UNIX & BASH

A Practical Guide for Physics Students

Theo Hugues
Department of Physics, Engineering Physics & Astronomy | Queen's University
Tuesday, May 5, 2026

This document was worked through during the SUITE 2026 workshop . Its purpose is to get anyone who has never used the command line before comfortable doing real work in a shell. The examples and exercises are drawn from particle astrophysics: organising detector run files, processing dark matter search outputs, automating simulation pipelines, and connecting to computing clusters. It is designed as a long-term reference , keep it handy throughout your research career.

Preface

Welcome to the world of Unix and Bash. Over the course of your research in astroparticle physics, you will regularly work with large datasets, simulation outputs from Geant4, raw detector data waiting to be reprocessed, or event files produced by analysis pipelines. All of these live on Linux-based computing clusters, and your ability to navigate, process, and analyse them efficiently depends on your fluency with the command line.

This guide is a hands-on introduction to Unix and the Bash shell. It assumes no prior experience with the command line, only curiosity and a willingness to experiment. Each section builds on the previous one, and every concept is illustrated with examples drawn from astroparticle physics workflows: organising simulation runs, grepping through event logs, automating data pipelines, and connecting to remote HPC clusters.

By the end of this workshop, you will be able to:

- Navigate a Unix filesystem confidently
- Manipulate files, directories, and data streams from the terminal
- Write Bash scripts to automate repetitive analysis tasks
- Use powerful text-processing tools (`grep`, `awk`, `sed`) to extract and transform data
- Connect to remote computing clusters via SSH
- Understand the foundations of job scheduling on HPC systems

> 💡 **Tip**
> All exercises in this guide run on any standard Linux or macOS system. If you are on Windows, follow the setup instructions in Section 2, no prior Unix knowledge is required.

1 | Setup: Getting Your Environment Ready

Before we can start issuing commands, everyone needs a working terminal. Follow the instructions for your operating system below.

1.1 macOS

macOS is built on a Unix foundation, you already have everything you need. Open a terminal via Applications -> Utilities -> Terminal, or search for "Terminal" in Spotlight (Cmd+Space).

You will see a prompt like:

```
student@macbook:~$
```

> 💡 **Tip**
> macOS Catalina and later use io`zsh` as the default shell instead of `bash`. For this workshop, start each session by typing `bash` to make sure we are all using the same shell.

1.2 Windows: MobaXTerm

Windows does not ship with a Unix shell, so we will use MobaXTerm, a free terminal emulator that provides a full Bash environment.

- Download the **Portable Edition** from: https://mobaxterm.mobatek.net/download-home-edition.html
- No installation required, just extract the .zip and run `MobaXterm_Personal_XX.X.exe`
- If you already have Cygwin or WSL2 installed and are comfortable with it, that works too, but some paths in this document may differ slightly.

1.3 Practice Directory

Download `PracticeDirectory_SUITE.zip` from the workshop page (Indico link distributed separately). Then:

Windows

Create a new folder at `C:\Users\<username>\Workshop_SUITE2026` and copy the zip file there. Open MobaXTerm and navigate to it:

```
cd /cygdrive/c/Users/<username>/Workshop_SUITE2026 (path may vary, try ~/)
unzip PracticeDirectory_SUITE.zip (apt-get install unzip if not installed)
ls
```

macOS

Create a folder at `/Users/<username>/Workshop_SUITE2026` and copy the zip file there. Open a terminal and run:

```
mkdir ~/Workshop_SUITE2026
cd ~/Workshop_SUITE2026
unzip PracticeDirectory_SUITE.zip
ls
```

1.4 Checking Your Shell

Let's make sure everyone is running the same shell. Type:

```
echo $SHELL
```

You should see something ending in `/bash`. If not, type `bash` and press Enter, then try again. Bash (Bourne Again SHell) is the most widely used shell in scientific computing, and the one used throughout this workshop.

