Bash Scripts + Git Programing Powerup

Data4ML

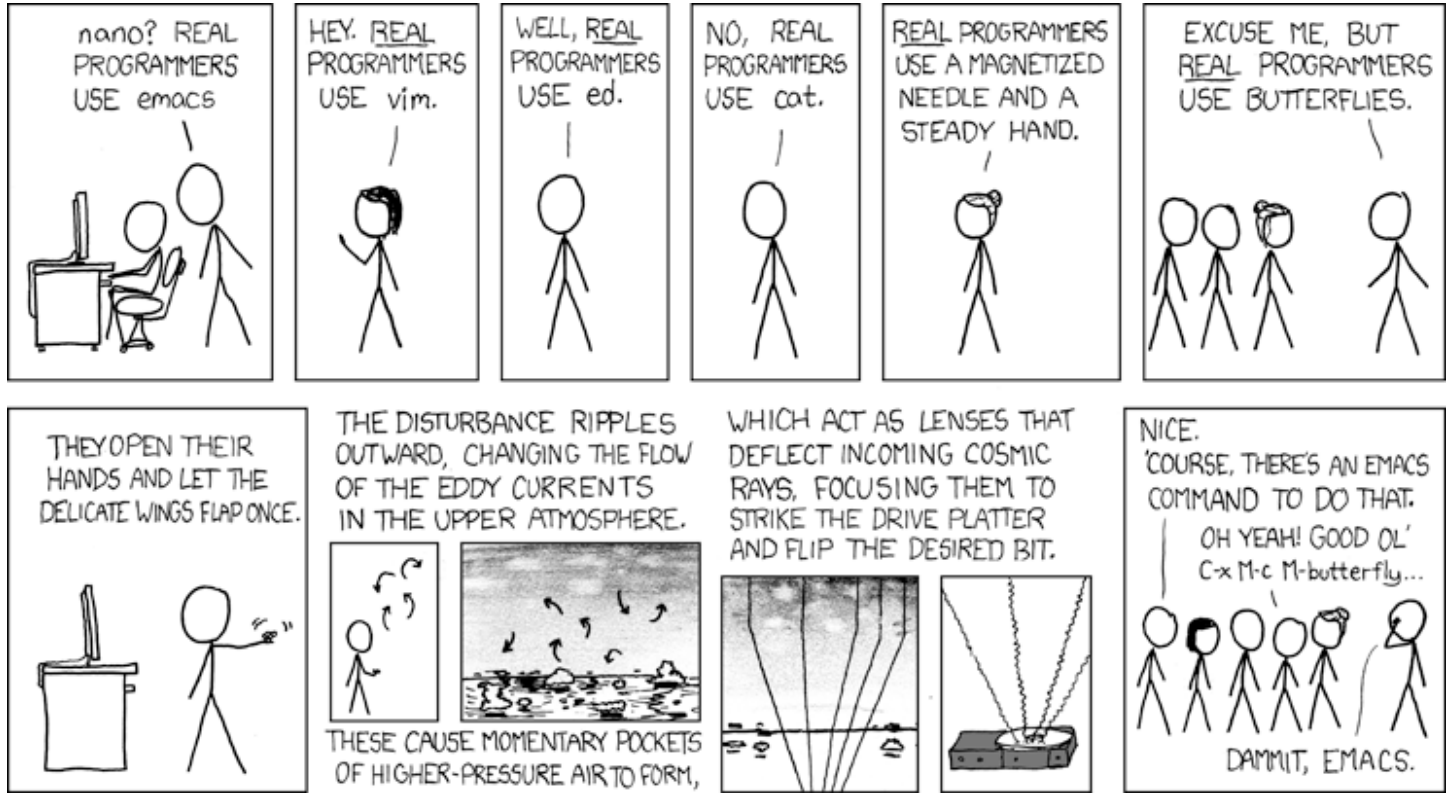
Summer 2022

Automating the Command Line

- We've covered a lot of Linux commands to move files around and transform text data
- What if you want to do run the same series for commands over and over again with different inputs
  - You could type out each command in your series one at a time for every input
    - Ok but time consuming and ERROR prone
  - It's Better to automate the process with a bash 'script'
- A bash script is just a text file that has a list of commands that get run all together when you run the file

