

way of the hax0r
n00b to console punk

Chapter 3

Filesystems

A command can be spelled correctly and still fail because it looked for a file in the wrong directory. Every program, including the shell, reads and writes files somewhere in a large tree of directories, and it needs to be told where. This chapter is about that tree: how it is organized on each operating system, how a path names a place in it, where you are standing in it, and how to create, copy, move, remove, and link the files it holds.

The concepts in this chapter are:

- the **filesystem tree**, its **root**, and how Linux, macOS, and Windows arrange it;
- **absolute paths** and the **path separator**;
- filenames: case, spaces, separators, and **extensions**;
- the **current working directory**, **relative paths**, and the **.** and **..** names;
- **drives** on Windows and **mount points** on UNIX-style systems;
- **hidden files** as an attribute or a naming convention;
- listing, creating, copying, moving, renaming, and removing files and directories;
- **links** and **junctions**, which give one file or directory a second path;
- listing more than one level at a time.

The first half changes nothing. The second half creates and deletes files, so it works in a disposable practice directory. Use a normal session without administrator privileges. The PowerShell examples work in Windows PowerShell 5.1 and PowerShell 7 unless a version is named, and Windows-specific paths are labeled.

Filesystem Tree

The earliest personal computers had no folders. MS-DOS 1.0 stored every file on a disk in one list, which was workable for a few dozen files and no more. MS-DOS 2.0 added the ability to put a folder inside a folder, and the same idea had been in UNIX from the start. [1], [2], [3] Folders inside folders let a system sort its files by purpose: commands in one place, each user's files in another, installed programs in a third.

A graphical file manager calls these containers **folders**, and the picture of a folder is a good one. On the command line the word is **directory**. The two words name the same thing; this book uses directory from here on.

Because a directory can hold directories, the whole collection forms a tree. A directory inside another is its **child** or **subdirectory**, the containing directory is its **parent**, and directories with the same parent are **siblings**. The directory at the top, the one with no parent, is the **root**.

The UNIX-Style Tree

Here is the top of a macOS filesystem, shortened to the directories that matter for this book. This is a diagram, not something to type:

```

/
+-- Applications/
+-- Library/
+-- System/
+-- Users/
|   +-- Shared/
|   +-- student/
|       +-- Desktop/
|       +-- Documents/
|       +-- Downloads/
+-- Volumes/
+-- bin/
+-- etc/
+-- tmp/
+-- usr/
    +-- bin/
    +-- local/

```

The root is written /. Everything on the machine is somewhere below it: the operating system under System and Library, installed programs under Applications, command-line programs under bin and usr/bin, and each person's files under Users. [4]

Linux uses the same layout with different names in a few places. User directories live under home instead of Users, and installed programs go under usr/bin, usr/local, or opt rather than a single Applications directory. [5] Both are UNIX-style systems, which Chapter 0 explained, so the shared layout is inherited and the naming differences grew afterward.

Purpose	macOS	Linux
Root	/	/
Home directories	/Users	/home
Your home directory	/Users/student	/home/student
Installed applications	/Applications	/opt, /usr/bin, /usr/local
Core commands	/bin, /usr/bin	/bin, /usr/bin
System configuration	/etc, /Library	/etc
Removable drives	/Volumes	/media, /mnt

The Windows Tree

Windows organizes its disk differently, and it grew from MS-DOS rather than UNIX, so the names and the layout are its own:

```

C:\
+-- Program Files\
+-- Program Files (x86)\
+-- Users\
|   +-- Public\
|   +-- student\
|       +-- Desktop\
|       +-- Documents\
|       +-- Downloads\
+-- Windows\
    +-- System32\

```

The root of the main disk is `C:\`, not `\` alone, because Windows names each disk with a **drive letter**; the section on additional storage explains why. The operating system lives under `Windows`, installed programs under `Program Files`, and user directories under `Users`, which is the one name Windows shares with macOS. [6] Command-line programs have no single home; `Windows\System32` holds the built-in ones, and installers add their own directories.

Absolute Paths

A **path** names a file or directory by listing the directories from the top of the tree down to it. Instead of commas between the names, a path uses a **separator**: `/` on UNIX-style systems and `\` on Windows. A path that starts at the root is an **absolute path**, and it names the same place no matter where you are. [7], [8]

Target	macOS	Linux	Windows
Your home directory	<code>/Users/student</code>	<code>/home/student</code>	<code>C:\Users\student</code>
A file in Documents	<code>/Users/student/Documents/notes.txt</code>	<code>/home/student/Documents/notes.txt</code>	<code>C:\Users\student\Documents\notes.txt</code>
A system command	<code>/bin/ls</code>	<code>/usr/bin/ls</code>	<code>C:\Windows\System32\cmd.exe</code>
A removable drive	<code>/Volumes/USB</code>	<code>/media/student/USB</code>	<code>D:\</code>

Read `/home/student/Documents/notes.txt` as “start at the root, enter home, enter student, enter Documents, and there is notes.txt.” The separators are not part of any name; they mark the boundaries between names.

