China-pub.com
下载

第10学时　文件与目录

操作系统中的文件为数据提供了一个非常方便的存储方式。操作系统为数据提供了一个名字（即文件名）和一个组织结构，这样你就可以在以后找到你要的数据。这个组织结构称为文件系统。然后你的文件系统再将文件分成各个组，称为目录，有时也称为文件夹。这些目录能够存放文件或其他目录。

在目录中嵌套目录的方法给计算机中的文件系统提供了一个树状结构。每个文件都是一个目录的组成部分，每个目录又是父目录的组成部分。除为你的文件提供一个组织结构外，操作系统还存放了关于文件的各种数据，比如上次读取文件是在什么时候，上次修改文件是在什么时候，谁创建了文件，当前文件有多大等等。所有的现代计算机操作系统几乎都采用这种组织结构。

在Macintosh系统中，仍然采用这种结构，不过它的高层目录称为卷，子目录称为文件夹。

Perl允许你访问这个组织结构，修改它的组织方法，并可查看关于文件的各种信息。 Perl用于这些操作的函数全部源自unix操作系统，但是在Perl运行的任何操作系统下，这些函数都能够很好地运行。Perl的文件系统的操作函数是可以移植的，也就是说，如果你使用 Perl的函数对你的文件进行操作并查询你的文件，那么在 Perl支持的任何操作系统下，运行你的代码都是没有问题的，只要目录结构相类似。

在本学时中，你将要学习：

- 如何获得目录列表。
- 如何创建和删除文件。
- 如何创建和删除目录。
- 如何获取关于文件的信息。

10.1　获得目录列表

从系统中获取目录信息的第一步是创建一个目录句柄。目录句柄与文件句柄相类似，不同之处是：不是通过读取文件句柄来获得文件的内容，而是使用目录句柄来读取目录的内容。若要打开目录句柄，可以使用opendir函数：

```
opendir dirhandle, directory
```

在这个语句中，dirhandle是要打开的目录句柄，directory是要读取的目录的名字。要是目录句柄不能打开，你就无权读取该目录的内容，或者该目录根本不存在。 opendir函数将返回假。目录句柄的结构应该与文件句柄相类似，它使用第 2学时介绍的变量名的创建规则，目录句柄应该全部使用大写字母，以避免与 Perl的关键字发生冲突。下面是目录句柄的一个例子：

```
opendir(TEMPDIR, '/tmp') || die "Cannot open /tmp: $!"
```

本学时中介绍的所有例子都使用 UNIX样式中的正斜杠，因为与反斜杠相比，它不易产生混乱，并且它可以同时用于UNIX和Windows操作系统。

China-pub.com 下载

目录句柄打开后，可以使用readdir函数来读取它的内容：

```
readdir dirhandle;
```

在标量上下文中，readdir函数返回目录中的下一项，如果目录中没有剩下任何项目，则返回undef。在列表上下文中，readdir返回所有的（剩余的）目录项。readdir返回的名字包括文件、目录的名字，而对于UNIX来说，则返回特殊文件的名字。它们返回时没有特定的次序。readdir返回目录项.和..。readdir返回的目录项不包含作为目录名的组成部分的路径名。

当完成目录句柄的操作后，应该使用closedir函数将它关闭：

```
closedir dirhandle;
```

下面这个例子说明如何读取一个目录：

```
opendir(TEMP, '/tmp') || die "Cannot open /tmp: $!";
@FILES=readdir TEMP;
closedir(TEMP);
```

在上面这个代码段中，整个目录被读入 @FILES中。不过，在大多数时候，你对 .和..文件是不感兴趣的。若要读取文件句柄并清除这些文件，可以输入下面的代码：

```
@FILES=grep(!/^\.\.?$/, readdir TEMP);
```

