Lecture 04-2: Data Science at the Command Line

DATA 503: Fundamentals of Data Engineering

Lucas P. Cordova, Ph.D.

2026-02-02

This lecture introduces data science at the command line, covering the OSEMN model for data science, setting up your environment with Docker, and essential Unix concepts including the shell, command-line tools, pipes, redirection, and file management.

Table of contents

1 Introduction to Data Science at the Command Line **1**

2 Getting Started **6**

3 Essential Unix Concepts **11**

4 Summary **23**

1 Introduction to Data Science at the Command Line

The command line has been around for over 50 years, yet it remains one of the most powerful tools for CS, DS, and beyond! Today we explore why.

1.1 Why the Command Line?

1.1.1 Data Science Meets Ancient Technology

How can a technology that is more than 50 years old be useful for a field that is only a few years young?

Today, data scientists can choose from an overwhelming collection of exciting technologies:

- Python, R, Julia
- Apache Spark
- Jupyter Notebooks
- GUI-based tools

And yet, the command line remains essential. Let us understand why.

1.1.2 Five Advantages of the Command Line

The command line offers five key advantages for data science:

Advantage	Description
Agile	REPL environment allows rapid iteration and exploration
Augmenting	Integrates with and amplifies other technologies
Scalable	Automate, parallelize, and run on remote servers
Extensible	Create your own tools in any language
Ubiquitous	Available on virtually every system

1.1.3 The Command Line Is Agile

Data science is interactive and exploratory. The command line supports this through:

Read-Eval-Print Loop (REPL)

- Type a command, press Enter, see results immediately
- Stop execution at will
- Modify and re-run quickly
- Short iteration cycles let you play with your data

Close to the Filesystem

- Data lives in files
- Command line offers convenient tools for file manipulation
- Navigate, search, and transform data files easily

1.1.4 The Command Line Is Augmenting

The command line does not replace your current workflow. It amplifies it.

Three ways it augments your tools:

1. **Glue between tools:** Connect output of one tool to input of another using pipes
2. **Delegation:** Python, R, and Spark can call command-line tools and capture output
3. **Tool creation:** Convert your scripts into reusable command-line tools

Every technology has strengths and weaknesses. Knowing multiple technologies lets you use the right one for each task.

1.1.5 The Command Line Is Scalable

On the command line, you do things by typing. In a GUI, you point and click.

Everything you type can be:

- Saved to scripts
- Rerun when data changes
- Scheduled to run at intervals
- Executed on remote servers
- Parallelized across multiple cores

Pointing and clicking is hard to automate. Typing is not.

1.1.6 The Command Line Is Extensible

The command line itself is over 50 years old, but its tools are being developed daily.

- Language agnostic: tools can be written in any programming language
- Open source community produces high-quality tools
- Tools can work together
- You can create your own tools

1.1.7 The Command Line Is Ubiquitous

The command line comes with any Unix-like operating system:

- macOS
- Linux (100% of top 500 supercomputers)
- Windows Subsystem for Linux
- Cloud servers
- Embedded systems

This technology has been around for more than five decades. It will be here for five more.

1.2 The OSEMN Model

1.2.1 Data Science Is OSEMN

The field of data science employs a practical definition devised by Hilary Mason and Chris H. Wiggins [1].

OSEMN (pronounced "awesome") defines data science in five steps:

1. **O**btaining data
2. **S**crubbing data
3. **E**xploring data
4. **M**odeling data
5. i**N**terpreting data

