

第16章 文件系统

(欢迎提出意见和建议：frebsdhandbook@163.com)

UNIX操作系统最久远的概念之一是它的文件系统，存储数据的方式。虽然绝大多数其他的操作系统其后一直沿用它，包括Microsoft平台，但没有哪一个能够完美地接近它的执行机制。在这一章，我们将看看那对你意味着什么。

文件系统

UNIX和Microsoft环境都用文件 (*file*) 来存储数据，并按顺序存放在目录中。一个文件可能是一个目录：也就是，它可能包含其他的文件。

文件名

UNIX和Microsoft之间的不同是从文件名 (*file name*) 开始的。传统的Microsoft文件名是很严格的：一个文件名包含8个字符，后面跟着一个点和另三个字符（也叫做扩展名）。可以用来组成一个文件名的字符有严格的限制，大写或小写字母都是一样的（内部处理时，Microsoft把名称转换成大写的）。目录之间用反斜线符号 (\) 隔开，这是与C编程语言的习惯相冲突的——看看第218页了解更多细节。

与此相反，UNIX文件名要灵活的多。它们可以包含任何除了斜线 (/) 以外的字符，这个斜线通常用来指出目录概念，它们最长可以有255个字符。在一些比较老的UNIX和早期的Linux版本中，文件名被限制在14个字符。

大写和小写的字母有不同的含义，所以在UNIX中*foo*, *FOO*和*Foo*是三个不同的名称。

权限

由于一个UNIX系统可能会被很多人使用，所以它包含了一种保护数据不被未经验证的人访问的方法。每个文件有三类与用来描述谁可以用什么方法访问的情况相关联的信息：

- 文件属主/主人 (*file owner*)，拥有这个文件的人的用户ID。
- 文件所在的组 (*file group*)，拥有这个文件的组的组ID。
- 文件属主，文件所在的组和其他人都可以处理这个文件。可能的动作是读，写或执行。

例如，你可能有一个访问私人数据的程序，你要确保只有你可以执行它。你可以通过设置权限来这样做，以便只有文件的属主可以执行它。或你可以在发展中有一个文本文档，你要确保你是唯一一个可以修改它的人。另一方面，与你一起工作的人也需要能够参考这个文档。你设置权限，以便只有文件属主可以写它，文件属主和它所在的组可以读它，因为它不是为公众准备，所以不允许其他人可以访问它。

通常，权限是通过三组rwx来表示的：r代表读权限，w代表写权限，x代表执行权限。三个组分别表示文件属主，文件属主所在的组，和其他用户的权限。如果权限被关闭，就用 (-) 来表示。因而，我们上面讨论的程序的权限是r-x-----（我可以读和执行这个程序，其他人不能做

The Complete FreeBSD(4th)

任何事情)。针对草拟文档的权限是rwr-----（我可以读和写，所在组可以读，其他人不能访问它）。

典型的FreeBSD文件访问权限对程序是rwxr-xr-x，对其他系统文件是rw-r--r--。在一些情况下，你将发现其他权限也是需求的。例如，文件`~/rhosts`，它被一些为用户确认的网络程序使用，可能包含用易读形式表示的用户口令。要帮助确保其他人无法读取它，网络程序将拒绝读它，除非它的权限是rw-----。UNIX中的绝大部分系统文件可能被描述成不正确的权限，所以你应该特别注意它们。

除了这些访问权限之外，可执行的也可以用两个位设置来指定进程的访问权限。如果`setuid`（设置user ID）位被设置，进程将总是运行，好像它已经被自己启动一样。如果`setgid`（设置group ID）位被设置，它将运行好像它已经被它的组启动。这通常被用来启动需要访问用户不能直接访问的资源的系统程序。我们将在第300页用`ps`命令看看这样的例子。

`ls`通过设置权限字符的第三个字母用s代替x来表示`setuid`位。相似的，它通过设置权限字符的第六个字母用s代替x来表示`setgid`位。

除了这个访问信息，权限包含了一个描述它表示哪种文件的字符。第一个字母可能是一个-，这指明了一个规则文件，字母d代表目录，或字母b或c代表一个设备节点。我们将在第16章看看设备节点。也有其他很多的字母没有使用。看看`ls`的联机手册了解一个完整的列表。