正则表达式（/^\.\.?$/）用于匹配也位于行尾的一个前导原义圆点（或两个圆点），而grep则用于清除它们。若要获得带有特定扩展名的全部文件，可以使用下面的代码：

```
@FILES=grep(/\.txt$/i, readdir TEMP);
```

readdir返回的文件名并不包含opendir使用的路径名。因此，下面的例子可能无法运行：

```
opendir(TD, "/tmp") || die "Cannot open /tmp: $!";
while($file=readdir TD) {
    # The following is WRONG
    open(FILEH, $file) || die "Cannot open $file: $!\n";
    :
}
closedir(TD);
```

除非你在运行代码时恰好在 /tmp目录中工作，否则open(FILEH，$file)语句的运行将会失败。例如，如果/tmp中存在文件myfile.txt，那么readdir便返回myfile.txt。当你打开myfile.txt时，实际上必须使用全路径名来打开 /tmp/myfile.txt。正确的代码如下所示：

```
opendir(TD, "/tmp") || die "Cannot open /tmp: $!";
while($file=readdir TD) {
    # Right!
    open(FILEH, "/tmp/$file") || die "Cannot open $file: $!\n";
    :
}
closedir(TD);
```

Globbing

读取目录中的文件名时使用的另一种方法称为 globbing。如果你熟悉DOS中的命令提示符，那么一定知道命令dir*.txt可用于输出以.txt结尾的所有文件的目录列表。在 UNIX中，globbing是由shell来完成的，但是ls*.txt几乎能产生相同的结果，即列出以 .txt结尾的所有文件。

Perl有一个操作符，能够进行这项操作，它称为 glob。Glob的句法是：

```
giob pattern
```

这里的pattern是你要匹配的文件名模式。 pattern可以包含目录名和文件名的各个部分。此外，pattern可以包含表10-1列出的任何一个特殊字符。在列表上下文中， glob返回与模式匹配的所有文件（和目录）。在标量上下文中，每查询一次glob，便返回一个文件。

China-pub.com
下载

glob的模式与正则表达式的模式不同。

表10-1 globbing的模式

字 符	匹配的模式	举 例
?	单个字符	f？d用于匹配fud、fid和fdd等。
*	任何数目的字符	f*d用于匹配fd、fdd、food和filled等
[chars]	用于匹配任何一个chars; MacPerl不支持这个特性	f[ou]d用于匹配fod和fud，但不能匹配fad
{a，b，…}	既可以匹配字符串a， 也可以匹配字符串b， MacPerl不支持这个特性	f*{txt，doc}用于匹配以f开头并且以.txt或.doc结尾的文件

对于UNIX爱好者来说，Perl的glob操作符使用C语言的shell样式的文件globbing，而不是Bourne（或Korn）shell的文件globbing。它适用于安装了Perl的任何UNIX系统，而不管你个人使用的是何种 shell。Bourne shell globbing和Korn shell globbing不同于C语言的shell globbing。它们在某些方面很相似，比如 *与?的运行特性相同，但是在其他方面差别很大，请注意。

下面请看几个globbing的例子：

```
# All of the .h files in /usr/include
my @hfiles=glob('/usr/include/*.h');
# Text or document files that contain 1999
my @curfiles=glob('*1999*.{txt,doc}')
# Printing a numbered list of filenames
$count=1;
while($name=glob('*')) {
    print "$count. $name\n";
    $count++;
}
```

下面是使用glob与opendir/readdir/closedir 之间的某些差别：

- glob只能返回有限数量的文件。对于较大的目录， glob可能会报告“太多的文件”，但是却不能返回任何文件。这是因为 glob当前是使用外部程序即shell来实现的，它只能返回有限数量的文件。opendir/readdir/closedir函数则不存在这个问题。
- glob返回模式中使用的路径名，而 opendir/readdir/closedir函数则不能。例如， glob（‘/usr/include/*’）返回/usr/include/*，作为任何匹配项的组成部分，而 readdir则不能返回它。
- glob的运行速度通常比opendir/readdir/closedir慢。同样，它之所以运行速度比较慢，是因为Perl必须启动一个外部程序来为它进行 globbing，而该程序将在对文件名排序后再返回这些文件名。

那么你究竟应该使用哪一个函数呢？这完全取决于你。不过，使用 opendir/readdir/closedir函数往往是个复杂得多的解决方案，在本书中的大多数程序例子中，我们都使用这个函数。

