Shell for Simulation Workflows

Practical shell fundamentals for PDE/CFD/DSMC projects

1 Basic Shell for Simulation Software

For most of simulation software (except GUIs), the shell terminal is the primary interface to the tools. You either find executables (`a.out`, `programm.exe`, ..) or scripts (`bash run.sh`, `python3 launch.py`, `julia solver.jl`, ..) that you run from the terminal.

> **Note**
>
> For reproducability, we try to avoid GUIs and prefer command-line interfaces (CLIs) that can be scripted and version-controlled (see Git).

> **Important**
>
> Especially when running our simulation program on a cluster (RWTH CLAIX, see Figure 1) or server, we will not have access to a GUI and must rely on the shell.

Figure 1

1.1 Overview

The Shell is a command-line interface (CLI) used in Linux/Unix operating systems. It allows users to interact with the system by typing commands, rather than using a graphical interface. It is text-based and provides powerful tools for file management, process control, and automation. Common shells include Bash, zsh, and fish.

You can excesss the shell through a terminal application on your operating system:

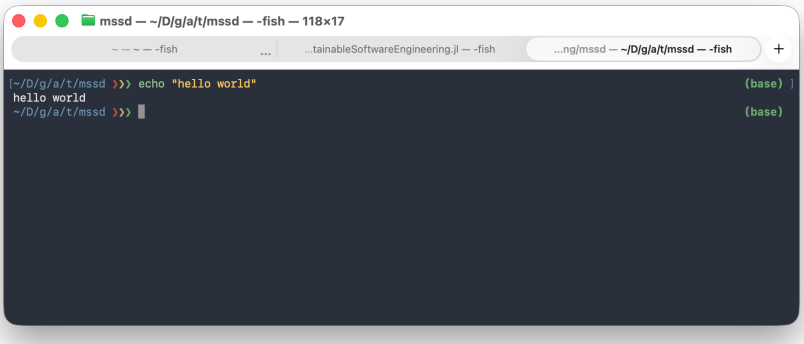
1. **Linux**: Terminal app
2. **macOS**: Terminal app
3. **Windows**: Windows Subsystem for Linux (WSL), Git Bash, PowerShell, …

1.2 Basic commands

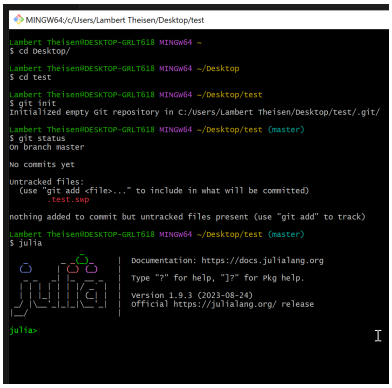
- `pwd`: Print working directory. Shows the current directory you are in.

```
pwd
```

```
/builds/rwth-acom/teaching/mssd/skills
```



