Intro to Open and Reproducible Research

Eric Crandall

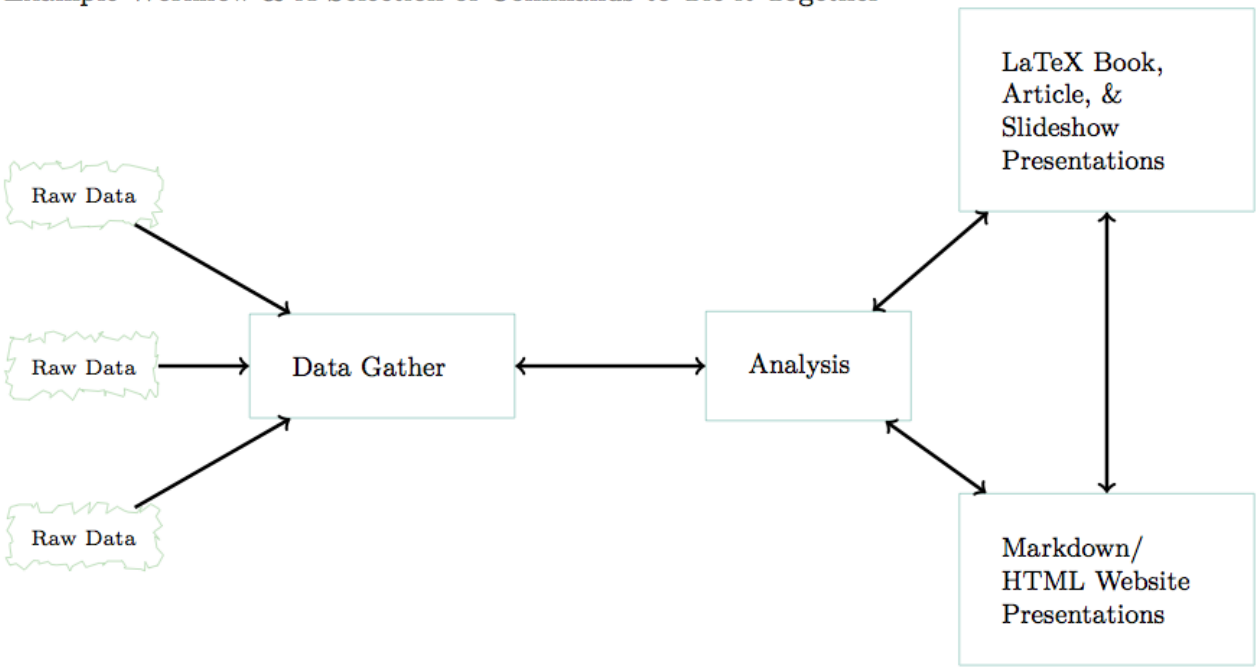
What is Reproducibility?

What is Reproducibility?

Gandrud 2014 gives this definition (especially for data analysis and computer science):

> *"The data and code used to make a finding are available and they are presented in such a way that it is (relatively) straightforward for an independent researcher to recreate the finding."*