China-pub.com 下载

为了完整起见，Perl提供了另一种方法，用于编写模式glob。只需将模式放入尖括号运算符（<>）中，就可以使尖括号运算符像glob函数那样来运行：

```
@cfiles=<*.c>;  # All files ending in .c
```

用于globbing的尖括号运算符的句法比较老，并且可能引起混乱。在本书中，为了清楚起见，将继续使用glob函数。

10.2 练习：UNIX的grep

当你进一步阅读本书的内容时，一些练习将为你展示更多的非常有用的工具。下面这个练习展示了一个UNIX的grep实用程序的简化版本。UNIX的grep（不要与Perl的grep相混淆）用于搜索文件中的模式。这个练习展示了一个实用程序，它提示你输入一个目录名和一个模式。目录中的每个文件均被搜索，以便寻找该模式，与该模式相匹配的文件行将被输出。

使用文本编辑器，键入程序清单10-1的程序，并将它保存为mygrep。务必按第1学时中介绍的方法使该程序成为可执行程序。另外，一定不要将该文件改名为UNIX系统上的grep，它可能被混淆为实际的grep实用程序。

完成上述操作后，键入下面的命令行，设法运行该程序：

```
perl mygrep
```

程序清单10-1 mygrep的完整清单

```
1:  #!/usr/bin/perl -w
2:
3:  use strict;
4:
5:  print "Directory to search: ";
6:  my $dir=<STDIN>; chomp $dir;
7:  print "Pattern to look for: ";
8:  my $pat=<STDIN>; chomp $pat;
9:
10:  my($file);
11:
12:  opendir(DH, $dir) || die "Cannot open $dir: $!";
13:  while ($file=readdir DH) {
14:      next if (-d "$dir/$file");
15:      if (! open(F, "$dir/$file") ) {
16:              warn "Cannot search $file: $!";
17:              next;
18:      }
19:      while(<F>) {
20:              if (/$pat/) {
21:                      print "$file: $_";
22:              }
23:      }
24:      close(F);
25:  }
26:  closedir(DH);
```

第1行：这一行包含到达解释程序的路径（可以修改它，使之适合系统的需要）和开关-w。请始终使警告特性处于激活状态。

第3行：use strict命令意味着所有变量都必须用my来声明，并且裸单词必须加上引号。

第5~8行：$dir（要搜索的目录）和$pat（要搜索的模式）是从STDIN中检索而来的。每

China-pub.com
下载

行结尾处的换行符均被删除。

第10行：$file被声明为符合use strict的专用变量。$file用在本程序的后面。

第12行：目录$dir被打开，如果这项操作没有成功，则输出一条出错消息。

第13行：从目录中检索各个条目，每次检索 1条，然后存入$file。

第14行：确实是目录本身（ -d）的任何目录条目均被拒绝。请注意，被核实的路径名是$dir/$file。核实该路径是必要的，因为当前目录中不一定存在 $file，它存在于$dir中。因此该文件的全路径名是$dir/$file。

第15~18行：文件被打开，再次使用全路径名 $dir/$file，如果文件没有打开，则被拒绝。

第19~23行：文件被搜索，逐行进行搜索，找出包含 $pat的这一行。匹配的行将被输出。

程序清单10-2显示了mygrep程序的输出举例。

程序清单10-2 mygrep的输出

```
1:   Directory to search: /home/clintp
2:   Pattern to look for: printer
3:   mailbox: lot of luck re-inking Epson printer ribbons with
4:   config.pl: # the following allows the user to pick a printer for
```

10.3 目录

到现在为此，本学时中一直在介绍目录结构的问题。为了打开文件，有时需要它的全路径名，readdir函数能够读取目录。但是，如果要浏览目录，添加和删除目录，或者清除目录，则需要更多的Perl功能。

