

Using the command line

Hitchhiker's Guide to Reproducible Research

Julia Wrobel and David Benkeser

 Course Website

Operating systems

- Windows
 - Not always programmer friendly
- Mac OSX
 - Better for programming
 - Under the hood, is just Unix
- Unix-based OS (Linux, Solaris, etc...)
 - Best for programming

We'll learn to interact with our computer like it's a Unix OS.

- Best practices for programming
- Needed for AWS and Docker (later)

Some terminology

Shell

- user interface for interacting with a computer
- the "outermost" layer of the operating system

Graphical user interface (GUI)

- visual interface (icons, menus, etc...) for interacting with computer
- "Point-and-click"

Command line interface (CLI)

- text-based interface for interacting with computer
- e.g., `bash`, `sh`, `tcsh`, `zsh`, ...

Some terminology

Shell script

- plain text file designed to be run by the shell

Terminal

- "terminal emulator"
- a program that lets you interact with the shell

Why use a terminal?

- work faster and more efficiently: no mouse or touchpad!
- better debugging when you break stuff (i.e. Git/GitHub)
- impress your friends and family?
- reproducibility!!!!

Terminal

If Windows, use [Ubuntu](#) for Windows

- Or other Linux distribution (e.g., [Debian](#))
- Biggest difference is how software installed

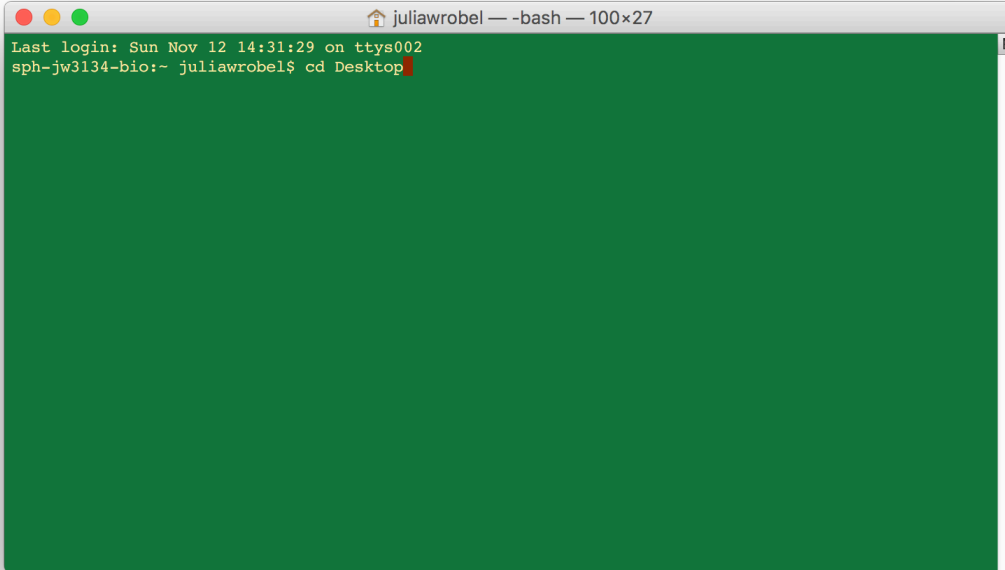
If Mac, use [Terminal](#)

- Or [iTerm2](#) -- more features

If Linux, whatever terminal emulator comes with your distribution.

Terminal

Your open terminal will look something like this:

A screenshot of a terminal window. The window title bar shows a home icon, the username 'juliawrobel', the shell '-bash', and the window size '100x27'. The terminal content shows the last login information: 'Last login: Sun Nov 12 14:31:29 on ttys002' and the current prompt: 'sph-jw3134-bio:- juliawrobel\$ cd Desktop'. The rest of the terminal area is filled with a solid green color.

```
juliawrobel — -bash — 100x27
Last login: Sun Nov 12 14:31:29 on ttys002
sph-jw3134-bio:- juliawrobel$ cd Desktop
```

You'll see a *prompt*, which is an alphanumeric string that (usually) ends in `$`. Commands are typed after the `$`.

