sdiff and More

Jerry Peek

As was said in last month's column (the first of this series), there are good graphical file-comparison programs available, and many of those take advantage of color and of a scrollbar to make file comparisons easy to see. However, the GUI tools aren't always the best way to compare files.

The topic last month was *diff*, which shows differences "vertically" — showing first the lines from one file, then lines from the other file. This month, let's look into *sdiff*, which shows differences "horizontally" or side-by-side. It also lets you merge two files interactively. *sdiff* has more than its share of oddities, though — let's explore those in detail, too, so you'll be ready to use this handy tool.

(The GNU *diffutils* documentation says that the name *sdiff* is obsolete for non-interactive use, without `-o`. It recommends using `diff -y` or `diff --side-by-side` instead. For consistency, though, this column uses *sdiff* everywhere.)

Why Use sdiff?

There are a number of good reasons to use the command-line *sdiff* instead of graphical difference programs:

- You have extra control over the output. For instance, `sdiff -I` ignores changed lines that match one or more regular expressions you specify. It can also ignore changes in tabs, whitespace, carriage returns, and more. (*sdiff* understands many of the *diff* options covered in last month's column.)
- The default (non-interactive) output is 130 columns wide, designed for printing on line printers — those clattering machines that use stacks of folded paper with holes on the edges. However, it's also good for printing on sheet-fed printers. Choose landscape mode, and use a constant-width font that maintains the position of the center "gutter" between the files. (There are examples of the gutter later in this column.)

TABLE ONE: Two short files, *file1a* and *file1b*, left and right, respectively

FILE *file1a*	**FILE** *file1b*
line 1	line one
line 2	line 2
line 2a	line 3
line 3	line 3b

- Although the non-interactive *sdiff* is designed for printing, it's also quite useful in a terminal window. Stretch a window wide enough to view the output, or use the handy pager program *less* to pan side-to-side through the output.
- *sdiff* can also merge two files interactively, letting you choose the lines from each file that make up the merged output file.

Let's dig in!

Showing Differences On-Screen

By default (without its `-o` option), *sdiff* compares two files and shows the two in a side-by-side format 130-characters (columns) wide. Since typical terminal windows are 80-characters wide, you'll need to make some adjustments to view the output sensibly. We'll cover some of those in the next section.

Let's compare two short files, *file1a* and *file1b*, shown in *Table One*.

To save printing space, let's use the option `-w 30` to make the output lines 30-characters wide, which is enough to show the short lines from both sample files. Here's the *sdiff* output:

```
$ sdiff -w 30 file1a file1b
line 1          | line one
line 2            line 2
line 2a         <
line 3            line 3
                > line 3b
```

The default *sdiff* output shows the first 61 characters from each line of both files. The `-w 30` forces the output to be much narrower. If lines in the first file are too short, *sdiff* outputs enough tabs and spaces at the end of the first file's line to make the complete 61 columns.

The two files' contents are separated by a *gutter*. If the two lines differ, the gutter has a space, a character that denotes the type of difference, and a tab. If the two lines are the same, the gutter is just a tab.

- A `|` (vertical bar) means that the lines are different. In the example, the first line is different.
- A `<` (left angle bracket) denotes that the line is only in the first file (in the left column). Here, `line 2a` only occurs in the first file (*file1a*).

- A > (right angle bracket) marks a line that's only in the second file (right column). Here, that's `line 3b`.

Cutting Clutter

Seeing common lines repeated can clutter the output. The `-l` (lowercase "L") or `--left-column` option outputs only the left column of common lines. Let's try it with the same two files, and also use the shell wildcard `[ab]` to avoid typing both filenames separately:

```
$ sdiff -w 30 -l file1[ab]
line 1          | line one
line 2          (
line 2a         <
line 3          (
                > line 3b
```

The `(` (left parenthesis) character in the gutter indicates that `line 2` and `line 3` are in both files, but it's only shown in the left column.

If you don't need to see the common lines at all, use `-s` or `--suppress-common-lines`, as in:

```
$ sdiff -w 30 -s file1[ab]
line 1          | line one
line 2a         <
                > line 3b
```

Incomplete Lines

You may see the marker `\` or `/` in the gutter. The reason for it is worth understanding — especially if you use the *Emacs* editor, which can cause the problem if you aren't careful.

Each line in a Linux text file ends with a newline character. In Emacs, you can create a file whose last line doesn't end with a newline if you don't press Enter at the end of the last line. When plain *diff* compares a file like that, it warns "No newline at end of file". *sdiff* works differently.

If both files have a line with the same text, but one of those lines is incomplete (doesn't have a newline), *sdiff* puts the marker `\` or `/` in the gutter. However, if only one of the files has a line, and that line is also incomplete, *sdiff* uses the marker `<` or `>` — and also doesn't output a final newline. (This is the behavior of GNU *sdiff* version 2.8.1.)

Figure One shows an example you can try. It uses `echo -n` and the shell's "append" operator, `>>`, to create files without a final newline. The file *file2a* has two newline-terminated lines; *file2b* has two lines, the last incomplete; and *file2c* has three lines, the last incomplete.

Notice the `$` prompt at the end of the last line. It's there because *sdiff* output `line 3` from *file2c* as-is — without a newline — and the shell did what it always does: emits its prompt string after the program exits.

