

第 3 章, UNIX 基础知识

(翻译中出现的任何问题或错误, 请广大读者及时反馈给我: freebsdhandbook@163.com)

3.1 概要

这一章将介绍 UNIX 的基础知识和 FreeBSD 的功能。如果你还是个 FreeBSD 的新手, 你在寻求帮助之前, 应当先阅读这一章。

读完这章, 你会了解到:

1. UNIX 的文件访问权限是如何工作的。
2. 进程, 后台, 和信号是什么。
3. 什么是 shell, 如何改变你的默认登陆环境。
4. 如何使用基本的文本编辑器。
5. 如何阅读联机手册了解更多信息。

3.2 权限

FreeBSD, 仍然保持着 BSD UNIX 的传统, 它的基本原理仍然是以几个关键的 UNIX 概念为基础的。首先, 最显著的就是: FreeBSD 是一个多用户的操作系统。这个系统能够允许许多毫不相关的任务同时工作。对于每一个用户来讲, 系统都能非常可靠地分享和管理着来自不同硬件设备、内存、CPU 时钟的处理请求。

因为系统支持多用户, 所以系统管理的每一件事情都必须设置谁有读, 写, 和执行的权限。这个权限用八进制的形式来表示, 把它分成三部分: 文件拥有者, 文件拥有者所在组和其他成员。这种表示方法如下所示:

值	权限	目录列表
0	No read, no write, no execute	---
1	No read, no write, execute	--X
2	No read, write, no execute	-W-

3	No read, write, execute	-WX
4	Read, no write, no execute	r---
5	Read, no write, execute	r-x-
6	Read, write, no execute	rw--
7	Read, write, execute	rwX

在一个长目录中，用 `ls -l` 命令列一个清单，就显示了文件所有者、文件所有者所在组和其它成员的权限信息。这里就是它的表示方法：

```
-rw- r- - r- -
```

从左到右，第一个字符是一个特殊的字符，它告诉你这是一个规则文件，一个目录，一个特殊字符，一个块设备，一个套接字，还是其它的伪设备文件。下面的三个字符“rw-”指明了文件所有者的权限。下面的三个字符“r--”指明了文件所有者所在组的权限。最后面的三个字符“r--”，给出了其他用户的权限。一个破折号表示这个权限被关闭了。在这样一个设置下，意味着只有文件所有者才能读、写文件，组能够读文件，其它用户只能读文件。按照上表的指示，这个文件的权限应该是 644，每个数字都代表着这个文件权限的三个部分。FreeBSD 是如何来控制设备的权限的呢？事实上，FreeBSD 把绝大多数的硬件设备看作是一个文件，就象其它文件能够被打开，阅读，写数据一样。这些特殊的设备文件保存在 `/dev` 目录下。

目录也可以当作文件来看待。它们有读、写和执行的权限。这里的可执行跟其它文件的可执行有一些差别。例如，当一个目录被标记为可执行时，意味着这个目录可以被查找到，并且可以在这个目录下列目录。如果你想了解如何设置权限的话，你可以参考 `chmod` 命令的有关说明。

3.3 目录结构

FreeBSD 使用的文件系统决定了许多基础的系统操作，文件系统的层次结构是非常重要的。在所有的目录中，`root (/)` 是最重要的。系统启动时，这个目录是最先被挂上的，而且它包含了基本的系统信息。`root` 目录也包含了你能够挂上的其它文件系统的装载点。装载点是其它文件系统能够被连接到 `root` 文件系统的目录。这些目录通常被指定在

`/etc/fstab`下。`/etc/fstab`是一个几种文件的表，装载点能够被系统参考。在`/etc/fstab`中的绝大多数文件系统，能够在启动时被自动挂上，除非它们包含 `noauto` 选项。你可以参考 `fstab` 的指南，了解更多有关`/etc/fstab`文件格式和它所包含选项的信息。

