内容目录

1.Linux 简介及基础知识

1.1 Linux 简介

Linux 内核最初只是由芬兰人李纳斯·托瓦兹（Linus Torvalds）在赫尔辛基大学上学时出于个人爱好而编写的。
Linux 是一套免费使用和自由传播的类 Unix 操作系统，是一个基于 POSIX 和 UNIX 的多用户、多任务、支持多线程和多 CPU 的操作系统。
Linux 能运行主要的 UNIX 工具软件、应用程序和网络协议。它支持 32 位和 64 位硬件。Linux 继承了 Unix 以网络为核心的设计思想，是一个性能稳定的多用户网络操作系统。

1.11 Linux 的发行版

Linux 的发行版说简单点就是将 Linux 内核与应用软件做一个打包。
目前市面上较知名的发行版有：Ubuntu、RedHat、CentOS、Debian、Fedora、SuSE、OpenSUSE、Arch Linux、SolusOS 等。

1.12 Linux 应用领域

今天各种场合都有使用各种 Linux 发行版，从嵌入式设备到超级计算机，并且在服务器领域确定了地位，通常服务器使用 LAMP（Linux + Apache + MySQL + PHP）或 LNMP（Linux + Nginx+ MySQL + PHP）组合。
目前 Linux 不仅在家庭与企业中使用，并且在政府中也很受欢迎。
•巴西联邦政府由于支持 Linux 而世界闻名。
•有新闻报道俄罗斯军队自己制造的 Linux 发布版的，做为 G.H.ost 项目已经取得成果.
•印度的 Kerala 联邦计划在向全联邦的高中推广使用 Linux。
•中华人民共和国为取得技术独立，在龙芯过程中排他性地使用 Linux。
•在西班牙的一些地区开发了自己的 Linux 发布版，并且在政府与教育领域广泛使用，如 Extremadura 地区的 gnuLinEx 和 Andalusia 地区的 Guadalinex。
•葡萄牙同样使用自己的 Linux 发布版 Caixa Mágica，用于 Magalh?es 笔记本电脑和 e-escola 政府软件。
•法国和德国同样开始逐步采用 Linux。

1.13 Linux vs Windows

目前国内 Linux 更多的是应用于服务器上，而桌面操作系统更多使用的是 Windows。主要区别如下

比较	Windows	Linux
界面	界面统一，外壳程序固定所有 Windows 程序菜单几乎一致，快捷键也几乎相同	图形界面风格依发布版不同而不同，可能互不兼容。GNU/Linux 的终端机是从 UNIX 传承下来，基本命令和操作方法也几乎一致。
驱动程序	驱动程序丰富，版本更新频繁。默认安装程序里面一般包含有该版本发布时流行的硬件驱动程序，之后所出的新硬件驱动依赖于硬件厂商提供。对于一些老硬件，如果没有了原配的驱动有时很难支持。另外，有时硬件厂商未提供所需版本的 Windows 下的驱动，也会比较头痛。	由志愿者开发，由 Linux 核心开发小组发布，很多硬件厂商基于版权考虑并未提供驱动程序，尽管多数无需手动安装，但是涉及安装则相对复杂，使得新用户面对驱动程序问题（是否存在和安装方法）会一筹莫展。但是在开源开发模式下，许多老硬件尽管在 Windows 下很难支持的也容易找到驱动。HP、Intel、AMD 等硬件厂商逐步不同程度支持开源驱动，问题正在得到缓解。
使用	使用比较简单，容易入门。图形化界面对没有计算机背景知识的用户使用十分有利。	图形界面使用简单，容易入门。文字界面，需要学习才能掌握。

学习	系统构造复杂、变化频繁，且知识、技能淘汰快，深入学习困难。	系统构造简单、稳定，且知识、技能传承性好，深入学习相对容易。
软件	每一种特定功能可能都需要商业软件的支持，需要购买相应的授权。	大部分软件都可以自由获取，同样功能的软件选择较少。

1.2 Linux 系统启动过程

linux 启动时我们会看到许多启动信息。

Linux 系统的启动过程并不是大家想象中的那么复杂，其过程可以分为 5 个阶段：

•内核的引导。
•运行 init。
•系统初始化。
•建立终端 。
•用户登录系统。

init 程序的类型：

- SysV: init, CentOS 5 之前, 配置文件： /etc/inittab。
- Upstart: init,CentOS 6, 配置文件： /etc/inittab, /etc/init/*.conf。
- Systemd： systemd, CentOS 7,配置文件： /usr/lib/systemd/system、/etc/systemd/system。

1.21 内核引导

当计算机打开电源后，首先是 BIOS 开机自检，按照 BIOS 中设置的启动设备（通常是硬盘）来启动。

操作系统接管硬件以后，首先读入 /boot 目录下的内核文件。

1.22 运行 init

init 进程是系统所有进程的起点，你可以把它比拟成系统所有进程的老祖宗，没有这个进程，系统中任何进程都不会启动。

init 程序首先是需要读取配置文件 /etc/inittab。

1.23 运行级别

许多程序需要开机启动。它们在 Windows 叫做"服务"（service），在 Linux 就叫做"守护进程"（daemon）。init 进程的一大任务，就是去运行这些开机启动的程序。

但是，不同的场合需要启动不同的程序，比如用作服务器时，需要启动 Apache，用作桌面就不需要。

Linux 允许为不同的场合，分配不同的开机启动程序，这就叫做"运行级别"（runlevel）。也就是说，启动时根据"运行级别"，确定要运行哪些程序。

Linux 系统有 7 个运行级别(runlevel)：

- 运行级别 0：系统停机状态，系统默认运行级别不能设为 0，否则不能正常启动
- 运行级别 1：单用户工作状态，root 权限，用于系统维护，禁止远程登陆
- 运行级别 2：多用户状态(没有 NFS)
- 运行级别 3：完全的多用户状态(有 NFS)，登陆后进入控制台命令行模式
- 运行级别 4：系统未使用，保留
- 运行级别 5：X11 控制台，登陆后进入图形 GUI 模式
- 运行级别 6：系统正常关闭并重启，默认运行级别不能设为 6，否则不能正常启动

1.24 系统初始化

在 init 的配置文件中有这么一行： si::sysinit:/etc/rc.d/rc.sysinit　它调用执行了/etc/rc.d/rc.sysinit，而 rc.sysinit 是一个 bash shell 的脚本，它主要是完成一些系统初始化的工作，rc.sysinit 是每一个运行级别都要首先运行的重要脚本。

它主要完成的工作有：激活交换分区，检查磁盘，加载硬件模块以及其它一些需要优先执行任务。

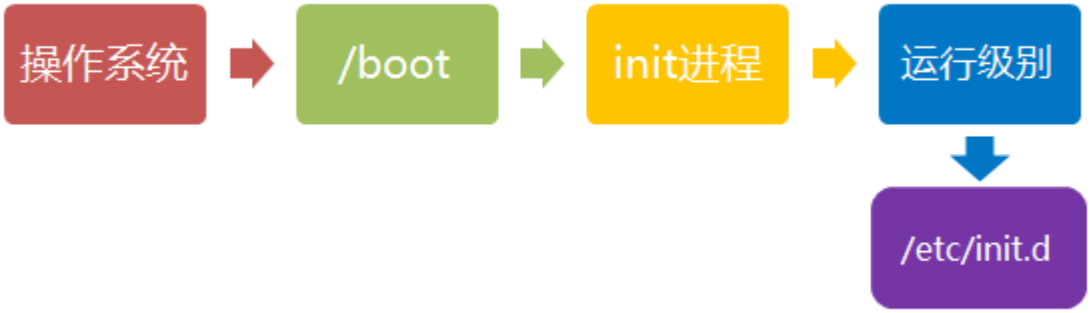
```
l5:5:wait:/etc/rc.d/rc 5
```

这一行表示以 5 为参数运行/etc/rc.d/rc，/etc/rc.d/rc 是一个 Shell 脚本，它接受 5 作为参数，去执行/etc/rc.d/rc5.d/目录下的所有的 rc 启动脚本，/etc/rc.d/rc5.d/目录中的这些启动脚本实际上都是一些连接文件，而不是真正的 rc 启动脚本，真正的 rc 启动脚本实际上都是放在/etc/rc.d/init.d/目录下。

而这些 rc 启动脚本有着类似的用法，它们一般能接受 start、stop、restart、status 等参数。

/etc/rc.d/rc5.d/中的 rc 启动脚本通常是 K 或 S 开头的连接文件，对于以 S 开头的启动脚本，将以 start 参数来运行。

而如果发现存在相应的脚本也存在 K 打头的连接，而且已经处于运行态了(以/var/lock/subsys/下的文件作为标志)，则将首先以 stop 为参数停止这些已经启动了的守护进程，然后再重新运行。
这样做是为了保证是当 init 改变运行级别时，所有相关的守护进程都将重启。
至于在每个运行级中将运行哪些守护进程，用户可以通过 chkconfig 或 setup 中的"System Services"来自行设定。



1.25 建立终端

rc 执行完毕后，返回 init。这时基本系统环境已经设置好了，各种守护进程也已经启动了。
init 接下来会打开 6 个终端，以便用户登录系统。在 inittab 中的以下 6 行就是定义了 6 个终端：

