



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Appendix 0 Part 1

TERMINAL AND CLI
FOUNDATIONS

Today's Learning Objectives

By the end of this session, you will:

1. **Understand** what a terminal is and why developers use it
2. **Know** the difference between a terminal, a shell, and an OS
3. **Navigate** your file system using text commands
4. **Read** command syntax like `git commit -m "message"`
5. **Understand** PATH and fix "command not found" errors
6. **See** why automation through scripting is a core skill

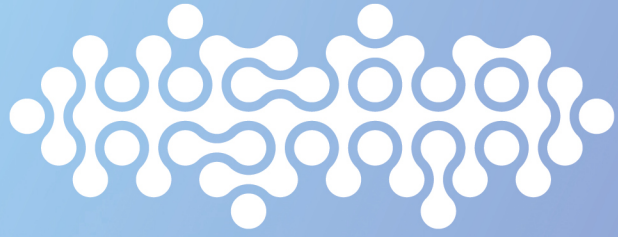
Why This Matters

Everything in this course runs through the terminal:

- Installing Git, Python, and tools
- Cloning repositories and managing code
- Running tests, linters, and formatters
- CI/CD pipelines and deployment

If "open a terminal and type this" sounds intimidating -- this session is for you.

Many professional developers started exactly where you are now.



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

What Is a Terminal?

1

A Text Conversation with Your Computer

Think of it like a chat:

```
You:      "Show me all files in this folder"
Computer: "document.txt  photo.jpg  notes.md"

You:      "Create a new folder called 'project'"
Computer: "(silence means success)"
```

In practice:

```
$ ls
document.txt  photo.jpg  notes.md

$ mkdir project
```

The `$` is the **prompt** -- it means "ready for your next command." Don't type it!

Quick Poll

Have you ever used a terminal before?

- **A)** Never -- this is all new to me
- **B)** A few times, but I'm not comfortable
- **C)** I use it occasionally
- **D)** I'm already comfortable with it

No wrong answer! This session brings everyone to the same level.

From Physical Devices to Software

1960s-70s: A "terminal" was a physical keyboard + screen connected to a mainframe

Today: Terminal emulators -- software that simulates that text conversation

Still the foundation of modern computing:

- Servers running websites
- Cloud infrastructure
- Automated testing pipelines
- Deployment systems

Every professional developer, sysadmin, and DevOps engineer uses the terminal daily.

Why Developers Still Use the Terminal

Practical reasons:

- **Speed** -- 3 seconds vs. 30 seconds
- **Automation** -- save commands in scripts
- **Precision** -- no hidden GUI assumptions

Professional reasons:

- **Reproducibility** -- share via text
- **Remote access** -- servers have no GUI
- **Tool ecosystem** -- Git, pip, pytest, ruff

Key insight: The GUI buttons in VS Code are wrappers around terminal commands. Understanding the terminal = understanding the entire pipeline.

GUI vs. Terminal: A Comparison

Creating a project folder with a README:

GUI: Open Explorer → Navigate → Right-click → New Folder → Name it → Open → Right-click → New File → Name it → **7 steps**

Terminal:

```
mkdir project && cd project && touch README.md
```

1 line. Done.

And you can save this line and reuse it forever.

What About AI Assistants?

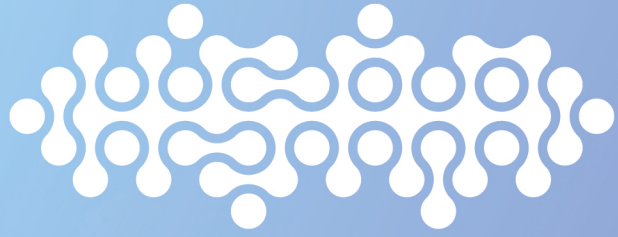
Tools like **GitHub Copilot CLI** can translate natural language to terminal commands.

So why learn terminal commands at all?

