Command Line Basics for R Users

Aravind Hebbali

2026-11-01

Contents

Preface 5
- Outline 7
- Topics this book does not cover 7
- About this book 8
- Using code examples 8
- Software information 8

About the Author 9
Introduction 9
- What is Shell/Terminal? 9
- Choose your setup 9
- Launch Terminal 10
  - mac 10
  - Windows 10
  - Linux 11
  - RStudio Terminal 11
- Prompt 12
- Get Started 12
- Help/Documentation 13
- Exercises 14

Navigating File System 14
- Introduction 14
- Change Working Directory 17
- List Directory Contents 18
  - List files one per line 18
  - List all files including hidden files 19
  - Display file permissions, ownership, size & modification date 20
  - Display size in human readable format 20
  - Sort list by size 21
  - Sort list by modification date 21
- R Functions 22
- Exercises 22

File Management 23
Create new file 23
Copy Files/Folders 24
Copy files in same folder 24
Copy files into different folder 24
Copy folders 25
Move/Rename Files 25
Move files in verbose mode 27
Do not overwrite existing files 27
Do not prompt for confirmation before overwriting files 29
Remove/Delete Files 30
Remove files 30
Recursive Deletion 31
Force Removal 31
Verbose Mode 32
Compare Files 32
Compare Files 33
Side By Side 33
Unified Format 34
Compare Recursively 34
File Details 35
R Functions 35
Exercises 35
Input/Output 36
Display Messages 36
Print Message 36
Redirect Output 36
Print & Concatenate Files 37
Print Content 37
Number All Output Lines 38
Concatenate Several Files 38
Head 39
Output the first parts of a file 39
Output the first n lines of a file 39
Output the first c bytes of a file 40
Output everything but the last 5 lines of a file 40
Output everything but the last 3 bytes of a file 40
Tail 41
Display the last parts of a file 41
Display the last 5 lines of a file 42
Display all contents from line 10 42
Display the last 10 bytes of a file 43
More 43

Less ... 43
Sort ... 44
Sort ... 44
Descending Order ... 47
Ignore case ... 49
Numeric Order ... 51
Preserve Only Unique Lines ... 54
Word Count ... 56
Count words, bytes and lines ... 56
Count lines in a file ... 56
Count words in a file ... 56
Count characters(bytes) in a file ... 56
Exercises ... 56
Search & Regular Expression ... 57
grep ... 57
Match Pattern in Input Text ... 57
Ignore Case ... 58
Highlight ... 59
Print Filename ... 60
Print Corresponding Line Number ... 60
Print Filename & Line Number ... 61
Invert Match ... 62
Recursive Search ... 63
find ... 63
Search Recursively ... 64
Search by Extension ... 64
Case-insensitive Mode ... 64
Case-sensitive Mode ... 64
Exercises ... 65
Data Transfer ... 65
wget ... 65
Download URL ... 66
Specify Filename ... 66
Download File ... 66
Download Multiple URLs ... 67
Quiet ... 67
No Verbose ... 67
Check Timestamp ... 67
curl ... 67
Download URL ... 68
Specify File ... 68
Download Silently ... 68
R Functions ... 68

Exercises ... 69
sudo ... 69
Exercises ... 69
File Compression ... 70
tar ... 70
List ... 71
Extract ... 71
Add ... 72
Create ... 72
gzip ... 72
Compress ... 72
Decompress ... 73
Specify Filename ... 74
zip & unzip ... 75
List ... 75
Extract ... 76
Compress ... 76
Add ... 78
R Functions ... 79
tar & tar.gz ... 79
zip & gzip ... 79
Exercises ... 79
System Info ... 80
uname ... 80
Print all available information about the system ... 80
Display free, used, swap memory in the system ... 81
Display file system disk space usage ... 81
Others ... 81
Funny Commands ... 81
R Functions ... 82
Exercises ... 82
R & the Shell ... 82
R ... 82
RStudio ... 83
RMarkdown ... 83
Rscript ... 84
Exercises ... 84
Pipes & Redirection ... 85
Why pipes matter for R users ... 85
The | operator ... 85
The canonical pipeline ... 85
Redirection and chaining recap ... 86
| versus |> and %>% ... 86

Exercises ... 87
Conclusion ... 87
What we have not covered ... 87
Summary ... 88
References ... 88
Feedback ... 88
Exercises ... 88
Shell ↔ R Cheat Sheet ... 89
Appendix: Git in 10 Minutes for RStudio Users ... 90
Configure once ... 90
The 6-command workflow ... 90
Branches ... 90
Ignore files ... 90
Undo safely ... 91
Exercises ... 91

Preface

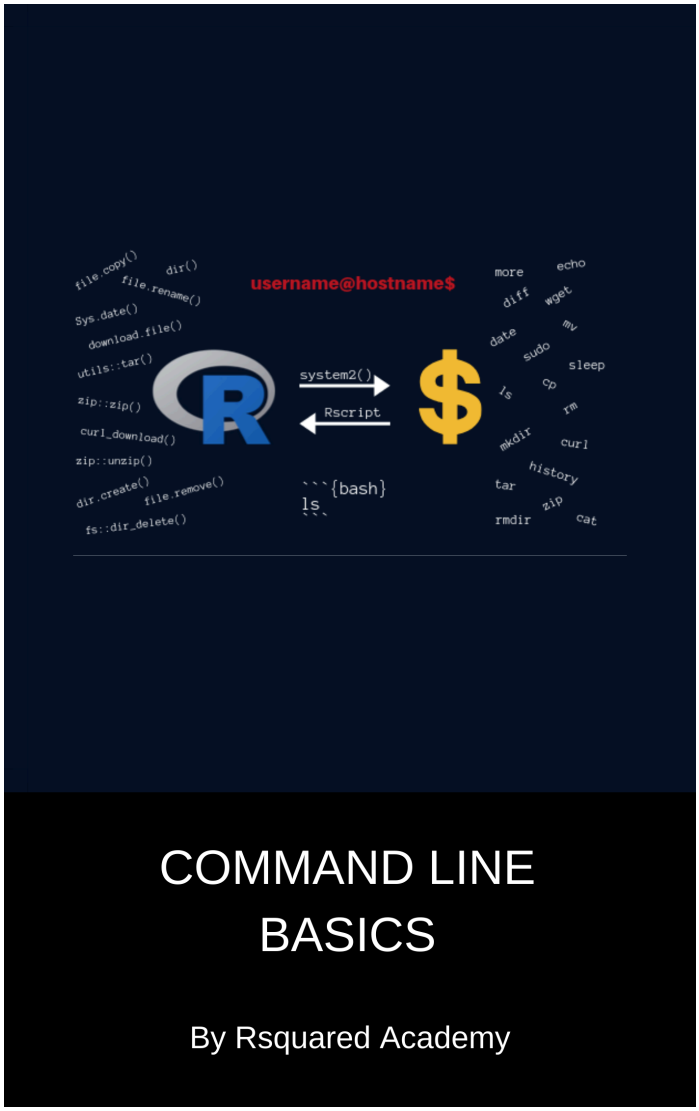

Figure 1: Cover image: Command Line Basics for R Users

This is the website for Command Line Basics for R Users. Visit the GitHub repository for this site.

This work by Aravind Hebbali is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

In this book, you will be introduced to shell commands. We have selected a set of commands we think will be useful in general to a wide range of audience. We have created a Posit Cloud project to ensure that all readers are using the same environment while using code examples. Our goal is to ensure that after reading this book, you should be able to use the shell for version control, managing cloud services (like deploying your own shiny server etc.), execute commands in R & RMarkdown and execute R scripts in the shell. Apart from learning shell commands, the book will also focus on

- exploring R release names

- mapping shell commands to R functions
- RStudio Terminal
- executing shell commands from R using `system2()` or `processx::run()`
- execute shell commands in RMarkdown
- execute R scripts in the shell

If you want a deeper understanding of using command line for data science, we suggest you read Data Science at the Command Line. Software Carpentry too has a lesson on shell. We have listed more references at the end of the book for the benefit of the readers.

Outline

We start by introducing what the shell is, how to launch it in different operating system and explore commands for navigating file system and file management.

- **Introduction** introduces the shell. It shows how to launch it in different operating systems. Apart from introducing the RStudio terminal, it will explore basic commands and different ways to view the documentation.
- **Navigating File System** explores how to navigate between different directories, list contents of a directory and create & delete files & directories.
- **File Management** describes how to create, copy, move, delete and compare files and directories.
- **Input/Output** demonstrates how to print text on the terminal and read content from files. It will cover sorting/viewing/searching file contents.
- **Search & Regular Expression** shows how to search text in file(s) using pattern matching and find files and folders using names and extensions.
- **Data Transfer** explores how to download files from the internet using `wget` and `curl`.
- **sudo** describes how to install, update and remove packages using `sudo`.
- **File Compression** demonstrates how to list, extract, create and add files to archives using `tar`, `zip` and `unzip`.
- **System Info** explains how to view system information using `uname`, `free`, `df` etc.
- **R & the Shell** demonstrates how to execute shell commands in R using `system2()`. It will also look at using shell commands in RMarkdown documents and how to execute R commands and scripts from the command line.
- **Pipes & Redirection** chains commands with `|`, streams data without loading it into R, and maps the shell pipe to R's `|>`.

Topics this book does not cover

While the book explores several shell commands and demonstrates how to execute them in R, it does not cover the following topics:

- shell scripting
- editing files
- file permissions
- user information
- awk
- sed

About this book

This book is focused on practical exploration of shell commands. We focus on a select set of commands we hope will be useful to beginners and motivate them to explore the command line further.

We do not assume any previous knowledge of command line. We have used the RStudio terminal and RStudio Cloud to remove any hurdles for beginners and to ensure all learners have access to the same environment to explore the commands.

We do assume the reader is familiar with R and RStudio.

Using code examples

All the codes used in this book are available in our GitHub repository. We encourage you to use Posit Cloud (it is free), especially Windows users as you will get access to Linux Bash and all the code and files/directories used in the examples.

Software information

The R session information when compiling this book is shown below:

```
sessionInfo()
```

```
R version 4.4.2 (2024-10-31)
Platform: x86_64-pc-linux-gnu
Running under: Ubuntu 24.04.5 LTS

Matrix products: default
BLAS:   /usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3
LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/libopenblasp-r0.3.26.so;
LAPACK version 3.12.0

locale:
 [1] LC_CTYPE=C.UTF-8       LC_NUMERIC=C           LC_TIME=C.UTF-8
 [4] LC_COLLATE=C.UTF-8     LC_MONETARY=C.UTF-8    LC_MESSAGES=C.UTF-8
 [7] LC_PAPER=C.UTF-8       LC_NAME=C              LC_ADDRESS=C
[10] LC_TELEPHONE=C         LC_MEASUREMENT=C.UTF-8 LC_IDENTIFICATION=C

time zone: UTC
tzcode source: system (glibc)

attached base packages:
[1] stats     graphics  grDevices datasets  utils     methods   base
```

```
loaded via a namespace (and not attached):
 [1] compiler_4.4.2  fastmap_1.2.0   cli_3.6.6       htmltools_0.5.9
 [5] tools_4.4.2     yaml_2.3.12     rmarkdown_2.32  knitr_1.52
 [9] jsonlite_2.0.0  xfun_0.61       digest_0.6.39   rlang_1.3.0
[13] renv_1.2.4      evaluate_1.0.5
```

About the Author

Aravind Hebbali is the founder of Rsquared Academy. He earned his Masters in Economics from Madras School of Economics. As an active R user, he has authored several R packages such as

- olsrr
- rfm
- descriptr
- blorr
- xplorerr

In 2015, he founded Rsquared Academy, a free and open source education initiative with focus on data science and analytics. Apart from self paced online courses, Rsquared Academy offers customized learning modules for corporates and universities.

You can find him on GitHub.

Introduction

What is Shell/Terminal?

Shell is a text based application for viewing, handling & manipulating files. It takes in commands and passes them on to the operating system. It is also known as

- CLI (Command Line Interface)
- Bash (Bourne Again Shell)
- Terminal

It is sufficient to know a handful of commands to get started with the shell.

Choose your setup

Pick the first tier that fits you. All tiers run every command in this book, except where noted.

- **Tier 0 (recommended, zero setup): Posit Cloud project** — one-click RStudio in your browser with Linux bash and all book files ready. Nothing to install.
- **Tier 1 (Windows without admin rights): Git Bash inside RStudio** — install Git for Windows, then open Git Bash via *Tools -> Terminal -> New Terminal* in RStudio. Runs everything except `sudo` chapter (`sudo apt-get` needs Linux) and the `free` command in the System Info chapter. For those two chapters, switch to the Tier 0 Posit Cloud project.
- **Tier 2 (Windows with admin rights): WSL2 + Windows Terminal** — full Ubuntu Linux on your machine (see below). Runs everything, including `sudo` and `free`.

- **macOS:** use the built-in Terminal (Zsh since Catalina). Every command in this book works identically in Zsh; see the BSD note below.
- **Linux:** use your native Bash terminal.

Do not run book commands in Windows PowerShell. PowerShell aliases `ls` to `Get-ChildItem` and `curl`/`wget` to `Invoke-WebRequest`, which reject standard Unix syntax. Always use Git Bash, WSL2, or Posit Cloud on Windows.

Note (macOS: Zsh default and BSD differences). macOS uses Zsh by default (since Catalina, 2019) and ships BSD coreutils instead of GNU coreutils. Book commands work the same, except: `head -n -5` (negative counts), `sed -i`, and the output format of `cal` and `date -d` differ from Linux. When a result looks slightly different on your Mac, it is still correct.

Launch Terminal

Although we recommend the Posit Cloud project so that everyone uses the same environment, you should still know how to launch a terminal on your own machine.

mac

Applications -> Utility -> Terminal

Figure 2: Terminal on macOS

Windows

Option 1 — Git Bash (no admin rights needed)

Install Git for Windows and launch **Git Bash**. Inside RStudio, select *Tools -> Terminal -> New Terminal*; if Git Bash is your configured terminal you can run every Tier 1 chapter right there.

Option 2 — WSL2 with Windows Terminal (full Ubuntu Linux)

1. Open PowerShell **as Administrator** and run `wsl --install -d Ubuntu`.
2. Reboot when prompted, then launch Ubuntu and create your Linux user.
3. Install Windows Terminal and open an Ubuntu tab.

You can learn more about the Windows Subsystem for Linux here.

Linux

- Applications -> Accessories -> Terminal
- Applications -> System -> Terminal

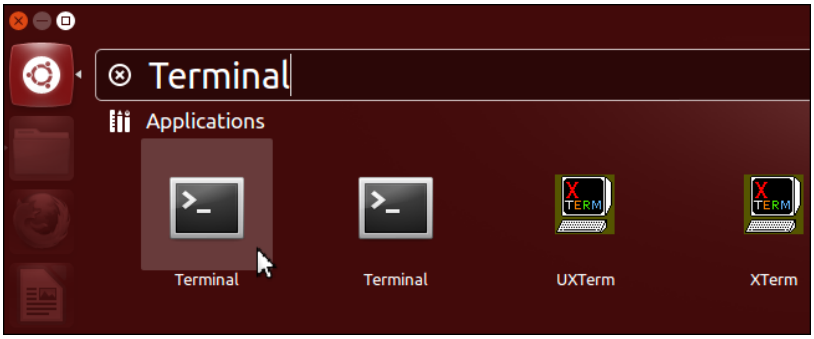

Figure 3: Terminal on Linux

RStudio Terminal

RStudio introduced the terminal with version 1.1.383. The terminal tab is next to the console tab. If it is not visible, use any of the below methods to launch it

- Shift + Alt + T
- Tools -> Terminal -> New Terminal

Note, the terminal depends on the underlying operating system. To learn more about the RStudio terminal, see the terminal pane documentation in RStudio (Tools -> Terminal). In this book, we will use the RStudio terminal on Posit Cloud to ensure that all users have access to Linux bash. You can try all the commands used in this book on your local system as well except in case of Windows users.