(a) macOS terminal



(a) Git Bash on Windows

- `ls`: List files and directories. Shows the contents of the current directory.

```
ls  # alias/function forces --color=never
```

```
01-shell.html
01-shell.qmd
01-shell.quarto_ipynb
01-shell_files
02-git.qmd
02-git.quarto_ipynb
03-gmsh.ipynb
04-paraview.ipynb
99-todo.qmd
99-todo.quarto_ipynb
length.txt
log.txt
output.csv
postprocess.sh
sleep.pid
solver.sh
test_file.txt
test_output.txt
test_repo
```

```
ls -l
```

```
total 2080
-rw-r--r--. 1 root root   66929 May  7 09:07 01-shell.html
-rw-rw-rw-. 1 root root   11892 May  4 12:50 01-shell.qmd
-rw-r--r--. 1 root root   47932 May  7 09:07 01-shell.quarto_ipynb
drwxr-xr-x. 4 root root      41 May  7 09:07 01-shell_files
-rw-rw-rw-. 1 root root   18051 May  4 12:50 02-git.qmd
-rw-r--r--. 1 root root   26098 May  7 09:04 02-git.quarto_ipynb
-rw-rw-rw-. 1 root root 1581904 May  5 14:12 03-gmsh.ipynb
-rw-rw-rw-. 1 root root  319617 May  5 14:12 04-paraview.ipynb
```

```
-rw-rw-rw-. 1 root root     217 May  4 12:50 99-todo.qmd
-rw-r--r--. 1 root root     763 May  7 09:04 99-todo.quarto_ipynb
-rw-r--r--. 1 root root       8 May  7 09:07 length.txt
-rw-r--r--. 1 root root      40 May  7 09:07 log.txt
-rw-r--r--. 1 root root      12 May  7 09:07 output.csv
-rw-r--r--. 1 root root     152 May  7 09:07 postprocess.sh
-rw-r--r--. 1 root root       5 May  7 09:07 sleep.pid
-rwxr-xr-x. 1 root root     113 May  7 09:07 solver.sh
-rw-r--r--. 1 root root      21 May  7 09:07 test_file.txt
-rw-r--r--. 1 root root      38 May  7 09:07 test_output.txt
drwxr-xr-x. 3 root root      34 May  7 09:01 test_repo
```

```
ls -a
```

```
.
..
.gitignore
01-shell.html
01-shell.qmd
01-shell.quarto_ipynb
01-shell_files
02-git.qmd
02-git.quarto_ipynb
03-gmsh.ipynb
04-paraview.ipynb
99-todo.qmd
99-todo.quarto_ipynb
length.txt
log.txt
output.csv
postprocess.sh
sleep.pid
solver.sh
test_file.txt
test_output.txt
test_repo
```

- `cd`: Change directory. Moves you to a different directory.

```
pwd; cd ..; pwd; cd skills; pwd
```

```
/builds/rwth-acom/teaching/mssd/skills
/builds/rwth-acom/teaching/mssd
/builds/rwth-acom/teaching/mssd/skills
```

- `echo`: Print text to the terminal. Useful for displaying messages or environment variable values in scripts.

```
echo "Hello, Shell!"
```

```
Hello, Shell!
```

```
NAME="Lambert"; echo "hello, my name is $NAME."
```

```
hello, my name is Lambert.
```

- `touch`: Create an empty file..

```
touch test_file.txt
```

```
ls -l test_file.txt
```

```
-rw-r--r--. 1 root root 21 May  7 09:07 test_file.txt
```

- `mkdir`: Make directory. Creates a new directory.

```
mkdir -p test_dir/subdir
```

```
ls -l test_dir
```

```
total 0
drwxr-xr-x. 2 root root 6 May  7 09:07 subdir
```

- `rmdir`: Remove directory. Deletes a directory and its contents.

```
mkdir new_dir
```

```
rmdir new_dir
```

- `cp`: Copy files and directories.

```
touch source_file.txt
```

```
cp source_file.txt copied_file.txt
```

```
ls -l source_file.txt copied_file.txt
```

```
-rw-r--r--. 1 root root 0 May  7 09:07 copied_file.txt
-rw-r--r--. 1 root root 0 May  7 09:07 source_file.txt
```

- `mv`: Move or rename files and directories.

```
mv copied_file.txt renamed_file.txt
```

```
ls -l renamed_file.txt
```

```
-rw-r--r--. 1 root root 0 May  7 09:07 renamed_file.txt
```

- `rm`: Remove files and directories. Use flag `-r` for recursive deletion.

```
ls -l source_file.txt renamed_file.txt
```

```
-rw-r--r--. 1 root root 0 May  7 09:07 renamed_file.txt
-rw-r--r--. 1 root root 0 May  7 09:07 source_file.txt
```

```
rm source_file.txt renamed_file.txt
```

```
ls -l source_file.txt renamed_file.txt
```

```
ls: cannot access 'source_file.txt': No such file or directory
ls: cannot access 'renamed_file.txt': No such file or directory
```

```
rm -r test_dir
```

```
ls -l test_dir
```

```
ls: cannot access 'test_dir': No such file or directory
```

