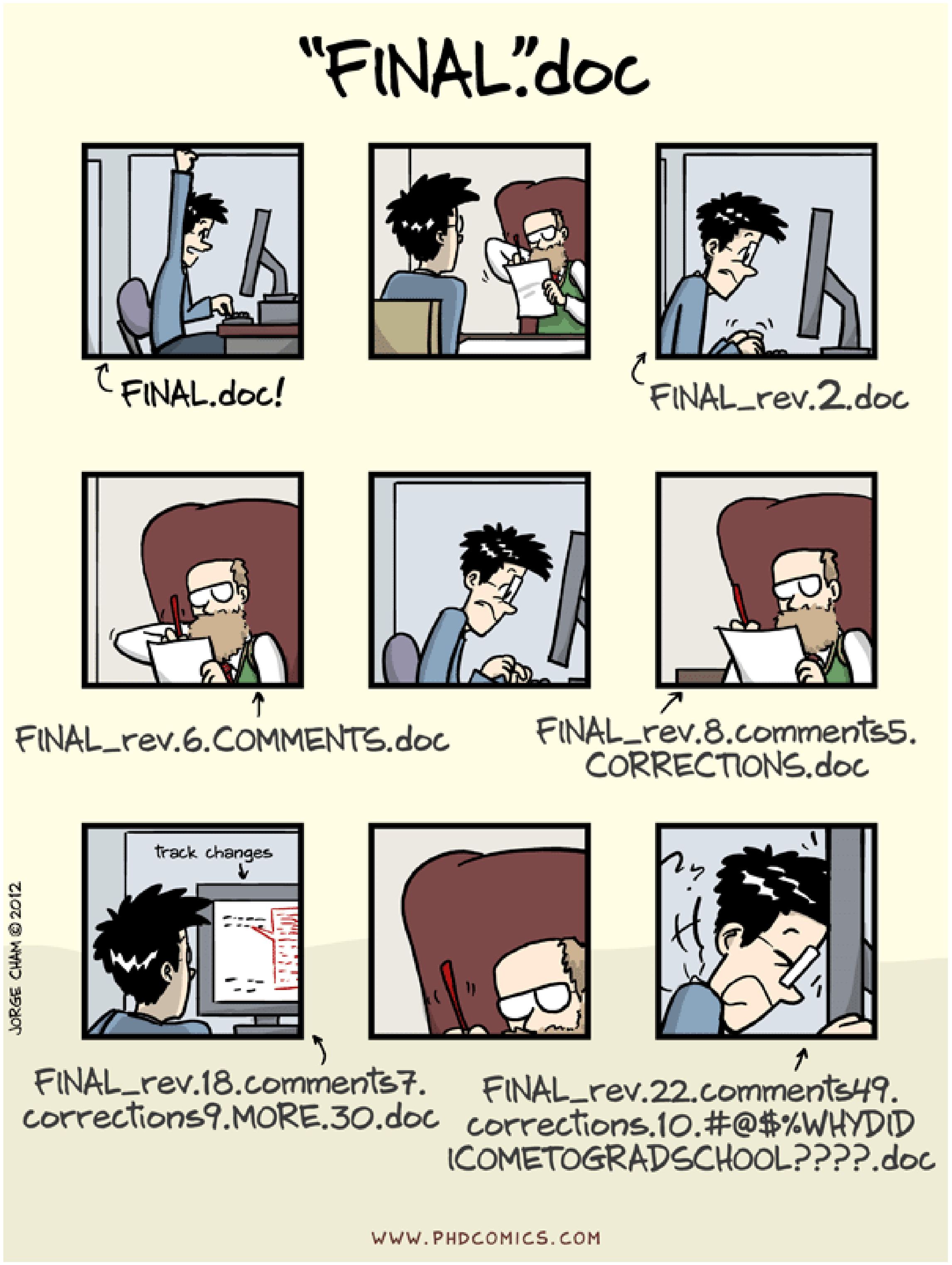
An introduction to Git and how to use it with RStudio

What is version control?


