Krembil Centre for Neuroinformatics

Using big data, artificial intelligence and brain modelling to fundamentally change our understanding of mental illness.

SUMMER SCHOOL 2020
Day 1
Understanding clinical research questions and reproducible science

Afternoon: Best practices and tools for reproducible science
Today’s Agenda

9:00 am - 10:30 am  
**Lecture 1: Welcome and Orientation + Neuroinformatics Across Scales**  
Erin Dickie + Sean Hill

10:45 am - 12:15 pm  
**Problems and opportunities in the diagnosis and treatment of major depression**  
Dr Victor Tang & Dr Brett Jones

1:00 pm - 2:30 pm  
**Workshop 1: Guiding principles for FAIR and open science**  
Erin Dickie & Sejal Patel

2:45 pm - 4:15 pm  
**Workshop 2: Tools for Reproducible Science**  
Erin Dickie & Sejal Patel

Day 1:  
Welcome!  
Understanding clinical research questions and reproducible science
This afternoon

Guiding principles and tools for reproducible science

Erin Dickie Ph.D.
Lead - Education and Knowledge Transfer
Krembil Centre for Neuroinformatics, Centre for Addiction and Mental Health, Toronto, Ontario
Twitter: @ErinWDickie  Github: @edickie

Sejal Patel Ph.D.
Post-doctoral Fellow - Whole Person Modeling
Krembil Centre for Neuroinformatics, Centre for Addiction and Mental Health, Toronto, Ontario
Github: @Sejal24
Teaching Assistants for this section

Kevin Kada
Whole Brain Modeling Lab
KCNI - CAMH

Taha Morshedzadeh
Whole Brain Modeling Lab
KCNI - CAMH

Kevin Witczak
CAMH Kimel TIGRlab
Github: @kimjetwav
Remember - many ways to engage

- **crowdcast**
  - (during sessions)
  - Use the chat or the ask question!

- **slack**
  - come chat with us in KCNI Summer School Slack :)

- **Gather**
  - virtually meet with us in gather.town

Tell us how the session went (post session survey):
https://forms.gle/ji18qLMZEZ9L16Ln6

KCNISchool@camh.ca
Outline for this afternoon

Why reproducible science?

The things you need to know to get through this course:
- Versioning and publishing code (github)
- Versioning and publishing software (docker)
- R with Rmarkdown (walk through)
- Python in ipython notebooks (walk through)
  - also google colab

The fancy bits:
- building your own binder environments
- building your own containers (docker & singularity)
### What is reproducible science?

<table>
<thead>
<tr>
<th>Analysis</th>
<th>Data</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Same</td>
<td>Same</td>
<td>Reproducible</td>
<td>Replicable</td>
</tr>
<tr>
<td>Different</td>
<td>Different</td>
<td>Robust</td>
<td>Generalisable</td>
</tr>
</tbody>
</table>

*Fig. 5 How the Turing Way defines reproducible research*
An article about computational science in a scientific publication is not the scholarship itself, it is merely advertising of the scholarship. The actual scholarship is the complete software development environment and the complete set of instructions which generated the figures.

Buckheit and Donoho
WaveLab and Reproducible Research, 1995
Reproducible Science - Why

For Yourself

For your colleagues

For science
Your number one collaborator is yourself six months ago: And they don’t answer emails
Think of reproducible science as “tidy” code and data. By keeping things tidy, labeled and organized, you create a space where you can invite guests.

Reproducible science leads to more meaningful collaborations where collaborators can review your code, learn from it, and build from your work.

