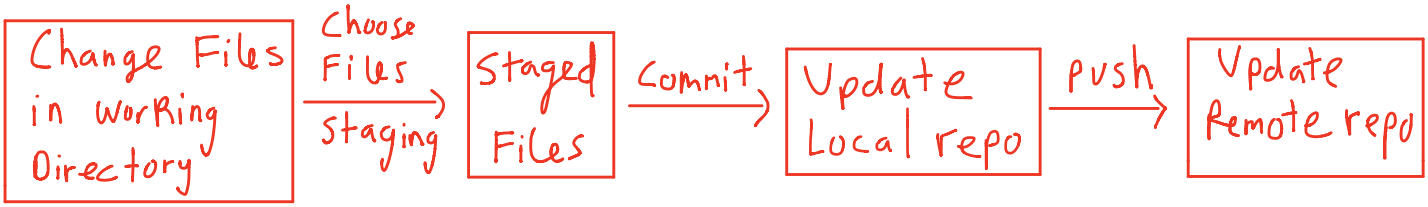
Introduction to Git for Social Scientists by Hammad Shaikh

Introduction and motivation

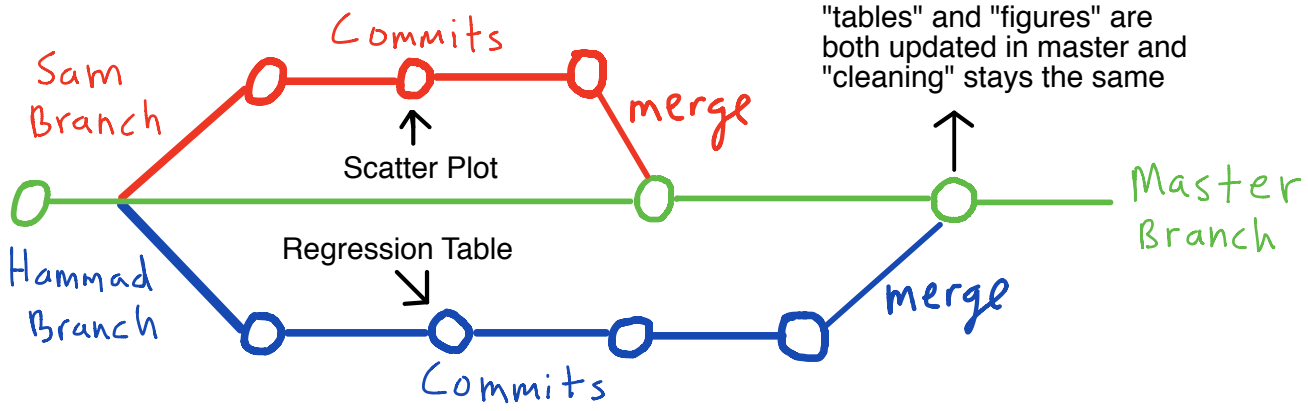
- Git is a program that does version control and allows you track changes as a script files is developed over time. Extremely useful for researchers doing a lot of coding work.
- Arguable more flexible than the version control done by dropbox.
  - More difficult in dropbox to work on the same script file with multiple collaborators.
  - Dropbox or google drive are still great for storing large data sets.
- Git can be integrated with online collaboration tools such as GitHub or BitBucket.
- Git can be used on both local **repositories** (i.e. a folder) on your computer or can also be connected to collaborative remote repositories hosted on a server (e.g. hosted on GitHub).
- Git can do version control on script files (Stata, Matlab, Python, R, etc), and also TeX files.
- Git is not suitable for doing version control with non-text documents such PDF files or images.
- A Git repository is essentially a history of **commits** and how they relate to each other.
  - A **commit** is message associated with the version of the script file.
- The Git components are 1) Make changes in **working directory**, 2) Add changes you want to **commit** to the **staging area**, 3) **commit** a snapshot of staging area (unique version of your file), and 4) **push** the **commit** to **remote repository** (not required for doing version control locally).