要列出和显示权限，使用带上-l选项的`ls`命令：

```
$ ls -l
total 2429
-rw-rw-r-- 1 grog wheel 28204 Jan 4 14:17 %backup%~
drwxrwxr-x 3 grog wheel 512 Oct 11 15:26 2.1.0-951005-SNAP
drwx----- 4 grog wheel 512 Nov 25 17:23 Mail
-rw-rw-r-- 1 grog wheel 149 Dec 4 14:18 Makefile
-rw-rw-r-- 1 grog wheel 108 Dec 4 12:36 Makefile.bak
-rw-rw-r-- 1 grog wheel 108 Dec 4 12:36 Makefile~
-rw-rw-r-- 1 grog wheel 0 Dec 4 12:36 depend
-rw-rw-r-- 1 root wheel 1474560 Dec 14 17:03 deppert.floppy
-rwxr-xr-x 1 grog wheel 100 Dec 19 15:24 doio
-rwxrwxr-x 1 grog wheel 204 Dec 19 15:25 doiovm
-rwxrwxr-x 1 grog wheel 204 Dec 19 15:16 doiovm~
-rwxr-xr-x 1 grog wheel 115 Dec 26 08:42 dovm
-rwxr-xr-x 1 grog wheel 114 Dec 19 15:30 dovm~
drwxr-xr-x 2 grog wheel 512 Oct 16 1994 emacs
drwxrwxrwx 2 grog wheel 512 Jan 3 14:07 letters
```

这个格式显示了下面的信息：

- 首先是，我们已经看到的权限。
- 然后，连接数（link count）。这是硬连接到文件的数量。对于一个规则文件，这通常是1，但目录至少是2。我们将在第302页看看连接。

- 接着是属主和所在组的名称和用字节表示的文件大小。你将注意到文件`deppert.floppy`属于`root`。这大概是一个意外，它可能会导致出现问题。顺便提一下，看看文件名和它的大小，很明显这是一个3½"寸软盘的映象，也就是说，是一个完整的软盘拷贝。
- 日期通常是文件被最后修改时的日期。对`ls`命令使用`-u`参数，你可以列出文件被访问的最后日期。
- 最后是文件的名称。正如你在这个例子中看到的，名称可能是各式各样的。
许多权限是很有意思的。目录都有`x`（执行）权限位设置。为了能够访问目录中的文件，这是必须的——那是术语`execute`定义一个目录的方法。如果我重新安排执行权限，我仍可以列出文件的名称，但我不能访问它们。

我是唯一一个可以访问目录`Mail`的人。这是一个邮件目录的普通权限。

修改文件权限和属主

经常，你可能要修改文件权限或属主。UNIX提供三个程序来这样做：

- 要修改文件的属主，使用`chown`。例如，要修改文件`deppert.floppy`的所有权，这在上面的列表中属于`root`，`root`将键入：

```
# chown grog deppert.floppy
```

注意，这个操作必须由`root`来执行。

- 要修改文件所在的组，使用`chgrp`，这与`chown`使用同样的方法来工作。要修改组所有权为`lemis`，你可以键入：

```
# chgrp lemis deppert.floppy
```

`chown`也可以修改属主和组。与前面两个例子不同，你可以键入：

```
# chown grog:lemis deppert.floppy
```

这将修改`grog`的属主，像以前一样，也要把组修改成`lemis`。

- 要修改权限，使用`chmod`程序。`chmod`有许多不同的格式，但不幸的是9字符表示不是它们中的一个。看看`chmod`的联机手册了解更多的信息，但你可以用表15-1所显示的格式只执行你需要的东西：

表15-1 `chmod`权限代号

规范	作用
<code>go-w</code>	拒绝给组和其他人写的权限。
<code>=rw,+X</code>	设置读和写权限为默认值，但保留当前设置的执行权限。
<code>+X</code>	如果它已经能够被任何人搜索/执行，创建一个被每个人搜索/执行的目录或文件。

u=rwx,g=rx	创建一个能被任何人读/执行的和只能被属主写的文件。
gO=	清除所有组和其他人的模式位。
g=u-w	设置与用户位相等的组位，但清除组的写入位。

新文件的权限

这并没有告诉我们新文件将使用什么权限。错误的选择可能会引起灾难。例如，如果文件是用权限rwxrwxrwx自动创建的，任何人都可以用任何方式访问它们。另一方面，用r-----创建它们可能导致需要做很多工作来设置你需要它们做的事情。UNIX用一个叫做umask(*User mask*)的程序来解决这个问题。这是一个默认的没有权限：它指出了哪个权限位不被允许。

好像这不会引起很大的混淆，它已经在八进制数的系统中被指定了，在这里正确的数字是0到7。每个八进制数表示3个位。与此相反，更普通的十六进制系统使用16个数字，0到9和a到f。UNIX的早期版本只运行在使用八进制数字的系统上，因此权限只能用三位，它只能理解用八进制表示的umask值。

一个例子：默认的，你要创建任何人都能读的文件，但只有你可以写。你设置mask成022。这对应于二进制位000010010。

第一位的0是必须，它用来指定这个数是十六进制的，不是拼凑的三个数字。如果你要设置权限，以便默认的没有人能读，你应当把它设置成0222。一些shell自动假设数字是八进制的，所以你可以忽略0，但那不是好习惯。相对应的位是0表示这个权限被允许：

rwxrwxrwx	<i>Possible permissions</i>
000010010	<i>umask</i>
rwxr-xr-x	<i>resultant permissions</i>

