Welcome to the land of AWK

Darren Drapkin

This is a brief introduction to AWK, one of my favourite programming languages.

I use AWK whenever I have an excuse to. Despite this, it is a serious programming language, not just a toy or demo, like 'goo.' It is not object oriented, though it dates from the time the first OO languages appeared. It is not lots of things. So, what is it really? What can it do — put in a positive sense? It is a kind of pattern/action file processing language. It can do most of what a shell can do, but it is very much its own thing.

The principle virtue of AWK is that it is designed for the writers of short programs that process flat file data sets and give a static output. Just the sort of thing that a lecturer may need to wow new data science students with.

This is not the only use case for it, however. Just think of all those things that the Linux kernel outputs under `/proc/` and `/sys/`. They look like files to the system, except that they are zero-length. They are the sort of thing that `top`, `htop` and the like look at for process information. They are where your battery charge indicator and other hardware monitors look for hardware information, or at least where they should look. A lot of time and effort has been put into getting the kernel not only to perform, but also to report on its doings.

If `top` does not work the way you want it to and `systemd` still gets your back up, then you may consider using AWK to parse `ps` output and deliver a series of synthesised command strings that you can paste into a shell.

If, like me, you keep a diary online, you may like to write a series of utility scripts that archive it and retrieve apposite passages as required.

I had a go at a set of AWK programmes, recently, that substitute for shell functions. There is a sort algorithm that is almost certainly broken; can we fix it? There is a cheeky built in feature masquerading as a serious programme. There is a program that sums a column of numbers up. There is a program that sums up values broken down into categories.

If you want to stretch things, you may consider writing an intelligent formatting script for printed/terminal output. Start with a begin pattern that identifies the file type. Lets say it's an `html` file that we are printing on a plain text terminal. What we would need would be a pattern that changes an `html <br>` tag into a `\n` character, another that underlines links, another that prints asterisks for bullet points, and so on.

I like AWK. I hope you will too. Especially if it is new to you, and if you have a text handling problem, or even one of the simpler AI problems, such as a magic filter, that can decide what to do with a file, based on its contents and some simple metadata that indicate what you want to do with it. A bit abstruse an example? How about this one?

What's a configuration file? I know! Its a text file, or should be. Have you thought of creating a script mantle for a `config` that is a bit too long to edit by hand?

Now, for the Kata at the end. You may have read, as I did, in the Unix-Haters Handbook, in the 1990s, 'If the PPID of a process is the PID of my shell, its mine and I can do what I want with it!' What is wrong with this assertion, if anything, and can AWK help?