BinAC 2 Training

Part 1: Linux and bash basics

University of Tübingen · High Performance and Cloud Computing Division

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN



BinAC 2

By 12:00 you can do this

```
$ for f in ~/training/fastq/*_R1.fastq.gz; do
>     echo "$f"
>     zcat "$f" | wc -l
> done
/home/tu/tu_tu/tu_abc01/training/fastq/sample_1_R1.fastq.gz
100000
/home/tu/tu_tu/tu_abc01/training/fastq/sample_2_R1.fastq.gz
125000
/home/tu/tu_tu/tu_abc01/training/fastq/sample_3_R1.fastq.gz
80000
```

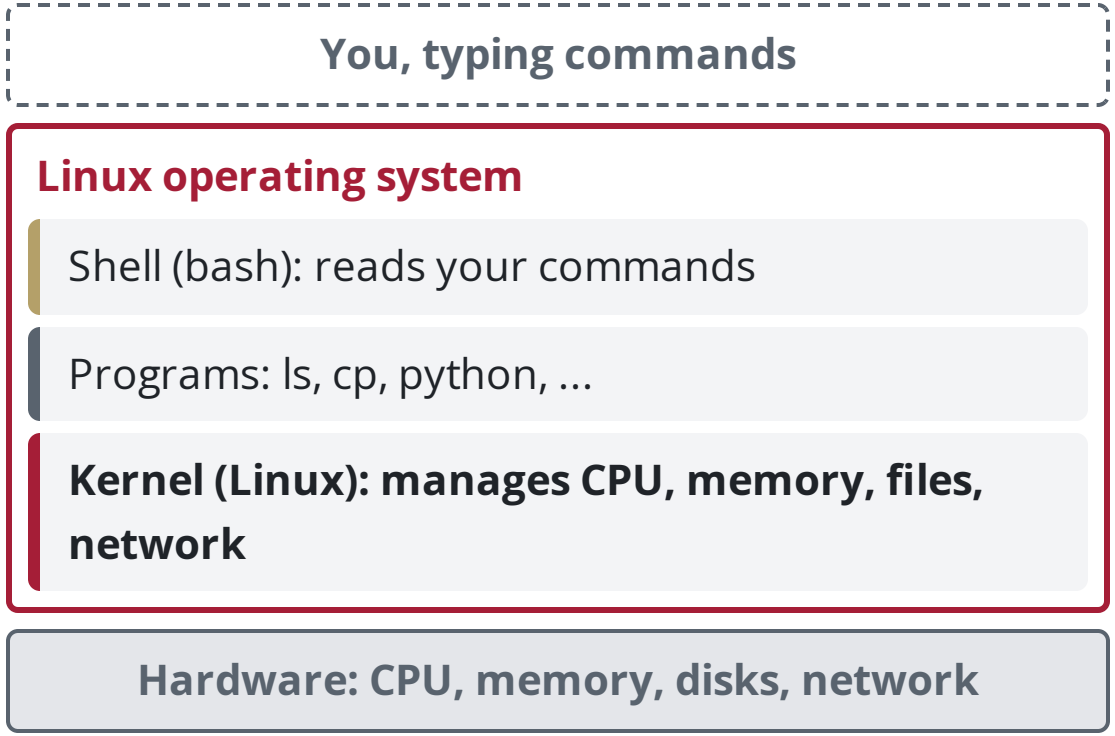
- Log in and find your way around BinAC 2
- Create, copy, search and inspect files, also many at once
- Copy data between your computer and BinAC 2
- Combine commands with pipes and write small scripts
- Find help yourself

Agenda

1. Linux, the shell and logging in
2. Help yourself: `--help`, `tldr`, `man`, `info`
3. The file system and paths
4. Working with files, wildcards
5. Copying data to and from BinAC 2
6. Viewing, editing and inspecting files
7. Permissions, pipes and redirection
8. Searching for files and text
9. Variables and shell scripts