PowerShell on macOS or Linux uses that system’s `/` paths, and PowerShell on Windows accepts either separator, but the operating system underneath still has one native form. An absolute path is also not portable between computers: the username, the drive, and the layout can all differ, so copy the form of a path from an example and the names from your own machine.

Names

A filename can be almost anything, and the exceptions cause most of the trouble.

Capitalization

Windows and macOS filesystems are normally **case-insensitive**: `Notes.txt` and `notes.txt` name the same file, although the system remembers how you capitalized it. Linux filesystems are **case-**

sensitive: those two names are two different files that can sit side by side. Both Windows and macOS can be configured the other way for particular directories or volumes, so the rule belongs to the filesystem, not to the operating system's name. [9], [10]

The habit that works everywhere is to type the name exactly as a listing shows it. A project that moves from a Mac to a Linux server breaks when one file is referred to with two spellings.

Slashes and Backslashes

A UNIX-style filename may contain a backslash, because only / is the separator there; a file named back\slash.txt is legal on Linux and macOS, if unwise. Windows forbids both / and \ in a name, along with :, *, ?, ", <, >, and |, because each of them means something to the system. [11] When a name from one system is not allowed on the other, rename it before copying.

Spaces

A space in a name is legal everywhere and inconvenient on the command line, where a space separates arguments. Chapter 2 showed the fix: quote the whole name, as in "My Documents", or in Bash and zsh escape the space with a backslash. Directories that Windows and macOS create for you contain spaces, Program Files and Application Support among them, so you cannot avoid learning this.

Extensions

An **extension** is the suffix after the last dot in a filename: .txt, .png, .md. It is part of the name, so notes, notes.txt, and notes.txt.txt are three names.

For most files the extension is a hint and nothing more. It tells you and your programs what to expect inside, but renaming picture.png to picture.txt changes nothing about the bytes; the file is still an image, and an image viewer will still open it if asked. Chapter 6 shows the file command, which reports what a file contains regardless of its name.

Windows is the exception. There, a few extensions decide whether a file can run as a program: .exe, .com, .bat, .cmd, and the others in the PATHEXT variable, plus .ps1 for PowerShell scripts. Double-clicking or typing the name of a file with one of those extensions runs it, and a file with any other extension opens in whatever program is associated with it. Chapter 8 returns to this when it explains what makes a file executable. [12]

Graphical file managers can hide extensions. In Finder, **Show all filename extensions** under **Settings > Advanced** turns them on; File Explorer has **File name extensions** on its View menu. Turn them on when comparing a window with a shell listing. [13], [14]

Common extensions you will see in this book:

Extension	Contents
.txt, .md	Plain text, Markdown text
.sh, .ps1, .bat, .cmd	Bash, PowerShell, and Command Prompt scripts
.py, .js, .c	Source code
.json, .yaml, .csv	Structured data as text
.log	A program's log, usually text

Extension	Contents
.zip, .tar.gz	Archives
.png, .jpg, .pdf	Images and documents, not text
.exe, .dll	Windows programs and libraries

Current Working Directory

At any moment, a shell, a program, or an application is running in a particular directory, called its **current working directory** or just the current directory. When a program is given a name that is not an absolute path, it looks for that name starting there. Each process has its own: two terminal tabs can sit in different directories, and changing one does not move the other.

Your prompt may show the current directory, but it can abbreviate or omit it. Ask the shell directly when you need to know.

Bash/zsh:

```
$ pwd
```

`pwd` means **print working directory**, and it prints an absolute path such as `/home/student` or `/Users/student`. [15], [16]

PowerShell:

```
$ Get-Location
```

Look at the Path field; on Windows it might be `C:\Users\student`. [17]

Command Prompt:

```
$ cd
```

`cd` with no argument displays the current drive and directory. [18]



Wut: Bare `cd` does different things in different shells. In Bash and zsh it takes you home, in Command Prompt it reports your location, and in PowerShell it takes you home in PowerShell 7 but does nothing in Windows PowerShell 5.1. Check what a command does in the shell you are using. [19]

Relative Paths, Here, and Up

A **relative path** starts from the current directory instead of the root. `Documents/notes.txt` means “the Documents directory here, and `notes.txt` inside it”, and it names a different file depending on where you are.

Every directory contains two special names that relative paths use:

- `.` means the current directory itself.
- `..` means its parent.

The two dots are one name, not two arguments. Because they exist in every directory, you can use them without knowing the directory's real name. [8], [20]

Imagine your current directory is `game` in this tree:

```
/home/student/
  +-- projects/
  |   +-- game/
  |       +-- notes.txt
  |       +-- website/
  +-- downloads/
```

Target	UNIX-style relative path	Windows-style relative path
The file inside <code>game</code>	<code>notes.txt</code> or <code>./notes.txt</code>	<code>notes.txt</code> or <code>.\notes.txt</code>
The parent, <code>projects</code>	<code>..</code>	<code>..</code>
The sibling, <code>website</code>	<code>../website</code>	<code>..\website</code>
downloads under home	<code>../../downloads</code>	<code>..\..\downloads</code>

Read the last row as “up to `projects`, up to home, down to `downloads`.” A relative path is convenient inside a project whose layout stays the same wherever it is copied; an absolute path is unambiguous but tied to one machine.

