The File System & the Shell

Modern Plain Text Computing

Week 03b

Kieran Healy

kieran.healy@duke.edu

September 2025

Files

Files

A file is just a stream of bytes, or data, some sort of resource that a program can read or interact with.

Files have a location in the file system.

In the UNIX way of thinking, “Everything is a file”

That is, lots of things that are not normally thought of as files (such as printers, or terminal screens, or connections to other computers) can be thought of as living in a named place somewhere in the filesystem.

The basic set of UNIX utilities can be thought of as tools that accept “files” (as a standard stream of input data), perform some specific action on them (read, print, move, copy, delete, count lines, find text, whatever) and then return a standard stream of output data that can be sent somewhere, e.g. to a terminal display, or used as input to another command, or become a file of its own.

File system hierarchy

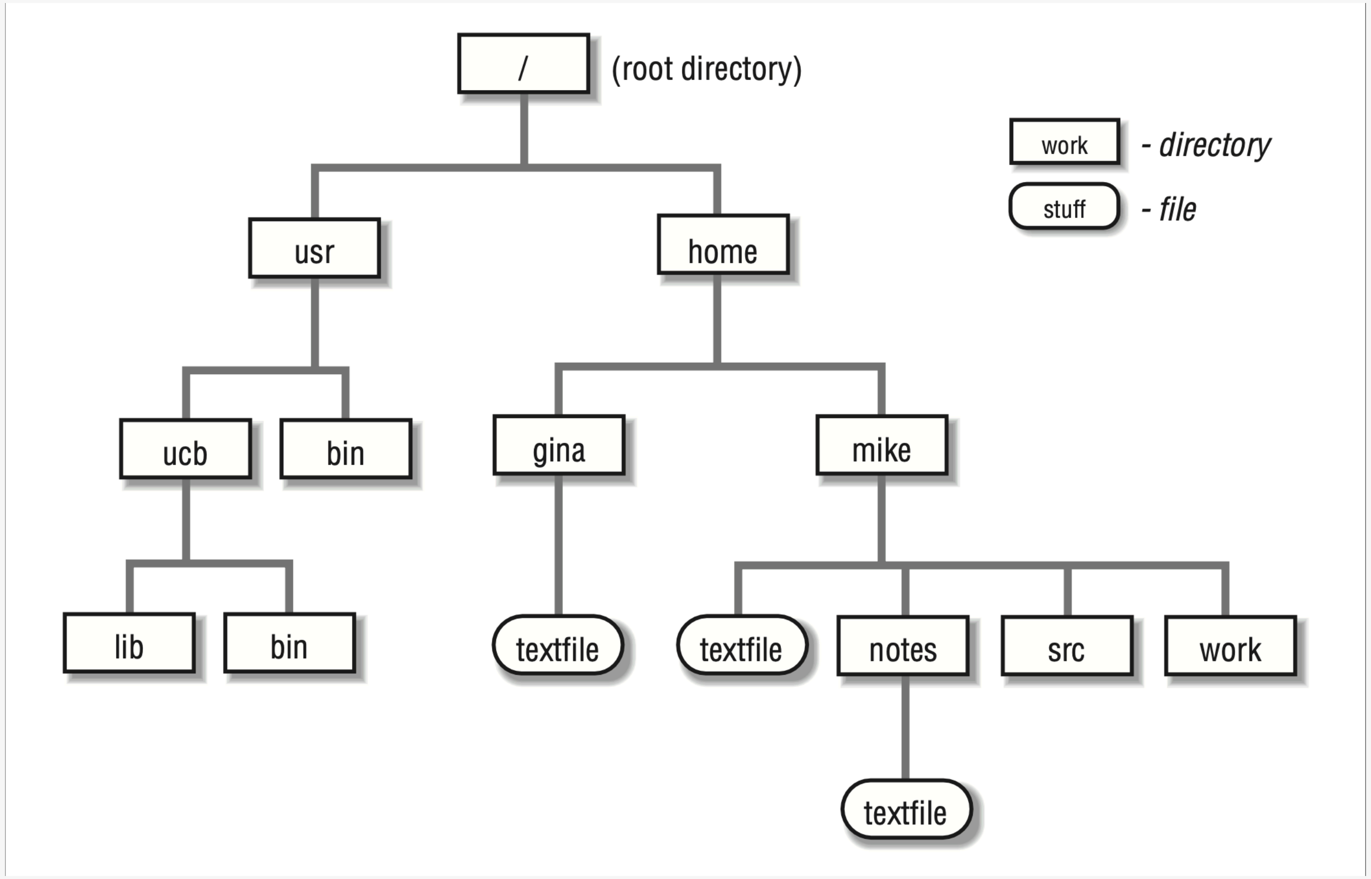


Illustration: Shelley Powers et al. *Unix Power Tools*, 3rd ed. (Sebastopol, CA: O'Reilly Media, 2002), 23.

Path conventions

`/` represents a division in the file hierarchy. You can think of it as a branch point on a tree, or as a new level of nesting in a series of boxes, or as the action “Go inside” or “Enter”.

On a Unix-like system, a full path to a file looks like this:

```
/Users/kjhealy/Documents/courses/mptc/slides/01b-slides.qmd
```

“Go inside the ‘`Users`’ folder, then inside the ‘`kjhealy`’ folder, then inside ‘`Documents`’ then inside ‘`courses`’ then ‘`mptc`’ then ‘`slides`’ and you will find the file `01b-slides.qmd`.”

Standard Unix locations

`/` : root. Everything lives inside or under the root.

`/bin/` : For *binaries*. Core user executable programs and tools.

`/sbin/` : System binaries. Essential executables for the super user (who is also called `root`)

`/lib/` : Support files for executables.

`/usr/` : Conventionally, stuff installed "locally" for users in addition to the core system. Will contain its own `bin/` and `lib/` subdirs.

`/usr/local` : Files that the local user has compiled or installed

`/opt/` : Like `/usr/`, another place for locally installed software to go.

Standard Unix locations

These locations get mapped together in the `$PATH`, which is an *environment variable* that tells the system where executables can be found.

```
❯ echo $PATH
/home/kjhealy/bin:/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games:/snap/bin
```

Delimited by `:` and searched in order from left to right.

To learn where a command is being executed from, use `which`

```
❯ which R
/usr/local/bin/R
```

Standard Unix locations

`/` : root. Everything lives inside or under the root.

`/bin/` : For *binaries*. Core user executable programs and tools.

`/sbin/` : System binaries. Essential executables for the super user (who is also called `root`)

`/lib/` : Support files for executables.

`/usr/` : Conventionally, stuff installed “locally” for users in addition to the core system. Will contain its own `bin/` and `lib/` subdirs.

`/usr/local` : Files that the local user has compiled or installed

`/opt/` : Like `/usr/`, another place for locally installed software to go.

`/etc/` : Editable text configuration. Config files often go here.

