UNIX中文教程

极客学院出版

前言

UNIX 是一种能够同时处理多个用户活动的计算机操作系统。

UNIX 由 AT&T 贝尔实验室的肯汤普森和丹尼斯 · 里奇开发完成，并于 1969 年左右面世。本教程对 UNIX 做了详细的介绍。

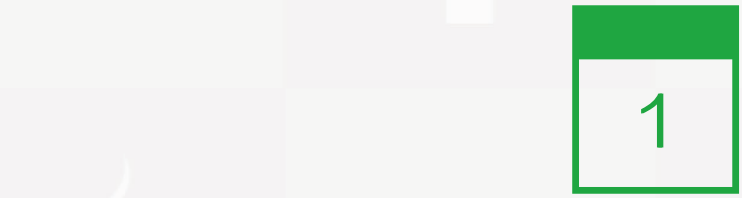
适用人群

本教程为初学者准备了帮助他们理解先进的理念基础，涵盖 UNIX 命令、 UNIX Shell 脚本和各种实用程序。

学习前提

我们假设您对操作系统和其功能的了解很少。对各种计算机的概念的基本了解也将帮助您理解本教程中提供的各种练习。

目录



1

UNIX 基础

什么是 UNIX？

UNIX 操作系统是一系列的程序，将计算机和用户联系在一起。

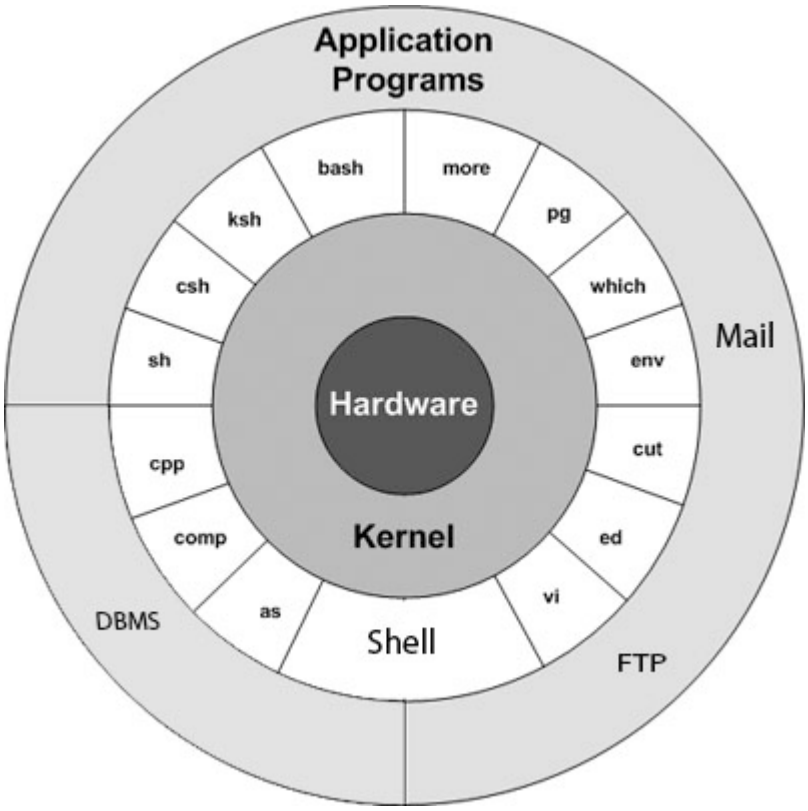
分配系统资源和协调计算机内部的所有详细信息的计算机程序被称为操作系统或内核。

用户通过一个称为 Shell 的程序内核进行通信。Shell 是一个命令行解释器; 它将用户输入的命令进行转换，并将它们转换为一种可以使内核理解的语言。

- UNIX 最初是 1969 年由一批在贝尔实验室的人员开发出来的，包括 Ken Thompson，Dennis Ritchie，Douglas McIlroy 和 Joe Ossanna。
- 在市场上有各种 UNIX 变体。例如 Solaris Unix, AIX, HP Unix and BSD 是一些例子。Linux 也是受欢迎的免费的 UNIX。
- 许多人可以同时使用 UNIX 计算机；因此 UNIX 被称为多用户系统。
- 用户也可以在同一时间运行多个程序；因此 UNIX 被称为多任务处理。

UNIX 体系结构

这里是一个 UNIX 系统基本框图:



总结所有版本的 UNIX 的主要概念包含以下四个基本要素:

- **内核**：内核是操作系统的核心。它与硬件和大多数任务像内存管理任务调度和文件管理交互。
- **Shell**：shell 是用于处理您的请求的实用程序。当您在您的终端键入命令时，Shell 将命令解释并调用你想要的程序。Shell 使用标准语法的所有命令。C Shell, Bourne Shell 和 Korn Shell 是最著名的 shell ，适用于大多数 UNIX 变体。
- **命令和实用程序**：有各种各样的命令和实用程序可供您使用。cp, mv, cat 和 grep 等是命令和实用程序的几个例子。有超过 250 标准命令，再加上通过第三方软件提供的其他命令。所有的命令都跟着各种可选的选项。
- **文件和目录**：在 UNIX 中的所有数据被都组织到文件中。所有文件被都组织到目录中。这些目录被组织成一个称为文件系统的树状结构。

系统启动

如果你有一台电脑安装了 UNIX 操作系统，然后你只需要打开其电源，使其运行。

只要你打开电源，系统开始启动，最后它会提示您登录到系统，登录到系统和使用它为您日复一日的活动。

登陆 UNIX

当你第一次连接到 UNIX 系统时，你通常会看到如下提示:

```
login:
```

登录

- 准备好您的用户名 (用户标识) 及密码。如果你还没有这些，请联系您的系统管理员。
- 在登录提示符下，键入您的用户名，然后按 ENTER 键。您的用户 id 是区分大小写，因此请确保您键入的 id 是系统管理员分配的。
- 在密码提示符下，键入您的密码，然后按 ENTER 键。您的密码也是区分大小写的。
- 如果您提供正确的用户 id 和密码，你将被允许进入系统。此时屏幕上回显的信息如下图所示。

```
login : amrood
amrood's password:
Last login: Sun Jun 14 09:32:32 2009 from 62.61.164.73
$
```

系统会为您提供 (有时称为 $ 提示) 一个命令提示符，你可以在下面键入你所有的命令。例如若要检查日历您需要键入 cal 命令，如下所示:

```
$ cal
 June 2009
Su Mo Tu We Th Fr Sa
1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30

$
```

修改密码

所有 UNIX 系统都需要密码以确保您的文件和数据的安全性，这个约束可以保证您的文件免受黑客破坏。这里是更改密码的步骤:

1. 开始时，在命令提示符处键入 passwd 如下所示。
2. 请输入您的旧密码即您目前使用的密码。
3. 输入你的新密码。总是保持您的密码足够复杂，没有人能猜出它。但前提是保证你记得住。
4. 您将需要再次键入该密码验证的密码。

```
$ passwd
Changing password for amrood
(current) Unix password:******
New UNIX password:*******
Retype new UNIX password:*******
passwd: all authentication tokens updated  successfully

$
```

注意：我用星 (*) 的位置是告诉您那是您输入当前密码和新密码的位置，当您键入字符时这些字符不会直接显示出来，而是以 * 号代替。

列出目录和文件

在 UNIX 中的所有数据被都组织到文件。所有文件被都组织成目录。这些目录被组织成一个称为文件系统的树状结构。

您可以使用 ls 命令列出所有的文件或目录在目录中。以下是使用 ls 命令与 –l 选项的示例。

```
$ ls –l
total 19621
drwxrwxr–x  2 amrood amrood  4096 Dec 25 09:59 uml
–rw–rw–r––  1 amrood amrood  5341 Dec 25 08:38 uml.jpg
drwxr–xr–x  2 amrood amrood  4096 Feb 15  2006 univ
drwxr–xr–x  2 root   root4096 Dec  9  2007 urlspedia
–rw–r––r––  1 root   root  276480 Dec  9  2007 urlspedia.tar
drwxr–xr–x  8 root   root4096 Nov 25  2007 usr
–rwxr–xr–x  1 root   root3192 Nov 25  2007 webthumb.php
–rw–rw–r––  1 amrood amrood 20480 Nov 25  2007 webthumb.tar
–rw–rw–r––  1 amrood amrood  5654 Aug  9  2007 yourfile.mid
–rw–rw–r––  1 amrood amrood166255 Aug  9  2007 yourfile.swf

$
```

以 d..... 开头的在这里表示目录。例如 uml, univ 和 urlspedia 是目录，其余的为文件。

你是谁?

当您登录到系统时，你可能愿意知道: **我是谁**?

最简单的方法来找出"你是谁"是输入 whoami 命令:

```
$ whoami
 amrood

$
```

在你的系统上试一试。此命令将列出与当前的登录名关联的帐户名称。你可以试试 who am i 命令以此来获取有关自己的信息。

已登录的是谁?

有时你可能想知道谁同时登录到计算机。

这里有三个命令可以用来获取你此信息，基于你想要了解其他用户的程度: users，who，和 w。

```
$ users
 amrood bablu qadir

$ who
amrood ttyp0 Oct 8 14:10 (limbo)
bablu  ttyp2 Oct 4 09:08 (calliope)
qadir  ttyp4 Oct 8 12:09 (dent)

$
```

尝试在您的系统上的 w 命令来检查输出。这将列出一些更多的与记录在系统中的用户相关联的信息。

登出

当您完成您的会话时，您需要登出您的系统，确保没有其他人伪装成您访问您的文件。

登出方法

1. 只需在命令提示符下，键入 logout 命令然后系统将清理一切和断开连接

系统关机

最一致的方法来关闭 UNIX 系统是正确通过命令行使用以下命令之一:

命令	描述
halt	立即使系统关机。
init 0	在关机之前使用预定义的脚本来同步和清理你的系统。
init 6	在系统完全关闭后重新启动系统，然后将它完全备份
poweroff	通过断电自动关闭系统。
reboot	重新启动
shutdown	关机

你通常需要超级用户或根 (在 UNIX 系统上最有特权的帐户) 来关闭系统，但在一些独立或个人拥有的 UNIX 机器上，管理员用户甚至常规用户都可以这样做。

文件管理

在 UNIX 中的所有数据被都组织成文件。所有文件被都组织成目录。这些目录被组织成一个称为文件系统的树状结构。

当您使用 UNIX 时，你将花费大部分时间用一种方式或另一种方式去处理文件。本教程将教你如何创建和删除文件，复制和重命名它们，创建链接到它们等。

在 UNIX 中有三种基本类型的文件:

1. **普通文件:** 一个普通的文件是系统上包含数据、 文本或程序指令的文件。在本教程中，你将使用普通文件。
2. **目录:** 目录存储特殊和普通文件。 UNIX 目录对于熟悉 Windows 或者 Mac OS 的用户，相当于文件夹。
3. **特殊文件:** 一些特殊的文件提供访问硬件，例如硬盘、 CD − ROM 驱动器、 调制解调器和以太网适配器。其他特殊文件类似于别名或快捷方式，使您能够访问单个文件使用不同的名称。

文件列表

为了列出存储在当前目录中的文件和目录。使用下面的命令:

```
$ls
```

这里是上述命令的示例输出:

```
$ls

binhosts  lib res.03
ch07   hw1pub test_results
ch07.bak   hw2res.01  users
docs   hw3res.02  work
```

命令 ls 支持 −l 选项，将帮助您获得有关列出的文件的详细信息:

```
$ls −l
total 1962188

drwxrwxr−x  2 amrood amrood  4096 Dec 25 09:59 uml
−rw−rw−r−−  1 amrood amrood  5341 Dec 25 08:38 uml.jpg
drwxr−xr−x  2 amrood amrood  4096 Feb 15  2006 univ
drwxr−xr−x  2 root   root4096 Dec  9  2007 urlspedia
```

```
-rw-r--r-- 1 root   root  276480 Dec  9  2007 urlspedia.tar
drwxr-xr-x  8 root   root4096 Nov 25  2007 usr
drwxr-xr-x  2200300  4096 Nov 25  2007 webthumb-1.01
-rwxr-xr-x  1 root   root3192 Nov 25  2007 webthumb.php
-rw-rw-r--  1 amrood amrood 20480 Nov 25  2007 webthumb.tar
-rw-rw-r--  1 amrood amrood  5654 Aug  9  2007 yourfile.mid
-rw-rw-r--  1 amrood amrood166255 Aug  9  2007 yourfile.swf
drwxr-xr-x 11 amrood amrood  4096 May 29  2007 zlib-1.2.3
$
```

这里是有关所有列出的列信息:

1. 第一列: 表示文件类型，给出了该文件的权限。后面是所有类型的文件的说明。
2. 第二列: 表示文件或目录所采取的内存块的数目。
3. 第三列: 表示该文件的所有者。这是创建此文件的 UNIX 用户。
4. 第四列: 表示用户组。每个 UNIX 用户会有一个相关联的组。
5. 第五列: 表示文件大小以字节为单位。
6. 第六列: 表示此文件被创建或最后一次修改的日期和时间。
7. 第七列: 表示文件或目录的名称。

在 `ls -l` 清单示例中，每个文件的行开头为 `d` ， `-` ，或 `l` 。这些字符指示列出的文件的类型。

前缀	描述
-	常规的文件，如 ASCII 文本文件，二进制可执行文件，或硬链接。
b	特殊块文件。块输入输出设备文件如物理硬盘驱动器。
c	字符特殊文件。原始的输入/输出设备文件如物理硬盘驱动器。
d	包含其他文件和目录列表的目录文件。
l	符号链接文件。链接到任何一个普通的文件。
p	命名的管道。进程间通信机制。
s	用于进程间通信的套接字。

元字符

元字符在 UNIX 中具有特殊的意义。例如 `*` 和 `?` 是元字符。我们使用 `*` 匹配 0 或多个字符，问号 `?` 与单个字符匹配。

举个例子:

```
$ls ch*.doc
```

显示名称以 ch 开头，并以 .doc 结束的所有文件:

```
ch01-1.doc   ch010.doc  ch02.docch03-2.doc
ch04-1.doc   ch040.doc  ch05.docch06-2.doc
ch01-2.doc ch02-1.doc c
```

在这里 * 作为元字符可以和任何字符相匹配。如果你只是想要显示以 .doc 结尾的所有文件，你可以使用以下命令:

```
$ls *.doc
```

隐藏文件

隐藏文件，是第一个字符是圆点或句点字符 (.) 的文件。 UNIX 程序 (包括 shell) 大多数使用这些文件来存储配置信息。

隐藏文件的一些常见的例子包括文件:

- **.profile:** Bourne shell (sh) 初始化脚本。
- **.kshrc:** Korn shell (ksh) 初始化脚本。
- **.cshrc:** C shell (csh) 初始化脚本。
- **.rhosts:** remote shell 配置文件。

若要列出不可见文件，请指定到 ls -a 选项:

```
$ ls -a

. .profile   docs lib test_results
...rhostshostspub users
.emacsbinhw1  res.01  work
.exrc ch07   hw2  res.02
.kshrcch07.bak   hw3  res.03
$
```

- 单个点 . : 这个代表当前目录。
- 两个点 .. : 这个代表父目录。

创建文件

您可以使用 vi 编辑器来创建任何 UNIX 系统上的普通文件。你只需要给出以下命令:

```
$ vi filename
```

上面的命令会打开一个给定的文件名的文件。您将需要按键 i 来进入编辑模式。一旦您处于编辑模式下你可以在如下图所示文件中写入您的内容:

```
This is unix file....I created it for the first time.....
I'm going to save this content in this file.
```

一旦你做完上一步，请执行以下步骤:

- 按键 esc 退出编辑模式。
- 一起按两个键 Shift + ZZ 完全退出文件。

现在你会有一个已经创建好的叫 filename 的文件在当前目录中。

```
$ vi filename
$
```

编辑文件

您可以使用 vi 编辑器编辑现有的文件。我们将在一个单独的教程中详细介绍。但总之，您可以打开现有的文件，如下所示:

```
$ vi filename
```

一旦文件被打开，您将能在编辑模式下按键 i ，然后您可以如您所想的编辑文件。如果您想要在一个文件里左右移动首先您需要按下键 esc 退出编辑模式来，然后您可以使用下列键在文件内部移动:

- l 键移动到右侧。
- h 键移动到左侧。
- k 键移动到上面。
- j 键移动到下面。

使用上面的键您可以将光标放在任何您想要编辑的地方。一旦您定位好然后您可以使用 i 键来在编辑模式下编辑该文件。当您编辑完文件您可以按下 esc 键然后按下 Shift + ZZ 键来从文件完全的退出。

显示文件的内容

你可以使用 cat 命令来查看文件的内容。以下是简单的示例来查看上面创建文件的内容:

```
$ cat filename
This is unix file....I created it for the first time.....
I'm going to save this content in this file.
$
```

你可以通过按如下方式使用 -b 选项和 cat 命令显示行号:

```
$ cat -b filename
1  This is unix file....I created it for the first time.....
2  I'm going to save this content in this file.
$
```

统计文件中字数

你可以使用 wc 命令来获取一个文件中的总的行数，字数和字符数。以下是简单的示例来查看有关上面创建的文件的信息:

```
$ wc filename
2  19 103 filename
$
```

这里是所有四个列的细节:

1. 第一列: 代表文件中的行数。
2. 第二列: 代表文件中的字数。
3. 第三列: 代表文件中的字符数。这是文件的实际大小。
4. 第四列: 代表文件名。

在获取有关这些文件的信息的时候，你可以给多个文件。这里是简单的语法:

```
$ wc filename1 filename2 filename3
```

复制文件

要使用 cp 命令文件的副本。该命令的基本语法如下:

```
$ cp source_file destination_file
```

下面是创建一个已有文件 filename 的副本的例子。

```
$ cp filename copyfile
$
```

现在你会发现多了一个文件 copyfile 在您的当前目录。此文件与原始文件 filename 完全相同。

删除文件

若要更改文件的名称使用 mv 命令。其基本的语法是:

```
$ mv old_file new_file
```

下面是把现有文件 filename 重命名为 newfile 的示例:

```
$ mv filename newfile
$
```

mv 命令将现有文件完全移动到新的文件。所以在这种情况下你只能发现 newfile 在你当前的目录中。

删除文件

若要删除现有文件使用 rm 命令。其基本的语法是:

```
$ rm filename
```

警告: 要删除一个文件可能会很危险，因为它可能包含有用的信息。所以在使用此命令时要小心。这推荐使用 –i 选项和 rm 命令。

以下是完全删除现有文件 filename 的示例:

```
$ rm filename
$
```

您可以在一行中删除多个文件，如下所示:

```
$ rm filename1 filename2 filename3
$
```

标准 UNIX 流

在正常情况下每个 UNIX 程序在它启动时打开的三个流 (文件):

- stdin : 这指作为标准输入，关联文件描述符为 0。它也可以表示为 STDIN 。UNIX 程序默认从 STDIN 中读取。
- stdout : 这指作为标准输出，关联文件描述符为 1。它也可以表示为 STDOUT 。UNIX 程序默认从 STDOUT 中读取。
- stderr : 这指作为标准错误，关联文件描述符为 2。它也可以表示为 STDERR 。UNIX 程序会将所有的错误信息写入 STDERR。

目录

目录是一个文件，它的作用是存储文件的名称和相关的信息。所有的文件，无论是普通，特殊，或目录都包含在目录中。

UNIX 使用层次结构来组织文件和目录。这种结构通常被称为一个目录树。树上有一个根节点，斜杠字符 (/)，所有其他目录包含在它之下。

主目录

主目录是当您第一次登录时所在的目录。

您的大部分工作将在主目录及您自定义的子目录中完成。

在任意目录下执行以下命令可以随时切换到主目录：

```
$cd ~
$
```

在这里 ~ 表示主目录。如果您想要跳转至任何其他用户的主目录中，可以使用以下命令：

```
$cd ~username
$
```

跳转至您最近的目录中可以使用下列命令：

```
$cd -
$
```

绝对/相对路径名

目录采用分层方式组织，其顶部为根目录 (/)。层次结构内的任何文件的位置由其路径描述。

路径由 / 来分隔。路径名是绝对的如果它是描述与根的关系，所以绝对路径名的开头总是 /。

这些是绝对文件名的一些例子。

```
/etc/passwd
/users/sjones/chem/notes
/dev/rdsk/Os3
```

路径也可以是相对于你当前的工作目录。相对路径永远不会以 `/` 开始。相对于用户 amrood 的主目录，一些路径可能看起来像这样：

```
chem/notes
personal/res
```