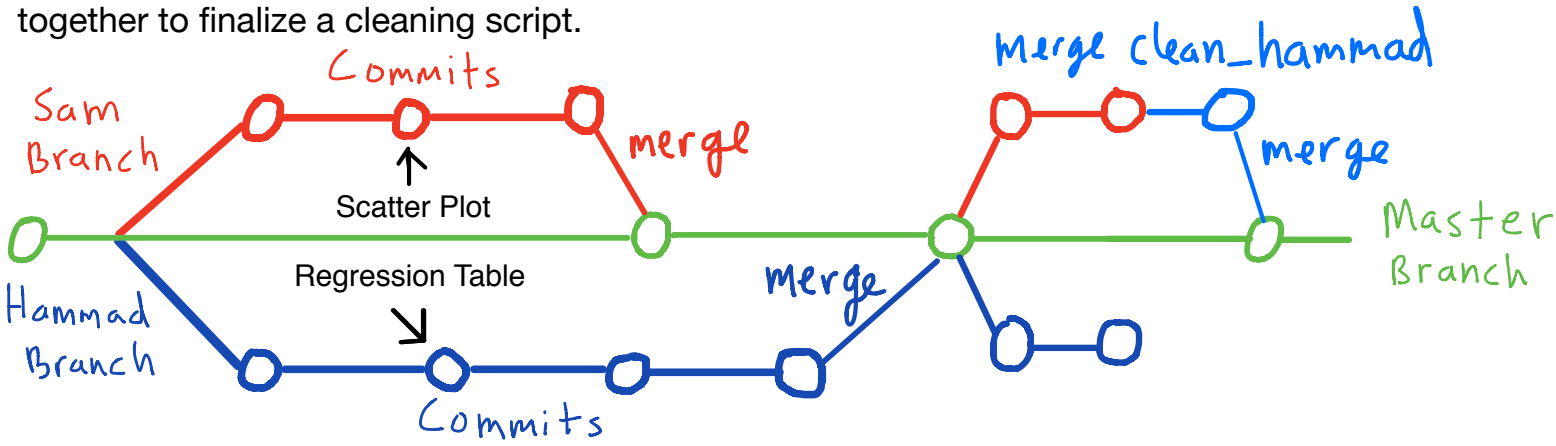
- Don't worry if the above terminology sounds confusing, we will go over concrete examples later.
- Git allows you to revert your code to any previous **commit**. This is helpful for replicating results of previous versions of your project and identifying why results changed after fixing coding bugs.
- For collaborative projects, you **push** to the remote repository to save the **commits** on the server, and **pull** from the remote repository to load commits of collaborators onto your computer.
- Hosting local repositories on a server does not only allow for collaboration opportunities but also a back-up of your script files. So your files are both on your local drive and on a online server.

Conceptual Example of Collaboration using a Remote Repository

- Suppose there are two coders, Hammad and Sam. The data is all cleaned up so Hammad and Sam want to analyze their data and produce Tables and Figures for their paper.
- The **master branch** has three script files (.do, .m, .py, etc) "cleaning", "tables", and "figures".
- Hammad will work on the "tables" script and Sam will work on on the "figures" script to produce Tables and Figures respectively. Hammad will create a branch "hammad_tables" and Sam will create another branch "sam_figures".
- Suppose Sam completes the figures first. He will **merge** his updates with the master branch. Hammad eventually finishes the Tables after Sam and also **merges** his updates to the master branch. This can be illustrated as:



- In the above example Sam and Hammad work on their own separate script files and **push** their **commits** back into their **remote repository branches**. Eventually both branches are **merged** with the master. Now any collaborator can **pull** the **master** and work from the latest scripts.
- A more complicated, but very useful feature of Git is **merging** two different versions of a script file together (e.g. Hammad and Sam want to edit the same file).
- Suppose Hammad and Sam want to process additional data they recently received using the "cleaning" script. Both have their own approach to cleaning the data, so decide to make their own "clean_sam" and "clean_hammad" branches to work simultaneously on the same script.
- Both coders make progress on the cleaning script and decide that the best data processing script would be some combination of both of their work. Hence they **merge** their two branches together to finalize a cleaning script.



- The above figure illustrates Sam **merging** his script with Hammad, and then **merging** to master.
- I would recommended using a **private remote repository** even for individual projects as you can leverage several features of the online platforms. But if your files contain confidential data, or you are working on computers not connected to the internet, you can still use Git locally!
- The most important feature of Git is **committing** major code changes. This allows you track versions of script file and takes less than a minute to do.
- Your coding time on a daily basis doesn't really change if you start using Git. More complicated features like **merging/reverting** will take longer, you only need to use them once in a while and they will save you time relative to the brute force alternatives to achieve the same tasks.