Standard Unix locations

`/home/` or `/Users/` : Where the accounts of individual system users live, like `/Users/kjhealy` or `/home/kjhealy`

```
❯ pwd
/home/kjhealy
❯  ls
bin  certbot.log  logrotate.conf  old  projects  public  staging
```

All of this is a matter of more or less established convention that varies by particular operating systems. E.g. on most Linux systems, individual user directories live in `/home`. On macOS they live in `/Users`. Windows is different again (and uses `\` for file paths rather than `/`.)

File system hierarchy

An edited version of the **root**, /, or top of my Mac’s file system tree:

```
├── Applications
├── bin
├── cores
├── dev
├── etc -> private/etc
├── home -> /System/Volumes/Data/home
├── Library
├── opt
│   ├── homebrew
├── private
│   ├── etc
│   ├── tftpboot
│   ├── tmp
│   └── var
├── sbin
├── System
├── tmp -> private/tmp
├── Users
│   ├── kjhealy
│   └── Shared
├── usr
│   ├── bin
│   ├── lib
│   ├── libexec
│   ├── local
│   ├── sbin
│   ├── share
│   ├── standalone
├── var -> private/var
└── Volumes
```

File system hierarchy

An edited version of the **User** or **home** tree, i.e. everyting inside `/Users/kjhealy` on my Mac:

```
├── Applications
├── bin
├── Box
├── Creative Cloud Files
├── Desktop
├── Documents
│   ├── bibs → /Users/kjhealy/Library/texmf/bibtex/bib
│   ├── bookdown
│   ├── comments
│   ├── completed
│   ├── courses
│   ├── data
│   ├── letters
│   ├── misc
│   ├── nonsense
│   ├── ordinal-society
│   ├── papers
│   ├── sites
│   ├── source
│   ├── talks
│   ├── teaching
│   ├── templates
│   ├── vita
├── Downloads
├── Dropbox
├── Library
├── Movies
├── Music
├── Pictures
├── Public
├── scratch
├── tmp
└── Zotero
```

Local and Remote Files

Local Files

So far we've been working with files on our own computer. These local files live somewhere in the file system on our own computer.

We're also mostly going to be confining ourselves, in any particular project, to files that are in or under our project directory. Like in the `mptc_oecd` project. While we're in an R session and working with `mptc_oecd`, we think of the project directory as our working directory, and the top of the project directory as the root of our little system of files and folders.

So `data-raw/countries_iso3.tsv` is a file that lives in the `data-raw` folder inside the project directory. `mptc_oecd.qmd` lives at the top level of the project directory.

But files can also be located remotely, on other computers, and we can access them over the internet or a network.

Remote Files: URLs

A URL or Uniform Resource Locator is a kind of address that locates a resource on the internet. It is, in effect, a path to a file that lives on another computer somewhere, one that is accessible by us (or by the public in general).

Remember, there’s no such thing as The Cloud, it’s just Someone Else’s Computer

Remote Files: URLs

A URL to the top or root level of a webserver looks like this:
https://kieranhealy.org/

A URL to a folder inside a webserver looks like this:
https://kieranhealy.org/publications/tos/

A URL to a specific file inside a webserver looks like this:
https://kieranhealy.org/files/misc/tos_cover_1024.png

Remote Files: URLs

As you can see, a URL is just a file path, apart from the `https://kieranhealy.org` bit at the start that tells your computer which webserver to connect to.

You might wonder why paths to folders, like `https://kieranhealy.org/publications/` appear in your browser as a web page. This is because the site is set up to serve a default file, usually called `index.html`, when you ask for a folder.

Can we get remote files via the Terminal or command line? Of course we can.

Curl

The address https://kjhealy.co/mptc/ shows a directory with some files in it. One is called `mortality.txt`. We use the `curl` command:

```
curl https://kjhealy.co/mptc/mortality.txt
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed

  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
100 16160  100 16160    0     0  33681      0 --:--:-- --:--:-- --:--:-- 33666
England and Wales, Total Population, Death rates (period 1x1),  Last modified: 02 Apr 2018;  Methods Protocol: v6
(2017)

  Year          Age             Female            Male            Total
  1841           0             0.136067        0.169189        0.152777
  1841           1             0.059577        0.063208        0.061386
  1841           2             0.036406        0.036976        0.036689
  1841           3             0.024913        0.026055        0.025480
  1841           4             0.018457        0.019089        0.018772
  1841           5             0.013967        0.014279        0.014123
  1841           6             0.010870        0.011210        0.011040
  1841           7             0.008591        0.008985        0.008788
  1841           8             0.006860        0.007246        0.007053
```

The contents of the file just appear in the terminal window.

Curl

We can redirect it to a file instead:

```
mkdir tmp
curl https://kjhealy.co/mptc/mortality.txt > tmp/mortality.txt
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed

  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
100 16160  100 16160    0     0  34155      0 --:--:-- --:--:-- --:--:-- 34164
```

```
ls -l tmp/
```

```
total 32
-rw-r--r--@ 1 kjhealy  staff  16160 Sep 23 13:57 mortality.txt
```

```
head tmp/mortality.txt
```

```
England and Wales, Total Population, Death rates (period 1x1),  Last modified: 02 Apr 2018;  Methods Protocol: v6
(2017)

  Year          Age             Female            Male           Total
  1841           0             0.136067        0.169189        0.152777
  1841           1             0.059577        0.063208        0.061386
  1841           2             0.036406        0.036976        0.036689
  1841           3             0.024913        0.026055        0.025480
  1841           4             0.018457        0.019089        0.018772
  1841           5             0.013967        0.014279        0.014123
  1841           6             0.010870        0.011210        0.011040
```

The Shell

What is it?

There are many shells

A command interpreter

```
echo "Hello there"
```

```
Hello there
```

Getting around the file system

Who and where

Who am I?

```
whoami
```

```
kjhealy
```

Where am I?

```
# Print working directory
pwd
```

```
/Users/kjhealy/Documents/courses/mptc
```

Listing files

What is in here?

```
# List files
ls
```

```
_extensions
_freeze
_motivation.qmd
_quarto.yml
_site
_targets
_targets.R
_variables.yml
_weekly-schedule.qmd
00_dummy_files
about
assets
assignment
avhrr
content
data
deploy.sh
example
files
```

Navigating the tree

Who am I?

```
whoami
```

```
kjhealy
```

Where am I?

```
pwd
```

```
/Users/kjhealy/Documents/courses/mptc
```

What is my purpose in life?

```
(Unix can't help you here)
```

Navigating the tree

```
cd files
ls
cd ..
```

```
01_1890_hollerith_codes.png
01_apple_macintosh.png
01_bryant_hard_drive.png
bib
examples
fars_spreadsheet_raw.png
misc
schedule.ics
scripts
```

