Open Source CFD

Tutorial #0: Linux Fundamentals for CFD

Getting Started with Ubuntu 24.04, Bash Shell, Git and python

Robert Castilla
Department of Fluid Mechanics

2025-26
Version 1.0
Learning Objectives:

- Install and setup linux distribution Ubuntu
- Get familiar with bash shell and principal commands
- Write and run a simple bash script
- Setup and use git as a version control system
- Install and setup python

This tutorial introduces the essential Linux skills needed for the CFD course. You'll learn to set up Ubuntu 24.04, navigate the filesystem using the bash shell, write simple scripts, and use Git for version control. These foundations will prepare you for working with OpenFOAM, FreeCAD, and managing your CFD projects effectively.

Contents

1. Getting Started with Ubuntu 24.04 LTS

1.1. What is Ubuntu?

Ubuntu is a Linux distribution based on Debian, known for being user-friendly andits extensive community support. Version 24.04 is a **Long Term Support (LTS)** release, meaning it receives security updates and maintenance until 2029.

Why Ubuntu for CFD?

- OpenFOAM is natively supported on Ubuntu
- FreeCAD and CfdOF work seamlessly
- Extensive documentation and community help
- Package manager (APT) simplifies software installation

1.2. Installation Options

You have two main ways to get Ubuntu:

Step 1: Dual Boot (Recommended): Install alongside Windows

- Best performance for CFD simulations
- Access to all system resources
- Requires partitioning your hard drive

Step 2: Virtual Machine: Run Ubuntu inside Windows

- Easier setup (using VirtualBox or Multipass)
- Slightly slower for heavy computations
- Good for testing and learning

TIP: You have already available an `ova` file that can be used to create a VirtualBox virtual machine.

1.3. Creating a Bootable USB Stick

1. **Download Ubuntu**: Get the ISO from `https://ubuntu.com/download`
2. **Create bootable USB**:
   - **Windows**: Use Rufus
   - **Linux**: Use `dd` command or `Startup Disk Creator`
   - **macOS**: Use balenaEtcher

TIP: Rufus (Windows) or balenaEtcher (macOS) are the most user-friendly options.

1.4. Installation Steps

1. Boot from USB (press `F12`, `F2`, or `Esc` during startup)
2. Select **"Try or Install Ubuntu"**
3. Choose language and keyboard layout
4. Connect to WiFi (optional but recommended)
5. Select **"Extended selection"** for more pre-installed apps
6. Enable **"Install third-party software"** for drivers and codecs
7. Choose installation type:
   - "Install alongside Windows" for dual boot
   - "Erase disk and install Ubuntu" for clean install
8. Create user account and password
9. Wait for installation to complete and reboot

WARNING: Back up your data before partitioning or installing any operating system!

1.5. Post-Installation Setup

After first boot, run these essential updates:

```
# Update package lists
sudo apt update

# Upgrade installed packages
sudo apt upgrade

# Update snap packages
sudo snap refresh

# Clean up unnecessary packages
sudo apt autoremove
sudo apt autoclean
```

Listing 1: Initial system update

IMPORTANT: Always run `sudo apt update` before installing new software to ensure you get the latest versions.

1.6. Installing Essential Software

```
# Build essentials (compilers, make, etc.)
sudo apt install build-essential

# Text editor
sudo apt install vim nano

# System monitoring
sudo apt install htop btop neofetch

# File utilities
sudo apt install tree ncdu

# Compression tools
sudo apt install unzip p7zip-full
```

Listing 2: Install basic tools

TIP: Use `htop` to monitor system resources while running CFD simulations.

2. The Bash Shell

2.1. What is a Shell?

The shell is a command-line interface that lets you interact with your operating system. **Bash** (Bourne Again SHell) is the default shell in Ubuntu and most Linux distributions.

