Bash Lecture 2 - Advanced

Bibliography and learning materials

★ Bibliography:

https://www.rigacci.org/docs/biblio/online/sysadmin/toc.htm

https://www.tldp.org/LDP/abs/html/

★ Learning Materials:

http://www.ee.surrey.ac.uk/Teaching/Unix/

https://github.com/gtaffoni/Learn-Python/blob/master/Lectures/ShellLecture01.pdf

https://github.com/gtaffoni/Learn-Python/blob/master/Lectures/ShellLecture02.pdf

https://github.com/bertocco/bash_lectures

Arguments of this lesson

★ Bash scripting programming:
  - Editors
  - Scripts and redirection
  - Variables
  - Main programming elements (if, for while,…)
  - Examples (using files)
  - Basic `sed`
  - Basic `awk`

Shell scripting abilities

Many shells have scripting abilities:

Put multiple commands in a script and the shell executes them as if they were typed from the keyboard.

Most shells offer additional programming constructs that extend the scripting feature into a programming language.

Editors

Editor

- ★In dictionary.cambridge.org: is a piece of software for editing text on a computer
- ★In www.merriam-webster.com: is a computer program that permits the user to create or modify data (such as text or graphics) especially on a display screen

Editor types

In Linux, text editor are of two kinds:

- ★ graphical user interface (GUI) based
  - gedit
  - bluefish
  - lime ………..
- ★ command line text editors (console or terminal)
  - nano
  - pico
  - vi/vim
  - emacs ……..

Nano

Nano is the built-in basic text editor for many popular linux distros.
It doesn't take any learning or getting used to, and all its commands and prompts are displayed at the bottom.

- ★ Use Nano if:
  - You're new to the terminal
  - you just need to get into a file for a quick change.

Compared to more advanced editors in the hands of someone who knows what they're doing, some tasks are cumbersome and non-customizable.

- ★ How to Nano:

from your terminal, enter `nano` and the filename you want to edit.
If the file doesn't already exist, it will once you save it.
Commands are listed across the bottom and are triggered with the Control (CTRL) key. For example, to find something in your file, hold CTRL and press W, tell it what you're searching for, and press Enter.
Press CTRL+X to exit, then follow the prompts at the bottom of the screen.

GNU emacs

Emacs has so many available features like a terminal, calculator, calendar, email client, web browser, and Tetris, it's often spoken of as an operating system itself.
Starting Emacs is relatively simple, but more you learn, the more there is to learn.

- ★ How to Emacs:

Emacs commands are accessed through keyboard combinations of CTRL or ALT and another keystroke. When you see shortcuts that read C-h or M-x, C stands for the control key and M stands for the Alt key (or Escape, depending on your system).

