

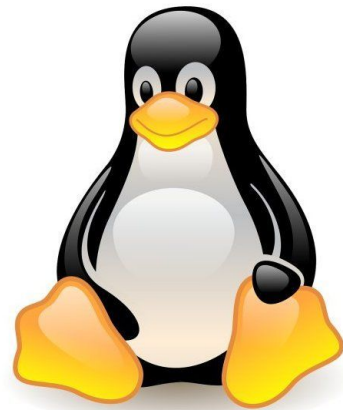


# Linux Plus

for

# AWS and DevOps

## Session 4





# Using Filter





# Table of Contents

- ▶ **stdin, stdout, stderr**
- ▶ **Filters**
- ▶ **Commands:**
  - cat, tee, grep, cut, tr, wc, sort, uniq, comm
- ▶ **Control Operators**



# 1 `stdin, stdout, stderr`



# stdin, stdout, stderr



stdin, stdout, and stderr are three standard streams that are established when a Linux command is executed.

Data streams, like water streams, have two ends. They have a source and an outflow. Whichever Linux command you're using provides one end of each stream. The other end is determined by the shell that launched the command. That end will be connected to the terminal window, connected to a pipe, or redirected to a file or other command, according to the command line that launched the command.

**stdin** – It stands for standard input, and is used for taking text as an input.

**stdout** – It stands for standard output, and is used to text output of any command you type in the terminal, and then that output is stored in the stdout stream.

**stderr** – It stands for standard error. It is invoked whenever a command faces an error, then that error message gets stored in this data stream.



2

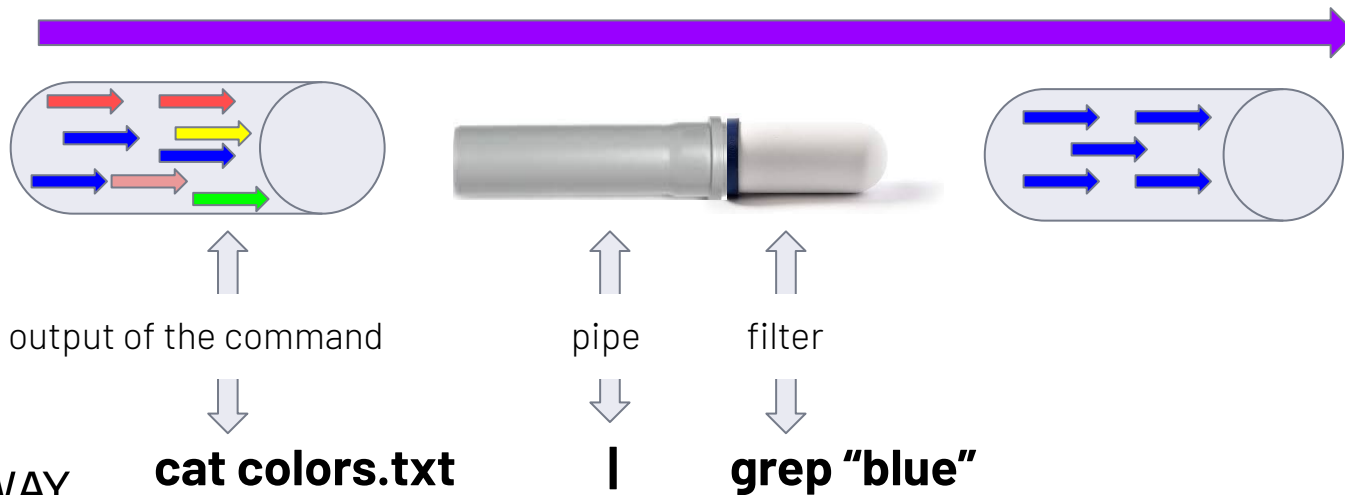
# Filters



# Filters

A filter is a program that takes data from one command, does some processing and gives output. Filter commands generally are used with a **pipe**.

