Computational Reproducibility (Research Reproducibility in Theory and Practice, Day 3, FSCI2021)

Daniel S. Katz

d.katz@ieee.org, @danielskatz

Slides and examples are in https://bit.ly/CompRepro

I ILLINOIS

NCSA | National Center for Supercomputing Applications

With thanks to many! See last slides for sources

Exercise

- Take a look at this dataset: https://osf.io/z274d/
- Download it via: https://osf.io/z274d/download
- Contains demographic data: tab-separated with a header row
- Using this data, create a graph that shows life expectancy in Canada between 1980 and 2000
- Write down how you did it, and give it to someone else, then ask them to reproduce it

```
year	pop	lifeexp	gdppercap	country	continent
1952	8425333	28.801	779.4453145	afghanistan	asia
1957	9240934	30.332	820.8530296	afghanistan	asia
…
```


Credit: Soderberg, C. K., Smith, J. F., & Katz, D. S. (2017, August 1). Computational Reproducibility Day. Retrieved from http://osf.io/sbnz7

Goals

- Reproducibility
- Of what?
  - Papers, results, figures
- By whom?
  - Future you, someone else knowledgeable in your field, anyone else
- When?
  - Tomorrow, six months, 5 years, 50 years
- How much?
  - Close enough (you decide what this means), not necessarily all the bits
  - Plausible vs. practical

Defining R* - terms

- Reproducibility, Replicability, Repeatability, etc.
- Confusing terms – see "Replicability vs. reproducibility — or is it the other way around?" (blog) and "Reproducibility vs. Replicability: A Brief History of a Confused Terminology" (paper) for some discussion
- Maybe these are getting to be more standardized? But still, define what you mean!

Goodman	Claerbout	ACM (2020+)	AMC (2020-)
		Repeatability	Repeatability
Methods Reproducibility	Reproducibility	Reproducibility	Replicability
Results Reproducibility	Replicability	Replicability	Reproducibility
Inferential Reproducibility			

Goodman, S. N., Fanelli, D., and Ioannidis, J. P. A. (2016). What does research reproducibility mean? *Sci. Transl. Med.*8:341ps12.
Claerbout, J. F., and Karrenbach, M. (1992). Electronic documents give reproducible research a new meaning. *SEG Expanded Abstracts* 11, 601–604.
Association of Computing Machinery (ACM) (2020). Artifact Review and Badging (Version 1.1).

Context: data science

- Organize and analyze large (or small) data sets to learn from them
  - Steps: capture/acquire, organize, process, analyze, communicate
- Examples
  - How fast are stars moving away from us, and how does this vary with their distance?
  - Which credit card transactions are fraudulent?
  - What does this German document say in English? What does this recording of someone speaking Spanish say?
  - Which patient scans contain tumors?
  - Who’s going to win the election?
  - If a patient has these symptoms, what disease do they have?
  - What treatment is best for this particular patient?
- Relevant: statistics, preregistration (declare your hypothesis before doing your analysis), random studies, false positives/negatives, sample size, confidence, power
- Typical outputs: data, tools and methods (algorithms, models, software), conclusions (understanding data)

Context: computational science

- Modeling or simulating a (physical) process in a predictive way, often using one or more equations
- Examples, simulation or analysis of:
  - Atmospheric or oceanic circulation, coupled together with other physical processes into a climate simulation
  - The interactions of atoms in one or more molecules (drug design)
  - The atoms and forces in a material (material design)
  - Engineering analysis of the stress or deformation of a structure under some load (mechanical engineering)
  - Electrical signals in a circuit board or a set of synapses (electrical engineering or neuroscience)
  - Microwaves focused on a breast tumor (patient-specific medicine)
- Often called computational science & engineering (CSE)
- Relevant: mathematics, error bounds
- Typical outputs: algorithm, method, software, conclusions (understanding processes)

Computational reproducibility principles

1. Provide structure
2. Control the source & changes
3. Use notebooks to explain and document
4. Automate steps
5. Automate everything
6. Capture the environment
7. Provide a license & make citable

