Contents

Welcome to the HPC filesystem tutorial. This guide will teach you all the necessary commands and info on how to work in linux within the context of an HPC environment.

For an extended overview of HPC-documentation I refer to the HPC-DOCS:
https://hpcugent.github.io/vsc_user_docs/

1. Introduction to Linux on HPC

1.1 Getting Started

To get started with the HPC-UGent infrastructure, you need to obtain a VSC account, see HPC manual. Keep in mind that you must keep your private key to yourself! You can look at your public/private key pair as a lock and a key: you give us the lock (your public key), we put it on the door, and then you can use your key to open the door and get access to the HPC infrastructure. Anyone who has your key can use your VSC account! Details on connecting to the HPC infrastructure are available in HPC manual connecting section.

NOTE: If you plan to work in the webbrowser interface only, then you don't need to upload public/private key. For this course, the webbrowse interface only is ok.

1.2 The prompt

The basic interface is the so-called shell prompt, typically ending with `$` (for bash shells).

NOTE: In the `code` examples below the `$` indicates your prompt. This is for information purpose only, you don't need to type this in your commands.

You use the shell by executing commands, and hitting `<enter>`. For example:

```
$ echo hello
hello
```

While typing (long) commands in the terminal, you can go to the start or end of the command line using `Ctrl-A` or `Ctrl-E`.

To go through previous commands, use `<up>` and `<down>`, rather than retyping them.

1.3 Basic Commands

Download example data

For this exercise you'll need to connect to the HPC and open a terminal. I've prepared some folders and files on which you can perform following commands.

TODO Start by copying the `/Linux_Basics_exampledata` to your home folder `~/`.

```
$ cp /path/to/ONT-SEQ_PA-Benin2024_Bioinfocourse/Linux_Basics_exampledata ~/
```

Commands & Examples **NOTE:** Make sure you are in your home folder before you start with the exercises.
To go to your home folder use following command:

```
$ cd ~/
```

Example 1: Lists files and directories in the current directory.
```
$ ls -l
```

Example 2: Changes the current directory.
```
$ cd /user/gent/433/vscXXXX//ONT-SEQ_PA-Benin2024_Bioinfocourse
```

Example 3: Prints the current working directory.
```
$ pwd
```

Example 4: Creates a new directory.
```
$ mkdir my_first-folder
```

Example 5: Removes an empty directory.
```
$ rmdir old_folder
```

Example 6: Deletes a file. WARNING: There is no trash, files are permanently deleted!
```
$ rm ./old_hello.py
```

Example 7: Deletes a directory and its contents recursively.
```
$ rm -r ./old_sequences
```

Example 8: Copies files or directories.
```
$ cp samplesheet96.txt ./my_first-folder
```

Example 9: Moves or renames files or directories.
```
$ mv ./my_first-folder/samplesheet96.txt ./my_first-folder/samplesheet96_$(date
+%Y-%m-%d).txt
```

Example 10: Searches for a file by name in a specified path.
```
$ find ~/ -name "samplesheet96*"
```

Example 11: Displays the contents of a file.
```
$ cat hello.py
```

Example 12: Opens a file for viewing one page at a time. Press q to quit.
```
$ less manual_cp.txt
```

Example 13: Shows the first 10 lines of a file.
```
$ head manual_cp.txt
```

Example 14: Shows the last 10 lines of a file.
```
$ tail manual_cp.txt
```

Example 15: Displays detailed information about files, including permissions.
```
$ ls -lh ~/Linux_Basics_exampledata
```

Example 16: Shows currently running processes.
```
$ ps
```

Example 17: Displays real-time system resource usage.
`$ top`

Example 18: Interactive process viewer (if installed).
`$ htop`

Example 19: Prints text to the terminal.
`$ echo "Hello, world!"`

Example 20: Displays the current date and time.
`$ date`

Example 21: Logs out or closes the terminal.
`$ exit`

1.4 Manipulating files and directories

Changing permissions: "chmod"
Each file and directory has particular permissions set on it, which can be queried using ls -l.