现在，主要的目录结构就是这些：

目录	描述
<code>/</code>	文件系统的根目录
<code>/bin/</code>	单用户和多用户环境下，用户使用的工具。
<code>/boot/</code>	操作系统启动过程中使用的程序和配置文件。
<code>/boot/default s/</code>	默认的启动配置文件；看看 <code>loader.conf</code> 。
<code>/dev/</code>	设备节点；看看 <code>intro</code> 。
<code>/etc/</code>	系统配置文件和脚本。
<code>/etc/default s/</code>	默认的系统配置文件；看看 <code>rc</code> 。
<code>/etc/mail/</code>	象 <code>sendmail</code> 这样的用于邮件传输代理的配置文件。
<code>/etc/namedb/</code>	Named 配置文件；看看 <code>named</code> 。
<code>/etc/periodi c/</code>	通过 <code>via</code> 每天，每周，每月运行的脚本；看看 <code>periodi c</code> 的联机手册。
<code>/etc/ppp/</code>	Ppp 配置文件；看看 <code>ppp</code> 联机手册。
<code>/mnt/</code>	系统管理员使用的用作一个临时加载点的空目录。
<code>/proc/</code>	处理文件系统；看看 <code>procfs, mount procfs</code> 联机手册。
<code>/root/</code>	Root 帐户的主目录。
<code>/sbin/</code>	单用户和多用户使用的系统程序和管理工具。
<code>/stand/</code>	在独立环境下使用的程序。
<code>/tmp/</code>	临时文件。
<code>/usr/</code>	主要是用户的工具和应用程序。
<code>/usr/bin/</code>	通常是工具，程序工具和应用程序。
<code>/usr/incl ude/</code>	标准 C include 文件。
<code>/usr/lib/</code>	文档库
<code>/usr/libdata/</code>	各种工具的数据文件。
<code>/usr/libexec/</code>	系统守护程序和系统工具（被其他程序执行的）。

/usr/local/	本地执行的，库等。也是默认的放置 ports 框架的地方。Ports 的文档放在/usr/local/share/doc/port 中。
/usr/obj/	通过建构/usr/src 目录树产生的特定结构的目标树。
/usr/ports	FreeBSD ports collection 。
/usr/sbin/	系统守护程序和系统工具（由用户执行的）。
/usr/share/	独立结构的文件。
/usr/src/	本地源代码文件。
/usr/X11R6/	X11R6 发行的可执行程序，库等。
/var/	多种日志，临时文件，和 spool 文件。
/var/log/	多种系统日志文件。
/var/mail/	用户邮箱文件。
/var/spool/	多种打印机和邮件系统 spooling 目录。
/var/tmp/	系统重新启动之间保存的临时文件。
/var/yp	NIS 地图。

3.4 挂上和卸载文件系统

文件系统可以形象化为一棵树，root 就是根：象这样/，/dev, /usr 等。根目录下的其他目录也可能有分支，这些分支也会有它们自己的分支，象这样/usr/local 等等。之所以要采用这种分离的文件系统有很多原因。/var 包含 log, spool 和不同的临时文件。当然，这个目录也可能被塞满。塞满根文件系统不是一个好主意，所以从/分离出一个/var 比较好。

在其他的文件系统中，采用这种目录树结构的另一个原因是他们可能会在另外一个物理磁盘上，或在另外一个虚拟磁盘上，象 NFS 文件系统或 CDROM 驱动器。

3.4.1 fstab 文件

在系统启动的过程中，在/etc/fstab 中列出的文件系统会被自动地挂上。/etc/fstab 文件包含了一个以下面的格式排列的列表：

```
device /mount-point fstype options dumpfreq passno
```

Device 是一个设备名，就象上一章磁盘命名规则所提到的。

mount-point 是一个目录，在它上面挂上文件系统。

Fstype 是要挂上的文件系统类型。FreeBSD 的默认文件系统是 ufs。

Options 选项既可以是 rw 可读写文件系统，也可以是 ro 只读文件系统，可根据其它选项的要求来定义。在系统按顺序启动过程中，一个普通的选项是 noauto，它通常不会被挂上。其他的选项可参看 mount 的联机手册。

