








13 FEB 2013

## The functional and performance differences of sed, awk and other Unix parsing utilities

 This article was written many years ago. Its technical content may be outdated.

There are a few Unix utilities that can be used to perform find and replace operations on files.

- [sed](#)  (**s**tream **e**ditor) extends the feature set of [ed](#)  (**e**dit text), an early Unix text editor that is counted as a forefather of [vi](#) .
- [awk](#)  (authors **A**ho, **W**einberger and **K**ernighan) is a utility that was originally used for formatting reports, and is very useful when working with tabulated data — its key feature being to automatically split input lines into their component fields.
- [grep](#)  (the `ed` command “`g/re/p`” would find the regular expression “`/re/`” in a file and print out the matched lines) is commonly used to find (but not replace) regular expressions in strings.

Both `sed` and `awk` accomplish the same task — given an input stream (a text file or incoming data over a network, for example), the utilities process the stream

sequentially line by line, finding and performing actions on text that match certain criteria specified by regular expressions

certain criteria specified by regular expressions.

For example, assume we have a file called `pangrams` [↗](#) that contains the following table of text:

	rank	length	text
1	1	36	How quickly daft jumping zebras vex.
2	2	38	Jackdaws love my big sphinx of quartz.
3	3	44	The quick brown fox jumps over the lazy dog

The format of a sed command is:

```
1 | sed s/search_pattern/replace_pattern/ inputfile
```

So if we wanted to replace every occurrence of "3" with "[3]" in the pangrams file, then we would execute the command:

```
1 | sed s/3/[3]/ pangrams
```

Which would print:

	rank	length	text
1	1	[3]6	How quickly daft jumping zebras vex.
2	2	[3]8	Jackdaws love my big sphinx of quartz.
3	[3]	44	The quick brown fox jumps over the lazy dog.

Alternatively, awk can perform more advanced actions than sed using the following syntax:

```
1 | awk search_pattern {actions}
```

So, if we wanted to only print the "length" field of lines with a "rank" of 2 or less, and in those fields we want to replace "3" with "[3]", then we could execute the command:

```
1 | awk '{if ($1 <= 2) {sub(/3/, "[3]"); print $2}}' pangrams
```

Which would print:

```
1 | [3]6
2 | [3]8
```

(Note that the column headings have also been removed because they did not fit the specified search pattern.)

As can be seen above, sed is much more readable for simple text parsing, but as may be expected it is not quite as flexible — despite having a primitive built-in scripting language (allowing simple conditional and goto statements), awk is much more powerful (in fact, awk is [Turing complete](#) ↗, with syntax similar to C, and has [PCRE matching](#) ↗ for regular expressions). sed's other advantage is that its execution time for simple parsing is shorter, compared to awk or a simple Python script acting on the same input.

To demonstrate this difference in performance, on a fairly low-powered Unix box I generated a file filled with two billion [base64 characters](#) ↗ and applied a simple regular expression search and replace operation using various utilities, which was repeated 10 times for each execution. (Output was redirected to /dev/null to ensure the write speed of the hard drive would not be the limiting factor for execution time.)

Utility	Operation type	Execution time (10 iterations)	Characters processed per second
grep	search only	41 seconds	489.3 million
sed	search & replace	4 minutes 4 seconds	82.1 million
awk	search & replace	4 minutes 46 seconds	69.8 million
Python script	search & replace	4 minutes 50 seconds	69.0 million
PHP script	search & replace	15 minutes 44 seconds	21.2 million

I noticed a few interesting characteristics in the results of these performance tests.

- All commands were limited by the CPU for the duration of the test (and bound to just one CPU. for how would you parallelise this operation?).

- Very low memory usage throughout, which is expected since the stream editors only analyse one line of the file at a time.
  - sed did perform better than awk — a 42 second improvement over 10 iterations.
  - Surprisingly (to me), the Python script performed almost as well as the built-in Unix utilities. It is possible that compiling the Python source at runtime gives a performance boost which offsets the interpreted Unix commands that are designed to be as efficient as possible. (Or maybe Python is just blazingly fast...) In either case, it certainly wouldn't be too much of a burden to use a Python script rather than a built-in Unix utility, especially if the script is to be run on a Windows machine or if more complex logic or data validation is required, such as an email [↗](#) or postal [↗](#) address, which are nigh on impossible to implement correctly without the aid of 3rd party libraries.
-