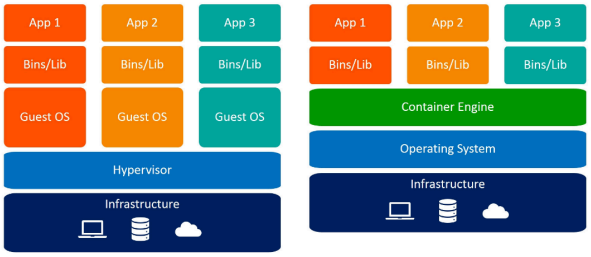
warning, a lot of stuff is skipped because deemed obvious for a linux user. For example:
*Unix-like command syntax* `command [options] [arguments]`*. The name of the command is associated to the action performed.*

# Linux installation

Skipped

# Docker

Open-source technology that permits the execution of applications quickly and efficiently by mean of Containers.
Containers share the kernel of the OS, the embedded depenedencies and configurations of the application. While VMs include the application, dependences, lireries and the complete OS.
So Docker offers all the advantages of VM but requiring fewer requirements from the host OS.




Virtual Machines Containers

The advantages are:

- Isolation
- Portability
- Agility (less requirements in terms of resources)
- Scalability
- Packetization

# Commands

list containers

```
sudo docker container ls -a
```

run create container:

```
sudo docker start <dockername>
```

stop running docker:

```
sudo docker stop <dockername>
```

remove a container:

```
sudo docker rm <dockername>
```

# SHH

Remote connection

```
ssh -X <username@hostname>
```

-X is to redirect grafical content, of course can be omitted

## Help commands

```
man <command> #manual page
apropos <term> #  searchesthe description lines from the top of the related man pages that include term
whatis <command> #gives smmary of what the command is used for
whereis <command> #gives path of command
```

You have to run mandb when you first install the package.

```
~ apropos halt
halt (8)              - Power off, reboot, or halt the machine
poweroff (8)          - Power off, reboot, or halt the machine
...

~whatis ifconfig
ifconfig (8)          - configure a network interface

~ whereis ifconfig
ifconfig: /usr/bin/ifconfig /usr/share/man/man8/ifconfig.8.gz
```