Navigating the tree

```
ls -l
```

```
total 936
drwxr-xr-x   3 kjhealy  staff      96 Jan  9  2024 _extensions
drwxr-xr-x@  8 kjhealy  staff     256 Sep 23 13:56 _freeze
-rw-r--r--@  1 kjhealy  staff    3757 Aug 17 10:36 _motivation.qmd
-rw-r--r--@  1 kjhealy  staff    3656 Sep 23 13:11 _quarto.yml
drwxr-xr-x@  2 kjhealy  staff      64 Sep 23 13:56 _site
drwxr-xr-x@  8 kjhealy  staff     256 Sep 23 13:56 _targets
-rw-r--r--@  1 kjhealy  staff    7552 Sep 23 13:50 _targets.R
-rw-r--r--@  1 kjhealy  staff    1009 Sep 23 13:33 _variables.yml
-rw-r--r--@  1 kjhealy  staff     974 Aug 16 21:28 _weekly-schedule.qmd
drwxr-xr-x@  3 kjhealy  staff      96 Sep 23 13:56 00_dummy_files
drwxr-xr-x@  4 kjhealy  staff     128 Sep 23 13:56 about
drwxr-xr-x@ 18 kjhealy  staff     576 Aug 25 05:58 assets
drwxr-xr-x@ 17 kjhealy  staff     544 Sep 23 13:56 assignment
lrwxr-xr-x   1 kjhealy  staff     135 Nov  5  2024 avhrr ->
/Users/kjhealy/Documents/data/misc/noaa_ncei/raw/www.ncei.noaa.gov/data/sea-surface-temperature-optimum-
interpolation/v2.1/access/avhrr
drwxr-xr-x@ 14 kjhealy  staff     448 Sep 23 13:56 content
drwxr-xr-x@  6 kjhealy  staff     192 Sep 23 07:42 data
```

Note the idea of commands having options, or *switches*.

Navigating the tree

```
ls /
```

```
Applications
bin
cores
dev
etc
home
Library
opt
private
sbin
System
tmp
Users
usr
var
Volumes
```

Path rules

If the path name begins with `/`, it is an *absolute* path, starting from the filesystem root.

If the path name begins with `~`, it will usually be expanded into an absolute path name starting at your home directory (`~`).

Path rules

If the pathname does not begin with a `/` or `~` then the path name is relative to the current directory.

Two relative special cases use entries that are in every Unix directory:

a. If the path name begins with `./`, the path is relative to the current directory, e.g., `./textfile`, though this can also execute the file if it is given executable file permissions.

b. If the path name begins with `../`, the path is relative to the parent of the current directory. For example, if your current directory is `/Users/kjhealy/Documents/papers` then `../data` means `/Users/kjhealy/Documents/data`

File permissions

Who is using this file system anyway?

```
drwxr-xr-x@  8 kjhealy  staff    256 Aug 15 16:35 R
-rw-r--r--@  1 kjhealy  staff   1210 Aug 15 20:29 README.md
```

Unix derives from a world there there are multiple users and groups of users who are all using slices (in terms of processor time and available permanent storage) of a large central computer.

File permissions

```
drwxr-xr-x@  8 kjhealy  staff    256 Aug 15 16:35 R
-rw-r--r--@  1 kjhealy  staff   1210 Aug 15 20:29 README.md
```

In Unix systems there are three kinds of owner: the *user* (here `kjhealy`), the *group* (here `staff`), and *others* or other users on the system.

File permissions

```
drwxr-xr-x@  8 kjhealy  staff    256 Aug 15 16:35 R
-rw-r--r--@  1 kjhealy  staff   1210 Aug 15 20:29 README.md
```

Three things you can do to a file:

read

write

execute

> For files, “read” means *open*; “write” means *edit, save, or delete*; “execute” means *run* if it’s an application or script.
>
> For directories, “read” means *list contents* with ls, “write’ means *create, delete, or rename*;”execute” means access or enter using `cd`

File permissions

```
❯ ls -l README.md
-rw-r--r--@  1 kjhealy  staff   1210 Aug 15 20:29 README.md
```

These permissions say `rw-r--r--` or

- The *user* can `rw-` read and write this file
- The *group* can `r--` read this file
- The *world* can `r--` read this file

Executable permissions are irrelevant here because it’s a text file.

File permissions

	user	group	all
	r w x	*r w x*	*r w x*
Abbreviation	`rw-`	`r--`	`r--`
As bits	110	100	100
As decimal	6	4	4

We change file permissions with the `chmod` command. So e.g. `chmod 644 README.md` means “change the permissions to `rw-r--r--`”.

A Tree

```
├── schedule
├── staging
│   ├── example
│   ├── content
│   ├── assignment
│   ├── slides
├── example
│   ├── 04-example-ggplot_files
├── projects
│   ├── 05-problem-set
├── R
├── content
├── assignment
├── html
│   ├── fonts
├── site_libs
│   ├── revealjs
│   ├── bootstrap
│   ├── quarto-html
```

Changing directories

```
## Change directory and list files
cd files
ls
cd ../slides
```

```
01_1890_hollerith_codes.png
01_apple_macintosh.png
01_bryant_hard_drive.png
bib
examples
fars_spreadsheet_raw.png
misc
schedule.ics
scripts
```

Some shell tools

Example files

Project at: https://github.com/kjhealy/mptc_text_examples

Download the zip file, for now via GitHub, and unzip it somewhere you can find it. Or, better, practice your `curl` skills and download it from `khealy.co`, like this:

```
# This time we use -o to specify the output file name, rather than using > to redirect STDOUT.
curl https://kjhealy.co/mptc/mptc_text_examples.zip -o mptc_text_examples.zip

# Once you've downloaded it, unzip it:
unzip mptc_text_examples.zip
```

What are we working with

```
ls files/examples/
```

```
_make-example
01_mptc_oecd_nocode.pdf
01_mptc_oecd_withcode.pdf
alice_in_wonderland.txt
alice_noboiler.txt
apple_mobility_daily_2021-04-12.csv
ascii_table.xlsx
bashrc.txt
basics.txt
census_edage.csv
congress
continent_sizes.csv
continent_tab.csv
continent_tab.tsv
countries_iso3.csv
countries.csv
country_iso3.tsv
country_tab.csv
country_tab.tsv
```

These files are in my course site project, so your file path will be different! It will be wherever you unzipped the files and the folder will be called `mptc_text_examples` if you got it via `curl`, or `mptc_text_examples_main` if you got it from GitHub.

wc, cat, head, and tail

```
wc files/examples/alice_in_wonderland.txt
```

```
    3761   29564  174392 files/examples/alice_in_wonderland.txt
```

