
بسم الله الرحمن الرحيم

LINUX COMMANDS

Description of Linux Commands
Part XI

Feel free

Collected and Written by Mostafa Fazli

28 December 2021

FUNCTIONS

Functions in Bash Scripting are a great way to reuse code.

Think of a function as a small script within a script. It's a small chunk of code which you may call multiple times within your script. They are particularly useful if you have certain tasks which need to be performed several times. Instead of writing out the same code over and over you may write it once in a function then call that function every time.

STRUCTURE

```
function_name () {
<commands>
}
```

=

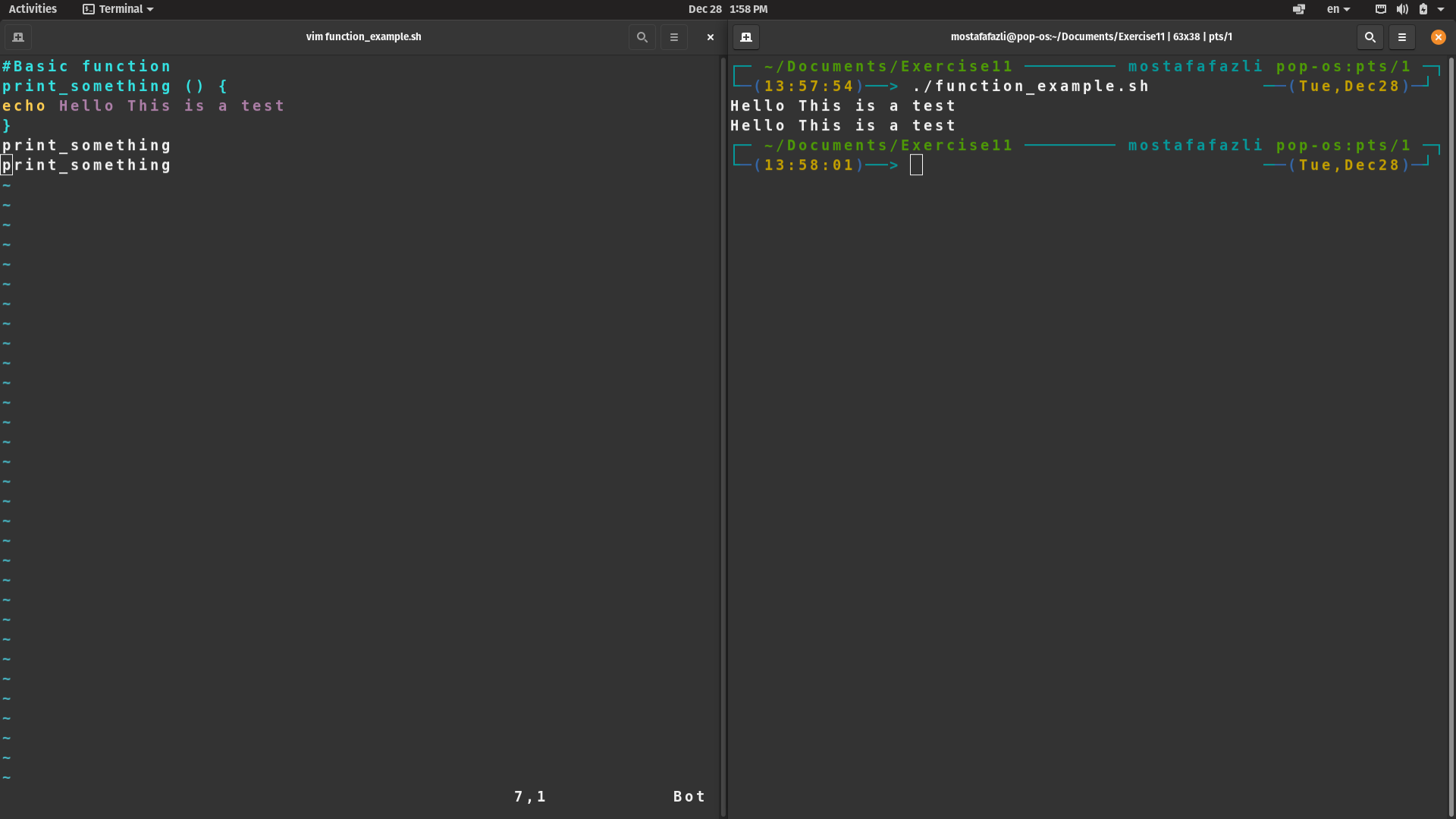
```
function function_name {
<commands>
}
```