**Pipe ('|')** is a mechanism that send the output of one command as input of another command.

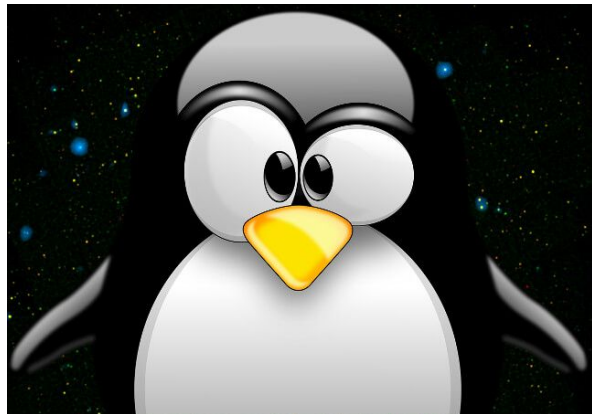




## ► Commands

3

- **cat**
- **tee**
- **grep**
- **cut**
- **tr**
- **wc**
- **sort**
- **uniq**
- **comm**







# Filters Commands

grep

The most common use of grep is to filter lines of text containing (or not containing) a certain string.

```
ubuntu@clarusway:~$ cat tennis.txt
Amelie Mauresmo, Fra
Justine Henin, BEL
Serena Williams, USA
Venus Williams, USA
ubuntu@clarusway:~$ cat tennis.txt | grep Williams
Serena Williams, USA
Venus Williams, USA
ubuntu@clarusway:~$
```



# Filters Commands

cut

The cut filter can select columns from files, depending on a delimiter or a count of bytes

**cut -d(delimiter) -f(columnNumber) <fileName>**

```
[ec2-user@ip-172-31-47-50 filters]$ cat tennis.txt
Amelie Mauresmo, Fra
Justine Henin, BEL
Serena Williams, USA
Venus Williams, USA
[ec2-user@ip-172-31-47-50 filters]$ cat tennis.txt | cut -d' ' -f1
Amelie
Justine
Serena
Venus
[ec2-user@ip-172-31-47-50 filters]$ cat tennis.txt | cut -d' ' -f2
Mauresmo,
Henin,
Williams,
Williams,
[ec2-user@ip-172-31-47-50 filters]$ cat tennis.txt | cut -d' ' -f3
Fra
BEL
USA
USA
```



# Filters Commands

tr

The command 'tr' stands for 'translate'.

It is used to translate, like from lowercase to uppercase and vice versa or new lines into spaces.

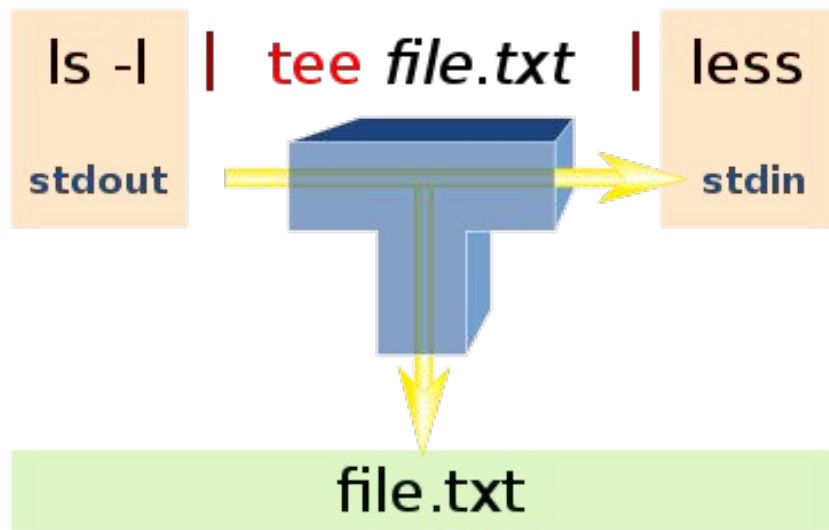
```
ubuntu@clarusway:~$ cat clarusway.txt
Way to Reinvent Yourself
ubuntu@clarusway:~$ cat clarusway.txt | tr 'ae' 'io'
Wiy to Roinvont Youusolf
ubuntu@clarusway:~$ cat count.txt
one
two
three
four
five
ubuntu@clarusway:~$ cat count.txt | tr '\n' ' '
one two three four five ubuntu@clarusway:~$
```



