附录 A

编写手册页

程序用户需要说明文件，程序设计者同样也需要它，当他们最近很少用到这个软件的时候。可惜的是，很少有计算机书籍提及软件说明文件的制作，所以就算用户想为程序写一份好文件，也不晓得怎么做，甚至不知从何开始。本附录便是为了填补这样的不足。

在UNIX中，简短的程序文件多半为手册页（manual page）的形式，以nroff/troff标记（markup）写成（注1），可通过man、nroff -man或groff -man以简单的ASCII文本显示，使用ditroff -man -Txxx、groff -man -Txxx或troff -man -Txxx显示设备xxx的排版方式，或在X windows下以groff -TX -man检查。

较冗长的软件说明文件一直以来都是提供使用手册或技术报告，通常是troff标记形式，并以PostScript或PDF的形式打印。troff标记完全不是以人类看得懂的方式定义，因此，GNU Project选择另一种完全不同的方式：Texinfo文件系统（注2）。Texinfo标记认为比通用的troff包更高级，且就像troff一样，也允许以简单ASCII文本以及TEX排版系统（注3）检查。最重要的是：它支持超文本链接，让用户在浏览整个在线文档时更方便。

注1：虽然nroff是在troff之前开发完成，但从用户的角度来看，这两个系统其实是类似的：ditroff与groff模拟这两者。

注2：见Robert J. Chassell与Richard M. Stallman的《Texinfo: The GNU Documentation Format》，由Free Software Foundation于1999年出版，ISBN: 1-882114-67-1。

注3：见Donald E. Knuth的《The TEXbook》，由Addison-Wesley于1984年出版，ISBN: 0-201-13448-9。

大部分你在UNIX系统里读到的在线文档，可能都是以早期的troff（注4）或Texinfo（注5）标记形成。Texinfo系统里的makeinfo程序可以产生ASCII、HTML、XML与DocBook/XML格式的输出。Texinfo文件可直接由TEX排版，其输出为DVI（device-independent）文件，该文件格式可通过后端DVI驱动程序转成数种设备格式。

当然能用的不只有这些标记格式。Sun Microsystems自Solaris 7开始，即以SGML格式提供（几乎）所有的手册页，而Linux Documentation Project（注6）推动XML（SGML子集）标记，有助于该单位将GNU/Linux文件转换为世界各国语言的目标。

那么，UNIX的程序设计者究竟应使用哪个标记系统呢？经验告诉我们，使用高级标记较佳，即便它较冗长，但绝对值得。SGML（HTML与XML）建立在严谨的语法上，所以在它们编译为可显示的页面之前，文件的逻辑架构仍是很好验证的。有了充分详细的标记，SGML文件即能可靠地转换为其他标记系统，事实上，有些书及杂志出版商正是这么做：作者以任意文件格式交稿，出版商将其转换为SGML，然后再使用troff、TEX或其他排版系统作为后端，产生打印机可读取的页面。

不幸的是SGML软件工具集仍不够充分，且未完整标准化，因此要达到软件文件最大的可移植性，可能还是使用troff或Texinfo标记之一比较好。以手册页来说，如果可以使用man命令，则使用troff格式较佳。

最后，有人仍希望能做出自动化转换两个标记系统的产物，不过这样的目标其实很难做到。你现在能做的，便是用troff标记的限制式子集编写手册页，让它们能自动转换为HTML与Texinfo。要达到此目标，你必须安装两个包：man2html与man2texi（注7）。

pathfind的手册页

即便完整介绍标记系统文件的书很多，不过你可以从我们这里的介绍，更轻松地学习、了解troff子集。我们在这里会逐步介绍，就像在8.1节里分段介绍pathfind脚本那样，最后会再将这些片段集结成完整的手册页文件，呈现于例A-1。

在开始前，我们先介绍一下nroff/troff标记语言。nroff建置在早期文本格式化系统的经验上，例如DEC的runoff及产生ASCII打印设备的输出结果。当贝尔实验室

注4：见 *http://www.troff.org/*。

注5：见 *http://www.gnu.org/software/texinfo/*。

注6：见 *http://www.tldp.org/*。

注7：可自 *http://www.math.utah.edu/pub/man2html* 与 *http://www.math.utah.edu/pub/man2texi* 取得。

需要照相凸版排版系统时，troff这个用于产生排版页面的新程序就诞生了。troff为最早期计算机排版的尝试中成功的一个。这两个程序都接受相同格式的输入，所以当我们说troff时，通常也是指nroff。

早期UNIX系统是在极小内存的微型计算机上执行，显然并不适于处理这些格式化程序。troff命令，一如许多UNIX命令，是隐密的与简短的。大部分出现在一行的开头，形式为一个点号，接上一或两个字母或数字。其字体的选择也是有限的：只有roman、粗体、斜体，及后来的等宽字体这几种形式而已。troff的文件里，空格与空白行是有意义的：输入两个空格字符，就会产生（大约）两个输出空格。

然而，简单的命令格式令troff文件的解析更容易，且许多前端处理程序已被开发为提供简单的方程序、图形、图片与表格的规格。它们消耗troff数据流，并产生比原始数据流稍大一些的输出。

虽然完整的troff命令其实内容庞大，但通过-man选项所选定的手册页风格只有一些命令。它不需要前端处理程序，所以手册页里没有方程或图片，表格也很少。

手册页文件的版面配置相当简单，六七行标准的顶层标题段落，穿插着一些文本的格式化段落，及缩进的、定标签的区块。就像你每次使用man命令所看到的那样。

手册页的检查方式，长久以来累积了相当多种类，在显示的文体上也有很大的不同，当标记是视觉的而非逻辑的时候较容易被预期。我们在此选择的字体，只是建议性，而非强制一定要使用。

现在是开始编写pathfind手册页的时候了，这是一个相当简单的程序，因此它的标记不致太难处理。

我们从注释性语句开始，因为每个计算机语言都应该要有。troff的注释从反斜线引用（backslash-quote）开始，直至结尾，但不包含end-of-line。当它们紧接在初始的点号之后，它们的行终结符也会从输出中消失：

```
.\" ============================================================
```

由于troff输入不能被缩进处理，所以它看起来非常密集。我们发现，在标头段落之前的等号注释行会让它们比较好辨识，且我们时常使用相当短的输入行。

每个手册页文件都以*Text Header*命令（.TH）开始，其至多可带有4个参数：大写命令名称、手册段落编号（数字的1为用户命令），以及可选用的再版日期与版本编号。这些参数用以构建执行中的页面标头与格式化后输出文件的页尾：

```
.TH PATHFIND 1 "" "1.00"
```

Section Heading（部分标题）命令（.SH）则只带有一个参数，如含有空格，请引用它。且请遵循手册页惯例，以大写字母表示：

```
.\" ============================================================
.SH NAME
```

NAME段落的主体，提供的是apropos（等同于man -k）命令所需的要件，它应该只有一行，结尾不带有任何标点符号。形式为command - description：

```
pathfind \(em find files in a directory path
```

标记\(em是手册页里可见的少数几个troff命令之一：它表示一个em -（破折号），也就是大约是字母m宽度的水平行。前置一个空格并且接着em -（破折号）。较旧的手册页用的是\-（负号），或只是-，但em -（破折号）为英语印刷样式的惯用法。

第二个段落为在命令行引用程序时，提供的简短概要说明。一开始仍为标头：

```
.\" ============================================================
.SH SYNOPSIS
```

接下来有时会是漫长的标记，最常出现的就是字体信息：

```
.B pathfind
[
.B \-\^\-all
]
[
.B \-\^\-?
]
[
.B \-\^\-help
]
[
.B \-\^\-version
]
```

选项-（连字号）以\-标记，以取得负号的排版方式，看起来会比稍短的原始连字号好。我们使用\^命令，防止在troff的输出中将连字号一起执行。nroff的输出下，空格字符会消失。程序名称与选项被设置为粗体字。字体转换的命令，像.B，可使用到6个参数（如它们包含空格，请引用它），然后每一个都紧邻着排版。当出现多个参数时，意即所有字间需要的空格都应该明确地提供。在此，方括号为默认的roman字体；在手册页中，它们界定可选用的值。虽然我们应该将关闭与开启的方括号置于同一行，但我们不这么做，因为让每个选项可以在三个连续行上完成可便于编辑。字体配对的命令虽可立即接上，让它们变成单唯一行，但它们很少被用在选项列表里。

除了断行，troff会以塞满段落的模式排版，因此所有东西看起来只有一行。以经验来

说，我们发现nroff的ASCII输出会在--version选项之后断行，但因为我们是在段落模式下，所以下一行会从最左边缘接上。这部分有点讨厌，所以我们只有在套用nroff时置入条件语句处理，但在troff里就不需要这么使用。这里是以临时缩进（temporary indentation）命令（.ti）加上参数+9n处理，即缩进9个空格符，大约是命令名称的宽度加上等宽字体的结尾空格符：

```
.if n .ti +9n
```

命令行很短，放在单一排版行上绰绰有余，所以对troff无须再作这类处理。这里是它大致的样式，但我们隐藏了注释，待程序加入了更多的选项，我们再添上：

```
.\" .if t .ti +\w'\fBpathfind\fP\ 'u
```

缩进总数的计算很复杂，因为它与字体成比例，且我们无法得知命令名称的宽度。\w'...'u的命令是在计算单引号里元素的宽度。因为文本被设置为粗体字，所以我们使用内部的字体包装：\fB...\fP，即转换为粗体字后，再转换回原先的字体。类似的字体转换命令还有roman（\fR）、斜体字（\fI），与等宽（\fC）字体。C表示的是Courier，这是广为流传的等宽字体。

接下来的命令行处理为：

```
envvar [ files-or-patterns ]
```

第三段落描述程序的选项。这部分置于所有进一步说明之前，是因为大部分在手册页里，它是最常读取的段落：

```
.\" ============================================================
.SH OPTIONS
```

部分选项会接上简短的备注说明，所以接下来处理这个：

```
.B pathfind
options can be prefixed with either one or two hyphens, and
can be abbreviated to any unique prefix.  Thus,
.BR \-v ,
.BR \-ver ,
and
.B \-\^\-version
are equivalent.
```

这个段落展现了一个新特色：成对字体命令（.BR），这里设置其参数是粗体与roman字体的文本之间没有任何空格。类似的命令还有：.IR与.RI斜体-roman配对，.IB与.BI粗体－斜体配对，当然还有已经介绍过的.RB。不过等宽字体并没有类似的用法，因为它是后来才加入的（原始的贝尔实验室排版程序并没有这样的字体），你必须改用\fC...\fP。

现在该是切开段落的时候了：

```
.PP
```

在nroff的输出中，一空行与一段落切分是一样的意思，但troff则使用少数的垂直空格作为段落切分。在段落之间使用.PP会是比较好的形式，一般来说，手册页输入文件绝不应含有任何空行。

接下来的段落如下：

```
To avoid confusion with options, if a filename begins with a
hyphen, it must be disguised by a leading absolute or
relative directory path, e.g.,
.I /tmp/-foo
or
.IR ./-foo .
```

现在，我们可以开始进行选项描述了。它们的标记应该算是用在手册页里最复杂的部分，不过要上手也很快。本质上，我们要的是有标签的（labeled）缩进段落，辅以段落第一行最左边的标签设置。近期许多标记系统均以项目列表构建此部分：起始－选项－列表、起始－选项、结束－选项、起始－选项、结束－选项等等，然后以结束－选项－列表作终结。不过，手册页标记不全然这么做，它只是起始于项目，但终结于下一个段落切分（.PP）或部分标头（.SH）。

起始项目的命令（.TP）可选择性地设置宽度参数，设置描述性段落从左边缘开始的缩进宽度。如参数省略，则使用默认的缩进。如标签长度大于缩进，则新的行立即自标签之后开始。段落缩进仍会影响接下来的.TP命令，所以只有选项列表里的第一个需要它。如同使用SYNOPSIS部分里封装的命令行的缩进，我们使用动态的缩进，根据最长选项名称的长度而定。由于我们有好几个选项要说明，所以这里以具有一连串破折号的注释行将之区分：

```
.\" ---------------------------------------------------------
.TP \w'\fB\-\^\-version\fP'u+3n
```

接在.TP命令之后的行会提供项目标签：

```
.B \-all
```

标签之后接的是选项描述：

```
Search all directories for each specified file, instead of
reporting just the first instance of each found in the
search path.
```

如果这个描述需要切分段落，则使用缩进段落（Indented Paragraph）命令（.IP）取代

原来的段落切分命令（.PP），这么做不会终结此列表。不过这份手册页很短，我们用不到 .IP。

接下来的选项描述就不再需要用到新的标记了，下面为完整的选项部分：

```
.\" ---------------------------------------------------------
.TP
.B \-?
Same as
.BR \-help .
.\" ---------------------------------------------------------
.TP
.B \-help
Display a brief help message on
.IR stdout ,
giving a usage description, and then terminate immediately
with a success return code.
.\" ---------------------------------------------------------
.TP
.B \-version
Display the program version number and release date on
.IR stdout ,
and then terminate immediately with a success return code.
```

手册页第4段为程序描述。这部分的长度由你决定：Shell能执行到数十页。然而，我们期待它是简短的，因为手册页时常会被查阅。pathfind相当简单，只要三段就能描述完成。前两段的标记是我们已经知道的：

```
.\" ===========================================================
.SH DESCRIPTION
.B pathfind
searches a colon-separated directory search path defined by
the value of the environment variable, \fIenvvar\fP, for
specified files or file patterns, reporting their full path on
.IR stdout ,
or complaining \fIfilename: not found\fP on
.I stderr
if a file cannot be found anywhere in the search path.
.PP
.BR pathfind 's
exit status is 0 on success, and otherwise is the number of
files that could not be found, possibly capped at the
exit code limit of 125.
.PP
```

最后一小部分是必须了解的手册页标记，显示在最后一段。在此，我们要以计算机输入与输出的等宽的缩进方式显示，而非一般填满段落的方式。字体的改变，是类似我们先前所提到的 \fC...\fP。当它出现于行起始时，我们会前置troff的no-op命令 \&，因为如果接下来的内文一开始就是点号时，就必须使用no-op。我们要计算机范例是缩

进的，所以将缩进范围边界以*Begin Right Shift*（.RS）与*End Right Shift*（.RE）命令界定。并且，我们还需要停止填满整个段落，所以在内文前后加上*no fill*（.nf）与*fill*（.fi）命令：

```
For example,
.RS
.nf
\&\fCpathfind PATH ls\fP
.fi
.RE
reports
.RS
.nf
\&\fC/bin/ls\fP
.fi
.RE
on most UNIX systems, and
.RS
.nf
\&\fCpathfind --all PATH gcc g++\fP
.fi
.RE
reports
.RS
.nf
\&\fC/usr/local/bin/gcc
/usr/bin/gcc
/usr/local/gnat/bin/gcc
/usr/local/bin/g++
/usr/bin/g++\fP
.fi
.RE
on some systems.
.PP
Wildcard patterns also work:
.RS
.nf
\&\fCpathfind --all PATH '??tex'\fP
.fi
.RE
reports
.RS
.nf
\&\fC/usr/local/bin/detex
/usr/local/bin/dotex
/usr/local/bin/latex
/usr/bin/latex\fP
.fi
.RE
on some systems.
```

最后部分提供其他相关命令的交叉引用信息；这些信息对读者可能相当有用，所以请彻

底执行。它的格式很简单：只是一个以字母顺序排列的单一段落，且其命令名称为粗体并辅以使用手册部分编号，各命令以逗点隔开，最后以点号结束：

```
.\" ============================================================
.SH "SEE ALSO"
.BR find (1),
.BR locate (1),
.BR slocate (1),
.BR type (1),
.BR whence (1),
.BR where (1),
.BR whereis (1).
.\" ============================================================
```