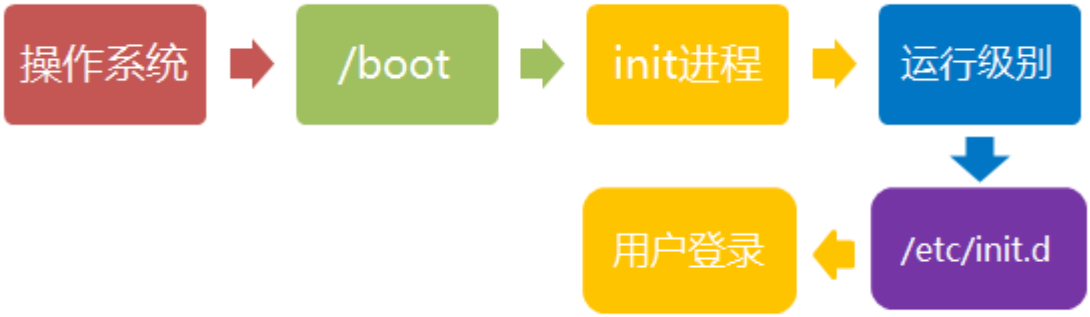
```
1:2345:respawn:/sbin/mingetty tty1
2:2345:respawn:/sbin/mingetty tty2
3:2345:respawn:/sbin/mingetty tty3
4:2345:respawn:/sbin/mingetty tty4
5:2345:respawn:/sbin/mingetty tty5
6:2345:respawn:/sbin/mingetty tty6
```

从上面可以看出在 2、3、4、5 的运行级别中都将以 respawn 方式运行 mingetty 程序，mingetty 程序能打开终端、设置模式。
同时它会显示一个文本登录界面，这个界面就是我们经常看到的登录界面，在这个登录界面中会提示用户输入用户名，而用户输入的用户将作为参数传给 login 程序来验证用户的身份。

1.26 用户登录系统

一般来说，用户的登录方式有三种：

- （1）命令行登录
- （2）ssh 登录
- （3）图形界面登录



对于运行级别为 5 的图形方式用户来说，他们的登录是通过一个图形化的登录界面。登录成功后可以直接进入 KDE、Gnome 等窗口管理器。

而本文主要讲的还是文本方式登录的情况：当我们看到 mingetty 的登录界面时，我们就可以输入用户名和密码来登录系统了。

Linux 的账号验证程序是 login，login 会接收 mingetty 传来的用户名作为用户名参数。

然后 login 会对用户名进行分析：如果用户名不是 root，且存在 /etc/nologin 文件，login 将输出 nologin 文件的内容，然后退出。

这通常用来系统维护时防止非 root 用户登录。只有/etc/securetty 中登记了的终端才允许 root 用户登录，如果不存在这个文件，则 root 用户可以在任何终端上登录。

/etc/usertty 文件用于对用户作出附加访问限制，如果不存在这个文件，则没有其他限制。

1.27 图形模式与文字模式的切换方式

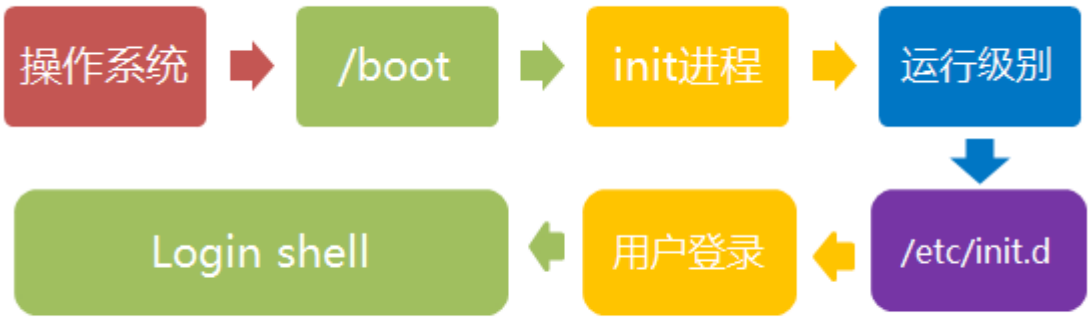
Linux 预设提供了六个命令窗口终端机让我们来登录。

默认我们登录的就是第一个窗口，也就是 tty1，这个六个窗口分别为 tty1,tty2 ... tty6，你可以按下 Ctrl + Alt + F1 ~ F6 来切换它们。

如果你安装了图形界面，默认情况下是进入图形界面的，此时你就可以按 Ctrl + Alt + F1 ~ F6 来进入其中一个命令窗口界面。

当你进入命令窗口界面后再返回图形界面只要按下 Ctrl + Alt + F7 就回来了。

如果你用的 vmware 虚拟机，命令窗口切换的快捷键为 Alt + Space + F1~F6. 如果你在图形界面下请按 Alt + Shift + Ctrl + F1~F6 切换至命令窗口。



1.28 Linux 关机

在 linux 领域内大多用在服务器上，很少遇到关机的操作。毕竟服务器上跑一个服务是永无止境的，除非特殊情况下，不得已才会关机。

正确的关机流程为：sync > shutdown > reboot > halt

关机指令为：shutdown ，你可以 man shutdown 来看一下帮助文档。

例如你可以运行如下命令关机：

```
sync 将数据由内存同步到硬盘中。

shutdown 关机指令，你可以 man shutdown 来看一下帮助文档。例如你可以运行如下命令关机：

shutdown –h 10 ‘This server will shutdown after 10 mins’ 这个命令告诉大家，计算机将在 10 分钟后关机，并且会显示在登陆用户的当前屏幕中。

Shutdown –h now 立马关机

Shutdown –h 20:25 系统会在今天 20:25 关机

Shutdown –h +10 十分钟后关机

Shutdown –r now 系统立马重启

Shutdown –r +10 系统十分钟后重启

reboot 就是重启，等同于 shutdown –r now

halt 关闭系统，等同于 shutdown –h now 和 poweroff
```

最后总结一下，不管是重启系统还是关闭系统，首先要运行 **sync** 命令，把内存中的数据写到磁盘中。

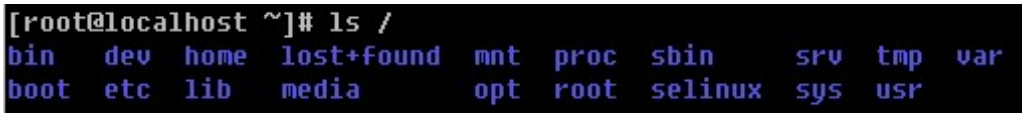
关机的命令有 **shutdown –h now halt poweroff** 和 **init 0** , 重启系统的命令有 **shutdown –r now reboot init 6**。

1.3 Linux 系统目录结构

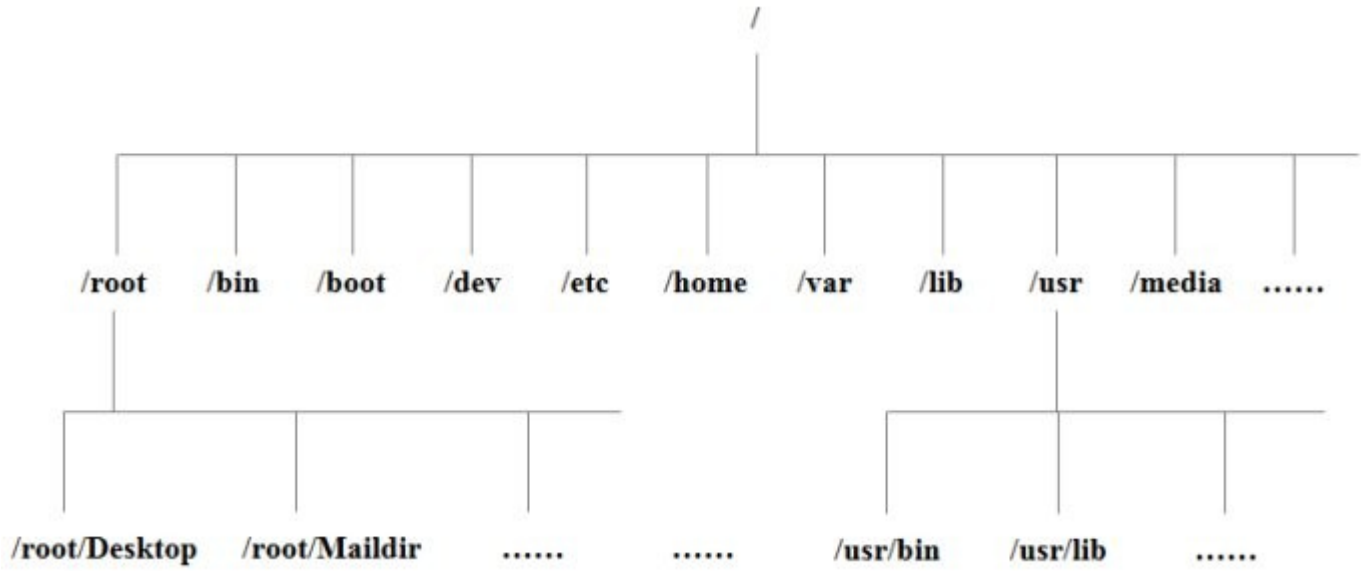
登录系统后，在当前命令窗口下输入命令：

```
ls /
```

你会看到如下图所示:



树状目录结构：



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，类似于 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。

1.4 Linux 文件基本属性

Linux 系统是一种典型的多用户系统，不同的用户处于不同的地位，拥有不同的权限。为了保护系统的安全性，Linux 系统对不同的用户访问同一文件（包括目录文件）的权限做了不同的规定。
在 Linux 中我们可以使用 ll 或者 ls –l 命令来显示一个文件的属性以及文件所属的用户和组，如：

```
[root@www /]# ls -l
total 64
dr-xr-xr-x   2 root root 4096 Dec 14  2012 bin
```

```
dr-xr-xr-x  4 root root 4096 Apr 19  2012 boot
......
```

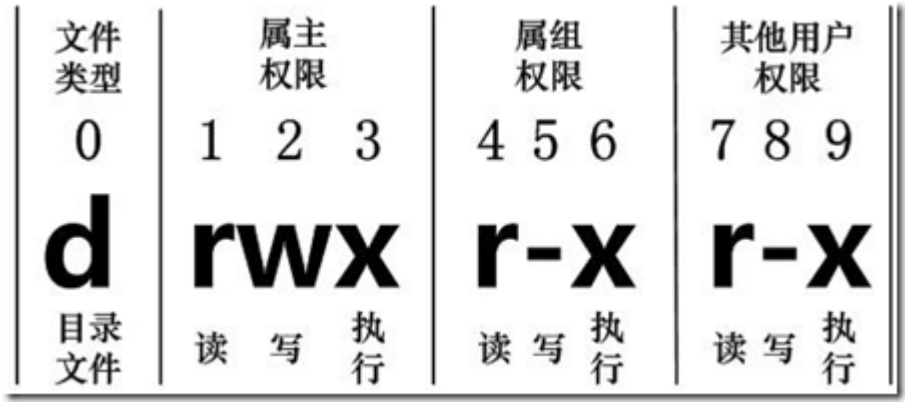
实例中，bin 文件的第一个属性用"d"表示。"d"在 Linux 中代表该文件是一个目录文件。

在 Linux 中第一个字符代表这个文件是目录、文件或链接文件等等。

- 当为[d]则是目录
- 当为[-]则是文件；
- 若是[l]则表示为链接文档(link file)；
- 若是[b]则表示为装置文件里面的可供储存的接口设备(可随机存取装置)；
- 若是[c]则表示为装置文件里面的串行端口设备，例如键盘、鼠标(一次性读取装置)。

接下来的字符中，以三个为一组，且均为『rwx』 的三个参数的组合。其中，[r]代表可读(read)、[w]代表可写(write)、[x]代表可执行(execute)。 要注意的是，这三个权限的位置不会改变，如果没有权限，就会出现减号[-]而已。

每个文件的属性由左边第一部分的 10 个字符来确定（如下图）。



从左至右用 0-9 这些数字来表示。

第 0 位确定文件类型，第 1-3 位确定属主（该文件的所有者）拥有该文件的权限。

第 4-6 位确定属组（所有者的同组用户）拥有该文件的权限，第 7-9 位确定其他用户拥有该文件的权限。

其中，第 1、4、7 位表示读权限，如果用"r"字符表示，则有读权限，如果用"-"字符表示，则没有读权限；

第 2、5、8 位表示写权限，如果用"w"字符表示，则有写权限，如果用"-"字符表示没有写权限；第 3、6、9 位表示可执行权限，如果用"x"字符表示，则有执行权限，如果用"-"字符表示，则没有执行权限。

1.41 Linux 文件属主和属组

```
[root@www /]# ls -l
total 64
drwxr-xr-x 2 root  root  4096 Feb 15 14:46 cron
drwxr-xr-x 3 mysql mysql 4096 Apr 21  2014 mysql
......
```

对于文件来说，它都有一个特定的所有者，也就是对该文件具有所有权的用户。

同时，在 Linux 系统中，用户是按组分类的，一个用户属于一个或多个组。

文件所有者以外的用户又可以分为文件所有者的同组用户和其他用户。

因此，Linux 系统按文件所有者、文件所有者同组用户和其他用户来规定了不同的文件访问权限。
在以上实例中，mysql 文件是一个目录文件，属主和属组都为 mysql，属主有可读、可写、可执行的权限；与属主同组的其他用户有可读和可执行的权限；其他用户也有可读和可执行的权限。
对于 root 用户来说，一般情况下，文件的权限对其不起作用。

1.42 更改文件属性

1、chgrp：更改文件属组

语法：

```
chgrp [-R] 属组名 文件名
```

参数选项

•-R：递归更改文件属组，就是在更改某个目录文件的属组时，如果加上-R 的参数，那么该目录下的所有文件的属组都会更改。

2、chown：更改文件属主，也可以同时更改文件属组

语法：

```
chown [–R] 属主名 文件名
chown [-R] 属主名：属组名 文件名
```

进入 /root 目录（~）将 install.log 的拥有者改为 bin 这个账号：

```
[root@www ~] cd ~
[root@www ~]# chown bin install.log
[root@www ~]# ls -l
-rw-r--r--  1 bin  users 68495 Jun 25 08:53 install.log
```

将 install.log 的拥有者与群组改回为 root：

```
[root@www ~]# chown root:root install.log
[root@www ~]# ls -l
-rw-r--r--  1 root root 68495 Jun 25 08:53 install.log
```

3、chmod：更改文件 9 个属性

Linux 文件属性有两种设置方法，一种是数字，一种是符号。
Linux 文件的基本权限就有九个，分别是 owner/group/others 三种身份各有自己的 read/write/execute 权限。
先复习一下刚刚上面提到的数据：文件的权限字符为：『-rwxrwxrwx』， 这九个权限是三个三个一组的！其中，我们可以使用数字来代表各个权限，各权限的分数对照表如下：

•r:4
•w:2
•x:1

每种身份(owner/group/others)各自的三个权限(r/w/x)分数是需要累加的，例如当权限为： [-rwxrwx---] 分数则是：

•owner = rwx = 4+2+1 = 7
•group = rwx = 4+2+1 = 7

•others= --- = 0+0+0 = 0

所以等一下我们设定权限的变更时，该文件的权限数字就是 770 啦！变更权限的指令 chmod 的语法是这样的：

```
chmod [-R] xyz 文件或目录
```

选项与参数：

•xyz : 就是刚刚提到的数字类型的权限属性，为 rwx 属性数值的相加。
•-R : 进行递归(recursive)的持续变更，亦即连同次目录下的所有文件都会变更

举例来说，如果要将.bashrc 这个文件所有的权限都设定启用，那么命令如下：

```
[root@www ~]# ls -al .bashrc
-rw-r--r--  1 root root 395 Jul  4 11:45 .bashrc
[root@www ~]# chmod 777 .bashrc
[root@www ~]# ls -al .bashrc
-rwxrwxrwx  1 root root 395 Jul  4 11:45 .bashrc
```

那如果要将权限变成 -rwxr-xr-- 呢？那么权限的分数就成为 [4+2+1][4+0+1][4+0+0]=754。

符号类型改变文件权限

还有一个改变权限的方法呦！从之前的介绍中我们可以发现，基本上就九个权限分别是(1)user (2)group (3)others 三种身份啦！ 那么我们就可以藉由 u, g, o 来代表三种身份的权限！

此外， a 则代表 all 亦即全部的身份！那么读写的权限就可以写成 r, w, x！也就是可以使用底下的方式来看：

```
        u       +(加入)   r
chmod   g       -(除去)   w   文件或目录
        o       =(设定)   x
        a
```

如果我们需要将文件权限设置为 -rwxr-xr-- ，可以使用 **chmod u=rwx,g=rx,o=r 文件名** 来设定:

```
#  touch test1    // 创建 test1 文件
# ls -al test1    // 查看 test1 默认权限
-rw-r--r-- 1 root root 0 Nov 15 10:32 test1
# chmod u=rwx,g=rx,o=r  test1    // 修改 test1 权限
# ls -al test1
-rwxr-xr-- 1 root root 0 Nov 15 10:32 test1
```

而如果是要将权限去掉而不改变其他已存在的权限呢？例如要拿掉全部人的可执行权限，则：

```
#  chmod  a-x test1
# ls -al test1
-rw-r--r-- 1 root root 0 Nov 15 10:32 test1
```

1.5 Linux 文件与目录管理

我们知道 Linux 的目录结构为树状结构，最顶级的目录为根目录 /。

其他目录通过挂载可以将它们添加到树中，通过解除挂载可以移除它们。

在开始本教程前我们需要先知道什么是绝对路径与相对路径。

•绝对路径：
路径的写法，由根目录 / 写起，例如： /usr/share/doc 这个目录。
•相对路径：
路径的写法，不是由 / 写起，例如由 /usr/share/doc 要到 /usr/share/man 底下时，可以写成： cd ../man 这就是相对路径的写法啦！

