Homegrown Tools & Self-Hosted Infrastructure

Weeks 1 + 2: The Shell + Scripts

ACUD MACHT NEU / School of Machines
Spring 2026

Sarah Grant

What We'll Build Over 12 Weeks

Week 1-2	Command Line	YOU ARE HERE
Week 3-4	Self-Hosted Server	
Week 5-6	Mesh Networks	
Week 7-8	Local AI	
Week 9-10	Peer-to-Peer	
Week 11-12	Integration	

01

What Is a Shell?

The Shell Is an Interface

A shell is the outermost layer between you and the operating system.

Every time you click an icon, drag a file, or open a program, your OS is executing the same operations you could type directly into a shell.

The GUI exposes what someone decided to put a button for.

The shell gives you the full vocabulary.

The Shell Loop

↻ then it loops back to the prompt

That's it. Everything you'll ever do in a terminal is this loop.

The Unix Philosophy

Small tools that do one thing well,

composed together with pipes.

```
$ ps aux | grep nginx | head -2
```

list all processes → find ones running nginx → show top 2

Activity 1: Open a Terminal

START HERE

```
$ open -a Terminal      # macOS

$ echo “Hello, I am $(whoami)”

$ echo $SHELL
```

What shell are you running?

GO FURTHER

```
$ bash --version    # or zsh --version

$ cat /etc/shells

$ echo $PS1
```

02

A Brief History of Shells

Origin of the Word "Shell"

1964 — Louis Pouzin (France) coins "shell"

Working at MIT, he developed RUNCOM — a tool for running command scripts. He imagined commands as building blocks, like a programming language.

1965 — Glenda Schroeder (MIT) builds the first shell

Implemented the Multics shell based on Pouzin's design. She also co-authored one of the earliest papers on electronic mail.

The "rc" in .bashrc and .vimrc? It stands for RUNCOM.

Shell Timeline

1971	Thompson shell	Ken Thompson, Bell Labs — first Unix shell
1977	Bourne shell (sh)	Stephen Bourne — syntax we still use today
1978	C shell (csh)	Bill Joy, UC Berkeley — history, aliases, job control
1983	KornShell (ksh)	David Korn, Bell Labs — proprietary until 2000
1989	Bash	GNU Project — free, became the default everywhere
1990	Zsh	Paul Falstad — superset of bash, now default on Mac
2005	fish	Friendly Interactive Shell — broke compatibility

Two Shell Families

THE BOURNE FAMILY

`sh → bash → zsh → ksh`

Share common scripting syntax

if/then/fi, for/do/done

Largely compatible with each other

This is the lineage we'll work in.

THE C FAMILY

`csh → tcsh`

C-like syntax (Bill Joy, Berkeley)

Introduced history (!!) and aliases

Scripting was unreliable

Mostly historical at this point.

The Outliers

fish (2005)

Designed for humans in 2005, not backward compatibility. Syntax highlighting, autosuggestions, cleaner scripting. But scripts don't transfer to bash/zsh. A walled garden.

PowerShell (Windows)

Philosophically different: pipes objects with properties, not text strings. Powerful for Windows admin but incompatible with the Unix world.

What's on your machine right now?

macOS	zsh (since Catalina 2019, was bash)
Linux	bash (most distros), some use zsh
Windows	cmd.exe / PowerShell, or bash via WSL

Why Apple Switched From Bash to Zsh

Apple used bash as the default shell from 2003 to 2019.

In 2007, bash moved to the GPLv3 license — which requires derivative works to remain open source. Apple didn't want that obligation.

So Apple froze bash at version 3.2 (from 2007!) for over a decade.

In 2019, they switched to zsh (MIT license — no copyleft).

Every time you open Terminal on a Mac, you're living with the consequences of a software licensing decision.

03

Navigating the File System

The File System Is a Tree

Everything on your computer lives in a hierarchy of folders.

```
/ (root)
├── home/
|   └── sarah/
|       ├── Documents/
|       ├── Desktop/
|       └── .bashrc
├── etc/          ← system configuration
├── usr/          ← installed programs
├── tmp/          ← temporary files
└── var/          ← logs, databases
```

Navigation Commands

```
# Where am I?
$ pwd
/Users/sarah/Documents/_teaching/schoolOfMa/afterhours/publish/homegrown-tools-self-hosted-infrastructures/week1-2

# What's here?
$ ls
afterhours-cli-cheatsheet.pdf  cli/  scripts/

# Show everything, including hidden files
$ ls -la

# Move around
$ cd scripts       # go into a folder
$ cd ..            # go up one level
$ cd ~             # go home
```

Activity 2: Explore Your File System

START HERE

```
$ pwd
$ ls
$ cd <folder>
$ cd ..
$ cd ~
$ ls -la
```

GO FURTHER

```
$ ls -la /etc
$ cd -
$ ls -la /etc | head -20
```

04

Making & Moving Things

File Operations

```
# Create
$ mkdir project                                  # make a folder
$ mkdir -p project/dev/scripts                   # make nested folders
$ touch notes.txt                                # create an empty file

# Copy and move
$ cp notes.txt readme.txt                        # copy a file
$ cp -r project/ archive/                        # copy a whole folder
$ mv readme.txt archive/                         # move a file
$ mv archive/readme.txt archive/readme-old.txt   # rename a file

# Delete (careful! no trash can!)
$ rm archive/readme-old.txt                      # delete a file
$ rm -r archive/                                 # delete a folder
```

Reading Files

```
# Print entire file
$ cat notes.txt

# Scroll through a long file (q to quit)
$ less notes.txt

# Just the beginning or end
$ head -n 10 notes.txt       # first 10 lines
$ tail -n 10 notes.txt       # last 10 lines

# Count things
$ wc -l notes.txt            # count lines
$ wc -w notes.txt            # count words
```

Activity 3: Build a Project Structure

START HERE

```
$ cd project

$ touch today.txt

$ echo "Things I learned today" >> today.txt

$ cat today.txt
```

05

Permissions as Power Structure

Who Can Do What?