Writing in the Terminal

- There is a surprisingly large number of ways to create a text file
  - Things like word that save *.docx file types won't work to create bash scripts
  - Sometimes fancier tools use graphics that don't work well over a remote connection
- We'll use nano which has familiar shortcuts like ctrl/cmd-c for copy and ctrl/cmd-v for paste
- People joke around a lot about text editor choice, but use what works for you



https://xkcd.com/378/

SSH

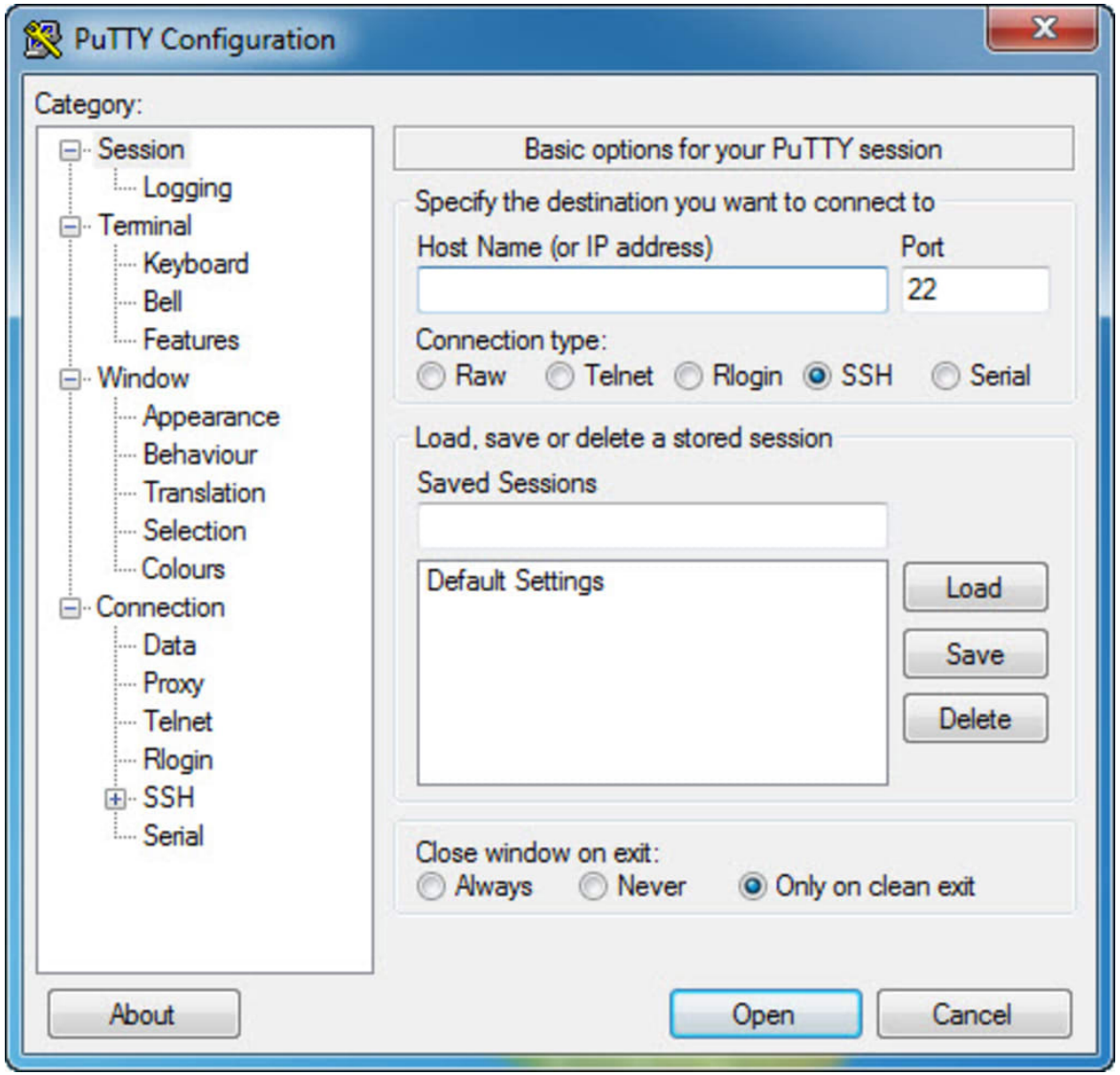
- So far we've been working in Open On Demand. It's more common to access machines using SSH (**S**ecure **Sh**ell)

