Introduction to AWK

Nguyen Le Duc Minh

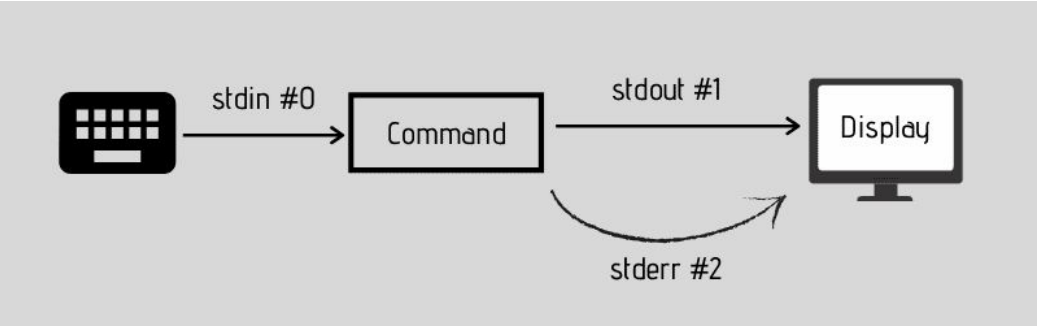
Contents

1. Simple manipulation with awk
2. Options in awk
3. Awk Program Structure (BEGIN and END block)
4. Built-In Variables in Awk
5. Unary Operators
6. Assignment Operators
7. AWK Statements

Data Streams

When you run a Linux command, there are three data streams that play a part in it:

- standard input (**stdin**)– usually the input from the keyboard.
- standard output (**stdout**) – displays the output from commands, usually to the terminal.
- standard error (**stderr**) – displays error output from commands. It is usually sent to the same output as standard output, but it can be redirected.



stdout

```
dminh@M /mnt/hdd/dminh/microbiome
$ ls
SRR18218331_1.fastq.gz  feature-table.qza  hello2.sh  qiime2.ipynb
SRR18218331_2.fastq.gz  hello.sh           input.sh   script.sh
(base)
dminh@M /mnt/hdd/dminh/microbiome
$ echo $?
0
(base)
dminh@M /mnt/hdd/dminh/microbiome
$ Ls
Ls: command not found
(base)
dminh@M /mnt/hdd/dminh/microbiome
$ echo $?
127
```

stderr

AWK

The awk command is a Linux tool and programming language that allows users to process text files. It is especially helpful when the lines in a text files are in a record format, i.e, when each line (record) contains multiple fields separated by a delimiter. You can also write programming logic using awk even when there are no input files that needs to be processed.

Syntax: awk [options] 'selection _criteria {action }' input-file > output-file

WHAT CAN WE DO WITH AWK?

1. AWK Operations:

(a) Scans a file line by line

(b) Splits each input line into fields

(c) Compares input line/fields to pattern

(d) Performs action(s) on matched lines

2. Useful For:

(a) Transform data files

(b) Produce formatted reports

3. Programming Constructs:

(a) Format output lines

(b) Arithmetic and string operations

(c) Conditionals and loops

1 Simple manipulation with awk

Default behavior of Awk: By default Awk prints every line of data from the specified file