3.4.2 mount 命令

mount 命令通常是用来挂上文件系统的。它的最基本的操作格式：

```
# mount device mountpoint
```

它有很多的选择参数，但绝大多数是这样的：

mount options

-a

挂上/etc/fstab 中的所有文件系统，也可以用-t 来修改。

-d

除了挂文件系统外，可以做所有事情。

-f

强迫挂上文件系统。

-r

挂上只读文件系统。

-t fstype

以给定的文件格式挂上给定的文件系统。如果加上-a 参数，就只能挂上给定类型的文件系统。“ufs”是默认的文件类型。

-u

在文件系统上升级 mount 选项。

-v

更加详细。

-w

修改文件系统为读，写。

-0 选项提供了一个用逗号分割的选项列表，包括下面这些：

nodev

不要解释文件系统上的特殊设备。有用的安全选项。

noexec

不允许在这个文件系统上执行程序。有用的安全选项

nosuid

不要在文件系统上解释 setuid 或 setgid 标记。有用的安全选项。

3.4.3 umount 命令

在 `umount` 命令后可能会加载的参数可能是一个挂载点，一个设备名，或是 `-a`, `-A` 选项。所有加 `-f` 参数的形式将会被强迫卸载，`-v` 参数就会太冗长。`-a`, `-A` 选项用来卸载所有挂上的文件系统。

3.5 进程

FreeBSD 是一个多任务的操作系统。这意味着可以同时有多个程序一起运行。你运行的每个程序叫做一个 **进程**。你运行的每个程序都至少要启动一个进程，系统中时刻都有很多进程在运行，以维持系统的功能。

每一个进程都有一个叫做进程 ID 或 PID 的号码，每个进程也会有一个主人（或叫属主）和它所在的组。主人和组的信息用来决定进程能够打开什么文件和设备，使用文件的权限。绝大多数进程都有一个父进程。父进程是启动其他进程的进程，你运行的任何命令也是进程。每个进程将使用你的 shell 作为它的父进程。除这个进程之外的一个特殊的进程叫做 `init`。Init 总是第一个进程，所以它的 PID 总是 1。当 FreeBSD 启动时，`init` 会被内核自动启动。

有两个命令可以用来查看系统的进程，`ps` 和 `top`。`ps` 命令用来显示当前运行的进程的列表，还可以显示它的 PID，它们使用多少内存，它们启动的命令行等等。`top` 命令显示了所有运行的进程，每隔几秒就刷新一次，以至你可以动态地观察你电脑的工作状况。默认情况下，`ps` 只显示正在运行的你自己的命令。例如：

```
% ps
```

```
PID  TT  STAT      TIME COMMAND
298  p0  Ss       0:01.10 tcsh
7078 p0  S        2:40.88 xemacs mdoc.xsl (xemacs-21.1.14)
37393 p0  I        0:03.11 xemacs freebsd.dsl (xemacs-21.1.14)
48630 p0  S        2:50.89
/usr/local/lib/netscape-linux/navigator-linux-4.77.bi
48730 p0  IW       0:00.00 (dns helper) (navigator-linux-)
72210 p0  R+       0:00.00 ps
390  p1  Is       0:01.14 tcsh
7059 p2  Is+     1:36.18 /usr/local/bin/mutt -y
6688 p3  IWs     0:00.00 tcsh
10735 p4  IWs     0:00.00 tcsh
20256 p5  IWs     0:00.00 tcsh
262  v0  IWs     0:00.00 -tcsh (tcsh)
270  v0  IW+     0:00.00 /bin/sh /usr/X11R6/bin/startx -- -bpp 16
280  v0  IW+     0:00.00 xinit /home/nik/.xinitrc -- -bpp 16
284  v0  IW      0:00.00 /bin/sh /home/nik/.xinitrc
285  v0  S       0:38.45 /usr/X11R6/bin/sawfish
```

正如你在这个例子中看到的，ps 的输出形式是根据数字的排列顺序来组织的。PID 是先前讨论的进程 ID。PID 从 1 开始，最高到 65536。当你完成以后，再从头开始。TT 显示了正在运行的 tty，也可以安全地略过。

