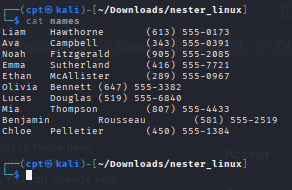
Same as sed, awk considered non-interactive text editor but provide greater/wider range of capabilities. This tool even has its own scripting languages, syntactically similar to C. Like sed, awk executes a set of instructions for each line of input. You can specify instructions on the command line or create a script file.

**awk ’***instructions***’** *files*

*Records and Fields*

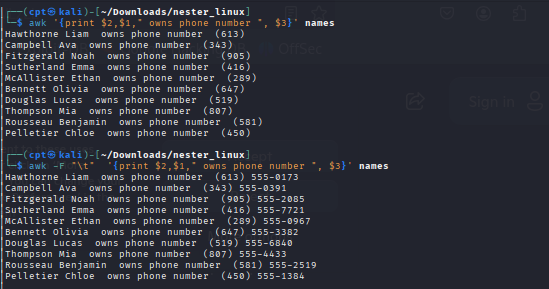
Awk makes the assumption that its input is structured and not just an endless string of characters. In the simplest case, it takes each input line as a record and each word, separated by spaces or tabs, as a field. (The characters separating the fields are often referred to as *delimiters*.) The following record in the file *names* has three fields, separated by either a space or a tab.



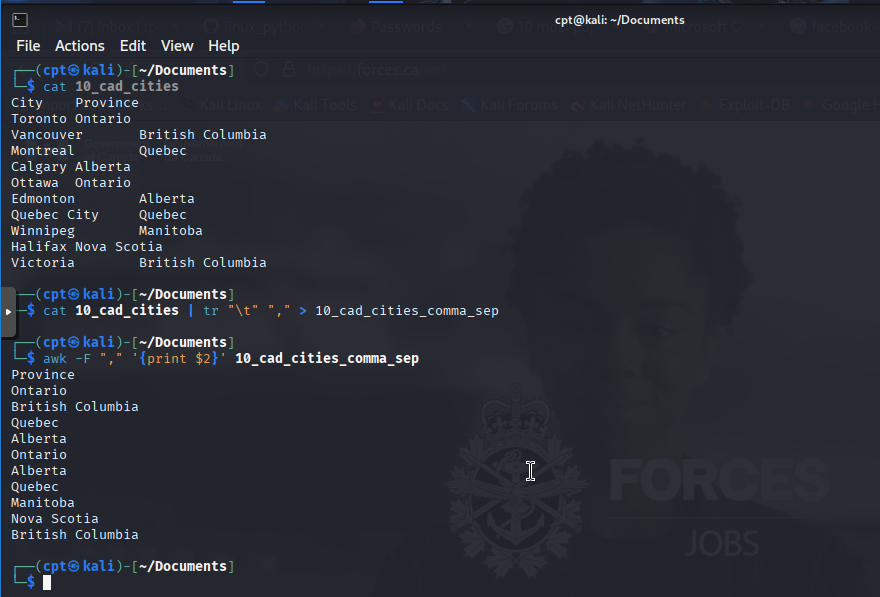
Two or more consecutive spaces and/or tabs count as a single delimiter.

*Referencing and Separating Fields*

Awk allows you to refer to fields in actions using the field operator $. This operator is followed by a number or a variable that identifies the position of a field by number. “$1” refers to the first field, “$2” to the second field, and so on. “$0” refers to the entire input record. The following example displays the last name first and the first name second, followed by the phone number. Notice that if explicit delimiter is not specified, both space and tab are considered a valid delimiter.

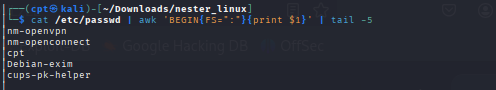


Use the *-F* option to change the field separator to a comma.



It is usually a better practice, and more convenient, to specify the field separator in the script itself. The system variable FS can be defined to change the field separator. Because this must be done before the first input line is read, we must assign this variable in an action controlled by the BEGIN rule.

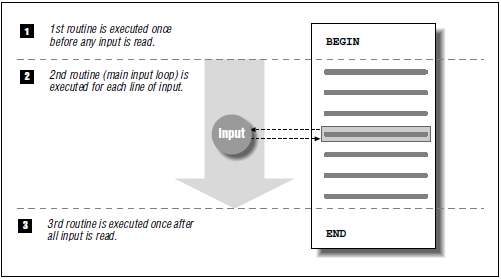
**BEGIN { FS = ":" }**



It’s important to understand the basic model that awk offers the programmer. Part of the reason why awk is easier to learn than many programming languages is that it offers such a well-defined and useful model to the programmer. An awk program consists of what we will call a main input loop. A loop is a routine that is executed over and over again until some condition exists that terminates it. You don’t write this loop, it is given—it exists as the framework within which the code that you do write will be executed. The main input loop in awk is a routine that reads one line of input from a file and makes it available for processing. The main input loop is executed as many times as there are lines of input. As you saw in the “Hello, world” examples, this loop does not execute until there is a line of input. It terminates when there is no more input to be read. Awk allows you to write two special routines that can be executed before any input is read and after all input is read. These are the procedures associated with the BEGIN and END rules, respectively. In other words, you can do some preprocessing before the main input loop is ever executed and you can do some post processing after the main input loop has terminated. The BEGIN and END procedures are optional.

You can think of an awk script as having potentially three major parts: what happens before, what happens during, and what happens after processing the input. Figure 7-1 shows the relationship of these parts in the flow of control of an awk

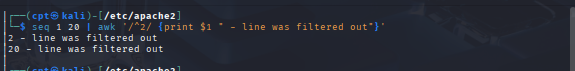
script.



When awk reads an input line, it attempts to match each pattern-matching rule in a script. Only the lines matching the particular pattern are the object of an action. If no action is specified, the line that matches the pattern is printed (executing the print statement is the default action). Consider the following script:

**/ˆ$/ { print "This is a blank line." }**

In the example below, we filtered out all the line that start with digit 2 and additional sentence was appended to it.



*Expressions*

The use of expressions in which you can store, manipulate, and retrieve data is quite different from anything you can do in sed, yet it is a common feature of most programming languages. A variable is an identifier that references a value. To define a variable, you only have to name it and assign it a value. The name can only contain letters, digits, and underscores, and may not start with a digit. Case distinctions in variable names are important: Salary and salary are two different variables. Variables are

not declared; you do not have to tell awk what type of value will be stored in a variable. Each variable has a string value and a numeric value, and awk uses the appropriate value based on the context of the expression. (Strings that do not consist of numbers have a numeric value of 0.) Variables do not have to be initialized; awk automatically initializes them to the empty string, which acts like 0 if used as

a number. The following expression assigns a value to x:

x = 1

x is the name of the variable, = is an assignment operator, and 1 is a numeric constant.

The following expression assigns the string “Hello” to the variable z:

z = "Hello"

A space is the string concatenation operator. The expression:

z = "Hello" "World"

concatenates the two strings and assigns “HelloWorld” to the variable z.

The dollar sign ($) operator is used to reference fields. The following expression

assigns the value of the first field of the current input record to the variable w:

w = $1

A variety of operators can be used in expressions. Arithmetic operators are listed below

+ Addition

- Subtraction

\* Multiplication

/ Division

% Modulo

ˆ Exponentiation

\*\* Exponentiation

Look at the following example, which counts each blank line in a file.

/ˆ$/ {x += 1} END{print x}