> 💡 **Tip**
> Most shells (zsh, fish, dash) are similar to Bash for everyday use. If you are forced to use a different shell on a cluster, most commands here will still work, small differences can always be Googled.

2 | Navigating the Filesystem

In Unix, everything, files, directories, devices, even running processes, is organised in a single hierarchical tree rooted at `/`.

2.1 Key Navigation Commands

Command	Description
`pwd`	Print Working Directory, where am I right now?
`ls`	List the contents of a directory
`cd DIR`	Change into directory DIR
`cd ..`	Go up one level
`cd ~`	Return to your home directory
`cd -`	Go back to the previous directory

2.2 Useful `ls` Options

The `ls` command has many options that become indispensable in practice:

Option	Effect
`ls -l`	Long listing: permissions, owner, size, date
`ls -a`	Show hidden files (names starting with `.`)
`ls -h`	Human-readable file sizes (KB, MB, GB)
`ls -t`	Sort by modification time (newest first)
`ls -r`	Reverse sort order
`ls -ltrah`	All of the above combined, the one to memorise

Try running `ls -ltrah` inside your PracticeDirectory_SUITE. Notice the difference from plain `ls`.

> 💡 **Tip**
> You can create a shortcut (alias) so that typing `ls` always runs `ls -ltrah`. We will cover aliases in the `.bashrc` section at the end of this workshop.

2.3 Paths: Absolute vs. Relative

A path describes the location of a file or directory in the filesystem. There are two kinds:

- **Absolute path**: starts from the root `/`. Example: `/home/student/data/run_001.root`

- **Relative path**: starts from your current directory. Example: `data/run_001.root` or `../../logs/`

The special symbols `.` (current directory) and `..` (parent directory) are used constantly:

```
# You are in /home/student/PracticeDirectory_SUITE/scripts
cd ../reco_data    # go to /home/student/PracticeDirectory_SUITE/reco_data
cd ~               # go to /home/student
cd -               # go back to reco_data
```

🔭 Exercise: Navigating the Filesystem

Work through the following steps in your terminal:

1. Type `pwd` from your home directory. Record the output.
2. Navigate to PracticeDirectory_SUITE using both an absolute and a relative path.
3. Use `cd` with `..` to navigate from `PracticeDirectory_SUITE/scripts/python` to `PracticeDirectory_SUITE/reco_data` in a single command.
4. Run `ls -ltrah` in PracticeDirectory_SUITE. What is the most recently modified item?
5. Run `ls -a ~`. Find a file starting with `.`. What do you think `.bashrc` does?

3 | Working with Files and Directories

The following commands are the bread and butter of daily Unix work. You will use them constantly, organising detector run files, cleaning up simulation outputs, and setting up project directories.

3.1 Creating and Removing

Command	Description
`mkdir DIR`	Create a new directory
`mkdir -p a/b/c`	Create nested directories in one command
`touch file.txt`	Create an empty file (or update its timestamp)
`rm file.txt`	Delete a file, permanently, no Recycle Bin
`rmdir DIR`	Remove an empty directory
`rm -rf DIR/`	Recursively delete a directory and all contents

> ⚠️ **Warning**
> `rm` is permanent. There is no undo, no Trash, no recovery. Always double-check the path before pressing Enter. Never run `rm -rf ~/` or `rm -rf /`, these could destroy your entire home directory or system.

3.2 Copying and Moving

Command	Description
`cp src.txt dst.txt`	Copy a file (overwrites dst if it exists, no warning!)
`cp -r src/ dst/`	Copy a directory recursively
`mv old.txt new.txt`	Rename a file
`mv file.txt dir/`	Move a file into a directory

Important: if you `mv` or `cp` onto an existing file, its contents are silently overwritten. The terminal will not warn you. This is one of the most common causes of accidental data loss for beginners.