Same reason you learn math even though calculators exist:

- You need to **understand** what the AI suggests before running it
- You need to **verify and debug** when things go wrong
- You need **mental models** to ask good questions

AI amplifies understanding -- it doesn't replace it.



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Terminals, Shells, and Operating Systems

2

Different OS, Different Defaults

OS	Default Terminal	Default Shell	Prompt
Windows	Windows Terminal	PowerShell	PS C:\Users\you>
macOS	Terminal.app	zsh	you@mac ~ %
Linux	GNOME Terminal	bash	you@pc:~\$

Even the **prompt symbol** differs: `>`, `%`, `$`

All serve the same purpose: **"I'm ready for your next command."**

Terminal vs. Shell -- The Key Distinction

- **Terminal** = the **window** (handles display, colors, fonts, copy/paste)
- **Shell** = the **language** running inside it (interprets your commands)

Analogy:

The terminal is the **phone**. The shell is the **language** you speak on the phone.

Same phone, different languages. Same terminal window, different shells.

Why confusing? "PowerShell" is both a shell AND a terminal app. "Git Bash" is a terminal running bash. Names are inconsistent -- the concept matters.



Quick Question

When someone says "open bash," what do they actually mean?

- **A)** Open a terminal application that runs the bash shell
- **B)** Install a new program called "bash"
- **C)** Open a web browser
- **D)** Open the Windows Command Prompt

Think about the terminal vs. shell distinction!

► Answer

Windows: Many Terminal Options

Terminal	What It Is	For This Course?
Command Prompt	Oldest Windows terminal (1980s)	Rarely needed
PowerShell	Modern Windows default	Works, but syntax differs
Git Bash	Bash shell on Windows (installed with Git)	Recommended
WSL	Full Linux inside Windows	Advanced option

Our recommendation: Use **Git Bash** or the **VS Code terminal** configured to use Git Bash. This way all course examples work on every OS.

The VS Code Terminal

Open it: ``Ctrl+``` (backtick) or **View** → **Terminal**

The VS Code terminal is a terminal application with **your choice of shell** inside it.

Click the dropdown (▼) next to + to see available shells:

- PowerShell (default on Windows)
- Git Bash
- Command Prompt
- JavaScript Debug Terminal

Set Git Bash as default:

`Ctrl+Shift+P` → "Terminal: Select Default Profile" → Git Bash

Bash vs. Zsh

bash:

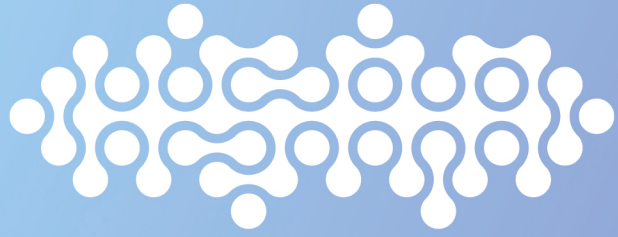
- Default on Linux, Git Bash
- Basic tab completion
- The scripting standard
- Config: `~/.bashrc`

zsh:

- Default on macOS (since 2019)
- Advanced tab completion
- Spelling correction
- Config: `~/.zshrc`

Both understand the same core commands: `cd`, `ls`, `mkdir`, `rm`, `cat`, `echo`, `grep` ...

Bottom line: Use what your system gives you. Don't overthink this.



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Navigating the File System

3

Reading the Prompt

```
user@computer:~/Documents$
```

This tells you:

- **Who you are:** user
- **Where you are:** ~/Documents (~ = home folder)
- **Ready for input:** \$

On Windows PowerShell:

```
PS C:\Users\you\Documents>
```

When tutorials show \$ at the beginning -- don't type it!

```
$ git status    # ← you only type: git status
```

Essential Navigation Commands

What You Want	Command	What It Does
Where am I?	<code>pwd</code>	Print working directory
What's here?	<code>ls</code>	List files and folders
Go into folder	<code>cd foldername</code>	Change directory
Go up one level	<code>cd ..</code>	Move to parent
Go home	<code>cd ~</code>	Move to home directory
Create folder	<code>mkdir name</code>	Make new directory
Create file	<code>touch name.txt</code>	Create empty file

These work in bash, zsh, Git Bash, **and** PowerShell (via aliases).