EXAMPLE

```
1.#!/bin/bash
2.# Basic function
3.print_something () {
4.echo Hello This is a test
5.}
6.print_something
7.print_something
```

RUN

```
1../function_example.sh
2.Hello I am a function
3.Hello I am a function
```


Activities
Terminal
Dec 28 1:58 PM
en
vim function_example.sh
#Basic function
print_something () {
echo Hello This is a test
}
print_something
print_something
7,1 Bot
mostafafazli@pop-os:~/Documents/Exercise11 | 63x38 | pts/1
~/Documents/Exercise11 mostafafazli pop-os:pts/1
(13:57:54)—> ./function_example.sh (Tue,Dec28)
Hello This is a test
Hello This is a test
~/Documents/Exercise11 mostafafazli pop-os:pts/1
(13:58:01)—> (Tue,Dec28)

FUNCTIONS

PARSSING ARGUMENT

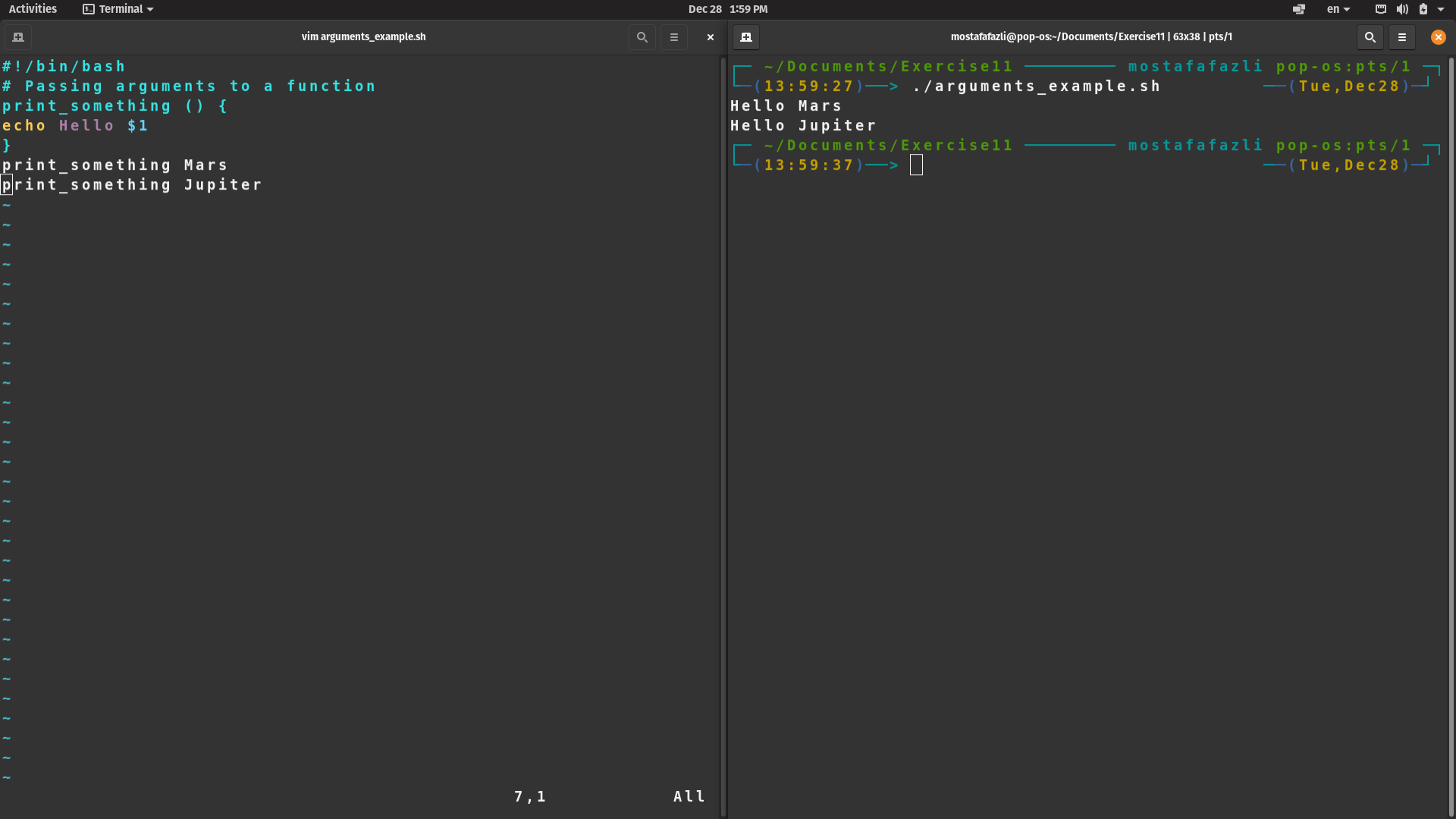
It is often the case that we would like the function to process some data for us.
We may send data to the function in a similar way to passing command line arguments to a script. We supply the arguments directly after the function name.
Within the function they are accessible as $1, $2, etc.

