

# Reproducible Research

## *Practices & Tools*

**Sarah Huber, Shahira Khair, Drew Leske**  
University of Victoria

COSS 2024 - june 10



Generated with DALL·E 3

# Introductions



Sarah Huber

UVic Research Computing Services

[sahuber@uvic.ca](mailto:sahuber@uvic.ca)



Shahira Khair

UVic Libraries

[skhair@uvic.ca](mailto:skhair@uvic.ca)



Drew Leske

UVic Research Computing Services

[dleske@uvic.ca](mailto:dleske@uvic.ca)





**We acknowledge and respect the Ləkʷəŋən (Songhees and Esquimalt) Peoples on whose territory the University of Victoria stands, and the Ləkʷəŋən and WSÁNEĆ Peoples whose historical relationships with the land continue to this day.**

# Learning Objectives

This hands-on session will provide researchers with tools and techniques to make their research process more transparent and reusable in remote computing environments.

In this workshop, you'll learn about:

- Organizing your file directories
- Writing readable metadata with readme files
- Automating your workflows with scripts
- Capture and share your computational environment

# Workshop Outline

## Introduction to reproducibility

- Reproducibility “crisis”
- Elements of reproducible research
- Tools for reproducibility

## Exercises

1. Organizing and Documenting Data and Code
2. Automating Workflows
3. Capturing Computational Environments

# What is Reproducibility?

A quality of research involving some element of computation.

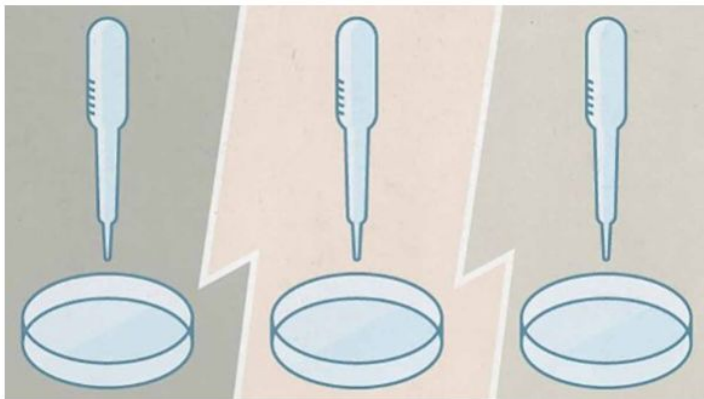
When the code and data are assembled in a way that another group can recreate your analyses and achieve the same result.

SPECIAL | 18 OCTOBER 2018

## Challenges in irreproducible research

Science moves forward by corroboration – when researchers verify others' results. Science advances faster when people waste less time pursuing false leads. No research paper can ever be considered to be the final word, but there are too many that do not stand up to further study.

There is growing alarm about results that cannot be reproduced. Explanations include increased levels of scrutiny, complexity of experiments and statistics, and pressures on researchers. Journals, scientists, institutions and funders all have a part in tackling reproducibility. Nature has taken substantive steps to improve the transparency and robustness in what we publish, and to promote awareness within the scientific community. We hope that the articles contained in this collection will help. [show less](#)



## *IS THERE A REPRODUCIBILITY CRISIS?*

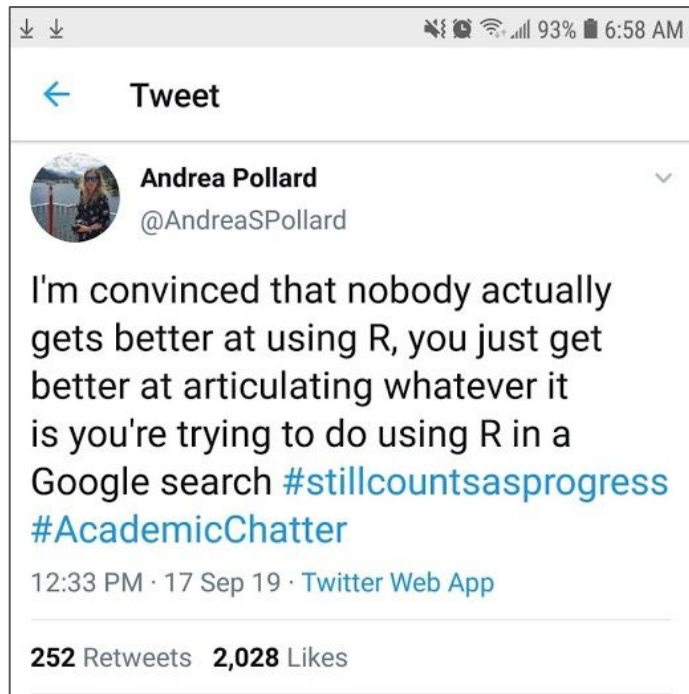
- ☐ Yes, significant crisis
- ☐ Yes, slight crisis
- ☐ No crisis
- ☐ Don't know



Who has ever tried to  
reproduce **your own**  
OR someone else's  
results?



<http://gph.is/1PaRTrX>



Question

Asked 9th Mar, 2020



Maggie Sabay

Grand Canyon University

I forgot to collect descriptive data and I have closed data collection for my dissertation. What should I do?

I am a novice writer working on my dissertation. As I started setting up data for analysis, I realized that I did not collect descriptive data. This was anonymous and there is no way to collect descriptive data at this time. Any suggestions on way forward?



*“An article about computational science in a scientific publication is not the scholarship itself, it is merely advertising of the scholarship.”*

[Buckheit and Donoho \(1995\)  
paraphrasing Jon Claerbout](#)

## **Scientific method in the 21st century...**

Most researchers are not formally trained as programmers, but increasingly need to deal with:

- Size and complexity of modern datasets
- Complexity of modern data analysis
- Sophistication and continuing advancement of software

# Spectrum of reproducibility

**High Reproducibility:** studies which provide the code, data, and full computational environment necessary to reproduce the results of the study

**Medium Reproducibility:** studies which provide the code and data but not the computational environment in which the code can be run

**Low Reproducibility:** studies which describe algorithms and results



# Iceberg of reproducibility

## Documentation

1

Usually written in separate effort from the study, describing the algorithms, results, environment