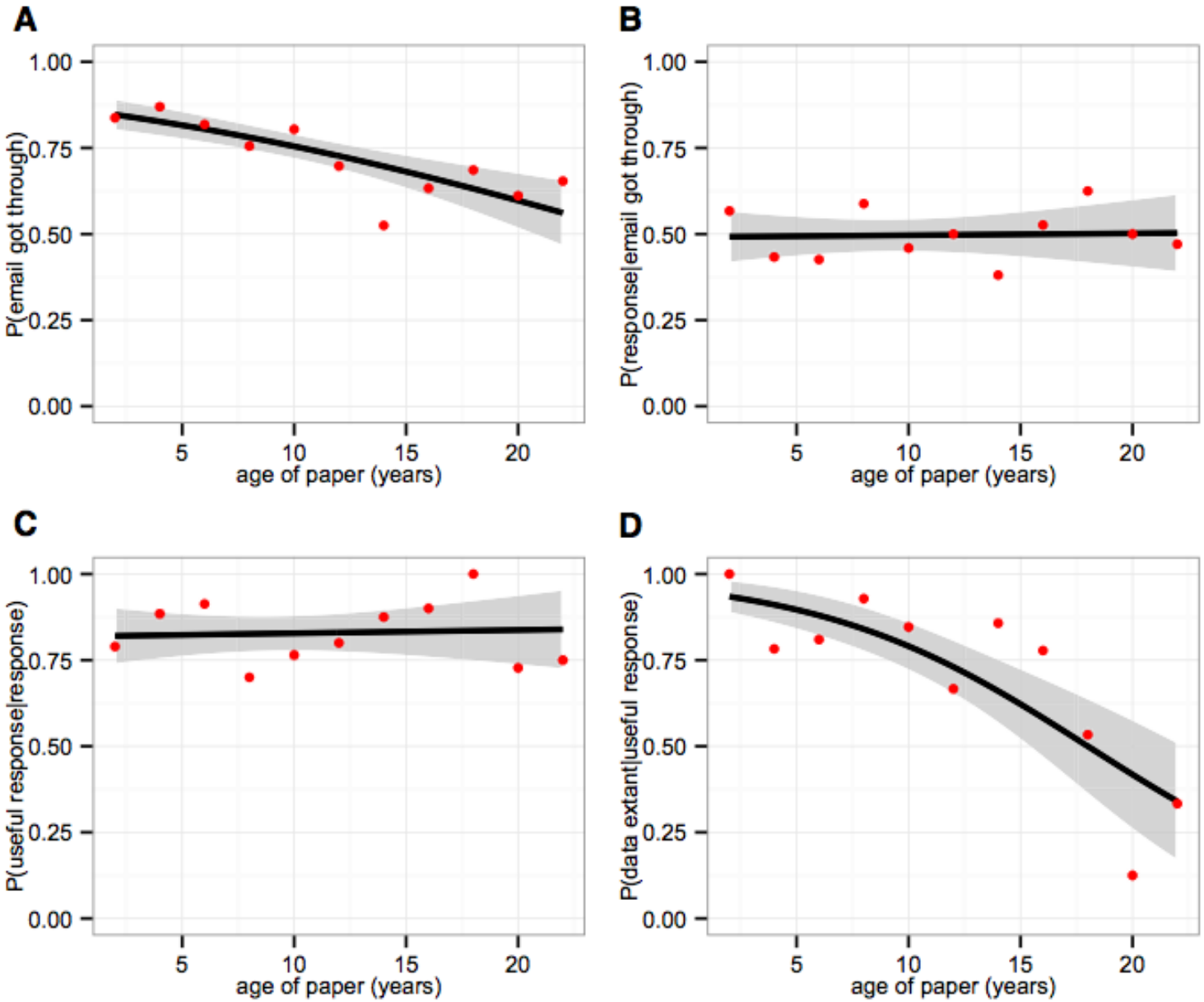
FIGURE 2.1
Example Workflow & A Selection of Commands to Tie it Together



This actually seldom happens.

Consider two interesting articles by Tim Vines: * The Availability of Research Data Declines Rapidly with Article Age + Contacted Authors of 516 datasets with morphological data for discriminant analysis published between 1991 and 2011 + Received only 101 datasets!

- "*of 516 articles published between 2 and 22 years ago. . . the odds of a data set being extant fell by 17% per year.*"



Why are Open Data and Reproducibility Important?

For Science?

- Standard to judge scientific claims
- Data and methods need to be openly available in order to be reproducible
- Avoiding effort duplication
- Encouraging cumulative knowledge development

Open Science

Open Science Links

- Open Data
- Reproducible Methods
- Open Access Publications

Figure 4: Big Data

Why are Open Data and Reproducibility Important for *You*?

- Better work habits
  - better, clearer documentation
- Better teamwork
- Re-analysis is easier
- Higher research impact

Gandrud 2014

Tools for Research Reproducibility

Tools for Research Reproducibility

- Open Source Everything
- R language
- Rstudio and knitR
- Markdown and LaTeX
- Unix operating system
- GitHub and git - version control (not covered in this class)
- Creative Commons Licensing
- Online Repositories (Dryad, Genbank,GBIF, GEOME, new ones all the time)

Reproducible Research Habits

Good habits to get into as a student!

