Bash Lecture 1
Bash and Linux command line introduction

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/bertocco/abilita_info_units_1920

Arguments of this lesson

- ★ How the shell works with you and linux
- ★ Features of a shell
- ★ Manipulating the shell environment

What is a shell?

★ SHELL is the human interface point for UNIX

SHELL is a program layer that provides an environment to enter commands and parameters to produce a given result.

To meet varying needs, UNIX has provided different shells.

- Bourne (sh)
- Bourne Again (bash)
- Korn (ksh)
- C shells (csh)
- TC-Shell (tcsh)
- Z shell (zsh)

Shell differences

★ They differ in

- the various options they give the user to manipulate the commands
- the complexity and capabilities of the scripting language.

Why bash?

★Flexible

★ More friendly than others

★The default in the most part of linux distributions

UNIX commands (1)

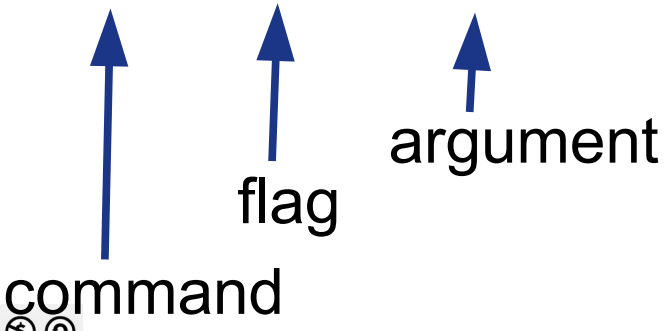
★ General form of a command:

command [flags] [argument1] [argument2] …

Example:

`ls -a -l` *or* `ls -al`

`ls -al mydir`



UNIX commands

- ★ Arguments can be optional or mandatory
- ★ All commands have a return code (0 if OK)

  Read return code: `echo $?`

  The return codes can be used as part of control logic in shell scripts

- ★ All UNIX commands have an help:
  `man command` or `man <number> command`

UNIX commands

★ Arguments can be optional or mandatory

Exercise: Try

- create the folder

  mkdir myFolder

- jump inside the created folder

  cd myFolder

- create two files

  touch pippo pluto

- jump outside the folder

  cd ..

- list the content of the folder where you are

  ls

  Argument NOT mandatory. Optional

- list a specific folder content

  ls myFolder

UNIX commands

★ Arguments can be optional or mandatory

Exercise: Try

- create a folder

  mkdir testFolder

- create (inside the folder) one or two files inside the folder

  cd testFolder ; touch filePippo ; touch filePluto

- or: cd testFolder ; touch filePippo filePluto

- inside the folder try the command to remove files (without arguments)

  pwd # to be sure to be in the folder

  rm

```
bertocco@firiel:~/work$ ls
filePippo  filePluto  pioo
bertocco@firiel:~/work$ rm
rm: missing operand
Try 'rm --help' for more information.
bertocco@firiel:~/work$
```

- An argument (indicating the file to remove) is mandatory

  rm filePippo # remove only the indicated file

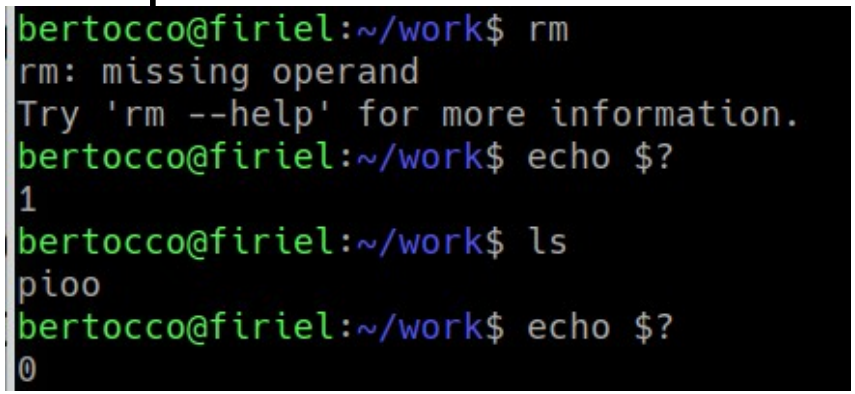
  rm * # remove all files

UNIX commands

★ All commands have a return code (0 if OK)