2

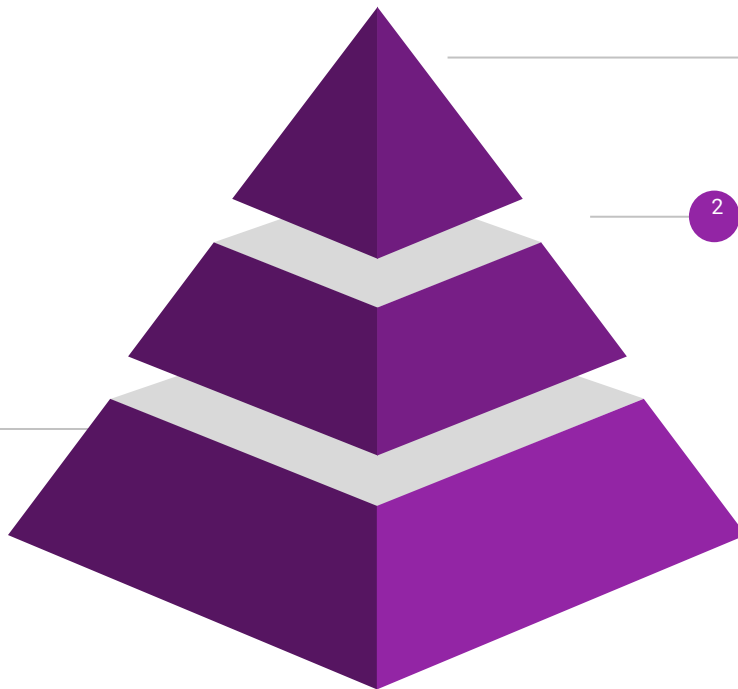
## Scripting and automation

Used to make the computational/analysis components of the study easily re-runnable. May capture some information on how the computational environment is set up or used

3

## Platform and environment

Captures the configuration of the computation environment (software, operating system)



[Tatman, VanderPlas, and Dane \(2018\)](#)

# Iceberg of reproducibility

A large, white iceberg floats on a calm, dark blue sea. The iceberg's surface is uneven and textured. A significant portion of the iceberg is submerged, appearing as a deep blue, translucent structure beneath the water's surface. The sky is a clear, pale blue.

## Documentation

Usually written in separate effort from the study, describing the algorithms, results, environment

## Scripting and automation

Used to make the computational/analysis components of the study easily re-runnable. May capture some information on how the computational environment is set up or used

## Platform and environment

Captures the configuration of the computation environment (software, operating system)

Adapted from [Tatman, VanderPlas, and Dane \(2018\)](#)

# Sharing $\neq$ Reproducible

- Could someone else:
- open it ?
  - understand it ?
  - run it ?
  - reuse it ?

*Consider...*

- Organization and documentation of files
- Different technical skills & experience of reusers
- Range of computing environments
- Impacts of proprietary software & licensing restrictions

# Testing reproducibility

- Queried Europe PMC for “jupyter OR ipynb”
- “My initial thought was that analysing the validity of the notebooks would simply involve searching the text of each article for a notebook reference, then downloading and executing it ... **It turned out that this was hopelessly naive...**”

<https://markwoodbridge.com/2017/03/05/jupyter-reproducible-science.html>

## Jupyter Notebooks and reproducible data science

### Introduction

One of the ideas pitched by [Daniel Mietchen](#) at the London [Open Research Data do-a-thon](#) for [Open Data Day 2017](#) was to [analyse Jupyter Notebooks mentioned in PubMed Central](#). This is potentially valuable exercise because these [notebooks](#) are an increasingly popular tool for documenting data science workflows used in research, and therefore play an important role in making the relevant analyses replicable.



# Challenge 1 = workflows are not portable

- A single step in your workflow could rely on multiple dependencies
- Documentation takes a lot of time and still can be imprecise
- Links break and code rots

# Challenge 2 = so many components

- Complex data analysis involves many inputs and outputs
- Annotated code, readme files, and other documentation are helpful but still separated from the data, analysis, and results

# Challenge 3 = collaborators have different skills + computer setups

*“A scientist unwilling to disenfranchise their collaborators could certainly elect to use more widely used tools, accepting frustration with inefficiency as the price for collaboration.*

*However, the price is often paid in reproducibility as well when those widely used, lowest-common denominator tools conflict with reproducibility goals”*

from [Kathryn Huff \(2018\). Lessons Learned. In \*The practice of reproducible research\*.](#)

# Challenge 4 = Computational environments change with time

- Software of all levels requires updates to remain usable
- Research software challenging to maintain
- Loss of backwards compatibility
- For what amount of time should functionality be preserved?
- Hardware also changes