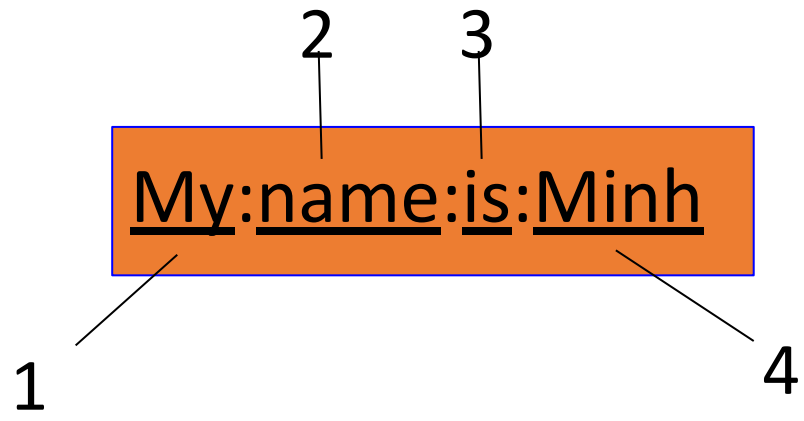
```
dminh@M /mnt/hdd/dminh/microbiome
$ cat example.txt
He likes attention 196
He likes bananas 51
He likes barbeque 44
He likes baseball 188
He likes basketball 57
He likes beer 281
He likes being 2026(base)
dminh@M /mnt/hdd/dminh/microbiome
$ awk '{print $0}' example.txt
He likes attention 196
He likes bananas 51
He likes barbeque 44
He likes baseball 188
He likes basketball 57
He likes beer 281
He likes being 2026
```

```
dminh@M /mnt/hdd/dminh/microbiome
$ grep "banana" example.txt
He likes bananas 51
(base)
dminh@M /mnt/hdd/dminh/microbiome
$ awk '/banana/{print $0}' example.txt
He likes bananas 51
```

To output words that contain certain letters and print out words that match a pattern you specify, use the slashes **//**

2 Options in awk

-F <value> - tells awk what field separator to use.




```
dminh@M /mnt/hdd/dminh/microbiome
$ echo "My name is Minh" | awk '{print $4}'
Minh
(base)
dminh@M /mnt/hdd/dminh/microbiome
$ echo "My:name:is:Minh" | awk '{print $4}'

(base)
dminh@M /mnt/hdd/dminh/microbiome
$ echo "My:name:is:Minh" | awk -F":" '{print $4}'
Minh
```

Ex: As you know the file /etc/passwd stores user account information.It's a plain text file and all fields are separated by a colon (:) symbol. How can I print the 6th field ?

-f <file> : To specify a file that contains awk script.

```
dminh@M /mnt/hdd/dminh/microbiome
$ awk '{print $0}' demo.awk
{print NR, $3}
(base)
dminh@M /mnt/hdd/dminh/microbiome
$ awk -f demo.awk example.txt
1 attention
2 bananas
3 barbeque
4 baseball
5 basketball
6 beer
7 being
```

3 Awk Program Structure (BEGIN and END block)

BEGIN: To perform an action before data is processed
END: To perform an action after the data is processed

```
dminh@M /mnt/hdd/dminh/microbiome
$ awk 'BEGIN{print "This is BEGIN block\n"}; {print $0}; END{print "\nThis is END block"}' example.txt
This is BEGIN block

He likes attention 196
He likes bananas 51
He likes barbeque 44
He likes baseball 188
He likes basketball 57
He likes beer 281
He likes being 2026

This is END block
```

4 Built-In Variables in Awk

Splitting a Line Into Fields : For each record i.e line, the awk command splits the record delimited by whitespace character by default and stores it in the $n variables. If the line has 5 words, it will be stored in $1, $2, $3, $4 and $5 respectively. Also, $0 represents the whole line.

```
dminh@M /mnt/hdd/dminh/microbiome
$ awk '{print $0}' example.txt
He likes attention 196
He likes bananas 51
He likes barbeque 44
He likes baseball 188
He likes basketball 57
He likes beer 281
He likes being 2026
(base)
dminh@M /mnt/hdd/dminh/microbiome
$ awk '{print $3, $4}' example.txt
attention 196
bananas 51
barbeque 44
baseball 188
basketball 57
beer 281
being 2026
(base)
dminh@M /mnt/hdd/dminh/microbiome
$ awk '{print $3, $4}' example.txt | column -t
attention   196
bananas     51
barbeque    44
baseball    188
basketball  57
beer        281
being       2026
```

NR (Number of Records) number of row input.