Ps 支持许多不同的选项来改变显示的信息。最有用的设置是 `auxww`。一个有关所有运行的进程的显示信息，不仅仅是你自己的。U 用来显示进程的使用者和内存的使用者。X 用

来显示后台运行的进程信息，`ww` 用来显示所有的命令行，一旦它比较长而超出屏幕时，就会截取其中一段。

Top 的输出也比较熟悉。可以看看下面的例子；

```
% top

last pid: 72257; load averages: 0.13, 0.09, 0.03 up 0+13:38:33
22:39:10

47 processes: 1 running, 46 sleeping

CPU states: 12.6% user, 0.0% nice, 7.8% system, 0.0% interrupt, 79.7% idle

Mem: 36M Active, 5256K Inact, 13M Wired, 6312K Cache, 15M Buf, 408K Free

Swap: 256M Total, 38M Used, 217M Free, 15% Inuse

  PID USERNAME PRI NICE  SIZE  RES STATE   TIME  WCPU   CPU COMMAND
 72257 ni k      28  0 1960K 1044K RUN     0:00 14.86% 1.42% top
 7078 ni k      2  0 15280K 10960K select 2:54 0.88% 0.88%
xemacs-21.1.14
 281 ni k      2  0 18636K 7112K select 5:36 0.73% 0.73% XF86_SVGA
 296 ni k      2  0 3240K 1644K select 0:12 0.05% 0.05% xterm
48630 ni k      2  0 29816K 9148K select 3:18 0.00% 0.00%
navigator-linu
 175 root      2  0 924K 252K select 1:41 0.00% 0.00% syslogd
 7059 ni k      2  0 7260K 4644K poll 1:38 0.00% 0.00% mutt
...
```

整个输出被分为两节。头上(最初 5 行)显示运行着的进程的 PID,系统的平均负载(反映系统的繁忙程度),系统的正常运行时间(从上次重起以来的时间),和当前时间。其它的图显示了当前运行了多少个进程,有多少内存和交换空间已经被占用,系统在不同 CPU 状态之间切换需要花费多长时间。

下面有一连串的竖直排列的 ps 命令的输出信息。你可以看到 PID,用户名,CPU 时钟的花费数量,正在运行的命令。Top 也默认地显示了进程所花费掉的内存空间的数量。这可以被分成两列,一列针对所有的内存数量,一列针对常驻内存数量。整个内存数量就是应用程序需要多少内存,常驻内存是在此刻使用的内存数量。在这个例子中,你可以看到 Netscape 使用了 30M 内存,当前只使用了 9MB。Top 每隔两秒自动刷新一次;这可以通过加上 s 选项来修改。

3.6 守护程序,信号和杀死进程

当你使用一个编辑器的时候,你可以要求它加载一些文件。因为它们提供了这样的功能,而且它们是与一个终端连在一起的。但,有一些程序却不能让用户输入信息,它们是不与终端连在一起的。例如,一个 web 服务器花费所有的时间来回应用户的请求,它通常不需要你有任何的输入。从一个站点到另一个站点传送 email 是另外一个这种类型的应用例子。我们叫这些程序为 daemons (守护程序)。Daemon 具有希腊神话的特征;今天的许多 web 服务器和 mail 服务器都使用这些。有时你可能需要与一个守护程序进行通讯。这些通讯就叫做 signal。你可以通过给它发送 signal (或是运行进程)来与守护程序通讯。如果你发送的 signal 有很多(有一些有特殊的用处),其它的与应用程序集成在一起,应用程序的文档会告诉你如何解释 signal。你可以发一个 signal 给你的一个进程。如果你设法发送一个 signal 给其他人的进程,它就会被忽略。Root 用户除外,它可以发送 signal 给每一个进程。

如果一个应用程序写入错误,就会设法访问内存,FreeBSD 会给进程发送 *Segmentation Violation* signal (SIGSEGV)。如果一个应用程序使用 alarm 系统来发出警告,那一段时间以后,它也会发送警告信号。

两个信号可能会中断一个进程, SIGTERM 和 SIGKILL。SIGTERM 是一个比较友好的杀死进程的方法;这个进程也会捕获信号,以便让你关机,关闭可能打开的一些日志文件。在