For example:

```
$ ls -l afile.txt
-rw-rw-r-- 1 vsc40000 agroup 2929176 Apr 12 13:29 afile.txt
```

The -rwxrw-r– specifies both the type of file (- for files, d for directories (see first character)), and the permissions for user/group/others:

1. each triple of characters indicates whether the read (r), write (w), execute (x) permission bits are set or not
2. the 1st part rwx indicates that the owner "vsc40000" of the file has all the rights
3. the 2nd part rw- indicates the members of the group "agroup" only have read/write permissions (not execute)
4. the 3rd part r– indicates that other users only have read permissions

The default permission settings for new files/directories are determined by the so-called umask setting, and are by default:

1. read-write permission on files for user/group (no execute), read-only for others (no write/execute)
2. read-write-execute permission for directories on user/group, read/execute-only for others (no write)

Any time you run `ls -l` you'll see a familiar line of `-rwx------` or similar combination of the letters `r`, `w`, `x` and `-` (dashes). These are the permissions for the file or directory.

```
$ ls -l
total 1
-rw-r--r--. 1 vsc40000 mygroup 4283648 Apr 12 15:13 articleTable.csv
drwxr-x---. 2 vsc40000 mygroup 40 Apr 12 15:00 Project_GoldenDragon
```

> Here, we see that `articleTable.csv` is a file (beginning the line with -) has read and write permission for the user `vsc40000` (`rw-`), and read permission for the group `mygroup` as well as all other users (`r--` and `r--`).

The next entry is `Project_GoldenDragon`. We see it is a directory because the line begins with a `d`. It also has read, write, and execute permission for the `vsc40000` user (`rwx`). So that user can look into the directory and add or remove files. Users in the `mygroup` can also look into the directory and read the files. But they can't add or remove files (`r-x`). Finally, other users can read files in the directory, but other users have no permissions to look in the directory at all (`---`).

Maybe we have a colleague who wants to be able to add files to the directory. We use `chmod` to change the modifiers to the directory to let people in the group write to the directory:

```
$ chmod g+w Project_GoldenDragon
$ ls -l
total 1
-rw-r--r--. 1 vsc40000 mygroup 4283648 Apr 12 15:13 articleTable.csv
drwxrwx---. 2 vsc40000 mygroup 40 Apr 12 15:00 Project_GoldenDragon
```

> The syntax used here is `g+x` which means group was given write permission. To revoke it again, we use `g-w`. The other roles are `u` for user and `o` for other.

You can put multiple changes on the same line: `chmod o-rwx,g-rxw,u+rx,u-w somefile` will take everyone's permission away except the user's ability to read or execute the file.

You can also use the `-R` flag to affect all the files within a directory, but this is dangerous. It's best to refine your search using find and then pass the resulting list to chmod since it's not usual for all files in a directory structure to have the same permissions.

Example 1: Apply chmod to all files (regular files only) within a directory
This will set 644 permissions for all files (not directories) `find /path/to/directory -type f -exec chmod 644 {} \;`

Example 2: Apply chmod to all directories within a directory
This will set 755 permissions for all directories (not files) `find /path/to/directory -type d -exec chmod 755 {} \;`

Example 3: Apply chmod to all files modified in the last 7 days
This will search for all files modified in the last 7 days and set 644 permissions `find /path/to/directory -type f -mtime -7 -exec chmod 644 {} \;`

Example 4: Apply chmod to files with a specific extension (e.g., .txt)
This will search for all .txt files and set 644 permissions `find /path/to/directory -type f -name "*.txt" -exec chmod 644 {} \;`

Example 5: Apply chmod recursively with different permissions for files and directories
First, set 644 permissions for all files (regular files) `find /path/to/directory -type f -exec chmod 644 {} \;`

Then, set 755 permissions for all directories `find /path/to/directory -type d -exec chmod 755 {} \;`

1.4.1.1 Exercises: Changing permissions with "chmod"

Here are some exercises on chmod based on your directory structure. Try each command and observe the changes using `ls -l` before and after.