Changing Directory

Each shell has a command to change the current directory, and each prints nothing when it succeeds. Check your location afterward rather than treating silence as an error, and read the error when there is one.

Bash/zsh:

```
$ cd /tmp
$ pwd
/tmp
$ cd ..
$ pwd
/
$ cd
$ pwd
/home/student
```

`cd` means **change directory**. With no argument it goes to your home directory, the value of the `HOME` variable. `cd ~` and `cd "$HOME"` do the same: the **tilde** expands to your home directory, and `~alice` expands to another user's home, where permissions allow. `cd -` returns to the directory you were in before the last `cd`, which is useful when bouncing between two places. [15], [21], [22]

Do not write `cd "~`: quoting the tilde stops it expanding, and the shell looks for a directory literally named `~`. [21], [23]

PowerShell:

```
$ Set-Location -LiteralPath /tmp
$ Set-Location -LiteralPath ..
$ Set-Location -LiteralPath $HOME
$ Get-Location
```

Set-Location changes the current location, and `cd` is its alias. `$HOME` and `~` both name your home directory. Set-Location - returns to the previous location in PowerShell 6.2 and later; Windows PowerShell 5.1 has Push-Location and Pop-Location instead. PowerShell does not expand `~alice`. [19], [24]

`-LiteralPath` tells Set-Location to use the path exactly as written, without interpreting characters such as square brackets as wildcard patterns. Chapter 2 used `-Path`; this chapter uses `-LiteralPath` so that the examples work for any name.

Command Prompt:

```
$ cd \
$ cd
$ cd /d "%USERPROFILE%"
$ cd
```

`cd \` goes to the root of the current drive. `%USERPROFILE%` is your home directory, and the `/d` option changes the drive as well as the directory, which the next section explains. Command Prompt has no `cd -`; `pushd` and `popd` save and restore a location. [18], [25], [26], [27]

Additional Storage

A second disk, a USB stick, or a network share has its own tree. The two families attach that tree to what you already have in different ways.

Windows: Drive Letters

Windows does not graft new storage into the existing tree. Each volume gets a **drive letter**, and each letter is the root of its own tree: `C:\` for the main disk, `D:\` for the next, and so on. A USB stick appears as the next free letter, and a network share can be mapped to one. The letter identifies a volume, not necessarily a separate physical disk. [11]

An absolute Windows path therefore starts with a letter: `D:\photos\trip.jpg`. Each drive also has its own remembered current directory, and switching drives is a separate act from changing directory:

```
$ D:
$ cd \photos
$ C:
```

In Command Prompt, typing a drive letter with a colon switches to that drive, and `cd /d D:\photos` does both in one step. In PowerShell, `Set-Location D:\photos` switches drive and directory together, and `Get-PSDrive -PSProvider FileSystem` lists the drives with their free space. [18], [28]



Trap: `C:\notes.txt`, `C:notes.txt`, and `\notes.txt` are three different paths. The first starts at the root of `C:`, the second at the remembered current directory of `C:`, and the third at the root of whatever drive is current. Use a letter followed by a backslash when you mean a drive root. [8]

Network paths look like `\\server\share\folder`. Recognize the form; you do not need one for this chapter.

UNIX: Mount Points

UNIX-style systems keep one tree. New storage is **mounted** at a directory, called the **mount point**, and from then on the directory's contents are the new volume's contents. `mount` with no arguments lists what is mounted where, and `df -h` shows each mounted filesystem with its free space. [7], [29]

```
$ df -h
```

A USB stick plugged into a Linux desktop is mounted automatically under `/media/student/` or `/run/media/student/`, named after the stick's label. On macOS it appears under `/Volumes/`. A path such as `/Volumes/USB/photos/trip.jpg` looks like any other absolute path, which is the point: programs need not know that the file is on removable media. [4], [5]

A **filesystem image** is a single file that contains a whole filesystem, such as an `.iso` for an installer or a `.dmg` on macOS. Mounting one makes its contents appear at a mount point, the same as a disk: `hdiutil attach file.dmg` on macOS, `sudo mount -o loop file.iso /mnt` on Linux, and `Mount-DiskImage file.iso` in PowerShell on Windows, where the image gets a drive letter. Mounting on Linux needs administrator rights, which Chapter 8 covers; the desktop does it for you when you double-click. [29], [30], [31]

Hidden Files

Some entries are left out of directory listings to reduce clutter, and the two families decide which ones differently.

Windows keeps a **hidden attribute** on each file, separate from its name. `dir` and File Explorer skip files with the attribute set, and `attrib +h name` or the file's Properties window sets it. [32], [33]

UNIX-style systems have no such attribute. By convention, a name that starts with a dot, such as `.bashrc` or `.git`, is left out of the default `ls` listing and of most file managers. Nothing about the file is hidden; `ls -a` shows it, any program can open it, and the dot is just the first character of its name. [34] Configuration files use this convention, which is why your home directory has more in it than `ls` shows.

Neither mechanism restricts access. Chapter 8 covers permissions, which do.