1.51 处理目录的常用命令

接下来我们就来看几个常见的处理目录的命令吧：
•ls: 列出目录
•cd：切换目录
•pwd：显示目前的目录
•mkdir：创建一个新的目录
•rmdir：删除一个空的目录
•cp: 复制文件或目录
•rm: 移除文件或目录

你可以使用 man [命令] 来查看各个命令的使用文档，如： man cp。

ls (列出目录)

在 Linux 系统当中， ls 命令可能是最常被运行的。
语法：

```
[root@www ~]# ls [-aAdfFhilnrRSt] 目录名称
[root@www ~]# ls [--color={never,auto,always}] 目录名称
[root@www ~]# ls [--full-time] 目录名称
```

选项与参数：
•-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)

```
[root@www ~]# ls -al ~
```

cd (切换目录)

cd 是 Change Directory 的缩写，这是用来变换工作目录的命令。
语法：

```
  cd [相对路径或绝对路径]
#使用 mkdir 命令创建 runoob 目录
[root@www ~]# mkdir runoob

#使用绝对路径切换到 runoob 目录
[root@www ~]# cd /root/runoob/
```

```
#使用相对路径切换到 runoob 目录
[root@www ~]# cd ./runoob/

# 表示回到自己的家目录，亦即是 /root 这个目录
[root@www runoob]# cd ~

# 表示去到目前的上一级目录，亦即是 /root 的上一级目录的意思；
[root@www ~]# cd ..
```

接下来大家多操作几次应该就可以很好的理解 cd 命令的。

pwd (显示目前所在的目录)

pwd 是 Print Working Directory 的缩写，也就是显示目前所在目录的命令。

```
[root@www ~]# pwd [-P]
```

选项与参数：

- **-P**：显示出确实的路径，而非使用连结 (link) 路径。

实例：单纯显示出目前的工作目录：

```
[root@www ~]# pwd
/root  <== 显示出目录啦～
```

实例显示出实际的工作目录，而非连结档本身的目录名而已。

```
[root@www ~]# cd /var/mail  <==注意，/var/mail 是一个连结档
[root@www mail]# pwd
/var/mail      <==列出目前的工作目录
[root@www mail]# pwd -P
/var/spool/mail  <==怎么回事？有没有加 -P 差很多～
[root@www mail]# ls -ld /var/mail
lrwxrwxrwx 1 root root 10 Sep  4 17:54 /var/mail -> spool/mail
# 看到这里应该知道为啥了吧？因为 /var/mail 是连结档，连结到 /var/spool/mail
# 所以，加上 pwd -P 的选项后，会不以连结档的数据显示，而是显示正确的完整路径啊！
```

mkdir (创建新目录)

如果想要创建新的目录的话，那么就使用 mkdir (make directory)吧。

语法：

```
mkdir [-mp] 目录名称
```

选项与参数：

- -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！

实例：请到/tmp 底下尝试创建数个新目录看看：

```
[root@www ~]# cd /tmp
[root@www tmp]# mkdir test   <==创建一名为 test 的新目录
```

```
[root@www tmp]# mkdir test1/test2/test3/test4
mkdir: cannot create directory `test1/test2/test3/test4':
No such file or directory       <== 没办法直接创建此目录啊!
[root@www tmp]# mkdir -p test1/test2/test3/test4
```

加了这个 -p 的选项，可以自行帮你创建多层目录!

实例：创建权限为 rwx--x--x 的目录。

```
[root@www tmp]# mkdir -m 711 test2
[root@www tmp]# ls -l
drwxr-xr-x  3 root  root 4096 Jul 18 12:50 test
drwxr-xr-x  3 root  root 4096 Jul 18 12:53 test1
drwx--x--x  2 root  root 4096 Jul 18 12:54 test2
```

上面的权限部分，如果没有加上 -m 来强制配置属性，系统会使用默认属性。

如果我们使用 -m ，如上例我们给予 -m 711 来给予新的目录 drwx--x--x 的权限。

rmdir (删除空的目录)

语法：

```
  rmdir [-p] 目录名称
```

选项与参数：

•-p：连同上一级『空的』目录也一起删除

删除 runoob 目录

```
[root@www tmp]# rmdir runoob/
```

将 mkdir 实例中创建的目录(/tmp 底下)删除掉!

```
[root@www tmp]# ls -l  <==看看有多少目录存在?
drwxr-xr-x  3 root  root 4096 Jul 18 12:50 test
drwxr-xr-x  3 root  root 4096 Jul 18 12:53 test1
drwx--x--x  2 root  root 4096 Jul 18 12:54 test2
[root@www tmp]# rmdir test  <==可直接删除掉，没问题
[root@www tmp]# rmdir test1  <==因为尚有内容，所以无法删除!
rmdir: `test1': Directory not empty
[root@www tmp]# rmdir -p test1/test2/test3/test4
[root@www tmp]# ls -l      <==您看看，底下的输出中 test 与 test1 不见了!
drwx--x--x  2 root  root 4096 Jul 18 12:54 test2
```

利用 -p 这个选项，立刻就可以将 test1/test2/test3/test4 一次删除。

不过要注意的是，这个 rmdir 仅能删除空的目录，你可以使用 rm 命令来删除非空目录。

cp (复制文件或目录)

cp 即拷贝文件和目录。

语法:

```
[root@www ~]# cp [-adfilprsu] 来源档(source) 目标档(destination)
```

```
[root@www ~]# cp [options] source1 source2 source3 .... directory
```

选项与参数：

- -a：相当於 -pdr 的意思，至於 pdr 请参考下列说明；(常用)
- -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；
- -f：为强制(force)的意思，若目标文件已经存在且无法开启，则移除后再尝试一次；
- -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)
- -s：复制成为符号连结档 (symbolic link)，亦即『捷径』文件；
- -u：若 destination 比 source 旧才升级 destination！

用 root 身份，将 root 目录下的 .bashrc 复制到 /tmp 下，并命名为 bashrc

```
[root@www ~]# cp ~/.bashrc /tmp/bashrc
[root@www ~]# cp -i ~/.bashrc /tmp/bashrc
cp: overwrite `/tmp/bashrc'? n  <==n 不覆盖，y 为覆盖
```

rm (移除文件或目录)

语法：

```
  rm [-fir] 文件或目录
```

选项与参数：

- -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！！！
-

将刚刚在 cp 的实例中创建的 bashrc 删除掉！

```
[root@www tmp]# rm -i bashrc
rm: remove regular file `bashrc'? y
```

如果加上 -i 的选项就会主动询问喔，避免你删除到错误的档名！

mv (移动文件与目录，或修改名称)

语法：

```
[root@www ~]# mv [-fiu] source destination
[root@www ~]# mv [options] source1 source2 source3 .... directory
```

选项与参数：

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

复制一文件，创建一目录，将文件移动到目录中

```
[root@www ~]# cd /tmp
```

```
[root@www tmp]# cp ~/.bashrc bashrc
[root@www tmp]# mkdir mvtest
[root@www tmp]# mv bashrc mvtest
```

将某个文件移动到某个目录去，就是这样做！

将刚刚的目录名称更名为 mvtest2

```
[root@www tmp]# mv mvtest mvtest2
```

1.52 Linux 文件内容查看

Linux 系统中使用以下命令来查看文件的内容：

•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

你可以使用 man [命令]来查看各个命令的使用文档，如： man cp。

cat

由第一行开始显示文件内容

语法：

```
cat [-AbEnTv]
```

选项与参数：

•-A： 相当於 -vET 的整合选项，可列出一些特殊字符而不是空白而已；
•-b： 列出行号，仅针对非空白行做行号显示，空白行不标行号！
•-E： 将结尾的断行字节 $ 显示出来；
•-n： 列印出行号，连同空白行也会有行号，与 -b 的选项不同；
•-T： 将 [tab] 按键以 ^I 显示出来；
•-v： 列出一些看不出来的特殊字符

检看 /etc/issue 这个文件的内容：

```
[root@www ~]# cat /etc/issue
CentOS release 6.4 (Final)
Kernel \r on an \m
```

tac

tac 与 cat 命令刚好相反，文件内容从最后一行开始显示，可以看出 tac 是 cat 的倒着写！如：

```
[root@www ~]# tac /etc/issue
```

```
Kernel \r on an \m
CentOS release 6.4 (Final)
```

nl

显示行号

语法：

```
nl [-bnw] 文件
```

选项与参数：

•-b： 指定行号指定的方式，主要有两种：
-b a： 表示不论是否为空行，也同样列出行号(类似 cat -n)；
-b t： 如果有空行，空的那一行不要列出行号(默认值)；
•-n： 列出行号表示的方法，主要有三种：
-n ln： 行号在荧幕的最左方显示；
-n rn： 行号在自己栏位的最右方显示，且不加 0；
-n rz： 行号在自己栏位的最右方显示，且加 0；
•-w： 行号栏位的占用的位数。

实例一：用 nl 列出 /etc/issue 的内容

```
[root@www ~]# nl /etc/issue
     1  CentOS release 6.4 (Final)
     2  Kernel \r on an \m
```

more

一页一页翻动

```
[root@www ~]# more /etc/man.config
#
# Generated automatically from man.conf.in by the
# configure script.
#
# man.conf from man-1.6d
....(中间省略)....
--More--(28%)  <== 重点在这一行喔！你的光标也会在这里等待你的命令
```