Configuring Git Locally

- If you are using Mac than you can use terminal to write Git commands.
- However on windows you we will need some setup to make them work with command prompt. Lots of articles online about how to do this: https://support.codebasehq.com/articles/getting-started/git-on-windows
- Check if Git is installed: `git --version`
  - If you don't have it, you will be prompted to install it
  - You can also download it from: https://git-scm.com/download
- Set name: `git config --global user.name "your_name"`
- Set email: `git config --global user.email "your_email"`

Text editors that work well with Git

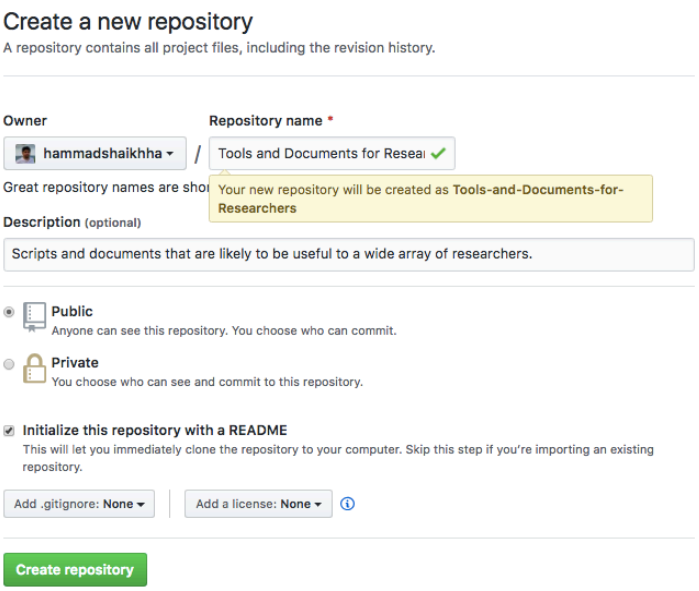
- 1) Atom, 2) Sublime Text, 3) TextMate, and 4) Nano are popular text editor to use with Git.
- Atoms user interface helps you identify the branch you are on and automatically updates your coding environment when you move across branches. I will use Atom in some of the examples.
- Change default settings to associate Git with your text editor: https://help.github.com/en/articles/associating-text-editors-with-git
- Set default code editor (I use Atom): `git config --global core.editor "atom -w"`
- Print Git configuration (checks your current configurations): `git config --list`

Setting up a Git Repository

- Create a new folder somewhere on your computer e.g. "Learning Git"
- Open up terminal and change directory into that folder.
  - cd *folder_path* (on Mac you can drag and drop folder into terminal to get the path)
- Initialize repository with Git: `git init`
  - Running `ls -a` should show a .git file
- The repository is now ready to track changes of your coding files (more on this later).

Connecting Git to a Remote Repository

- Popular online platforms compatible with Git are 1) GitHub, 2) BitBucket, and 3) GitLab
  - Make an account on your favourite platform
- Make a repository on the online platform and **clone** it to your local drive
  - Cloning using GitHub: https://help.github.com/en/articles/cloning-a-repository
- Change directory into your cloned repository and type: `git config --list`
  - You should see `remote.origin.url: link to your repository`
- I am going to setup a "Tools and Documents for Researchers" remote repository on GitHub:



- I will now clone the repository to my local machine.

```
Hammads-MacBook-Air:GitHub hammadshaikh$ git clone https://github.com/hammadshaikhha/Tools-and-Documents-for-Researchers.git
Cloning into 'Tools-and-Documents-for-Researchers'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
Checking connectivity... done.
```

Git Jargon

- Git involves using a bunch of fancy terminology to represent different tasks.
- **Repository**: a folder initialized with Git that contains the different versions of your project.
- **Origin**: remote repository hosted on a server like GitHub or Bitbucket.
- **Fetch:** Checks the remote repository for new commits (doesn't download or merge any of them)
- **Clone:** Used to copy a remote repository (e.g. from GitHub) onto your local drive.
- **Commit:** Permanent unique snapshot of a collection of files that are tracked by Git.
  - Stored in the .git directory, each commit has a unique ID.
  - **Head:** refers to the current commit.
- **Tag:** A label that can be attached to a important commit (e.g. highlight major changes)
- **Diff:** The difference between two commits (identifies differences across versions of script).
- **.gitignore:** A text file that indicates a set of file extensions that Git should ignore.
- **Branch:** A separate sequence of commits (e.g. a "test" branch for experimenting).