Working with Files

Everything from here on creates, changes, or deletes files, so make a place where mistakes do not matter. Go home, create `hax0r-practice`, and move into it:

Bash/zsh:

```
$ cd
$ mkdir hax0r-practice
$ cd hax0r-practice
```

PowerShell:

```
$ Set-Location -LiteralPath $HOME
$ New-Item -ItemType Directory -Path hax0r-practice
$ Set-Location -LiteralPath hax0r-practice
```

Command Prompt:

```
$ cd /d "%USERPROFILE%"
$ mkdir hax0r-practice
$ cd hax0r-practice
```

`mkdir` means **make directory**. PowerShell's `New-Item` creates files or directories depending on `-ItemType`, and on Windows PowerShell also defines a `mkdir` function that wraps it. Creating a directory that already exists is an error in all three; `mkdir -p a/b/c` in Bash and zsh, `New-Item -ItemType Directory a/b/c`, and `mkdir a\b\c` in Command Prompt create several levels at once. [35], [36], [37]

Check that your prompt or `pwd` shows `hax0r-practice` before every command in the rest of the chapter.

Listing Files

You saw the listing commands in Chapter 2. Here they are with the options that show what you need for the rest of the chapter:

Bash/zsh:

```
$ ls
$ ls -l
$ ls -a
$ ls -F
```

`ls` alone prints names. `-l` is the **long listing**, one line per entry with permissions, owner, size, and modification time; `-a` includes dot files; `-F` marks directories with a trailing `/` and executables with `*`. Options combine, so `ls -laF` is common. [38]

PowerShell:

```
$ Get-ChildItem
$ Get-ChildItem -Force
```

The default listing is already a long one, with a `Mode` column where `d` marks a directory on Windows and a `UnixMode` column on Linux and macOS. `-Force` includes hidden entries. [39]

Command Prompt:

```
$ dir
$ dir /a
```

dir shows sizes and times and labels directories <DIR>; /a includes hidden entries. [33]

Create Files

An empty file is enough to practice with. Create one, then a second with a line of text in it:

Bash/zsh:

```
$ touch notes.txt
$ printf 'hello\n' > hello.txt
$ ls -l
```

touch creates an empty file, or updates the modification time of an existing one. The > sends the output of printf into a file instead of the screen, creating the file; Chapter 7 explains redirection fully. The long listing shows hello.txt at six bytes and notes.txt at zero. [40]

PowerShell:

```
$ New-Item -ItemType File -Path notes.txt
$ Set-Content -Path hello.txt -Value 'hello'
$ Get-ChildItem
```

New-Item -ItemType File creates an empty file, and Set-Content writes text, creating the file if needed. [36], [41]

Command Prompt:

```
$ type nul > notes.txt
$ echo hello > hello.txt
$ dir
```

Command Prompt has no touch; type nul > notes.txt copies nothing into a new file, which is the usual idiom. [42], [43]



Trap: > replaces a file's contents without asking. printf 'x' > notes.txt empties a file you spent an hour on and writes one character. Check the name after > before you press Enter.



Trap: Windows locks files that a program has open. While Word has report.docx open, or a running program has its log open, a command that tries to overwrite, rename, or delete the file fails with "The process cannot access the file because it is being used by another process", and some programs open files in a way that blocks reading too. Close the program, or find it with the tools in Chapter 9, before retrying. Linux and macOS allow the operation and let the program keep its old copy. [44]

Copy

Copying makes a second, independent file: changing one does not change the other.

Bash/zsh:

```
$ cp hello.txt copy.txt
$ cp -r . ../hax0r-backup
$ ls ../hax0r-backup
```

cp takes a source and a destination. If the destination is an existing directory, the copy goes inside it with the same name. Copying a directory needs **recursion**, which -r requests: cp descends into the directory and copies everything below it. The second command copies the whole practice directory, ., to a new sibling named hax0r-backup. [45]

PowerShell:

```
$ Copy-Item -Path hello.txt -Destination copy.txt
$ Copy-Item -Path . -Destination ../hax0r-backup -Recurse
$ Get-ChildItem ../hax0r-backup
```

-Recurse is required for a directory's contents. [46]

Command Prompt:

```
$ copy hello.txt copy.txt
$ xcopy . ../hax0r-backup /e /i
$ dir ../hax0r-backup
```

copy handles files. Copying a directory tree uses xcopy, where /e copies subdirectories including empty ones and /i says the destination is a directory; robocopy is the more capable modern tool, with defaults that differ from copy, so read its help first. [47], [48], [49]

Overwrites

If the destination of a copy already exists, most tools replace it, and the replaced file is gone.

- GNU cp overwrites silently; -i asks before each overwrite, and -n skips existing files.
- Copy-Item overwrites silently; -Confirm asks, and -WhatIf shows what would happen without doing it.
- In an interactive Command Prompt, copy asks before overwriting; in a batch file it does not unless you add /-Y. [45], [47], [50]

Try the safe forms, and answer n if asked:

Bash/zsh:

```
$ cp -i hello.txt notes.txt
```

PowerShell:

```
$ Copy-Item -Path hello.txt -Destination notes.txt -WhatIf
```

Command Prompt:

```
$ copy hello.txt notes.txt
```

-WhatIf never acts, so it never asks. It works with most PowerShell commands that change something and is the best habit in this chapter. [50]

The Windows open-file rule applies to copies too. On Windows, open `notes.txt` in Notepad, then run the copy line above: Notepad does not lock its files, so the copy succeeds, but the same command against a file open in Word or Excel fails with the “being used by another process” message. Which programs lock their files is up to each program.