Communicating to your peers

**Problem:** today’s scientific papers can fail to communicate all details of the methods needed to reproduce the study.
Problems and Suggested Solutions

<table>
<thead>
<tr>
<th>Problems</th>
<th>Suggested Solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low statistical power</td>
<td>(one or many) increasing the amount of available data for analysis</td>
</tr>
<tr>
<td>Flexibility and exploration in data analysis</td>
<td>pre-registration of methods and analysis plans</td>
</tr>
<tr>
<td>Multiple comparisons</td>
<td>sharing of both thresholded and un-thresholded result maps (for meta analysis)</td>
</tr>
<tr>
<td>Software errors</td>
<td>avoid the trap of the ‘not invented here’ philosophy: when the problem at hand can</td>
</tr>
<tr>
<td>“As complexity of a software program increases, the</td>
<td>be solved using software tools from a well-established project, these should be</td>
</tr>
<tr>
<td>likelihood of undiscovered bugs quickly reaches</td>
<td>chosen instead of re-implementing the same method in custom code</td>
</tr>
<tr>
<td>certainty”</td>
<td></td>
</tr>
<tr>
<td>Insufficient study reporting</td>
<td>Because the computer code is often necessary to understand exactly how a data set</td>
</tr>
<tr>
<td></td>
<td>has been analysed, releasing the analysis code is particularly useful and should be</td>
</tr>
<tr>
<td></td>
<td>standard practice.</td>
</tr>
</tbody>
</table>
Towards the [neuroimaging] paper of the future

“All code for data collection and analysis would be stored in a version-control system and would include software tests to detect common problems”

“The repository would use a continuous integration system to ensure that each revision of the code passes appropriate software tests.”

“The entire analysis workflow (including both successful and failed analyses) would be completely automated in a workflow engine and packaged in a software container or virtual machine to ensure computational reproducibility.”
The many bits of a project

Fig. 3 The Turing Way project illustration by Scriberia. Used under a CC-BY 4.0 licence. DOI: 10.5281/zenodo.3332807.
FAIR data principles

Findable: The first step in (re)using data is to find them! Descriptive metadata (information about the data such as keywords) are essential.

Accessible: Once the user finds the data and software they need to know how to access it. Data could be openly available but it is also possible that authentication and authorisation procedures are necessary.

Interoperable: Data needs to be integrated with other data and interoperate with applications or workflows.

Reusable: Data should be well-described so that they can be used, combined, and extended in different settings.

Fig. 31 The Turing Way project illustration by Scriberia. Used under a CC-BY 4.0 licence. DOI: 10.5281/zenodo.3332807.
Data management recommendations

0. Organise your project as if you have no memory

1. Raw data is read-only. (i.e. never modify your source data)

2. Name files for both humans and computers to understand

3. Always know where your data comes from

4. Track all the changes the humans or computers make. Use version control.
Here are some tips for naming files within a research project, which are both human- and machine-readable [Cow20][Hod15]:

- Name your files consistently
- Keep it short but descriptive
- Avoid special characters or spaces to keep it machine-compatible
- Use capitals or underscores to keep it human-readable
- Use consistent date formatting, for example ISO 8601: YYYY-MM-DD to maintain default order
- Include a version number when applicable
- Share/establish a naming convention when working with collaborators
- Record a naming convention in your data management plan

[the turing way - https://the-turing-way.netlify.app/project-design/filenaming.html]
Beware the WOMBAT

Waste
Of
Money
Brains
And
Time

Do not reinvent the wheel

If a (well-maintain) tool or pipeline exists for your analysis - use it
  - don’t recode your own

If a data naming standard exists for your data type
  - use it
  - don’t invent you own

- Michael Hanke @eknahm, BrainHack Leipzig 2012
If it exists - use a data standard!

International Neuroinformatics Coordinating Facility (INCF) hosts a curated list of data standards that are useful for our research.

- [https://www.incf.org/resources/sbps](https://www.incf.org/resources/sbps)
- Brain Imaging Data Structure (BIDS) for neuroimaging and EEG
- neurodata without borders (NWB) for neurophysiology
- NeuroML for models
The impact of open data

Open science is a cultural change

These practices and tools can be difficult to incorporate into a full project with multiple stages of analysis...but they are especially useful in teaching.

The materials in this course are examples of reproducible science tools.
Outline for this afternoon

Why reproducible science?