"FINAL".doc
FINAL.doc!
FINAL_rev.2.doc
FINAL_rev.6.COMMENTS.doc
FINAL_rev.8.comments5.CORRECTIONS.doc
track changes
FINAL_rev.18.comments7.corrections9.MORE.30.doc
FINAL_rev.22.comments49.corrections.10.#@$%WHYDIDICOMETOGRADSCHOOL????.doc
JORGE CHAM © 2012
WWW.PHDCOMICS.COM

You wrote your project in R, and you're ready to submit it, and you saved it as "final.html" Next you sent it to your instructor for his evaluation. He gave you a few suggestions on your submission, so you made a couple of small changes and renamed your submission "final_thisone." A few days passed, and you realized there was a typo in your figure, and you renamed it "final_thisone_really." A couple of more days passed, you got comments back from the instructor, asking for stronger conclusions. So you made a few changes consistent with what he asked and renamed your file "final_submitted." And then… new email, next class, your instructor wants more! He thinks you should rewrite the methods section to better reflect what you did in the analysis question … and the file becomes "final_submitted_really".

Two weeks pass, you gaze into your Moodle, Blackboard or Canvas grade sections and see… OMG I got a "0" on the assignment!! You got an email from the instruction apologizing, saying he inadvertently deleted your submission, would you please just re-submit. Easy, after all this project is in a sub-folder of your 'project_folder' on your USB. You open the project folder, and find that you are **mesmerized** by the number of final versions, all 33labeled "final something or other," and alas, you forget which one is the correct one. Great, now what do I submit? Sounds familiar?

Version control is a category of software that keep track of changes to your files for you. It allows you to:

- keep the entire history of a file and inspect a file throughout its life time
- tag particular versions so you can go back to them easily
- facilitate collaborations and makes contributions transparent
- experiment with code and 'new' features without breaking the main project

Version control is designed to manage your R project. It provides a convenient way to keep track of files in your project folder. Git is the "lab notebook" of the digital world. Because of its powerful and useful features, it is used for many different projects and can easily track data sets, keep track of your capstone or final project, images ('snip shots' you turned in through Moodle), and much more.

What is Git?

Git is a particular implementation of version control. It's powerful and has many features: here are a few that serve to introduce the use of Git.

What is GitHub?

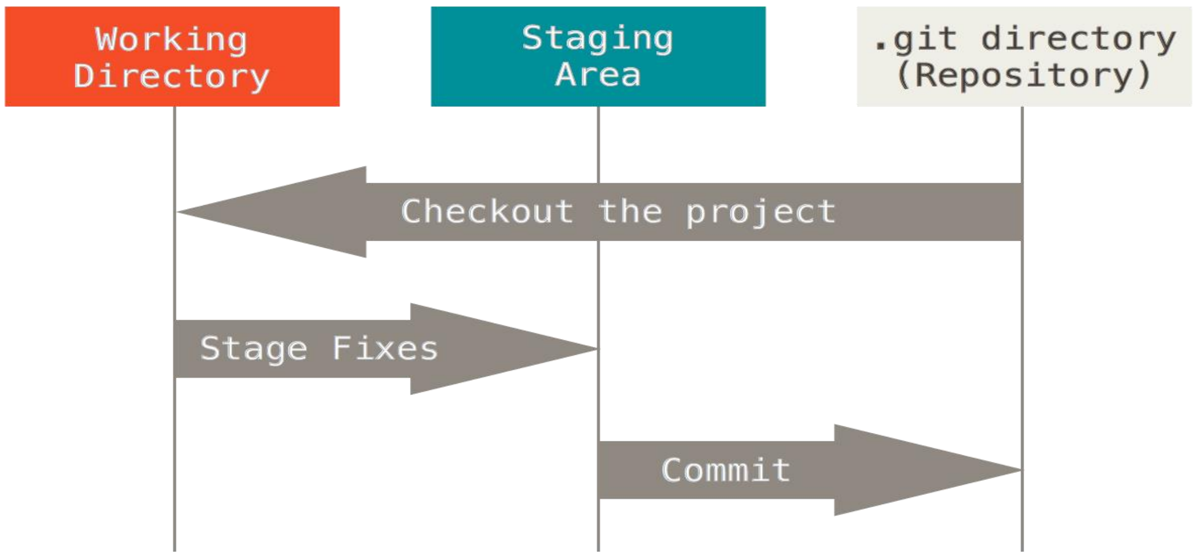
An R project (Rstudio) can incorporate Git directly in your project folder. This means that the entire history of your project is stored locally. A storage area of your project and its history is called a **repository**.