Exercise 1: Make `hello.sh` Executable
Goal: Grant execute (x) permission to the script so it can run as a program.

```
chmod +x hello.sh
ls -l hello.sh
```

Check if the x permission appears for the user (`rwxr--r--`). Now try running it:

```
./hello.sh
```

Exercise 2: Remove Read Permissions from `manual_cp.txt`
Goal: Prevent yourself and others from reading the file.

```
chmod a-r manual_cp.txt
ls -l manual_cp.txt
```

Now try opening it:

```
cat manual_cp.txt  # Should give a "Permission denied" error
```

Restore permissions after testing:

```
chmod u+r manual_cp.txt
```

Exercise 3: Grant Full Access to `samplesheet96.txt` for Everyone
Goal: Allow all users to read, write, and execute `samplesheet96.txt`.

```
chmod 777 samplesheet96.txt
ls -l samplesheet96.txt
```

*Now everyone can modify and execute the file. This is **not recommended** for sensitive files!*

Exercise 4: Restrict `old_hello.py` to Read and Write for Owner Only
Goal: Make the file private so only the owner can read and modify it.

```
chmod 600 old_hello.py
ls -l old_hello.py
```

Now other users cannot read or modify it.

Exercise 5: Add a Sticky Bit to `sequences`
Goal: Ensure that only the file owner can delete their own files inside `sequences`, even if

others have write access.

```
chmod +t sequences
ls -ld sequences  # Check for the "t" at the end of the permissions (drwxr-xr-t)
```

Now, even if other users have write permissions, they cannot delete files they don't own.

Exercise 6: Recursively Set Read & Execute for All Users in `old_sequences`
Goal: Ensure all files in `old_sequences` are readable and executable but not writable by others.

```
chmod -R a+rx old_sequences
ls -l old_sequences/
```

Check if all files inside now have `r-x` for everyone.

Exercise 7: Set Group Write Permissions for `Tabular-file.tab`
Goal: Allow your group members to edit the file.

```
chmod g+w Tabular-file.tab
ls -l Tabular-file.tab
```

Now, users in the same group (`vsc43352`) can modify the file.

Exercise 8: Remove Execute Permission from `hello.py`
Goal: Prevent accidental execution of a Python script.

```
chmod -x hello.py
ls -l hello.py
```

Now, you must explicitly use `python3 hello.py` instead of `./hello.py`.

Exercise 9: Convert Between Symbolic and Octal Modes
Goal: Change `samplesheet96_2025-04-02.txt` to the same permissions as `samplesheet96.txt` using octal mode. 1. **Check the original permissions:** `bash    ls -l samplesheet96.txt` 2. **Use stat to see the octal mode:** `bash    stat -c "%a" samplesheet96.txt` Suppose it returns 644. 3. **Apply the same mode to the new file:** `bash    chmod 644 samplesheet96_2025-04-02.txt`

Bonus Challenge: Set sequences to 750
Goal: Make `sequences` accessible only to the owner and group, while others have no access.

```
chmod 750 sequences
ls -ld sequences
```

Now only you and your group can access it, while others are blocked.

Final Check After completing the exercises, list all files again to see the changes:

```
ls -l
```

Compressing files (zip, unzip, tar)
Files should usually be stored in a compressed file if they're not being used frequently. This means they will use less space and thus you get more out of your quota. Some types of files (e.g., CSV files with a lot of numbers) compress as much as 9:1. The most commonly used compression format on Linux is gzip. To compress a file using gzip, we use:

```
$ ls -lh myfile
-rw-r--r--. 1 vsc40000 vsc40000 4.1M Dec 2 11:14 myfile
$ gzip myfile
$ ls -lh myfile.gz
-rw-r--r--. 1 vsc40000 vsc40000 1.1M Dec 2 11:14 myfile.gz
```

Note: if you zip a file, the original file will be removed. If you unzip a file, the compressed file will be removed. To keep both, we send the data to stdout and redirect it to the target file:

```
$ gzip -c myfile > myfile.gz
$ gunzip -c myfile.gz > myfile
```

"zip" and "unzip"

Windows and macOS seem to favour the zip file format, so it's also important to know how to unpack those. We do this using unzip:

```
$ unzip myfile.zip
```

If we would like to make our own zip archive, we use zip:

```
$ zip myfiles.zip myfile1 myfile2 myfile3
```

Working with tarballs: "tar"

Tar stands for "tape archive" and is a way to bundle files together in a bigger file.