10.3.1 浏览目录

当你运行软件时，操作系统将对你所在的目录保持跟踪。当你登录到一台 UNIX计算机中并且运行一个软件包时，通常要进入你的主目录。如果你键入操作系统的命令 pwd，shell便向你显示你是在什么目录中。如果你使用 DOS或Windows操作系统，并且打开一个命令提示符，该提示符就能反映出你当时是在什么目录中，比如 C:\WINDOWS。另外，你也可以在DOS提示符后面键入操作系统命令 cd，这样， DOS就会告诉你是在什么目录中。你当前使用的目录称为当前目录，即你的当前工作目录。

如果你使用程序编辑器或者集成式编辑器 /调试器，并且直接从那里运行你的Perl程序，那么“ 当前目录 ”可能不是你想像的那个目录。它可能是 Perl程序所在的目录，编辑器所在的目录或者是任意其他的目录，这取决于你使用何种编辑器。如果要确定当前目录究竟是什么，请在你的 Perl程序中使用cwd函数。

如果没有全路径名，也可以打开文件，比如： open（FH,“ file ”）||die可以在你的当前目录中打开。若要改变当前目录，可以使用下面这个 chdir函数：

```
chdir newdir;
```

chdir函数将当前工作目录改为 newdir。如果newdir目录不存在，或者你不拥有对 newdir的访问权，那么chdir返回假。chdir对目录的改变是暂时的，一旦 Perl程序运行结束，就返回运

China-pub.com 下载

行Perl程序之前所在的目录。

如果运行的chdir函数不包含一个目录作为其参数，那么 chdir就会将你的目录改为你的主目录。在UNIX系统上，主目录通常是你登录时进入的这个目录。在 Windows 95、Windows NT或DOS计算机上，chdir会使你进入HOME环境变量中指明的这个目录。如果 HOME没有设置，chdir将根本不改变当前目录。

Perl并不配有任何内置函数来确定当前目录是个什么目录，因为某些操作系统编写时所采用的方法，使得函数很难做到这一点。若要确定当前目录，必须同时使用两个语句。在程序的某个位置，最好是在靠近程序开始的地方，必须使用语句 use Cwd，然后，当你想要检查当前目录时，使用cwd函数：

```
use Cwd;
print "Your current directory is: ", cwd, "\n";
chdir '/tmp' or warn "Directory /tmp not accessible: $!";
print "You are now in: ", cwd, "\n";
```

只能执行use Cwd语句一次，此后，可以根据需要多次使用 cwd函数。

语句use Cwd实际上让Perl加载一个称为Cwd的模块，使Perl语言增加一些新的函数，如 Cwd。如果上面介绍的这个代码段返回一条出错消息，说 Can't locate cwd .pm in @INC（在@INC中找不到Cwd.pm），或者你不完全理解各个模块，那么现在不必为此而担心，第 14学时将要详细介绍模块方面的知识。

10.3.2 创建和删除目录

若要创建一个新目录，可以使用 Perl的mkdir函数，mkdir函数的句法如下：

```
mkdir newdir, permissions;
```

如果目录newdir能够创建，那么mkdir函数返回真。否则，它返回假，并且将 $ ！设置为mkdir运行失败的原因。只有在Perl的UNIX实现代码中，permissions才真的十分重要，不过在所有版本中都必须设置 permissions。对于下面这个例子，使用的值是 0755。这个值将在本学时后面部分中的“ UNIX系统 ”这一节中介绍。对于 DOS和Windows用户来说，只使用 0755这个值就足够了，可以省略冗长的说明。

```
print "Directory to create?";
my $newdir=<STDIN>;
chomp $newdir;
mkdir( $newdir, 0755 ) || die "Failed to create $newdir: $!";
```

