Where to go from here

Finally it's all over. The package is ported, you've installed the software, and it really *does* work. This time, we're done!

Well, we said that once before, before we started testing, and we were wrong. We're wrong here, too:

- In the course of the port, you may find a bug or a misfeature and fix it. If you do so, you have effectively created a new version of the package. You should send in information about these changes to the author. If this is a popular package, you might consider reporting the changes to the Usenet group that exists for the package.
- You no longer need the space on disk, so you can clean up the archive and write it to tape. It's a good idea to maintain enough documentation to be able to retrieve it again.
- Sometime, maybe very soon, somebody will come out with a fix for a bug that will probably bite you some time, or with a feature that could really be of use to you. Your experience with this port will help you to port the new version.

None of this is much work now, and it will save you grief later on. Let's look at it in a little more detail.

Reporting modifications

Once you have the software running, you should report any changes to the author or maintainer of the software. In order for this to be of any use, you need to supply the following information:

- A description of the problems you ran into. Don't spare details here: remember the pain you went to to figure out what was going wrong, and you had an interest in solving the problem. If you're the first person to run into the problem, it probably hasn't hurt anybody else, least of all the author. He probably gets lots of mail saying "*xfoo* is broke", and he may not believe what you have to say until you prove it to him.
- How you fixed them. Again, lots of detail. The author probably understands the package better than you do. If you explain the problem properly, he may come up with a better

fix.

- The fixes themselves. *diff*s, lists of differences between the previous version and your versions, are the method of choice. We'll look at them in the rest of this section.

diff

diff is a program that compares two related source files and outputs information about how to create the second file from the first. You typically use it after making modifications to a file in order to describe how the modified file differs from the original. The resultant output file is also called a *diff*. We saw the application of *diff*s in Chapter 3, *Care and feeding of source trees*, page 29. Here we'll look at how to make them.

It's useful to recognize and understand diff formats, since you occasionally have to apply them manually. *diff* compares two source files and attempts to output a reasonably succinct list of the differences between them. In *diff* terminology, the output is grouped into *hunks*, information about a relatively local groups of differences.

Like most useful programs, *diff* has grown in the course of time, and modern versions can output in a bewildering number of formats. Fortunately, almost all diffs nowadays use the *context* format. We'll look at some others anyway so that you can recognize them.

In the following examples, we compare the files *eden.1*:

```
        A doctor, an architect, and a computer scientist
were arguing about whose profession was the oldest.  In the
course of their arguments, they got all the way back to the
Garden of Eden, whereupon the doctor said, "The medical
profession is clearly the oldest, because Eve was made from
Adam's rib, as the story goes, and that was a simply
incredible surgical feat."
        The architect did not agree.  He said, "But if you
look at the Garden itself, in the beginning there was chaos
and void, and out of that, the Garden and the world were
created.  So God must have been an architect."
        The computer scientist, who had listened to all of
this said, "Yes, but where do you think the chaos came
from?"
```

and *eden.2*:

```
        A doctor, an architect, and a computer scientist
were arguing about whose profession was the oldest.  In the
course of their arguments, they came to discuss the Garden
of Eden, whereupon the doctor said, "The medical profession
is clearly the oldest, because Eve was made from Adam's rib,
as the story goes, and that was a simply incredible surgical
feat."
        The architect did not agree.  He said, "But if you
look at the Garden itself, in the beginning there was chaos
and void, and out of that, the Garden and the world were
created.  So God must have been an architect."
        The computer scientist, who had listened to all of
```

```
this, said, "Yes, but where do you think the chaos came
from?"
```

normal format diffs

As the name implies, the *normal* format is the default. You don't need to specify any format flags:

```
$ diff eden.1 eden.2
3,7c3,7
< course of their arguments, they got all the way back to the
< Garden of Eden, whereupon the doctor said, "The medical
< profession is clearly the oldest, because Eve was made from
< Adam's rib, as the story goes, and that was a simply
< incredible surgical feat."
---
> course of their arguments, they came to discuss the Garden
> of Eden, whereupon the doctor said, "The medical profession
> is clearly the oldest, because Eve was made from Adam's rib,
> as the story goes, and that was a simply incredible surgical
> feat."
13c13
< this said, "Yes, but where do you think the chaos came
---
> this, said, "Yes, but where do you think the chaos came
```