1.2.2 OSEMN: Expectation vs Reality

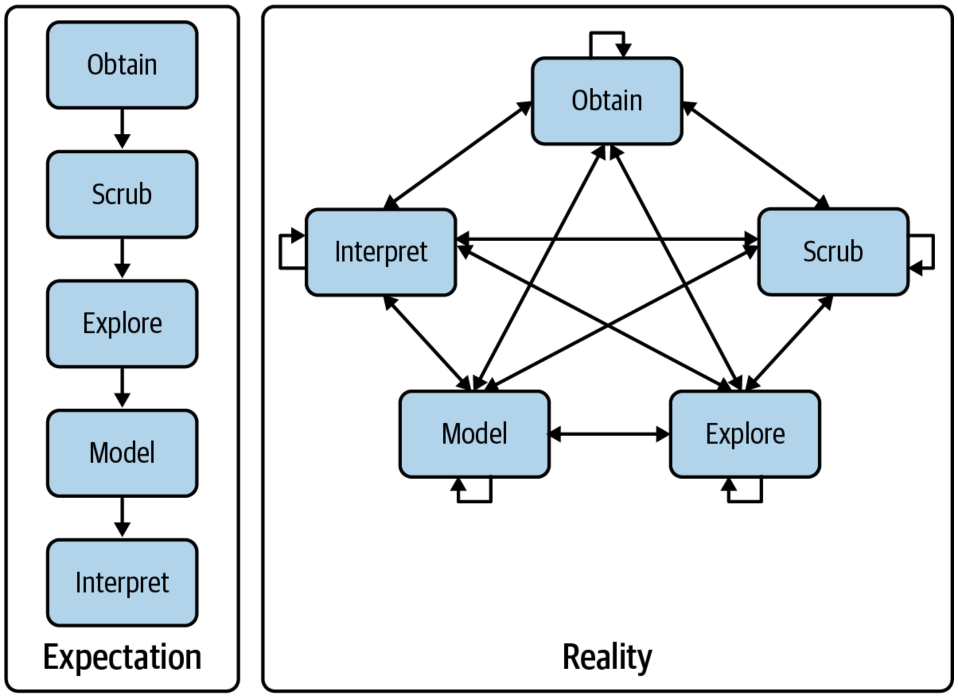


Figure 1: The OSEMN model shows data science as an iterative, nonlinear process

In practice, you move back and forth between steps. After modeling, you may return to scrubbing to adjust features.

1.2.3 The OSEMN Steps

1.2.3.1 Obtain

Download, query APIs, extract from files, or generate data yourself.

Common formats: plain text, CSV, JSON, HTML, XML

1.2.3.2 Scrub

Clean the data before analysis:

- Filter lines
- Extract columns
- Replace values

- Handle missing data
- Convert formats

80% of data project work is in cleaning the data.

1.2.3.3 Explore

Get to know your data:

- Look at the data
- Derive statistics
- Create visualizations

1.2.3.4 Model

Create statistical models:

- Clustering
- Classification
- Regression
- Dimensionality reduction

1.2.3.5 Interpret

Draw conclusions, evaluate results, and communicate findings.

This is where human judgment matters most.

2 Getting Started

2.1 Setting Up Your Environment

2.1.1 Docker: Your Portable Environment

We use Docker to ensure everyone has the same environment with all necessary tools.

What is Docker?

- A Docker **image** bundles applications with their dependencies
- A Docker **container** is an isolated environment running an image
- Like a lightweight virtual machine, but uses fewer resources

Why Docker for this course?

- Consistent environment across Windows, macOS, and Linux
- All tools pre-installed and configured
- No dependency conflicts
- Easy to reset if something goes wrong

2.1.2 Installing Docker

1. Download Docker Desktop from docker.com
2. Install and launch Docker Desktop
3. Open your terminal (or Command Prompt on Windows)
4. Pull the course Docker image:

```
$ docker pull lucascordova/dataeng
```

2.1.3 Running the Docker Container

Run the Docker image:

```
$ docker run --rm -it lucascordova/dataeng
```

You are now inside an isolated environment with all the necessary command-line tools installed.

Test that it works:

```
$ cowsay "Data engineering is awesome!"
```

```
 -------------------------------
< Data engineering is awesome!  >
 -------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

2.1.4 Mounting a Local Directory

To get data in and out of the container, mount a local directory:

2.1.4.1 macOS/Linux

```
$ docker run --rm -it -v "$(pwd)":/data lucascordova/dataeng
```

2.1.4.2 Windows (Command Prompt)

```
C:\> docker run --rm -it -v "%cd%":/data lucascordova/dataeng
```

2.1.4.3 Windows (PowerShell)