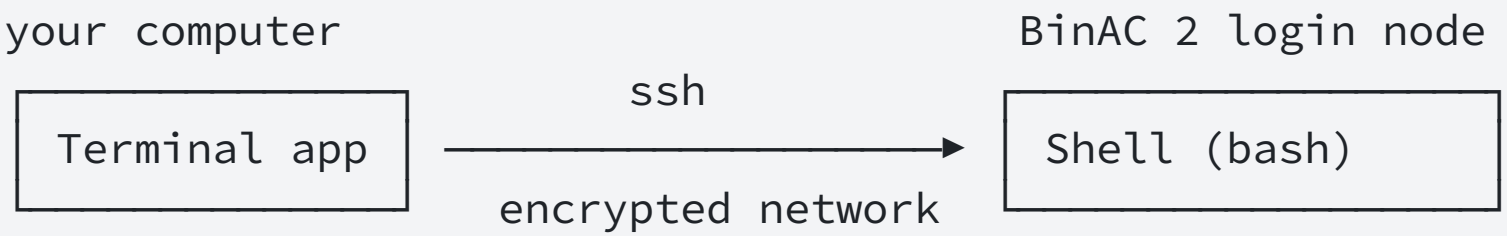
Break around 10:30 · Lunch at 12:00

What is Linux?



- **Kernel:** the core, talks to the hardware
- **Operating system:** kernel plus tools and programs, packaged as a **distribution**
- Many basic tools like `ls`, `cp` and `bash` come from the GNU project, so the whole system is also called **GNU/Linux**
- BinAC 2 runs **Rocky Linux 9**; others: Ubuntu, Debian, Fedora
- Almost all supercomputers run Linux

Terminal, SSH and shell



- **Terminal:** the window on your computer that shows text
- **SSH:** the secure connection to BinAC 2
- **Shell:** the program that reads your command, runs it, prints the result

The **prompt** tells you where you are:

```
[tu_abc01@login01 ~]$
 └──┬──┘ └──┬──┘ └─ current directory (~ = home)
    │       │
  user    host
```

Logging in

Connecting with SSH

```
ssh USERNAME@login.binac2.uni-tuebingen.de
```

- After login, every command runs **on BinAC 2**, not on your computer: check the prompt
- Your local terminal: Linux/macOS Terminal, Windows Terminal or MobaXterm
- Log in with your **service password**, not your university password
- First login: answer `yes` to "Are you sure you want to continue connecting?"
- The password is not shown while you type: that is normal

Anatomy of a command

```
ls -t --reverse /home
│  │  │         └── argument
│  │  └── long option (same as -r)
│  └── short option (sort by time)
└── command
```

- Options change the behaviour, arguments say what to work on
- `man ls` or `ls --help` shows all options

Things that surprise new users

```
$ cp input.txt backup.txt
$                                    ← no message: it worked
$ ls Input.txt
ls: cannot access 'Input.txt': No such file or directory
```

- **No news is good news:** most commands print nothing when they succeed
- **No undo, no recycle bin:** `rm`, `mv`, `cp` and `>` delete or overwrite without asking (`-i` makes `rm`, `mv` and `cp` ask first)
- **Case sensitive:** `data.txt`, `Data.txt` and `DATA.TXT` are three different files
- **No spaces in file names:** use `my_data.txt`, not `my data.txt`

Read the error message

```
$ cd trainig
bash: cd: trainig: No such file or directory
```

Read from the right: **what** went wrong, with **which argument**, in **which command**

Message	Usual cause and fix
`No such file or directory`	typo, case, wrong directory: `pwd`, `ls`, `Tab`
`command not found`	typo, or software not loaded (`module load`)
`Permission denied`	not executable (`chmod u+x`), or not your file
`Is a directory`	file and directory mixed up

Keyboard survival kit

Keys	Action
`Ctrl+C`	cancel the running command (it does **not** copy!)
`Ctrl+Shift+C` / `Ctrl+Shift+V`	copy / paste in most Linux terminals
`↑` `↓`	previous / next command
`Ctrl+R`	search your command history, `Enter` to run
`Tab`	complete a name, press twice to see all options
`q`	leave `man` and `less`
`Ctrl+L` or `clear`	clear the screen

Help yourself

Nobody remembers all options

Look things up in this order:

1. `command --help` : short overview, built into most programs
2. `tldr command` : practical examples
3. `man command` : the full manual
4. Search engine: error message or question
5. LLM (chatbot): explanations and first drafts
6. Our documentation and support