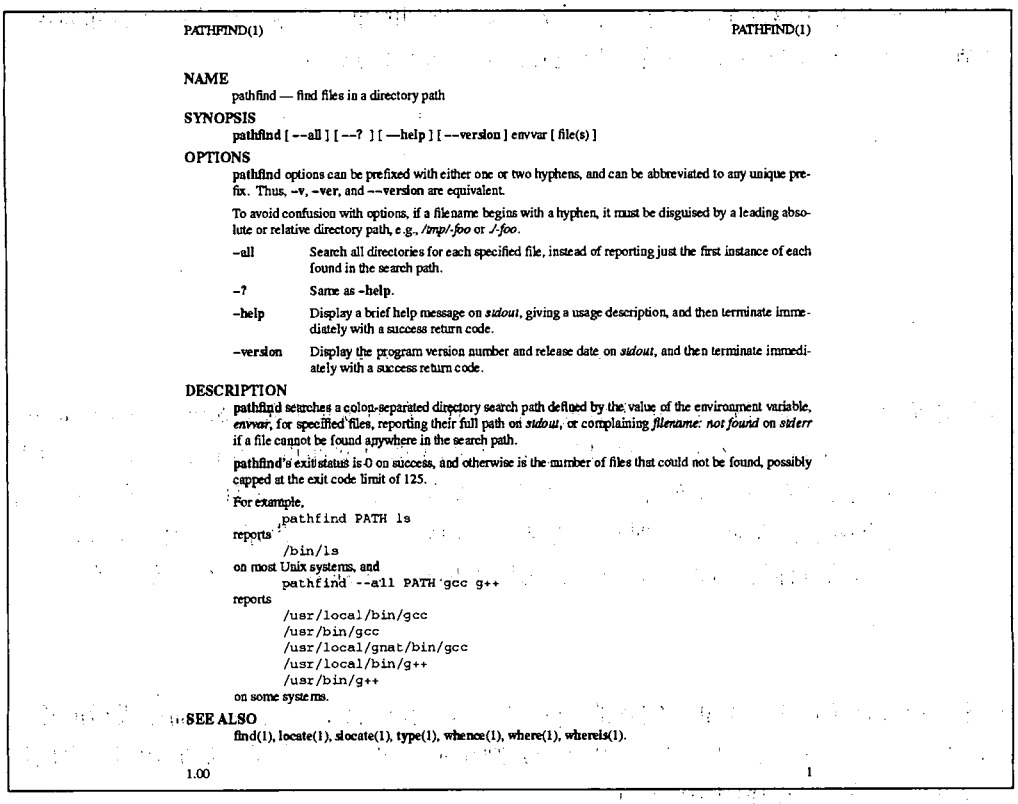
我们几乎已经介绍完所有常见的手册页标记了。唯一的重要遗漏便是*Subsection Heading*命令（.SS），不过它很少见，只出现在较冗长的手册页文件里，其运行与.SH命令类似，只不过它在排版输出中使用较小的字体。来自nroff的ASCII输出，在视觉上并无差异。另有两个行内命令，有时你可能会需要用到.\|.\|.表示省略符号（即...），与\(bu表示项目标记（即·），时常作为以下标签段落列表中的标签，像这样：

```
.TP \w'\(bu'u+2n
\(bu
```

至此已检查过手册页的分析。完整的troff输入，我们收集在例A-1，而排版后的输出（来自groff -man，默认产生PostScript）则显示于图A-1。有了我们的指南，你应该可以开始着手编写程序的手册页了。

例A-1：pathfind的troff手册页标记

```
.\" ============================================================
.TH PATHFIND 1 "" "1.00"
.\" ============================================================
.SH NAME
pathfind \(em find files in a directory path
.\" ============================================================
.SH SYNOPSIS
.B pathfind
[
.B \-\^\-all
]
[
.B \-\^\-?
]
[
.B \-\^\-help
]
[
.B \-\^\-version
]
```



PATHFIND(1) PATHFIND(1)

NAME

pathfind — find files in a directory path

SYNOPSIS

pathfind [--all] [--?] [—help] [--version] envvar [file(s)]

OPTIONS

pathfind options can be prefixed with either one or two hyphens, and can be abbreviated to any unique prefix. Thus, -v, -ver, and --version are equivalent.

To avoid confusion with options, if a filename begins with a hyphen, it must be disguised by a leading absolute or relative directory path, e.g., */tmp/-foo* or *./-foo*.

-all Search all directories for each specified file, instead of reporting just the first instance of each found in the search path.

-? Same as -help.

-help Display a brief help message on *stdout*, giving a usage description, and then terminate immediately with a success return code.

-version Display the program version number and release date on *stdout*, and then terminate immediately with a success return code.

DESCRIPTION

pathfind searches a colon-separated directory search path defined by the value of the environment variable, *envvar*, for specified files, reporting their full path on *stdout*, or complaining *filename: not found* on *stderr* if a file cannot be found anywhere in the search path.

pathfind's exit status is 0 on success, and otherwise is the number of files that could not be found, possibly capped at the exit code limit of 125.

For example,

```
pathfind PATH ls
```

reports

```
/bin/ls
```

on most Unix systems, and

```
pathfind --all PATH gcc g++
```

reports

```
/usr/local/bin/gcc
/usr/bin/gcc
/usr/local/gnat/bin/gcc
/usr/local/bin/g++
/usr/bin/g++
```

on some systems.

SEE ALSO

find(1), locate(1), slocate(1), type(1), whence(1), where(1), whereis(1).

1.00 1

图 A-1：pathfind 排版后的手册页

```
.if n .ti +9n
.\" .if t .ti +\w'\fBpathfind\fP\ 'u
envvar [ files-or-patterns ]
.\" ------------------------------------------------------
.SH OPTIONS
.B pathfind
options can be prefixed with either one or two hyphens, and
can be abbreviated to any unique prefix.  Thus,
.BR \-v ,
.BR \-ver ,
and
.B \-\^\-version
are equivalent.
.PP
To avoid confusion with options, if a filename begins with a
hyphen, it must be disguised by a leading absolute or
relative directory path, e.g.,
.I /tmp/-foo
or
.IR ./-foo .
.\" ------------------------------------------------------
```

```
.TP \w'\fB\-\^\-version\fP'u+3n
.B \-all
Search all directories for each specified file, instead of
reporting just the first instance of each found in the
search path.
.\" ------------------------------------------------------
.TP
.B \-?
Same as
.BR \-help .
.\" ------------------------------------------------------
.TP
.B \-help
Display a brief help message on
.IR stdout ,
giving a usage description, and then terminate immediately
with a success return code.
.\" ------------------------------------------------------
.TP
.B \-version
Display the program version number and release date on
.IR stdout ,
and then terminate immediately with a success return code.
.\" ======================================================
.SH DESCRIPTION
.B pathfind
searches a colon-separated directory search path defined by
the value of the environment variable, \fIenvvar\fP, for
specified files or file patterns, reporting their full path on
.IR stdout ,
or complaining \fIfilename: not found\fP on
.I stderr
if a file cannot be found anywhere in the search path.
.PP
.BR pathfind 's
exit status is 0 on success, and otherwise is the number of
files that could not be found, possibly capped at the
exit code limit of 125.
.PP
For example,
.RS
.nf
\&\fCpathfind PATH ls\fP
.fi
.RE
reports
.RS
.nf
\&\fC/bin/ls\fP
.fi
.RE
on most UNIX systems, and
.RS
.nf
```

```
\&\fCpathfind --all PATH gcc g++\fP
.fi
.RE
reports
.RS
.nf
\&\fC/usr/local/bin/gcc
/usr/bin/gcc
/usr/local/gnat/bin/gcc
/usr/local/bin/g++
/usr/bin/g++\fP
.fi
.RE
on some systems.
.PP
Wildcard patterns also work:
.RS
.nf
\&\fCpathfind --all PATH '??tex'\fP
.fi
.RE
reports
.RS
.nf
\&\fC/usr/local/bin/detex
/usr/local/bin/dotex
/usr/local/bin/latex
/usr/bin/latex\fP
.fi
.RE
on some systems.
.\" ============================================================
.SH "SEE ALSO"
.BR find (1),
.BR locate (1),
.BR slocate (1),
.BR type (1),
.BR whence (1),
.BR where (1),
.BR whereis (1).
.\" ============================================================
```

手册页语法检查

检查手册页的格式化是否正确，通常是视觉地直接看，你只要使用下面其中一个命令，打印输出即可：

```
groff -man -Tps pathfind.man | lp
troff -man -Tpost pathfind.man | /usr/lib/lp/postscript/dpost | lp
```

或者使用这样的命令，在屏幕上以 ASCII 或是排版输出：

```
nroff -man pathfind.man | col | more
groff -man -Tascii pathfind.man | more
groff -man -TX100 pathfind.man &
```

col命令会处理由nroff针对平行的与垂直的操作所产生的特殊转义符，不过groff的输出就不需要用到col了。

有些UNIX系统里会提供简单的语法检查程序：checknr，命令为：

```
checknr pathfind.man
```

它在我们的系统上不会产生警告信息。checknr的功能是找出不符合的字体，但对手册页格式的了解不多。

很多UNIX系统里都有deroff，它是一套简单的过滤程序，用以去除troff标记。你可以像这样执行拼字检查，：

```
deroff pathfind.man | spell
```

为了避免来自拼字检查程序对于troff标记错误的抱怨。找出文件里难以察觉的问题还有两个好用的工具：叠词查找(doubled-word finder，注8)与界定符平衡检查(delimiter-balance checker，注9)。

手册页格式转换

转换为HTML、Texinfo、Info、XML与DVI文件很简单：

```
man2html pathfind.man
man2texi --batch pathfind.man
makeinfo pathfind.texi
makeinfo --xml pathfind.texi
tex pathfind.texi
```

碍于长度，我们不在这里展现.html、.texi、.info与.xml文件的输出。如果你好奇可以自己做做看，再一窥其内容，了解它们的标记格式。

手册页的安装

一直以来，都是使用man命令，在环境变量MANPATH定义的查找路径下之各个子目录中——通常像这样/usr/man:/usr/local/man，寻找手册页。

注8： *http://www.math.utah.edu/pub/dw/*。

注9： *http://www.math.utah.edu/pub/chkdelim/*。

有些近期的man版本只是假设，在程序查找路径PATH中的每个目录，可以放置/../man字符串在尾端，以找出相对应的手册页目录，而不需用到MANPATH。

在各个手册页目录下，通常是寻找前置man与cat，及结尾是部分编号的成对子目录。在各子目录内，文件名也以部分编号作为结尾。因此/usr/man/man1/ls.1为ls命令文件的troff文件，而/usr/man/cat1/ls.1则保存nroff的格式化输出。man使用的是后者，当它存在时，可避免重新执行不必要的格式化。

当有部分厂商采用全然不同的手册页树状结构组织时，它们的man实例仍能认得过去存在过的实现方式。因此，大部分GNU软件的安装将可执行文件置于$prefix/bin中，并将手册页置于$prefix/man/man1，其中prefix默认为/usr/local，且在各类系统下都能运行得当。

系统管理者通常会在固定一段期间安排执行catman或makewhatis，以更新来自手册页NAME部分中含有单行描述的文件。该文件是给apropos、man -k与whatis命令所使用的，目的在于提供手册页的简单索引。如果这么做还找不到你想要的东西，就只能求助于grep，使用全文查找了。

附录 B

文件与文件系统

如果想要有效率地利用计算机，那么对文件与文件系统就要有基本的了解。本附录要呈现的是 UNIX 文件系统重点功能的概要：什么是文件？文件如何命名、包括了哪些东西？如何将它们层级式地聚集在文件系统里？它们有哪些特性？

什么是文件

简单的说，一个文件就是存在于计算机系统里的一堆数据，而且可以用单一实体的方式从计算机程序中引用。文件还提供让进程执行可以继续的数据存储机制，一般常用于重新启动计算机（注 1）。

早期计算机上，文件属于计算机系统外部的东西：多半是放在磁带、纸带（paper tape），或打孔卡（punched cards）上。谁拿着卡片就能管理里面的文件，想用它的人只要愿意把一大叠打孔卡从地上抱起来就可以。

过了一段时间，磁碟就开始广泛使用。它们的物理大小一直在缩减，从整只手臂的大小缩到只有你拇指的宽度，不过它们的容量却是一直在长大，从 20 世纪 50 年代年中期的 5MB，到 2004 年的 400 000MB。成本与访问时间已经至少下降了 3 倍。而今，能使用的磁碟种类已经相当多了。

注 1：部分系统将特殊的快速文件系统置放在中央的随机访问内存（random-access memory，RAM）里，让进程之间得以共享临时文件。以一般 RAM 技术来说，这类文件系统需搭配不断电系统，因为它们常会在系统重启后重建一个新的。然而，有些嵌入式计算机系统（embedded computer system）使用非挥发性（nonvolatile）的 RAM，可提供长期的文件系统。

光学存储设备，例如CD-ROM与DVD，已经是廉价又高容量的选择了：20世纪90年代，CD-ROM大举取代软盘（flexible magnetic disk；floppy）与商用软件发布所使用的磁带。

另外可以用的还有非挥发性的固态（solid-state）存储设备；它们最后应该会取代某些具有移动机制部分的设备，后者会因为逐渐耗损而失效。不过在编写本书的时候，成本上的考虑远大于它的替代性，它们不但容量比较小，且仅能重写几次而已。

文件如何命名

早期计算机操作系统无法命名文件：文件的处理是由它们的所有者提出，再由人工计算机操作人员一次处理一个。不久马上就有人发现，如果文件能自动地处理就更好了：文件需要人类可用来归类与管理的名称，而计算机也可使用该名称识别它们。

当我们可以指定名称给文件后，马上就会发现必须处理名称冲突的问题，因为很可能出现相同名称指定予两个或更多个不同文件的情况。现代文件系统解决这个问题的方法是将独一无二的文件名逻辑式地组合在一起，称为目录（directory）或文件夹（folder）。这部分在本附录稍后“UNIX层级式文件系统”里将有所介绍。

在文件命名时，使用的是从主机操作系统里字符集取得的字符。早期的计算上，字符集各有相当大的差异，但因为必须在相异的系统间交换数据便凸显了标准化的需求。

1963年，*American Standards Association*（注2）以冗长的*American Standard Code for Information Interchange*名称，提出7位字符集，之后即以其初始字母ASCII（发音为*ask-ee*）广为流传。7个位允许呈现$2^7 = 128$个不同的字符，已经足以处理拉丁字母的大写与小写、数字与一些特殊符号及标点符号字符，包括空格以及剩下的33个控制字符。后者未指定可打印的图形表现方式。它们之中，有些是用作将行加以标示与分页，但大部分是用于特定用途。几乎所有计算机系统都已支持ASCII。想了解ASCII字符集，请使用命令`man ascii`。

不过，这么多的世界语言，使用ASCII表示是不够的：它能贮藏的字符太少了。因为大部分的系统现在都使用8位字节，作为最小的定址存储单位，它允许$2^8 = 256$个不同字符。系统设计师也立即将该256元素集合的上半部分拿来使用，将ASCII留在下半部分。可惜的是他们未遵循国际标准，所以出现了几百种不同的各种字符指定方式，有时它们会被称为内码页（code page）。即使单一128个额外字符集的空间，对完整的欧洲语系

注2：之后重新命名为*American National Standards Institute (ANSI)*。

仍嫌不足，因此 *International Organization for Standardization* (*ISO*)便开发了一系列代码页（或称内码页）：ISO 8859-1（注 3）、ISO 8859-2、ISO 8859-3 等。

20 世纪 90 年代，共同开发的单一万国字符集 Unicode（注 4）开始运作。这最终需要每个字符大约有 21 个位，但许多操作系统下的现行实例只使用到 16 个位。UNIX 系统使用一个可变动的位宽度编码：*UTF-8*（注 5），允许已存在的 ASCII 文件成为有效的 Unicode 文件。

会讨论字符集上是因为：除了独特的 IBM 大型计算机使用 *EBCDIC*（注 6）字符集外，所有现行系统都将ASCII字符集纳入128以下的位置。因此将文件名限制在ASCII子集，我们就可以让这个名称通用于所有地方了。现在的 Internet 与 World Wide Web 便证明了文件可以在不同的系统间进行交换。

原始的UNIX文件系统设计者，决定这256个元素集合都可用于文件名，但有两个例外：一个是控制字符 NUL（此字符的所有位都为零），这是许多程序语言里，用来表示字符串结尾的字符；另一个则是斜杠 (/) 字符，这是用来保留重要用途的字符，稍后会介绍。

此选择是相当宽容，不过我们强烈建议你再加强进一步的限制，理由如下：

- 因为文件名是人们也要使用，所以它的名称必须是可视字符：看不到的控制字符不适合。
- 文件名不单单是人类要用，计算机也要用：人们可从上下文认出作为文件名的字符字符串，但计算机程序需要更精确的规则。
- 在文件名里使用 Shell 的 meta 字符（也就是大部分的标点符号）必须特殊处理，因此最好都避免。
- 初始的连字号会让文件名看起来像 UNIX 命令的选项。

注 3：可到 *http://www.iso.ch/iso/en/CatalogueListPage.CatalogueList* 查找ISO Standards 目录。

注 4：《The Unicode Standard, Version 4.0》，由 Addison-Wesley 于 2003 年出版，ISBN: 0-321-18578-1。

注 5：见 *RFC 2279: UTF-8*，《a transformation format of ISO 10646》，见 *ftp://ftp.internic.net/rfc/ rfc2279.txt*。

