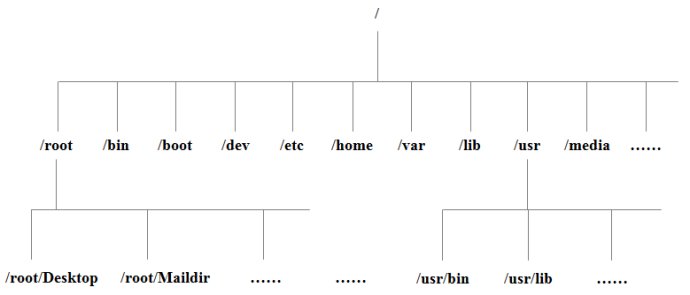
**Linux 系统目录结构**



以下是对这些目录的解释：

/bin：

bin是Binary的缩写, 这个目录存放着最经常使用的命令。

/boot：

这里存放的是启动Linux时使用的一些核心文件，包括一些连接文件以及镜像文件。

/dev ：

dev是Device(设备)的缩写, 该目录下存放的是Linux的外部设备，在Linux中访问设备的方式和访问文件的方式是相同的。

/etc：

这个目录用来存放所有的系统管理所需要的配置文件和子目录。

/home：

用户的主目录，在Linux中，每个用户都有一个自己的目录，一般该目录名是以用户的账号命名的。

/lib：

这个目录里存放着系统最基本的动态连接共享库，其作用类似于Windows里的DLL文件。几乎所有的应用程序都需要用到这些共享库。

/lost+found：

这个目录一般情况下是空的，当系统非法关机后，这里就存放了一些文件。

/media：

linux系统会自动识别一些设备，例如U盘、光驱等等，当识别后，linux会把识别的设备挂载到这个目录下。

/mnt：

系统提供该目录是为了让用户临时挂载别的文件系统的，我们可以将光驱挂载在/mnt/上，然后进入该目录就可以查看光驱里的内容了。

/opt：

这是给主机额外安装软件所摆放的目录。比如你安装一个ORACLE数据库则就可以放到这个目录下。默认是空的。

/proc：

这个目录是一个虚拟的目录，它是系统内存的映射，我们可以通过直接访问这个目录来获取系统信息。

这个目录的内容不在硬盘上而是在内存里，我们也可以直接修改里面的某些文件，比如可以通过下面的命令来屏蔽主机的ping命令，使别人无法ping你的机器：

echo 1 > /proc/sys/net/ipv4/icmp\_echo\_ignore\_all

/root：

该目录为系统管理员，也称作超级权限者的用户主目录。

/sbin：

s就是Super User的意思，这里存放的是系统管理员使用的系统管理程序。

/selinux：

这个目录是Redhat/CentOS所特有的目录，Selinux是一个安全机制，类似于windows的防火墙，但是这套机制比较复杂，这个目录就是存放selinux相关的文件的。

