

Help! My Computer has a Terminal

Authored: Alex Villa -- 2025

Purpose

The purpose of this document is to highlight critical aspects and concepts of using, developing, and working with Unix-based terminals (not Windows Command Prompt!). It serves as a high-level introduction and overview of key components rather than an in-depth tutorial on specific commands and pipelines. The additional sections toward the bottom cover related topics that, while not exclusively terminal-focused, are essential for effective computational work.

The **overarching goal** is to turn from **Help! My Computer has a Terminal** to **Help! My Terminal has a Computer**.

-Alex

Table of Contents

- ◆ Shell
- ◆ Free Software
 - ◆ Free and Open Source Software
- ◆ Utilities
- ◆ SSH
- ◆ Editors (Emacs)
- ◆ Version Control
 - *Most importantly it allows for Accountability*
 - ◆ **History Examination and Navigation**
 - ◆ **Commit Creation**
- ◆ Communication
- ◆ History
 - ◆ History Rewriting
- ◆ Python
- ◆ Toolchain
- ◆ Building
- ◆ Documentation
- ◆ High Performance Computing
- ◆ Floating Point






- ◆ Real Programming Languages
 - There are so much more R.P.L properties and guidelines, that I can not place all of them in here, but they are all still extremely relevant!
- ◆ Library Discipline
- ◆ Operating Systems
- ◆ Software Engineering
- ◆ Extra Topics
 - ◆ Scientific Computing
 - ◆ Machine Learning
 - ◆ AI
- ◆ Acknowledgments
- ◆ References



Shell

"GUI makes easy tasks easy, while CLI makes difficult tasks possible."

"In the beginning..... was the command line." - Neal Stephenson

Learn to use the debugger, so you can ask more questions without recompiling.

- ◆ File Systems 
 - ◆ Navigation 
 - ◆ `pwd` : show where you are
 - ◆ `cd` : change directory (absolute vs. relative paths)
 - ◆ `ls` : list contents, with flags (-l, -a, -h)
 - ◆ Home 
 - ◆ Meaning of ~ (your personal home directory)
 - ◆ The “root” of the entire tree (/) vs. “root user”
 - ◆ Quick jumps: `cd ~`, `cd /`, `cd -` (previous dir)
 - ◆ Root 
 - ◆ Permissions 
 - ◆ Permission Model
 - ◆ Owner / Group / Others
 - ◆ Read (r), write (w), execute (x) bits

- ◆ Viewing Permissions
- ◆ `ls -l` output breakdown
- ◆ Special flags: `setuid`, `setgid`, sticky bit (`s / t`)
- ◆ Changing Permissions & Ownership
- ◆ `chmod` symbolic vs. numeric modes (e.g. `u+rx`, `644`)
- ◆ `chown user:group file`
- ◆ When and why to use `sudo` 
 - ◆ Hint: `substitute user, do`
- ◆ Special Files
- ◆ Redirection/Pipes 
 - ◆ **Redirect stdout** to a file (`cmd > out.txt`), **append** (`>>`), or **redirect stdin** from a file (`cmd < in.txt`).
 - ◆ **Pipe** one command's output into another's `stdin`:
 - ◆ `<()`
 - ◆ `less` - A pager for viewing (and searching) long text streams without loading everything at once.
 - ◆ `$()` - (Command Substitutions): -Capture the output of a command and substitute it into another command or assignment.
 - ◆ Completion - - - Hit **Tab** to auto-complete filenames, commands, or even variable names. Speeds up typing and helps discover options.
 - ◆ `history`
 - ◆ `history` shows recent commands; use `!123` to rerun line 123, or `!!` for the last command.
 - ◆ E.g. `!grep` reruns the last command that started with "grep."
 - ◆ `pushd`
 - ◆ Maintain a directory stack: `pushd` dir changes into dir and pushes old cwd; `popd` returns to the previous.
 - ◆ Handy when jumping back and forth between projects.
 - ◆ Job Control
 - ◆ Run jobs in the background (`cmd &`), list them (jobs), bring one to foreground (fg %1) or background (bg %1).
 - ◆ PATH/type
 - ◆ `PATH` is the list of dirs the shell searches for executables.
 - ◆ type `cmd` tells you if cmd is a builtin, function, alias, or external program and where it lives.
 - ◆ `help`
 - ◆ Shell-builtin help: `help ls`
 - ◆ Env(vars)