在任何时候要确定你所在的文件系统层次结构时，请输入命令 pwd 打印当前工作目录：

```
$pwd
/user0/home/amrood

$
```

目录列表

要列出目录中的文件可以使用下面的语法：

```
$ls dirname
```

以下是示例，列出 `/usr/local` 目录中包含的所有文件：

```
$ls /usr/local

X11   bin  gimp   jikes   sbin
ace   doc  includelib share
atalk etc  info   man ami
```

创建目录

通过下面的命令创建目录:

```
$mkdir dirname
```

在这里，dirname 是您想要创建的目录的绝对或相对路径名。例如，命令：

```
$mkdir mydir
$
```

在当前目录中创建目录 mydir。这里是另一个示例：

```
$mkdir /tmp/test-dir
$
```

此命令在 /tmp 目录中创建目录 test-dir。命令 mkdir 不产生任何输出如果它成功创建请求的目录。

如果你在命令行上给出多个目录，mkdir 创建每个目录。例如：

```
$mkdir docs pub
$
```

在当前目录下创建目录 docs 和 pub 。

创建父目录

有时当你想要创建一个目录，其父目录可能不存在。在这种情况下，mkdir 发出一个错误消息，如下所示：

```
$mkdir /tmp/amrood/test
mkdir: Failed to make directory "/tmp/amrood/test";
No such file or directory
$
```

在这种情况下，您可以指定 mkdir 命令的 -p 选项。它为您创建所有必要的目录。例如：

```
$mkdir -p /tmp/amrood/test
$
```

上面的命令创建所需的父目录。

删除目录

可以按如下方式使用 rmdir 命令删除目录：

```
$rmdir dirname
$
```

注意: 删除目录时请确保它是空的，这意味着不应该在这个目录里有任何文件或子目录。

您可以一次创建多个目录如下：

```
$rmdir dirname1 dirname2 dirname3
$
```

上面的命令删除目录 dirname1、dirname2 和 dirname2，前提是它们是空的。如果成功删除，rmdir 命令不生成任何输出。

更改目录

你可以使用 cd 命令来做比更改主目录更多的事：你可以使用它来跳转到任何目录，其参数为一个有效的绝对或相对路径。语法如下所示:

```
$cd dirname
$
```

在这里，dirname 是你想要跳转到的目录的名称。例如，命令:

```
$cd /usr/local/bin
$
```

更改目录 `/usr/local/bin` 。从该目录，您可以使用下面的相对路径跳转到 `/usr/home/amrood` 目录:

```
$cd ../../home/amrood
$
```

重命名目录

mv (move) 命令也可以用于重命名目录。语法如下所示:

```
$mv olddir newdir
$
```

您可以重命名目录 mydir 为 yourdir，如下所示:

```
$mv mydir yourdir
$
```

目录 . (点) 和 .. (点点)

文件名 `.` (点) 表示当前的工作目录；和文件名 `..` (点点) 代表当前工作目录的上一级，通常被称为父目录。

如果我们输入要显示的当前工作目录文件的列表，使用 `-a` 选项列出所有的文件与 `-l` 选项提供长列表，这是结果。

```
$ls -la
drwxrwxr-x4teacher   class   2048  Jul 16 17.56 .
drwxr-xr-x60   root  1536  Jul 13 14:18 ..
----------1teacher   class   4210  May 1 08:27 .profile
```

```
-rwxr-xr-x1teacher   class   1948  May 12 13:42 memo
$
```

文件权限

文件所有权是 UNIX 的一个重要的组成部分，提供了一种安全的方法来存储文件。在 UNIX 中每个文件有以下属性:

- 所有者权限：所有者的权限决定文件的所有者可以对文件执行的操作。
- 组权限：组权限决定了属于该组的成员对他所拥有的文件能够执行的操作。
- 其他人权限：其他人权限表示其他所有人对于该文件能够进行的操作。

权限表示符

当使用 ls –l 命令的时候，会将与文件相关的各种权限展示出来，如下：

```
$ls –l /home/amrood
–rwxr–xr––  1 amrood   users 1024  Nov 2 00:10  myfile
drwxr–xr––– 1 amrood   users 1024  Nov 2 00:10  mydir
```

输出的第一列表示的是与文件或者目录相关的访问模式或者权限。

权限被分为三组，组中的每个位置代表一个特定的权限，这个顺序是：读(r)、写(w)和执行(x):

- 前三个字符 (2–4) 表示文件的所有者的权限。例如 `–rwxr–xr––` 代表，文件的所有者拥有读 (r)、写 (w) 和执行 (x) 的权限。
- 第二组的三个字符 (5–7) 包含了该文件所属组的权限。例如 `–rwxr–xr––` 表示了所属组拥有读 (r) 和执行 (x) 的权限，但没有写权限。
- 最后一组三个字符 (8–10) 代表其他人的权限。例如 `–rwxr–xr––` 代表其他人只有读 (r) 的权限。

文件访问模式

文件的权限是 UNIX 系统安全性的第一道防线。UNIX 权限的基本组成部分是**读，写，执行**权限，如下所述:

1. **读：**分配对文件的内容进行读取和查看文件的权限。
2. **写：**分配对文件的内容进行修改或者删除的权限。
3. **执行：**允许用户将该文件作为一个程序进行执行的权限。

目录访问模式

目录访问模式采用和其他文件用相同的方式组织。但是有一些差异，还是需要提到:

1. **读**：访问目录意味着用户可以读取目录下的内容。用户可以查看目录内的文件名。
2. **写**：这个权限意味着用户可以在目录下面删除或者新建文件。
3. **执行**：执行一个目录并没有真正的意义，因此将它当作可以遍历目录的权限。

用户为了执行 ls 或者 cd 命令就必须先访问了 bin 目录。

改变权限

改变文件或目录的权限，您可以使用 chmod(change mode)命令。有两种方法可以使用 chmod：符号模式和绝对模式。

符号模式中使用 chmod

对于初学者来说使用符号模式是最简单的来修改文件或目录的权限方法。可以用下表中的符号来添加、删除或指定你想要设置的权限。

Chmod 操作符	描述
+	给文件或者目录添加指定的权限。
−	删除文件或者目录的权限。
=	设置指定的权限。

如下是以 testfile 文件为示例。对 testfile 文件运行 `ls −l` 就会像下面一样显示文件的权限：

```
$ls −l testfile
−rwxrwxr−− 1 amrood users 1024 Nov 2 00:10 testfile
```

接下来将前面表格中的 chmod 命令都对 testfile 运行一下，下面的是在 `ls −l` 运行之后，你可以看到文件权限的改变：

```
$chmod o+wx testfile
$ls −l testfile
−rwxrwxrwx 1 amrood users 1024 Nov 2 00:10 testfile
$chmod u−x testfile
$ls −l testfile
```

```
-rw-rwxrwx  1 amrood   users 1024  Nov 2 00:10  testfile
$chmod g=rx testfile
$ls -l testfile
-rw-r-xrwx  1 amrood   users 1024  Nov 2 00:10  testfile
```

下面将展示如何将上面的命令组合成一行：

```
$chmod o+wx,u-x,g=rx testfile
$ls -l testfile
-rw-r-xrwx  1 amrood   users 1024  Nov 2 00:10  testfile
```

chmod 命令中使用绝对权限

用chmod命令修改权限的第二种方法，是使用一个数字来指定文件的一些列权限。

每个权限被分配了一个数值，如下表所示， 并且给每个权限集的总和提供了一个数值。

数值	权限八进制表示	参照
0	没有权限	---
1	可执行的权限	--x
2	写权限	-w-
3	执行和写权限: 1 (执行) + 2 (写) = 3	-wx
4	读取权限	r--
5	读取和执行权限: 4 (读取) + 1 (执行) = 5	r-x
6	读取和写权限: 4 (读) + 2 (写) = 6	rw-
7	所有权限: 4 (读) + 2 (写) + 1 (执行) = 7	rwx

如下是针对 testfile 文件的示例。运行 ls -l 命令会显示与该文件相关的权限如下：

```
$ls -l testfile
-rwxrwxr--  1 amrood   users 1024  Nov 2 00:10  testfile
```

对 testfile 运行上面表格中每个 chmod 示例命令，如下是在 ls -l 之后的，你可以从下面命令中看出权限的改变情况：

```
$ chmod 755 testfile
$ls -l testfile
-rwxr-xr-x  1 amrood   users 1024  Nov 2 00:10  testfile
$chmod 743 testfile
$ls -l testfile
-rwxr---wx  1 amrood   users 1024  Nov 2 00:10  testfile
$chmod 043 testfile
```

```
$ls -l testfile
----r---wx  1 amrood   users 1024  Nov 2 00:10  testfile
```

改变所有者和所属组

在 UNIX 上创建一个帐户时，系统会给每个用户分配一个所有者 ID 和组 ID。所有上面提到的权限也会基于所有者和组进行分配。

如下的两个命令可以改变一个文件的所有者和组：

1. chown：chown 表示的是 “change owner”，并且它是被用来改变一个文件的所有者。
2. chgrp：chgrp 表示的是 “change group”，并且它是被用来一个文件所属的组。

改变所有者关系

chown 命令用来改变一个文件的所有者，它的基本语法如下：

```
$ chown user filelist
```

上面命令中的 user 既可以是系统中的用户名，也可以是系统中用户的 id(uid)。 示例：

```
$ chown amrood testfile
$
```

改变 testfile 文件的所有者为 amrood 用户。

注意：超级用户，root 用户，拥有不受限制的权限，能够更改所有文件的所有者，但是普通用户只能修改他们所拥有的文件的所有者。

改变组关系

chgrp 命令被用来修改文件所属的组。基本语法如下：

```
$ chgrp group filelist
```

上面命令中的 group 既可以是系统中存在的组的名称，也可以是系统中存在的组的 ID(GID)。

示例：

```
$ chgrp special testfile
$
```

改变给定的文件的组为 special 组。

SUID 和 SGID 文件权限

通常执行一个命令时，为了完成该任务它必须拥有某些特殊的权限。

举一个例子，当你使用 passwd 命令改变了你的密码后，您的新密码存储在文件 `/etc/shadow` 中。

作为一个普通用户，出于安全原因你没有读或写访问这个文件的权限，但是当你改变你的密码时，你需要拥有对这个文件写权限。这意味着 passwd 程序必须给你额外的权限，以便您可以编写文件 `/etc/shadow`，也就是需要额外的权限。

通过设置用户 ID(SUID)和组 ID(SGID) 位可以给程序额外的权限。

当您执行一个启用了 SUID 的程序，你继承了程序所有者的权限。启动改程序的用户就可以不用设置 SUID 直接运行该程序。

这对于 SGID 同样是适用的。通常程序是按组的权限进行执行，除非你的组改变了该程序所属组的拥有者。

如果 SUID 和 SGID 权限是可用的，它们将会以小写的 “s” 出现。SUID 的 “s” 位通常位于权限中所有者执行权限的旁边。如下：

```
$ ls -l /usr/bin/passwd
-r-sr-xr-x 1   root   bin  19031 Feb 7 13:47  /usr/bin/passwd*
$
```

上面的显示了 SUID 被设置了并且该命令被 root 用户所拥有。在使用大写字母 S 而不是小写字母表示执行位没有设置。

如果对一个目录设置了防删除位(sticky bit)，那么只有你是如下任意一种用户时你才可以删除该文件：

- 该目录的拥有者
- 被删除文件的拥有者
- 超级用户，root 用户

你可以使用如下的方式设置任何目录的 SUID 和 SGID 位。

```
$ chmod ug+s dirname
$ ls -l
drwsr-sr-x 2 root root  4096 Jun 19 06:45 dirname
$
```

环境

UNIX 中一个重要的概念是环境，它是由环境变量所定义。一些环境变量是由系统设置，有一些是由用户设置的，还有一些是被 Shell，或任何会加载另一个程序的程序所设置。

一个变量是由一个字符组成的串，并且我们会给它赋值。给变量赋的值可以是一个数字，文本，文件名，设备或任何其他类型的数据。

例如，首先，我们设置一个名称为 TEST 的变量，然后我们使用 echo 命令查看它的值:

```
$TEST="Unix Programming"
$echo $TEST
Unix Programming
```

注意，设置环境变量不使用 `$` 符号，但在访问他们的时候，我们使用 `$` 符号作为前缀。这些变量保存它们的值，直到我们退出 shell。

当你登录到系统，shell 经过初始化阶段，在该阶段会设置各种环境变量。这通常会涉及到两步的处理过程，shell 会读取以下文件:

- `/etc/profile`
- profile

处理流程如下:

1. shell 程序检查 /etc/profile 文件是否存在。
2. 如果该文件存在，shell 程序会读取该文件。否则，就会跳过该文件。同时也不会显示任何错误信息。
3. shell 程序检查 .profile 文件是否在你的根目录下面存在。您的根目录就是你在登录之后进入的目录。
4. 如果该文件存在，shell 程序就会读取它。否则，shell 程序跳过它，不会显示任何错误信息。

一旦这两个文件读取完成，shell 显示一个等待输入命令:

```
  $
```

这是提示,在它后面你可以输入命令来执行。

注意：Shell 初始化的详细过程通常利用的是 Bourne Shell，但是其他的一些文件处理是利用 bash 和 ksh shell 程序。

.profile 文件

/etc/profile 文件是由 UNIX 的系统管理员维护的，并且该文件中包含了 Shell 初始化的信息，这个信息可以被任何系统中的任何用户查看。

如果你有对 .profile 文件操作的权限，那么你就可以在这个文件中添加你想要的尽可能多的定制 Shell 信息。

- 你使用的终止符的类型
- 命令存在的一系列文件的列表
- 一些列的变量设置你的终端显示的效果

你可以在你的根目录下面查看 .profile 文件。利用 vi 编辑器打开它，查看其中设置的所有环境变量。

设置终结符的类型

通常您所使用的终端的类型由 login 或 getty 程序自动配置。有时，自动配置过程会推测你的终端类型是不对的。

如果您的终端设置错误，命令的输出可能看起来很奇怪，或者你可能无法与 Shell 正常交互。

确保这不是这种情况，大多数用户的终端最少相同的特性如下:

```
$TERM=vt 100
$
```

设置 PATH 变量

当你在命令提示符下输入任何命令，Shell 只有确定了命令所在的目录才能执行命令。

Shell 是在环境变量 PATH 中寻找命令所在的目录。通常，它设置如下:

```
$PATH=/bin:/usr/bin
$
```

这里的每一个由冒号，：，分开的实体是目录。如果你请求 Shell 执行一个命令，但是它不能在 PATH 环境变量中找到任何命令所在的路径,这时会出现一个类似如下的消息:

```
$hello
hello: not found
$
```

还有类似于 PS1 和 PS2 这样的变量，将会在下一节说明。

PS1 和 PS2 变量

shell 显示给你的命令提示符存储在变量 PS1 中。你可以改变这个变量成任何你想要的字符。只要你改变它，它就会从你改变后开始起作用。

例如，如果你输入如下的命令:

```
$PS1='=>'
=>
=>
=>
```

你的提示输入符将会变成 =>。设置 PS1 的值，让它显示工作目录，输入如下的命令:

```
=>PS1="[\u@\h \w]\$"
[root@ip-72-167-112-17 /var/www/tutorialspoint/unix]$
[root@ip-72-167-112-17 /var/www/tutorialspoint/unix]$
```

该命令的结果是,显示用户的用户名、机器名称(主机名)，和工作目录。

有相当多的转义序列，可以用作 PS1 的参数，尽量让自己只关注最关键的部分,不要让下面的信息对你造成过多的压力。

转义序列	描述
\t	将当前的时间表示成 HH:MM:SS 的形式
\d	将当前的日期表示成 周 月 日
\n	新的一行。
\s	当前的环境变量。
\W	工作目录。
\w	工作目录的完整路径。
\u	当前用户的用户名。
\h	当前机器的主机名称。
\#	当前命令的编号。每输入一条命令编号加 1。
\$	如果有效的 UID 是 0(也就是说，如果你以 root 用户进行登录)，命令提示符会变成 #，否则，提示符是 $。

你可以通过修改 .profile 文件，在每次登录的时候进行上面的那些转换。这样每次登录就会自动的改变 PS1 的值。

当你输入一个不完整的命令是，shell 将再次显示一个命令输入符，等待你再次完成命令并回车。

默认二级提示 >(大于号)，但可以改变通过设置 PS2 变量进行修改：

下面的示例使用默认的二级提示:

```
$ echo "this is a
> test"
this is a
test
$
```

下面是一个通过重新定义 PS2 变量自定义输入符的示例:

```
$ PS2="secondary prompt->"
$ echo "this is a
secondary prompt->test"
this is a
test
$
```

环境变量

以下是部分重要的环境变量的列表。这些变量将按照上面提到的方式被设置和访问:

变量	描述
DISPLAY	包含显示设备的标识符，默认情况下它的值是 X11。
HOME	表明当前用户的根目录，默认的参数中会内置 cd 命令。
IFS	表明系统内部所使用字段分隔符，它通常用在解析器分割单词中。
LANG	LANG 扩展系统默认的语言：LC_ALL 可以用来覆盖这个变量。例如，如果它的值是 pt_BR，那么系统的语言就被设置成(Brazillian)Portuguese 和地区被设置成 Brazil。
LD_LIBRARY_PATH	许多 UNIX 系统动态链接器，包含以冒号分隔的目录列表，在执行后，动态连接器构建过程图像过程中，在搜索其他目录之前，先搜索共享对象。
PATH	命令的搜索路径。它是由冒号分隔开一系列目录，也就是 shell 寻找命令所在的目录。
PWD	当前的工作目录，由 cd 命令设置的。
RANDOM	每次被引用的时候就会生成一个 0 到 32,767 范围内的一个随机整数。

SHLVL	每次一个 bash 实例被启动这个值就会加 1。这个变量对于决定内置的退出命令是否终止当前会话是很有用的。
TERM	显示类型。
TZ	时间区域。它能被赋值为 GMT，AST 等。
UID	数值类型标识当前用户，它在 shell 启动的时候被初始化。

如下是几个简单的例子显示几个环境变量：

```
$ echo $HOME
/root
]$ echo $DISPLAY
$ echo $TERM
xterm
$ echo $PATH
/usr/local/bin:/bin:/usr/bin:/home/amrood/bin:/usr/local/bin
$
```

实用工具

到现在为止你肯定对 UNIX 系统已经有了一些大概的理解和一些命令的基本使用方式。本教程将介绍一些非常基本的但重要的 UNIX 实用工具。

打印文件

在 UNIX 系统中，您打印一个文件之前，您可能想要重新格式化它调整它的边距，高亮显示一些单词等等。大多数文件也可以打印而不用重新格式化，但未经处理的打印可能不那么好看。

UNIX 系统的许多版本中都包含了两个强大的文本格式化命令，nroff 和 troff。他们不包含在本教程中，但是你可以在在网上查到很多关于讲解这两个命令的使用方式的资料。

pr 命令

pr 命令可以对终端显示屏上或者打印机上显示的文件进行小幅度的格式化。例如，如果在你的文件中有一长串名字，你可以将它格式化成两列或者多列在屏幕上显示。

如下是 pr 命令的语法示例：

```
pr option(s) filename(s)
```

pr 命令仅仅只是格式化显示在屏幕上或者打印的副本文件，它不会修改源文件。如下的列表显示一些 `pr` 命令中可选的参数：

操作	描述
-k	产生 k 列的输出
-d	将输出用两个空格隔开(并不是所有的 pr 版本适用)。
-h "header"	将下一个项目作为头部信息。
-t	去掉打印中的头部和上/下边距。
-l PAGE_LENGTH	设置一页存放的数据行数为 PAGE_LENGTH(66)。默认的文本行数为 56 行。
-o MARGIN	设置每行之间的间隔为 MARGIN(0) 个空格。
-w PAGE_WIDTH	设置页一行的字符个数为 PAGE_WIDTG(72) 个字符。这个参数仅仅对多文本列输出可用。