默认的，文件被不带x位的创建，但是目录需要用允许的x位创建，所以对于这个umask，一个文件将用权限rw-r--r--创建。

umask是一个shell命令。要设置它，只要键入：

```
$ umask 022
```

在你的shell初始化文件中设置这个更好——看看第239页了解更多细节。

创建一个太过严格的umask要小心一点。例如，你将在用一个像377这样的umask时会遇到很多问题，这可以创建你只能读，而其他人只能访问的文件。如果你不接受x位，你将不能访问你创建的目录，你也不能运行你编译的程序。

使一个程序可执行

文件权限会导致出现一个问题，应当值得认真注意。许多操作系统要求一个可执行的程序有一个特定的命名惯例，如COMMAND.COM或FOO.BAT，在MS-DOS中，这分别表示一个特定类型的二进制可执行和脚本文件。在UNIX中，为了能执行一个程序，你不需要一个特定的后缀，

但它必须设置x位。

有时这个位是关闭的，例如，如果你用`ftp`通过网络拷贝。结果看起来是这样的：

```
$ ps
bash: ps: Permission denied
$ ls -l /bin/ps
-r--r--r-- 1 bin kmem 163840 May 6 06:02 /bin/ps
$ su
you need to be super user to set ps permission
Password:
password doesn't echo
# chmod +x /bin/ps
make it executable
# ps
now it works
PID TT STAT TIME COMMAND
226 p2 S 0:00.56 su (bash)
239 p2 R+ 0:00.02 ps
146 v1 Is+ 0:00.06 /usr/libexec/getty Pc ttyv1
147 v2 Is+ 0:00.05 /usr/libexec/getty Pc ttyv2
# ^D exit su
$ ps
ps: /dev/mem: Permission denied hey! it's stopped working
```

哼？它只工作在`su`下，当我变成一个纯粹的凡人时就停止工作？这里将发生什么？

可能有像`ps`这样的程序的另外一个问题：一些版本必须能够访问一些特殊的文件，在这种情况下是`/dev/mem`，一个标记系统内存的特殊文件。要这样做，我们必须设置`setgid`位，`s`。我们需要先变成超级用户：

```
$ su
you need to be super user to set ps permission
Password:
password doesn't echo
# chmod g+s /bin/ps
set the setgid bit
# ls -l /bin/ps
see what it looks like
-r-xr-sr-x 1 bin kmem 163840 May 6 06:02 /bin/ps
# ^D
exit su
$ ps
now it still works
PID TT STAT TIME COMMAND
226 p2 S 0:00.56 su (bash)
239 p2 R+ 0:00.02 ps
146 v1 Is+ 0:00.06 /usr/libexec/getty Pc ttyv1
147 v2 Is+ 0:00.05 /usr/libexec/getty Pc ttyv2
```

在这个例子中，最后结果中的权限是对`ps`的正确权限。它不可能为每个标准的程序检查权限。如果你猜测你设置了不正确的权限，参考一下Live Filesystem CD-ROM上的文件权限设置。

`setuid`和`setgid`程序可能有一个安全问题。如果叫做`ps`的程序是其他一些东西，如特洛伊木马（Trojan Horse），将会发生什么？我们设置权限来允许它侵入系统。结果，比较新版本的FreeBSD可以选择`ps`方法来做这个工作，它不再需要设置`setgid`。

连接

在UNIX中，文件通过`inodes`来定义，你不能直接访问在磁盘上的结构。它们包含元数据（`metadata`）——所有有关文件的信息，如属主，权限和时间戳。它们不能包含的是你把它看

作产生一个文件的东西：它们没有任何数据，它没有任何名称。而是，inode包含有关在磁盘上定位数据块的信息。它不知道有关名称的任何东西：那是目录的工作。

一个目录是简单的一个特殊类型的文件，它包含名称和inode号码的列表：换句话说，它们分配一个名字给一个inode，然后给一个文件。可以有不止一个名字可以指向同一个inode，所以文件可以有不止一个名称。这个名称与inode之间的关系叫做一个连接（link），有时会混淆为hard link。

因为inode编号与文件系统相关，文件必须与引用它们的目录在同样的文件系统中。

目录记录互相之间是独立的：每个都指向inode，所以它们完全是相等的。Inode包含一个连接数（link count），用来跟踪有多少目录记录指向它：当你删除最后的记录时，文件数据被删除。

另外，symbolic links有时叫做soft links，是不受同样文件系统限制的（甚至不是同一个系统！），它们引用另一个文件名，不是文件自己。如果你删除一个文件，这种差异就很明显：如果文件是硬连接的，其他名称仍存在，你可以通过它们访问文件。如果你删除了一个有指向它的符号连接，文件将丢失，符号连接就根本找不到它。