2.2. Opening the Terminal

- Press `Ctrl+Alt+T`
- Search for "Terminal" in applications menu
- Right-click desktop → "Open in Terminal"

2.3. Navigating the Filesystem

Command	Meaning	Example
`pwd`	Print Working Directory	`pwd` → `/home/username`
`ls`	List directory contents	`ls -la` (detailed view)
`cd`	Change Directory	`cd Documents`
`cd ..`	Go up one level	`cd ../..` (up two levels)
`cd ~`	Go to home directory	`cd ~`
`cd -`	Go to previous directory	`cd -`

Table 1: Essential navigation commands

```
# Where am I?
pwd

# What's in this directory?
ls -la

# Go to Documents
cd ~/Documents

# Create a new directory
mkdir cfd_tutorials

# Go into it
cd cfd_tutorials

# Go back home
```

```
cd
```

Listing 3: Practice navigation

2.4. File Operations

Command	Purpose	Example
`cp`	Copy	`cp <file1> <file2>`
`mv`	Move/Rename	`mv <oldname> <newname>`
`rm`	Remove	`rm <file>`
`rm -r`	Remove directory	`rm -r <folder>`
`mkdir`	Create directory	`mkdir <folder>`
`touch`	Create empty file	`touch <file>`
`cat`	Display file	`cat <file>`
`less`	View file page by page	`less <file>`

Table 2: File and directory operations

WARNING: Be careful with `rm -r`! It permanently deletes files without a trash bin.

2.5. Understanding Paths

- **Absolute path**: Starts from root directory

  ```
  /home/username/Documents/cfd_tutorials
  ```

- **Relative path**: Relative to current location

  ```
  Documents/cfd_tutorials # If you're in /home/username
  ../Downloads # One level up, then into Downloads
  ```

TIP: Use `Tab` for auto-completion! Type the first few letters and press `Tab`.

2.6. Useful Shortcuts

Shortcut	Function
`Ctrl+C`	Cancel current command
`Ctrl+Z`	Suspend current process
`Ctrl+D`	Exit shell/logout
`Ctrl+L`	Clear screen
`Ctrl+A`	Go to beginning of line
`Ctrl+E`	Go to end of line
`Ctrl+R`	Search command history
`Up/Down arrows`	Navigate command history
`Tab`	Auto-complete filenames/commands

Table 3: Keyboard shortcuts for terminal efficiency

2.7. Wildcards and Pattern Matching

```
# List all .txt files
ls *.txt

# Remove all backup files
rm *.bak

# List files starting with ’data’ followed by any characters
ls data*

# List files with single-character wildcard
ls data?.csv # matches data1.csv, data2.csv, but not data10.csv
```

Listing 4: Using wildcards

2.8. Input/Output Redirection

Operator	Meaning	Example
`>`	Redirect output to file (overwrite)	`ls > filelist.txt`
`»`	Redirect output to file (append)	`echo "new" » file.txt`
`<`	Take input from file	`sort < unsorted.txt`
`|`	Pipe output to another command	`ls -la | grep "txt"`

Table 4: Redirection and pipes

```bash
# Count files in directory
ls -1 | wc -l

# Find specific files
ls -la | grep "cfd"

# Sort and display unique
cat data.txt | sort | uniq
```

Listing 5: Pipe examples

2.9. Process Management

```bash
# List running processes
ps aux

# Interactive process viewer
htop

# Run command in background
./longsimulation.sh &

# Bring background job to foreground
fg

# Kill a process
kill -9 PID
```

Listing 6: Managing running processes

3. Introduction to Bash Scripting

3.1. What is a Shell Script?

A shell script is a text file containing a series of commands that can be executed as a program. This automates repetitive tasks.

3.2. Your First Script

1. Create a new file:

   ```bash
   vim firstscript.sh
   ```

2. Add the following content:

```bash
#!/bin/bash
# This is a comment - my first script

echo "Hello, CFD world!"
echo "Current directory: $(pwd)"
echo "Files here:"
ls -la
```

Listing 7: hello.sh

3. Make it executable:

```bash
chmod +x firstscript.sh
```

4. Run it:

```bash
./firstscript.sh
```

IMPORTANT: The line `#!/bin/bash` (shebang) tells the system which interpreter to use. Without it, the script won't execute properly.