/srv：

该目录存放一些服务启动之后需要提取的数据。

/sys：

这是linux2.6内核的一个很大的变化。该目录下安装了2.6内核中新出现的一个文件系统 sysfs 。

sysfs文件系统集成了下面3种文件系统的信息：针对进程信息的proc文件系统、针对设备的devfs文件系统以及针对伪终端的devpts文件系统。

该文件系统是内核设备树的一个直观反映。

当一个内核对象被创建的时候，对应的文件和目录也在内核对象子系统中被创建。

/tmp：

这个目录是用来存放一些临时文件的。

/usr：

这是一个非常重要的目录，用户的很多应用程序和文件都放在这个目录下，类似与windows下的program files目录。

/usr/bin：

系统用户使用的应用程序。

/usr/sbin：

超级用户使用的比较高级的管理程序和系统守护程序。

/usr/src：内核源代码默认的放置目录。

/var：

这个目录中存放着在不断扩充着的东西，我们习惯将那些经常被修改的目录放在这个目录下。包括各种日志文件。

在linux系统中，有几个目录是比较重要的，平时需要注意不要误删除或者随意更改内部文件。

/etc： 上边也提到了，这个是系统中的配置文件，如果你更改了该目录下的某个文件可能会导致系统不能启动。

/bin, /sbin, /usr/bin, /usr/sbin: 这是系统预设的执行文件的放置目录，比如 ls 就是在/bin/ls 目录下的。

值得提出的是，/bin, /usr/bin 是给系统用户使用的指令（除root外的通用户），而/sbin, /usr/sbin 则是给root使用的指令。

/var： 这是一个非常重要的目录，系统上跑了很多程序，那么每个程序都会有相应的日志产生，而这些日志就被记录到这个目录下，具体在/var/log 目录下，另外mail的预设放置也是在这里。

* . ：代表当前的目录，也可以使用 ./ 来表示；
* .. ：代表上一层目录，也可以 ../ 来代表。

如果一个目录或文件名以一个点 . 开始，表示这个目录或文件是一个隐藏目录或文件(如：.bashrc)。即以默认方式查找时，不显示该目录或文件

* ls: 列出目录 list system
* -a ：全部的文件，连同隐藏档( 开头为 . 的文件) 一起列出来(常用)
* -d ：仅列出目录本身，而不是列出目录内的文件数据(常用)
* -l ：长数据串列出，包含文件的属性与权限等等数据；(常用)

将家目录下的所有文件列出来(含属性与隐藏档):ls -al

* cd：切换目录change directory

cd ~ 切换到当前用户的目录

* pwd：显示目前的目录Print Working Directory（显示工作目录）
* mkdir：创建一个新的目录

-m ：配置文件的权限喔！直接配置，不需要看默认权限 (umask) 的脸色～

-p ：帮助你直接将所需要的目录(包含上一级目录)递归创建起来！

mkdir -p test1/test2/test3/test4 创建递归目录

mkdir -m 711 test2 创建带权限的目录

* rmdir：删除一个空的目录

rmdir –p test 连同上一级的目录一起删除

rmdir -p test1/test2/test3/test4

* cp: 复制文件或目录

**d：**若来源档为连结档的属性(link file)，则复制连结档属性而非文件本身；

**-i：**若目标档(destination)已经存在时，在覆盖时会先询问动作的进行(常用)

**-l：**进行硬式连结(hard link)的连结档创建，而非复制文件本身；

-p：连同文件的属性一起复制过去，而非使用默认属性(备份常用)；

-r：递归持续复制，用於目录的复制行为；(常用)

* rm: 移除文件或目录

-f ：就是 force 的意思，忽略不存在的文件，不会出现警告信息；

-i ：互动模式，在删除前会询问使用者是否动作

-r ：递归删除啊！最常用在目录的删除了！这是非常危险的选项！！！

**mv (移动文件与目录，或修改名称)**

-f ：force 强制的意思，如果目标文件已经存在，不会询问而直接覆盖；

-i ：若目标文件 (destination) 已经存在时，就会询问是否覆盖！输入yes确认

-u ：若目标文件已经存在，且 source 比较新，才会升级 (update)