- `cat`: Concatenate and display file contents. Useful for quickly viewing the contents of a file.

```
echo 'This is a test file.' > test_file.txt  # note the > operator to write to the file
```

```
cat test_file.txt
```

```
This is a test file.
```

- `grep`: Search for patterns in files. Useful for finding specific information in log files or code.

```
echo -e 'line1: error\nline2: warning\nline3: info' > log.txt
```

```
grep 'error' log.txt
```

```
line1: error
```

```
grep -n 'warning' log.txt
```

```
2:line2: warning
```

- `find`: Search for files and directories. Useful in combination with wildcards (see GNU Bash: Pattern Matching).

```
find . -name '*.txt'
```

```
./test_file.txt
./log.txt
./test_output.txt
./length.txt
./test_repo/file.txt
```

- `man`: Manual. Displays the manual page for a command, providing detailed information on its usage and options.

```
man echo | col -bx | head -n 10  # col -bx to strip formatting
```

```
bash: line 2: col: command not found
```

- `|`, `>`, `>>`, `&&`, `||`, `;`, `&`: Shell operators for piping, redirection, and command chaining.

```
echo 'Hello, World!' | tr '[:lower:]' '[:upper:]'  # pipe output to another command (tr for up
```

```
HELLO, WORLD!
```

```
echo 'This is a test.' > test_output.txt  # redirect output to a file (overwrite)
```

```
echo 'This is another line.' >> test_output.txt  # redirect output to a file (append)
```

```
echo 'First command' && echo 'Second command'  # execute second command only if first succeeds
```

```
First command
Second command
```

```
false || echo 'First command failed, executing second command'  # execute second command only
```

```
First command failed, executing second command
```

```
echo 'Command 1'; echo 'Command 2'  # execute commands sequentially regardless of success
```

```
Command 1
Command 2
```

```
sleep 0.1 & echo 'This runs in the background while sleep is running'  # run command in the ba
```

```
This runs in the background while sleep is running
```

1.3 System Information and Utils

- **df**: Report file system disk space usage.

```
df -h | head -n 5
```

```
Filesystem      Size  Used Avail Use% Mounted on
overlay         297G  116G  181G  39% /
tmpfs            64M     0   64M   0% /dev
shm             512M     0  512M   0% /dev/shm
/dev/sdb1       200G   61G  140G  31% /cache
```

- **top**: Display Linux tasks.

```
top -n 1 | head -n 20
```

```
top: failed tty get
```

- **du**: Estimate file space usage.

```
du -sh *
```

```
68K 01-shell.html
12K 01-shell.qmd
48K 01-shell.quarto_ipynb
0   01-shell_files
20K 02-git.qmd
28K 02-git.quarto_ipynb
1.6M    03-gmsh.ipynb
316K    04-paraview.ipynb
4.0K    99-todo.qmd
4.0K    99-todo.quarto_ipynb
4.0K    length.txt
4.0K    log.txt
4.0K    output.csv
4.0K    postprocess.sh
4.0K    sleep.pid
4.0K    solver.sh
4.0K    test_file.txt
4.0K    test_output.txt
176K    test_repo
```

- **htop**: Interactive process viewer (if installed).
- **kill**: Terminate processes by PID.

```
sleep 10 & echo $! > sleep.pid
```

```
cat sleep.pid
```

```
2907
```

```
kill $(cat sleep.pid)  # terminate the background process
```

```
bash: line 2: kill: (2907) - No such process
```

1.4 Tips

- Use the **Tab** key for auto-completing commands and file names.
- Use **up and down arrow keys** to navigate through command history.
- Use **Ctrl + C** to terminate a running command.
- Use **Ctrl + L** to clear the terminal screen.
- Use **Ctrl + R** to search through command history.