若要删除目录，可以使用 rmdir函数。rmdir函数的句法如下：

```
rmdir pathname;
```

如果目录pathname可以删除，rmdir函数返回真。如果 Pathname无法删除，rmdir返回假，并将 $ ！设置为rmdir运行失败的原因，如下所示：

```
print "Directory to be removed?";
my $baddir=<STDIN>;
chomp $baddir;
rmdir($baddir) || die "Failed to remove $baddir: $!";
```

rmdir函数只删除完全是空的目录。这意味着在目录被删除之前，首先必须删除目录中的

China-pub.com
下载

所有文件和子目录。

10.3.3 删除文件

若要从目录中删除文件，可以使用unlink函数：

```
unlink list_of_files;
```

unlink函数能删除 list_of_files中的所有文件，并返回已经删除的文件数量。如果list_of_files被省略，$_中指定的文件将被删除。请看下例：

```
unlink <*.bat>;
$erased=unlink 'old.exe', 'a.out', 'personal.txt';
unlink @badfiles;
unlink;       # Removes the filename in $_
```

若要检查文件列表是否已被删除，必须对想要删除的文件数量与已删除的文件数量进行比较，如下面这个例子所示：

```
my @files=<*.txt>;
my $erased=unlink @files;

# Compare actual erased number, to original number
if ($erased != @files) {
    print "Files failed to erase: ",
          join(',', <*.txt>), "\n";
}
```

在上面这个代码段中，被 unlink删除的文件数量将存放在 $erased中。unlink运行后，$erased的值将与@files中的元素的数量进行比较，它们应该相同。如果不同，便输出一条出错消息，显示“剩余的”文件。

用unlink函数删除的文件将被绝对地清除掉，它们将无法恢复，并且不是被转入“回收站”，因此使用unlink函数时应该格外小心。

10.3.4 给文件改名

在Perl中给文件或目录改名是很简单的，可以使用 rename函数，如下所示：

```
rename oldname, newname;
```

rename函数取出名字为oldname的文件，将它的名字改为 newname。如果改名成功，该函数返回真。如果 oldname和newname是目录，那么这些目录将被改名。如果改名不成功，rename返回假，并将$！设置为不成功的原因，如下所示：

```
if (! rename "myfile.txt", "archive.txt") {
    warn "Could not rename myfile.txt: $!";
}
```

如果你设定了路径名，而不只是设定文件名，那么 rename函还会将文件从一个目录移到另一个目录，如下例所示：

如果文件newname已经存在，该文件将被撤销。

```
rename "myfile.txt", "/tmp/myfile.txt";   # Effectively it moves the file.
```

如果文件是在不同的文件系统上，那么 rename函将不会把文件从一个目录移到另一个目录中。

China-pub.com 下载

10.4 UNIX系统

下面介绍Perl用户在UNIX操作系统下工作的情况。如果你不是在 UNIX系统上使用Perl，那么可以跳过本节，你不会遗漏任何重要的内容。如果你对 UNIX非常有兴趣，可以阅读这一节的内容。

作为UNIX用户来说，应该知道Perl与UNIX系统之间有着很深的渊源关系，有些 Perl函数直接来自 UNIX命令和操作系统函数。这些函数中，大部分是你不使用的。有些函数，如unlink，虽然源于UNIX，但其含义却与UNIX毫无关系。每个操作系统都可以用来删除文件，Perl能够确保unlink会对操作系统执行正确的操作。 Perl作出了很大的努力，以确保有关的特性（如文件I/O）能够在操作系统之间移植，并且它在可能的情况下将所有的兼容性问题隐藏起来，使你不必为此而担心。

Perl语言中嵌入了许多UNIX函数和命令，而Perl语言已经移植到许多非UNIX操作系统中，这满足了UNIX开发人员和管理员的要求，使他们能够将 UNIX工具包中的一些工具带到任何地方去使用。

正如下一节的标题所示，这个描述不能被视为 UNIX文件系统访问许可权以及如何操作文件的完整说明。若要了解它的完整说明，请参见你的操作系统文档，或者参阅关于UNIX的其他著作，如《UNIX 24学时教程》。

文件访问许可权的简要介绍

在第1学时中，我们讲到，为了使Perl程序能够像一个标准命令那样来运行，我们提供了一个命令chmod 755 scriptname，但是没有具体介绍它的含义。755是赋予文件scriptname的访问许可权的一种描述。UNIX中的chmod命令用于设置文件的访问许可权。