3.3. Variables in Bash

```bash
#!/bin/bash

# Defining variables (no spaces around =)
name="Student"
course="CFD"
simulation_time=3600

# Using variables (with $)
echo "Hello, $name!"
echo "Welcome to $course course."

# Command substitution - store command output
files=$(ls -la)
echo "Files in current directory: $files"

# Arithmetic
a=10
b=20
sum=$((a + b))
echo "Sum: $sum"
```

Listing 8: variables.sh

3.4. Special Variables

Variable	Meaning
`$0`	Script name
`$1`, `$2`, ...	Positional parameters (arguments)
`$#`	Number of arguments
`$*`	All arguments as single string
`$@`	All arguments as separate strings
`$?`	Exit status of last command
`$$`	Process ID of current script

Table 5: Special shell variables

```
#!/bin/bash
echo "Script name: $0"
echo "First argument: $1"
echo "Second argument: $2"
echo "Number of arguments: $#"
echo "All arguments: $@"

if [ $# -eq 0 ]; then
echo "Error: No arguments provided!"
exit 1
fi
```

Listing 9: arguments.sh

3.5. Conditional Statements

Operator	Meaning
File tests	
`-e file`	File exists
`-f file`	File exists and is regular file
`-d file`	File exists and is directory
Numeric comparisons	
`-eq`	Equal to
`-ne`	Not equal to
`-lt`	Less than
`-gt`	Greater than
String comparisons	
`=`	Strings equal
`!=`	Strings not equal
`-z`	String is empty

Table 6: Common test operators

```bash
#!/bin/bash

# File tests
if [ -e "mesh.geo" ]; then
echo "mesh.geo exists"
else
echo "mesh.geo not found"
fi

# String comparison
name="OpenFOAM"
if [ "$name" = "OpenFOAM" ]; then
echo "Correct solver"
fi

# Numeric comparison
value=10
if [ $value -gt 5 ]; then
echo "Value is greater than 5"
fi
```

Listing 10: conditions.sh

3.6. Loops

```bash
#!/bin/bash

# For loop over explicit values
echo "=== For loop with values ==="
for fruit in apple banana orange; do
echo "I like $fruit"
done

# For loop with range
echo "=== For loop with range ==="
for i in {1..5}; do
echo "Iteration $i"
done

# For loop over files
echo "=== For loop over files ==="
for file in *.txt; do
echo "Processing $file"
wc -l "$file"
done

# While loop
echo "=== While loop ==="
counter=1
while [ $counter -le 5 ]; do
echo "Counter: $counter"
counter=$((counter + 1))
done
```

Listing 11: loops.sh

3.7. Reading User Input

```bash
#!/bin/bash

# Basic input
echo -n "Enter your name: "
read name
echo "Hello, $name!"

# Prompt with message
read -p "Enter Reynolds number: " Re
echo "Reynolds number: $Re"

# Hidden input (for passwords)
read -s -p "Enter password: " password
echo "Password accepted."
```

```
# Read multiple values
read -p "Enter x y z coordinates: " x y z
echo "Coordinates: ($x, $y, $z)"
```

Listing 12: input.sh

4. Version Control with Git

4.1. What is Git?

Git is a distributed version control system created by Linus Torvalds in 2005 for Linux kernel development. It tracks changes to files, enables collaboration, and maintains complete history of your projects.

Why Git for CFD?

- Track changes to simulation setups
- Collaborate with team members
- Revert to working versions when experiments fail
- Share cases with reproducibility

4.2. Installing and Configuring Git

```
sudo apt install git
git --version # Verify installation
```

Listing 13: Install Git

4.3. First-Time Setup

```
# Set your identity (required for commits)
git config --global user.name "Your Name"
git config --global user.email "your.email@example.com"

# Set default editor
git config --global core.editor "vim"

# View all settings
git config --list
```

Listing 14: Configure Git

4.4. Creating Your First Repository

```
# Create project directory
mkdir pitzDaily_case
cd pitzDaily_case

# Initialize Git repository
git init

# Check status (should show empty repo)
git status
```

Listing 15: Initialize a repository

After `git init`, you'll have a hidden `.git` directory containing all version control information.

