

way of the hax0r
n00b to console punk

Chapter 0

Introduction

Welcome to *way of the hax0r*. This is a series of books about understanding technology well enough to do interesting, useful, and occasionally ridiculous things with it. We are starting with *n00b to console punk*. You bring the curiosity; we will build the skills.

There is a bunch of technology to learn: programming languages, operating systems, networks, electronics, databases, and more tools than anyone can keep track of. You do not need to learn it all at once. Our first stop is the console, where you can give a computer commands, see what happened, and start connecting tools to solve your own problems. Getting comfortable there will make everything that follows easier to explore.

What Is Hacking?

Richard Stallman gives us a good starting point in *On Hacking*: [1]

hacking means exploring the limits of what is possible, in a spirit of playful cleverness.

That is the meaning we will use throughout this series. You learn how something works, ask what else it could do, and try an experiment. Sometimes you solve a practical problem. Sometimes you make something wonderfully unnecessary just to see whether you can.

Suppose you turn an old laptop into a scoreboard for a game you invented. Or connect a few small programs so that an hour of repetitive work takes a few seconds. Or make a printer play a tune because you noticed that its moving parts make different sounds. The interesting part is the idea, the understanding behind it, and the cleverness of making it work.

A **hacker** enjoys that process. The *Jargon File*, a collection of hacker vocabulary and culture, includes this definition: [2]

One who enjoys the intellectual challenge of creatively overcoming or circumventing limitations.

Notice that neither definition requires you to know everything already. You need curiosity, a willingness to experiment, and enough persistence to work through the parts you do not understand yet. Expertise grows out of doing that repeatedly.

This Started Before the Internet You Know

In June 1959, Peter Samson wrote *An Abridged Dictionary of the TMRC Language* for MIT's Tech Model Railroad Club. It already included entries for **hack** and **hacker**. The dictionary was deliberately funny; in his later annotations, Samson describes a hack as an unconventional application of technology and recalls making a room-sized computer play music. [3]

The vocabulary was forming around model railroads, electronics, and early computers, long before today's websites and apps. The pleasure of getting a machine to do something unexpected was already there.

In 1962, Steve Russell and collaborators including Peter Samson, Dan Edwards, and Martin Graetz brought *Spacewar!* to life on a PDP-1 computer at MIT. They made a game in which players controlled spaceships on a computer display. The Computer History Museum preserves the program and its history. [4] It is a useful picture of the spirit we are after: learn the machine, explore its possibilities, and do something that people do not realize is possible.

Stallman recalls a broader hacking community developing at MIT and other universities in the 1960s and 1970s, with activities ranging from software to elaborate practical jokes. [1] Today's hackathons carry that same spirit of experimenting and building together.

What About Black Hats and White Hats?

You have probably heard **hacker** used to mean someone who breaks into computer systems. That is a common use of the word, especially in news stories and computer security. In that context, **black-hat hackers** use their skills for malicious purposes, such as stealing information or disrupting systems. **White-hat hackers**, also called ethical hackers, work with authorization to find weaknesses and help fix them. [5]

Those labels describe security work and the intentions and permission behind it. They do not describe the whole world of hacking, and they are not part of this series. A person building a compiler, modifying a game, or making an unusual musical instrument can be a hacker without doing security research at all. Security researchers can also share the curiosity and ingenuity we are talking about. The meanings overlap, but they are not interchangeable.

Stallman uses **cracking** for security breaking to distinguish it from the broader activity of hacking. [1] You will encounter both vocabularies. In this series, *hacking* means playful, informed exploration, and our projects are about building, understanding, and experimenting.

Wait, What Is 1337?

1337 is pronounced *leet*, a shortened form of *elite*. It is written in **leetspeak**, which replaces letters with numbers or other characters that resemble them. Read 1 as l, 3 as e, and 7 as t: 1337 becomes leet. You will also see l337, with an actual letter l at the beginning.

This spelling game comes from older online subcultures, including bulletin board systems, or **BBSes** — computers people connected to for messages and files before the modern web. The *Jargon File* documents those roots, the letter substitutions, and the deliberately exaggerated spelling **hax0r** for *hacker*. [6], [7] So **1337 hax0r** means *elite hacker*, and **l337 skillz** means *elite skills*. Yes, we are having some fun with the title.

Our series uses this progression:

n00b → **console punk** → **hacker & modder** → **guru** → **1337 hax0r**

These are our milestones, borrowed from slang and arranged for this series. There is no official council of hax0rz handing out certificates. A **n00b** is a beginner. A **console punk** can explore from the command line, combine tools, and investigate when something goes wrong. Later books build toward creating your own solutions, understanding systems deeply, and helping other people understand them too.

