Merging and More with diff3

Jerry Peek

As you've seen over the course of the last two "Power Tools" columns, one reason to use the command-line GNU *diffutils* is that, compared to windowed difference programs, *diffutils* can give you much more control over how variations are found and displayed. Another compelling argument is that some *diffutils* can create new files from merged versions of other files — either interactively, or by using the output of one utility as input to another. You can also use `diff --from-file` to compare one file to two or more other files. *patch* can also merge files — automatically.

The *diff3* utility compares *three* files, and can also merge the changes from two other files into a common ancestor file. Let's dig in and see how it works.

What's different about diff3?

Most *diffutils* compare two files, but *diff3* compares three. It's often used to compare two differing files that are both based on a common ancestor. For instance, if two programmers copy and then modify the same program file, *diff3* could show both sets of changes.

diff3 can also merge changes from two files into a third file — typically in a case like the one just mentioned. The options `-m` and `--merge` do this.

The `-e` and `--ed` options create an *ed* script that lets you merge changes into a separate file. *ed* scripts were common before *patch* came on the scene, and remain effective. However, because *ed* scripts generally use absolute line numbers, such scripts can cause corruption or fail completely when used to edit files that are somewhat different than the file the script was based upon. *patch* is far more forgiving.

Showing differences with diff3

The *diff3* output format is different than the formats of *diff*. Let's start with the short description (edited slightly) from the *Unix Seventh Edition man* page, and then look at a few examples.

```
diff3 [ -ex3 ] file1 file2 file3

diff3 compares three versions of a file,
and publishes disagreeing ranges of text
flagged with these codes:

====            all three files differ

====1           file1 is different

====2           file2 is different

====3           file3 is different

The type of change needed to convert a
given range of one file  to another is
indicated in one of two ways:

f : n1 a
Text is to be appended after line number
n1 in file f, where f is 1, 2, or 3.

f : n1 , n2 c
Text is to be changed in the range line n1
to line n2.  If there's just one line,
only n1 is given.
```

Now for examples. Let's look at three sets of three files. In each set, one line in one of the three files is different.

In the first example, shown in *Table One*, the first line of the first file (file *1a*) is different.

If you pass those three filename arguments to *diff3* (using the shell wildcard `1?` to expand the three names sorted alphanumerically), the output is:

```
$ diff3 1?
====1
1:1c
   line 1 file a
2:1c
3:1c
   line 1
```

The first output line, starting with equal signs and a `1`, says, "Here's a difference in the first file." Next, is the difference. In the first file (`1:`), line 1 contains (`1c`) the text "line 1 file a". In both the second and third files (`2:` and `3:`), line 1 contains (`1c`) the text "line 1". Because the text is the same in both the second and third files, the codes for both of those

TABLE ONE: The first file of three differs

FILE *file1a*	**FILE** *file1b*	**FILE** *file1c*
line 1	file a	line 1 line 1
line 2	line 2	line 2

files (`2:1c` and `3:1c`) are shown together, and the line is shown only once afterward.

For the next example (*Table Two*), the first line of the second file, file *2b*, is different:

```
$ diff3 2?
====2
1:1c
3:1c
    line 1
2:1c
    line 1 file b
```

Again, the `2` after the hunk delimiter `====2` means, "This change is in the second file." You can see that both the first file (`1:1c`) and third file (`3:1c`) have a first line that reads "line 1". In the second file, though (`2:1c`), line 1 reads "line 1 file b".

You're probably seeing the pattern now. But, to be complete, *Table Three* shows a change in the third file *3c*:

```
$ diff3 3?
====3
1:1c
2:1c
    line 1
3:1c
    line 1 file c
```

The other type of change that *diff3* indicates is an appended line. *Table Four* shows three example files; the second file has a new line at the end. Here's the *diff3* output:

```
$ diff3 4?
====2
1:2a
3:2a
2:3c
    line 3 file b
```

It shows that in file1 (`1:2a`) and file 3 (`3:2a`), the hunk contains no lines. (In other words, the new text isn't inserted into these two files.) But in file 2, the new line 3 is changed (`2:3c`) to read "line 3 file b".

If you'd like more examples, see the online manual page at http://www.gnu.org/software/diffutils/manual/. (It's worth reading, anyway.)

Merging with diff3

One advantage of plain *diff* over *diff3* is that *diff*'s output can be used as input to *patch* — to merge the same changes into other files, at other times. *diff3* output, however, is for you to look at.