3.3 Viewing File Contents

```
head -n 5 rawdata.txt      # show first 5 lines
tail -n 5 rawdata.txt      # show last 5 lines
tail -f run.log            # follow a log file in real time (Ctrl+C to
stop)
cat rawdata.txt            # print entire file to screen
less rawdata.txt           # page through a large file (q to quit)
```

```
wc -l rawdata.txt          # count lines
wc -w rawdata.txt          # count words
```

> **Tip**
> If you accidentally run `cat` on a huge file and your terminal fills with text, press Ctrl+C to interrupt the command. If that fails, try Ctrl+Z, then type `kill %1`. This will happen to you, now you know what to do.

When you run `python stdout_example.py` from `scripts/python/`, you will notice it prints the numbers 1 through 9 — not 10. This is because `range(1,10)` in Python excludes the upper bound. This is a common source of off-by-one surprises and a good reminder to always sanity-check your output.

3.4 Archiving: zip and tar

Compressed archives are the standard way to distribute and back up data in HEP and astroparticle physics. The two most common tools are `zip` and `tar`.

```
# zip
zip -r archive.zip MyDirectory/    # create archive
unzip archive.zip                  # extract archive

# tar (more common on Linux clusters)
tar -czf archive.tar.gz MyDirectory/   # create compressed archive
tar -xf archive.tar.gz                 # extract
```

The `tar` flags: `-c` = create, `-z` = compress with gzip, `-f` = operate on the named file, `-x` = extract.

3.5 File Permissions

Unix controls access to every file through a permission system. This matters enormously on shared computing clusters used by your collaboration.

A typical `ls -l` output line looks like:

```
-rw-r--r--  1 theo  physics  270 May 5 15:38 fake_program.py
```

The permission string `-rw-r--r--` breaks down as:

Part	Meaning
`-`	File type: `-` = regular file, `d` = directory, `l` = symbolic link
`rw-`	Owner (you): read + write, no execute
`r--`	Group: read only
`r--`	Everyone else: read only

To change permissions, use `chmod` with octal notation (`r=4`, `w=2`, `x=1`):

```
chmod 755 script.sh    # rwxr-xr-x: owner full, others read+execute
chmod 644 data.txt     # rw-r--r--: owner read/write, others read
chmod 700 private.sh   # rwx------: owner only
```

🔭 Exercise: Organising a Detector Run

Imagine you are organising output files from a liquid argon dark matter run, let's randomly select DEAP-3600. Set up the following structure and practice the commands you have learned.

6. Navigate to PracticeDirectory_SUITE and create a directory called `deap_run`.
7. Inside `deap_run`, create subdirectories `raw/`, `processed/`, and `logs/`.
8. Create an empty file `raw/run_001.dat` and another `raw/run_002.dat`.
9. Copy both `.dat` files into `processed/`.
10. Rename `processed/run_001.dat` to `processed/run_001_calibrated.dat`.
11. Delete the original `raw/run_001.dat`.
12. Set permissions `644` on both files in `processed/`.
13. Finally, remove the entire `deap_run` directory.

4 | Input/Output, Redirection, and Pipes

One of the most powerful ideas in Unix is that every command reads from standard input (stdin) and writes to standard output (stdout). By connecting these streams, you can build sophisticated data processing pipelines from simple building blocks, exactly what is needed in astroparticle physics data analysis.

4.1 Redirecting Output

```
# Redirect stdout to a file (overwrites!)
echo '# Event list header' > events.txt

# Append to a file (does NOT overwrite)
echo 'Run 1  33  2  12' >> events.txt

# Redirect stderr (error messages) to a separate file
myprogram > output.log 2> errors.log

# Redirect both stdout and stderr together
myprogram > all_output.log 2>&1

# Discard output entirely
verbose_program > /dev/null
```

4.2 Pipes

The pipe character `|` sends the output of one command directly into the input of the next, no intermediate files needed. Think of it as a detector chain: raw signal enters one end, passes through filters and amplifiers, and refined data exits the other.

```
# Count the total number of events in a file
cat rawdata.txt | wc -l

# Find how many runs are above 1000 photons
awk '$7 > 1000 {print}' rawdata.txt | wc -l

# List the 5 most recently modified files
ls -lt | head -5

# Find unique run IDs in an analysis log
grep 'RunID' analysis.log | awk '{print $2}' | sort | uniq
```

> 💡 **Tip**
> Pipes are one of the most distinctive and powerful features of Unix. With `grep`, `awk`, `sort`, `uniq`, `wc`, and `head`/`tail`, you can answer complex questions about large datasets in a single line, no Python script needed.