We can ask for a count of lines only:

```
wc -l files/examples/alice_in_wonderland.txt
```

```
    3761 files/examples/alice_in_wonderland.txt
```

wc, cat, head, and tail

cat *concatenates* and *prints* the files given to it.

```
cat files/examples/jabberwocky.txt
```

```
’Twas brillig, and the slithy toves
      Did gyre and gimble in the wabe:
All mimsy were the borogoves,
      And the mome raths outgrabe.

“Beware the Jabberwock, my son!
      The jaws that bite, the claws that catch!
Beware the Jubjub bird, and shun
      The frumious Bandersnatch!”

He took his vorpal sword in hand;
      Long time the manxome foe he sought—
So rested he by the Tumtum tree
      And stood awhile in thought.

And, as in uffish thought he stood,
      The Jabberwock, with eyes of flame,
Came whiffling through the tulgey wood,
      And burbled as it came!
```

wc, cat, head, and tail

The top:

```
head files/examples/alice_in_wonderland.txt
```

```
The Project Gutenberg eBook of Alice's Adventures in Wonderland

This ebook is for the use of anyone anywhere in the United States and
most other parts of the world at no cost and with almost no restrictions
whatsoever. You may copy it, give it away or re-use it under the terms
of the Project Gutenberg License included with this ebook or online
at www.gutenberg.org. If you are not located in the United States,
you will have to check the laws of the country where you are located
before using this eBook.
```

The bottom:

```
tail files/examples/alice_in_wonderland.txt
```

```

Most people start at our website which has the main PG search
facility: www.gutenberg.org.

This website includes information about Project Gutenberg™,
including how to make donations to the Project Gutenberg Literary
Archive Foundation, how to help produce our new eBooks, and how to
subscribe to our email newsletter to hear about new eBooks.
```

wc, cat, head, and tail

There are 29 lines of boilerplate at the start of the book:

```
head -n 29 files/examples/alice_in_wonderland.txt
```

```
The Project Gutenberg eBook of Alice's Adventures in Wonderland

This ebook is for the use of anyone anywhere in the United States and
most other parts of the world at no cost and with almost no restrictions
whatsoever. You may copy it, give it away or re-use it under the terms
of the Project Gutenberg License included with this ebook or online
at www.gutenberg.org. If you are not located in the United States,
you will have to check the laws of the country where you are located
before using this eBook.

Title: Alice's Adventures in Wonderland


Author: Lewis Carroll

Release date: June 27, 2008 [eBook #11]
                Most recently updated: March 30, 2021

Language: English
```

wc, cat, head, and tail

And 351 at the end:

```
tail -n 351 files/examples/alice_in_wonderland.txt | head -n 20
```

```
            *** END OF THE PROJECT GUTENBERG EBOOK ALICE'S ADVENTURES IN WONDERLAND ***




Updated editions will replace the previous one—the old editions will
be renamed.

Creating the works from print editions not protected by U.S. copyright
law means that no one owns a United States copyright in these works,
so the Foundation (and you!) can copy and distribute it in the United
States without permission and without paying copyright
royalties. Special rules, set forth in the General Terms of Use part
of this license, apply to copying and distributing Project
Gutenberg™ electronic works to protect the PROJECT GUTENBERG™
concept and trademark. Project Gutenberg is a registered trademark,
and may not be used if you charge for an eBook, except by following
the terms of the trademark license, including paying royalties for use
of the Project Gutenberg trademark. If you do not charge anything for
```

wc, cat, head, and tail

We can use tail to skip the boilerplate at the top:

```
tail -n +29 files/examples/alice_in_wonderland.txt | head
```

```
Alice’s Adventures in Wonderland

by Lewis Carroll

THE MILLENNIUM FULCRUM EDITION 3.0

Contents

 CHAPTER I.     Down the Rabbit-Hole
```

`wc`, `cat`, `head`, and `tail`

The shell can be treated like a programming language. That is, it has variables and also flow control (loops, if-then-else, etc).

We can use some shell variables along with `tail` twice to skip the boilerplate at the top and bottom, and put the result into a file of its own using `>` to redirect the output from `STDOUT`:

```
# This sets HEADSKIP to 29 and ENDSKIP to 351;
# We can refer to them with $HEADSKIP and $ENDSKIP
HEADSKIP=29
ENDSKIP=351

# The backticks ` ` here mean "Evaluate this command"; then put the result in a variable
BOOKLINES=`cat files/examples/alice_in_wonderland.txt| wc -l | tr ' ' '\n' | tail -1`

# This line does the arithmetic using expr and makes the result a variable
GOODLINES=$(expr $BOOKLINES - $HEADSKIP - $ENDSKIP)

# Now we use $HEADKSIP and $GOODLINES and create a new file
tail -n +$HEADSKIP files/examples/alice_in_wonderland.txt |
  head -n $GOODLINES > files/examples/alice_noboiler.txt
```

wc, cat, head, and tail

Now our wc will be different:

```
wc files/examples/alice_in_wonderland.txt

wc files/examples/alice_noboiler.txt
```

```
    3761   29564  174392 files/examples/alice_in_wonderland.txt
    3381   26524  154465 files/examples/alice_noboiler.txt
```

uniq, sort, and cut

A data file:

```
head files/examples/countries.csv
```

```
cname,iso3,iso2,continent
Afghanistan,AFG,AF,Asia
Algeria,DZA,DZ,Africa
Armenia,ARM,AM,Asia
Australia,AUS,AU,Oceania
Austria,AUT,AT,Europe
Azerbaijan,AZE,AZ,Asia
Bahrain,BHR,BH,Asia
Belarus,BLR,BY,Europe
Belgium,BEL,BE,Europe
```

How many lines?

```
wc -l files/examples/countries.csv
```

```
     214 files/examples/countries.csv
```

How many unique lines?

```
uniq files/examples/countries.csv | wc -l
```

```
     214
```

uniq, sort, and cut

```
# Omit the header line
tail -n +2 files/examples/countries.csv | sort -r | head
```

```
Zimbabwe,ZWE,ZW,Africa
Zambia,ZMB,ZM,Africa
Yemen,YEM,YE,Asia
Western Sahara,ESH,EH,Africa
Wallis and Futuna,WLF,WF,Oceania
Viet Nam,VNM,VN,Asia
Vanuatu,VUT,VU,Oceania
Uzbekistan,UZB,UZ,Asia
Uruguay,URY,UY,South America
United States,USA,US,North America
```

`uniq`, `sort`, and `cut`

This doesn’t *quite* work because of the way the data is coded:

```
tail -n +2 files/examples/countries.csv | sort -t , -k4 -k1
```

```
Algeria,DZA,DZ,Africa
Angola,AGO,AO,Africa
Benin,BEN,BJ,Africa
Botswana,BWA,BW,Africa
Burkina Faso,BFA,BF,Africa
Burundi,BDI,BI,Africa
Cabo Verde,CPV,CV,Africa
Cameroon,CMR,CM,Africa
Central African Republic,CAF,CF,Africa
Chad,TCD,TD,Africa
Comoros,COM,KM,Africa
Congo,COG,CG,Africa
Côte d'Ivoire,CIV,CI,Africa
Djibouti,DJI,DJ,Africa
Egypt,EGY,EG,Africa
Equatorial Guinea,GNQ,GQ,Africa
Eritrea,ERI,ER,Africa
Ethiopia,ETH,ET,Africa
Gabon,GAB,GA,Africa
```

uniq, sort, and cut

cut slices out columns defined by a delimiter (by default \t or tab)

```
cut -d , -f 2,4 files/examples/countries.csv
```

```
iso3,continent
AFG,Asia
DZA,Africa
ARM,Asia
AUS,Oceania
AUT,Europe
AZE,Asia
BHR,Asia
BLR,Europe
BEL,Europe
BRA,South America
KHM,Asia
CAN,North America
CHN,Asia
HRV,Europe
CZE,Europe
DNK,Europe
DOM,North America
ECU,South America
```

Again in this case it doesn’t quite behave as you might think!