10 Things Every Graduate Student Should Do By Carly Strasser

Stop Using Excel!

- OK, maybe not entirely - its good for quick visualizing, data entry, etc.
- It tends to be a crutch.
  - Stops you from thinking carefully about your data structure
  - Stops you from learning better ways to handle data
- Proprietary software
- Easy to mess up your data, no provenance
- Dates!
- At least keep your raw data in text format

Learn to Code

Any language.

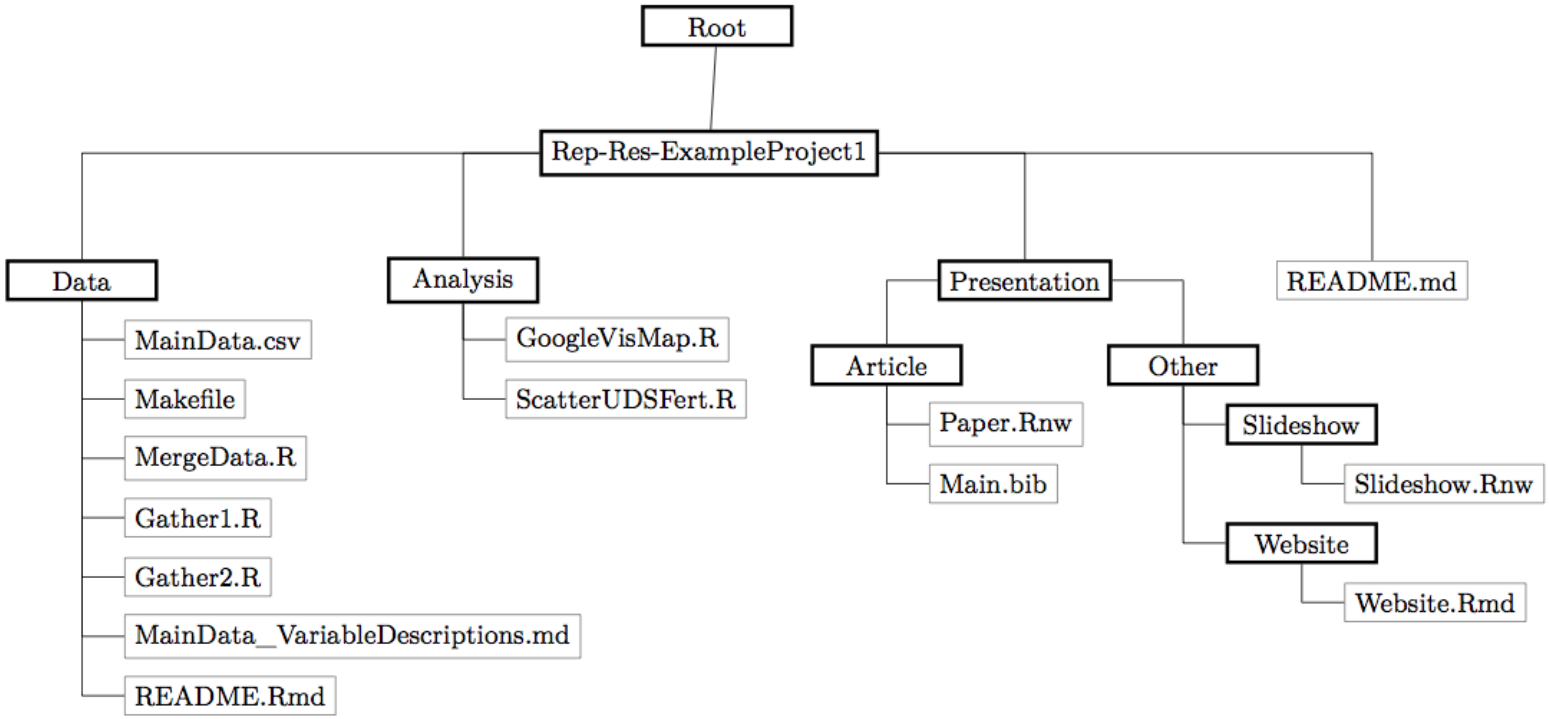
R is a great starting place.