First thing – get a terminal

- On a Mac
  - Click the Launchpad icon in the Dock, type Terminal in the search field, then click Terminal.
  - In the Finder, open the /Applications/Utilities folder, then double-click Terminal.
- On Windows
  - Open your computer's Start menu. Click the Windows icon on the bottom-left corner of your desktop or press the ⊞ Win key on your keyboard
  - Type `cmd` or `Command Prompt`. After opening the Start menu, type this on your keyboard to search the menu items. Command Prompt will show up as the top result.
  - Click the `Command Prompt` app on the menu. This will open the Command Prompt terminal in a new window.
- Using Binder
  - Go to https://github.com/danielskatz/repro-fdtd1d, click on launch binder
  - Once binder starts the repo, use “New” -> “Terminal” to get a terminal

Principle 1 – Provide structure

- Use directories for different things, all inside a project directory, with a top-level readme and license
  - E.g., data, docs, models, notebooks, references, reports, src (for Python data science, Cookiecutter Data Science is an example)
- Use relative paths, so that you can move and share
  - (../data/file.dat)
- Use names that have meaning (and avoid using "final")
  - 00-dsk-data_acquisition.py

```
My_project
  |--data
  |--docs
  |--notebooks
  |--references
  |--reports
  |--src
```

Principle 2 – Control the source & changes

1. For data, store the original (raw) data archivally somewhere and build other versions elsewhere using scripts (including accessing the data from the archive)
   - Note: GitHub is not archival, and isn't good for large datasets
     - Two previous versions of this class used data in https://raw.githubusercontent.com/csoderberg/test_study/master/gapminder_copy.txt but this no longer exists
   - However, it is still in OSF: https://osf.io/z274d/
   - Get it via

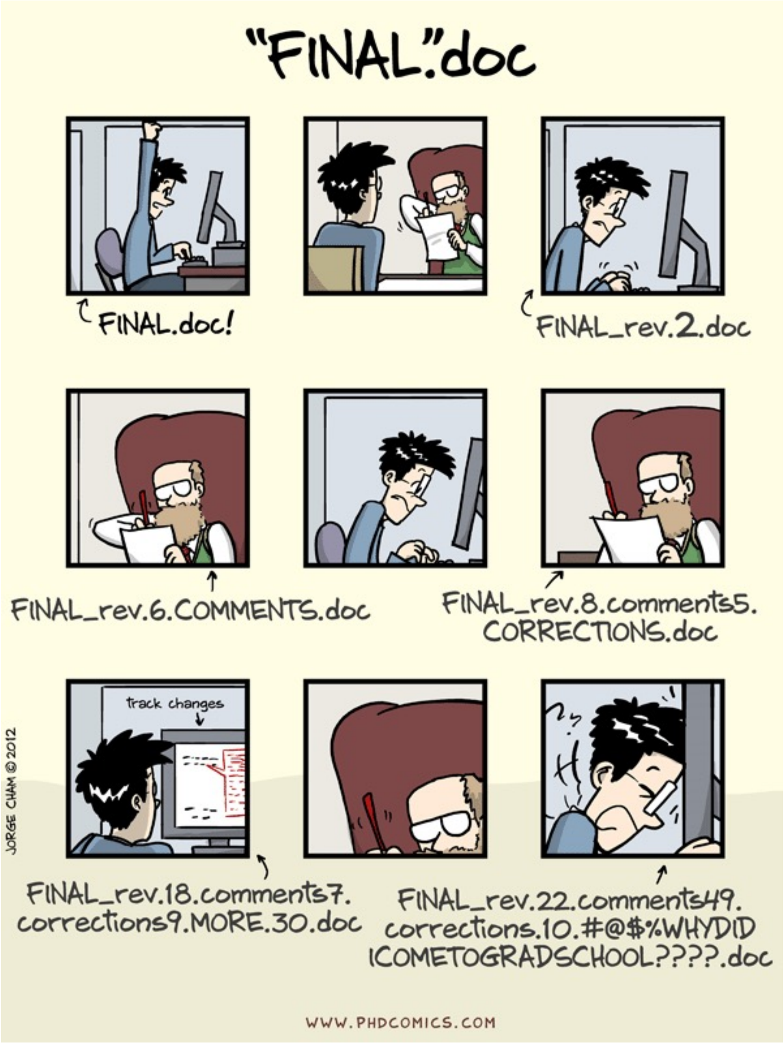
```
wget https://osf.io/z274d/download -O gapminder_copy.txt
```

(You may have to install wget – Google it)

```
My_project
  |--data
  |     |--raw
  |     |--derived
  |     |--results
  |--docs
  |--notebooks
  |--references
  |--reports
  |--src
```