Finding files and finding text

find

find is for locating files and directories by name:

```
# Everything in the `files/` subdirectory
find files
```

```
files
files/misc
files/misc/home-tree.txt
files/misc/root-tree.txt
files/.DS_Store
files/schedule.ics
files/01_apple_macintosh.png
files/01_bryant_hard_drive.png
files/fars_spreadsheet_raw.png
files/examples
files/examples/country_iso3.tsv
files/examples/jabberwocky.txt
files/examples/country_tab.csv
files/examples/ulysses.txt
files/examples/_make-example
files/examples/_make-example/mypaper.md
files/examples/_make-example/fig1.r
files/examples/_make-example/Makefile
files/examples/_make-example/README.md
```

find

We can use *globbing* (or *wildcards*) to narrow our search:

```
# Everything underneath the `files/` subdirectory
# whose name ends in `.csl`
find files -name "*.csl"
```

```
files/bib/samplesyllabus.csl
files/bib/american-political-science-association.csl
files/bib/chicago-fullnote-bibliography-no-bib.csl
files/bib/chicago-fullnote-bibliography.csl
files/bib/chicago-syllabus-no-bib.csl
files/bib/apa.csl
files/bib/chicago-author-date.csl
files/bib/chicago-note-bibliography.csl
```

`find`

Here we use the `.` to mean “Search in the current folder”

```
find . -name "*.xlsx"
```

```
./files/examples/symptoms.xlsx
./files/examples/fars_crash_report.xlsx
./files/examples/ascii_table.xlsx
./data/schedule.xlsx
./data/data_sources.xlsx
```

find

The -exec option lets us do things with each result.

The {} expands to each found file in turn.

Here we use echo to see what the rm (remove) command would do.

The quoted semicolon ";" or \; is required to end the line

```
find files -name "*.png" -exec echo rm {} ";"
```

```
rm files/01_apple_macintosh.png
rm files/01_bryant_hard_drive.png
rm files/fars_spreadsheet_raw.png
rm files/01_1890_hollerith_codes.png
```

If we omitted the echo here the found files really would be deleted one at a time.

find

We can also use xargs to act on search results:

```
# Everything underneath the `files/` subdirectory
# whose name ends in `.png`
find files -name "*.png"
```

```
files/01_apple_macintosh.png
files/01_bryant_hard_drive.png
files/fars_spreadsheet_raw.png
files/01_1890_hollerith_codes.png
```

Convert all these png files to jpg:

```
# Convert everything underneath the `files/` subdirectory
# whose name ends in `.png` to `.jpg` format, keeping the original files.
find files -name '*.png' -print0 | xargs -0 -r mogrify -format jpg
```

find

Check:

```
find files -name '*.png'
find files -name '*.jpg'
```

```
files/01_apple_macintosh.png
files/01_bryant_hard_drive.png
files/fars_spreadsheet_raw.png
files/01_1890_hollerith_codes.png
files/01_apple_macintosh.jpg
files/01_bryant_hard_drive.jpg
files/fars_spreadsheet_raw.jpg
files/01_1890_hollerith_codes.jpg
```

Delete them (with another method of deletion):

```
find files  -name '*.jpg' -type f -delete
```

Perspective

Obviously you will not be doing this sort of thing every day of the week. But you may well want to programmatically rename, move, convert, or otherwise maniplate files in batches from time to time. Especially if there are a lot of them, the shell can help you.

Naming things

Naming files

The better your names for things, the easier they will be to find (and programmatically work with)

In civilized operating systems, names containing spaces and special characters (such as `? ! , . # $ * <space>` and the like) are not a problem.

However, the more you work programatically, the more you will want to avoid them.

Jenny Bryan's 5 minute Normconf talk is a good overview of good habits

Naming files

Names should tell you something about what the file is

Names should avoid spaces and punctuation

Names should follow some reasonable convention

Names with numbers should sort in useful ways

Names should not be used to track the versions of files

Naming files

Find all files in or below the project directory that end in `.qmd`:

```
find . -name "*.qmd"
```

```
./schedule/index.qmd
./staging/example/04-example.qmd
./staging/example/11-example.qmd
./staging/example/08-example.qmd
./staging/example/07-example.qmd
./staging/example/09-example.qmd
./staging/example/05-example.qmd
./staging/example/06-example.qmd
./staging/example/03-example.qmd
./staging/content/09-content.qmd
./staging/content/10-content.qmd
./staging/content/06-content.qmd
./staging/content/03-content.qmd
./staging/content/11-content.qmd
./staging/content/08-content.qmd
./staging/content/07-content.qmd
./staging/content/12-content.qmd
./staging/assignment/04-assignment.qmd
./staging/assignment/03-assignment.qmd
```

Naming files

Find all files in or below the current directory that start with two characters followed by `-example` and end with any other number of characters:

```
find . -name "??-example*"
```

```
./staging/example/04-example.qmd
./staging/example/11-example.qmd
./staging/example/08-example.qmd
./staging/example/07-example.qmd
./staging/example/09-example.qmd
./staging/example/05-example.qmd
./staging/example/06-example.qmd
./staging/example/03-example.qmd
./example/04-example-ggplot.html
./example/01-example-oecd.html
./example/04-example-ggplot.qmd
./example/03-example-shell.qmd
./example/01-example-oecd.qmd
./example/05-example-dplyr.qmd
./example/05-example-dplyr.html
./example/04-example-ggplot_files
./example/03-example-shell.html
./_freeze/example/01-example-oecd
./_freeze/example/05-example-dplyr
```

Sort order

```
mkdir tmp
touch tmp/{1..15}.txt
```

See how these sort:

```
ls tmp/
```

```
1.txt
10.txt
11.txt
12.txt
13.txt
14.txt
15.txt
2.txt
3.txt
4.txt
5.txt
6.txt
7.txt
8.txt
9.txt
```

Not what we want.