More Essential Commands

What You Want	Command	What It Does
Show file contents	<code>cat file.txt</code>	Display file content
Copy a file	<code>cp source dest</code>	Copy file
Move/rename	<code>mv old new</code>	Move or rename
Delete a file	<code>rm file.txt</code>	Remove (no undo!)
Clear screen	<code>clear</code>	Clean up display

Warning: `rm` has **no recycle bin**. Deleted files are gone. Be careful!

Understanding File Paths

Absolute path (full address):

```
# Linux/macOS
/home/user/Documents/README.md

# Windows
C:\Users\user\Documents\README.md
```

Relative path (from current location):

```
project/README.md
./project/README.md
../Downloads/file.zip
```

Special symbols: `.` = current dir, `..` = parent dir, `~` = home dir, `/` = root

Tip: Git Bash on Windows accepts forward slashes `/` -- use them!

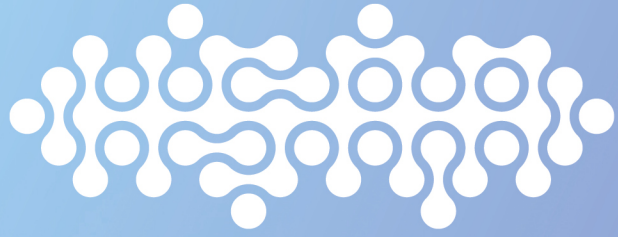


Hands-On: Navigate the File System

Follow along in your terminal:

```
pwd                # Where am I?
ls                 # What's in this folder?
mkdir terminal-practice # Create a new folder
cd terminal-practice # Go into it
pwd                # Confirm location
touch hello.txt     # Create an empty file
echo "Hello Terminal" > hello.txt # Write into it
cat hello.txt        # Read the file
cd ..                # Go back up
```

Take 3 minutes. Try each command. Ask if anything is unclear!



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Command Syntax and Help

4

Anatomy of a Command

```
git commit -m "Add README file"
```

Diagram illustrating the components of the command:

- argument: the value for -m
- flag: modifies behavior
- subcommand: the action
- program: the tool

Not every command has all parts:

```
git status      # program + subcommand
ls              # just a program
ls -la          # program + flags
cd Documents    # program + argument
```

Reading --help Output

Every good CLI tool includes built-in documentation:

```
git --help          # list all subcommands
git commit --help   # detailed help for commit
```

How to read the synopsis:

```
git commit [-a | --all] [-m <msg>] [<pathspec>...]
```

Notation	Meaning
[...]	Optional
<...>	Replace with your value
	Choose one
...	Multiple values allowed

Common Patterns Across All CLI Tools

These patterns repeat everywhere:

```
# Getting help
git --help          python --help          ruff --help

# Checking version
git --version       python --version       uv --version
```

Short vs. long flags:

```
git commit -m "message"      # short: -m
git commit --message "message" # long: --message
```

Combining short flags:

```
ls -l -a      # two separate flags
ls -la        # combined -- same result
```



Interactive: Parse This Command

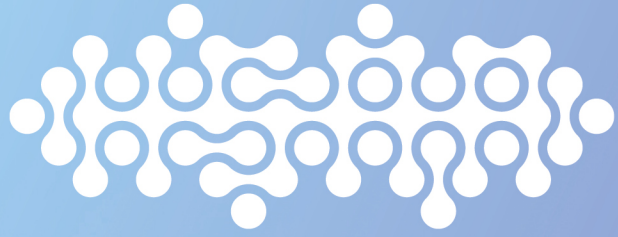
Identify the parts:

```
git clone --depth 1 https://github.com/user/repo.git
```

Discuss with your neighbor:

1. What is the **program**?
2. What is the **subcommand**?
3. What is the **flag** and what might it do?
4. What is the **argument**?