Here is code to paste cells from excel into an R data frame!

```
data <- read.table(pipe("pbpaste"),header=T)
```

Make a plan for managing data in each project

- Keep all data (and ideally analyses) in a text file
- Think about your file structure



Make a plan for managing data in each project - 2

- Document everything
- Explicitly tie your files together

Keep an electronic (online) notebook

- Document everything!!
  - repeat your own analysis!!
  - show others what you did
- Dokuwiki is great (keeps things in plain text)
- I now use GitHub and Rmarkdown notebooks
- Go open if you're brave

Communicate Your Science

Start doing outreach now!

NSF vs. House Committee on Science

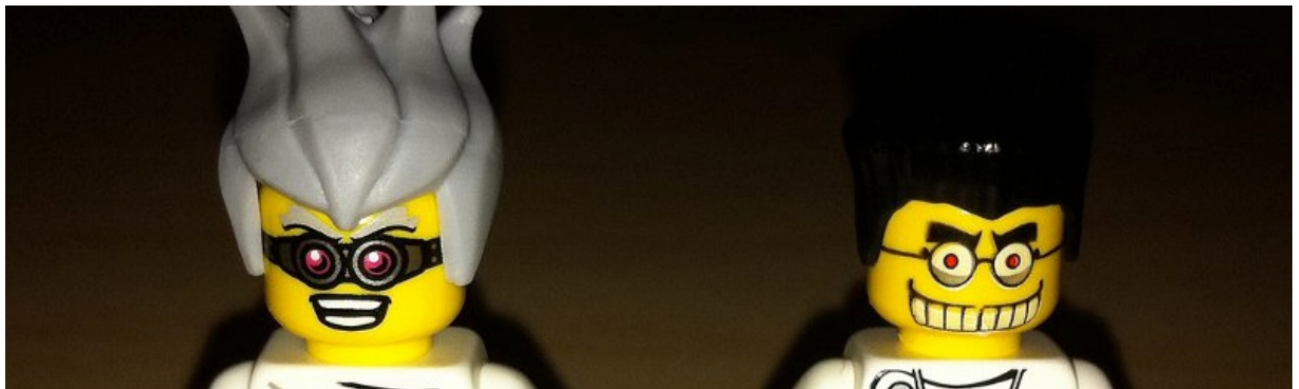
KQED Science npr PBS

Radio | TV | News | Arts | Food | Science | Education

Why Scientists are Seen as Competent but Untrustworthy (and Why it Matters)

Dr. Barry Starr, Tech Museum & Stanford University | October 6, 2014 | 1 Comment

Share: Print



Publish Open-Access Articles

- Biology Failed the Internet
  - Physics moved to pre-print servers a long time ago!
  - Open access journals (e.g. PLoS) were supposed to be a stop-gap measure!
- Many schools now have open-access funds
- The dark side of open-access
- Most journals only own the "typesetting" (because that is all they did!).
  - Therefore you may legally post a pre-print
  - BiorXiv
  - PeerJ
  - Most universities now have pre-print platforms
- ResearchGate, Academia.edu etc. are social networks whereby you are sharing articles with your "friends"

Rstudio

R studio is an Integrated Developer Environment for R * Is an IDE (integrated development environment) that *sits on top of* R and makes it easier to interact with R. * Organizes your work in R in neatly-contained packages of work (typically data and code) called “projects” * Nothing mysterious about these—just collections of files stored together in a single directory on your computer.

Git and GitHub

Git and GitHub

- Thanks to Eric Anderson for portions of the git/github part of this lecture

Git

- A version control system, or VCS, tracks the history of changes as people and teams collaborate on projects together. As the project evolves, teams can run tests, fix bugs, and contribute new code with the confidence that any version can be recovered at any time. Developers can review project history to find out:
  - Which changes were made?
  - Who made the changes?
  - When were the changes made?
  - Why were changes needed?
- All of this is stored as "commits" inside an invisible directory called .git

```
/reproducible_research/--% ls .git
COMMIT_EDITMSG  config          hooks           info
HEAD            description     index           logs
```