`--help` and `-h`

```
cp --help
grep --help | less
git help commit           # programs with subcommands
help cd                   # shell built-ins like cd
```

- `--help` works for most programs
- `-h` is **not always help**:
  - `ls -h`, `df -h`: human readable sizes
  - `grep -h`: hide file names
- If in doubt, use `--help`
- Subcommands (`git`, `conda`, `module`): `git help commit`, `conda create --help`

`tldr` : examples first

```
tldr tar
```

```
  Archiving utility.
  Often combined with a compression method, such as `gzip` or `bzip2`.

  [c]reate an archive and write it to a [f]ile:

      tar cf path/to/target.tar path/to/file1 path/to/file2 ...

  [c]reate a g[z]ipped archive and write it to a [f]ile:

      tar czf path/to/target.tar.gz path/to/file1 path/to/file2 ...
```

- Community maintained examples for common tasks
- The letters in `[ ]` explain the short options: `c` create, `z` gzip, `f` file
- Not complete: check `--help` or `man` for details

`man`: the full manual

```
man ls
```

Section	Content
NAME	one line: what the command does
SYNOPSIS	how to call it
DESCRIPTION, OPTIONS	all options in detail
EXAMPLES, SEE ALSO	at the end, if present

Reading the SYNOPSIS: `ls [OPTION]... [FILE]...`

- `[ ]` optional, `...` can be repeated
- `UPPER_CASE`: replace with your own value

Navigating a man page

Key	Action
`Space` / `b`	page down / up
`↓` `↑`	line down / up
`/text`	search forward, `n` next match, `N` previous
`g` / `G`	start / end of the page
`h`	help with all keys
`q`	quit

- Find an option quickly: `/-t` or `/--sort`
- The same keys work in `less`, which you will use later today

`info`: the long manual of GNU tools

```
info ls           # manual for ls
info coreutils    # whole manual
```

- Many man pages end with: "The full documentation is maintained as a Texinfo manual": that is `info`
- Navigation differs from `man` and `less`: lost? `q` and start again

Key	Action
`Space` / **`Backspace`**	page down / up
`Enter`	follow link (lines with `*`)
`n` / **`p`**	next / previous section
`u`	up one level
`l`	back to where you were
`s`	search
`q`	quit

`apropos`: which command do I need?

```
apropos copy              # same as: man -k copy
apropos -a copy file      # all keywords must match
```

```
cp (1)                - copy files and directories
```

- Searches the NAME line of all man pages
- Use general words: `copy`, `compress`, `space`, `archive`
- Too many results? Add a second keyword with `-a`

Exercise: Find it out

Goal: answer questions about commands without asking us

1. Which option of `ls` sorts by modification time?
2. What does `wc -c /etc/hostname` count?
3. How do you make `cp` ask before overwriting a file?
4. Find a command that compresses files

Expected result: four answers, and for each the command that told you

> **Help yourself:** `--help`, `tldr`, `man`, `apropos`

Bonus: which `ls` option sorts by file size?

The file system

The file system is a tree

```
/                                ← root directory: Top most directory in hierarchy
├── etc
├── home
│   └── tu
│       ├── tu_abc01             ← your home directory (~)
│       │   ├── notes.txt
│       │   └── training
│       │       ├── data
│       │       │   └── input.txt
│       │       └── results
│       └── tu_xyz02             ← another user
└── usr
    └── bin
        └── ls                   ← commands are files, too
```