EXAMPLE

```
1.#!/bin/bash
2.# Passing arguments to a function
3.print_something () {
4.echo Hello $1
5.}
6.print_something Mars
7.print_something Jupiter
```

RUN

```
1../arguments_example.sh
2.Hello Mars
3.Hello Jupiter
```


Activities
Terminal
Dec 28 1:59 PM
en
vim arguments_example.sh
#!/bin/bash
# Passing arguments to a function
print_something () {
echo Hello $1
}
print_something Mars
print_something Jupiter
7,1 All
mostafafazli@pop-os:~/Documents/Exercise11 | 63x38 | pts/1
~/Documents/Exercise11 mostafafazli pop-os:pts/1
(13:59:27)--> ./arguments_example.sh (Tue,Dec28)
Hello Mars
Hello Jupiter
~/Documents/Exercise11 mostafafazli pop-os:pts/1
(13:59:37)-->

FUNCTIONS

RETURN VALUES

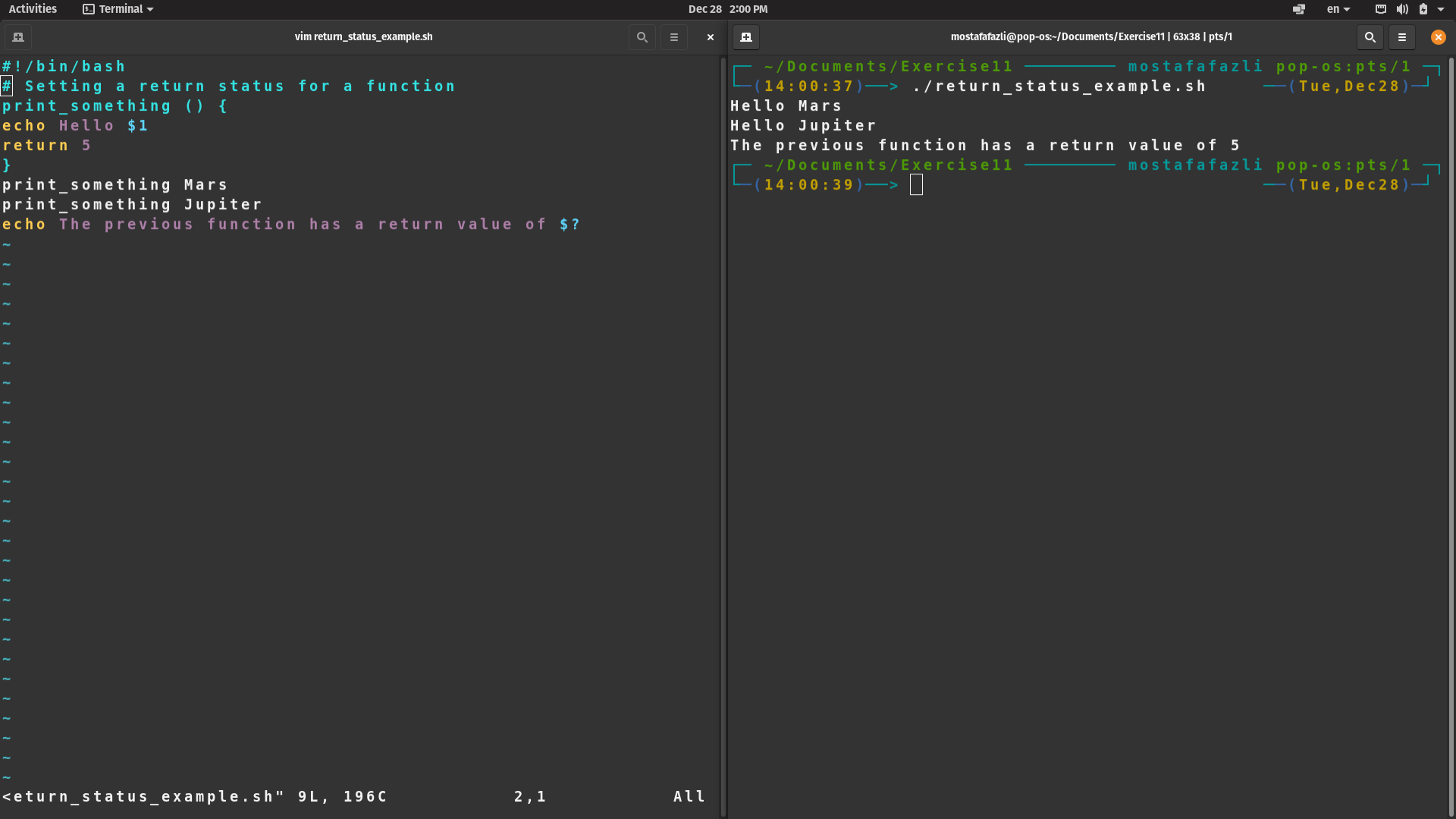
Most other programming languages have the concept of a return value for functions, a means for the function to send data back to the original calling location. Bash functions don't allow us to do this. They do however allow us to set a return status. Similar to how a program or command exits with an exit status which indicates whether it succeeded or not. We use the keyword return to indicate a return status.

EXAMPLE

```
1.#!/bin/bash
2.# Setting a return status for a function
3.print_something () {
4.echo Hello $1
5.return 5
6.}
7.print_something Mars
8.print_something Jupiter
9.echo The previous function has a return value of $?