Read return code: `echo $?`

The return codes can be used as part of control logic in shell scripts



Exercise: try

ls

echo $? # the result will be 0 : correct command

rm

echo $? # the result will be 1: command failure

UNIX commands

★ All UNIX commands have an help:
`man command` or
`man <number> command

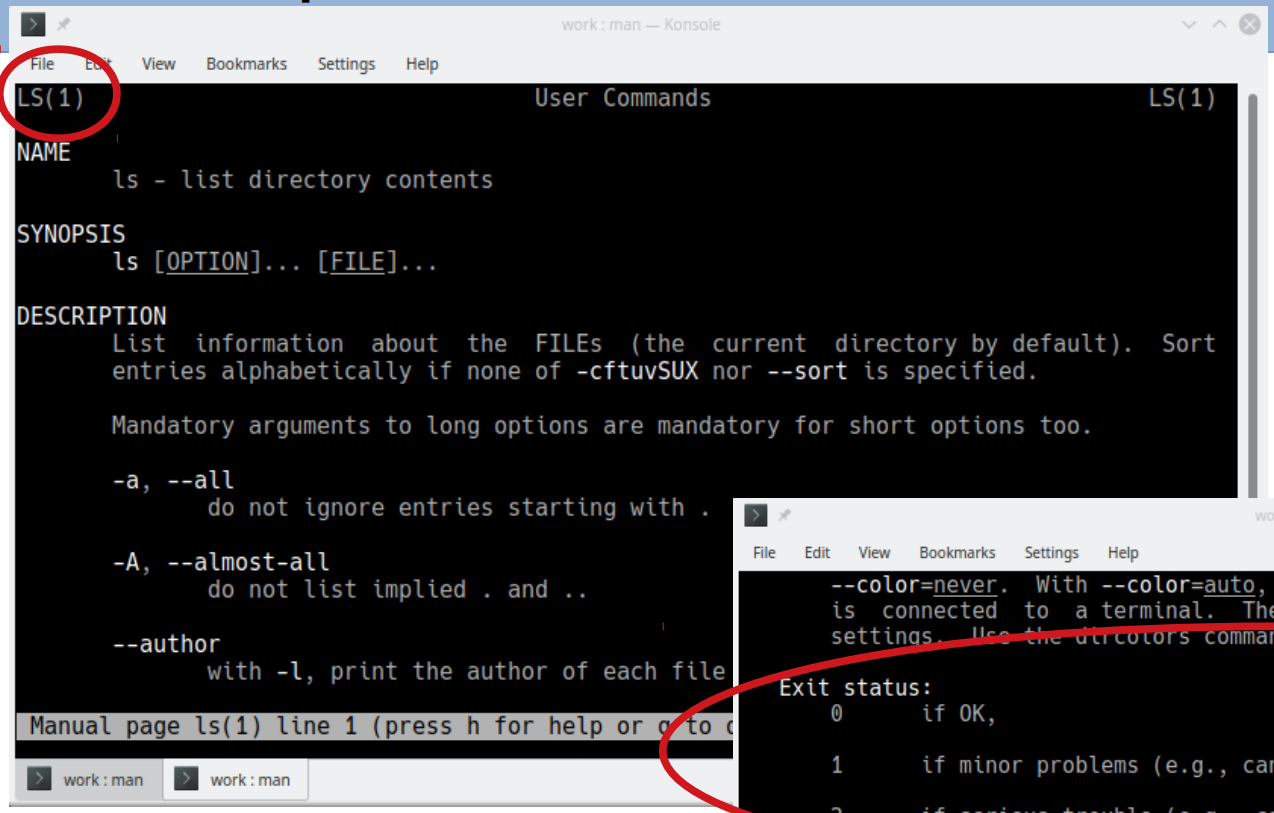
Example:

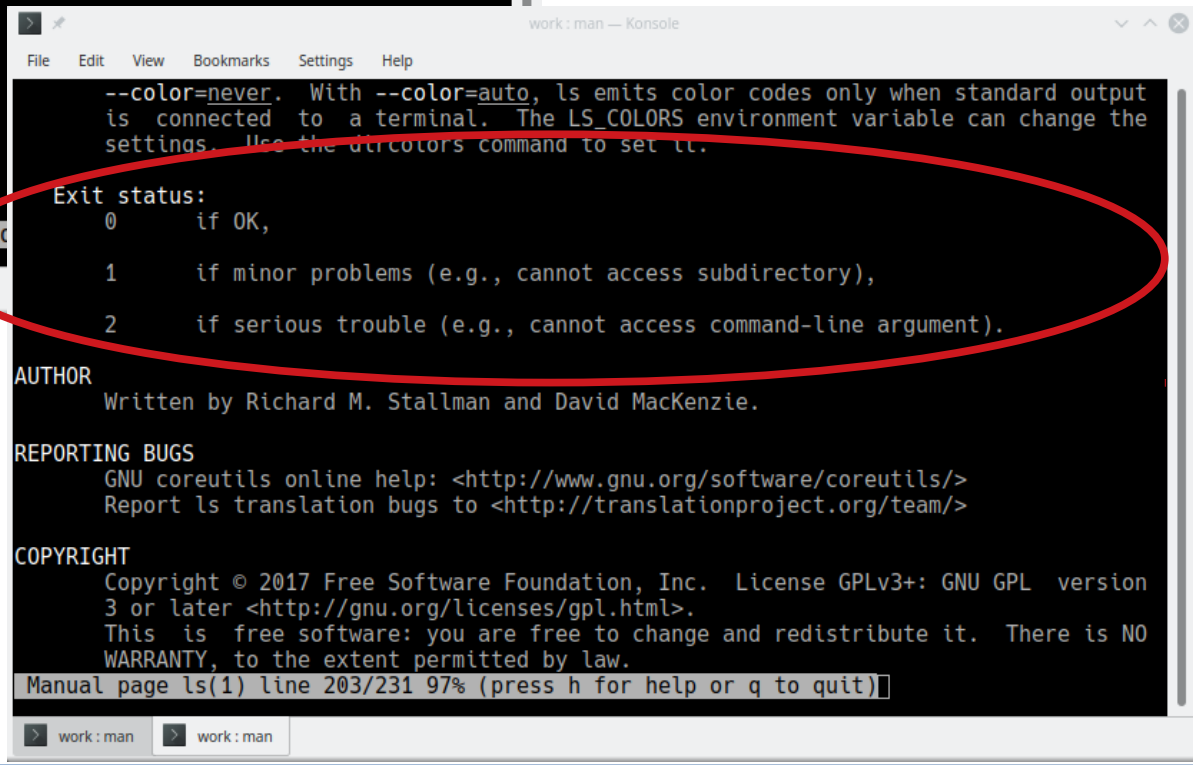
man ls

man wait

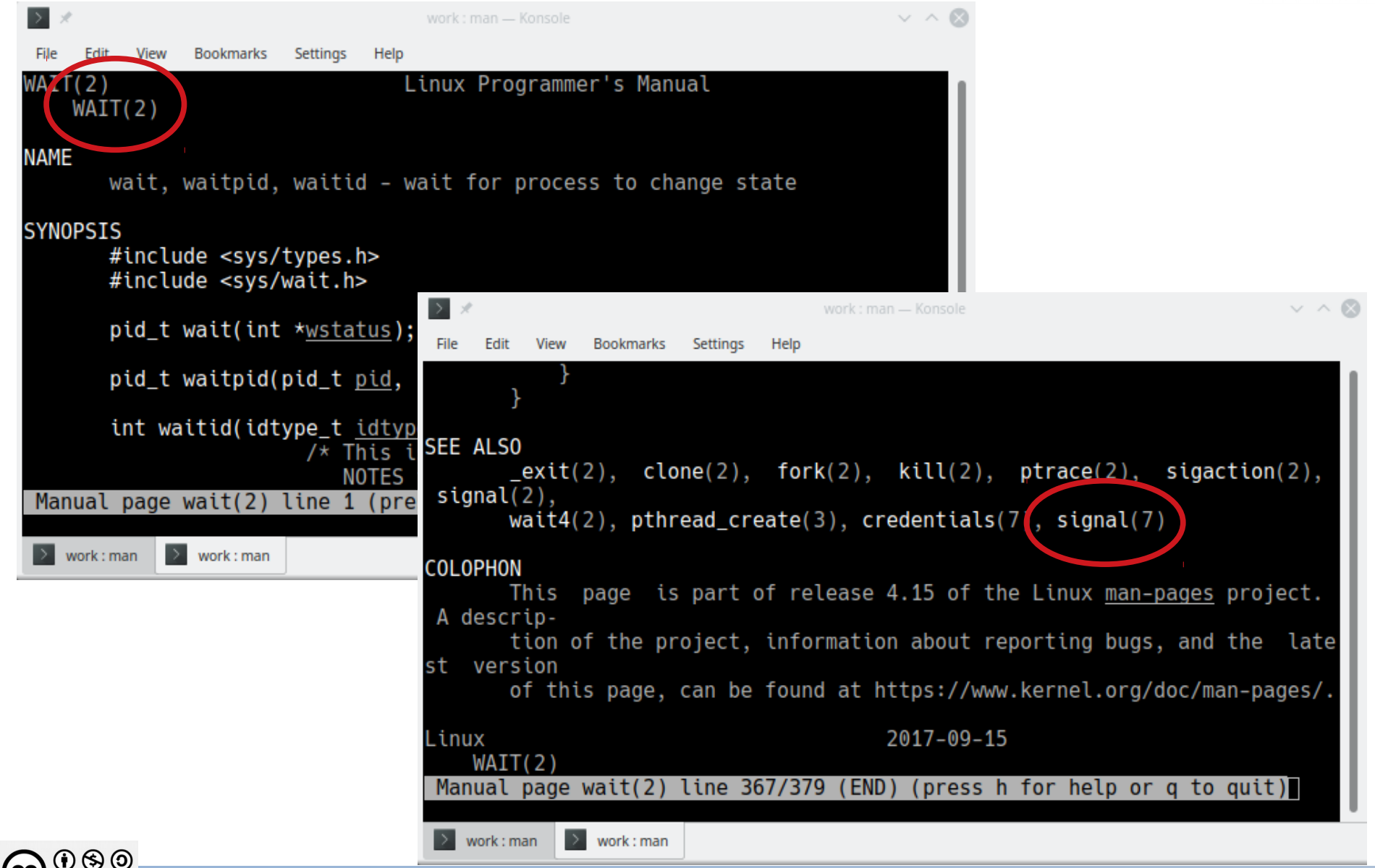
man 7 signal

`ls` help





`wait` help



UNIX commands (2)

★ All commands:

- accept inputs from the **standard input**,

  is where UNIX gets the input for a command

- display output on **standard output**

  is where UNIX displays output from a command

- display error message on **standard error**

  is where UNIX displays any errors as a result of the execution of a command

★ UNIX has redirection capabilities: to redirect one or more of these (see “advanced bash” lesson)

`ls` (1)

`ls` can be used to inquire about the various attributes of one or more files or directories.

You must have read permission to a directory to be able to use the ls command on that directory and the files under that directory.

The `ls` command generates the output to standard output, which can be redirected, using the UNIX redirection operator >, to a file.

`ls` (2)

You can provide the names of one or more filenames or directories to the ls command. The file and directory names are optional. If you do not provide them, UNIX processes the current directory.

Be default, the list of files within a directory is sorted by filename. You can modify the sort order by using some of the flags.

You should also be aware that the files starting with . (period) will not be processed unless you use the -a flag with the ls command. This means that the entries . (single period) and .. (two consecutive periods) will not be processed by default.

`ls`: Exercices

- Try and understand differences:

ls; ls -l; ls -al

- Try and understand differences:

ls -trl; ls -tl

- Try an output redirection:

ls -l > myfileout.txt

File manipulation commands

It exists a set of file manipulation commands to manage files and directories.

