

# Awk One-Liners Explained

by

[@pkrumins](#)

Peteris Krumins

[peter@catonmat.net](mailto:peter@catonmat.net)

<http://www.catonmat.net>

good coders code, great reuse

---

# Contents

---

|                                                                                                     |          |
|-----------------------------------------------------------------------------------------------------|----------|
| <b>Contents</b>                                                                                     | <b>i</b> |
| <b>Preface</b>                                                                                      | <b>v</b> |
| <b>1 Introduction</b>                                                                               | <b>1</b> |
| 1.1 Awk One-Liners . . . . .                                                                        | 1        |
| <b>2 Line Spacing</b>                                                                               | <b>4</b> |
| 2.1 Double-space a file . . . . .                                                                   | 4        |
| 2.2 Another way to double-space a file . . . . .                                                    | 5        |
| 2.3 Double-space a file so that no more than one blank line appears between lines of text . . . . . | 6        |
| 2.4 Triple-space a file . . . . .                                                                   | 6        |
| 2.5 Join all lines . . . . .                                                                        | 7        |
| <b>3 Numbering and Calculations</b>                                                                 | <b>8</b> |
| 3.1 Number lines in each file separately . . . . .                                                  | 8        |
| 3.2 Number lines for all files together . . . . .                                                   | 8        |
| 3.3 Number lines in a fancy manner . . . . .                                                        | 9        |
| 3.4 Number only non-blank lines in files . . . . .                                                  | 9        |
| 3.5 Count lines in files . . . . .                                                                  | 9        |
| 3.6 Print the sum of fields in every line . . . . .                                                 | 10       |
| 3.7 Print the sum of fields in all lines . . . . .                                                  | 10       |
| 3.8 Replace every field by its absolute value . . . . .                                             | 11       |
| 3.9 Count the total number of fields (words) in a file . . . . .                                    | 11       |
| 3.10 Print the total number of lines containing word "Beth" . . . . .                               | 12       |
| 3.11 Find the line containing the largest (numeric) first field . . . . .                           | 12       |
| 3.12 Print the number of fields in each line, followed by the line . . . . .                        | 13       |
| 3.13 Print the last field of each line . . . . .                                                    | 13       |

---

|          |                                                                                                    |           |
|----------|----------------------------------------------------------------------------------------------------|-----------|
| 3.14     | Print the last field of the last line . . . . .                                                    | 13        |
| 3.15     | Print every line with more than 4 fields . . . . .                                                 | 14        |
| 3.16     | Print every line where the value of the last field is greater<br>than 4 . . . . .                  | 14        |
| <b>4</b> | <b>Text Conversion and Substitution</b>                                                            | <b>15</b> |
| 4.1      | Convert Windows/DOS newlines (CRLF) to Unix newlines<br>(LF) from Unix . . . . .                   | 15        |
| 4.2      | Convert Unix newlines (LF) to Windows/DOS newlines (CRLF)<br>from Unix . . . . .                   | 16        |
| 4.3      | Convert Unix newlines (LF) to Windows/DOS newlines (CRLF)<br>from Windows/DOS . . . . .            | 16        |
| 4.4      | Convert Windows/DOS newlines (CRLF) to Unix newlines<br>(LF) from Windows/DOS . . . . .            | 17        |
| 4.5      | Delete leading whitespace (spaces and tabs) from the begin-<br>ning of each line (ltrim) . . . . . | 18        |
| 4.6      | Delete trailing whitespace (spaces and tabs) from the end of<br>each line (rtrim) . . . . .        | 18        |
| 4.7      | Delete both leading and trailing whitespaces from each line<br>(trim) . . . . .                    | 18        |
| 4.8      | Insert 5 blank spaces at beginning of each line . . . . .                                          | 19        |
| 4.9      | Align all text to the right right on a 79-column width . . . . .                                   | 19        |
| 4.10     | Center all text on a 79-character width . . . . .                                                  | 20        |
| 4.11     | Substitute (find and replace) "foo" with "bar" on each line . . . . .                              | 20        |
| 4.12     | Substitute "foo" with "bar" only on lines that contain "baz" . . . . .                             | 21        |
| 4.13     | Substitute "foo" with "bar" only on lines that don't contain<br>"baz" . . . . .                    | 22        |
| 4.14     | Change "scarlet" or "ruby" or "puce" to "red" . . . . .                                            | 22        |
| 4.15     | Reverse order of lines (emulate "tac") . . . . .                                                   | 22        |
| 4.16     | Join a line ending with a backslash with the next line . . . . .                                   | 23        |
| 4.17     | Print and sort the login names of all users . . . . .                                              | 23        |
| 4.18     | Print the first two fields in reverse order on each line . . . . .                                 | 24        |
| 4.19     | Swap first field with second on every line . . . . .                                               | 25        |
| 4.20     | Delete the second field on each line . . . . .                                                     | 25        |
| 4.21     | Print the fields in reverse order on every line . . . . .                                          | 25        |
| 4.22     | Remove duplicate, consecutive lines (emulate "uniq") . . . . .                                     | 26        |
| 4.23     | Remove duplicate, nonconsecutive lines . . . . .                                                   | 27        |
| 4.24     | Concatenate every 5 lines of input with a comma . . . . .                                          | 28        |