```

RUN

```
1../return_status_example.sh
2.Hello Mars
3.Hello Jupiter
4.The previous function has a return value of 5
```


Activities
Terminal
Dec 28 2:00 PM
en
vim return_status_example.sh
#!/bin/bash
# Setting a return status for a function
print_something () {
echo Hello $1
return 5
}
print_something Mars
print_something Jupiter
echo The previous function has a return value of $?
<eturn_status_example.sh" 9L, 196C 2,1 All
mostafafazli@pop-os:~/Documents/Exercise11 | 63x38 | pts/1
~/Documents/Exercise11 mostafafazli pop-os:pts/1
(14:00:37)--> ./return_status_example.sh (Tue,Dec28)
Hello Mars
Hello Jupiter
The previous function has a return value of 5
~/Documents/Exercise11 mostafafazli pop-os:pts/1
(14:00:39)--> (Tue,Dec28)

Typically a return status of 0 indicates that everything went successfully. A non zero value indicates an error occurred.

TIPS

FUNCTIONS

RETURN VALUES

You can get value of last function return by $?

EXAMPLE

```
1.#!/bin/bash
2.# Setting a return status for a function
3.sum(){
1.    return ‘expr $1 + $2’
4.}
5.sum 67 89
6.total=$?
7.echo $total
```

RUN

```
1../return_status_example.sh
2.156
```

FUNCTIONS

PROCESS MULTIPLICATION

For Multiplication two numbers, you should use ‘x’ instead of ‘*’

Because of ‘*’(star) symbol for address, when you use ‘*’, Bash consider address for that.