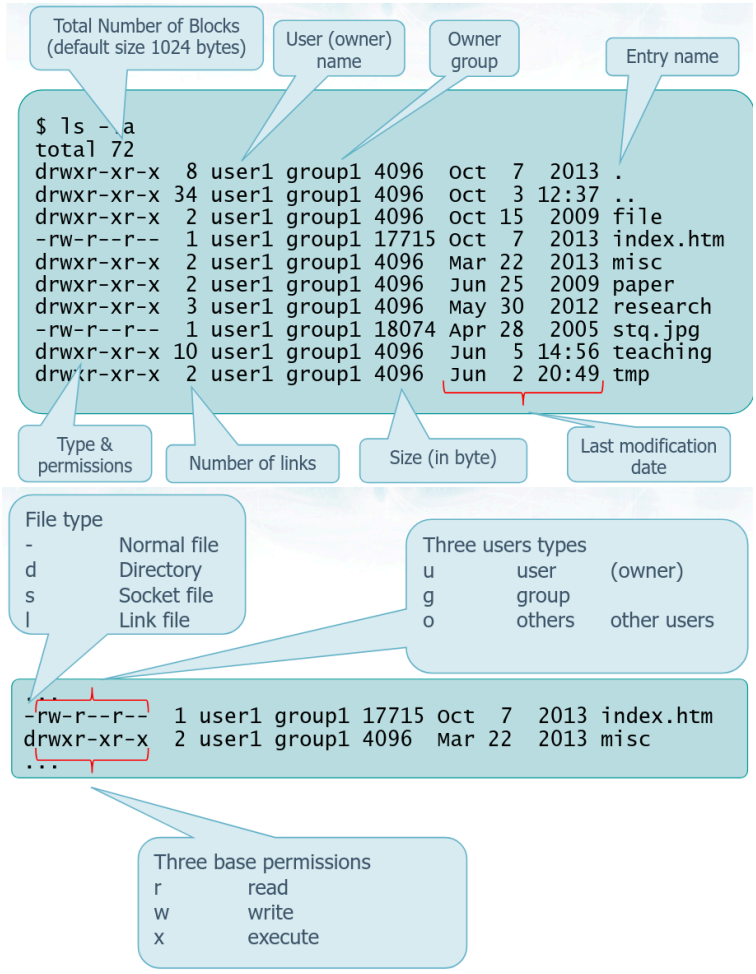
## Command parsing

You can use '\' to continue a long command on the next line.
Two commands can be written on the same line using ';'

## ls

```
ls [-options] [file ...]
```

-a shows hidden files
-l long list format
-t sort by date
-r reverse order
-R recursive, includes files in subdir




-|rwx|rwx|rwx
0 1 2 3
0: - normal file, d directory, s socket file, l symbolic link
1: user permissions
2: group permissions
3: other's permissions ($other = totalUsers - user - group$)

## Files move, copy, remove

```
mv [options] file1 file2 ... dest
```

```
cp [options] src1 src2 ... dest
```

```
rm [options] file1 file2 …
```

Options:
-f force, doenst ask confirmation
-i ask confirmation for each file
-r -R recursively to all sub dir

## Directory move, cpy, remove

```
mkdir dir
```

`rmdir dir` //same as `rm -fr dir`

## Permission for directories

There exist 3 permissions:
r, w, x
read, write, execute

```
cp file1 file2
```

fails if file1 has not read permissions or if file2 has not write permissions

```
cd dir
```

fails if dir has not execution permissions

# Permission management

It is possible to change file permissions if you have the rights, i.e., if you are the owner of the file.

## User

There are commands to change personal generalities (i.e., the user) of files on a UNIX system:
To become a different user:
`su username`
The password of the new user is requested
To run commands as a superuser (or other user):
`sudo command`
The password of the user is required
To know which user you are:
`whoami`

## Files

It is possible to change file permissions

```
chmod [options] permissions file
```

Examples:

```
chmod g+r filename
```

```
chmod +x filename
```

```
chmod +xw filename
```

```
chmod uo+rx filename #user other add read execute. Adds to user and others read and execute
permissions to that file.
```

The owner can always change the permissions of the file.

Can also use this system:

| | r | w | x |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 |
| 2 | 0 | 1 | 0 |
| 3 | 0 | 1 | 1 |
| 4 | 1 | 0 | 0 |
| 5 | 1 | 0 | 1 |

| | r | w | x |
|---|---|---|---|
| 6 | 1 | 1 | 0 |
| 7 | 1 | 1 | 1 |

So if i want
rwx|r-x|---
750

```
chmod 750 file0.txt
```

rw-|r-w|-w-
652

```
chmod 652 file1.txt
```

```
chmod go-w #removes write permission from group and other
```

```
polycore% ls -l
total 0
-rw-r--r-- 1 gandalf gandalf 0 Sep 29 10:55 file_con_permessi.txt
polycore% chmod 777 file_con_permessi.txt
polycore% ls -l
total 0
-rwxrwxrwx 1 gandalf gandalf 0 Sep 29 10:55 file_con_permessi.txt
polycore% chmod u-w file_con_permessi.txt
polycore% ls -l
total 0
-r-xrwxrwx 1 gandalf gandalf 0 Sep 29 10:55 file_con_permessi.txt
```

## Directory

Changing the owner of a directory entry
`bash chown [options] user entry`
Changing the group of a directory entry

```
chgrp [options] group entry
```

```
chown [options] user[:group] entry #it means that it gives the permission to user and to group, so you write `chown -options gabriele:teachers file.txt`
```

```
chown [options] uid[:gid] entry
```