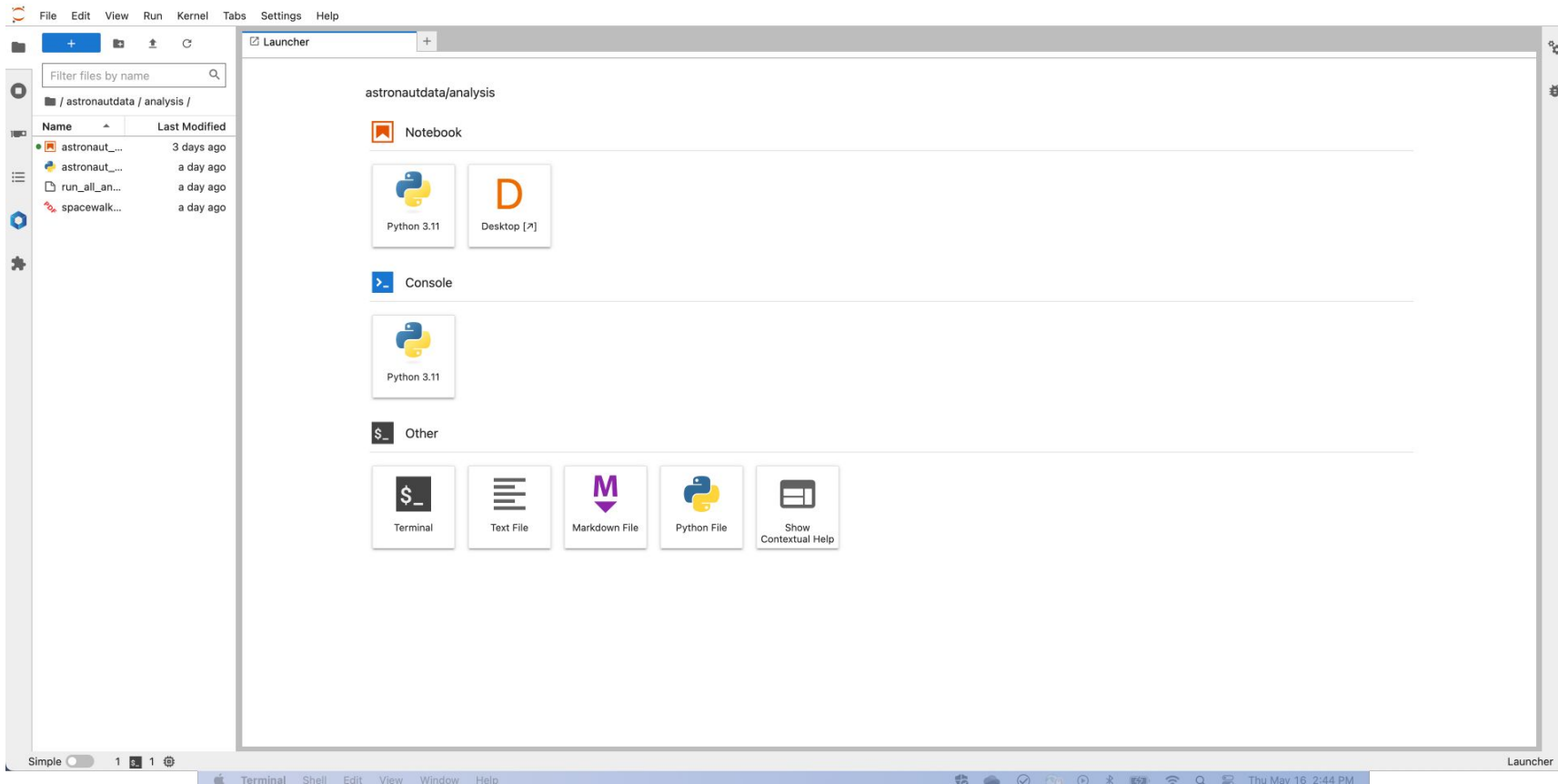


**Discussion break**



Generated with DALL·E 3

# Today's computing environment



# Today's computing environment

Login info at <https://training.computeontario.ca>  
(Click and scroll to bottom)

The screenshot shows the Compute Ontario training portal. At the top, there is a navigation bar with the Compute Ontario logo on the left and links for Home, Dashboard, My courses, 2024 Summer School, Compute Ontario, Alliance Canada, and Help. On the right side of the navigation bar are notification icons, a user profile icon labeled 'SH', and an 'Edit mode' toggle switch.

On the left side of the main content area is a sidebar menu. It has a close button (X) and a menu icon (three dots). The menu is divided into two sections: 'General' and 'Monday, June 10: 13:30 to 16:30 Session'. Under 'General' are links for 'Need/Want to Unenrol from...', 'Need Help? Need to Contac...', 'Announcements', 'Compute Ontario Summer ...', 'Attendance', 'Attendance Certificate', 'Live Course Video Links', and 'Instructor(s):Unless explicit...'. Under the session section is a link for 'Magic Castle Username an...'. The session section is currently selected and highlighted with a blue background.

The main content area displays the breadcrumb '2024-COSS-ReprodRes / Monday, June 10: 13:30 to 16:30 Session / Magic Castle Username and Password'. Below this is the title 'Magic Castle Username and Password' with a link icon. Under the title are three tabs: 'URL' (which is active and underlined), 'Settings', and 'More'. Below the tabs is a large light blue rectangular area with a 'View' button in the top left corner. Below this area is a text instruction: 'Click on Magic Castle Username and Password to open the resource.'

In the bottom right corner of the page, there is a small circular help icon with a question mark.


# Activity scenario

You've recently joined a new lab and have inherited the last post-doc's work with all their data on astronauts...

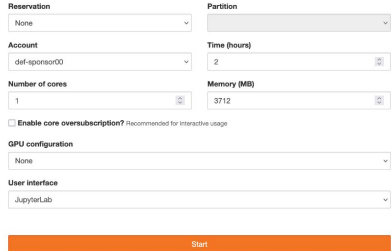


# Exercises- accessing the computing environment

- Go to <https://jupyter.coss-b.c3.ca> in your web browser
- Sign in using your provided username and password.
- Change “Time” to about 2.5 hours, and click Start
- Let us know if any questions!



The image shows the JupyterHub sign-in page. It has a header with the JupyterHub logo. Below it is a 'Sign in' button. Underneath is a form with 'Username:' and 'Password:' labels, each followed by a text input field. At the bottom of the form is an orange 'Sign in' button. Below the form are two links: 'Create Account' and 'Reset Password'.

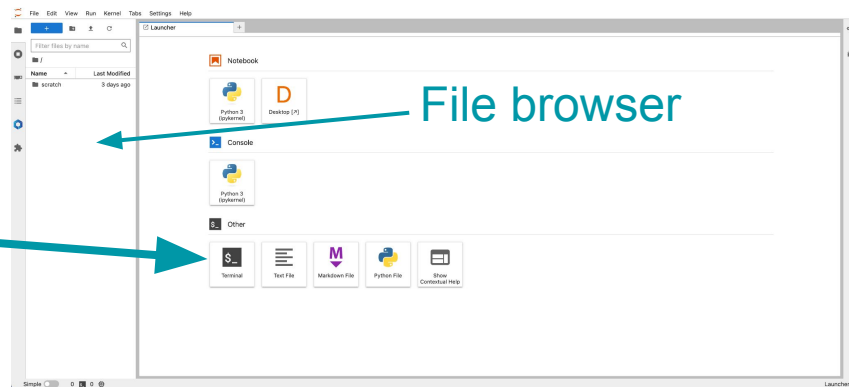


The image shows the JupyterHub 'Server Options' page. The header includes the JupyterHub logo, 'Home', 'Token', 'Services', and a user profile 'User735' with a 'Logout' link. The main content area is titled 'Server Options' and contains several configuration sections: 'Reservation' (dropdown set to 'None'), 'Partition' (dropdown set to 'None'), 'Account' (dropdown set to 'def-sponsor00'), 'Time (hours)' (input field set to '2'), 'Number of cores' (input field set to '1'), and 'Memory (MB)' (input field set to '3712'). There is a checkbox for 'Enable core oversubscription?' which is unchecked. Below this is a 'GPU configuration' dropdown set to 'None'. Then there is a 'User interface' dropdown set to 'JupyterLab'. At the bottom is a large orange 'Start' button. Two blue arrows point to the 'Time (hours)' field and the 'Start' button.

