31.2.5 Mixing Reads and Writes

It is possible to specify both read and write access. All you do is add a "+" symbol in front of the redirection. But as in the shell, using a less-than on a file never creates a new file; it only opens an existing one. On the other hand, using a greater-than always clobbers (truncates to zero length) an existing file, or creates a brand-new one if there isn't an old one. Adding a "+" for read-write doesn't affect whether it only works on existing files or always clobbers existing ones.

```
open(WTMP, "+< /usr/adm/wtmp")
    || die "can't open /usr/adm/wtmp: $!";

open(SCREEN, "+> lkscreen")
    || die "can't open lkscreen: $!";

open(LOGFILE, "+>> /var/log/applog"
    || die "can't open /var/log/applog: $!";
```

The first one won't create a new file, and the second one will always clobber an old one. The third one will create a new file if necessary and not clobber an old one, and it will allow you to read at any point in the file, but all writes will always go to the end. In short, the first case is substantially more common than the second and third cases, which are almost always wrong. (If you know C, the plus in Perl's open is historically derived from the one in C's fopen(3S), which it ultimately calls.)

In fact, when it comes to updating a file, unless you're working on a binary file as in the WTMP case above, you probably don't want to use this approach for updating. Instead, Perl's **-i** flag comes to the rescue. The following command takes all the C, C++, or yacc source or header files and changes all their foo's to bar's, leaving the old version in the original filename with a ".orig" tacked on the end:

```
$ perl -i.orig -pe 's/\bfoo\b/bar/g' *.[Cchy]
```

This is a short cut for some renaming games that are really the best way to update textfiles. See the second question in *perlfaq5* for more details.

31.2.6 Filters

One of the most common uses for open is one you never even notice. When you process the ARGV filehandle using <ARGV>, Perl actually does an implicit open on each file in @ARGV. Thus a program called like this:

```
$ myprogram file1 file2 file3
```

Can have all its files opened and processed one at a time using a construct no more complex than:

```
while (<>) {
    # do something with $_
}
```

If @ARGV is empty when the loop first begins, Perl pretends you've opened up minus, that is, the standard input. In fact, $ARGV, the currently open file during <ARGV> processing, is even set to "-" in these circumstances.

You are welcome to pre-process your @ARGV before starting the loop to make sure it's to your liking. One reason to do this might be to remove command options beginning with a minus. While you can always roll the simple ones by hand, the Getopts modules are good for this:

```
use Getopt::Std;

# -v, -D, -o ARG, sets $opt_v, $opt_D, $opt_o
getopts("vDo:");
```