For example, you used my GitHub repository to download course materials. And GitHub provides the ability to collaborate with other people, and also to store your repository not only on your USB, but also in your GitHub account.

The GitHub website lets you store your repository publicly for free (you need to pay if you want to keep your repositories private, and you can also get an 'education' account with your `.edu` email address; that will give you some free private accounts). Storing your repositories on GitHub has many advantages. Using Git within RStudio offers a number of advantages: among them 'point and click' features to many common operations so that you don't have to remember how to manage project in the 'shell' using the command line. We will use both the command line (shell or terminal) and the 'Git' tab within RStudio

The Git workflow

Among the most used commands are "commit", "pull," "push," "branch," and 'history'; details of how these pieces fit together are shown below.



The typical workflow goes like this: - you create/edit/modify a file inside your (RStudio project) repository - you **stage** the changes, (Rstudio, 'point and click') and then commit these changes. You can only stage a file that has been changed. That's why we typically modify a file, use 'knit' to check it out and then when it 'knots' we stage and commit. This creates a permanent snapshot of the file in the Git directory along with a message that **you** write to describe what you did in that file.

When you start a new project, the files in your working directory are **untracked**, you will first need to **add** them to your repository before Git can keep track of them and their history. You should see the ‘Git’ tab in the upper right window of your RStudio project.   This is the tab that manages all the Git opertions.   If that tab is not present, open a shell (tools->shell), make sure the command window is set in your project file (>T:\project_folders\myProject) for example, and then type, “git init” at the prompt. Do not type the “’s.   Then close and reopen the project. The Git tab will now be available in your upper right pane in Rstudio.   Remember, you have to be in Rstudio, using a project.

At this point everything is still on your hard drive. To upload your modifications (i.e., your commits) to GitHub you need to **push** to your GitHub account repository

If you are working with other people you are also committing your shared repository on GitHub, you will need to **pull** to bring their modifications into your local copy of the repository.

Commits are cheap. Commit often and provide useful messages so you can keep track of what you are doing. Don’t do this:

COMMENT
DATE
CREATED MAIN LOOP & TIMING CONTROL
14 HOURS AGO
ENABLED CONFIG FILE PARSING
9 HOURS AGO
MISC BUGFIXES
5 HOURS AGO
CODE ADDITIONS/EDITS
4 HOURS AGO
MORE CODE
4 HOURS AGO
HERE HAVE CODE
4 HOURS AGO
AAAAAAAA
3 HOURS AGO
ADKFJSLKDFJSDKLFJ
3 HOURS AGO
MY HANDS ARE TYPING WORDS
2 HOURS AGO
HAAAAAAAAANDS
2 HOURS AGO
AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

(From xkcd.com)

Branching

Branching is a great feature of version control. It allows you to **duplicate** your existing repository, develop or experiment with a new feature independently, and if you like what you are doing you can **merge** these modifications back into your project.

Branching is particularly important with Git as it is the mechanism that is used when you are collaborating on external projects!

RStudio can create branches directly; i.e. you can do either:

- create them in GitHub and pull the changes in your repository;
- create them using the Rstudio GUI (in the Git tab)
- create them from the Shell (Tools > Shell) and type `git checkout -b new-branch` where ‘new-branch’ is the branch name

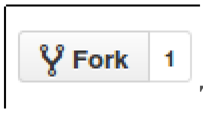
Pull requests

With a **pull** request you are asking someone who maintains a repository to pull your changes into their repository.

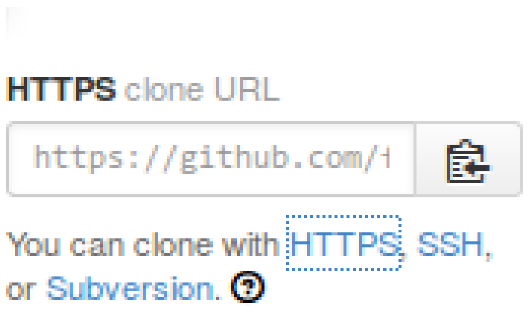
To issue a pull request:

1. First, you need a copy of the repository that contains the code you want to improve. To do so, you will **fork** it from its original location. You will also need to tell Git that this a copy of another project (a.k.a., **upstream**)
2. You need to import the code onto your computer. To do so, you will **clone** your fork to your computer.
3. Now that the code is on your computer, you can edit and modify it. Once you are done you can **push** it to your repository.
4. Once the code is in your repository, you can issue a pull request so the original owner of the code (**upstream**) can review, comment, accept (or reject) your proposed modifications.
5. At some point, you will need to **pull** changes that occur upstream into your own repository if you want to keep contributing to the project.