Take 2 minutes, then we'll discuss.



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

PATH and Environment Variables

5

The "Command Not Found" Problem

At some point, you **will** see this:

```
$ ruff check  
bash: ruff: command not found
```

Or on PowerShell:

```
ruff: The term 'ruff' is not recognized as the name of  
a cmdlet, function, script file, or operable program.
```

This does NOT mean the program is broken.

It usually means the shell doesn't know **where to find** the program.

What Is the PATH?

PATH = a list of folders the shell searches when you type a command

Think of it like a **phone book**: the shell looks up the command name in a list of known addresses.

```
# bash / Git Bash (separated by colons)
$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/home/user/.local/bin
```

```
# PowerShell (separated by semicolons)
PS> echo $env:PATH
C:\Windows\System32;C:\Program Files\Git\bin
```

When you type `git`, the shell checks each folder **in order** until it finds `git`.

Fixing "Command Not Found"

Step 1: Is the program installed? If not, install it.

Step 2: Is its folder in PATH?

```
which git          # bash / Git Bash
where.exe git      # PowerShell (not just "where"!)
Get-Command git    # PowerShell-native alternative
```

Note: `where` alone in PowerShell is an alias for `where-object` -- not what you want!

Step 3: Did you restart the terminal after installation?

Many installers modify PATH, but changes only apply to **new** terminal windows.

Step 4 (advanced): Add the folder to PATH manually. (Ask for help if needed.)

90% of the time, Step 3 fixes it!

Other Environment Variables

PATH is the most important, but there are others:

Variable	What It Stores
HOME	Your home directory
USER	Your username
LANG	Language setting
EDITOR	Preferred text editor (used by git)

```
echo $HOME      # bash / Git Bash
echo $env:USERPROFILE # PowerShell
```

Environment variables = **global settings** for your terminal session.



Hands-On: Explore Your PATH

Try these in your terminal:

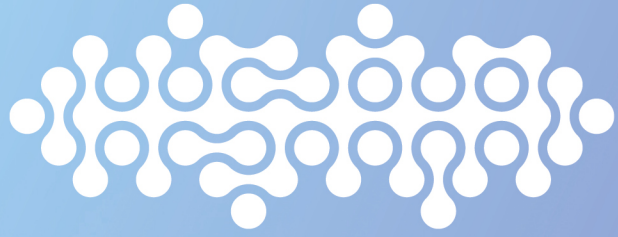
```
# See your PATH
echo $PATH

# Find where git lives
which git

# Count the directories in your PATH
# (hint: count the colons or semicolons)
```

Questions to discuss:

- How many directories does your shell search?
- Where is git installed on your machine?
- What happens if two folders in PATH both contain `git`?



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

What Is a Program?

6

Programs Are Just Files

A program is a file that contains instructions your computer can execute. **That's it.**

"Installing" a program = copying files to your hard drive + adding the location to PATH

"Running" a program = the OS reads the file and follows the instructions

Aspect	Windows	Linux / macOS
Identifies executables by	File extension (<code>.exe</code>)	File permissions
Typical program file	<code>git.exe</code>	<code>git</code> (no extension)
Making something runnable	Give it <code>.exe</code> extension	<code>chmod +x script.sh</code>

Compiled vs. Interpreted Programs

Compiled (e.g., `git`, C programs):

- Source code → machine code (once)
- Runs directly, no extra software
- Standalone `.exe` or binary

Interpreted (e.g., Python):

- Read and run line by line
- Needs the interpreter installed
- `python script.py`

This is why installing Python is required before running `.py` files.

Python is the **interpreter** that reads your scripts.

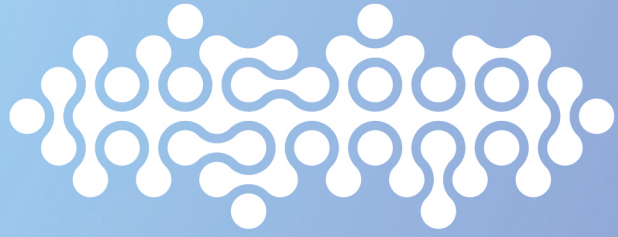
Programs vs. IDE Extensions

Why install Python AND the Python VS Code extension?