注 6：EBCDIC = Extended Binary-Coded Decimal Interchange Code，发音为 *eb-see-dick* 或 *eb-kih-dick*。这是一套在 1964 年首次出现在 IBM System/360 系统上的 8 位字符，包括了旧式 6 位的 IBM BCD 集合作为子集。System/360 及其后来的产物都是应用在计算机上最久的架构，许多全球企业都使用它们。IBM 也支持使用 ASCII 字符集的 GNU/Linux 实例，见 *http://www.ibm.com/linux/*。

部分非UNIX文件系统允许在文件名里使用大、小写字符，但在比较文件时却会忽略大小写的不同。UNIX原始的文件系统则不是这样，对它们来说：readme、Readme，与README是三个不同的文件名（注7）。

UNIX文件名惯用的方式是全为小写，因为它好读、好输入。某些常见的重要文件名，例如AUTHORS、BUGS、ChangeLog、COPYRIGHT、INTALL、LICENSE、Makefile、NEWS、README，与TODO则惯用大写或混用大小写。因为大写字母在ASCII字符集里位于小写字母之前，因此这些文件在进行目录列表时，会一开始就出现，而更容易被看到。然而，以现行UNIX系统而言，排序顺序视locale而定，所以将环境变量LC_ALL设为C，即可得到传统ASCII的排序方式。

为了在其他操作系统上也能使用，最好是将文件名限制在拉丁字母的字符、数字、连字号、下划线及单一个点号。

文件名可以多长？这根据文件系统而定，而有些软件本身的缓冲区有固定大小，限制了所能处理的最大文件名。早期UNIX系统有14个字符的限制。但从20世纪80年代中期，UNIX系统的设计便普遍允许使用到255个字符。POSIX定义了NAME_MAX常数为该长度，不包含终结的NUL字符，且要求最小值为14。X/Open Portability Guide要求最小值为255。你可以使用getconf（注8）命令找出你系统的限制。下面是你在大部分UNIX系统里会看到的报告：

```
$ getconf NAME_MAX .                    在当前的文件系统下，文件名可以多长？
255
```

文件位置的完整规格有另一个且更大的限制，我们会在本附录“文件系统架构”部分提及。

警告： 我们在这里，对使用空格字符于文件名中的做法提出警告。有些窗口式的桌面环境操作系统，其文件名是从滚动菜单中被选定，或输入到对话方块中，这让它们的用户会以为在文件名里使用空格字符是没问题的。其实不是!文件名不单只是在这个小小的对话框里被用到，唯一明智的做法应是在有限的字符集里选择字符，作为你的文件名。特别是UNIX Shell，它的命令是可以使用空格字符加以分隔的。

因为文件名里可能出现空白或其他特殊字符，在Shell脚本里，你应该要记得将任何可能含有文件名的Shell变量的计算总是以引号括起。

注7： Mac OS X里支持的旧式HFS式文件系统会视大小写为相同，所以将软件移植到该系统上，可能会出现意料之外的情况。Mac OS X也支持一般视大小写为不同的UNIX文件系统。

注8： 几乎所有UNIX系统里都有，除了Mac OS X与FreeBSD（5.0前的版本）外。getconf的源代码可以在glibc的发布包里找到：*ftp://ftp.gnu.org/gnu/glibc*。

UNIX 的文件里有什么

UNIX 另一个了不起的成就，就是以简单的观点来看文件：UNIX 的文件，不过是零或多个不知名的数据字节集结而成的流。

大部分的其他操作系统将文件看成两种：二进制与纯文本的数据、计数长度（counted-length）与固定长度与可变长度的记录、索引与随机与顺序的访问，等等。这马上就成了一种梦魇：简单的复制文件操作，可能就因为文件类型的不同而必须以不同的方式完成，且必须所有软件都能处理数种文件，复杂度会更高。

UNIX 的文件复制其实没什么：

```
try-to-get-a-byte
 while (have-a-byte)
 {
    put-a-byte
    try-to-get-a-byte
 }
```

这种循环的排序可被实例在许多程序语言中，它最棒的地方是在：程序无须知道数据从哪里来，它可以从文件、磁带设备、管道、网络连接、内核数据结构，或是从任何未来设计者所设计的数据来源而来。

你会说，那我需要一个特殊文件，文件尾端有一个具指针的目录指向稍早的数据，且该数据本身是加密的。在 UNIX 中的答案是：没问题！你只要让应用程序了解你这个完美的文件格式，完全不会带给文件系统或操作系统该复杂度。它们不必了解这些细节。

然而，UNIX 仍对允许的文件稍作区分。人为建立的文件，通常含有数行文本，以分行字符作为结尾，且不会出现无法打印的 ASCII 控制字符。这样的文件可以被编辑、显示于屏幕上、被打印、以电子邮件传送，还能通过网络传递到其他计算机系统上，其数据也保证是被维护的很完整。用于处理文本文件的程序，包括我们在本书讨论的诸多软件工具，其设计上是使用大的但大小固定的缓冲区来保存文本行。如果给它们过长的行，或具有无法打印的字符的输入文件，则它们可能会出现无法预知的行为。处理文本文件时，建议你将行的长度限制在读取时较舒适的大小，例如 50 到 70 个字符。

文本文件以 ASCII linefeed（LF）字符，在 ASCII 表里为十进制值 10，表示行的界线。此字符指的是换行字符。许多程序语言在字符字符串里，以 `\n` 表示此字符。这种表示方式，比其他系统的一组 carriage-return/linefeed 字符的表示方式简单多了。在 C 与 C++ 程序语言里的广泛用法以及后来开发的一些语言，都以单一换行字符作为文本文件里每个行的终结；这是由于它们有很多都是源自于 UNIX。

在共享文件系统的混用操作系统环境下，常会需要为使用不同行结尾符的文本文件作转换。*dosmacux*包（注9）提供了一组方便的工具来做这件事，同时保留了文件的时间戳。

UNIX里所有的其他文件可被认为是二进制文件：每一个包含在其中的字节，都有256种可能的值。因此，文本文件可以算是二进制文件的子集。

不同于某些操作系统的是，没有字符会被抢夺以表示end-of-file：UNIX文件系统单纯地在文件中保留字节数的计数。

尝试读取超越文件字节计数时，则返回一个end-of-file的暗示，所以它不可能看到任何磁盘块之前的内容。

有些操作系统禁用空文件，但UNIX不这么做。有时，它表示的只是一个文件的存在，重点不在它的内容。例如时间戳、文件锁定，以及THIS-PROGRAM-IS-OBSOLETE这样的警告，都是有用的空文件范例。

UNIX将文件视为字节流的想法，鼓励操作系统的设计者，实现出看起来像文件但非传统式文件想法的数据。许多UNIX版本，实现一个进程信息虚拟文件系统（pseudofilesystem）：只要输入man proc就可以知道你系统提供的有哪些。我们在13.7节里已做过详细的讨论。在/proc树状结构下的文件，并未真实存在于大型存储设备下，它只是提供察看进程表格及执行中进程的内存空间，或了解操作系统内部信息（例如处理器、网络、内存与磁盘系统）的详细数据的方式。

以我们写本书时所使用的系统为例，我们可以找到与存储设备相关的细节，类似这样（命令参数上，斜杠表示的意义将在下节讨论）：

```
$ cat /proc/scsi/scsi                          显示磁盘设备信息
Attached devices:
Host: scsi0 Channel: 00 Id: 00 Lun: 00
  Vendor: IBM      Model: DMVS18V           Rev: 0077
  Type:   Direct-Access                     ANSI SCSI revision: 03
Host: scsi1 Channel: 00 Id: 01 Lun: 00
  Vendor: TOSHIBA  Model: CD-ROM XM-6401TA Rev: 1009
  Type:   CD-ROM                            ANSI SCSI revision: 02
```

UNIX层级式文件系统

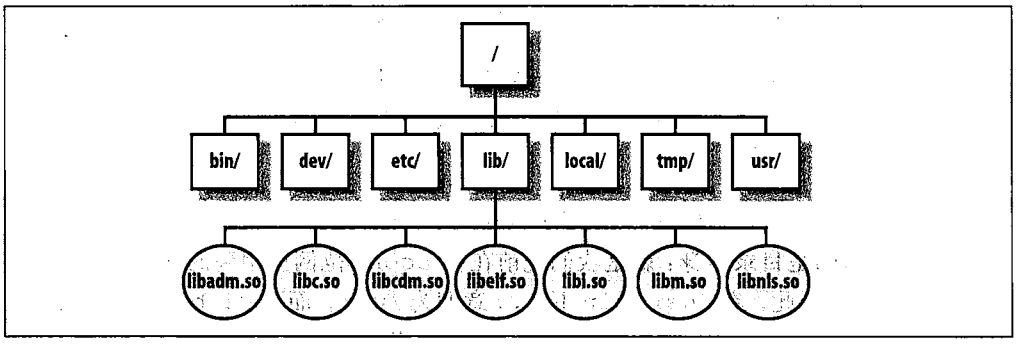
大量的文件就可能有文件名冲突的风险，如果要所有名称都独一无二，管理上也相当困难。UNIX处理的方式，便是将文件组织在目录（directory）下：每个目录形成它自己

注9： *http://www.math.utah.edu/pub/dosmacux/.*

的名称空间，独立于所有其他的目录。目录也可提供默认属性给文件，这个主题我们将在稍后的“文件所有权与权限”部分做简短介绍。

文件系统架构

目录可以嵌套配置为任意深度，使得UNIX文件系统形成树状结构。UNIX在此不使用文件夹（folder）是因为纸本文件的文件夹无法嵌套配置。文件系统目录结构的本源为根目录（root directory），它有一个特殊而简单的名称：/（ASCII的斜杠）。/myfile指的就是根目录下，叫作myfile的文件名称。斜杠还有另一个目的：用来界定名称，以记录目录的嵌套架构。图B-1呈现的是文件系统顶层架构的一小部分。



图B-1：文件系统目录结构

UNIX目录下可包括任意数目的文件。不过，大部分现行UNIX文件系统的设计与文件系统程序界面，都假定目录是被连续地查找。因此在大型目录下寻找文件的时间，便与目录里的文件数成比例。当文件超过百个，最好以子目录重新组织。

在嵌套式目录的完整列表下，要到达一文件，是以路径名称（pathname）或称为路径的方式引用。它有时会包含文件名本身，有时则不会，视当时的情况而定。文件名的完整路径，包含名称本身，能有多长？一直以来，UNIX的文件都未提供答案，但POSIX定义PATH_MAX常数来限制其长度，包含终结的NUL字符，要求最大值为256，但X/Open Portability Guide则要求到1024。你可以使用getconf命令查询你系统里的限制，以我们的系统为例：

```
$ getconf PATH_MAX .                    在当前文件系统下，路径名称的最大长度为何?
1023
```

其他我们测试过的UNIX系统，也有报告1024或4095的。

C程序语言的ISO标准称此值为FILENAME_MAX，且它必须定义在标准标头文件stdio.h

里。我们检查过许多UNIX版本，还发现255、1024，与4095的值。Hewlett-Packard HP-UX的10.20与11.23的值只有14，但它们的getconf报告则为1023与1024。

因为UNIX系统支持多个文件系统，文件名长度也为文件系统的特性之一，与操作系统无关，所以编译期常数所定义的这些限制是没有意义的。高级语言的程序员多半被建议使用pathconf()或fpathconf()函数库调用，以取得这些限制值：它们需要传递一个路径名称，或是一个打开的文件描述代码，使得特定的文件系统可以被识别。也就是为什么我们在先前的例子里，传递当前的目录（点号）给getconf。

UNIX目录本身就是文件，只不过它拥有特殊属性且限制性访问。所有UNIX系统都包括顶层目录bin，保存（时常是二进制）可执行程序，包括很多在本书中使用过的那些。这个目录的完整路径名称为/bin，它很少包含子目录。

另一个普遍性顶层目录为usr，不过它一定含有其他目录，/usr/bin就是其中一个。它和/bin是不同的，本附录稍后的“文件系统实现概况”会说明，如何让两个bin目录看起来一样（注10）。

所有的UNIX目录，就算是空的，也至少包括两个特殊目录：.（点号）与..（点号点号）。第一个指的是目录本身：就是我们先前在getconf范例里用到的那个；第二个指的则是父目录。因此，在/usr/bin下，..意即为/usr，而../lib/libc.a意思就是/usr/lib/libc.a——这是C语言执行期程序库存放的惯例位置。

根目录的父目录就是自己，所以/、/..、/../..、/../../..都是一样的。

路径结尾如果以斜杠结束，则它是一个目录。如果最后字符非斜杠，那么最后一个组成部分是目录还是其他类型的文件，则只能咨询文件系统而得知。

POSIX要求路径里连续的斜杠被视为单一斜杠。这要求在我们参考到最早期的UNIX文件里并未明白指定，但自20世纪70年代中期起，Version 6源代码一开始，即完成此斜杠减少（注11）。因此：/tmp/x、/tmp//x，与//tmp//x都指同一个文件。

注10：DEC/Compaq/Hewlett-Packard OSF/1 (Tru64)、IBM AIX、SGI IRIX，与Sun Solaris都做得到。Apple Mac OS X、BSD系统、GNU/Linux，与Hewlett-Packard HP-UX则不能做到。

注11：见John Lion的书：《Lions' Commentary on UNIX 6th Edition, with Source Code》，1996年由Peer-to-Peer Communications出版，ISBN 1-57398-013-7。此修正出现在kernel行7535（sheet 75），注释说明于p.19-2的“Multiple slashes are acceptable”。如果程序码以if取代while，则此减少不会发生。

在这本书里，有很多的注脚提供World Wide Web的来源位置（URL），其语法是以UNIX的路径名称所形成。URL前置通信协议的名称与主机名称，例如：proto://host，指的即为UNIX风格的路径名称，置于主机的网页目录树下。网页服务器需要这些信息，找到它们在文件系统里的适当位置。URL自20世纪90年代晚期开始广泛使用，使得UNIX路径名称为人们所熟悉。

层级式文件系统

如果斜杠为根目录，则每个文件系统里只会有一个，那么UNIX要如何支持多个文件系统，但又可以避免根目录名称冲突的情况呢？答案很简单：UNIX允许将某个文件系统，逻辑性地置于另一个文件系统内一个已存在的任意目录之上。该操作称为加载（mounting），相关命令为mount与umount：分别为加载与卸载文件系统。

当另一个操作系统加载在一个目录之上时，该目录先前的内容都无法看见也无法访问，只有在卸载以后它们才会再出现。

文件系统加载会让人觉得单一文件系统树会无限长大的幻觉，只需通过简单地加入更多或更大的存储设备即可。正规的文件名称惯例/a/b/c/d/...即指出对用户与软件而言，无须关心其设备为何，这点不同于其他操作系统：后者会将设备名称放置在路径名称之中。

完成加载命令需要充分的信息，因此系统管理员将这些细节存储在一个特殊文件里，通常是/etc/fstab或/etc/vfstab，视UNIX的版本而定。该文件一如大部分的UNIX组态文件：都为一般文本文件，其格式可参考手册页*fstab*(4或5)或*vfstab*(4)。

当共享的磁盘是唯一可用的文件系统媒体时，加载与卸载便需要特殊权限，通常只有系统管理员可以做这件事。不过，对一些个人拥有的媒体，例如软盘、光盘或DVD，桌上计算机的用户需要能够自己做这件事。许多UNIX系统进行了功能的扩充，所以有某些设备也允许非特权用户进行加载与卸载。这里是自GNU/Linux系统下使用的例子：

```
$ grep owner /etc/fstab | sort              哪些设备允许用户加载 ?
/dev/cdrom    /mnt/cdrom      iso9660 noauto,owner,kudzu,ro 0 0
/dev/fd0      /mnt/floppy     auto    noauto,owner,kudzu 0 0
/dev/sdb4     /mnt/zip100.0   auto    noauto,owner,kudzu 0 0
```

这里设置让用户可以使用CD-ROM、软盘，与Iomega Zip，使用方式如下：

```
mount /mnt/cdrom                            使光驱呈可用状态
cd /mnt/cdrom                               改变到它的顶层目录
ls                                          列出其文件
...
cd                                          改变根目录
umount /mnt/cdrom                           释放光驱
```

mount命令不使用参数与特殊权限时：只会简单地报告所有当前加载的文件系统。下列为独立的网页服务器范例：

```
$ mount | sort                                    显示已加载的文件系统列表，并排序它
/dev/sda2 on /boot type ext3 (rw)
/dev/sda3 on /export type ext3 (rw)
/dev/sda5 on / type ext3 (rw)
/dev/sda6 on /ww type ext3 (rw)
/dev/sda8 on /tmp type ext3 (rw)
/dev/sda9 on /var type ext3 (rw)
none on /dev/pts type devpts (rw,gid=5,mode=620)
none on /dev/shm type tmpfs (rw)
none on /nue/proc type proc (rw)
none on /proc/sys/fs/binfmt_misc type binfmt_misc (rw)
none on /proc type proc (rw)
```