这行数字分别代表赋予文件所有者、文件所属小组以及其他的非文件所有者和非文件所属小组的访问许可权。在上面这个例子中，文件所有者拥有的访问许可权是 7，文件所属小组和其他人的访问许可权是5。表10-2列出了每种访问许可权的值。

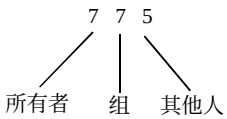

表10-2 文件访问许可权

访问许可权值	权 限
7	所有者／组／其他人可以读、写和执行该文件
6	所有者／组／其他人可以读、写该文件
5	所有者／组／其他人可以读和执行该文件
4	所有者／组／其他人可以读该文件
3	所有者／组／其他人可以写和执行该文件
2	所有者／组／其他人可以写该文件
1	所有者／组／其他人可以执行该文件

若要在Perl中设置文件的访问许可权，可以使用 UNIX的内置函数chmod：

```
chmod mode, list_of_files;
```

chmod函能够改变list_of_files的所有文件的访问许可权，并且返回已经改变访问许可权的文件数量。 mode的前面必须有一个数字 0（如果它是一个八进制直接量数字的话），然后是你

China-pub.com
下载

想指明其访问许可权的数字。下面是chmod命令的一些例子：

```
chmod 0755, 'file.pl';      # Grants RWX to owner, RX to group and non-owners
chmod 0644, 'mydata.txt';   # Grants RW to owner, and R to group and non-owners
chmod 0777, 'script.pl';    # Grants RWX to everyone (usually, not very smart)
chmod 0000, 'cia.dat';      # Nobody can do anything with this file
```

在本学时前面部分的内容中，我们介绍了 mkdir函数。mkdir的第一个参数是文件访问许可权，它与chmod使用的许可权是相同的：

```
mkdir "/usr/tmp", 0777;  # Publically readable/writable
mkdir "myfiles", 0700;   # A very private directory
```

UNIX文件的访问许可权常常称为它的“方式”。因此chmod是“change mode（改变方式）”的缩写。

10.5 你应该了解的关于文件的所有信息

如果你想要详细而全面地了解关于一个文件的信息，可以使用 Perl的stat函数。stat函数源于UNIX系统，它的返回值在UNIX系统中与非UNIX系统中略有不同。Stat的句法如下所示：

```
stat filehandle;
stat filename;
```

stat函数即可以用来检索已经打开的文件句柄的信息，也可以检索关于某个特定文件的信息。在任何操作系统下，stat均可返回一个包含13个元素的列表，来描述文件的属性。列表中的实际值随着运行的操作系统的不同而有所差异，因为有些操作系统包含的特性是其他操作系统所没有的。表10-3显示了stat返回值中的每个元素的含义。

表10-3 stat函数的返回值

编 号	名 字	UNIX系统	Windows系统
0	dev	设备号	驱动器号（C：通常是2，D：通常是3，等等）
1	ino	索引节号	总是0
2	mode	文件的方式	无
3	nlink	链接号	通常为0；Windows NT；文件系统允许链接
4	uid	文件所有者的用户ID（UID）	总是0
5	gid	文件所有者的组ID （GID）	总是0
6	rdev	特殊文件信息	驱动器号（重复）
7	size	文件大小（以字节计）	文件大小（以字节计）
8	atime	上次访问的时间	上次访问的时间
9	mtime	上次修改的时间	上次修改的时间
10	ctime	Inode修改时间	文件的创建时间
11	blksz	磁盘块的大小	总是0
12	blocks	文件中的块的数量	总是0

表10-3中的许多值你可能永远不会使用，但是为了完整起见，我们在表中将它们列出来。对于含义比较含糊的值，尤其是UNIX中的返回值，你可以查看操作系统的参考手册，以了解它的含义。

下面是将stat用于文件的一个例子：

China-pub.com 下载

```
@stuff=stat "myfile";
```

通常情况下，为了清楚起见， stat的返回值被拷贝到标量的一个赋值列表：

```
($dev, $ino, $mode, $nlink, $uid, $gid, $rdev, $size,
$atime, $mtime, $ctime, $blksize, $blocks)=stat("myfile");
```