A typical VCS for a non-computer programmer

- Start writing `my_manuscript.doc`.
- At some point worry that MS Word is going to eat your file, so,
  - Make a "backup" called `my_manuscript_A.doc`
- Then, before overhauling the discussion, save the current file as `my_manuscript_B.doc`.
- Email it to your coauthors and then have a series of files with other extensions such as the initials of their names when they edit them and send them back.
- Etc.
- Disadvantages:
  - Hard to find a good record of what is in each version. (Wait! I liked the introduction I wrote three weeks ago. . . where is that now?)
  - A terrible system if you have multiple files that are dependent on one another
  - If you decide that you want to merge the changes you made

GitHub (and others) is a distributed version control system (DVCS)

- Git stores "snapshots" of your collection of files in a repository, which can be stored on GitHub
- For our work, the "collection of files" will be "the stuff in your RStudio project"
  - Another reason it is nice to keep everything you need for a project together in a "project directory"
- When you clone or repository, **you** get the whole version history
- When someone else clones that repository, **they also** get the whole version history.
- Git has well-developed features for merging changes made in different repositories
- Unlike once popular centralized version control systems (rcs, cvs, subversion), DVCSs like GitHub don't need a constant connection to a central repository. Developers can work anywhere and collaborate asynchronously from any time zone

GitHub (and others) is a distributed version control system (DVCS)

- Git and a DVCS system allow multiple people to work on multiple versions ("branches") of a piece of software at the same time, without breaking the main branch. This approach can be used to add features or fix bugs.
- To eliminate unnecessary work, Git and other VCSs give each contributor a unified and consistent view of a project, surfacing work that's already in progress. Seeing a transparent history of changes, who made them, and how they contribute to the development of a project helps team members stay aligned while working independently.
- Without version control, team members are subject to redundant tasks, slower timelines, and multiple copies of a single project.