在使用 pr 命令之前，如下是查看 food 文件的内容：

```
$cat food
Sweet Tooth
Bangkok Wok
Mandalay
Afghani Cuisine
Isle of Java
Big Apple Deli
Sushi and Sashimi
Tio Pepe's Peppers
.....
$
```

接着让我们利用 pr 命令将输出变成两列，同时头部显示 *Restaruants*:

```
$pr -2 -h "Restaurants" food
Nov  7  9:58 1997  Restaurants   Page 1

Sweet Tooth            Isle of Java
Bangkok Wok            Big Apple Deli
Mandalay               Sushi and Sashimi
Afghani Cuisine        Tio Pepe's Peppers
.....
$
```

lp 和 lpr 命令

命令 lp 或 lpr 将文件打印到纸上，而不是在屏幕上显示。一旦你准备使用 pr 命令格式化文本，您可以使用这些命令在任何与你电脑连接的打印机上打印你的文件。

您的系统管理员可能已经建立了一个站点作为默认打印机。为了在默认的打印机上打印一个文件命名 food 的文件，你可以使用 lp 或 lpr 命令，如下示例:

```
$lp food
request id is laserp-525  (1 file)
$
```

lp 命令显示了打印机的 ID，您可以使用它来取消打印作业或检查它的状态。

- 如果您正在使用 lp 命令，您可以使用 -nNum 选项参数设置打印副本的份数。对于 lpr 命令,您也可以使用参数 -Num 起到相同的作用。
- 如果有多个打印机连接到共享网络中，对于 lp 命令你可以使用 -dprinter 参数来选择你想使用的打印机，对于 lpr 命令你可以使用 -Pprinter 参数达到相同的效果。这里 printer 值得是打印机的名称。

lpstat 和 lpg 命令

lpstat 命令显示在打印机队列中的作业:请求的 ID，所有者，文件大小，当打印任务被发送给打印机的时候，请求的状态同样也发送了给打印机。

如果你想看到所有输出请求而不仅仅是你自己的，你可以使用 `pstat -o` 命令。请求会按照他们将会被打印的顺序显示出来:

```
$lpstat -o
laserp-573  john  128865  Nov 7  11:27  on laserp
laserp-574  grace  82744  Nov 7  11:28
laserp-575  john   23347  Nov 7  11:35
$
```

lpg 显示的信息与 lpstat -o 显示的稍微有些不同:

```
$lpq
laserp is ready and printing
Rank   Owner      Job  Files                 Total Size
active john       573  report.ps             128865 bytes
1st    grace      574  ch03.ps ch04.ps        82744 bytes
2nd    john       575  standard input         23347 bytes
$
```

在第一行显示打印机状态。如果打印机是禁用或纸用完了，你可以在第一行看到不同的信息。

cancel 和 lprm 命令

cancel 命令终止 lp 命令发出的打印请求。lprm 命令终止 lpr 发出的打印请求。您可以指定打印机的 ID (由 lp 或 lpq 发出的请求)或名称来终止打印任务。

```
$cancel laserp-575
request "laserp-575" cancelled
$
```

为了取消当前正在打印的任务，可以忽视它的 ID，仅仅输入 cancel 命令和打印机的名称即可:

```
$cancel laserp
request "laserp-573" cancelled
$
```

lprm 命令将取消活动的工作，如果它属于你。否则，你可以使用工作的编号作为该命令的参数，或者使用破折号(-)删除你所有的工作:

```
$lprm 575
dfA575diamond dequeued
cfA575diamond dequeued
$
```

lprm 命令将会告诉你从打印机队列中删除的任务的文件名。

发送邮件

您可以使用 UNIX 邮件命令发送和接收邮件。如下是发送电子邮件的语法:

```
$mail [-s subject] [-c cc-addr] [-b bcc-addr] to-addr
```

如下是 mail 命令中重要的参数:

参数	描述
-s	在命令行中指定邮件的主题。
-c	给列表中的用户发送副本。用户列表是由逗号分开的用户名列表。
-b	发送密文副本给列表中的用户。各个列表由逗号分隔开。

下面是示例发送测试消息到 admin@yahoo.com。

```
$mail -s "Test Message" admin@yahoo.com
```

接下来该输入你的消息部分，消息输入部分是在行首的 “control-D" 的之后。如果想要结束，你仅仅只需要输入一个点类型(.)，如下:

```
Hi,
This is a test
.
Cc:
```

你可以发送一个完整的文件通过利用重定向 `<` 操作符，如下:

```
$mail -s "Report 05/06/07" admin@yahoo.com < demo.txt
```

为了检查是否有收到邮件，在 UNIX 系统中你可以简单的输入如下的命令:

```
$mail
no email
```

管道和过滤器

你可以连接两个命令在一起，这样一个程序的输出就可以作为下一个程序的输入。两个或两个以上的命令以这种方式连接形成一个管道。

为了形成管道，在同一行中利用一个竖线 (|) 将两个命令隔开。

如果一个程序将另一个程序的输出作为输入数据，接着对输入的数据执行一些操作，并将结果写入标准输出，它就称为一个过滤器。

grep 命令

grep 程序用固定的模式搜索一个文件或多个文件。它的语法是:

```
$grep pattern file(s)
```

“grep” 这个名字来源于 ed(UNIX 行编辑器)命令， g/re/p 这意味着“利用正则表达式进行全局搜索并打印所有包含它的行。”

正则表达式是一些纯文本 (例如，一个词) 和 / 或特殊字符，它被用于模式匹配。

最简单的 grep 使用就是匹配由一个词组成的模式。它可可以管道中使用，因此只有那些输入行中包含一个给定的字符串，才会被发送到标准输出。如果你不指定 grep 读取的文件名，它读取标准输入，这也是所有过滤程序工作的方式:

```
$ls -l | grep "Aug"
-rw-rw-rw-  1 john  doc    11008 Aug  6 14:10 ch02
-rw-rw-rw-  1 john  doc     8515 Aug  6 15:30 ch07
-rw-rw-r--  1 john  doc     2488 Aug 15 10:51 intro
-rw-rw-r--  1 carol doc     1605 Aug 23 07:35 macros
$
```

如下是各种可选的参数，你可以在 grep 命令中进行使用:

参数	描述
-v	打印所有没有匹配的行。
-n	打印所有成功匹配的行和行号。
-l	打印匹配的文件名和匹配的行("l"来自字母 letter)。
-c	仅仅打印成功匹配到行的个数。
-i	同时匹配大小写。

接下来，让我们使用一个正则表达式,它让 grep 命令找到包含 “carol” 字母的行，紧随其后的可以是零个或多个字母，正则表达式中表示方法是 “.*”)，之后接着是 “Aug” 字符。

如下是使用 -i 参数，表示对字母大小写不敏感：

```
$ls -l | grep -i "carol.*aug"
-rw-rw-r--   1 carol doc      1605 Aug 23 07:35 macros
$
```

sort 命令

sort 命令是按字母顺序或者数字顺序对行文本进行排序。下面的示例是对 food 文件中的文本进行排序:

```
$sort food
Afghani Cuisine
Bangkok Wok
Big Apple Deli
Isle of Java
Mandalay
Sushi and Sashimi
Sweet Tooth
Tio Pepe's Peppers
$
```

sort 命令默认是按字母顺序进行排序。有很多可选参数，可以控制排序：

参数	描述
-n	数值顺序进行排序 (例如: 10 将会被排到 2 之后)，忽略空格和 tab 符。
-r	将排序的顺序反转。
-f	将大小写排在一起。
+x	排序的时候忽略第一个 x 字段。

两个或者两个以上的命令就可以形成管道。拿前面提到的 grep 命令为例，我们可以按照文件的大小进一步对 August 文件进行排序。

如下管道包含了 ls，grep，和 sort 命令：

```
$ls -l | grep "Aug" | sort +4n
-rw-rw-r--  1 carol doc      1605 Aug 23 07:35 macros
-rw-rw-r--  1 john  doc      2488 Aug 15 10:51 intro
-rw-rw-rw-  1 john  doc      8515 Aug  6 15:30 ch07
-rw-rw-rw-  1 john  doc     11008 Aug  6 14:10 ch02
$
```

上面的管道将会按照文件的大小对 August 目录下的文件进行排序，并将它们打印到终端屏幕。排序参数 `+4n` 会跳过 4 个字段(由空格分隔的字段)，接着在按照数值顺序对行进行排序。

pg 和 more 命令介绍

过长的输出通常会在您的屏幕上被压缩，但是如果你通过使用 more 或 pg 命令作为过滤器，知道屏幕显示满了文本之后才会停止。

假设你有一个很长的目录列表。为了让它容易阅读,我们就要对它进行排序，通过使用 more 命令对管道的输出进行处理:

```
$ls -l | grep "Aug" | sort +4n | more
-rw-rw-r--  1 carol doc      1605 Aug 23 07:35 macros
-rw-rw-r--  1 john  doc      2488 Aug 15 10:51 intro
-rw-rw-rw-  1 john  doc      8515 Aug  6 15:30 ch07
-rw-rw-r--  1 john  doc     14827 Aug  9 12:40 ch03
   .
   .
   .
-rw-rw-rw-  1 john  doc     16867 Aug  6 15:56 ch05
--More--(74%)
```

屏幕将会充满文本数据，这些文本是按照文件大小顺序的。在屏幕的底端是一个 more 命令，你可以敲入命令让屏幕滚动显示更多的数据。

当屏幕上显示完成的时候，你接着可以使用在讨论部分说的任何关于 more 程序的命令。

进程管理

如果用户在 UNIX 操作系统上执行了一个程序，那么操作系统会为这个程序创建一个运行它的特定环境。这个环境包含系统运行该程序所需的一切资源，使得好像系统中没有运行其他程序一样。

用户如果在 UNIX 操作系统中输入一个指令，操作系统就会创建（启动）一个相应的进程。比如，如果用户希望使用 ls 指令来列出目录内的文件列表时，系统就启动了一个进程来完成这个任务。简单的说，进程就是一个可执行程序的实例。

操作系统通过一个 5 位的 ID 号码来追踪进程，这个 ID 号码 通常被称为pid 或进程 ID。操作系统中的每一个进程都有唯一 pid。

由于所有的进程 ID 是循环使用的，所以 pid 是会重复的。不过，在操作系统中，不存在两个进程拥有统一进程 ID 的情况。

创建进程

如果用户创建一个进程（执行一个指令），那么可以两种方式来运行它。

- 前台进程
- 后台进程

前台进程

默认情况下，任何一个用户创建的进程都会在前端执行。该进程可以从键盘获取输入信息并且可以将执行结果反馈到显示器上。

我们可以使用 ls 指令来观察这个过程。如果用户希望列出当前目录下的所有文件，用户需要在终端命令行键入如下指令：

```
$ls ch*.doc
```

这个指令将会显示所有文件名称以 ch 开头，以 `.doc` 结尾的文件。

```
ch01-1.doc   ch010.doc  ch02.docch03-2.doc
ch04-1.doc   ch040.doc  ch05.docch06-2.doc
ch01-2.doc   ch02-1.doc
```

该指令对应的进程在前台进行，输出结果直接显示在显示屏上，如果 ls 执行需要获取输入，那么该进程会等待来自键盘的输入信息。

当程序在前台执行的时候，用户无法执行其他的指令（创建其他的进程），这是因为系统会提示其他进程无法创建直到当前进程执行完毕。

后台进程

后台进程不需要键盘输入的信息就可以执行。如果后台进程需要键盘等外设的输入信息的话，那么它会等待。

后台进程的优点是用户可以执行其他的指令。用户此时不需要等待进程结束就可以执行其他的进程。

开启一个后台进程的最简单的方法就是在指令的末尾添加 `&` 标识符。

```
$ls ch*.doc &
```

这个指令也会示所有文件名称以 ch 开头，以 `.doc` 结尾的文件。

```
ch01-1.doc   ch010.doc  ch02.docch03-2.doc
ch04-1.doc   ch040.doc  ch05.docch06-2.doc
ch01-2.doc   ch02-1.doc
```

如果这里的 ls 指令希望得到输入信息，它会转换为停止状态直到用户将他转到前台并获取到从键盘来的输入信息。

第一行显示了后台进程的信息——作业号和进程 ID，用户需要使用作业号来完成前景和后台之间的切换。

如果用户按下回车键，可以看到如下信息：

```
[1]  +  Done ls ch*.doc &
$
```

第一行表示 ls 指令的后台进程已经成功执行。第二行提示可以执行其他的指令。

列出处于执行状态的进程

可以使用 ps 指令来显示当前操作系统处于执行状态的进程，结果如下：

```
$ps
PID    TTY    TIMECMD

18358  ttyp3  00:00:00sh
```

```
18361   ttyp3   00:01:31abiword
18789   ttyp3   00:00:00ps
```

使用 ps 指令时，通常会选择 –f 选项。该选项可以显示更为详细的内容。

```
$ps –f
UID     PID  PPID C STIME      TTY     TIME CMD
amrood   6738 3662 0 10:23:03    pts/6   0:00 first_one
amrood   6739 3662 0 10:22:54    pts/6   0:00 second_one
amrood   3662 3657 0 08:10:53    pts/6   0:00 –ksh
amrood   6892 3662 4 10:51:50    pts/6   0:00 ps –f
```

下面是 –f 选项列出内容的的相关解释。

列名称	意义
UID	执行该进程的用户ID
PID	进程编号
PPID	该进程的父进程编号
C	该进程所在的CPU利用率
STIME	进程执行时间
TTY	进程相关的终端类型
TIME	进程所占用的CPU时间
CMD	创建该进程的指令

如下是其他配合 ps 指令的选项：

选项	意义
–a	显示所有用户的信息
–x	显示关于没有终端的进程的信息
–u	显示类似 –f 的其他附加信息
–e	显示扩展信息

停止进程执行

用户可以使用多种方式来停止一个进程。通常情况下，可以通过终端指令来完成，比如，同时按下 CTRL+C 键就可以停止当前执行的指令。这种方式仅在程序以前台的方式执行的情况下起作用。

如果一个进程以后台的方式在执行，那么首先用户需要通过 ps 指令来获取它的作业编号，然后用户可以使用 kill 指令来杀掉该进程。如下：

```
$ps –f
UID  PID  PPID C STIMETTY   TIME CMD
```

```
amrood   6738 3662 0 10:23:03 pts/6 0:00 first_one
amrood   6739 3662 0 10:22:54 pts/6 0:00 second_one
amrood   3662 3657 0 08:10:53 pts/6 0:00 -ksh
amrood   6892 3662 4 10:51:50 pts/6 0:00 ps -f
$kill 6738
Terminated
```

这里的 kill 指令收终止指令为 first_one 对应的进程。如果某个进程无视常规的 kill 指令。用户可以使用 kill -9 后跟进程编号的方式来终止指令，如下：

```
$kill -9 6738
Terminated
```

父进程和子进程

UNIX 系统中的每一个进程都有两个 ID 号码：进程 ID（pid）和父进程 ID（ppid）.系统中的每一个用户进程均有父进程。

大部分使用 shell 执行的指令均有他们各自的父进程。使用 `ps -f` 指令可以显示出每个进程相对应的进程 ID和其父进程 ID.

僵尸进程和孤儿进程

通常情况下，当某个子进程被杀掉后，其父进程会被 SIGCHILD 信号通知。然后，该父进程会做一些必要的操作或者启动一个新的子进程。然而，有时候是父进程先于子进程被杀掉。这种情况下，被称为“所有进程的父进程”的 init 进程就称为该子进程的父进程。这些子进程也称为孤儿进程。

当某个进程被杀掉后，ps 指令列出的列表里显示该进程标志位 Z 状态。它就是一个僵尸进程。该进程处于死亡状态并且不会被再次使用。这些进程不同于孤儿进程。他们是已经完成任务的进程，但是仍在进程表中留有一个入口。

守护进程

守护进程是操作系统相关的后台进程，他们通常以 root 权限执行，并且会相应其他进程的请求。

守护进程没有控制终端。它也不能打开 `/dev/tty` 。如果用户使用"ps -ef" 指令来查看 tty 域，所有的守护进程在该域都会显示? 。

更详细的来讲，守护进程就是执行在后台的进程，且它会等待某个事件的发生，从而相应该事件。比如打印机守护进程一直在等待打印的指令。

如果用户的某个程序需要长时间的执行，那么可以将它设计为守护进程的启动方式。

top 指令

top 指令是用于显示以不同条件排序进程的指令。

它是一个频繁更新的交互式诊断工具，会动态的显示如下和相关进程的如下信息：物理内存、虚拟内存、CPU 利用率、负载率。

下面是一个简单的例子来执行 top 指令，且查看不同继承的CPU使用率。

```
$top
```

作业编号与进程号

后台的且阻塞状态的进程通常使用作业编号来维护。该作业编号不同于进程编号。

此外，作业可以包含多个进程，这些进程可以串行执行，也可以并行执行，所以使用作业编号比跟踪单个的进程会更加简单。

通信工具

如果用户在分布式环境下工作，那么用户就需要与远程用户通信，用户也需要远程方式访问 UNIX 主机。

如下是一些 UNIX 操作系统中的实用工具，这些工具专用于分布式环境下的用户间的网络通信。

ping 工具

ping 指令会发送一个应答请求到网络中某个主机。该指令主要用于检测远端主机是否可以正常通信。

ping 指令可以用于如下用途：

- 追踪并区分硬件或软件的问题。
- 确定网络和远端主机的状态。
- 测试、测量或网络管理。

语法

如下是使用 ping 指令的语法：

```
$ping hostname or ip-address
```

上述指定会持续打印响应信息。用户可以同时按下 CTRL+C 按键来结束信息的打印.