Move and Rename

Moving and renaming are the same operation: the entry gets a new path, and within one filesystem no data is copied. UNIX has one command for both; Windows has two.

Bash/zsh:

```
$ mkdir projects
$ mv copy.txt renamed.txt
$ mv renamed.txt projects/
$ ls projects
```

`mv` renames when the destination is a new name and moves when the destination is an existing directory. It overwrites an existing destination file silently; `-i` asks first. [51]

PowerShell:

```
$ New-Item -ItemType Directory -Path projects
$ Rename-Item -Path copy.txt -NewName renamed.txt
$ Move-Item -Path renamed.txt -Destination projects
$ Get-ChildItem projects
```

`Rename-Item` changes the name in place and `Move-Item` changes the location; `Move-Item` can also give the item a new name at the destination. `Move-Item` refuses to replace an existing file unless you add `-Force`. [52], [53]

Command Prompt:

```
$ mkdir projects
$ ren copy.txt renamed.txt
$ move renamed.txt projects
$ dir projects
```

`ren`, also `rename`, changes a name; `move` changes location and asks before overwriting in an interactive session. [54], [55]

A move to a different filesystem, such as from your home directory to a USB stick or from `C:` to `D:`, cannot just rename the entry. The tool copies the data to the destination and then deletes the original,

which takes as long as a copy and can leave a partial copy if it is interrupted. The commands are the same; the time and the risk are not.

Remove

Removing a file in the shell does not move it to a trash can. The name is gone immediately, and the only way back is a backup.

Bash/zsh:

```
$ rm notes.txt
$ rmdir projects
rmdir: failed to remove 'projects': Directory not empty
$ rm -r projects
$ ls
```

`rm` removes files. `rmdir` removes only an empty directory, which is a useful safety check; the error wording differs between GNU and macOS. `rm -r` removes a directory and everything under it, with no confirmation. `rm -i` asks before each file, and `rm -f` suppresses errors and prompts. Do not combine `-r` and `-f` until you can explain exactly which path you are giving, because `rm -rf` with a mistyped path is the classic way to lose a directory. [56], [57]

PowerShell:

```
$ Remove-Item -Path notes.txt
$ Remove-Item -Path projects
```

The second command stops and asks, because the directory has contents and you did not say `-Recurse`. Answer `N`, preview with `-WhatIf`, then remove it: [58]

```
$ Remove-Item -Path projects -Recurse -WhatIf
$ Remove-Item -Path projects -Recurse
```

Command Prompt:

```
$ del notes.txt
$ rd projects
$ rd /s projects
```

`del`, also `erase`, removes files. `rd`, also `rmdir`, removes an empty directory; `/s` removes the contents too and asks once, and `/q` suppresses the question. [59], [60]



Trap: A wildcard in a remove command is expanded or interpreted before anything is deleted, and a space in the wrong place changes what it names. `rm * .bak` removes every file in the directory, not the `.bak` files. List the same pattern first with `ls`, `Get-ChildItem`, or `dir`; when the listing shows exactly the files you expect, edit that line into the remove command.

Links and Junctions

A **link** gives an existing file or directory a second name. It is the one feature in this chapter that makes the tree less straightforward, because two paths lead to the same thing.

A **symbolic link**, or **symlink**, is a small file that holds a path to its target. Opening the link opens the target; deleting the target leaves a link that points nowhere. A symbolic link can point across filesystems, and it can point to a directory.

A **hard link** is a second directory entry for the same file data. Neither name is the original; the file stays until every hard link to it is removed. Hard links cannot cross filesystems, and on most systems they cannot point to directories. Windows adds a **junction**, an older kind of directory link that works without special privileges. [7], [61]

```
hax0r-practice/
+-- hello.txt -----+
|                               | same data on disk
+-- hard.txt -----+ (hard link: two names, one file)
|
+-- link.txt --> "hello.txt" (symlink: a file holding a path)
|
+-- backup-link --> "../hax0r-backup/" (symlink to a directory)
```

Bash/zsh:

```
$ ln -s hello.txt link.txt
$ ln hello.txt hard.txt
$ ln -s ../hax0r-backup backup-link
$ ls -l
```

`ln -s` creates a symbolic link, which the long listing shows as `link.txt -> hello.txt` with an `l` in the first column. `ln` without `-s` creates a hard link, which looks like an ordinary file; `ls -li` shows that `hello.txt` and `hard.txt` share an **inode** number, the filesystem's internal identity for the data. [62]

PowerShell:

```
$ New-Item -ItemType SymbolicLink -Path link.txt -Target hello.txt
$ New-Item -ItemType HardLink -Path hard.txt -Target hello.txt
$ New-Item -ItemType Junction -Path backup-link -Target ../hax0r-backup
$ Get-ChildItem | Select-Object Mode, Name, LinkType, Target
```