🔭 Exercise: Building Analysis Pipelines

Navigate to `PracticeDirectory_SUITE/raw_data/`. The file `rawdata.txt` contains 10,000 lines of simulated detector data.

14. Use `wc -l` to confirm there are 10,000 lines.
15. Use `head` and `tail` to see the first and last 5 lines. What columns do you see?
16. How many lines start with a date between June 6 and June 9, 2023, inclusive? (Answer: 3419) Hint: `grep 2023060[6-9] rawdata.txt | wc -l`
17. How many lines contain `5.0` somewhere in them? Watch out for decimal points, recall that `.` is special in grep patterns. (Answer: 2208)
18. Use `diff rawdata.txt rawdata_adjusted.txt` to find what changed between the two files. What was swapped?

5 | Text Processing: grep, find, and Searching

In astroparticle physics, you will spend a great deal of time extracting information from large text files: event lists, run logs, configuration files, and simulation outputs. The tools in this section are indispensable.

5.1 grep — Searching Text

`grep` searches for lines matching a pattern (plain text or regular expression). It is one of the most-used commands in scientific computing.

```
# Find all lines mentioning a run number
grep 'Run 1' reco_data/Backgrounds/alphas.txt     # matches Run 1, but
also Run 10-19 (substring!)
grep 'Run 1,' reco_data/Backgrounds/alphas.txt    # add the comma to match
Run 1 exactly

# Case-insensitive search
grep -i 'neutron' analysis.log

# Count matching lines
grep -c 'TRIGGER' run.log

# Show line numbers
grep -n 'ERROR' run.log

# Search recursively through all files in a directory
grep -r 'dark matter' .

# Invert match — lines NOT matching
grep -v '^#' config.ini

# Match with context: 2 lines after, 1 line before
grep -A 2 -B 1 'OVERFLOW' run.log

# Use a regular expression: lines starting with a date
```

> 💡 **Tip**
> The characters `.` and `*` have special meaning in grep patterns. To match a literal dot (e.g. in `5.0`), escape it with a backslash: `grep '5\.0'`. Without the backslash, `.` matches any single character, and you will get more results than expected.

5.2 find — Locating Files

`find` searches the filesystem for files matching criteria you specify. It is invaluable on large computing clusters where directory trees can be enormous.

```
# Find all .root files anywhere under /data
find /data -name '*.root'
```

```
# Find files modified in the last 24 hours
find . -mtime -1

# Search for files containing 'neutrons' in the name
find reco_data/ -name '*neutron*'

# Find and list large files (>100 MB)
find /scratch -size +100M
```

5.3 A Note on awk and sed

`awk` and `sed` are two powerful stream-processing tools that complement `grep`. `awk` excels at working with columnar data, for example, extracting the third column from a tab-separated event file, or computing the sum of a column. `sed` is ideal for find-and-replace operations on text streams.

A full treatment is beyond today's scope, but here are two quick examples relevant to physics data work:

```
# awk: print only events with energy > 1.0 TeV (column 1)
awk '$1 > 1.0 {print}' events.dat

# awk: compute total photon count from column 2
awk '{sum += $2} END {print "Total:", sum}' run_summary.txt

# sed: replace 'Run' with 'EVENT' everywhere in a file
sed 's/Run/EVENT/g' run.log > event.log
```

Good resources: sed guide at grymoire.com/Unix/Sed.html, awk guide at grymoire.com/Unix/Awk.html

🔭 Exercise: Searching Detector Data

Work in PracticeDirectory. You will use `grep` and `find` to answer questions about the data files.