Options:
-R recursive

# Output the content of a file

Output and concatenate files:
`cat filename1 filename2 ]`
Output the first n lines of a file
`head [options] filename ...` //default is 10, can also remove lines from print by doing -3, doenst print last 3
Output the last num lines of a file
`tail [options] filename ...`

Options:
-n num, number of lines
-f, outputs appended data as the file grows (i.e., the file
is continuously re-read)

browse page-wise through text file:
`pg [options] filename ...`
to view a text file:
`more [options] filename ...`
Like the previous command but allows the use of arrows to move in the text (advanced version of more):
`less [options] filename ...`

`more` and `less` can be nagivated with the following inputs:

- space: Next page
- return: Next line
- b: Previous page
- /str: Find next occurrence of string str
- ?str: Find previous occurrence of string str
- q: Quit

# Counts

`wc filename`
prints:
number_lines, number_word, number_char

options:
-l prints only n lines
-w prints only n words
-c prints only n byts
-m prints only number of characters

It always outputs the name of the file as the first line

# File comparison

```
diff [options] file1 file2
```

lists the line number of the lines added, deleted, changed

```
diff [options] dir1 dir2
```

OPTIONS:
-q reports only when files differ
-b Ignores spaces at the end of the line, merges the others
-i case insensitive
-w ignores completely al white spaces
-B ingnores all blank lines

# Hard and soft links

Symbolic or soft link:

- Particular type of file that simply contains a path (i.e., the name) of another object (file or directory)
- Allows references between different file-systems (partitions)
- If you remove the file the link remains pending

Physical or hard link

- Association between an object name and its content (pointer from directory-entry to i-node)
- It is not possible to create hard links between different file-systems, or hard links to a directory
- The file is removed only when it is removed the last of its hard links

## Link creation

```
ln [options] souce [destination]
```

Default:
Creates a hard link. If the destination is not present, creates a link with the same filename on the working director.

Options:
-s creates a symbolic link
-f forces creation, removes file if already exist
-f, -F allow the superuser to attempt to create a hard link to a directory

# Archive management

Data storage and compression can be managed using the `tar` command
**Archiving**:

```
tar czvf <file>.tgz <dir>
```

Options:
-c
-z,j,J compression gzip, bzip2, 7z
-v vebose
-f specify the name of the archive

**Extracting**

```
tar xzvf <file>.tgz <dir>
```

-x Extracts the files from the archive
-z, j, J Compression (gzip, bzip2, 7z)
-v Verbose (print some messages and statistics)
f Specify the name of the archive (always present)

# Disk space occupation

To control disk occupancy, it is possible to use the df command

```
df [options] [disk ...]
```

OPTIONS:
-B SIZE, scale sizes by SIZE before printing them. Size can be 1K, 10K, 1M, 1G, 1T etc.
--block-size=SIZE
-k correpsonds to --blocksize=1k

To control direcotry occupation

```
du [options] directory
```

Options:
-a Occupation of each file
--summarize, -s Prints only the total
--block-size=1K, -k Occupation in kB

# pushd

pushd is like cd, but after setting a path it reCan also use this system:mebers it, so if i run pushd, i jump to that directory.

# Running time

to check how long a programm takes to run write

```
time program
```

if i have a compiled c program:

```
time compiled_c_program
```

it will print:
`./compiled_c_program 0.01s user 0.06s system 2% cpu 2.626 total`

# GCC

gcc is an open source compile and linker. It supports C and C++.

```
gcc <options> <arguments>
```

Compilation of a set of files that produces the corresponding object files

```
gcc -c file1.c
gcc -c file2.c
gcc -c main.c
```

Link of the object files produces the executable file:

```
gcc -o myexe file1.o file2.o main.o
```

Compilation and linking with a single command:

```
gcc -o myexe file1.c file2.c main.c
```