On Windows, creating a symbolic link requires an elevated session or Developer Mode, and the first command fails with a privilege error in an ordinary session; the junction on the third line needs neither. Junction is a Windows feature; on Linux and macOS use `SymbolicLink` for the directory link instead, and the other two commands work as written. [36], [63]

Command Prompt:

```
$ mklink link.txt hello.txt
$ mklink /h hard.txt hello.txt
```

```
$ mklink /j backup-link ..\hax0r-backup
```

mklink creates a file symlink by default, /d a directory symlink, /h a hard link, and /j a junction, with the same privilege rule: symlinks need elevation or Developer Mode, junctions and hard links do not. [64]

Removing a link by its own name removes only the link. Going through the link is different: `rm -r backup-link/*` removes the contents of the real backup directory, and a trailing slash, as in `rm -r backup-link/`, can do the same depending on the implementation. Check what a path is before recursing through it.

Advanced Listing

The plain listing answers “what is here.” These options answer “what is here, sorted the way I need” and “what is anywhere below here.”

Bash/zsh:

```
$ ls -lh
$ ls -lt
$ ls -lS
$ ls -R
```

-h prints sizes as 4.0K or 2.3M instead of bytes, -t sorts by modification time with the newest first, -S sorts by size, and -r reverses any sort. -R lists every subdirectory too, which is the first way to see a whole tree. [38]

The `tree` command draws the tree with indentation, like the diagrams in this chapter; it is installed with a package manager on Linux and macOS. `find` walks a tree and prints every path, with tests to narrow the result:

```
$ find . -type d
$ find . -name '*.txt'
```

Chapter 6 covers `find` properly; for now, `find .` with no tests is `ls -R` with full paths. [65], [66]

PowerShell:

```
$ Get-ChildItem | Sort-Object Length -Descending
$ Get-ChildItem | Sort-Object LastWriteTime
$ Get-ChildItem -Recurse
```

PowerShell has no sort options on the listing command; instead, the listing is piped to `Sort-Object`, which Chapter 7 explains. -Recurse lists the whole tree. [39]

Command Prompt:

```
$ dir /o-s
$ dir /od
$ dir /s
$ tree
```

/o sorts, with -s for largest first and d for oldest first, and /s searches subdirectories. Windows includes tree, which draws directories and, with /f, the files in them. [33], [67]

Clean Up

Leave the practice directory, then remove it and the backup:

Bash/zsh:

```
$ cd
$ rm -r hax0r-practice hax0r-backup
```

PowerShell:

```
$ Set-Location -LiteralPath $HOME
$ Remove-Item -Path hax0r-practice, hax0r-backup -Recurse
```

Command Prompt:

```
$ cd /d "%USERPROFILE%"
$ rd /s hax0r-practice
$ rd /s hax0r-backup
```

Leaving first matters: Windows refuses to remove the current directory, and on any system the relative name hax0r-practice does not exist inside itself. Recreate the directory whenever a later chapter asks for one.

Key Points

- Files live in a tree of directories with one root on UNIX-style systems and one root per drive letter on Windows; macOS and Linux share the layout with different names, and Windows has its own.
- An absolute path lists the directories from the root, separated by / or \; a relative path starts from the current directory, where . is here and .. is the parent.
- Windows and macOS ignore case in names, Linux does not; Windows forbids the separator characters in names; spaces need quotes; extensions are hints except for the executable extensions on Windows.
- pwd, Get-Location, and bare cd show the current directory; cd, Set-Location, and cd /d change it, and cd - or Set-Location - returns to the previous one.
- New storage becomes a drive letter on Windows and a mount point on UNIX-style systems.
- Hidden files are an attribute on Windows and a leading dot elsewhere; neither restricts access.
- mkdir, touch, cp, mv, rm, ln; New-Item, Copy-Item, Move-Item, Rename-Item, Remove-Item; and mkdir, copy, move, ren, del, rd, mklink create, copy, move, remove, and link.
- Overwrites replace files silently in most tools, nothing goes to a trash can, and Windows refuses to touch a file another program holds open; -i, -Confirm, and -WhatIf let you check first.
- ls -R, Get-ChildItem -Recurse, dir /s, tree, and find show more than one level.

Level Up

n00bs finish the chapter here. You can read a path, find and change your location, and create, copy, move, remove, and link files without a file manager. Hackers look at the tree from the filesystem's side.

Read the Filesystem Hierarchy Standard for Linux and Apple's file system overview, and explain why `/usr/bin` and `/usr/local/bin` are separate. Then list your home directory with `ls -la` and identify what every dot file is for.

Investigate `cd -`, `pushd`, and `popd` in Bash and `zsh`, `Push-Location` and `Pop-Location` in PowerShell, and `pushd` and `popd` in Command Prompt, including how the Windows versions handle drive changes and network paths. [15], [16], [24], [26], [27] Explain what a directory stack is and when it beats `cd -`.

Run `ls -li` on a file and its hard link, then read about inodes and explain what `rm` actually removes. Find out what `stat` reports on Linux and macOS, and what `Get-Item` reports in PowerShell, about the same file.