The things you need to know to get through this course:

  - Versioning and publishing code (github)
  - Versioning and publishing software (docker)
  - R with Rmarkdown (walk through)
  - Python in ipython notebooks (walk through)
    - also google colab

The fancy bits:

  - building your own binder environments
  - building your own containers (docker & singularity)
Using version control

**version control:** (‘git’) is a tool for tracking what changes to a folder (usually a folder filled with code) when and by who...

- Like MS Word’s “track changes”
  - ...but for code
  - ..and on steroids..
GitHub is a website where everyone shares their code with themselves, their teams and with the world.

It has a lot of useful features for:

- working with teams
- reading other people’s code
- integrating with other platforms
  - continuous integration (CI) to test for bugs
  - Dockerhub
- hosting documentation websites and wiki’s
- releasing versions
to post questions for help cloning to local computer

scroll down to the readme for relevant info (course schedule)

to download the complete (old) 2020 school code
Version Control (git)

- **Your files**
- **Your local git repo**
- **Github**

### Commands

- **git status**
  List what has not been committed (repo)

- **git diff**
  Tells you what changes haven’t been committed (file)

- **git init**
  Starts a new repo

- **git clone**
  Copies a repo from GitHub or GitLab to your local computer
Git for your own project demo
Step 1: build repo on github/gitlab

Step 2:
- cd ~/code
- git clone http:link/to/git/repo.git

<table>
<thead>
<tr>
<th>Time Event</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>First thing in the morning</td>
<td>git pull</td>
</tr>
<tr>
<td>Before coffee break</td>
<td>git commit bin/scriptx.py -m “I changed x”</td>
</tr>
<tr>
<td>Before going to lunch</td>
<td>git commit bin/scripty.py -m “modifying y”</td>
</tr>
<tr>
<td>Done adding function foo</td>
<td>git commit bin/sciprtx.py -m “now does foo”</td>
</tr>
<tr>
<td>Before going home</td>
<td>git push</td>
</tr>
</tbody>
</table>
"Git, the toothbrush of science"
Git with friends..

Let go look at the lesson’s repo on github

- when working with larger groups - it’s usually better to put in a little more work to make sure that your changes don’t clash with other people’s changes
- some projects have “CONTRIBUTING.md” files that lay down some best practice steps

- “fork” the repo
  - make a copy under your own user space
- create a new “branch” that specific to you change
- make your changes inside the “branch”
- when you are done make a “pull request” back to the “upstream” repo
Git submodules - what? and why?

**What?** - a git **“submodule”** is link from one git repository to another repository

- *think of the kcni-school-lessons repo as an index of lesson repos*
- *on github you see the “submodule” as a hyperlink to another git repo* at a specific commit
- *on your computer - a submodule looks like a git repo inside another git repo.*

**Why?** - it can allow you to keep project more modular

- meaning you can use it to link to a common code base that is shared across multiple projects
- it can help git run faster (for very large projects)
Get the tutorial scripts (with all the submodules) by typing

```bash
git clone --recurse-submodules \  https://github.com/krembilneuroinformatics/kcni-school-lessons.git
```

Get the updated tutorial scripts (with all the submodules) after cloning

```bash
cd kcni-school-lessons
git pull --recurse-submodules
```
But what …
you might find, when navigating the kcni-school-lesson that the submodules folders are empty…this is because the submodule needs to be “updated”

cd kcni-school-lessons
cd day1
git submodule init
git submodule update example-python-repo
Git submodules - to create your own

A good tutorial for everything:


For example here’s some code to add SPM12 toolbox code into kcni-school-lesson’s day 6 code.

Get add a submodule to your repo

```
cd kcni-school-lessons/day6/toolboxes
git submodule add https://github.com/spm/spm12.git
```

Get updated the submodule content to the newest commit

```
cd kcni-school-lessons
git submodule update --remote spm12
```
Outline for this afternoon

Why reproducible science?
The things you need to know to get through this course:
  Versioning and publishing code (github)
  Versioning and publishing software (docker)
  R with Rmarkdown (walk through)
  Python in ipython notebooks (walk through)
    -also google colab
The fancy bits:
  building your own binder environments
  building your own containers (docker & singularity)
Containers - Why

Every box here is something that could be installed differently (or not at all) by the next user.