Callouts

Before we return to the progression, here are the three kinds of callout boxes that mark details worth slowing down for. Each has its own icon and label:



Tip: Pay special attention to this detail. Tips highlight useful habits and ideas that will help you understand the tools and use them well.



Trap: Watch out for a common mistake. Traps point out assumptions or actions that can get you into trouble, so you can recognize them before they catch you.



Wut: This part may not behave the way you expect. Wut callouts explain counter-intuitive details; pause and check your mental picture of what is happening.

We Are Skipping Script Kiddie

You may have expected **script kiddie** between n00b and console punk. We are going to skip it.

The term is usually an insult for someone who runs other people's code without understanding how it works. It has roots in security slang, but the *Jargon File* also describes the broader habit of copying and pasting code without having, or wanting, an explanation of what it does. [8]

Script kiddies can accomplish things. Copy, paste, run, and sometimes you get an impressive result. The problem is the mindless part: when the result is wrong, the example no longer fits, or the tool breaks, there is no understanding to build on. As a learning strategy, that is a dead end.

We expect AI to make this habit more common, because it makes producing code and commands without understanding them easier. The output can look sophisticated while the person running it has learned very little. Getting another answer every time something fails can keep you busy without making you more capable.

You can use examples, documentation, and AI while learning. The work is to understand the answer, check it, change it deliberately, and explain why the changed version behaves differently. Being new is fine. Remaining dependent on instructions you do not understand is the habit we want to avoid.



Trap: A command that worked once is not a command you have mastered. Before you reuse it, explain what goes in, what it changes, and what comes out. If you cannot, you have found the next thing to learn.

Why the Console Comes First

For now, think of the **console** as the text interface where you type commands and read results. Usually you will open a **terminal** application and interact with a **shell**, the program that interprets your commands. We will get precise about those terms as we use them.

The console puts a lot of the computer within reach. You can move through directories, inspect files, run programs, search their output, and connect one program's output to another program's input. You can turn a sequence of commands into a **script** so the computer can repeat the work for you. That gives you a place to explore programming tools, servers, data, and the operating system itself.

Until you master the basics of the console, it is hard to explore those deeper topics independently. A tutorial tells you to run a command, and suddenly you are stuck on where to type it, which directory it expects, or what its error message means. We will build that foundation here, so those details become things you can investigate yourself.

We will work layer by layer: understand a command, understand the files it uses, connect it to another command, and automate a sequence you can already explain. Each layer gives you something solid to stand on for the next one. Over time, that is how you develop *l337* skillz.

Two Families of Operating Systems

Linux and macOS behave alike in this book, and Windows behaves differently. The reason is ancestry.

UNIX was written at Bell Labs starting in 1969 by Ken Thompson, Dennis Ritchie, and their colleagues. It introduced the ideas this book depends on: one directory tree with / at its root, small programs that read and write streams of text, a shell that connects them with pipes, and permissions on every file. [9] UNIX spread to universities, and the University of California at Berkeley produced its own version, **BSD**. The behavior these systems share was later written down as the **POSIX** standard, so that a program or shell script written for one runs on the others. [10]

Linux is not a descendant of that code. Linus Torvalds started writing its kernel in 1991 as a new implementation of the same design, and a Linux system combines that kernel with GNU tools that reimplement the classic UNIX commands. [11] That is why this book says “GNU” when a Linux command differs from its BSD counterpart.

macOS is a descendant. Its core, Darwin, contains code from BSD, and macOS is a certified UNIX system. [12], [13] A Mac's Terminal therefore runs the same shells and most of the same commands as Linux, with the BSD versions of the tools; Chapter 13 covers the differences.

Windows grew from a different root. MS-DOS, released in 1981, was a single-user system for small computers, and it took drive letters such as C: and the carriage-return-plus-line-feed line ending from CP/M rather than from UNIX. [14], [15] When MS-DOS 2.0 added directories, it separated their names with a backslash, because the slash already introduced command options. [16], [17]

Windows NT, the basis of every Windows since 1993, was a new design that kept those conventions for compatibility and added its own permission model and its own shells.

When a command or a path works on Linux and macOS but not on Windows, or the other way round, this ancestry is usually the reason. This book calls Linux and macOS **UNIX-style** systems and treats Windows on its own terms, not as a UNIX with different spelling. WSL, covered in Chapter 13, runs a real Linux inside Windows, which is why a Windows machine can follow both tracks.