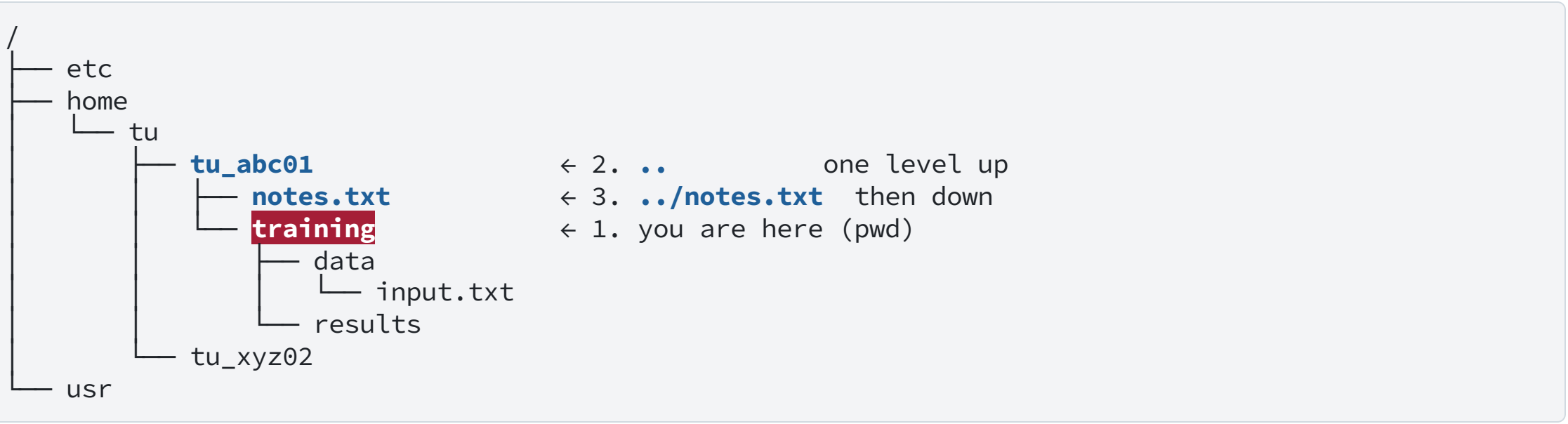
Directories contain files and other directories, separated by `/`

A path is a walk through the tree

```
/                          ← start at the root   /
├── etc
├── home                                         /home
│   └── tu                                       /home/tu
│       ├── tu_abc01                             /home/tu/tu_abc01
│       │   ├── notes.txt
│       │   └── training                         /home/tu/tu_abc01/training
│       │       ├── data                         /home/tu/tu_abc01/training/data
│       │       │   └── input.txt                /home/tu/tu_abc01/training/data/input.txt
│       │       └── results
│       └── tu_xyz02
└── usr
```

Each step down adds `/NAME`. The **absolute path** is the whole walk from the root.

A relative path starts where you are



From `training`, `../notes.txt` and `/home/tu/tu_abc01/notes.txt` lead to the **same file**.

Absolute and relative paths

You are in `/home/tu/tu_abc01/training` (check with `pwd`)

Path	Type	Points to
`/home/tu/tu_abc01/training/data/input.txt`	absolute	`input.txt`
`data/input.txt`	relative	`input.txt`
`../notes.txt`	relative	`notes.txt`
`.` / **`..`**	relative	this directory / one level up
`~`	shortcut	`/home/tu/tu_abc01`

- **Absolute:** starts with `/`, the route from the root, works from anywhere
- **Relative:** starts where you are, shorter, depends on `pwd`

Quiz: Where am I?

You are in `/home/tu/tu_abc01/training/data`. Where does `cd ../results` take you?

1. `/home/tu/tu_abc01/results`
2. `/home/tu/tu_abc01/training/results`
3. `/home/tu/tu_abc01/training/data/results`
4. Error: `results` does not exist

Navigating

Command	Meaning
`pwd`	print working directory
`ls`	list directory contents
`cd DIRECTORY`	change directory
`cd ..`	one level up
`cd` or `cd ~`	back to your home directory

- `Tab` completes names, `↑` recalls previous commands

Hidden files

```
ls -a          # show all files, including hidden ones
ls -la ~       # long list of your home directory
```

- Names starting with `.` are hidden: `.bashrc`, `.ssh`, `.cache`
- Mostly configuration files of programs
- `.` and `..` appear in every directory: this one and one level up
- Hidden directories like `.cache` or `.conda` can fill your home quota

Working with files and directories

```
mkdir results                # create directory
cp data.txt backup.txt       # copy
mv backup.txt results/       # move or rename
rm results/backup.txt        # delete (no recycle bin!)
rm -r results                # delete directory recursively
```

Exercise: Navigate and organize

Goal: create, copy and rename files and directories

1. Create a directory `training` in your home directory
2. Create a subdirectory `training/data`
3. Copy `/pfs/10/project/training_material/example.txt` into it
4. Rename the copy to `input.txt`