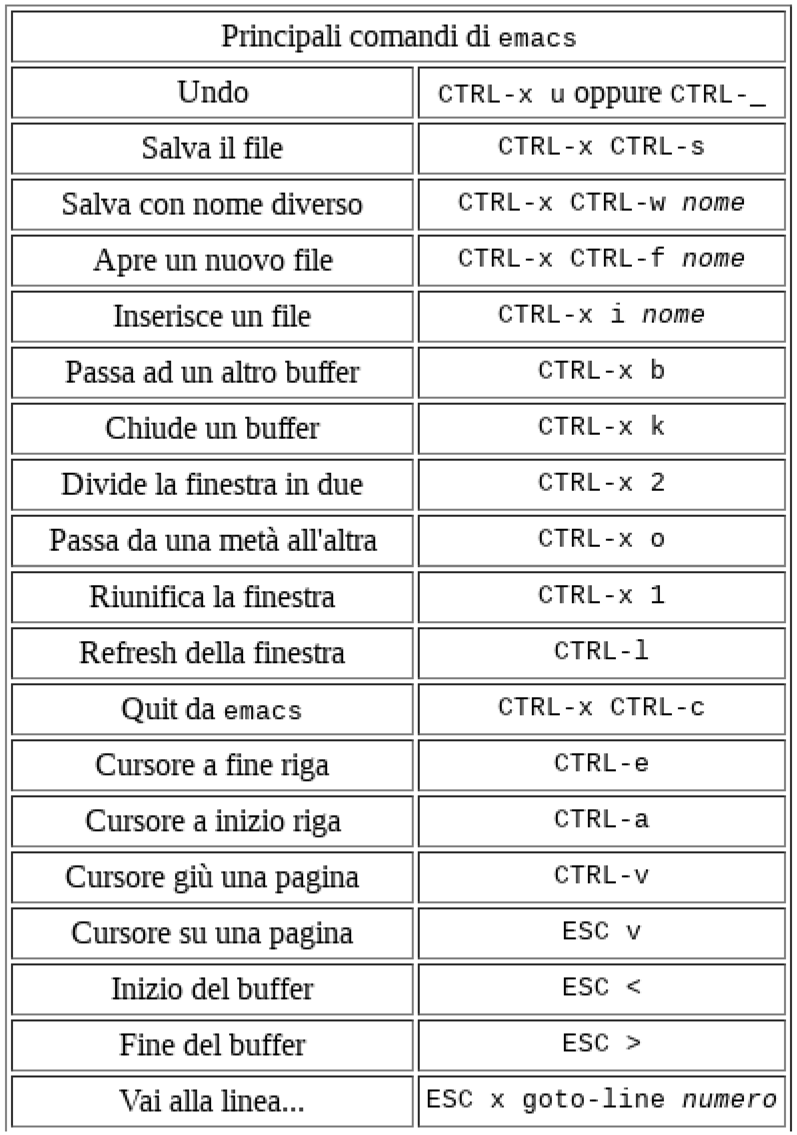
Enter `emacs` in your terminal, and access the built-in tutorial with C-h t. That means, while holding CTRL, press H, then T.

Or, try key combination C-h r to open the manual within Emacs. You can also use the manual as a playground; just remember to quit without saving by pressing key combination C-x C-c.

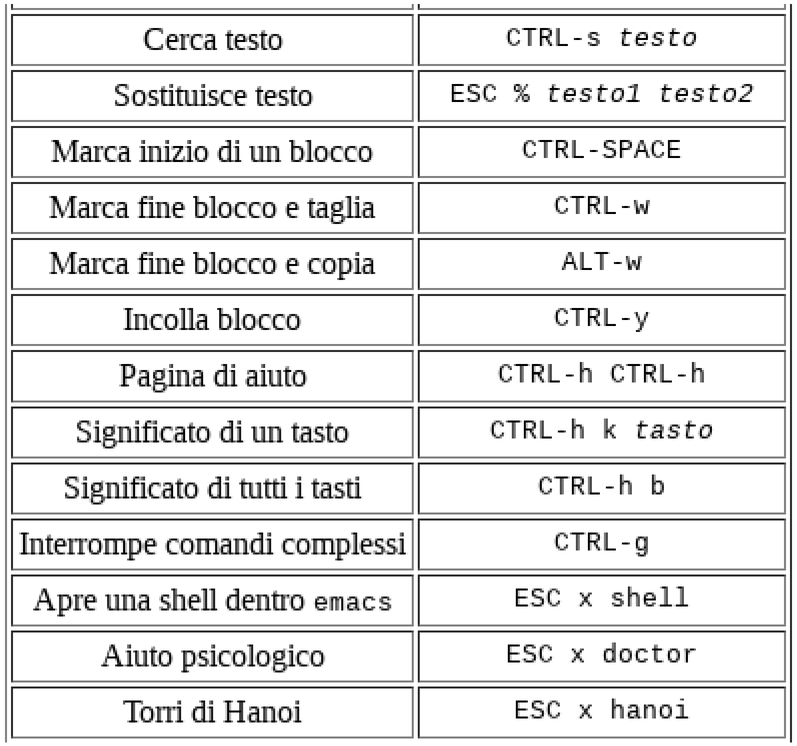
Helpful Emacs links

- ★ Emacs wiki
  https://www.emacswiki.org/emacs/SiteMap
- ★ GNU Guided Tour
  https://www.gnu.org/software/emacs/tour/
- ★ Cornell Emacs Quick Reference
  https://www.cs.cornell.edu/courses/cs312/2006fa/software/quick-emacs.html