- In a Linux or a mac machine just type this command in a terminal

ssh talapas-ln1.uoregon.edu

One a windows machine you'll need to install Putty -https://www.putty.org/

talapas-ln1.uoregon.edu goes under **Host Name**



jsearcy — jsearcy@talapas-ln1:~ — ssh talapas-ln1.uoregon.edu — 80×24

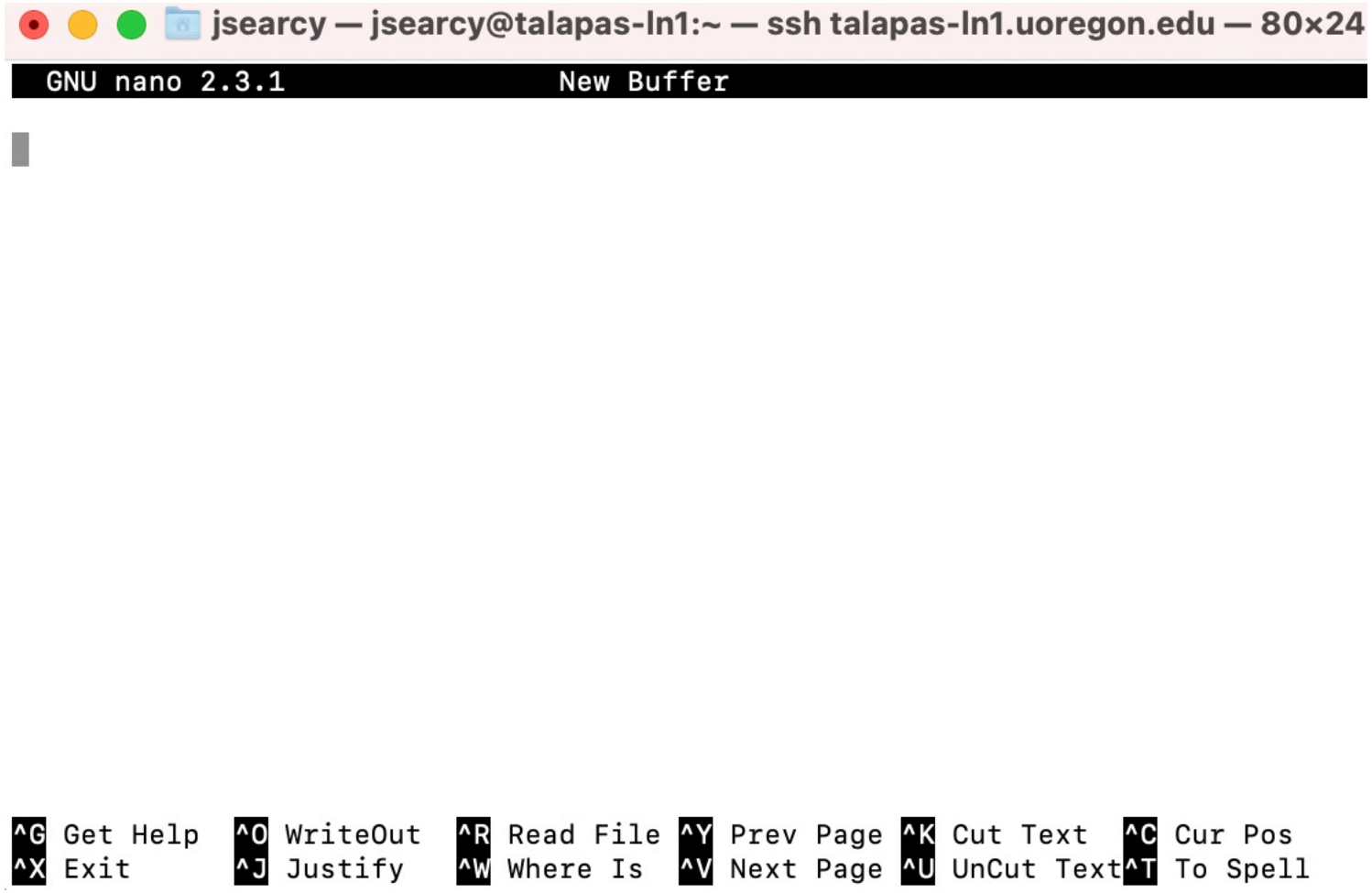
```
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
dyn-10-108-24-10:~ jsearcy$ ssh talapas-ln1.uoregon.edu
Enter passphrase for key '/Users/jsearcy/.ssh/id_rsa':
Last login: Mon Jun 27 15:07:20 2022 from dyn-10-108-24-10.wless.uoregon.edu
# Welcome to Talapas!
# Data on Talapas is NOT backed up. Data management is each users responsibilit
.

# Need support? Please visit the Talapas Knowledge Base:
# https://hpcrcf.atlassian.net/wiki/spaces/TCP/overview

# Storage usage in GB as of Mon Jun 27 15:01:04 2022
Fileset          User             UsedByUser  UsedByAll      Quota  Use%
home             jsearcy                  23          -         25    92
bgmp             jsearcy                  37      11810      65536    18
datascience      jsearcy                 780       1732       2048    85
hpcrcf           jsearcy                 303      18248      65536    28
maplab           jsearcy                  24         40       2048     2
mcguirelab       jsearcy                   0       7260       8192    89
nereus           jsearcy                   0      23730      32768    72
packages         jsearcy                   2      11792      16384    72
system           jsearcy                   0       2540       8192    31
(base) [jsearcy@talapas-ln1 ~]$
```