This will cause my_fancy_day1_script.R to:
1) crash/not work at all
2) produce unexpected/different results
3) maybe still work?
Containers - what and why?

Docker - is a tool for sharing software + the dependencies
- the install instructions are stored script called “Dockerfile”
- it’s like a virtual machine
  - without a display
  - that takes up a little less disk space
  - that can be installed in one line

Some Docker vocabulary
- **image**: your install of the software
- **container**: one instance of that software that is *usually* still running.

Dockerhub is a website that hosts docker images. So that anyone - anywhere is the world can run it!
Containers for scaling up analysis

Develop and test on your local computer

Deploy software on a high performance cluster
Containers for medical science

There is some speculation that (secure) cloud computing will become important of health research.

Deploy in the cloud

Develop and test on your local computer
How to run our code

Run on your Local Computer using Docker
- will use less internet bandwidth while you watch the stream
- you will have a copy of the files locally
- requires installation of Docker Desktop

Run on the web with Binder
- no local installation needed
- may take some time to boot up
- limited resources for the computer

Run on SciNet using a guest account
- no local installation needed
- a few extra set-up steps needed
- good compute resources
To follow along

To interact with the school lesson code on your local computer you will need:

- A terminal for pulling/cloning data from github.
  - mac os terminal or WSL work

- On windows - gitbash is an option

https://www.docker.com/products/docker-desktop
- installs on window, mac or linux

Also useful, but not necessary:
Step 1: Install Docker Desktop

Installing Docker *should* not be harder than installing any other program on your computer. Download link and install instructions at: https://www.docker.com/products/docker-desktop.

To check your install open up a terminal (in windows this is Powershell or WSL) and type:

docker run hello-world
Docker Desktop install gotcha’s

1. On Windows - you need to enable Hyper-V or WSL virtualization
2. You also need “Share the drive” with docker.
   a. Settings->Resources->File Sharing
If you can’t install Docker on your local computer (because you probably don’t have enough administrative rights - or you don’t have enough space on you home computer) We have a plan!

You can run the software on the binder instance or SciNet teach cluster!

Instructions are available at: https://github.com/edickie/kcni-school-lessons
Step 2: download and run the KCNIschool docker

In the same terminal window where you typed “docker pull hello-world” now type:

```
git clone --recurse-submodules \  
  https://github.com/krembilneuroinformatics/kcni-school-lessons.git
cd kcni-school-lessons
docker compose up rstudio
```

Then you should see lots of things happening! What is happening? -

*docker is downloading ~ 5G of software for our lessons into an “image”*

Copy and paste this line from:
https://github.com/edickie/kcni-school-lessons/tree/master/envs/README.md
Step 3: open rstudio in browser

After typing

```
cd kcni-school-lessons
docker compose up rstudio
```

You will finally see the message 

```
[services.d] done.
```

point you browser to: 

http://localhost:8787/
Outline for this afternoon

Why reproducible science?

The things you need to know to get through this course:

- Versioning and publishing code (github)
- Versioning and publishing software (docker)
- R with Rmarkdown (walk through)
- Python in ipython notebooks (walk through)
  - also google colab

The fancy bits:

- building your own binder environments
- building your own containers (docker & singularity)
R Programming

R is a language and environment for statistical computing and graphics for data visualization

- Similar to S programming language and environment
R GUI and RStudio

Old way of coding in R

New way of coding R
Tidyverse (dplyr package) vs Base R

- Here are few examples between base R function and the equivalent functions in the dplyr package found within tidyverse