例子

下面是检测网络中某主机是否可达的例子：

```
$ping google.com
PING google.com (74.125.67.100) 56(84) bytes of data.
64 bytes from 74.125.67.100: icmp_seq=1 ttl=54 time=39.4 ms
64 bytes from 74.125.67.100: icmp_seq=2 ttl=54 time=39.9 ms
64 bytes from 74.125.67.100: icmp_seq=3 ttl=54 time=39.3 ms
64 bytes from 74.125.67.100: icmp_seq=4 ttl=54 time=39.1 ms
64 bytes from 74.125.67.100: icmp_seq=5 ttl=54 time=38.8 ms
--- google.com ping statistics ---
22 packets transmitted, 22 received, 0% packet loss, time 21017ms
```

```
rtt min/avg/max/mdev = 38.867/39.334/39.900/0.396 ms
$
```

如果某个主机不可达，那么会显示如下信息：

```
$ping giiiiigle.com
ping: unknown host giiiiigle.com
$
```

FTP 工具

FTP 就是文件传输协议（File Transter protocol）的简称。使用该工具可以帮助用户在主机间上传或下载文件。

FTP 工具拥有自己的 UNIX 指令，可以完成如下任务：

- 链接并登陆到远程主机。
- 浏览目录。
- 列出目录内容。
- 上传或下载文件。
- 按照 ascii、ebcdic 或 binary 方式传输文件。

语法

如下是使用 ftp 指令的语法：

```
$ftp hostname or ip-address
```

上述指令会触发一个输入账号和密码的登陆界面。如果用户输入的账号和密码认证通过，则用户可以访问相应输入账户的根目录，然后就可以执行多种操作。

下面是一些常用操作：

指令	描述
put filename	从本地往远程服务器上传文件
get filename	从远程服务器往本地下载文件
mput file list	从本地往远程服务器批量上传文件
mget file list	从远程服务器往本地批量下载文件
prompt off	关闭文件提醒,在 mput 与 mget 时不会每操作一个文件就询问一次。

prompt on	开启文件提醒
dir	列出远程服务器上当前目录下的所有文件
cd dirname	切换本地主机上的目录到指定目录下
lcd dirname	切换远程服务器上的目录到指定目录下
quit	注销当前登陆

需要注意的是，上传和下载文件时的本地主机目录都是当前目录。如果用户希望上传或下载文件的目录为特定的目录，那么用户需要先将当前目录切换到指定目录后再进行上传或下载操作。

例子

下面是一些关于 ftp 操作的例子：

```
$ftp amrood.com
Connected to amrood.com.
220 amrood.com FTP server (Ver 4.9 Thu Sep 2 20:35:07 CDT 2009)
Name (amrood.com:amrood): amrood
331 Password required for amrood.
Password:
230 User amrood logged in.
ftp> dir
200 PORT command successful.
150 Opening data connection for /bin/ls.
total 1464
drwxr-sr-x   3 amrood   group   1024 Mar 11 20:04 Mail
drwxr-sr-x   2 amrood   group   1536 Mar  3 18:07 Misc
drwxr-sr-x   5 amrood   group512 Dec  7 10:59 OldStuff
drwxr-sr-x   2 amrood   group   1024 Mar 11 15:24 bin
drwxr-sr-x   5 amrood   group   3072 Mar 13 16:10 mpl
-rw-r--r--   1 amrood   group 209671 Mar 15 10:57 myfile.out
drwxr-sr-x   3 amrood   group512 Jan  5 13:32 public
drwxr-sr-x   3 amrood   group512 Feb 10 10:17 pvm3
226 Transfer complete.
ftp> cd mpl
250 CWD command successful.
ftp> dir
200 PORT command successful.
150 Opening data connection for /bin/ls.
total 7320
-rw-r--r--   1 amrood   group   1630 Aug  8 1994  dboard.f
-rw-r-----   1 amrood   group   4340 Jul 17 1994  vttest.c
-rwxr-xr-x   1 amrood   group 525574 Feb 15 11:52 wave_shift
-rw-r--r--   1 amrood   group   1648 Aug  5 1994  wide.list
```

```
-rwxr-xr-x  1 amrood   group   4019 Feb 14 16:26 fix.c
226 Transfer complete.
ftp> get wave_shift
200 PORT command successful.
150 Opening data connection for wave_shift (525574 bytes).
226 Transfer complete.
528454 bytes received in 1.296 seconds (398.1 Kbytes/s)
ftp> quit
221 Goodbye.
$
```

Telnet 工具

用户在工作经常会遇到这样的需求：用户需要连接到远程 UNIX 主机且需要在远程主机上进行操作。Telnet 就是一个允许用户对远程服务器进行连接、登陆且可以进行远程操作的工具。

一旦用户使用 Telnet 工具登陆到了远程服务器上，那么用户就可以像在本地主机操作那样操作远程服务器来执行任务。下面是 Telnet 对话的一个例子：

```
C:>telnet amrood.com
Trying...
Connected to amrood.com.
Escape character is '^]'.

login: amrood
amrood's Password:
******************************************************
*  *
*  *
*WELCOME TO AMROOD.COM  *
*  *
*  *
******************************************************

Last unsuccessful login: Fri Mar  3 12:01:09 IST 2009
Last login: Wed Mar  8 18:33:27 IST 2009 on pts/10

  {  do your work }

$ logout
Connection closed.
C:>
```

finger 工具

finger 指令用于显示指定主机上有关用户的信息。这里的主机可以是本地主机，也可以是远程服务器。

由于安全原因，finger 也能在其他系统中使用。

下面是使用 finger 指令的简单语法。

检测本地主机中登陆用户的信息的例子如下：

```
$ finger
Login Name   Tty  Idle  Login Time   Office amrood   pts/0  Jun 25 08:03 (62.61.164.115)
```

获取本地主机上指定有效用户的信息的例子如下：

```
$ finger amrood
Login: amrood   Name: (null)
Directory: /home/amrood Shell: /bin/bash
On since Thu Jun 25 08:03 (MST) on pts/0 from 62.61.164.115
No mail.
No Plan.
```

检测远程服务器中所有登陆用户的信息的例子如下：

```
 $ finger @avtar.com
Login Name   Tty  Idle  Login Time   Office
amrood   pts/0  Jun 25 08:03 (62.61.164.115)
```

获取远程服务器上的指定有效用户信息的例子如下：

```
$ finger amrood@avtar.com
Login: amrood   Name: (null)
Directory: /home/amrood Shell: /bin/bash
On since Thu Jun 25 08:03 (MST) on pts/0 from 62.61.164.115
No mail.
No Plan.
```

vi 编辑器使用向导

在 UNIX 操作系统中，文档编辑器有很多种，其中最友好的工具就是 vi. 该文档编辑器可以让用户在文件中基于上下文执行编辑操作。

现在，用户可以使用 vi 文档编辑器的升级版 VIM.这里的 VIM 中的 M 就是源自改善的意思。

vi 通常会被认为是 UNIX 编辑器的实施标准，原因如下：

- 几乎在所有 UNIX 系列操作系统中都有效。
- 它的具体实现都非常的类似。
- 仅仅占用非常少的资源。
- 与其他编辑器相比，用户界面更加友好。

用户可以使用 vi 编辑器来编辑已经存在的文件，当然也可以使用它创建一个新的文件。此外，用户也可以使用它来浏览一个文本文件。

开始使用 vi 编辑器

用户可以通过下面几种方法来开启 vi 编辑器：

指令	描述
vi filename	如果文件不存在，则创建这个文件，否则打开这个文件
vi -R filename	以只读的方式打开一个已存在的文件
view filename	以只读的方式打开一个已存在的文件

下面是一个创建新文件 testfile 的例子，当然，前提是该文件在当前目录内不存在。

```
$vi testfile
```

然后用户会在屏幕上看到如下显示：

```
|
~
~
~
~
~
~
```

```
~
~
~
~
~
~
"testfile" [New File]
```

此时，应该注意到的是在每一行都的开头都会有一个波浪号（~），这个波浪号代表该行并没有被使用。如果某一行没有以波浪号开头，那么这就意味着有空格存在，也可能是换行符或其他的不易看见的符号。

所以，在我们正式使用 vi 编辑器之前，让我们理解一些微小但是很重要的。

操作模式

使用 vi 编辑器时通常会在下面两个模式之间来回切换：

- **指令模式**：该模式下，用户可以完成一些诸如保存文件、执行指令、移动光标、剪切或粘贴某行（字符）、查找或替代的管理功能。在该模式下，用户键入的所有内容都会被系统当做指令来执行。
- **插入模式**：该模式下，用户可以往文件内，键入文本。在该模式下，用户键入的所有内容都会被当做是文件输入而最后保存在文本中。

vi 编辑器通常会以指令模式打开。如果用户希望键入文本的话，用户必须把当前模式切换为插入模式。切换模式到插入模式时，用户可以简单的按 i 键。离开插入模式，可以按 Esc 键。

值得注意的是，如果用户不清楚当前处于哪种模式，用户可以按两次 Esc 键，编辑器就会回到指令模式。用户可以使用 vi 编辑器打开一个文本，并且键入一些字符，然后切换到指令模式，体验一下这前后的差别。

离开 vi 编辑器

离开 vi 编辑器的指令是 `：q` 。在指令模式下，一旦键入冒号后跟字母 q，再单击回车。如果文档有修改的话，编辑器会报出提示。如果不想看到这样的提示信息，用户可以使用 `：q!` 。这个指令可以在不保存更改的情况下离开 vi 编辑器。

保存文档的指令时 `：w` 。用户可以将其与上述的离开指令融合在一起使用，比如 `：wq` ，然后单击回车。

保存文件更新且退出的最快捷的方法就是 ZZ 指令。当用户在指令模式下键入 zz 指令后，效果等同于 `：wq` 。

用户也可以通过再： w 后指定特定的文件名来指定不同的文件名来保存。比如，如果用户希望将文件保存为 filename2，用户可以键入：w filename2， 然后点击回车。

在文件内移动光标

为了在文件内不影响文本内容的情况下移动光标，用户必须在指令模式（连续两次按下 Esc 键）下来操作。下面是文件内移动光标的相关指令：

指令	描述
k	向上移动一行
j	向下移动一行
h	将光标往左移动一个字符
l	将光标往右移动一个字符

下面是两个需要注意的重点：

- vi 编译器是大小写敏感的，所以用户在指令模式下进行操作时要注意大小写。
- 大部分 vi 编辑器中的指令都可以通过在动作行为前加个数字来表明希望发生的次数。比如，2j 意味着希望光标往下移动两行。

还有其他一些在文件内移动光标的方法。需要格外注意的是必须在指令模式下来操作。下面是其他在文件内移动光标的方法。

指令	描述
0 或 l	光标回到行首
$	光标回到行末尾
w	光标移动到下一个字
b	光标移动到前一个字
(	光标移动到当前句子的开头
)	光标移动到下个句子的开头
E	光标移动空白分割单词的末尾
{	光标往后移动一段
}	光标往前移动一段
[[	光标往后移动一节
]]	光标往前移动一节
n|	光标移动到当前行的n列
1G	光标移动到文件的第一行
G	光标移动到文件的最后一行
nG	光标移动到文件的第 n 行
： n	光标移动到文件的第 n 行
fc	光标往前移动到 c

Fc	光标往后移动到 c
H	光标移动到屏幕的首部
nH	光标移动到屏幕从顶部查第 n 行
M	光标移动到屏幕的中间
L	光标移动到屏幕的底部
nL	光标移动到从屏幕底部往上第 n 行
：x	光标移动到行号为 x 的行

Control指令

下面是一些和 Control 键配合使用的指令：

指令	描述
CTRL+d	向前移动半个屏幕
CTRL+f	向前移动整个屏幕
CTRL+u	向后移动半个屏幕
CTRL+b	向后移动整个屏幕
CTRL+e	屏幕往上移动一行
CTRL+y	屏幕往下移动一行
CTRL+u	屏幕往上移动半个页面
CTRL+f	屏幕往下移动一个页面
CTRL+l	重绘屏幕

编辑文件

只有在插入模式下才可以对文本进行编辑。从指令模式切换为插入模式可以有很多种方法：

指令	描述
i	当前位置前插入文本
I	当前行首插入文本
a	当前位置后插入文本
A	当前行末尾插入文本
o	在光标位置下方新建一行来输入文本
a	在光标位置上方新建一行来输入文本

删除字符

下面是关于用来在打开的文件中删除字符或行的指令列表：

指令	描述
x	删除光标位置下的字符
X	删除光标位置前的字符
dw	删除光标位置到下一个字间的所有字符
d^	删除光标位置到行首间的所有字符
d$	删除光标位置到行末尾间的所有字符
D	删除光标位置到当前行末尾间的所有字符
dd	删除一整行

正如前面所述，vi 中的大部分指令都可以在前面加数字来表示希望执行的次数。比如，2x 意味着会删除当前光标位置下的两个字符，2dd 意味着删除两行。

本教程建议在学习后面相关内容前可以多多练习以上内容。

更改指令

用户可以在不删除文本的情况下对字符、字和行进行更改。下面是相关指令：

指令	描述
cc	删除当前行，仅留下用户键入的文本
cw	删除光标所在的单词，并进入插入模式
r	替换掉光标下的字符，vi 在替换结束后回到指令模式
R	覆写当前光标处的多个字符，只有使用 Esc 才可以停止覆写
s	将当前字符替换为用户键入的字符，之后，仍处于插入模式
S	删除光标所在行，替换为用户键入的文本，之后，系统仍处于插入模式

复制和粘贴指令

用户可以从一个地方复制一行或一个字，然后粘贴到其他地方，相关指令如下：

指令	描述
yy	复制当前行
yw	复制当前字

p	在光标后粘贴
P	在光标前粘贴

高级指令

有一些高级的指令可以简化日常编辑操作：

指令	描述
J	当前行与下一行想连接，定义一个数字可以连接好多行
<<	当前行往左跳转，一个 shift 的宽度
>>	当前行往右跳转，一个 shift 的宽度
~	当前光标下，大小写切换
U	将当前行重新回到光标刚到这里的状态
u	撤销该文件的最后一个更改，再次键入u就是重新实现更新
:f	在屏幕上显示当前光标的半分比、文件名级文件总数
:f filename	重命名当前文件到 filename
:w filename	写到文件 filename
:e filename	打开另一个文件 filename
:cd dirname	切换当前目录到目录 dirname
:e #	在两个已打开的文件间切换
:n	用户在使用 vi 打开多个文件的情况下，使用该指令一次切换到下一个文件
:p	用户在使用 vi 打开多个文件的情况下，使用该指令一次切换到上一个文件
:N	用户在使用 vi 打开多个文件的情况下，使用该指令一次切换到下一个文件
:r file	读取文件 file，然后在当前行的后面进入插入模式
:nr file	读取文件 file，然后在当前行的后面n行进入插入模式

字或字符搜索

vi 编辑器有两种搜索方式：字符串和字符。对于字符串搜索，需要使用到 `/` 和 `?` 指令。当用户开始键入这些指令的时候，这些指令就会显示在屏幕的底部，这样，用户就可以加入指定的字符串来进行搜索了。

这两个指令仅仅在搜索方向上有所不同：

- `/` 指令是从上往下搜索。
- `?` 指令时从下往上搜索。

n 和 N 指令用于以同样或相反的方向重复上述搜索指令。在搜索指令中，一些字符串拥有特殊的意义。就需要使用转意符（\）。

指令	描述
^	在行的首部开始搜索
.	匹配单个字符
*	匹配0个或更多个前面的字符
$	在行末尾开始搜索
[	启动一个匹配或者不匹配的表达式
<	在表达式中，来找到一个字的开头或结尾
>	参考上面的<

集指令

用户可以通过如下指令来改变 vi 界面的外观和友好程度：set 指令。下述指定必须在指令模式下键入。

指令	描述
:set ic	搜索的时候忽略大小写
:set ai	设置自动缩进
:set noai	不设置自动缩进/td>
:set nu	在左侧显示行号
:set sw	设置制表符的宽度。比如，:set sw=4 意味着将制表符宽度设置为4
:set ws	如何循环搜索被设置，则如果在文件底部没有搜索到，会重新从文件开头开始搜索。
:set wm	如果这个选项有一个值大于零,编辑器会“自动换行”。
:set ro	更改文件读写类型为只读
:set term	输出终端类型
:set bf	忽略控制字符

运行指令

vi 编辑器可以运行指令集。为此，用户必须在指令模式下输入 `:!` 指令。

比如，如果用户在保存文件前，希望检测文件是否已存在，那么用户可以键入 `:! ls` ,然后用户就看以在屏幕上看到 ls 指令的的输出。

按任意键后，会回到 vi 界面。

替换文本

`:s/` 指令可以快速替换字或者一组字。下面是语法：

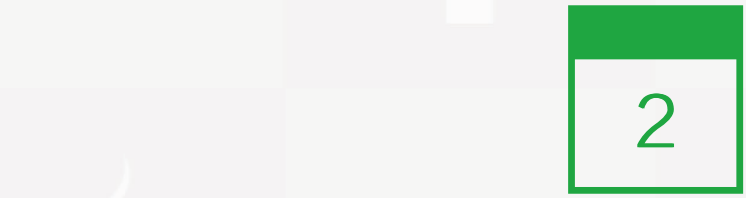
```
:s/search/replace/g
```

g 代表全局。该指令的意思就是出现在光标所在行的所有匹配字符都会被替换。

注意

下面是使用 vi 编辑器时的重要提示：

- 用户必须在指令模式下来键入指令（连续按两次 Esc 键，来保证当前处于指令模式）。
- 用户要注意指令的大小写。
- 用户只能在插入模式下来键入文本。



2 UNIX 程序设计

什么是 Shell 脚本

Shell 是用户访问 Unix 操纵系统的接口。它接收用户的输入，然后基于该输入执行程序。程序执行完后，结果会显示在显示器上。

Shell 就是运行指令、程序和 Shell 脚本的运行环境。就和操作系统可以有很多种类一样，Shell 也有很多种。每一种 Shell 都有其特定的指令和函数集。

Shell 提示符

提示符 `$` 被称为命令提示符。当显示命令提示符后，用户就可以键入命令。

Shell 在用户按 Enter 键后，从用户输入设备读入输入信息，它通过查看用户输入的第一个单词，来获知用户想要执行的命令。一个字即使字符不分割组成的字符串，一般是空格和制表符分割字。

下面是在显示器上显示当前日期和时间的 date 指令的例子：

```
$date
Thu Jun 25 08:30:19 MST 2009
```

用户也可以定制自己喜欢的命令提示符，方法是改变环境变量 PS1。

Shell 类型

Unix 系统中有两种主要的 shell：

- Bourne shell：如果用户使用 bourne shell，默认命令提示符是 `$` 。
- C shell:如果用户使用 bourne shell，默认命令提示符是 `%` 。

Bourne shell 也有如下几种子分类：

- Bourne shell (sh)
- Korn shell (ksh)
- Bourne Again shell (bash)
- POSIX shell (sh)

C shell不同的类型如下：

- C shell (csh)
- TENEX/TOPS C shell (tcsh)

最初的 UNIX Shell 是 Stephen R. Bourne 在 1970 年代中期写的。当时,他在新泽西的 AT&T 贝尔实验室工作。

Bourne shell是第一个出现在 Unix 系统中的 shell,因此它被称为标准的“shell”。

Bourne shell通常是安装在大多数版本的 Unix 中的 `/bin/sh` 目录。由于这个原因,在不同版本的 Unix 上也会选择这种 Shell 来编写脚本。

在本教程中,我们将覆盖 Bourne shell 中的大部分概念。

Shell 脚本

Shell 脚本的主要形式就是一系列的命令，这些命令会顺序执行。良好风格的 Shell 会有相应的注释。

Shell 脚本有条件语句（A 大于 B）、循环语句、读取文件和存储数据、读取变量且存储数据，当然，Shell 脚本也包括函数。

Shell 脚本和函数都是翻译型语言，所以他们并不会被编译。

在后面的部分，我们会尝试写一些脚本。他们是一些写有命令的简单文本文件。

脚本例子

假设我们创建一个名为 test.sh 的脚本。注意所有脚本的后缀名都必须为 .sh。假设之前，用户已经往里面添加了一些命令，下面就是要启动这个脚本。例子如下:

```
#!/bin/sh
```

这个命令告诉系统，后面的是 bourne shell*它应念成 shebang，因为 # 被称为 hash，！ 称为 bang*

为了创建包含这些指令的脚本，用户需要先键入 shebang 行，然后键入指令:

```
#!/bin/bash
pwd
ls
```

Shell 注释

可以像下面一样来为脚本添加注释：

```
#!/bin/bash

# Author : Zara Ali
# Copyright (c) Tutorialspoint.com
# Script follows here:
pwd
ls
```

现在用户已经保存了上述内容，然后就可以执行了：

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

执行脚本方式如下：

```
$./test.sh
```

这会输出如下结果：

```
/home/amrood
index.htm  unix-basic_utilities.htm  unix-directories.htm
test.shunix-communication.htmunix-environment.htm
```

注意：如果想要执行当前目录下的脚本，需要使用如下方式 **./program_name**

扩展的 Shell 脚本：

Shell 脚本有几个构造告诉 Shell 环境做什么和什么时候去做。当然，大多数脚本比上面复杂得多。

毕竟，Shell 是一种真正的编程语言，它可以有变量，控制结构等等。无论多么复杂的脚本，它仍然只是一个顺序执行的命令列表。

以下脚本使用 **read** 命令从键盘输入并分配给变量 PERSON，最后打印 STDOUT。

```
#!/bin/sh

# Author : Zara Ali
# Copyright (c) Tutorialspoint.com
# Script follows here:

echo "What is your name?"
```

```
read PERSON
echo "Hello, $PERSON"
```

下面是运行该脚本的例子：

```
$./test.sh
What is your name?
Zara Ali
Hello, Zara Ali
$
```

变量

变量就是被赋值后的字符串。那个赋给变量的值可以是数字、文本、文件名、设备或其他类型的数据。

本质上，变量就是执行实际数据的指针。Shell 可以创建、赋值和删除变量。

变量名

变量名仅能包含字母、数字或者下划线。

约定俗成的，UNIX Shell 的变量名都使用大写。

下面是一些有效的变量名的例子：

```
_ALI
TOKEN_A
VAR_1
VAR_2
```

下面是一些无效的变量名的例子：

```
2_VAR
-VARIABLE
VAR1-VAR2
VAR_A!
```

不能使用 `！` 、 `*` 、 `-` 等字符的原因是，这些字符在 Shell 中有特殊用途。

定义变量

变量可以按照如下方式来定义：

```
variable_name=variable_value
```

比如：

```
NAME="Zara Ali"
```

上述例子定义了变量 NAME，然后赋值 "Zara Ali".这种类型的变量是常规变量，这种变量一次只能赋值一个。

Shell 可以随心所欲的赋值。比如：

```
VAR1="Zara Ali"
VAR2=100
```

访问变量

为了获取存储在变量内的值，需要在变量名前加 $.