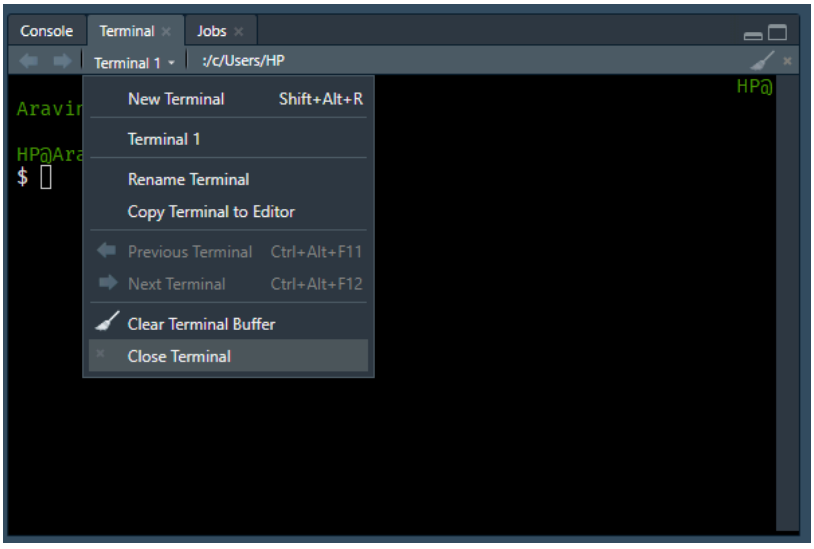

Figure 4: RStudio terminal

Prompt

As soon as you launch the terminal, you will see the hostname, machine name and the prompt. In case of mac & Linux users, the prompt is `$`. For Windows users, it is `>`.

OS	Prompt
macOS	$
Linux	$
Windows	>

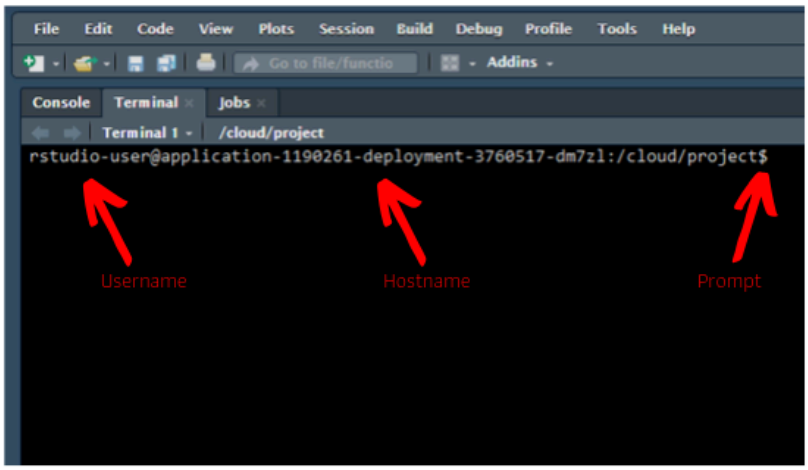

Figure 5: Anatomy of the shell prompt

Get Started

To begin with, let us learn to display

- basic information about the user
- the current date & time
- the calendar
- and clear the screen.

Command	Description
`whoami`	Who is the user?
`date`	Get date, time and timezone
`cal`	Display calendar
`clear`	Clear the screen

`whoami` prints the effective user id i.e. the name of the user who runs the command. Use it to verify the user as which you are logged into the system.

```
whoami
```

```
runner
```

`date` will display or change the value of the system's time and date information.

```
date
```

```
Sun Nov  1 11:34:02 UTC 2026
```

`cal` will display a formatted calendar and `clear` will clear all text on the screen and display a new prompt. You can clear the screen by pressing `Ctrl + L` as well.

```
cal
```

```
   November 2026
Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30
```

In R, we can get the user information from `Sys.info()` or `whoami()` from the whoami package. The current date & time are returned by `Sys.date()` & `Sys.time()`. To clear the R console, we use Ctrl + L.

Command	R
`whoami`	`Sys.info()` / `whoami::whoami()`
`date`	`Sys.date()` / `Sys.time()`
`cal`	
`clear`	`Ctrl + L`

Help/Documentation

Before we proceed further, let us learn to view the documentation/manual pages of the commands.

Command	Description
`man`	Display manual pages for a command
`whatis`	Single line description of a command

`man` is used to view the system's reference manual. Git Bash ships **no** manpages, so if `man` is unavailable in your terminal, append `--help` to any command instead (works everywhere) and consult tldr.sh for practical examples.

```
man ls
```

```
ls --help | head -n 12
```

```
Usage: ls [OPTION]... [FILE]...
List information about the FILEs (the current directory by default).
Sort entries alphabetically if none of -cftuvSUX nor --sort is specified.

Mandatory arguments to long options are mandatory for short options too.
  -a, --all                  do not ignore entries starting with .
  -A, --almost-all           do not list implied . and ..
      --author               with -l, print the author of each file
  -b, --escape               print C-style escapes for nongraphic characters
      --block-size=SIZE      with -l, scale sizes by SIZE when printing them;
                             e.g., '--block-size=M'; see SIZE format below
```

`whatis` displays short manual page descriptions (each manual page has a short description available within it). Like `man`, it needs the system manual database, so it is shown but not executed here.

```
whatis ls
```

You will find tldr.sh very useful while exploring new commands and there is a related R package, tldrrr as well.

```
# devtools::install_github("kirillseva/tldrrr")
tldrrr::tldr("pwd")
```

Exercises

1. Run `whoami`, then `date`, then `cal` one after another in your terminal. For each command, write down in one sentence what was displayed. Then run `clear` (or press `Ctrl + L`) and confirm you get a fresh empty prompt.
2. Run `ls --help | head -n 12`. Identify one option flag mentioned there and write down what the help text says it does. Then try `man ls` — note whether it opens a manual page or reports no manual on your system (Git Bash ships no manpages; `--help` is the fallback).
3. Visit tldr.sh in a browser (or run `tldr ls` if installed) and find one practical example for `ls` you had not seen in this chapter.

Worked solutions in `solutions/introduction.md` (attempt first).

Navigating File System

Introduction

In this chapter, we will learn commands that will help us

- navigate between different folders/directories
- return current working directory
- list all the files & folders in a directory
- create and delete directories

Command	Description
`pwd`	Print working directory
`ls`	List directory contents
`cd`	Change current working directory
`mkdir`	Create directory
`rmdir`	Remove/delete directory

`pwd` displays the name of the **p**resent **w**orking **d**irectory.

```
pwd
```

```
/home/runner/work/bash-intro/bash-intro/cline
```

`ls` displays information about files and directories in the current directory along with their associated metadata such as

- size
- ownership
- modification date

With no options, it will list the files and directories in the current directory, sorted alphabetically.

```
ls
```

```
analysis.R
bash.R
bash.Rmd
bash.sh
cran_log_sample.csv.gz
imports_blorr.txt
imports_olsrr.txt
lorem-ipsum.txt
main_project.zip
myfiles
mypackage
myproject
package_names.txt
pkg_names.txt
r
release_names.tar
release_names.tar.gz
release_names.txt
release_names_18.txt
release_names_19.txt
urls.txt
zip_example.zip
```

`cd` (**c**hange **d**irectory) changes the current working directory. It is among the most used commands as it allows the user to move around the file system.

```
cd r
pwd
```

```
/home/runner/work/bash-intro/bash-intro/cline/r
```

`mkdir` will create new directory. It will allow you to set file mode (permissions associated with the directory) i.e. who can open/modify/delete the directory.

```
mkdir rfiles
ls
```

```
analysis.R
bash.R
bash.Rmd
bash.sh
cran_log_sample.csv.gz
imports_blorr.txt
imports_olsrr.txt
lorem-ipsum.txt
main_project.zip
myfiles
mypackage
myproject
package_names.txt
pkg_names.txt
r
release_names.tar
release_names.tar.gz
release_names.txt
release_names_18.txt
release_names_19.txt
rfiles
urls.txt
zip_example.zip
```

`rmdir` will remove empty directories from the file system. It can be used to remove multiple empty directories as well. If the directory is not empty, `rmdir` will not remove it and instead display a warning that the directory is not empty.

```
rmdir rfiles
ls
```

```
analysis.R
bash.R
bash.Rmd
bash.sh
cran_log_sample.csv.gz
imports_blorr.txt
imports_olsrr.txt
```

```
lorem-ipsum.txt
main_project.zip
myfiles
mypackage
myproject
package_names.txt
pkg_names.txt
r
release_names.tar
release_names.tar.gz
release_names.txt
release_names_18.txt
release_names_19.txt
urls.txt
zip_example.zip
```

Change Working Directory

Let us focus a bit more on changing working directory. The below table shows commands for changing working directory to

- up one level
- previous working directory
- home directory
- and root directory

Command	Description
`cd .`	Navigate into directory
`cd ..`	Go up one level
`cd -`	Go to previous working directory
`cd ~`	Change directory to home directory
`cd /`	Change directory to root directory

All files and directories stem from one main directory, the root directory. All the other directories in the system are sub-directories of the root directory and the root directory has no parent directory. It is represented by a single slash (`/`). `cd /` will change the current working directory to the root directory.

```
cd /
pwd
```

```
/
```

The parent directory i.e. the directory one level up from the current directory which contains the directory we are in now is represented by two dots (`..`). `cd ..` will change us into the parent directory of the current directory.

```
cd ..
pwd
```

```
/home/runner/work/bash-intro/bash-intro
```

The home directory is the directory we are placed in, by default, when we launch a new terminal session. It is represented by the tilde (~).

```
cd ~
pwd
```

```
/home/runner
```

To change into the previous working directory, use `cd -`.

```
cd -
pwd
```

```
bash: line 1: cd: OLDPWD not set
/home/runner/work/bash-intro/bash-intro/cline
```

The current working directory is represented by a single dot (`.`). `cd .` will change us into the current directory i.e. it will do nothing.

List Directory Contents

`ls` will list the contents of a directory. Using different arguments, we can

- list hidden files
- view file permissions, ownership, size & modification date
- sort by size & modification date

Command	Description
`ls`	List directory contents
`ls -l`	List files one per line
`ls -a`	List all files including hidden files
`ls -la`	Display file permissions, ownership, size & modification date
`ls -lh`	Long format list with size displayed in human readable format
`ls -lS`	Long format list sorted by size
`ls -ltr`	Long format list sorted by modification date

List files one per line

```
ls -l
```

```
total 200
-rw-r--r-- 1 runner runner    12 Jun  2  2021 analysis.R
```

```
-rw-r--r-- 1 runner runner   430 Jun  2  2021 bash.R
-rw-r--r-- 1 runner runner   145 Jun  2  2021 bash.Rmd
-rw-r--r-- 1 runner runner 16242 Jun  2  2021 bash.sh
-rw-r--r-- 1 runner runner 48185 Sep 25 10:55 cran_log_sample.csv.gz
-rw-r--r-- 1 runner runner    35 Jun  2  2021 imports_blorr.txt
-rw-r--r-- 1 runner runner    34 Jun  2  2021 imports_olsrr.txt
-rw-r--r-- 1 runner runner 39501 Jun  2  2021 lorem-ipsum.txt
-rw-r--r-- 1 runner runner  9291 Jun  2  2021 main_project.zip
drwxr-xr-x 2 runner runner  4096 Jun  2  2021 myfiles
drwxr-xr-x 9 runner runner  4096 Jun  2  2021 mypackage
drwxr-xr-x 5 runner runner  4096 Jun  2  2021 myproject
-rw-r--r-- 1 runner runner  1498 Jun  2  2021 package_names.txt
-rw-r--r-- 1 runner runner  1082 Jun  2  2021 pkg_names.txt
drwxr-xr-x 2 runner runner  4096 Jun  2  2021 r
-rw-r--r-- 1 runner runner 10240 Jun  2  2021 release_names.tar
-rw-r--r-- 1 runner runner   632 Jun  2  2021 release_names.tar.gz
-rw-r--r-- 1 runner runner   546 Jun  2  2021 release_names.txt
-rw-r--r-- 1 runner runner    65 Jun  2  2021 release_names_18.txt
-rw-r--r-- 1 runner runner    53 Jun  2  2021 release_names_19.txt
-rw-r--r-- 1 runner runner   157 Jun  2  2021 urls.txt
-rw-r--r-- 1 runner runner  4398 Jun  2  2021 zip_example.zip
```

List all files including hidden files

```
ls -a
```

```
.
..
analysis.R
bash.R
bash.Rmd
bash.sh
cran_log_sample.csv.gz
imports_blorr.txt
imports_olsrr.txt
lorem-ipsum.txt
main_project.zip
myfiles
mypackage
myproject
package_names.txt
pkg_names.txt
r
release_names.tar
release_names.tar.gz
release_names.txt
release_names_18.txt
release_names_19.txt
urls.txt
zip_example.zip
```

Display file permissions, ownership, size & modification date

```
ls -la
```

```
total 208
drwxr-xr-x  6 runner runner  4096 Nov  1 11:34 .
drwxr-xr-x 17 runner runner  4096 Nov  1 11:34 ..
-rw-r--r--  1 runner runner    12 Jun  2  2021 analysis.R
-rw-r--r--  1 runner runner   430 Jun  2  2021 bash.R
-rw-r--r--  1 runner runner   145 Jun  2  2021 bash.Rmd
-rw-r--r--  1 runner runner 16242 Jun  2  2021 bash.sh
-rw-r--r--  1 runner runner 48185 Sep 25 10:55 cran_log_sample.csv.gz
-rw-r--r--  1 runner runner    35 Jun  2  2021 imports_blorr.txt
-rw-r--r--  1 runner runner    34 Jun  2  2021 imports_olsrr.txt
-rw-r--r--  1 runner runner 39501 Jun  2  2021 lorem-ipsum.txt
-rw-r--r--  1 runner runner  9291 Jun  2  2021 main_project.zip
drwxr-xr-x  2 runner runner  4096 Jun  2  2021 myfiles
drwxr-xr-x  9 runner runner  4096 Jun  2  2021 mypackage
drwxr-xr-x  5 runner runner  4096 Jun  2  2021 myproject
-rw-r--r--  1 runner runner  1498 Jun  2  2021 package_names.txt
-rw-r--r--  1 runner runner  1082 Jun  2  2021 pkg_names.txt
drwxr-xr-x  2 runner runner  4096 Jun  2  2021 r
-rw-r--r--  1 runner runner 10240 Jun  2  2021 release_names.tar
-rw-r--r--  1 runner runner   632 Jun  2  2021 release_names.tar.gz
-rw-r--r--  1 runner runner   546 Jun  2  2021 release_names.txt
-rw-r--r--  1 runner runner    65 Jun  2  2021 release_names_18.txt
-rw-r--r--  1 runner runner    53 Jun  2  2021 release_names_19.txt
-rw-r--r--  1 runner runner   157 Jun  2  2021 urls.txt
-rw-r--r--  1 runner runner  4398 Jun  2  2021 zip_example.zip
```

Display size in human readable format

```
ls -lh
```

```
total 200K
-rw-r--r-- 1 runner runner   12 Jun  2  2021 analysis.R
-rw-r--r-- 1 runner runner  430 Jun  2  2021 bash.R
-rw-r--r-- 1 runner runner  145 Jun  2  2021 bash.Rmd
-rw-r--r-- 1 runner runner  16K Jun  2  2021 bash.sh
-rw-r--r-- 1 runner runner  48K Sep 25 10:55 cran_log_sample.csv.gz
-rw-r--r-- 1 runner runner   35 Jun  2  2021 imports_blorr.txt
-rw-r--r-- 1 runner runner   34 Jun  2  2021 imports_olsrr.txt
-rw-r--r-- 1 runner runner  39K Jun  2  2021 lorem-ipsum.txt
-rw-r--r-- 1 runner runner 9.1K Jun  2  2021 main_project.zip
drwxr-xr-x 2 runner runner 4.0K Jun  2  2021 myfiles
drwxr-xr-x 9 runner runner 4.0K Jun  2  2021 mypackage
drwxr-xr-x 5 runner runner 4.0K Jun  2  2021 myproject
-rw-r--r-- 1 runner runner 1.5K Jun  2  2021 package_names.txt
-rw-r--r-- 1 runner runner 1.1K Jun  2  2021 pkg_names.txt
drwxr-xr-x 2 runner runner 4.0K Jun  2  2021 r
-rw-r--r-- 1 runner runner  10K Jun  2  2021 release_names.tar
-rw-r--r-- 1 runner runner  632 Jun  2  2021 release_names.tar.gz
```

```
-rw-r--r-- 1 runner runner  546 Jun  2  2021 release_names.txt
-rw-r--r-- 1 runner runner   65 Jun  2  2021 release_names_18.txt
-rw-r--r-- 1 runner runner   53 Jun  2  2021 release_names_19.txt
-rw-r--r-- 1 runner runner  157 Jun  2  2021 urls.txt
-rw-r--r-- 1 runner runner 4.3K Jun  2  2021 zip_example.zip
```