Every file on your computer has an owner, a group, and a set of permissions.

These same permission structures exist at the scale of servers, networks, and the internet.

```
-rwxr-xr--  1  sarah  staff  hello.sh

owner: rwx              group: r-x            others: r--

r = read    w = write    x = execute
```

Permission Commands

```
# Make a file executable (so you can run it)
$ chmod +x script.sh

# Who am I?
$ whoami
sarah

# Run something as the superuser (root)
$ sudo apt install htop

# sudo = "superuser do"
# root has full control of the machine
# Regular users have limited access
# This is the same model as server administration
```

Activity 4: Examining Permissions

START HERE

```
$ ls -la

$ whoami

$ touch /etc/test

$ sudo touch /etc/test
```

GO FURTHER

```
$ find /usr/bin -type f “ head -20

$ ls -la /etc/passwd

$ groups $(whoami)

$ ls /etc/init.d
```

06

Installing Programs

Package Managers

Software you install comes from repositories — servers maintained by someone who has decided what's trustworthy.

Linux (Debian/Ubuntu)

apt

```
sudo apt install htop
```

macOS

Homebrew

```
brew install htop
```

Windows

winget

```
winget install htop
```

What Happens When You Install?

1 **CHECK**

Package manager queries a repository (a server hosting software)

2 **DOWNLOAD**

Fetches a compressed package (binaries, config, docs, dependencies)

3 **PLACE**

Puts files in the right locations (/usr/bin, /etc, /usr/share/man)

4 **RECORD**

Logs what was installed so it can update or remove cleanly

Where Does Your Software Come From?

Who maintains the repository? Who decides what's available? Who do you trust?

When Ubuntu ships default packages, Canonical has made choices about trust.

When Apple restricts what you install outside the App Store, that's control.

When you compile from source, you opt out — but take responsibility.

In Weeks 3-4, you'll run your own server with your own applications. You'll go from apt install to being the one who serves the software.

Activity 5: Install Your First Package

START HERE

1. Update your package list: sudo apt update (or brew update)
2. Install a program: sudo apt install tree (or brew install tree)
3. Run it: tree afterhours/ (from Activity 3!)
4. Try: which tree — where did it get installed?

GO FURTHER

1. Install htop and run it — a live system monitor (q to quit)
2. Check what's installed: apt list --installed | wc -l
3. Read a man page: man ls — then try options you didn't know
4. Install cowsay and try: echo "hello" | cowsay

07

Your First Script

Writing a Script

```
# 1. Create the file
$ nano hello.sh

# 2. Write this inside:
#!/bin/bash

echo "Hello, I am $(whoami)"
echo "Today is $(date)"
echo "I'm in $(pwd)"
echo "Files here:"
ls -la

# 3. Save (Ctrl+O) and exit (Ctrl+X)
```

Running a Script

```
# Make it executable
$ chmod +x hello.sh

# Run it
$ ./hello.sh
Hello, I am sarah
Today is Sat Feb  7 14:30:00 CET 2026
I'm in /home/sarah/afterhours/week-01
Files here:
total 8
-rwxr-xr-x  1 sarah sarah  128 Feb 7 hello.sh
drwxr-xr-x  2 sarah sarah 4096 Feb 7 notes/
```

What Just Happened?

`#!/bin/bash`

The "shebang" — tells the system which shell to use

`chmod +x`

Changed the permission so the file can be executed

`$(command)`

Command substitution — runs a command and inserts its output

This is automation and where infrastructure maintenance begins.

Activity 6: Write a Script

START HERE

1. cd afterhours/week-01 (your project from earlier!)
2. nano hello.sh — type the script from the slide
3. chmod +x hello.sh
4. ./hello.sh — watch it run!
5. Add a line: echo "My shell is $SHELL"

GO FURTHER

1. Add a line that counts files: echo "$(find . -type f | wc -l) files"
2. Make it take your name: read -p "Name: " name; echo "Hi $name"
3. Save the output: ./hello.sh > report.txt && cat report.txt
4. Add the date to the filename: ./hello.sh > "report-$(date +%F).txt"

→ Builds on everything: navigation, files, permissions, and commands — all woven into one script you wrote.

Weeks 1-2 Recap

What we did these first 2 weeks…

- **Opened a terminal** — and found your shell
- **Learned the history** — from Pouzin & Schroeder to your machine
- **Navigated the file system** — pwd, ls, cd
- **Created and organized files** — mkdir, touch, cp, mv, rm
- **Examined permissions** — the power structure of your computer
- **Installed software** — via package managers
- **Wrote your first script** — and made it executable

For Next Week

Read:

"The Art of Command Line" — github.com/jlevy/the-art-of-command-line

Practice:

Try to do everyday tasks without the GUI — organizing files, finding things, reading logs. Get comfortable being uncomfortable.

Explore:

What is your computer doing when you're not looking? What processes run? What files exist that you didn't create? Poke around.

Next week: text processing, pipes, and writing scripts that automate real tasks.

$ exit

teaching@chootka.com

Office hours: Mon + Fri 10:00–12:00