Principle 2 – Control the source & changes

2. For software, use a version control system to save versions and changes, and explain the reason for the changes
   - Git is the standard these days
   - Basics
     - Software is stored somewhere (e.g., GitHub, GitLab), either privately or publicly
     - New versions can be added
       - Author, changes, message about change stored
     - Multiple people can make changes in different parts of a project or even a file, and these can be merged together, mostly automatically
   - See Software Carpentry's "Version Control with Git"
   - Version numbers
     - Consider releases, use semantic versioning
       - A release is a tagged version
       - major.minor.patch (API-breaking.API-maintaining.bug-fixes)



"Piled Higher and Deeper" by Jorge Cham
www.phdcomics.com

Principle 2 – Control the source & changes

3. For published documents, people, etc. find a permanent identifier (PID, e.g., DOI, PubMed ID, ORCID) and use it to find the details (e.g., for references)
   - Get data from ORCID for a person (in Python):

     ```
     import requests
     import json
     print(requests.get('https://pub.orcid.org/v3.0/0000-0001-5934-7525',
               headers={'content-type':'application/json'}).json())
     ```

   - Get metadata about a paper from a DOI (in bash):

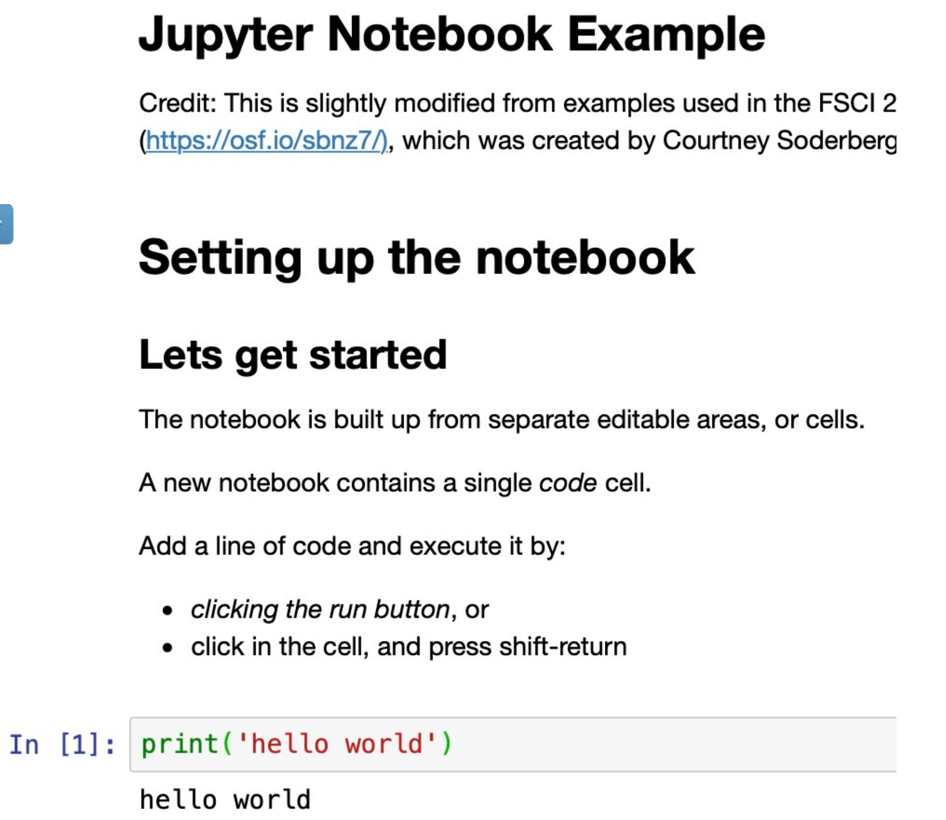
     ```
     curl https://api.crossref.org/works/10.1145/3307681.3325400/transform/application/vnd.crossref.unixsd+xml
     ```

   - Get bibtex for a paper from a DOI (in bash):

     ```
     curl -LH "Accept: application/x-bibtex" https://doi.org/10.1145/3307681.3325400
     curl https://data.crosscite.org/application/x-bibtex/10.1145/3307681.3325400
     ```