To use these commands, the user needs to have right on the file to manage.

drwxrwxr-x 2 bertocco bertocco 20480 Dec 14 09:00 BACKUP

owner *group* *size* *last_access_date* *file-name*

drwxrwxr-x permissions representation:

d means it is a directory (- for a file)

rwx means readable, writable, executable by owner

rwx readable, writable, executable by group

r-x readable, NOT writable, executable by

File Permissions

Understand the meaning of:

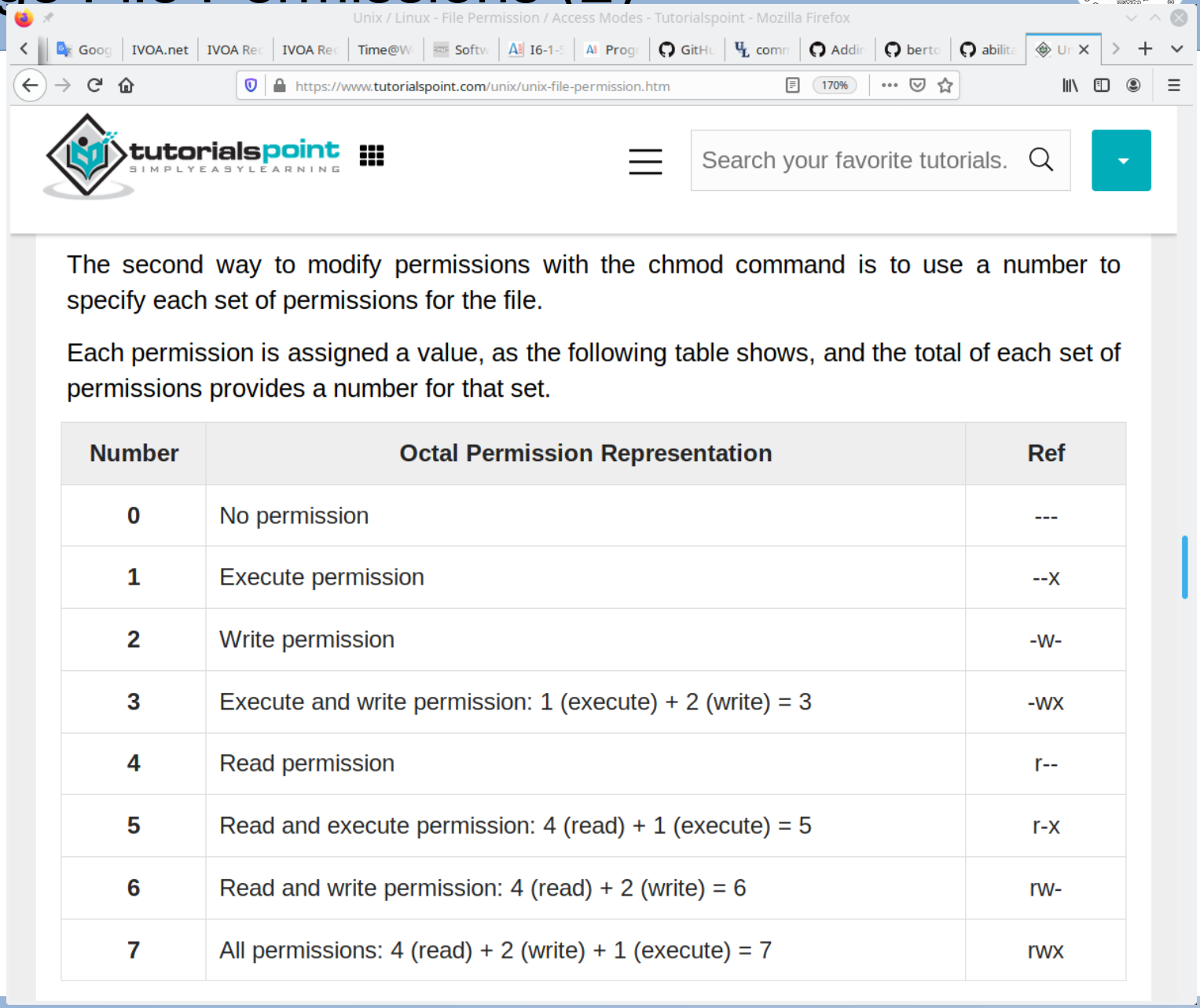
```
drwxrwxr-x   2 bertocco bertocco      4096 Apr 26  2018 config
-rw-rw-r--   1 bertocco bertocco     10240 Mar 13  2017 config.tar
-rw-------   1 bertocco bertocco 960065536 Dec  3 22:02 core.3040
-rw-rw-r--   1 bertocco bertocco   7290880 May  8  2017 demo_EGIconf.tar
drwxr-xr-x.  4 bertocco bertocco      4096 Dec  7 15:57 Desktop
drwx------. 12 bertocco bertocco      4096 Aug 13 19:01 dev
drwxr-xr-x. 14 bertocco bertocco      4096 Nov 28 17:18 Documents
drwxr-----. 13 bertocco bertocco      8192 Dec 10 12:35 Downloads
drwxrwxr-x   2 bertocco bertocco       147 Apr 24  2018 exchange
-rw-r--r--   1 bertocco bertocco       181 Apr 13  2017 filmatini_utili.txt
```