Options:
-c file Compilation only
-o file Specifies the executable name; generally indicates
the name of the final executable (after the link
operation)
-g gcc does not produce optimized code, but inserts additional information useful for debugging (see gdb)
-Wall Output a warning for all possible code errors
-Idir Specify further directories where searching header files. More than one directory can be specified (-Idir1 - Idir2 …)
-lm Specifies to use the math library
-Ldir Specifies the search directories for pre-existing libraries to be linked

# Makefile

Makefile is a scripting language to automatically perform repetitive tasks and avoid redoing unnecessary tasks. An example would be only re-compiling files that have been modified since the previous command.
It is made up of two utilities: `makefile` and `make`.

Two phases:

1. Write a Makefile file
   A text file similar to a script
2. The Makefile file is intepreted with the `make` utility
   This way you can compile and link]

Options for `make`:

- `-n` Does not execute, just displays the commands
- `-` i Ignores possible errors and proceeds with the next commands
- `-f <fileName>`

- `-d` Output debug information during the execution
- `--debug=[options]` Options:
  - a = print all info
  - b = basic info
  - v = verbose = basic + other
  - i = implicit = verbose + other

The `make` command executes by default the file "makefile" if it exists, if that doesnt exist it tries "Makefile". It can also take a file as input parameter.

Structure:

```
target: dependency
     command
```

empty lines and comments # are ignored. Each rule specifies a target, some dependencies, and actions; it can occupy one or more lines. Very long lines can be splitted by inserting "\". Don't forget the <tab>.

The default behavior is to execute the first rule, but if target are specified, that one is executed.

```
make <targetName>
make -f <myMakefile> <targetName>
```

Each rule includes

- Target Name
  - Usually the name of a file
  - Sometimes the name of an action (which is named "phony" target)
- dependency list that must be verified to execute the target
- Command, or list of commands
  - Each command is preceded by a mandatory TAB character, invisible but necessary

Examples:

```
target:
     gcc -Wall -g -o pgrm pgrm.c -lm

clean:
     rm -rf pgrm
     rm -rf *~
```

There exist very powerful rules for improving modularity and make more efficient the writing of makefiles

- Use of macros
- Use of implicit rules
  - The dependence between .o and .c is automatic
  - The dependence between .c and .h is automatic
  - Recursive dependencies are analyzed automatically
  - etc.

```
CC = gcc
#CC = g++
FLAGS = -Wall -g
includedir = ./dirI
INCLUDEDIR = -I. -I$(includedir)
LIB = -lm

.PHONY: clean distclean

target: mainVet.o inVet.o outVet.o
```

```
#    gcc -Wall -g -o myExe mainVet.o inVet.o outVet.o
     $(CC) $(FLAGS) $(INCLUDEDIR) -o myExe mainVet.o inVet.o outVet.o $(LIB)


mainVet.o: mainVet.c my.h
     $(CC) $(FLAGS) $(INCLUDEDIR) -c mainVet.c $(LIB)

inVet.o: inVet.c my.h
     $(CC) $(FLAGS) $(INCLUDEDIR) -c inVet.c $(LIB)

outVet.o: outVet.c my.h
     $(CC) $(FLAGS) $(INCLUDEDIR) -c outVet.c $(LIB)

clean:
     rm -rf *~
     rm -rf *bak*
     rm -rf core

distclean: clean
     rm -rf myExe
     rm -rf *.o
```

File with dependencies

```
target: file1.o file2.o
     gcc -Wall -o myExe file1.o file2.o
file1.o: file1.c myLib1.h
     gcc -Wall -g -I./dirI -c file1.c
file2.o: file2.c myLib1.h myLIb2.h
     gcc -Wall -g -I./dirI -c file2.c
```

Execution of multiple targets in the presence of dependencies

- It checks if target dependencies are more recent than the current target
- In this case, dependencies are performed before the execution of the current target
- This process iterates recursively