Expected result: `ls -l ~/training/data` shows only `input.txt`

> **Help yourself:** `mkdir --help`, `tldr mv`

Bonus: create `training/data` with a single command

Break

15 minutes · back at 10:45

Wildcards: many files at once

```
ls *.txt                    # * : any characters, also none
ls sample_1*                # sample_1.csv, sample_10.csv, ...
ls sample_?.csv             # ? : exactly one character
ls sample_??.csv            # ?? : exactly two characters
cp data/*.fastq.gz backup/  # copy all FASTQ files
```

- The **shell** expands the pattern before the command runs (called **globbing**)
- `*` does **not** match hidden files: `ls *` does not show `.bashrc`
- `echo PATTERN` shows what a pattern matches
- Before `rm` with a wildcard: run `ls` with the same pattern first

Quiz: Which files match?

A directory contains: `sample_1.csv`, `sample_10.csv`, `sample_A.csv`, `sample_1.txt`, `Sample_2.csv`

Which files does `ls sample_?.csv` list?

1. `sample_1.csv`
2. `sample_1.csv`, `sample_A.csv`
3. `sample_1.csv`, `sample_10.csv`, `sample_A.csv`
4. `sample_1.csv`, `sample_A.csv`, `Sample_2.csv`

Exercise: Pick files with wildcards

Goal: select groups of files with `*` and `?`

1. Copy the example data: `cp -r /pfs/10/project/training_material/data ~/training/fastq`
2. Go into `~/training/fastq` and list all `.fastq.gz` files
3. List all files that belong to `sample_2`
4. Create `~/training/results` and copy all `.csv` files into it

Expected result: `ls ~/training/results` shows only `.csv` files

> **Help yourself:** try your pattern with `echo` first

Bonus: what does `echo *.xyz` print, if no file matches?

Copying data to and from BinAC 2

Run these on **your own computer**, not on BinAC 2:

```
scp results.csv USERNAME@login.binac2.uni-tuebingen.de:training/       # upload a file
scp USERNAME@login.binac2.uni-tuebingen.de:training/results.csv .       # download into .
rsync -av data/ USERNAME@login.binac2.uni-tuebingen.de:training/data/   # sync, skips unchanged files
```

- The path after `:` is relative to your home directory on BinAC 2
- Graphical tools (WinSCP, FileZilla, Cyberduck) use **SFTP**: same host, same login
- Large data belongs in a workspace or in PROJECT (this afternoon), not in HOME

Did everything arrive?

A checksum is a fingerprint of the content: same checksum, same file.

```
sha256sum data.fastq.gz                    # on BinAC 2
sha256sum data.fastq.gz                    # on your computer, compare
```

Many files at once:

```
sha256sum *.fastq.gz > checksums.sha256  # create the list
sha256sum -c checksums.sha256            # check: OK or FAILED per file
```

- `md5sum` works the same way; sequencing facilities often send `.md5` files
- macOS: `shasum -a 256 FILE`, Windows PowerShell: `Get-FileHash FILE`
- `rsync` verifies transferred files itself; check after `scp` and for data from others

Exercise: Get a file home

Goal: download a file from BinAC 2 to your computer

1. Open a **second terminal on your own computer**
2. Download `~/training/data/input.txt` from BinAC 2 into your current directory
3. Check on your computer that the file is there

Expected result: `ls input.txt` on your computer shows the file

> **Help yourself:** `tldr scp`, `man scp` (Linux, macOS), the slide "Copying data"

Bonus: upload it again, as `~/training/upload.txt`

Viewing and editing text files

Command	Use
`cat FILE`	print the whole file, also works where `less` is missing (containers)
`cat A B > C`	join files A and B into C ("concatenate")
`cat -A FILE`	show invisible characters: `^M` = Windows line ending
`less FILE`	page through (`q` quits, `/` searches)
`head -n 20 FILE`	first 20 lines
`tail -f FILE`	follow a growing file, e.g. a log (`Ctrl+C` stops)
`nano FILE`	simple editor