Change File Permissions (1)

Chenge read permission (similarly for write ‘w’ and execute ‘x’):

```
-rw-rw-r-- 1 bertocco bertocco 0 Dec 14 15:30 pippo
$ chmod -r pippo     # remove all read permissions. Check:
$ ls -l pippo
--w--w---- 1 bertocco bertocco 0 Dec 14 15:30 pippo
$ chmod +r pippo     # add all read permissions, Check:
$ ls -l pippo
-rw-rw-r-- 1 bertocco bertocco 0 Dec 14 15:30 pippo
$ chmod -r pippo     # remove a new time all permissions, to restart from
--w--w---- 1 bertocco bertocco 0 Dec 14 15:30 pippo
$ chmod u+r pippo     # add read permission to user
$ ls -l pippo
-rw--w---- 1 bertocco bertocco 0 Dec 14 15:30 pippo
$ chmod g+r pippo     # add read permission to group
$ ls -l pippo
-rw-rw---- 1 bertocco bertocco 0 Dec 14 15:30 pippo
$ chmod a+r pippo     # add read permission to all
$ ls -l pippo
-rw-rw-r-- 1 bertocco bertocco 0 Dec 14 15:30 pippo
```

Change File Permissions (2)



The second way to modify permissions with the chmod command is to use a number to specify each set of permissions for the file.

Each permission is assigned a value, as the following table shows, and the total of each set of permissions provides a number for that set.

Number	Octal Permission Representation	Ref
0	No permission	---
1	Execute permission	--x
2	Write permission	-w-
3	Execute and write permission: 1 (execute) + 2 (write) = 3	-wx
4	Read permission	r--
5	Read and execute permission: 4 (read) + 1 (execute) = 5	r-x
6	Read and write permission: 4 (read) + 2 (write) = 6	rw-
7	All permissions: 4 (read) + 2 (write) + 1 (execute) = 7	rwx

Main file manipulation commands

Create an empty file in the current path

- `touch testFile` creates a file

Create an empty file in a specified path

- `touch one_dir_more/target_dir/testFile

Create an empty directory in the current path

- `mkdir  mydir`

Create an empty directory in the specified path forcing intermediate folders if needed

- `mkdir -p /*onedir/*twodir/threedir`

Main file manipulation commands

Delete a single file

- `rm file1`

Delete a single file in the specified path

- rm one_dir_more/target_dir/testFile

Delete all files in the current path

- `rm *`

*** is a UNIX wildcard: here it matches all files**

Delete all files in the specified path

- `rm one_dir_more/target_dir/*`

Delete an empty folder

- `rm myFolder` works only if myFolder is empty

Delete recursively all files and folders contained in myFolder and myFolder itself

- `rm -r myFolder`

Copy file1 on file2

overwrite file2 if already exists create it if it does not exist:

- `cp file1 file2`

Copy file1 on file2, but asking before: “are you sure?”

- `cp -i file1 file2`

Copy file1 on file2 removing the -i flag if set

- `\cp file1 file2`

BE VERY CAREFUL!!!