- ◆ `env` lists all env vars; `export NAME=value` sets one for child processes.
- ◆ `export EDITOR=emacs`
- ◆ `$?` - Exit Status
 - ◆ After a command runs, `$?` holds its return code (0 = success, non-zero = error).
 - ◆ `grep foo file.txt; echo "Status was $?"`
- ◆ `time`
 - ◆ Measures how long a command takes (real/user/sys).
 - ◆ `time tar czf archive.tar.gz bigfolder/`

Free Software

Free Software

"Free software is a matter of liberty, not price. To understand the concept, you should think of 'free' as in 'free speech,' not as in 'free beer.'" - Richard Stallman

"The power of Open Source is the power of the people. The people rule." - Philippe Kahn

"Proprietary software keeps users divided and helpless." - Richard Stallman

https://en.wikipedia.org/wiki/Free_software 

<https://www.gnu.org/philosophy/free-sw.en.html> 

<https://www.gnu.org/philosophy/free-sw.en.html> 

<https://www.fsf.org/> 

<https://github.com/johnjago/awesome-free-software?tab=readme-ov-file> 

- ◆ As defined by the GNU Project, emphasizes user freedom rather than price. It means users have the freedom to run, copy, distribute, study, and modify the software.
 - ◆ Four Essential Freedoms:
 - ◆ **Source Code Access**
 - ◆ **Not about price**
 - ◆ **Ethical implications**
 - ◆ **Distinction from freeware**
 - ◆ Freeware, while often free to use, may have restrictions on modification, distribution, or other freedoms, making it distinct from free software.

Free and Open Source Software

Free and open-source software (FOSS) refers to software that is both free of cost and whose source code is publicly available for anyone to use, modify, and distribute. Free **Software + Open Source Software = Free and Open Source Software (FOSS)**

Utilities

- ◆ **man** -> The built-in manual-page browser for virtually every Unix command.
- ◆ **cut** -> Extracts columns or fields from each line of input.
 - ◆ **-d** 'DELIM' specify a delimiter (default is TAB)
 - ◆ **-f** LIST select fields by number or range
 - ◆ **-c** LIST select character positions
 - ◆ Example from ChatGPT

```
# Given "name,age,city" CSV, pull just the age:
```

```
echo "Alice,30,Denver" | cut -d',' -f2
```

```
# → 30
```

- ◆ **sort** -> Orders lines of text.
- ◆ **grep** -> Filters lines matching a pattern.
 - ◆ **-i** case-insensitive
 - ◆ **-E** extended regex
 - ◆ **-v** invert match (show non-matches)
 - ◆ **-r** recursive through directories
 - ◆ **-n** show line numbers
- ◆ **diff** -> Compares two files (or directories) line by line.
- ◆ **tar** -> Archives (and optionally compresses) collections of files into a single file.

SSH

SSH 

Securely log into a remote machine or run commands over an encrypted channel.

- ◆ [scp](https://en.wikipedia.org/wiki/Secure_copy_protocol  -> Securely copy files to/from a remote host over SSH)

```
scp localfile.txt user@host:/remote/path/  
scp user@host:/remote/file .
```


- ◆ `~^z` -> - **Suspend/Resume Jobs (Ctrl+Z, fg, bg)**
 - ◆ Temporarily pause a foreground job and resume it in the foreground or background.
 - ◆ **Ctrl+Z**: suspend the current job (puts it in “Stopped” state).
 - ◆ **jobs**: list stopped/background jobs.
 - ◆ **fg %n**: resume job number n in foreground.
 - ◆ **bg %n**: resume job number n in background.

```
ping google.com    # press Ctrl+Z to suspend  
jobs               # see “[1]+  Stopped”  
bg %1              # resume ping in background
```

- ◆ **screen** / **tmux** -> - Terminal multiplexers that let you run multiple shell sessions within a single window, detach/reattach, and manage panes.
 - ◆ Start a session: `screen -S name` or `tmux new -s name`
- ◆ **X11** -> - Run GUI applications on a remote host and display them locally over SSH.
- ◆ Config of forward: Configuring SSH Forwarding in `~/.ssh/config`

Editors (Emacs)

Use Emacs. (M-/ is enough of a reason.) - Davis

- ◆ Emacs and Vim can do rectangles just as well as your current editor.
- ◆ Nano is a great beginner editor: <https://www.nano-editor.org/> . Does it quick and simple
- ◆ Vim and Emacs allow for more control and features.

