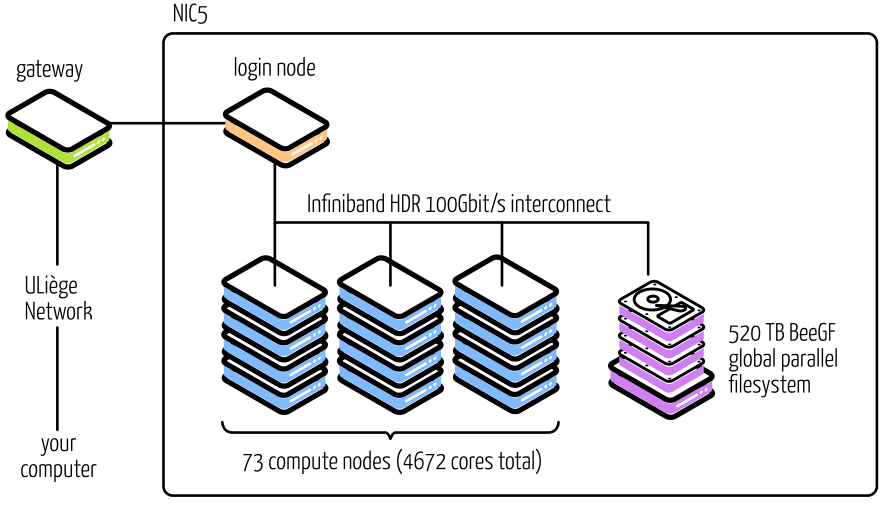
Introduction to a High-Performance Computing Environment

Orian Louant
orian.louant@uliege.be

Tuesday 28th September, 2021

NIC5 Overview



The Login Node

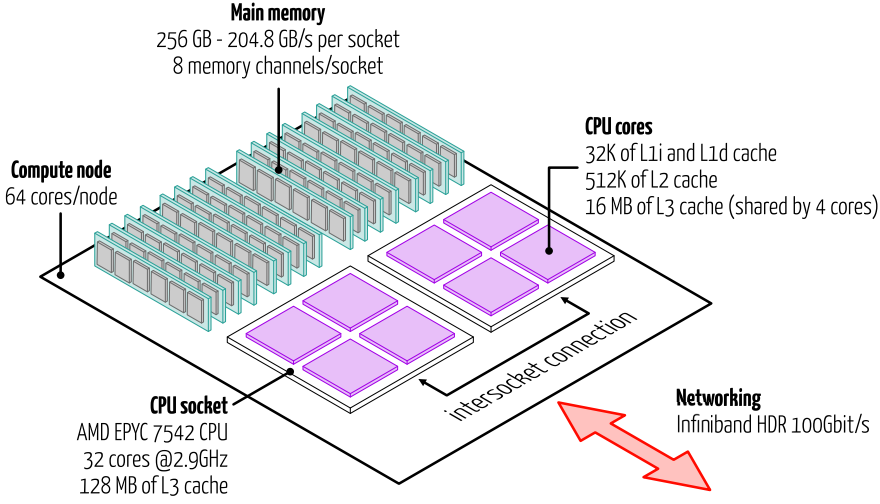
The login node is the machine to which you are connected when connecting to NIC5 (or any cluster).

- HPC clusters is shared environment: you are not alone, other users use the resources at the same time as you
- this is particularly true for the login node

The login node is intended for small tasks (in terms of resource usage)

- edit files prepare your job, submit your job, ...
- compile and debug your code with a limited number of threads/processes (1-4)

NIC5 Compute nodes



Basics of Slurm

Resource Sharing on a Supercomputer

Contrary to the login node, you cannot access the compute node directly

- resource sharing on a supercomputer is often organized by a piece of software called a resource manager or job scheduler
- the scheduler is responsible for the allocation of the compute node
- users submit jobs, which are scheduled and allocated resources by the resource manager.

The CÉCI clusters use Slurm as resource manager and job scheduler

Slurm Overview

Slurm is responsible for

- register user request for computational resources and put the job in the queue
- when resources are available, launch (an) eligible pending job(s)
- monitor the running jobs and check if they don't use more resources than allocated

The basic Slurm commands are the following

- `sinfo` : view information about Slurm nodes and partitions
- `squeue` : view information about jobs located in the Slurm scheduling queue
- `sbatch` : submit a batch script to Slurm
- `scancel` : Cancel a job

sinfo

View information about Slurm nodes and partitions.