GitHub Example for Software

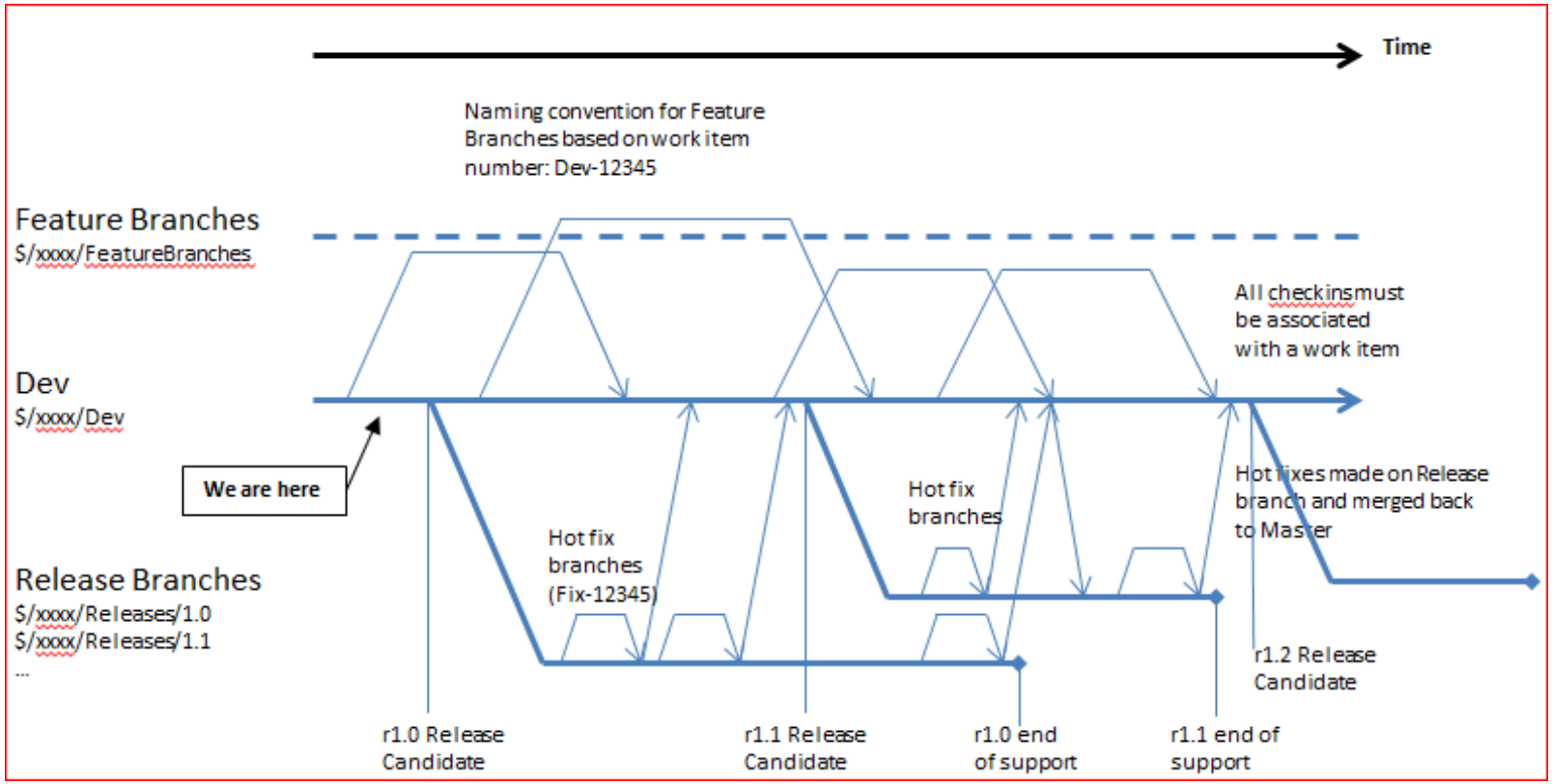


Figure 7: The Github Flow

What's a repository?

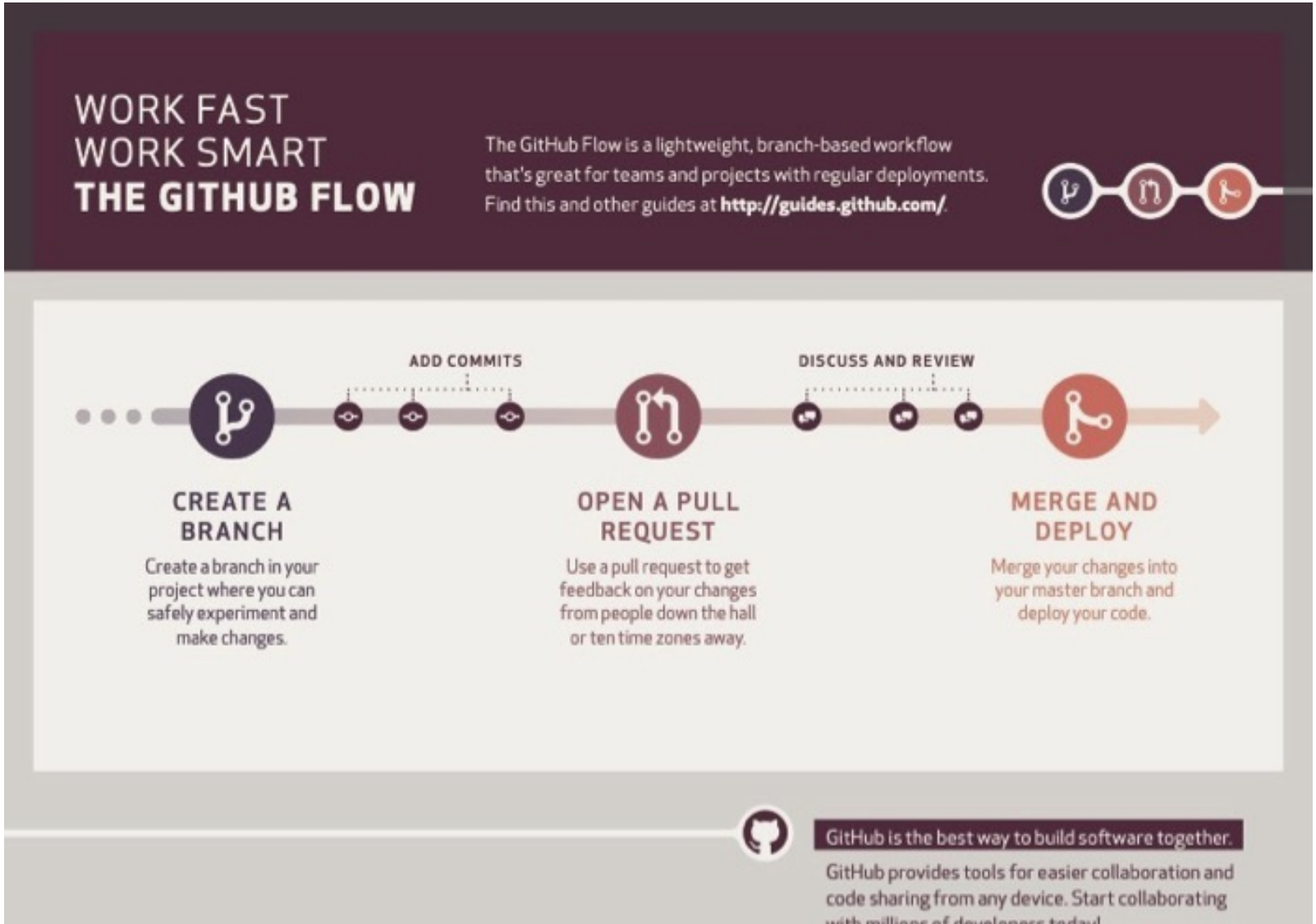
- A *repository*, or Git project, encompasses the entire collection of files and folders associated with a project (i.e. a directory), *along with each file's revision history*.
- The file history appears as snapshots in time called *commits*, and can be organized into multiple lines of development called branches.
- Because Github is a DVCS, repositories are self-contained units and anyone who owns a copy of the repository can access the entire codebase and its history.
- Using the command line or other ease-of-use interfaces, a git repository also allows for: interaction with the history, cloning, creating branches, committing, merging, comparing changes across versions of code, and more.