Main Emacs commands



Principali comandi di `emacs`	
Undo	`CTRL-x u` oppure `CTRL-_`
Salva il file	`CTRL-x CTRL-s`
Salva con nome diverso	`CTRL-x CTRL-w` *`nome`*
Apre un nuovo file	`CTRL-x CTRL-f` *`nome`*
Inserisce un file	`CTRL-x i` *`nome`*
Passa ad un altro buffer	`CTRL-x b`
Chiude un buffer	`CTRL-x k`
Divide la finestra in due	`CTRL-x 2`
Passa da una metà all'altra	`CTRL-x o`
Riunifica la finestra	`CTRL-x 1`
Refresh della finestra	`CTRL-l`
Quit da `emacs`	`CTRL-x CTRL-c`
Cursore a fine riga	`CTRL-e`
Cursore a inizio riga	`CTRL-a`
Cursore giù una pagina	`CTRL-v`
Cursore su una pagina	`ESC v`
Inizio del buffer	`ESC <`
Fine del buffer	`ESC >`
Vai alla linea...	`ESC x goto-line` *`numero`*



Cerca testo	`CTRL-s` *`testo`*
Sostituisce testo	`ESC %` *`testo1 testo2`*
Marca inizio di un blocco	`CTRL-SPACE`
Marca fine blocco e taglia	`CTRL-w`
Marca fine blocco e copia	`ALT-w`
Incolla blocco	`CTRL-y`
Pagina di aiuto	`CTRL-h CTRL-h`
Significato di un tasto	`CTRL-h k` *`tasto`*
Significato di tutti i tasti	`CTRL-h b`
Interrompe comandi complessi	`CTRL-g`
Apre una shell dentro `emacs`	`ESC x shell`
Aiuto psicologico	`ESC x doctor`
Torri di Hanoi	`ESC x hanoi`

http://www.di.unipi.it/~bozzo/fino/appunti/node2.htmla

vi/vim

Vi, typically comes with your distro-of-choice.
Vim is a vi successor with some improvements. It runs by default on OS X and some Linux distributions when `vi` is run.

VI has two modes of operation (is a “modal” editor):

- Command mode for navigating files: commands which cause action to be taken on the file
- Insert mode for editing text: in which entered text is inserted into the file.

Because Vi is navigated through the use of keyboard commands and shortcuts, it is better experienced than explained.

How to Vi or Vim

Enter `vi` or `vim` in your terminal.
When you enter Vi, you begin in command mode and navigate using keyboard commands and the H, J, K, and L keys to move left, down, up, and right, respectively (but arrows use is possible in the most recent versions).

To enter in editing mode press:
‘a’ to append to the file
‘i’ to insert
pressing the <Esc> (Escape) key turns off the Insert mode.