若要按“文件访问许可权的简要介绍”这一节中所说的3字符形式输出文件的访问许可权，可以使用下面这个代码，其中的@stuff包含了访问许可权：

```
printf "%04o\n", $mode&0777;
```

上面这个代码所包含的元素你可能不了解，这没有什么关系。其中有些元素尚未向你介绍。在$ mode中由stat检索的元素包含了许多“额外的”信息。 &0777只是提取你感兴趣的这部分信息。最后，% 0是一个printf格式，用于输出采用0~7格式的八进制数字， UNIX希望用这种格式对文件访问许可权进行格式化。

八进制是以8为基数的数字表示法。由于历史原因，它用于 UNIX系统，不过它也用于 Perl。如果你仍然对它不太理解，请不必担心。如果需要显示文件的访问许可权，只要使用前面介绍的 printf函数即可。它并不是经常出现。

表10-3中列出的3个时间戳，即访问时间、修改时间和更改（即创建）时间均以特定格式来存储。时间戳以格林威治时间 1970年1月1日零点起的秒数来存储。若要以便于使用的格式来输出时间，可以使用localtime函数，如下所示：

```
print scalar localtime($mtime);
```

该函数用于输出文件的修改时间，格式为 Sat Jul 3 23:35:11 EDT 1999。访问时间是指上次读取文件（或打开文件以便读取）的时间，修改时间是指上次将数据写入文件的时间。在UNIX系统下，“更改”时间是指更改关于文件的信息（文件的所有者、链接的数量、访问许可权等）的时间，它并不是文件的创建时间，不过由于巧合，它常常就是文件的创建时间。在Microsoft Windows下，ctime域实际上用于存放文件的创建时间。

有时你可能想从stat返回的列表中仅仅检索1个值。若是这样，可以用括号将整个 stat函数括起来，并使用下标将你想要的值标出来：

```
print "The file has", (stat("file"))[7], " bytes of data";
```

10.6 练习：对整个文件改名

这个练习为你的工具包提供了另一个小型工具。假定有一个目录名，一个要查找的模式和要改变成的模式，使用这个实用程序，可以对目录名中的所有文件进行改名。例如，如果一个目录中包含文件名Chapter_01.rtf、Chapter_02.rtf、Chapter_04.rtf等，你就可以将所有文件改名为Hour_01.rtf、Hour_02.rtf、Hour_04.rtf等。当使用基于图形用户界面的文件浏览器时，想要用命令提示符来执行这种操作通常并不容易，而且显得很笨。

使用文本编辑器，键入程序清单 10-3中的程序，并将它保存为 Renamer。务必按照第1学时介绍的方法使该程序成为可执行程序。

完成上述操作后，键入下面的命令行，设法运行该程序：

```
perl Renamer
```

China-pub.com
下载

程序清单10-3 Renamer程序的完整清单

```
1:  #!/usr/bin/perl -w
2:
3:  use strict;
4:
5:  my($dir, $oldpat, $newpat);
6:  print "Directory: ";
7:  chomp($dir=<STDIN>);
8:  print "Old pattern: ";
9:  chomp($oldpat=<STDIN>);
10: print "New pattern: ";
11: chomp($newpat=<STDIN>);
12:
13: opendir(DH, $dir) || die "Cannot open $dir: $!";
14: my @files=readdir DH;
15: close(DH);
16: my $oldname;
17: foreach(@files) {
18:     $oldname=$_;
19:     s/$oldpat/$newpat/;
20:     next if (-e "$dir/$_");
21:     if (! rename "$dir/$oldname", "$dir/$_") {
22:             warn "Could not rename $oldname to $_: $!
23:     } else {
24:             print "File $oldname renamed to $_\n";
25:     }
26: }
```

