

way of the hax0r
n00b to console punk

Chapter 6

Working with Files

You can navigate to a file and open it in an editor. Often that is more than you need: you want to know what a file is, look at its first lines, or find which file in a project mentions a word. Opening each candidate by hand does not scale past a handful of files. The shell has tools that read, measure, and search files without editing them, and they work on a thousand files as easily as one.

The concepts in this chapter are:

- **paggers**, which show a long file one screen at a time;
- **inspection**: file type, first and last lines, and counts;
- searching **by name** with `find`, `Get-ChildItem`, and `dir`;
- searching **by content** with `grep`, `rg`, `Select-String`, and `findstr`;
- **globs** versus **regular expressions**, two pattern languages that look alike and are not.

None of these commands modify files. Set up a small practice set first so the examples have something to find.

A Practice Set

Recreate `hax0r-practice` from Chapter 3, move into it, and create three files in two directories.

Bash/zsh:

```
$ mkdir -p src docs
$ printf 'alpha\nbeta gamma\nGamma ray\ndelta\n' > notes.txt
$ printf 'todo: fix bug\ndone: write docs\n' > docs/todo.md
$ printf 'int main() { return 0; }\n' > src/main.c
```

PowerShell:

```
$ New-Item -ItemType Directory -Path src, docs
$ Set-Content -Path notes.txt -Value 'alpha', 'beta gamma', 'Gamma ray', 'delta'
$ Set-Content -Path docs/todo.md -Value 'todo: fix bug', 'done: write docs'
$ Set-Content -Path src/main.c -Value 'int main() { return 0; }'
```

`Set-Content` writes each string in the list as one line.

Command Prompt:

```
$ mkdir src docs
$ (echo alpha& echo beta gamma& echo Gamma ray& echo delta) > notes.txt
$ (echo todo: fix bug& echo done: write docs) > docs\todo.md
$ echo int main() { return 0; } > src\main.c
```

The parentheses group several commands so that one redirection collects all their output; & separates commands on one line. [1]

Read a Long File

cat prints a whole file to the terminal. For anything longer than a screen, use a **pager**, which shows one screen and waits.

Bash/zsh:

```
$ less notes.txt
```

less is the pager you used with man in Chapter 2. Space or f moves forward a screen, b moves back, g and G jump to the start and end, /text searches forward, n repeats the search, and q quits. less -N shows line numbers. [2]

more is an older pager that only moves forward; it exists on every UNIX-style system and in Windows.

PowerShell:

```
$ Get-Content notes.txt | Out-Host -Paging
```

PowerShell has no built-in pager of its own. Out-Host -Paging pauses after each screen in the console, and on Windows the more command wraps the Windows more.com pager. Many people install less on Windows through Git for Windows and use it from PowerShell. [3]

Command Prompt:

```
$ more notes.txt
```

Space shows the next screen, Enter the next line, and q quits. [4]

Inspect Before You Open

Four questions about a file can be answered without opening it: what kind of file it is, what its first lines are, what its last lines are, and how big it is.

Bash/zsh:

```
$ file src/main.c
src/main.c: C source, ASCII text
$ head -n 2 notes.txt
alpha
beta gamma
```

```
$ tail -n 1 notes.txt
delta
$ wc notes.txt
 4  6 33 notes.txt
```

file inspects the bytes and reports a type, which is more reliable than the extension. head and tail print the first or last lines; -n sets how many, and the default is ten. wc counts lines, words, and bytes; wc -l counts only lines. [5], [6], [7], [8]

tail -f logfile keeps printing as a file grows, which is the standard way to watch a log. Press Ctrl+C to stop.

PowerShell:

```
$ Get-Content notes.txt -TotalCount 2
$ Get-Content notes.txt -Tail 1
$ Get-Content notes.txt | Measure-Object -Line -Word -Character
```

Get-Content reads a file as a sequence of lines; -TotalCount and -Tail limit which ones. Measure-Object counts what it receives; its character count excludes the line endings, so it is smaller than the wc byte count. Get-Content -Wait follows a growing file the way tail -f does. PowerShell has no file command; Format-Hex from Chapter 5 shows the bytes, and the file's extension is otherwise your only hint. [9], [10]

Command Prompt:

```
$ type notes.txt
$ find /c /v "" notes.txt
```

type prints a file. Command Prompt has no head or tail; more +3 notes.txt starts at line 3, and the find command, which is a text search tool and not the UNIX find, counts lines with /c /v "". For anything more, use PowerShell. [11], [12]

Find Files by Name

Searching by name walks a directory tree and reports entries whose names match a pattern.

Bash/zsh:

```
$ find . -name '*.md'
./docs/todo.md
$ find . -type d
$ find . -type f -name 'main*'
```

find takes a starting directory and then tests. -name matches the filename against a glob, -type f and -type d select files or directories, and tests combine with an implicit "and". Quote the pattern so that the shell does not expand it first; Chapter 2 explained why. [13]

find reports hidden entries too. Add -iname for a case-insensitive match, -newer other.txt for files modified more recently than another, and -mmmin -60 for files modified in the last hour.