这里显示，根文件系统加载在磁盘设备/dev/sda5下。其他文件系统则分别加载为/boot、/export等等。

系统管理员可使用来下命令卸载/ww树：

```
# umount /ww                                      在此，# 为根提示符号
```

如果/ww目录下有任何文件正在使用，则此命令的执行结果会失败。你可以使用lsof (list-open-files) 命令（注12），以追踪正被防止卸载的进程。

文件系统实现概况

文件系统实现的细节很有趣，但也太复杂，且超出这本书的范畴。我们建议你参考更好的书，例如《The Design and Implementation of the 4.4BSD Operating System》（注13）与《UNIX Internals: The New Frontiers》（注14），进一步了解。

从较高层的观点来看文件系统实现其实是相当有帮助的，因为这么做可以从用户的角度去看UNIX的文件系统。文件系统建立时，一个管理员指定的固定大小表格（注15）也随之建立，以保存文件系统中与文件相关的信息。每个文件都会与此表格的一个实现产生相关，每个实现都为一个文件系统数据结构，被称为*inode*（*index node*的缩写，发

注12：*ftp://vic.cc.purdue.edu/pub/tools/UNIX/lsof/*。其他UNIX版本下的替代命令可使用fstat与fuser。

注13：作者Marshall Kirk McKusick, Keith Bostic, Michael J. Karels, 与John S. Quarterman, 于1996年由Addison-Wesley出版，ISBN 0-201-54979-4。

注14：作者Uresh Vahalia，于1996年由Prentic-Hall出版，ISBN 0-13-101908-2。

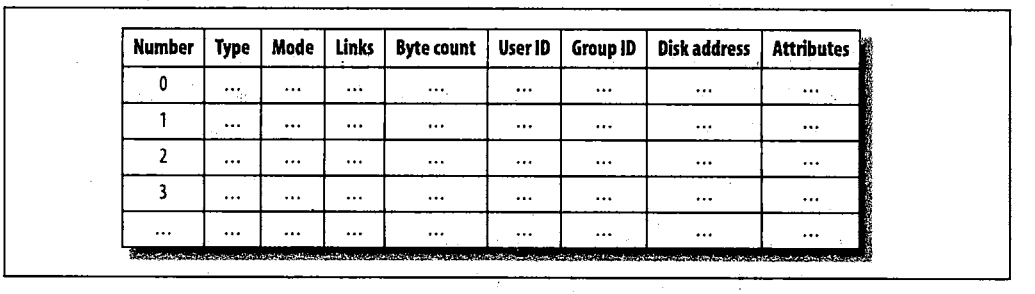
注15：部分高级文件系统设计允许根据需要加大表格。

音为*eye node*)。*indoe*的内容视特定的文件系统设计而定，因此单一系统下可能含有数种不同的形式。程序员都被隔绝在stat()与fstat()系统调用的差异之外（见*stat*(2)手册页)。参考man inode可以了解你系统上实际结构的相关信息。

由于inode结构与存储设备的其他低层细节都与系统息息相关，因此通常不太可能将某厂商的UNIX文件系统加载在另一个厂商的UNIX文件系统下。不过我们可以通过软件*Network File System (NFS)*解决这个问题，它可以跨网络共享各个不同厂商所提供的UNIX文件系统。

由于inode表格为固定大小，因此有可能出现文件系统已满，但存储设备仍有大量可用空间的情况：还有空间可用来置放文件的数据，而没有空间放它的*metadata*（数据的数据)。

如图B-2所示，inode条目包括了系统辨认文件时所需的所有数据，只有一件事除外：它的文件名。这似乎很令人惊讶，事实上，是有许多使用类似文件系统设计的操作系统将文件名包括在类似于inode的条目中。



Number	Type	Mode	Links	Byte count	User ID	Group ID	Disk address	Attributes
0	...	...	...	...	...	...	...	...
1	...	...	...	...	...	...	...	...
2	...	...	...	...	...	...	...	...
3	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

图B-2：Inode表格内容

在UNIX下，文件名伴随其inode编号存储在目录里，如图B-3所示。早期20世纪70年代小型计算机里的UNIX系统，仅在目录下，为每个文件配置16个字节：2个字节给inode编号（文件编号限制为2^{16} = 65 536)，剩下的14个字节则给文件名使用，只比一些其他系统的8+3限制好一点。

现代UNIX文件系统允许较长文件名，不过传统上还是有最大长度的限制，请参考本附录先前“文件系统架构”提供的getconf范例。

对目录的所有者，及早期一些需打开与读取目录以寻找文件的UNIX软件而言，目录只能读取，不能写入。更复杂的目录设计在20世纪80年代问世，opendir()、readdir()与closedir()程序库调用的建立，让程序员看不到它们的架构，这些调用也成为现在POSIX的一部分（见*opendir*(3)手册页)。为加强程序库的访问，部分现行的UNIX实现，禁止在目录文件上进行读取运算。

i-Node number	Filename
2155329	.
737046	..
1294503	ch04.xml
2241988	README
3974649	Makefile
720277	ch04.ps
2945369	CVS
523023	CH-AA-SHELL-EXTENSIONS.txt
351882	ch04.xml.~1~
...etc...	...etc...

图 B-3：目录表格内容

注意：为什么UNIX要将文件名与剩下的文件metadata加以分隔呢？理由至少有两个：

- 通常用户列出目录内容的目的，只是为了提醒自己：文件是不是就在这个目录下。如果文件名存在inode里，当你在目录下寻找各个文件名时，可能得访问磁盘一到多次。将名称存储在目录文件里，再多的名称都只需自单一磁盘块里取出即可。
- 如果文件名与inode各自独立，则同一个物理文件，就能拥有数个文件名，只需通过不同的目录条目引用到相同inode即可。这些引用甚至不需要在同一个目录里！这是文件别名的概念，在UNIX下称之为连接（link），是一个相当方便且广为使用的功能。在6种不同的UNIX版本下，我们就发现 `/usr` 下有10%~30%的文件为连接。

UNIX文件系统设计里，还有一个很有用的结果就是重新命名文件或目录，或是在同一个UNIX实现文件系统内移动它，速度都很快：只有名称需要改变或移动，而不会动到内容。在文件系统之间移动文件，则需要对文件所有块进行读取与写入的操作。

如果文件有数个名称，那么哪一个才能删除文件？应该在删除后，所有名称立即消失，还是只有某一个被删除？这部分是由文件系统的设计师决定支持别名还是连接；UNIX选择后者。UNIX的inode条目包括了连接到文件内容的计数。文件删除将引发连接计数的减少，但只有计数为零时，文件块最终才会重新指派给可用空间的列表。

因为目录条目包括的只是inode数字，所以它只可以引用同一个物理文件系统内的文件。我们已知道UNIX文件系统通常会包含数个加载点，我们怎么才能在这个文件系统里做一个连到另一个文件系统里的连接？解决方式是使用另一种连接：软连接（soft link），或称为符号连接（symbolic link），有时直接称为 *symlink*，这是为了与第一种硬连接

(hard link) 加以区别。符号连接表示的是“该目录条目指向另一个目录条目”(注 16),而非 inode 条目。指向的条目 (pointed-to entry) 为一般 UNIX 路径名称,因此可以指向文件系统内任何位置,就算是跨加载点也可以。

符号连接也表示可能在文件系统里产生无穷循环的可能性,为防止这样的情况发生,如需制作一连串的连接,请不要超过 8 个 (传统上)。以下为两个元素的循环:

```
$ ls -l                                          显示连接循环
total 0
lrwxrwxrwx    1 jones    devel                   3 2002-09-26 08:44 one -> two
lrwxrwxrwx    1 jones    devel                   3 2002-09-26 08:44 two -> one
$ file one                                       one 是什么文件?
one: broken symbolic link to two
$ file two                                       two 是什么文件?
two: broken symbolic link to one
$ cat one                                        试着显示 one 文件
cat: one: Too many levels of symbolic links
```

基于技术上的理由 (其中一个可能会造成循环),目录通常不能有硬连接,但可以有符号连接。此规则的例外就是 . 与 .. 目录,它们在目录被建立时即自动地产生。

设备作为 UNIX 文件

UNIX 另一个先进的做法,便是将文件的概念延展到系统上的设备。所有的 UNIX 系统都拥有名为 /dev 的顶层目录,在该目录下,则是一些难懂的文件名,例如 /dev/audio、/dev/sda1 与 /dev/tty03。这些设备文件由特殊软件模块控制,也就是设备驱动程序 (device driver),它会知道如何与特定的外部设备进行沟通。虽然设备名称会因系统的不同而有很大的差异,但它们的功能都是提供一个类似文件的“打开 - 处理 - 关闭”访问模式。

注意: 将设备整合到层级式文件系统里,是 UNIX 最棒的点子 (The integration of devices into the hierarchical file system was the best idea in UNIX.)。

—— Rob Pike et al.,《The Use of Name Spaces in Plan 9》, 1992.

/dev 树里的实体以特殊工具 mknod 所建立,该工具通常隐藏在 MAKEDEV 这个 Shell 脚本里,且需要系统管理员权限才能执行:见 *mknod*(1) 与 *MAKEDEV*(8) 的手册页。

大部分 UNIX 的用户很少需要引用 /dev 树下的成员,不过有两个例外:/dev/null 与 /dev/tty,这些我们在 2.5.5.2 节里已做过介绍。

注 16: 在 inode 中的文件类型会记录文件是符号连接,且在大部分的文件系统设计中,它指到的文件名称会被存储在符号连接的数据块中。

20世纪90年代，一些UNIX版本引进随机伪设备（random pseudodevice），/dev/urandom，作为永远非空的随机字节流。许多密码与安全性软件，需要这样的数据来源。我们在第10章已经展示过，使用/dev/urandom构建一个难猜的临时文件名。

UNIX文件到底可以多大

UNIX的文件大小通常受限于两个硬性条件：在inode条目里所分配到的位数，用来保存文件大小（以字节计），以及文件系统本身的大小。除此之外，有些UNIX内核还提供管理员设置文件大小的限制。大部分UNIX文件系统所使用的数据结构会在一个文件中记录数据块列表，加诸的限制是大约168万个块，其中一个块的大小基本上是1 024到65 536个字节，可在文件系统建立期被设置且固定住。最后，而文件系统备份设备的容量，也可能会加上更进一步与站台相关的限制。

没有名称的文件

UNIX操作系统另一个特点，就是打开供输入或输出的文件名称，不会被保留在内核的数据结构中。因此，在命令行上针对标准输入、标准输出或是标准错误输出而被重定向的文件名，都不为被引用的进程所知。想想看：我们的文件系统里已经有几百万个文件了，这三个没有名称也没有不好！不过为了弥补这个缺陷，近期有些UNIX系统提供了这样的名称/dev/stdin、/dev/stdout与/dev/stderr，或是比较难记的/dev/fd/0、/dev/fd/1与/dev/fd/2。GNU/Linux与SunSolaris还支持/proc/PID/fd/0。下面我们可以来看看你的系统是否支持它们；你要不就是执行成功，如下所示：

```
$ echo Hello, world > /dev/stdout
Hello, world
```

要不就是失败，如下所示：

```
$ echo Hello, world > /dev/stdout
/dev/stdout: Permission denied.
```

很多UNIX程序发现它们在重定向文件时需要名称，所以一般惯例是使用连字号作为文件名，使用连字号不表示文件名为连字号，而是指标准输入或标准输出，视上下文而定。我们在这里强调这是习惯用法，因为并非所有UNIX软件都如此应用。如果你就是不喜欢这样的文件，你也可以前置目录名称伪装它，例如./--data。部分程序遵循惯例，详见2.5.1节使用双连字号选项--，表示命令行自此之后是一个文件，而非一个选项，不过这种方式依然并非统一用法。

大部分现行UNIX文件系统都使用32位整数，以保存文件大小，且由于文件定位系统调用，可以在文件中前后移动，因此该整数必须带有正负号。最大可能的文件大小为2^{31} - 1个字节，约为2GB（注17）。到了20世纪90年代初期，许多磁盘设计都小于此数字，但在2000年时磁盘却能容纳100GB甚至更大的空间，甚至还能将数个磁盘结合成单一逻辑性磁盘，所以现在才能有更大型的文件系统。

UNIX厂商已渐渐升级至可处理64位的文件系统上，即能支持至8亿GB。但你想想，如果写了一个这么大的文件，以当前合理的执行速度10MB/s来看，这个文件要执行27800年！这个移植绝对会产生相当大的影响，因为所有现行使用“随机访问文件定位系统调用”的软件都必须被更新。为避免这种大规模的升级，很多厂商仍允许在较新系统上使用旧式的32位大小，只要不超过2GB限制即可正常运作。

UNIX文件系统建立时，基于效能上的理由会保留一小部分空间，通常是10%，给由root执行的进程使用。文件系统本身所需的inode表格空间，通常是放在特殊的初级块，只供磁盘控制器硬件可以访问。因此，磁盘有效空间通常只有磁盘厂商估计的80%。

有些系统里会提供降低这些保留空间的命令：在大型磁盘上，我们会建议你使用它。在BSD及商用UNIX系统上，你可以参考*tunefs*(8)手册页，GNU/Linux的系统则可参考*tune2fs*(8)。

内置Shell命令ulimit可控制系统资源的限制。-a选项将显示所有资源的值。在我们的系统上，结果如下：

```
$ ulimit -a                                    显示当前用户的进程限制
...
file size (blocks)          unlimited
...
```

你的系统可能会由于本地管理的政策不同而有不一样的结果。

在某些UNIX站台，磁盘限制是启动的（详见*quota*(1)的手册页），它可以进一步限制单一用户所能使用的文件系统空间总量。

UNIX文件属性

本附录稍早，在“文件系统实现概况”的部分曾提及UNIX文件系统的实现，并说明inode的条目记录中包括了*metadata*：除了名称之外，有关文件的相关信息。现在我们要讨论的就是这些属性，因为它们与文件系统的用户息息相关。

注17： GB = gigabyte，约十亿字节。在计算机里，使用G如果非度量衡单位，即表示2^{30} = 1,073,741,824。

文件所有权与权限

或许，与单一用户的个人计算机文件系统比起来，UNIX文件的最大不同之处就在于所有权（ownership）与权限（permissions）了。

所有权

在很多的个人计算机上，任何的进程或用户都能读取或写入所有文件，因此计算机病毒现在对读者来说非常熟悉。但是因为UNIX用户能访问的文件系统是受到限制的，所以要替换或破坏重要的文件系统元件很难：病毒很少对UNIX系统造成问题。

UNIX文件有两种所有权（或称所有权）：用户（user）与组（group），它们各有自己的权限。一般来说，文件的所有者应具完整的访问权，而该所有者的工作组的成员拥有的权限会有些许限制，除此之外的其他人，权限就再更少一些。前述最后的类别，在UNIX的文件里称之为其他人（other）。文件所有权可使用`ls`命令的冗长模式来显示。

新的文件通常会继承其创造者的所有者与组成员，如果要给予适当的权限，则只有系统管理员通过`chown`与`chgrp`命令改变它们的属性。

在inode条目记录中，用户与组都以数字识别而非名称。因为人们通常偏好以名称识别，因此系统管理员提供对应的表格，一直以来我们都称为密码文件：`/etc/passwd`与组文件`/etc/group`。在大型站点，这些文件多半会替换为某种网络分布式的数据库形式。这些文件或是数据库，任何登录的用户都可读取，不过现今偏向使用程序库调用`setpwent()`、`getpwent()`与`endpwent()`访问密码数据库、使用`setgrent()`、`getgrent()`与`endgrent()`访问组数据库：参考*getpwent*(3)与*getgrent*(3)的手册页。如果你的站点使用数据库取代`/etc`下的文件，你可以试试Shell命令：`ypcat passwd`检查密码数据库，或`ypmatch joines passwd`寻找用户`jones`的条目记录。如果你的站台使用NIS+而非NIS，则`yp`命令应改为`niscat passwd.org_dir`与`nismatch name=jones passwd.org_dir`。

重点部分是通过用户与组标识符的数字值来控制访问。如果一文件系统通过用户`smith`以user ID 100被加载或导入，则一个文件系统的user ID 100指定给用户`jones`，那么`jones`便能完整访问`smith`的文件。就算目标系统下还有另一个`smith`用户也一样。这类的考虑在大型组织的UNIX文件系统下就相当重要了，因它面向全局性可访问的UNIX文件系统：用户与组的识别必须涵盖整个组织范围，是必须的考虑。问题不是只有这里讲的这么简单：用户与组的标识符也有诸多限制。旧式UNIX系统仅能为每一个配置16位，也就是总计为2^{16} = 65 536个值。较新的UNIX系统则允许32位的标识