Nano

- Type – nano



Anatomy of a Shell Script

- **Copy the following**

- **#!** – This is the first line it tells your computer what this file is (bonus you can modify this so it works for python scripts and R scripts too)
- # without a ! is a comment, everything is interpreted.
- $ this lets you access variables
  - $1 $2 lets you access command line arguments

```
GNU nano 2.3.1                File: test.sh                          Modified

#!/bin/bash

# This is a comment

#Below is a command to print Hello World on the Terminal
echo "Hello world!"

#Command Line Arguments this is how you access things typed on the CLI
echo $1

echo $2

#This sets a variable

MYVAR='42 is the answer'

#Use the $ sign to access it
echo $MYVAR

Save modified buffer (ANSWERING "No" WILL DESTROY CHANGES) ?
 Y Yes
 N No           ^C Cancel
```

Running your Script

- **Running a script can be done it two ways**
  - **Executing it (this requires the file have execute permissions)**
    - Runs in a new shell
    - No variable you set in your script are available after the script closes
  - **Sourcing**
    - Runs in your current environment, just like you typed it by hand
    - Your variables will live after the program exits
    - You can type **source your_script.sh**

      Or
    - **. your_script.sh**

```
(base) [jsearcy@talapas-ln1 ~]$ ./test.sh
-bash: ./test.sh: Permission denied
(base) [jsearcy@talapas-ln1 ~]$ source test.sh
Hello world!


42 is the answer
(base) [jsearcy@talapas-ln1 ~]$ echo $MYVAR
42 is the answer
(base) [jsearcy@talapas-ln1 ~]$ unset MYVAR
(base) [jsearcy@talapas-ln1 ~]$ source test.sh arg1 "is neat"
Hello world!
arg1
is neat
42 is the answer
(base) [jsearcy@talapas-ln1 ~]$ echo $MYVAR
42 is the answer
(base) [jsearcy@talapas-ln1 ~]$ . ./test.sh arg1 "is neat"
Hello world!
arg1
is neat
42 is the answer
(base) [jsearcy@talapas-ln1 ~]$ unset MYVAR
(base) [jsearcy@talapas-ln1 ~]$ chmod +x ./test.sh
(base) [jsearcy@talapas-ln1 ~]$ ./test.sh exec this
Hello world!
exec
this
42 is the answer
(base) [jsearcy@talapas-ln1 ~]$ echo $MYVAR

(base) [jsearcy@talapas-ln1 ~]$
```

Everything’s a Variable

- Pretty much everything in the shell is a variable and you can change it

```
[jsearcy@talapas-ln1 ~]$echo $PS1
[\u@\h \W]$
[jsearcy@talapas-ln1 ~]$export PS1='[\u New Prompt]\$'
[jsearcy New Prompt]$
```

For loops

Loop Variable

```
for i in ‘1’ ‘2’ ‘3’
do
echo $i
done
```

Loop list

In your loop the loop variable takes on the values in your list

```
(base) [jsearcy@talapas-ln1 my_test_repo]$ for i in '1' '2' '3'; do echo $i ; done
1
2
3
(base) [jsearcy@talapas-ln1 my_test_repo]$
```

backticks

- Sometimes you want to save the output of a command or use it as a loop list, you can do this with back ticks

```
(base) [jsearcy@talapas-ln1 my_test_repo]$ x=echo hi
-bash: hi: command not found
(base) [jsearcy@talapas-ln1 my_test_repo]$ x=`echo hi`
(base) [jsearcy@talapas-ln1 my_test_repo]$ echo $x
hi
(base) [jsearcy@talapas-ln1 my_test_repo]$ for i in `ls *`; do echo $i; done
gc_counter.sh
gc_counter.sh~
MT
test_fasta_human_chimp_ND5.fasta
```

Exercise

- Make a directory called - my_test_repo

- Copy this file into it

/projects/datascience/shared/Data4ML1_examples/test_fasta_human_chimp_ND5.fasta

- Write a bash script called gc_counter.sh
  - Counts the number of C's, G's, and the number of all nucleotides each sequence
  - Prints each number to the screen
  - Bonus: divide to G and C percentates