Choose one editor and learn one editor well.

Version Control

Version Control 

Version control lets even one person understand their own work.

Version Control is a system that tracks changes for files over time, allowing one to see what changed, when, and who made the changes.

Most importantly it allows for Accountability

History Examination and Navigation

- ◆ **git checkout <commit|branch>**

Switches your working directory to a specific commit or branch.

- ◆ Example: `git checkout feature/login` moves to the feature/login branch.

- ◆ **git log**

Shows the commit history in reverse chronological order.

- ◆ Common flags: `--oneline`, `--graph`, `--decorate`.

- ◆ **git diff [<commit> [<commit>]]**

Displays changes between the working directory, staging area, and/or commits.

- ◆ Example: `git diff HEAD~1 HEAD` compares the last two commits.
- ◆ **gitk**. Opens a graphical history viewer for commits, branches, and merges.
- ◆ Launch with: `gitk --all` to see every branch.

Commit Creation

- ◆ **git add <file|dir>**

Stages changes (new, modified, or deleted files) for the next commit.

- ◆ **git reset [--mixed|--soft|--hard] <commit>**

Unstages or rewinds commits:

- ◆ `--mixed` (default) resets index but keeps working files.
- ◆ `--soft` moves HEAD but preserves index and working files.
- ◆ `--hard` resets everything to the specified commit.

- ◆ **git commit -m "message"**

Records staged changes with a descriptive message.

- ◆ **git commit -- <path>**

Creates a commit only for changes in the specified path(s), ignoring other staged changes.

Communication

- ◆ **git fetch [<remote>]**

Downloads commits, tags, and refs from the remote without merging.

- ◆ **git pull [<remote> [<branch>]]**

Equivalent to git fetch followed by git merge.

Note: avoid pull for opening pull requests; use it only to update local branches.

- ◆ `git push [<remote> `` <branch>]`

Uploads your local commits to the remote repository's branch.

History

History Rewriting

"With Great Power Comes Great Responsibility".

History Rewriting in version control should not be done constantly. History is there to show and highlight your mistakes on purpose! It's to help revise the next version to be better than the last!

- ◆ `commit --amend` : Modifies the last commit (change message or add forgotten files) (A personal favorite)
- ◆ `rebase -i` : Interactive rebase to edit, reorder, squash, or delete multiple commits (Another personal favorite)

Python




Yes you can do that in Python.

- ◆ `-i` for `REPL` [↗](#) : Interactive mode for executing Python commands line-by-line
- ◆ `subprocess` [↗](#) : module for spawning processes, executing shell commands, and handling input/output pipes
- ◆ `NumPy` [↗](#) - A crucial library for numerical computing with efficient n-dimensional arrays(tensors) and mathematical functions. Lacks `XPU` support, look into `CuPyNumeric` [↗](#) for a GPU numerical computing library!
- ◆ `Pandas` [↗](#) - Data manipulation library providing DataFrames for structured data analysis and processing
- ◆ `Matplotlib` [↗](#) - Plotting library for creating static, animated, and interactive visualizations
- ◆ `pdf` [↗](#) - Libraries (PyPDF2, reportlab) for reading, writing, and manipulating PDF documents programmatically









Toolchain









Collection of software development tools that work together to transform source code into executable programs. For the most part it can includes a compiler, linker, assembler

, and some utilities for debugging, profiling, etc.

- ◆ **Compiler** : translates high-level source code (C++, Java, etc.) into lower-level code. It performs optimizations and error checking
- ◆ **Assembler** : Converts human-readable assembly language into machine code (binary instructions) that the processor can directly execute
- ◆ **Linker** : Combines multiple object files and libraries into a single executable, resolving symbol references and memory addresses between different code modules
- ◆ **Debugging tools**: Software that helps developers find and fix bugs by allowing them to pause execution, examine variable values, step through code line-by-line, and analyze program behavior.

The typical flow: source code → compiler → assembly → assembler → object files → linker → executable program, with debugging tools available throughout the process.