If the target is not a file name, it is a "phony" target that should always be executed
To be sure that is always executed, as in maybe someone creates a file with that name use: `.PHONY : target`

Definition of macro: macro=name (with or without spaces).
Use of the macro: `$(macro)`.

```
CC=gcc
FLAGCS=-Wall -g
SRC=main.c bst.c list.c util.c

project: $(SRC)
     $(CC) $(FLAGS)-o project $(SRC) -lm
```

```
CC=gcc
FLAGS=-Wall -g
SDIR=source
HDIR=header
ODIR=obj
project: $(ODIR)/main.o $(ODIR)/bst.o
     $(CC) $(FLAGS)-o $@ $^
$(ODIR)/main.o: $(SDIR)/main.c $(HDIR)/main.h
     $(CC) $(FLAGS) -c $^
$(ODIR)/bst.o: $(SDIR)/bst.c $(HDIR)/bst.h
     $(CC) $(FLAGS) -c $^
```

- `$@`: Represents the Target name. In the first rule, `$@` becomes `project`.

- `$^`: Represents all the dependencies. In the first rule, it expands to `$(ODIR)/main.o$(ODIR)/bst.o`.
- `$<`: (Not in your snippet, but good to know) Represents only the *first* dependency.

# GDB

Software package used to analyze the behavior of another program in order to identify and eliminate errors (bugs).
It can be used

- As a "stand-alone" tool
  - Particularly inconvenient use
- Integrated with many editors (e.g., emacs)
- Embedded in some graphical IDE

| **Action** | **Command** |
|---|---|
| Execution commands | run (r)<br>next (n)<br>next <NumberOfSteps><br>step (s)<br>step <NumberOfSteps><br>stepi (si)<br>finish (f)<br>continue (c) |
| Breakpoint commands | info break<br>break (b), ctrl-x-blank<br>break LineNumber<br>break FunctionName<br>break fileName:LineNumber<br>disable BreakpointNumber<br>enable BreakpointNumber |
| Print commands | print (p)<br>print expression<br>display expression |
| Print commands | down (d)<br>up (u)<br>Info args<br>Info locals |
| Code listing commands | list (l)<br>list LineNumber<br>list FirstLine, LastLine |
| Miscellaneous commands | file fileName<br>exec filename<br>kill |

# Regular expressions (RE)

A regular expression (or pattern) is an expression that specifies a set of strings:
Compact operators are used to represent complex sequences of characters. Example:
`a | b*` represents the set of strings {a, φ, b, bb, bbb, bbbb, …}

Expressions are useful to find if a match exists between objects.

| **Operator** | **Meaning** |
|---|---|
| `[…]` | Specifies a list or range of symbols |
| `(…)` | Manages operator precedence; groups sub-expressions; allows reference to previous expressions (backward reference) |
| `\|` | Logical OR between regular expressions |

| Anchor | Meaning |
|---|---|
| `\<` | Beginning of word |
| `\>` | End of word |
| `^` | Beginning of line |
| `$` | End of line |

| Special characters | Meaning |
|---|---|
| `\+` `\?` `\.` | Characters `+`, `?`, `.` |
| `\n` | New line |
| `\t` | TAB |

| Quantifiers and ranges | Meaning |
|---|---|
| `*` | `[0, ∞]` times |
| `+` | `[1, ∞]` times |
| `?` | `[0, 1]` times |
| `[c1c2c3]` | Any character in brackets |
| `[c1-c5]` | Any character in range |
| `[^c1-c5]` | Any character **not** in range |
| `{n}` | Exactly `n` times |
| `{n1,n2}` | `n1` to `n2` times |

| Characters | Meaning |
|---|---|
| `c` | Any symbol `c` (excluding special ones) |
| `.` | Any character (excluding `\n`) |
| `\c` | Any control character |
| `\s` | A space or TAB |
| `\d` | A digit `[0-9]` |
| `\D` | Not a digit `[^0-9]` |
| `\w` | Letters, numbers and `_` `[0-9A-Za-z_]` |
| `\W` | Not letters, numbers and `_` `[^0-9A-Za-z_]` |

