### grep

通过man命令，我们可以查看到grep的基本用法如下：

NAME

grep, egrep, fgrep - print lines matching a pattern

SYNOPSIS

grep [OPTIONS] PATTERN [FILE...]

grep [OPTIONS] [-e PATTERN | -f FILE] [FILE...]

DESCRIPTION

grep searches the named input FILEs (or standard input if no files are named, or if a single yphen-minus (-) is given as file name) for lines containing a match to the given PATTERN. Bydefault, grep prints the matching lines.

其中[options]最常用的有如下几个：

-i：不区分大小写搜索

--color：关键字突出颜色显示

-n ：顺便输出行号

-c ：计算找到 '搜寻字符串' 的次数

-v:  显示没有被模式匹配到的行

-o：只显示被模式匹配到的字符串

-A N：打印匹配行之前的N行

-B N：打印匹配行之后的N行

-C N：打印匹配行前后的N行

grep 常常根据正则表达式去搜索匹配。

### sed

sed 是一种在线编辑器，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”（pattern space），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。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 [-nefr] [动作]

选项与参数：

-n ：使用安静(silent)模式。在一般 sed 的用法中，所有来自 STDIN 的数据一般都会被列出到终端上。但如果加上 -n 参数后，则只有经过sed 特殊处理的那一行(或者动作)才会被列出来。

-e ：直接在命令列模式上进行 sed 的动作编辑；

-f ：直接将 sed 的动作写在一个文件内， -f filename 则可以运行 filename 内的 sed 动作；

-r ：sed 的动作支持的是延伸型正规表示法的语法。(默认是基础正规表示法语法)

-i ：直接修改读取的文件内容，而不是输出到终端。

动作说明： [n1[,n2]]function

n1, n2 ：不见得会存在，一般代表『选择进行动作的行数』，举例来说，如果我的动作是需要在 10 到 20 行之间进行的，则『 10,20[动作行为] 』

function：

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，因为是删除啊，所以 d 后面通常不接任何咚咚；

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 ：列印，亦即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哩！通常这个 s 的动作可以搭配正规表示法！例如 1,20s/old/new/g 就是啦！

参考博客：http://www.cnblogs.com/ggjucheng/archive/2013/01/13/2856901.html

### awk

awk是一个强大的文本分析工具，相对于grep的查找，sed的编辑，awk在其对数据分析并生成报告时，显得尤为强大。简单来说awk就是把文件逐行的读入，以空格为默认分隔符将每行切片，切开的部分再进行各种分析处理。

awk工作流程是这样的：先执行BEGIN，然后读取文件，读入有/n换行符分割的一条记录，然后将记录按指定的域分隔符划分域，填充域，$0则表示所有域,$1表示第一个域,$n表示第n个域， 默认域分隔符是"空白键" 或 "[tab]键"，随后开始执行模式所对应的动作action。接着开始读入第二条记录······直到所有的记录都读完，最后执行END操作。

例子：

显示/etc/passwd的账户和账户对应的shell,而账户与shell之间以tab键分割：

#cat /etc/passwd |awk -F ':' '{print $1"\t"$7}'

root /bin/bash

daemon /bin/sh

bin /bin/sh

sys /bin/sh

awk有许多内置变量用来设置环境信息，这些变量可以被改变，下面给出了最常用的一些变量。

ARGC 命令行参数个数

ARGV 命令行参数排列

ENVIRON 支持队列中系统环境变量的使用

FILENAME awk浏览的文件名

FNR 浏览文件的记录数

FS 设置输入域分隔符，等价于命令行 -F选项

NF 浏览记录的域的个数

NR 已读的记录数

OFS 输出域分隔符

ORS 输出记录分隔符

RS 控制记录分隔符

有三种方式调用awk:

1.命令行方式

awk [-F field-separator] 'commands' input-file(s)

其中，commands 是真正awk命令，[-F域分隔符]是可选的。 input-file(s) 是待处理的文件。

在awk中，文件的每一行中，由域分隔符分开的每一项称为一个域。通常，在不指名-F域分隔符的情况下，默认的域分隔符是空格。

2.shell脚本方式

将所有的awk命令插入一个文件，并使awk程序可执行，然后awk命令解释器作为脚本的首行，一遍通过键入脚本名称来调用。

相当于shell脚本首行的：#!/bin/sh

可以换成：#!/bin/awk

3.将所有的awk命令插入一个单独文件，然后调用：

awk -f awk-script-file input-file(s)

其中，-f选项加载awk-script-file中的awk脚本，input-file(s)跟上面的是一样的。

更多详解参考博客：

http://www.cnblogs.com/losbyday/p/5854701.html

http://www.cnblogs.com/losbyday/p/5854707.html

<http://www.cnblogs.com/losbyday/p/5854725.html>