**Linux 文件内容查看**

cat 由第一行开始显示文件内容

tac 从最后一行开始显示，可以看出 tac 是 cat 的倒著写！

nl 显示的时候，顺道输出行号！

more 一页一页的显示文件内容

less 与 more 类似，但是比 more 更好的是，他可以往前翻页！

head 只看头几行

tail 只看尾巴几行

Linux 磁盘管理

* df：列出文件系统的整体磁盘使用量
* du：检查磁盘空间使用量
* fdisk：用于磁盘分区

df

-a ：列出所有的文件系统，包括系统特有的 /proc 等文件系统；

-k ：以 KBytes 的容量显示各文件系统；

-m ：以 MBytes 的容量显示各文件系统；

-h ：以人们较易阅读的 GBytes, MBytes, KBytes 等格式自行显示；

-H ：以 M=1000K 取代 M=1024K 的进位方式；

-T ：显示文件系统类型, 连同该 partition 的 filesystem 名称 (例如 ext3) 也列出；

-i ：不用硬盘容量，而以 inode 的数量来显示

>df 列出所有文件系统

>df –h 将容量结果以易读的容量格式显示出来

>df –aT 将系统内的所有特殊文件格式及名称都列出来

**du**

* -a ：列出所有的文件与目录容量，因为默认仅统计目录底下的文件量而已。

该命令会打印所有目录下文件，

* -h ：以人们较易读的容量格式 (G/M) 显示；
* -s ：列出总量而已，而不列出每个各别的目录占用容量；
* -S ：不包括子目录下的总计，与 -s 有点差别。
* -k ：以 KBytes 列出容量显示；
* -m ：以 MBytes 列出容量显示；

>du –sm /\* 检查根目录底下每个目录所占用的容量, 通配符 \* 来代表每个目录

Vim



**Shell 教程**

Shell 是一个用 C 语言编写的程序

Linux 的 Shell 种类众多，常见的有：

* Bourne Shell（/usr/bin/sh或/bin/sh）
* Bourne Again Shell（/bin/bash）
* C Shell（/usr/bin/csh）
* K Shell（/usr/bin/ksh）
* Shell for Root（/sbin/sh）

Bash 也是大多数Linux 系统默认的 Shell

#!/bin/bash  
echo "Hello World !"

#! 是一个约定的标记，它告诉系统这个脚本需要什么解释器来执行，即使用哪一种 Shell。

echo 命令用于向窗口输出文本。

Test.sh

扩展名为 sh（sh代表shell），扩展名并不影响脚本执行，见名知意就好，如果你用 php 写 shell 脚本，扩展名就用 php 好了

### 运行 Shell 脚本有两种方法：

1作为可执行程序运行shell

./test.sh

直接写 test.sh，linux 系统会去 PATH 里寻找有没有叫 test.sh 的，而只有 /bin, /sbin, /usr/bin，/usr/sbin 等在 PATH 里，你的当前目录通常不在 PATH 里，所以写成 test.sh 是会找不到命令的，要用 ./test.sh 告诉系统说，就在当前目录找

**2作为解释器参数**

这种方式运行的脚本，不需要在第一行指定解释器信息，写了也没用

**Shell 变量**

1定义变量时，变量名不加美元符号$

2变量名和等号之间不能有空格

3变量名的命名须遵循如下规则

* 首个字符必须为字母（a-z，A-Z）。
* 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（可用help命令查看保留关键字）

定义变量：your\_name="runoob.com"

除了显式地直接赋值，还可以用语句给变量赋值：for file in `ls /etc` 将/tec下所有文件名循环出来

**使用变量**

**your\_name="qinjx"**

**echo $your\_name**

**echo ${your\_name} 推荐使用的方式，在关键时候可以避免歧义**

已定义的变量，可以被重新定义

**只读变量**

使用 readonly 命令可以将变量定义为只读变量，只读变量的值不能被改变

**#!/bin/bash**

**myUrl="http://www.w3cschool.cc"**

**readonly myUrl**

**myUrl=**[**http://www.runoob.com**](http://www.runoob.com)

**删除变量**

unset variable\_name

unset 命令不能删除只读变量，也就是不能删除常量

**示例代码：**

**for skill in Ada Coffe Action Java; do**

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

**done**

**echo后面要有双引号，不然打印不出效果**

**变量类型**

**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。//shell内部定义的变量**

**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。//可以访问的系统变量**

**3) shell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 //包含内部定义变量和系统变量**

**Shell 字符串**

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）。

**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

**示例 在双引号内使用变量，也展示了字符串拼接的一种方法**

**your\_name="yangqiang"**

**greeting="hello, "$your\_name" !"**

**greeting\_1="hello, ${your\_name} !"**

**echo $greeting $greeting\_1**

**打印：hello,yangqiang ! hello,yangqiang !**

**获取字符串长度**

**string="abcd"**

**echo ${#string} #输出 4**

**提取子字符串**

**string="runoob is a great site"**

**echo ${string:1:4} # 输出 unoo 用两个冒号来处理**

**查找子字符串**

查找字符 "i 或 s" 的位置：

string="runoob is a great company"