比如，下面的脚本可以访问变量 NAME 中的值，然后将之打印到 STDOUT:

```
#!/bin/sh

NAME="Zara Ali"
echo $NAME
```

会出现下面的值:

```
Zara Ali
```

只读变量

Shell 使用只读命令提供了使变量只读化的功能。这样的变量，都不能被改变。

比如，下面的脚本中，对变量 NAME 的值进行修改，系统会报错:

```
#!/bin/sh

NAME="Zara Ali"
readonly NAME
NAME="Qadiri"
```

会出现如下结果:

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

删除变量

变量的删除会告诉 Shell 从变量列表中删除变量从而，无法对其进行跟踪。一旦用户删除了一个变量,将无法访问存储在变量中。

下面是使用 unset 指令的例子:

```
unset variable_name
```

上述指令会取消已定义变量。下面是简单的例子：

```
#!/bin/sh

NAME="Zara Ali"
unset NAME
echo $NAME
```

上述例子不会显示任何信息，不能使用 unset 指令取消被标记为只读模式的变量。

变量类型

Shell 脚本被执行的时候，主要存在如下三种变量类型：

- **局部变量**：该类型变量只会在当前 Shell 实例内有效。他们无法适用于由 Shell 启动的程序。他们仅在命令提示符处进行设置。
- **环境变量**：环境变量对 Shell 的任何子进程都有效。部分程序是需要正确的调用函数才需要环境变量。通常，Shell 脚本只会定义程序运行需要的环境变量。
- **Shell 变量**：该类型变量是由 Shell 设置的专用变量，是用来正确调用函数用的。有时这些变量是环境变量，有时是局部变量。

特殊变量

之前的教程就在命名变量时，使用某些非字符数值作为字符变量名提出警告。这是因为这些字符用于作为特殊的 UNIX 变量的名称。这些变量是预留给特定功能的。

例如，$ 字符代表进程的 ID 码，或当前 Shell 的 PID：

```
$echo $$
```

以上命令将输出当前 Shell 的 PID：

```
29949
```

下面的表列出了一些特殊变量，可以在你的 Shell 脚本中使用它们：

变量	描述
$0	当前脚本的文件名。
$n	这些变量对应于调用一个脚本时的参数。n 是一个十进制正整数，对应于特定参数的位置(第一个参数是 $1，第二个参数是 $2 等等)。
$#	提供给脚本的参数数量。
$*	所有的参数都表示两个引用。如果一个脚本接收了两个参数，即 $* 相当于 $1 $2。
$@	所有的参数都是两个单独地引用。如果一个脚本接收了两个参数，即 $@ 相当于 $1 $2。
$?	执行最后一个命令的退出态。
$$	当前 shell 的进程号。对于 shell 脚本，即他们正在执行的进程的 ID。
$!	最后一个后台命令的进程号。

命令行参数

命令行参数 $1，$2，$3，……$9 是位置参数，$0 指向实际的命令，程序，shell 脚本或函数。$1，$2，$3，……$9 作为命令的参数。

以下脚本使用与命令行相关的各种特殊变量：

```
#!/bin/sh

echo "File Name: $0"
echo "First Parameter : $1"
echo "Second Parameter : $2"
echo "Quoted Values: $@"
```

```
echo "Quoted Values: $*"
echo "Total Number of Parameters : $#"
```

这是一个运行上述脚本的示例：

```
$./test.sh Zara Ali
File Name : ./test.sh
First Parameter : Zara
Second Parameter : Ali
Quoted Values: Zara Ali
Quoted Values: Zara Ali
Total Number of Parameters : 2
```

特殊参数 $* 和 $@

存在一些特殊参数，使用它们可以访问所有的命令行参数。除非他们包含在双引号 "" 中，否则 $* 和 $@ 运行是相同的。

这两个参数都指定所有的命令行参数，但 $* 特殊参数将整个列表作为一个参数，各个值之间用空格隔开。而 $@ 特殊参数将整个列表分隔成单独的参数。

我们可以编写如下所示的 Shell 脚本，使用 $* 或 $@ 特殊参数来处理数量未知的命令行参数：

```
#!/bin/sh

for TOKEN in $*
do
  echo $TOKEN
done
```

作为示例，运行上述脚本:

```
$./test.sh Zara Ali 10 Years Old
Zara
Ali
10
Years
Old
```

注意：这里 do……done 是一种循环，我们将在后续教程中介绍它。

退出态

$? 变量代表前面的命令的退出态。

退出态是每个命令在其完成后返回的数值。一般来说，大多数命令如果它们成功地执行，将 0 作为退出态返回，如果它们执行失败，则将 1 作为退出态返回。

一些命令由于一些特定的原因，会返回额外的退出状态。例如，一些命令为了区分不同类型的错误，将根据特定类型的失败原因返回各种不同的退出态值。

下面是一个成功命令的例子:

```
$./test.sh Zara Ali
File Name : ./test.sh
First Parameter : Zara
Second Parameter : Ali
Quoted Values: Zara Ali
Quoted Values: Zara Ali
Total Number of Parameters : 2
$echo $?
0
$
```

数组

一个 Shell 变量只能够容纳一个值。这种类型的变量称为标量变量。

Shell 数组变量可以同时容纳多个值，它支持不同类型的变量。数组提供了一种变量集分组的方法。你可以使用一个数组变量存储所有其他的变量，而不是为每个必需的变量都创建一个新的名字。

Shell 变量中讨论的所有命名规则都将适用于命名数组。

定义数组值

一个数组变量和一个标量变量之间的差异可以解释如下。

假如你想描绘不同学生的名字，你需要命名一系列变量名作为一个变量集合。每一个单独的变量是一个标量变量，如下所示：

```
NAME01="Zara"
NAME02="Qadir"
NAME03="Mahnaz"
NAME04="Ayan"
NAME05="Daisy"
```

我们可以使用一个数组来存储所有上面提到的名字。下面是创建一个数组变量的最简单的方法，将值赋给数组的一个索引。表示如下：

```
array_name[index]=value
```

这里 array_name 是数组的名称，index 是数组中需要赋值的索引项，value 是你想要为这个索引项设置的值。

例如，以下命令:

```
NAME[0]="Zara"
NAME[1]="Qadir"
NAME[2]="Mahnaz"
NAME[3]="Ayan"
NAME[4]="Daisy"
```

如果使用 ksh shell，数组初始化的语法如下所示：

```
set -A array_name value1 value2 ... valuen
```

如果使用 bash shell，数组初始化的语法如下所示：

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

访问数组值

在为数组变量赋值之后，你可以访问它。如下所示：

```
${array_name[index]}
```

这里 array_name 是数组的名称，index 是将要访问的值的索引。下面是一个最简单的例子：

```
#!/bin/sh

NAME[0]="Zara"
NAME[1]="Qadir"
NAME[2]="Mahnaz"
NAME[3]="Ayan"
NAME[4]="Daisy"
echo "First Index: ${NAME[0]}"
echo "Second Index: ${NAME[1]}"
```

这将产生以下结果：

```
$./test.sh
First Index: Zara
Second Index: Qadir
```

你可以使用以下方法之一，来访问数组中的所有项目：

```
${array_name[*]}
${array_name[@]}
```

这里 array_name 是你感兴趣的数组的名称。下面是一个最简单的例子：

```
#!/bin/sh

NAME[0]="Zara"
NAME[1]="Qadir"
NAME[2]="Mahnaz"
NAME[3]="Ayan"
NAME[4]="Daisy"
echo "First Method: ${NAME[*]}"
echo "Second Method: ${NAME[@]}"
```

这将产生以下结果：

```
$./test.sh
First Method: Zara Qadir Mahnaz Ayan Daisy
Second Method: Zara Qadir Mahnaz Ayan Daisy
```

基本操作符

每一种 Shell 都支持各种各样的操作符。我们的教程基于默认的 Shell(Bourne)，所以在我们的教程中涵盖所有重要的 Bourne Shell 操作符。

下面列出我们将讨论的操作符：

- 算术运算符。
- 关系运算符。
- 布尔操作符。
- 字符串运算符。
- 文件测试操作符。

最初的 Bourne Shell 没有任何机制来执行简单算术运算，它使用外部程序 awk 或者最简单的程序 expr。

下面我们用一个简单的例子说明，两个数字相加：

```
#!/bin/sh

val=`expr 2 + 2`
echo "Total value : $val"
```

这将产生以下结果：

```
Total value : 4
```

注意以下事项：

- 操作符和表达式之间必须有空格，例如 2+2 是不正确的，这里应该写成 2 + 2。
- 完整的表达应该封闭在两个单引号 " 之间。

算术运算符

下面列出 Bourne Shell 支持的算术运算符。

假设变量 a 赋值为 10，变量 b 赋值为 20：

运算符	描述	例子
+	加法 – 将操作符两边的数加起来	`expr $a + $b` = 30

-	减法 - 用操作符左边的操作数减去右边的操作数	`expr $a - $b` = -10
*	乘法 - 将操作符两边的数乘起来	`expr $a \* $b` = 200
/	除法 - 用操作符左边的操作数除以右边的操作数	`expr $b / $a` = 2
%	取模 - 用操作符左边的操作数除以右边的操作数，返回余数	`expr $b % $a` = 0
=	赋值 - 将操作符右边的操作数赋值给左边的操作数	a=$b 将 b 的值赋给了 a
==	相等 - 比较两个数字，如果相同，返回 true	[$a == $b] = false
!=	不相等 - 比较两个数字，如果不同，返回true。	[$a != $b] = true

这里需要非常注意是，所有的条件表达式和操作符之间都必须用空格隔开，例如 [$a == $b] 是正确的,而 [$a==$b] 是不正确的。

所有的算术计算都是针对长整数操作的。

关系运算符

Bourne Shell 支持以下的关系运算符，这些运算符是专门针对数值数据的。它们不会对字符串值起作用，除非他们的值是数值数据。

例如，下面的操作符将检查 10 和 20 之间的关系以及 “10” 和 “20” 的关系，但不能用于判断 “ten” 和 “twenty” 的关系。

假设变量 a 赋值为 10， 变量 b 赋值为 20：

运算符	描述	例子
-eq	检查两个操作数的值是否相等，如果值相等，那么条件为真。	[$a -eq $b] is not true.
-ne	检查两个操作数的值是否相等，如果值不相等，那么条件为真。	[$a -ne $b] is true.
-gt	检查左操作数的值是否大于右操作数的值，如果是的，那么条件为真。	[$a -gt $b] is not true.
-lt	检查左操作数的值是否小于右操作数的值，如果是的，那么条件为真。	[$a -lt $b] is true.
-ge	检查左操作数的值是否大于等于右操作数的值，如果是的，那么条件为真。	[$a -ge $b] is not true.
-le	检查左操作数的值是否小于等于右操作数的值，如果是的，那么条件为真。	[$a -le $b] is true.

这里需要非常注意是，所有的条件表达式和操作符之间都必须用空格隔开，例如 [$a <= $b] 是正确的，而 [$a<=$b] 是不正确的。

布尔操作符

Bourne Shell 支持以下的布尔操作符。

假设变量 a 赋值为 10， 变量 b 赋值为 20：

运算符	描述	例子
!	这表示逻辑否定。如果条件为假，返回真，反之亦然。	[! false] is true.
-o	这表示逻辑 OR。如果操作对象中有一个为真，那么条件将会是真。	[$a -lt 20 -o $b -gt 100] is true.
-a	这表示逻辑 AND。如果两个操作对象都是真，那么条件才会为真，否则为假。	[$a -lt 20 -a $b -gt 100] is false.

字符串运算符:

Bourne Shell 支持以下字符串运算符。

假设变量 a 赋值为 "abc"， 变量 b 赋值为 "efg":

示例说明

运算符	描述	例子
=	检查两个操作数的值是否相等，如果是的，那么条件为真。	[$a = $b] is not true.
!=	检查两个操作数的值是否相等，如果值不相等，那么条件为真。	[$a != $b] is true.
-z	检查给定字符串操作数的长度是否为零。如果长度为零，则返回true。	[-z $a] is not true.
-n	检查给定字符串操作数的长度是否不为零。如果长度不为零，则返回true。	[-z $a] is not false.
str	检查字符串str是否是非空字符串。如果它为空字符串，则返回false。	[$a] is not false.

文件测试操作符:

下列操作符用来测试与Unix文件相关联的各种属性。

假设一个文件变量 file，包含一个文件名 "test"，文件大小是100字节，在其上有读、写和执行权限:

示例说明

运算符	描述	例子
-b file	检查文件是否为块特殊文件，如果是，那么条件为真。	[-b $file] is false.
-c file	检查文件是否为字符特殊文件，如果是，那么条件变为真。	[-c $file] is false.
-d file	检查文件是否是一个目录文件，如果是，那么条件为真。	[-d $file] is not true.
-f file	检查文件是否是一个不同于目录文件和特殊文件的普通文件，如果是，那么条件为真。	[-f $file] is true.
-g file	检查文件是否有组ID(SGID)设置，如果是，那么条件为真。	[-g $file] is false.

-k file	检查文件是否有粘贴位设置，如果是，那么条件为真。	[-k $file] is false.
-p file	检查文件是否是一个命名管道，如果是，那么条件为真。	[-p $file] is false.
-t file	检查文件描述符是否可用且与终端相关，如果是，条件为真实。	[-t $file] is false.
-u file	检查文件是否有用户id(SUID)设置，如果是，那么条件为真。	[-u $file] is false.
-r file	检查文件是否可读，如果是，那么条件为真。	[-r $file] is true.
-w file	检查文件是否可写，如果是，那么条件为真。	[-w $file] is true.
-x file	检查文件是否可执行，如果是，那么条件为真。	[-x $file] is true.
-s file	检查文件大小是否大于0，如果是，那么条件为真。	[-s $file] is true.
-e file	检查文件是否存在。即使文件是一个目录目录，只有存在，条件就为真。	[-e $file] is true.

决策

编写 Shell 脚本时，可能存在一种情况，你需要在两条路径中选择一条路径。所以你需要使用条件语句，确保你的程序做出正确的决策并执行正确的操作。

UNIX Shell 支持条件语句，这些语句基于不同的条件，用于执行不同的操作。在这里，我们将介绍以下两个决策语句：

- if……else语句
- case…… esac语句

if……else 语句：

if……else 语句是非常有用的决策语句，它可以用来从一个给定的选项集中选择一个选项。

Unix Shell 支持以下形式的 if……else 的语句：

- if...fi statement
- if...else...fi statement
- if...elif...else...fi statement

大部分的 if 语句使用关系运算符检查关系，这部分知识在前一章已经讨论过。

case…… esac 语句

你可以使用多个 if……elif 语句执行一个多路分支。然而，这并不总是最好的解决方案，特别是当所有的分支都依赖于一个单一变量的值。

Unix Shell 支持 case……esac 语句，可以更确切地处理这种情况，它比重复 if……elif 语句更加有效。

`case...esac` 语句只有一种形式，详细说明如下：

- case...esac statement

Unix Shell 的 case……esac 语句非常类似于 switch……case 语句，switch……case 语句在其他编程语言如 C 或 C++ 和 PERL 等中实现。

循环

循环是一个强大的编程工具，可以使您能够重复执行一系列命令。针对 Shell 程序员，有 4 种循环类型：

- while 循环
- for 循环
- until 循环
- select 循环

根据不同的情况使用不同的循环。例如只要给定条件仍然是 true，while 循环将执行给定的命令。而 until 循环是直到给定的条件变成 true，才会执行。

一旦你有了良好的编程实践，你就会开始根据情况使用适当的循环。while 循环和 for 循环在大多数其他编程语言如 C、C++ 和 PERL 等中都有实现。

嵌套循环

所有的循环都支持嵌套的概念，这意味着可以将一个循环放到另一个相似或不同的循环中。这个嵌套可以根据您的需求高达无限次。

下面是一个嵌套 while 循环的例子，基于编程的要求，其他循环类型也可以以类似的方式嵌套：

嵌套 while 循环

可以使用 while 循环作为另一个 while 循环体的一部分。

语法

```
while command1 ; # this is loop1, the outer loop
do
   Statement(s) to be executed if command1 is true

   while command2 ; # this is loop2, the inner loop
   do
  Statement(s) to be executed if command2 is true
   done
```

```
   Statement(s) to be executed if command1 is true
done
```

例子

下面是循环嵌套的一个简单例子，在循环内部添加另一个倒计时循环，用来数到九：

```
#!/bin/sh

a=0
while [ "$a" -lt 10 ]# this is loop1
do
   b="$a"
   while [ "$b" -ge 0 ]  # this is loop2
   do
  echo -n "$b "
  b=`expr $b - 1`
   done
   echo
   a=`expr $a + 1`
done
```

这将产生以下结果。重要的是要注意 echo -n 是如何工作的。这里 `-n` 选项让输出避免了打印新行字符。

```
0
1 0
2 1 0
3 2 1 0
4 3 2 1 0
5 4 3 2 1 0
6 5 4 3 2 1 0
7 6 5 4 3 2 1 0
8 7 6 5 4 3 2 1 0
9 8 7 6 5 4 3 2 1 0
```

循环控制

到目前为止你已经学习过创建循环以及用循环来完成不同的任务。有时候你需要停止循环或跳出循环迭代。

在本教程中你将学到以下语句用于控制 Shell 循环：

- break 语句
- continue 语句

无限循环

所有循环都有一个有限的生命周期。当条件为假或真时它们将跳出循环，这取决于这个循环。

一个循环可能会由于未匹配到适合得条件而无限执行。一个永远执行没有终止的循环会执行无数次。因此，这种循环被称为无限循环。

例子

这是一个使用 while 循环显示数字 0 到 9 的简单的例子：

```
#!/bin/sh

a=10

while [ $a -ge 10 ]
do
   echo $a
   a=`expr $a + 1`
done
```

这个循环将永远持续下去，因为 a 总是大于或等于 10，它永远不会小于 10。所以这正是无限循环的一个恰当的例子。

break 语句

所有在 break 语句之前得语句执行结束后执行 break 语句，break 语句用于跳出整个循环。然后执行循环体后面的代码。然后在循环结束后运行接下来的代码。

语法

以下 break 语句将用于跳出一个循环：

```
break
```

break 语句也可以使用这种格式来退出嵌套循环式：

```
break n
```

在这里 n 指定封闭循环执行的次数然后退出循环。

例子

这里是一个简单的例子，用来说明只要 a 变成 5 循环将终止：

```
#!/bin/sh

a=0

while [ $a -lt 10 ]
do
   echo $a
   if [ $a -eq 5 ]
   then
  break
   fi
   a=`expr $a + 1`
done
```

这会产生以下结果：

```
0
1
2
3
4
5
```

这里是一个简单的嵌套 for 循环的例子。如果 var1 等于 var2 以及 var2 等于 0 ，则这个脚本将跳出这个双重循环：

```
#!/bin/sh

for var1 in 1 2 3
do
   for var2 in 0 5
   do
  if [ $var1 -eq 2 -a $var2 -eq 0 ]
  then
 break 2
  else
 echo "$var1 $var2"
  fi
   done
done
```

这会产生以下结果。在内循环中，有一个 break 命令，其参数为 2。这表明，你应该打破外循环和内循环才能满足条件。

```
1 0
1 5
```

continue 语句

continue 语句类似于 break 命令，二者不同之处在于，continue 语句用语结束当前循环，能引起当前循环的迭代的退出，而不是整个循环。

这个语句在当程序发生了错误，但你想执行下一次循环的时候是非常有用的。

语法

```
continue
```

正如 break 语句，一个整型参数可以传递给 continue 命令以从嵌套循环中跳过命令。

```
continue n
```

在这里 n 指定封闭循环执行的次数然后进入下一次循环。

例子

下面是使用 continue 语句的循环，它返回 continue 语句并且开始处理下一个语句：

```
#!/bin/sh

NUMS="1 2 3 4 5 6 7"

for NUM in $NUMS
do
  Q=`expr $NUM % 2`
  if [ $Q -eq 0 ]
  then
 echo "Number is an even number!!"
 continue
  fi
  echo "Found odd number"
done
```

这会产生以下结果：

```
Found odd number
Number is an even number!!
Found odd number
Number is an even number!!
Found odd number
Number is an even number!!
Found odd number
```

替代

什么是替代?