19. In `reco_data/Backgrounds/`, run `grep 'Run 1' alphas.txt`. How many lines match? Is it what you expected? Now try `grep 'Run 1,' alphas.txt`. What is the difference, and why?
20. Use `grep -r 'dark matter' .` to search the entire PracticeDirectory recursively. Which file contains it?
21. Use `find` to locate a file called `neutrons.txt` anywhere under PracticeDirectory.
22. Use `grep` with a range pattern to find all lines in `rawdata.txt` containing a 2-digit number between 20 and 29 (hint: `2[0-9]`).
23. (Advanced) Combine `grep` and `wc` in a pipeline to count how many unique run dates appear in `rawdata.txt`.

6 | Editing Files: vim

We are now at the stage where we need to write our own files — scripts, configuration files, and analysis code. There are many text editors available, and the right choice depends on context.

In your day-to-day work on your own machine, a modern editor like VS Code is an excellent choice. It has syntax highlighting, Git integration, a built-in terminal, and a Remote-SSH extension that lets you edit files on a computing cluster directly from your laptop as if they were local. VS Code also ships with GitHub Copilot integration — an AI assistant that can suggest code completions and help you debug scripts inline.

However, for this workshop we will use `vim`, for the following practical reasons:

- It is available on virtually every Unix system — including computing clusters where you have no choice
- It requires no graphical display, which is essential for remote SSH connections where no GUI is available
- It is extremely powerful once you know a handful of commands
- It is highly customisable

Knowing at least the basics of vim is not optional if you work in HEP or astroparticle physics — at some point you will be on a remote machine with nothing else available. Think of it as a survival skill.

To open a file in vim (creating it if it does not exist):

```
vim my_first_script.sh
```

6.1 The Two Essential Modes

vim has multiple modes. The two you will use constantly are:

Mode	How to enter it	What you can do
`Normal`	Default on open; press `ESC` from anywhere	Navigate, delete, copy, paste, search
`Insert`	Press `i` in Normal mode	Type text as in a regular editor

The most common beginner mistake is trying to type text immediately after opening vim and seeing random behaviour. Remember: press `i` first.

6.2 Essential Commands

In Normal Mode

Key / Command	Action
`i`	Enter insert mode (cursor stays)
`ESC`	Return to normal mode

Key / Command	Action
`u`	Undo
`Ctrl+r`	Redo
`/pattern`	Search for pattern; `n` for next, `N` for previous
`dd`	Delete (cut) the current line
`yy`	Copy (yank) the current line
`p`	Paste below current line

In Command Mode (enter with `:`)

Command	Action
`:w`	Write (save) the file
`:q`	Quit (fails if unsaved changes)
`:wq`	Save and quit
`:q!`	Quit, discarding all changes
`:N`	Jump to line N
`:%s/old/new/gc`	Replace every `old` with `new`, asking for confirmation

> 💡 **Tip**
> If you ever get completely lost in vim and cannot figure out what mode you are in, press `ESC` several times and then type `:q!` to quit without saving. You can always reopen the file. This will happen to everyone at least once.

> 🔭 **Exercise: Your First File in vim**
> In PracticeDirectory_SUITE, create a new file called `detector_notes.txt` using vim.
>
> 24. Open vim: `vim detector_notes.txt`
> 25. Press `i` to enter insert mode. Type a few lines describing a hypothetical dark matter detector run (date, run number, number of events).
> 26. Press `ESC` to return to normal mode.
> 27. Use `:%s/run/Run/g` to capitalise the word 'run' throughout.
> 28. Save with `:w` and quit with `:q`. Verify the file exists with `cat detector_notes.txt`.

7 | Bash Scripting

A Bash script is a text file containing a sequence of commands that the shell executes in order. Scripts let you automate repetitive tasks: processing 500 detector runs, renaming files to a consistent convention, or generating a summary report after each simulation. This is one of the most valuable skills you will develop as a researcher.