符，遗憾的是，它们有许多都被加诸严格的限制，大大地限制了标识符的数目，这些数字对大型企业而言仍嫌不足。

权限

UNIX文件系统权限有三种类型：读取（read）、写入（write）与执行（execute）。它们每一个在inode数据结构里都只需要单一位，即可指出权限的存在与否。它们会分别针对用户、组，与其他人设置权限。文件权限可通过ls命令的冗长模式显示，通过chmod命令变更。因为权限每个设置都只需要三个位，因此它可以单一八进制（注18）数字表示，chmod命令也接受3个或4个八进制数字的参数或符号形式。

chmod

语法

chmod [options] *mode file(s)*

主要选项

-f

强制变更，如果可能的话（如果失败，不要显示信息）。

-R

将变更递归地应用到整个目录。

用途

变更文件或目录的权限。

行为模式

必需的参数*mode*，可以是绝对性的3个或4个八进制数字之一个权限掩码，或是一或多个字母的符号表示：a（全部，同于ugo）、g（组）、o（其他人）、或u（用户），再接上=（设置）、+（加入），或-（除去），最后则是一或多个r（读取）、w（写入），或x（执行）。多符号的设置需以逗点分隔，因此，755的模式，等同于u=rwx,go=rx、a=rx,u+w与a=rwx,go-w。

警告

递归的形式是相当危险，请谨慎使用！它可能会因误用chmod -R应用，而需要从备份媒体中恢复整个文件树。

注18：BSD系统例外：它们提供*sappnd*与*uappnd*标志，可使用chflags设置之。

注意： 有些操作系统支持额外的权限。其中有个很有用、但UNIX没有的权限，叫作附加权限（注19）：它在日志文件上特别好用，可用来确保数据只能被加入，但现存的数据不能被更改。当然，如果此文件能被删除，就能再替换为变更数据后的文件，所以附加权限提供的安全性只是错觉。

默认权限的设置会应用至每一个新建立的文件：它们由umask命令控制，以给定的参数设置默认值，如未提供参数，则直接显示默认值。umask的值为三个八进制数字，表示要被拿走的权限：通常值为077，指的是给用户完整的权限（读取、写入、执行），而组与其他人都不具任何权限。其结果为新建立之文件，限制在只有拥有它们的用户可以访问。

现在让我们来看看这些文件权限：

```
$ umask                                          显示当前的权限掩码
2
$ touch foo                                      建立一个空文件
$ ls -l foo                                      列出与文件相关的信息
-rw-rw-r--    1 jones    devel                   0 2002-09-21 16:16 foo
$ rm foo                                         删除文件
$ ls -l foo                                      再次列出与文件相关的信息
ls: foo: No such file or directory
```

一开始，权限掩码为2（确切说法为002），即删除其他人的写入权限。touch只是更新文件最后写入的时间戳，如有需要时建立文件。ls -l命令为冗长式文件列出的惯用语法。它报告了-的文件类型（一般文件）与权限字符串rw-rw-r--（指的是用户与组具读取与写入权限，其他人则具读取权限）等信息。

我们将掩码改为023之后重建文件，以删除组的写入权限与其他人的写入与执行的权限。会看到这样的权限字符串：rw-r--r--，也就是我们所预期的：删除组与其他人的写入权限：

```
$ umask 023                                      重设权限掩码
$ touch foo                                      建立空文件
$ ls -l foo                                      列出文件相关信息
-rw-r--r--    1 jones    devel                   0 2002-09-21 16:16 foo
```

权限运作

什么是执行权限？文件通常不具此权限，除非它们是可以执行的程序或脚本。通常这类程序的连接器都会自动地加上执行权限，不过脚本不会，我们得自行使用chmod变更。

注19： 默认权限。

在复制一个拥有执行权限的文件，例如/bin/pwd，该权限会被保留，除非umask的值使得它们被删除：

```
$ umask                                    显示当前的权限掩码
023
$ rm -f foo                                删除任何存在的文件
$ cp /bin/pwd foo                          复制一份系统命令
$ ls -l /bin/pwd foo                       列出文件相关信息
-rwxr-xr-x    1 root     root        10428 2001-07-23 10:23 /bin/pwd
-rwxr-xr--    1 jones    devel       10428 2002-09-21 16:37 foo
```

最后结果rwxr-xr--反映部分权限的消失：组的写入访问消失、其他人的写入与执行也不存在。

最后，我们使用符号形式的参数执行chmod，为所有人加入执行权限：

```
$ chmod a+x foo                            为所有人加入执行权限
$ ls -l foo                                列出冗长式文件信息
-rwxr-xr-x    1 jones    devel       10428 2002-09-21 16:37 foo
```

最后的权限字符串为rwxr-xr-x：用户、组与其他人均可执行。这里要注意的是，权限掩码不会对chmod操作造成影响：掩码只在文件建立的时候有影响。至于复制的文件，行为模式和原始的pwd命令一样：

```
$ /bin/pwd                                 尝试系统版本
/tmp
$ pwd                                      以及 Shell 内置版本
/tmp
$ ./foo                                    还有我们对系统版本所复制的
/tmp
$ file foo /bin/pwd                        查看这些文件的信息
foo:      ELF 32位 LSB executable, Intel 80386, version 1,
          dynamically linked (uses shared libs), stripped
/bin/pwd: ELF 32位 LSB executable, Intel 80386, version 1,
          dynamically linked (uses shared libs), stripped
```

请注意我们在引用foo时加上目录前置字符：基于安全性理由，绝不要在PATH列表里包括当前目录。如果你一定要这么做，也请你将它放在最后一个！

警告： 如果你试过上述这些，在试图执行/tmp下的命令时，可能会得到permission-denied的回应。在提供这样功能的系统上，如GNU/Linux，系统管理员有时会以没有执行权限的模式加载这个目录（tmp）；请检查/etc/fstab下是否有noexec选项。使用这个选项的另一个理由是为了避免特洛伊木马脚本（参考第15章）在像/tmp这样公开可写入的目录下被执行。你仍然可以将它们放入Shell中而执行，但是你需要知道为什么要这么做。

下面是你删除可执行权限又试图执行程序时，会发生的事：

```
$ chmod a-x foo                                  删除所有人的可执行权限
$ ls -l foo                                      列出冗长式文件信息
-rw-r--r--    1 jones    devel        10428 2002-09-21 16:37 foo
$ ./foo                                          试图执行程序
bash: ./foo: Permission denied
```

这里指的不是文件是否有像可执行程序一样的执行能力(ability),而是是否拥有执行权限(possession of execute permission),决定了它能否像命令一样被执行。这是UNIX里一个很重要的安全功能。

当你提供执行权限给不应该具有此权限的文件时:

```
$ umask 002                                      删除默认的都可写入权限
$ rm -f foo                                      删除任何已存在的文件
$ echo 'Hello, world' > foo                      建立一个单行文件
$ chmod a+x foo                                  使之可执行
$ ls -l foo                                      显示我们做的变更
-rwxrwxr-x    1 jones    devel           13 2002-09-21 16:51 foo
$ ./foo                                          试图执行程序
./foo: line 1: Hello,: command not found
$ echo $?                                        显示退出状态码
127
```

Shell会要求内核执行./foo及得到失败报告,其使用设置为ENOEXEC的程序库错误指示器。Shell接下来会试着自己执行它。在命令行上Hello, world被解释为命令Hello、参数world。因为在查找路径下找不到这样的命令,所以Shell报告一连串的错误信息,并回传127退出状态码,详见6.2节。

检查权限时,依序为用户、组,最后才是其他人。它们是由所属的进程决定该设置哪些权限位。因此很可能文件属于你,但你却不能读,而你的组成员及系统里的其他人却可以。像这样:

```
$ echo 'This is a secret' > top-secret   建立单行文件
$ chmod 044 top-secret                   对组与其他人,删除所有权限只保留读取权限
$ ls -l                                  显示我们的变更
----r--r--    1 jones    devel          17 2002-10-11 14:59 top-secret
$ cat top-secret                         试着显示文件
cat: top-secret: Permission denied
$ chmod u+r top-secret                   允许所有者读取文件
$ ls -l                                  显示我们的变更
-r--r--r--    1 jones    devel          17 2002-10-11 14:59 top-secret
$ cat top-secret                         这时,便能显示了!
This is a secret
```

所有UNIX文件系统都另提供额外的权限位:*set-user-ID*、*set-group-ID*与*sticky*(粘连)位。为兼容旧系统并避免增加已存在的行长度,ls不使用三个额外的权限字符来显示这些权限,而是将x改为其他字母。详见*chmod*(1)、*chmod*(2)与*ls*(1)手册页。基于安全性理由,Shell脚本绝不应该设置set-user-ID或set-group-ID权限位:我们发

现太多这类脚本里可怕的安全性漏洞。这些权限位与Shell脚本的安全性议题，在第15章已说明过。

有时我们会在商用软件上应用仅允许执行的权限（--x--x--x），以禁止复制、除虫，与追踪操作，但程序仍可以执行。

目录权限

现在为止，我们讨论的都是一般文件的权限。在目录上，这些权限的解读会稍有不同。目录的读取，即能列出它的内容，例如使用ls。写入，则表示你能在目录下建立或删除文件，即便你对目录下的文件不具写入权限：该特权保留给操作系统，以维持文件系统的一致性。执行访问，即你可以访问文件以及该目录下的子目录（当然受它们自己权限的管制），特别是你还可以跟随该目录下的路径名称。

由于目录的执行与读取较难区分，我们在这里举例解释：

```
$ umask                                          显示当前的权限掩码
22

$ mkdir test                                     建立子目录
$ ls -Fld test                                   显示目录权限
drwxr-xr-x  2 jones devel 512 Jul 31 13:34 test/

$ touch test/the-file                            建立空目录
$ ls -l test                                     目录内容冗长式列出
-rw-r--r--  1 jones devel 0 Jul 31 13:34 test/the-file
```

至此，都为一般行为模式。现在，我们删除读取权限，但留下执行权限：

```
$ chmod a-r test                                 删除所有人读取目录的权限
$ ls -lFd test                                   显示目录权限
d-wx--x--x  2 jones devel 512 Jan 31 16:39 test/

$ ls -l test                                     试图列出目录内容
ls: test: Permission denied

$ ls -l test/the-file                            列出文件本身
-rw-r--r--  1 jones devel 0 Jul 31 13:34 test/the-file
```

第二个ls失败是因为缺乏读取权限，但因为有执行权限，所以第三个ls成功。这里呈现的是：删除目录的读取权限并不能防止目录下的文件被访问，用户只要知道文件名就可以这么做。

当我们删除执行访问，却未恢复读取权限时：

```
$ chmod a-x test                                 删除所有人执行目录的权限
$ ls -lFd test                                   列出目录
d-w-------  3 jones devel 512 Jul 31 13:34 test/
```

```
$ ls -l test                                    试图列出目录内容
ls: test: Permission denied

$ ls -l test/the-file                           试图列出文件
ls: test/the-file: Permission denied

$ cd test                                       试图改变目录
test: Permission denied.
```

目录树不再允许所有用户浏览，root 除外。

最后我们恢复读取，但不要恢复执行访问，再重复刚刚做的事：

```
$ chmod a+r test                                加入所有人读取目录的权限
$ ls -lFd test                                  显示目录权限
drw-r--r--  2 jones devel 512 Jul 31 13:34 test/

$ ls -l test                                    试图列出目录内容
ls: test/the-file: Permission denied
total 0

$ ls -l test/the-file                           试图列出文件
ls: test/the-file: Permission denied

$ cd test                                       试图改变目录
test: Permission denied.
```

缺乏对目录的执行权限，是无法浏览它的内容，或是不能使它成为当前的工作目录。

目录设置黏着位时，里头所含的所有文件就只有它们的所有者或目录所有者才能删除。此功能最常用在公用的可写入目录，例如/tmp、/var/tmp（过去的/usr/tmp）这些，还有邮件进来的目录，以防止用户删除不属于他们的文件。

某些系统上，目录设置set-group-ID位时，新建立的文件的组ID即为此目录的组ID而非所有者所属组。可惜的是，这样的权限位并非在所有系统下都如此处理。另有一些系统，其行为模式需视加载的文件系统为何而定，所以你应该在你系统里，再确认一次mount命令的手册页。当有好几个用户协同开发项目时，set-group-ID位的设置就相当好用了。我们可以为此项目建立一个特殊的组，并建立成员，然后再将项目的目录设置给该组。

部分系统结合set-group-ID位的设置与group-execute位，这种用法太过复杂，已超出本书范围，在此不进行介绍。

目录读取与执行权限

为什么读取目录与通过该目录至其子目录有不同的含义？答案很简单：它是为了在看不到父目录的情况下，仍能看到子目录下的文件。最常见的使用就是在用户的网页结构下。根目录通常为rwx--x--x这样的权限，防止组或其他人列出目录内容或检查文件。但网页目录的起始，假设是$HOME/public_html，包含其子目录，我们可以给予它们rwxr-xr-x这样的权限，且在那之下的文件，则至少拥有rw-r--r--的权限。

另一个例子是，假设为了安全性的理由，系统管理员想要对先前未防护的文件子目录进行读取保护（read-protect）。他需要做的就只是删除该子目录顶层的根目录（单一目录）之读取与执行权限chmod a-rx *dirname*即可；这使所有这之下的文件都立即无法新的打开（但已打开的则不受影响），即便他们拥有个别文件的使用权限。

注意： 有些UNIX系统支持访问控制列表（access control lists，ACL）。它可以提供较细的访问控制，可针对个别的用户与组指定非默认的权限。可惜的是，ACL工具的设置与显示在各系统间都不尽相同，使其难以在异构环境中使用，在本书中做进一步讨论也不适当。如果你想了解更多，可以试着使用man -k acl或man -k 'access control list'，在你的系统下查找相关的命令。

文件时间戳

UNIX文件的inode条目记录包括三个重要时间戳：访问时间、inode变更时间与修改时间。这些时间一般是自*epoch*(注20)算起的秒数计之，epoch的UNIX系统时间为00:00:00 UTC, January 1, 1970，不过有些UNIX实现提供更好的计时单位。以UTC（注21）[国际标准时间（Coordinated Universal Time），早期为格林威治时间（Greenwich Mean Time，GMT)] 计算的时间，表示该时间戳不受本地时区设置影响。

访问时间是通过数个系统调用而被更新，包括那些读取与写入文件的操作。

注20： **epoch**，ep'ok，名词。用以编号之后年度的一个固定时间点。

注21： 经委员会一致通过：UTC为不受语言影响的字母缩略字，法文的展开为Temps Universel Coordonné。见*http://www.npl.co.uk/time/time_scales.html*、*http://aa.usno.navy.mil/faq/docs/UT.html*，与*http://www.boulder.nist.gov/timefreq/general/misc.htm*，了解更多与时间标准有关的信息。

inode变更时间是在文件建立之初，以及inode metadata被修改时被设置。

修改时间的变更是在文件块被更改，而非metadata（文件名、用户、组、连接计数或权限）变更时。

touch命令，或utime()系统调用，可用于改变文件访问与修改时间，但不会改变inode变更时间。近期的GNU touch版本提供选项，可针对文件标明时间。ls -l命令显示的是修改时间，但加上-c选项，则可显示inode变更时间；加上-u选项，会显示访问时间。

这些时间戳都不够完美。inode变更时间表示两种完全不同的目的，应该已经个别分开地被记录下来。因此，它并不能告诉你这个文件首度出现在UNIX文件系统里的确切时间。

访问时间在以read()系统调用读取文件时被更新，而不是在使用mmap()对应文件到内存以及以该方式读取文件时。

修改时间可能稍具可靠性，不过文件复制命令通常都会重设输出文件的修改时间为当前时间，即便它的内容完全没有变更，这并非我们所希望的，所以，复制命令cp提供-p选项，让你可以保留文件的修改时间。

最后备份的时间不会被记录：即备份系统必须保留补助性的数据，以追踪自最后一次备份至今已进行修改的文件。

注意： 文件系统备份软件，在保留文件时间戳这部分都相当谨慎处理。否则在每次备份之后，所有文件都看起来像刚被读取。使用打包工具，例如tar，作备份的系统，都必须更新inode变更时间，使得该时间戳无法再用于其他用途。

基于某些目的，有人希望能将读取、写入、更名、改变metadata的时间戳分开记录，这样的分隔方式在UNIX里是不可能的。

文件连接

尽管我们在本附录之前的“文件系统实现概况”讨论过，硬连接与软连接（符号连接）的工具非常多。但它们其实遭到一些非难，意见无非是同一个东西，给它多个名称只会混淆用户，因为连接是让两个已隔离的文件树接在一起。移动了含有连接的子树就会切断连接，让文件系统产生不一致的情况。图B-4展现的是因删除而切断了软连接的情况，而图B-5则是告诉你如何保留这样的连接，这完全是看你在建立连接时，是以相对还是绝对路径而定。