Use of NR built-in variables (Display Line Number)

```
dminh@M /mnt/hdd/dminh/microbiome
$ awk '{print NR, $0}' example.txt
1 He likes attention 196
2 He likes bananas 51
3 He likes barbeque 44
4 He likes baseball 188
5 He likes basketball 57
6 He likes beer 281
7 He likes being 2026
```

Another use of NR built-in variables (Display Line From 2 to 4)

```
dminh@M /mnt/hdd/dminh/microbiome
$ awk 'NR==2, NR==4 {print NR, $0}' example.txt
2 He likes bananas 51
3 He likes barbeque 44
4 He likes baseball 188
```

NF (Number of Fields) number of fields input record and displays the last field of the file.

Use of NF built-in variables (Display Last Field)

```
dminh@M /mnt/hdd/dminh/microbiome
$ awk 'END{printf("Number of fields per record: %s\n", NF)}' example.txt
Number of fields per record: 4
(base)
dminh@M /mnt/hdd/dminh/microbiome
$ awk '{print $NF}' example.txt
196
51
44
188
57
281
2026
```

Awk is used to display the number of rows and fields of data

```
dminh@M /mnt/hdd/dminh/microbiome
$ awk 'END{print NR,NF}' example.txt
7 4
```

FS (Input field separator variable) Contains the character used to divide fields on the input line

RS (Record Separator variable) Stores the current record separator character. The default input line is the input record, which makes a newline the default record separator. The command is useful if the input is a comma-separated file (CSV).

```
dminh@M /mnt/hdd/dminh/microbiome
$ awk '{print $0}' example2.txt
He likes attention 196-He likes bananas 51-He likes barbeque 44-He likes baseball 188-He likes basketball 57-He likes beer 281-He likes being 2
026
(base)
dminh@M /mnt/hdd/dminh/microbiome
$ awk 'BEGIN{FS=" "; RS="-"}{print $0}' example2.txt
He likes attention 196
He likes bananas 51
He likes barbeque 44
He likes baseball 188
He likes basketball 57
He likes beer 281
He likes being 2026
```

```
dminh@M /mnt/hdd/dminh/microbiome
$ awk 'BEGIN{FS="-";RS="\n"}{print $1}' example2.txt
He likes attention 196
(base)
dminh@M /mnt/hdd/dminh/microbiome
$ awk 'BEGIN{FS="-";RS="\n"}{print $2}' example2.txt
He likes bananas 51
(base)
dminh@M /mnt/hdd/dminh/microbiome
$ awk 'BEGIN{FS="-";RS="\n"}{print $3}' example2.txt
He likes barbeque 44
```

OFS: Output Field Separator Variable

```
dminh@M /mnt/hdd/dminh/microbiome
$ awk  'BEGIN{FS=" "; OFS=":::"}{print $1,$2,$3}' example.txt
He:::likes:::attention
He:::likes:::bananas
He:::likes:::barbeque
He:::likes:::baseball
He:::likes:::basketball
He:::likes:::beer
He:::likes:::being
(base)
dminh@M /mnt/hdd/dminh/microbiome
$ awk  'BEGIN{FS=" "; OFS="//"}{print $1,$2,$3}' example.txt
He//likes//attention
He//likes//bananas
He//likes//barbeque
He//likes//baseball
He//likes//basketball
He//likes//beer
He//likes//being
```

Concatenator in the print statement “,” concatenates two parameters with a space which is the value of awk OFS by default.

So, Awk OFS value will be inserted between fields in the output as shown example.