Sort order

```
rm -f tmp/*.txt
touch tmp/{01..15}.txt
ls tmp/
```

```
01.txt
02.txt
03.txt
04.txt
05.txt
06.txt
07.txt
08.txt
09.txt
10.txt
11.txt
12.txt
13.txt
14.txt
15.txt
```

Sort order

```
rm -f tmp/*.txt
touch tmp/{a..d}{01..03}.txt
ls -l tmp/
rm -rf tmp/
rm -rf ../tmp/
```

```
total 0
-rw-r--r--@ 1 kjhealy  staff  0 Sep 23 13:57 a01.txt
-rw-r--r--@ 1 kjhealy  staff  0 Sep 23 13:57 a02.txt
-rw-r--r--@ 1 kjhealy  staff  0 Sep 23 13:57 a03.txt
-rw-r--r--@ 1 kjhealy  staff  0 Sep 23 13:57 b01.txt
-rw-r--r--@ 1 kjhealy  staff  0 Sep 23 13:57 b02.txt
-rw-r--r--@ 1 kjhealy  staff  0 Sep 23 13:57 b03.txt
-rw-r--r--@ 1 kjhealy  staff  0 Sep 23 13:57 c01.txt
-rw-r--r--@ 1 kjhealy  staff  0 Sep 23 13:57 c02.txt
-rw-r--r--@ 1 kjhealy  staff  0 Sep 23 13:57 c03.txt
-rw-r--r--@ 1 kjhealy  staff  0 Sep 23 13:57 d01.txt
-rw-r--r--@ 1 kjhealy  staff  0 Sep 23 13:57 d02.txt
-rw-r--r--@ 1 kjhealy  staff  0 Sep 23 13:57 d03.txt
```

In general keep your names lower-case.

Dates

Use the one true YMD format, ISO 8601:

YYYY-MM-DD

Naming files

Be consistent in your use of naming conventions

No need to get too clever, but …

```
data_clean/
data_raw/
docs/
figures/
R/01_clean-data.R
R/02_process-data.R
R/03_descriptive-figs-tables.R
R/04_brms-model.R
paper/
README.md
```

Unix naming conventions

Dotfiles and underscores

```
ls -l
```

```
total 936
drwxr-xr-x   3 kjhealy  staff      96 Jan  9  2024 _extensions
drwxr-xr-x@  8 kjhealy  staff     256 Sep 23 13:56 _freeze
-rw-r--r--@  1 kjhealy  staff    3757 Aug 17 10:36 _motivation.qmd
-rw-r--r--@  1 kjhealy  staff    3656 Sep 23 13:11 _quarto.yml
drwxr-xr-x@  2 kjhealy  staff      64 Sep 23 13:56 _site
drwxr-xr-x@  8 kjhealy  staff     256 Sep 23 13:56 _targets
-rw-r--r--@  1 kjhealy  staff    7552 Sep 23 13:50 _targets.R
-rw-r--r--@  1 kjhealy  staff    1009 Sep 23 13:33 _variables.yml
-rw-r--r--@  1 kjhealy  staff     974 Aug 16 21:28 _weekly-schedule.qmd
drwxr-xr-x@  3 kjhealy  staff      96 Sep 23 13:56 00_dummy_files
drwxr-xr-x@  4 kjhealy  staff     128 Sep 23 13:56 about
drwxr-xr-x@ 18 kjhealy  staff     576 Aug 25 05:58 assets
drwxr-xr-x@ 17 kjhealy  staff     544 Sep 23 13:56 assignment
lrwxr-xr-x   1 kjhealy  staff     135 Nov  5  2024 avhrr ->
/Users/kjhealy/Documents/data/misc/noaa_ncei/raw/www.ncei.noaa.gov/data/sea-surface-temperature-optimum-
interpolation/v2.1/access/avhrr
drwxr-xr-x@ 14 kjhealy  staff     448 Sep 23 13:56 content
drwxr-xr-x@  6 kjhealy  staff     192 Sep 23 07:42 data
```

Unix naming conventions

```
ls -la
```

```
total 1032
drwxr-xr-x   3 kjhealy  staff     96 Jan  9  2024 _extensions
drwxr-xr-x@  8 kjhealy  staff    256 Sep 23 13:56 _freeze
-rw-r--r--@  1 kjhealy  staff   3757 Aug 17 10:36 _motivation.qmd
-rw-r--r--@  1 kjhealy  staff   3656 Sep 23 13:11 _quarto.yml
drwxr-xr-x@  2 kjhealy  staff     64 Sep 23 13:56 _site
drwxr-xr-x@  8 kjhealy  staff    256 Sep 23 13:56 _targets
-rw-r--r--@  1 kjhealy  staff   7552 Sep 23 13:50 _targets.R
-rw-r--r--@  1 kjhealy  staff   1009 Sep 23 13:33 _variables.yml
-rw-r--r--@  1 kjhealy  staff    974 Aug 16 21:28 _weekly-schedule.qmd
drwxr-xr-x@ 48 kjhealy  staff   1536 Sep 23 13:57 .
drwxr-xr-x@ 38 kjhealy  staff   1216 Sep 16 08:51 ..
-rw-r--r--@  1 kjhealy  staff  10244 Sep 22 08:48 .DS_Store
drwxr-xr-x@ 16 kjhealy  staff    512 Sep 23 13:55 .git
-rw-r--r--@  1 kjhealy  staff    383 Aug 19 09:19 .gitignore
-rw-r--r--   1 kjhealy  staff     71 Jan  9  2024 .gitmodules
-rw-r--r--@  1 kjhealy  staff    821 Aug 16  2023 .luarc.json
drwxr-xr-x@ 34 kjhealy  staff   1088 Sep 23 13:56 .quarto
-rw-r--r--@  1 kjhealy  staff  16656 Sep  8 11:34 .Rhistory
```

Unix naming conventions

Files and folders beginning with a period, `.`, are “hidden”

They won’t show up via `ls`

By convention they are often used for configuration information

In the world of R, files or folders beginning with an underscore, `_`, are often “generated” or are visible configuration files. (This is a weak convention.)

The structure of plain-text config files will depend on the thing they are configuring. It might just a list of words or options, or it might be a structured file based on a Markup language like YAML or TOML, or it might be written to be parsed in R or Python, etc.

Files have extensions by convention. These exist to help the user and they can be useful when writing scripts. And specific applications or processes may expect to look for and use files with specific names or extensions. But the operating system in general doesn’t care about them.