可以很容易地决定使用哪种连接——看看《UNIX Power Tools》一书了解更多细节。

目录结构

虽然Microsoft平台有一个层次式的目录结构，但是目录名称没有标准化：很难知道一个特殊的程序或数据文件可能是在哪儿。虽然每个厂商喜欢做一些修改以确保它们不完全兼容，但UNIX系统还是有一个标准的层次式目录。在它的发展过程中，UNIX已经好几次改变了它的目录结构，但它仍然比Microsoft的标准更加完善。最近，大概最明显的变化，发生在System V.4和4.4BSD，这两个几乎作了同样的修改。

几乎每一个版本的UNIX都至少有两个文件系统（root file system）和/usr，即使它们只在一个简单的磁盘上。这个安排被认为比只用一个简单文件系统更加可靠：只有一个文件系统很可能会由于严重损坏，而导致它根本不能被挂上，你必须从磁带备份中读取，或使用像fsck或fsdb这样的程序来修补它们。我们已经在第97页讨论了这个问题，我们建议在与/一样的文件系统上加上/usr。

标准目录

文件系统的物理划分不会影响目录的名称或内容，这是标准的。表15-2给出了标准FreeBSD目录的情况。

The Complete FreeBSD(4th)

表15-2 FreeBSD目录结构

目录名称	用法
/	Root文件系统。包含内核，启动程序，和其他文件系统的加载点。它不包含其他东西。
/bin	在系统启动时通常必须使用的可执行程序。名称最初是binary的缩写，但这里的许多文件是shell脚本。
/boot	启动系统时使用的文件。从FreeBSD版本5开始这个目录将包含内核和它相关的kld。
/cdrom	CDROM驱动器的加载点。
/compat	用于包含模拟其他系统的代码，如Linux。
/dev	设备节点的目录。这个名称是device的缩写。我们将在第311页详细看看这个目录的内容。
/etc	在系统启动时使用的文件。不像System V，/etc不包含内核编译文件，这不是系统启动时必须的。不像早期的UNIX版本，它也不包含可执行文件——它们已经被移到/sbin。
/mnt	一个针对软盘和其他临时文件系统的加载点。
/modules	包含kernel loadable modules的目录，在运行时可以被启动的内核部分。从FreeBSD版本5开始，在这个目录中的文件将被移到/boot/kernel。
/proc	进程文件系统（process file system）。这个目录包含当前提到的运行着的进程的虚拟内存的伪文件。
/root	用户root的主目录。在传统的UNIX文件系统中，root的主目录是/，但这有点杂乱。
/sbin	系统启动时必须的系统可执行程序。这些典型的存储在/etc中的系统管理文件。
/stand	standalone程序的目录。事实上，绝大多数程序是一样的文件，/stand/sysinstall，我们已经在第6章详细讨论了。从FreeBSD版本5开始，sysinstall将被移到/sbin，为了更容易地在一个live system上使用它。
/usr	第二个文件系统。看看我们上面的讨论。
/usr/X11R6	X11视窗系统。
/usr/X11R6/bin	可执行的X11程序。
/usr/X11R6/include	X11编程的头文件。

The Complete FreeBSD(4th)

<i>/usr/X11R6/lib</i>	针对X11的库文件。
<i>/usr/X11R6/man</i>	针对x11的联机手册。
<i>/usr/bin</i>	系统启动时不需要的标准可执行程序。你使用的绝大多数程序将存储在这里。
<i>/usr/games</i>	游戏。
<i>/usr/include</i>	给程序员使用的头文件。
<i>/usr/lib</i>	给程序员使用的库文件。FreeBSD没有目录 <i>/lib</i> 。
<i>/usr/libexec</i>	不能被用户直接启动的可执行文件,例如C编译器(由 <i>/usr/bin/gcc</i> 启动)或由init启动的 <i>getty</i> 程序。
<i>/usr/local</i>	不是操作系统一部分的额外程序。它在已有的 <i>bin</i> , <i>include</i> , <i>lib</i> , <i>man</i> , <i>sbin</i> , 和 <i>share</i> 子目录中并列 <i>/usr</i> 目录。这是你存放安装程序的地方。
<i>/usr/sbin</i>	系统启动时不需要的系统管理程序。
<i>/usr/share</i>	其他只读文件,主要是提供信息的。子目录包含 <i>doc</i> (FreeBSD文档), <i>games</i> , <i>info</i> (GNU <i>info</i> 文档), <i>locale</i> (国际化信息)和 <i>man</i> (联机手册)。
<i>/var</i>	经常修改数据的文件系统,如 <i>mail</i> , <i>news</i> 和日志文件。如果 <i>/var</i> 不是一个分离的文件系统,你应当在另一个文件系统上创建一个目录和符号连接 <i>/var</i> 。
<i>/var/log</i>	系统日志文件的目录。
<i>/var/mail</i>	在这个系统上接受邮件的地方。
<i>/var/spool</i>	Spool数据,如等待打印的数据(<i>/var/spool/lpd</i>), <i>/var/spool/mqueue</i> (正在发出的邮件), UUCP数据(<i>/var/spool/uucp</i>), 和 <i>/var/spool/ftp</i> (匿名FTP)。
<i>/var/tmp</i>	临时文件。