Here we have two partitions, **batch** and **debug** with maximum allowed run time of 2 days and 6 hours respectively. These two partitions are available for computing (**up**).

```
 $ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
batch*       up 2-00:00:00     41    mix nic5-w[001-002,...]
batch*       up 2-00:00:00     28  alloc nic5-w[003-008,...]
hmem         up 2-00:00:00      3   idle nic5-w[071-073]
bio          up 62-00:00:0      1    mix nic5-w074
```

sinfo

The first line corresponds to the nodes in the **batch** partition that are in a **mix** state. This means that all the resources available on the node listed in the last column are not fully allocated.

```
 $ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
batch*       up 2-00:00:00     41    mix nic5-w[001-002,...]
batch*       up 2-00:00:00     28  alloc nic5-w[003-008,...]
hmem         up 2-00:00:00      3   idle nic5-w[071-073]
bio          up 62-00:00:0      1    mix nic5-w074
```

sinfo

The second line corresponds to the nodes in the **batch** partition that are in a **alloc** state. This means that all the resources available on the node listed in the last column are allocated.

```
 $ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
batch*       up 2-00:00:00     41    mix nic5-w[001-002,...]
batch*       up 2-00:00:00     28  alloc nic5-w[003-008,...]
hmem         up 2-00:00:00      3   idle nic5-w[071-073]
bio          up 62-00:00:0      1    mix nic5-w074
```

sinfo

The third line corresponds to the nodes in the **hmem** and **bio** partitions. You will not use these partitions.

```
 $ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
batch*       up 2-00:00:00     41    mix nic5-w[001-002,...]
batch*       up 2-00:00:00     28  alloc nic5-w[003-008,...]
hmem         up 2-00:00:00      3   idle nic5-w[071-073]
bio          up 62-00:00:0      1    mix nic5-w074
```

squeue

The **squeue** command allows you to view information about jobs that are currently running or waiting in the queue.

```
$ squeue
  JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
1549050 batch           Ti  aslassi PD       0:00      4 (Resources)
1549113 batch      SigX1.5  dtanner PD       0:00      1 (Priority)
...
1539952 batch     blast_te rdugauqu  R 1-05:11:32      1 nic5-w036
1547140 batch      CHL_APF meulders  R 1-03:23:31     13 nic5-w[016,030, ...]
```

squeue

This job is in a pending state (**PD**). This means that the job is waiting for the scheduler to grant permission to start. The reason why this job is waiting is the lack of **resources**.

```
$ squeue
  JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
1549050 batch           Ti  aslassi PD       0:00      4 (Resources)
1549113 batch      SigX1.5  dtanner PD       0:00      1 (Priority)
...
1539952 batch     blast_te rdugauqu  R 1-05:11:32      1 nic5-w036
1547140 batch      CHL_APF meulders  R 1-03:23:31     13 nic5-w[016,030, ...]
```

squeue

This job is in a pending state (**PD**) too. The reason why this job is waiting is that the **priority** of the user is too low. The scheduler runs jobs with higher priority first.

```
$ squeue
  JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
1549050 batch           Ti  aslassi PD       0:00      4 (Resources)
1549113 batch      SigX1.5  dtanner PD       0:00      1 (Priority)
...
1539952 batch     blast_te rdugauqu  R 1-05:11:32      1 nic5-w036
1547140 batch      CHL_APF meulders  R 1-03:23:31     13 nic5-w[016,030, ...]
```

squeue

This job is running (**R**) for 1 day, 5 hours and 11 minutes. It has been allocated 1 compute node (**nic5-w036**)

```
$ squeue
  JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
1549050 batch           Ti  aslassi PD       0:00      4 (Resources)
1549113 batch      SigX1.5  dtanner PD       0:00      1 (Priority)
...
1539952 batch     blast_te rdugauqu  R 1-05:11:32      1 nic5-w036
1547140 batch      CHL_APF meulders  R 1-03:23:31     13 nic5-w[016,030, ...]
```

squeue

This job is running (**R**) for 1 day, 3 hours and 23 minutes. It has been allocated 13 compute nodes

```
 $ squeue
  JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
1549050 batch           Ti  aslassi PD       0:00      4 (Resources)
1549113 batch      SigX1.5  dtanner PD       0:00      1 (Priority)
...
1539952 batch     blast_te rdugauqu  R 1-05:11:32      1 nic5-w036
1547140 batch      CHL_APF meulders  R 1-03:23:31     13 nic5-w[016,030, ...]
```

squeue

You can use **`squeue`** with the **`--me`** option to only see your jobs

```
$ squeue --me
  JOBID PARTITION     NAME     USER ST     TIME  NODES NODELIST(REASON)
1549033     batch     job1     user PD     0:00      6 (Priority)
1549034     batch     job2     user PD     0:00     12 (Priority)
1549035     batch     job3     user PD     0:00      4 (Priority)
1549036     batch     job4     user PD     0:00     10 (Priority)
```