关机之前，通常需要完成当前正在做的工作。有时，如果它是处在一个不能打断的任务中，一个进程可以忽略 SIGTERM。

SIGKILL 无法被进程忽略。它会发出这样的信号 “ I do not care what you are doing, stop right now ”。如果你发送 SIGKILL 给一个进程，FreeBSD 将会停止那个进程。

你可能要用到的其他 signal 是 SIGHUP, SIGUSR1 和 SIGUSR2。这些是普通用途的 signal。当他们被发送时，不同的应用程序将做不同的事情。

建议你改变一下你的 web 服务器的配置文件----你最好告诉 web 服务器重新读一下它的配置文件。你需要重起 httpd, 但这将会在你的 web 服务器上增加一些消耗，而这可能是你不太欢迎的。绝大多数守护程序通过重新读取它们的配置文件来对 SIGHUP 信号作出回应。不同的后台程序将有不同的行为。所以，要带着问题来阅读守护程序的联机手册。

可以使用 kill 命令来发送 signal，例如：

发送一个 signal 给处理器

这个例子显示了如何发送一个信号给 inetd。Inetd 的配置文件是 */etc/inetd.conf*。当它接收到 SIGHUP 时，inetd 将重新读取这个配置文件。

1. 寻找你要发送信号的进程 ID。可以使用 ps, grep 命令。Grep 命令被用来搜索输出，寻找你要指定的字符。这个可以有一个普通用户来执行，而 inetd 需要是 root 用户，所以 ps 必须带上 ax 选项

```
% ps -ax | grep inetd
```

```
198 ?? IWs 0:00.00 inetd -wW
```

这儿，inetd 的 PID 是 198。有时，grep inetd 命令也需要出现在这个输出中。这是因为 ps 必须要找到当前运行的进程的列表。

2. 使用 kill 来发送信号。因为 inetd 只有 root 用户才能运行，你必须使用 su 来变成一个 root 用户。

```
% su
```

```
Password:
```

```
# /bin/kill -s HUP 198
```

就象普通的 unix 命令一样，如果它成功执行，kill 将不会输出任何信息。如果你想发送一个 signal 给一个进程，你会看到“kill: PID: Operation not permitted”。如果你打错了 PID，你有可能把信号发错了某个进程，这样会很糟糕，也有可能把信号发了一个当前不在使用的 PID，你将可能看到“kill: PID: No such process”这样的信息。

为什么使用/bin/kill：许多 shell 提供了内建命令 kill；shell 将直接发送信号，比运行/bin/kill 要好。这点非常有用，但不同的 shell 需要用不同的语法来指定信号的名字。

重点：在系统中随意地杀死进程是个坏主意。特别地，init 进程 ID 是 1，非常特殊。运行/bin/kill -s KILL 1 是一个快速关闭你系统的方法。在你键入 kill 之前，请你仔细检查你执行的 kill 的参数。

3.7 Shell

在 FreeBSD 中，许多工作是通过一个叫命令解释器{俗称“外壳 (shell)”}的命令行接口来完成的。shell 的主要工作是接收输入的命令然后执行它们。许多 shell 也能够用来帮助完成每天的工作，如：文件管理，文件查找，命令行编辑，宏命令，以及其它环境设置。FreeBSD 有许多种 shell，如：sh, Bourne Shell 和 csh, C-shell。许多其它的 shell，如 tcsh, bash 拥有更强大的功能，你可以在 FreeBSD 的软件包中找到。

你使用哪个 shell？那还正是个问题。如果你是一个 C 程序员，你可能觉得象 tcsh 这种 C 类型的 shell，用起来比较舒服。如果你使用 Linux 或你是一个 UNIX 的新手，你可能会试一试 bash。这里要指出的是每一种 shell 都有它自己的特点，你可以根据你的喜好自由地选择。Shell 的一个基本特征是文件名的自动补充功能。通常在你输入一个命令或文件名时，你先输入几个字母，然后按 TAB 键，命令或文件名会自动补上剩下的字母。例如：我有两个文件：foobar 和 foo.bar。我要删除 foo.bar。所以我就输入：rm foo[TAB]. [TAB]。Shell 就会打出 rm foo[BEEP]. Bar。[BEEP]是铃声。它能够告诉我，因为匹配的问题 shell 不能自动补充文件名。因为 foobar 和 foo.bar 同时以 fo 开始，但它无法完成 foo。一旦我输入.，然后键入 TAB，shell 就能补充完文件名。