<table>
<thead>
<tr>
<th>dplyr</th>
<th>base</th>
</tr>
</thead>
<tbody>
<tr>
<td>arrange(df, x)</td>
<td>df[order(x), , drop = FALSE]</td>
</tr>
<tr>
<td>distinct(df, x)</td>
<td>df[!duplicated(x), , drop = FALSE], unique()</td>
</tr>
<tr>
<td>filter(df, x)</td>
<td>df[which(x), , drop = FALSE], subset()</td>
</tr>
<tr>
<td>mutate(df, z = x + y)</td>
<td>df$z &lt;- df$x + df$y, transform()</td>
</tr>
<tr>
<td>pull(df, 1)</td>
<td>df[[1]]</td>
</tr>
<tr>
<td>pull(df, x)</td>
<td>df$x</td>
</tr>
<tr>
<td>rename(df, y = x)</td>
<td>names(df)[names(df) == &quot;x&quot;] &lt;- &quot;y&quot;</td>
</tr>
<tr>
<td>relocate(df, y)</td>
<td>df[union(&quot;y&quot;, names(df))]</td>
</tr>
<tr>
<td>select(df, x, y)</td>
<td>df[c(&quot;x&quot;, &quot;y&quot;), subset()</td>
</tr>
<tr>
<td>select(df, starts_with(&quot;x&quot;))</td>
<td>df[grepl(names(df), &quot;^x&quot;)]]</td>
</tr>
<tr>
<td>summarise(df, mean(x))</td>
<td>mean(df$x), tapply(), aggregate(), by()</td>
</tr>
<tr>
<td>slice(df, c(1, 2, 5))</td>
<td>df[c(1, 2, 5), , drop = FALSE]</td>
</tr>
</tbody>
</table>

- Note: through the different workshops you will encounter the use of both tidyverse and base R coding style
Long debate in the R community of which way of R coding is better

- Generally Tidyverse is easier to learn than Base R
- Tidyverse follow some logical flow when coding which is easier to understand
Two main ways to write and execute code

- R script file can be used to write your code and the file extension is .R
  - code and output are on two different panels

- However using R Notebooks is becoming used more often
  - The output of the code is below each code chunk
  - Documenting the code and reporting can be done beside the code using text elements to get a fully formatted
  - Can use other programing language such as Bash or Python
  - Easy to share with collaborator
  - Work will with version control system
  - R Notebook is way to work with R Markdown files
R Markdown has a file extension .Rmd

- Provides an authoring framework for data science
- Fully formatted document into PDF, HTML or Word
- Combination of:
  - Written in plain text
  - Special characters for text formatting
  - R code within it to produce outputs such as table and plots
- Generate high quality reports that can be shared with an audience
R Script Demo

## Load Libraries
library(tidyverse)
library(ggplot2)

## Demo Data
# Use built in R datasets, esoph. Data from a case-control study of esophageal cancer in Ille-et-Vilaine, France.
# Is a data frame with records of 88 individuals with their alcohol and tobacco status, and how many cases of noncases in each combination category.

data("esoph")
summary(esoph)

## Demo Question:
Which age group has the highest number of alcohol consumption that are cases?

```r
> # Use built in R datasets, esoph. Data from a case-control study of esophageal cancer in Ille-et-Vilaine, France.
> # Is a data frame with records of 88 individuals with their alcohol and tobacco status, and how many cases of noncases in each combination category.
> data("esoph")
> summary(esoph)

    agegp  alcgp  tobgp ncases ncontrols
   25-34:15 0-39/day:23 0-9g/day:24 Min. :0.000  Min. :0.000
   35-44:15 40-79:23 10-19:24 1st Qu.:0.000  1st Qu.:1.000
   45-54:16 80-119:21 20-29:20 Median :1.000  Median :4.000
   55-64:16 120+:21 30+:20  Mean :2.273  Mean :8.887
   65-74:15 3rd Qu.:4.000 3rd Qu.:10.000
   75+:11  Max. :17.000  Max. :60.000
```
R Notebook Example + Demo

[Image of R Notebook and RStudio interface]
This is an [R Markdown](http://rmarkdown.rstudio.com) Notebook. When you execute code within the notebook, the results appear beneath the code.