```
PS C:\> docker run --rm -it -v ${PWD}:/data lucascordova/dataeng
```

The `-v` option maps your current directory to `/data` inside the container.

2.1.5 Exiting the Container

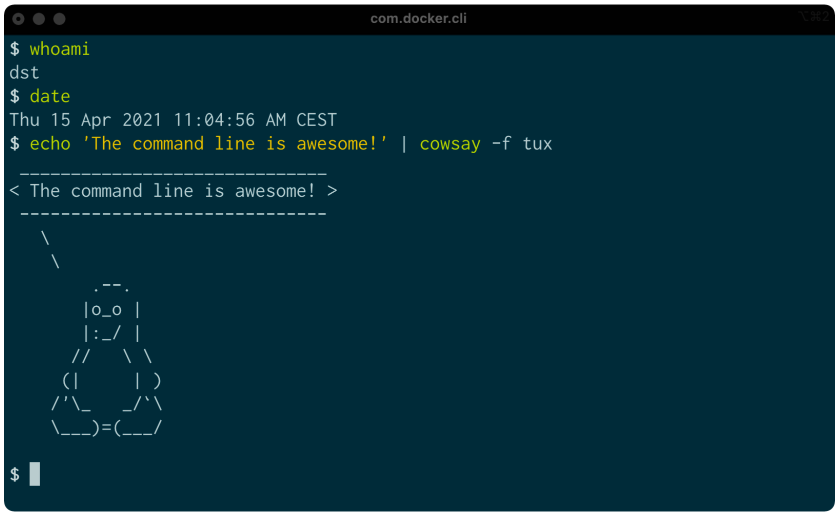
When you are done, exit the container by typing:

```
$ exit
```

The container is removed (due to `--rm` flag), but any files you saved to the mounted directory persist on your computer.

2.2 What Is the Command Line?

2.2.1 The Terminal

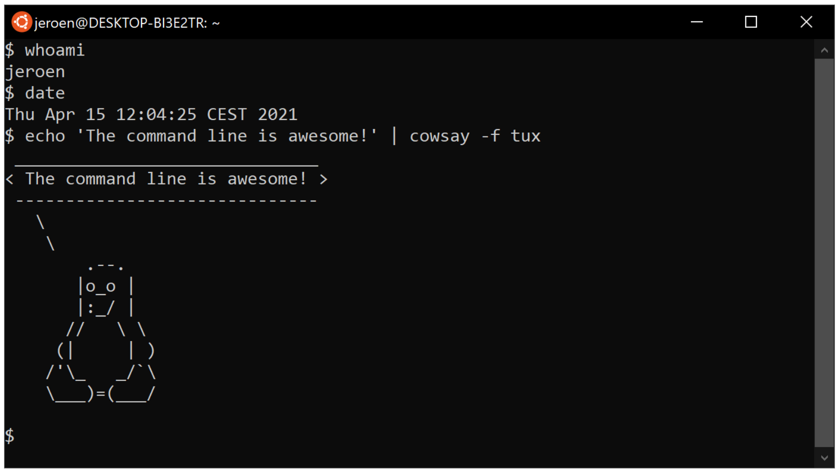


```
$ whoami
dst
$ date
Thu 15 Apr 2021 11:04:56 AM CEST
$ echo 'The command line is awesome!' | cowsay -f tux
 ______________________________
< The command line is awesome! >
 ------------------------------
   \
    \
        .--.
       |o_o |
       |:_/ |
      //   \ \
     (|     | )
    /'\_   _/`\
    \___)=(___/