EXAMPLE

```
1.#!/bin/bash
2.# multiplication of two numbers
3.calc(){
4.    case $2 in
5.     ‘+’)result=‘expr $1 + $3’;;
6.     ‘-’)result=‘expr $1 - $3’;;
7.     ‘/’)result=‘expr $1 / $3’;;
8.     ‘x’)result= $(($1 * $3));;
9.sum 167 x  89
10.total = $?
11.echo $total
```

RUN

```
1../multiplication_example.sh
2.14863
```

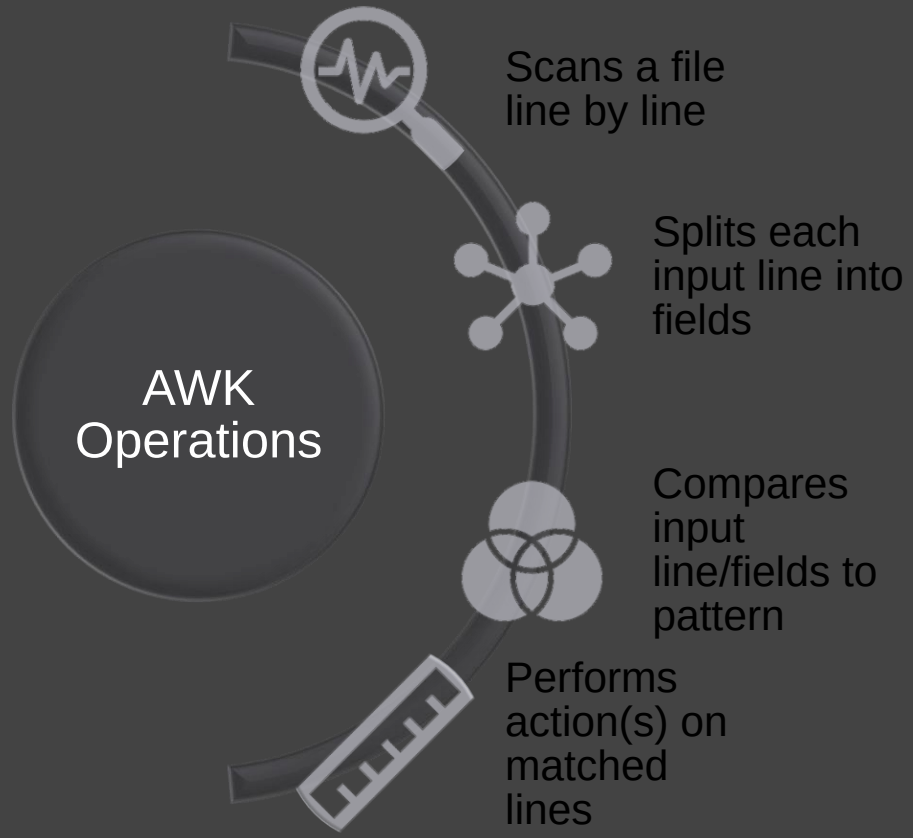

awk

AWK

Awk is a scripting language used for manipulating data and generating reports. The awk command programming language requires **no compiling** and allows the user to use variables, numeric functions, string functions, and logical operators.

Awk is a utility that enables a programmer to write tiny but effective programs in the form of statements that define text patterns that are to be searched for in each line of a document and the action that is to be taken when a match is found within a line. Awk is mostly used for **pattern scanning** and **processing**. It searches one or more files to see if they contain lines that matches with the specified patterns and then perform the associated actions.