What's a repository?

- Working in repositories keeps development projects organized and protected. Developers are encouraged to fix bugs, or create fresh features, without fear of derailing mainline development efforts.
- Through platforms like GitHub, Git also provides more opportunities for project transparency and collaboration. Public repositories help teams work together to build the best possible final product.

The Github Flow

Video



The GitHub Flow

The GitHub flow has six steps, each with distinct benefits when implemented:

- Create a branch:Topic branches created from the canonical deployment branch (usually master) allow teams to contribute to many parallel efforts. Short-lived topic branches, in particular, keep teams focused and results in quick ships.
- Add commits:Snapshots of development efforts within a branch create safe, revertible points in the project's history.
- Open a pull request:Pull requests publicize a project's ongoing efforts and set the tone for a transparent development process.

The GitHub Flow

- Discuss and review code:Teams participate in code reviews by commenting, testing, and reviewing open pull requests. Code review is at the core of an open and participatory culture.
- Merge:Upon clicking merge, GitHub automatically performs the equivalent of a local 'git merge' operation. GitHub also keeps the entire branch development history on the merged pull request.
- Deploy:Teams can choose the best release cycles or incorporate continuous integration tools and operate with the assurance that code on the deployment branch has gone through a robust workflow.

Alternatives to Github exist!

- Bitbucket
- Gitlab
- Gitbucket

All work on a "freemium" model in which they provide free service to low-end users (like us), but charge for services needed by power users (like private repositories, more file storage space)

GitHub Tutorial

Do this in GitHub

1. Go to https://github.com/ericcrandall/reproducible_research
2. Click "Fork" in the upper right-hand corner and follow dialogue prompts to create fork in your account

Do This in RStudio

1. Select File -> New Project
2. Create project from: Choose "Version Control"
   1. Choose "Git"
   2. Input the url of this repository 1.https://github.com/*yourusername*/reproducible_research
   3. and put it somewhere
   4. I suggest a repository coming off your home directory called `github`.
   5. (browse to where you want to put it in the "create project as subdirectory of:")
   6. So it should be in `yourhome/github/reproducible_research`
   7. Hit `Create Project`

The status/staging panel

- RStudio keeps git constantly scanning the project directory to find any files that have changed or which are new.
- By clicking a file's little "check-box" you can stage it.
- Some symbols:
  - **Blue-M**: a file that is already under version control that has been modified.
  - **Yellow-?**: a file that is not under version control (yet...)
  - **Green-A**: a file that was not under version control, but which has been staged to be committed.
  - **Red-D**: a file under version control has been deleted. To make it really disappear, you have to stage its disappearance and commit.
  - **Purple-R** a file that was renamed. (Note that git in Rstudio seems to be figuring this out on its own.)