7.1 Anatomy of a Script

```
#!/bin/bash
# The first line is the shebang: it tells Unix which interpreter to use.
# Lines starting with # are comments, ignored by the shell.

echo '=================================='
echo '  DEAP-3600 Analysis Pipeline'
echo '=================================='
echo "Started at: $(date)"
echo "Running on: $(hostname)"
echo "User: $(whoami)"
```

To run a script:

```
chmod +x myscript.sh    # give yourself execute permission
./myscript.sh           # run it

# Alternatively (no execute permission needed):
bash myscript.sh
```

7.2 Variables

```
#!/bin/bash

# Variables: no spaces around the = sign
DETECTOR='DEAP-3600'
RUN_NUMBER=4821
DATA_DIR='/data/deap/runs'

echo "Analysing ${DETECTOR} run ${RUN_NUMBER}"
echo "Data directory: ${DATA_DIR}"

# Command substitution: store output of a command in a variable
N_FILES=$(ls ${DATA_DIR}/*.root | wc -l)
echo "Found ${N_FILES} ROOT files to process"
```

Note: spaces around `=` are not allowed. `VAR=value` works; `VAR = value` does not. This trips up almost everyone who comes from Python.

7.3 Conditionals

```
#!/bin/bash

DATA_FILE='alphas.txt'

# Check if a file exists
if [ -f "${DATA_FILE}" ]; then
    echo "File found: ${DATA_FILE}"
    wc -l ${DATA_FILE}
else
    echo "ERROR: File not found, ${DATA_FILE}"
    exit 1
fi

# Numeric comparison
N_EVENTS=$(grep -c 'Run' alphas.txt)
if [ ${N_EVENTS} -gt 1000 ]; then
    echo "Good run: ${N_EVENTS} events"
else
    echo "WARNING: Only ${N_EVENTS} events, low statistics"
fi
```

7.4 Loops

```
#!/bin/bash

# --- FOR LOOP: iterate over a list ---
for DETECTOR in DEAP-3600 DarkSide-20k MiniCLEAN; do
    echo "Processing data from: ${DETECTOR}"
done

# --- FOR LOOP: iterate over files ---
for ROOT_FILE in /data/deap/runs/*.root; do
    RUN_ID=$(basename ${ROOT_FILE} .root)
    echo "Processing run: ${RUN_ID}"
    # ./process_run.sh ${ROOT_FILE}
done

# --- WHILE LOOP ---
COUNTER=1
while [ ${COUNTER} -le 10 ]; do
    echo "Iteration ${COUNTER}"
    COUNTER=$((COUNTER + 1))
done
```

7.5 Running Scripts: Four Methods

Method	Behaviour
`. script.sh`	Runs in current shell. Variables persist. `exit` closes your terminal.
`source script.sh`	Identical to `.` (bash-specific). Variables persist.
`bash script.sh`	Opens a subshell. Variables defined inside do not leak out.
`./script.sh`	Runs using the shebang line. Requires execute permission.

In practice, use `bash script.sh` while testing (safe) and `./script.sh` once it is working and you want to treat it as a proper executable.

🔭 Exercise: Automating a Multi-Run Analysis

In `PracticeDirectory_SUITE/scripts/bash/`, look at `script_example.sh` and study its structure. Then write a new script `run_summary.sh` that:

29. Defines a list of detector names: `DEAP-3600 DarkSide-20k ArDM`.
30. For each detector, creates a directory `results/<detector_name>/` if it does not already exist.
31. Inside each directory, creates a file `summary.txt` containing the detector name and the current date.
32. After the loop, prints: `Done. Processed 3 detectors. Results in ./results/`
33. (Advanced) Add a check at the start: if `results/` already exists, print a warning and ask the user if they want to overwrite it.

7.6 Final Challenge: The Fruits Pipeline

This is the capstone exercise that tests most of what we have covered. In `PracticeDirectory_SUITE/scripts/bash/Challenge/`, there is a file `fruits_data.txt` containing 10,000 lines of random words, including these five fruits: `strawberry`, `banana`, `apple`, `orange`, `grape`.