What is in this file?

```
file input.txt                # what kind of file is it?
file data.fastq.gz            # gzip compressed data
file job.sh                   # ... with CRLF line terminators?
wc -l input.txt               # number of lines
stat input.txt                # size, owner, permissions, exact time of last change
```

- File extensions can lie: `file` looks at the content
- Script written on Windows? Fix the line endings with `dos2unix job.sh` (error otherwise: `/bin/bash^M: bad interpreter`)

Editing with `nano`

```
nano notes.txt                # opens the file, creates it if missing
```

Keys	Action
`Ctrl+O`, then **`Enter`**	save ("write out")
`Ctrl+X`	exit
`Ctrl+K` / **`Ctrl+U`**	cut line / paste
`Ctrl+W`	search

- `^X` at the bottom of the screen means `Ctrl+X`
- `nano` is enough for job scripts. `vim` and `emacs` are more powerful, but harder to learn

Exercise: Write a file

Goal: create, save and change a text file with `nano`

1. Create `~/training/notes.txt` with `nano` and write three lines
2. Save, exit, and print the file with `cat`
3. Open it again, delete the second line, save and exit

Expected result: `cat ~/training/notes.txt` shows two lines

> **Help yourself:** the key list at the bottom of the `nano` screen

Bonus: view the file with `less` and search for a word

Looking at directories and comparing files

`tree` : a directory as a tree

```
$ tree ~/training
/home/tu/tu_abc01/training
├── data
│   └── input.txt
├── fastq
│   ├── sample_1_R1.fastq.gz
│   └── ...
├── notes.txt
└── results
    └── ...  (the .csv files)
```

`tree -L 2` : only 2 levels, `tree -d` : only directories

`diff` : what changed?

```
$ diff old.txt new.txt
2c2
< threads=4
---
> threads=8
```

- `<` only in the first file, `>` only in the second
- `diff -u` : easier to read
- `diff -q dir1 dir2` : which files differ

Permissions: who may do what?

```
$ ls -l
-rwxr-x--- 1 tu_abc01 bw16f003 1.2K Nov 16 10:00 job.sh
drwxr-x--- 2 tu_abc01 bw16f003 4.0K Nov 16 10:00 results
```

`-` **file,** `d` **directory**, then `rwx` for **user (owner)**, **group (e.g. your project)**, **others**

	File	Directory
`r`	read the content	list the names inside
`w`	change the content	create, delete, rename inside
`x`	run it as a program	enter it, reach the files inside

```
chmod u+x job.sh        # make executable for yourself
chmod g+rx results      # let your group look inside
```

Pipes and redirection

```
ls -l > listing.txt                # write output to file
ls -l >> listing.txt               # append
grep "ERROR" job.log | wc -l       # count lines with ERROR
sort names.txt | uniq -c           # count unique lines
```

- `|` passes the output of one command to the next
- `>` overwrites, `>>` appends
- `2>` redirects error messages

Quiz: What is in the file?

```
echo hello > out.txt
echo world > out.txt
cat out.txt
```

What does `cat` print?

1. `hello`
2. `world`
3. `hello` and `world`, on two lines
4. An error: `out.txt` already exists

Exercise: Count the reads

Goal: combine commands with `|` and `>`

A FASTQ file stores each read in **4 lines**. `zcat` prints a compressed file.

1. Go into `~/training/fastq` and show the first 8 lines of `sample_1_R1.fastq.gz`
2. Count the lines of this file
3. Write the line count into `~/training/results/count.txt`

Expected result: `cat ~/training/results/count.txt` shows a number; reads = number / 4

> **Help yourself:** `zcat --help`, `head --help`, `wc --help`

Bonus: why is `zcat FILE | grep -c "^@"` not a reliable way to count reads?

Searching: what are you looking for?