Unix naming conventions

Here's the `.gitignore` file for this project:

```
.Rproj.user
.Rhistory
.RData
.Ruserdata

/.quarto/
/_site/
/renv/

/staging/

/_freeze/
/_targets/

about/*.pdf
about/*.html
assignment/*.html
example/*.html
schedule/*.html
syllabus/*.html
data/dfstrat.csv
slides/*.pdf
slides/*.html
slides/fonts/*
```

Customizing your shell

Bash (often the Linux default)

A `.bashrc` file to configure non-login shells for Bash:

```
# Put the contents of this file in your ~/.bashrc file
# ~/.bashrc: executed by bash(1) for non-login shells.
# see /usr/share/doc/bash/examples/startup-files (in the package bash-doc)
# for examples

# If not running interactively, don't do anything
case $- in
    *i*) ;;
      *) return;;
esac

# don't put duplicate lines or lines starting with space in the history.
# See bash(1) for more options
HISTCONTROL=ignoreboth

# append to the history file, don't overwrite it
shopt -s histappend

# for setting history length see HISTSIZE and HISTFILESIZE in bash(1)
HISTSIZE=1000
HISTFILESIZE=2000

# check the window size after each command and, if necessary,
# update the values of LINES and COLUMNS.
```

Zsh (the Mac default)

```
# Put the contents of this file in your ~/.zshrc file.
# Source: https://github.com/belak/zsh-utils?tab=readme-ov-file

[[ ! -d "$HOME/.antigen" ]] && git clone https://github.com/zsh-users/antigen.git "$HOME/.antigen"
source "$HOME/.antigen/antigen.zsh"

# Set the default plugin repo to be zsh-utils
antigen use belak/zsh-utils --branch=main

# Specify completions we want before the completion module
antigen bundle zsh-users/zsh-completions

# Specify plugins we want
antigen bundle editor@main
antigen bundle history@main
antigen bundle prompt@main
antigen bundle utility@main
antigen bundle completion@main

# Specify additional external plugins we want
antigen bundle zsh-users/zsh-syntax-highlighting

# Load everything
antigen apply
```

Caution

ⓘ Don’t blindly install things

Installing things via shell scripts should only be done from trusted sources!

The Unix way of thinking

Stepping back

Your computer stores files and runs commands.

The files are stored in a large hierarchy called a filesystem.

You issue instructions to run particluar commands at a command line that is provided by a shell, which is how you the user talk to the operating system.

Unix commands and utilities generally try to do a *specific* thing to files or running processes.

The Unix conception of a ‘file’ is very flexible. Connections to other computers can act like files.

Unix commands are often composable using pipes.

The Unix pipe

Unix commands work with some *input* and may produce some *output*

Unix systems have the concepts of “standard input”, “standard output”, and “standard error” as streams where things come from, where they go to, and where problems are reported.

The idea of a sequence of commands or, more generally, *functions* that can be composed or pipelined in a smooth sequence is a very general and very powerful idea that we will soon see in action in R and that you may come across in many other settings as well.

The Unix pipe

The output of the `ls` command again:

```
ls
```

```
_extensions
_freeze
_motivation.qmd
_quarto.yml
_site
_targets
_targets.R
_variables.yml
_weekly-schedule.qmd
00_dummy_files
about
assets
assignment
avhrr
content
data
deploy.sh
example
files
```

The Unix pipe

We can send, or *pipe*, this output to another command, instead of to the terminal:

```
ls | wc -l
```

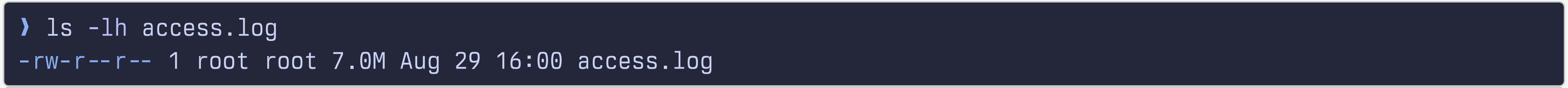
```
      36
```

The `wc` command counts the number of words in a file, or in whatever is sent to it via `STDIN`.

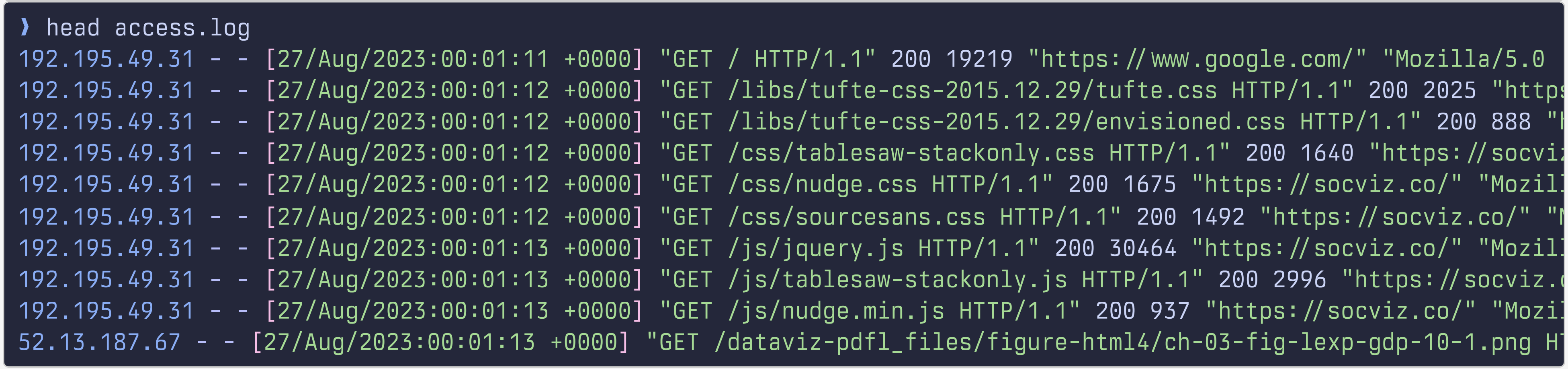
The `-l` switch to `wc` means ‘just count lines instead of words’

The Unix pipe

Like with pipelines in R, we can compose sequences of actions at the prompt:



```
❯ ls -lh access.log
-rw-r--r-- 1 root root 7.0M Aug 29 16:00 access.log
```



```
❯ head access.log
192.195.49.31 - - [27/Aug/2023:00:01:11 +0000] "GET / HTTP/1.1" 200 19219 "https://www.google.com/" "Mozilla/5.0
192.195.49.31 - - [27/Aug/2023:00:01:12 +0000] "GET /libs/tufte-css-2015.12.29/tufte.css HTTP/1.1" 200 2025 "https
192.195.49.31 - - [27/Aug/2023:00:01:12 +0000] "GET /libs/tufte-css-2015.12.29/envisioned.css HTTP/1.1" 200 888 "
192.195.49.31 - - [27/Aug/2023:00:01:12 +0000] "GET /css/tablesaw-stackonly.css HTTP/1.1" 200 1640 "https://socvi
192.195.49.31 - - [27/Aug/2023:00:01:12 +0000] "GET /css/nudge.css HTTP/1.1" 200 1675 "https://socviz.co/" "Mozil
192.195.49.31 - - [27/Aug/2023:00:01:12 +0000] "GET /css/sourcesans.css HTTP/1.1" 200 1492 "https://socviz.co/" "
192.195.49.31 - - [27/Aug/2023:00:01:13 +0000] "GET /js/jquery.js HTTP/1.1" 200 30464 "https://socviz.co/" "Mozil
192.195.49.31 - - [27/Aug/2023:00:01:13 +0000] "GET /js/tablesaw-stackonly.js HTTP/1.1" 200 2996 "https://socviz.
192.195.49.31 - - [27/Aug/2023:00:01:13 +0000] "GET /js/nudge.min.js HTTP/1.1" 200 937 "https://socviz.co/" "Mozil
52.13.187.67 - - [27/Aug/2023:00:01:13 +0000] "GET /dataviz-pdfl_files/figure-html4/ch-03-fig-lexp-gdp-10-1.png H
```