Write a Bash script that:

34. Creates a directory for each fruit named after the fruit.
35. Inside each directory, creates `<fruitname>.txt` containing the count of lines that fruit appears on.
36. In the Challenge folder, creates `fruits_summary.txt` listing each fruit and its count (`fruit: N`).
37. Twist: the string in the file is `orange`, but your summary file should call it `clementine`.

You have all the tools to solve this. Everything you need was covered in this workshop.

8 | Remote Access: SSH and File Transfer

Astroparticle physics computing almost always involves working on remote clusters, CCIN2P3, CCDB, TRIUMF, SNOLAB computing nodes, or university HPC systems. SSH (Secure Shell) is the primary tool for this.

8.1 Connecting with SSH

```
# Basic connection
ssh username@cluster.triumf.ca

# Run a single command without an interactive session
ssh username@cluster.triumf.ca 'ls /data/deap/runs | wc -l'

# Enable X11 forwarding (for graphical tools like ROOT)
ssh -X username@cluster.triumf.ca
```

8.2 SSH Keys (Passwordless Login)

Once you are using a cluster regularly, typing a password every connection becomes tedious. SSH key pairs solve this.

```
# Generate a key pair (press Enter to accept defaults)
ssh-keygen -t ed25519 -C 'your.email@university.ca'

# Copy your public key to the cluster
ssh-copy-id username@cluster.triumf.ca

# Now log in without a password
ssh username@cluster.triumf.ca
```

> 💡 **Tip**
> Create a `~/.ssh/config` file to save shortcuts. Adding `Host cc` / `HostName cluster.triumf.ca` / `User yourusername` lets you simply type `ssh cc` to connect.

8.3 Transferring Files: scp and rsync

```
# Copy a file TO the remote cluster
scp analysis_config.txt username@cluster.triumf.ca:/home/username/

# Copy a file FROM the cluster
scp username@cluster.triumf.ca:/data/deap/output.root ./

# Copy an entire directory recursively
scp -r results/ username@cluster.triumf.ca:/data/deap/
```

```
# rsync: smarter than scp (only transfers changed files)
rsync -avh --progress local_data/ username@cluster.triumf.ca:/data/

# rsync dry-run: preview what would be transferred
rsync -avhn local_data/ username@cluster.triumf.ca:/data/
```

9 | Additional Topics

If time permits in the workshop, or for self-study, here are several additional topics that you will encounter in your research.

9.1 The .bashrc File and Aliases

The file `~/.bashrc` is a script that runs every time you open a new terminal. It is where you define persistent aliases, environment variables, and custom functions. Open it with vim and add lines like:

```
# In ~/.bashrc

# Aliases — shortcuts for long commands
alias ll='ls -ltrah'
alias gs='git status'
alias mydata='cd /data/deap/runs'

# Add a custom scripts directory to PATH
export PATH=$PATH:$HOME/scripts

# Source after editing (no need to restart terminal):
# source ~/.bashrc
```

After saving changes to `.bashrc`, run `source ~/.bashrc` to apply them in your current session.

9.2 Passing Arguments to Scripts

```
#!/bin/bash
# $0 = script name, $1 = first argument, $# = number of arguments

if [ $# -lt 2 ]; then
    echo "Usage: $0 <run_number> <detector>"
    exit 1
fi

RUN=$1
DETECTOR=$2
echo "Processing run ${RUN} from ${DETECTOR}"
```

Call the script as: `./process.sh 4821 DEAP-3600`

9.3 Job Scheduling: A Brief Overview

On shared HPC clusters, you submit jobs to a scheduler (SLURM is most common at Canadian facilities) rather than running them directly. The scheduler queues your jobs and dispatches them when resources are available.

```
#!/bin/bash
#SBATCH
--job-name=deap_analys
is
#SBATCH
--output=logs/job_%j.o
ut
#SBATCH
--time=02:00:00
#SBATCH --mem=8G
#SBATCH
--cpus-per-task=4

echo "Job
${SLURM_JOB_ID}
started at $(date)"
python run_analysis.py
--run 4821 --detector
DEAP-3600
echo "Job finished at
$(date)"
```