What This Book Covers

The chapters build on each other, so read them in order the first time:

1. **Terminal and Shell:** the window you type in, the program that answers, and how to tell them apart.
2. **Anatomy of a Command:** names, arguments, options, quoting, what the shell changes before a program runs, and where programs come from.
3. **Filesystems:** where you are, how paths work, and how to create, copy, move, and remove files safely.
4. **Stop Fighting the Keyboard:** editing the command line, history, and completion.
5. **Edit Something:** text editors you can use from the shell, and what plain text really is.
6. **Working with Files:** reading, inspecting, and searching files by name and by content.
7. **Make Commands Work Together:** pipes, redirection, exit status, and the difference between text and object pipelines.
8. **Who Can Do What?:** users, permissions, and elevation.
9. **Programs That Keep Running:** processes, jobs, and stopping things.
10. **Make the Shell Yours:** startup files, aliases, functions, prompts, and key bindings.
11. **From Commands to Scripts:** turning what you can explain into something the computer repeats.
12. **Work Across Machines:** SSH, transfers, downloads, and persistent sessions.
13. **Cross the OS Boundary:** WSL, path conversion, encodings, archives, and checksums.
14. **Keep Your Work:** Git basics and keeping secrets out of repositories.
15. **Earn Console Punk:** a project that uses everything above.

Four appendices collect what you will look up most: command comparisons across shells, keyboard shortcuts by shell and terminal, startup-file locations, and Vim survival keys.

Each instructional chapter follows the same form. It starts by explaining why the topic matters and listing the concepts it covers. It explains each concept in a way that holds on Windows, macOS, and Linux, then shows how it works in each shell with examples. It ends with **Key Points**, a **Level Up** section, and **Exercises**.

How to Work Through This Book

Keep a terminal open while you read. Try the examples, predict what they will do, and compare your prediction with the result. Change one thing and try again. When something surprises you, investigate it. Write down what you learned in words you understand.



Tip: After an example works, give yourself a slightly different problem and solve it without copying the answer. That is a good way to find out which parts you understand and which parts need another look.

Level Up marks the point where a n00b can stop and a hacker keeps going: it offers tools, questions, and alternatives to investigate on your own. The **Exercises** ask four kinds of question: what does this do, what is the problem, why is it like this, and how do you do something. Answer them in writing, then check your answers in a shell.

By the end of this book, the command prompt should feel like a place where you can ask questions and try ideas. That is our first milestone: console punk. Let's get there.

References

- [1] R. Stallman, “On hacking.” Available: <https://stallman.org/articles/on-hacking.html>
- [2] E. S. Raymond, Ed., “The jargon file: hacker.” Available: <https://www.catb.org/jargon/html/H/hacker.html>
- [3] P. R. Samson, “An abridged dictionary of the TMRC language.” 1959. Available: <https://www.gricer.com/tmrc/dictionary1959.html>
- [4] Computer History Museum, “Spacewar! Original 1962 game code.” Available: <https://www.computerhistory.org/pdp-1/playspacewar/>
- [5] M. Kosinski, “What is hacking?” IBM Think. Available: <https://www.ibm.com/think/topics/cyber-hacking>
- [6] E. S. Raymond, Ed., “The jargon file: elite.” Available: <https://www.catb.org/jargon/html/E/elite.html>
- [7] E. S. Raymond, Ed., “The jargon file: Crackers, phreaks, and lamers.” Available: <https://www.catb.org/jargon/html/crackers.html>
- [8] E. S. Raymond, Ed., “The jargon file: Script kiddies.” Available: <https://www.catb.org/jargon/html/S/script-kiddies.html>
- [9] D. M. Ritchie and K. Thompson, “The UNIX time-sharing system,” *Communications of the ACM*, vol. 17, no. 7, pp. 365–375, 1974, Available: <https://doi.org/10.1145/361011.361061>
- [10] The Open Group, “The open group base specifications issue 8, IEEE std 1003.1-2024 (POSIX).” Available: <https://pubs.opengroup.org/onlinepubs/9799919799/>
- [11] L. Torvalds, “What would you like to see most in minix? (Comp.os.minix, august 25, 1991), collected in linux history.” Available: <https://www.cs.cmu.edu/~awb/linux.history.html>
- [12] Apple, “Kernel programming guide: BSD overview.” Available: <https://developer.apple.com/library/archive/documentation/Darwin/Conceptual/KernelProgramming/BSD/BSD.html>
- [13] The Open Group, “Register of certified UNIX products.” Available: <https://www.opengroup.org/openbrand/register/>
- [14] Microsoft, “MS-DOS v1.25, v2.0, v4.0 source code.” Available: <https://github.com/microsoft/MS-DOS>
- [15] R. Chen, “Why is the line terminator CR+LF? (The old new thing).” Available: <https://devblogs.microsoft.com/oldnewthing/20040318-00/?p=40193>
- [16] R. Chen, “Why is the DOS path character backslash? (The old new thing).” Available: <https://devblogs.microsoft.com/oldnewthing/20050624-13/?p=35213>

- [17] Microsoft, “Naming files, paths, and namespaces.” Available: <https://learn.microsoft.com/en-us/windows/win32/fileio/naming-a-file>