Sort list by size

```
ls -lS
```

```
total 200
-rw-r--r-- 1 runner runner 48185 Sep 25 10:55 cran_log_sample.csv.gz
-rw-r--r-- 1 runner runner 39501 Jun  2  2021 lorem-ipsum.txt
-rw-r--r-- 1 runner runner 16242 Jun  2  2021 bash.sh
-rw-r--r-- 1 runner runner 10240 Jun  2  2021 release_names.tar
-rw-r--r-- 1 runner runner  9291 Jun  2  2021 main_project.zip
-rw-r--r-- 1 runner runner  4398 Jun  2  2021 zip_example.zip
drwxr-xr-x 2 runner runner  4096 Jun  2  2021 myfiles
drwxr-xr-x 9 runner runner  4096 Jun  2  2021 mypackage
drwxr-xr-x 5 runner runner  4096 Jun  2  2021 myproject
drwxr-xr-x 2 runner runner  4096 Jun  2  2021 r
-rw-r--r-- 1 runner runner  1498 Jun  2  2021 package_names.txt
-rw-r--r-- 1 runner runner  1082 Jun  2  2021 pkg_names.txt
-rw-r--r-- 1 runner runner   632 Jun  2  2021 release_names.tar.gz
-rw-r--r-- 1 runner runner   546 Jun  2  2021 release_names.txt
-rw-r--r-- 1 runner runner   430 Jun  2  2021 bash.R
-rw-r--r-- 1 runner runner   157 Jun  2  2021 urls.txt
-rw-r--r-- 1 runner runner   145 Jun  2  2021 bash.Rmd
-rw-r--r-- 1 runner runner    65 Jun  2  2021 release_names_18.txt
-rw-r--r-- 1 runner runner    53 Jun  2  2021 release_names_19.txt
-rw-r--r-- 1 runner runner    35 Jun  2  2021 imports_blorr.txt
-rw-r--r-- 1 runner runner    34 Jun  2  2021 imports_olsrr.txt
-rw-r--r-- 1 runner runner    12 Jun  2  2021 analysis.R
```

Sort list by modification date

```
ls -ltr
```

```
total 200
-rw-r--r-- 1 runner runner  4398 Jun  2  2021 zip_example.zip
-rw-r--r-- 1 runner runner   157 Jun  2  2021 urls.txt
-rw-r--r-- 1 runner runner    53 Jun  2  2021 release_names_19.txt
-rw-r--r-- 1 runner runner    65 Jun  2  2021 release_names_18.txt
-rw-r--r-- 1 runner runner   546 Jun  2  2021 release_names.txt
-rw-r--r-- 1 runner runner   632 Jun  2  2021 release_names.tar.gz
-rw-r--r-- 1 runner runner 10240 Jun  2  2021 release_names.tar
drwxr-xr-x 2 runner runner  4096 Jun  2  2021 r
-rw-r--r-- 1 runner runner  1498 Jun  2  2021 package_names.txt
drwxr-xr-x 2 runner runner  4096 Jun  2  2021 myfiles
-rw-r--r-- 1 runner runner  9291 Jun  2  2021 main_project.zip
-rw-r--r-- 1 runner runner    34 Jun  2  2021 imports_olsrr.txt
-rw-r--r-- 1 runner runner    35 Jun  2  2021 imports_blorr.txt
```

```
-rw-r--r-- 1 runner runner  1082 Jun  2  2021 pkg_names.txt
drwxr-xr-x 5 runner runner  4096 Jun  2  2021 myproject
drwxr-xr-x 9 runner runner  4096 Jun  2  2021 mypackage
-rw-r--r-- 1 runner runner 39501 Jun  2  2021 lorem-ipsum.txt
-rw-r--r-- 1 runner runner 16242 Jun  2  2021 bash.sh
-rw-r--r-- 1 runner runner   145 Jun  2  2021 bash.Rmd
-rw-r--r-- 1 runner runner   430 Jun  2  2021 bash.R
-rw-r--r-- 1 runner runner    12 Jun  2  2021 analysis.R
-rw-r--r-- 1 runner runner 48185 Sep 25 10:55 cran_log_sample.csv.gz
```

R Functions

In R, `getwd()` will return the current working directory. You can use `here()` from the here package as well. To change the current working directory, use `setwd()`. The fs package provides useful functions for file operations.

Command	R
`pwd`	`getwd()` / `here::here()`
`ls`	`dir()` / `list.files()` / `list.dirs()` / `fs::dir_ls()` / `dir_info()`
`cd`	`setwd()`
`mkdir`	`dir.create()` / `fs::dir_create()`
`rmdir`	`fs::dir_delete()`

Exercises

1. From inside the book's `cline/` folder, run `pwd`, then `mkdir practice_nav` followed by `ls` — you should see `practice_nav` appear. Then run `rmdir practice_nav` followed by `ls` and confirm it is gone.
2. Still in `cline/`, run `cd r`, then `pwd` (path should end in `/r`). Then `cd ..` and `pwd` (back in `cline/`), then `cd -` and `pwd`. Finish with `cd ~`, `pwd` (home directory), and `cd -`, `pwd` to return. Write down the directory shown after each `pwd`.
3. In `cline/`, run `ls -l`, then `ls -la`, then `ls -lh`, then `ls -lS`, then `ls -ltr`. Note what changes each time: hidden entries (starting with `.`) appear with `-a`, human-readable sizes with `-h`, and different ordering with `-lS` (by size) versus `-ltr` (by time, oldest first). Which variant would you use to find the most recently modified file?

Worked solutions in `solutions/navigating.md` (attempt first).

File Management

In this chapter, we will explore commands for file management including:

- create new file/change timestamps
- copying files
- renaming/moving files
- deleting files
- comparing files

Command	Description
`touch`	Create empty file(s)/change timestamp
`cp`	Copy files & folders
`mv`	Rename/move file
`rm`	Remove/delete file
`diff`	Compare files

Create new file

`touch` modifies file timestamps which is information associated with file modification. It can be any of the following:

- access time (the last time the file was read)
- modification time (the last time the contents of the file was changed)
- change time (the last time the file's metadata was changed)

If the file does not exist, it will create an empty file of the same name. Let us use `touch` to create a new file `myanalysis.R`.

```
touch myanalysis.R
ls
```

```
analysis.R
bash.R
bash.Rmd
bash.sh
cran_log_sample.csv.gz
imports_blorr.txt
imports_olsrr.txt
lorem-ipsum.txt
main_project.zip
myanalysis.R
myfiles
mypackage
myproject
package_names.txt
pkg_names.txt
r
```

```
release_names.tar
release_names.tar.gz
release_names.txt
release_names_18.txt
release_names_19.txt
urls.txt
zip_example.zip
```

Copy Files/Folders

`cp` makes copies of files and directories. The general form of the command is **cp source destination**. By default, it will overwrite files without prompting for confirmation so be cautious while copying files or folders.

Copy files in same folder

Let us create a copy of `release_names.txt` file and name it as `release_names_2.txt`.

```
cp release_names.txt release_names_2.txt
ls
```

```
analysis.R
bash.R
bash.Rmd
bash.sh
cran_log_sample.csv.gz
imports_blorr.txt
imports_olsrr.txt
lorem-ipsum.txt
main_project.zip
myanalysis.R
myfiles
mypackage
myproject
package_names.txt
pkg_names.txt
r
release_names.tar
release_names.tar.gz
release_names.txt
release_names_18.txt
release_names_19.txt
release_names_2.txt
urls.txt
zip_example.zip
```

Copy files into different folder

To copy a file into a different directory/folder, we need to specify the name of the destination folder. If the copied file should have a different name, then we need to specify the new name of the file as well. Let us copy the `release_names.txt` file into the `r_releases` folder (we will retain the same name for the file as we are copying it into a different folder).

```
mkdir -p r_releases
cp release_names.txt r_releases/release_names.txt
```

Let us check if the file has been copied by listing the files in the `r_releases` folder using `ls`.

```
ls r_releases
```

```
release_names.txt
```

Copy folders

How about making copies of folders? Use the `-r` option to copy entire folders. Let us create a copy of the `r` folder and name it as `r2`. The `-r` option stands for `--recursive` i.e. copy directories recursively.

```
cp -r r r2
ls
```

```
analysis.R
bash.R
bash.Rmd
bash.sh
cran_log_sample.csv.gz
imports_blorr.txt
imports_olsrr.txt
lorem-ipsum.txt
main_project.zip
myanalysis.R
myfiles
mypackage
myproject
package_names.txt
pkg_names.txt
r
r2
r_releases
release_names.tar
release_names.tar.gz
release_names.txt
release_names_18.txt
release_names_19.txt
release_names_2.txt
urls.txt
zip_example.zip
```

Move/Rename Files

`mv` moves and renames files and directories. Using different options, we can ensure

- files are not overwritten
- user is prompted for confirmation before overwriting files
- details of files being moved is displayed

Command	Description
`mv`	Move or rename files/directories
`mv -f`	Do not prompt for confirmation before overwriting files
`mv -i`	Prompt for confirmation before overwriting files
`mv -n`	Do not overwrite existing files
`mv -v`	Move files in verbose mode

Let us move the `release_names_2.txt` file to the `r_releases` folder.

```
mv release_names_2.txt r_releases
```

Use `ls` to verfiy if the file has been moved. As you can see, `release_names_2.txt` is not present in the current working directory.

```
ls
```

```
analysis.R
bash.R
bash.Rmd
bash.sh
cran_log_sample.csv.gz
imports_blorr.txt
imports_olsrr.txt
lorem-ipsum.txt
main_project.zip
myanalysis.R
myfiles
mypackage
myproject
package_names.txt
pkg_names.txt
r
r2
r_releases
release_names.tar
release_names.tar.gz
release_names.txt
release_names_18.txt
release_names_19.txt
urls.txt
zip_example.zip
```

Let us check if `release_names_2.txt` is present in the `r_releases` folder. Great! We have successfully moved the file into a different folder.

```
ls r_releases
```

```
release_names.txt
release_names_2.txt
```

Move files in verbose mode

To view the details of the files being moved/renamed, use the `-v` option. In the below example, we first stage `release_names_3.txt` and then move it into the `r_releases` folder using `mv`.

```
cp release_names.txt release_names_3.txt
mv -v release_names_3.txt r_releases
```

```
renamed 'release_names_3.txt' -> 'r_releases/release_names_3.txt'
```

Do not overwrite existing files

How do we ensure that files are not overwritten without prompting the user first? In the below example, we will try to overwrite the `release_names_2.txt` in the `r_releases` folder using `mv` and see what happens. But first, let us look at the contents of the `release_names_2.txt` file using the `cat` command.

We will look into the `cat` command in more detail in the next chapter but for the time being it is sufficient to know that it prints contents of a file. The file contains release names of different R versions.

```
cat r_releases/release_names_2.txt
```

```
Unsuffered Consequences
Great Pumpkin
December Snowflakes
Gift-Getting Season
Easter Beagle
Roasted Marshmallows
Trick or Treat
Security Blanket
Masked Marvel
Good Sport
Frisbee Sailing
Warm Puppy
Spring Dance
Sock it to Me
Pumpkin Helmet
Smooth Sidewalk
Full of Ingredients
World-Famous Astronaut
Fire Safety
Wooden Christmas Tree
Very Secure Dishes
Very, Very Secure Dishes
Supposedly Educational
Bug in Your Hair
Sincere Pumpkin Patch
Another Canoe
```

```
You Stupid Darkness
Single Candle
Short Summer
Kite Eating Tree
```

In our current working directory, we will create another file of the same name i.e. `release_names_2.txt` but its contents are different from the file in the `r_releases` folder. It contains the string `release_names` and nothing else. We will now move this file into the `r_releases` folder but use the option `-n` to ensure that the file in the `r_releases` folder is not overwritten. We can confirm this by printing the contents of the file in the `r_releases` folder.

The `echo` command is used to print text to the terminal or to write to a file. We will explore it in more detail in the next chapter.

```
echo "release_names" > release_names_2.txt
mv -n release_names_2.txt r_releases
cat r_releases/release_names_2.txt
```

```
mv: not replacing 'r_releases/release_names_2.txt'
Unsuffered Consequences
Great Pumpkin
December Snowflakes
Gift-Getting Season
Easter Beagle
Roasted Marshmallows
Trick or Treat
Security Blanket
Masked Marvel
Good Sport
Frisbee Sailing
Warm Puppy
Spring Dance
Sock it to Me
Pumpkin Helmet
Smooth Sidewalk
Full of Ingredients
World-Famous Astronaut
Fire Safety
Wooden Christmas Tree
Very Secure Dishes
Very, Very Secure Dishes
Supposedly Educational
Bug in Your Hair
Sincere Pumpkin Patch
Another Canoe
You Stupid Darkness
Single Candle
Short Summer
Kite Eating Tree
```

As you can observe, the contents of the file in the `r_releases` folder has not changed. In the next section, we will learn to overwrite the contents using the `-f` option.

Do not prompt for confirmation before overwriting files

What if we actually intend to overwrite a file and do not want to be prompted for confirming the same. In this case, we can use the `-f` option which stands for `--force` i.e. do not prompt before overwriting. Let us first print the contents of the `release_names_2.txt` file in the `r_releases` folder.

```
cat r_releases/release_names_2.txt
```

```
Unsuffered Consequences
Great Pumpkin
December Snowflakes
Gift-Getting Season
Easter Beagle
Roasted Marshmallows
Trick or Treat
Security Blanket
Masked Marvel
Good Sport
Frisbee Sailing
Warm Puppy
Spring Dance
Sock it to Me
Pumpkin Helmet
Smooth Sidewalk
Full of Ingredients
World-Famous Astronaut
Fire Safety
Wooden Christmas Tree
Very Secure Dishes
Very, Very Secure Dishes
Supposedly Educational
Bug in Your Hair
Sincere Pumpkin Patch
Another Canoe
You Stupid Darkness
Single Candle
Short Summer
Kite Eating Tree
```

Now we will create another file of the same name in the current working directory but with different content and use the `-f` option to overwrite the file in the `r_releases` folder. You can see that the contents of the file in the `r_releases` folder has changed.

```
echo "release_names" > release_names_2.txt
mv -f release_names_2.txt r_releases
cat r_releases/release_names_2.txt
```

```
release_names
```

Remove/Delete Files

The `rm` command is used to delete/remove files & folders. Using additional options, we can

- remove directories & sub-directories
- forcibly remove directories
- interactively remove multiple files
- display information about files removed/deleted

Command	Description
`rm`	Remove files/directories
`rm -r`	Recursively remove a directory & all its subdirectories
`rm -rf`	Forcibly remove directory without prompting for confirmation or showing error messages
`rm -i`	Interactively remove multiple files, with a prompt before every removal
`rm -v`	Remove files in verbose mode, printing a message for each removed file

Remove files

Let us use `rm` to remove the file `myanalysis.R` (we created it earlier using the `touch` command).

```
rm myanalysis.R
ls
```

```
analysis.R
bash.R
bash.Rmd
bash.sh
cran_log_sample.csv.gz
imports_blorr.txt
imports_olsrr.txt
lorem-ipsum.txt
main_project.zip
myfiles
mypackage
myproject
package_names.txt
pkg_names.txt
r
r2
r_releases
release_names.tar
release_names.tar.gz
release_names.txt
```

```
release_names_18.txt
release_names_19.txt
urls.txt
zip_example.zip
```

Recursive Deletion

How about folders or directories? We can remove a directory and all its contents including sub-directories using the option `-r` which stands for `--recursive` and removes directories and their contents recursively. Let us first stage a disposable copy as `myproject1`, then remove it and all its contents.

```
cp -r myproject myproject1
rm -r myproject1
ls
```

```
analysis.R
bash.R
bash.Rmd
bash.sh
cran_log_sample.csv.gz
imports_blorr.txt
imports_olsrr.txt
lorem-ipsum.txt
main_project.zip
myfiles
mypackage
myproject
package_names.txt
pkg_names.txt
r
r2
r_releases
release_names.tar
release_names.tar.gz
release_names.txt
release_names_18.txt
release_names_19.txt
urls.txt
zip_example.zip
```

Force Removal

Use the `-f` option which stands for `--force` to forciby remove directory and all its contents without prompting for confirmation or showing error messages. Let us first stage a disposable copy as `myproject2`, then remove it and all its contents.

```
cp -r myproject myproject2
rm -rf myproject2
ls
```

```
analysis.R
bash.R
bash.Rmd
bash.sh
cran_log_sample.csv.gz
imports_blorr.txt
imports_olsrr.txt
lorem-ipsum.txt
main_project.zip
myfiles
mypackage
myproject
package_names.txt
pkg_names.txt
r
r2
r_releases
release_names.tar
release_names.tar.gz
release_names.txt
release_names_18.txt
release_names_19.txt
urls.txt
zip_example.zip
```

Verbose Mode

Remove files in verbose mode, printing a message for each removed file. This is useful when you want to see the details of the files being removed. In the below example, we will remove all files with `.txt` extension from the `myfiles` folder. Instead of specifying the name of each text file, we use the wildcard * along with `.txt` i.e. any file with the extension `.txt` will be removed.

```
cd myfiles
rm -v *.txt
```

```
removed 'release_names.txt'
removed 'release_names_18.txt'
removed 'release_names_19.txt'
```

Compare Files

`diff` stands for difference. It is used to compare files line by line and display differences. It also indicates which lines in one file must be changed to make the files identical. Using additional options, we can

- ignore white spaces while comparing files
- show differences sidy by side
- show differences in unified format
- compare directories recursively
- display names of files that differ

Command	Description
`diff`	Compare files & directories
`diff -w`	Compare files; ignoring white spaces
`diff -y`	Compare files; showing differences side by side
`diff -u`	Compare files; show differences in unified format
`diff -r`	Compare directories recursively
`diff -rq`	Compare directories; show the names of files that differ

Compare Files

Let us compare the contents of the following files

- `imports_olsrr.txt`
- `imports_blorr.txt`

The files contain the names of R packages imported by the olsrr and blorr packages respectively (**Full disclosure: both the above R pakages are developed by Rsquared Academy.**).