Try executing this chunk by clicking the *Run* button within the chunk or by placing your cursor inside it and pressing *Ctrl+Shift+Enter*.
R Cheatsheets!

```r
library(tidyverse)
library(ggplot2)
data("esoph")
summary(esoph)
# baseR: which age ago has the highest number of alcohol consumption
case_df1a <- esoph[c("agegp", "alcgp", "ncases")]
case_df1b <- case_df1a[order(case_df1b$ncases, decreasing = TRUE), ]
head(case_df1b)
# Tidyverse: which age ago has the highest number of alcohol consumption that are cases
case_df2 <- esoph %>% select(c(agegp, alcgp, ncases)) %>% arrange(desc(ncases))
head(case_df2)
# Tidyverse plot the number of cases in each alcgp group by ages
ggplot(case_df2, aes(x = alcgp, y = ncases, fill = agegp)) +
  geom_bar(stat = "identity")
```

Help menu:
- R Help
- About RStudio
- RStudio Docs
- RStudio Community Forum
- Cheatsheets
  - RStudio IDE Cheat Sheet
  - Data Transformation with dplyr
  - Data Visualization with ggplot2
  - List manipulation with purrr
  - Package Development with devtools
  - Web Applications with shiny
  - Interfacing Spark with sparklyr
  - R Markdown Cheat Sheet
  - R Markdown Reference Guide
  - Browse Cheatsheets...
Outline for this afternoon

Why reproducible science?
The things you need to know to get through this course:

- Versioning and publishing code (github)
- Versioning and publishing software (docker)
- R with Rmarkdown (walk through)
- Python in ipython notebooks (walk through)
  - also google colab

The fancy bits:

- building your own binder environments
- building your own containers (docker & singularity)
Jupyter notebooks - let's open the one in our example-python-repo

docker compose up jupyter
Jupyter file browser interface

- Files
- Running
- Clusters

Select items to perform actions on them.

File Tree

- step1.ipynb
- step2.ipynb: Running Notebook in green
- README.md
Jupyter file browser interface

add new cell

Run current cell

This cell is written in markdown

(double click on it to edit)

In [12]:
```
# this cell is python code - the output shows up below
print("Hello World")
```

Hello World

Things we should learn

- bit of the interface
Now let’s check out the same notebook on google colaboratory (colab)

Instructions at:
https://github.com/krembilneuroinformatics/kcni-school-lessons/tree/master/day1
Recap: opening the notebook

https://github.com/krembilneuroinformatics/example-python-repo

example_notebook.ipynb
The colab interface (README.md)

This cell is written in markdown

(double click on it to edit)

```
# this cell is python code - the output shows up below
print("Hello World")
```

Note: this notebook uses the seaborn plotting package - which is listed in requirements.txt

Things we should learn

- bit of the interface
- python block
- markdown blocks
- plotting
colab strengths and caveates

strengths

- more power, more disk space and more RAM than binder
- easy to use
- can be left running for days (no timeouts, unlike binder)
- can connect to google drive for more cloud storage space.

caveats
to remember for this course

- each colab notebook is an island
  - it is not aware of the other data scripts or notebooks in the same folder
- You always start with the same (cleanish) linux/python-3.7.1 environment
  - so you need to install all other software inside your notebook
  - the current env is old so some newer packages may not work
Outline for this afternoon

Why reproducible science?

The things you need to know to get through this course:
- Versioning and publishing code (github)
- Versioning and publishing software (docker)
- R with Rmarkdown (walk through)
- Python in ipython notebooks (walk through)
  - also google colab

The fancy bits:

- building your own binder environments
- building your own containers (docker & singularity)
Binder - repros runnable on the web

https://mybinder.org/v2/gh/<user>/<repository>/<branch>[other-stuff]

[other-stuff] in the web address will determine what interface you see
Binder how - the files

For python (jupyter)

- requirements.txt: tells binder what python packages to install
- runtime.txt: tells binder what version of python to install

see example in:
day1/example-python-repo

For and R (with rstudio)