$ 
```

Figure 2: Command line on macOS

The **terminal** is the application where you type commands. It enables you to interact with the shell.

2.2.2 The Terminal on Linux



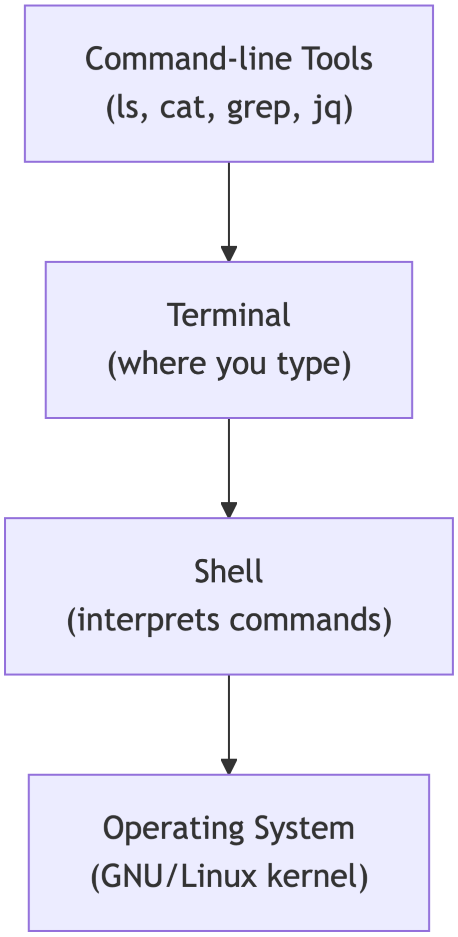
```
$ whoami
jeroen
$ date
Thu Apr 15 12:04:25 CEST 2021
$ echo 'The command line is awesome!' | cowsay -f tux
 ______________________________
< The command line is awesome! >
 ------------------------------
   \
    \
        .--.
       |o_o |
       |:_/ |
      //   \ \
     (|     | )
    /'\_   _/`\
    \___)=(___/

$
```

Figure 3: Command line on Ubuntu

Ubuntu is a distribution of GNU/Linux. The same commands work across different Unix-like systems.

2.2.3 The Four Layers

The environment consists of four layers:



Layer	Purpose
Command-line tools	Programs you execute
Terminal	Application for typing commands
Shell	Program that interprets commands (bash, zsh)
Operating system	Executes tools, manages hardware

2.2.4 Understanding the Prompt

When you see text like this in the book or slides:

```
$ seq 3
1
2
3
```

- The `$` is the **prompt** (do not type it)
- `seq 3` is the command you type
- `1`, `2`, `3` is the output

The prompt may show additional information (username, directory, time), but we show only `$` for simplicity.

3 Essential Unix Concepts

3.1 Executing Commands

3.1.1 Your First Commands

Try these commands in your terminal:

```
$ pwd
/home/dst
```

The tool `pwd` outputs the name of your current directory.

```
$ cd /data/ch02
$ pwd
/data/ch02
```

The tool `cd` changes directories. Values after the command are called **arguments** or **options**.

3.1.2 Command Arguments and Options

Commands often take arguments and options:

```
$ head -n 3 movies.txt
Matrix
Star Wars
Home Alone
```

This command has three arguments:

Argument	Type	Purpose
`-n`	Option (short form)	Specifies number of lines
`3`	Value	The number of lines to show
`movies.txt`	Filename	The file to read

The long form of `-n` is `--lines`.

3.2 Five Types of Command-Line Tools

3.2.1 The Tool Umbrella

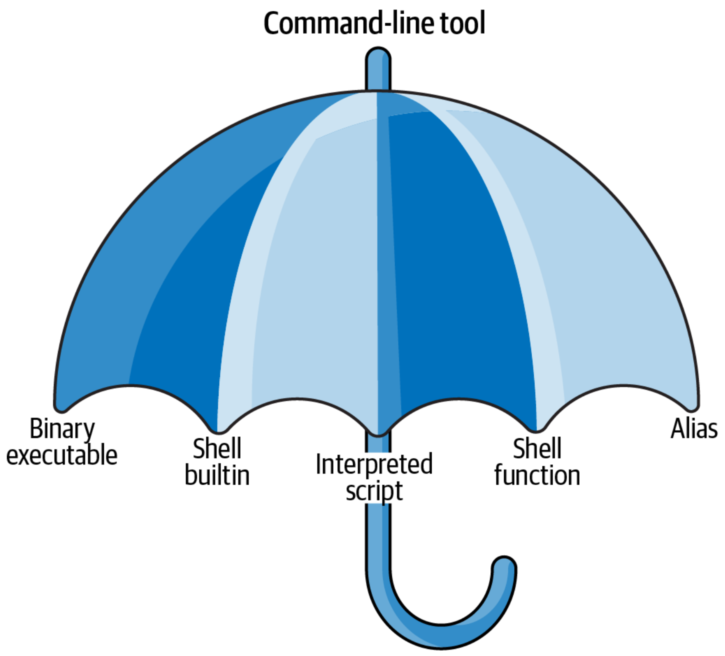


Figure 4: Command-line tool is an umbrella term for five types

Each command-line tool is one of five types:

- Binary executable
- Shell builtin
- Interpreted script
- Shell function
- Alias

3.2.2 Binary Executables

Programs compiled from source code to machine code.

- Created by compiling C, C++, Rust, Go, etc.
- Cannot read the file in a text editor
- Fast execution
- Examples: `ls`, `grep`, `cat`