# Exercises- downloading the data

1. Open Terminal
  - a. From Launcher
  - b. Or under *File-> New -> Terminal*
2. From Terminal, enter the below command to download the data:

```
$ git clone https://github.com/dleske/reproducibility-workshop-202
```



3. Navigate to “exercise\_1” folder in file browser (left)



# Organize your files

- Create **one repository** that holds all your related research files

```
Basic directory
```

```
|--astro_dataset
```

# Think about the workflow....

- How will files be created?
- In what format?
- In what order?
- With what stability?

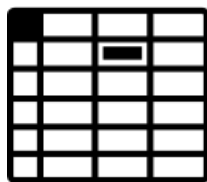
# For example...



Questionnaire  
(physical)



Scan  
(.pdf)



Spreadsheet  
(.csv)



NVIVO  
(.nvp)



Transcript  
(.txt)



Figures  
(.jpeg)



Journal Article  
(.txt)



Codes,  
annotations  
(.txt)



Interview  
(.mp3)



# Organize your files

- Create one repository that holds all your related research files
- **Organize your files to distinguish your data, code, and results**

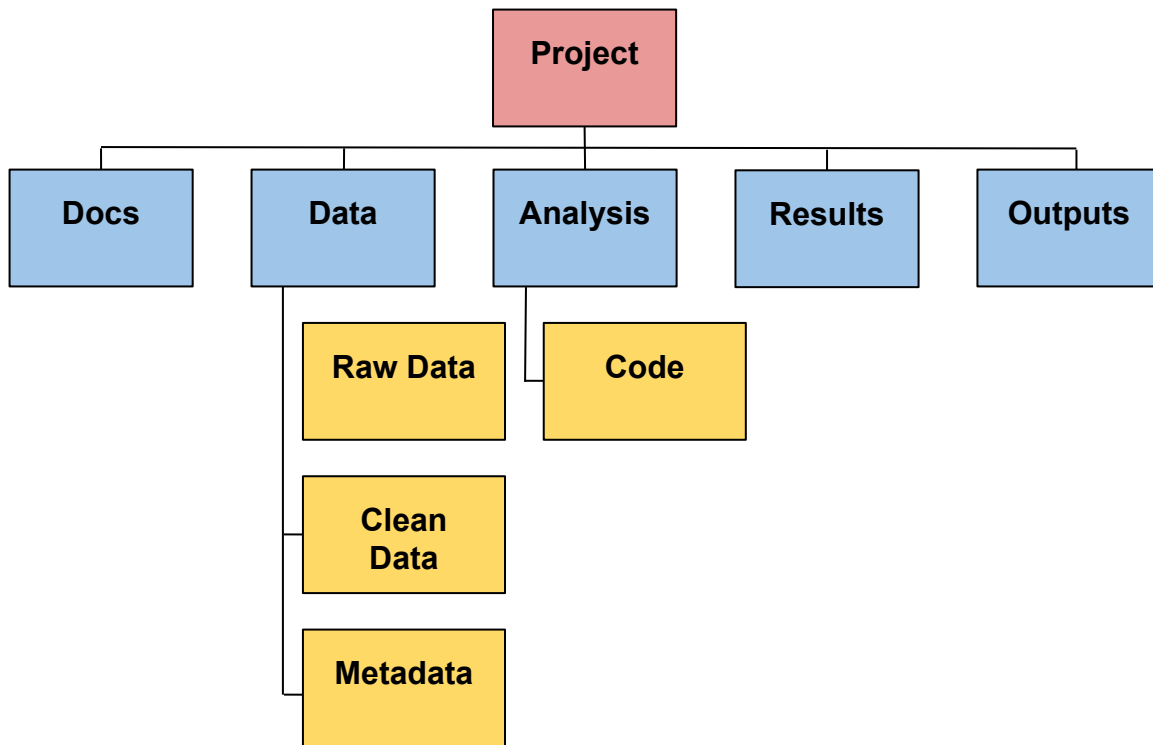
```
Basic directory
```

```
|--astro_dataset
```

# Filesystem

A filesystem organizes a computer's files and directories into a tree structure:

- The first directory in the filesystem is the root directory.
- Each directory can contain more files and child directories.



# Organize your files

- Create one repository that holds all your related research files
- Organize your files to distinguish your data, code, and results
  - **Separate input and output data**

Basic directory

```
|--astro_dataset
|   |  -- data_raw
|   |  |  -- launch_dat.csv
|   |  |  -- transcripts.csv
```

# Organize your files

- Create one repository that holds all your related research files
- Organize your files to distinguish your data, code, and results
  - **Separate input and output data**

Basic directory

```
|--astro_dataset
|   |  -- data_raw
|   |  |  -- launch_dat.csv
|   |  |  -- transcripts.csv
|   |  -- data_clean
|   |  |  -- clean_launch_dat.csv
|   |  |  -- coded_transcripts.csv
```



# Organize your files

- Create one repository that holds all your related research files
- Organize your files to distinguish your data, code, and results
  - Separate input and output data
  - **Separate scripts from data**

Basic directory

```
|--astro_dataset
|   | -- data_raw
|   |   | -- launch_dat.csv
|   |   | -- transcripts.csv
|   | -- data_clean
|   |   | -- clean_launch_dat.csv
|   |   | -- coded_transcripts.csv
|   | -- src
|   |   | -- analysis_launch_dat.R
|   |   | -- process_launch_dat.R
```

# Organize your files

- Create one repository that holds all your related research files
- Organize your files to distinguish your data, code, and results
  - Separate input and output data
  - Separate data from scripts
  - **Separate results from data and scripts**

Basic directory

```
|--astro_dataset
|   |  -- data_raw
|   |    |  -- launch_dat.csv
|   |    |  -- transcripts.csv
|   |  -- data_clean
|   |    |  -- clean_launch_dat.csv
|   |    |  -- coded_transcripts.csv
|   |  -- results
|   |    |  -- t-test_results.txt
|   |    |  -- fig1_freq.jpg
|   |    |  -- fig2_distribution.jpg
|   |  -- src
|   |    |  -- analysis_launch_dat.R
|   |    |  -- process_launch_dat.R
```

# Organize your files

- Create one repository that holds all your related research files
- Organize your files to distinguish your data, code, and results
  - Separate input and output data
  - Separate data from scripts
  - Separate results from data and scripts
- **Use clear and explicit file names**