PowerShell:

```
$ Get-ChildItem -Recurse -Filter *.md
$ Get-ChildItem -Recurse -Directory
$ Get-ChildItem -Recurse -File -Filter main*
```

-Recurse descends into subdirectories. -Filter matches names with a wildcard, -Directory and -File select the kind of entry, and -Force includes hidden entries. The result is a list of objects, so `Select-Object FullName` or `Where-Object` can refine it; Chapter 7 covers that. [14]

Command Prompt:

```
$ dir /s /b *.md
$ where /r . main*
```

`dir /s` searches subdirectories and `/b` prints bare paths. `where /r` searches a tree for files matching a pattern. [15], [16]

Find Files by Content

Searching by content reads each file and reports the lines that match a pattern. This is the tool for “which file mentions this?”.

grep

`grep` is on every UNIX-style system. It prints matching lines, and with a directory and `-r` it searches recursively:

```
$ grep -n gamma notes.txt
2:beta gamma
$ grep -in gamma notes.txt
2:beta gamma
3:Gamma ray
$ grep -rn docs .
./docs/todo.md:2:done: write docs
$ grep -rl main .
./src/main.c
```

`-n` adds line numbers, `-i` ignores case, `-r` recurses, and `-l` lists only the filenames. `-c` counts matching lines, and `-v` inverts the match to show lines that do not contain the pattern. [17]

ripgrep

ripgrep, run as `rg`, is a separate tool you install with a package manager. It searches recursively by default, skips hidden files and anything listed in `.gitignore`, and is fast on large trees:

```
$ rg gamma
notes.txt
2:beta gamma
$ rg -in gamma
```

Its options resemble `grep`'s where they overlap. `rg --files` lists the files it would search, which shows what its defaults skip; `rg -uu` searches hidden and ignored files too. [18]

PowerShell

`Select-String` is PowerShell's `grep`:

```
$ Select-String -Path notes.txt -Pattern gamma
notes.txt:2:beta gamma
notes.txt:3:Gamma ray
$ Get-ChildItem -Recurse -File | Select-String -Pattern docs
```

It ignores case by default; `-CaseSensitive` changes that. The output is objects with `Path`, `LineNumber`, and `Line` properties, which is why the second example can hand it a list of files from `Get-ChildItem`. [19]

Command Prompt

`findstr` searches files for text:

```
$ findstr /n gamma notes.txt
$ findstr /s /i /n gamma *.txt *.md
```

`/n` adds line numbers, `/i` ignores case, and `/s` searches subdirectories. Its pattern language is its own limited form of regular expression, and it treats a space in the pattern as “or” unless you add `/c:"exact phrase"`. [20]

Globs Are Not Regular Expressions

Two pattern languages appeared in this chapter, and they use some of the same characters for different things.

A **glob** matches filenames. `*` matches any run of characters, `?` matches one character, and `[abc]` matches one of the listed characters. Bash and zsh expand globs before running a command; `find -name`, `Get-ChildItem -Filter`, `dir`, and `-like` in PowerShell interpret them directly. [21], [22]

A **regular expression**, or **regex**, matches text. `.` matches any one character, `*` means “zero or more of the previous item”, `^` and `$` anchor to the start and end of a line, and `[abc]` again matches one listed character. `grep`, `rg`, `Select-String`, and `-match` in PowerShell use regular expressions. [23], [24]

Compare:

Pattern	As a glob	As a regex
<code>*.txt</code>	any name ending in <code>.txt</code>	zero or more of the previous item, then any character, then <code>txt</code>
<code>a.c</code>	exactly the name <code>a.c</code>	<code>a</code> , any character, <code>c</code> : matches <code>abc</code> and <code>a.c</code>

Pattern	As a glob	As a regex
<code>main*</code>	names starting with <code>main</code>	<code>mai</code> followed by zero or more <code>n</code>

So `grep 'a.c' notes.txt` would match a line containing `abc`, and `ls a.c` would list only a file named exactly that. When a search finds too much or too little, ask which language the tool speaks.

Regular expressions have more: `+` for one or more, `?` for optional, `|` for alternatives, and `{2}` for a count, most of which need `grep -E` or `rg`. Chapter 7 uses them in pipelines, and they deserve a book of their own.



Tip: Test a pattern against a small file before pointing it at a whole tree. `grep -n` on `notes.txt` tells you in one line whether the pattern means what you think.