4.5. Basic Git Workflow

The typical Git workflow has three stages:

1. **Working Directory**: Where you edit files
2. **Staging Area** (index): Where you prepare changes
3. **Repository**: Where commits are permanently stored

```
Working Directory    git add    Staging Area    git commit    Repository
(files)             -------->    (index)          -------->   (commits)
^                                                                 |
|                                                                  |
+------------------------ git checkout <--------------------------+
```

Figure 1: Git workflow stages

4.6. Tracking Changes

```
# Create a file
echo "# pitzDaily Simulation" > README.md
echo "U uniform (10 0 0);" > 0/U

# Check status
git status

# Stage files
```

```
git add README.md
git add 0/U

# Or stage all changes
git add .

# Check status again (files now staged)
git status

# Commit with message
git commit -m "Initial case setup: README and velocity BC"

# View commit history
git log --oneline
```

Listing 16: First commit

4.7. Checking Differences

```
# See unstaged changes
git diff

# See staged changes
git diff --staged

# Compare with previous commit
git diff HEAD^

# Show specific commit
git show COMMIT_HASH
```

Listing 17: Viewing changes

4.8. Viewing History

```
# Simple log
git log --oneline

# Detailed log with graph
git log --graph --oneline --decorate --all

# Show last 3 commits
git log -3

# Search commits by message
git log --grep="mesh refinement"
```

Listing 18: git log examples

The output shows commit hash, author, date, and message:

```
abc1234 (HEAD -> main) Update boundary conditions
def5678 Add mesh refinement study
ghi9012 Initial commit
```

4.9. Ignoring Files

Create `.gitignore` to exclude files from version control:

```
# Ignore OpenFOAM processor directories
processor*/

# Ignore log files
*.log
log.*

# Ignore backup files
*~
*.bak

# Ignore large result files
*.vtk
*.foam

# Ignore IDE files
.vscode/
.idea/
```

Listing 19: .gitignore

4.10. Undoing Changes

```
# Unstage a file (keep changes)
git reset HEAD filename

# Discard changes in working directory
git checkout -- filename

# Amend last commit (fix message or add forgotten files)
git commit --amend -m "Better commit message"

# Revert to previous commit (creates new commit)
git revert HEAD
```

Listing 20: Fixing mistakes

WARNING: Be careful with `git reset` and `git checkout` as they can permanently discard changes!

4.11. Cloning and Pulling

```
# Clone a repository
git clone https://github.com/username/pitzDaily_case.git

# Get latest changes
git pull

# Fetch without merging
git fetch
```

Listing 21: Working with existing repositories

4.12. Branching (Advanced)

Branches allow parallel development:

```
# List branches
git branch

# Create new branch
git branch mesh-refinement

# Switch to branch
git checkout mesh-refinement

# Create and switch in one command
git checkout -b turbulence-study

# Merge branch into main
git checkout main
git merge mesh-refinement

# Delete branch
git branch -d mesh-refinement
```

Listing 22: Branch basics

4.13. Removing the local repository

The local repository can be easily removed by just deleting the `.git` folder.

```
# Removing git repository
rm -rf .git
```

Listing 23: Branch basics

TIP: The `-f` option is to `force` the removing without asking for confirmation

WARNING: Be cautious with this command. It can not be undone!

It is also possible to work with remote repositories, GitHub for instance, but it is an advanced topic that is out of scope of the present Tutorial

5. Simple computations with python

5.1. What is python?

Python is a high-level computation language. It is simular to Matlab, but it is open-source and free. You can make from simple mathematical operations to very complex program, by using the huge repositoty of modules and packages available.

Why python for CFD?

- Data analysis of results
- Automation of tasks

5.2. Installation of python and an IDE (Integrated Development Enviroment)

By default, python version 3 is installed in Ubuntu 24.04

```
# python version
python3 --version

# basic interface for python
python3
```

Listing 24: python basics

However, it is preferable to make a personal environment and install an IDE. There are several options for an IDE for python:

1. Jupyter Lab
2. Spyder
3. PyCharm

4. Google Colab
5. ...

All these IDE's have pros and cons. In this tutorial we install and configure the web browser version of Jupyter Lab.