Git and Git-hub

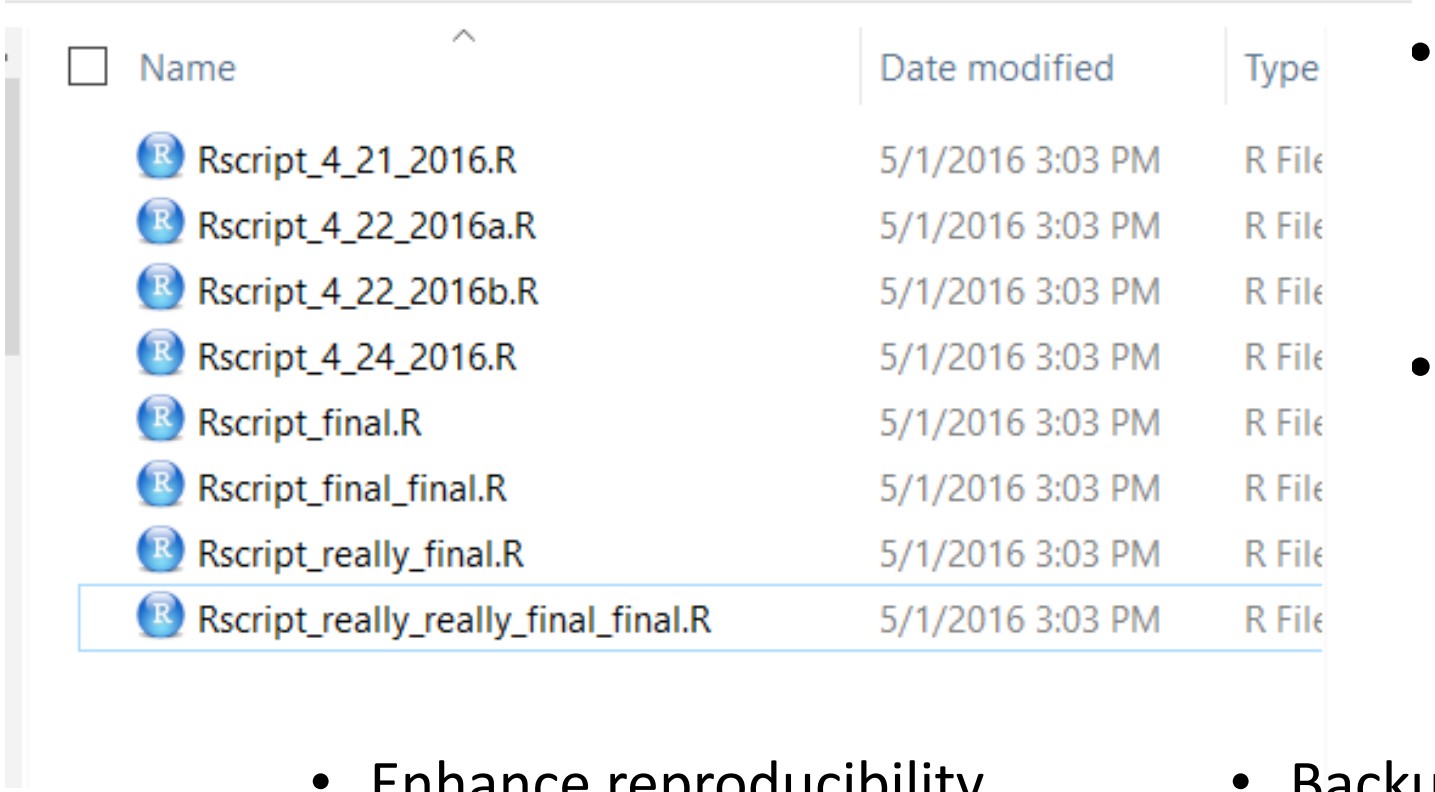
- Your code python R etc. represents a large time investment, and it also a place where mistakes or changes can affect your science
  - It’s a good idea to track these changes we do this with something called version control
  - As a bonus it let’s you share and publish your code

- Git is a kind of version control

- GitHub is a service owned by Microsoft that lets you store a remote copy of your code and share it through there servers
  - It also lets you do things like build websites

https://github.com/

What is version control?



Name	Date modified	Type
Rscript_4_21_2016.R	5/1/2016 3:03 PM	R File
Rscript_4_22_2016a.R	5/1/2016 3:03 PM	R File
Rscript_4_22_2016b.R	5/1/2016 3:03 PM	R File
Rscript_4_24_2016.R	5/1/2016 3:03 PM	R File
Rscript_final.R	5/1/2016 3:03 PM	R File
Rscript_final_final.R	5/1/2016 3:03 PM	R File
Rscript_really_final.R	5/1/2016 3:03 PM	R File
Rscript_really_really_final_final.R	5/1/2016 3:03 PM	R File

- Version control is an organized way of maintaining a record of changes
- Git is a system for distributed version control – not the only one, but popular among scientists

- Enhance reproducibility
- Fix mistakes by reverting to earlier versions
- Improve project structure