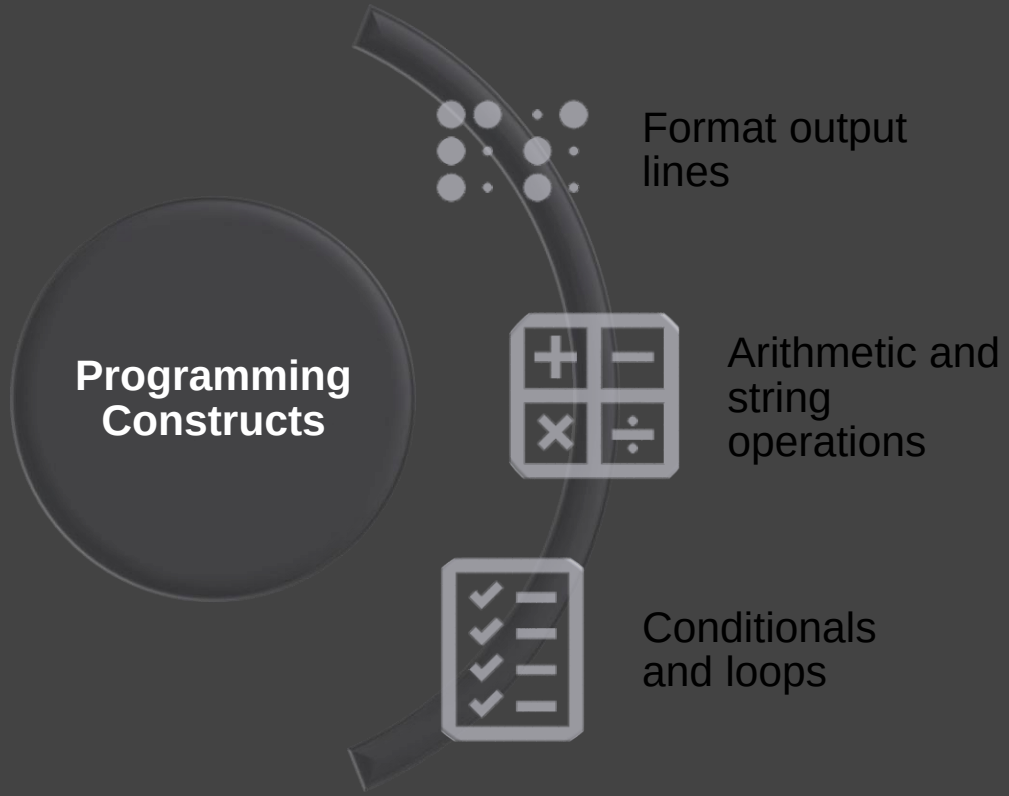
The awk command was named using the initials of the three people who wrote the original version in 1977: Alfred Aho, Peter Weinberger, and Brian Kernighan. These three men were from the legendary AT&T Bell Laboratories Unix pantheon.


AWK
Operations
Scans a file line by line
Splits each input line into fields
Compares input line/fields to pattern
Performs action(s) on matched lines

Useful For

Transform data files

Produce formatted reports


Programming Constructs
Format output lines
Arithmetic and string operations
Conditionals and loops

AWK

STRUCTURE

awk options 'selection _criteria {action }' input-file > output-file

AWK

OPTIONS:

-f program-file => Reads the AWK program source from the file program-file, instead of from the first command line argument.

-F => Use fs for the input field separator

$cat > employee.txt

ajay manager account 45000

sunil clerk account 25000

varun manager sales 50000

amit manager account 47000

tarun peon sales 15000

deepak clerk sales 23000

sunil peon sales 13000

satvik director purchase 80000

AWK

OPTIONS:

'{print}'

Use print by default just prints every lines from file

* it works like cat command

$ awk '{print}' employee.txt

ajay manager account 45000

sunil clerk account 25000

varun manager sales 50000

amit manager account 47000

tarun peon sales 15000

deepak clerk sales 23000

sunil peon sales 13000

satvik director purchase 80000

AWK

OPTIONS:

Print the lines which match the given pattern

the awk command prints all the line which matches with the ‘PATTERN’

$ awk '/manager/ {print}' employee.txt

ajay manager account 45000

varun manager sales 50000

amit manager account 47000

AWK

OPTIONS:

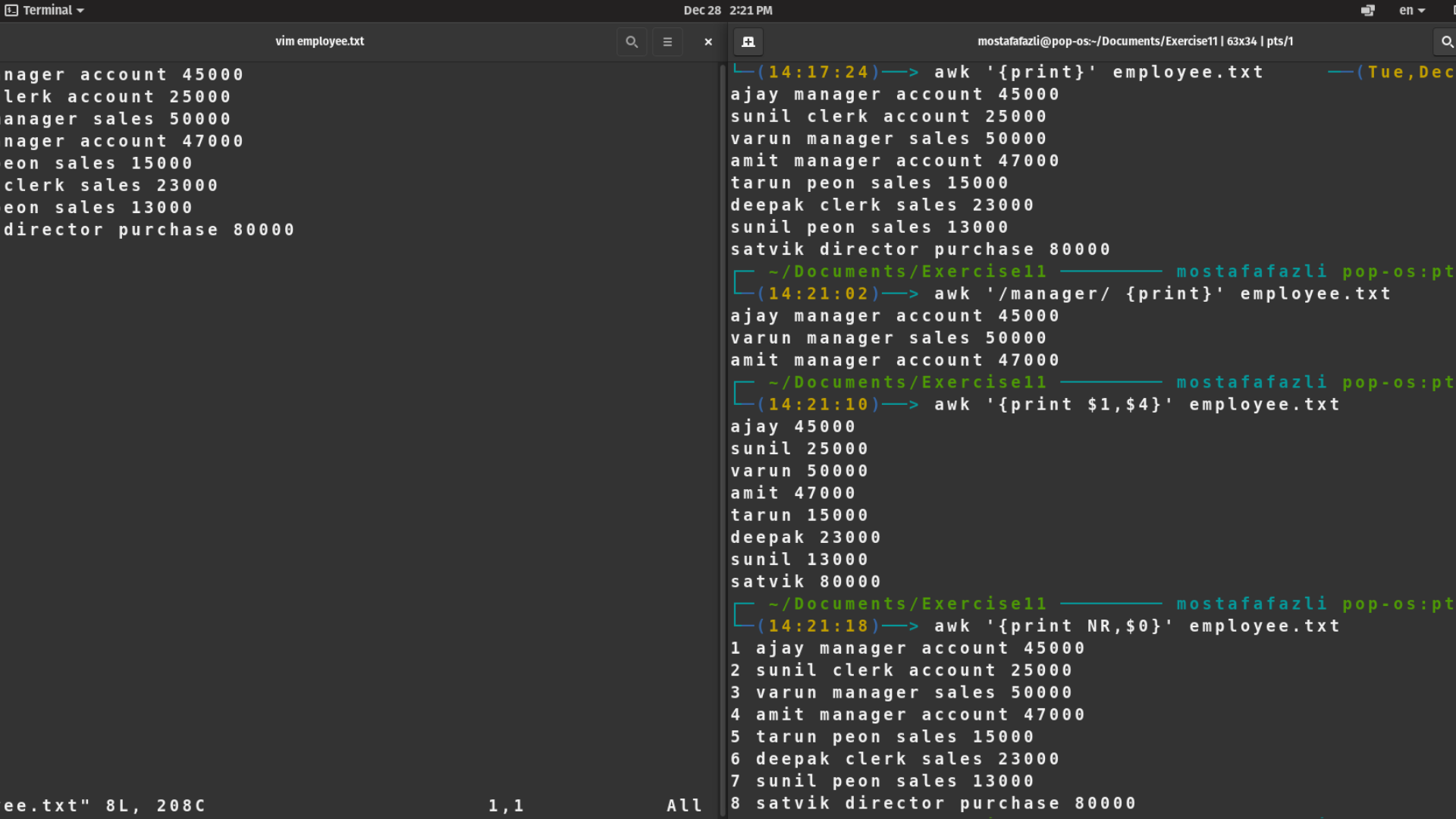
Splitting a Line Into Fields

awk command splits the record delimited by whitespace character by default and stores it in the $n variables. If the line has 4 words, it will be stored in $1, $2, $3 and $4 respectively.

*Also, $0 represents the whole line.

```
$ awk '{print $1,$4}' employee.txt
```

```
ajay 45000
sunil 25000
varun 50000
amit 47000
tarun 15000
deepak 23000
sunil 13000
satvik 80000
```