`diff` uses certain special symbols and gives instructions to make the files identical. The instructions are on how to change the first file to make it identical to the second file. We list the symbols below

- **a** for add
- **c** for change
- **d** for delete

We will use the `-w` option to ignore white spaces while comparing the files.

```
diff -w imports_olsrr.txt imports_blorr.txt || true
```

```
4a5
> caret
```

Let us interpret the results. `4a5` indicates **after line 4 in file 1, add line 5 from file 2** to make both the files identical i.e. add `caret` which is line 5 in `imports_blorr.txt` after line 4 in `imports_olsrr.txt` which will make both the files identical.

Let us change the file order and see the instructions from `diff`.

```
diff -w imports_blorr.txt imports_olsrr.txt || true
```

```
5d4
< caret
```

`5d4` indicates **delete line 5 from file 1 to match both the files at line4** i.e. delete `caret` which is line 5 in `imports_blorr.txt` to make both the files identical.

Side By Side

To view the differences between the files side by side, use the `-y` option.

```
diff -y imports_olsrr.txt imports_blorr.txt || true
```

```
car                                 | car
checkmate                           | checkmate
cli                                 | cli
clisymbols                          | clisymbols
                                    > caret
```

Unified Format

To view the differences between the files in a unified format, use the `-u` option.

```
diff -u imports_olsrr.txt imports_blorr.txt || true
```

```
--- imports_olsrr.txt   2021-06-02 11:44:47.000000000 +0000
+++ imports_blorr.txt   2021-06-02 11:44:47.000000000 +0000
@@ -1,4 +1,5 @@
-car
-checkmate
-cli
-clisymbols
+car
+checkmate
+cli
+clisymbols
+caret
```

Compare Recursively

To compare recursively, use the `-r` option. Let us compare the `mypackage` and `myproject` folders.

```
diff -r mypackage myproject || true
```

```
Only in mypackage: .Rbuildignore
Only in mypackage: DESCRIPTION
Only in mypackage: LICENSE
Only in mypackage: NAMESPACE
Only in mypackage: NEWS.md
Only in mypackage: R
Only in myproject/data: processed
Only in myproject/data: raw
Only in mypackage: docs
Only in mypackage: man
Only in myproject: output
Only in myproject: run_analysis.R
Only in mypackage: tests
Only in mypackage: vignettes
```

File Details

To compare directories and view the names of files that differ, use the `-rq` option. In the below example, we look at the names of files that differ in `mypackage` and `myproject` folders.

```
diff -rq mypackage myproject || true
```

```
Only in mypackage: .Rbuildignore
Only in mypackage: DESCRIPTION
Only in mypackage: LICENSE
Only in mypackage: NAMESPACE
Only in mypackage: NEWS.md
Only in mypackage: R
Only in myproject/data: processed
Only in myproject/data: raw
Only in mypackage: docs
Only in mypackage: man
Only in myproject: output
Only in myproject: run_analysis.R
Only in mypackage: tests
Only in mypackage: vignettes
```

R Functions

In R, file operations can be performed using functions from both base R and the fs package.

Command	R
`touch`	`file.create()` / `fs::file_create()` / `fs::file_touch()`
`cp`	`file.copy()` / `fs::file_copy()` / `fs::dir_copy()`
`mv`	`file.rename()` / `fs::file_move()`
`rm`	`file.remove()` / `fs::file_delete()`
`diff`	

Exercises

1. In `cline/`, run `touch myexercise.R` then `ls` and confirm the new empty file appears. Then run `cp release_names.txt release_names_ex.txt` and `ls` — you should see both files. Clean up with `rm myexercise.R release_names_ex.txt`.
2. In `cline/`, run `mkdir -p ex_releases`, then `cp release_names.txt ex_releases/` and check with `ls ex_releases` — you should see 1 file. Then run `touch myexercise2.R` followed by `mv -v myexercise2.R ex_releases/` and observe the verbose message. Verify with `ls ex_releases` (2 files), then clean up with `rm -rf ex_releases`.
3. In `cline/`, run `diff imports_olsrr.txt imports_blorr.txt || true`. Note the short difference block with `a`/`d`/`c` instructions and which package name is the extra line. Then re-run with `diff -y` (side-by-side) and `diff -u` (unified) — describe in one sentence each how the views differ.

Worked solutions in `solutions/file-management.md` (attempt first).

Input/Output

In this chapter, we will explore commands that will

- display messages
- print file contents
- sort file contents

Command	Description
`echo`	Display messages
`cat`	Print contents of a file
`head`	Prints first ten lines of a file by default
`tail`	Prints last ten lines of a file by default
`more`	Open a file for interactive reading, scrolling & searching
`less`	Open a file for interactive reading, scrolling & searching
`sort`	Sort a file in ascending order

Display Messages

The `echo` command prints text to the terminal. It can be used for writing or appending messages to a file as well.

Command	Description
`echo`	Display messages
`echo -n`	Print message without trailing new line
`echo > file`	Write message to a file
`echo >> file`	Append message to a file
`echo -e`	Enable interpretation of special characters

Print Message

Let us start with a simple example. We will print the text `Action of the Toes` to the terminal. It is the release name for R version 3.6.1.

```
echo Action of the Toes
```

```
Action of the Toes
```

Redirect Output

What if we want to redirect the output? Instead of printing the text to the terminal, we want to write it to a file. In such cases, use > along with the file name to redirect the output to the file. Keep in mind that > will overwrite files. If you want to append to files instead of overwriting, use >>.

```
echo Great Truth > release.txt
```

Print & Concatenate Files

The `cat` command reads data from files, and outputs their contents. It is the simplest way to display the contents of a file at the command line. It can be used to overwrite or append new content to files as well. `cat` stands for **catenate** and can be used to

- display text files
- copy text files into a new document
- append the contents of a text file to the end of another text file, combining them

Command	Description
`cat`	Print & concatenate files
`cat >`	Concatenate several files into the target file
`cat >>`	Append several files into the target file
`cat -n`	Number all output lines

Print Content

Let us print the content of the `release_names.txt` file (it contains R release names).

```
cat release_names.txt
```

```
Unsuffered Consequences
Great Pumpkin
December Snowflakes
Gift-Getting Season
Easter Beagle
Roasted Marshmallows
Trick or Treat
Security Blanket
Masked Marvel
Good Sport
Frisbee Sailing
Warm Puppy
Spring Dance
Sock it to Me
Pumpkin Helmet
Smooth Sidewalk
Full of Ingredients
World-Famous Astronaut
Fire Safety
Wooden Christmas Tree
Very Secure Dishes
Very, Very Secure Dishes
Supposedly Educational
Bug in Your Hair
Sincere Pumpkin Patch
```

```
Another Canoe
You Stupid Darkness
Single Candle
Short Summer
Kite Eating Tree
```

Number All Output Lines

If you want to number the output line, use the `-n` option.

```
cat -n release_names.txt
```

```
     1  Unsuffered Consequences
     2  Great Pumpkin
     3  December Snowflakes
     4  Gift-Getting Season
     5  Easter Beagle
     6  Roasted Marshmallows
     7  Trick or Treat
     8  Security Blanket
     9  Masked Marvel
    10  Good Sport
    11  Frisbee Sailing
    12  Warm Puppy
    13  Spring Dance
    14  Sock it to Me
    15  Pumpkin Helmet
    16  Smooth Sidewalk
    17  Full of Ingredients
    18  World-Famous Astronaut
    19  Fire Safety
    20  Wooden Christmas Tree
    21  Very Secure Dishes
    22  Very, Very Secure Dishes
    23  Supposedly Educational
    24  Bug in Your Hair
    25  Sincere Pumpkin Patch
    26  Another Canoe
    27  You Stupid Darkness
    28  Single Candle
    29  Short Summer
    30  Kite Eating Tree
```

Concatenate Several Files

To concatenate the contents of several files into a target file, use `>`. In the below example, we concatenate the contents of the files `release_names_18.txt` and `release_names_19.txt` into a single file `release_names_18_19.txt`. In this case we are not printing the contents of the file to the terminal and instead we concatenate the contents from both the files and redirect the output to the target file.

```
cat release_names_18.txt release_names_19.txt > release_names_18_19.txt
cat release_names_18_19.txt
```

```
Someone to Lean On
Joy in Playing
Feather Spray
Eggshell IglooGreat Truth
Planting of a Tree
Action of the Toes
```

Head

The `head` command will display the firt 10 lines of a file(s) by default. It can be used to display the first few lines or bytes of a file as well.

Command	Description
`head`	Output the first parts of a file
`head -n`	Output the first n lines of a file
`head -c`	Output the first c bytes of a file
`head -n -x`	Output everything but the last x lines of a file
`head -c -x`	Output everything but the last x bytes of a file

Output the first parts of a file

Let us use `head` to display the first 10 lines of the `release_names.txt` file.

```
head release_names.txt
```

```
Unsuffered Consequences
Great Pumpkin
December Snowflakes
Gift-Getting Season
Easter Beagle
Roasted Marshmallows
Trick or Treat
Security Blanket
Masked Marvel
Good Sport
```

Output the first n lines of a file

Using the `n` option, we can specify the number of lines to be displayed. In the below example, we display the first 5 lines.

```
head -n 5 release_names.txt
```

```
Unsuffered Consequences
Great Pumpkin
December Snowflakes
```

```
Gift-Getting Season
Easter Beagle
```

Output the first c bytes of a file

The `c` option can be used to display characters or bytes instead of lines. Let us display the first 5 bytes of the `release_names.txt` file.

```
head -c 5 release_names.txt
```

```
Unsuf
```

Output everything but the last 5 lines of a file

To display the last parts of a file, use `-` while specifying the number of lines. In the below example, we display the last 5 lines of the file.

```
head -n -5 release_names.txt
```

```
Unsuffered Consequences
Great Pumpkin
December Snowflakes
Gift-Getting Season
Easter Beagle
Roasted Marshmallows
Trick or Treat
Security Blanket
Masked Marvel
Good Sport
Frisbee Sailing
Warm Puppy
Spring Dance
Sock it to Me
Pumpkin Helmet
Smooth Sidewalk
Full of Ingredients
World-Famous Astronaut
Fire Safety
Wooden Christmas Tree
Very Secure Dishes
Very, Very Secure Dishes
Supposedly Educational
Bug in Your Hair
Sincere Pumpkin Patch
```

Output everything but the last 3 bytes of a file

In this example, we display the last 3 bytes of the file using the `c` option and `-` while specifying the number of bytes.

```
head -c -3 release_names.txt
```

```
Unsuffered Consequences
Great Pumpkin
```

```
December Snowflakes
Gift-Getting Season
Easter Beagle
Roasted Marshmallows
Trick or Treat
Security Blanket
Masked Marvel
Good Sport
Frisbee Sailing
Warm Puppy
Spring Dance
Sock it to Me
Pumpkin Helmet
Smooth Sidewalk
Full of Ingredients
World-Famous Astronaut
Fire Safety
Wooden Christmas Tree
Very Secure Dishes
Very, Very Secure Dishes
Supposedly Educational
Bug in Your Hair
Sincere Pumpkin Patch
Another Canoe
You Stupid Darkness
Single Candle
Short Summer
Kite Eating Tre
```

Tail

The `tail` command displays the last 10 lines of a file(s) by default. It can be used to display the last few lines or bytes of a file as well.

Command	Description
`tail`	Display the last part of a file
`tail -n num`	Show the last num lines of a file
`tail -n +num`	Show all contents of the file starting from num line
`tail -c num`	Show last num bytes of a file
`tail -f`	Keep reading file until Ctrl + C
`tail -F`	Keep reading file until Ctrl + C; even if the file is rotated

Display the last parts of a file

Let us use `tail` to display the last 10 lines of the file.

```
tail release_names.txt
```

```
Very Secure Dishes
Very, Very Secure Dishes
Supposedly Educational
Bug in Your Hair
Sincere Pumpkin Patch
Another Canoe
You Stupid Darkness
Single Candle
Short Summer
Kite Eating Tree
```

Display the last 5 lines of a file

As we did in the previous section, use `n` to specify the number of lines to be displayed.

```
tail -n 5 release_names.txt
```

```
Another Canoe
You Stupid Darkness
Single Candle
Short Summer
Kite Eating Tree
```

Display all contents from line 10

We can use `tail` to display all contents of a file starting from a specific line. In the below example, we display all contents of the file starting from the 10th line using the `n` option and `+` prefix while specifying the number of lines.

```
tail -n +10 release_names.txt
```

```
Good Sport
Frisbee Sailing
Warm Puppy
Spring Dance
Sock it to Me
Pumpkin Helmet
Smooth Sidewalk
Full of Ingredients
World-Famous Astronaut
Fire Safety
Wooden Christmas Tree
Very Secure Dishes
Very, Very Secure Dishes
Supposedly Educational
Bug in Your Hair
Sincere Pumpkin Patch
Another Canoe
You Stupid Darkness
Single Candle
Short Summer
Kite Eating Tree
```

Display the last 10 bytes of a file

Use the `c` option to display the last 7 bytes of a file.

```
tail -c 7 release_names.txt
```

```
Tree
```

More

The `more` command displays text, one screen at a time. It opens a file for

- interactive reading
- scrolling
- and searching

Press `space` to scroll down the page, the forward slash (`/`) for searching strings, `n` to go to the next match and `q` to quit.

Command	Description
`more`	Open a file for interactive reading, scrolling & searching
`space`	Page down
`/`	Search for a string; press `n` to go the next match
`q`	Quit

Less

The `less` command is similar to `more` but offers more features. It allows the user to scroll up and down, go to the beggining and end of the file, forward and backward search and the ability to go the next and previous match while searching the file.

Command	Description
`less`	Open a file for interactive reading, scrolling & searching
`space`	Page down
`b`	Page up
`G`	Go to the end of file
`g`	Go to the start of file
`/`	Forward search
`?`	Backward search
`n`	Go to next match
`N`	Go to previous match
`q`	Quit

Sort

The `sort` command will sort the contents of text file, line by line. Using additional options, we can

- sort a file in ascending/descending order
- ignore case while sorting
- use numeric order for sorting
- preserve only unique lines while sorting

Using the `sort` command, the contents can be sorted numerically and alphabetically. By default, the rules for sorting are:

- lines starting with a number will appear before lines starting with a letter.
- lines starting with a letter that appears earlier in the alphabet will appear before lines starting with a letter that appears later in the alphabet.
- lines starting with a lowercase letter will appear before lines starting with the same letter in uppercase.