FIGURE ONE: Files with incomplete lines

```
$ echo" line 1 nl" & gt; file2a
$ cp file2a file2b
$ echo" line 2 nl" & gt;& gt; file2a
$ echo –n" line 2" & gt;& gt; file2b
$ diff file2[ ab]
2c2
& < line 2 nl
—
& > line 2
\ No newline at end of file
$ sdiff –w 30 file2[ ab]
line 1 nl line 1 nl
line 2 nl / line 2
$ sdiff –w 30 file2b file2a
line 1 nl line 1 nl
line 2 \ line 2 nl
$ cp file2a file2c
$ echo –n" line 3" & gt;& gt; file2c
$ diff file2[ ac]
2a3
& > line 3
\ No newline at end of file
$ sdiff –w 30 file2[ ac]
line 1 nl line 1 nl
line 2 nl line 2 nl
& > line 3$
```

Alignment Problems

If there's misalignment in the characters in the gutter, or at the start of each line in the second file, your terminal window may not be using a constant-width font. Be sure you're using a typewriter-like font, such as *Courier*, where, for instance, a period (`.`) requires the same horizontal space as an uppercase `W`.

sdiff tries to handle tab characters correctly on both the input file and the output lines to your terminal. If it doesn't seem to work, though, try the *sdiff* option `-t` or `--expand-tabs`. It expands tabs in the input files to spaces on output.

If problems persist, your terminal might not be handling tabs correctly. Try piping the output of *sdiff* to the *expand* utility, which converts tabs into the proper number of space characters to keep the default eight-character tabstop settings. (However, don't use *expand* with the `-o` option, which you'll see momentarily. Pipe buffering can keep you from seeing all of the text.) If the output of *expand* looks right, then your terminal's tabstop settings may be wrong. In that case, try using the *reset* utility or open another terminal window.

See Power, *pg. 52*

Power, *from pg. 17*

Line-Width Problems

If a line from one of the input files is wider than the space available in the *sdiff* output, the line is truncated, even if the difference is in that truncated line part!

One useful adjustment is to make the output *wider*: 160 columns or more. This lets you compare two files with 80-character lines and see all of each file. The option `-w nnn` sets the total output line width, so `-w 168` makes lines wide enough for two 80-character-wide files and the 8-character gutter. Of course, if your terminal isn't 168 characters wide (and you can't stretch it that far or use a smaller font), the output lines will either wrap or be truncated.

A handy way to show *sdiff* output that's wider than your terminal is by piping the *sdiff* output to `less -S`. (That's an uppercase "S".) This truncates each line at the screen width; you can scroll right and left across the full line width with the right-arrow and left-arrow keys. (This won't work with interactive *sdiff*, though.)

Merging Files With sdiff

To merge two files interactively, use `sdiff -o outfile` or `sdiff --output=outfile`, where `outfile` is the destination of the merged text. *sdiff* starts from the beginning of the files, outputting the first set of common lines to `outfile`. Then it shows the first hunk of differing lines and prompts with the `%` character. At the prompt, you can type one of these commands and press Enter:

- `l` copies the left version to `outfile`, while `r` copies the right version.
- `el` invokes an editor on the left version, then copies the edited output to `outfile`. `er` does the same for the right version.
- `eb` concatenates both versions into a temporary file, lets you edit the temporary file, then copies the result to `outfile`.
- `ed` is like `eb`, but it puts a header with filename and line numbers before each version. (Remove the header when you edit the temporary file.)
- `e` discards both versions and opens an editor on an empty file. The result is copied to `outfile`.
- `q` quits early.
- Enter gives a brief summary of the commands.

FIGURE TWO: Interactive file merging with `sdiff -o`

```
$ export EDITOR=ed
$ sdiff -o file1.merge -w 30 file1[ab]
line 1          | line one
%l
line 2            line 2
line 2a         <
%r
line 3            line 3
                > line 3b
%e
0
a
That's all.
.
w
12
q
$ cat file1.merge
line 1
line 2
line 3
That's all.
```

Figure Two shows an example editing session. The editor used is *ed*, the simple line-oriented editor, so you can see what's happening, but you'll probably prefer your usual editor (which may already be set as `EDITOR` by default).

When *sdiff* shows `line 2a`, which is only in the left file, press `r` to ignore the line (output nothing to *file1.merge*). When *sdiff* shows `line 3b`, press `e` to open an empty file with *ed*. In the editor, add the single line `That's all.` to the file, and save it.

Quick-and-Dirty Merging

Interactive merging with *sdiff* can be tedious. Here's a kludge that's often easier. Make a `diff -u` (unified *diff* format) listing of both files, with enough context to include all lines. Then edit the merged listing, removing the lines you don't want. Finally, remove the first character (`+`, `-`, and space) from the remaining lines.

To Be Continued…

Next time, we'll look into the little-known *diff3*, including using it to merge files automatically. We'll also cover hints about using *patch* to merge.

Jerry Peek is a longtime contributor to Linux Magazine, *a freelance writer, and an instructor who has used Unix and Linux for 25 years. He's happy to hear from readers; visit the online email form at* http://www.jpeek.com/contact.html.