Basic directory

```
|--astro_dataset
|   | -- data_raw
|   |   | -- launch_dat.csv
|   |   | -- transcripts.csv
|   | -- data_clean
|   |   | -- clean_launch_dat.csv
|   |   | -- coded_transcripts.csv
|   | -- results
|   |   | -- t-test_results.txt
|   |   | -- fig1_freq.jpg
|   |   | -- fig2_distribution.jpg
|   | -- src
|   |   | -- analysis_launch_dat.R
|   |   | -- process_launch_dat.R
```

# Organize your files

- Create one repository that holds all your related research files
- Organize your files to distinguish your data, code, and results
  - Separate input and output data
  - Separate data from scripts
  - Separate results from data and scripts
- Use clear and explicit file names
- Include a **readme file** in your main directory

## Basic directory

```
|--astro_dataset
|  |  -- data_raw
|  |  |  -- launch_dat.csv
|  |  |  -- transcripts.csv
|  |  -- data_clean
|  |  |  -- clean_launch_dat.csv
|  |  |  -- coded_transcripts.csv
|  |  -- results
|  |  |  -- t-test_results.txt
|  |  |  -- fig1_freq.jpg
|  |  |  -- fig2_distribution.jpg
|  |  -- src
|  |  |  -- analysis_launch_dat.R
|  |  |  -- process_launch_dat.R
|  |  -- README.txt
```

# How to write a README

[readme template](#)

## #### Project-level Description

Provide a description of your dataset, highlighting its purpose and features.

## #### Attribution

Acknowledge contributors, include a main contact, and cite sources.

## #### License

Specify the project's license to clarify how others can use your code.

## #### Installation / Methodology

Provide instructions on install, including prerequisites. Explain how to navigate folder structure and files, how to run code.

## #### File-level Description

Describe files to ensure completeness of understanding - how many variables do they contain, define codes, specify units, etc.

# Exercise 1

1. Import the [dataset](#) into Jupyter notebook

```
$ git clone https://github.com/dleske/reproducibility-workshop-202
```

2. Review and organize files following best practices
3. Draft a readme using this [template](#)

Automating your workflow with scripts



# Common workflow styles

- GUI (graphical user interface)
- Command line interfaces
- Executable scripts

```
top - 18:15:44 up 38 days, 22:12, 0 users, load average: 4.38, 4.29, 4.46
Tasks: 1422 total, 2 running, 1408 sleeping, 0 stopped, 12 zombie
%Cpu(s): 6.5 us, 0.0 sy, 0.0 ni, 93.4 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 257372.6 total, 85146.2 free, 161519.2 used, 10707.2 buff/cache
MiB Swap: 0.0 total, 0.0 free, 0.0 used. 88593.8 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
2689516		20	0	16.9g	12.1g	208332	S	399.3	4.8	510:20.79	pt_main_thread
2867822		20	0	1589888	242640	51476	R	16.8	0.1	0:07.43	python
2883967		20	0	23112	5820	3672	R	1.0	0.0	0:00.41	top
1912368		20	0	695636	164268	8460	S	0.7	0.1	0:40.41	batchspawner-si
2		20	0	0	0	0	S	0.3	0.0	35:14.65	kthreadd
5260		20	0	0	0	0	S	0.3	0.0	15:51.00	txg_sync
2802468		20	0	244212	6728	5448	S	0.3	0.0	0:00.66	slurmstepd
2815235		20	0	694760	166588	22200	S	0.3	0.1	0:06.97	batchspawner-si
1	root	20	0	243096	16164	9220	S	0.0	0.0	7:17.60	systemd
3	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_gp
4	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_par_gp
5	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	slub_flushwq
7	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kworker/0:0H-events_highpri
11	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	mm_percpu_wq
12	root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcu_tasks_rude_
13	root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcu_tasks_trace
14	root	20	0	0	0	0	S	0.0	0.0	0:19.29	ksoftirqd/0
15	root	20	0	0	0	0	I	0.0	0.0	12:55.42	rcu_sched
16	root	rt	0	0	0	0	S	0.0	0.0	0:04.04	migration/0
17	root	rt	0	0	0	0	S	0.0	0.0	0:00.11	watchdog/0

# Scripting languages



- Bash

- awk/sed

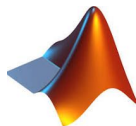
The **AWK**  
Programming  
Language



- Python



- R



- Matlab



- Perl



- Julia

# Basic bash commands

\$ ls            lists files and directories

\$ pwd          print working directory

\$ cd           change directory

\$ mkdir        create new directory

\$ touch        update/create new file

\$ cp           copy files (add -R for copying folders)

\$ mv           move files

\$ rm           remove (add -R for deleting folders)

# A first bash script

```
#!/bin/bash
```

```
echo "Hello world" #print out the text "Hello world"
```

```
cd src #move into to "src" directory
```

```
python my_python_script.py #Run using a python interpreter
```

# Sample use cases for scripts

- Analyze output files in separate folders (or the same ones)
- Prepare input files for multiple jobs
- Compile a new software
- Set up a virtual environment

*If you have to do it more than once, write a script.*

*If you want to remember how you did it, write a script.*



# Document your code

- Do everything with a script
- Specify software environment version
- Specify all dependencies (& versions!)
- Use relative paths
- Annotate scripts explaining steps and decisions

**\*\*Print out configuration\*\***

○ R: sessionInfo()

○ Py: IPython.sys\_info()  
pip freeze

# Using loops

## Bash

```
#!/bin/bash

# Loop through all subfolders

for dir in $(pwd); do

    if [ -d "$dir" ]; then

        echo "$dir"

    fi

done
```

## Python

```
import os

for item in os.listdir(os.getcwd()):

    item_path = os.path.join(os.getcwd(), item)

    if os.path.isdir(item_path):

        print(item_path)
```



# Using variables

## Bash

```
#!/bin/bash

current_directory=$(pwd)