Using additional options, the rules for sorting can be changed. We list the options in the below table.

Command	Description
`sort`	Sort lines of text files
`sort -r`	Sort a file in descending order
`sort --ignore-case`	Ignore case while sorting
`sort -n`	Use numeric order for sorting
`sort -u`	Preserve only unique lines while sorting

Sort

Let us sort the contents of the `pkg_names.txt` file. It contains names R packages randomly selected from CRAN.

```
sort pkg_names.txt
```

```
ASIP
AdMit
AnalyzeTS
AzureStor
AzureStor
BIGDAWG
BIOMASS
BIOMASS
BenfordTests
BinOrdNonNor
BioCircos
ClimMobTools
CombinePValue
```

Eagle
FField
ICAOD
MARSS
MIAmaxent
MIAmaxent
MIAmaxent
MVB
MVTests
MaXact
MaxentVariableSelection
OptimaRegion
OxyBS
PathSelectMP
PropScrRand
RJDBC
RPyGeo
SCRT
SMARTp
SPEDInstabR
SemiParSampleSel
SetMethods
SmallCountRounding
SpatioTemporal
SphericalK
SuppDists
Survgini
TIMP
TSeriesMMA
VineCopula
WGScan
WPKDE
accept
accept
addhaz
alfr
aweek
aweek
bayesbio
blink
breakfast
cbsem
corclass
crsra
cyclocomp
dagitty
disparityfilter
edfReader
errorlocate

```
expstudies
fermicatsR
foretell
gLRTH
gazepath
generalhoslem
geoknife
hdnom
hindexcalculator
ibd
interplot
kfigr
logNormReg
ltxsparklines
lue
mbir
mcmcabn
mev
mgcViz
mined
mlflow
mongolite
mongolite
mvShapiroTest
odk
overlapping
pAnalysis
pls
pmdplyr
poisbinom
randtests
redcapAPI
rgw
rless
rsed
rstudioapi
solitude
splithalfr
sspline
sybilccFBA
tailr
tailr
tictactoe
viridisLite
vqtl
widyr
widyr
```

Descending Order

Using the `-r` option which stands for `--reverse` the contents of the file can be sorted in descending/reverse order. Let us now sort the contents of the `pkg_names.txt` file in reverse order.

```
sort -r pkg_names.txt
```

```
widyr
widyr
vqtl
viridisLite
tictactoe
tailr
tailr
sybilccFBA
sspline
splithalfr
solitude
rstudioapi
rsed
rless
rgw
redcapAPI
randtests
poisbinom
pmdplyr
pls
pAnalysis
overlapping
odk
mvShapiroTest
mongolite
mongolite
mlflow
mined
mgcViz
mev
mcmcabn
mbir
lue
ltxsparklines
logNormReg
kfigr
interplot
ibd
hindexcalculator
hdnom
geoknife
generalhoslem
gazepath
gLRTH
```

foretell
fermicatsR
expstudies
errorlocate
edfReader
disparityfilter
dagitty
cyclocomp
crsra
corclass
cbsem
breakfast
blink
bayesbio
aweek
aweek
alfr
addhaz
accept
accept
WPKDE
WGScan
VineCopula
TSeriesMMA
TIMP
Survgini
SuppDists
SphericalK
SpatioTemporal
SmallCountRounding
SetMethods
SemiParSampleSel
SPEDInstabR
SMARTp
SCRT
RPyGeo
RJDBC
PropScrRand
PathSelectMP
OxyBS
OptimaRegion
MaxentVariableSelection
MaXact
MVTests
MVB
MIAmaxent
MIAmaxent
MIAmaxent
MARSS

```
ICAOD
FField
Eagle
CombinePValue
ClimMobTools
BioCircos
BinOrdNonNor
BenfordTests
BIOMASS
BIOMASS
BIGDAWG
AzureStor
AzureStor
AnalyzeTS
AdMit
ASIP
```

Ignore case

To ignore case while sorting contents, use the `--ignore-case` option. Time to sort the `pkg_names.txt` file while ignoring case.

```
sort --ignore-case pkg_names.txt
```

```
accept
accept
addhaz
AdMit
alfr
AnalyzeTS
ASIP
aweek
aweek
AzureStor
AzureStor
bayesbio
BenfordTests
BIGDAWG
BinOrdNonNor
BioCircos
BIOMASS
BIOMASS
blink
breakfast
cbsem
ClimMobTools
CombinePValue
corclass
crsra
cyclocomp
dagitty
```

disparityfilter
Eagle
edfReader
errorlocate
expstudies
fermicatsR
FField
foretell
gazepath
generalhoslem
geoknife
gLRTH
hdnom
hindexcalculator
ibd
ICAOD
interplot
kfigr
logNormReg
ltxsparklines
lue
MARSS
MaXact
MaxentVariableSelection
mbir
mcmcabn
mev
mgcViz
MIAmaxent
MIAmaxent
MIAmaxent
mined
mlflow
mongolite
mongolite
MVB
mvShapiroTest
MVTests
odk
OptimaRegion
overlapping
OxyBS
pAnalysis
PathSelectMP
pls
pmdplyr
poisbinom
PropScrRand
randtests

```
redcapAPI
rgw
RJDBC
rless
RPyGeo
rsed
rstudioapi
SCRT
SemiParSampleSel
SetMethods
SmallCountRounding
SMARTp
solitude
SpatioTemporal
SPEDInstabR
SphericalK
splithalfr
sspline
SuppDists
Survgini
sybilccFBA
tailr
tailr
tictactoe
TIMP
TSeriesMMA
VineCopula
viridisLite
vqtl
WGScan
widyr
widyr
WPKDE
```

Numeric Order

To sort numerically, use the `-n` option which stands for `--numeric-sort`. In this example, we will use a different file, `package_names.txt` where the package names are prefixed by random numbers between 1 and 100.

```
sort -n package_names.txt
```

```
1. cyclocomp
2. odk
3. redcapAPI
4. TIMP
5. pls
6. BinOrdNonNor
7. bayesbio
8. MVTests
9. pAnalysis
```

10. aweek
11. hdnom
12. ltxsparklines
13. MaXact
14. RJDBC
15. MIAmaxent
16. randtests
17. ASIP
18. gazepath
19. mcmcabn
20. rless
21. corclass
22. vqtl
23. disparityfilter
24. SCRT
25. RPyGeo
26. blink
27. gLRTH
28. splithalfr
29. sspline
29. sspline
30. logNormReg
31. BIGDAWG
31. BIGDAWG
32. SPEDInstabR
33. tailr
33. tailr
34. ibd
35. fermicatsR
36. mlflow
37. CombinePValue
38. BenfordTests
39. mev
40. MaxentVariableSelection
41. rstudioapi
42. OptimaRegion
43. accept
44. expstudies
45. solitude
45. solitude
46. cbsem
47. SMARTp
48. geoknife
49. SemiParSampleSel
50. mbir
51. interplot
52. ClimMobTools
53. MVB
54. OxyBS

55. hindexcalculator
56. MARSS
57. generalhoslem
58. alfr
59. AdMit
60. Eagle
61. PropScrRand
62. lue
63. dagitty
64. viridisLite
65. mined
65. mined
66. SuppDists
67. tictactoe
68. AzureStor
68. AzureStor
69. FField
70. rsed
70. rsed
71. kfigr
72. overlapping
72. overlapping
73. VineCopula
74. crsra
75. pmdplyr
76. errorlocate
77. SetMethods
78. sybilccFBA
79. mvShapiroTest
80. SpatioTemporal
81. mgcViz
82. breakfast
83. WPKDE
84. BIOMASS
85. edfReader
86. mongolite
87. WGScan
88. SphericalK
89. foretell
90. widyr
91. rgw
92. BioCircos
93. PathSelectMP
94. ICAOD
95. TSeriesMMA
96. poisbinom
97. AnalyzeTS
98. SmallCountRounding

```
99. Survgini
100. addhaz
```

Preserve Only Unique Lines

The `-u` option which stands for `--unique` will preserve only unique lines while sorting the contents of the file. In the below example, we remove all duplicate lines from the `pkg_names.txt` while sorting.

```
sort -u pkg_names.txt
```

```
ASIP
AdMit
AnalyzeTS
AzureStor
BIGDAWG
BIOMASS
BenfordTests
BinOrdNonNor
BioCircos
ClimMobTools
CombinePValue
Eagle
FField
ICAOD
MARSS
MIAmaxent
MVB
MVTests
MaXact
MaxentVariableSelection
OptimaRegion
OxyBS
PathSelectMP
PropScrRand
RJDBC
RPyGeo
SCRT
SMARTp
SPEDInstabR
SemiParSampleSel
SetMethods
SmallCountRounding
SpatioTemporal
SphericalK
SuppDists
Survgini
TIMP
TSeriesMMA
VineCopula
WGScan
```

WPKDE
accept
accept
addhaz
alfr
aweek
bayesbio
blink
breakfast
cbsem
corclass
crsra
cyclocomp
dagitty
disparityfilter
edfReader
errorlocate
expstudies
fermicatsR
foretell
gLRTH
gazepath
generalhoslem
geoknife
hdnom
hindexcalculator
ibd
interplot
kfigr
logNormReg
ltxsparklines
lue
mbir
mcmcabn
mev
mgcViz
mined
mlflow
mongolite
mvShapiroTest
odk
overlapping
pAnalysis
pls
pmdplyr
poisbinom
randtests
redcapAPI
rgw

```
rless
rsed
rstudioapi
solitude
splithalfr
sspline
sybilccFBA
tailr
tictactoe
viridisLite
vqtl
widyr
```

Word Count

wc will print newline, word, and byte counts for file(s). If more than one file is specified, it will also print total line.

Count words, bytes and lines

```
wc release_names.txt
```

```
 30  73 546 release_names.txt
```

Count lines in a file

```
wc -l release_names.txt
```

```
30 release_names.txt
```

Count words in a file

```
wc -w release_names.txt
```

```
73 release_names.txt
```

Count characters(bytes) in a file

```
wc -c release_names.txt
```

```
546 release_names.txt
```

Exercises

1. Run `echo Great Truth > release_ex.txt` followed by `cat release_ex.txt` — you should see just one line. Then run `echo Action of the Toes >> release_ex.txt` and `cat release_ex.txt` again — you should now see 2 lines. Explain in your own words the difference between `>` and `>>`, then clean up with `rm release_ex.txt`.
2. In `cline/`, run `cat -n release_names.txt` (content with line numbers), then `head -n 5 release_names.txt` (first 5 lines) and `tail -n 5 release_names.txt` (last 5 lines). Use `wc -l release_names.txt` to get the total line count and check your head/tail outputs are consistent with it.
3. In `cline/`, run `sort pkg_names.txt`, then `sort -r pkg_names.txt` (first and last lines swapped), then `sort -u pkg_names.txt` (duplicates removed — compare counts with `wc -`

`l`). Then run `sort -n package_names.txt` on the number-prefixed file — lines should order numerically rather than alphabetically.

Worked solutions in `solutions/input-output.md` (attempt first).

Search & Regular Expression

In this chapter, we will explore commands that will

- search for a given string in a file
- find files using names
- and search for binary executable files

Command	Description
`grep`	Search for a given string in a file
`find`	Find files using filenames
`which`	Search for binary executable files

grep

The `grep` command is used for pattern matching. Along with additional options, it can be used to

- match pattern in input text
- ignore case
- search recursively for an exact string
- print filename and line number for each match
- invert match for excluding specific strings

`grep` processes text line by line, and prints any lines which match a specified pattern. `grep`, which stands for **g**lobal **r**egular **e**xpression **p**rint is a powerful tool for matching a regular expression against text in a file, multiple files, or a stream of input.

Command	Description
`grep`	Matches pattern in input text
`grep -i`	Ignore case
`grep -RI`	Search recursively for an exact string
`grep -E`	Use extended regular expression
`grep -Hn`	Print file name & corresponding line number for each match
`grep -v`	Invert match for excluding specific strings

Match Pattern in Input Text

Using `grep` let us search for packages that inlcude the letter `R` in their names.

```
grep R package_names.txt
```

```
14. RJDBC
30. logNormReg
27. gLRTH
35. fermicatsR
42. OptimaRegion
61. PropScrRand
25. RPyGeo
47. SMARTp
24. SCRT
56. MARSS
85. edfReader
32. SPEDInstabR
98. SmallCountRounding
```

Ignore Case

In the previous case, `grep` returned only those packages whose name included `R` but not `r` i.e. it did not ignore the case of the letter. Using the `-i` option, we will now search while ignoring the case of the letter.

```
grep -i R package_names.txt
```

```
14. RJDBC
58. alfr
64. viridisLite
99. Survgini
30. logNormReg
27. gLRTH
71. kfigr
72. overlapping
90. widyr
33. tailr
40. MaxentVariableSelection
33. tailr
72. overlapping
16. randtests
12. ltxsparklines
91. rgw
35. fermicatsR
21. corclass
68. AzureStor
42. OptimaRegion
61. PropScrRand
74. crsra
80. SpatioTemporal
23. disparityfilter
49. SemiParSampleSel
76. errorlocate
88. SphericalK
```

```
28. splithalfr
89. foretell
25. RPyGeo
50. mbir
51. interplot
6. BinOrdNonNor
47. SMARTp
38. BenfordTests
79. mvShapiroTest
92. BioCircos
55. hindexcalculator
41. rstudioapi
57. generalhoslem
24. SCRT
95. TSeriesMMA
82. breakfast
56. MARSS
70. rsed
68. AzureStor
85. edfReader
20. rless
75. pmdplyr
32. SPEDInstabR
3. redcapAPI
70. rsed
98. SmallCountRounding
```

Highlight

The `--color` option will highlight the matched strings.

```
grep -i --color R package_names.txt
```

```
14. RJDBC
58. alfr
64. viridisLite
99. Survgini
30. logNormReg
27. gLRTH
71. kfigr
72. overlapping
90. widyr
33. tailr
40. MaxentVariableSelection
33. tailr
72. overlapping
16. randtests
12. ltxsparklines
91. rgw
35. fermicatsR
21. corclass
```

```
68. AzureStor
42. OptimaRegion
61. PropScrRand
74. crsra
80. SpatioTemporal
23. disparityfilter
49. SemiParSampleSel
76. errorlocate
88. SphericalK
28. splithalfr
89. foretell
25. RPyGeo
50. mbir
51. interplot
6. BinOrdNonNor
47. SMARTp
38. BenfordTests
79. mvShapiroTest
92. BioCircos
55. hindexcalculator
41. rstudioapi
57. generalhoslem
24. SCRT
95. TSeriesMMA
82. breakfast
56. MARSS
70. rsed
68. AzureStor
85. edfReader
20. rless
75. pmdplyr
32. SPEDInstabR
3. redcapAPI
70. rsed
98. SmallCountRounding
```

Print Filename

If there is more than one file to search, use the `-H` option to print the filename for each match.

```
grep -i --color -H bio package_names.txt
```

```
package_names.txt:84. BIOMASS
package_names.txt:92. BioCircos
package_names.txt:7. bayesbio
```

Print Corresponding Line Number

The `-n` option will print the corresponding line number of the match in the file.

```
grep -i --color -n bio package_names.txt
```

```
59:84. BIOMASS
71:92. BioCircos
88:7. bayesbio
```