文件系统的类型

FreeBSD支持很多种类型的文件系统。最重要的是：

- *ufs*是UNIX File System。所有本地磁盘文件都是这个类型。
- *Cd9660*是用启用UNIX类文件名的叫做*Rock Ridge Extensions*的ISO 9660 CD-ROM格式。可以为所有的CDROM使用这个文件系统类型,即使它们没有Rock Ridge Extensions。
- *nfs*是Network File System,一种通过网络共享文件系统的方式。我们将在第31章讨论这个问题。
- 你可以用*ext2fs*文件系统访问Linux文件系统。

你可以用`msdos`和`ntfs`文件系统访问Microsoft文件。看看`mount_msdos`和`mount_ntfs`的联机手册了解更多细节。

挂上文件系统

Microsoft系统通过在启动时分配的字母来指定分区。分区之间没有明显的关系，你只能通过系统分配的方法进行很少的控制。与此相反，所有的UNIX分区都是与`root file system`相关的，这简单地叫做`/`。这个灵活性有一个问题：你可以选择把所有独立的文件系统放在整个文件系统中。你可以用`mount`命令来指定位置。例如，你可以在目录`/cdrom`中挂上一个CDROM，但如果你有三个驱动器挂在你的SCSI控制器上，你可能更喜欢把它们挂在目录`/cd0`、`/cd1`和`/cd2`上（这个编号是与UNIX的传统的以0开始的编号一致的。当然，并不阻止你选择其他的编号方法）。为了挂上一个文件系统，你必须指定被挂上的设备，它在哪儿被挂上，文件系统的类型（除非它是`ufs`）。挂载点（被挂上的目录）必须存在。要在`/cd1`上挂上你的第二个CDROM，你可以键入：

```
# mkdir /cd1                                only if it doesn't exist
# mount -t cd9660 -o ro /dev/cd1a /cd1
```

当系统启动的时候，它呼叫启动脚本`/etc/rc`，这是在其他自动挂上文件系统的目录中。所有你必须做的是提供信息：挂上什么，在哪里挂上？这是在文件`/etc/fstab`中说明的。如果你使用System V环境，你将注意到格式方面的很大不同——看看`fstab`的联机手册了解更多信息。一个典型的`/etc/fstab`看起来是这样的：

```
/dev/ad0s1a /          ufs      rw 1 1   root file system
/dev/ad0s1b none    swap     sw 0 0   swap
/dev/ad0s1e /usr    ufs      rw 2 2   /usr file system
/dev/da1e  /src    ufs      rw 2 2   additional file system
/dev/da2s1 /linux  ext2fs   rw 2 2   Linux file system
/dev/ad1s1 /C:     msdos    rw 2 2   Microsoft file system
proc      /proc   procfs   rw 0 0   pr oc pseudo-file system
/dev/cd0a /cdrom  cd9660   ro 0 0   CD-ROM
presto:/  /presto/root  nfs  rw 0 0   NFS file systems on other systems
presto:/usr  /presto/usr    nfs  rw 0 0
presto:/home /presto/home  nfs  rw 0 0
presto:/S    /S             nfs  rw 0 0
```

文件的格式是相当简单的：

- 第一列给出了设备的名称（如果它是一个真实文件系统），一个针对一些文件系统的关键字，像`proc`，或用于NFS加载的远程文件系统的名称。
- 第二列指出了加载点。
- 第三列指出了文件系统的类型。在硬盘上的本地文件系统总是`ufs`，在CDROM上的文件系统是`cd9660`。远程文件系统总是`nfs`。用`swap`来指定交换分区，用`proc`指定`proc`文件系统。
- 第四列包含了可以读或写（`rw`）的文件系统，只读的文件系统（如CDROM）`ro`，和交换分

区sw。

- 第5和第6列被dump和fsck程序使用。你通常不需要修改它们。为一个root文件系统键入1，为其他ufs文件系统键入2,为其他一些键入0。

卸载文件系统

当你挂上一个文件系统时，系统假设它是位于那个地方，它可以确保高效地把数据写回到文件系统。这是我们在第92页讨论的一样的结果。结果，如果你要停止使用一个文件系统，你必须告诉系统有关它的事情，以至它可以刷新余下的数据。你可以用`umount`命令来这样做。注意拼写——在命令名中没有n。

