Awk One-Liners Explained

by

@pkrumins
Peteris Krumins
peter@catonmat.net
http://www.catonmat.net
good coders code, great reuse

Contents

Preface

Thanks!

Thank you for purchasing my "Awk One-Liners Explained" e-book! This is my first e-book that I have ever written and I based it on article series "Famous Awk One-Liners Explained" that I wrote on my www.catonmat.net blog. I went through all the one-liners in the articles, improved them, fixed a lot of mistakes, added an introduction to Awk one-liners and two new chapters. The two new chapters are Awk Special Variables that summarizes some of the most commonly used Awk variables and Idiomatic Awk that explains what idiomatic Awk is.

You might wonder why I called the article series "famous"? Well, because I based the articles on the famous awk1line.txt file by Eric Pement. This file has been circulating around Unix newsgroups and forums for years and it's very popular among Unix programmers. That's how I actually learned the Awk language myself. I went through all the one-liners in this file, tried them out and understood how they exactly work. Then I thought it would be a good idea to explain them on my blog, which I did, and after that I thought, why not turn it into a book? That's how I ended up writing this book.

I have also planned writing two more books called "Sed One-Liners Explained" and "Perl One-Liners Explained". The sed book will be based on Eric Pement's sed1line.txt file and "Famous Sed One-Liners Explained" article series and the Perl book will be based on my "Famous Perl One-Liners Explained" article series. I am also going to create perl1line.txt file of my own. If you're interested, subscribe to my blog and follow me on Twitter. That way you'll know when I publish all of this!

Credits

I'd like to thank Eric Pement who made the famous awk1line.txt file that I learned Awk from and that I based this book on. I'd also like to thank waldner and pgas from #awk channel on FreeNode IRC network for always helping me with Awk, Madars Virza for proof reading the book before I published it and correcting several glitches, Antons Suspans for proof reading the book after I published it, Abraham Alhashmy for giving advice on how to improve the design of the book, everyone who commented on my blog while I was writing the Awk one-liners article series, and everyone else who helped me with Awk and this book.

One

Introduction

1.1 Awk One-Liners

Knowing Awk makes you really powerful when working in the shell. Check this out, suppose you want to print the usernames of all users on your system. You can do it very quickly with this one-liner:

```
awk -F: '{print $1}' /etc/passwd
```

This is really short and powerful, isn't it? As you know, the format of `/etc/passwd` is colon separated:

```
root:x:0:0:0:/root:/bin/bash
```

The one-liner above says: Take each line from `/etc/passwd`, split it on the colon `-F:` and print the first field `$1` of each line.

Here are the first few lines of output when I run this program on my system:

```
root
bin
daemon
adm
lp
sync
...
```

Exactly what I expected.

Now compare it to a C program that I just wrote that does the same:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_LINE_LEN 1024

int main() {
    char line[MAX_LINE_LEN];
    FILE *in = fopen("/etc/passwd", "r");
    if (!in) exit(EXIT_FAILURE);

    while (fgets(line, MAX_LINE_LEN, in) != NULL) {
        char *sep = strchr(line, ':');
        if (!sep) exit(EXIT_FAILURE);
        *sep = '\0';
        printf("%s\n", line);
    }
    fclose(in);
    return EXIT_SUCCESS;
}
```

This is much longer and you have to compile the program, only then you can run it. If you make any mistakes, you have to recompile again.

That's why one-liners are called one-liners. They are short, easy to write and they do one and only one thing really well. I am pretty sure you're starting to see how mastering Awk and one-liners can make you much more efficient when working in the shell, with text files and with computers in general.

Here is another one-liner, this one numbers the lines in some file:

```
awk '{ print NR ". " $0 }' somefile
```

Isn't this beautiful? The `NR` special variable keeps track of current line number so I just print it out, followed by a dot and `$0` that, as you'll learn, contains the whole line. And you're done.

I know that a lot of my book readers would argue that Perl does exactly the same, so why should you learn Awk? My answer is very simple, yes, Perl does exactly the same, but why not be the master of the shell? Why not learn Awk, sed, Perl and other utilities? Besides Perl was created based on ideas from Awk, so why not learn Awk to see how Perl evolved. That gives you a unique perspective on programming languages, doesn't it?

Overall, this book contains 70 well explained one-liners. Once you go through them, you should have a really good understanding of Awk and you'll be the master shell problem solver. Enjoy this book!

Two

Line Spacing

2.1 Double-space a file

```
awk '1; { print "" }'
```

So how does this one-liner work? A one-liner is an Awk program and every Awk program consists of a sequence of pattern-action statements `pattern { action statement }`. In this case there are two statements `1` and `{ print "" }`. In a pattern-action statement either the pattern or the action may be missing. If the pattern is missing, the action is applied to every single line of input. A missing action is equivalent to `{ print }`. The first pattern-action statement is missing the action, therefore we can rewrite it as:

```
awk '1 { print }; { print "" }'
```

An action is applied to the line only if the pattern matches, i.e., pattern is true. Since `1` is always true, this one-liner translates further into two print statements:

```
awk '{ print }; { print "" }'
```

Every print statement in Awk is silently followed by the `ORS` – Output Record Separator variable, which is a newline by default. The first print statement with no arguments is equivalent to `print $0`, where `$0` is the variable holding the entire line (not including the newline at the end). The second print statement seemingly prints nothing, but knowing that each print statement is followed by `ORS`, it actually prints a newline. So there we have it, each line gets double-spaced.