# Loop through all subfolders

for dir in "$current_directory"/*; do

    if [ -d "$dir" ]; then

        echo "$dir"

    fi

done
```

## Python

```
import os

current_dir = os.getcwd()

for item in os.listdir(current_dir):

    item_path = os.path.join(current_dir, item)

    if os.path.isdir(item_path):

        print(item_path)
```

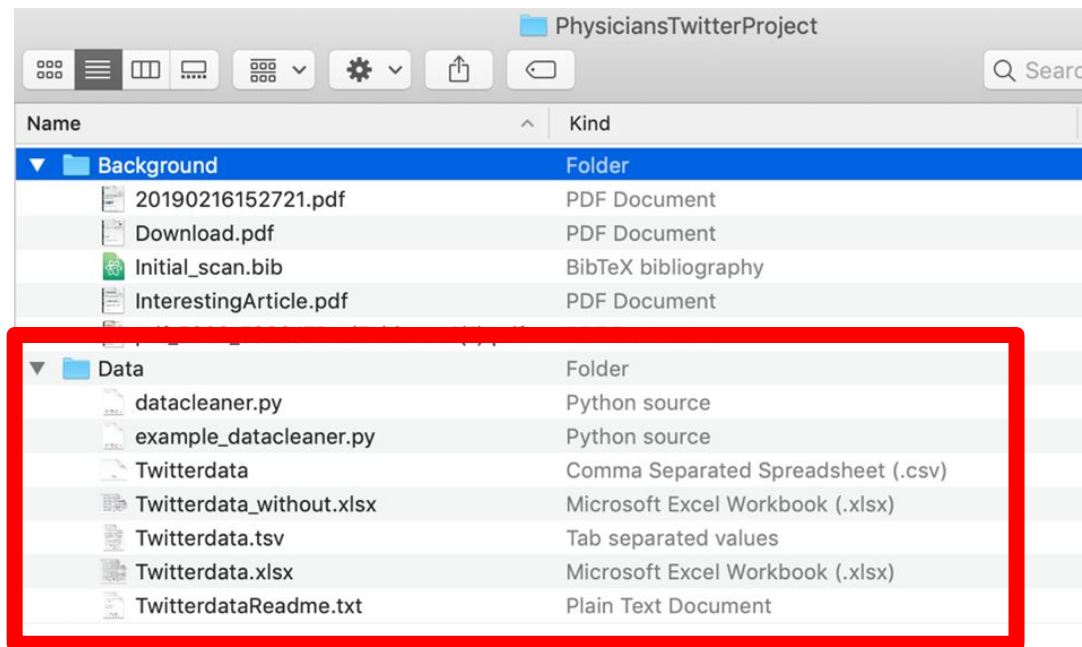
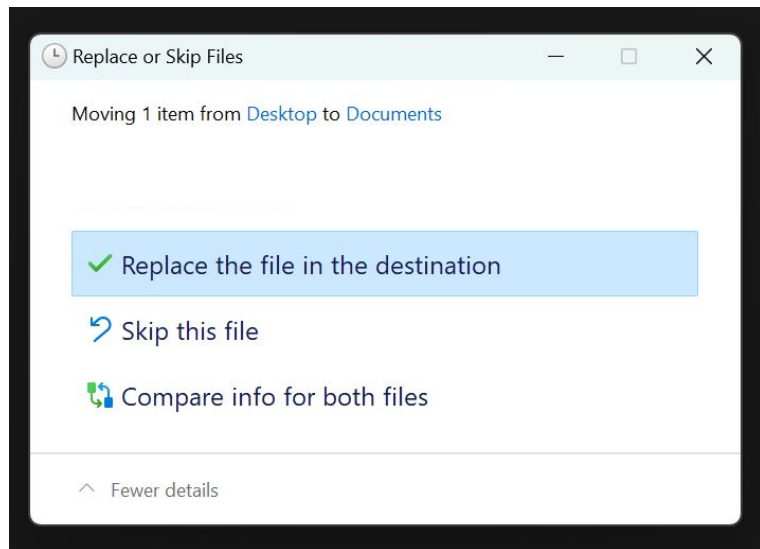
# Using version control

- Combines nicely with scripts and automation!
- Track changes
- Git example:

```
$ git diff
diff --git a/paper/manuscript.tex b/paper/manuscript.tex
index 3fe30be..7b6f0f1 100644
--- a/paper/manuscript.tex
+++ b/paper/manuscript.tex
@@ -20,7 +20,7 @@ This document was drafted with the assistance of Microsoft Copilot.