| Regular expression | Meaning |
|---|---|
| `ABCDEF` | String `"ABCDEF"` |
| `a*b` | Any number of `a` followed by a single `b` |
| `ab?` | `a` or `ab` |
| `a{5,15}` | 5 to 15 repetitions of `a` |
| `(fred){3,9}` | 3 to 9 repetitions of string `"fred"` |
| `.+` | Any non-empty sequence (entire line) |
| `myfunc.*(.*)` | A function with name beginning with `"myfunc"` |
| `^ABC.*` | A line beginning with `"ABC"` |
| `.*h$` | A line ending with `"h"` |
| `hello\>` | Word ending with `"hello"` |
| `a+b+` | One or more `a` followed by one or more `b` |
| | |
| `.*b.*3` | Matches string `"./fbar3"` |
| `[a-zA-Z0-9]` | A letter or a digit |
| `A\|b` | `A` or `b` |

| Regular expression | Meaning |
|---|---|
| `\w{8}` | An 8-character word |
| `((4\.[0-2])\|(2\.[0-2]))` | Numbers `4.0`, `4.1`, `4.2` or `2.0`, `2.1`, `2.2` |
| `(.)\1` | Two times the same character (e.g. `"aa"`) |
| `(.)(.).\2\1` | Any 5-character palindrome string (e.g. `radar`, `civic`, `12321`) |

# find

Allows for searching and listing the file, directories or links that match a given criterion. It possibly executes a shell command on every listed file.
Notice that it outputs the relative path of the matching files, not their basenames. This is important to write the REs to match a give path and the actions to be performed.

`find directory options actions`

In details, the find command

1. Visits the directory subtree
2. Outputs the list of pathnames satisfying the options
3. Possibly performs the actions on every file of the list

## `directory`

Specifies the search directory tree in which
execute the command

```
.
/usr/bin
./subDirA/subDirB
```

## `options`

| Option | Meaning |
|---|---|
| `-name pattern` | Match with the file name. The initial path (basename) is removed. In some versions, you can place the pattern in double quotes to specify a regular expression. |
| `-iname pattern` | Same as `-name`, but case insensitive. |
| `-path pattern` | Like `-name`, but you need to specify the full path + filename. |
| `-ipath pattern` | Same as `-path`, but case insensitive. |
| `-regex expr` | Specifies a regular expression that matches the found relative path (full path) |
| `-iregex` | is equal but case insensitive |
| `-regextype type` | Indicates the type of regular expressions used: posix-basic, posix-egrep, posix-extended, etc. You must specify the type before the regular expression (regextype must precede regex) |
| `-atime [+,-]n` | Last access time. `n=1` specifies from 0 to 24 hours back. `n` value with sign: `+` means ≤, `-` means ≥ |
| `-ctime [+,-]n` | Last status change time. `n=1` specifies from 0 to 24 hours back. `n` value with sign: `+` means ≤, `-` means ≥ |
| `-mtime [+,-]n` | Last modification time. `n=1` specifies from 0 to 24 hours back. `n` value with sign: `+` means ≤, `-` means ≥ |
| `-size [+,-]n[bckwMG]` | File dimension. Sign `+` means ≥, `-` means ≤. Next character indicates the size: `b` blocks (of 512 bytes), `c` bytes, `k` kbytes, `w` word (2 bytes), `M` Mbytes, `G` Gbytes |
| `-type type` | File type: `f` per regular file (i.e., text files, executable, etc.), `d` for directories, `p` for pipes, `l` for symbolic links, `s` for sockets |
| `-user name` | File owner identifier (user name) |
| `-group name` | File group identifier (group name) |
| `-readable` | File access permissions |