在 more 这个程序的运行过程中，你有几个按键可以按的：

•空白键 (space)：代表向下翻一页；
•Enter　　：代表向下翻『一行』；
•/字串　　：代表在这个显示的内容当中，向下搜寻『字串』这个关键字；
•:f　　：立刻显示出档名以及目前显示的行数；
•q　　：代表立刻离开 more ，不再显示该文件内容。
•b 或 [ctrl]-b：代表往回翻页，不过这动作只对文件有用，对管线无用。

less

一页一页翻动，以下实例输出/etc/man.config 文件的内容：

```
[root@www ~]# less /etc/man.config
#
# Generated automatically from man.conf.in by the
# configure script.
#
# man.conf from man-1.6d
....(中间省略)....
:  <== 这里可以等待你输入命令!
```

less 运行时可以输入的命令有:

•空白键　: 向下翻动一页;
•[pagedown]: 向下翻动一页;
•[pageup] : 向上翻动一页;
•/字串　: 向下搜寻『字串』的功能;
•?字串　: 向上搜寻『字串』的功能;
•n　　: 重复前一个搜寻 (与 / 或 ? 有关!)
•N　　: 反向的重复前一个搜寻 (与 / 或 ? 有关!)
•q　　: 离开 less 这个程序;

head

取出文件前面几行

语法:

```
head [-n number] 文件
```

选项与参数:

•-n : 后面接数字，代表显示几行的意思

```
[root@www ~]# head /etc/man.config
```

默认的情况中，显示前面 10 行! 若要显示前 20 行，就得要这样:

```
[root@www ~]# head -n 20 /etc/man.config
```

tail

取出文件后面几行

语法:

```
tail [-n number] 文件
```

选项与参数:

•-n : 后面接数字，代表显示几行的意思
•-f : 表示持续侦测后面所接的档名，要等到按下[ctrl]-c 才会结束 tail 的侦测

```
[root@www ~]# tail /etc/man.config
# 默认的情况中，显示最后的十行! 若要显示最后的 20 行，就得要这样:
[root@www ~]# tail -n 20 /etc/man.config
```

1.53 Linux 链接概念

Linux 链接分两种，一种被称为硬链接（Hard Link），另一种被称为符号链接（Symbolic Link）。默认情况下，`ln` 命令产生硬链接。

硬连接

硬连接指通过索引节点来进行连接。在 Linux 的文件系统中，保存在磁盘分区中的文件不管是什么类型都给它分配一个编号，称为索引节点号(Inode Index)。在 Linux 中，多个文件名指向同一索引节点是存在的。比如：A 是 B 的硬链接（A 和 B 都是文件名），则 A 的目录项中的 inode 节点号与 B 的目录项中的 inode 节点号相同，即一个 inode 节点对应两个不同的文件名，两个文件名指向同一个文件，A 和 B 对文件系统来说是完全平等的。删除其中任何一个都不会影响另外一个的访问。

硬连接的作用是允许一个文件拥有多个有效路径名，这样用户就可以建立硬连接到重要文件，以防止“误删”的功能。其原因如上所述，因为对应该目录的索引节点有一个以上的连接。只删除一个连接并不影响索引节点本身和其它的连接，只有当最后一个连接被删除后，文件的数据块及目录的连接才会被释放。也就是说，文件真正删除的条件是与之相关的所有硬连接文件均被删除。

软连接

另外一种连接称之为符号连接（Symbolic Link），也叫软连接。软链接文件有类似于 Windows 的快捷方式。它实际上是一个特殊的文件。在符号连接中，文件实际上是一个文本文件，其中包含的有另一文件的位置信息。比如：A 是 B 的软链接（A 和 B 都是文件名），A 的目录项中的 inode 节点号与 B 的目录项中的 inode 节点号不相同，A 和 B 指向的是两个不同的 inode，继而指向两块不同的数据块。但是 A 的数据块中存放的只是 B 的路径名（可以根据这个找到 B 的目录项）。A 和 B 之间是“主从”关系，如果 B 被删除了，A 仍然存在（因为两个是不同的文件），但指向的是一个无效的链接。

通过实验加深理解

```
[oracle@Linux]$ touch f1       #创建一个测试文件 f1
[oracle@Linux]$ ln f1 f2       #创建 f1 的一个硬连接文件 f2
[oracle@Linux]$ ln -s f1 f3    #创建 f1 的一个符号连接文件 f3
[oracle@Linux]$ ls -li         # -i 参数显示文件的 inode 节点信息
total 0
9797648 -rw-r--r--  2 oracle oinstall 0 Apr 21 08:11 f1
9797648 -rw-r--r--  2 oracle oinstall 0 Apr 21 08:11 f2
9797649 lrwxrwxrwx  1 oracle oinstall 2 Apr 21 08:11 f3 -> f1
```

从上面的结果中可以看出，硬连接文件 f2 与原文件 f1 的 inode 节点相同，均为 9797648，然而符号连接文件的 inode 节点不同。

```
[oracle@Linux]$ echo "I am f1 file" >>f1
[oracle@Linux]$ cat f1
I am f1 file
```

[oracle@Linux]$ cat f2

```
I am f1 file
[oracle@Linux]$ cat f3
I am f1 file
[oracle@Linux]$ rm -f f1
[oracle@Linux]$ cat f2
I am f1 file
[oracle@Linux]$ cat f3
cat: f3: No such file or directory
```

通过上面的测试可以看出：当删除原始文件 f1 后，硬连接 f2 不受影响，但是符号连接 f1 文件无效

总结

依此您可以做一些相关的测试，可以得到以下全部结论：

•1).删除符号连接 f3,对 f1,f2 无影响；

•2).删除硬连接 f2，对 f1,f3 也无影响；

•3).删除原文件 f1，对硬连接 f2 没有影响，导致符号连接 f3 失效；

•4).同时删除原文件 f1,硬连接 f2，整个文件会真正的被删除。

1.6 Linux 磁盘管理

Linux 磁盘管理好坏直接关系到整个系统的性能问题。

Linux 磁盘管理常用三个命令为 df、du 和 fdisk。

•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

df 命令参数功能：检查文件系统的磁盘空间占用情况。可以利用该命令来获取硬盘被占用了多少空间，目前还剩下多少空间等信息。

语法：

```
df [-ahikHTm] [目录或文件名]
```

选项与参数：

•-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

实例 1

将系统内所有的文件系统列出来！

```
[root@www ~]# df
Filesystem    1K-blocks    Used Available Use% Mounted on
/dev/hdc2      9920624  3823112  5585444  41% /
/dev/hdc3      4956316   141376  4559108   4% /home
/dev/hdc1       101086    11126    84741  12% /boot
tmpfs           371332        0   371332   0% /dev/shm
```

在 Linux 底下如果 df 没有加任何选项，那么默认会将系统内所有的 (不含特殊内存内的文件系统与 swap) 都以 1 Kbytes 的容量来列出来！

实例 2

将容量结果以易读的容量格式显示出来

```
[root@www ~]# df -h
Filesystem        Size  Used Avail Use% Mounted on
/dev/hdc2         9.5G  3.7G  5.4G  41% /
/dev/hdc3         4.8G  139M  4.4G   4% /home
/dev/hdc1          99M   11M   83M  12% /boot
tmpfs             363M    0  363M   0% /dev/shm
```

实例 3

将系统内的所有特殊文件格式及名称都列出来

```
[root@www ~]# df -aT
Filesystem    Type 1K-blocks    Used Available Use% Mounted on
/dev/hdc2     ext3   9920624 3823112   5585444  41% /
proc          proc         0       0         0   -  /proc
sysfs         sysfs        0       0         0   -  /sys
devpts        devpts       0       0         0   -  /dev/pts
/dev/hdc3     ext3   4956316  141376   4559108   4% /home
/dev/hdc1     ext3    101086   11126     84741  12% /boot
tmpfs         tmpfs   371332       0    371332   0% /dev/shm
none          binfmt_misc      0       0         0   -  /proc/sys/fs/binfmt_misc
sunrpc        rpc_pipefs       0       0         0   -  /var/lib/nfs/rpc_pipefs
```

实例 4

将 /etc 底下的可用的磁盘容量以易读的容量格式显示

```
[root@www ~]# df -h /etc
Filesystem        Size  Used Avail Use% Mounted on
/dev/hdc2         9.5G  3.7G  5.4G  41% /
```

---

du

Linux du 命令也是查看使用空间的，但是与 df 命令不同的是 Linux du 命令是对文件和目录磁盘使用的空间的查看，还是和 df 命令有一些区别的，这里介绍 Linux du 命令。

语法：

```
du [-ahskm] 文件或目录名称
```

选项与参数：

- -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。
- -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；

实例 1

列出目前目录下的所有文件容量

```
[root@www ~]# du
8     ./test4    <==每个目录都会列出来
8     ./test2
....中间省略....
12    ./.gconfd   <==包括隐藏文件的目录
220   .         <==这个目录(.)所占用的总量
```