- **Master:** The primary branch in which the project is developed.
- **Stash:** Setting aside a set of edits without committing them (can return to them later).
  - Useful when switching from a branch with uncompleted work to another branch.
- **Pull:** Retrieves content from remote repository to update your local repository.
- **Push:** Upload local repository content into remote repository.
- **Merge:** Combine sequence of commits across multiple branches into one unified history.
- **Merge Conflict:** Arises from competing changes done on the same file.
  - Need to chose which changes to keep for finalized version.
- **Staging area:** contains changes to your files that will be included in next commit.
  - Acts as a extra blanket of security, by adding files to the staging area we can review our changes before committing them.

Terminal Basics

- Go into directory: `cd folder_name`
- Go back a step from current directory: `cd ..`
- List files in directory: `ls`
  - Show hidden files starting with ".": `ls -a`
- Make new folder: `mkdir folder_name`
- Remove folder: `rmdir folder_name`
- Show contents of a file: `cat file_name`
- Access previous commands: up arrow
- Show history of terminal commands: history
  - Save history in text file: history > history.txt
- Its possible to maintain version control without extra terminal coding by using a Git user interface tools such as 1) Source Tree or 2) GitKraken.
  - But its still good to learn Git using terminal first before getting into these tools.

Ignoring irrelevant files for version control

- By default Git will track of all files in the git initialized folder. This is not ideal as we don't want to do version control on everything, but just the key script files.
- Create a .txt file called .gitignore
- The .gitignore file is used to ignore junk files that we don't want to do version control with
  - If you are on Mac include .DS_Store in your .gitignore
- Save the .gitignore in the same folder, e.g. "Learning Git"
- The .gitignore file I use for this document: https://github.com/hammadshaikhha/Tools-and-Documents-for-Researchers/blob/master/.gitignore.txt
- A list of common .gitignore files: https://github.com/github/gitignore
  - TeX.gitignore, Python.gitignore, R.gitignore, and macOS.gitignore are likely relevant for the audience of this document
- Open source software for constructing gitignore files: https://gitignore.io/
- Once you get more experience with Git you can setup a global .gitignore for all Git folders
  - `git config --global core.excludesfile ~/.gitignore_global`

Committing the .gitignore text file

- Recall we created a .gitignore text file previously, now we will commit this. In terminal type:
  - `git status`: will show .gitignore as a untracked file
  - `git add .gitignore`: move .gitignore to the staging area
  - `git commit -m` "Create gitignore file": commit the .gitignore with a message

Popular Git Commands:

- Version control using Git can be done from terminal or using some type of user interface. In this document I will cover both methods but focus mainly terminal.
- Although not exhaustive, the list below includes commonly used terminal commands.

Most commonly used git commands for solo projects (e.g. working on your own locally)

- `git status:` can run anytime, gives current status of repository
  - Includes list of files that have been modified and the branch you are on
- `git log:` shows you history of commits with author, commit ID, message, and date
  - `git log` --oneline: condensed version of log with just commit ID and message
- `git add` *file_name*: Choose what file you want to commit, moves these files to a "staging area"
  - Use `git add .` to add all modified files to staging area
- `git commit` -m "*commit_message*": make commit with a message
- `git help` *command_name*: shows manual with description of command (press q to exit)
- `git diff` *commitid1..commitid2*: shows difference between two commits
  - `git difftool` *commitid1..commitid2*: uses your local difftool (I use KDiff3)
    - KDiff3: http://kdiff3.sourceforge.net/
  - http://www.mergely.com/editor is an online diff tool, very nice GUI

Common git commands for collaborative projects (e.g. remote repository)