You will normally want to unpack these files more often than you make them. To unpack a `.tar` file you use:

```
$ tar -xf tarfile.tar
```

Often, you will find `gzip` compressed `.tar` files on the web. These are called tarballs. You can recognize them by the filename ending in `.tar.gz`. You can uncompress these using gunzip and then unpacking them using tar. But tar knows how to open them using the `-z` option:

```
$ tar -zxf tarfile.tar.gz
$ tar -zxf tarfile.tgz
```

Order of arguments

Note: Archive programs like `zip`, `tar` use arguments in the "opposite direction" of copy commands.

```
$ cp source1 source2 source3 target
$ zip zipfile.zip source1 source2 source3
$ tar -cf tarfile.tar source1 source2 source3
```

If you use `tar` with the source files first then the first file will be overwritten. You can control the order of arguments of tar if it helps you remember:

$ tar -c source1 source2 source3 -f tarfile.tar Exercises: Compressing files (zip, unzip, tar)

Here are some **zip, unzip, and tar** exercises based on your directory structure. Each exercise includes commands and explanations.

Exercise 1: Create a `.zip`file from a local file.
Goal: Create a `.zip` archive containing `manual_cp.txt`.

```
zip manual_cp.zip manual_cp.txt
ls -l manual_cp.zip
```

Now the file is compressed into `manual_cp.zip`.

Exercise 2: Extract `manual_cp.zip`
Goal: Extract the zipped file back to its original form.

```
unzip manual_cp.zip
ls -l
```

The extracted file should appear in the directory.

Exercise 3: Compress Multiple Files into One ZIP Archive
Goal: Create a `.zip` file containing `hello.py`, `hello.sh`, and `Tabular-file.tab`.

```
zip my_archive.zip hello.py hello.sh Tabular-file.tab
ls -l my_archive.zip
```

All three files are stored in `my_archive.zip`.

Exercise 4: Compress the `sequences` Directory Using ZIP
Goal: Create a `.zip` archive of the `sequences` directory.

```
zip -r sequences.zip sequences
ls -l sequences.zip
```

The `-r` flag ensures that all files inside the directory are included.

Exercise 5: Extract `sequences.zip`
Goal: Extract the entire directory from the `.zip` archive.

```
unzip sequences.zip
ls -l
```

The `sequences` directory is restored.

Exercise 6: Create a Tarball (`.tar`) Without Compression
Goal: Archive multiple files into a `.tar` file without compression.

```
tar -cf archive.tar hello.py hello.sh Tabular-file.tab
ls -l archive.tar
```

*The `.tar` file now contains all three files but is **not compressed**.*

Exercise 7: Create a Gzipped Tarball (`.tar.gz`)
Goal: Archive and compress the `old_sequences` directory.

```
tar -czf old_sequences.tar.gz old_sequences
ls -l old_sequences.tar.gz
```

The `-z` flag enables gzip compression, creating a smaller file.

Exercise 8: Extract a `.tar.gz` File
Goal: Extract the `old_sequences.tar.gz` archive.

```
tar -xzf old_sequences.tar.gz
ls -l
```

The `old_sequences` directory is restored.

Exercise 9: List Contents of a `.tar.gz` File Without Extracting
Goal: View files inside `old_sequences.tar.gz` without extracting.

```
tar -tzf old_sequences.tar.gz
```

This shows all files inside the tarball.

Exercise 10: Extract Only `samplesheet96.txt` From `archive.tar`
Goal: Extract a single file from a tar archive.

```
tar -xf archive.tar samplesheet96.txt
ls -l samplesheet96.txt
```

Only `samplesheet96.txt` is extracted.