| Option | Meaning |
|---|---|
| `-writable` | File access permissions |
| `-executable` | File access permissions |
| `-mindepth n` | Search limited to a subtree section: mindepth and maxdepth indicate the minimum and maximum depth from directory (-maxdepth 1 means search only on directory) |
| `-maxdepth n` | Search limited to a subtree section: mindepth and maxdepth indicate the minimum and maximum depth from directory (-maxdepth 1 means search only on directory) |
| `-quit` | Quit the search after the first match |

```
find . -name "*.c" #File with ".c" extension
find . -regex ".*\.c"  #RE equivalent
find /usr/bin -iname "a.*"
find . -size +500c #All files with dimension >500 bytes
find . -readable \
-regex "\./a+b.*\..*" #All readable files in the current directory (.) with name beginning by ab, aab, aaab,etc., and any extension
```

### `actions`

| Option | Meaning |
|---|---|
| `-print` | Default action. Print a name for each line |
| `-fprint` | Like the previous one, but it performs the output on a file |
| `-print0` | Like `-print`, but without going in the next line |
| `-execdir command` | Executes command |
| `-exec command` | Secure POSIX version of the previous one. Expands the command by including the path and the name |
| `-delete` | Deletes the founded filename |

The execution of a command is performed by means of the option `exec` or `execdir`:

```
find directory options -exec command '{}' ';'
find directory options -exec command \{} \;
```

Where:

- the command is executed in the directory:
  - in which the pathname has been found if you use `execdir`
  - in the current directory if you use `exec`
- find substitutes string '{}' (or \{}) with the current pathname of the list
- String ';' (or \;) terminates the command executed by find

Examples:

```
root -exec cat \{} \; #Outputs the content of each file of the root filesystem belonging to user root, and concatenates their content
find / -user root -exec cat '{}' >> file.txt ';' #same but ridirected into file
find . -name "*.txt" -exec head -n 2 \{} \; #outputs first two lines of each txt file
```

## Filters

In UNIX/Linux a filter is a command that

- Takes its input from standard input
- Process (filters) it according to some parameters and options

- Produces its output on standard output

Most popular filters
`awk`, `cat`, `cut`, `compress`, `grep`, `head`, `perl`, `sed`, `sort`, `tail`, `tr`, `uniq`, `wc`

Some of these commands

- Are quite complex
  - grep, sort
- Are scripting languages
  - sed, awk
- Often
  - Use regular expressions
  - Are used in pipe (by means of the `|` operator) with other commands

## `cut`

Selects and outputs sections from each line of files.

```
cut [options] file
```

Main options:

- `--characters=LIST`, `-c LIST`
  - Select only the characters at the positions in LIST
- `--fields=LIST`, `-f LIST`
- Selects the (comma separated) list of fields
  - Format
    - n (=n), -n (≤n), n- (≥n), n1-n2 (≥n1 && ≤n2)
    - Examples: 3, -3, 3-, 3-5
- `--delimiter=DELIM`, `-d DELIM`
  - Uses DELIM to separate fields rather than the default <tab> delimiter.

Examples:

```
cut -f 1,3 file.txt #Selects fields 1 and 3 of all file lines
cut -f 1-3,5-6 -d " " foo.txt #Selects fields 1 to 3 (1, 2 and 3) and 5 to 6 of file foo.txt fields
are delimited by space rather than by TAB
```

## `tr`

Translate, compress, and/or delete characters from standard input, writing to standard output. Must be used by redirecting the output of other commands to its input.

```
tr [options] set1 [set2]
```

Main options

- `--delete`, `-d`
  - Delete the characters in set1
- `--squeeze-repeats`, `-s`
  - replace each input sequence of a repeated character that is listed in set1 with a single occurrence of that character
- `--complement`, `-c`, `-C`
  - Uses the complement of set1