当它遇到包含一个或多个特殊字符的表达式时 Shell 执行替代。

例

以下是一个例子，在这个例子中，变量被其真实值所替代。同时，“\n” 被替换为换行符:

```
#!/bin/sh

a=10
echo -e "Value of a is $a \n"
```

这会产生以下结果。在这里 -e 选项可以解释反斜杠转义。

```
Value of a is 10
```

下面是没有 -e 选项的结果:

```
Value of a is 10\n
```

这里有以下转义序列可用于 echo 命令:

Escape	Description
\\	反斜杠
\a	警报(BEL)
\b	退格键
\c	抑制换行
\f	换页
\n	换行
\r	回车
\t	水平制表符
\v	垂直制表符

默认情况下，你可以使用 -E 选项来禁用反斜线转义解释。

你你可以使用 -n 选项来禁用换行的插入。

命令替换：一种机制，通过它， Shell 执行给定的命令，然后在命令行替代他们的输出。

语法

当给出如下命令时命令替换就会被执行：

```
command
```

当执行命令替换确保你使用反引号，而不是单引号字符。

例

命令替换一般是用来分配一个命令的输出变量。下面的示例演示命令替换：

```
#!/bin/sh

DATE=`date`
echo "Date is $DATE"

USERS=`who | wc -l`
echo "Logged in user are $USERS"

UP=`date ; uptime`
echo "Uptime is $UP"
```

这会产生以下结果：

```
Date is Thu Jul  2 03:59:57 MST 2009
Logged in user are 1
Uptime is Thu Jul  2 03:59:57 MST 2009
03:59:57 up 20 days, 14:03,  1 user,  load avg: 0.13, 0.07, 0.15
```

变量代换

变量代换使 Shell 程序员操纵基于状态变量的值。

下面的表中是所有可能的替换：

Form	Description
${var}	替代 *var* 的值
${var:-word}	如果 *var* 为空或者没有赋值，*word* 替代 var。*var* 的值不改变。

${var:=word}	如果 *var* 为空或者没有赋值， *var* 赋值为 word 的值。
${var:?message}	如果 *var* 为空或者没有赋值，*message* 被编译为标准错误。这可以检测变量是否被正确赋值。
${var:+word}	如果 *var* 被赋值，*word* 将替代 var。*var* 的值不改变。

例

以下是例子用来说明上述替代的各种状态：

```
#!/bin/sh

echo ${var:-"Variable is not set"}
echo "1 - Value of var is ${var}"

echo ${var:="Variable is not set"}
echo "2 - Value of var is ${var}"

unset var
echo ${var:+"This is default value"}
echo "3 - Value of var is $var"

var="Prefix"
echo ${var:+"This is default value"}
echo "4 - Value of var is $var"

echo ${var:?"Print this message"}
echo "5 - Value of var is ${var}"
```

这会产生以下结果：

```
Variable is not set
1 - Value of var is
Variable is not set
2 - Value of var is Variable is not set

3 - Value of var is
This is default value
4 - Value of var is Prefix
Prefix
5 - Value of var is Prefix
```

引用机制

元字符

UNIX Shell 提供有特殊意义的各种元字符，同时利用他们在任何 Shell 脚本，并导致终止一个字，除了引用。

举个例子，在列出文件中的目录时？ 匹配一个一元字符，并且 * 匹配多个字符。下面是一个 Shell 特殊字符（也称为元字符）的列表：

```
* ? [ ] ' " \ $ ; & ( ) | ^ < > new-line space tab
```

在一个字符前使用 \ ,它可能被引用（例如，代表它自己）。

例子

下面的例子，显示了如何打印 * 或？ ：

```
#!/bin/sh

echo Hello; Word
```

这将产生下面的结果。

```
Hello
./test.sh: line 2: Word: command not found

shell returned 127
```

现在，让我们尝试使用引用字符：

```
#!/bin/sh

echo Hello\; Word
```

这将产生以下结果：

```
Hello; Word
```

$ 符号是一个元字符，所以它必须被引用，以避免被 Shell 特殊处理：

```
#!/bin/sh

echo "I have \$1200"
```

这将产生以下结果：

```
I have $1200
```

是以下四种形式的引用：

Quoting	Description
单引号	所有在这些引号之间的特殊字符会失去它们特殊的意义
双引号	所有在这些引号之间的特殊字符会失去它们特殊的意义除了以下字符： – $ – ` – \$ – \' – \" –\\
反斜杠	任何直接跟在反斜杠后的字符会失去它们特殊的意义
反引号	所有在这些引号之间的特殊字符会被当做命令而被执行

单引号

考虑包含许多特殊的 Shell 字符的 echo 命令：

```
echo <–$1500.**>; (update?) [y|n]
```

在每个特殊字符前加反斜杠会显得异常繁琐，并且不容易阅读：

```
echo \<–\$1500.\*\*\>\; \(update\?\) \[y\|n\]
```

有一个简单的方法来引用一大组字符。将一个单引号（'）放在字符串的开头和结尾：

```
echo '<–$1500.**>; (update?) [y|n]'
```

单引号内的任何字符被引用正如每个字符前均加上一个反斜杠。所以，现在这个 echo 命令将正确地显示。

如果要输出的一个字符串内出现一个单引号，你不应该把整个字符串置于单引号内，相反你应该在单引号前使用反斜杠（\）如下：

```
echo 'It\'s Shell Programming'
```

双引号

尝试执行以下 Shell 脚本。这个 Shell 脚本使用了单引号：

```
VAR=ZARA
echo '$VAR owes <-$1500.**>; [ as of (`date +%m/%d`) ]'
```

这将输出以下结果：

```
$VAR owes <-$1500.**>; [ as of (`date +%m/%d`) ]
```

所以这不是你想显示的内容。很明显，单引号防止变量替换。如果想替换的变量值和如预期那样使引号起作用，那么就需要把命令放置在双引号内，如下：

```
VAR=ZARA
echo "$VAR owes <-\$1500.**>; [ as of (`date +%m/%d`) ]"
```

这将产生以下结果：

```
ZARA owes <-$1500.**>; [ as of (07/02) ]
```

除以下字符外，双引号使所有字符的失去特殊含义：

- $ 参数替代。
- 用于命令替换的反引号。
- \$ 使美元标志在字面上显示。
- \` 使反引号在字面上显示。
- \" 启用嵌入式双引号。
- \ 启用嵌入式反斜杠。
- 所有其他\字符在字面上显示（而不是特殊意义）。

单引号内的任何字符被引用正如每个字符前均加上一个反斜杠。所以，现在这个 echo 命令将正确地显示。

如果要输出的字符串内出现一个单引号，你不应该把整个字符串置于单引号内，相反你应该在单引号前使用反斜杠（\）如下：

```
echo 'It\'s Shell Programming'
```

反引号

置于反引号之间的任何 Shell 命令将执行命令

语法

下面是一个简单的语法，把任何 Shell 命令置于反引号之间：

例子

```
var=`command`
```

例子

下面将执行 date 命令，产生的结果将被存储在 DATA 变量中。

```
DATE=`date`

echo "Current Date: $DATE"
```

这将输出以下结果：

```
Current Date: Thu Jul  2 05:28:45 MST 2009
```

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。

输出重定向

一个命令的输出通常用于标准输出，也可以很容易地将输出转移到一个文件。这种能力被称为输出重定向：

如果记号 `> file` 添加到任何命令，这些命令通常将其输出写入到标准输出，该命令的输出将被写入文件，而不是你的终端：

检查下面的 who 命令，它将命令的完整的输出重定向在用户文件中。

```
$ who > users
```

请注意，没有输出出现在终端中。这是因为输出已被从默认的标准输出设备（终端）重定向到指定的文件。如果你想检查 *users* 文件，它有完整的内容：

```
$ cat users
oko tty01   Sep 12 07:30
ai  tty15   Sep 12 13:32
ruthtty21   Sep 12 10:10
pat tty24   Sep 12 13:07
steve   tty25   Sep 12 13:03
$
```

如果命令输出重定向到一个文件，该文件已经包含了一些数据，这些数据将会丢失。考虑这个例子：

```
$ echo line 1 > users
$ cat users
line 1
$
```

您可以使用 `>>` 运算符将输出添加在现有的文件如下：

```
$ echo line 2 >> users
$ cat users
line 1
```

```
line 2
$
```

输入重定向

正如一个命令的输出可以被重定向到一个文件中，所以一个命令的输入可以从文件重定向。大于号 `>` 被用于输出重定向，小于号 `<` 用于重定向一个命令的输入。

通常从标准输入获取输入的命令可以有自己的方式从文件进行输入重定向。例如，为了计算上面 *user* 生成的文件中的行数，你可以执行如下命令：

```
$ wc –l users
2 users
$
```

在这里，它产生的输出为 2 行。你可以通过从 *user* 文件进行 wc 命令的标准输入重定向：

```
$ wc –l < users
2
$
```

请注意，两种形式的 wc 命令产生的输出是有区别的。在第一种情况下，用行数列出该文件的用户的名称，而在第二种情况下，它不是。

在第一种情况下，wc 知道，它是从文件用户读取输入。在第二种情况下，只知道它是从标准输入读取输入，所以它不显示文件名。

Here 文档

here document 被用来将输入重定向到一个交互式 Shell 脚本或程序。

在一个 Shell 脚本中，我们可以运行一个交互式程序，无需用户操作，通过提供互动程序或交互式 Shell 脚本所需的输入。

Here 文档的一般形式是：

```
command << delimiter
document
delimiter
```

这里的 Shell 将 `<<` 操作符解释为读取输入的指令，直到它找到含有指定的分隔符线。然后所有包含行分隔符的输入行被送入命令的标准输入。

分隔符告诉 Shell here 文档已完成。没有它，Shell 不断的读取输入。分隔符必须是一个字符且不包含空格或制表符。

以下是输入命令 wc -l 来进行计算行的总数：

```
$wc -l << EOF
    This is a simple lookup program
    for good (and bad) restaurants
    in Cape Town.
EOF
3
$
```

可以用 *here document* 编译多行，脚本如下：

```
#!/bin/sh

cat << EOF
This is a simple lookup program
for good (and bad) restaurants
in Cape Town.
EOF
```

这将产生以下结果：

```
This is a simple lookup program
for good (and bad) restaurants
in Cape Town.
```

下面的脚本用 vi 文本编辑器运行一个会话并且将输入保存文件在 test.txt 中。

```
#!/bin/sh

filename=test.txt
vi $filename <<EndOfCommands
i
This file was created automatically from
a shell script
^[
ZZ
EndOfCommands
```

如果用 vim 作为 vi 来运行这个脚本，那么很可能会看到以下的输出：

```
$ sh test.sh
Vim: Warning: Input is not from a terminal
$
```

运行该脚本后，你应该看到以下内容添加到了文件 test.txt 中：

```
$ cat test.txt
This file was created automatically from
a shell script
$
```

丢弃输出

有时你需要执行命令，但不想在屏幕上显示输出。在这种情况下，你可以通过重定向到文件 `/dev/null` 以丢弃输出：

```
$ command > /dev/null
```

在这里 command 是要执行的命令的名字。文件 `/dev/null` 是一个自动丢弃其所有的输入的特殊文件。

为了丢弃一个命令的输出和它的错误输出，你可以使用标准重定向来将 STDOUT 重定向到 STDERR：

```
$ command > /dev/null 2>&1
```

在这里，2 代表 STDERR ， 1 代表 STDOUT。可以上通过将 STDERR 重定向到 STDERR 来显示一条消息，如下：

```
$ echo message 1>&2
```

重定向命令

以下是可以用来重定向的命令的完整列表：

命令	描述
pgm > file	pgm 的输出被重定向到文件
pgm < file	pgm 程序从文件度它的输入
pgm >> file	pgm 的输出被添加到文件
n > file	带有描述符 n 的输出流重定向到文件
n >> file	带有描述符 n 的输出流添加到文件
n >& m	合并流 n 和 流 m 的输出
n <& m	合并流 n 和 流 m 的输入

<< tag	标准输入从开始行的下一个标记开始。
|	从一个程序或进程获取输入,并将其发送到另一个程序或进程。

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）。

函数

函数允许你将一个脚本的整体功能分解成更小的逻辑子部分，然后当需要的时候可以被调用来执行它们各自的任务。

使用函数来执行重复性的任务是一个创建代码重用的很好的方式来。代码重用是现代面向对象编程的原则的重要组成部分。

Shell 函数类似于其他编程语言中的子程序和函数。

创建函数

声明一个函数，只需使用以下语法：

```
function_name () {
   list of commands
}
```

函数的名字是 function_name，在脚本的其它地方你可以用函数名调用它。函数名后必须加括号，在其后加花括号，其中包含了一系列的命令。

例子

以下是使用函数的简单例子：

```
#!/bin/sh

# Define your function here
Hello () {
   echo "Hello World"
}

# Invoke your function
Hello
```

当你想执行上面的脚本时，它会产生以下结果：

```
$./test.sh
Hello World
$
```

函数的参数传递

你可以定义一个函数，在调用这些函数的时候可以接受传递的参数。这些参数可以由 $1，$2 等表示。

以下是一个例子，我们传递两个参数 *Zara* 和 *Ali* ，然后我们在函数中捕获和编译这些参数。

```
#!/bin/sh

# Define your function here
Hello () {
   echo "Hello World $1 $2"
}

# Invoke your function
Hello Zara Ali
```

这将产生以下结果：

```
$./test.sh
Hello World Zara Ali
$
```

函数返回值

如果你从一个函数内部执行一个 exit 命令，不仅能终止函数的执行，而且能够终止调用该函数的 Shell 程序。

如果你只是想终止该函数的执行，有一种方式可以跳出定义的函数。

根据实际情况，你可以使用 return 命令从你的函数返回任何值，其语法如下：

```
return code
```

这里的 *code* 可以是你选择的任何东西，但很明显，考虑到将脚本作为一个整体，你应该选择有意义的或有用的东西。

例子

下面的函数返回一个值 1：

```
#!/bin/sh
```

```
# Define your function here
Hello () {
  echo "Hello World $1 $2"
  return 10
}

# Invoke your function
Hello Zara Ali

# Capture value returnd by last command
ret=$?

echo "Return value is $ret"
```

这将产生以下结果：

```
$./test.sh
Hello World Zara Ali
Return value is 10
$
```

嵌套函数

函数更有趣的功能之一就是他们可以调用本身以及调用其他函数。调用自身的函数被称为递归函数。

下面简单的例子演示了两个函数的嵌套：

```
#!/bin/sh

# Calling one function from another
number_one () {
  echo "This is the first function speaking..."
  number_two
}

number_two () {
  echo "This is now the second function speaking..."
}

# Calling function one.
number_one
```

这将产生以下结果：

```
This is the first function speaking...
This is now the second function speaking...
```

从 Prompt 函数调用

你可以把常用函数的定义放置到文件 *.profile* 中，这样当你载入的时候可以得到它们并且在 prompt 命令中使用它们。

或者，你可以将多个函数定义在一个文件中，比如 *test.sh*，然后通过键入以下内容当前 Shell 中执行该文件：

```
$. test.sh
```

这样做可以使 *test.sh* 内定义的任何函数被读入，定义到当前 Shell ，如下：

```
$ number_one
This is the first function speaking...
This is now the second function speaking...
$
```

要从 Shell 删除函数的定义，你可以使用带 `.f` 选项的 unset 命令。这也是用来删除 Shell 中一个变量的定义的命令。

```
$unset .f function_name
```

Manpage 帮助

UNIX 命令都有许多可选的和强制性的选项，而忘记这些命令的完整语法是很常见的事情。

因为没有人能记住每个 UNIX 命令及其所有选项,所以在 UNIX 的最早版本中，用户就可以获得在线帮助。

UNIX 版本的帮助文件被称为 Manpage。如果你知道所有命令的名称,但你不知道如何使用它,这时 Manpage 可以在每一步中帮助你。

语法

UNIX 系统工作时，以下是获取 UNIX 命令细节的简单指令

```
$man command
```

举例

你现在想象任何一个命令并且你需要得到与这个命令对应的帮助。假设你想知道 `pwd` 命令的用法,那么你只需要使用以下命令:

```
$man pwd
```

上面的命令将打开一个帮助，它会给你提供关于 `pwd` 命令的完整的信息。自己试试根据你的命令提示符来获得更多的细节。

使用以下命令，你可以获得更详细的关于 `man` 命令的细节:

```
$man man
```

Man Page 选项

Man Page通常被划分为不同的部分，而且由于不同 Man page 作者的偏好不同，划分情况也不一样。下面是一些常见的部分:

部分	描述
名称	命令的名字
描述	一般描述命令和他的作用
摘要	一般命令所使用的参数

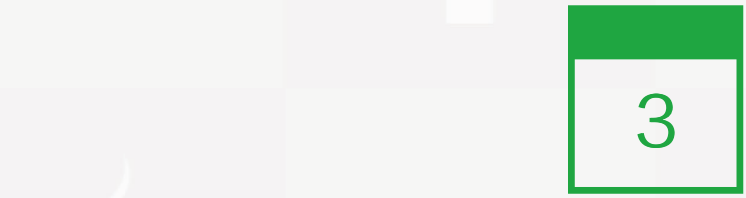
选项	描述命令的所有参数和选项
另请参阅	列出其他的在 Manpage 中与该命令直接相关的命令或功能相似的命令
漏洞	描述存在于命令或输出中的问题或错误
例子	常见的用法示例,让读者了解如何使用命令。
作者	Manpage 或其他命令的作者。

所以最后,我想说，当你需要使用 UNIX 命令或它的系统文件信息时， Man page 是一个至关重要的资源。

有用的 Shell 命令

下面的链接中给出了最重要和频繁使用 UNIX Shell命令列表。

如果你不知道如何使用命令，可以使用 Man page 获得命令的完整细节。



3

UNIX 进阶

正则表达式和 SED

正则表达式是一个字符串,可以用来描述几个字符序列。UNIX 的这些命令中会用到正则表达式,包括 ed、sed、awk、grep,以及 vi。

本教程将教你如何使用正则表达式和 sed。

这里 sed 代表的流编辑器是一个面向流的编辑器，是专门为执行脚本创建的。因此你的所有输入都会被送到 STDOUT 并且它不改变输入文件。

调用 sed

在我们开始之前,让我们以确保你有/etc/passwd文本文件的本地副本。

如前所述,可以通过一个 pipe 发送数据而调用s ed，如下所示:

```
$ cat /etc/passwd | sed
Usage: sed [OPTION]... {script-other-script} [input-file]...

  -n, --quiet, --silent
 suppress automatic printing of pattern space
  -e script, --expression=script
.....................................
```

cat 命令转储 `/etc/passwd` 的内容到 sed 是通过 pipe 进入 sed 的模式空间。sed 使用模式空间的内部工作缓冲区来做它的工作。

sed 的一般语法:

下面是 sed 的一般语法

```
  /pattern/action
```

在这里,pattern 是一个正则表达式，action 则是在下表中给出的命令。当省 pattern 时，如上面我们已经看到的，action 会执行每一行命令。

围绕 pattern 的斜杠字符(/)是不可省略的，因为它们是作为分隔符使用。

范围	描述

p	输出该行
d	删除该行
s/模式1/模式2/	替代第一次出现的模式1和模式2

用 sed 删除所有行

再次调用 sed ,但这一次使用 sed 的编辑命令删除一行记录，使用字母 d 表示其：

```
$ cat /etc/passwd | sed 'd'
$
```

除了通过 pipe 发送一个文件来调用 sed，你可以指导 sed 从文件中读取数据，示例如下。

下面的命令与前面是完全一样的，尝试一下，里面不包括 cat 命令：

```
$ sed -e 'd' /etc/passwd
$
```

sed地址

sed 也可以理解为所谓的地址。地址可以是文件中的一个位置，也可以是一个特殊的编辑命令适用的范围。当 sed 遇到没有地址的情况时，它会对文件中的每一行执行其操作。

下面的命令将一个基本的地址添加到您使用的 sed 命令中：

```
$ cat /etc/passwd | sed '1d' |more
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
$
```

注意，数字 1 添加在删除命令前面。这告诉 sed 在文件的第一行执行编辑命令。在这个例子中，sed将删除 `/etc/password` 文件的第一行并打印文件的其他部分。

sed 地址范围

所以如果你想从文件中删除一行，您可以指定一个地址范围如下：

```
$ cat /etc/passwd | sed '1, 5d' |more
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
$
```

以上命令应用的范围是 1 到 5 行。所以将删除这五行。

尝试以下地址范围：

范围	描述
'4,10d'	删除第 4 到 10 行
'10,4d'	只删除第 10 行，因为 sed 不能反方向工作
'4,+5d'	这将匹配文件中的第 4 行,删除这一行之后,继续删除下一个五行,然后停止其删除操作并输出其他行
'2,5!d'	这将删除除 2 到 5 行外的所有其他行。
'1~3d'	删除第一行后，跳过接下来的三行，然后删除第四行。sed 继续这种模式直到文件的末尾。
'2~2d'	sed 删除第二行，跳过下一行后，删除下面的一行，并重复，直到到达文件的末尾。
'4,10p'	输出 4 到 10 行之间的内容。
'4,d'	产生语法错误。
',10d'	也产生语法错误。

注意：在使用 `p action` 的时候,您应该使用 `-n` 选项来避免重复输出。检查以下两个命令的 betweek 差异：

```
$ cat /etc/passwd | sed -n '1,3p'
```

上面的命令不加 - n 的情形如下：

```
$ cat /etc/passwd | sed '1,3p'
```

替换命令

替换命令，用 s 表示，将取代你指定的任何其他字符串。

用一个字符串替代另一个，你需要告诉 sed 你第一个字符串的结束位置和想要替换的字符串的开始位置。传统上是由正斜杠(/)将两个字符串分开的。

以下命令将替换第一次出现的 root 和 amrood 字符串。

```
$ cat /etc/passwd | sed 's/root/amrood/'
amrood:x:0:0:root user:/root:/bin/sh
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
.....
```

非常重要的一点是，sed 替代只在一个命令行的某一字符串第一次出现时才能使用。如果字符串 root 在一行里面出现不止一次，只有第一个 root 字符串被替换。

sed 去做一个全局替换，需要添加字母 g 到命令末尾，命令如下：

```
$ cat /etc/passwd | sed 's/root/amrood/g'
amrood:x:0:0:amrood user:/amrood:/bin/sh
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
.....
```

替换标志

除了 g 标志外，还有许多其他有用的标志可以使用，而且您每次可以指定多余一个标志。

标志	描述
g	替换所有可以匹配的字符而不仅仅是第一个
NUMBER	仅仅替换第 NUMBLER 个匹配的字符
p	如果发生了替换，则输出模式空间
w FILENAME	如果发生了替换，则将结果写到 FILENAME
I or i	以不区分大小写的方式匹配
M or m	除了拥有特殊正则表达式字符`^`和`$`的正常的行为外,这个标志使`^`匹配换行符后的空字符串，使$匹配换行符前的空字符串。

使用一个可替换的字符串分隔符

你会发现自己不得不对包含斜杠字符的字符串做一个替换。在这种情况下，您可以对 s 后的字符来指定一个不同的分隔符。

```
$ cat /etc/passwd | sed 's:/root:/amrood:g'
amrood:x:0:0:amrood user:/amrood:/bin/sh
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
```

在上面的例子中：`/` 作为定界符使用,而不是斜线 `/`。因为我们试图搜索 `/root`，而不是简单的 root 字符串。

使用空串的执行替换

使用一个空的替换字符串去删除 `/etc/passwd` 文件的 root 字符串。

```
$ cat /etc/passwd | sed 's/root//g'
:x:0:0::/:/bin/sh
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
```

地址替换

如果你想只在第 10 行用字符串 sh 替换字符串 quiet，你可以指定如下：

```
$ cat /etc/passwd | sed '10s/sh/quiet/g'
root:x:0:0:root user:/root:/bin/sh
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/quiet
```

同样的,做一个地址范围替换，你可以做如下操作：

```
$ cat /etc/passwd | sed '1,5s/sh/quiet/g'
root:x:0:0:root user:/root:/bin/quiet
daemon:x:1:1:daemon:/usr/sbin:/bin/quiet
bin:x:2:2:bin:/bin:/bin/quiet
sys:x:3:3:sys:/dev:/bin/quiet
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
```

正如你从输出所看到的，前五行里面的字符串 sh 都改为了 quiet，但是其他行里面的 sh 都丝毫没有改变。

匹配命令

你可以使用 p 参数和 – n 参数输出所有匹配的行，如下所示：

```
$ cat testing | sed –n '/root/p'
root:x:0:0:root user:/root:/bin/sh
[root@ip–72–167–112–17 amrood]# vi testing
root:x:0:0:root user:/root:/bin/sh
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
```

使用正则表达式

在进行模式匹配时，您可以使用正则表达式，它提供了更多的灵活性。

检查下面的例子中以 daemon 开始的行然后删除：

```
$ cat testing | sed '/^daemon/d'
root:x:0:0:root user:/root:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
```

下面是将删除以 sh 结尾的所有行的例子：

```
$ cat testing | sed '/sh$/d'
sync:x:4:65534:sync:/bin:/bin/sync
```

下表列出了四个在正则表达式里面非常有用的特殊字符。

字符	描述
^	匹配一行的起始
$	匹配一行的结尾
.	匹配任何的单个字符
*	匹配零个或多个以前出现的字符
[chars]	为了匹配任何字符串的字符。您可以使用`-`字符来表示字符的范围。

匹配字符

来看看在其他的表达式里面如何演示元字符的使用。例如下面的模式:

表达式	描述
/a.c/	匹配包含字符串如a+c，a-c，abc, match, 还有 a3c
/a*c/	匹配相同的字符串还有字符串比如ace，yacc，以及arctic
/[tT]he/	匹配字符The和the
/^$/	匹配空白行
/^.*$/	不管任何情况，都匹配一整行
/ */	匹配一个或多个空格
/^$/	匹配空行

下表给出了一些常用的字符:

集	描述
[a-z]	匹配一个小写字母
[A-Z]	匹配一个大写字母
[a-zA-Z]	匹配一个字母
[0-9]	匹配数字
[a-zA-Z0-9]	匹配单个字母或数字

字符类关键词

通常来说，一些特殊的关键字对 regexp 来说也是适用的,尤其是 GNU 实用程序会使用 regexp。对 sed 正则表达式来说这些都是非常有用的，因为这样既简化了表达式又增强了可读性。

例如，字符 a 到 z 以及字符 A 到 Z 构成了这样一个用关键字 `[[:alpha:]]` 表示的类。

使用字母表的字符类关键词，这个命令输出 `/etc/syslog.conf` 文件里面以字母表的字母开始的行:

```
$ cat /etc/syslog.conf | sed -n '/^[[:alpha:]]/p'
authpriv.* /var/log/secure
mail.* -/var/log/maillog
cron.* /var/log/cron
uucp,news.crit /var/log/spooler
local7.*   /var/log/boot.log
```

下表是 GNU sed 的可用的字符类关键词的一个完整的列表。

字符类	描述
[[:alnum:]]	字母(a - z A-Z 0 - 9)
[[:alpha:]]	字母(a - z A-Z)
[[:blank:]]	空白字符(空格或制表键)
[[:cntrl:]]	控制字符
[[:digit:]]	数字[0 - 9]
[[:graph:]]	任何可见字符(不包括空格)
[[:lower:]]	小写字母的[a -ź]
[[:print:]]	可打印字符(无控字符)
[[:punct:]]	标点字符
[[:space:]]	空白
[[:upper:]]	大写字母的[A -Z]
[[:xdigit:]]	十六进制数字[0 - 9 a - f A-F]

&引用

sed 元字符 & 代表被匹配的 pattern 的内容。例如，假设您有一个名为 phone.txt 的文件，里面都电话号码，如下所示：

```
5555551212

5555551213

5555551214

6665551215

6665551216

7775551217
```

你想让前三个数字被括号括起来以更容易阅读。要做到这一点，您可以使用 & 替换字符，如下所示：

```
$ sed – e ' s / ^[[数位:]][[数位:]][[数位:]](&)/ g phone.txt


(555)5551212

(555)5551213

(555)5551214

(666)5551215

(666)5551216

(777)5551217
```

先匹配 3 位数字，然后使用 & 取代那些括号括起来的数字。

使用多个 sed 命令

您可以在一个 sed 命令下使用多个 sed 命令，如下：

```
$ sed –e 'command1' –e 'command2' ... –e 'commandN' files
```

这里 commandN 到 command1 都是我们之前讨论的 sed 类型命令。这些命令应用于每个文件列表的行。

以相同的机制，我们可以以下面的方式写上面的电话号码：

```
$ sed – e ' s / ^[[数位:]]\ \ { 3 } /(&)/ g \

                – e ' s /)[[数位:]]\ \ { 3 } / & – / g phone.txt

(555)555 – 1212

(555)555 – 1213

(555)555 – 1214

(666)555 – 1215

(666)555 – 1216

(777)555 – 1217
```

注意：在上面的例子中，不是重复字符类关键字 [[:digit:]] 三次,而是代之以 \{3\} ，这意味着前三次正则表达式相匹配。

引用

& 元字符是有用的，但更有用的功能是能够在正则表达式中定义特定区域，通过定义正则表达式的特定的一部分，您可以引用字符引用这部分。

反向引用时,你必须首先定义一个区域,然后回顾这个区域。定义一个区域是在你感兴趣的区域插入 \ 和括号。你周围的第一区域被通过 \1 引用,第二个地区用 \2 引用，等等。

假设 phone.txt 有以下文本:

```
(555)555 – 1212

(555)555 – 1213

(555)555 – 1214

(666)555 – 1215

(666)555 – 1216

(777)555 – 1217
```

现在试试下面的命令:

```
$ cat phone.txt | sed 's/\(.*)\)\(.*–\)\(.*$\)/Area \
  code: \1 Second: \2 Third: \3/'
Area code: (555) Second: 555– Third: 1212
Area code: (555) Second: 555– Third: 1213
Area code: (555) Second: 555– Third: 1214
Area code: (666) Second: 555– Third: 1215
Area code: (666) Second: 555– Third: 1216
Area code: (777) Second: 555– Third: 1217
```

注意:在上面的例子中每个括号内的正则表达式将引用 \1 ，， \2 等等。

文件系统基础知识

文件系统是一个分区或磁盘上的文件的逻辑集合。一个分区是一个信息的容器，如果需要可以跨整个硬盘。

你的硬盘可以有不同的分区，但通常只包含一个文件系统，如一个文件系统涵盖 `/file` 系统，另一个包含 `/home` 文件系统。

一个文件系统分区允许不同文件系统的逻辑维护和管理。

UNIX 中一切都被认为是一个文件，包括物理设备，如 DVD-ROMs、USB 设备、软盘驱动器等等。

目录结构

UNIX 使用文件系统层次结构，就像一棵倒置的树，根目录(/) 是文件系统的底部，所有其他的目录都从那里蔓延。

UNIX 文件系统是文件和目录的集合，具有以下属性：

- 它有一个根目录 (/)，包含其他的文件和目录。
- 使用名字唯一地标识每个文件或目录，这个名字可以是它所在的目录，或者一个独特的标识符，通常被称为一个 inode。
- 按照惯例，根目录的 inode 编号为 2，lost+found 目录的 inode 编号为 3。Inode 编号 0 和 1 暂不使用。文件的 inode 编号可以通过 ls 命令的 -i 选项指定。
- 它是自包含的。一个文件系统和其他文件系统之间没有依赖关系。

目录有特定的目的，通常存储相同类型的信息以实现更容易定位文件的目的。以下是主要的 UNIX 版本上存在的目录：

目录	描述
/	这是根目录，只包含顶层文件结构所需的目录。
/bin	这是可执行文件所在的地方。他们提供给所有用户使用。
/dev	这些是设备驱动程序。
/etc	上级目录的命令，配置文件，磁盘配置文件，有效的用户列表，组，以太网，主机等各种发送重要信息的地方。
/lib	包含共享库文件，例如其他内核相关文件。
/boot	包含系统启动相关的文件。

/home	包含用户的主目录和其他账户。
/mnt	用来挂载其他临时文件系统，比如分别针对光盘和软盘的 CD-ROM 驱动器和软盘驱动器。
/proc	标记为一个包含所有进程的文件，这些进程使用进程编号或其他信息标记。这个文件是一个动态的系统。
/tmp	包含系统启动期间所有的临时文件。
/uer	用于各种各样的用途，可以被许多用户使用。包括行政命令、共享文件、库文件等等。
/var	通常包含变长文件，如日志和打印文件和任何其他类型的文件，该文件包含的数据的量可能是变化的。
/sbin	包含二进制(可执行的)文件，通常用于系统管理。比如 fdisk 和 ifconfig 功能。
/kernel	包含内核文件。

浏览文件系统

既然已经了解了文件系统的基本知识，现在就可以开始导航到所需要的文件。以下列出导航到文件系统可以使用的命令：

命令	描述
cat filename	显示文件名。
cd dirname	移动到确定的目录。
cp file1 file2	复制一个文件/目录到指定位置。
file filename	识别文件类型(二进制、文本等)。
find filename dir	发现一个文件/目录。
head filename	显示一个文件的开始。
less filename	从结束或开始位置浏览一个文件。
ls dirname	显示指定目录的内容。
mkdir dirname	创建指定目录。
more filename	从头到尾浏览一个文件。
mv file1 file2	移动一个文件/目录的位置或重命名一个文件/目录。
pwd	显示用户当前所在的目录。
rm filename	删除一个文件。
rmdir dirname	删除一个目录。
tail filename	显示一个文件的结束。
touch filename	创建一个空白文件或修改现有文件的属性。
whereis filename	显示一个文件的位置。
which filename	如果文件在你的路径内，显示它的位置，。

df命令

管理分区空间的第一种方式是 df (磁盘空闲)命令。命令 df -k(磁盘空闲)以千字节的形式显示磁盘空间的使用情况，如下所示：

```
$df -k
Filesystem  1K-blocks  Used   Available Use% Mounted on
/dev/vzfs10485760   7836644 2649116  75% /
/devices0 0   0   0% /devices
$
```

一些目录，比如 `/devices`，以千字节形式显示使用为 0，且可用列以及能力都为 0%。这些特殊的(或虚拟的)文件系统，虽然他们驻留在磁盘上，但他们本身不占用磁盘空间。

在所有 UNIX 系统上 `df -k` 的输出通常都是相同的。它一般包括：

列	描述
Filesystem	物理文件系统名称。
kbytes	存储介质上的可用空间总字节。
used	被文件使用过的空间的总字节。
avail	可用空间的总字节。
capacity	被文件使用的空间和总额的比例。
Mounted on	文件系统正在安装的。

您可以使用 `-h` (可读的)选项来设置显示，使用易于理解的符号，合适的大小等输出格式。

du 命令

du (磁盘使用量) 命令使您能够按指定目录来显示一个特定的目录中磁盘空间的使用情况。

如果你想判断一个特定的目录正在使用多少空间，这个命令是很有用的。以下命令将显示被每个目录消耗的块的数量。根据系统的不同，一个块可能需要 512 字节或 1 千字节。

```
$du /etc
10 /etc/cron.d
126/etc/default
6  /etc/dfs
...
$
```

`-h` 选项使输出更容易理解：

```
$du -h /etc
5k/etc/cron.d
63k   /etc/default
3k/etc/dfs
...
$
```

挂载文件系统

文件系统必须安装以用于系统的正常使用。为了查看您的系统上目前安装(可用)的文件系统，可以使用这个命令：

```
$ mount
/dev/vzfs on / type reiserfs (rw,usrquota,grpquota)
proc on /proc type proc (rw,nodiratime)
devpts on /dev/pts type devpts (rw)
$
```

UNIX 协定的 `/mnt` 目录，就是临时挂载的地方(例如 CD-ROM 驱动器，远程网络驱动器，软盘驱动器)。如果你需要挂载文件系统，您可以使用 mount 命令，语法如下：

```
mount -t file_system_type device_to_mount directory_to_mount_to
```

例如，如果你想挂载 CD-ROM 到目录 `/mnt/cdrom`，你可以输入：

```
$ mount -t iso9660 /dev/cdrom /mnt/cdrom
```

假设您的 CD-ROM 设备称为 `/dev/cdrom`，你想挂载到 `/mnt/cdrom`。可以参考安装手册页获得更具体的信息或类型，在命令行输入 `-h` 得到帮助信息。

安装之后，您可以使用 cd 命令通过挂载点来浏览可用的新文件系统。

卸载文件系统

通过识别挂载点或设备，从你的系统中卸载(删除)文件系统。使用 umount 命令实行。

例如，可以使用以下命令卸载光盘：

```
$ umount /dev/cdrom
```

mount 命令使你能够访问你的文件系统，但在大多数现代 UNIX 系统中，加载函数使这个过程对用户不可见且不需要用户干预。

用户和组配额

提供用户和组配额的机制：单个用户或特定组中的所有用户所使用的空间量可以由管理员定义的值限制。

配额操作有两种限制。如果空间的数量或磁盘块的数量开始超过管理员定义的限制，允许用户采取行动：

- **软限制**：如果用户超过限制的定义，有一个宽限度，允许用户释放一些空间。
- **硬限制**：当达到硬限制使，忽略宽限度，没有进一步的文件或块可以分配。

有许多命令来管理配额:

命令	描述
quota	显示组中一个用户的磁盘使用情况和限制。
edquota	这是一个配额编辑器。可以使用这个命令编辑用户或组配额。
quotacheck	扫描文件系统，为了磁盘使用、制造、检查和修复配额文件
setquota	这也是一个命令行配额编辑器。
quotaon	系统宣布应该在一个或多个文件系统上启用磁盘配额。
quotaoff	系统宣布应该禁用一个或多个文件系统上的磁盘配额。
repquota	打印指定文件系统的磁盘使用和配额的汇总