Shell 的另外一个功能就是环境变量。环境变量是存储在 shell 环境空间中的可变键对。这个空间能够被 shell 的任何程序调用，而且包含了许多程序配置。这儿是一个普通环境变量的列表：

变量	详细说明
USER	现在登陆的使用者名称。
PATH	以冒号分隔的目录列表以便寻找执行文件的路径。
DISPLAY	X11 显示连接的网络名称,如果有的话。
SHELL	目前用的 shell 。
TERM	使用者终端的名称。用来决定终端机的能力。
TERMCAP	完成几个终端功能的终端退出代码的数据库记录。
OSTYPE	操作系统的种类，如 FreeBSD。
MACHTYPE	现在系统所用的 CPU。
EDITOR	使用者喜欢的编辑器。
PAGER	使用者喜欢的文字呼叫器。
MANPATH	以冒号分隔的目录以便寻找联机手册。

在各 shell 之间，设置一个环境变量稍微有点不同。例如，象 tcsh 和 csh 的 C 风格 shell，你可以使用 `setenv` 来设置或查看环境变量。而在象 sh 和 bash 的 Bourne shell 下，你可以使用 `set` 和 `export` 来查看和设置环境变量。例如，在 csh 或 tcsh 下，设置或修改环境编辑器，可以把编辑器设置成 `/usr/local/bin/emacs`：

```
%setenv EDITOR /usr/local/bin/emacs
```

在 Bourne shell 下：

```
%export EDITOR="/usr/local/bin/emacs"
```

你也可以在命令列用 `$` 放在变量的前面来取得环境变量。echo `$TERM` 就会显示出 `$TERM` 的设置值，因为 shell 取得了 `$TERM` 并把它传给 echo 显示出来。

Shell 里有很多特殊字符代表着一些资料，我们叫做 meta-characters。最常用的就是* 这个字符，它代表文件名的任何字符。这些 meta-characters 会被用在文件名称的全域样式上。举例来说，键入 echo *是和 ls 有同样的效果，因为 shell 将所有符合*的文件传到命令列给 echo 显示出来。

为了防止这些特殊的字符被 shell 转译，我们可以在前面放一个反斜线(\)让它们跳出来。echo \$TERM 会显示出你所设置的终端机。而 echo \ \$TERM 就会直接显示出 \$TERM 这几个字母。

3.7.1 改变你的 shell

改变你的 shell 的最简单的方法是用 chsh 命令。运行 chsh 就可以把你带入系统设置的编辑器中。如果编辑器没有设置的话，你就可以用 vi 来代替。直接在此改变“shell:”。你可以给 chsh 加上-s 的参数；这样，就不需要你输入一个编辑器来设置 shell 了。例如，如果你想把你的 shell 设置成 bash，你只要这样做：

```
% chsh -s /usr/local/bin/bash
```

运行不带参数的 chsh，编辑 shell 将同样能达到目的。

注意：你使用的 shell 出现在 */etc/shell* 文件中。如果你在安装软件的时候，已经安装了一个 shell，那么这个就已经做好了。如果你是手工安装 shell，那你就必须做。例如，如果你手工安装 bash，只要把它放在 */usr/local/bin* 中，你只要这样做：

```
#echo "/usr/local/bin/bash" >>/etc/shells
```

然后再执行 chsh。

3.8 文本编辑器

在 FreeBSD 中，许多配置信息都是通过编辑一个文本文件来完成的。所以，熟悉一个文本编辑器是非常必要的。FreeBSD 带有许多的编辑器，可以在 ports collection 中找到。