- ◆ **nm / objdump**: binary analysis tools for examining symbols, sections, and assembly code in compiled executables and object files
- ◆ **Shared Libraries** : Dynamically linked code modules (.so/.dll) that multiple programs can use simultaneously, reducing memory usage and enabling code reuse
- ◆ **"unit " test systems**  - Framework to write and run automated tests for individual components (functions, classes, methods) in isolation. Can provides tools to make assertions about expected behavior, organize tests into suites, and report pass/fail results.
Test Early and Test Often
 - ◆ **CPPUnit** : C++ testing framework for creating automated unit tests with assertions and test fixtures
 - ◆ **GoogleTest** : Google's C++ testing framework with rich assertion macros, test discovery, and parameterized tests
- ◆ **Debuggers** 
 - ◆ **TotalView**  (For MPI): Specialized debugger for parallel programs running across multiple processes/nodes in MPI environments.
 - ◆ **GDB** /**LLDB** : Command-line debuggers for setting breakpoints, examining variables, and stepping through code execution.

- ◆ **Strace** : System call tracer that monitors and logs all system calls made by a process for debugging I/O and system interactions.
- ◆ **Sanitizers** : Runtime error detection tool that instruments code during compilation to catch bugs like memory corruption, undefined behavior, and threading issues while the program runs.
 - ◆ UndefinedBehavior: **UndefinedBehavior** -- detects undefined behavior like integer overflow, null pointer dereference, and uninitialized variables at runtime.
 - ◆ Address: Finds memory corruption bugs including buffer overflows, use-after-free, and heap corruption.
 - ◆ Memory: Detects uninitialized memory reads and provides more detailed memory error reporting than AddressSanitizer.
 - ◆ Leak: Identifies memory leaks by tracking allocated memory that's never freed
 - ◆ Thread: Detects data races and threading bugs in multithreaded programs.
 - ◆ **Valgrind** : A Comprehensive memory debugging tool that detects memory leaks, corruption, and threading errors through dynamic analysis.
- ◆ **gcov** : Code coverage tool that measures which lines of code are executed during testing to identify untested code paths.
- ◆ **Profilers**
 - ◆ **Gprof** : GNU profiler that analyzes function call frequency and execution time to identify performance bottlenecks.
 - ◆ **perf** : Linux performance analysis tool for CPU profiling, cache misses, and hardware event monitoring.
 - ◆ **nvprof** : NVIDIA profiler for analyzing GPU kernel performance and memory usage in CUDA applications.
 - ◆ **I/O** : Profilers that monitor file system operations, disk usage, and network activity to optimize data access patterns.

Building

"The build that works on my machine is not the build that matters."

Building is the process of transforming source code into executable software.

- ◆ **Make** : Build automation tool that uses Makefiles to define dependencies and compilation rules for building projects.
- ◆ **CMake** : Cross-platform build system generator that creates native build files (Makefiles, Visual Studio projects) from high-level configuration.

- ◆ [Spack](#): Package manager for HPC environments that builds and installs software with different versions, compilers, and dependencies.
- ◆ [Environment Modules](#) (No module unload/purge): System for dynamically modifying user environment variables to manage different software versions and dependencies without conflicts.

Documentation

Written material that explains how software works, how to use it, and how it's structured.

"Good code is its own best documentation." - Steve McConnell

"Code tells you how; Comments tell you why." - Jeff Atwood

"Any fool can write code that a computer can understand. Good programmers write code that humans can understand." - Martin Fowler

- ◆ [Doxygen](#): Documentation generator that extracts comments from source code to create API documentation in HTML, PDF, or other formats. (Extremely useful for when code is well documented in the code!)
- ◆ [Sphinx](#): Python-based documentation tool that converts reStructuredText markup into professional documentation websites and PDFs.
- ◆ [LaTeX](#): Typesetting system for creating high-quality documents, papers, and books with precise formatting and mathematical notation.
- ◆ **Markdown**
 - ◆ [Pandoc](#): Universal document converter that transforms **Markdown**, **LaTeX**, **HTML**, and other formats between each other.
 - ◆ [GitLab Wiki](#): Web-based wiki system integrated with **GitLab** repositories for collaborative documentation using Markdown
 - ◆ Also can be done for [Github](#):
- ◆ [InkScape](#): Vector graphics editor for creating scalable diagrams, illustrations, and technical drawings
- ◆ **Dot**: Graph description language and layout engine from **Graphviz** for generating network diagrams and flowcharts from text descriptions
- ◆ [Gnuplot](#): Command-line plotting utility for creating 2D and 3D graphs from data files or mathematical functions

High Performance Computing





Scalability is about doing something in a bigger way. Efficiency is about doing something in a better way.

