Introduction to UNIX Commands

Why Talk About UNIX?

- Almost all supercomputers in the world use Linux as their operating system, the CECI clusters are no exception
- A Linux operating system is based on the linux kernel which is the central part of the operating system managing the operations of the computer and the hardware
- Linux, which stands for (Linux Is Not UniX) is a UNIX-like operating: the filesystem, environment and commands are very similar to UNIX

The command shell

- When you login to a CÉCI cluster, you connect to a login node and have access to a shell
- A shell is a program that takes commands from the keyboard and gives them to the operating system

The following is an overview of the commands that may be useful for your work on the CÉCI clusters.

List Files and Directories

The current directory

At login, your current directory (the folder where you are currently working) is your home directory.

You can print the absolute path (with respect to the root of the filesystem) of your current directory pwd command.

```
$ pwd
/home/user
```

List Files and Directories

To list the files and folder in a directory, you can use the `ls` command.

```
 $ ls
file1 file2 dir1 dir2 dir3
```

Listing Files and Folders

- Files and folders starting with a dot (.) are hidden.
- To list the files and folder, including the ones that are hidden, use `ls` with the option `-a` (all).

```
 $ ls -a
.      .hidden_dir   dir1    dir3    file2
..     .hidden_file  dir2    file1
```

Listing Files and Folders

- Files and folders starting with a dot (.) are hidden.
- To list the files and folder, including the ones that are hidden, use `ls` with the option `-a` (all).

```
$ ls -a
.    .hidden_file   file1  folder1  folder3
..   .hidden_folder file2  folder2
```

Notice the `.` and `..`? These are the current and parent directories.

Listing Files and Folders

Alternatively, you can use `ll` which gives a more readable output.

```
 $ ll
-rw-r--r--  1 user  group     0B Sep 23 20:17 file2
-rw-r--r--  1 user  group     0B Sep 23 20:17 file1
drwxr-xr-x  2 user  group    64B Sep 23 20:17 dir3
drwxr-xr-x  2 user  group    64B Sep 23 20:17 dir2
drwxr-xr-x  2 user  group    64B Sep 23 20:17 dir1
```

The current directory

- The hidden `.` represents the current directory
- It may used to run a program or a script located in the current directory

```
 $ myscript
-bash: myscript: command not found
```

Here, the OS is looking in all the executable paths (directories which are supposed to contain executables) and did not find `myscript`.

The current directory

- The hidden `.` represents the current directory
- It may used to run a program or a script located in the current directory

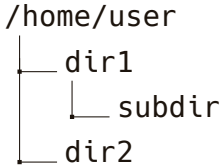
```
 $ ./myscript
Hello, I'm a script!
```

By using `./` we specify that the script is in the current directory.

Changing Directory

Changing directory

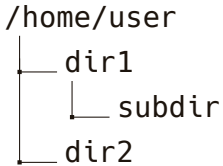
To change the current directory use the `cd` command followed by the name of the directory




```
 $ pwd
/home/user
 $ cd dir1
 $ pwd
/home/user/dir1
```

Changing directory

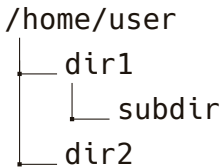
To go to the parent directory (the directory in which your working directory is located) you can use `..`




```
 $ pwd
/home/user/dir1
 $ cd ..
 $ pwd
/home/user
```

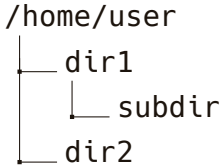
Changing directory

To change to your home directory, you can use ~. The ~ can also be used to specify a path relative to your home directory.




```
 $ cd ~
 $ pwd
/home/user
 $ cd ~/dir1/
 $ pwd
/home/user/dir1
```

Changing directory

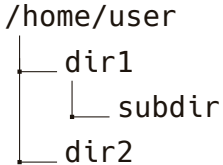


Moving multiple levels down the filesystem hierarchy is possible by separating the directory names by `/`

```
 $ pwd
/home/user
 $ cd dir1/subdir
 $ pwd
/home/user/dir1/subdir
```