Rename a file:

- `mv file1 file2` moves file1 on file2

  # same result using two commands (but less neat)

- `cp file1 file2`; `rm file1` # but not the best way

Main file manipulation commands: summary

- `touch` creates a file
- `mkdir  mydir` creates a directory (where you are)
- `mkdir -p /*onedir*/twodir/threedir`
- `rmdir mydir` delete an empty directory
- `rm -rf` force to recursively delete a non empty directory
- `cp file1 file2` copy file1 on file2 (overwriting it if already exists, creating file2 if it does not exist)
- `cp -i file1 file2` before copy asks “are you sure?”
- `\cp file1 file2` remove the -i flag if set
- `rm file1` removes file1
- `mv file1 file2` moves file1 on file2 # it is the same of :
- `cp file1 file2`; `rm file1`

BE VERY CAREFUL!!!

File manipulation commands: Exercises

- Create a file
- Create a directory
- Create a directory tree
- Create files in the directory tree
- Remove a file
- remove a directory (emprty and not empty)
- remove a directory tree (emprty and not empty)
- Rename a file

`cat`

`cat` is used to display a text file or to concatenate multiple files into a single file.

By default, the cat command generates outputs into the standard output and accepts input from standard input.

The cat command takes in one or more filenames as its arguments. The files are concatenated in the order they appear in the argument list.

`cat`: Exercices

- Display file on a terminal

  cat testfile

- concatenate multiple files for display on the terminal

  cat testfile1 testfile2 testfile3

- concatenate these files into a file called testfile, use the redirection operator > as follows:

  cat testfile1 testfile2 testfile2 > testfile

- If the file testfile already exists, it is overwritten with the concatenated files testfile1, testfile2 and testfile3. If testfile already exists and you want to concatenate at the end of the existing file, instead of using the redirection operator >, you must use the >> (two consecutive greater than sign) operator as follows:

  cat testfile1 testfile2 testfile2 >> testfile

`ln`

`ln` provides alternate names for the same file.

It links a file name to another one and it is possible to link a file to another name in the same directory or the same name in another directory.

The flags that can be used with the ln command are as follows:

-s to create a soft link to another file or directory. In a soft link, the linked file contains the name of the original file. When an operation on the linked filename is done, the name of the original file in the link is used to reference the original file.

-f to ensure that the destination filename is replaced by the linked filename if the file already exists. Remove destination files, if existing

`ln`: hard link

Hard link (ln without flags):

In sourceFile targetFile

- The hard link refers directly to the physical location of another file. It points to the same “node” in the filesystem that the original file points to
- The hard link acts like a mirror of the original file. The content pointed by the two names is synchronized
- When the source of the link “sourceFile” is moved or removed, the hard link still refer to the source
- The hard link cannot refer to a directory and cannot cross file system boundaries (ex.: can not link a file in a mounted removable disk from home)

```
bertocco@firiel:~/tmp$ ln /media/bertocco/7445-67A4/ODMEC_1/ios-stream.smil ciccio
ln: failed to create hard link 'ciccio' => '/media/bertocco/7445-67A4/ODMEC_1/ios-stream.smil': Invalid cross-device link
bertocco@firiel:~/tmp$
```

`ln -s`: soft/symbolic link

Soft/symbolic link (ln with flag s):

ln -s sourceFile targetFile

- A symbolic link is a symbolic path indicating the location of the source file. It is some way an alias to a file-name.
- A symbolic link can refer to a directory
- A symbolic link can cross file system boundaries (can link a file in a mounted removable disk from home).

```
bertocco@firiel:~/tmp$ ln -s /media/bertocco/7445-67A4/ODMEC_1/ios-stream.smil ciccio
bertocco@firiel:~/tmp$ ls -l
total 0
lrwxrwxrwx 1 bertocco bertocco 49 nov 27 15:37 ciccio -> /media/bertocco/7445-67A4/ODMEC_1/ios-stream.smil
bertocco@firiel:~/tmp$
```

- If the sourceFile of the link is moved or removed, then the symbolic link is not updated (the ‘alias’ results broken).