Print Filename & Line Number

Let us print both the file name and the line number for each match.

```
grep -i --color -Hn R package_names.txt
```

```
package_names.txt:1:14. RJDBC
package_names.txt:3:58. alfr
package_names.txt:8:64. viridisLite
package_names.txt:14:99. Survgini
package_names.txt:15:30. logNormReg
package_names.txt:16:27. gLRTH
package_names.txt:18:71. kfigr
package_names.txt:20:72. overlapping
package_names.txt:21:90. widyr
package_names.txt:22:33. tailr
package_names.txt:23:40. MaxentVariableSelection
package_names.txt:26:33. tailr
package_names.txt:27:72. overlapping
package_names.txt:30:16. randtests
package_names.txt:31:12. ltxsparklines
package_names.txt:32:91. rgw
package_names.txt:33:35. fermicatsR
package_names.txt:37:21. corclass
package_names.txt:38:68. AzureStor
package_names.txt:41:42. OptimaRegion
package_names.txt:42:61. PropScrRand
package_names.txt:43:74. crsra
package_names.txt:51:80. SpatioTemporal
package_names.txt:52:23. disparityfilter
package_names.txt:54:49. SemiParSampleSel
package_names.txt:55:76. errorlocate
package_names.txt:57:88. SphericalK
package_names.txt:61:28. splithalfr
package_names.txt:62:89. foretell
package_names.txt:63:25. RPyGeo
package_names.txt:64:50. mbir
package_names.txt:65:51. interplot
package_names.txt:66:6. BinOrdNonNor
package_names.txt:67:47. SMARTp
package_names.txt:68:38. BenfordTests
package_names.txt:69:79. mvShapiroTest
package_names.txt:71:92. BioCircos
package_names.txt:75:55. hindexcalculator
package_names.txt:78:41. rstudioapi
package_names.txt:80:57. generalhoslem
package_names.txt:84:24. SCRT
```

```
package_names.txt:85:95. TSeriesMMA
package_names.txt:87:82. breakfast
package_names.txt:96:56. MARSS
package_names.txt:97:70. rsed
package_names.txt:98:68. AzureStor
package_names.txt:100:85. edfReader
package_names.txt:101:20. rless
package_names.txt:102:75. pmdplyr
package_names.txt:103:32. SPEDInstabR
package_names.txt:104:3. redcapAPI
package_names.txt:106:70. rsed
package_names.txt:107:98. SmallCountRounding
```

Invert Match

Use the `-v` option to select non-matching lines. In the below example, we search for packages whose name does not include `R` while ignoring the case.

```
grep -v -i R package_names.txt
```

```
36. mlflow
10. aweek
31. BIGDAWG
22. vqtl
29. sspline
39. mev
66. SuppDists
15. MIAmaxent
31. BIGDAWG
29. sspline
60. Eagle
83. WPKDE
11. hdnom
26. blink
18. gazepath
52. ClimMobTools
44. expstudies
65. mined
81. mgcViz
45. solitude
9. pAnalysis
65. mined
94. ICAOD
48. geoknife
45. solitude
67. tictactoe
46. cbsem
93. PathSelectMP
96. poisbinom
17. ASIP
5. pls
```

```
84. BIOMASS
59. AdMit
77. SetMethods
53. MVB
2. odk
86. mongolite
4. TIMP
97. AnalyzeTS
87. WGScan
63. dagitty
69. FField
13. MaXact
73. VineCopula
7. bayesbio
34. ibd
8. MVTests
19. mcmcabn
43. accept
78. sybilccFBA
62. lue
100. addhaz
37. CombinePValue
1. cyclocomp
54. OxyBS
```

Recursive Search

Use the `-r` option to search recursively. In the below example, we search all files with the `.txt` extension for the string `bio` while ignoring the case.

```
grep -i --color -r bio *.txt
```

```
package_names.txt:84. BIOMASS
package_names.txt:92. BioCircos
package_names.txt:7. bayesbio
pkg_names.txt:BIOMASS
pkg_names.txt:BioCircos
pkg_names.txt:BIOMASS
pkg_names.txt:bayesbio
```

find

The `find` command can be used for searching files and directories. Using additional options, we can

- search files by extension type
- ignore case while searching files/directories

`find` is a powerful tool for working with the files. It can be used on its own to locate files, or in conjunction with other programs to perform operations on those files.

Command	Description
`find`	Find files or directories under the given directory; recursively
`find -name '*.txt'`	Find files by extension
`find -type d -iname`	Find directories matching a given name, in case-insensitive mode
`find -type d -name`	Find directories matching a given name, in case-sensitive mode

Search Recursively

Let us use `find` to search for the file `release_names.txt` recursively. The `-name` option is used to specify the name of the file we are searching.

```
find -name release_names.txt
```

```
./r_releases/release_names.txt
./release_names.txt
```

There are two files with the name `release_names.txt` present in the current working directory and in `r_releases` directory.

Search by Extension

Let us search for all files with `.txt` extension in the `r_releases` folder.

```
find r_releases -name '*.txt'
```

```
r_releases/release_names_3.txt
r_releases/release_names_2.txt
r_releases/release_names.txt
```

There are 3 files with the `.txt` extension in `r_releases` folder.

Case-insensitive Mode

Search for all folders with the name `R` or `r`. Here we use the `-iname` option to ignore case while searching. The `-type` option is used to specify whether we are searching for files or folders. Since we are searching for folder/directory, we use it along with `d` i.e. directory to indicate that we are searching for directories and not files.

```
find -type d -iname R
```

```
./mypackage/R
./r
```

Case-sensitive Mode

Search for all folders with the name `r`. It should exclude any folder with the name `R`.

```
find -type d -name r
```

`./r`

Exercises

1. In `cline/`, run `grep R package_names.txt`, then repeat with `grep -i R package_names.txt`. Describe the difference: the first matches only uppercase `R`, the second matches both cases and returns more lines.
2. Search for `bio` case-insensitively in `package_names.txt` with `grep -i -Hn bio package_names.txt`. Each matching line should print with its filename prefix and line number.
3. In `cline/`, run `find r_releases -name '*.txt'` — you should see the 3 `.txt` files. Then run `find -type d -name R` and `find -type d -iname R` and compare: `-name` is case-sensitive while `-iname` matches both `R` and `r` directories.

Worked solutions in `solutions/regex.md` (attempt first).

Data Transfer

In this chapter, we will explore commands that will allow us to download files from the internet.

Command	Description
`wget`	Download files from the web
`curl`	Transfer data from or to a server
`hostname`	Name of the current host
`ping`	Ping a remote host
`nslookup`	Name server details

Remote-download commands below are shown but not executed during automated builds (rendering stays offline so builds never depend on the network). Each tool has an offline demo using committed fixtures, and every output you see in this chapter was produced from those fixtures.

wget

The `wget` command will download contents of a URL and files from the internet. Using additional options, we can

- download contents/files to a file
- continue incomplete downloads
- download multiple files
- limit download speed and number of retries

Command	Description
`wget url`	Download contents of a url
`wget -O file url`	Download contents of url to a file
`wget -c`	Continue an incomplete download

Command	Description
`wget -P folder_name -i urls.txt`	Download all urls stored in a text file to a specific directory
`wget --limit-rate`	Limit download speed
`wget --tries`	Limit number of retries
`wget --quiet`	Turn off output
`wget --no-verbose`	Print basic information
`wget --progress-dot`	Change progress bar type to dot
`wget --timestamping`	Check if the timestamp of the file has changed before downloading
`wget --wait`	Wait between retrievals

Download URL

Let us first use `wget` to download contents of a URL. Note, we are not downloading file as such but just the content of the URL. We will use the URL of the home page of R project.

```
wget https://www.r-project.org/
```

Downloading a bare URL saves the response under a default name such as `index.html`, which gets confusing with multiple URLs. Save the contents to an explicitly named file instead (we can specify the name of the file which should help avoid confusion).

Specify Filename

In this example, we download contents from the same URL and in addition specify the name of the file in which the content must be saved. Here we save it in a new file, `rhomepage.html` using the `-O` option followed by the filename.

```
wget -O rhomepage.html https://www.r-project.org/
```

Download File

How about downloading a file instead of a URL? In this example, we will download a logfile from the RStudio CRAN mirror. It contains the details of R downloads and individual package downloads. If you are a package developer and would want to know the countries in which your packages are downloaded, you will find this useful. We will download the file for 29th September and save it as `sep_29.csv.gz`. (The 2019 CRAN-log URLs below illustrate the pattern but the archive layout has since changed, so treat them as illustration and run the offline demo instead.)

```
wget -O sep_29.csv.gz http://cran-logs.rstudio.com/2019/2019-09-29.csv.gz
```

Try the same `-O` pattern offline (no network needed) against a committed fixture:

```
wget -O wget_demo.csv.gz file://$PWD/cran_log_sample.csv.gz
ls -lh wget_demo.csv.gz
```

```
file:///home/runner/work/bash-intro/bash-intro/cline/cran_log_sample.csv.gz:
Unsupported scheme.
-rw-r--r-- 1 runner runner 0 Nov  1 11:34 wget_demo.csv.gz
```

Download Multiple URLs

How do we download multiple URLs? One way is to specify the URLs one after the other separated by a space or save all URLs in a file and read them one by one. In the below example, we have saved multiple URLs in the file `urls.txt`.

```
cat urls.txt
```

```
http://cran-logs.rstudio.com/2019/2019-09-26.csv.gz
http://cran-logs.rstudio.com/2019/2019-09-27.csv.gz
http://cran-logs.rstudio.com/2019/2019-09-28.csv.gz
```

We will download all the above URLs and save them in a new folder `downloads`. The `-i` indicates that the URLs must be read from a file (local or external). The `-P` option allows us to specify the directory into which all the files will be downloaded.

```
wget -P downloads -i urls.txt
```

Quiet

The `--quiet` option will turn off `wget` output. It will not show any of the following details:

- name of the file being saved
- file size
- download speed
- eta etc.

```
wget --quiet http://cran-logs.rstudio.com/2019/2019-10-06.csv.gz
```

No Verbose

Using the `-nv` or `--no-verbose` option, we can turn off verbose without being completely quiet (as we did in the previous example). Any error messages and basic information will still be printed.

```
wget --no-verbose http://cran-logs.rstudio.com/2019/2019-10-13.csv.gz
```

Check Timestamp

Let us say we have already downloaded a file from a URL. The file is updated from time to time and we intend to keep the local copy updated as well. Using the `--timestamping` option, the local file will have timestamp matching the remote file; if the remote file is not newer (not updated), no download will occur i.e. if the timestamp of the remote file has not changed it will not be downloaded. This is very useful in case of large files where you do not want to download them unless they have been updated.

```
wget --timestamping http://cran-logs.rstudio.com/2019/2019-10-13.csv.gz
```

curl

The `curl` command will transfer data from or to a server. We will only look at downloading files from the internet.

Command	Description
`curl url`	Download contents of a url
`curl url -o file`	Download contents of url to a file
`curl url > file`	Download contents of url to a file
`curl -s`	Download in silent or quiet mode

Download URL

Let us download the home page of the R project using `curl`.

```
curl https://www.r-project.org/
```

Specify File

Let us download another log file from the RStudio CRAN mirror and save it into a file using the `-o` option.

```
curl http://cran-logs.rstudio.com/2019/2019-09-08.csv.gz -o sept_08.csv.gz
```

Same pattern, offline, using a committed fixture:

```
curl -o curl_demo.csv.gz file://$PWD/cran_log_sample.csv.gz
ls -lh curl_demo.csv.gz
```

```
  % 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 48185  100 48185    0     0   332M      0 --:--:-- --:--:-- --:--:--  332M
-rw-r--r-- 1 runner runner 48K Nov  1 11:34 curl_demo.csv.gz
```

Another way to save a downloaded file is to use > followed by the name of the file as shown in the below example.

```
curl http://cran-logs.rstudio.com/2019/2019-09-01.csv.gz > sep_01.csv.gz
```

Download Silently

The `-s` option will allow you to download files silently. It will mute `curl` and will not display progress meter or error messages.

```
curl http://cran-logs.rstudio.com/2019/2019-09-01.csv.gz -o sept_01.csv.gz -s
```

R Functions

In R, we can use `download.file()` to download files from the internet. The following packages offer functionalities that you will find useful.

- curl
- R.utils
- pingr

Command	R
`wget`	`download.file()`
`curl`	`curl::curl_download()`
`hostname`	`R.utils::getHostname.System()`
`ping`	`pingr::ping()`
`nslookup`	`curl::nslookup()`

Exercises

1. Offline `wget` copy (no network used): from inside `cline/`, run `wget -O wget_demo.csv.gz file://$PWD/cran_log_sample.csv.gz`, then verify with `ls -lh wget_demo.csv.gz` — the new file should exist with non-zero size. Clean up with `rm wget_demo.csv.gz`.
2. Offline `curl` copy: from inside `cline/`, run `curl -o curl_demo.csv.gz file://$PWD/cran_log_sample.csv.gz`, then compare with the original using `cmp cran_log_sample.csv.gz curl_demo.csv.gz` — silence means identical. Clean up with `rm curl_demo.csv.gz`.
3. Inspect `urls.txt` with `cat`, then write down (do not execute) the `wget -P downloads -i urls.txt` command you would use to fetch all listed URLs into a `downloads/` folder. Explain in one sentence each what `-P` and `-i` do.

Worked solutions in `solutions/data-transfer.md` (attempt first).

sudo

sudo **S**uper **U**ser **DO** is a prefix of commands that only superuser or root users are allowed to run. It is similar to **run as administrator** option in Windows. It is used to install, update and remove software. We will use it in the next section to install & update packages.

Command	Description
`dpkg --list`	List installed packages
`sudo apt-get update`	Update packages
`sudo apt-get install`	Install packages
`sudo apt-get remove`	Remove packages (retain configuration, plugins and settings)
`sudo apt-get purge`	Remove packages including personalized settings
`sudo apt-get autoremove`	Remove any dependencies no longer in use

Exercises

1. Safe listing (no `sudo` needed): run `dpkg --list | head -n 10` and `apt list --installed 2>/dev/null | head -n 10`. Describe in one sentence what kind of information each shows — both list installed software without changing the system.

2. Without running them, write the commands `sudo apt-get update`, `sudo apt-get install <pkg>`, `sudo apt-get remove <pkg>`, `sudo apt-get purge <pkg>`, and `sudo apt-get autoremove`. State which of `remove`/`purge` keeps configuration files and which deletes them.
3. Explain in 2–3 sentences why `apt-get install`/`remove`/`purge` require the `sudo` prefix while `dpkg --list` and `apt list --installed` do not (superuser privileges modify system software; read-only queries are safe for any user).

Worked solutions in `solutions/sudo.md` (attempt first).

File Compression

In this chapter, we will explore file compression using the following

- `tar`
- `gzip`
- `zip`
- `unzip`

tar

The `tar` command is used for file compression. It works with both `tar` and `tar.gz` extensions. It is used to

- list files
- extract files
- create archives
- append file to existing archives

`tar` creates, maintains, modifies, and extracts files that are archived in the tar format. Tar stands for **t**ape **ar**chive and is an archiving file format.

Command	Description
`tar tvf`	List an archive
`tar tvfz`	List a gzipped archive
`tar xvf`	Extract an archive
`tar xvfz`	Extract a gzipped archive
`tar cvf`	Create an uncompressed tar archive
`tar cvfz`	Create a tar gzipped archive
`tar rvf`	Add a file to an existing archive
`tar rvfz`	Add a file to an existing gzipped archive

We will use different options along with the `tar` command for listing, extracting, creating and adding files. The `vf` (**v** stands for verbosely show .tar file progress and **f** stands for file name type of the archive file) option is common for all the above operations while the following are specific.

- `t` for listing
- `x` for extracting
- `c` for creating
- `r` for adding files

While dealing with `tar.gz` archives we will use `z` in addition to `vf` and the above options.