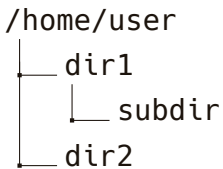
Changing directory

Moving multiple levels up the filesystem hierarchy is possible by using multiple `..` separated by `/`




```
 $ pwd
/home/user/dir1/subdir
 $ cd ../..
 $ pwd
/home/user/
```

Changing directory



Moving to a directory located up the filesystem hierarchy is possible by using `..`, the name of the directory and `/` as the separator

```
 $ pwd
/home/user/dir1/
 $ cd ../dir2
 $ pwd
/home/user/dir2
```

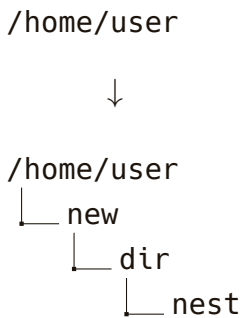
Create Directories and Files

Create Directories

To create a directory use the `mkdir` command followed by the name of the new directory

```
$ mkdir dir
```

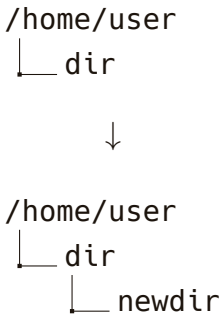
Create Directories



You can create multiples nested directories in one command with the `-p` option. Use `/` as the delimiter of the name of the directories

```
$ mkdir -p new/dir/nest
```

Create Directories

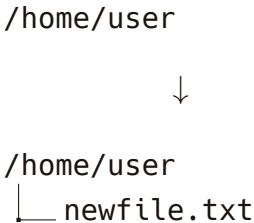


If the directory already exists there is no need for the `-p` option

```
$ mkdir dir/newdir
```

Create Directories and Files

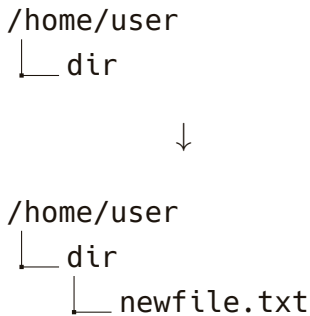
Create Files



To create a file use the `touch` command followed by the name of the new file

```
$ touch newfile.txt
```

Create Files

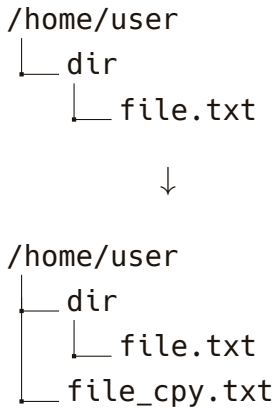


To create a file in a directory use the `touch` command followed by the name of the directory and the name of the new file separated by `/`

```
$ touch dir/newfile.txt
```

Copying Files and Directories

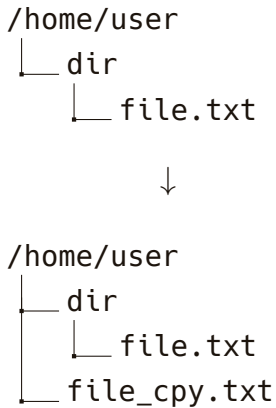
Copying Files



To copy a file, use the `cp` command followed by the path to the file to copy and the path to where you want the copy to be located

```
$ cp dir/file.txt file_cpy.txt
```

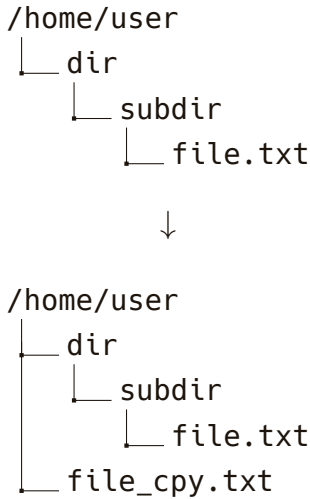
Copying Files



The same operation can be done from `dir` directory. In this case, we use `..` to indicate that we want to copy the file to the parent directory

```
$ cd dir
$ cp file.txt ../file_cpy.txt
```