- `git clone` *`remote_url`*`:` cloning repository on local machine
- `git fetch: updates the remote tracking of branches`
- `git pull:` bringing down changes in remote repo (e.g. on GitHub) to your local machine
  - Equivalent to executing i) `git fetch origin` then ii) `git merge origin master`
- `git branch` *branch_name*: make new branch which is separate from the master
  - `git branch –a`: identify current branch and show all other branches
  - `git branch –d` *branch_name*: delete branch
- `git checkout` *branch_name*: shows commit on the specified branch (move branches)
  - `git checkout` *commit_id*: check the files in the *commit_id*
  - `git checkout` -b *branch_name*: create new branch and checkout to it
    - `git checkout` *commit_id* -b *-branch_name*: create branch from given commit
  - `git checkout` *`branch_name`* `–f:` ignore local changes when switching branches
- `git push`: push local changes up to remote repo, opposite of `git pull`
  - `git push origin` *branch_name*: push local commits to a remote branch
  - `git push –f origin HEAD^:` *branch_name* (Undo previous push on remote repo)
- `git merge` *branch_name*: merge one branch with another branch
- `git diff` *local_branch_name* origin/*remote_branch* -- *file_name*
  - Get difference between local branch file to the corresponding remote branch file

Other useful git commands

- `git rebase:` move branch commits in front of master
  - Advantage: show commits in linear order, Disadvantage: re-writes history
- `git revert` *commit_id*`:` creates new commit to undo changes from a previous commit
- `git reset:` opposite of git add, remove files from staged area
  - `git reset ––hard HEAD:` permanently delete any uncommitted changes
  - `git reset` *commit_id*: Delete all commits beyond *commit_id*
- `git stash:` save your work in progress without committing to use at some later point
  - `git stash pop:` removes stash from list and applies it to working directory
- `git rm` -- *file_name*: remove file from working directory
- `git tag –a` *`tagname`*

Example of using Git locally with LaTeX

- Please read "Configuring Git Locally" section above before going through this example.

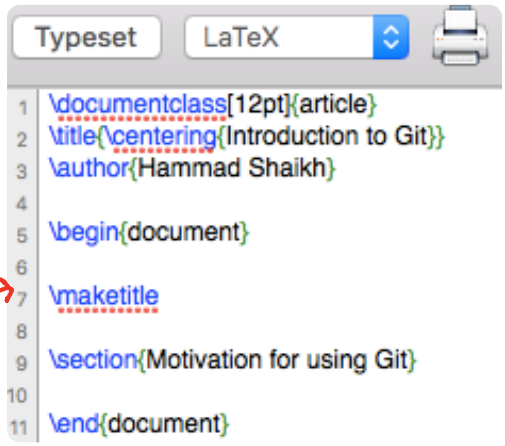
Create Git repository with a simple LaTeX (I am using TeX shop) document

- `git init:` Initialize repository
- `git status:` Check for any modified files

```
[Hammads-MacBook-Air:git TeX hammadshaikh$ git status
 On branch master

 Initial commit

nothing to commit (create/copy files and use "git add" to track)
```

No files added

Add .tex



Check status of initialized git repo with new .tex file

- `git status:` check for our .tex file

```
[Hammads-MacBook-Air:git TeX hammadshaikh$ git status
 On branch master

 Initial commit

 Untracked files:
   (use "git add <file>..." to include in what will be committed)

         IntroGit.tex

 nothing added to commit but untracked files present (use "git add" to track)
```

Move IntoGit.tex to staging area

- `git add IntroGit.tex:` Move .tex to staging area
- `git status:` Check that .tex file has moved to staging area

```
[Hammads-MacBook-Air:git TeX hammadshaikh$ git add IntroGit.tex
[Hammads-MacBook-Air:git TeX hammadshaikh$ git status
 On branch master

 Initial commit

 Changes to be committed:
   (use "git rm --cached <file>..." to unstage)

         new file:   IntroGit.tex
```

Commit change to TeX file

- `git commit -m "Create Intro to Git document"`
- `git log --oneline:` Check log of our commit

```
[Hammads-MacBook-Air:git TeX hammadshaikh$ git commit -m "Create Intro to Git document"
 [master (root-commit) d19943d] Create Intro to Git document
  1 file changed, 16 insertions(+)
  create mode 100644 IntroGit.tex
[Hammads-MacBook-Air:git TeX hammadshaikh$ git log --oneline
 d19943d Create Intro to Git document
```