Staging Files

- You can click the check box next to various files to stage them to be part of a commit.
  - I generally stage all changes for every commit
  - But one could conceive of being more strategic...
  - at the command line, staging or adding all files to a commit is achieved by:

```
git add .
```

The Commit window

- Click "commit" to reach the commit window
- Shows a "diff" of your changes.
- In other words, what has changed between the last committed version of a file and its current state.
- Green = additions, red = deletions
- Holy smokes this is convenient
- (Note: all this output is available from the command line, but the Rstudio interface is very nice, IMHO)

Making a Commit

- Super easy:
  - After staging the files you want to commit...
  - Write a brief message (first line short, then as much after that as you want) and hit the commit button.
- Tradition is to use present tense when describing your changes.
  - as in "Add new data file, update file slurping code"
- This can be really handy when trying to find where you made an error!
- Spending a little time to write informative commit messages can pay off.
- At the command line, a commit is achieved thusly:

```
git commit -m "my commit message"
```

The History window

- Easy inspection of past commits.
- See what changes were made at each commit.
- At the command line you can see this with

```
git log
```

How does git store and keep track of things

- Everything is stored in the .git folder inside the RStudio project.
- The "working copy" gets checkout out of there
- Committed changes are recorded to the directory

What is inside of the .git directory?

We can use R to list the files.

```
# check out this file-system command in R
dir(path = ".git", all.files = TRUE, recursive = TRUE
```

```
##  [1] "COMMIT_EDITMSG"
##  [2] "config"
##  [3] "description"
##  [4] "HEAD"
##  [5] "hooks/applypatch-msg.sample"
##  [6] "hooks/commit-msg.sample"
##  [7] "hooks/fsmonitor-watchman.sample"
##  [8] "hooks/post-update.sample"
##  [9] "hooks/pre-applypatch.sample"
## [10] "hooks/pre-commit.sample"
## [11] "hooks/pre-merge-commit.sample"
## [12] "hooks/pre-push.sample"
```

How does git know a file has changed?

- Does it just look at the modification date?
- NO! It "fingerprints" every file, so it knows when it has changed from the most recent committed version.
  - Demonstration. Change a file. Save, then undo the change and save again. . . Git knows the file has been changed back to its "former self"
- SHA-1 hashes.
- You will see things like `ed00c10ae6cf7bcc35d335d2edad7e71bc0f6770` all over in Git-land.
- You can treat them as very specific names for different commits.

How can I make git ignore certain files?

- The `.gitignore` file!
- File names (and patterns) in the `.gitignore` file are ignored *recursively* (down into subdirectories), by default.
- Files won't be ignored if they are already in the repository.
- Example: `*.html` ## Go for it everyone! Git to playing
- Make some changes and commit them yourselves.
- Add some new files to the project, and commit those.
- Get familiar with the diff window.
- Check the history after a few commits.

Intro to Rmarkdown

- Designed as a text *markup* language that would be
  - Simple
  - Expressive
  - Intuitive
  - Capable of conveying intent even without being compiled into HTML or PDF
- There are many Markdown interpreters. The Rstudio folks have been using pandoc to crunch Markdown into other formats. It provides many useful extensions.
- Customizations of style are mostly separate from the **content**.
- This presentation was made in Rmarkdown!

To Do

Add Merge Conflicts Add Revert Add Rebase Add "github pages" * Open the shell (Tools->Shell...) and issue these two commands, replacing the name "John Doe" with yours, and his email with yours. + Use the email address that you gave to GitHub.

```
git config --global user.name "John Doe"
git config --global user.email johndoe@example.com
```