```
$ file /usr/bin/ls
/usr/bin/ls: ELF 64-bit LSB pie executable...
```

3.2.3 Shell Builtins

Command-line tools provided by the shell itself.

- Part of the shell (bash, zsh)
- May differ between shells
- Cannot be easily inspected
- Examples: `cd`, `pwd`, `echo`

```
$ type cd
cd is a shell builtin
```

3.2.4 Interpreted Scripts

Text files executed by an interpreter (Python, R, Bash).

```python
#!/usr/bin/env python

def factorial(x):
    result = 1
    for i in range(2, x + 1):
        result *= i
    return result

if __name__ == "__main__":
    import sys
    x = int(sys.argv[1])
    print(factorial(x))
```

Advantages: Readable, editable, portable

3.2.5 Shell Functions

Functions executed by the shell itself:

```
$ fac() { (echo 1; seq $1) | paste -s -d\* - | bc; }
$ fac 5
120
```

- Similar to scripts but typically smaller
- Defined in shell configuration (`.bashrc`, `.zshrc`)
- Good for personal productivity shortcuts

3.2.6 Aliases

Macros that expand to longer commands:

```
$ alias l='ls --color -lhF --group-directories-first'
$ alias les=less
```

Now typing `l` expands to the full `ls` command with options.

- Save keystrokes for common commands
- Fix common typos
- Cannot take parameters (use functions for that)

3.2.7 Identifying Tool Types

Use the `type` command to identify what kind of tool you have:

```
$ type -a pwd
pwd is a shell builtin
pwd is /usr/bin/pwd

$ type -a cd
cd is a shell builtin

$ type -a l
l is an alias for ls --color -lhF --group-directories-first
```

3.3 Combining Tools

3.3.1 The Unix Philosophy

Most command-line tools follow the Unix philosophy:

> Do one thing and do it well.

- `grep` filters lines
- `wc` counts lines, words, characters
- `sort` sorts lines
- `head` shows first lines

The power comes from **combining** these simple tools.

3.3.2 Standard Streams

Every tool has three standard communication streams:

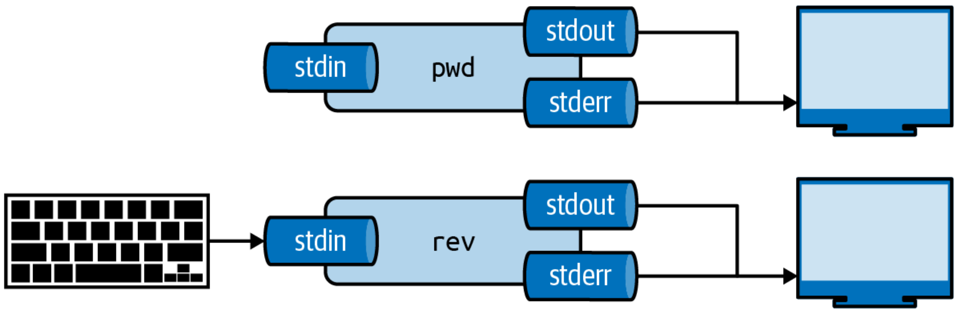


Figure 5: Standard input (stdin), standard output (stdout), and standard error (stderr)

Stream	Abbreviation	Default
Standard input	stdin	Keyboard
Standard output	stdout	Terminal
Standard error	stderr	Terminal

3.3.3 The Pipe Operator

The pipe operator (|) connects stdout of one tool to stdin of another:

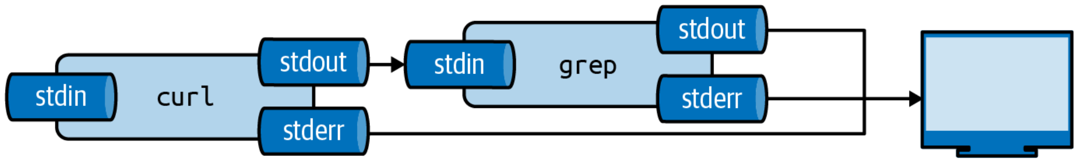


Figure 6: Piping output from curl to grep