Using the terminal through RStudio

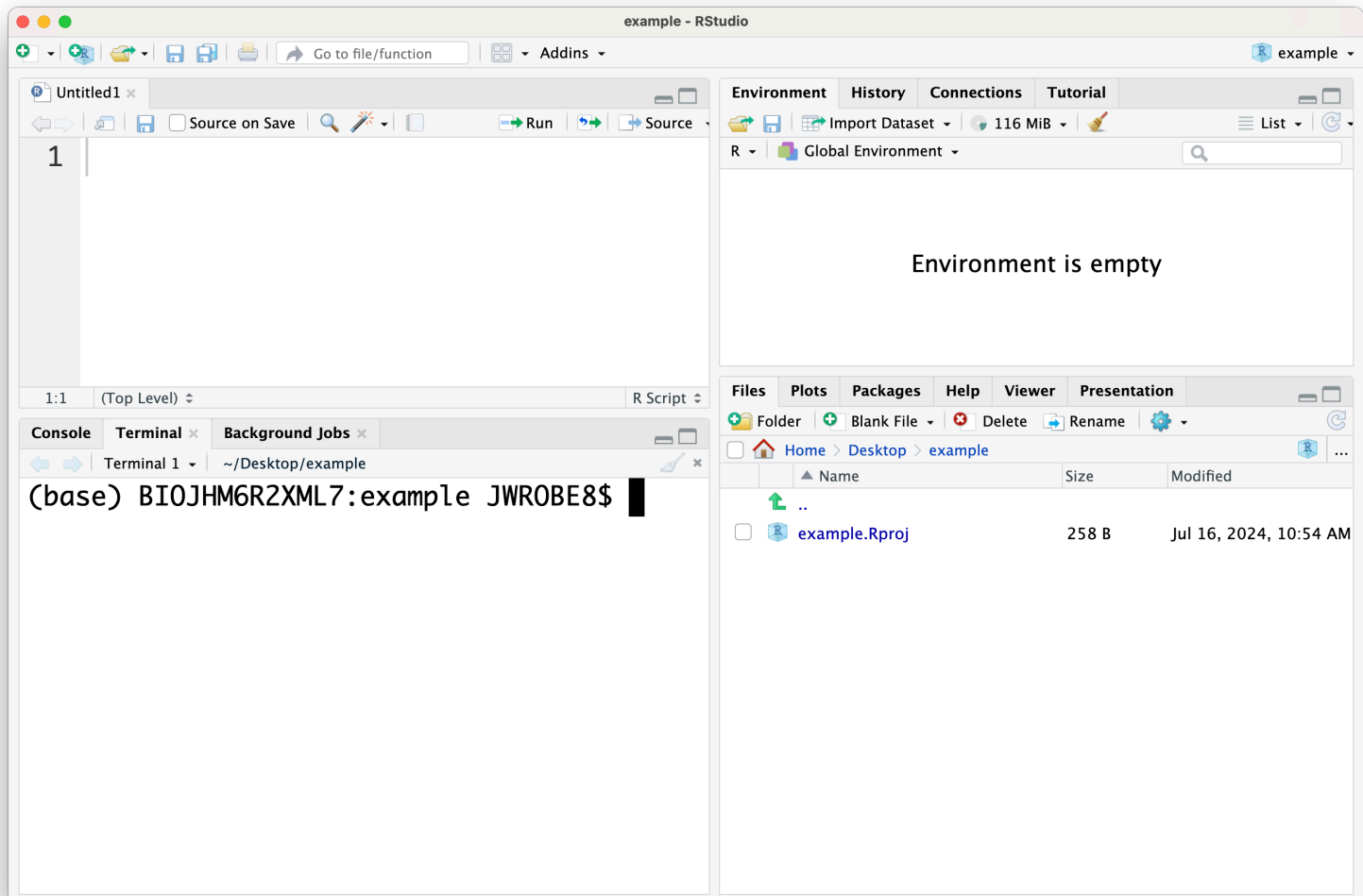
You can also use the terminal directly in Rstudio!

- Rstudio introduced the terminal tab in 2017
- For our course, this may be the simplest approach

Display the Terminal tab: If the tab isn't visible, you can display it by going to Tools > Terminal > Move Focus to Terminal.

- Can also use the keyboard shortcut Shift+Alt+M or Shift+Option+M on Mac.

RStudio terminal



Working directories

What is a working directory?

Moving around directories

Folders within your computer are called **directories**. You can navigate around to different directories, remove or create directories, remove or create files, move files around, and list their contents all from the terminal.

These next sections may seem fairly basic but these are also the commands I use the most often, so I'm going to spend some time on them.

Moving around directories

Command	Action
<code>pwd</code>	print working directory
<code>cd</code>	change directory
<code>ls</code>	list files in directory

- **cd**: Takes you to the home directory
 - `cd ..`: Moves up one directory
 - `cd ../..`: Moves up two directories
- **ls -a**: list **hidden** files as well as other files
 - Hidden files are often part of the instructions for the OS or a particular application
 - Usually invisible when searching through folders
 - Examples: `.git`, `.gitignore`, `.Rhistory`

Absolute vs. relative file paths

Absolute paths

- `/Users/juliawrobel`
- `~/Documents`
- `/`

Relative paths

- `./Documents`
- `../Documents`
- `../.. /`

- Absolute paths include the **whole path** for a directory
- Relative paths depend on the working directory that they are executed in
 - The `./` means "in the current directory"
 - The `../` means "in one directory up from the current directory".