- runtime.txt: tells binder what version of R to use
- install.R: tells binder what R packages to install

see example in:
day1/example-r-repo
Binder - how - the URL’s

The standard format is:

http://mybinder.org/v2/gh/<github-user>/ <github-repo>/ <branch>

For this repo this is:

- http://mybinder.org/v2/gh/krembilneuroinformatics/example-python-repo/HEAD

But if directly to one particular python notebook. You could add the filepath to the end

?filepath=<filepath>

- Example:
  https://mybinder.org/v2/gh/krembilneuroinformatics/example-python-repo.git/HEAD?filepath=example_notebook.ipynb

For the RStudio environment, we must add the following at the end of the URL: ?urlpath=rstudio

- Example:
  http://mybinder.org/v2/gh/krembilneuroinformatics/example-r-repo/HEAD?urlpath=rstudio

Help at mybinder.org
let’s look at two example repo’s in the kcni-school-lessons/day1
Outline for this afternoon

Why reproducible science?
The things you need to know to get through this course:
  Versioning and publishing code (github)
  Versioning and publishing software (docker)
  R with Rmarkdown (walk through)
  Python in ipython notebooks (walk through)
  -also google colab

The fancy bits:
  building your own binder environments
  building your own containers (docker & singularity)
Step 1: write a Dockerfile

In an empty folder (new github repo)
Create a file named:

**Dockerfile**

*Everything else in this folder will be copied inside the docker image.*

The Dockerfile contains the instructions for software installation. Commands are:

- **FROM** (at top line): points to another Docker image to start from
- **RUN**: will run an installation command)
  - -run is followed by shell install commands
  - can add && to package lines into the same “layer”
- **ENTRYPOINT**: will determine one command that is run “by default”

```bash
FROM <base-image>

# stuff to install
RUN <installation command>

# more stuff (layer 2)
RUN <installation line1>
   && <installation line2>
```

Browse [https://hub.docker.com/](https://hub.docker.com/) & kcni-school-lessons/envs for examples
Step 1: write a Dockerfile

In an empty folder (new github repo)
Create a file named:

Dockerfile

Everything else in this folder will be copied inside the docker image.

The Dockerfile contains the instructions for software installation. Commands are:

- **FROM** (at top line): points to another Docker image to start from
- **RUN**: will run an installation command)
  - -run is followed by shell install commands
- **ENV**: can set environment variables
- **ENTRYPOINT**: will determine one command that is run “by default”

```bash
FROM rocker/verse:4.1.0

# adding plotting packages to from day 1 demo
RUN install2.r --error \
  --deps TRUE \
  ggrepel \
  ggthemes \
  here

## adding data grabbing packages
RUN apt-get update -qq \
    && apt-get -y --no-install-recommends \
    install wget curl git
```

Browse [https://hub.docker.com/](https://hub.docker.com/) & kcni-school-lessons/envs for examples
Building your docker image

Use “docker build” to build and test your image on your local computer. Note “-t” is for tag - or the name you will give to your docker the last argument is the folder

```
cd my_docker_folder/
docker build -t my_new_docker ./
```

You can connect your github repo to dockerhub and have dockerhub re-build your container everytime you make a push to your repo

https://hub.docker.com/
Running docker images

To run your docker:

docker run [options] <dockerhub_user>/<image>:<version>

example: docker run [options] edickie/rstudio-school:latest

Important options when you run and image:

- **--publish, -p**: allows port forwarding from the inside of the docker to the outside
  - we need this to connect rstudio or the inside of the docker to your computer's browser
- **--volume, -v**: use this to connect data outside the docker readable/writable by the docker software.
  - Docker can only “see” data that is connected to it (not everything on your computer)
Running the kcnischool-rstudio image