\section{Introduction}
The age of astronauts has been a topic of interest for many years.
-This paper presents a few simple statistical analysis of the age of astronauts.
+This paper presents a statistical analysis of the age of astronauts and some related analysis around age and spacewalk records.
```

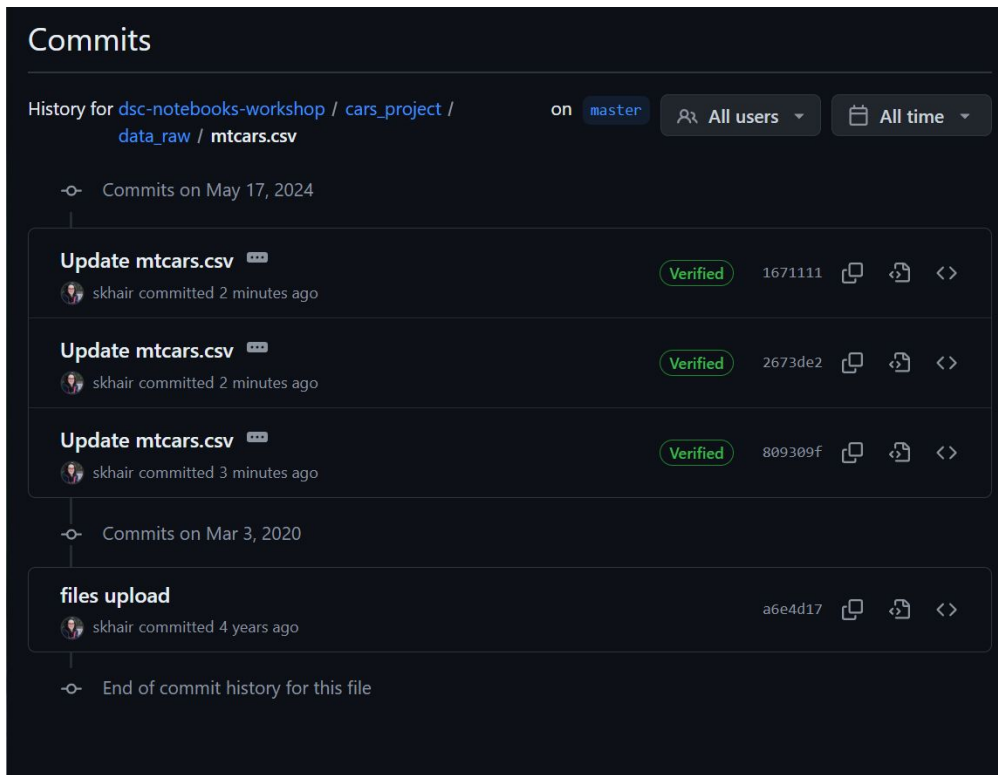
# Version control



**Don't do this...**

# Version control

- **Audit** — see who made what changes and when.
- **Undo/redo work** — go back to a previous version as needed.



The screenshot shows the commit history for the file `mtcars.csv` in the `cars_project / data_raw` directory. The interface is dark-themed. At the top, it says "Commits" and "History for dsc-notebooks-workshop / cars\_project / data\_raw / mtcars.csv". There are filters for "on master", "All users", and "All time". A section titled "Commits on May 17, 2024" contains three commits, all titled "Update mtcars.csv" by user "skhair". Each commit is marked as "Verified" and includes a commit hash (1671111, 2673de2, 809309f) and icons for file diff, code diff, and commit details. A section titled "Commits on Mar 3, 2020" contains one commit titled "files upload" by user "skhair" with commit hash "a6e4d17" and the same icons. At the bottom, it says "End of commit history for this file".

## Exercise 2:

1. Navigate to “exercise\_2” folder in file browser
2. Review the different subfolders, files, and scripts.
  - a. What are the input files, and what is produced?
  - b. What does each script do?
3. You’ve received new data! Update the collection to use data from the aggregated file available for download from [https://object-arbutus.cloud.computecanada.ca/RCSWorkshopMedia/2024-06-10-COSS-Reproducible-Research/astronauts\\_full.csv](https://object-arbutus.cloud.computecanada.ca/RCSWorkshopMedia/2024-06-10-COSS-Reproducible-Research/astronauts_full.csv) instead of the current input file.
4. Re-create the manuscript to see what’s changed!

**Capturing environment**

# Recap:

## 1. Documentation

- A “how-to” for reproducing your science. A great start! Everybody does this. (Right?)
- Small and portable
- Easy to archive and readable forever unless it's saved in WordStar
- Tell me what to do
- Tell me where to find the software tools I need

BUT:

- The tools aren't there anymore?
- The instructions are confusing/vague/incomplete

# Recap:

## 2. Scriptage

- An automated playbook for reproducing your science. Awesome!
- Everybody can do this. (Right? Mostly. Lots of help available!)
- Still small and portable!
- Easy to archive and runnable forever unless it's, uh, AmigaDOS
- Show & tell me what to do, can function as documentation!
- Encapsulates where to find the software tools needed

BUT:

- The tools aren't there anymore? (404!)



# So now what?

You could package up the necessary software and include that?

- Licensing - no way around that, probably, we'll have to set that aside for now, we'll have to assume open-source or freely distributable software.
- Do you package the binary, or the source?
- What about dependencies? Libraries, OS components

What if you could capture your software environment,  
and put that on a shelf or send it to someone?



# Virtual machines

A virtual machine (VM) is a software representation of a computer, its operating system, software and configuration.

You can treat this like a regular computer, add whatever software and configure it however you like, and then make copies.

VMs are based on *images*. You take an image, create a VM from it, and add whatever you want, like installing your favourite language and libraries.

You can create your own images from a VM by taking a *snapshot*. This captures the entire configured state of the VM.

A VM instantiated from this snapshot will then be identical to the original.

# Virtual machines, cont'd.

*But.*

- VM images can be huge and contain stuff you don't really need
- VM images can be tricky and/or tedious to create
- Multiple competing formats
- Portability is fantastic so long as where you want to run it supports that type of image
- VMs package up everything in case you need it.



# Containers

Two ways to look at containers:

1. “VM Lite”
2. A mechanism for packaging software.

Containers encapsulate software and configuration but rely on the host machine’s kernel and hardware.

(Note we are talking about *application containers*, not *system containers*, which really are like “VM Lite” but have their own drawbacks, portability issues, etc.)

# Containers, cont'd.

Yay!

- Smaller than VMs
- Only have what you need
- Easier to create—*we're going to make one in a few minutes!*
- More portable: two primary formats, each of which is runnable everywhere\*

Boo.

- Relies on host system support, host system kernel
- ...

\* Sort of (sssh!)



# Containers, cont'd redux.

Two main types:

- Docker: microservices, Kubernetes, ...
  - Composed of layers
  - Hugely popular, used by all the major cloud providers, run 92.442% of the internet\*, used in fighter planes\*\*
  - Sort of open source, mostly!
- Apptainer (né Singularity): for *science*. Can run on HPC.
  - Single image file
  - Built for reproducibility of research!
  - Open source!

\* *Made that up*

\*\* *Did not make that up - the F22 Raptor uses Kubernetes in its avionics*

# So, Apptainer

- Designed for HPC
- Designed for research
- Designed for reproducibility
- Support HPC usage profiles (MPI, InfiniBand, ...)



# The Bad News

*Not magic.*

Our national HPC environments use common software builds across heterogeneous environments so within the Alliance your analysis codes can run anywhere. Beluga down? Try a few cycles on Cedar.

This software stack is amazing and does what it sets out to do very well. But it does not lend itself to containerization.

I don't have a solution for you on this. At least not a magic one.



# Real-world example: containerizing Exercise 2

I set out to take Sarah's scripted analysis and make that into a portable container and ran into some problems.

- Uses `lmod` (`module load this-n-that`) - no easy way to reproduce this inside the container
- Had to *reproduce* as best I could, instead of *capture*, the environment
- Had to figure out the important parts of `module load scipy-stack/2023b`
- Adjusted the scripts to not depend on being in specific directories

Takes 15 minutes to build on a dedicated VM. Not great for a class exercise.

Definition file, etc. in repo under `exercise_3_pt1`.

Finished image under `/home/user0952/share/analysis2.sif`.

## Exercise 3: Let's build a container!

We'll build an Apptainer container that is:

- Based on a specific software image (Python 3.11.5)
- Installs a specific software package and version
- Does a thing.

We will build the image definition in parts, from scratch.

The *solution* is in ``exercise_3_pt2`` if you get stuck.

## Exercise 3: Creating the definition file

A definition file for an Apptainer image is a text file with a simple format, traditionally with the extension “**.def**”.

Log in to the course computing environment using the supplied credentials, and use your favourite editor to start editing a file “**figgy.def**”.

If you're interested after the course:

- [https://apptainer.org/docs/user/latest/quick\\_start.html](https://apptainer.org/docs/user/latest/quick_start.html)
- [https://apptainer.org/docs/user/latest/definition\\_files.html](https://apptainer.org/docs/user/latest/definition_files.html)

## Exercise 3: The header section

In the header section we define the base for our image. Generally we build on other images.

- Using a Docker image
- We specify image and tag
- Apptainer supports *multi-stage builds* which we are not using here, but it's shown here to plant a seed.