`ln`: hard link vs soft/symbolic link

Create a simple source file

- create a hard link
- create a soft link
- inspect the content of the hard link
- inspect the content of the soft link:

contents are the same

```
bertocco@firiel:~/symlink$ cat sourceFile.txt
Ciao ciao Pippo!
bertocco@firiel:~/symlink$ ln sourceFile.txt hardLinkTargetFile.txt
bertocco@firiel:~/symlink$ ln -s sourceFile.txt symbolicTargetFile.txt
bertocco@firiel:~/symlink$ cat hardLinkTargetFile.txt
Ciao ciao Pippo!
bertocco@firiel:~/symlink$ cat symbolicTargetFile.txt
Ciao ciao Pippo!
```

`ln`: hard link vs soft/symbolic link

Modify the source file

- inspect the content of the hard link
- inspect the content of the soft link:

contents have the same modifications

```
bertocco@firiel:~/symlink$ echo "Provo a modificare source" >> sourceFile.txt
bertocco@firiel:~/symlink$ cat hardLinkTargetFile.txt
Ciao ciao Pippo!
Provo a modificare source
bertocco@firiel:~/symlink$ cat symbolicTargetFile.txt
Ciao ciao Pippo!
Provo a modificare source
```

`ln`: hard link vs soft/symbolic link

list the folder content to check the file status

- remove the source file
- list the folder content to check the file status

the soft link is broken, the hard link is untouched

```
bertocco@firiel:~/symlink$ cat sourceFile.txt
Ciao ciao Pippo!
Provo a modificare source
bertocco@firiel:~/symlink$ ls -l
total 8
-rw-rw-r-- 2 bertocco bertocco 43 nov 27 16:41 hardLinkTargetFile.txt
-rw-rw-r-- 2 bertocco bertocco 43 nov 27 16:41 sourceFile.txt
lrwxrwxrwx 1 bertocco bertocco 14 nov 27 16:41 symbolicLinkTargetFile.txt -> sourceFile.txt
bertocco@firiel:~/symlink$ rm sourceFile.txt
bertocco@firiel:~/symlink$ ls -l
total 4
-rw-rw-r-- 1 bertocco bertocco 43 nov 27 16:41 hardLinkTargetFile.txt
lrwxrwxrwx 1 bertocco bertocco 14 nov 27 16:41 symbolicLinkTargetFile.txt -> sourceFile.txt
bertocco@firiel:~/symlink$ cat symbolicLinkTargetFile.txt
cat: symbolicLinkTargetFile.txt: No such file or directory
bertocco@firiel:~/symlink$ cat hardLinkTargetFile.txt
Ciao ciao Pippo!
Provo a modificare source
```

`ln`: Exercices

★If you want to link testfile1 to testfile2 in the current directory, execute the following command:

ln testfile1 testfile2

This creates a hard linked testfile2 linking it to tesftfile1. In this case, if one of the files is removed, the other will remain unaltered.

★If testfile is in the current directory and is to be linked to testfile in the directory /u/testuser/testdir, execute the following command:

ln testfile u/testuser/testdir

★To create a symbolic link of testfile1 in the current directory, execute the following command:

ln -s testfile1 testfile2

This creates a linked testfile2, which will contain the name of testfile1. If you remove testfile1, you will be left with an orphan testfile2, which points to nowhere.

Command `echo` and strings

★When one or more strings are provided as arguments, echo by default repeats those stings on the screen.

Example (try)

echo This is a pen.

It is not necessary to surround the strings with quotes, as it does not affect what is written on the screen. If quotes (either single or double) are used, they are not repeated on the screen (try `echo “This is a pen.”`).

★`echo` can also show the value of a particular variable if the name of the variable is preceded directly (i.e., with no intervening spaces) by the dollar character ($), which tells the shell to substitute the value of the variable for its name. Example (try):

x=5; echo The number is $x.

Command `echo` examples

★echo This is a pen.

★x=5
echo The number is $x.
echo “The number is $x.”

★Simple backup script
OF=/home/me/my-backup-$(date +%Y%m%d).tgz
tar -czf $OF <path>/dir_or_file_to_tar
ls -l (to check the result)

Exercises

★ Verify that you are using bash:

`echo $SHELL`

★Explore help command

type `help`

★ Explore help for `ls` command

type `man command_name`

★List files `ls` or `ls -l` and check differences

★ List all files `ls -al`

★List files by date (direct and reverse order) `ls -trl`