Examples:

```
tr -d abcd < file.txt #Outputs file.txt eliminating  characters a, b, c, d
cat file.txt | tr ab BA #Outputs the lines of file.txt, in which 'a' is substituted with 'B', and 'b'
```

```
is substituted with 'a'
echo ciao | tr ia IA #Outputs cIAo
echo "a b        c" | tr -s ' ' #squeezes (compresses) the sequence of spaces to a single space.
Outputs a b c
```

## `uniq`

Report or omit repeated lines of the input file. The file must be sorted! Without options eliminates the repeated lines.

```
uniq [options] [inFile] [outFile]
```

Main options

- `--count`, `-c`
  - prefix lines by the number of occurrences
- `--repeated`, `-d`
  - only print duplicate lines, one for each group
- `--skip-fields=N`, `-f N`
  - avoid comparing the first N fields
- `--ignore-case`, `-I`
  - Case insensitive

Examples:

```
uniq -c file.txt #Eliminates duplicate lines of a sorted text file outputs the others and inserts the
number of occurrences of the duplicated lines

uniq -d a.x #Outputs the duplicated lines only
```

## `basename`

Eliminates the directories from a pathname and possibly its extensions

```
basename pathname [extension]
```

Example:

```
> basename /home/user1/current/file.txt
file.txt
> basename /home/user1/current/file.txt ".txt"
file
> basename /home/user1/current/file.txt .txt
file
> basename /home/user1/current/file.txt txt
file.
```

## `sort`

Sort the input file in alphabetic order
`sort [options] [file]`
Main options

- `--ignore-leading-blanks`, `-b`
- Ignore the initial spaces
- `--dictionary-order`, `-d`
- Considers spaces and alphabetic characters only
- `--ignore-case`, `-f`
- Transforms lowercase characters in uppercase characters (Case insensitive)
- `--numeric-sort`, `-n`

- Sort in numeric order
- `--reverse`, `-r`
- Sort in reverse order
- `--key=c1`,`[,c2]`, `-k c1[,c2]`
- Sort on the basis of the selected fields
- `--merge`, `-m`
- Merges sorted files, no sort is performed without
  other options
- `--output=f`, `-o=f`
- Writes its output on file f rather than on standard output

Examples

```
sort file.txt #Sorts the lines of the file file.txt interpreting them as a sequence of ASCII
characters
cat file1.txt file2.txt | \
sort -r –k 1,3 -f #Concatenates files file1.txt and file2.txt, and sorts the rows of the two files in
descending order using the fields 1, 2 and 3 and ignoring the difference between uppercase and
lowercase letters
```

## `grep`

Global Regular Expression Print. Searches the input files for lines containing a
match to the given pattern. If no files are specified, or if the file "-" is given, grep searches standard input. By default, grep prints the matching lines.
Versions
 grep
- Standard version
 egrep, fgrep, rgrep
- equivalent to `grep –E`, `grep -F`, `grep -R`
- Uses Extended RE for matching the pattern

```
grep [options] pattern [file]
```

Main options

- `--regexp=PATTERN`, `-e PATTERN`
  - Specifies the search patterns
  - Allows you to specify multiple patterns
- `--line-number`, `-n`
  - Outputs the matching line number
- `--recursive`, `-r`, `-R`
  - Search recursively the sub-trees
- `--inverse-match`, `-v`
  - Outputs only the lines that do not match
- `--ignore-case`, `-I`
  - Case insensitive
- `--after-context=N`, `-A N`
  - Outputs N lines after each match line (in addition to the line in which the match was found)
- `--before-context=N`, `-B N`
  - Outputs N lines before each match line (in addition to the line in which the match was found)
- `--with-filename`, `-H`
  - Outputs the filename for each matching line

```
grep abc file.txt #Outputs the file lines that include the string "abc"
grep –e "l." –e a file.txt #Outputs the file lines that include character 'l' followed by any other
character, or include character 'a'
```

```
grep –H –A 4 abc file.txt #Outputs the file lines that include string "abc", and the next 4 lines, preceded by the filename
```