Question	Command
Where is the file named X?	`find . -name "X"`
Which files contain a word?	`grep -rl "word" DIRECTORY`
In which lines of a file is the word?	`grep -n "word" FILE`
Which command does Y?	`apropos Y`
How did I run that command again?	`Ctrl+R` or `history | grep WORD`

`grep` options: `-i` ignore case, `-r` all files below a directory, `-n` line numbers, `-l` file names only, `-c` count, `-v` lines **without** the word

`find`: search for files

```
find ~/training -name "*.csv"          # by name, quote the pattern!
find . -iname "readme*"                # ignore case: README, Readme, readme
find . -type f -name "sample_*"        # only files
find . -type d -name results           # only directories
find . -size +1G                       # larger than 1 GiB
find . -mtime -7                       # changed in the last 7 days
find . -name "*.fastq.gz" | wc -l      # how many?
```

- Quotes: otherwise the shell expands `*.csv` before `find` sees it
- Several conditions can be combined: all of them must match
- Search in your own directories, not from `/`: that takes a long time and prints many errors
- On `/pfs` (WORK and PROJECT): use `lfs find` with the same options, it is faster and spares the file system

Variables

```
echo $HOME              # your home directory
echo $USER              # your user name
echo $PATH              # directories where the shell looks for commands
data=~/training/fastq # set a variable: no spaces around =
ls "$data"              # use it, with quotes
env                     # list all environment variables
```

- `$NAME` reads a variable, `NAME=value` sets it
- Variables exist only in the current shell, until you log out
- Job scripts use variables like `$SLURM_CPUS_PER_TASK` (this afternoon)

Exercise: Use variables

Goal: set and use your own variables

1. Print your user name and your home directory using variables
2. Set a variable `results` to `~/training/results`
3. List the contents of `results` and change into it

Expected result: `pwd` ends with `training/results`

> **Help yourself:** the previous slide

Bonus: what happens with `results = ~/training/results` (spaces around `=`)?

Variables: good habits

```
sample="sample 1"            # lowercase for your own variables
ls $sample                   # wrong: ls looks for "sample" and "1"
ls "$sample"                 # right: one argument
echo "$sample_R1.fastq.gz"   # wrong: reads variable sample_R1 (empty)
echo "${sample}_R1.fastq.gz"
today=$(date +%F)            # store the output of a command
```

- **Always quote:** `"$sample"`, otherwise spaces and `*` in the value break the command
- **Braces** when text follows: `"${sample}_R1"`
- **Lowercase** for your own variables, UPPERCASE is for environment variables: `PATH=~/data` breaks every command in this shell
- **Meaningful names:** `input_dir`, not `x`

Shell scripts

```
#!/bin/bash
# count.sh: count lines in all .txt files

for f in *.txt; do
    echo "$f"
    wc -l "$f"
done
```

```
chmod u+x count.sh
./count.sh
```

- `#!/bin/bash` : first line, says which program runs the script
- `#` starts a comment
- `for ... do ... done` : runs the block once per file, `$f` is the current file
- `./` : the current directory is not in `$PATH`

Exercise: Your first script

Goal: write, run and redirect a bash script

1. Write a script `info.sh` that prints the current date and your hostname
2. Make it executable and run it
3. Redirect its output into `info.txt`

Expected result: `cat info.txt` shows date and hostname

> **Help yourself:** `tldr date`, `tldr hostname`, `man chmod`

Bonus: print the date in the format `2026-11-16`

Long commands over several lines

A `\` at the end of a line continues the command on the next line:

```
find ~/training \
    -type f \
    -name "*.fastq.gz" \
    -mtime -7
```

- Easier to read and to change, one option per line
- `\` must be the **very last character**: a space after it breaks the command
- No comments after `\`

Now you can read this

```
for f in ~/training/fastq/*_R1.fastq.gz; do    # wildcard, loop
    echo "$f"                                  # variable, quoted
    zcat "$f" | wc -l                          # pipe: lines / 4 = reads
done
```

Summary

- Log in with `ssh`; commands, options, arguments; read the error message
- Help yourself: `--help`, `tldr`, `man`, `info`, `apropos`
- Paths: absolute and relative; `cd`, `ls`, `pwd`, `tree`
- Files: `cp`, `mv`, `rm`, `mkdir`, many at once with wildcards
- Copying data with `scp` and `rsync`, checking it with `sha256sum`
- Viewing and editing: `cat`, `less`, `head`, `nano`, `file`, `stat`, `diff`
- Permissions: `ls -l`, `chmod`
- Pipes, redirection, `grep` and `find`
- Variables and simple shell scripts

This afternoon: using BinAC 2 for real work