How does one do this using RStudio and GitHub?

1. Go to the repository page on GitHub for the project you would like to improve, and click on the “Fork” button in the top right corner. 

This will create a copy of the repository in your GitHub account. For instance, if I fork `https://github.com/jamesbrownlow/DataScience`, it will create a copy (a repository) at your GitHub account.
2. In RStudio, go to File > New Project, and choose “Version Control”, select “Git”, and type the repository URL found in *your* copy of the repository listed in the



HTTPS clone URL

https://github.com/

You can clone with HTTPS, SSH, or Subversion.

right column on the GitHub website. For example this could be https://github.com/jamesbrownlow/DataScience; then choose an appropriate location on your hard-drive/USB to store the project (like your project folder!).

3. After a few seconds, the content of the repository should appear in the “Files” pane in RStudio.
4. At this point, we need to tell Git that this project has an upstream version. This is most easily done using the Rstudio shell. Go to Tools > Shell, and enter the address of the upstream repository (in our example https://github.com/jamesbrownlow/DataScience): `` `git remote add upstream https://github.com/jamesbrownlow/DataScience` ``

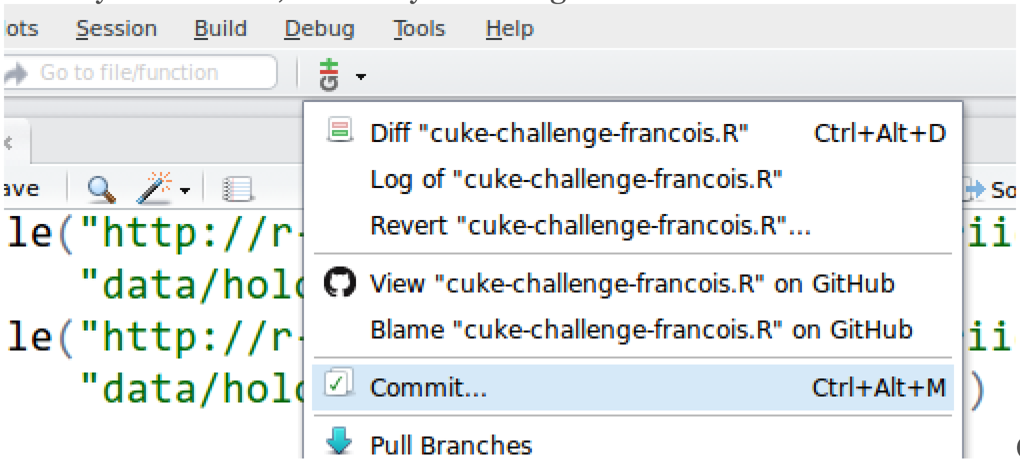
   Make sure that it worked by typing `git remote -v`, it should display 4 lines, 2 that start with `origin` and the address of your fork, and 2 that start with `upstream` and the address of the upstream repository. Note that here we used `upstream` to name the upstream repository but we could have given it another name. In this case, `upstream` is just easy to remember and accurate. Keep your shell window open.

5. We are now going to create a **branch** for our changes so they are self-contained! A branch may be created within RStudio (using the branch symbol from the ‘GIT’ tab in the project file), or enter a command in the (Tools -> shell) to create it: `` `git checkout -b proposed-fixes master` ``

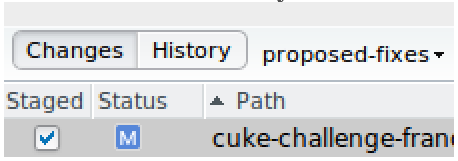
   Here, `proposed-fixes` is an arbitrary name we give to our branch. Ideally, you want to choose a name that summarize what your proposed changes are about. After seeing the message `Switched to a new branch...`, you can close the shell window.