echo `expr index "$string" is` # 输出 8

注意： 以上脚本中 "`" 是反引号，而不是单引号 "'"，不要看错了哦。就是Esc下面那个键

**Echo `expr index “$string” isb` 会输出8 尽管b没有匹配上,这是一个或的关系**

**Shell 数组**

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

用括号来表示数组，数组元素用"空格"符号分割开

**array\_name=(value0 value1 value2 value3)**

**或**

**array\_name[0]=value0**

**array\_name[1]=value1**

**array\_name[n]=valuen**

可以不使用连续的下标，而且下标的范围没有限制

**${数组名[下标]} 读取数组中单个item**

**echo ${array\_name[@]} 获取数组中所有元素//echo $array\_name[@] 这样是没效果的**

**获取数组的长度**

# 取得数组元素的个数

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

# 或者

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

# 取得数组单个元素的长度

lengthn=${#array\_name[n]}

**//@ \* 是获取所有元素,而单个下标就获取下标对应的元素**

**Echo ${array\_name} 这样只会显示array\_name第0个元素**

**Shell 注释**

**以"#"开头的行就是注释，会被解释器忽略。**

**sh里没有多行注释，只能每一行加一个#号**

**注释一段代码的方法;将这一段代码写成一个函数，将调用注释掉就相当于注释整段代码**

**Linux中的变量不需要声明类型,var=http://www.yqstudio.com 这也是合法的**

**Linux中截取字符串的八种方法; 其中\*表示通配符**

假设有变量 var=http://www.aaa.com/123.htm

1. # 号截取，删除左边字符，保留右边字符。

echo ${var#\*//}

其中 var 是变量名，# 号是运算符，\*// 表示从左边开始删除第一个(从左边数) // 号及左边的所有字符

即删除 http://

结果是 ：www.aaa.com/123.htm

**2. ## 号截取，删除左边字符，保留右边字符。**

**echo ${var##\*/}**

**##\*/ 表示从左边开始删除最后（最右边）一个 / 号及左边的所有字符**

**即删除 http://www.aaa.com/**

**结果是 123.htm**

**3. %号截取，删除右边字符，保留左边字符**

**echo ${var%/\*}**

**%/\* 表示从右边开始，删除第一个（从右边数）/ 号及右边的字符**

**结果是：http://www.aaa.com**

**4. %% 号截取，删除右边字符，保留左边字符**

**echo ${var%%/\*}**

**%%/\* 表示从右边开始，删除最后（从右边数）一个 / 号及右边的字符**

**结果是：http:**

**前提：注：（左边的第一个字符是用 0 表示，右边的第一个字符用 0-1 表示）**

**0-7就表示从右边开始第7个字符**

**5. 从左边第几个字符开始，及字符的个数**

echo ${var:0:5}

其中的 0 表示左边第一个字符开始，5 表示字符的总个数。

结果是：http:

**6. 从左边第几个字符开始，一直到结束。**

echo ${var:7}

其中的 7 表示左边第8个字符开始，一直到结束。

结果是 ：www.aaa.com/123.htm

**7. 从右边第几个字符开始，及字符的个数，这个截取的方向还是向右**

echo ${var:0-7:3}

其中的 0-7 表示右边算起第七个字符开始，3 表示字符的个数。

结果是：123

**8. 从右边第几个字符开始，一直到结束。这个截取的方向还是向右**

echo ${var:0-7}

表示从右边第七个字符开始，一直到结束。

结果是：123.htm

**Shell 传递参数**

我们可以在执行 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……

**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";**

**执行文件**

$ chmod +x test.sh

$ ./test.sh 1 2 3 #参数之间用空格隔开

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

**如果执行的时候;>./test.sh 1,2,3会怎样**

第一个参数为：1,2,3

第二个参数为：

第三个参数为：

**参数不光可以传递数字，还可以传递字符串**

**>./test.sh 1 two ‘three’ #也是可以的**

**>./test.sh “one” ‘two’ three**

**>./test.sh one two three**

**打印的效果都是一样的**

|  |  |
| --- | --- |
| **参数处理** | **说明** |
| **$#** | **传递到脚本的参数个数** |
| **$\*** | **以一个单字符串显示所有向脚本传递的参数。 如"$\*"用「"」括起来的情况、以"$1 $2 … $n"的形式输出所有参数。** |
| **$$** | **脚本运行的当前进程ID号** |
| **$!** | **后台运行的最后一个进程的ID号** |
| **$@** | **与$\*相同，但是使用时加引号，并在引号中返回每个参数。 如"$@"用「"」括起来的情况、以"$1" "$2" … "$n" 的形式输出所有参数。** |
| **$-** | **显示Shell使用的当前选项，与**[**set命令**](http://www.runoob.com/linux/linux-comm-set.html)**功能相同。** |
| **$?** | **显示最后命令的退出状态。0表示没有错误，其他任何值表明有错误。** |

**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）**

echo "shell 传递参数演示";

echo "执行的文件名:$0";

echo "第一个参数:$1";

echo "第二个参数:$2";

echo "第三个参数:$3";

echo "传递参数的个数:$#"

echo "传递的参数作为一个字符串显示:$\*"

echo "传递的参数作为一个字符串显示2::['$\*']"

echo "当前脚本运行的进程号:$$"

echo "后台运行的最后一个进程的ID号:$!"

echo "传递的参数作为字符串显示:$@"

echo "传递的参数作为字符串显示2:['$@']"

echo "显示shell使用的当前选项$-"

echo "显示最后命令的退出状态,0表示没有错误，其他值表示有错:$?"

echo "演示$\*和$@的区别"

echo "-----\$\*----"

for i in "$\*";do

echo $i

done

echo "---\$@---"

for i in "$@";do

echo $i

done

**开始执行**

root@yq-pc:/opt/workspace# ./param.sh 'one' "two" three

shell 传递参数演示

执行的文件名:./param.sh

第一个参数:one

第二个参数:two

第三个参数:three

传递参数的个数:3

传递的参数作为一个字符串显示:one two three

传递的参数作为一个字符串显示2::['one two three']

当前脚本运行的进程号:12999

后台运行的最后一个进程的ID号:

传递的参数作为字符串显示:one two three

传递的参数作为字符串显示2:['one two three']

显示shell使用的当前选项hB

显示最后命令的退出状态,0表示没有错误，其他值表示有错:0

演示one two three和one two three的区别

-----$\*----

one two three

---$@---

one

two

three

**Shell 数组**

数组中可以存放多个值。Bash Shell 只支持一维数组，初始时不要指定数组的长度，元素用空格隔开

**Shell 基本运算符**

**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，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，例如 2+2 是不对的，必须写成 2 + 2，这与我们熟悉的大多数编程语言不一样。

完整的表达式要被 ` ` 包含，注意这个字符不是常用的单引号，在 Esc 键下边

val=`expr 2 + 2`

echo "两数之和为 : $val"

|  |  |  |
| --- | --- | --- |
| **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。** |