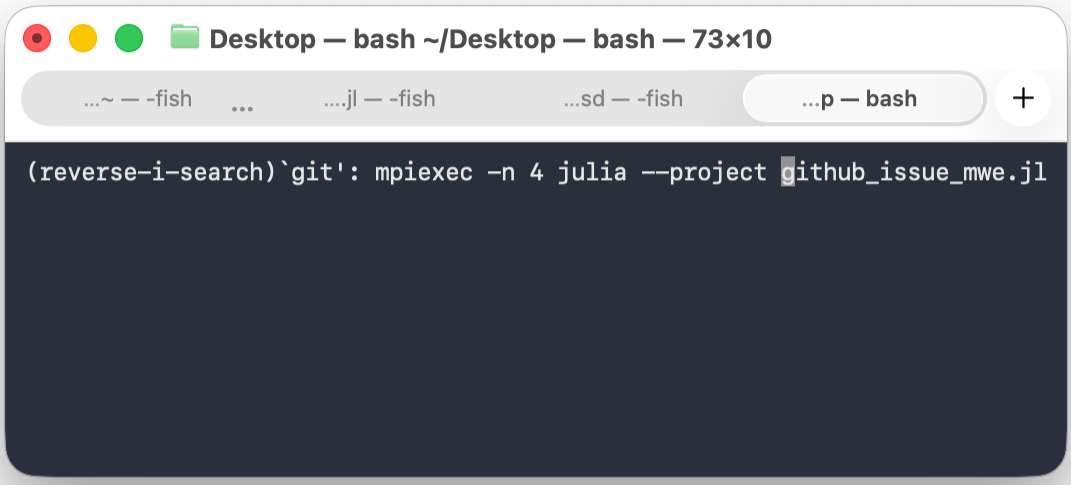


Figure 4: Shell Search

2 Scripting

Scrripting and automation are key for reproducibility and efficiency in simulation workflows. You can write shell scripts (`.sh` files) to automate sequences of commands, set up environments, and run simulations with specific parameters.

An example script:

```
#!/bin/bash
# This is a simple shell script to run a simulation and post-process results
echo "Running simulation..."
echo "Filename: $1"

# Write some data to filename
echo "x,y,z" > $1
echo "1,2,3" >> $1
echo "Simulation complete. Output written to $1"
```

2.1 An example solver

```
echo "#!/bin/bash
echo 'Running solver...'
echo 'x,y,z' > output.csv
echo '1,2,3' >> output.csv
echo 'Write output...'" > solver.sh
```

```
ls -l solver.sh
chmod +x solver.sh  # make script executable
ls -l solver.sh
```

```
-rwxr-xr-x. 1 root root 113 May  7 09:07 solver.sh
-rwxr-xr-x. 1 root root 113 May  7 09:07 solver.sh
```

```
./solver.sh  # run the script
sh solver.sh  # alternative way to run the script
```

```
Running solver...
Write output...
Running solver...
Write output...
```

```
cat output.csv  # check the output
```

```
x,y,z
1,2,3
```

2.2 An example post-processing script

```
echo "#!/bin/bash
echo 'Calculate length of vector...'
awk -F, 'NR>1 {print sqrt(\$1^2 + \$2^2 + \$3^2)}' output.csv > length.txt
echo 'Write length to file...'" > postprocess.sh
cat postprocess.sh
```

```
#!/bin/bash
echo 'Calculate length of vector...'
awk -F, 'NR>1 {print sqrt($1^2 + $2^2 + $3^2)}' output.csv > length.txt
echo 'Write length to file...'
```

```
sh postprocess.sh  # run the post-processing script
cat length.txt  # check the output
```

```
Calculate length of vector...
Write length to file...
3.74166
```

3 Integrated Development Environments (IDEs)

IDEs (e.g., VS Code, Vim, Emacs, Jetbrains, Atom) combine a text editor with additional features like terminal integration, version control, debugging tools, and extensions for specific languages and frameworks.