First, install all the needed packages

```
# packages installation
sudo apt install python3-pip python3-venv python3-full

# Make the local virtual enviroment
python3 -m venv my_env

# Activate the enviroment
source ./my_venv/bin/activate
```

Listing 25: Installing python packages and creating the environment

After the activation of the virtual environment, you will get a `(my_env)` at the beginning of the prompt in the terminal, meaning that you are working in a local virtual environment (not in the global python system) and all the python modeules installed will be in this local venv.

TIP: You can create as much virtual enviroments as you want. Each one of these `venv`'s can have different modules and/or different versions.

To exit form the local virtual enviroment, just deactivate it

```
# venv deactivation
deactivate
```

Listing 26: Deactivation of the virtual environment

Install the usual modules needed for Engineering computations in python. Be sure to have the virtual enviroment activated.

```
# modules installation
pip install jupyter numpy scipy matplotlib pandas

# Run jupyter
jupyter lab
```

Listing 27: Installing python modules and running jupyter lab

Jupyter Lab runs in the web broser with a local server.

6. Practice Exercises

6.1. Exercise 1: Filesystem Navigation

1. Create directory structure: `cfd-course/tutorials/pitzDaily`
2. Navigate to this directory
3. Create files: `README.md`, `0/U`, `constant/transportProperties`
4. List all files recursively
5. Count number of files in the project

6.2. Exercise 2: Bash Scripting

Create a script `setup_case.sh` that:

1. Checks if directory exists, creates if not
2. Prompts user for Reynolds number
3. Creates basic OpenFOAM file structure
4. Prints summary of what was created

Solution template:

```
#!/bin/bash
# CFD case setup script

case_name=$1
if [ -z "$case_name" ]; then
read -p "Enter case name: " case_name
fi

# Your code here...
```

6.3. Exercise 3: Git Practice

1. Initialize Git repository in your case directory
2. Create and commit initial files
3. Make changes, view differences
4. View commit history
5. Create `.gitignore` for CFD files

6.4. Exercise 4: Combine Skills

Create a script that:

1. Creates a new Git repository
2. Sets up standard CFD case structure
3. Makes initial commit
4. Displays repository status and log

7. Common Issues and Solutions

7.1. Permission Denied

```
# Make script executable
chmod +x script.sh

# Check permissions
ls -la script.sh
```

7.2. Command Not Found

```
# Check if installed
which <command>

# Install if missing
sudo apt install <package>
```

8. Additional Resources

8.1. Online Tutorials

- Ubuntu Official Tutorials
- The Bash Shell - Southampton course
- Official Git Documentation

8.2. Cheat Sheets

- Ubuntu command line interface (CLI)
- Ubuntu keyboard shortcuts
- Bash commands reference
- Git quick reference

8.3. Books

- "The Ultimate Ubuntu Handbook" by Ken VanDine
- "Beginning Ubuntu for Windows and Mac Users" by Nathan Haines
- "Pro Git" by Scott Chacon (free online)

8.4. Community

- Ask Ubuntu
- Stack Overflow (tag: bash, git)
- Ubuntu Discourse

IMPORTANT: Keep practicing! The command line becomes intuitive with regular use.

A. Quick Reference Cards

A.1. Ubuntu Shortcuts

`Ctrl + Alt + D`	Show desktop
`Alt + Tab`	Switch applications
`Ctrl + Alt + T`	Open terminal
`PrtScn`	Take screenshot

A.2. Bash Commands Summary

`ls -la`	List all files with details
`cp -r`	Copy recursively
`grep <pattern> <file>`	Search in files
`history`	Show command history
`man <command>`	Show manual
`which <program>`	Locate program

A.3. Git Commands Summary

`git init`	Initialize repository
`git add`	Stage changes
`git commit -m "msg"`	Commit staged changes
`git status`	Check status
`git log`	View history
`git diff`	Show changes

Acknowledgments: The author acknowledges the assistance of DeepSeek, an AI assistant created by DeepSeek Company, for providing valuable support in structuring this document, formatting LaTeX code, and developing the tutorial content.

Disclaimer: All content has been reviewed, modified, and validated by the author. The author takes full responsibility for the accuracy, completeness, and educational quality of this material.