```
$ curl -s "https://www.gutenberg.org/files/11/11-0.txt" | grep " CHAPTER"
 CHAPTER I.     Down the Rabbit-Hole
 CHAPTER II.    The Pool of Tears
 CHAPTER III.   A Caucus-Race and a Long Tale
...
```

3.3.4 Chaining Multiple Tools

You can chain as many tools as needed:

```
$ curl -s "https://www.gutenberg.org/files/11/11-0.txt" |
> grep " CHAPTER" |
> wc -l
12
```

This pipeline:

1. Downloads Alice in Wonderland
2. Filters lines containing " CHAPTER"
3. Counts the number of lines

Think of piping as automated copy and paste.

3.4 Redirecting Input and Output

3.4.1 Output Redirection

Save output to a file using `>`:

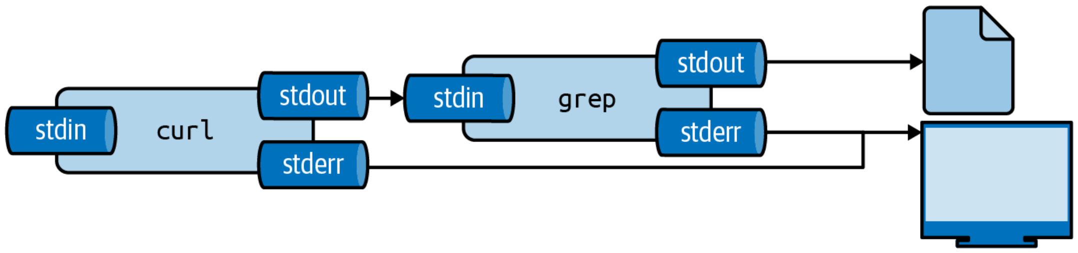


Figure 7: Redirecting output to a file

```
$ curl -s "https://example.com/data.txt" | grep "pattern" > results.txt
```

- Creates file if it does not exist
- **Overwrites** file if it exists
- Standard error still goes to terminal

3.4.2 Appending Output

Use `>>` to append instead of overwrite:

```
$ echo -n "Hello" > greeting.txt
$ echo " World" >> greeting.txt
$ cat greeting.txt
Hello World
```

The `-n` option tells `echo` not to add a trailing newline.

3.4.3 Input Redirection

Read from a file using `<`:

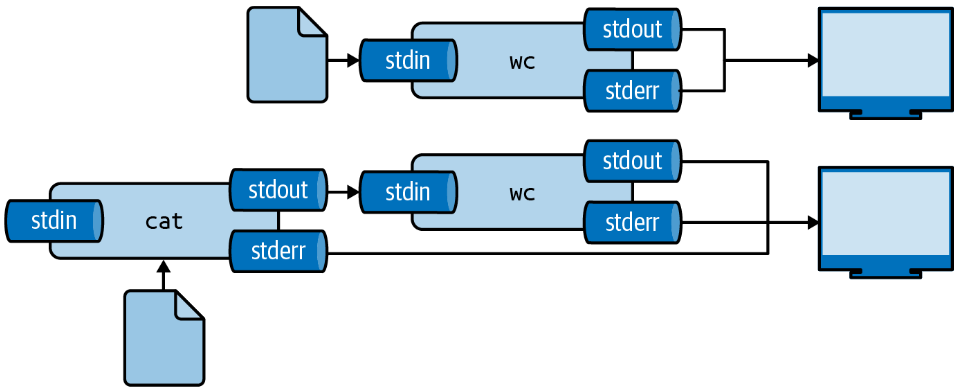


Figure 8: Two ways to use file contents as input