直接输入 du 没有加任何选项时，则 du 会分析当前所在目录的文件与目录所占用的硬盘空间。

实例 2

将文件的容量也列出来

```
[root@www ~]# du -a
12    ./install.log.syslog   <==有文件的列表了
8     ./.bash_logout
8     ./test4
8     ./test2
....中间省略....
12    ./.gconfd
220   .
```

实例 3

检查根目录底下每个目录所占用的容量

```
[root@www ~]# du -sm /*
7     /bin
6     /boot
.....中间省略....
0     /proc
```

```
.....中间省略....
1     /tmp
3859   /usr   <==系统初期最大就是他了啦!
77     /var
```

通配符 * 来代表每个目录。

与 df 不一样的是，du 这个命令其实会直接到文件系统内去搜寻所有的文件数据。

---

fdisk

fdisk 是 Linux 的磁盘分区表操作工具。

语法：

```
fdisk [-l] 装置名称
```

选项与参数：

•-l：输出后面接的装置所有的分区内容。若仅有 fdisk -l 时， 则系统将会把整个系统内能够搜寻到的装置的分区均列出来。

实例 1

列出所有分区信息

```
[root@AY120919111755c246621 tmp]# fdisk -l

Disk /dev/xvda: 21.5 GB, 21474836480 bytes
255 heads, 63 sectors/track, 2610 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00000000

   Device Boot      Start         End      Blocks   Id  System
/dev/xvda1   *           1        2550    20480000   83  Linux
/dev/xvda2            2550        2611      490496   82  Linux swap / Solaris

Disk /dev/xvdb: 21.5 GB, 21474836480 bytes
255 heads, 63 sectors/track, 2610 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x56f40944

   Device Boot      Start         End      Blocks   Id  System
/dev/xvdb2               1        2610    20964793+  83  Linux
```

实例 2

找出你系统中的根目录所在磁盘，并查阅该硬盘内的相关信息

```
[root@www ~]# df /        <==注意：重点在找出磁盘文件名而已
Filesystem         1K-blocks     Used Available Use% Mounted on
/dev/hdc2           9920624   3823168   5585388  41% /

[root@www ~]# fdisk /dev/hdc  <==仔细看，不要加上数字喔！
The number of cylinders for this disk is set to 5005.
There is nothing wrong with that, but this is larger than 1024,
and could in certain setups cause problems with:
1) software that runs at boot time (e.g., old versions of LILO)
2) booting and partitioning software from other OSs
      (e.g., DOS FDISK, OS/2 FDISK)

Command (m for help):    <==等待你的输入！
```

输入 m 后，就会看到底下这些命令介绍

```
Command (m for help): m   <== 输入 m 后，就会看到底下这些命令介绍
Command action
      a   toggle a bootable flag
      b   edit bsd disklabel
      c   toggle the dos compatibility flag
      d   delete a partition          <==删除一个 partition
      l   list known partition types
      m   print this menu
      n   add a new partition         <==新增一个 partition
      o   create a new empty DOS partition table
      p   print the partition table    <==在屏幕上显示分割表
      q   quit without saving changes  <==不储存离开 fdisk 程序
      s   create a new empty Sun disklabel
      t   change a partition's system id
      u   change display/entry units
      v   verify the partition table
      w   write table to disk and exit  <==将刚刚的动作写入分割表
      x   extra functionality (experts only)
```

离开 fdisk 时按下 q，那么所有的动作都不会生效！相反的， 按下 w 就是动作生效的意思。

```
Command (m for help): p  <== 这里可以输出目前磁盘的状态

Disk /dev/hdc: 41.1 GB, 41174138880 bytes      <==这个磁盘的文件名与容量
255 heads, 63 sectors/track, 5005 cylinders    <==磁头、扇区与磁柱大小
Units = cylinders of 16065 * 512 = 8225280 bytes <==每个磁柱的大小

      Device Boot    Start       End    Blocks   Id  System
/dev/hdc1   *         1        13      104391   83  Linux
/dev/hdc2            14      1288    10241437+  83  Linux
/dev/hdc3          1289      1925     5116702+  83  Linux
/dev/hdc4          1926      5005    24740100    5  Extended
```

```
/dev/hdc5        1926      2052    1020096  82  Linux swap / Solaris
# 装置文件名 启动区否 开始磁柱   结束磁柱  1K 大小容量 磁盘分区槽内的系统

Command (m for help): q
```

想要不储存离开吗？按下 q 就对了！不要随便按 w 啊！

使用 p 可以列出目前这颗磁盘的分割表信息，这个信息的上半部在显示整体磁盘的状态。

磁盘格式化

磁盘分割完毕后自然就是要进行文件系统的格式化，格式化的命令非常的简单，使用 mkfs（make filesystem）命令。

语法：

```
mkfs [-t 文件系统格式] 装置文件名
```

选项与参数：

•-t ： 可以接文件系统格式，例如 ext3, ext2, vfat 等(系统有支持才会生效)

实例 1

查看 mkfs 支持的文件格式

```
[root@www ~]# mkfs[tab][tab]
mkfs      mkfs.cramfs  mkfs.ext2   mkfs.ext3   mkfs.msdos   mkfs.vfat
```

按下两个[tab]，会发现 mkfs 支持的文件格式如上所示。

实例 2

将分区 /dev/hdc6（可指定你自己的分区） 格式化为 ext3 文件系统：

```
[root@www ~]# mkfs -t ext3 /dev/hdc6
mke2fs 1.39 (29-May-2006)
Filesystem label=           <==这里指的是分割槽的名称(label)
OS type: Linux
Block size=4096 (log=2)       <==block 的大小配置为 4K
Fragment size=4096 (log=2)
251392 inodes, 502023 blocks    <==由此配置决定的 inode/block 数量
25101 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=515899392
16 block groups
32768 blocks per group, 32768 fragments per group
15712 inodes per group
Superblock backups stored on blocks:
                32768, 98304, 163840, 229376, 294912

Writing inode tables: done
Creating journal (8192 blocks): done <==有日志记录
```

```
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 34 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
# 这样就创建起来我们所需要的 Ext3 文件系统了！简单明了！
```

磁盘检验

fsck（file system check）用来检查和维护不一致的文件系统。
若系统掉电或磁盘发生问题，可利用 fsck 命令对文件系统进行检查。
语法：

```
fsck [-t 文件系统] [-ACay] 装置名称
```

选项与参数：

- -t : 给定档案系统的型式，若在 /etc/fstab 中已有定义或 kernel 本身已支援的则不需加上此参数
- -s : 依序一个一个地执行 fsck 的指令来检查
- -A : 对/etc/fstab 中所有列出来的 分区（partition）做检查
- -C : 显示完整的检查进度
- -d : 打印出 e2fsck 的 debug 结果
- -p : 同时有 -A 条件时，同时有多个 fsck 的检查一起执行
- -R : 同时有 -A 条件时，省略 / 不检查
- -V : 详细显示模式
- -a : 如果检查有错则自动修复
- -r : 如果检查有错则由使用者回答是否修复
- -y : 选项指定检测每个文件是自动输入 yes，在不确定那些是不正常的时候，可以执行 # fsck -y 全部检查修复。

实例 1

查看系统有多少文件系统支持的 fsck 命令：

```
[root@www ~]# fsck[tab][tab]
fsck      fsck.cramfs  fsck.ext2   fsck.ext3   fsck.msdos   fsck.vfat
```

实例 2

强制检测 /dev/hdc6 分区:

```
[root@www ~]# fsck -C -f -t ext3 /dev/hdc6
fsck 1.39 (29-May-2006)
e2fsck 1.39 (29-May-2006)
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
vbird_logical: 11/251968 files (9.1% non-contiguous), 36926/1004046 blocks
```

如果没有加上 -f 的选项，则由于这个文件系统不曾出现问题，检查的经过非常快速！若加上 -f 强制检查，才会一项一项的显示过程。

磁盘挂载与卸除

Linux 的磁盘挂载使用 mount 命令，卸载使用 umount 命令。
磁盘挂载语法：

```
mount [-t 文件系统] [-L Label 名] [-o 额外选项] [-n]  装置文件名  挂载点
```

实例 1

用默认的方式，将刚刚创建的 /dev/hdc6 挂载到 /mnt/hdc6 上面！

```
[root@www ~]# mkdir /mnt/hdc6
[root@www ~]# mount /dev/hdc6 /mnt/hdc6
[root@www ~]# df
Filesystem      1K-blocks     Used Available Use% Mounted on
.....中间省略.....
/dev/hdc6         1976312    42072   1833836   3% /mnt/hdc6
```

磁盘卸载命令 umount 语法：

```
umount [-fn] 装置文件名或挂载点
```

选项与参数：
•-f ： 强制卸除！可用在类似网络文件系统 (NFS) 无法读取到的情况下；
•-n ： 不升级 /etc/mtab 情况下卸除。

卸载/dev/hdc6

```
[root@www ~]# umount /dev/hdc6
```

1.7 Linux 中的块文件设备和字符文件设备

系统中能够随机（不需要按顺序）访问固定大小数据片（chunks）的设备被称作块设备，这些数据片就称作块。最常
见的块设备是硬盘，除此以外，还有软盘驱动器、CD-ROM 驱动器和闪存等等许多其他块设备。注意，它们都是以安装文件系统的方式使用的——这也是块设备
的一般访问方式。