You can use **`squeue`** with a combination of the **`--me --start`** options to have an estimate of when your job is scheduled to sta

```
$ squeue --me --start
  JOBID PARTITION NAME USER ST           START_TIME  NODES SCHEDNODES
1549033     batch job1 user PD  2021-09-27T13:07:37      7 nic5-w[001,...
```

sbatch

The **`sbatch`** command allows you to submit a batch script to Slurm. This script is a file with specific commands to Slurm as well as commands to execute on the compute nodes.

```
$ sbatch your_job_script.sh
Submitted batch job 1549033
```

Upon successful submission of the job to the queue, **`sbatch`** will return the ID that is assigned to your job. The job ID an important piece of information as it allows you to alter your job. For example, to cancel a job

```
$ scancel 1549033
```

You don't need to write down the IDs of your jobs. You can get it them at any time with **`squeue`**

sbatch

In a submission script, lines prefixed with `#SBATCH` and followed by a command are understood by Slurm as resource requests.

`--time`	Limit on the run time of the job
`--ntasks`	Maximum of number tasks (MPI ranks)
`--cpus-per-task`	Number of processors per task (threads)
`--mem-per-cpu`	Minimum memory required per allocated CPU
`--partition`	Request a specific partition for the resource allocation

sbatch

Important note about the Slurm terminology:

- a CPU is a core, not a CPU socket
- a task is an independent process, if you request more than one task, these tasks can be located on different compute nodes

You can control the way the tasks are distributed on the node using a combination of the `--nodes` and `--ntasks-per-node`

Typical Script for an OpenMP Program

```
#!/bin/bash -l
#SBATCH --time=0-01:00:00 # dd-hh:mm:ss
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=4
#SBATCH --mem-per-cpu=1024 # megabytes
#SBATCH --partition=batch

export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

module load releases/2020b
module load GCCcore/10.2.0

cd $SLURM_SUBMIT_DIR

./your_program
```

Typical Script for an OpenMP Program

```
#!/bin/bash -l
#SBATCH --time=0-01:00:00 # dd-hh:mm:ss
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=4
#SBATCH --mem-per-cpu=1024 # megabytes
#SBATCH --partition=batch

export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

module load releases/2020b
module load GCCcore/10.2.0

cd $SLURM_SUBMIT_DIR

./your_program
```

You have to start your script with this line

Typical Script for an OpenMP Program

```
#!/bin/bash -l
#SBATCH --time=0-01:00:00 # dd-hh:mm:ss
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=4
#SBATCH --mem-per-cpu=1024 # megabytes
#SBATCH --partition=batch

export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

module load releases/2020b
module load GCCcore/10.2.0

cd $SLURM_SUBMIT_DIR

./your_program
```

Maximum run time: 1 hour

Typical Script for an OpenMP Program

```
#!/bin/bash -l
#SBATCH --time=0-01:00:00 # dd-hh:mm:ss
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=4
#SBATCH --mem-per-cpu=1024 # megabytes
#SBATCH --partition=batch

export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

module load releases/2020b
module load GCCcore/10.2.0

cd $SLURM_SUBMIT_DIR

./your_program
```

Typical Script for an OpenMP Program

```
#!/bin/bash -l
#SBATCH --time=0-01:00:00 # dd-hh:mm:ss
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=4
#SBATCH --mem-per-cpu=1024 # megabytes
#SBATCH --partition=batch

export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

module load releases/2020b
module load GCCcore/10.2.0

cd $SLURM_SUBMIT_DIR

./your_program
```

Typical Script for an OpenMP Program

```
#!/bin/bash -l
#SBATCH --time=0-01:00:00 # dd-hh:mm:ss
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=4
#SBATCH --mem-per-cpu=1024 # megabytes
#SBATCH --partition=batch

export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

module load releases/2020b
module load GCCcore/10.2.0

cd $SLURM_SUBMIT_DIR

./your_program
```

Typical Script for an OpenMP Program

```
#!/bin/bash -l
#SBATCH --time=0-01:00:00 # dd-hh:mm:ss
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=4
#SBATCH --mem-per-cpu=1024 # megabytes
#SBATCH --partition=batch

export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

module load releases/2020b
module load GCCcore/10.2.0

cd $SLURM_SUBMIT_DIR

./your_program
```