**条件表达式要放在方括号之间，并且要有空格，例如: [$a==$b] 是错误的，必须写成 [ $a == $b ] #[和$a之间，$b和]之间有空格**

**if [ $a == $b ]**

**then**

**echo "a 等于 b"**

**fi**

**if [ $a != $b ]**

**then**

**echo "a 不等于 b"**

**fi**

**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。

**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：**

**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。

**if [ $a -lt $b ] #[ $a -lt $b ] 注意空格**

**then**

**echo "$a -lt $b: a 小于 b"**

**else**

**echo "$a -lt $b: a 不小于 b"**

**fi**

**布尔运算符**

**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。*

**a=10**

**b=20**

**if [ $a != $b ]**

**then**

**echo "$a != $b : a 不等于 b"**

**else**

**echo "$a != $b: a 等于 b"**

**fi**

**if [ $a -lt 100 -a $b -gt 15 ]**

**then**

**echo "$a -lt 100 -a $b -gt 15 : 返回 true"**

**else**

**echo "$a -lt 100 -a $b -gt 15 : 返回 false"**

**fi**

**if [ $a -lt 100 -o $b -gt 100 ]**

**then**

**echo "$a -lt 100 -o $b -gt 100 : 返回 true"**

**else**

**echo "$a -lt 100 -o $b -gt 100 : 返回 false"**

**fi**

**逻辑运算符**

**&& 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**

a=10

b=20

if [[ $a -lt 100 && $b -gt 100 ]]

then

echo "返回 true"

else

echo "返回 false"

fi

if [[ $a -lt 100 || $b -gt 100 ]]

then

echo "返回 true"

else

echo "返回 false"

fi

**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。** |

**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。

**file="/var/www/runoob/test.sh"**

**if [ -r $file ]**

**then**

**echo "文件可读"**

**else**

**echo "文件不可读"**

**fi**

**if [ -w $file ]**

**then**

**echo "文件可写"**

**else**

**echo "文件不可写"**

**fi**

**if [ -d $file ]**

**then**

**echo "文件是个目录"**

**else**

**echo "文件不是个目录"**

**fi**

**Shell echo命令**

**.显示结果定向至文件**

**echo "It is a test" > myfile**

**echo “yq” > myfiel.txt**

**显示命令执行结果**

**>echo `date`**

**2017年 05月 23日 星期二 15:43:22 CST**

# Shell printf 命令

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

**test命令可以检测真假，应用于数值，字符串，文件**

**Shell 流程控制**

如果else分支没有语句执行，就不要写这个else

a=10

b=20

if [ $a == $b ]

then

echo "a 等于 b"

elif [ $a -gt $b ]

then

echo "a 大于 b"

elif [ $a -lt $b ]

then

echo "a 小于 b"

else

echo "没有符合的条件"

fi

if else语句经常与test命令结合使用，如下所示：

num1=$[2\*3]

num2=$[1+5]

if test $[num1] -eq $[num2]

then

echo '两个数字相等!'

else

echo '两个数字不相等!'

fi

**最后显示：两个数字相等**

**for 循环**

for var in item1 item2 ... itemN

do

command1

command2

...

commandN

done

for loop in 1 2 3 4 5

do

echo "The value is: $loop"

done

**while 语句**

int=1

while(( $int<=5 ))

do

echo $int

let "int++"

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

echo '按下 <CTRL-D> 退出'

echo -n '输入你最喜欢的电影名: '

while read FILM

do

echo "是的！$FILM 是一部好电影"

done

将输入的信息存储在变量FILM中，而且是循环读取，也就是说会一直等待键盘输入

无限循环

while true

do

command

done

或者

for (( ; ; ))

演示 vim while.sh

While true

Do

Echo “套路才是王道”

Done

>chmod u+x,g+x,o+x while.sh

>./while.sh > while.log

程序执行的结果是一直将”套路才是王道” 输出到while.log文件中

until 循环

until循环执行一系列命令直至条件为真时停止

until循环与while循环在处理方式上刚好相反。

一般while循环优于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

// vim until.sh

X=5

Until (( x>10000 ))

Do

Echo “x=:$x”

Let “x++”

Done

>chmod u+x,g+x,o+x until.sh

>./until.sh

程序会一直执行，知道条件为真才停止

Shell case语句为多选择语句

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

匹配模式不一定是数字

>vim case.sh

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'