你必须这样做，甚至是用只读的媒介如CD-ROM：系统假设它可以从一个挂上的文件系统访问数据，如果它不能，就变得十分不好。有可能，它会锁定媒介，以至你不能从设备删除它们，直到你卸下它们。使用`umount`是很直接的：只要告诉它`umount`什么，可能是设备名或目录名。例如，要卸下上面的例子中挂上的CDROM，你只要键入下面这些命令：

```
# umount /dev/cd1a
# umount /cd1
```

在卸下一个文件系统之前，`umount`检查没有人使用它。如果有人正在使用它，它将拒绝卸下它，并提示`umount: /cd1: Device busy`。这个信息经常出现，因为你已经把你的目录改成你要删除的文件系统上的目录。例如（在提示符中显示了不可用的目录名）：

```
=== root@freebie (/dev/tty2) /cd1 16 -> umount /cd1
umount: /cd1: Device busy
=== root@freebie (/dev/tty2) /cd1 17 -> cd
=== root@freebie (/dev/tty2) ~ 18 -> umount /cd1
=== root@freebie (/dev/tty2) ~ 19 ->
```

FreeBSD设备

UNIX使用与它引用普通文件一样的方式来引用设备。与普通的文件相比，它们被叫做 *special files*。它们不是真正的文件：它们是在内核中有关设备支持的信息，术语 *device node* 更加精确。按照惯例，它们被存储在目录 `/dev` 中。一些设备没有设备节点 (*device node*)，例如以太网接口：它们与 `inconfig` 程序不一样处理。

UNIX系统区分两种类型的设备：块设备 (*block devices*) 和字符设备 (*character devices*)。FreeBSD不再有块设备。我们在第41页讨论了这个原因。

主要和次要的编号

像所有传统的UNIX系统一样，FreeBSD用主编号 (*major number*) 和次编号 (*minor number*) 来引用设备。主编号事实上用来指定驱动程序的号码，次编号是驱动程序用来区分独立设备和它应当如何对待的号码。

通常，主编号和次编号是被存储在同样的机器关键字中。最初，关键字是16位长，主编号和次编号是每个8位长，这限制了它们最大只能是255。System V.4提高了关键字的大小到32位，14位给主编号，18位给次编号，最大分别能得到16383和262143。4.4BSD也引入了32位驱动号码，但在同样的地方放置主编号，只有8位的长度。次编号占用了关键字的其余部分，是24位长。当前，Linux系统有16位设备号码，但这可能以后会改变。

这里是一个预览：

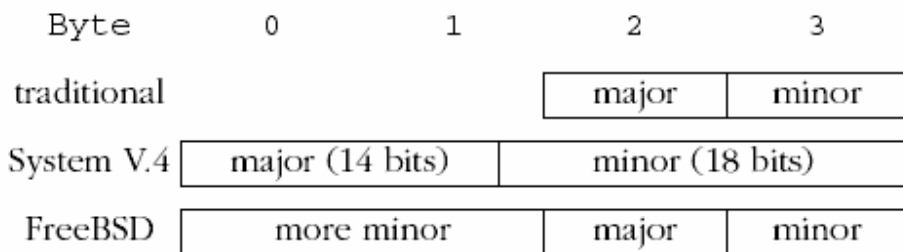


图15-3 主编号和次编号

FreeBSD为每个特殊的设备使用次编号的高位，如磁盘的控制端口。通常，你是看不到它们的，但如果你在数百万个中找到一个次编号，它不是一个错误，它是一个特性而已。

devfs

设备节点会产生一个问题：当你创建它们时，你如何知道内核中支持什么？如果你修改了内核会怎么样？

这真是一个问题。特别是，当设法访问一个特定的文件时，你可能会得到两个不同的错误：

- ENOENT, "no such file or directory", 意思是没有这个名称的设备节点。如果你向内核添加这个支持，这可能会发生，但不会创建设备节点。我们将在下一节看看添加设备节点。
- ENXIO, "device configured", 意思是设备节点存在，但内核不支持设备。在这种情况下，你必须添加对设备的支持，我们将在第20章考虑这个问题。

不知道创建哪个设备的节点是一个问题。安装脚本通过为没有内核支持的设备创建许多设备节点来补偿它。标准的安装会创建将近1000个设备节点。甚至这些还不够。例如，系统支持在一个控制器上很多SCSI设备的联合，最多到15个，但默认的配置只为磁带机和四个磁盘创建设备节点。如果你添加另外一个磁带驱动器或第五个SCSI磁盘，你将必须添加设备节点以和它们进行交谈。

FreeBSD的版本5已经用*device file system*来解决这个问题，也叫做*devfs*。*devfs*是一个伪文件系统，可以动态地为那些内核支持的设备创建设备节点，它将使你不需要手工地创建设备。