The full command to run the KCNI rstudio image is:

docker run --rm -it \
  -e DISABLE_AUTH=true \ 
  -p 127.0.0.1:8787:8787 \ 
  -v <path/to/your/data>:/home/rstudio/kcni-school-lessons \ 
edickie/kcnischool-rstudio:latest

where - <path/to/my/data> is the name of the folder on your computer where you cloned the kcni-school-lessons repo.
We put a “docker-compose.yml” file at the base of the kcni-school-lessons repo. This file contains defaults of the docker options - so to run the docker you type:

```
docker compose up rstudio
```

Later in the week we will start jupyter with:

```
docker compose up jupyter
```
## Docker vs Singularity

<table>
<thead>
<tr>
<th></th>
<th>Pro</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Docker</td>
<td>• Strong and building dev community</td>
<td>Needs “root” access</td>
</tr>
<tr>
<td></td>
<td>• can be pushed and pulled to dockerhub</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Works on any system (Windows, Mac, Linux)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• “Layers” decrease the hard disk space</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Image of Docker logo]</td>
<td></td>
</tr>
<tr>
<td>Singularity</td>
<td>Does not need “root” access.</td>
<td>• Smaller dev community</td>
</tr>
<tr>
<td></td>
<td>Can converted from Docker</td>
<td>• Singularity hub exists, but is less used</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Only works on Linux</td>
</tr>
<tr>
<td></td>
<td>[Image of Singularity logo]</td>
<td></td>
</tr>
</tbody>
</table>
Docker vs Singularity

take home...

- we write a Docker spec
- we run singularity containers on High performance computers and shared servers (like the CAMH SCC or SciNet)
## Translating Docker Usage to Singularity

<table>
<thead>
<tr>
<th>Feature</th>
<th>Docker</th>
<th>Singularity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Running</td>
<td>Docker run -it --rm</td>
<td>Singularity run</td>
</tr>
<tr>
<td>Mounting or binding a path</td>
<td>-v, --volume</td>
<td>-B, --bind</td>
</tr>
<tr>
<td>Attaching the “workdir”</td>
<td>-w, --workdir</td>
<td>-W</td>
</tr>
<tr>
<td>Port forwarding</td>
<td>-p, --publish</td>
<td>N/A</td>
</tr>
<tr>
<td>Removing the outside environment</td>
<td>N/A</td>
<td>-e, --cleanenv</td>
</tr>
<tr>
<td>Change the mount to $HOME</td>
<td>N/A</td>
<td>-H, --home</td>
</tr>
</tbody>
</table>
Building a singularity container from docker
Direct from dockerhub to singularity

ssh <username>@teach.scinet.utoronto.ca

# note - in this example I am adding the new container to my $SCRATCH folder
mkdir $SCRATCH/test_sing_img

# singularity build <output-image>
docker://<dockerhubuser>/<dockernname>:<version>
singularity build singularity build \\
edickie_kcnischool-jupyter_latest-2021-07-02.sif \\
docker://edickie/kcnischool-jupyter:latest
A note about “latest”

Most dockers will allow you to download the latest version of their software by typing “latest” instead of a version number.

Problem - for reporting and debugging...you need the version number.

So - for a real analysis - make sure to build a specific version and put the version number in the filename of the singularity image.
Remember - many ways to engage

?  🧑‍💻crowdcast

(during sessions)
Use the chat or the ask question!

You can always return to the session and re-watch the videos after the session ends

💻slack

virtually meet with us in gather.town

come chat with us in KCNI Summer School Slack :)

✉️KCNISchool@camh.ca

Tell us how the session went (post session survey):
https://forms.gle/ji18qLMZEZ9L16Ln6
7. Ioannidis JPA. Why most published research findings are false. PLoS Med. 2, e124 (2005). This landmark paper outlines the ways in which common practices can lead to inflated levels of false positives.

8. Simmons JP, Nelson LD & Simonsohn U. False-positive psychology: undisclosed flexibility in data collection and analysis allows presenting anything as significant. Psychol. Sci 22, 1359–1366 (2011). This paper highlights the impact of common ‘questionable research practices’ on study outcomes and proposes a set of guidelines to prevent false-positive findings. [PubMed: 22006061]