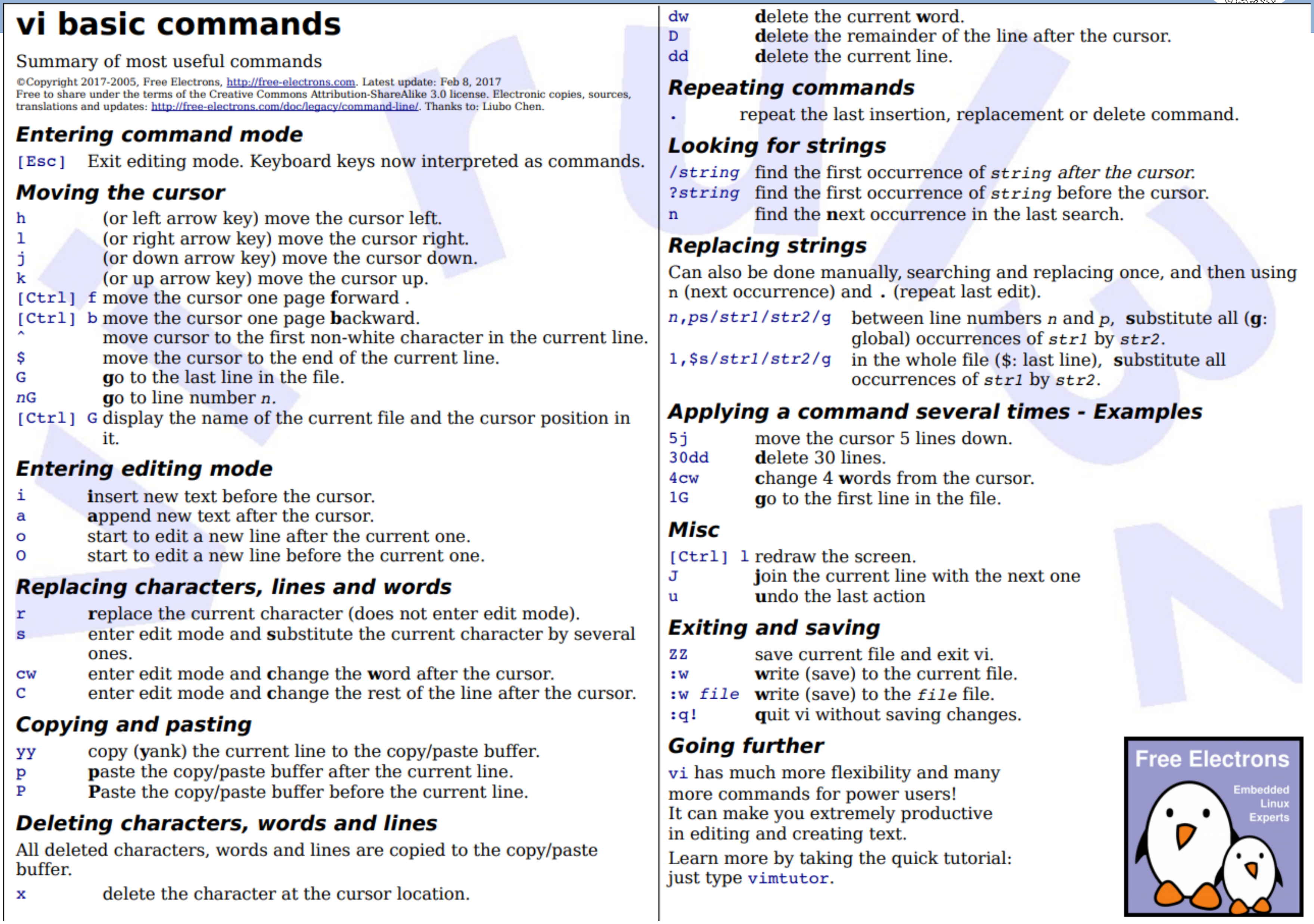
To exit Vim without saving, press ESC to enter command mode, then press : (colon) to access the command line (a prompt appears at the very bottom) and enter q!.
To save and quit, you could use that prompt and the key combination :wq, or hold down SHIFT and press Z two times (the shortcut SHIFT+ZZ).

The : (colon) operator begins many commands like :help for help, or :w to save.

If you’re stuck at the prompt and don’t remember the operator you want to use, enter : (colon), then press CTRL+D for a list of possibilities.