创建设备节点

那么，如果你需要创建设备节点，应当做些什么呢？比较困难的方法是使用*mknod*命令。

The Complete FreeBSD(4th)

比较容易的方法是使用脚本`/dev/MAKEDEV`。例如，默认的，FreeBSD只提供针对四个SCSI磁盘的定义。如果你要添加第五个SCSI磁盘，键入：

```
# cd /dev
# ./MAKEDEV da4          cr eate the device
# ./MAKEDEV da4s0a      cr eate the slice entries too
```

`MAKEDEV`假设你是在目录中，就像在这个例子中指出的。

`MAKEDEV`选择的名称不是最合适的。你可能很难决定如何告诉它建立你需要的设备。检查文件`/dev/MAKEDEV`开头的注释了解更多细节。例如，如果你要创建第二个磁带机驱动器，你将发现：

```
# Device "make" file. Valid arguments:
# all makes all known devices, standard number of units (or close)
# std standard devices
# jail suitable for a jail(8)
# local configuration specific devices
...
# Tapes:
# wt* QIC-interfaced (e.g. not SCSI) 3M cartridge tape
# sa* SCSI Sequential Access Devices
```

一些名称后面的星号(*)指出你应当指定创建的设备号码，或在一些情况下特定的设备号码。在这种情况下，要为第二个磁带机创建设备节点，你可以键入：

```
# cd /dev
# ./MAKEDEV sa1
```

也需要注意`./MAKEDEV`：它首先删除已存在的记录，所以你可以以较少的几个设备节点结束。如果你碰到这种事情，你可以用一个更特殊的应用程序重新编译它们，像上面这个例子一样。

FreeBSD设备预览

每一个UNIX系统当它继承设备名称和用法时都有它自己的特性。即使你习惯了UNIX，你将发现下面这样的表的用法。

表15-4 FreeBSD设备名称

设备	描述
acd0	第一个ata (IDE) CDROM驱动器。
ad0	第一个ata (IDE或相似的) 磁盘驱动器，块设备。看看第2章，了解磁盘驱动器命名的完整介绍。
bpf0	Berkeley过滤封包——看看 bpf 的描述。
cd0a	第一个SCSI CD-ROM驱动器。

The Complete FreeBSD(4th)

ch0	SCSI CD-ROM交换器(juke box)。
console	系统控制台，接收控制台信息的设备。最初，它是/dev/ttyv0，但它可能会被修改。
cuaa0	在呼出模式时的第一个串行端口。
cuaia0	在呼出模式时的第一个串行端口，初始状态。注意字母i代表 <i>initial</i> 。
cuala0	在呼出模式时的第一个串行端口，锁定状态。注意字母l代表 <i>lock</i> 。
da0	第一个SCSI磁盘驱动器，块设备。看看第2章有关磁盘命名的章节。
esa0	第一个SCSI磁带机驱动器，处于关闭模式。
fd	伪设备的文件描述符：一个包含伪设备的目录，当打开的时候，用一样的编号返回一个文件描述符的副本。例如，如果你打开/dev/fd/0，你将在你的stdin输入上得到另一个处理（文件描述符0）。
fd0	第一个软盘驱动器，被访问作为一个文件系统。
fd0a	第一个软盘驱动器，被访问作为一个文件系统。软磁盘没有与硬盘使用一样的方法来分区，名称fd0a, fd0b, fd0c, fd0d, fd0e, fd0f, fd0g和fd0h都是指同一个设备。
kmem	内核虚拟内存伪设备。
lpctl0	第一个并行打印机的控制端口。
lpt0	第一个并行打印机。
mem	物理虚拟内存伪设备。
nsa0	第一个SCSI磁带机驱动器，没有重绕模式。
nwt0	第一个QIC-36磁带机驱动器，没有重绕模式。
null	“bit bucket”。如果你再也不想看到它，向这个设备写入数据。
ptyp0	第一个主伪终端。主伪终端被命名成ptyp0到ptypv, ptyq0到ptyqv, ptyr0到ptyrv, ptyS0到ptysv, ptyP0到ptyPv, ptyQ0到ptyQv, ptyR0到ptyRv and ptyS0到ptySv。
sa0	第一个SCSI磁带机驱动器。
tty	当前控制的终端。
ttyd0	在呼入模式上的第一个传行终端。
ttyid0	在呼入模式上的第一个传行终端，初始状态。
ttyld0	在呼入模式上的第一个传行终端，锁定状态。
ttyp0	第一个副伪终端。副伪终端被命名成ttyp0到ttypv, ttyq0到ttyqv, ttyr0到ttyrv, ttys0到ttysv, ttyP0到 ttyPv, ttyQ0到ttyQv, ttyR0到ttyRv and ttyS0到ttySv。一些进程如xterm，只能用到ttyp0到ttysv。
ttyv0	第一个虚拟tty。这是系统用它来启动的窗口。最多10个虚拟tty，这些可以通过在