The Unix pipe

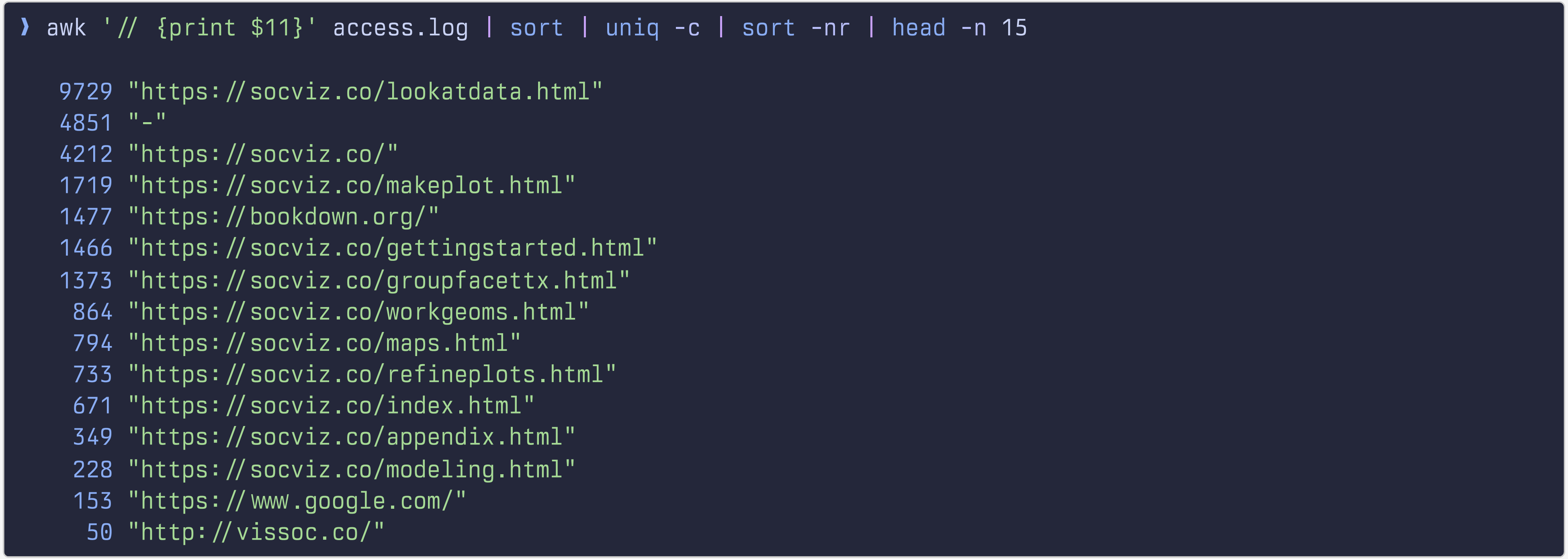
Like with pipelines in R, we can compose sequences of actions at the prompt:

```
❯ head access.log | awk '// {print $11}'

"https://www.google.com/"
"https://socviz.co/"
"https://socviz.co/"
"https://socviz.co/"
"https://socviz.co/"
"https://socviz.co/"
"https://socviz.co/"
"https://socviz.co/"
"https://socviz.co/"
"-"
```

The Unix pipe

Like with pipelines in R, we can compose sequences of actions at the prompt:



```
❯ awk '// {print $11}' access.log | sort | uniq -c | sort -nr | head -n 15

   9729 "https://socviz.co/lookatdata.html"
   4851 "-"
   4212 "https://socviz.co/"
   1719 "https://socviz.co/makeplot.html"
   1477 "https://bookdown.org/"
   1466 "https://socviz.co/gettingstarted.html"
   1373 "https://socviz.co/groupfacettx.html"
    864 "https://socviz.co/workgeoms.html"
    794 "https://socviz.co/maps.html"
    733 "https://socviz.co/refineplots.html"
    671 "https://socviz.co/index.html"
    349 "https://socviz.co/appendix.html"
    228 "https://socviz.co/modeling.html"
    153 "https://www.google.com/"
     50 "http://vissoc.co/"
```

The Unix pipe

We can do a lot with a pipeline:

```
curl -s 'http://api.citybik.es/v2/networks/citi-bike-nyc' |
   jq '.network.stations[].free_bikes' |
  gpaste -sd+ |
  bc
```

```
30820
```

This is the number of Citi Bikes available in New York City at the time these slides were made.

We usually won't use the Unix command line or shell to things like this. We'll do it in R. You could also do it in other languages. But basic shell competence remains extremely handy for many more common tasks.

CitiBike example courtesy of Jeroen Janssens

Shell Scripting

Shell Scripts

If you find yourself doing the same task repeatedly, think about whether it makes sense to write a script

Shell scripts can become mini-programs, but can also be just one or two lines that pull together a few commands

They really show their strength when there’s some fiddly thing you want to do to a lot of files or directories

Shell Scripts

```
#!/usr/bin/env bash

echo "Hello World!"
```

`#!` or “shebang” line saying where the interpreter is

`chmod 755 script.sh` or `chmod +x script.sh` to make executable

The interpreter doesn’t have to be the shell: other languages can be scripted too

Shell Scripts

```
#!/usr/bin/env bash

# Make a thumbnail for each PNG
for i in *.png; do

  FILENAME=$(basename -- "$i") # Full filename
  EXTENSION="${FILENAME##*.}" # Extension only
  FILENAME="${FILENAME%.*}" # Filename without extension

  convert "$i" -thumbnail 500 "$FILENAME-thumb.$EXTENSION";

done;
```

Shell Scripts

The shell can talk to the clipboard:

```
echo I am sending this sentence to the clipboard | pbcopy
```

Back from the clipboard:

```
pbpaste | wc -c
```

```
      44
```

On Windows with Cygwin the corresponding commands are `getclip` and `putclip`.

In an era of Generative AI and LLMs, why are we covering this stuff?

Because Unix is still everywhere

And will be for a long time to come, I’m afraid.

“Why am I doing this?”

As soon as you try to do anything of any sort of technical complexity, or just simple reproducibility, with your computer—even using the newest and coolest tools—I promise you’ll eventually find yourself in a world governed by the metaphors and methods Unix originated, and, very likely, in a literal Unix-derived environment.

That is, you will be in some sort of folder-based hierarchy; you will edit plain-text files in order to configure, launch, generate, or capture the output of applications; and you will do this by way of instructions written down as a series of commands that follow some sort of regular syntax. The details of those instructions (and the particular conventions they use) will vary depending on the task at hand. But in essence you will always be doing the same thing.