```
$ cat greeting.txt | wc -w
2

$ < greeting.txt wc -w
2
```

Both achieve the same result. The second form avoids starting an extra process.

3.4.4 Redirecting Standard Error

Suppress error messages by redirecting stderr to `/dev/null`:

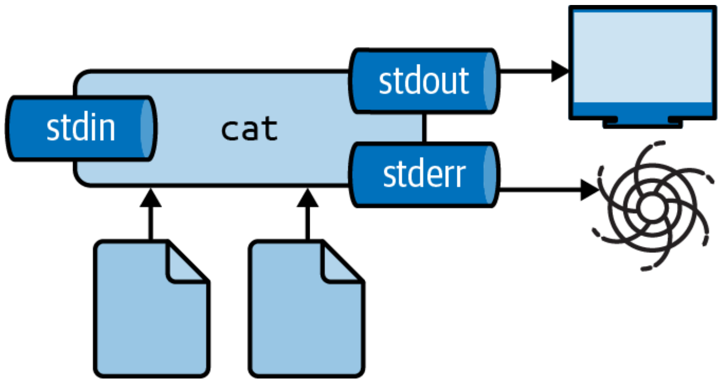


Figure 9: Redirecting stderr to /dev/null

```
$ cat movies.txt 404.txt
Matrix
Star Wars
cat: 404.txt: No such file or directory
```

```
$ cat movies.txt 404.txt 2> /dev/null
Matrix
Star Wars
```

The 2 refers to standard error (file descriptor 2).

3.4.5 The Sponge Problem

Warning: Do not read from and write to the same file in one command!

```
$ < dates.txt nl > dates.txt  # BAD: results in empty file!
```

The output file is opened (and emptied) before reading starts.

Solutions:

1. Write to a different file, then rename
2. Use `sponge` to absorb all input before writing

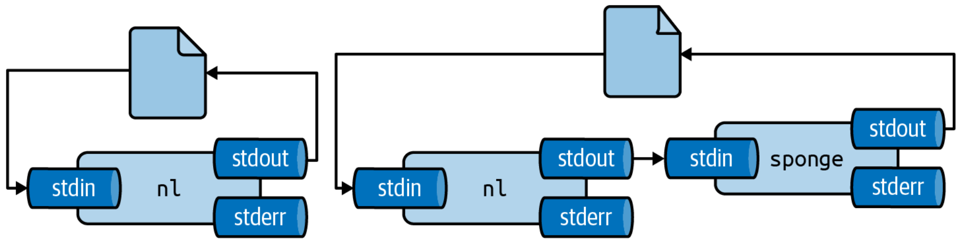


Figure 10: sponge soaks up input before writing

```
$ < dates.txt nl | sponge dates.txt  # GOOD
```

3.4.6 Using tee for Intermediate Output

The `tee` command writes to both a file and stdout:

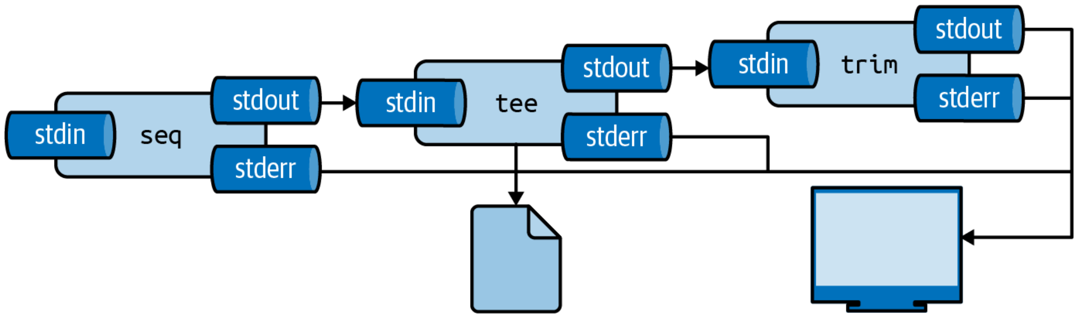


Figure 11: tee writes to file while passing data through

```
$ seq 0 2 100 | tee even.txt | head -5
0
2
4
6
8
```

Useful for saving intermediate results while continuing a pipeline.

3.5 Working with Files and Directories

3.5.1 Listing Directory Contents