Adding/removing files

Command	Action
<code>mkdir</code>	make a new directory
<code>rm</code>	delete a file or directory
<code>mv</code>	move a file or directory
<code>cp</code>	copy a file or directory

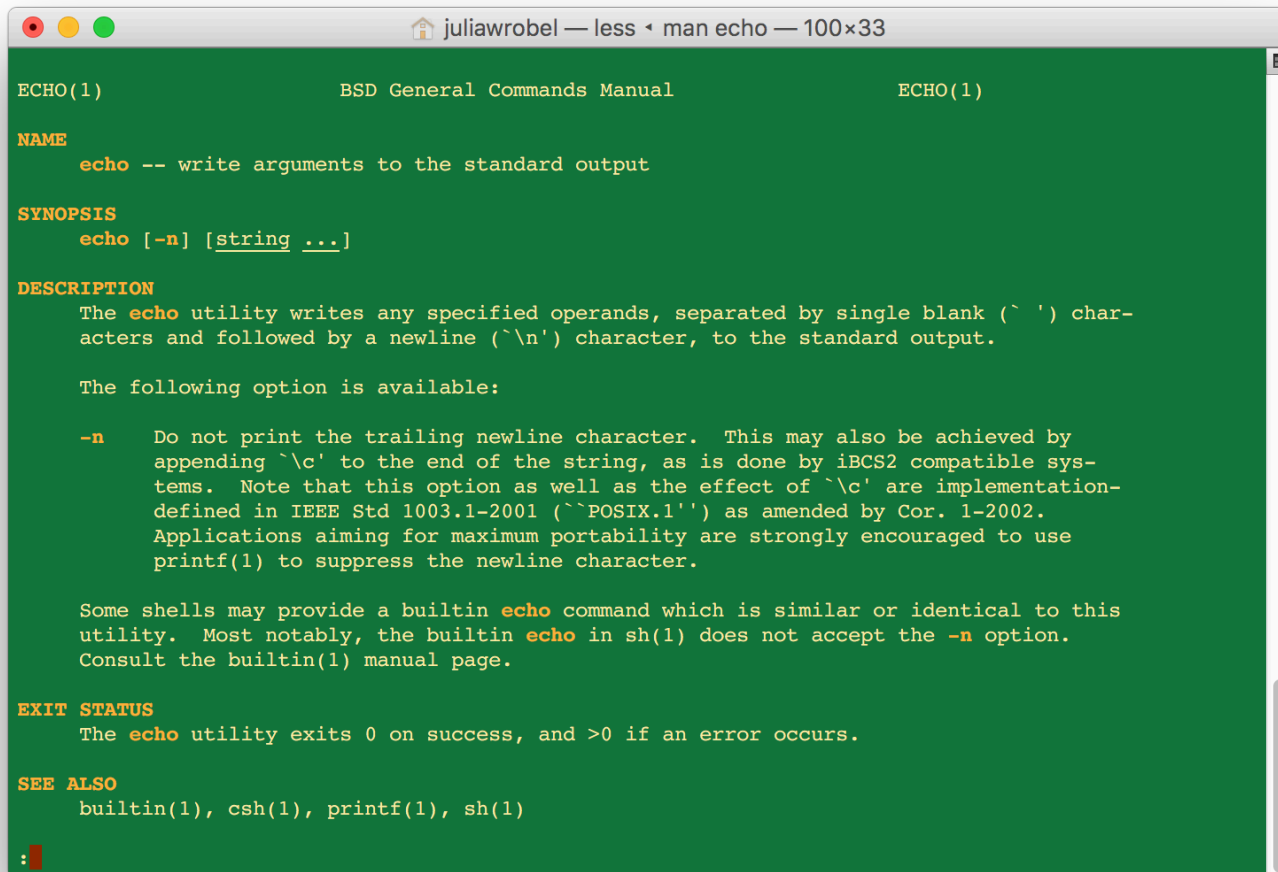
Structure of a bash command

```
command [options] [arguments]
```

1. **command**: the bash function you want to run, e.g. `ls`, `cd`, `echo`, etc.
2. **options**: also called "flags", these are additional parameters to modify the behavior of the command, e.g.,
 - `ls -R` lists all directories and contents recursively
 - `ls -aR` recursively lists all files and hidden files
3. **arguments**: inputs to the command, such as file names or other data that tell the command what to operate on
 - `rm` what? `cp` what?
 - `ls \Documents` lists all files in the documents folder