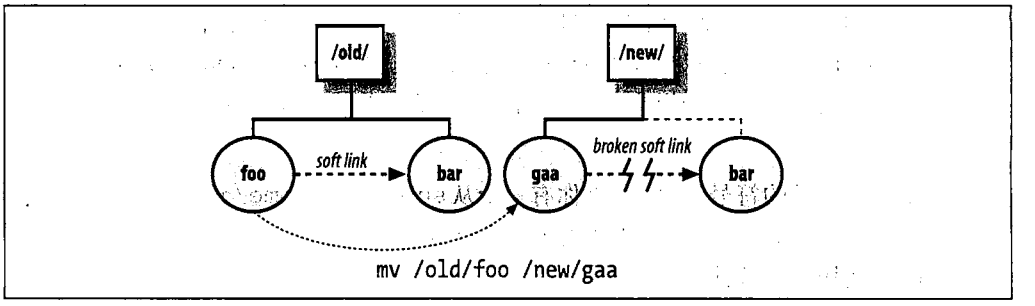


图 B-4：移动切断了软连接

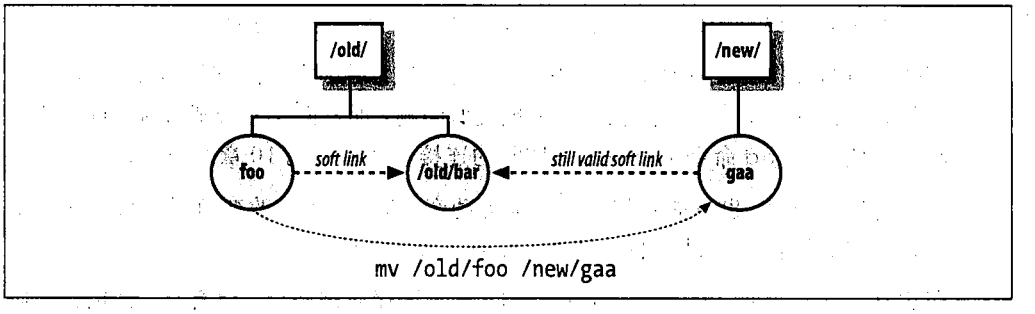


图 B-5：移动可以保留绝对符号连接

以下为硬连接与软连接会出现的其他问题：

- 当连接的文件更新时，不管它是被文件复制命令或是程序（例如，文本编辑程序）所替换，硬连接是否仍被保留根据更新的方式而定。如果是打开已存在的文件供输出及重写入，其inode编号保留不变，则硬连接仍会保留。然后，如果系统崩溃或磁盘溢满产生错误，则在更新期间可能导致遗失整个文件。比较小心的程序员可能就会在临时名称下编写新的版本，而且只有确定复制完成时，才删除原始的那个（因此连接计数减1）并更改副本的名字。剩下的隐匿性很快，所以针对失误的窗口是较小的。替换文件会产生一个新的inode编号及连接计数1，并切断硬连接。

  我们测试了许多文本编辑器，发现似乎都是使用第一种方式，保留硬连接。emacs编辑器则允许在两种方式择一（注22）。相对地，如果你编辑或重写的文件是软连接，那么你编修的就是原始数据；且只要它的路径名称仍未改变，则所有指向它的其他软连接，都会反映此更新过的内容。

注22：将变量backup-by-copying-when-linked设为非-nil (non-nil)，及backup-by-copying设为nil，即可保留硬连接。可参考emacs手册里的*Copying versus Renaming*。

以硬连接而言，两种更新方式都可能导致新文件的所有者与组改变：更新位置（update-in-place）会保留所有者与组，但复制与更名（copy-and-rename）则会将值重设为执行此操作的用户。因此，两种连接的行为模式在文件修改之后时常是不一致的。

- 再来看看目录的符号连接：如果你有一个从subdir到/home/jones/somedir的符号连接，那么当你将文件树移到另一个没有/home/jones/somedir的文件系统下时，连接便会被截断。
- 在连接里通常使用相对路径比较好，而且是只有在目录位于同级或更低级的情况下：所以从subdir到../anotherdir的符号连接，只有在文件树至少比被移动的文件树高一层目录处开始才会被保留。否则，连接会被切断。
- 切断的符号连接无法在切断当时被发现，只有在之后你引用此连接时才会知道：这已经为时已晚。你的电话簿也可能出现类似问题：朋友搬了家没通知你，自此断了联系。使用find命令可以找出被切断的连接，请参考第10章的说明。
- 符号连接到目录，也可能对相对性目录更动产生问题：当你改变符号连接的父目录时，会移到被指向的目录的父目录，而非连接本身的父目录。
- 在建立文件打包时，符号连接会有问题：有时连接应被保留，但有时，打包文件应只是包括文件本身的副本而不是连接。

文件大小与时间戳的变化

每个文件包括的inode实体记录包含了它的字节大小，如果文件为空时它可以是零。ls输出的冗长模式，将大小显示在第5栏：

```
$ ls -l /bin/ksh                                    列出冗长模式的文件信息
-rwxr-xr-x    1 root     root       172316 2001-06-24 21:12 /bin/ksh
```

GNU版本的ls提供-S选项，以文件大小递减排序列出：

```
$ ls -lS /bin | head -n 8                    显示8个最大文件，并由大到小排列
total 7120
-rwxr-xr-x    1 rpm      rpm       1737960 2002-02-15 08:31 rpm
-rwxr-xr-x    1 root     root       519964 2001-07-09 06:56 bash
-rwxr-xr-x    1 root     root       472492 2001-06-24 20:08 ash.static
-rwxr-xr-x    2 root     root       404604 2001-07-30 12:46 zsh
-rwxr-xr-x    2 root     root       404604 2001-07-30 12:46 zsh-4.0.2
-rwxr-xr-x    1 root     root       387820 2002-01-28 04:10 vi
-rwxr-xr-x    1 root     root       288604 2001-06-24 21:45 tcsh
```

当文件系统使用空间已满，想要找出罪魁祸首时，-S选项就派得上用场了。当然，如果你的ls不提供此选项，你也只要使用ls -l *files* | sort -k5nr便能得到相同的结果。

注意：如果你怀疑某个正在执行的进程灌爆了文件系统，在Sun Solaris下，可以使用下列方式找到这个打开中的大文件（如果你想看到的不只是属于你的文件，请以root的身份执行）：

```
# ls -lS /proc/*/fd/*                          列出所有打开的文件
-rw------- 1 jones jones 111679057 Jan 29 17:23 /proc/2965/fd/4
-r--r--r-- 1 smith smith  946643 Dec  2 03:25 /proc/15993/fd/16
-r--r--r-- 1 smith smith  835284 Dec  2 03:32 /proc/15993/fd/9
...
```

本例中，删除2965可能就能删除这个大文件——至少你知道是jones引用它的了。

GNU/Linux也有/proc这样的工具机制，不过上面这个Solaris的解决方案在GNU/Linux下并不适用，因为它所报告的文件大小在GNU/Linux上是不正确的。

磁盘可用空间（disk-free）命令df用来报告当前磁盘的使用情况，或者你可加上-i选项了解inode的使用情况。磁盘使用情况命令du则可报告个别目录内容下的总使用空间，或辅以-s选项输出简洁的摘要。这些例子在第10章里都有。find命令搭配-mtime与-size选项，可以找出最近建立的或大小不寻常的文件，同样请参考第10章的说明。

在ls命令下使用-s选项可显示额外的开头栏位，其提供文件的块（block）大小：

```
$ ls -lgs /lib/lib* | head -n 4          以冗长模式列出前4个匹配文件的信息
2220 -r-xr-xr-t    1 sys  2270300 Nov  4  1999 /lib/libc.so.1
  60 -r--r--r--    1 sys    59348 Nov  4  1999 /lib/libcpr.so
 108 -r--r--r--    1 sys   107676 Nov  4  1999 /lib/libdisk.so
  28 -r--r--r--    1 sys    27832 Nov  4  1999 /lib/libmalloc.so
```

块大小与操作系统及文件系统息息相关：为了找到一个块的大小，可以字节为单位的文件大小除以用块为单位的文件大小，然后进制成2的次方，即可得知。以上述为例，我们发现2270300/2220 = 1022.6，所以其块大小为2^{10} = 1024字节。随着存储设备的技术越来越精进，我们以块大小算出来的值可能与它所呈现在设备上的值有所不同。且厂商与某些GNU的ls版本也不一致，因此有时以此法取得的块大小不见得可靠－除非是在同系统下使用同一个ls命令作对照。

注意：有时，你可能会遇到块小到有点奇怪的文件：像这样的文件多半有洞（hole），这是因为使用直接访问的方式写入字节在指定的位置。数据库程序就常这么做，因为它们是以松散式的表格存储在文件系统里。文件系统下的inode架构，处理有hole的文件时不会有问题，但对于读取这样文件的程序而言，它看到的可能是（想像的）磁盘块所对应至hole的一连串零字节。

复制如此的文件会以实体的零磁盘块填满hole，这可能会增加文件的大小。虽然建立原始文件的软件不会感觉到它，但它是提供功能齐备的备份工具所需要处理的一个文件系统功能。GNU的tar提供--sparse选项以请求检查这类文件，不过其他的tar实例则不提供。另外，GNU的cp也支持--sparse选项，以处理这类带有hole的文件。

使用管理的输出/恢复（dump/restore）工具，可能是在复制文件树时，唯一可避免填满hole的方法了。各系统里的此类工具都有很大的差异，所以我们在本书中不做讨论。

你可能还会发现在最后两个范例的输出上有个地方不同：时间戳的表示方式。为尽量缩减行宽度，ls通常是以*Mmm dd hh:mm*表示最近6个月内的时间戳，而以*Mmm dd yyyy*表示6个月前的时间。有些人会觉得这样很麻烦，而现行许多窗口系统都已经没有旧式ASCII终端那种80个字符的行限制了，因此这种做法已经不是那么必要。不过大部分的人仍认为太长的行会很难阅读，且近期的GNU ls版本也致力于将显示的结果保持在简短的样式。

GNU的ls会依locale的设置，显示近似*yyyy-mm-dd hh:mm:ss*这样的格式，以符合*ISO 8601:2000: Data elements and interchange formats-Information interchange-Representation of dates and times*的定义，不过就像先前的例子会去除秒数部分。

GNU的ls里，选项--full-time可用来揭露文件系统里完整的时间戳记录，如第10章所述。

其他的文件metadata

剩下还有一些文件的属性记录在inode条目里，是我们还未提及的。不过在ls -l的输出里，还看到的部分就只有文件类型（file type）了，它记录在每行的第一个字符，就在权限的前面。-（连字号）指的是一般文件、d为目录，而l为符号连接。

这三种是我们在一般目录下常看到的，但在/dev下，你还会遇到至少这两种：b指块设备，c为字符设备。它们都与本书无关。

两种其他较少见的文件类型，例如p指的是命名的管道（named pipe），s指的是Socket（一种特殊的网络连接）。Socket为较高级的范畴，本书不作介绍。命名的管道则在程序与Shell脚本里偶尔用到：它们可以允许用户端和服务器端通过文件系统命名空间来沟通，并提供将一个进程的输出导向两个或两个以上不相关进程的方式。它们广义化一般的管道，后者只有一个写入与一个读取。

GNU *coreutils*包里的stat命令会显示stat()系统调用的结果，回传文件的inode信息。下面为SGI IRIX里的使用范例：

```
$ stat /bin/true                                    报告文件的 inode 信息
  File: `/bin/true'
  Size: 312             Blocks: 8          IO Block: 65536  regular file
Device: eeh/238d        Inode: 380         Links: 1
Access: (0755/-rwxr-xr-x)  Uid: (    0/    root)   Gid: (    0/     sys)
Access: 2003-12-09 09:02:56.572619600 -0700
```

```
Modify: 1999-11-04 12:07:38.887783200 -0700
Change: 1999-11-04 12:07:38.888253600 -0700
```

这里显示的stat子集信息，比ls更细微。

GNU的stat也支持设计更精细的报告，让你选择其中的子集输出。例如，软件安装包可使用它们，以找出文件系统是否仍有足够的空间可执行安装。详见stat手册页。

只有少数的UNIX版本（FreeBSD、GNU/Linux、NetBSD与SGI IRIX）支持原始的stat命令。这里举三个例子如下：

```
$ /usr/bin/stat /usr/bin/true    reeBSD 5.0（较长的输出，已缩减长度以符合解说页面）
1027 1366263 -r-xr-xr-x 1 root wheel 5464488 3120 "Dec  2 18:48:36 2003"
"Jan 16 13:29:56 2003" "Apr  4 09:14:03 2003" 16384 8 /usr/bin/true

$ stat -t /bin/true              GNU/Linux 简洁的 inode 信息
/bin/true 312 8 81ed 0 0 ee 380 1 0 0 1070985776 941742458 941742458 65536

$ /sbin/stat /bin/true           SGI IRIX 系统工具程序
      /bin/true:
        inode 380; dev 238; links 1; size 312
        regular; mode is rwxr-xr-x; uid 0 (root); gid 0 (sys)
        projid 0        st_fstype: xfs
        change time - Thu Nov  4 12:07:38 1999 <941742458>
        access time - Tue Dec  9 09:02:56 2003 <1070985776>
        modify time - Thu Nov  4 12:07:38 1999 <941742458>
```

UNIX文件的所有权与隐私权议题

我们已提及太多与文件权限相关的议题，让你了解如何控制文件与目录的读取、写入与执行的访问。你可以，也应该注意文件权限的选择，以掌控能访问你文件的有哪些人。

访问控制中最重要的工具就是umask命令了，因为它可以针对接下来建立的所有文件限制指定的权限。通常你会使用默认值，而它是设置在你Shell启动时所读取的文件里，以类似sh的Shell而言为$HOME/.profile文件，见14.7节。如Shell有支持，系统管理员通常会在对应的系统面起始文件内放置umask的设置。在协力合作的研究环境下，你应选择022掩码值，删除组与其他人的写入权限。以学生使用的环境来看，077的掩码值较适合，可剔除所有除了所有者（与root）以外的访问。

需要非默认权限时，Shell脚本应于开始处明白地直接下达umask命令，这个操作必须在所有文件建立之前做。不过，像这样的设置不会影响在命令行上被重定向的那些文件，因为它们在脚本起始时，已被打开。

第二个重要的工具就是chmod命令了：你应该好好了解它。即便是在开放所有人读取的公开环境下，文件与目录仍应多作限制。包括邮件文件、网页浏览器历史记录与缓存、

私有信件、财务与个人数据、营销计划等。邮件客户端与浏览器通常使用的是默认的限制性权限，但你以文本编辑器建立的文件就必须自行使用chmod变更了。如果你想要更谨慎行事，那么请不要使用文本编辑程序建立文件：你可以先以touch建立空文件，执行chmod后再编辑它。

你应该还记得，系统管理员拥有你文件系统的完整访问权，他可以读取任何文件。虽然大部分的系统管理员认为未经文件所有者的允许，查看用户文件是不道德的，但部分组织却认为，所有计算机文件，包括电子邮件，都属于公司财产，应24小时监控。这点的合法性其实很模糊，且世界各国标准不一。

加密与数据安全性

如果你希望存储的文件除了你之外（几乎）没有任何人可以读取，那么就需要用到加密。由于各国政府的输出条例将加密工具视为武器，因此大部分UNIX厂商不会在它的标准发布包里随附加密软件。在你开始安装网络上找到的或是商用的加密软件之前，我们有以下几点建议：

- 安全性是一个程序，而非产品。有本书可以让你有更深入的了解：《Secrets and Lies: Digital Security in a Networked World》（Wiley）。
- 你是否曾忘记你的加密密钥，或是离职员工留下的密钥不正确，这都可能漏失你的数据：良好的加密方式，通常无法在你要的时间之内破解。
- 就像员工离职你可能会换门锁一样，你应该相信使用前职员的加密密钥是不可靠的，你应该使用新的密钥重新加密曾以先前的密钥加密过的所有文件。
- 如果加密文件提升安全性，使得用户很不方便，它们可能会直接停用加密。

如果你想了解更多与加密算法相关的历史，建议你从《The Code Book: The Evolution of Secrecy from Mary, Queen of Scots, to Quantum Cryptography》（Doubleday）这本书开始。如果你觉得很有兴趣，想了解更多算法的细节，你可以继续看《Applied Cryptography: Protocols, Algorithms, and Source Code in C》（Wiley）。*http://www.math.utah.edu/pub/tex/bib/index-table.html* 里，还有更多相关学术议题的参考文献供你研究。