The first line of each hunk specifies the line range: `3,7c3,7` means "lines 3 to 7 of the first file, lines 3 to 7 of the second file". `13c13` means "line 13 of the first file, line 13 of the second file, has changed (c)". Instead of `c` you will also see `d` (lines deleted) and `a` (lines added). After this header line come the lines of the first file, with a leading `<` character, then a divider (`---`) and the lines of the second file with a leading `>` character. This example has two hunks.

ed format diffs

ed format diffs have the dubious advantage that the program *ed* can process them. You can create them with the `-e` flag. In this example, we also use shell syntax to shorten the input line. Writing `eden.[12]` is completely equivalent to writing `eden.1 eden.2`.

```
$ diff -e eden.[12]
13c
this, said, "Yes, but where do you think the chaos came
.
3,7c
course of their arguments, they came to discuss the Garden
of Eden, whereupon the doctor said, "The medical profession
is clearly the oldest, because Eve was made from Adam's rib,
as the story goes, and that was a simply incredible surgical
feat."
.
```

Just about everybody who has *diff* also has *patch*, and nowadays not everybody has *ed*. In addition, this format is extremely dangerous, since there is no information about the old

content of the file: you can't be sure if the patch will be applied in the right place. As a result, you almost never see this form.

context diffs

You select a context diff with the flag `-c`:

```
$ diff -c eden.[12]
*** eden.1      Tue May 10 14:21:47 1994
--- eden.2      Tue May 10 14:22:38 1994
***************
*** 1,14 ****
        A doctor, an architect, and a computer scientist
  were arguing about whose profession was the oldest.  In the
! course of their arguments, they got all the way back to the
! Garden of Eden, whereupon the doctor said, "The medical
! profession is clearly the oldest, because Eve was made from
! Adam's rib, as the story goes, and that was a simply
! incredible surgical feat."
        The architect did not agree.  He said, "But if you
  look at the Garden itself, in the beginning there was chaos
  and void, and out of that, the Garden and the world were
  created.  So God must have been an architect."
        The computer scientist, who had listened to all of
! this said, "Yes, but where do you think the chaos came
  from?"
--- 1,14 ----
        A doctor, an architect, and a computer scientist
  were arguing about whose profession was the oldest.  In the
! course of their arguments, they came to discuss the Garden
! of Eden, whereupon the doctor said, "The medical profession
! is clearly the oldest, because Eve was made from Adam's rib,
! as the story goes, and that was a simply incredible surgical
! feat."
        The architect did not agree.  He said, "But if you
  look at the Garden itself, in the beginning there was chaos
  and void, and out of that, the Garden and the world were
  created.  So God must have been an architect."
        The computer scientist, who had listened to all of
! this, said, "Yes, but where do you think the chaos came
```

The output here gives us significantly more information: the first two line gives the name and modification timestamp of the files. Then the hunks start, with a row of * as a leader. The next line is line number information for the first file (lines 1 to 14), after which come the lines themselves, surrounded by a number of lines of *context*, unchanged information. You can specify the number of lines of context, but by default *diff* includes 2 lines either side of the changes. The lines that have been modified are flagged with an exclamation mark (!) at the beginning of the line. In this case, the file is so small that the two modifications have been merged into one large one, and the whole file gets repeated, but in a larger file *diff* would include only the information immediately surrounding the changes. This format is more reliable than normal diffs: if the original source file has changed since the diff, the context

information helps establish the correct location to apply the patch.

unified context diffs

unified diffs are similar to normal context diffs. They are created with the `-u` flag:

```
$ diff -u eden.[12]
--- eden.1      Tue May 10 14:21:47 1994
+++ eden.2      Tue May 10 14:22:38 1994
@@ -1,14 +1,14 @@
        A doctor, an architect, and a computer scientist
 were arguing about whose profession was the oldest.  In the
-course of their arguments, they got all the way back to the
-Garden of Eden, whereupon the doctor said, "The medical
-profession is clearly the oldest, because Eve was made from
-Adam's rib, as the story goes, and that was a simply
-incredible surgical feat."
+course of their arguments, they came to discuss the Garden
+of Eden, whereupon the doctor said, "The medical profession
+is clearly the oldest, because Eve was made from Adam's rib,
+as the story goes, and that was a simply incredible surgical
+feat."
        The architect did not agree.  He said, "But if you
 look at the Garden itself, in the beginning there was chaos
 and void, and out of that, the Garden and the world were
 created.  So God must have been an architect."
        The computer scientist, who had listened to all of
-this said, "Yes, but where do you think the chaos came
+this, said, "Yes, but where do you think the chaos came
 from?"
```