Install	Where	What It Does
Python (program)	Your computer	Actually runs <code>.py</code> files
Python (extension)	Inside VS Code	Adds syntax highlighting, autocomplete
Ruff (program)	Your computer	Checks code style in terminal
Ruff (extension)	Inside VS Code	Shows Ruff warnings as underlines

The extension calls the program -- it doesn't replace it.

The program is the **engine**. The extension is the **dashboard**.



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Automation and Scripting

7

From Single Commands to Automation

The progression:

Single command → **Chained commands** → **Shell script** → **CI/CD pipeline**

```
# 1. Single
mkdir project

# 2. Chained (&& means "if that succeeded, also do...")
mkdir project && cd project && git init

# 3. Script (save in setup.sh, run with: bash setup.sh)

# 4. CI/CD (runs automatically when you push code)
```

Each level builds on the previous. **The terminal is where automation lives.**

A Real Script Example

setup.sh -- Create and initialize a project:

```
#!/bin/bash
PROJECT_NAME="my-project"

mkdir "$PROJECT_NAME"
cd "$PROJECT_NAME"
git init
echo "# $PROJECT_NAME" > README.md
git add .
git commit -m "Initial commit"
echo "Project $PROJECT_NAME created!"
```

Run it: `bash setup.sh`

Faster, no typos, repeatable. Need 10 projects? Run it 10 times.

Automation in This Course

You will encounter all of these -- all built on terminal automation:

- **Pre-commit hooks** -- scripts that check code before every commit
- **GitHub Actions** -- automated pipelines on push
- **Pytest** -- finds and runs all your tests automatically
- **Ruff** -- checks your code style automatically

The pattern is always the same: take manual steps and automate them.



Hands-On: Write Your First Script

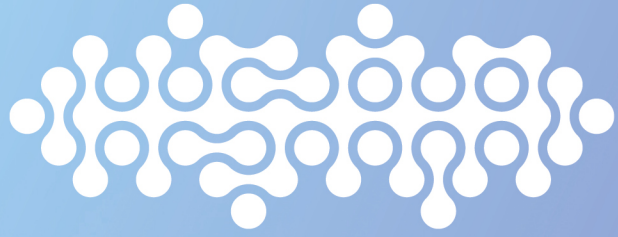
1. Create a file called `setup.sh` (use VS Code or any editor)
2. Paste the script from the previous slide
3. Open a terminal and navigate to the file
4. Run it:

```
bash setup.sh
```

5. Verify:

```
ls my-project  
cd my-project && git log
```

Take 5 minutes. Raise your hand when you see the commit!



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Quick Reference and Summary

8

Keyboard Shortcuts for the Terminal

Shortcut	What It Does
Up/Down arrows	Scroll through previous commands
Tab	Auto-complete file names and commands
Ctrl+C	Cancel the current command
Ctrl+L	Clear the screen
Ctrl+R	Search command history

Tab completion is your best friend -- start typing and press Tab!

Ctrl+C is your emergency brake -- use it when something is stuck.

Key Takeaways

1. A **terminal** is a text interface; a **shell** is the language inside it
2. Different OS = different defaults, but **core commands are the same**
3. Commands follow: `program subcommand --flag argument`
4. **PATH** tells the shell where to find programs
5. Programs are just files -- extensions are just conventions
6. **Automation** through scripting is a core professional skill

You now have the foundation to work confidently in the terminal!

Next Steps

Ready to use these skills? Go to:

- **Install tools:** [Lecture 1: Installing Git](#)
- **Set up VS Code:** [Lecture 1: VS Code Setup](#)
- **Git workflows:** [Appendix 1: Advanced Git](#)

Full lecture reference: [Appendix 0 Part 1: Terminal and CLI Foundations](#)



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Questions?

THANK YOU