The Stata Journal (2008)
8, Number 2, pp. 290–292

Stata tip 60: Making fast and easy changes to files with filefilter

Alan R. Riley
StataCorp
College Station, TX
ariley@stata.com

Stata has a command `filefilter` (see [D] **filefilter**) that makes it possible to perform global search-and-replace operations on a file, saving the result to another file. Think of it as a command which copies a file, but in the process of copying it, can search for one text pattern and replace it with another.

As its manual entry points out, `filefilter` has been designed to read and write the input and output files using buffers for speed, and is thus fast at converting even large files. `filefilter` can be used on files which are too large to open in a traditional text editor, and because Stata is programmable, it is possible to use `filefilter` to perform complicated global search-and-replace operations, which would not be possible in most text editors. `filefilter` can even make changes to binary files.

`filefilter` is often used to preprocess files (perhaps to remove invalid characters or to change a delimiter) before reading them into Stata and can be used in many other situations as a useful file-processing tool.

For example, if you have a log file named `x.log` in Windows (where the end-of-line (EOL) character combination is `\r\n`), and you want to convert the file to have Unix-style EOL characters (`\n`), you can type in Stata

```
. filefilter x.log y.log, from(\r\n) to(\n)
```

which will replace every occurrence of the Windows EOL character sequence with the Unix EOL character. Equivalently, you could type

```
. filefilter x.log y.log, from(\W) to(\U)
```

because `filefilter` understands `\W` as a synonym for the Windows EOL character sequence `\r\n` and `\U` as a synonym for the Unix EOL character sequence `\n`.

(For the rest of this tip, I will write `\W` as the EOL marker, but be sure to use the EOL shorthand in `filefilter` appropriate for your operating system.)

Let's put `filefilter` to use on another example. Imagine that we want to replace all occurrences of multiple blank lines in a file with a single blank line for readability. Changing a file in this way may be desirable after, for example, the command `cleanlog` (Sieswerda 2003) which reads a log file (plain text or SMCL) and removes all command syntax and other extraneous material, leaving behind only output. However, in doing so, `cleanlog` leaves behind multiple adjacent blank lines.

pr0039

Consider the following lines from a file. (The file below obviously did not result from `cleanlog`, but it will serve for the purpose of this example.) I will write `EOL` everywhere the file contains an end-of-line character sequence.

```
here is a line.  the next two lines are blank in the original file.EOL
EOL
EOL
here is another line.  the next line is blank in the original file.EOL
EOL
this is the last line of the file.EOL
```

The first `filefilter` syntax you might think of would be

```
. filefilter x.log y.log, from(\r\n\r\n) to(\r\n)
```

but it will not do what we want. Because there are EOL characters at the end of nonblank lines, if all adjacent pairs of EOL characters (\`W`\`W`) were replaced with single EOL characters (\`W`), the file above would end up looking like

```
here is a line.  the next two lines are blank in the original file.EOL
here is another line.  the next line is blank in the original file.EOL
this is the last line of the file.EOL
```

with no blank lines at all. To have a blank line between sections of output, there must be two adjacent EOL characters: one at the end of a line, and another on a line all by itself (the blank line).

Thus, to compress multiple adjacent blank lines down to single blank lines, we need to replace every occurrence of three adjacent EOL characters with two EOL characters:

```
. filefilter x.log y.log, from(\W\W\W) to(\W\W)
```

We still have not quite achieved the desired result. If we issue the above command only once, there may still be adjacent empty lines left in the file. We actually need to call `filefilter` multiple times, each time changing every three newlines to two newlines. I will assume that `x.log` is the original file, and will use `y.log` and `z.log` as output files with `filefilter` so that the original file will be left unchanged:

```
. filefilter x.log y.log, from(\W\W\W) to(\W\W)
. filefilter y.log z.log, from(\W\W\W) to(\W\W)
. filefilter z.log y.log, from(\W\W\W) to(\W\W) replace
. filefilter y.log z.log, from(\W\W\W) to(\W\W) replace
. filefilter z.log y.log, from(\W\W\W) to(\W\W) replace
...
```

The above should continue until no more changes are made. We can automate this by checking the return results from `filefilter` to see if the `from()` pattern was found. If it was not, we know there were no changes made, and thus, no more changes to be made:

```
filefilter x.log y.log, from(\W\W\W) to(\W\W)
local nchanges = r(occurrences)
while `nchanges´ != 0 {
    filefilter y.log z.log, from(\W\W\W) to(\W\W) replace
    filefilter z.log y.log, from(\W\W\W) to(\W\W) replace
    local nchanges = r(occurrences)
}
...
```

After the code above is executed, `y.log` will contain the desired file, and `z.log` can be discarded. It is possible that the code above will call `filefilter` one more time than is necessary, but unless we have an extremely large file that takes `filefilter` some time to process, we won't even notice.

While it may seem inefficient to use `filefilter` to make multiple passes through a file until the desired result is achieved, it is a fast and easy way to make such modifications. For very large files, Stata's `file` command ([P] **file**) or Mata's I/O functions ([M-4] **io**) could be used to perform such processing in a single pass, but they require a higher level of programming effort.

Reference

Sieswerda, L. E. 2003. cleanlog: Stata module to clean log files. Boston College Department of Economics, Statistical Software Components S432401. Downloadable from http://ideas.repec.org/c/boc/bocode/s432401.html.