第13~15行：$dir指明的目录中的条目被读入@files。

第17~19行：来自@files的每个文件均被赋给$_，并且该名字被保存在$ oldname中。然后，$_中的原始文件名被改为19行上的新名字。

第20行：在对文件改名之前，这一行要确认目标文件名并不存在。否则，该程序可能将文件改名为一个现有文件名，撤消其原始数据。

第21~25行。该文件被改名，如果改名失败，则输出一条警告消息。请注意，原始目录名必须附加到文件名上，例如，$/dir/$oldname，因为@files不包含全路径名，所以必须使用全路径名来进行改名。

程序清单10-4显示了该程序的示例输出。

程序清单10-4 Rename程序的输出示例

```
1:  Directory: /tmp
2:  Old Pattern: Chapter
3:  New Pattern: Hour
4:  File Chapter_02.rtf renamed to Hour_02.rtf
5:  File Chapter_10.rtf renamed to Hour_10.rtf
```

10.7 课时小结

本学时我们介绍了如何使用Perl中的mkdir、rm和rename函数来创建、删除目录条目并对其改名。另外，还介绍了如何使用stat来查询文件系统，以便了解关于文件的信息，不光是了解文件的内容。在本学时中，两个练习提供了一些简单而实用的工具，使你的程序具备更高的效能。

China-pub.com 下载

10.8 课外作业

10.8.1 专家答疑

问题：我在运行下面这个程序时遇到了问题。虽然目录中有文件，但是无法读取，原因何在？

```
opendir(DIRHANDLE, "/mydir") || die;
@files=<DIRHANDLE> ;
closedir(DIRHANDLE);
```

解答：问题出在第2行上。DIRHANDLE是个目录句柄，不是文件句柄。你无法用尖括号（<>）操作符来读取目录句柄。读取目录的正确方法是 @files=readdir DIRHANDLE。

问题：为什么glob(“ *.* ”)不能匹配目录中的所有文件？

解答：因为“ *.* ”只能匹配文件名中有圆点的文件名。若要匹配目录中的所有文件，请使用glob(*.*)。glob函数的模式可以在许多不同操作系统之间进行移植，因此它的运行特性与DOS中的*.*不同。

问题：我修改了mygrep这个练习，以便使用 opendir和更多的循环来搜索子目录，但是它似乎存在某些错误。为什么？

解答：总之，你不要这样做。向下搜索目录树是个老问题，它并不很容易，以前曾经多次解决过这个问题，而你自己没有必要这样去做。（从事全部这项工作称为“ 重新发明车轮 ”。）如果你只是因为好玩而这样做，这很好，但是不要在这上面耗费太多的时间。请等到第 15学时，你将会了解到如何使用 File∷Find这个方法。它用起来更加简单，但是更加重要，它能够进行程序调试。

问题：如果我将*.bat改为*.tmp，程序清单10-3中的程序就会出错，为什么？

解答：该程序并不希望你键入 *.bat作为要搜索的模式。在正则表达式中使用 *.bat是无效的，*必须放在另外某个字符的后面。如果你输入了 *\.bat，该程序完全可以接受这个输入，不过它不会像你期望的那样运行，因为文件名中从来没有原义字符 *。

为了纠正这个错误，你可以为该程序提供它期望的输入（简单字符串），也可以将程序清单的第19行改为s/\Q$oldpat/$newpat/，这样，正则表达式模式中的“ 特殊字符 ”将不起作用。

10.8.2 思考题

1) 若要输出文件foofile的上次修改时间，应该使用：
   a. print glob(“ foofile ”)；
   b. print (stat(“ foofile ”))[9];
   c. print scalar localtime (stat (“ foofile ”))[9]；
2) unlink函数返回的是:
   a. 实际删除的文件数量。
   b. 真或假，根据函数运行是否成功的情况而定。
   c. 试图删除的文件数量。

10.8.3 解答

1) 答案是b或c。如果选择b，输出的时间为1970年以来的秒数，没有什么用处。如果

China-pub.com
下载

选择c，则输出格式很好的时间。

2) 答案是a。不过，选择c，在某种情况下也是可以的。如果没有文件可以删除， unlink返回0，表示假。

10.8.4 实习

- 设法编写一个程序，列出目录中的所有文件、它的子目录中的所有文件等。这只作为一个编程练习。