---

|          |                                                                                                                                 |           |
|----------|---------------------------------------------------------------------------------------------------------------------------------|-----------|
| <b>5</b> | <b>Selective Printing and Deleting of Certain Lines</b>                                                                         | <b>30</b> |
| 5.1      | Print the first 10 lines of a file (emulates "head -10") . . . . .                                                              | 30        |
| 5.2      | Print the first line of a file (emulates "head -1") . . . . .                                                                   | 31        |
| 5.3      | Print the last 2 lines of a file (emulates "tail -2") . . . . .                                                                 | 31        |
| 5.4      | Print the last line of a file (emulates "tail -1") . . . . .                                                                    | 32        |
| 5.5      | Print only the lines that match a regular expression <code>"/regex/"</code><br>(emulates "grep") . . . . .                      | 32        |
| 5.6      | Print only the lines that do not match a regular expression<br><code>"/regex/"</code> (emulates "grep -v") . . . . .            | 33        |
| 5.7      | Print the line immediately before a line that matches <code>"/regex/"</code>                                                    | 33        |
| 5.8      | Print the line immediately after a line that matches <code>"/regex/"</code><br>(but not the line that matches itself) . . . . . | 34        |
| 5.9      | Print lines that match any of "AAA" or "BBB", or "CCC" .                                                                        | 34        |
| 5.10     | Print lines that contain "AAA", "BBB", and "CCC" in this<br>order . . . . .                                                     | 34        |
| 5.11     | Print only the lines that are 65 characters in length or longer                                                                 | 35        |
| 5.12     | Print only the lines that are less than 64 characters in length                                                                 | 35        |
| 5.13     | Print a section of file from regular expression to end of file .                                                                | 36        |
| 5.14     | Print lines 8 to 12 (inclusive) . . . . .                                                                                       | 36        |
| 5.15     | Print line number 52 . . . . .                                                                                                  | 36        |
| 5.16     | Print section of a file between two regular expressions (inclu-<br>sive) . . . . .                                              | 37        |
| 5.17     | Print all lines where 5th field is equal to "abc123" . . . . .                                                                  | 37        |
| 5.18     | Print any line where field #5 is not equal to "abc123" . . . . .                                                                | 38        |
| 5.19     | Print all lines whose 7th field matches a regular expression .                                                                  | 38        |
| 5.20     | Print all lines whose 7th field doesn't match a regular ex-<br>pression . . . . .                                               | 38        |
| 5.21     | Delete all blank lines from a file . . . . .                                                                                    | 39        |
| <b>6</b> | <b>String and Array Creation</b>                                                                                                | <b>40</b> |
| 6.1      | Create a string of a specific length (generate a string of x's<br>of length 513) . . . . .                                      | 40        |
| 6.2      | Insert a string of specific length at a certain character posi-<br>tion (insert 49 x's after 6th char) . . . . .                | 41        |
| 6.3      | Create an array from string . . . . .                                                                                           | 42        |
| 6.4      | Create an array named "mdigit", indexed by strings . . . . .                                                                    | 42        |
| <b>A</b> | <b>Awk Special Variables</b>                                                                                                    | <b>44</b> |

---

|          |                                                                    |           |
|----------|--------------------------------------------------------------------|-----------|
| A.1      | FS – Input Field Separator . . . . .                               | 44        |
| A.2      | OFS – Output Field Separator . . . . .                             | 45        |
| A.3      | NF – Number of Fields on the current line . . . . .                | 46        |
| A.4      | NR – Number of records seen so far (current line number) . . . . . | 47        |
| A.5      | RS – Input Record Separator . . . . .                              | 47        |
| A.6      | ORS – Output Record Separator . . . . .                            | 48        |
| <b>B</b> | <b>Idiomatic Awk</b>                                               | <b>49</b> |
|          | <b>Index</b>                                                       | <b>51</b> |

---

## 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](http://www.catonmat.net). 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.