bash help files

To see available options check `man` command.



```
juliawrobel — less ◀ man echo — 100×33
ECHO(1) BSD General Commands Manual ECHO(1)
NAME
  echo -- write arguments to the standard output
SYNOPSIS
  echo [-n] [string ...]
DESCRIPTION
  The echo utility writes any specified operands, separated by single blank ( ` ` ) characters and followed by a newline ( ` `n` ) character, to the standard output.

  The following option is available:

  -n   Do not print the trailing newline character.  This may also be achieved by appending ` `c` to the end of the string, as is done by iBCS2 compatible systems.  Note that this option as well as the effect of ` `c` are implementation-defined in IEEE Std 1003.1-2001 ( ` `POSIX.1` `) as amended by Cor. 1-2002.  Applications aiming for maximum portability are strongly encouraged to use printf(1) to suppress the newline character.

  Some shells may provide a builtin echo command which is similar or identical to this utility.  Most notably, the builtin echo in sh(1) does not accept the -n option.  Consult the builtin(1) manual page.
EXIT STATUS
  The echo utility exits 0 on success, and >0 if an error occurs.
SEE ALSO
  builtin(1), csh(1), printf(1), sh(1)
```

Solving computing problems

- `man [command]`
- Google (with `site:stackoverflow.com?`)
- ChatGPT
- Ask friends/classmates
- **Try stuff!**

Breakout exercises are designed to force you to try new things.

- Learning how to learn!

Try it out!

Open the terminal and perform the following tasks:

1. What is your current working directory?
2. In your current directory, make a directory named `tmp`
3. Navigate into this new folder called `tmp`
4. Add an empty file named `tmp_file`
5. Within `tmp`, add an empty directory named `another_tmp`
6. Within `another_tmp`, add an additional empty file named `another_tmp_file`

Hint: if you need some help, check out the solutions under [Exercise 1](#).

Group exercise

Next, get into breakout groups and perform the following tasks:

1. list all files with sizes displayed in bytes/kilobytes/megabytes
2. remove `tmp_file`
3. rename `another_tmp_file` to `such_a_cool_file`
4. remove `tmp` directory *and* all its contents

Redirects and pipes

Command	Action
>	redirect output to file
>>	redirect output and append to file
<	have input come from a file
	output of command becomes input of next

Piping and redirects give you **flexibility** in coding.

To bash or not to bash

```
# download hamlet text from web and save
# in file called hamlet.txt
curl -L http://bit.ly/hamlet_txt > hamlet.txt

# lines the string "Ham" mentioned
grep "Ham" hamlet.txt

# lines with "Oph" and "Ham"
grep "Ham" hamlet.txt | grep "Oph"

# count Hamlet's lines
grep "Ham\." hamlet.txt | wc -l

# see the start of Hamlet's first 5 lines
grep "Ham\." hamlet.txt | head -5

# see the start of Hamlet's last line
grep "Ham\." hamlet.txt | tail -1
```

Wild cards

Command	Action
*	match anything
?, ??, ...	match a single character
[...]	match a range of characters

```
# files in cwd with .txt extension
ls -l *.txt
# all files in cwd named a_file with three character extension
ls a_file.???
# .txt files in cwd name a_file, b_file, ..., e_file
ls [a-e]_file.txt
```

Useful command line shortcuts

Key stroke	Action
<code>↑</code>	move to previous command
<code>↓</code>	move to next command
<code>tab</code>	autocomplete command or file
<code>ctrl+c</code>	cancel (running) command
<code>ctrl+z</code>	suspend command
<code>ctrl+r</code>	search for command in history
<code>ctrl+l</code>	clear the screen

File permissions

If you run `ls -l`, the far left column shows file permissions:

- e.g., `-rw-r--r--` or `drwxr-xr-x`

First character is **file type**. Then comes **read** (`r`), **write** (`w`), and execute (`x`) permissions for **user**, **group**, **others**.

Executing `+x` (essentially) makes the file executable for everyone.

See slide notes for more options for `chmod`.

Vim

Vim is a minimalistic text editor that is built into Unix and is often the default text editor within the terminal.

Key Features:

1. Modal Editing:

- Normal Mode: Navigate and manipulate text.
- Insert Mode: Insert and edit text.
- Visual Mode: Select and highlight text.

2. Commands

- Performs complex text manipulations with minimal keystrokes.
- Examples: `dd` (delete line), `yy` (yank/copy line), `p` (paste).

Basic vim commands

Opening and exiting vim

- `vim filename`: open a file in vim
- `:w`: save changes
- `:q`: quit vim
- `:wq`: save changes and quit

Navigating in vim

- `h, j, k, l`: move left, down, up, and right
- `gg`: go to the beginning of the file
- `G`: go to the end of the file

Editing text

- `i`: enter Insert mode before the cursor
- `a`: enter Insert mode after the cursor

Group exercise

In your breakout group, write bash code that executes each of the following.

1. Write a sentence or two about Atlanta, or anything you'd like, and store it in a file called "atlanta.txt" within a folder called "atlanta".
2. Use vim to edit this file in some way (e.g. add some more text).
3. Save the results and exit vim.