Principle 3 – Use notebooks to explain & document

- Notebooks are great for showing what code does
- And teaching people how to use it
- Intersperse cells with text, equations, runnable code, outputs, images
  - Demo: go to https://github.com/danielskatz/repro-fdtd1d, click on launch binder
  - Once binder starts the repo, click on `Notebook_Demonstration.ipynb`
- This uses binder (mybinder.org) – you can too
- Turn a Git repo into a collection of interactive notebooks, making your code immediately reproducible by anyone, anywhere
  - Use `requirements.txt` to tell binder what dependencies to install in the environment
  - Also take a look at binderhub and jupyterhub if you want to run your own instance
- But don't write code to do the same task in multiple notebooks
  - Pull it out (refactor it) into a (reusable) package, then import that package in the notebooks



Jupyter Notebook Example

Credit: This is slightly modified from examples used in the FSCI 2
(https://osf.io/sbnz7/), which was created by Courtney Soderberg

Setting up the notebook

Lets get started

The notebook is built up from separate editable areas, or cells.

A new notebook contains a single *code* cell.

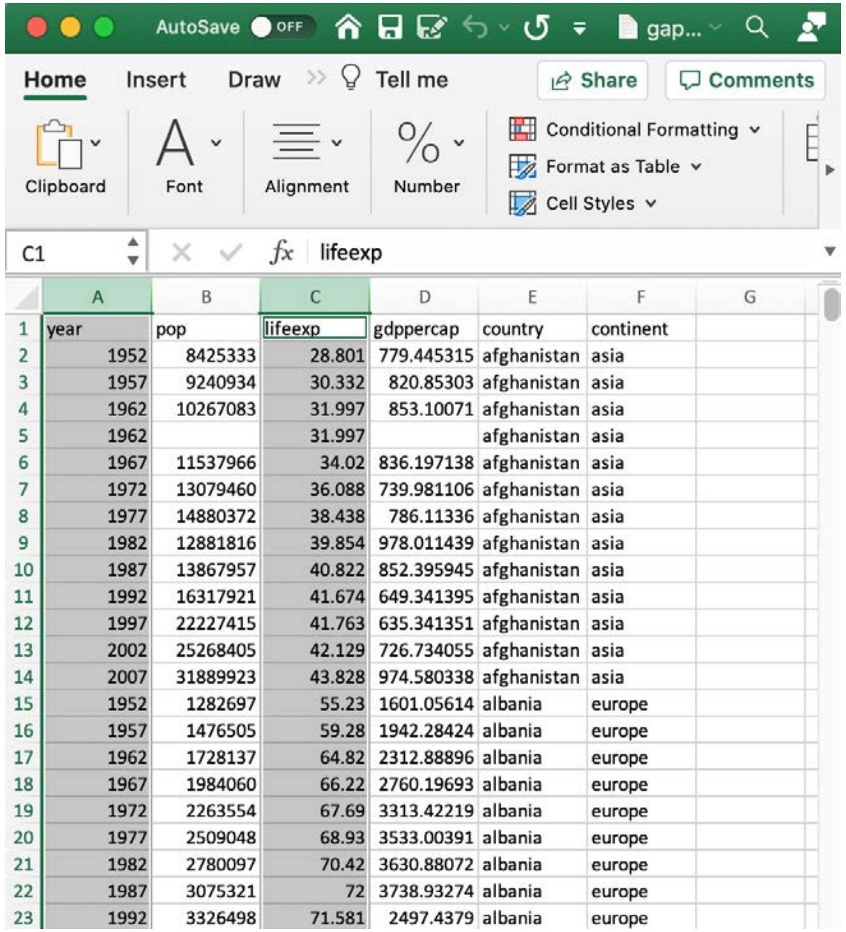
Add a line of code and execute it by:

- *clicking the run button*, or
- click in the cell, and press shift-return

In [1]:
```
print('hello world')
```
```
hello world
```

Principle 4 – Automate steps

1. Anything you do by hand is subject to irreproducible errors
   – GUIs can be intuitive, but they don't support scalability or reproducibility well
      - Imagine having to extract a column of data from 1000 Excel files
   – Goal: capture what you do in some way so that you can repeat it in one step, such as
      - Capture a set of commands in a single script
      - Use pyexcel to read and write data to/from Excel files
      - Script GUI actions – see AutoHotkey_L for an example of how this can be done for Windows programs