ORS: Output Record Separator Variable

ORS stores the output record separator, which separates the output lines when Awk prints them. The default is a newline character.

```
dminh@M /mnt/hdd/dminh/microbiome
$ awk  'BEGIN{ORS="/"}{print $1,$2,$3}' example.txt
He likes attention/He likes bananas/He likes barbeque/He likes baseball/He likes basketball/He likes beer/He likes being/(base)
dminh@M /mnt/hdd/dminh/microbiome
$ awk  'BEGIN{ORS="\n\n"}{print $1,$2,$3}' example.txt
He likes attention

He likes bananas

He likes barbeque

He likes baseball

He likes basketball

He likes beer

He likes being
```

Using in gsub awk

```
dminh@M /mnt/hdd/dminh/microbiome
$ awk '{gsub("bananas", "BANANAS", $0); print $0}' example.txt
He likes attention 196
He likes BANANAS 51
He likes barbeque 44
He likes baseball 188
He likes basketball 57
He likes beer 281
He likes being 2026
```

```
dminh@M /mnt/hdd/dminh/microbiome
$ awk '{gsub("likes", "LIKES", $2); print $0}' example.txt
He LIKES attention 196
He LIKES bananas 51
He LIKES barbeque 44
He LIKES baseball 188
He LIKES basketball 57
He LIKES beer 281
He LIKES being 2026
```

gsub stands for global substitution. It replaces every occurrence of regex with the given string (sub).

5 Unary Operators

+ : The number (returns the number itself)
- : Negate the number
++ : Auto Increment
-- : Auto Decrement

```
dminh@M /mnt/hdd/dminh/microbiome
$ awk '{print ++$4}' example.txt
197
52
45
189
58
282
2027
(base)
dminh@M /mnt/hdd/dminh/microbiome
$ awk '{$4++; print $4}' example.txt
197
52
45
189
58
282
2027
```

```
dminh@M /mnt/hdd/dminh/microbiome
$ awk '{$4--; print $4}' example.txt
195
50
43
187
56
280
2025
(base)
dminh@M /mnt/hdd/dminh/microbiome
$ awk '{print --$4}' example.txt
195
50
43
187
56
280
2025
```

++ in the print statement the original value is printed
++ in a separate statement the resulting value is printed

6 Assignment Operators

awk uses = as the assignment operator

"=" -Assignment

"+=" -Shortcut addition assignment

"-=" -Shortcut subtraction assignment

"*=" -Shortcut multiplication assignment

"/=" -Shortcut division assignment

"%=" -Shortcut modulo division assignment

```
dminh@M /mnt/hdd/dminh/microbiome
$ awk '{sum+=$4; print sum}' example.txt
196
247
291
479
536
817
2843
```

```
dminh@M /mnt/hdd/dminh/microbiome
$ awk '{sum+=$4}; END{print sum}' example.txt
2843
```

7 AWK Statements

Awk Simple If statement

The simple if statement tests a condition, and if the condition returns true, performs the corresponding action(s).

Syntax: awk '{if (condition) {statement} }' [input_file]

```
dminh@M /mnt/hdd/dminh/microbiome
$ awk '{if(NR<5) print $0}' example.txt
He likes attention 196
He likes bananas 51
He likes barbeque 44
He likes baseball 188
(base)
dminh@M /mnt/hdd/dminh/microbiome
$ awk '{if($4<100) print $0}' example.txt
He likes bananas 51
He likes barbeque 44
He likes basketball 57
```

Awk If-Else statement

In the awk "If Else" statement you can also provide list of actions to perform if the condition is false. In the following syntax, if the condition is true statement 1 will be performed, if the condition is false statement 2 will be performed.