# Filters Commands

tee

tee is almost the same as cat, except that it has two identical outputs.



tee - read from standard input and write to standard output and files



# Filters Commands

wc

Counting words, lines and characters is easy with wc. It counts the number of newlines, words, characters, and bytes in text files.

- `wc <fileName>` (Counts words, lines and characters)
- `wc -l <fileName>` (Counts only lines)
- `wc -w <fileName>` (Counts only words)
- `wc -c <fileName>` (Counts only characters)

```
ubuntu@clarusway:~$ cat count.txt
one
two
three
four
five
ubuntu@clarusway:~$ wc count.txt
 5  5 24 count.txt
ubuntu@clarusway:~$ wc -l count.txt
5 count.txt
ubuntu@clarusway:~$ wc -w count.txt
5 count.txt
ubuntu@clarusway:~$ wc -c count.txt
24 count.txt
ubuntu@clarusway:~$
```



# Filters Commands

sort

The sort filter will default to an alphabetical sort.

sort -r	the flag returns the results in reverse order
sort -f	the flag does case insensitive sorting

```
ubuntu@clarusway:~$ cat marks.txt
John-10
James-9
Aaron-8
Oliver-7
Walter-6
ubuntu@clarusway:~$ sort marks.txt
Aaron-8
James-9
John-10
Oliver-7
Walter-6
ubuntu@clarusway:~$
```



# Filters Commands

uniq

With the help of uniq command you can form a **sorted list** in which every word will occur only once.

**sort -u = the unique findings**

```
ubuntu@clarusway:~$ cat marks.txt
John
James
Aaron
Oliver
Walter
Aaron
John
James
John
John

ubuntu@clarusway:~$ sort marks.txt | uniq

Aaron
James
John
Oliver
Walter
ubuntu@clarusway:~$
```



# Filters Commands

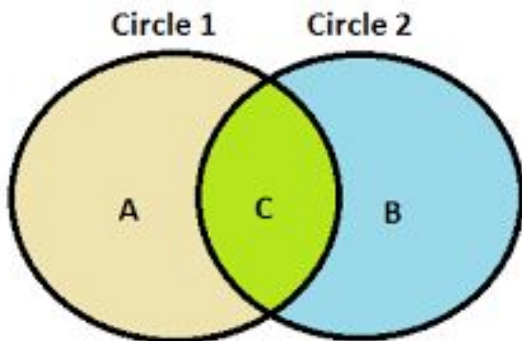
comm

The 'comm' command compares two files or streams.

By default, 'comm' will always display three columns.

First column indicates non-matching items of first file, second column indicates non-matching items of second file, and third column indicates matching items of both the files.

Both the files has to be in sorted order for 'comm' command to be executed.



```
ubuntu@clarusway:~$ cat file1
Aaron
James
John
Oliver
Walter
ubuntu@clarusway:~$ cat file2
Guile
James
John
Raymond
ubuntu@clarusway:~$ comm file1 file2
Aaron
      Guile
          James
          John
Oliver
      Raymond
Walter
ubuntu@clarusway:~$
```





# Homework Exercise 1

1. Create a file named countries.csv with the following content

```
Country,Capital,Continent
USA,Washington,North America
France,Paris,Europe
Canada,Ottawa,North America
Germany,Berlin,Europe
```

2.
  - a. Cut only “Continent” column
  - b. Remove header
  - c. Sort the output
  - d. List distinct values
  - e. Save final output to “continents.txt” file
3. Display content of continents.txt file





4

# Using Control Operators



# Table of Contents

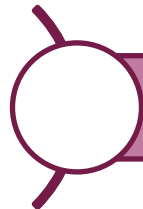


## ► Control Operators

- Semicolon (;)
- Ampersand (&)
- Dollar Question Mark (\$?)
- Double Ampersand (&&)
- Double Vertical Bar (||)
- Combining && and ||
- Pound Sign (#)
- Escaping Special Characters (\)
- End of line Backslash



# Control Operators



We put more than one command on the command line using control operators.