	A	B	C	D	E	F	G
1	year	pop	lifeexp	gdppercap	country	continent	
2	1952	8425333	28.801	779.445315	afghanistan	asia	
3	1957	9240934	30.332	820.85303	afghanistan	asia	
4	1962	10267083	31.997	853.10071	afghanistan	asia	
5	1962		31.997		afghanistan	asia	
6	1967	11537966	34.02	836.197138	afghanistan	asia	
7	1972	13079460	36.088	739.981106	afghanistan	asia	
8	1977	14880372	38.438	786.11336	afghanistan	asia	
9	1982	12881816	39.854	978.011439	afghanistan	asia	
10	1987	13867957	40.822	852.395945	afghanistan	asia	
11	1992	16317921	41.674	649.341395	afghanistan	asia	
12	1997	22227415	41.763	635.341351	afghanistan	asia	
13	2002	25268405	42.129	726.734055	afghanistan	asia	
14	2007	31889923	43.828	974.580338	afghanistan	asia	
15	1952	1282697	55.23	1601.05614	albania	europe	
16	1957	1476505	59.28	1942.28424	albania	europe	
17	1962	1728137	64.82	2312.88896	albania	europe	
18	1967	1984060	66.22	2760.19693	albania	europe	
19	1972	2263554	67.69	3313.42219	albania	europe	
20	1977	2509048	68.93	3533.00391	albania	europe	
21	1982	2780097	70.42	3630.88072	albania	europe	
22	1987	3075321	72	3738.93274	albania	europe	
23	1992	3326498	71.581	2497.4379	albania	europe	

Principle 4 – Automate steps

2. Scripts for simple things (steps)
   – Shell scripts
     - See Software Carpentry lesson "The Unix Shell"
       – "The Unix shell has been around longer than most of its users have been alive. It has survived so long because it's a power tool that allows people to do complex things with just a few keystrokes. More importantly, it helps them combine existing programs in new ways and automate repetitive tasks so they aren't typing the same things over and over again."
     - At the simplest, the shell is the process you interact with when you type in a terminal window
     - Multiple commands can be placed in a script and rerun
     - And the shell supports variables and control flow (e.g. if-then, loops)

Principle 4 – Automate steps

- Go to https://github.com/danielskatz/repro-fdtd1d, click on launch binder
- Once binder starts the repo, use "New" -> "Terminal" to get a terminal

```
mkdir raw
mkdir proc
```

Make directories

Get raw input files (into raw directory)

```
wget https://raw.githubusercontent.com/danielskatz/parsl-example/master/data/0001.jpg -O raw/0001.jpg
wget https://raw.githubusercontent.com/danielskatz/parsl-example/master/data/0002.jpg -O raw/0002.jpg
wget https://raw.githubusercontent.com/danielskatz/parsl-example/master/data/0003.jpg -O raw/0003.jpg
wget https://raw.githubusercontent.com/danielskatz/parsl-example/master/data/0004.jpg -O raw/0004.jpg
```

```
python3 bin/sharpen_image.py raw/0001.jpg proc/0001_sharp.jpg
python3 bin/sharpen_image.py raw/0002.jpg proc/0002_sharp.jpg
python3 bin/sharpen_image.py raw/0003.jpg proc/0003_sharp.jpg
python3 bin/sharpen_image.py raw/0004.jpg proc/0004_sharp.jpg
```

Process raw input files

```
python3 bin/local_build_mosaic.py 2 proc/mosaic.jpg proc/0001_sharp.jpg proc/0002_sharp.jpg proc/0003_sharp.jpg
proc/0004_sharp.jpg
```

Further process processed files

Principle 4 – Automate steps

- Go to https://github.com/danielskatz/repro-fdtd1d, click on launch binder
- Once binder starts the repo, use “New” -> “Terminal” to get a terminal

- Automate by:

```
sh script/build_mosaic.sh
```

Contains all the commands from the previous slide

Principle 4 – Automate steps

3. Can use notebooks like scripts/programs with tools such as
   - nbclient, a very lightweight python API for executing notebooks
   - Papermill, a tool for parameterizing and executing Jupyter Notebooks
   - Jupytext, a converter between notebooks and code and vice versa

4. An interesting-looking new project
   - nbmake-action - A Notebook-First Continuous Integration Framework
     - A GitHub Action for testing notebooks, runs them from top-to-bottom
     - Intended to raise the quality of scientific material through better automation
     - For scientists/developers who have written docs in notebooks and want to CI test them after every commit