- ◆ **SLURM**: Workload manager and job scheduler for allocating compute resources and managing job queues on HPC clusters
- ◆ give/take (if active): Resource sharing mechanism that allows jobs to dynamically borrow unused resources from other allocations
- ◆ **MPI**: Message Passing Interface standard for parallel programming across distributed memory systems
- ◆ **OpenMP**: API for shared-memory parallel programming using compiler directives and threading
- ◆ **PBS**: Portable Batch System for job scheduling and resource management on compute clusters

Floating Point

The Cauchy-Schwartz inequality is violated by floating-point dot products.

"The condition of equality of two floating point values is not only unnecessary, but usually incorrect."

- ◆ $[0,1]$: Interval notation representing real numbers between 0 and 1, highlighting precision.
- ◆ non-associativity: A **Critical** property where $(a + b) + c \neq a + (b + c)$ due to rounding errors in floating-point arithmetic!
- ◆ **What every computer scientist should know about floating-point arithmetic** 
- ◆ **IEEE 754** : International standard defining formats and operations for binary floating-point arithmetic used by most processors.
- ◆ **Machine Epsilon** : Smallest floating-point number that when added to 1.0 gives a result different from 1.0, measuring precision limits.
- ◆ **Catastrophic Cancellation** : Severe loss of precision when subtracting two nearly equal floating-point numbers.

Real Programming Languages

"Simplicity is prerequisite for reliability." - Edsger Dijkstra

"There are only two kinds of languages: the ones people complain about and the ones nobody uses."- Bjarne Stroustrup





- ◆ **Conservation of Difficulty:** Principle that complexity cannot be eliminated, only moved around - making one part simpler often makes another part more complex.
- ◆ **Principle of Least Surprise:** Language constructs should behave in ways that are intuitive and predictable to programmers familiar with the domain.
- ◆ **Orthogonality:** Language features should be independent and composable without unexpected interactions or special cases.
- ◆ **Fail Fast:** Errors should be detected and reported as early as possible, preferably at compile time rather than runtime.
- ◆ **Locality of Reference:** Related code and data should be kept close together both spatially and temporally.
- ◆ **Progressive Disclosure:** Simple things should be simple, but complex things should still be possible.
- ◆ **Zero-Cost Abstraction:** High-level language features shouldn't impose runtime overhead compared to hand-written low-level code.
- ◆ **Backward Compatibility:** maintaining support for legacy code while evolving language features, creating complexity and design compromises.
- ◆ **Performance Trade-offs:** Tension between abstraction levels and execution efficiency, where higher-level features often incur runtime costs.
- ◆ **Type Systems:** compile-time constraints that catch errors early but require explicit declarations and can limit expressiveness.

There are so much more R.P.L properties and guidelines, that I can not place all of them in here, but they are all still extremely relevant!

Library Discipline






Library Discipline is about designing libraries that are **predictable, safe, and composable**. *Remember* we are trying to avoid hidden global state, side effects, and dependencies that make code unreliable in larger systems or concurrent environments.

The goal for libraries is to "play well with others" and don't surprise users with unexpected behavior.

- ◆ **Global State**  (impedes even very weak thread safety): Shared mutable state that makes libraries unsafe for concurrent use and creates hidden dependencies between function calls.
- ◆ **Process Globals**: *A piece of state that affects the entire process and is shared by all threads, libraries, and components within that process.*
 - ◆ **cwd** : current working directory that affects file operations globally and can cause unexpected behavior when changed.
 - ◆ **Environment** : Environment variables that modify program behavior invisibly and create hidden coupling between components.
 - ◆ **Standard Streams** : **stdin** / **stdout** / **stderr** that are shared across the entire process and can interfere with each other.
 - ◆ **Bad Examples**
 - ◆ **matplotlib** : Too heavy use of global state for plot configuration that makes concurrent plotting difficult and unpredictable.
 - ◆ **subprocess** : Modifies process-wide state like signal handlers and can interfere with other parts of the application
- ◆ **The meaning of version numbers**: Semantic versioning (major.minor.patch) where changes indicate compatibility guarantees and breaking changes.
 - ◆ https://en.wikipedia.org/wiki/Software_versioning 
 - ◆ <https://semver.org/> 
- ◆ **Deprecations** : Gradual removal of features through warnings and staged elimination, allowing users time to migrate without breaking existing code.