6. Open the script you would like to modify, and change the code as needed. Save your modifications.

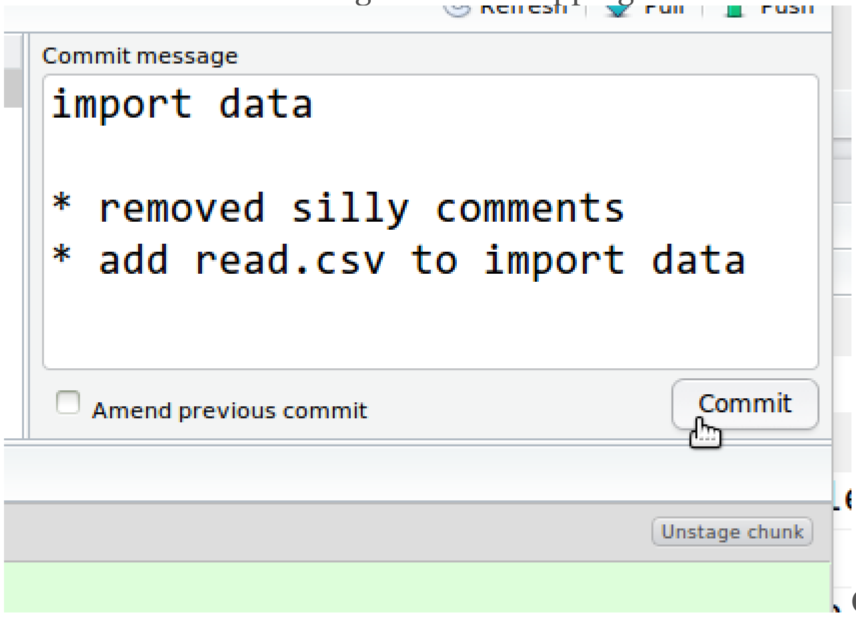
7. When you are done, commit your changes:


ots Session Build Debug Tools Help
Go to file/function
Diff "cuke-challenge-francois.R" Ctrl+Alt+D
Log of "cuke-challenge-francois.R"
Revert "cuke-challenge-francois.R"...
View "cuke-challenge-francois.R" on GitHub
Blame "cuke-challenge-francois.R" on GitHub
Commit... Ctrl+Alt+M
Pull Branches

Click the staged checkbox for the files that are affected by the modifications you want to commit to your repository:



Make sure that you are in the correct branch, (`proposed-fixes` appears next to the History button), and write a commit message: The first line should be short (< 50 characters), additional details can be given after skipping a line:



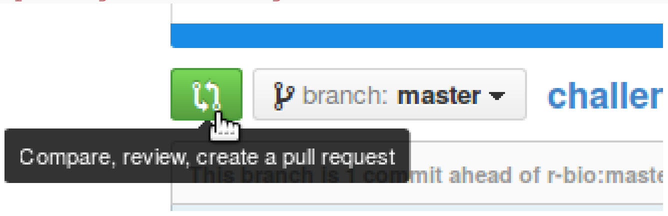
```
import data

* removed silly comments
* add read.csv to import data
```

Close the window, and go back to the shell. There, type:

```
git push origin proposed-fixes
```

This command sends your modification to your fork on GitHub.

8. Go to GitHub to view your fork, in our example that would be: `https://github.com/jamesbrownlow/DataScience`, and click on the green icon:


branch: master
challer
Compare, review, create a pull request
This branch is 1 commit ahead of r-bio:maste

9. Make sure that the correct branch is selected in the `compare` drop-down menu in the top right, and click on the green button: “Create pull request”.
10. Leave a message with some explanation for your proposed changes and suggestions and click on the green button “Create Pull Request”. This will send a notification to the owner of the repository who will review your request.

What to do if you receive a pull request?

If you want to accept the pull request, you can click on the “Merge pull request” button in your repository, and after adding an optional comment, click again on the “Accept pull request” button.

After that, you need to import (pull) these changes into your local copy of the repository. You should be able to do pull the changes by clicking on “Pull branches” in the “Git” menu of your R project in RStudio.

Exploring further

There are many resources on the web to learn about Git and GitHub:

- Git Tower provides a nice infographics that illustrates how the different parts of the Git workflow fit together. Git workflow
- An excellent (but maybe too comprehensive?) is the Pro Git Book., included in the original GitHub distribution. Chapters 2 and 3 in this book are a great way to learn how to use GitHub to contribute to a project.