Helpful VI links

- ★ Basic vi Commands
  https://www.cs.colostate.edu/helpdocs/vi.html
- ★ Swathmore's Tips and Tricks
  https://www.cs.swarthmore.edu/oldhelp/vim/home.html
- ★ Linux Academy's Vim Reference Guide
  https://linuxacademy.com/blog/wp-content/uploads/2016/06/vim-2.png



vi basic commands

Summary of most useful commands

Entering command mode

`[Esc]`	Exit editing mode. Keyboard keys now interpreted as commands.

Moving the cursor

`h`	(or left arrow key) move the cursor left.
`l`	(or right arrow key) move the cursor right.
`j`	(or down arrow key) move the cursor down.
`k`	(or up arrow key) move the cursor up.
`[Ctrl] f`	move the cursor one page **f**orward .
`[Ctrl] b`	move the cursor one page **b**ackward.
`^`	move cursor to the first non-white character in the current line.
`$`	move the cursor to the end of the current line.
`G`	**g**o to the last line in the file.
`n``G`	**g**o to line number *n*.
`[Ctrl] G`	display the name of the current file and the cursor position in it.

Entering editing mode

`i`	**i**nsert new text before the cursor.
`a`	**a**ppend new text after the cursor.
`o`	start to edit a new line after the current one.
`O`	start to edit a new line before the current one.

Replacing characters, lines and words

`r`	**r**eplace the current character (does not enter edit mode).
`s`	enter edit mode and **s**ubstitute the current character by several ones.
`cw`	enter edit mode and **c**hange the **w**ord after the cursor.
`C`	enter edit mode and **c**hange the rest of the line after the cursor.

Copying and pasting

`yy`	copy (**y**ank) the current line to the copy/paste buffer.
`p`	**p**aste the copy/paste buffer after the current line.
`P`	**P**aste the copy/paste buffer before the current line.

Deleting characters, words and lines

All deleted characters, words and lines are copied to the copy/paste buffer.

`x`	delete the character at the cursor location.
`dw`	**d**elete the current **w**ord.
`D`	**d**elete the remainder of the line after the cursor.
`dd`	**d**elete the current line.

Repeating commands

`.`	repeat the last insertion, replacement or delete command.

Looking for strings

`/`*`string`*	find the first occurrence of *`string`* *after the cursor.*
`?`*`string`*	find the first occurrence of *`string`* before the cursor.
`n`	find the **n**ext occurrence in the last search.

Replacing strings

Can also be done manually, searching and replacing once, and then using n (next occurrence) and **.** (repeat last edit).

`n,p``s/`*`str1`*`/`*`str2`*`/g`	between line numbers *n* and *p*, **s**ubstitute all (**g**: global) occurrences of *`str1`* by *`str2`*.
`1,$s/`*`str1`*`/`*`str2`*`/g`	in the whole file ($: last line), **s**ubstitute all occurrences of *`str1`* by *`str2`*.

Applying a command several times - Examples

`5j`	move the cursor 5 lines down.
`30dd`	**d**elete 30 lines.
`4cw`	**c**hange 4 **w**ords from the cursor.
`1G`	**g**o to the first line in the file.

Misc

`[Ctrl] l`	redraw the screen.
`J`	**j**oin the current line with the next one
`u`	**u**ndo the last action

Exiting and saving

`ZZ`	save current file and exit vi.
`:w`	**w**rite (save) to the current file.
`:w `*`file`*	**w**rite (save) to the *`file`* file.
`:q!`	**q**uit vi without saving changes.

Going further

`vi` has much more flexibility and many more commands for power users!
It can make you extremely productive in editing and creating text.
Learn more by taking the quick tutorial: just type `vimtutor`.

Free Electrons
Embedded Linux Experts

The editor war

Technologies available in Information Technology are a lot.

Often, to solve a problem, you can choose between different instruments. The rule to base your choose is:

It does not exist “the best tool” but “the best tool to solve *your specific problem*”.

Sometimes different tools are more or less equivalent.