```
Bootstrap: docker
From: python:3.11.5
Stage: build
```

Reference:

- [https://apptainer.org/docs/user/latest/definition\\_files.html#header](https://apptainer.org/docs/user/latest/definition_files.html#header)

## Exercise 3: The %files section

**We don't need this for our exercise.** I'm just telling you about it because it's often important (see `exercise\_3\_pt1` for a practical example).

The files section allows you to copy files from the host system into your container image.

(Remember, don't actually add this to your definition!)

```
%files  
source/ /analysis
```

Reference:

- [https://apptainer.org/docs/user/latest/definition\\_files.html#files](https://apptainer.org/docs/user/latest/definition_files.html#files)

## Exercise 3: The %post section

In the %post section we define a script that installs and configures software inside our image.

Text in this section defines a script which is run in building the image. Make sure it's indented properly so Apptainer knows when the script is done.

```
%post
    pip install pyfiglet==1.0.2
```

Reference:

- [https://apptainer.org/docs/user/latest/definition\\_files.html#post](https://apptainer.org/docs/user/latest/definition_files.html#post)

## Exercise 3: The %runscript section

The `%runscript` section defines the script (indented like `%post`) that will be executed when the container is run.

In our example, we run the `pyfiglet` tool that we installed in the `%post` section, but we could have a longer script with additional commands.

Note the ``"$@"``: this allows us to pass arguments to the script when running our container.

```
%runscript  
    pyfiglet "$@"
```

Reference:

- [https://apptainer.org/docs/user/latest/definition\\_files.html#runscript](https://apptainer.org/docs/user/latest/definition_files.html#runscript)

## Exercise 3: The %test section

An optional `%test` section allows us to verify some aspect of our image during the build. If the script defined here exits with a non-zero status, the build fails. It can also be invoked afterwards with ``apptainer test``.

Our test here is pretty trivial, but it verifies that the program was installed and runs. A longer example is available in the definition file provided.

```
%test  
pyfiglet --version
```

Reference:

- [https://apptainer.org/docs/user/latest/definition\\_files.html#test](https://apptainer.org/docs/user/latest/definition_files.html#test)



## Exercise 3: The %help section

The %help section is also optional and makes information available to the user if they invoke the ``apptainer run-help`` command on the container.

```
%help
```

```
This container runs a Python Figlet script that prints  
a banner to the console from the text given on the  
command line.
```

Reference:

- [https://apptainer.org/docs/user/latest/definition\\_files.html#help](https://apptainer.org/docs/user/latest/definition_files.html#help)

## Exercise 3: The %labels section

The %labels section, also optional, allows the specification of metadata in the container image. These are arbitrary, simple key-value pairs.

```
%labels  
  Maintainer Drew Leske  
  Version 0.0.1
```

Reference:

- [https://apptainer.org/docs/user/latest/definition\\_files.html#labels](https://apptainer.org/docs/user/latest/definition_files.html#labels)

## Exercise 3: Other sections

There are other sections we don't need for our example and haven't included but sometimes very useful:

- **%arguments** allows for customizing the image on build through the command line
- **%setup**: for preparation *on the build host* before **%post** (careful!)
- **%environment**: for setting env. variables for the container runtime

Maybe less useful:

- **%app**: advanced topic, left as an exercise to the reader
- **%startscript**: used when operating the container as an *instance*

## Exercise 3: Building the container

That's it! Save and close, and you now have a definition file. 🥳 Shall we try building it?

**HOLD ON THOUGH**



## Exercise 3: Building the container (on the cluster)

Building containers can consume a non-trivial amount of resources, and for our trivial container, the building is more intensive than the running. The login node is not the place to do this as it will impact others. We'll submit this to the cluster as a job.

I've provided a job file for this in the ``exercise_3_pt2`` directory: `make-figgy.job`.

Copy this into the same directory as your `figgy.def` file, and then run:

```
$ sbatch make-figgy.job  
Submitted batch job 114
```

## Exercise 3: Building the container (on the cluster)

Watch for the job to finish:

```
$ watch squeue
```

You should see your job. When it disappears Ctrl-C and you should have a job output (ex. `slurm-114.out`) and if the build was successful, a new image file (ex. `figgy.sif`).

No image file?

- Have look at the output file and see if you can figure out what the issue was, and try again.
- If all else fails, grab mine: `cp ~user0952/share/figgy.sif .`

## Exercise 3: Try some things with your container

Let's experiment with our new container!

- Look at the metadata
- Get help on running it
- Run the container

```
apptainer inspect figgy.sif
apptainer run-help figgy.sif
apptainer run figgy.sif hi
./figgy.sif hi hello hooray
```

# Next steps

- Copy your container off the MC and run it somewhere else, like another Linux distro (so long as the apptainer runtime is there)
- Read up on Apptainer:  
<https://apptainer.org/docs/user/latest/introduction.html>
- Explore the Alliance's documentation on Apptainer:  
<https://docs.alliancecan.ca/wiki/Apptainer>



# Afterparty!



# TL;DR on Reproducibility

- Develop an organized filesystem
- Document your dataset with a readme
  - Authors and source info
  - Variable codes, units, etc.
  - Software version and dependencies
  - License
- Automate your workflow with scripts
- Document your code
  - Specify dependencies
  - Add annotations
  - Use relative paths
- Implement version control practices
- Use containers to capture your computer environment



# Git repository with our teaching materials:

<https://github.com/dleske/reproducibility-workshop-2024/>

## Additional Resources:

Content in today's workshop was repurposed from an earlier 2019 workshop for data curators:

- Khair & Sawchuk. (2019). "Curating Data Sets for Reproducibility", GitHub repository. <https://research-reuse.github.io/>

Suggested readings on reproducibility:

- Broman, K. (n.d.) Initial steps toward reproducible research. Accessed May 2024 from <https://kbroman.org/steps2rr/>
- The practice of reproducible research: Case studies and lessons from the data-intensive sciences. Kitzes, J., Turek D., & Deniz, F. (Eds.). (2018). Oakland, CA: University of California Press. <https://www.practicereproducibleresearch.org>