List

Let us list all the files & folders in `release_names.tar`. As mentioned above. to list the files in the archive, we use the `t` option.

```
tar -tvf release_names.tar
```

```
-rwxrwxrwx aravind/aravind 546 2019-09-16 10:29 release_names.txt
-rwxrwxrwx aravind/aravind  65 2019-09-16 10:28 release_names_18.txt
-rwxrwxrwx aravind/aravind  53 2019-09-16 10:29 release_names_19.txt
```

Extract

Let us extract files from `release_names.tar` using the `x` option in addition to `vf`.

```
tar -xvf release_names.tar
ls
```

```
release_names.txt
release_names_18.txt
release_names_19.txt
analysis.R
bash.R
bash.Rmd
bash.sh
cran_log_sample.csv.gz
curl_demo.csv.gz
imports_blorr.txt
imports_olsrr.txt
lorem-ipsum.txt
main_project.zip
myfiles
mypackage
myproject
package_names.txt
pkg_names.txt
r
r2
r_releases
release.txt
release_names.tar
release_names.tar.gz
release_names.txt
release_names_18.txt
release_names_18_19.txt
```

```
release_names_19.txt
urls.txt
wget_demo.csv.gz
zip_example.zip
```

Add

To add a file to an existing archive, use the `r` option. Let us add `release_names_18.txt` and `release_names_19.txt` to the archive we created in the previous step.

```
tar -rvf release_names.tar release_names_18.txt release_names_19.txt
```

```
release_names_18.txt
release_names_19.txt
```

Create

Using the `c` option we can create tar archives. In the below example, we are using a single file but you can specify multiple files and folders as well.

```
tar -cvf pkg_names.tar pkg_names.txt
```

```
pkg_names.txt
```

gzip

Command	Description
`gzip`	Compress a file
`gzip -d`	Decompress a file
`gzip -c`	Compress a file and specify the output file name
`zip -r`	Compress a directory
`zip`	Add files to an existing zip file
`unzip`	Extract files from a zip files
`unzip -d`	Extract files from a zip file and specify the output file name
`unzip -l`	List contents of a zip file

`gzip`, `gunzip`, and `zcat` commands are used to compress or expand files in the GNU GZIP format i.e. files with `.gz` extension

Compress

Let us compress `release_names.txt` file using `gzip`.

```
gzip release_names.txt
ls
gunzip release_names.txt.gz
```

```
analysis.R
bash.R
```

```
bash.Rmd
bash.sh
cran_log_sample.csv.gz
curl_demo.csv.gz
imports_blorr.txt
imports_olsrr.txt
lorem-ipsum.txt
main_project.zip
myfiles
mypackage
myproject
package_names.txt
pkg_names.tar
pkg_names.txt
r
r2
r_releases
release.txt
release_names.tar
release_names.tar.gz
release_names.txt.gz
release_names_18.txt
release_names_18_19.txt
release_names_19.txt
urls.txt
wget_demo.csv.gz
zip_example.zip
```

Decompress

Use the `-d` option with `gzip` to decompress a file. In the below example, we stream the decompressed contents of `cran_log_sample.csv.gz` to the terminal while keeping the fixture compressed for later chapters. You can also use `gunzip` (or `gzip -d` on a disposable copy) for the same result.

```
gzip -dc cran_log_sample.csv.gz | head -n 3
ls
```

```
"date","time","size","r_version","r_arch","r_os","package","version","country","ip_id"
"2019-09-15","02:28:38",47952,NA,NA,NA,"aws.s3","0.3.12","US",1
"2019-09-15","02:28:37",5899,NA,NA,NA,"aws.ec2metadata","0.2.0","US",1
analysis.R
bash.R
bash.Rmd
bash.sh
cran_log_sample.csv.gz
curl_demo.csv.gz
imports_blorr.txt
imports_olsrr.txt
lorem-ipsum.txt
main_project.zip
```

```
myfiles
mypackage
myproject
package_names.txt
pkg_names.tar
pkg_names.txt
r
r2
r_releases
release.txt
release_names.tar
release_names.tar.gz
release_names.txt
release_names_18.txt
release_names_18_19.txt
release_names_19.txt
urls.txt
wget_demo.csv.gz
zip_example.zip
```

Specify Filename

Use `-c` and `>` to specify a different file name while compressing using `gzip`. In the below example, `gzip` will create `releases.txt.gz` instead of `release_names.txt.gz`.

```
gzip -c release_names.txt > releases.txt.gz
ls
```

```
analysis.R
bash.R
bash.Rmd
bash.sh
cran_log_sample.csv.gz
curl_demo.csv.gz
imports_blorr.txt
imports_olsrr.txt
lorem-ipsum.txt
main_project.zip
myfiles
mypackage
myproject
package_names.txt
pkg_names.tar
pkg_names.txt
r
r2
r_releases
release.txt
release_names.tar
release_names.tar.gz
release_names.txt
```

```
release_names_18.txt
release_names_18_19.txt
release_names_19.txt
releases.txt.gz
urls.txt
wget_demo.csv.gz
zip_example.zip
```

zip & unzip

`zip` creates ZIP archives while `unzip` lists and extracts compressed files in a ZIP archive.

List

Let us list all the files and folders in `main_project.zip()` using `unzip` and the `-l` option.

```
unzip -l main_project.zip
```

```
Archive:  main_project.zip
  Length      Date    Time    Name
---------  ---------- -----   ----
        0  2019-09-23 12:37   myproject/
        0  2019-09-20 08:32   myproject/.gitignore
        0  2019-09-23 12:37   myproject/data/
        0  2019-09-20 08:32   myproject/data/processed/
        0  2019-09-20 08:32   myproject/data/raw/
        0  2019-09-20 08:32   myproject/output/
        0  2019-09-20 08:32   myproject/README.md
       13  2019-09-20 08:32   myproject/run_analysis.R
        0  2019-09-20 08:32   myproject/src/
        0  2019-09-23 12:37   mypackage/
        0  2019-09-20 08:41   mypackage/.gitignore
        0  2019-09-20 08:41   mypackage/.Rbuildignore
        0  2019-09-20 08:40   mypackage/data/
        0  2019-09-20 08:41   mypackage/DESCRIPTION
        0  2019-09-20 08:40   mypackage/docs/
        0  2019-09-20 08:41   mypackage/LICENSE
        0  2019-09-20 08:40   mypackage/man/
        0  2019-09-20 08:41   mypackage/NAMESPACE
        0  2019-09-20 08:41   mypackage/NEWS.md
        0  2019-09-20 08:40   mypackage/R/
        0  2019-09-20 08:41   mypackage/README.md
        0  2019-09-20 08:41   mypackage/src/
        0  2019-09-20 08:40   mypackage/tests/
        0  2019-09-20 08:40   mypackage/vignettes/
        0  2019-09-23 12:37   myfiles/
       12  2019-09-20 10:00   myfiles/analysis.R
        7  2019-09-20 10:01   myfiles/NEWS.md
        9  2019-09-20 10:01   myfiles/README.md
      546  2019-09-20 09:59   myfiles/release_names.txt
       65  2019-09-20 09:59   myfiles/release_names_18.txt
       53  2019-09-20 10:00   myfiles/release_names_19.txt
```

```
       12  2019-09-20 10:00   myfiles/visualization.R
    15333  2019-10-01 11:28   bash.sh
        0  2019-09-16 07:12   r/
---------                     -------
    16050                     34 files
```

Extract

Using `unzip`, let us now extract files and folders from `zip_example.zip`.

```
unzip zip_example.zip
```

```
Archive:  zip_example.zip
   creating: zip_example/
  inflating: zip_example/bash.sh
  inflating: zip_example/pkg_names.txt
```

Using the `-d` option, we can extract the contents of `zip_example.zip` to a specific folder. In the below example, we extract it to a new folder `myexamples`.

```
unzip zip_example.zip -d myexamples
```

```
Archive:  zip_example.zip
   creating: myexamples/zip_example/
  inflating: myexamples/zip_example/bash.sh
  inflating: myexamples/zip_example/pkg_names.txt
```

Compress

Use the `-r` option along with `zip` to create a ZIP archive. In the below example, we create a ZIP archive of `myproject` folder.

```
zip -r myproject.zip myproject
ls
```

```
  adding: myproject/ (stored 0%)
  adding: myproject/.gitignore (stored 0%)
  adding: myproject/output/ (stored 0%)
  adding: myproject/README.md (stored 0%)
  adding: myproject/data/ (stored 0%)
  adding: myproject/data/processed/ (stored 0%)
  adding: myproject/data/raw/ (stored 0%)
  adding: myproject/src/ (stored 0%)
  adding: myproject/run_analysis.R (stored 0%)
analysis.R
bash.R
bash.Rmd
bash.sh
cran_log_sample.csv.gz
curl_demo.csv.gz
imports_blorr.txt
imports_olsrr.txt
lorem-ipsum.txt
main_project.zip
```

```
myexamples
myfiles
mypackage
myproject
myproject.zip
package_names.txt
pkg_names.tar
pkg_names.txt
r
r2
r_releases
release.txt
release_names.tar
release_names.tar.gz
release_names.txt
release_names_18.txt
release_names_18_19.txt
release_names_19.txt
releases.txt.gz
urls.txt
wget_demo.csv.gz
zip_example
zip_example.zip
```

We can compress multiple directories using `zip`. The names of the directories must be separated by a space as shown in the below example where we compress `myproject` and `mypackage` into a single ZIP archive.

```
zip -r packproj.zip myproject mypackage
ls
```

```
  adding: myproject/ (stored 0%)
  adding: myproject/.gitignore (stored 0%)
  adding: myproject/output/ (stored 0%)
  adding: myproject/README.md (stored 0%)
  adding: myproject/data/ (stored 0%)
  adding: myproject/data/processed/ (stored 0%)
  adding: myproject/data/raw/ (stored 0%)
  adding: myproject/src/ (stored 0%)
  adding: myproject/run_analysis.R (stored 0%)
  adding: mypackage/ (stored 0%)
  adding: mypackage/NAMESPACE (stored 0%)
  adding: mypackage/DESCRIPTION (stored 0%)
  adding: mypackage/LICENSE (stored 0%)
  adding: mypackage/.gitignore (stored 0%)
  adding: mypackage/man/ (stored 0%)
  adding: mypackage/docs/ (stored 0%)
  adding: mypackage/R/ (stored 0%)
  adding: mypackage/README.md (stored 0%)
  adding: mypackage/NEWS.md (stored 0%)
```

```
  adding: mypackage/data/ (stored 0%)
  adding: mypackage/.Rbuildignore (stored 0%)
  adding: mypackage/src/ (stored 0%)
  adding: mypackage/vignettes/ (stored 0%)
  adding: mypackage/tests/ (stored 0%)
analysis.R
bash.R
bash.Rmd
bash.sh
cran_log_sample.csv.gz
curl_demo.csv.gz
imports_blorr.txt
imports_olsrr.txt
lorem-ipsum.txt
main_project.zip
myexamples
myfiles
mypackage
myproject
myproject.zip
package_names.txt
packproj.zip
pkg_names.tar
pkg_names.txt
r
r2
r_releases
release.txt
release_names.tar
release_names.tar.gz
release_names.txt
release_names_18.txt
release_names_18_19.txt
release_names_19.txt
releases.txt.gz
urls.txt
wget_demo.csv.gz
zip_example
zip_example.zip
```

Add

To add a new file/folder to an existing archive, specify the name of the archive followed by the name of the file or the folder. In the below example, we add the `bash.sh` file to the `myproject.zip` archive created in a previous step.

```
zip myproject.zip bash.sh
```

```
  adding: bash.sh (deflated 78%)
```

R Functions

tar & tar.gz

In R, we can use the `tar()` and `untar()` functions from the `utils` package to handle `.tar` and `.tar.gz` archives.

Command	R
`tar tvf`	`utils::untar('archive.tar', list = TRUE)`
`tar tvfz`	`utils::untar('archive.tar.gz', list = TRUE)`
`tar xvf`	`utils::untar('archive.tar')`
`tar xvfz`	`utils::untar('archive.tar.gz')`
`tar cvf`	`utils::tar('archive.tar')`
`tar cvfz`	`utils::tar('archive.tar', compression = 'gzip')`

zip & gzip

The zip package has the functionalities to handle ZIP archives. The `tar()` and `untar()` functions from the `utils` package can handle GZIP archives.

Command	R
`gzip`	`utils::tar(compression = 'gzip')` / `R.utils::gzip()`
`gzip -d`	`utils::untar()` / `R.utils::gunzip()`
`gzip -c`	`utils::untar(exdir = filename)`
`zip -r`	`zip::zip()`
`zip`	`zip::zipr_append()`
`unzip`	`zip::unzip()`
`unzip -d`	`zip::unzip(exdir = dir_name)`
`unzip -l`	`zip::zip_list()`

Exercises

1. List without extracting: run `tar -tvf release_names.tar` and `unzip -l main_project.zip`. You should see listings of the archived files and folders, with no new files created.
2. Stream without decompressing: run `gzip -dc cran_log_sample.csv.gz | head -n 3` — you should see the first 3 CSV lines while the fixture stays compressed.
3. Create and extend: copy a text file to `myarchive_src.txt`, run `tar -cvf myarchive.tar myarchive_src.txt`, then append another file with `tar -rvf myarchive.tar release_names.txt` and verify with `tar -tvf myarchive.tar`. Separately, run `zip myarchive.zip myarchive_src.txt` and verify with `unzip -l myarchive.zip`. Clean up all `myarchive*` files afterwards.

Worked solutions in `solutions/file-compression.md` (attempt first).

System Info

In this chapter, we will explore commands that will allow us to

- display information about the system
- display memory usage information
- display file system disk space usage
- exit the terminal
- run commands a superuser
- shutdown the system

Command	Description
`uname`	Display important information about the system
`free`	Display free, used, swap memory in the system
`df`	Display file system disk space usage
`exit`	Exit the terminal
`sudo`	Run command as super user
`shutdown`	Shutdown the system

uname

The `uname` command is used to view important information about the system. Using additional options, we can

- print details about operating system
- hardware & software related information

Command	Description
`uname`	Print details about the current machine and the operating system running on it
`uname -mp`	Hardware related information; machine & processor
`uname -srv`	Software related information; operating system, release number and version
`uname -n`	Nodename of the system
`uname -a`	Print all available information system

Print all available information about the system

```
uname -a
```

```
Linux runnervm48cm3 7.0.0-1017-azure #17~24.04.1-Ubuntu SMP PREEMPT Thu Sep 24
18:46:57 UTC 2026 x86_64 x86_64 x86_64 GNU/Linux
```

Display free, used, swap memory in the system

```
free
```

```
               total        used        free      shared  buff/cache   available
Mem:        16367520     1354740    12629604       51156     2766632    15012780
Swap:        3145724           0     3145724
```

Display file system disk space usage

```
df
```

```
Filesystem      1K-blocks     Used Available Use% Mounted on
/dev/root       151263856 62894804  88352668  42% /
tmpfs             8183760       84   8183676   1% /dev/shm
tmpfs             3273504     1024   3272480   1% /run
tmpfs                5120        0      5120   0% /run/lock
efivarfs           131072       32    131036   1% /sys/firmware/efi/efivars
/dev/nvme0n1p16    901520    86444    751948  11% /boot
/dev/nvme0n1p15    106832     6250    100582   6% /boot/efi
tmpfs             1636752       12   1636740   1% /run/user/1001
```

Others

In this section, let us look at a few other useful commands that will allow us to

- see how long a command takes to execute
- delay activity
- display and clear command history list

Command	Description
`time`	See how long a command takes to execute
`sleep`	Delay activity in seconds
`sleep 1m`	Delay activity in minutes
`sleep 1h`	Delay activity in hours
`history`	Display command history list with line numbers
`history -c`	Clear the command history list

Funny Commands

Below are a few funny commands for you to try out. Use `sudo apt-get install` to install fortune and banner before trying them.

Command	Description
`fortune`	Poignant, inspirational & silly phrases
`yes`	Output a string repeatedly until killed
`banner`	ASCII banner

Command	Description
`rev`	Reverse each character

R Functions

In R, we can use `Sys.sleep()` to delay activity and `history()` to view command history.

Command	R
`sleep`	`Sys.sleep()`
`history`	`history()`

Exercises

1. Run `uname -mp` (hardware: machine and processor), `uname -srv` (software: OS, release, version), and `uname -a` (all information). State which option you used for each and what you observed.
2. Run `free` and `df` without options. Describe in one sentence each what memory/swap details and disk-space details you observe. (Git Bash users: these commands are unavailable — run them in the Tier 0 Posit Cloud project instead.)
3. Run `time sleep 2` — the timing report should show about 2 seconds. Then run `history` and find those commands in the list with their line numbers. Do not run `history -c` for this exercise.

Worked solutions in `solutions/sysinfo.md` (attempt first).

R & the Shell

In this chapter, we will learn to

- execute shell commands from R
- execute shell script in RStudio
- execute shell commands in RMarkdown
- execute R commands & script in the shell

R

Now, let us turn our attention to executing commands from R using `system2()`. Here we will focus on the following

- execute a command without arguments
- execute commands with arguments
- redirect output

Let us try to execute a command without any additional arguments. We will execute the `ls` command to list all files and directories. Use `system2()` and specify the command using the `command` argument. Whenever you are trying to execute a command from R, the first argument or input should be the command and it must be enclosed in quotes.

```
system2(command = "ls")
```

Great! Now, how do we specify the options? The additional options of a command must be stored as a character vector and specified using the `args` argument. In the below example, we delete the `myexamples` folder we created earlier while decompressing the `zip_example.zip` file.

```
system2(command = "rm",
        args    = c("-r", "myexamples"))
```

In some cases, we might want to redirect the output. Let us say we are writing message to a file using the `echo` command. In this cases, we want the output to be redirected to the `release.txt` file. The `stdout` argument can be used to redirect output to a file or the R console. In the below example, we redirect the output to a file.

```
system2(command = "echo",
        args    = c("Great Truth"),
        stdout  = "release.txt")
```

In the next example, we redirect the output to the R console by setting the value of the `stdout` argument to `TRUE`. If you are curious, set the value to `FALSE` and see what happens.

```
system2(command = "diff",
        args    = c("imports_olsrr.txt", "imports_blorr.txt"),
        stdout  = TRUE)
```

The `run()` command from the processx package can be used to execute shell commands as well.