您可以使用 Manpage 帮助 查看这里提到每个命令的完整语法。

用户管理

在 UNIX 系统中，有三种类型的账户：

- root 帐户：这也被称为超级用户，这类用户对系统拥有完整的和不受约束的控制权。超级用户可以运行任何命令，而不受任何限制。这类用户应该承担作为一个系统管理员的任务。
- 系统账户：系统账户是为操作系统特定组件的需要提供的，例如邮件账户和 sshd 账户。这些账户通常是为了满足系统上一些特定的功能的需要而设定的，对它们进行的任何修改都可能会对系统造成负面影响。
- 用户帐号：用户帐户提供交互式访问系统的用户和用户组。通常给普通用户分配这些账户，通常附带有对关键系统文件和目录有限的访问权限。

UNIX 支持组帐号（Group Account）的概念，在逻辑上是许多账户的群组。每个帐户都可能是任何组账号的一部分。UNIX 组在处理文件权限和流程管理中发挥了重要的作用。

管理用户和组

下面列出三个主要的用户管理文件：

- /etc/passwd：此文件保存用户帐户和密码信息。这个文件包含了 UNIX 系统上大多数的账户信息。
- /etc/shadow：此文件包含相应帐户的加密密码。不是所有的系统都支持这个文件。
- /etc/group：此文件包含每个帐户的组信息。
- /etc/gshadow：此文件包含安全组帐号信息。

使用 cat 命令检查上述所有文件。

大多数 UNIX 系统可用以下命令来创建和管理帐户和组：

命令	描述
useradd	将账户添加到系统。
usermod	修改账户属性。
userdel	从系统删除账户。
groupadd	将组添加到系统。
groupmod	修改组属性。
groupdel	从系统中删除组。

可以使用 Manpage 帮助 查看这里提到每个命令的完整语法。

创建一个组

在创建任何账户之前需要先创建组，否则将不得不使用系统中现有的组。你会在 `/etc/groups` 文件中找到所有组的列表。

所有默认组都是系统帐户组成的特定组，不推荐普通账户使用。所以下面给出用来创建一个新组帐户的语法:

```
groupadd [-g gid [-o]] [-r] [-f] groupname
```

下面列出详细的参数:

选项	描述
-g GID	组 ID 的数值。
-o	这个选项允许给组添加一个非唯一的 GID。
-r	这个标志表示给组添加一个系统账户。
-f	如果指定的组已经存在，这个选项会导致成功退出。附带 -g 时，如果指定 GID 已经存在，就选择其他(独特的) GID。
groupname	创建一个真实的组名称。

如果你没有指定任何参数，那么系统将使用默认值。

以下示例将使用默认值创建开发人员组，这为大部分的管理员接受。

```
$ groupadd developers
```

修改组

修改一个组,使用 groupmod 语法:

```
$ groupmod -n new_modified_group_name old_group_name
```

将 developers_2 组的名称改为 developer，例如:

```
$ groupmod -n developer developer_2
```

下边描述如何将 developer 的 GID 更改为 545:

```
$ groupmod -g 545 developer
```

删除一个组

删除现有的组，需要的所有东西就是 groupdel 命令和组名。例如删除 developer 组，命令是：

```
$ groupdel developer
```

这个操作只是删除了组，而不涉及任何跟组相关的文件。这些文件仍然可以被它们的主人访问。

创建一个帐户

让我们看看如何在 UNIX 系统上创建一个新的帐户。下面是用来创建一个用户帐户的语法：

```
useradd -d homedir -g groupname -m -s shell -u userid accountname
```

下面列出详细的参数:

选项	描述
-d homedir	指定账户的主目录。
-g groupname	指定该账户所属的组账户。
-m	如果它不存在，则创建主目录。
-s shell	指定该帐户的默认 shell。
-u userid	您可以为账户指定一个用户id。
accountname	创建一个真实的帐户名称

如果你没有指定任何参数，那么系统将使用默认值。useradd 命令将修改 `/etc/passwd` 文件、`/etc/shadow` 文件、`/etc/group` 文件并创建一个主目录。

下面的示例将创建一个帐户：mcmohd，主目录设置为 `/home/mcmohd`，组为 developers。将 Korn Shell 分配给这个用户。

```
$ useradd -d /home/mcmohd -g developers -s /bin/ksh mcmohd
```

上面的命令执行之前，必须确保你已经使用 groupadd 命令创建了 developers 组。

创建一个帐户之后，你可以使用 passwd 命令设置它的密码，如下所示：

```
$ passwd mcmohd20
Changing password for user mcmohd20.
New UNIX password:
Retype new UNIX password:
passwd: all authentication tokens updated successfully.
```

当您输入 passwd 账户名，它会假定你是超级用户，从而更改密码。否则你只能使用这样的命令改变你的密码，而不能更改指定帐户的密码。

修改一个账户

usermod 命令允许从命令行更改现有的账户。它使用和 useradd 命令相同的参数，加上 `-l` 参数，允许更改帐户名称。

例如，将账户名称 mcmohd 更改为 mcmohd20 并相应地改变主目录，需要执行以下命令：

```
$ usermod -d /home/mcmohd20 -m -l mcmohd mcmohd20
```

删除一个账户

userdel 命令可以用来删除现有的用户。这是一个非常危险的命令，必须小心使用。

这个命令只有一个参数或可用的选项：`.r`，用来删除帐户的主目录和邮件文件。

例如，删除帐户 mcmohd20，需要发出以下命令：

```
$ userdel -r mcmohd20
```

如果为了备份的目的，想保留它的主目录，省略 `-r` 选项。可以根据需要在稍后的时间删除主目录。

系统性能

本教程的目的是介绍一些可用的免费性能分析工具来对 UNIX 系统性能进行监控和管理，并提供了如何分析和解决 UNIX 环境中性能问题的指导。

UNIX 有以下需要进行监控和调整的主要资源类型：

- CPU
- 内存
- 磁盘空间
- 通信线路
- I/O 时间
- 网络时间
- 应用程序

性能组件

主要有以下五个组件，随着总系统时间的推移而变化：

组件	描述
用户状态 CPU	在用户状态下，CPU 运行用户程序的所花费的时间。它包括用于执行库函数调用的时间，但是不包括自己在内核中花费的时间。
系统状态 CPU	在系统状态下，CPU 花费在运行程序上的时间。所有的 I/O 例程均需要内核服务。程序员可以通过使用阻塞 I/O 传输来影响这个值。
I/O 时间和网络时间	这些是花费在移动数据和 I/O 请求服务的时间
虚拟内存的性能	这包括了上下文的切换和交换。
应用程序	花费在运行其他程序的时间--系统没有为这个应用程序提供服务是因为另一个应用程序在当前状态下持有 CPU。

性能工具

UNIX 提供了以下重要工具来估量和调整 UNIX 系统的性能：

命令	描述
nice/renice	修改运行程序的调度优先级。
netstat	打印网络连接，路由表，接口状态，无效连接和多播成员。
time	测量一个普通命令的运行时间或资源的使用情况。
uptime	系统平均负载。
ps	显示当前进程的快照。
vmstat	显示虚拟内存信息的统计。
gprof	显示调用关系图资料。
prof	进程性能分析。
top	显示系统的任务。

系统日志

UNIX 系统有一个非常灵活和强大的日志系统，它让你能够记录几乎任何你能想象的东西，然后你可以操作日志来获取你需要的信息。

许多版本的 UNIX 提供了一个名为 *syslog* 的通用日志工具，有信息需要记录的单独程序要将信息发送到 syslog。

Unix *syslog* 是一个主机可配置的，统一的系统日志工具。该系统采用集中式的系统日志进程，其运行程序 `/etc/syslogd` 或者 `/etc/syslog` 。

系统记录器的操作是相当简单的。程序发送日志条目到 *syslogd*，其将会在配置文件 `/etc/syslogd.conf` 或 `/etc/syslog` 中查找，当找到一个匹配后，将日志消息写入到期望的日志文件中。

现有你应该了解的四种基本日志术语：

术语	描述
Facility	此标识符用来描述提交的日志信息的应用程序或进程。例如邮件，内核和 FTP。
Priority	一个显示消息重要性的指示器。syslog 作为准则定义了消息的级别，从调试信息到关键事件。
Selector	一个或更多的 facility 和 level 的结合体 。当一个输入事件匹配一个 selector 时，一个 action 会被执行。
Action	传入的消息匹配 selector 时会发生的事情。Action 可以将消息写入日志文件，将消息回传到控制台或其他设备，将消息写入到一个登录用户，或将消息发送到另一个日志服务器。

Syslog Facilities

下面是 selector 可用的 facility。不是所有的 facility 都存在于所有版本的 UNIX。

Facility	描述
auth	需要用户名和密码的相关活动（getty，su，login）
authpriv	类似于 auth 的认证，但是记录的文件只能被授权的用户读取。
console	用于捕获信息，这些信息一般会传向系统控制台。
cron	与 cron 系统有关的计划任务信息。
daemon	所捕获的所有系统守护进程信息。
ftp	ftp 守护进程相关的信息。
kern	内核信息。

local0.local7	用户自定义使用的本地信息。
lpr	与打印服务系统有关的信息。
mail	与邮件系统相关的信息。
mark	用于生产日志文件中时间戳的伪事件。
news	与网络新闻传输协议(nntp)有关的信息。
ntp	与网络时间协议有关的信息。
user	普通用户进程产生的信息。
uucp	UUCP 子系统生成的信息。

Syslog 优先级

syslog 的优先级(Priority)如下表：

Priority	描述
emerg	紧急情况，如即将发生的系统崩溃，通常会广播到所有用户。
alert	需要立即修改的情况，如系统数据库的损坏。
crit	关键的情况，如一个硬件的错误。
err	普通错误。
warning	警告
notice	不是一个错误的情况，但是可能需要用特定方式的处理一下。
info	报告性的消息。
debug	用于调试程序的消息。
none	没有重要级别，通常用于指定非日志的消息。

facility 和 level 的组合能够让你辨别记录了什么和这些日志信息去哪儿了。

每个程序尽职尽责地向系统记录器发送消息，记录器基于 selector 定义的 level 决定跟踪什么和舍弃什么信息。

当你指定了一个 level，系统会记录这一 level 及更高 level 的一切信息。

文件 /etc/syslog.conf

文件 /etc/syslog.conf 用于配置记录消息的位置。一个典型的 syslog.conf 文件看起来应该像这样：

```
*.err;kern.debug;auth.notice /dev/console
daemon,auth.notice          /var/log/messages
lpr.info                    /var/log/lpr.log
mail.*                      /var/log/mail.log
ftp.*                       /var/log/ftp.log
auth.*                      @prep.ai.mit.edu
```

```
auth.*                root,amrood
netinfo.err             /var/log/netinfo.log
install.*             /var/log/install.log
*.emerg                 *
*.alert               |program_name
mark.*                 /dev/console
```

文件中的每一行包含两部分：

- 一个消息 selector，其指定了哪种消息用来记录。例如，内核的所有错误信息或所有调试信息。
- 一个 action，其指明了对接收的消息该怎么处理。例如，写入一个文件中或者将消息发送到用户的终端。

下面是上述配置的注意事项：

- 消息 selector 有两部分：facility 和 priority。例如， kern.debug 选择了所有由内核(facility)产生的的调试信息(priority)。
- 消息 selectetor kern.debug 选择了所有 priority 大于 debug 的信息。
- 在任何 facility 和 priority 位置上的星号，表示“所有”的意思。例如， *.debug 表示所有 facility 的调试信息，而 kern.* 表示内核所产生的所有信息。
- 你也可以用逗号来指定多个 facility。两个或两个以上的 selectetor 可以用分号组合在一起。

日志记录 Action

action 部分指定了下面五个 action 中的其中一个：

1. 将信息记录到一个文件或设备。例如， `/var/log/lpr.log` 或者 `/dev/console` 。
2. 发送一个消息给一个用户。你可以用逗号分开指定多个用户名（ 例如，root，amrood ）。
3. 发送一个消息给所有用户。在这种情况下，action 部分包含了一个星号（ 例如，* ）。
4. 用管道发送消息到程序。在这种情况下，程序是在 UNIX 管道符号（ | ）后指定。
5. 将消息发送到另一台主机上的 syslog。在这种情况下，action 部分包含了一个前面有 at 符号的主机名（ 例如，@jikexueyuan.com ）。

logger 命令

UNIX 提供了 **logger** 命令，这是处理系统日志记录的一个非常有用的命令。**logger** 命令发送日志消息到 syslog d 守护进程，从而驱使系统记录日志。

这意味着我们可以随时用命令行检查 syslogd 守护进程及其配置。logger 命令提供了一种在命令行上添加一行条目到系统日志文件中的方法。

该命令的格式是：

```
logger [-i] [-f file] [-p priority] [-t tag] [message]...
```

下面是具体的参数细节：

选项	描述
-f filename	使用文件 filename 的内容作为消息来记录。
-i	日志的每一行都记录进程的 id。
-p priority	指定输入消息的优先级 priority（指定的 selector），优先级 priority 可以是数字或者指定为 facility.level 对的格式。默认参数是 user.notice。
-t tag	用指定 tag 标记记录到日志中的每一行。
message	字符串参数，它的内容以特定顺序连接在一起，由空格分开。

日志轮换

日志文件有快速增长的特点，并消耗大量的磁盘空间。大多数 UNIX 发行版系统使用了工具（如 newsyslog 或 logrotate）启用日志轮换功能。

这些工具由 cron 守护进程在一个频繁的时间间隔里调用。你可以在 newsyslog 或 logrotate 的手册页中获取更多的细节内容。

重要日志文件的位置

所有的系统应用程序创建自己的日志文件在 `/var/log` 和它的子目录里。下面这里有几个重要的应用，其相应的日志目录：

应用	目录
httpd	/var/log/httpd
samba	/var/log/samba
cron	/var/log/
mail	/var/log/
mysql	/var/log/

信号和 Traps

信号是发送给程序的软件中断，表明已发生的重要事件。这些事件的范围可以是从用户请求到非法内存访问错误。这些信号，如中断信号，表明用户要求程序做一些非一般流程控制下的事情。

以下是一些你可能会遇到的并且需要在程序中使用的常见信号：

信号名	信号值	描述
SIGHUP	1	控制终端意外终止或者是控制进程结束时发出
SIGINT	2	用户发送一个中断信号时发出（Ctrl+C）。
SIGQUIT	3	用户发送一个退出信号时发出（Ctrl+D）。
SIGFPE	8	当非法的数学操作执行时发出。
SIGKILL	9	一个进程获取到此信号，它必须立即退出，并且不能进行任何清除操作。
SIGALRM	14	时钟信号（用于定时器）
SIGTERM	15	软件终止信号（kill 命令缺省产生此信号）。

信号列表

有一个简单的方法可以列出你的系统所支持的所有信号。仅仅用命令 kill –l 就可以显示系统所有支持的信号：

```
$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
 5) SIGTRAP      6) SIGABRT      7) SIGBUS       8) SIGFPE
 9) SIGKILL     10) SIGUSR1     11) SIGSEGV     12) SIGUSR2
13) SIGPIPE     14) SIGALRM     15) SIGTERM     16) SIGSTKFLT
17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU
25) SIGXFSZ     26) SIGVTALRM   27) SIGPROF     28) SIGWINCH
29) SIGIO       30) SIGPWR      31) SIGSYS      34) SIGRTMIN
35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3  38) SIGRTMIN+4
39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12
47) SIGRTMIN+13 48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14
51) SIGRTMAX-13 52) SIGRTMAX-12 53) SIGRTMAX-11 54) SIGRTMAX-10
55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7  58) SIGRTMAX-6
59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

Solaris, HP-UX 和 Linux 之间实际的信号列表会有所不同。

默认动作

每一个信号都有一个与之相随的默认动作。一个信号的默认动作是当脚本或程序接收到一个信号时，它的执行表现。

这些可能的默认动作是：

- 终止进程。
- 忽略信号。
- 内核映像转储。当收到信号时，此动作将创建一个称为 core 的文件，其包含进程的内存映像。
- 停止进程。
- 继续停止的进程。

发送信号

有几种向程序或者脚本传递信号的方法。其中用户最常见的是在正在运行的脚本的时候，按下 Ctrl+C 或者 INTERRUPT 键。

当你按下 Ctrl+C 键，信号 SIGINT 被发送到脚本中，根据之前定义的默认动作，脚本将会被终止。

另外一种常见的传送信号的方法是使用 kill 命令，其语法如下所示：

```
$ kill -signal pid
```

这里的 signal 要么是信号值，要么是信号名称。 pid 是信号要发送到的进程的 ID。 例如：

```
$ kill -1 1001
```

发送 HUP 信号（即意外终止信号）到运行进程 ID 为 1001 的程序。 用下面的命令来发送一个 kill 信号到同样的进程：

```
$ kill -9 1001
```

这条命令会杀死运行进程 ID 为 1001 的程序。

捕获信号

当一个 Shell 程序正在执行的过程中，你在终端按下了 Ctrl+C 或 Break 键，通常情况下，程序会立即终止，并且你的命令提示返回。但这可能并不总是你所期望的。例如，这可能会留下一堆未清理的临时文件。

捕获这些信号是很容易的，捕获命令（trap）的语法如下：

```
$ trap commands signals
```

这里的 commands 可以是任何有效的 UNIX 命令，甚至可以是一个用户定义的函数。

signals 可以是一个你想捕获的任何数量信号的列表。

在 Shell 脚本中，trap 有三种常见的用途：

1. 清除临时文件
2. 忽略信号

清除临时文件

作为 trap 命令的一个例子，下面的命令展示了，如果有人试图从终端中止程序时，你如何先删除一些文件，然后退出：

```
$ trap "rm -f $WORKDIR/work1$$ $WORKDIR/dataout$$; exit" 2
```

如果程序收到了信号值为 2 的信号，trap 命令将会被执行。从 Shell 程序上执行 trap 的这一点开始，work1$$ 和 dataout$$ 这两个文件会自动被删除。

所以如果用户中断程序的运行，这个 trap 将会被执行，你可以确保这两个文件将被清理掉。rm 后面的这个 exit 命令，它的存在是必要的。如果没有它，程序会在它中断点（也就是信号接收的时刻）继续执行。

信号值 1 是由挂断产生的：要么是有人故意挂断终端或者是终端被意外断开。

在这种情况下，把信号值 1 增加到信号列表中，你可以通过这样修改上面的 trap 命令，从而删除那两个指定的文件：

```
$ trap "rm $WORKDIR/work1$$ $WORKDIR/dataout$$; exit" 1 2
```

现在，如果终端被挂起或者是 Ctrl+C 被按下，这些文件将被删除。

指定的 trap 命令如果包含多个命令，那么它们必须括在引号中。还要注意，当 trap 命令执行时，shell 会扫描一遍命令行，当接收到信号列表的其中一个信号时，也会再执行一次扫描。

所以在上面的例子中，当 trap 命令执行后，WORKDIR 和 $$ 的值将会被替换。如果你想要这种替换仅仅发生在信号值 1 或者 2 接收到的时候，你可以用单引号把多个命令引起来：

```
$ trap 'rm $WORKDIR/work1$$ $WORKDIR/dataout$$; exit' 1 2
```

忽略信号

如果 trap 命令的 command 字段是空的，那么当接收到指定的信号后，信号将被忽略。例如下面的一个命令：

```
$ trap '' 2
```

指定的中断信号将被忽略。当执行某些可能不想被打断的操作时，你可能会想忽略掉些特定的信号。按如下所示，你可以指定要忽略的多个信号：

```
$ trap '' 1 2 3 15
```

注意，如果要一个信号被忽略掉，那么第一个参数必须是指定的空，它不等同于如下的命令，下面的命令有自己单独的意义：

```
$ trap  2
```

如果你忽略了一个信号，那么你所有的子 Shell 也会忽略此信号。然而，如果你指定了接收一个信号所采取的动作，那么所有的子 shell 接收到此信号后会采取相同动作。

重置 Traps

当你改变了信号接收后的默认动作，你可以通过一个简单的省略第一个参数的 trap 命令将默认动作重置回来。

那么命令

```
$ trap 1 2
```

将信号 1 和 2 接收后采取的动作重置为其原有的默认动作。