Principle 4 – Automate steps

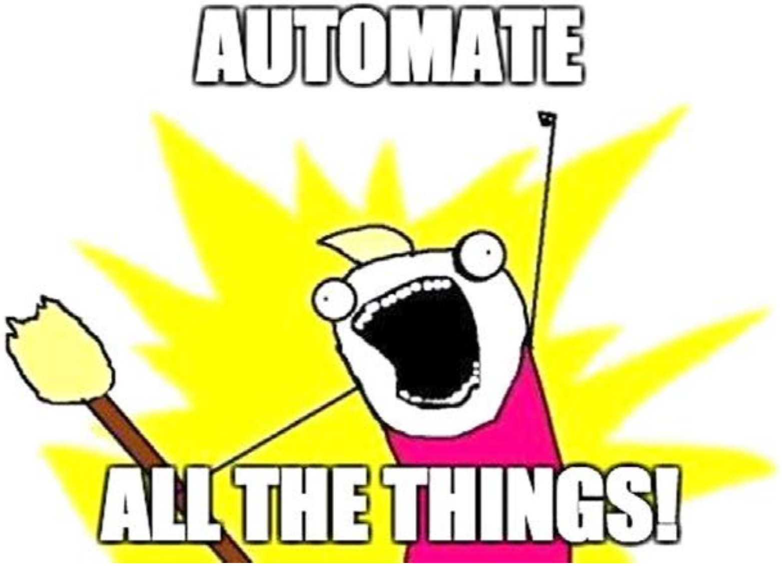
4. Make randomness repeatable
   - Many simulations and data analysis involve random seeds, used to start generating a series of "random" numbers
     - Capture these seeds as part of your step so that you can repeat the same "randomness"
     - And get the "same" results
   - But be aware of tradeoffs
     - Example
       - When adding a list of floating point numbers, order can matter due to numerical roundoff
       - When using parallel computing, order can change with the same or different numbers of processes
       - Can force order at the cost of performance (extra sync/lock/messages)
       - Better to know what accuracy counts
       - Or to have a debug mode and a production mode

ILLINOIS NCSA

Principle 5 – Automate everything

GNU Make

- Make or something make-like to handle multiple steps (dependencies) and not redo what isn't needed
  - In the previous example, what happens if we just change the final program?
  - We don't really want to have to rerun the whole script
  - Learn about make (GNU make, gmake) from Software Carpentry's lesson
  - Short version
    - A program that defines rules for how to make one thing from others (dependencies)
    - Can use variables to make rules general
    - Make knows how to only make a thing when its dependencies have changes
- Other options: workflow (management) systems & languages, e.g., in bioinformatics, snakemake, cwl, wdl, nextflow, … (there's a CWL wiki page with 298 examples)
- Consider continuous integration to automatically rebuild/test when things change
  - Integrate with GitHub Actions, CircleCI, Travis CI, etc.



Principle 5 – Automate everything

```
.PHONY: clean all
all: proc/mosaic.jpg

clean:
	-rm -rf raw proc

raw:
	mkdir raw

proc:
	mkdir proc

raw/0001.jpg: | raw
	wget https://raw.githubusercontent.com/danielskatz/parsl-example/master/data/0001.jpg -O raw/0001.jpg

[…]

raw/0004.jpg: | raw
	wget https://raw.githubusercontent.com/danielskatz/parsl-example/master/data/0004.jpg -O raw/0004.jpg

proc/0001_sharp.jpg: raw/0001.jpg bin/sharpen_image.py | proc
	python3 bin/sharpen_image.py raw/0001.jpg proc/0001_sharp.jpg

[…]

proc/0004_sharp.jpg: raw/0004.jpg bin/sharpen_image.py | proc
	python3 bin/sharpen_image.py raw/0004.jpg proc/0004_sharp.jpg

proc/mosaic.jpg: bin/local_build_mosaic.py proc/0001_sharp.jpg proc/0002_sharp.jpg proc/0003_sharp.jpg proc/0004_sharp.jpg
	python3 bin/local_build_mosaic.py 2 proc/mosaic.jpg proc/0001_sharp.jpg proc/0002_sharp.jpg proc/0003_sharp.jpg proc/0004_sharp.jpg
```