Operating Systems

"An operating system is a collection of things that don't fit into a language. There shouldn't be one." – Dan Ingalls

- ◆ **Process vs Thread** : A Processes have separate memory spaces providing isolation, while threads share memory within a process for lighter-weight concurrency.
- ◆ **Virtual Memory** : Abstraction that gives each process the illusion of having dedicated memory while actually sharing physical RAM through paging.
- ◆ **Race Conditions** : Timing-dependent bugs where program correctness depends on unpredictable execution order of concurrent operations.
- ◆ **Deadlock** : A situation where processes wait indefinitely for resources held by each other, requiring careful resource ordering to prevent.
- ◆ **System Calls** : A controlled interface for user programs to request privileged operations from the kernel, crossing the user-kernel boundary.

- ◆ **Scheduling Trade-offs** ⚡: Balance between throughput, responsiveness, and fairness when deciding which processes get CPU time.

Software Engineering

"If debugging is the process of removing software bugs, then programming must be the process of putting them in." - Edsger Dijkstra

- ◆ **Technical Debt** ⚡: Accumulated shortcuts and compromises in code that slow future development, requiring eventual refactoring investment.
- ◆ **Separation of Concerns** ⚡: A design principle that each module should handle one responsibility, reducing coupling and improving maintainability.
- ◆ **Code Smells** ⚡ - symptoms of deeper design problems like long functions, duplicate code, or inappropriate intimacy between classes.
- ◆ **Premature Optimization** ⚡: Focusing on performance before understanding actual bottlenecks, often sacrificing readability for negligible gains.
 - ◆ "Premature optimization is the root of all evil." - Donald Knuth
- ◆ **YAGNI (You Aren't Gonna Need It)** ⚡: Principle of not implementing features until they're actually required, avoiding speculative complexity.
- ◆ **Conway's Law** ⚡: Organizational structure inevitably shapes software architecture, as teams build systems reflecting their communication patterns.



Extra Topics

Scientific Computing

- ◆ **Numerical Stability** ⚡: Property of algorithms that small input changes produce small output changes, avoiding catastrophic error amplification
- ◆ **Condition Number** ⚡: Measure of how sensitive a problem is to small changes in input data, indicating computational difficulty
- ◆ **Discretization Error** ⚡: Approximation error introduced when converting continuous mathematical problems into discrete computational forms
- ◆ **Iterative Solvers** ⚡: Algorithms that approximate solutions through repeated refinement, used when direct methods are impractical
- ◆ **Vectorization** ⚡: Technique of applying operations to entire arrays simultaneously rather than element-by-element loops for performance

- ◆ **Computational Complexity** 🔗: Analysis of algorithm resource requirements (time, memory) as problem size scales, determining feasibility

Machine Learning

- ◆ **Overfitting** 🔗 - model memorizes training data instead of learning generalizable patterns, performing poorly on new data.
- ◆ **Bias-Variance Tradeoff** 🔗 - fundamental tension between model simplicity (high bias) and complexity (high variance) in prediction accuracy.
- ◆ **Feature Engineering** 🔗 - process of selecting, transforming, and creating input variables that best represent the underlying problem.
- ◆ **Cross-Validation** 🔗 - technique for evaluating model performance by splitting data into training and testing subsets multiple times.
- ◆ **Gradient Descent** 🔗 - optimization algorithm that iteratively adjusts model parameters by following the steepest descent of the loss function.
- ◆ **Supervised vs Unsupervised Learning** 🔗 - distinction between learning from labeled examples versus discovering patterns in unlabeled data.

AI

- ◆ **Transformer Architecture** 🔗 - attention-based neural network model that processes sequences in parallel, forming the foundation of modern language models.
- ◆ **Large Language Models (LLMs)** 🔗 - neural networks trained on vast text corpora to generate human-like text and perform various language tasks.
- ◆ **Training vs Inference** 🔗: Distinction between the computationally expensive process of teaching models and the relatively cheap process of using them.
- ◆ **Hallucination** 🔗 - When AI models generate plausible-sounding but factually incorrect or nonsensical information.
- ◆ **Prompt Engineering** 🔗: Craft of designing input text to elicit desired behaviors and outputs from language models.
- ◆ **Fine-tuning** 🔗: Process of adapting pre-trained models to specific tasks or domains with additional targeted training.

Acknowledgments

This document draws significant inspiration from S. Davis Herring (herring@lanl.gov 🔗), whose original terminal workshop have been invaluable in shaping this overview.

References

Software Carpentry

Software Carpentry with Python

ProGit

Git from the bottom up

Version control with GIT

Software Carpentries Version Control

Unix Shell

GNU Project

Wikipedia