This is the case of editors emacs and vi:

https://en.wikipedia.org/wiki/Editor_war

Choose your editor

Try an editor and its tutorial,
watch videos on how to use it for your intended purpose,
spend a day or two using it with real files training your fingers.

The best editor for you is the one that makes you feel like you’re easily getting things done.

What is a script

A script is, in the simplest case, a list of system commands stored in a file.

Place commands in a script is useful

- to avoid having to retype them time and again
- to be able to modify and customize the script for a particular application
- to use the script as a program/command

The sha-bang #!

Every script starts with the sha-bang (#!) at the head, followed by the full path name of an interpreter.
Examples:

```
#!/bin/sh
#!/bin/bash
#!/usr/bin/perl
```

This tells your system that the file is a set of commands to be fed to the command interpreter indicated by the path.

The #! is a special marker that designates a file type, or in this case an executable shell script (type `man magic` for more details on this fascinating topic).

The command interpreter executes the commands in the script, starting at the top (the line following the sha-bang line), and ignoring comments.

Execute the script

- ★ The script execution requires the script has “execute” permissions:
  chmod +rx scriptname (gives everyone read/execute permission)
  chmod u+rx scriptname (gives only the script owner read/execute permission)

- ★ The script can be executed issuing:
  ./scriptname

- ★ The script can be made available as a command:
  - moving the script to /usr/local/bin (as root), making it available to all users as a system wide executable. The script could then be invoked by simply typing scriptname [ENTER] from the command-line.
  - Including the directory containing the script in the user's $PATH

Exercise: a first script

- ★ Write a script that upon invocation
  1) Says “Hello!”
  2) shows the time and date
  3) The script then saves this information to a logfile
- ★ Make the script executable
- ★ Execute the script
- ★ Make the script available as a command

Special characters (1)

★ Special characters have a meaning beyond its literal meaning

Comments [#]. Lines beginning with a # (with the exception of #!)

```
# This line is a comment.
```

Comments may also occur following the end of a command.

```
echo "A comment will follow." # Comment here.
```

Comments may also follow whitespace at the beginning of a line.

```
    # Note
```

Command separator [semicolon ;]. Permits putting two or more commands on the same line.

```
echo hello; echo world
```

Escape [backslash \]. This is a mechanism to express litterally a special character.
For example the \ may be used to escape " and ' echoing a string:

```
echo This is a double quote \”  # This is a double quote
```

Special characters (2)

Command substitution [backquotes or backticks `]. The `command` construct makes available the output of command for assignment to a variable.

a=`pwd`   (or a=$(pwd)) - (backtick AltGR+’ on Linux, ALT+096 on Windows)

acho $a   # display the path of your location

Wild card [asterisk *]. The * character serves as a "wild card", it matches every filename in a given directory or every character in a string.

Run job in background [and &]. A command followed by an & will run in the background.

```
    bash$ sleep 10 &
    [1] 850
    [1]+  Done                    sleep 10
```

Within a script, commands and even loops may run in the background.

To bring the script in foreground type `fg` or `CTRL Z fg`

To bring the script in background type `fg` or `CTRL Z bg`

Complete reference:

https://www.tldp.org/LDP/abs/html/special-chars.html

Exercise: special characters

- Write a commented command and execute it
- Write two commands on the same row and execute them
- Make the echo of a string containing one or more escaped characters
- Make the echo of a command (like ls or pwd) output
- Use wildcard to list all files starting with ‘a’ in your directory

Redirection (1)

Each UNIX command (or program) is connected to three communication channels between the command and its environment:

- Standard input (stdin) where the command read its input
- Standard output (stdout) where the command writes its output
- Standard error (stderr) where the command writes its error

When a command is executed via an interactive shell, the streams are typically connected to the text terminal on which the shell is running, but can be changed with redirection or with a pipeline

redirect stdout to a file	redirect stderr and stdout to a file
redirect stderr to a file	redirect stderr and stdout to stdout
redirect stdout to stderr	redirect stderr and stdout to stderr
redirect stderr to stdout	

Standard Input, Standard Output and Standard Error Symbols:

standard input	0<
standard output	1>
standard error	2>

Redirection (2)

Redirection [> &> >& >>].