Typical Script for an OpenMP Program

```
#!/bin/bash -l
#SBATCH --time=0-01:00:00 # dd-hh:mm:ss
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=4
#SBATCH --mem-per-cpu=1024 # megabytes
#SBATCH --partition=batch

export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

module load releases/2020b
module load GCCcore/10.2.0

cd $SLURM_SUBMIT_DIR

./your_program
```

Set the number of OpenMP threads

Typical Script for an OpenMP Program

```
#!/bin/bash -l
#SBATCH --time=0-01:00:00 # dd-hh:mm:ss
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=4
#SBATCH --mem-per-cpu=1024 # megabytes
#SBATCH --partition=batch

export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

module load releases/2020b
module load GCCcore/10.2.0

cd $SLURM_SUBMIT_DIR

./your_program
```

← Run your program

Typical Script for an MPI+OpenMP Program

```
#!/bin/bash -l
#SBATCH --time=0-01:00:00
#SBATCH --ntasks=8
#SBATCH --cpus-per-task=8
#SBATCH --mem-per-cpu=1024 # megabytes
#SBATCH --partition=batch

export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

module load releases/2020b
module load OpenMPI/4.0.5-GCC-10.2.0

cd $SLURM_SUBMIT_DIR

mpirun ./your_mpi_program
```

Typical Script for an MPI+OpenMP Program

```
#!/bin/bash -l
#SBATCH --time=0-01:00:00
#SBATCH --ntasks=8
#SBATCH --cpus-per-task=8
#SBATCH --mem-per-cpu=1024 # megabytes
#SBATCH --partition=batch

export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

module load releases/2020b
module load OpenMPI/4.0.5-GCC-10.2.0

cd $SLURM_SUBMIT_DIR

mpirun ./your_mpi_program
```

8 tasks (MPI processes)

Typical Script for an MPI+OpenMP Program

```
#!/bin/bash -l
#SBATCH --time=0-01:00:00
#SBATCH --ntasks=8
#SBATCH --cpus-per-task=8
#SBATCH --mem-per-cpu=1024 # megabytes
#SBATCH --partition=batch

export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

module load releases/2020b
module load OpenMPI/4.0.5-GCC-10.2.0

cd $SLURM_SUBMIT_DIR

mpirun ./your_mpi_program
```

Set the number of OpenMP threads per MPI process

Typical Script for an MPI+OpenMP Program

```
#!/bin/bash -l
#SBATCH --time=0-01:00:00
#SBATCH --ntasks=8
#SBATCH --cpus-per-task=8
#SBATCH --mem-per-cpu=1024 # megabytes
#SBATCH --partition=batch

export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

module load releases/2020b
module load OpenMPI/4.0.5-GCC-10.2.0

cd $SLURM_SUBMIT_DIR

mpirun ./your_mpi_program
```

← Run your program with `mpirun`

Environment Modules

Access to Softwares : Modules

Preinstalled software can be listed, enabled or disabled through the use of the module command. In the framework of this course, we are mainly interested in compilers.

- `module avail` : list available software
- `module load` : set up the environment to use the software
- `module list` : list currently loaded software
- `module purge` : clears the environment

Modules: Releases

On the CÉCI clusters, the modules are organized in releases. These releases are superset of modules. The default release is **releases/2019b**

This release is not ideal (older compiler not fully aware of the NIC5 CPU features). It's recommended to switch to the **2020b** release.

```
module load releases/2020b
```

and even to make it permanent with

```
echo -e "module load releases/2020b" >> .bashrc
```

Modules

The main module of interest for us is the GCC compiler

```
 $ module av gcc/ GCC/
------------------- Releases ( 2020b ) --------------------
   GCC/10.2.0
 $ module load GCC/10.2.0
 $ module list
Currently Loaded Modules:
...  3) releases/2020b  5) zlib/1.2.11-GCCcore-10.2.0     7) GCC/10.2.0
...  4) GCCcore/10.2.0  6) binutils/2.35-GCCcore-10.2.0
```

Modules

We look for the **`GCC`** module using the **`module av`** command. The result of the command tells us that there is a module named **`GCC/10.2.0`** available.

```
 $ module av gcc/ GCC/
------------------- Releases ( 2020b ) -------------------
   GCC/10.2.0
 $ module load GCC/10.2.0
 $ module list
Currently Loaded Modules:
...  3) releases/2020b  5) zlib/1.2.11-GCCcore-10.2.0     7) GCC/10.2.0
...  4) GCCcore/10.2.0  6) binutils/2.35-GCCcore-10.2.0
```