Key Points

- `less` pages through a file; `q` quits, `/` searches. PowerShell uses `Out-Host -Paging` or an installed `less`; Command Prompt uses `more`.
- `file`, `head`, `tail`, and `wc` describe a file without opening it. PowerShell uses `Get-Content -TotalCount`, `-Tail`, and `Measure-Object`.
- `find`, `Get-ChildItem -Recurse`, and `dir /s /b` search by name; quote glob patterns in Bash and `zsh`.
- `grep -rn`, `rg`, `Select-String`, and `findstr` search by content; `rg` skips hidden and ignored files by default.
- Globs match filenames and regular expressions match text; `*` and `.` mean different things in each.

Level Up

`n00bs` finish the chapter here. You can page through a file, describe it without opening it, and find files by name or content. Hackers make searching part of how they think.

Learn `find`'s `-exec` and `xargs`, which run a command on every file found, and explain the risk when filenames contain spaces. Compare with `Get-ChildItem | ForEach-Object` in PowerShell.

Try `fd`, a `find` alternative with simpler syntax and `ripgrep`'s ignore rules, and `fzf` for interactive filtering of any list. Install one and explain what it does that the built-in tool does not.

Explore the tools that understand file structure: `jq` for JSON, `xmllint` for XML, and PowerShell's `ConvertFrom-Json`. Why is `grep` a poor tool for a JSON file?

Look up `bat`, a `cat` replacement with syntax highlighting, and `hexyl` or `xxd` for binary files. Then read the manual for `less` and find three features you did not know it had.

Exercises

1. **What does this do?** Predict the output of `grep -c a notes.txt` and of `grep -v a notes.txt`, then run both.

2. **What does this do?** Predict what `find . -name '*.c' -o -name '*.md'` prints, then check what `-o` means in the manual.
3. **What is the problem?** `find . -name *.md` prints only one file even though there are several, and sometimes prints an error instead. Explain what the shell did before `find` ran.
4. **What is the problem?** `rg TODO` finds nothing in a project, but `grep -r TODO .` finds twelve matches. Give two reasons `ripgrep` might skip those files.
5. **Why is it like this?** `Select-String` ignores case by default and `grep` does not. Explain how each tool's origin explains its default.
6. **Why is it like this?** `wc` and `Measure-Object` report different character counts for the same file. Explain what each one counts.
7. **How do you** find every file under your home directory modified in the last day, in Bash and in PowerShell?
8. **How do you** search a project for lines that start with `import`, and only those lines, in `grep`, `rg`, and `Select-String`?

References

- [1] Microsoft, “Windows commands: cmd.” Available: <https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/cmd>
- [2] Greenwood Software, “Less: Manual page.” Available: <https://github.com/gwsw/less/blob/master/less.nro.VER>
- [3] Microsoft, “Out-host.” Available: <https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.core/out-host>
- [4] Microsoft, “Windows commands: more.” Available: <https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/more>
- [5] Ian F. Darwin and Christos Zoulas, “File(1): Determine file type.” Available: <https://man7.org/linux/man-pages/man1/file.1.html>
- [6] GNU Project, “GNU coreutils manual: Head: Output the first part of files.” Available: https://www.gnu.org/software/coreutils/manual/html_node/head-invocation.html
- [7] GNU Project, “GNU coreutils manual: Tail: Output the last part of files.” Available: https://www.gnu.org/software/coreutils/manual/html_node/tail-invocation.html
- [8] GNU Project, “GNU coreutils manual: Wc: Print newline, word, and byte counts.” Available: https://www.gnu.org/software/coreutils/manual/html_node/wc-invocation.html
- [9] Microsoft, “Get-content.” Available: <https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.management/get-content>
- [10] Microsoft, “Measure-object.” Available: <https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/measure-object>
- [11] Microsoft, “Windows commands: type.” Available: <https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/type>
- [12] Microsoft, “Windows commands: find.” Available: <https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/find>
- [13] GNU Project, “GNU findutils manual: Finding files.” Available: https://www.gnu.org/software/findutils/manual/html_mono/find.html
- [14] Microsoft, “Get-ChildItem.” Available: <https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.management/get-childitem>
- [15] Microsoft, “Windows commands: dir.” Available: <https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/dir>
- [16] Microsoft, “Windows commands: where.” Available: <https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/where>

- [17] GNU Project, “GNU grep manual.” Available: <https://www.gnu.org/software/grep/manual/grep.html>
- [18] Andrew Gallant, “Ripgrep user guide.” Available: <https://github.com/BurntSushi/ripgrep/blob/master/GUIDE.md>
- [19] Microsoft, “Select-string.” Available: <https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/select-string>
- [20] Microsoft, “Windows commands: findstr.” Available: <https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/findstr>
- [21] GNU Project, “Bash reference manual: Filename expansion.” Available: https://www.gnu.org/software/bash/manual/html_node/Filename-Expansion.html
- [22] Microsoft, “About_wildcards.” Available: https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_wildcards
- [23] GNU Project, “GNU grep manual: Regular expressions.” Available: https://www.gnu.org/software/grep/manual/html_node/Regular-Expressions.html
- [24] Microsoft, “About_regular_expressions.” Available: https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_regular_expressions