This is in script/Makefile-explicit

To run it, from the terminal in binderhub, use:
make -f script/Makefile-explicit

Principle 5 – Automate everything

```
LANGUAGE=python3
FILE_NOS=0001 0002 0003 0004
RAW_FILES=$(FILE_NOS:%=raw/%.jpg)
PROC_FILES=$(FILE_NOS:%=proc/%_sharp.jpg)
SHARPEN = bin/sharpen_image.py
MOSAIC = bin/local_build_mosaic.py
RAW_SOURCE_DIR=https://raw.githubusercontent.com/danielskatz/parsl-example/master/data

.PHONY: clean all
all: proc/mosaic.jpg

clean:
	-rm -rf raw proc

raw:
	mkdir raw

proc:
	mkdir proc

$(RAW_FILES): | raw
	wget $(@:raw/%.jpg=$(RAW_SOURCE_DIR)/%.jpg) -O $@

proc/%_sharp.jpg: raw/%.jpg $(SHARPEN) | proc
	$(LANGUAGE) $(SHARPEN) $(@:proc/%_sharp.jpg=raw/%.jpg) $@

proc/mosaic.jpg: $(MOSAIC) $(PROC_FILES)
	$(LANGUAGE) $(MOSAIC) 2 $@ $(PROC_FILES)
```

And make the automation as general as possible

This is in script/Makefile

To run it, from the terminal in binderhub, use:
make -f script/Makefile

Principle 6 – Capture the environment

- Containers
  - Use docker to ensure the exact same software environment everywhere; lightweight & practical
    - For HPC, will likely need to use singularity or shifter instead
  - To specify an environment
    - In Python, use virtualenv (and `pip freeze > requirements.txt`) or pipenv or conda
    - In R, use add_dependencies_to_description() or use renv package or rocker
- VMs (heavier weight than containers, includes OS)
- Reproducible builds - a set of software development practices that create an independently-verifiable path from source to binary code
  - Reliant on package identification and management, e.g., Guix, PyPI, CRAN, …
- Lots of tools and systems – see "Publishing computational research - a review of infrastructures for reproducible and transparent scholarly communication" for a 2020 survey of 11

Principle 7 – Provide a license & make citable

- Copyright defines ownership, license gives permission to do something
- But facts aren't copyrightable, while works of authorship are (at least in the US)
  - A particular arrangement of facts might be eligible for copyright protection if that arrangement demonstrates sufficient creativity, but not if the arrangement is something uncreative like chronological or alphabetical order
  - Even with creative arrangement, underlying facts cannot be copyrighted; it's perfectly legal for someone else to pull them out, rearrange them, and use them in something new
  - See "Who 'owns' your data?"
- If you are employed, your employer may own the copyright to things you create at work, and maybe even outside
  - Common in the US and in universities, but students own work they develop in their own coursework (though not if they are paid to do it, such as in a research assistantship)
- Use a common license, don't create your own
  - Common licenses are understood, uncommon one will prevent people from using your work just because they may not understand the license

Principle 7 – Provide a license & make citable

- Creative Common licenses for text and data
  - CC0 – waive copyright, dedicate to the public domain (not really a license)
  - CC BY (Attribution): material is free to use and adapt, but credit must be given
  - CC BY-SA (Attribution-ShareAlike): free to use and adapt, but credit must be given and adapted material must also be distributed with this same license
  - CC BY-ND (Attribution-NoDerivs): free to use, but credit must be given and can't be adapted
  - CC BY-NC (Attribution-NonCommercial): free to use and adapt but credit must be given and can't be used commercially
  - CC BY-NC-SA (Attribution-NonCommercial-ShareAlike): free to use and adapt, but credit must be given, can't be used commercially, and adapted material must also be distributed with this same license
  - CC BY-NC-ND (Attribution-NonCommercial-NoDerivs): free to use, but credit must be given, can't be used commercially, and can't be adapted
- Creative Commons provides a guide/decision tree
- Be aware someone might argue that the data are facts and not subject to copyright, so the license doesn't hold
- Scholarly norms and principles of attribution/credit/provenance/authority might hold more sway
- (for more, see "CC BY and data: Not always a good fit")

Principle 7 – Provide a license & make citable

- Open Source Initiative licenses for software
  - Don’t use a CC license for software
  - At high level, two types of licenses
  - Permissive: MIT, Apache, BSD, …
  - Copyleft (“viral”): GPL, LGPL
- Use choosealicense.com to pick one
- Pick a very common one if possible
- How to apply (MIT):
  - Create a text file (typically named LICENSE or LICENSE.txt) in the root of your source code and copy the text of the license into the file. Replace [year] with the current year and [fullname] with the name (or names) of the copyright holders.