Control Operator	Usage
; semicolon	More than one command can be used in a single line.
& ampersand	Command ends with & (and) doesn't wait for the command to finish.
\$? dollar question mark	Used to store exit code of the previous command.
&& double ampersand	Used as logical AND.
 double vertical bar	Used as logical OR.
Combining && and	Used to write if then else structure in the command line.
# pound sign	Anything was written after # will be ignored.



# Semicolon (;)

You can put two or more commands on the same line separated by a **semicolon (;)**

```
ubuntu@clarusway: $ cat days.txt
sunday
monday
tuesday
wednesday
thursday
friday
saturday
ubuntu@clarusway: $ cat count.txt
one
two
three
four
five
ubuntu@clarusway: $ cat days.txt ; cat count.txt
sunday
monday
tuesday
wednesday
thursday
friday
saturday
one
two
three
four
five
ubuntu@clarusway: $
```



# Amperсанд (&)

When a line ends with an ampersand &, the shell will not wait for the command to finish. You will get your shell prompt back, and the command is executed in background. You will get a message when this command has finished executing in background.

```
ubuntu@clarusway:~$ sleep 20 &
[1] 3396
ubuntu@clarusway:~$
[1]+  Done                  sleep 20
ubuntu@clarusway:~$
```

- Look at the above snapshot, command "**sleep 20 &**" has displayed a message after 15 seconds.
- Meanwhile, in the shell prompt, we can write any other command.

# Dollar Question Mark (\$?)



This control operator is used to check the status of last executed command. If status shows '0' then command was successfully executed, if shows '1' then command was a failure and '127' means command not found error.

```
ubuntu@clarusway:~$ ls
Repo          count.txt    days.txt    file1.txt    file2.txt    key.txt      keypc.pem   marks.txt   tennis.txt
clarusway.txt days          file1       file2        get-docker.sh key.txt.pub  keypc.pub   temp.txt    testdir
ubuntu@clarusway:~$
ubuntu@clarusway:~$ echo $?
0
ubuntu@clarusway:~$ rmdir *
rmdir: failed to remove 'Repo': Directory not empty
rmdir: failed to remove 'clarusway.txt': Not a directory
rmdir: failed to remove 'count.txt': Not a directory
rmdir: failed to remove 'days': Not a directory
rmdir: failed to remove 'days.txt': Not a directory
rmdir: failed to remove 'file1': Not a directory
rmdir: failed to remove 'file1.txt': Not a directory
rmdir: failed to remove 'file2': Not a directory
rmdir: failed to remove 'file2.txt': Not a directory
rmdir: failed to remove 'get-docker.sh': Not a directory
rmdir: failed to remove 'key.txt': Not a directory
rmdir: failed to remove 'key.txt.pub': Not a directory
rmdir: failed to remove 'keypc.pem': Not a directory
rmdir: failed to remove 'keypc.pub': Not a directory
rmdir: failed to remove 'marks.txt': Not a directory
rmdir: failed to remove 'temp.txt': Not a directory
rmdir: failed to remove 'tennis.txt': Not a directory
rmdir: failed to remove 'testdir': Directory not empty
ubuntu@clarusway:~$ echo $?
1
ubuntu@clarusway:~$
```



# Double Ampersand (&&)

The command shell interprets the && as the logical AND. When using this command, the second command will be executed only when the first one has been successfully executed.

```
ubuntu@clarusway:~$ cat days.txt && cat count.txt
sunday
monday
tuesday
wednesday
thursday
friday
saturday
one
two
three
four
five
ubuntu@clarusway:~$ cd Repo && ls
m
ubuntu@clarusway:~/Repo$
```