另一种基本的设备类型是字符设备。字符设备按照字符流的方式被有序访问，像串口和键盘就都属于字符设备。如果一个硬件设备是以字符流的方式被访问的话，那就应该将它归于字符设备；反过来，如果一个设备是随机（无序的）访问的，那么它就属于块设备。
这两种类型的设备的根本区别在于它们是否可以被随机访问——换句话说就是，能否在访问设备时随意地从一个位置跳转到另一个位置。举
个例子，键盘这种设备提供的就是一个数据流，当你敲入“fox”
这个字符串时，键盘驱动程序会按照和输入完全相同的顺序返回这个由三个字符组成的数据流。如果让

键盘驱动程序打乱顺序来读字符串，或读取其他字符，都是没
有意义的。所以键盘就是一种典型的字符设备，它提供的就是用户从键盘输入的字符流。对键盘进行读操作会得到一个字符流，首先是“f”，然后是“o”，最后
是“x”，最终是文件的结束（EOF）。当没人敲键盘时，字符流就是空的。硬盘设备的情况就不大一样了。硬盘设备的驱动可能要求读取磁盘上任意块的内容，
然后又转去读取别的块的内容，而被读取的块在磁盘上位置不一定要连续，所以说硬盘可以被随机访问，而不是以流的方式被访问，显然它是一个块设备。

内核管理块设备要比管理字符设备细致得多，需要考虑的问题和完成的工作相比字符设备来说要复杂许多。这是因为字
符设备仅仅需要控制一个位置—当前位置—而块设备访问的位置必须能够在介质的不同区间前后移动。所以事实上内核不必提供一个专门的子系统来管理字符设备，
但是对块设备的管理却必须要有一个专门的提供服务的子系统。不仅仅是因为块设备的复杂性远远高于字符设备，更重要的原因是块设备对执行性能的要求很高；对
硬盘每多一分利用都会对整个系统的性能带来提升，其效果要远远比键盘吞吐速度成倍的提高大得多。另外，我们将会看到，块设备的复杂性会为这种优化留下很大
的施展空间。

简单来讲，块设备可以随机存取，而字符设备不能随机存取，那裸设备又该如何解释呢?

难道裸设备，如磁盘裸设备也不能随机读取吗？那在数据库中用裸设备建一个 2g 的数据文件，为了存取最后一个数据块，难道 ORACLE 还要把前面的所有数据块都读一遍，显然不符合事实，如果这样解释呢，操作系统不能随机读取，并不意味着数据库也不能随机读取。

块设备通过系统缓存进行读取，不是直接和物理磁盘读取。字符设备可以直接物理磁盘读取，不经过系统缓存。（如键盘，直接相应中断）

2.Shell 简介及基础知识

2.1 一个简单的问题

照着教程写的代码，却跑不出来。

出现这种问题的原因，或许是电脑上的 sh 指向为 dash 而非 bash。

majc0806@majc0806-OptiPlex-3020:~/文档/每日总结$ ll /bin/sh

lrwxrwxrwx 1 root root 4 8 月 15 11:07 /bin/sh -> dash*

我们可以看到，确实是指向了 dash。

如今 Debian 和 Ubuntu 中，/bin/sh 默认已经指向 dash，这是一个不同于 bash 的 shell，它主要是为了执行脚本而出现，而不是交互， 它速度更快,但功能相比 bash 要少很多，语法严格遵守 POSIX 标准。

在运行程序的时候，可以使用 bash file.sh 而非 sh file.sh

2.2 什么是 shell?

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 脚本

Shell 脚本（shell script），是一种为 shell 编写的脚本程序。

业界所说的 shell 通常都是指 shell 脚本，但读者朋友要知道，shell 和 shell script 是两个不同的概念。

由于习惯的原因，简洁起见，本文出现的 "shell 编程" 都是指 shell 脚本编程，不是指开发 shell 自身。

Shell 环境

Shell 编程跟 java、php 编程一样，只要有一个能编写代码的文本编辑器和一个能解释执行的脚本解释器就可以了。

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）

……

本教程关注的是 Bash，也就是 Bourne Again Shell，由于易用和免费，Bash 在日常工作中被广泛使用。同时，Bash 也是大多数 Linux 系统默认的 Shell。

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。

运行 Shell 脚本有两种方法：

1、作为可执行程序

将上面的代码保存为 test.sh，并 cd 到相应目录：

```
chmod +x ./test.sh  #使脚本具有执行权限
./test.sh  #执行脚本
```

注意，一定要写成 ./test.sh，而不是 test.sh，运行其它二进制的程序也一样，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

这种运行方式是，直接运行解释器，其参数就是 shell 脚本的文件名，如：

/bin/sh test.sh

/bin/php test.php

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。

2.3 Shell 变量

定义变量时，变量名不加美元符号（$，PHP 语言中变量需要），如：

your_name="runoob.com"

注意，变量名和等号之间不能有空格，这可能和你熟悉的所有编程语言都不一样。同时，变量名的命名须遵循如下规则：

•命名只能使用英文字母，数字和下划线，首个字符不能以数字开头。
•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）。
有效的 Shell 变量名示例如下：

RUNOOB

LD_LIBRARY_PATH

_var

var2

无效的变量命名：

?var=123

user*name=runoob

除了显式地直接赋值，还可以用语句给变量赋值，如：

for file in `ls /etc`

或

for file in $(ls /etc)

以上语句将 /etc 下目录的文件名循环出来。

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，如：

```
your_name="qinjx"
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：

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

如果不给 skill 变量加花括号，写成 echo "I am good at $skillScript"，解释器就会把$skillScript 当成一个变量（其值为空），代码执行结果就不是我们期望的样子了。

推荐给所有变量加上花括号，这是个好的编程习惯。

已定义的变量，可以被重新定义，如：

```
your_name="tom"
echo $your_name
your_name="alibaba"
echo $your_name
```

这样写是合法的，但注意，第二次赋值的时候不能写$your_name="alibaba"，使用变量的时候才加美元符（$）。

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。

下面的例子尝试更改只读变量，结果报错：

```
#!/bin/bash
myUrl="http://www.google.com"
readonly myUrl
myUrl="http://www.runoob.com"
```

运行脚本，结果如下：

```
/bin/sh: NAME: This variable is read only.
```

删除变量

使用 unset 命令可以删除变量。语法：

unset variable_name

变量被删除后不能再次使用。unset 命令不能删除只读变量。

实例

#!/bin/sh

myUrl="http://www.runoob.com"

unset myUrl

echo $myUrl

以上实例执行将没有任何输出。

变量类型

运行 shell 时，会同时存在三种变量：

•**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。
•**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

2.4 Shell 字符串

字符串是 shell 编程中最常用最有用的数据类型（除了数字和字符串，也没啥其它类型好用了），字符串可以用单引号，也可以用双引号，也可以不用引号。单双引号的区别跟 PHP 类似。

单引号

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独一个的单引号（对单引号使用转义符后也不行），但可成对出现，作为字符串拼接使用。

双引号

your_name='runoob'

str="Hello, I know you are \"$your_name\"! \n"

```
echo $str
```

```
Hello, I know you are "runoob"!
```

输出结果为：

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

拼接字符串

```
your_name="runoob"
# 使用双引号拼接
greeting="hello, "$your_name" !"
greeting_1="hello, ${your_name} !"
echo $greeting  $greeting_1
# 使用单引号拼接
greeting_2='hello, '$your_name' !'
greeting_3='hello, ${your_name} !'
echo $greeting_2  $greeting_3
```

输出结果为：

```
hello, runoob ! hello, runoob !
hello, runoob ! hello, ${your_name} !
```

获取字符串长度

```
string="abcd"
echo ${#string} #输出 4
```

提取子字符串

以下实例从字符串第 **2** 个字符开始截取 **4** 个字符：

```
string="runoob is a great site"
echo ${string:1:4} # 输出 unoo
```

查找子字符串

查找字符 **i** 或 **o** 的位置(哪个字母先出现就计算哪个)：

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

echo `expr index "$string" io` # 输出 4

注意： 以上脚本中 ` 是反引号，而不是单引号 '，不要看错了哦。

2.5 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……

实例

以下实例我们向脚本传递三个参数，并分别输出，其中 **$0** 为执行的文件名：

```
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";
```

为脚本设置可执行权限，并执行脚本，输出结果如下所示：

```
$ chmod +x test.sh
$ ./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
```

另外，还有几个特殊字符用来处理参数：

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。

echo "Shell 传递参数实例！";

echo "第一个参数为：$1";

echo "参数个数为：$#";

echo "传递的参数作为一个字符串显示：$*";

执行脚本，输出结果如下所示：

$ chmod +x test.sh

$ ./test.sh 1 2 3

Shell 传递参数实例!

第一个参数为：1

参数个数为：3

传递的参数作为一个字符串显示：1 2 3

$* 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）。

echo "-- \$* 演示 ---"