- Backup versions in remote repositories**
- Facilitate collaboration**

The mechanics of version control

Single user:

Multiple users:

Image credit: Software Carpentries

Some Git vocabulary

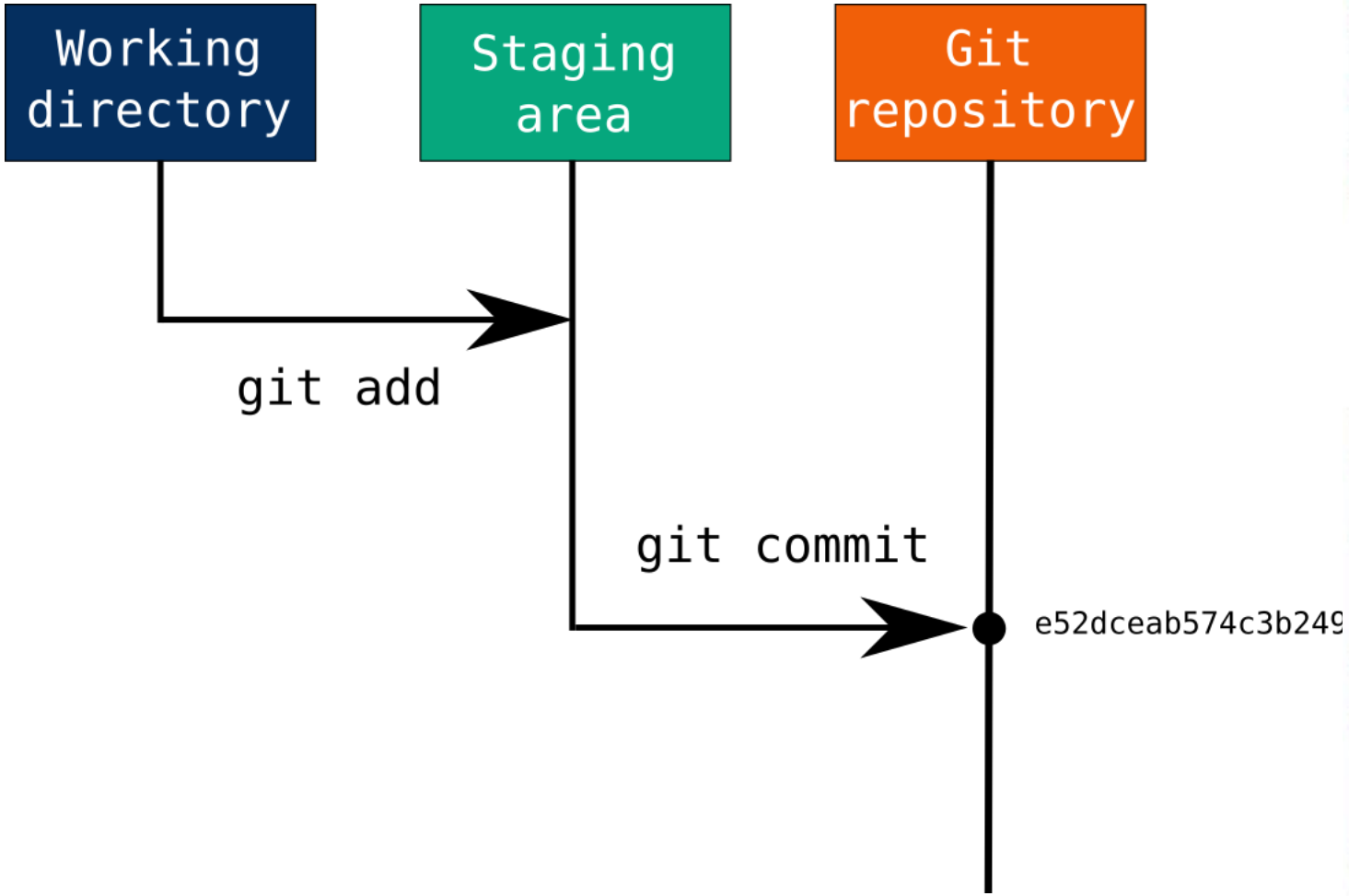
Repository/repo – the collection of files and directories associated with a project and tracked with version control