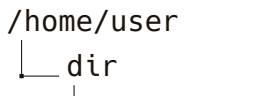
Copying Files



We use `..` multiple times to copy the file to a location multiple levels up in the filesystem hierarchy

```
$ cd dir/subdir
$ cp file.txt ../../file_cpy.txt
```

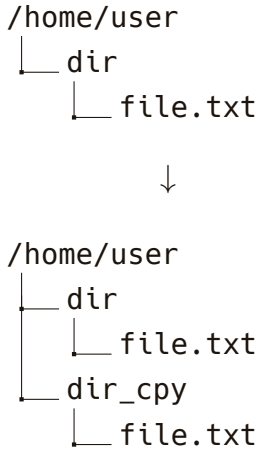
Copying Directories



When we try to copy a directory, we get an error message looking like this

```
 $ cp dir dir_cpy
cp: omitting directory 'dir'
```

Copying Directories

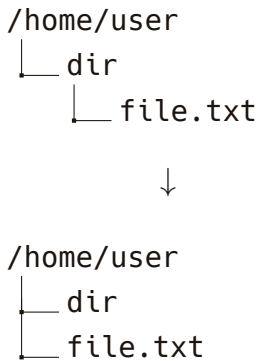


To copy a directory, we need to use the recursive option `-r`

```
$ cp -r dir dir_cpy
```

Moving Files

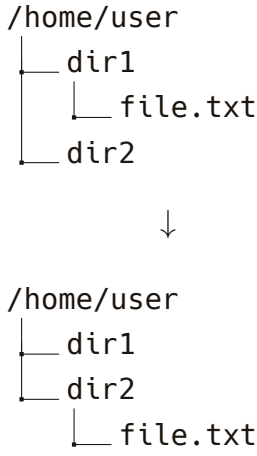
Moving Files



Moving a file is done using the `mv` followed by the path to the file to move and the path to the new location of the file

```
$ mv dir/file.txt file.txt
```

Moving Files

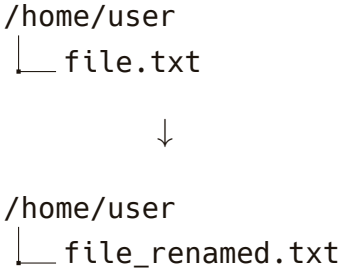


Moving a file is done using the `mv` followed by the path to the file to move and the path to the new location of the file

```
$ mv dir1/file.txt dir2/file.txt
```

Renaming Files and Directories

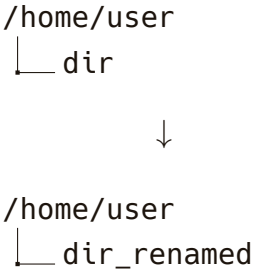
Renaming Files and Directories



The `mv` command can also be used to rename files

```
$ mv file.txt file_renamed.txt
```

Renaming Files and Directories



The `mv` command can also be used to rename directories

```
$ mv dir dir_renamed
```

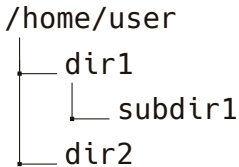
Absolute and Relative Path

Absolute and Relative Path

A path to a directory or a file can be specified either by using an absolute path or a relative path.

- An absolute path is defined as specifying the location of a file or directory from the root directory: starting with a `/`
- A Relative path is defined as the path related to the present working directory: it never starts with a `/`

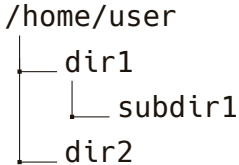
Absolute and Relative Path



Here we use the absolute path to move to the `subdir1` directory

```
 $ pwd
/home/user
 $ cd /home/user/dir1/subdir1
 $ pwd
/home/user/dir1/subdir1
```