Modules

We load the **`GCC/10.2.0`** module with the **`module load`** command.

```
 $ module av gcc/ GCC/
-------------------- Releases ( 2020b ) --------------------
   GCC/10.2.0
 $ module load GCC/10.2.0
 $ module list
Currently Loaded Modules:
...  3) releases/2020b  5) zlib/1.2.11-GCCcore-10.2.0     7) GCC/10.2.0
...  4) GCCcore/10.2.0  6) binutils/2.35-GCCcore-10.2.0
```

Modules

Using the **`module list`** command we can see the **`GCC`** module is now loaded as well as its dependencies

```
 $ module av gcc/ GCC/
------------------- Releases ( 2020b ) -------------------
   GCC/10.2.0
 $ module load GCC/10.2.0
 $ module list
Currently Loaded Modules:
...  3) releases/2020b  5) zlib/1.2.11-GCCcore-10.2.0     7) GCC/10.2.0
...  4) GCCcore/10.2.0  6) binutils/2.35-GCCcore-10.2.0
```

Copying Files to and From NIC5

Copying to and from a Windows Computer

- MobaXTerm comes with a built-in SFTP client
- it's accessible from the "SFTP" tab on the left of the MobaXTerm window
- you can drag and drop file to the SFTP panel to copy files from your computer to cluster
- you can drag and drop file from the SFTP panel to copy files from the cluster to your computer

Copying to and from a Linux/MacOS computer

Copying files to and from the cluster from an UNIX-like OS can be done with the **`scp`** command.

- to copy file to the cluster the first argument to the command is the path to the file on your computer
- the second argument starts with **`nic5:`** to indicate a remote machine followed by the destination path

```
scp ~/INFO0939/project1.c nic5:~/src/
```

- Copying files from the cluster is the other way around

```
scp nic5:~/src/project1.c ~/INFO0939/
```

Copying to and from a Linux/MacOS computer

Copying directories to and from the cluster from an UNIX-like OS can be done with the **`scp`** command with **`-r`** option.

For example

```
scp -r ~/INFO0939 nic5:
```

copy the **`INFO0939`** directory from your computer to NIC5.

Introduction to UNIX

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 log into 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.

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 folders 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 folders, 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 folders, 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 be 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`** .

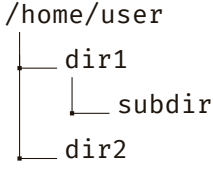
The current directory

- The hidden `.` represents the current directory
- It may be 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.

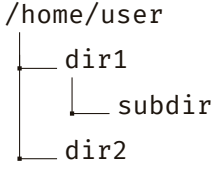
Changing directory



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

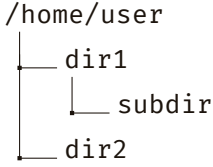


To go to the parent directory (the directory in which your working directory is located) you can use `..`

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

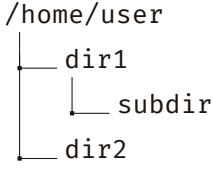
Changing directory

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
```

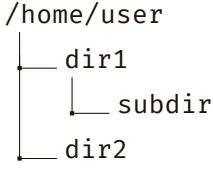
Changing directory



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
```

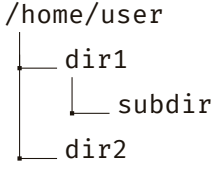
Changing directory



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

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

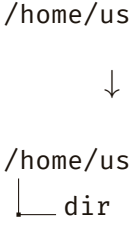
Changing directory



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
```

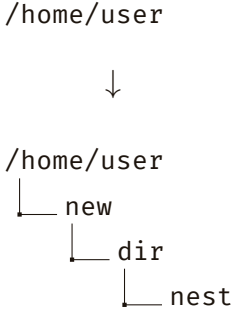
Create Directories



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

```
$ mkdir dir
```

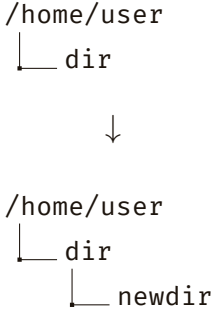
Create Directories



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

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

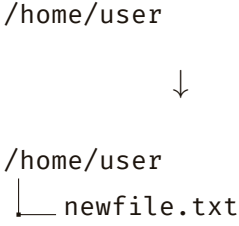
Create Directories



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