```
$ ls /data/ch10
alice.txt  count.py  count.R  Untitled1337.ipynb

$ ls -lhF /data/ch10
total 176K
-rw-r--r-- 1 dst dst 164K Jun 29 14:25 alice.txt
-rwxr-xr-x 1 dst dst  408 Jun 29 14:25 count.py*
-rw-r--r-- 1 dst dst  460 Jun 29 14:25 count.R
-rw-r--r-- 1 dst dst 1.7K Jun 29 14:25 Untitled1337.ipynb
```

Common options:

Option	Meaning
`-l`	Long format (permissions, size, date)
`-h`	Human-readable sizes
`-F`	Append indicators (/ for directories)
`-a`	Show hidden files

3.5.2 Creating and Navigating Directories

```
$ mkdir logs                # Create directory
$ mkdir -p data/raw/2024    # Create nested directories

$ cd /data                  # Change to absolute path
$ cd ..                     # Go up one level
$ cd ~                      # Go to home directory
$ cd -                      # Go to previous directory
```

3.5.3 Moving and Copying Files

```
$ mv hello.txt /data/ch02      # Move file
$ mv hello.txt goodbye.txt     # Rename file
$ mv old_dir new_dir           # Rename directory

$ cp server.log server.log.bak # Copy file
$ cp -r src/ backup/           # Copy directory recursively
```

3.5.4 Removing Files and Directories

```
$ rm bye.txt             # Remove file
$ rm -r /data/ch02/old # Remove directory and contents
$ rm -i *.txt            # Interactive: ask before each removal
```

Warning: There is no recycle bin on the command line. Deleted files are gone.

Use the -v (verbose) option to see what is happening:

```
$ mkdir -v backup
mkdir: created directory 'backup'
```

3.6 Getting Help

3.6.1 Manual Pages

The man command displays the manual page for a tool:

```
$ man tar
TAR(1)                    GNU TAR Manual                    TAR(1)

NAME
       tar - an archiving utility

SYNOPSIS
       tar [options] [pathname ...]
...
```

Navigation:

- Space or `f`: next page
- `b`: previous page
- `/pattern`: search
- `q`: quit

3.6.2 The –help Option

Many tools support `--help`:

```
$ jq --help
jq - commandline JSON processor [version 1.6]

Usage: jq [options] <jq filter> [file...]
...
```

Some tools use `-h` instead.

3.6.3 TLDR Pages

For concise, example-focused help, use `tldr`:

```
$ tldr tar

  tar

  Archiving utility.

  - [c]reate an archive and write it to a [f]ile:
    tar cf target.tar file1 file2 file3
```

```
- E[x]tract a (compressed) archive [f]ile:
  tar xf source.tar[.gz|.bz2|.xz]
```

Shows practical examples instead of exhaustive documentation.

3.6.4 Shell Builtins Help

For shell builtins like `cd`, check the shell manual:

```
$ man zshbuiltins  # for zsh
$ man bash         # for bash (search for the builtin)
```

Or use the `help` command (in bash):

```
$ help cd
```

4 Summary

4.1 Key Takeaways

4.1.1 What We Learned

1. **OSEMN Model:** Obtain, Scrub, Explore, Model, iNterpret - data science is iterative
2. **Command Line Advantages:** Agile, augmenting, scalable, extensible, ubiquitous
3. **Docker Setup:** Use `lucascordova/dataeng` for a consistent environment
4. **Tool Types:** Binary executables, shell builtins, scripts, functions, aliases
5. **Pipes and Redirection:** Combine simple tools into powerful pipelines
6. **File Management:** Navigate, create, copy, move, and remove files

4.1.2 Looking Ahead

Next steps:

- Practice the commands covered today
- Explore the Docker environment
- Complete the command-line exercises

Questions?

4.2 References

4.2.1 References

1. Mason, H. & Wiggins, C. H. (2010). A Taxonomy of Data Science. *dataists blog.* http://www.dataists.com/2010/09/a-taxonomy-of-data-science
2. Janssens, J. (2021). *Data Science at the Command Line* (2nd ed.). O'Reilly Media. https://datascienceatthecommandline.com
3. Raymond, E. S. (2003). *The Art of Unix Programming.* Addison-Wesley.
4. Patil, D. J. (2012). *Data Jujitsu.* O'Reilly Media.
5. Docker Documentation. https://docs.docker.com
6. GNU Coreutils Manual. https://www.gnu.org/software/coreutils/manual