Command	Description
`sbatch job.sh`	Submit a job script
`squeue -u $USER`	Show your running and pending jobs
`scancel JOB_ID`	Cancel a job
`sinfo`	Show available partitions and nodes

9.4 tmux: Keeping Sessions Alive

`tmux` is a terminal multiplexer that lets you run processes that survive disconnection from SSH. Essential for long analysis jobs.

```
tmux new-session -s myanalysis    # start a named session
# ... run your work ...
# Ctrl+B, then D to detach (leaves session running)
tmux attach -t myanalysis         # reconnect later
```

Appendix: Quick Reference Card

Navigation

Command	Description
`pwd`	Show current directory
`ls -ltrah`	List with sizes, times, hidden files
`cd DIR`	Change into DIR
`cd ~`	Go to home directory
`cd -`	Go to previous directory

File Operations

Command	Description
`cp src dst`	Copy file (overwrites silently)
`mv src dst`	Move or rename
`rm file`	Delete file (permanent)
`mkdir -p a/b`	Create nested directories
`cat / head / tail`	View file contents
`less FILE`	Page through file (q to quit)
`wc -l FILE`	Count lines
`diff A B`	Show differences between two files
`chmod 755 FILE`	Set permissions

Searching and Text Processing

Command	Description
`grep PATTERN FILE`	Search for pattern in file
`grep -r PATTERN DIR`	Search recursively
`grep -i PATTERN FILE`	Case-insensitive search
`grep -v PATTERN FILE`	Lines NOT matching
`find . -name '*.root'`	Find files by name
`awk '{print $2}'`	Print column 2
`sed 's/A/B/g'`	Replace A with B globally
`sort | uniq`	Remove duplicate lines

Redirection and Pipes

Command	Description
`cmd > file`	Redirect stdout to file (overwrite)
`cmd >> file`	Append stdout to file
`cmd1 | cmd2`	Pipe output of cmd1 into cmd2
`cmd 2> err.log`	Redirect stderr
`cmd > out 2>&1`	Redirect both streams

Bash Scripting

Syntax	Description
`#!/bin/bash`	Shebang line, first line of every script
`VAR=value`	Assign variable (no spaces around =)
`echo ${VAR}`	Use variable
`$(command)`	Command substitution
`if [ cond ]; then`	Conditional
`for X in list; do`	Loop over list
`chmod +x script.sh`	Make script executable

SSH and File Transfer

Command	Description
`ssh user@host`	Connect to remote machine
`scp file user@host:path`	Copy file to remote
`scp user@host:file ./`	Copy file from remote
`rsync -avh src/ dst/`	Efficient sync (incremental)
`ssh-keygen -t ed25519`	Generate SSH key pair

vim Quick Reference

Key / Command	Action
`i`	Enter insert mode
`ESC`	Return to normal mode

Key / Command	Action
`u`	Undo
`:w`	Save
`:q!`	Quit without saving
`:wq`	Save and quit
`/pattern`	Search for pattern
`:%s/A/B/g`	Replace A with B everywhere

Further Resources

- **The Linux Command Line** by William Shotts, freely available at linuxcommand.org. The definitive introductory book.
- **Software Carpentry: The Unix Shell**, interactive lessons at software-carpentry.org, designed specifically for researchers.
- **explainshell.com**, paste any shell command to get a breakdown of every option.
- **GNU Bash Manual**, the official reference at gnu.org/software/bash/manual/
- **TRIUMF Computing**, internal wiki for cluster access and job submission.

Good luck with your research! Remember: the command line rewards practice. Do not be afraid to experiment, you can always make a new directory, and Ctrl+C cancels a runaway command. Clear skies and high statistics.