Bonus Challenge: Tar a Directory and Extract It to Another Location
Goal: Archive `sequences` and extract it elsewhere.

```
tar -czf sequences.tar.gz sequences
mkdir test_restore
tar -xzf sequences.tar.gz -C test_restore
ls test_restore/
```

The `sequences` directory is extracted into `test_restore` instead of the current directory.

Final Check
After completing the exercises, list your directory contents:

```
ls -l
```

2. Nanopore data

Remember that each of you extracted DNA and did PCR on his/her own samples in Benin last year.
You can find more detailed list with extra information in the teams *General* channel: `Document > General > Specieslist-overview`

To keep the running time of the analysis minimal (more data is longer runtime), I suggest to select **no more than 10 Barcodes** each. ### 2.1 Data structure

TODO
But first things first.
Have a look at the content of the data folder **PATH TO CHARED DATAFOLDER**. You should be able to navigate to this folder by now.
Hint: you can use the `ls` command, or `cd` to access the directory and use the command `tree` to create an overview of the content.

Main folder
The structure looks like this:

```
path/to/inputdata/ONT-SEQ_PA-Benin2024_Input-Data/
.
+-- ONT-SEQ-24_PA-01
|   +-- fastq_pass
|   |   +-- barcode01
|   |   +-- ...
|   |   \-- barcode24
|   +-- reference.fa
|   +-- report_AMF003_20241018_1203_c8d4ac6d.html
|   \-- samplesheet_ONT-SEQ-24_01-withreference.txt
+-- ONT-SEQ-24_PA-02
|   +-- fastq_pass
|   |   +-- barcode01
|   |   +-- ...
|   |   +-- barcode24
|   +-- reference.fa
|   +-- report_axf308_20241022_1153_b65829bf.html
|   \-- samplesheet_ONT-SEQ-24_02-withreference.txt
+-- ONT-SEQ-25_PA-04
|   +-- fastq_pass
|   |   +-- barcode01
|   |   +-- ...
|   |   +-- barcode96
|   +-- report_FBA60703_20250120_1336_292963a6.html
|   \-- samplesheet_ONT-SEQ-25_04.txt
+-- ONT-SEQ-25_PA-05
|   +-- fastq_pass
|   |   +-- barcode01
|   |   +-- ...
|   |   +-- barcode48
|   +-- report_FBA60703_20250128_1308_e2391328.html
|   \-- samplesheet_ONT-SEQ-25_05.txt
+-- information
\-- scripts
```

fastq-pass folder
Each `barcode` folder contains the reads for this particular barcode in `fastq.gz` format:

```
../fastq_pass/barcode01$tree
.
+-- AMF003_pass_barcode01_c8d4ac6d_b8b1e3e8_0.fastq.gz
+-- AMF003_pass_barcode01_c8d4ac6d_b8b1e3e8_10.fastq.gz
+-- AMF003_pass_barcode01_c8d4ac6d_b8b1e3e8_11.fastq.gz
+-- AMF003_pass_barcode01_c8d4ac6d_b8b1e3e8_12.fastq.gz
+-- AMF003_pass_barcode01_c8d4ac6d_b8b1e3e8_13.fastq.gz
+-- AMF003_pass_barcode01_c8d4ac6d_b8b1e3e8_14.fastq.gz
+-- AMF003_pass_barcode01_c8d4ac6d_b8b1e3e8_15.fastq.gz
```

4 Sequence experiments were performed on your samples: *ONT-SEQ-24_PA-0X*.
You can find more detailed list with extra information in the teams *General* channel:
`Document > General > Specieslist-overview`

1. ONT-SEQ-24_PA-01: 24 amplicons - ITS+LSU sequences `(*)`
2. ONT-SEQ-24_PA-02: 24 amplicons - ITS+LSU sequences `(*)`
3. ONT-SEQ-24_PA-04: 96 amplicons - ITS sequences

4. ONT-SEQ-24_PA-05: 48 amplicons - ITS

`(*)` That means both ITS and LSU amplicon products for the **same** sample where mixed befor sequencing. The bioinformatic tool we will able to construct both ITS and LSU sequences at the same time.