Look at the tools that protect you from your own commands: `trash-cli` on Linux, `trash` on macOS, and the `-WhatIf` habit in PowerShell. Explore `cp --reflink` on filesystems that support it, and `robocopy /mir` on Windows, and explain what each does that a plain copy does not.

Plug in a USB stick and watch where it appears: `mount`, `df -h`, `/Volumes`, or a new drive letter. Then read about what "safely remove" does that pulling the stick out does not.

Exercises

1. **What does this do?** With `game` as your current directory in the diagram, predict where `cd ../../downloads` takes you, then what happens if you run the same command again.
2. **What does this do?** Predict the result of `mv notes.txt projects` when `projects` is an existing directory, and when it is not.
3. **What is the problem?** `cd Downloads` reports "No such file or directory" on Linux, but the file manager shows a Downloads folder. Give two different causes and the command that distinguishes them.
4. **What is the problem?** On Windows, `del report.docx` fails with "being used by another process" even though you have permission to delete it. Explain what is happening and what to do first.
5. **What is the problem?** `rm -r backup-link` removed only the link, but `rm -r backup-link/*` emptied the real backup directory. Explain the difference and how to check what a path is before recursing through it.
6. **Why is it like this?** A move within one disk finishes instantly, but the same move to a USB stick takes as long as a copy. Explain what the filesystem does in each case.
7. **Why is it like this?** Renaming `picture.png` to `picture.txt` does not turn it into text, but renaming `tool.txt` to `tool.exe` on Windows changes what double-clicking does. Explain both.
8. **How do you** copy an entire directory tree in each of the three shells, and what happens in each if you leave out the recursive option?
9. **How do you** find the five largest files in a directory, in Bash and in PowerShell?

References

- [1] Microsoft, “MS-DOS v1.25, v2.0, v4.0 source code.” Available: <https://github.com/microsoft/MS-DOS>
- [2] R. Chen, “Why is the DOS path character backslash? (The old new thing).” Available: <https://devblogs.microsoft.com/oldnewthing/20050624-13/?p=35213>
- [3] 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>
- [4] Apple, “File system programming guide: File system basics.” Available: <https://developer.apple.com/library/archive/documentation/FileManagement/Conceptual/FileSystemProgrammingGuide/FileSystemOverview/FileSystemOverview.html>
- [5] Linux Foundation, “Filesystem hierarchy standard, version 3.0.” Available: https://refspecs.linuxfoundation.org/FHS_3.0/fhs/index.html
- [6] Microsoft, “KNOWNFOLDERID (windows shell).” Available: <https://learn.microsoft.com/en-us/windows/win32/shell/knownfolderid>
- [7] The Open Group, “POSIX.1-2024: General concepts, pathname resolution.” Available: https://pubs.opengroup.org/onlinepubs/9799919799/basedefs/V1_chap04.html
- [8] Microsoft, “File path formats on windows systems.” Available: <https://learn.microsoft.com/en-us/dotnet/standard/io/file-path-formats>
- [9] Microsoft, “Case sensitivity.” Available: <https://learn.microsoft.com/en-us/windows/wsl/case-sensitivity>
- [10] Apple, “File system formats available in disk utility on mac.” Available: <https://support.apple.com/guide/disk-utility/file-system-formats-dsku19ed921c/mac>
- [11] Microsoft, “Naming files, paths, and namespaces.” Available: <https://learn.microsoft.com/en-us/windows/win32/fileio/naming-a-file>
- [12] Microsoft, “About_environment_variables.” Available: https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_environment_variables
- [13] Apple, “Show or hide filename extensions on mac.” Available: <https://support.apple.com/guide/mac-help/mchlp2304/mac>
- [14] Microsoft, “File explorer in windows.” Available: <https://support.microsoft.com/en-us/windows/experience/fileexplorer/file-explorer-in-windows>
- [15] GNU Project, “Bash reference manual: Bourne shell builtins.” Available: https://www.gnu.org/software/bash/manual/html_node/Bourne-Shell-Builtins.html
- [16] Zsh Developers, “The z shell manual: Shell builtin commands.” Available: <https://zsh.sourceforge.io/Doc/Release/Shell-Builtin-Commands.html>