最后，在这个网络计算机的时代，你很可能会被网络与文件系统或是操作系统分开，除非你的网络通信确定相当安全，否则你的数据其实很危险。无线网络的弱点又更多，软件可以无声无息地窃听你的网络通信，利用现行无线加密通信协议的弱点，解出加密的通信数据。远端访问你的电子邮件，还有互动式的信息系统，可能都是不安全的。如果

你还在使用telnet，或非匿名式的ftp连接计算机，请立即切换为安全的Shell（secure Shell）（注23）。旧式的这些通信软件会以明码文本传递所有数据，包括用户名称与密码；网络攻击者可以轻松地取得这类数据。*secure Shell*软件使用健全的公钥加密，完成安全交换数据的操作，它会以随机产生长的加密密钥与其他许多较简单与较快的加密算法一起使用。

用户的数据会等到加密通道建立才开始传送，而标准的加密方法也经过深度的考虑，普遍相信是十分安全的。攻击者看到你的封包时，是经过随机字节组流加密后的样子，不过来源与目地端地址都看得到，也可能拿来分析。secure Shell也会为X Window System的数据建立安全通道，不过如果攻击者在你和你的计算机间动手脚，这么做也是于事无补的。网吧、键盘探测、无线网络等等，都可能让攻击大行其道，而令secure Shell无用武之地。

UNIX扩展文件名惯例

有些操作系统，使用主要名称、一个点号，及1～3个字符的文件类型或文件扩展作为文件名的形式。这些扩展有其重要目的：指出文件内容属于哪种特定的数据类型。例如，扩展文件名为pas指的是文件内容为Pascal的源代码，而exe指的则为二进制可执行文件。

这里并不保证文件扩展必会反映文件内容，不过大部分用户觉得这么做很好用，便遵循这一惯例。

UNIX也提供不少的通用文件扩展，但UNIX的文件名并未强制必须有一个点号。有时，文件扩展只是个惯例而已（对大部分的脚本语言来说），但编译器通常会要求特定的文件扩展及使用主文件名（截去扩展部分），以形成其他相关文件的名称。较常见的几种扩展见表B-1。

表B-1：常见的UNIX文件扩展

扩展	内容
1	数字的1。手册页的Section 1（用户命令）
a	程序库打包文件
awk	awk语言原始文件

注23：例如*http://www.columbia.edu/kermit/*、*http://www.ssh.com/*与*http://www.openssh.org/*。要更深入了解SSH，可参考《The Secure Shell: The Definitive Guide》（O'Reilly）一书。

表B-1：常见的UNIX文件扩展（续）

扩展	内容
bz2	以bzip2压缩的文件
c	C语言原始文件
cc C cpp cxx	C++语言原始文件
eps ps	PostScript页面描述语言原始文件
f	Fortran 77语言原始文件
gz	由gzip压缩的文件
f90	Fortran 90/95/200x语言原始文件
h	C语言标头文件
html htm	超文本标记语言（HyperText Markup Language）文件
o	目标文件（来自大部分编译程序语言）
pdf	可移植式文件格式文件
s	汇编语言源文件（例如，编译器的输出，以反映符号编码选项-S）
sh	Bourne系列Shell脚本
so	共享对象库（部分系统称之为动态载入库）
tar	磁带打包文件（产生自tar工具程序）
,v	cvs与rcs的历史文件
z	以pack压缩的文件（极少见）
Z	以compress压缩的文件

此表格最明显的就是少了exe。虽然许多操作系统都使用它作为二进制可执行程序的扩展文件名，且允许在使用程序时，省略扩展文件名。但UNIX本身并未在可执行文件上应用任何特定的扩展（文件权限已经做了这件事了），UNIX的软件极少允许省略文件名扩展。

有些UNIX文本编辑程序提供用户建立临时备份文件，这么一来，就算是在编辑较久的通信期中，也能每隔一段时间就将文件记录在文件系统里。对这类备份文件的命名，使用惯例有几种：将（#）或（~）字符前置于文件开头或后置于文件结尾，或以~加上数字的方式编排，例如.~1~、.~2~等等。后者的文件名产生编号方式模仿其他文件系统所提供的，UNIX本身并未提供这样的功能，不过其实UNIX的文件命名规则相当弹性，用户可以自行这么设置。

文件产生编号在其他系统下可保留文件的多个版本，而省略编号通常指的就是最高编号。UNIX提供更好的方式，处理文件版本的历史记录：通过软件工具，保留与主文件不同

部分的历史记录，并附上注释性的描述，说明改变部分。这类的包一开始是AT&T的*Source Code Control System*（sccs），现今则为*Revision Control System*（rcs）（见附录C“其他程序”）与*Concurrent Versions System*（cvs）较常见。

小结

我们带你将UNIX的文件系统完整看过一遍，现在，你应该对下面这些功能已经相当熟悉了：

- 文件为0至多个8位字节的流，文本文件里的行只能以换行字符来标示行界限。
- 字节通常以ASCII字符解释，但UTF-8编号与Unicode字符集让UNIX文件系统、管道与网络通信，能支持全世界书写世界上数百万种不同字符，且大部分现存文件或软件也能有效使用。
- 文件有属性，例如时间戳、所有权与权限。这可以让我们更有效地设置访问控制层级及隐私权，提供比其他桌面环境操作系统更好的支持，并剔除大部分计算机病毒的问题。
- 可以在目录的单一节点设置适当权限，控制整个目录树的访问。
- 极大的文件几乎不会造成什么问题，而在现行技术下，新的文件系统设计也能容许越来越大的文件。
- 文件名与路径名称的最大长度，已超出你会用到的最大长度。
- 简明的层级式目录架构，辅以斜杠分隔的路径组成部分，搭配mount命令的使用，几乎可以拥有无限大小的逻辑式文件系统。
- 尽可能地将所有数据看成文件，且鼓励这么做，可以简化人为的数据处理与使用。
- 文件名可使用NUL与斜杠以外的所有字符，但实务上，为了可移植性、可靠度，及Shell通配字符的考虑，大大地限制了应该可被使用的字符。
- 文件名的字母大小写将视为不同（除了Mac OS X的非UNIX HFS文件系统）。
- 虽然文件系统本身并未规定文件名结构，但许多程序仍预期文件名应有.加上扩展名称，并利用此扩展建立相关文件。Shell通过它们对通配字符样式的支持，如ch01.*与*.xml，来鼓励此实现。
- 文件名存储在目录文件里，而与文件相关的信息、文件*metadata*则另存储于inode实体记录中。
- 在相同文件系统内的文件与目录要移动或更名是相当快的，因为只有它们包含的目录实体需被更新，文件数据块本身则不会被访问。

- 硬连接与软连接允许同一实体文件拥有多个名称。硬连接只限在单一实体文件系统里才能做，但软连接则可指向逻辑文件系统的任一位置。
- inode表格大小为文件系统安装时就已固定，所以文件系统即使仍有空间置放文件数据，也可能出现已满的状态。

附录 C

重要的 UNIX 命令

现今 UNIX 系统都随附相当多命令。很多是特殊用途，也有很多是日常处理使用的，它们都能应用在交互模式下，或写在 Shell 脚本里。不过，我们不可能包括得了系统里所有的命令，也没有这么做的必要（像《UNIX in a Nutshell》的书籍对此部分有非常深入的描述）。

我们仍尽可能地找出有用的命令，也就是UNIX的用户或程序设计人员首先应了解的那些，简单做个介绍。这里也可能包括早期UNIX使用的旧式命令。本附录只是当你有志于成为UNIX的开发者时，建议你研究的命令列表。为求简洁，我们重新做了分类，用表这些命令列表，并进行简单的说明。

Shell 与内置命令

首先，我们先了解 Bourne Shell 的部分，特别是 POSIX 整理过的那些。bash 与 ksh93 都为 POSIX 兼容，而另外还有一些 Shell 在语法上也与 Bourne Shell 一致：

bash	GNU Project 的 Bourne-Again Shell。
ksh	Korn Shell —— 原始版本或分支体系版本，视操作系统而定。
pdksh	Public Domain Korn Shell。
sh	原始 Bourne Shell，特别是在商用的 UNIX 系统上。
zsh	Z-Shell。

你应该了解 Shell 内置命令的运作方式：

.	在当前 Shell 下，读取与执行给定的文件。
break	切断 for、select、until 或 while 循环。
cd	更改当前的目录。

command	规避函数的查找，直接执行正规的内置命令。
continue	开始 for、select、until，或while循环的下一个重复。
eval	将给定的文本视为 Shell 命令。
exec	无参数的情况下，改变 Shell 打开的文件。如带有参数，则以其他程序置换 Shell。
exit	退出 Shell 脚本，可选地带有特定的退出码。
export	将变量导出到接下来的程序环境中。
false	什么事也不做，指非成功的状态。用于 Shell 循环中。
getopts	处理命令行选项。
read	将输入行读进一个或多个 Shell 变量里。
readonly	将变量标记为只读，例：不可更改的。
return	返回自 Shell 函数而来的值。
set	显示Shell变量与变量值、设置Shell选项、设置命令行参数（$1、$2、…）。
shift	一次移动一个或多个命令行参数。
test	计算表达式，检测其为字符串、数字或文件属性相关的。
trap	管理操作系统信号。
true	什么事也不做，指成功的状态。用于 Shell 循环中。
type	指出命令的特性（关键字、内置命令、外部命令等等）。
typeset	声明变量与管理它们的类型与属性。
ulimit	设置或显示系统对每个进程所加诸的限制。
unset	删除 Shell 变量与函数。

下列为编写日常处理的 Shell 脚本的好用命令：

basename	显示路径名称的最后元件，并可选用地删除副文件名。主要用于命令替换。
dirname	显示除了路径名称最后组成部分以外的所有信息。主要用于命令替换。
env	处理命令的环境。
id	显示用户与组 ID 及名称信息。
date	显示现在的日期与时间，可选用地受用户提供的格式字符串所控制。
who	显示已登录的用户列表。
stty	处理当前终端设备的状态。

文本处理

下面的命令是做文本处理用。

awk	优雅又实用的程序语言，为许多大型 Shell 脚本的重要组成部分。
cat	连接文件。
cmp	简单的文件比较程序。

cut	剪下选定的列或字段。
dd	阻绝与解除阻绝数据的专门程序，也可执行ASCII与EBCDIC之间的转换。dd在产生设备文件原貌的副本时特别好用。需要特别注意的是，执行字符集转换时使用iconv较为合适。
echo	将参数打印到标准输出。
egrep	扩展的grep。使用扩展正则表达式(Extended Regular Expressions；ERE)进行匹配。
expand	展开制表字符与空格字符。
fgrep	快速grep。此程序使用与grep不同的算法匹配固定字符串。大部分的UNIX系统可同时查找多个固定字符串——不过并非全部。
fmt	将文本格式化为段落的简单工具。
grep	源自原始的ed行编辑命令g/re/p，“全局性匹配正则表达式并打印”。使用基本正则表达式（Basic Regular Expressions；BREs）进行匹配。
iconv	一般用途的字符编码转换工具。
join	自多个文件结合匹配的记录。
less	设计精良的交互式分页(pager)程序用以于终端上查看信息，一次显示屏幕所能显示的内容（页）。现已有GNU Project提供此程序，其名称为对应的more程序双关语。
more	原始的BSD UNIX交互式分页程序。
pr	将文件格式化，供行打打印机使用。
printf	echo的精装版，提供要被打印参数的控制方式。
sed	流编辑器，以ed行编辑器的命令集为基础。
sort	排序文本文件。命令行参数提供排序键值的指定与优先级控制。
spell	批次拼字检查程序。你也可以使用aspell或ispell封装成名为spell的Shell脚本。
tee	将标准输入拷贝到标准输出，或到一至多个指名的输出文件。
tr	转换、删除或减少重复字符的执行。
unexpand	将空格字符转换成适当数量的制表字符。
uniq	删除或计算已排序输入中的重复行。
wc	计算行、单词、字符或字节。

文件

与文件处理相关的命令：

bzip2, bunzip2	极高品质的文件压缩与解压缩。
chgrp	更改文件与目录的组。
chmod	更改文件与目录的权限（模式）。

chown	更改文件或目录的所有权。
cksum	显示文件的校验和（checksum）、POSIX 标准算法。
comm	显示或省略两个排序后的文件之间具有唯一性或共有的行。
cp	复制文件与目录。
df	显示可用磁盘空间。
diff	比较文件，显示其差异。
du	显示文件与目录所使用的磁盘块。
file	通过文件开头部分的检查，判断文件里的数据类型。
find	向下一个或多个目录阶层，寻找匹配于指定条件的文件系统对象（文件、目录、特殊文件）。
gzip, gunzip	高品质的文件压缩与解压缩。
head	显示一个或多个文件的前 *n* 行。
locate	以文件名称在系统里查找一文件。此程序使用定期自动重建的文件数据库中进行查找。
ls	列出文件。可使用选项控制要显示的信息。
md5sum	打印文件校验和，其使用 Message Digest 5（MD5）算法求出校验和。
mktemp	建立独一无二的临时文件，并显示其名称。非所有系统都可使用。
od	八进制输出；以八进制、十六进制或作为字符数据来打印文件内容。
patch	通过读取 diff 的输出，将给定的文件更新为新版本。
pwd	显示当前的工作目录。通常内置在现代的 Shell 中。
rm	删除文件与目录。
rmdir	只删除空目录。
strings	查找二进制文件中可打印的字符串，并显示它们。
tail	显示文件的最后 *n* 行。加上 -f 则继续打印（成长）文件的内容。
tar	磁带打包程序。现常被拿来作为软件发布的格式。
touch	更新文件的修改或访问时间。
umask	设置默认的文件建立权限掩码。
zip, unzip	文件打包与压缩 / 解压缩程序。ZIP 格式可使用于多种操作系统下，相当具有可移植性。

进程

以下为建立、删除，或管理进程所使用的命令：

at	在指定时间执行工作。at调度的工作只执行一次，而cron则为定期执行。
batch	在系统负载较不忙碌时，执行工作。
cron	在指定时间执行工作。

crontab	编辑每个用户的“cron表格”文件，指定应执行哪些命令，于何时执行。
fuser	寻找正在使用特定文件或socket的进程。
kill	传送信号到一或多个进程。
nice	在进程执行前，更改其优先级。
ps	进程状态。显示与正在执行中进程有关的信息。
renice	进程已被启动后，更改其优先级。
sleep	停止执行一段指定的秒数。
top	交互式显示系统上密集使用CPU的工作。
wait	Shell内置命令，等待一个或多个进程完成。
xargs	读取标准输入上的字符串，作为参数，尽可能地传递给指定的命令。多半会搭配find使用。

其他程序

还有些其他范畴的命令：

cvs	Concurrent Versions System，功能强大的源代码管理程序。
info	GNU Info系统，供在线文件浏览使用。
locale	显示可用的locale相关信息。
logger	通常是通过*syslog*(3)，传送信息到系统日志文件。
lp, lpr	将打印缓冲区文件传送给打印机。
lpq	显示正在处理中与在队列等待中的打印工作列表。
mail	传送电子邮件。
make	控制文件的编译与重新编译。
man	显示命令、程序库函数、系统调用、设备、文件格式与管理性命令的在线手册页。
scp	安全进行文件的远端复制。
ssh	安全的Shell。在执进程序或交互式登录的机器之间提供加密的连接。
uptime	显示系统已开机多久及其负载信息。

在其他种类中，针对Revision Control System（RCS）的相关命令如下：

ci	签入文件到RCS。
co	自RCS中签出文件。
rcs	在RCS控制下处理文件。
rcsdiff	对RCS控制下的文件的两个不同版本，执行diff。
rlog	为一至多个RCS所管理的文件，打印签入（check-in）日志。

参考书目

UNIX 程序员的手册

1. 《UNIX Time-sharing System: UNIX Programmers Manual》, Seventh Edition, Volumes 1, 2A, 2B. Bell Telephone Laboratories, Inc., January 1979.