As with context diffs, there is a header with information about the two files, followed by a hunk header specifying the line number range in each of the two files. Unlike a normal context diff, the following hunk contains the old text mingled with the new text. The lines prefixed with the character `-` belong to the first file, those prefixed with `+` belong to the second file—in other words, to convert the old file to the new file you remove the lines prefixed with `-` and insert the lines prefixed with `+`.

There are still other formats offered by various flavours of *diff*, but these are the only important ones.

What kind of diff?

As we've seen, *ed* style diffs are out of the question. You still have the choice between regular diffs, context diffs and unified context diffs. It's not that important which kind of diff you choose, but context diffs are easier to apply manually. Unified context diffs take up less space than regular context diffs, but there are still versions of *patch* out there that don't understand unified diffs. Until that changes, it's probably best to settle for regular context diffs. You may have noticed that all the examples in Chapter 3, *Care and feeding of source trees*, were regular context diffs.

Living with diff

Diff is a straightforward enough program, but you might run into a couple of problems:

- After a large port, it makes sense to make diffs of the whole directory hierarchy. This requires that you have copies of all the original files. You can use *rcsdiff*, part of the RCS package, but it only does diffs one at a time. I find it easier to maintain a copy of the complete original source tree, and then run *diff* with the option `-r` (descend recursively into directories):

  ```
  $ diff -ru /S/SCO/Base/gcc-2.6.3 /S/Base/Core/gcc-2.6.3 >SCO.diffs
  ```

  This command will create a single file with all the diffs and a list of files which only exist in the first directory. This can be important if you have added files, but it also means that you should do a *make clean* before running *diff*, or you will have entries of this kind for all the object files you create.

- Another problem that may occur is that one of the files does not have a newline character at the end of the last line. This does not normally worry compilers, but *diff* sees fit to complain. This is particularly insidious, because *patch* doesn't like the message, and it causes *patch* to fail.

Saving the archive

Most of us have had the message *Don't forget to make backups* drummed into us since we were in elementary school, but nowhere does it make more sense than at the end of a port. Don't forget where you put it! After archiving your port of *xfoo*, you may not look at it again for three years. When the new version comes out, you try to port it, but all sorts of things go wrong. Now is the time to get out the old version and read your notes—but where is it?

It's beyond the scope of this book to go into backup strategies, but you should do some thinking about the subject. One good idea is to keep separate (DAT or Exabyte) tapes of old ports, and just add additional archives at the end. That way you don't have to worry about overwriting them accidentally: the tapes are small and cheap enough that you can afford to keep their contents almost indefinitely. If you don't choose this method (maybe because the media don't fit into your QIC-150 tape drive), you need to think carefully about how to track the archives and when they are no longer needed.

Not done after all?

Of course, it may be that this optimistic finish is completely out of place. After what seems like months of frustration, you finally decide that you are never going to get this &%%$@# to work, and you give up. You can never rule out this possibility—as I said in Chapter 1, *Introduction*, I hope this book made it easier, but it's not a magic scroll.

Even if you *do* give up, you have some tidying up to do: you obviously can't send the author your bug fixes, but you can at least report the bugs. What he does with them depends on his interest and his contractual obligations, but even with free software, which is free of obligations of this nature, the author may be interested enough to fix the problem. One way or

another, you should go to the trouble to report problems you experience, even if you can't fix them and there is no support obligation.

A final word: if you give up on a port after getting this far, this book has failed for you. I don't want that to happen. Please contact me, too (`grog@lemis.de`, or via O'Reilly and Associates) and explain the problem. Like the authors of the software, I don't guarantee to do anything about it, but I might, and your experience might help to make the next edition of this book more useful.