最容易和最简单的编辑器就是 `ee` 了，它非常容易掌握。要启动 `ee`，你只需要在命令行键入 `ee` 文件名。例如，要编辑文件 `/etc/rc.conf`，只要键入 `ee /etc/rc.conf`。一进入 `ee`，许多编辑功能就都列在屏幕的顶部。`^` 字符是键盘的 `ctrl` 键，所以 `^e` 键盘的 `ctrl` 键加上 `e` 键。要离开 `ee`，只要键入 `escape` 键，然后选择离开。编辑器会提示你保存刚才的修改。FreeBSD 也带有很多功能强大的编辑器，象 `vi`，`emacs` 和 `vim` 等。这些编辑器都有很强大的功能，你需要认真地学习。

3.9 设备和设备节点

一个设备大都是一个系统中与硬件相关的活动所使用的，包括磁盘，打印机，图形卡和键盘。当 FreeBSD 启动时，显示的大都是检测到的设备。你可以查看 `/var/run/dmesg.boot` 文件来看看启动信息。

例如，`acd0` 是第一个 IDE CDROM 驱动器，而 `kbd0` 则是键盘。在 `unix` 系统中的绝大多数设备必须呼叫设备的节点才能来访问一个特殊文件，这些都放在 `/dev` 目录下。

3.9.1 创建设备节点

当你在系统中添加了一个新的设备或编译支持额外的设备时，需要重建一个设备驱动。

3.9.1.1 MAKEDEV 脚本

在系统中没有 `DEVFS`，设备节点可以使用 `MAKEDEV` 脚本来创建：

```
# cd /dev

# sh MAKEDEV ad1
```

这个例子中将创建第二个 IDE 驱动器的设备节点。

3.9.1.2 devfs (设备文件系统)

设备文件系统或 `devfs`，提供了访问内核设备的命名方式。`Devfs` 获得了这个特殊的文件系统，代替了创建和修改设备节点。`devfs` 将在 FreeBSD 5.0 中作为默认使用项。

3.10 更多信息

3.10.1 联机手册

绝大多数 FreeBSD 的参考文档是以手册的形式出现的。系统的每个程序都有一个简短的联机手册。这些手册能够用 man 命令来阅读。例如：

```
% man command
```

command 是你希望了解的命令的名字。例如，要知道 ls 的用法：

```
% man ls
```

联机手册被分成好几节：

1. 用户命令
2. 系统呼叫和错误编号
3. 在 C 库中的功能
4. 设备驱动器
5. 文件格式
6. 游戏和其他娱乐方式
7. 其它一些凌乱的信息
8. 系统维护和操作命令
9. 内核开发

在一些示例中，同一个主题可能会出现在手册的很多地方。例如，chmod 用户命令和 chmod 系统呼叫。在这个例子中，你可以告诉 man 命令，你要指定哪一节：

```
% man 1 chmod
```

这将显示用户命令 chmod 的参考文档。参考一个联机手册的特殊的章节通常被附带在文章的后面，所以第一个 chmod 引用了 chmod 用户命令和第二个 chmod 引用了系统呼叫。

如果你知道命令的名字，就比较好，你只需要知道如何使用。但如果你无法想起命令的名字是什么？你可能要使用 man 加上 -k 选项在命令描述中搜索关键字：

```
% man -k mail
```

这个命令将出现在一个带有关键字 mail 的命令列表中。这与使用 apropos 命令具有相同的功能。所以，你可以在 /usr/bin 中寻找所有的奇特的命令，只要这样做：

```
% cd /usr/bin
```

```
% man -f *
```

或

```
% cd /usr/bin
```

```
% whatis *
```

可以达到同样的目的。

3.10.2 GNU Info 文件

FreeBSD 包括许多自由软件基金会提供的应用程序和工具。除了联机手册之外，这些程序都提供叫做 info 文件的超文本文件，它可以用 info 命令来阅读，或者如果你有 emacs，可以使用 emacs 的 info 模式来阅读。

使用 info 命令：

```
% info
```

要看看概要介绍，可以键入 h。要看看快速命令参考，可以键入？。