The 2 other experiments (5 and 6) contain only ITS sequences per specimen.

Sequence folder

This folder contains the following:

- `fastq_pass` the passed fastq files i.e. your sequence reads

- `reference.fa` a reference file with an ITS and a LSU sequence, only for the reads from 1st and 2nd experiment

- `report...html` a sequence report file, general info on the sequence run (including all samples)

- `samplesheet_....txt`a samplesheet (with the MBB number and sequence barcode).

```
/ONT-SEQ_PA-Benin2024_Input-Data/ONT-SEQ-24_PA-01$tree -L 1
.
+-- fastq_pass
+-- reference.fa
+-- report_AMF003_20241018_1203_c8d4ac6d.html
\-- samplesheet_ONT-SEQ-24_01-withreference.txt
```

Take a look inside the folder and the samplesheet file. *Hint: use the command `cd`, `ls` and `cat`
You will notice that opening a `html` file is not possible in the terminal.
To look at this file, you'll need to download it.

2.2 Download Test dataset

TODO
Before we start with the actual field data, we first are going to run a test dataset, to see if everything runs smoothly.

1. Copy the folder `/ONT-SEQ_PA-Benin2024_Input-Data/test_data` to your home folder

2. Verify the content and check if you have the sequence folders, the samplesheet and the reference file (for variant mode).

```
vsc43352@gligar09:~$ls
test_data
vsc43352@gligar09:~$ cd test_data
vsc43352@gligar09:~$/test_data$tree -L 2
.
+-- consensus_mode
|   +-- barcode01
|   +-- barcode02
|   \-- samplesheet_ONT-SEQ-25_05.txt
\-- variant_mode
    +-- barcode12
    +-- barcode14
    +-- reference.fa
    \-- samplesheet_ONT-SEQ-24_02-withreference.txt

6 directories, 3 files
```

2.3 Download Actual dataset

Each *ONT-SEQ-24_PA-0X* folder contains a samplesheet.

```
../ONT-SEQ_PA-Benin2024_Input-Data/ONT-SEQ-24_PA-01$cat samplesheet_ONT-SEQ-24_01-wi

barcode,alias,type,ref
barcode01,MBB-24-001_barcode01_ONT-SEQ-24_01,test_sample,ITS LSU
barcode02,MBB-24-002_barcode02_ONT-SEQ-24_01,test_sample,ITS LSU
barcode03,MBB-24-003_barcode03_ONT-SEQ-24_01,test_sample,ITS LSU
barcode04,MBB-24-004_barcode04_ONT-SEQ-24_01,test_sample,ITS LSU
...
```

In this samplesheet `ONT-SEQ-24_PA-01-withrefeence.txt` you can see that (for this sequence experiment only!)
`barcode01 = MBB-24-001`.

Have a look in the master species list in teams:
You can find more detailed list with extra information in the teams *General* channel:
`Document > General > Specieslist-overview`.

Important: Sequence selection!

- First decide from which sequence experiment you want to select samples
- Then select no more than 10 (i.e. 10 barcode folders)

Example workflow - commands to run:

- I want to select samples from sequence experiment `ONT-SEQ-24_PA-01`
- I would like to build consensus sequences for barcode01 - 10

```
$ cd $VSC_DATA # change directory to your VSC-DATA folder
$ mkdir Analysis01 # make a directory to copy your sequence reads to
$ cd Analysis01 # change to the created folder
$ cp -r ../ONT-SEQ_PA-Benin2024_Input-Data/ONT-SEQ-24_PA-01/barcode01 $VSC_DATA/Anal
$ ls $VSC_DATA/Analysis01 # check the content of the folder, the barcode01 folder sh
```

- Don't forget to copy also the samplesheet from this `ONT-SEQ-24_PA-01` folder to your `$VSC_DATA/Analysis01` folder. Once copied over, you can delete the lines with the barcodes you don't use.

3. EPI2ME: wf-amplicon