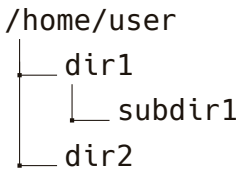
Absolute and Relative Path



The same operation can be done by using a relative path

```
 $ pwd
/home/user/
$ cd dir1/subdir1
 $ pwd
/home/user/dir1/subdir1
```

Absolute and Relative Path



To move to the `dir2` directory using a relative path, we can use `..` to move up in the hierarchy before selecting `dir2`

```
 $ pwd
/home/user/dir1/subdir1
 $ cd ../../dir2
 $ pwd
/home/user/dir2
```

Get the Content of Files

Get the Content of Files

The `cat` command allows to view the content of a file

```
 $ cat file.txt
This is the content of the file
This is the second line of the file
```

Get the Content of Files

The `cat` command also allows to view the content of multiple files in one command

```
 $ cat file1.txt file2.txt
This is the content of the file1.txt
This is the content of the file2.txt
```

Get the Content of Files

Sometimes, the content of the file is too large fit in the terminal windows. In this case, the `less` command comes in handy.

```
$ less file.txt
```

Then you can use the ↓ and ↑ keys to navigate. Press `q` to quit.

Wildcards

Wildcards

A wildcard is a symbol or a set of symbols that stands in for other characters. It can be used to substitute for any other character or characters in a string

- `?` matches a single character. For example, `a??c` will match any string with a length of 4, starting with a and ending with c.
- `*` matches any character or set of characters. For example, `a*c` will match any string with of any length, starting with a and ending with c.
- `[val]` matches characters enclosed in square brackets. For example, `a[d-f]c` will match "adc", "aec" and "afc" but not "agc".

Wildcards

To list the files starting with any character or set of characters you can use the * wildcard

```
 $ ls
file.in file.out
 $ ls *.in
file.in
```

Wildcards

If we have four files all starting with "file", two files ending with a digit and two files ending with a letter.

We can selectively list the files ending with a letter using a wildcard

```
 $ ls
file1 file2 filea fileb
 $ ls file[a-z]
filea fileb
```

Wildcards

If we have four files all starting with "file", two files ending with a digit and two files ending with a letter.

We can selectively list the files ending with a digit using a wildcard

```
 $ ls
file1 file2 filea fileb
 $ ls file[0-9]
file1 file2
```

Wildcards

To list the files with the letter ”i” as the second letter, we can use the ? wildcard

```
 $ ls
file.txt video.mp4 test.out test.in
 $ ls ?i*
file.txt video.mp4
```

Wildcards

To list the files an extension of size 3 we can combine the `?` and `*` wildcards

```
 $ ls
file.txt video.mp4 test.out test.in
 $ ls *.???
file.txt video.mp4 test.out
```

Pipes and Redirections

UNIX Philosophy

An important design philosophy of UNIX was minimalism and modularity. Linux systems follow the same principles.

To achieve this goal, the programs are designed so that

- Each program do one thing well
- Expect the output of every program to become the input to another

Redirecting to a file

The output of programs can be redirected to a file using the > operator

```
 $ ls > file.txt
 $ cat file.txt
file1 file2 dir1 dir2 dir3
```

- If the file does not exist, it will be created
- If the file already exists the old content will be discarded and replaced by the new one

Redirecting to a file

To redirect the output of a program to the end of a file, we can use the >> operator

```
 $ ls >> file.txt
 $ cat file.txt
Old content of file.txt
file1 file2 dir1 dir2 dir3
```

- If the file does not exist, it will be created
- If the file already exists the new content will be added at the end of it

Pipes

A pipe transfer the standard output of a program to another program.

For example, if the number of files in a directory is huge, you may be interested in redirecting the output of `ls` to `less`. To do so, use the `|` operator.

```
$ ls | less
```

Pipes

Let's determine the number of frames in the LaTeX document used to create these slides.

```
$ cat unix-intro.tex | grep begin{frame} | wc -l
     25
```

- We use `cat` to get the content of the file
- Then `grep` is used to find all instances of `begin{frame}`
- Finally we use `wc` with the `-l` option to count the number of lines

That’s All Folks!