Principle 7 – Provide a license & make citable

- Citeable isn't required for reproducibility, but it's a good idea if you want credit
- Make your data citable
  - Deposit it in an archival repository (e.g., Zenodo, OSF, see re3data.org for more) along with metadata, receive a DOI, advertise the DOI and metadata (suggested citation)
- Make your software citable
  - Less well-defined practice
  - GitHub is not an archival repository
  - Can follow data practice (can link GitHub repo to Zenodo to automatically deposit new releases - guides.github.com/activities/citable-code)
  - Record metadata in the repository (using CodeMeta or citation.cff), some repositories will pick up
  - Also can use Software Heritage ("archive.org for software") to cite archive of GitHub software
  - See cite.research-software.org for more

Exercise(s)

- Try out one of the project structure tools, or look at them and try to organize a project you have similarly
  - Python: Cookiecutter Data Science
  - R: ProjectTemplate
- Redo the exercise from the beginning in a more reproducible manner
- Automate a paper you have written
  - Or try to do this for a paper someone else has written (start by finding the data and code, see how far you can get)

Final thoughts

- “I was inspired more than 15 years ago by John Claerbout […] He pointed out to me, in a way paraphrased in Buckheit and Donoho (1995): ‘an article about computational result is advertising, not scholarship. The actual scholarship is the full software environment, code and data, that produced the result.’” - David Donoho (in https://doi.org/10.1093/biostatistics/kxq028)

- “You shouldn’t try to do these things all at once; start with one, or part of one. Then in your next project, do that plus another thing.” – Karl Broman (in https://kbroman.org/steps2rr/)

- It's no secret that good analyses are often the result of very scattershot and serendipitous explorations. […] That being said, once started it is not a process that lends itself to thinking carefully about the structure of your code or project layout, so it's best to start with a clean, logical structure and stick to it throughout. (in https://drivendata.github.io/cookiecutter-data-science/)

Resources (1)

- Organizing projects:
  - Python: Cookiecutter Data Science - https://drivendata.github.io/cookiecutter-data-science
  - R: ProjectTemplate - http://projecttemplate.net/
- Guidelines:
  - Karl Broman's initial steps toward reproducible research (R, explains python too) - https://kbroman.org/steps2rr/
- Reproducible papers:
  - PINGA lab's template (computational science, GitHub, Python, LaTeX) - https://www.leouieda.com/blog/paper-template.html
  - Manubot (markdown, git, collaboration) - https://manubot.org
  - Akhaghi (C/C++, LaTeX) - https://gitlab.com/makhlaghi/reproducible-paper
- Book:
  - The Practice of Reproducible Research: Case Studies and Lessons from the Data-Intensive Sciences - http://www.practicereproducibleresearch.org/

Resources (2)

- Short courses/MOOCs:
  - Essential skills for reproducible research computing - https://barbagroup.github.io/essential_skills_RRC/
  - Reproducible Research using Jupyter Notebooks - https://reproducible-science-curriculum.github.io/workshop-RR-Jupyter/
  - Duke UPGG Informatics Orientation Bootcamp - https://duke-gcb.github.io/2019-08-12-Duke/
  - Reproducible Research and Data Analysis (under development) - https://opensciencemooc.eu/modules/reproducible-research-and-data-analysis/
  - Reproducible research: Methodological principles for a transparent science - https://learninglab.inria.fr/en/mooc-recherche-reproductible-principes-methodologiques-pour-une-science-transparente/
  - Make (Software Carpentry's lesson) - http://swcarpentry.github.io/make-novice/

Resources (3)

- Tools:
  - Popper - https://github.com/getpopper/popper
  - Reana - http://www.reanahub.io
  - ReproZip - https://www.reprozip.org
  - Sciunit - https://sciunit.run
- Other:
  - Reproducible PI Manifesto - https://lorenabarba.com/gallery/reproducibility-pi-manifesto/
  - Container information - https://slurm.schedmd.com/containers.html
  - Python virtual environments: https://towardsdatascience.com/python-virtual-environments-made-easy-fe0c603fe601 and https://towardsdatascience.com/comparing-python-virtual-environment-tools-9a6543643a44
  - Computational science example (from FSCI 2018 & 2019): https://github.com/danielskatz/repro-fdtd1d
  - Make your code ready for publication (sharable and citable) workshop - https://gitlab.com/hifis/hifis-workshops/make-your-code-ready-for-publication/workshop-materials
  - Software Citation Principles - https://doi.org/10.7717/peerj-cs.86