- [17] Microsoft, “Get-location.” Available: <https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.management/get-location>
- [18] Microsoft, “Windows commands: cd.” Available: <https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/cd>
- [19] Microsoft, “Set-location.” Available: <https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.management/set-location?view=powershell-7.4>
- [20] Apple, “Specify files and folders in terminal on mac.” Available: <https://support.apple.com/guide/terminal/apd3cf6fe02-3ec8-48f1-951f-866e52955fc8/mac>
- [21] GNU Project, “Bash reference manual: Tilde expansion.” Available: https://www.gnu.org/software/bash/manual/html_node/Tilde-Expansion.html
- [22] Zsh Developers, “The z shell manual: parameters.” Available: <https://zsh.sourceforge.io/Doc/Release/Parameters.html>
- [23] Zsh Developers, “The z shell manual: Shell grammar, quoting.” Available: <https://zsh.sourceforge.io/Doc/Release/Shell-Grammar.html#Quoting>
- [24] Microsoft, “Push-location.” Available: <https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.management/push-location>
- [25] Microsoft, “Recognized environment variables.” Available: <https://learn.microsoft.com/en-us/windows/deployment/usmt/usmt-recognized-environment-variables>
- [26] Microsoft, “Windows commands: pushd.” Available: <https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/pushd>
- [27] Microsoft, “Windows commands: popd.” Available: <https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/popd>
- [28] Microsoft, “Get-PSDrive.” Available: <https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.management/get-psdrive?view=powershell-7.5>
- [29] Linux man-pages Project, “Mount(8): Mount a filesystem.” Available: <https://man7.org/linux/man-pages/man8/mount.8.html>
- [30] Apple, “Hdiutil(1): Manipulate disk images (macOS manual page).” Available: <https://keith.github.io/xcode-man-pages/hdiutil.1.html>
- [31] Microsoft, “Mount-DiskImage.” Available: <https://learn.microsoft.com/en-us/powershell/module/storage/mount-diskimage>
- [32] Microsoft, “File attribute constants (windows file management).” Available: <https://learn.microsoft.com/en-us/windows/win32/fileio/file-attribute-constants>
- [33] Microsoft, “Windows commands: dir.” Available: <https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/dir>
- [34] GNU Project, “GNU coreutils: Which files are listed.” Available: https://www.gnu.org/software/coreutils/manual/html_node/Which-files-are-listed.html
- [35] GNU Project, “GNU coreutils manual: Mkdir: Make directories.” Available: https://www.gnu.org/software/coreutils/manual/html_node/mkdir-invocation.html
- [36] Microsoft, “New-item.” Available: <https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.management/new-item>
- [37] Microsoft, “Windows commands: mkdir.” Available: <https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/mkdir>

- [38] GNU Project, “GNU coreutils manual: Ls: List directory contents.” Available: https://www.gnu.org/software/coreutils/manual/html_node/ls-invocation.html
- [39] Microsoft, “Get-ChildItem.” Available: <https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.management/get-childitem>
- [40] GNU Project, “GNU coreutils manual: Touch: Change file timestamps.” Available: https://www.gnu.org/software/coreutils/manual/html_node/touch-invocation.html
- [41] Microsoft, “Set-content.” Available: <https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.management/set-content>
- [42] Microsoft, “Windows commands: type.” Available: <https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/type>
- [43] Microsoft, “Windows commands: echo.” Available: <https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/echo>
- [44] Microsoft, “Creating and opening files (windows file management).” Available: <https://learn.microsoft.com/en-us/windows/win32/fileio/creating-and-opening-files>
- [45] GNU Project, “GNU coreutils manual: Cp: Copy files and directories.” Available: https://www.gnu.org/software/coreutils/manual/html_node/cp-invocation.html
- [46] Microsoft, “Copy-item.” Available: <https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.management/copy-item>
- [47] Microsoft, “Windows commands: copy.” Available: <https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/copy>
- [48] Microsoft, “Windows commands: xcopy.” Available: <https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/xcopy>
- [49] Microsoft, “Windows commands: robocopy.” Available: <https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/robocopy>
- [50] Microsoft, “About_CommonParameters.” Available: https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_commonparameters
- [51] GNU Project, “GNU coreutils manual: Mv: Move (rename) files.” Available: https://www.gnu.org/software/coreutils/manual/html_node/mv-invocation.html
- [52] Microsoft, “Move-item.” Available: <https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.management/move-item>
- [53] Microsoft, “Rename-item.” Available: <https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.management/rename-item>
- [54] Microsoft, “Windows commands: ren.” Available: <https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/ren>
- [55] Microsoft, “Windows commands: move.” Available: <https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/move>
- [56] GNU Project, “GNU coreutils manual: Rm: Remove files or directories.” Available: https://www.gnu.org/software/coreutils/manual/html_node/rm-invocation.html
- [57] GNU Project, “GNU coreutils manual: Rmdir: Remove empty directories.” Available: https://www.gnu.org/software/coreutils/manual/html_node/rmdir-invocation.html
- [58] Microsoft, “Remove-item.” Available: <https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.management/remove-item>

- [59] Microsoft, “Windows commands: del.” Available: <https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/del>
- [60] Microsoft, “Windows commands: rd.” Available: <https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/rd>
- [61] Microsoft, “Symbolic links (Win32 file management).” Available: <https://learn.microsoft.com/en-us/windows/win32/fileio/symbolic-links>
- [62] GNU Project, “GNU coreutils manual: Ln: Make links between files.” Available: https://www.gnu.org/software/coreutils/manual/html_node/ln-invocation.html
- [63] Microsoft, “Enable your device for development.” Available: <https://learn.microsoft.com/en-us/windows/apps/get-started/enable-your-device-for-development>
- [64] Microsoft, “Windows commands: mklink.” Available: <https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/mklink>
- [65] Steve Baker, “Tree: List contents of directories in a tree-like format.” Available: <https://oldmanprogrammer.net/source.php?dir=projects/tree>
- [66] GNU Project, “GNU findutils manual: Finding files.” Available: https://www.gnu.org/software/findutils/manual/html_mono/find.html
- [67] Microsoft, “Windows commands: tree.” Available: <https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/tree>