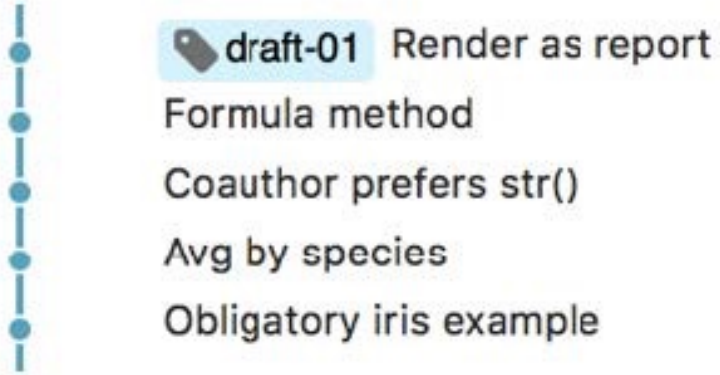
Commit – a snapshot of a repository's history that is recorded by Git

Diff – Changes in the repository's content associated with the commit

Branches – Concurrent work (changes to file content) can occur in parallel branches, so that you can focus on developing one aspect of the repository/project independently

The basic Git workflow





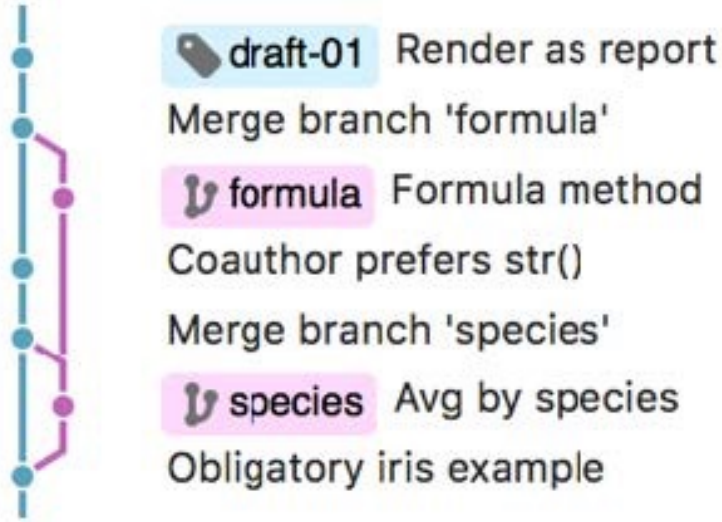


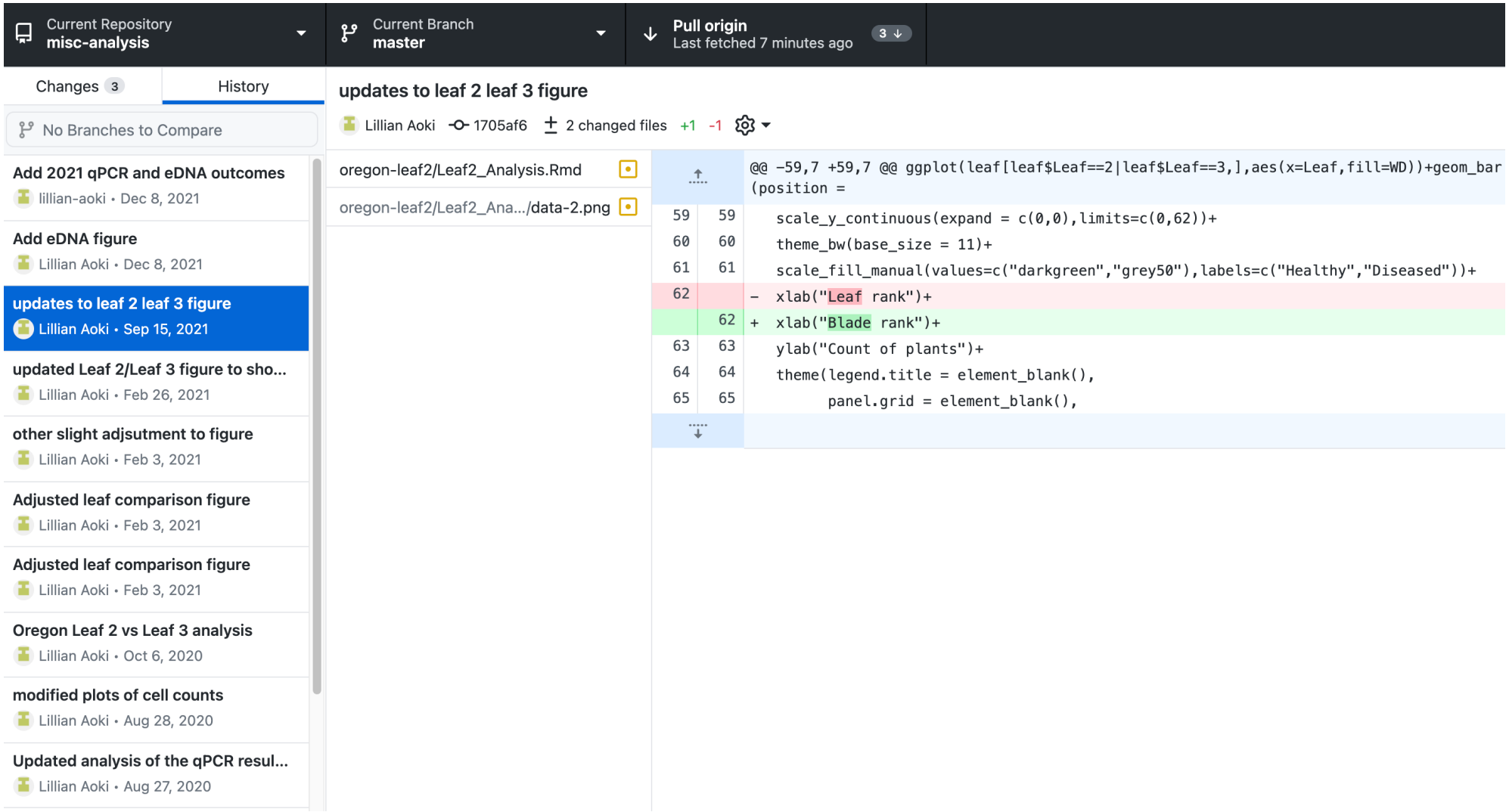
Image credit: J. Bryan; M. Joseph

Use commits to anchor your code

- if you make a mistake, you can’t fall past the previous commit.
- Use more commits when you’re in uncertain or dangerous territory.
- Commits are also helpful to others, because they show your journey, not just the destination.

Hadley Wickham, R Packages

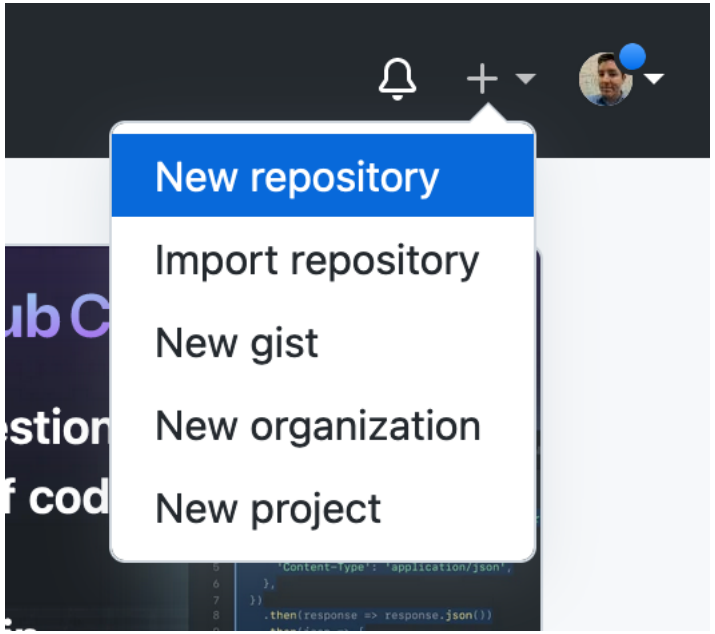
Command line vs. Git client



You’ll need a free github account for the following

https://github.com/ Sign-up or Sign in

Create a new repository



Owner * **Repository name** *

jsearcy1 / my_test_repo

Great repository names are short and memorable. Need inspiration? How about **urban-tribble**?

Description (optional)

- ○ **Public**
  Anyone on the internet can see this repository. You choose who can commit.
- ◉ **Private**
  You choose who can see and commit to this repository.

Initialize this repository with:

Skip this step if you're importing an existing repository.

☐ **Add a README file**
This is where you can write a long description for your project. Learn more.

Add .gitignore

Choose which files not to track from a list of templates. Learn more.

.gitignore template: None

Choose a license

A license tells others what they can and can't do with your code. Learn more.

License: None

ⓘ You are creating a private repository in your personal account.

Create repository

Tokens

- Github won’t let you use your password to login when using applications or commands on the command line
- You’ll need to generate a token
  - https://github.com/settings/tokens/new

Quick setup — if you've done this kind of thing before

Set up in Desktop or HTTPS SSH https://github.com/jsearcy1/my_test_repo.git

Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.

...or create a new repository on the command line

```
echo "# my_test_repo" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/jsearcy1/my_test_repo.git
git push -u origin main
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/jsearcy1/my_test_repo.git
git branch -M main
git push -u origin main
```

...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

Import code

ProTip! Use the URL for this page when adding GitHub as a remote.

Your repo is setup

- Everytime you make changes you want to keep
  - git add <files>
  - git commit –m “a message for everyone. I changed x for y reason”
  - git push

- See what’s being tracked and what has changed
  - git status
  - git diff