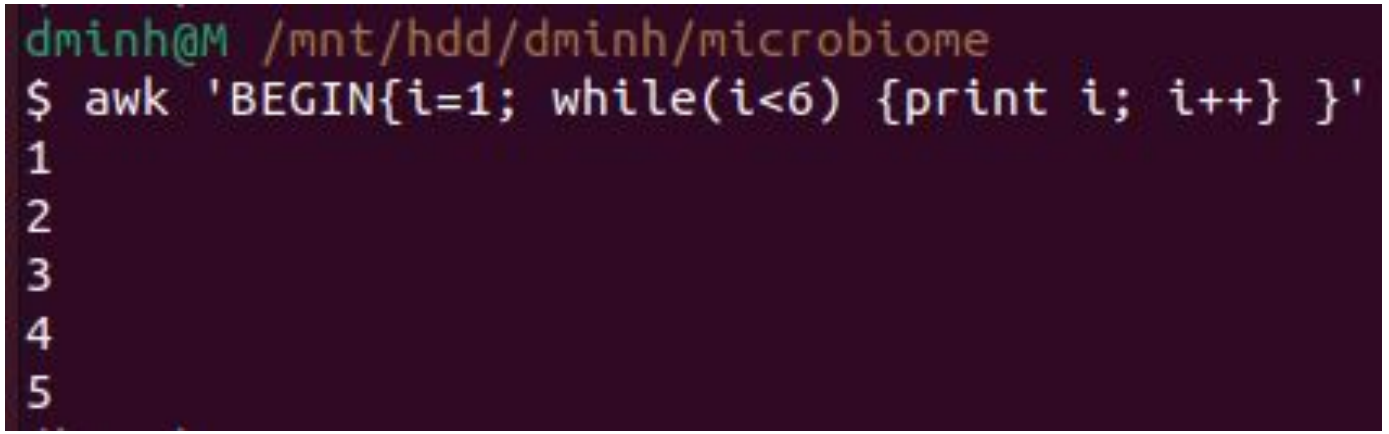
Syntax:

```
if (condition) {
    statements
} else {
        statements }
```

```
dminh@M /mnt/hdd/dminh/microbiome
$ awk '{if($4<100) print($0, "==>", "LOW"); else print($0, "==>", "HIGH")}' example.txt
He likes attention 196 ==> HIGH
He likes bananas 51 ==> LOW
He likes barbeque 44 ==> LOW
He likes baseball 188 ==> HIGH
He likes basketball 57 ==> LOW
He likes beer 281 ==> HIGH
He likes being 2026 ==> HIGH
```

Awk While Loop

Awk while looping statements are used to perform a set of actions again and again in succession. Awk keeps executing a statement as long as the loop condition is true.



Awk For Loop

```
dminh@M /mnt/hdd/dminh/microbiome
$ awk 'BEGIN{for(i=1; i<=10; i++) printf("The square of %d is %d\n",i, i*i)}'
The square of 1 is 1
The square of 2 is 4
The square of 3 is 9
The square of 4 is 16
The square of 5 is 25
The square of 6 is 36
The square of 7 is 49
The square of 8 is 64
The square of 9 is 81
The square of 10 is 100
```

break statement

```
dminh@M /mnt/hdd/dminh/microbiome
$ awk 'BEGIN{
    for (i=1; i<=10; i++) {
        printf("The square of %d is %d\n",i, i*i)
        if (i==5) {
            print "Reached 5, STOP."
            break
        }
    }
}'
The square of 1 is 1
The square of 2 is 4
The square of 3 is 9
The square of 4 is 16
The square of 5 is 25
Reached 5, STOP.
```

Exercise: Normalizing counts

```
dminh@M /mnt/hdd/dminh/microbiome
$ awk 'BEGIN { total=2843} {
    norm = $4/total
    print($1, $2, $3, norm)
}' example.txt
He likes attention 0.0689413
He likes bananas 0.0179388
He likes barbeque 0.0154766
He likes baseball 0.0661273
He likes basketball 0.0200492
He likes beer 0.0988393
He likes being 0.712628
```

Classwork

Use data awk_example.tsv to make the following requests.

- Check the number of rows and columns in the data

- Displays the contents of the last line of the “Row.names” field and the total number of lines contained in that field “Row.names”

- Show first 10 lines of these field : Row.names, AveExpr and P.Value

- Change “Row.names” to “ProbesName” using awk

- Filter data with P.Value > 0.95

- Extract data from line 1 to line 1000 and save it to a new file