RStudio

In RStudio, commands can be executed from shell scripts by pressing `Ctrl + Enter`. Instead of sending the command to the R console, it is redirected to the terminal where it is executed as shown below.

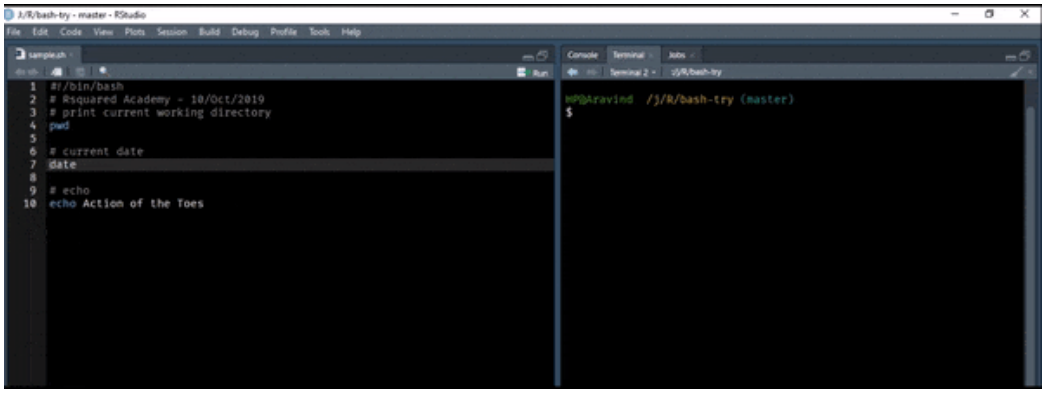

Figure 6: Running shell commands from an RStudio shell script with Ctrl+Enter

RMarkdown

RMarkdown supports `bash`, `sh` and `awk`. This post was initially created using `sh` as the underlying operating system is Windows. Later, we used `bash` after installing the Windows subsystem for Linux. Click here to learn more.

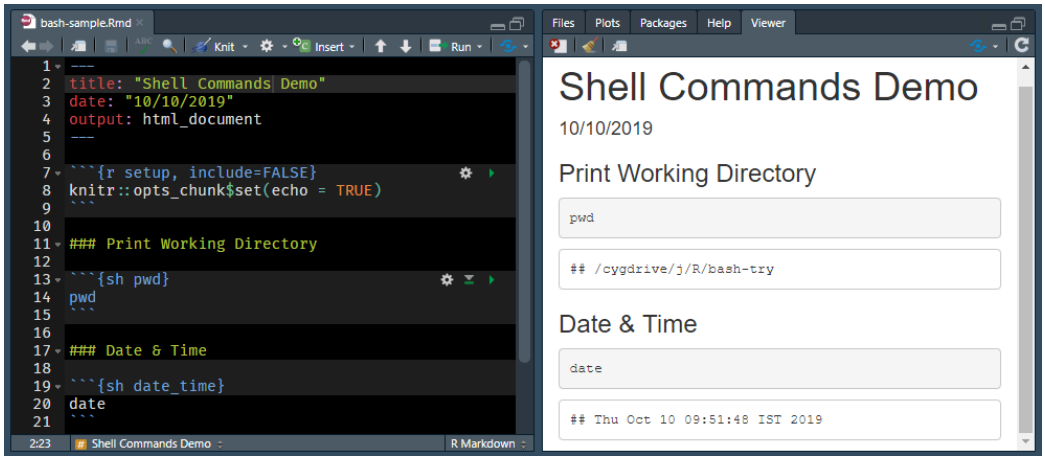

Figure 7: Using the bash language engine in RMarkdown

Rscript

In this section, we will learn to execute R commands and scripts in the command line using:

- `R -e`
- `Rscript -e`
- `R CMD BATCH`

The `-e` option allows us to specify R expression(s). `R -e` will launch R and then execute the code specified within quotes. Use semi-colon to execute multiple expressions as shown below. You will be able to run the below commands only if you are able to launch R from the command line. Windows users need to ensure that R is added to the path environment.

```
R -e "head(mtcars); tail(mtcars)"
```

`Rscript -e` will run code without launching R.

```
Rscript -e "head(mtcars)"
```

We can use `Rscript` to execute a R script as well. In the below example, we execute the code in `analysis.R` file.

```
Rscript analysis.R
```

If you are more interested, try the littler package as well.

Exercises

1. From R, run `system2(command = "ls")` to list files, then `system2(command = "rm", args = c("-r", "myexamples"))` as shown in the chapter. State what you expect in the R console and in the working directory in each case.
2. From R, run `system2(command = "echo", args = c("Great Truth"), stdout = "release.txt")`, then check `release.txt` contains the text. Then run `system2(command = "diff", args = c("imports_olsrr.txt", "imports_blorr.txt"), stdout = TRUE)` and

note the differences print to the R console instead of a file. Clean up `release.txt` afterwards if it is not part of the fixtures.

3. From the shell, run `Rscript -e "head(mtcars)"` and `Rscript analysis.R`, and compare with `R -e "head(mtcars); tail(mtcars)"`. State the expected difference: running code without launching the full R console versus executing a script file.

Worked solutions in `solutions/r-shell.md` (attempt first).

Pipes & Redirection

So far every command has worked on one file at a time. Real data work chains small tools together: decompress a log, skip its header, pull out one column, count values, and keep the top 10 — without ever opening the 5,000-row file in an editor or loading it into R first.

Why pipes matter for R users

Before calling `readr::read_csv()` on a large download, you want to know: how many rows? Which columns? What are the most frequent values? The shell answers in milliseconds by **streaming** data through a pipeline. Nothing is stored in between — each command reads, transforms, and passes onward.

The | operator

The pipe `|` connects the standard output of one command to the standard input of the next:

```
gzip -dc cran_log_sample.csv.gz | head -n 3
```

```
"date","time","size","r_version","r_arch","r_os","package","version","country","ip_id"
"2019-09-15","02:28:38",47952,NA,NA,NA,"aws.s3","0.3.12","US",1
"2019-09-15","02:28:37",5899,NA,NA,NA,"aws.ec2metadata","0.2.0","US",1
```

`gzip -dc` decompresses to standard output (`-d`) and keeps the original file (`-c`), so the fixture stays compressed. We use `gzip -dc` instead of `zcat` because `zcat` fails on macOS (it expects `.Z` files there).

> On macOS, several coreutils differ from Linux GNU versions (e.g. `date`, `sed -i`, `head -n -5`). The commands in this chapter behave identically on both.

The canonical pipeline

Our question: which packages were downloaded most in the CRAN log sample? Build the answer one stage at a time:

```
gzip -dc cran_log_sample.csv.gz | tail -n +2 | cut -d',' -f7 | tr -d '"' | sort |
uniq -c | sort -nr | head -n 10
```

```
    124 rsconnect
    122 magrittr
    115 aws.s3
    108 aws.ec2metadata
     93 ggplot2
```

```
     47 stringr
     47 rlang
     45 dplyr
     44 testthat
     44 ellipsis
```

Stage by stage:

Stage	Purpose
`gzip -dc cran_log_sample.csv.gz`	Stream the uncompressed CSV (5,001 lines: header + 5,000 rows)
`tail -n +2`	Skip line 1, the header (`"date","time",…`), so `"package"` is not counted as a download
`cut -d',' -f7`	Extract field 7, the `package` column (field 3 is `size` — verify with the header)
`tr -d '"'`	Strip quotes so `"aws.s3"` and `aws.s3` group together
`sort`	Group identical names (required before `uniq -c`)
`uniq -c`	Count each group
`sort -nr`	Sort numerically (`-n`), descending (`-r`)
`head -n 10`	Keep the top 10

The result: `rsconnect` (124 downloads) tops the sample, followed by `magrittr` (122) and `aws.s3` (115).

> `cut -d','` splits on every comma naively — it breaks on CSVs with quoted commas inside fields. For such files use a real CSV parser (`readr::read_csv()` in R, or `xsv`/`csvkit` on the shell).

Redirection and chaining recap

Chapter 4 covered > (overwrite) and >> (append). Two more operators complete the set:

```
sort < pkg_names.txt > sorted_pkgs.txt
```

< feeds a file into a command's standard input. And && runs the next command only if the previous one succeeded:

```
mkdir -p pipe_demo && cp release_names.txt pipe_demo/ && ls pipe_demo
```

```
release_names.txt
```

```
rm -rf pipe_demo
```

| versus |> and %>%

The shell pipe and the R pipe rhyme but differ:

	Shell `|`	R `|>` / `%>%`
Passes	raw text stream	R objects
Evaluation	streaming (constant memory)	eager (each step completes first)
Errors	exit codes (`echo $?`), easy to ignore silently	conditions that stop unless handled

In R, the equivalent of the counting stage is `table()` + `sort()`:

```
logs <- readr::read_csv("cran_log_sample.csv.gz", show_col_types = FALSE)
sort(table(logs$package), decreasing = TRUE)[1:10]
```

Use the shell to **inspect and slim down** data before it ever enters R; use R once the data fits comfortably in memory.

Command	R
`cmd1 | cmd2`	`cmd2(cmd1(x))` / `x |> cmd1() |> cmd2()`
`>` / `>>`	`writeLines()` / `readr::write_csv()`
`<`	`readLines()`
`&&`	`&&`
`sort | uniq -c | sort -nr`	`sort(table(x), decreasing = TRUE)`

Exercises

1. Run the canonical pipeline from this chapter. Confirm the top line reads `124 rsconnect` and the output has exactly 10 lines (check with `... | wc -l`).
2. Which countries download most? Reuse the pipeline but extract field 9 (`country`) instead of field 7. Confirm the top line reads `1959 NL`.
3. Drop the `tail -n +2` stage and re-run. You should see `"package"` appear in the counts with a count of 1 — proof the header row pollutes results when not skipped.

Worked solutions in `solutions/pipes.md` (attempt first).

Conclusion

What we have not covered

- shell scripting
- editing files
- file permissions
- user information
- pipes
- awk
- sed

Summary

- Shell is a text based application for viewing, handling and manipulating files
- It is also known by the following names
  - CLI (Command Line Inteface)
  - Terminal
  - Bash (Bourne Again Shell)
- Use `system2()` and `processx::run()` in R to execute shell commands
- Use `Rscript -e` or `R -e` to execute R scripts from the command line
- RStudio includes a Terminal (from version 1.1.383)
- Execute commands from shell script in RStudio using Ctrl + Enter
- RMarkdown supports `bash`, `sh` and `awk`

References

- https://tldr.sh/
- https://blog.rstudio.com/2017/08/11/rstudio-v1-1-preview-terminal/
- https://support.rstudio.com/hc/en-us/articles/115010737148-Using-the-RStudio-Terminal
- https://bookdown.org/yihui/rmarkdown/language-engines.html#shell-scripts
- https://github.com/kirillseva/tldrrr
- https://fs.r-lib.org/articles/function-comparisons.html
- https://livefreeordichotomize.com/2017/09/28/r-release-names/
- https://stackoverflow.com/questions/13478375/is-there-any-authoritative-documentation-on-r-release-nicknames
- http://www2.imm.dtu.dk/courses/02333/cygwin_tutorial/index.html
- https://tutorial.djangogirls.org/en/intro_to_command_line/
- https://ss64.com/
- https://www.learnenough.com/command-line-tutorial

Feedback

If you see mistakes or want to suggest changes, please create an issue on the source repository or reach out to us at support@rsquaredacademy.com.

Exercises

1. Navigation + R mapping: from the shell, navigate into `cline/` and run `ls`. Then do the same from R with `system2(command = "ls")` and by running R from the shell with `Rscript -e "dir()"`. Confirm all three approaches list the same files.
2. Search + R mapping: from the shell, search `package_names.txt` for a pattern with `grep -i`. Then repeat the search from R by wrapping the same command in `system2(..., stdout = TRUE)`. Confirm both return the same matches.
3. End-to-end mini-workflow: navigate to a folder, locate a `.R` script by searching, and execute it from the shell with `Rscript`. List the three steps you took and the final evidence (script output or created file) proving the workflow succeeded.

Worked solutions in `solutions/conclusion.md` (attempt first).

Shell ↔ R Cheat Sheet

One page mapping every shell command in this book to its R equivalent. Keep it beside your terminal.

Shell	R equivalent
`pwd`	`getwd()` / `here::here()`
`ls`	`dir()` / `list.files()` / `fs::dir_ls()`
`ls -la`	`fs::dir_info()`
`cd dir`	`setwd("dir")`
`mkdir dir`	`dir.create("dir")` / `fs::dir_create()`
`rmdir dir`	`fs::dir_delete()`
`touch file`	`file.create()` / `fs::file_touch()`
`cp a b`	`file.copy()` / `fs::file_copy()`
`mv a b`	`file.rename()` / `fs::file_move()`
`rm file`	`file.remove()` / `fs::file_delete()`
`echo text`	`cat()` / `message()`
`cat file`	`readLines()`
`head -n 5 file`	`head(readLines("file"), 5)`
`tail -n 5 file`	`tail(readLines("file"), 5)`
`wc -l file`	`length(readLines("file"))`
`sort file`	`sort()`
`sort -u file`	`unique(sort())`
`grep pattern file`	`grep()` / `stringr::str_detect()`
`find dir -name '*.txt'`	`list.files(dir, pattern = "\\.txt$")`
`wget URL` / `curl -O URL`	`download.file()` / `curl::curl_download()`
`tar -xvf f.tar`	`utils::untar()`
`zip -r z.zip dir`	`zip::zip()`
`unzip f.zip`	`zip::unzip()`
`gzip -dc f.gz`	`gzcon()` / `readr::read_csv()`
`sleep 5`	`Sys.sleep(5)`
`history`	`history()`
`diff a b`	`waldo::compare()` / `diffobj::diffObj()`
`system2("ls")`	run shell from R

Shell	R equivalent
`Rscript script.R`	run R from shell

Licensed CC BY-NC-SA 4.0.

Appendix: Git in 10 Minutes for RStudio Users

You already know the shell — Git is the natural next step for version-controlling R projects. Everything below also maps to RStudio's Git pane (Tools -> Version Control).

Configure once

```
git config --global user.name "Your Name"
git config --global user.email "you@example.com"
```

The 6-command workflow

Step	Shell	RStudio Git pane
1. Check state	`git status`	Diff / status list
2. Stage	`git add file.R` (or `git add -u`)	Tick the Staged box
3. Commit	`git commit -m "feat: short message"`	Commit button
4. Sync down	`git pull`	Pull button (blue down arrow)
5. Sync up	`git push`	Push button (green up arrow)
6. History	`git log --oneline -10`	History button

Branches

```
git branch feature          # create
git switch feature          # move to it
git switch -c hotfix        # create + move in one step
git switch master           # back to production branch
```

Rule of thumb from this book's workflow: do work on a feature branch, keep `master` deployable, and push only at the end of a phase.

Ignore files

Create a `.gitignore` so generated files never get committed:

```
cat .gitignore
```

```
.Rhistory
.RData
cline/
docs/
```

This book's own `.gitignore` keeps the `cline/` working directory and rendered `docs/` output out of version control — the same pattern works for your R projects (`data/`, `output/`, `.Rhistory`).

Undo safely

- Wrong stage: `git restore --staged file.R`
- Wrong edit (file still uncommitted): `git restore file.R`
- Never rewrite history on shared branches — prefer a new commit that fixes the mistake.

Exercises

1. In a scratch folder, run `git init`, create `notes.R`, then `git status`, `git add notes.R`, `git commit -m "docs: first notes"`, and `git log --oneline`. You should see exactly one commit.
2. Edit `notes.R`, run `git status` (file shows as modified), stage and commit again. `git log --oneline` should now show two commits.
3. Add a `.gitignore` containing `*.html`, create `report.html`, and run `git status` — the HTML file should not appear. Remove it with `rm report.html`.

Worked solutions in `solutions/git-appendix.md` (attempt first).