- **VS Code**: All-in-one solution, rich extensions, user-friendly
- **Vim/Emacs**: Lightweight, highly customizable, steep learning curve

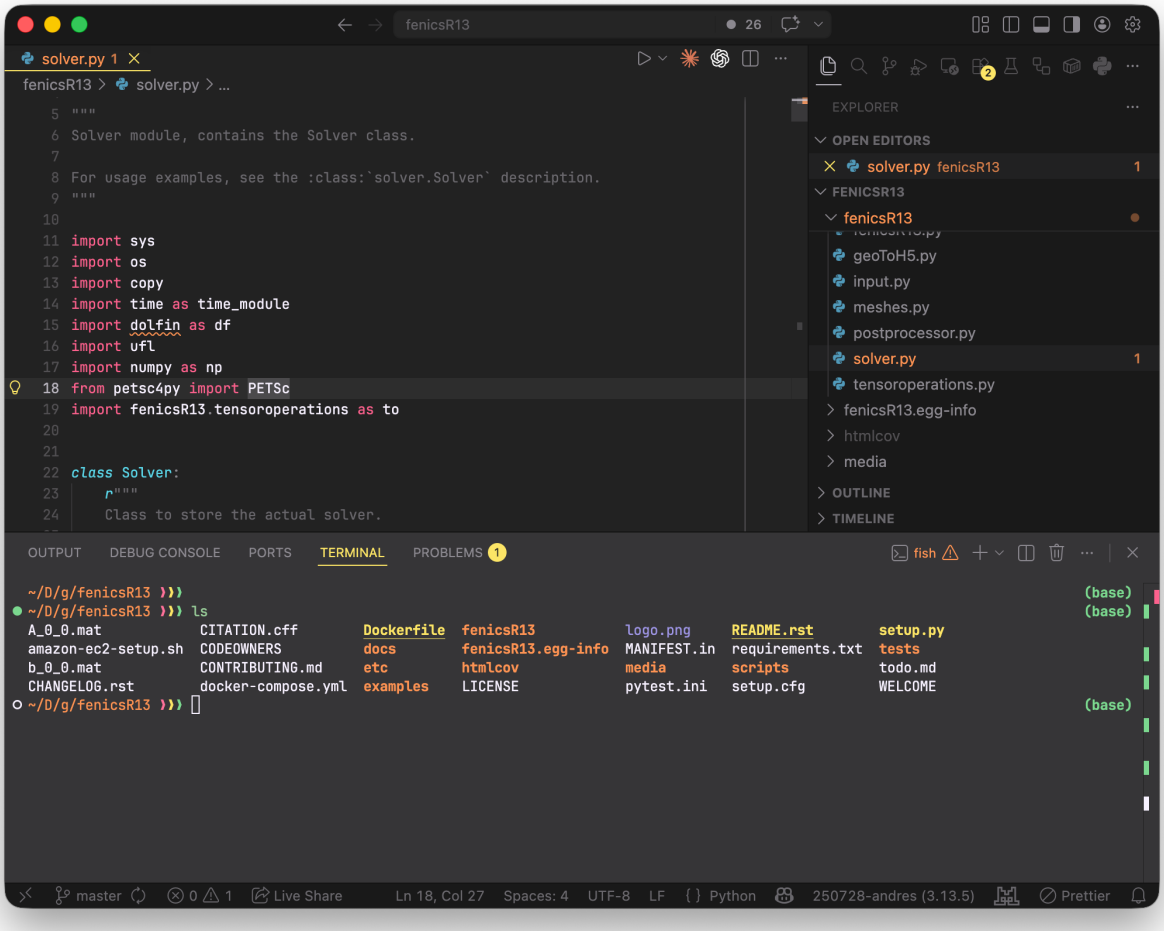


Figure 5: fenicsR13 Solver in VS Code

4 References and Further Reading

- MIT: The Missing Semester of Your CS Education
- Bash Guide
- Shell Scripting Tutorial
- Random collection of useful commands by Dr. Georgii Oblapenko

5 Exercises

1. Write a shell script that takes a filename as an argument, checks if the file exists, and prints its contents if it does, or an error message if it doesn't.
2. Use `curl` or `wget` to download our course website and use `grep` to extract all dates mentioned on the page.

6 Questions

1. What are the advantages of using the shell for running simulations compared to GUIs?
2. How can you automate a simulation workflow using shell scripts?
3. What are some common shell operators and how do they work?
4. How can you manage and navigate the file system using shell commands?

More good exercises can be found in the MIT Missing Semester course.