;; //一个语句块的结束标志

2) echo '你选择了 2'

;; //一个语句块的结束标志

3) echo '你选择了 3'

;; //一个语句块的结束标志

4) echo '你选择了 4'

;; //一个语句块的结束标志

\*) echo '你没有输入 1 到 4 之间的数字'

;;

Esac //esac（就是case反过来）作为结束标记

执行程序

>chmod u+x,g+x,o+x case.sh

>./case.sh

输入 1 到 4 之间的数字:

你输入的数字为:

3

你选择了 3

Shell 里面也有break continue等结构 和C语言一样

while :

do

echo -n "输入 1 到 5 之间的数字: "

read aNum

case $aNum in

1|2|3|4|5) echo "你输入的数字为 $aNum!"

;;

\*) echo "你输入的数字不是 1 到 5 之间的!"

continue

echo "游戏结束"

;;

esac

done

**Shell 函数**

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n(0-255

funWithReturn(){

echo "这个函数会对输入的两个数字进行相加运算..."

echo "输入第一个数字: "

read aNum

echo "输入第二个数字: "

read anotherNum

echo "两个数字分别为 $aNum 和 $anotherNum !"

return $(($aNum+$anotherNum)) 括号前面的$必须加

}

funWithReturn

echo "输入的两个数字之和为 $? !"

函数返回值在调用该函数后通过 $? 来获得。

注意：所有函数在使用前必须定义。这意味着必须将函数放在脚本开始部分，直至shell解释器首次发现它时，才可以使用。调用函数仅使用其函数名即可

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 $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...

funWithParam(){

echo "第一个参数为 $1 !"

echo "第二个参数为 $2 !"

echo "第十个参数为 $10 !"

echo "第十个参数为 ${10} !"

echo "第十一个参数为 ${11} !"

echo "参数总数有 $# 个!"

echo "作为一个字符串输出所有参数 $\* !"

}

funWithParam 1 2 3 4 5 6 7 8 9 34 73

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

另外，还有几个特殊字符用来处理参数：这和通过命令行传递参数的方式有些类似

参数处理 说明

$# 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号

$! 后台运行的最后一个进程的ID号

$@ 与$\*相同，但是使用时加引号，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。

$? 显示最后命令的退出状态。0表示没有错误，其他任何值表明有错误。

命令行传递参数:> test.sh 1 2 3 4 5 6 7 8 9 #这是在命令行

函数传递参数:>func 1 2 3 4 5 6 7 8 9 #这是在脚本内部

**Shell 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

通常标准输入就是键盘；标准输出就是显示器

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

*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）*

**以追加的方式和不以追加的方式的区别：在操作之前是把文件清空还是在尾部添加**

**$ who > users**

**$ wc -l users**

**2 users**

**或**

**$ wc -l < users**

**2**

**注意两者的区别：一个显示输入源一个不显示输入源**

**同时替换输入和输出**

**command1 < infile > outfile #执行顺序是从左向右**

**执行command1，从文件infile读取内容，然后将输出写入到outfile中**

**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 重定向到 file

**>command 2 > out**

**将命令标准错误输出到out**

**>command 2 >> out**

**将标准错误输出到out**

**如果希望将 stdout 和 stderr 合并后重定向到 file，可以这样写**

**$ command > file 2>&1**

# Shell 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**

**Test1.sh 内容如下**

**url=”http:www.yqstudio.cn”**

**test2.sh内容如下**

**. ./test1.sh 或者source ./test1.sh**

**Echo “个人工作室:$url”**

**>chmod +x test2.sh**

**>./test2.sh**

**被包含的文件test1.sh不需要可执行权限**

**总结：**

**在Linux中 \* 是通配符**

**示例：当前文件夹下有很多shell脚本，还有文件夹 电子文档，现在想将shell文件全部移动到一个文件夹**

**>cp \*.sh linux\_shell**