TABLE TWO: The second file differs from the other two

FILE *file2a*	**FILE** *file2b*	**FILE** *file2c*
line 1	file 1	line b line 1
line 2	line 2	line 2

TABLE THREE: An example where the third file differs

FILE *file3a*	**FILE** *file3b*	**FILE** *file3c*
line 1	file 1	line 1 lfile c
line 2	line 2	line 2

TABLE FOUR: A third line in second file

FILE *file4a*	**FILE** *file4b*	**FILE** *file4c*
line 1	line 1	line 1
line 2	line 2	line 2
	line 3	file b

When you're comparing two different files to a third, as *diff3* does, changes can conflict. For instance, line 2 may be different in all three files. In those cases, you have to decide which change to keep. If there are no conflicts, though, *diff3* can merge two files automatically, and writes the result to its standard output. (To merge more than three files, merge the first three, then re-merge the next into the result from the first three, and so on.)

When merging files, the order of the three command-line arguments is important. The *diffutils* manual gives an example: Both you and someone else have modified an ancestor file named *older*. Your version is named *mine* and the other person's is named *yours*. You want to merge into *mine* the changes that would turn *older* into *yours*. In other words, the other person's changes should be found, then those changes are merged with your changes. The command-line arguments should be in this order:

```
$ diff3 mine older yours
```

Let's see examples of both a clean merge and a merge with a conflict, reusing the files from *Table Three*.

The *yours* file (the third argument, file *3c*) has been modified, but the *mine* file hasn't. It's easy to merge those changes because only one of the two "newer" files has changed:

```
$ diff3 -m 3?
line 1 file c
line 2
```

In the same way, in the situation shown in *Table One*, only *mine* has been edited, so merging is easy:

```
$ diff3 -m 1?
line 1 file a
line 2
```

The problem comes when both *mine* and *yours* are different than the original version. When *older* is different from both newer files, *diff3* flags that, by default. The conflict is shown between a pair of `<<<<<<<` and `>>>>>>>` markers; other lines without conflicts are emitted as-is. Here's the conflict in this case:

```
$ diff3 -m 2?
<<<<<<< 2b
line 1 file b
=======
line 1
>>>>>>> 2c
line 2
```

That "conflict" may not be a problem for you, though. Here, both you and the other person made the same change to the *older* file (changing "line 1 file b" to "line 1"). To handle cases like these, *diff3* gives you other options to control which output is shown:

- `-x` or `--overlap-only` outputs only the overlapping changes (conflicts where all three files are different)
- `-3` or `--easy-only` outputs only non–overlapping changes
- `-A` or `--show-all` outputs all changes.

See the documentation for more information and examples.

Merging with patch

patch is such a popular and easy-to-use tool that it doesn't need much introduction here. Briefly, though, it's an automated way to apply *diff* output to other files. You create a *diff* listing that shows the changes between the current version of one or more files and the new version(s). You then send that *diff* listing to others. In turn, they feed the listing to *patch*, and it makes those same changes in the remote files.

patch does merging automatically and fairly intelligently. Here are some good tips from the *diffutils* manual:

- *patch* is smart about ignoring extraneous text around the *diff* listing: mail headers and footers, indentation, and so on. If you receive a *diff* listing via email, you can often just feed the mail message to *patch* without cleaning it up. (If you're using a graphical email program, save the email message to a file, then *cd* from a shell window to the right directory and run `patch <path-to-message-file>`.)
- The `--dry-run` option shows diagnostics without modifying files. If you aren't sure what may happen to your files, try a dry run before patching "for real."
- In traditional versions of *patch*, backup files were enabled by default. With the POSIX and GNU versions, though, backups are made only in certain circumstances. Either use a revision-control system (like *RCS* or *Subversion*) to keep old versions of the file, or use the `-b` option to force backups.
- When *patch* does make a backup file, by default, that name is the original filename with the suffix `.orig`. But GNU *patch* also follows the GNU backup system — the environment variable `GNU SIMPLE_BACKUP_SUFFIX`, for instance. (There's more information in the August 2005 "Power Tools" column *What's GNU, Part Three*, online at http://www.linux-mag.com/id/2163.)

Running the *bash* command `export SIMPLE_BACKUP_SUFFIX=".old"` sets that suffix for all GNU programs that make backups. To set the suffix only for *patch*, you can make an alias that defines the environment variable per-command. (This *bash* feature works for any Linux utility, not just *patch*.) For example, if the *patch* binary is at */usr/bin/patch*, try…

```
$ alias patch='SIMPLE_BACKUP_SUFFIX=".old"
/usr/bin/patch'
$ patch myfile < patchfile
patching file myfile
$ ls myfile*
myfile myfile.old
```

- To make a *patch* file, don't use *diff3*. Try `diff -aNru` instead. See "Avoiding Common Mistakes" in the *diffutils* manual.

And Even More…

There's a lot more to discover about *diffutils* in these three columns. The detailed and very helpful documentation for *diffutils* is online at http://www.gnu.org/software/diffutils/manual/.

Jerry Peek is a freelance writer and instructor who has used Unix and Linux for 25 years. He's happy to hear from readers; see http://www.jpeek.com/contact.html.