```
$ mkdir dir/newdir
```

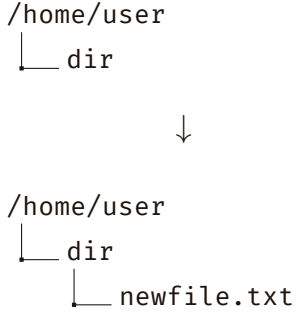
Create Files



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

```
$ touch newfile.txt
```

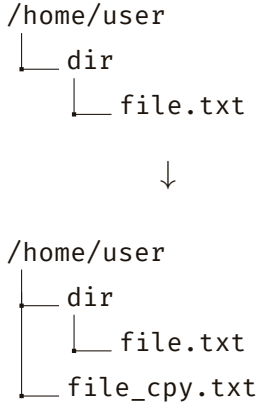
Create Files



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
```

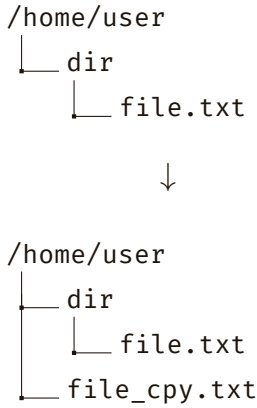
Copying Files



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
```

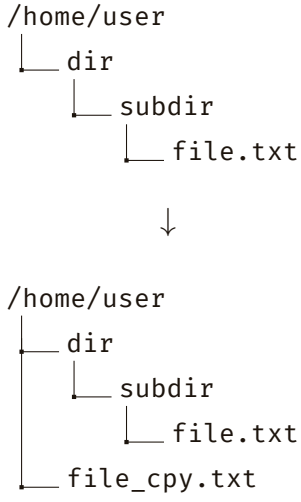
Copying Files



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
```

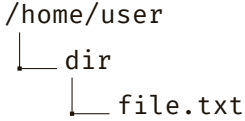
Copying Files



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
```

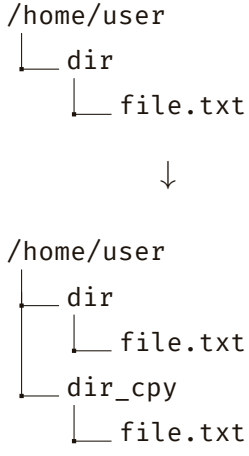
Copying Directories



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

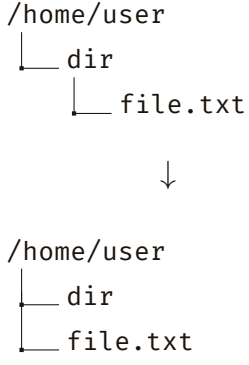


To copy a directory, we need to use the recursive option

`-r`

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

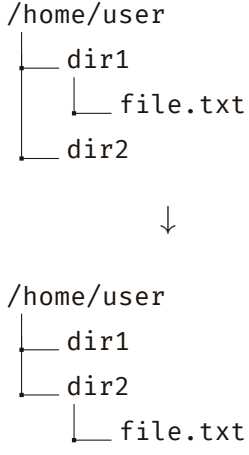
Moving Files



Moving 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
```

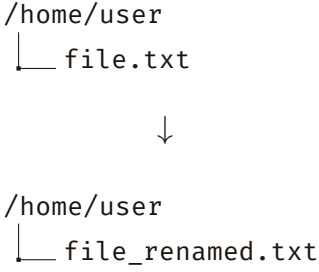
Moving Files



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
```

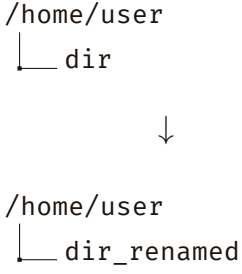
Renaming Files and Directories



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

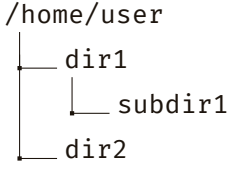
```
$ mv dir dir_renamed
```

Absolute and Relative Path

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 `/`

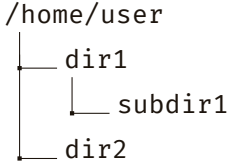
Absolute and Relative Path



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
```

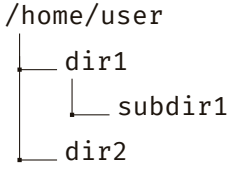
Absolute and Relative Path



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

The **cat** command allows you 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

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
```

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 transfers 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