# Double Vertical Bar (||)

The command shell interprets the (||) as the logical OR. This is opposite of logical AND. Means second command will execute only when first command will be a failure.

```
ubuntu@clarusway: $ cat days.txt || echo "clarusway" ; echo one
sunday
monday
tuesday
wednesday
thursday
friday
saturday
one
ubuntu@clarusway: $ zecho days.txt || echo "clarusway" ; echo one

Command 'zecho' not found, did you mean:

  command 'aecho' from deb netatalk
  command 'echo' from deb coreutils

Try: sudo apt install <deb name>

clarusway
one
ubuntu@clarusway: $
```



# Combining && and ||

You can use this logical AND and logical OR to write an if-then-else structure on the command line. This example uses echo to display whether the rm command was successful.

```
ubuntu@clarusway: $ cat file1
Aaron
James
John
Oliver
Walter
ubuntu@clarusway: $ rm file1 && echo It worked! || echo It failed!
It worked!
ubuntu@clarusway: $ rm file1 && echo It worked! || echo It failed!
rm: cannot remove 'file1': No such file or directory
It failed!
ubuntu@clarusway: $
```



# Pound Sign (#)

Everything written after a pound sign (#) is ignored by the shell. This is useful to write a shell comment but has no influence on the command execution or shell expansion.

```
ubuntu@clarusway:~$ mkdir test           # We create a directory
ubuntu@clarusway:~$ cd test              # We enter the directory
ubuntu@clarusway:~/test$ ls              # is it empty ?
ubuntu@clarusway:~/test$
```

# Escaping Special Characters (\)

Escaping characters are used to enable the use of control characters in the shell expansion but without interpreting it by the shell.

```
ubuntu@clarusway:~$ echo this is \* symbol.  
this is * symbol.  
ubuntu@clarusway:~$ echo this \ \ \ \is \ \ \ \clarusway.  
this   is   clarusway.  
ubuntu@clarusway:~$ echo escaping \\\ \#\ \&\ |"\ |'  
escaping \ # & " '  
ubuntu@clarusway:~$
```

# End of Line Backslash (\)



Lines ending in a backslash are continued on the next line. The shell does not interpret the newline character and will wait on shell expansion and execution of the command line until a newline without backslash is encountered.

```
ubuntu@clarusway:~$ echo This command line \  
> > is split in three \  
> > parts  
ubuntu@clarusway:~$ This command line is split in three parts
```



# Homework Exercise 2

1.   a. Search for “clarusway.txt” in the current directory  
      b. If it exists display its content  
      c. If it does not exist print message “Too early!”
2.   Create a file named “clarusway.txt” that contains “Congratulations”
3.   Repeat Step 1



Students, write your response!



# Homework Exercise 3

## diff

diff (1)            - compare files line by line  
diff (1p)          - compare two files

<https://www.geeksforgeeks.org/diff-command-linux-examples/#:~:text=diff%20stands%20for%20difference.,make%20the%20two%20files%20identical.>

<https://www.linuxtechi.com/diff-command-examples-linux/>

# Other Linux Tools



- ▶ **sed**
- ▶ **awk**







# sed command

SED command in UNIX stands for stream editor and it can perform lots of functions on file like searching, find and replace, insertion or deletion.

Though most common use of SED command in UNIX is for substitution or for find and replace.

By using SED you can edit files even without opening them, which is much quicker way to find and replace something in file, than first opening that file in VI Editor and then changing it.

SED is a powerful text stream editor. Can do insertion, deletion, search and replace(substitution). SED command in unix supports regular expression which allows it perform complex pattern matching.

## Syntax:

```
sed OPTIONS... [SCRIPT] [INPUTFILE...]
```



# awk command

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.

Awk is abbreviated from the names of the developers – Aho, Weinberger, and Kernighan.

## Syntax:

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



# THANKS!

## Any questions?

You can find me at:

- ▶ @sumod
- ▶ sumod@clarusway.com

