Enough sed to be useful

brianstorti.com/enough-sed-to-be-useful

January 20, 2015

`sed` is a text editor that is probably already installed in your machine and can help you be more productive. It can make the boring and time consuming task of editing multiple files a breeze, and it shouldn't take more than a few minutes to learn the basics.

The stream editor

`sed` is a non-interactive editor. It means that, unlike most of the text editors that you're probably used to, like `vim` or `sublime`, it just reads a set of commands from a script and execute these commands in a given file.
This is the command syntax that is the most commonly used:

```
$ sed -e 'script commands here' file.txt
```

This way we can pass an inline command. We could also change the `-e` flag for a `-f` and give it a filename where the script is defined:

```
$ sed -f scriptfile file.txt
```

> Just read sed's manpage if you want to learn more about the other arguments that it accepts.

The structure of a sed command

A `sed` command consists of an **address** and an **editing instruction**.
The address tells `sed` in which lines the command should be executed, and the editing instruction tells it what to do with these lines.

The address

An address can be a line number, a regular expression or a special symbol, like `$` for the last line and `\n` for newlines.

This address is optional, and if no address is provided, `sed` will just execute the command for every line. You can also provide one or two addresses, one meaning that the command should be executed just for lines that match that given address, and two meaning that the command should be executed for the lines between the first and the second address (inclusive). These addresses are separated by a comma.

```
$ sed -e '1,3 command' file.txt
# will execute the command for the first, second and third line.

$ sed -e '/PATTERN/ command' file.txt
# will execute the command just for lines that match the pattern

$ sed -e '/BEGIN/,/END/ command' file.txt
# will execute the command starting in the line that matches BEGIN, until the
lines that matches END
```

The editing instruction

These instructions are single characters that tell `sed` what to do with the current line. The most used (and, maybe, the most useful) editing instructions is `s`, to substitute a pattern:

```
$ echo 'foo bar foo baz' | sed -e 's/foo/FOO/'
# FOO bar foo baz
```

These commands can also receive an argument (or flag), that will tell them how to operate. For instance, we could tell `s` to substitute all occurrences in that line, instead of just the first one, passing the `g` flag:

```
$ echo 'foo bar foo baz' | sed -e 's/foo/FOO/g'
# FOO bar FOO baz
```

If you want to execute more than one command for a matched address, just put them all around curly brackets (`{}`):

```
$ sed -e '/bar/{
s/a/A/g
a\
this line was appended
}' file.txt
```

This command will read `file.txt`, and for every line that matches the pattern `/bar/`, it will `s`ubstitute “a” for “A”, and then `a`ppend a new line that says “this line was appended”. You can also do different things to different addresses, in the same script:

```
$ sed -e '/bar/{
s/a/A/g
a\
this line was appended
}
/foo/d' file.txt
```

So this script will do the same things as the one before, but it will also `d`elete all the lines that match `/foo/`.

And that’s pretty much it. You know how to provide addresses and tell `sed` what to do with these lines, that’s the basic structure of how you will interact with `sed` most of the time. Of course there are a lot more editing instructions, and a quick look at the manpage can show you more about how powerful `sed` can be.

Some caveats

If you are following the examples, you probably have noticed that `sed` never actually changes the original file, it just shows you the result of the script execution.
`sed` stands for “stream editor” for a reason. Like most UNIX programs, it receives an input and directs the results of the script execution to the standard output. What if you really want to change the file with your script?

We can pass a `-i` argument to `sed`, that’s pretty useful. It tells `sed` to create a backup, with the given extension, that has the content of the file before the script was applied:

```
$ sed -i '.bkp' -e '/foo/d' file.txt
```

This command will execute the script (removing all the lines that match `/foo/`), and will create a `file.txt.bkp`, with the original content.
If you are confident your script works fine, just pass an empty string, and `sed` will replace the original file:

```
$ sed -i'' -e '/foo/d' file.txt
# file.txt is changed, and not backup is created.
```

`sed` works better with friends

And its best friend is usually `find`. This is a utility to, well, find something in your filesystem. We’ll usually use it to find files that we want to change.

Just pretend you made a terrible mistake and logged some users’ password. You are rotating your logs, so you can have dozens of log files, and now you need to find and remove all the lines with the word “password”, from all your log files. Now that you know how `sed` works, this should be easy:

```
$ find . -name "*.log" -exec sed -i'' -e '/password/d' {} \;
```

`find` will locate every log file, and then execute a `sed` command, and what it does is probably already familiar for you by now. `{}` just means “add the file that you found here”, so there is nothing different from the examples that we have seen so far.

Conclusion

`sed` is a very powerful tool that can save us precious time with just a few keystrokes. We always try to use the best tool for the job, and `sed` is a good candidate for a lot of common jobs that we have to do quite frequently.

I covered here just the basics of how to use it to perform some simple tasks, but your imagination is the limit. If you want to get inspiration, or just want to see some cool things that can be done, this file has a bunch of interesting one-liners.

Interested in learning Kubernetes?

I just published a new book called Kubernetes in Practice, you can use the discount code **blog** to get 10% off.

Get fresh articles in your inbox

If you liked this article, you might want to subscribe. If you don't like what you get, unsubscribe with one click.