The Complete FreeBSD(4th)

	<i>/etc/ttys</i> 文件中添加适当的 <i>getty</i> 信息来激活。看看第26章了解更多细节。
tw0	TW-523电源行接口驱动程序。
zero	当读取时总是返回值为0的哑巴设备。

你将注意到许多与串行口相关联的不同模式。看看第26章了解更多细节。

虚拟终端

正如我们已经看到的，UNIX是一个多任务的操作系统，但一个PC一般只有一个屏幕。FreeBSD使用虚拟终端来解决这个问题。当在文本模式时，你可以结合Alt键和一个功能键来切换最多16个不同的屏幕。设备被命名为*/dev/ttyv0*到*/dev/ttyv15*，相对应按键是Alt-F1到Alt-F16。默认的，三个虚拟终端是活动的：*/dev/ttyv0*到*/dev/ttyv2*。系统的控制台就是虚拟终端*/dev/ttyv0*，当你启动机器时你已经看到的那些。要激活额外的虚拟终端，就编辑*/etc/ttys*文件。你将发现：

```
ttyv0 "/usr/libexec/getty Pc" cons25 on secure
ttyv1 "/usr/libexec/getty Pc" cons25 on secure
ttyv2 "/usr/libexec/getty Pc" cons25 on secure
ttyv3 "/usr/libexec/getty Pc" cons25 off secure
```

关键字on和off指出了终端的状态：要启用一个，把它设置成on。要启用其他的虚拟终端，用相应的终端名添加一行，范围在*/dev/ttyv4*和*/dev/ttyv15*之间。

另外，如果它们不存在，你可能必须创建设备节点，默认的，系统在*/dev*目录中包含四个虚拟终端设备。如果你使用了超过这个数的终端，你必须创建它们，用*MAKEDEV*或用*mknod*。当计算你需要多少设备时，注意如果你要运行X11，你需要为X服务器准备一个没有*getty*的终端设备。例如，如果你已经启用了*/dev/ttyv3*，*/dev/ttyv4*和*/dev/ttyv5*，你也要运行X，你将需要总共7个虚拟终端(*/dev/ttyv0*到*/dev/ttyv6*)。对于*MAKEDEV*，你要指定你需要多少个虚拟终端。

```
# cd /dev
# ./MAKEDEV vty7 make 7 vtys
```

另外，你可以用*mknod*来这样做：

```
# cd /dev
# ls -l ttyv0
crw----- 1 root wheel 12, 0 Nov 28 10:25 ttyv0
# mknod ttyv3 c 12 3
# mknod ttyv4 c 12 4
# mknod ttyv5 c 12 5
# mknod ttyv6 c 12 6
```

在这个例子中，你为了检查虚拟终端的*major device number*为*/dev/ttyv0*列出了记录（在这个例子中，那是12。它从一个版本到另一个版本可能会发生变化）。你必须为*mknod*指定这个号码。了解更多主要和次要号码的更多细节，请参看第308页。

你编辑完*/etc/ttys*文件以后，可以创建设备节点，为了启动终端，你必须告诉系统重新读取

它。用这个命令来这样做：

```
# kill -1 1
```

进程1是*inet*——看看第252页了解更多细节。

伪终端

除了虚拟终端，FreeBSD提供了其他叫做*pseudo-terminal*的终端。它们成对出现：*master device*也叫做*pty*（读作*pity*），只被使用接口的进程使用，有一个像*/dev/ptyp0*的名字。*slave device*看起来像一个终端，有一个像*/dev/ttyp0*这样的名字。任何进程都不需要特殊的接口知识就可以打开它。这些终端被用在如*xterm*, *telnet*和*rlogin*这样的网络连接上。你不需要为伪终端使用一个*getty*。

你需要为每个终端类型的连接准备一个伪终端，例如为一个*xterm*。它很容易用完它们。如果你这样做，你可以配置更多。另外，产生设备节点。例如，要生成另一组32个伪终端，你可以键入：

```
# cd /dev
# ./MAKEDEV pty1
```

你可以最多生成256个伪终端。它们被命名成*ttyp0*到*ttypv*, *ttyq0*到*ttyqv*, *ttyr0*到*ttyrv*, *ttys0*到*ttysv*, *ttyP0*到*ttyPv*, *ttyQ0*到*ttyQv*, *ttyR0*到*ttyRv*和*ttyS0*到*ttySv*。要创建每组32个终端，使用组号：第一组是*pty0*，第八组是*pty7*。注意一些进程，如*xterm*，只可以使用*ttyp0*到*ttysv*。