- Redirect stdout to file (overwrite filename if it already exists):

```
scriptname > filename
scriptname  >> filename       # appends the output of ‘scriptname’ to file ‘filename’. If
                              #  filename does not already exist, it is created
```

- Redirect stderr to file (overwrite filename if it already exists):

```
scriptname 2> filename
```

- Redirect both the stdout and the stderr of command to filename:

command &> filename redirects both the stdout and the stderr of command to filename

- Redirects stdout of command to stderr:

```
command >&2
```

- Redirects stderr of command to stdout:

```
command 2>&1
```

Redirection: Examples

- Stdout redirected to file

```
find . -name pippo > find-output.txt
```

- Stderr redirected to file

```
find . -name pippo 2> find-errors.txt
```

- discards any errors that are generated by the find command

```
find / -name "*" -print 2> /dev/null
```

/dev/null is a simple device (implemented in software and not corresponding to any hardware device on the system).

/dev/null looks empty when you read from it.

Writing to /dev/null does nothing: data written to this device simply "disappear."

Often a command's standard output is silenced by redirecting it to /dev/null, and this is perhaps the null device's commonest use in shell scripting:

```
command > /dev/null
```

- Redirect both stdout and stderr to file

```
find .  -name pippo &> out_and_err.txt
```

- Redirect stderr to stdout: `find .  -name filename 2>&1`
- Redirect stdout to stderr: `find .  -name filename 1>&2`

Special characters (3)

Pipe [|]. Passes the output (stdout) of a previous command to the input (stdin) of the next one, or to the shell. This is a method of chaining commands together.

```
echo ls -l | bash
#  Passes the output of "echo ls -l" to the shell,
#+ with the same result as a simple "ls -l".
```

```
cat *.lst | sort | uniq
# Merges and sorts all ".lst" files, then deletes duplicate lines.
```

A pipe sends the stdout of one process to the stdin of another. In a typical case, a command, such as cat or echo, pipes a stream of data to a command that transforms it in input for processing:

```
cat $filename1 $filename2 | grep $search_word
```

Redirection with pipe and tee examples

Examples of redirection of the output of a command to be used as input of another:

- Display the output of a command (in this case ls) by pages:
  ```
  ls -la | less
  ```
- Count files in a directory:
  ```
  ls -l | wc -l
  ```
- Count the number of rows containing of the word “canadesi” in the file vialactea.txt
  ```
  grep canadesi vialactea.txt | wc -l
  ```
- Count the number of words in the rows containing the word “canadesi”

\`tee\` is useful to redirect output both to stdout and to a file. Example:

```
find . -name filename.ext 2>&1 | tee -a log.txt
```

This will take stdout and append it to log file. The stderr will then get converted to stdout which is piped to tee which appends it to the log and sends it to stdout which will either appear on the tty or can be piped to another command.

To go deep: https://stackoverflow.com/questions/2871233/write-stdout-stderr-to-a-logfile-also-write-stderr-to-screen

Exercise: redirection

Create a directory and file tree like this one:

```
my_examples  /ex1.dir
             /ex2.txt
             /ex3.dir
             /ex3.dir/file1.txt
             /ex3.dir/file2.txt
             /ex3.dir/file3.txt
```

Remove read permissions to directory *ex1.dir*

Redirect output on a file. Error is displayed on terminal

Redirect error on a file. Output is displayed on terminal

Verify the content of the files

Stderr redirected to file

Redirect output and errors simultaneously

Use pipe to redirect the output of a command to another command and to a file

Use tee to redirect output both to stdout and to a file