```
for i in "$*"; do
   echo $i
done


echo "-- \$@ 演示 ---"
for i in "$@"; do
   echo $i
done
```

执行脚本，输出结果如下所示：

```
$ chmod +x test.sh
$ ./test.sh 1 2 3
-- $* 演示 ---
1 2 3
-- $@ 演示 ---
1
2
3
```

2.6 Shell 数组

数组中可以存放多个值。Bash Shell 只支持一维数组（不支持多维数组），初始化时不需要定义数组大小（与 PHP 类似）。

与大部分编程语言类似，数组元素的下标由 0 开始。

Shell 数组用括号来表示，元素用"空格"符号分割开，语法格式如下：

```
array_name=(value1 ... valuen)
```

实例

```
my_array=(A B "C" D)
```

我们也可以使用下标来定义数组:

```
array_name[0]=value0
array_name[1]=value1
array_name[2]=value2
```

读取数组

读取数组元素值的一般格式是：

${array_name[index]}

实例

```
my_array=(A B "C" D)

echo "第一个元素为: ${my_array[0]}"
echo "第二个元素为: ${my_array[1]}"
echo "第三个元素为: ${my_array[2]}"
echo "第四个元素为: ${my_array[3]}"
```

执行脚本，输出结果如下所示：

```
$ chmod +x test.sh
$ ./test.sh
第一个元素为: A
第二个元素为: B
第三个元素为: C
第四个元素为: D
```

获取数组中的所有元素

使用@ 或 * 可以获取数组中的所有元素，例如：

```
my_array[0]=A
my_array[1]=B
my_array[2]=C
my_array[3]=D

echo "数组的元素为: ${my_array[*]}"
echo "数组的元素为: ${my_array[@]}"
```

执行脚本，输出结果如下所示：

```
$ chmod +x test.sh
```

$ ./test.sh

数组的元素为: A B C D

数组的元素为: A B C D

获取数组的长度

获取数组长度的方法与获取字符串长度的方法相同，例如：

my_array[0]=A

my_array[1]=B

my_array[2]=C

my_array[3]=D

echo "数组元素个数为: ${#my_array[*]}"

echo "数组元素个数为: ${#my_array[@]}"

执行脚本，输出结果如下所示：

$ chmod +x test.sh

$ ./test.sh

数组元素个数为: 4

数组元素个数为: 4

2.7 Shell 基本运算符

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

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。

例如，两个数相加(**注意使用的是反引号 ` 而不是单引号 '**)：

#!/bin/bash

```
val=`expr 2 + 2`

echo "两数之和为 : $val"
```

运行实例 »

执行脚本，输出结果如下所示：

```
两数之和为 : 4
```

两点注意：

•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。

算术运算符

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。

注意：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]。

实例

算术运算符实例如下：

```
a=10
b=20

val=`expr $a + $b`
echo "a + b : $val"

val=`expr $a - $b`
echo "a - b : $val"

val=`expr $a \* $b`
echo "a * b : $val"

val=`expr $b / $a`
echo "b / a : $val"

val=`expr $b % $a`
echo "b % a : $val"

if [ $a == $b ]
then
  echo "a 等于 b"
fi
if [ $a != $b ]
then
  echo "a 不等于 b"
fi
```

执行脚本，输出结果如下所示：

```
a + b : 30
a - b : -10
a * b : 200
b / a : 2
b % a : 0
a 不等于 b
```

注意：

- 乘号(*)前边必须加反斜杠(\)才能实现乘法运算；
- if...then...fi 是条件语句，后续将会讲解。
- 在 MAC 中 shell 的 expr 语法是：**$((表达式))**，此处表达式中的 "*" 不需要转义符号 "\" 。

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 $b] 返回 false。
-ne	检测两个数是否不相等，不相等返回 true。	[$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。

实例

关系运算符实例如下：

```
a=10
b=20

if [ $a -eq $b ]
then
  echo "$a -eq $b : a 等于 b"
else
  echo "$a -eq $b: a 不等于 b"
fi
if [ $a -ne $b ]
then
  echo "$a -ne $b: a 不等于 b"
else
  echo "$a -ne $b : a 等于 b"
fi
if [ $a -gt $b ]
then
  echo "$a -gt $b: a 大于 b"
else
  echo "$a -gt $b: a 不大于 b"
fi
if [ $a -lt $b ]
then
  echo "$a -lt $b: a 小于 b"
else
  echo "$a -lt $b: a 不小于 b"
fi
if [ $a -ge $b ]
```

```
then
  echo "$a -ge $b: a 大于或等于 b"
else
  echo "$a -ge $b: a 小于 b"
fi
if [ $a -le $b ]
then
  echo "$a -le $b: a 小于或等于 b"
else
  echo "$a -le $b: a 大于 b"
fi
```

执行脚本，输出结果如下所示：

```
10 -eq 20: a 不等于 b
10 -ne 20: a 不等于 b
10 -gt 20: a 不大于 b
10 -lt 20: a 小于 b
10 -ge 20: a 小于 b
10 -le 20: a 小于或等于 b
```

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。

实例

布尔运算符实例如下：

```
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 小于 100 且 $b 大于 15 : 返回 true"
else
  echo "$a 小于 100 且 $b 大于 15 : 返回 false"
fi
if [ $a -lt 100 -o $b -gt 100 ]
then
  echo "$a 小于 100 或 $b 大于 100 : 返回 true"
else
  echo "$a 小于 100 或 $b 大于 100 : 返回 false"
fi
if [ $a -lt 5 -o $b -gt 100 ]
then
  echo "$a 小于 5 或 $b 大于 100 : 返回 true"
else
  echo "$a 小于 5 或 $b 大于 100 : 返回 false"
fi
```

执行脚本，输出结果如下所示：

10 != 20 : a 不等于 b

10 小于 100 且 20 大于 15 : 返回 true

10 小于 100 或 20 大于 100 : 返回 true

10 小于 5 或 20 大于 100 : 返回 false

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

实例

逻辑运算符实例如下：

```
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
```

执行脚本，输出结果如下所示：

```
返回 false
返回 true
```

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。

实例

字符串运算符实例如下：

```
a="abc"
b="efg"

if [ $a = $b ]
then
  echo "$a = $b : a 等于 b"
else
```

```
  echo "$a = $b: a 不等于 b"
fi
if [ $a != $b ]
then
  echo "$a != $b : a 不等于 b"
else
  echo "$a != $b: a 等于 b"
fi
if [ -z $a ]
then
  echo "-z $a : 字符串长度为 0"
else
  echo "-z $a : 字符串长度不为 0"
fi
if [ -n "$a" ]
then
  echo "-n $a : 字符串长度不为 0"
else
  echo "-n $a : 字符串长度为 0"
fi
if [ $a ]
then
  echo "$a : 字符串不为空"
else
  echo "$a : 字符串为空"
fi
```

执行脚本，输出结果如下所示：

```
abc = efg: a 不等于 b
abc != efg : a 不等于 b
-z abc : 字符串长度不为 0
```

-n abc：字符串长度不为 0

abc：字符串不为空

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。

实例

变量 file 表示文件"/var/www/runoob/test.sh"，它的大小为 100 字节，具有 rwx 权限。下面的代码，将检测该文件的各种属性：

```
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 [ -x $file ]
then
  echo "文件可执行"
else
  echo "文件不可执行"
fi
if [ -f $file ]
```

```
then
  echo "文件为普通文件"
else
  echo "文件为特殊文件"
fi
if [ -d $file ]
then
  echo "文件是个目录"
else
  echo "文件不是个目录"
fi
if [ -s $file ]
then
  echo "文件不为空"
else
  echo "文件为空"
fi
if [ -e $file ]
then
  echo "文件存在"
else
  echo "文件不存在"
fi
```

执行脚本，输出结果如下所示：

```
文件可读
文件可写
文件可执行
文件为普通文件
文件不是个目录
文件不为空
```

文件存在

2.8 Shell echo 命令

Shell 的 echo 指令与 PHP 的 echo 指令类似，都是用于字符串的输出。命令格式：

echo string

您可以使用 echo 实现更复杂的输出格式控制。

1.显示普通字符串:

echo "It is a test"

这里的双引号完全可以省略，以下命令与上面实例效果一致：

echo It is a test

2.显示转义字符

echo "\"It is a test\""

结果将是:

"It is a test"

同样，双引号也可以省略

3.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的值指定给 shell 变量

#!/bin/sh

read name

echo "$name It is a test"

以上代码保存为 test.sh，name 接收标准输入的变量，结果将是:

[root@www ~]# sh test.sh

OK   #标准输入

OK It is a test   #输出

4.显示换行

echo -e "OK! \n" # -e 开启转义

echo "It it a test"

输出结果：

OK!

It it a test

5.显示不换行

#!/bin/sh

echo -e "OK! \c" # -e 开启转义 \c 不换行

echo "It is a test"

输出结果：

OK! It is a test

6.显示结果定向至文件

echo "It is a test" > myfile

7.原样输出字符串，不进行转义或取变量(用单引号)

echo '$name\'"

输出结果：

$name\"

8.显示命令执行结果

echo `date`

注意： 这里使用的是反引号 `, 而不是单引号 '。

结果将显示当前日期

Thu Jul 24 10:08:46 CST 2014