   这些是第7版（Seventh Edition）UNIX系统的参考手册（Volume 1）及描述性报告（Volumes 2A & 2B），UNIX系统第7版是所有当前商用UNIX系统的直属祖先。

   它们由Holt Rinehart & Winston出版，但是现在已经很久未再印刷了。然而，它们在Bell Labs网站有在线形式，采用的格式有`troff`、PDF及PostScript。参考*http://plan9.bell-labs.com/7thEdMan*。

2. 你的UNIX程序设计手册。最有启发性的指导书之一，你可以从头读到尾（注1）（当UNIX系统已经长大，这会比以前更为困难）。如果你的UNIX厂商将它的文件印刷出书，则会较容易些。否则，需要从Seventh Edition手册开始，然后也需要阅读你的本地端文件。

使用 UNIX 思路进行程序设计

我们期待此书已经帮助你用现代文字学习“思考UNIX”。此列表的前两本书是UNIX“工具箱”程序设计方法论的原始展现。第3本书检查在UNIX下可用的、更广的程序设计

注1：在我作为合同程序员的某个夏日，我用几次午餐的时间从头至尾地阅读了这本手册。没想到我在那么短的时间里学到这么多知识。

工具。第4本与第5本是一般性的程序设计图书，且非常值得阅读。我们注意到由Brian Kernighan编写的任何书籍都值得仔细阅读，通常是要读许多遍。

1. 《Software Tools》, Brian W. Kernighan and P. J. Plauger. Addison-Wesley, Reading, MA, U.S.A., 1976. ISBN 0-201-03669-X.

   一本展现相当于UNIX grep、sort、ed等程序设计与源代码的很棒书籍（注2）。其程序使用Ratfor（Rational Fortran），是具有仿C控制结构的Fortran预处理器。

2. 《Software Tools in Pascal》, Brian W. Kernighan and P. J. Plauger. Addison-Wesley, Reading, MA, U.S.A., 1981. ISBN 0-201-10342-7.

   前一本书转成Pascal版。仍然值得阅读；Pascal提供了许多Fortran没有的事情。

3. 《The UNIX Programming Environment》, Brian W. Kernighan and Rob Pike. Prentice-Hall, Englewood Cliffs, NJ, U.S.A., 1984. ISBN 0-13-937699-2 (hardcover), 0-13-937681-X (paperback).

   本书的焦点在UNIX，并在该环境中使用工具。特别地，它增加重要的题材在Shell、awk及lex的使用上。参考 *http://cm.bell-labs.com/cm/cs/upe*。

4. 《The Elements of Programming Style》, Second Edition, Brian W. Kernighan and P. J. Plauger. McGraw-Hill, New York, NY, U.S.A., 1978. ISBN 0-07-034207-5.

   Strunk & White有名的《The Elements of Style》之后的经典书籍，本书描述可以用在任何环境的优良程序设计实务。

5. 《The Practice of Programming》, Brian W. Kernighan and Rob Pike. Addison-Wesley Longman, Reading, MA, U.S.A., 1999. ISBN 0-201-61586-X.

   与前一部书相似，但具有更强的技术性。参考 *http://cm.bell-labs.com/cm/cs/tpop*。

6. 《The Art of UNIX Programming》, Eric S. Raymond. Addison-Wesley, Reading, MA, U.S.A., 2003. ISBN 0-13-124085-4.

7. 《Programming Pearls》, First Edition, Jon Louis Bentley. Addison-Wesley, Reading, MA, U.S.A., 1986. ISBN 0-201-10331-1.

8. 《Programming Pearls》, Second Edition, Jon Louis Bentley. Addison-Wesley, Reading, MA, U.S.A., 2000. ISBN 0-201-65788-0.

   参考 *http://www.cs.bell-labs.com/cm/cs/pearls/*。

9. 《More Programming Pearls: Confessions of a Coder》, Jon Louis Bentley. Addison-Wesley, Reading, MA, U.S.A., 1988. ISBN 0-201-11889-0.

注2： 这本书长久地改变了我的生活。

Bentley 编写的象征 UNIX 精神的优秀书籍，且具有极少语言、算法设计，与更多的美妙范例。这些应该是在每个正规程序员的书架上。

10. 《Linux and the UNIX Philosophy》, Mike Gancarz. Digital Press, Bedford, MA, U.S.A., 2003. ISBN 1-55558-273-7.

Awk and Shell

1. 《The AWK Programming Language》, Alfred V. Aho, Brian W. Kernighan, and Peter J. Weinberger. Addison-Wesley, Reading, MA, U.S.A., 1987. ISBN 0-201-07981-X.

   awk 程序语言的原始定义。相当值得阅读。参考 *http://cm.bell-labs.com/cm/cs/awkbook*。

   《Effective awk Programming》, Third Edition, Arnold Robbins. O'Reilly, Sebastopol, CA, U.S.A., 2001. ISBN 0-596-00070-7.

   awk 的指导手册，涵盖 POSIX 标准的 awk。它也可作为 gawk 的用户指引。

2. 《The New KornShell Command and Programming Language》, Morris I. Bolsky and David G. Korn. Prentice-Hall, Englewood Cliffs, NJ, U.S.A., 1995. ISBN 0-13-182700-6.

   在 Korn Shell 上最可靠的作品。

3. 《Hands-On KornShell93 Programming》, Barry Rosenberg. Addison-Wesley Longman, Reading, MA, U.S.A., 1998. ISBN 0-201-31018-X.

标准

正式的标准文件是重要的，因为它们表示实现者与计算机系统用户之间的“合约”。

1. 《IEEE Standard 1003.1-2001: Standard for Information Technology-Portable Operating System Interface》(POSIX®) IEEE, New York, NY, U.S.A., 2001.

   这是次新的 POSIX 标准。它结合系统调用界面与 Shell 及工具标准在一份文件中。实际上此标准包括许多册，可在线取得（注 3），通过 PDF 电子格式打印（注 4），及由 CD-ROM 提供：

注 3：参考网址：*http://www.opengroup.org/online pubs/007904975*。

注 4：参考网址：*http://www.standards.ieee.org/*。

基本定义

此册提供标准的历史、术语的定义及文件格式与输出入格式的规范。ISBN 0-7381-3047-8；PDF: 0-7381-3010-9/SS94956；CD-ROM: 0-7381-3129-6/SE94956。

基本原理（信息型）

不是标准的正式部分，它不会在实例上加诸需求，此册提供 POSIX 标准中为何事情是如此的说明。ISBN 0-7381-3048-6；PDF: 0-7381-3010-9/SS94956；CD-ROM: 0-7381-3129-6/SE94956。

系统界面

描述C或C++程序员能看到操作系统的界面。ISBN 0-7381-3094-4；PDF: 0-7381-3010-9/SS94956；CD-ROM: 0-7381-3129-6/SE94956。

Shell 与工具

对于本书的读者更为相关：它描述在 Shell 与工具层次的操作系统。ISBN 0-7381-3050-8；PDF: 0-7381-3010-9/SS94956；CD-ROM: 0-7381-3129-6/SE9。

2. 《IEEE Standard 1003.1-2004: Standard for Information Technology-Portable Operating System Interface》(POSIX®) IEEE, New York, NY, U.S.A., 2004.

   最新的 POSIX 标准，当本书付印时发行。它是前一版的修订版，且以相似的方式组织。此标准包括许多册：基本定义（Volume 1）、系统界面（Volume 2）、Shell 与工具（Volume 3）和基本原理（Volume 4）。

   此标准可从 *http://www.standards.ieee.org/* 订购 CD-ROM（产品编号 SE95238，ISBN 0-7381-4049-X）或是 PDF（产品编号 SS95238，ISBN 0-7381-4048-1）。

3. 《The Unicode Standard》, Version 4.0, The Unicode Consortium. Addison-Wesley, Reading, MA, U.S.A., 2003. ISBN 0-321-18578-1.

4. XML 的标准；可从 *http://www.w3.org/TR/REC-xml/* 中取得。

安全与加密

1. 《PGP: Pretty Good Privacy》, Simson Garfinkel. O'Reilly, Sebastopol, CA, U.S.A., 1995. ISBN 1-56592-098-8.

2. 《The Official PGP User's Guide》, Philip R. Zimmermann. MIT Press, Cambridge, MA, U.S.A., 1995. ISBN 0-262-74017-6.

3. 《Practical UNIX & Internet Security》, Third Edition, Simson Garfinkel, Gene

Spafford, and Alan Schwartz. O'Reilly, Sebastopol, CA, U.S.A., 2003. ISBN 0-596-00323-4.

4. 《SSH, The Secure Shell: The Definitive Guide》, Second Edition, Daniel J. Barrett, Richard E. Silverman, and Robert G. Byrnes. O'Reilly Media, Sebastopol, CA, U.S.A., 2005. ISBN 0-596-00895-3.

5. 《Secrets and Lies: Digital Security in a Networked World》, Bruce Schneier. Wiley, New York, NY, U.S.A., 2000. ISBN 0-471-25311-1.

   本书深刻地揭露计算机安全性对每一个世界公民在生活上、数据上及个人自由上的含意。Bruce Schneier 如同 Brian Kernighan、Jon Bentley 与 Donald Knuth 一样，总是值得阅读的作家之一。

6. 《The Code Book: The Evolution of Secrecy from Mary, Queen of Scots, to Quantum Cryptography》, Simon Singh. Doubleday, New York, NY, U.S.A., 1999. ISBN 0-385-49531-5.

7. 《Applied Cryptography: Protocols, Algorithms, and Source Code in C》, Second Edition, Bruce Schneier. Wiley, New York, NY, U.S.A., 1996. ISBN 0-471-12845-7 (hardcover), 0-471-11709-9 (paperback).

8. 《Cryptographic Security Architecture: Design and Verification》, Peter Gutmann. Springer-Verlag, New York, NY, U.S.A., 2004. ISBN 0-387-95387-6.

UNIX 内部

1. 《Lions' Commentary on UNIX 6th Edition, with Source Code》, John Lions. Peer-to-Peer Communications, 1996. ISBN 1-57398-013-7.

2. 《The Design and Implementation of the 4.4BSD Operating System》, Marshall Kirk McKusick, Keith Bostic, Michael J. Karels, and John S. Quarterman. Addison-Wesley, Reading, MA, U.S.A., 1996. ISBN 0-201-54979-4.

3. 《UNIX Internals: The New Frontiers》, Uresh Vahalia. Prentice Hall, Englewood Cliffs, NJ, U.S.A., 1996. ISBN 0-13-101908-2.

O'Reilly 书籍

这里是 O'Reilly 书籍的列表。当然，还有许多与 UNIX 相关的 O'Reilly 书籍，可参考 *http://www.oreilly.com/catalog*。

1. 《Learning the bash Shell》, Third Edition, Cameron Newham and Bill Rosenblatt. O'Reilly, Sebastopol, CA, U.S.A., 2005. ISBN 0-596-00965-8.

2. 《Learning the Korn Shell》, Second Edition, Bill Rosenblatt and Arnold Robbins. O'Reilly, Sebastopol, CA, U.S.A., 2002. ISBN 0-596-00195-9.

3. 《Learning the UNIX Operating System》, Fifth Edition, Jerry Peek, Grace Todino, and John Strang. O'Reilly, Sebastopol, CA, U.S.A., 2001. ISBN 0-596-00261-0.

4. 《Linux in a NutShell》, Third Edition, Ellen Siever, Stephen Spainhour, Jessica P. Hekman, and Stephen Figgins. O'Reilly, Sebastopol, CA, U.S.A., 2000. ISBN 0-596-00025-1.

5. 《Mastering Regular Expressions》, Second Edition, Jeffrey E. F. Friedl. O'Reilly, Sebastopol, CA, U.S.A., 2002. ISBN 0-596-00289-0.

6. 《Managing Projects with GNU make》, Third Edition, Robert Mecklenburg, Andy Oram, and Steve Talbott. O'Reilly Media, Sebastopol, CA, U.S.A., 2005. ISBN: 0-596-00610-1.

7. 《sed and awk》, Second Edition, Dale Dougherty and Arnold Robbins. O'Reilly, Sebastopol, CA, U.S.A., 1997. ISBN 1-56592-225-5.

8. 《sed and awk Pocket Reference》, Second Edition, Arnold Robbins. O'Reilly, Sebastopol, CA, U.S.A., 2002. ISBN 0-596-00352-8.

9. 《UNIX in a NutShell》, Third Edition, Arnold Robbins. O'Reilly, Sebastopol, CA, U.S.A., 1999. ISBN 1-56592-427-4.

其他书籍

1. 《CUPS: Common UNIX Printing System》, Michael R. Sweet. SAMS Publishing, Indianapolis, IN, U.S.A., 2001. ISBN 0-672-32196-3.

2. 《SQL in a NutShell》, Kevin Kline and Daniel Kline. O'Reilly, Sebastopol, CA, U.S.A., 2000. ISBN 1-56592-744-3.

3. 《HTML & XHTML: The Definitive Guide》, Chuck Musciano and Bill Kennedy. O'Reilly, Sebastopol, CA, U.S.A., 2002. ISBN 0-596-00026-X.

4. 《The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary》, Eric S. Raymond. O'Reilly, Sebastopol, CA, U.S.A., 2001. ISBN 0-596-00131-2 (hardcover), 0-596-00108-8 (paperback).

5. 《Texinfo: The GNU Documentation Format》, Robert J. Chassell and Richard M. Stallman. Free Software Foundation, Cambridge, MA, U.S.A., 1999. ISBN 1-882114-67-1.

6. 《The TEXbook》, Donald E. Knuth. Addison-Wesley, Reading, MA, U.S.A., 1984. ISBN 0-201-13448-9.

7. 《The Art of Computer Programming, Volume 2: Seminumerical Algorithms》, Third Edition, Donald E. Knuth. Addison-Wesley, Reading, MA, U.S.A., 1997. ISBN 0-201-89684-2.

8. 《Literate Programming》, Donald E. Knuth. Stanford University Center for the Study of Language and Information, Stanford, CA, U.S.A., 1992. ISBN 0-937073-80-6 (paperback) and 0-937073-81-4 (hardcover).

9. 《Herman Hollerith —Forgotten Giant of Information Processing》, Geoffrey D. Austrian. Columbia University Press, New York, NY, U.S.A. 1982. ISBN 0-231-05146-8.

10. 《Father Son & Co. — My Life at IBM and Beyond》, Thomas J. Watson Jr. and Peter Petre. Bantam Books, New York, NY, U.S.A., 1990. ISBN 0-553-07011-8.

11. 《A Quarter Century of UNIX》, Peter H. Salus. Addison-Wesley, Reading, MA, U.S.A., 1994. ISBN 0-201-54777-5.

作者简介

Arnold Robbins是亚特兰大人，他是一位专业程序员与技术作家。他也是一位快乐的丈夫，4个可爱孩子的父亲，还是犹太教徒（Babylonian与Jerusalem）。1997年末，他与家人移居到以色列。

Arnold从1980年开始使用UNIX系统，那是一台执行第6版UNIX（Sixth Edition UNIX）的PDP-11。从1984年开始他已经着手Shell脚本程序编写，先从增强型Bourne Shell开始，然后使用Korn Shell与bash。

自1987年开始，Arnold也已经是重度的awk用户，当时他使用的是gawk——awk的GNU版。身为POSIX 1003.2的委员，他曾协助awk的POSIX标准建立。他目前是gawk及其文件的维护者。

他曾是个系统管理者以及UNIX与网络持续教育课程的老师。他也曾拥有一家创业型软件公司，但是他已不想再提起。他希望有一天能将他自己的网站放在*http://www.skeeve.com*。

O'Reilly已经使他很忙了。他是畅销书《Learning the vi Editor》、《Effective awk Programming》、《sed and awk》、《Learning the Korn Shell》、《UNIX in a Nutshell》以及许多口袋参考书的合著者。

Nelson H. F. Beebe是犹他大学数学系研究所教授，具有化学、物理、数学、电子计算机科学以及计算工具管理等后台。他曾在几个主要制造商的计算机上工作过许多年。他总是会在个人计算机中为每种UNIX打分。他是许多程序语言（包括awk）、浮点算术、软件可移植性、科学软件与计算机图形的专家，而且长期从事在早期UNIX时代的电子文件与排版。

封面介绍

我们的外观是取自于读者建议、我们自己的经验及经销商的回馈。与众不同的封面结合我们对技术主题特有的表现方式，将个性与生活融入较为艰涩的主题。

本书的动物封面是非洲的帐蓬陆龟(Psammobates tentorius),其龟甲呈现几何形状。Psammobates的含义是“热爱沙地”,其栖息地包括干燥的草原、沙漠边缘及沿海沙地,属于典型的干燥炎热气候。所以帐蓬陆龟只能在南非的草原与沙漠外围找到,这一点也不令人惊讶。这种类型的所有品种都是小型的,大小约从5英寸到10英寸,而且在它们的甲壳上有黄色放射型的标记。帐蓬陆龟特别突出的特征是具有拱起的龟甲。

陆龟以它们的长寿出名,而且乌龟与陆龟也是今日最古老的物种。它们在2亿年前的恐龙时代就已存在。所有的龟类都依赖温度,也就是它们只在温度不是太极端时才吃东西。在酷暑与寒冬期间,陆龟会冬眠且群体停止进食。在春天,陆龟的食物包括多汁植物、纤维植物及牧草。