Terminal
Dec 28 2:21 PM
en
vim employee.txt
nager account 45000
lerk account 25000
anager sales 50000
nager account 47000
eon sales 15000
clerk sales 23000
eon sales 13000
director purchase 80000
ee.txt" 8L, 208C
1,1
All
mostafafazli@pop-os:~/Documents/Exercise11 | 63x34 | pts/1
(14:17:24)──> awk '{print}' employee.txt
ajay manager account 45000
sunil clerk account 25000
varun manager sales 50000
amit manager account 47000
tarun peon sales 15000
deepak clerk sales 23000
sunil peon sales 13000
satvik director purchase 80000
~/Documents/Exercise11 mostafafazli pop-os:pt
(14:21:02)──> awk '/manager/ {print}' employee.txt
ajay manager account 45000
varun manager sales 50000
amit manager account 47000
~/Documents/Exercise11 mostafafazli pop-os:pt
(14:21:10)──> awk '{print $1,$4}' employee.txt
ajay 45000
sunil 25000
varun 50000
amit 47000
tarun 15000
deepak 23000
sunil 13000
satvik 80000
~/Documents/Exercise11 mostafafazli pop-os:pt
(14:21:18)──> awk '{print NR,$0}' employee.txt
1 ajay manager account 45000
2 sunil clerk account 25000
3 varun manager sales 50000
4 amit manager account 47000
5 tarun peon sales 15000
6 deepak clerk sales 23000
7 sunil peon sales 13000
8 satvik director purchase 80000

AWK

OPTIONS:

NR

Use the NR command for Display line number

$ awk '{print NR,$0}' employee.txt

1 ajay manager account 45000

2 sunil clerk account 25000

3 varun manager sales 50000

4 amit manager account 47000

5 tarun peon sales 15000

6 deepak clerk sales 23000

7 sunil peon sales 13000

8 satvik director purchase 80000

AWK

OPTIONS:

NF

Use the NF command for Display last field

*NF stands for Number of Fields

$ awk '{print $1,$NF}' employee.txt

ajay 45000

sunil 25000

varun 50000

amit 47000

tarun 15000

deepak 23000

sunil 13000

satvik 80000

AWK

OPTIONS:

NR (between)

Another use of NR built-in variables (Display Line From 3 to 6)

$ awk 'NR==3, NR==6 {print NR,$0}' employee.txt

3 varun manager sales 50000

4 amit manager account 47000

5 tarun peon sales 15000

6 deepak clerk sales 23000

AWK

OPTIONS:

Advanced commands

To print the first item along with the row number(NR) separated with " – " from each line

$cat > test.txt

A B C

Tarun A12 1

Man B6 2

Praveen M42 3

```
$ awk '{print NR "- " $1 }' test.txt
```

1 - A

2 - Tarun

3 – Manav

4 - Praveen

AWK

OPTIONS:

To return the second row/item

$ awk '{print $2}' test.txt

A12

B6

M42

AWK

OPTIONS:

find the length of the longest line present

```
$ awk '{ if (length($0) > max) max = length($0)
} END { print max }' test.txt
```

```
13
```

AWK

OPTIONS:

To count the lines

```
$ awk 'END { print NR }' test.txt
```

```
3
```

AWK

OPTIONS:

combination commands

You can combination other command with this command by pipeline

```
who | awk '{print $1}'
```

```
mosfazli
knight
lenovo_gaming
```

```
cuments/Exercise11 ──────────────────────────────────────────────── mostafafazli pop-os:pt
2:10)──> who | awk '{print $1}'                                          ──(Tue,Dec
fazli
cuments/Exercise11 ──────────────────────────────────────────────── mostafafazli pop-os:pt
2:14)──>
                                                                         ──(Tue,Dec
```

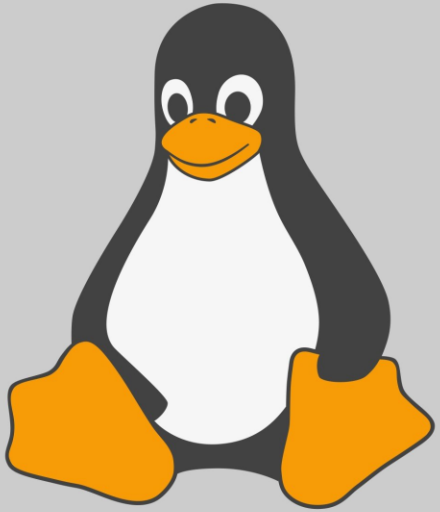
AWK

OPTIONS:

You can combination other command with this command by pipeline with some rules

```
who | awk '{print $1,$4}'
```

```
mosfazli 15:22
knight 12:11
lenovo_gaming 01:23
```



LINUX