Add some text and compile PDF

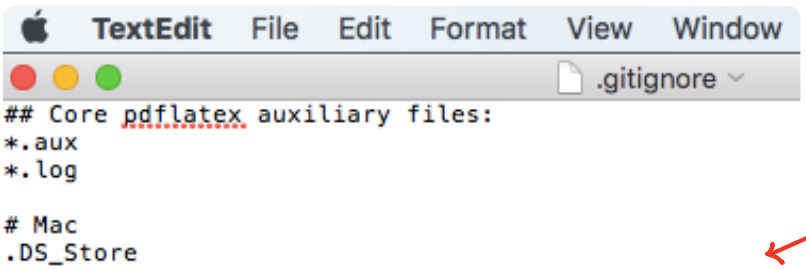
- `git status:` Check for the updated .tex file

```
Hammads-MacBook-Air:git TeX hammadshaikh$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   IntroGit.tex

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        .DS_Store
        IntroGit.aux
        IntroGit.log
        IntroGit.pdf
        IntroGit.synctex.gz

no changes added to commit (use "git add" and/or "git commit -a")
```

We don't want to track this

Add a .gitignore file to ignore LaTeX and Mac junk files (skip if .gitignore already setup)

- touch .gitignore
- open .gitignore



Text file with extensions to ignore

- git status
- git add .gitignore.txt
- git commit -m "Create gitignore for mac and tex junk"
- git status

Notice now that ignored files do not show under git status

```
Hammads-MacBook-Air:git TeX hammadshaikh$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   IntroGit.tex

no changes added to commit (use "git add" and/or "git commit -a")
```

Add motivation section

- git add .
- git commit -m "Complete draft of Git motivation section"

Write a new section and commit it

- git status
- git add .
- git commit -m "Complete draft of getting started with Git"

Check differences in .tex document across commits using Terminal

- git diff commit_id1..commit_id2
  - Let *commit_id1* correspond to "getting started" and *commit_id2* with "motivation section"

- Scroll down and Git will show the new edits you made

```
 Git is a program that allows you to keep track of several versions of your
coding files. It keeps a history of all the changes and you can revert to an
y point in the history. Git can be used locally on your own computer, or you
 can also host a remote repository that can be used to collaborate with othe
rs and keep back ups of your script files.

 \section{Getting started with Git}
+We can get started with Git by installing the program, configuring the sett
ings, and initializing a new folder with Git. In this tutorial we will get s
tarted with tracking versions of a TeX file using Git.
```

Terminal shows the newly added section

- You may be thinking the terminal diff view isn't ideal, I agree.

Use KDiff3 tool to show difference across commits (http://kdiff3.sourceforge.net/)

- `git difftool commit_id1..commit_id2`
- This diff view is much better and this should illustrate the benefits of version control

```
01  \documentclass[12pt]{article}
02  \title{\centering{Introduction to Git}}
03  \author{Hammad Shaikh}
04
05  \begin{document}
06
07  \maketitle
08
09  \section{Motivation for using Git}
10  Git is a program that allows you to keep track
    of several versions of your coding files. It
    keeps a history of all the changes and you can
    revert to any point in the history. Git can be
    used locally on your own computer, or you can
    also host a remote repository that can be used
    to collaborate with others and keep back ups of
    your script files.
11
12  \section{Getting started with Git}
13
14  \end{document}
```

```
01  \documentclass[12pt]{article}
02  \title{\centering{Introduction to Git}}
03  \author{Hammad Shaikh}
04
05  \begin{document}
06
07  \maketitle
08
09  \section{Motivation for using Git}
10  Git is a program that allows you to keep track
    of several versions of your coding files. It
    keeps a history of all the changes and you can
    revert to any point in the history. Git can be
    used locally on your own computer, or you can
    also host a remote repository that can be used
    to collaborate with others and keep back ups of
    your script files.
11
12  \section{Getting started with Git}
13  We can get started with Git by installing the
    program, configuring the settings, and
    initializing a new folder with Git. In this
    tutorial we will get started with tracking
    versions of a TeX file using Git.
14
15  \end{document}
```

Edit the Motivation section

- `git status`
- `git add .`
- `git commit –m "Motivation for using Git section updated"`

```
9  \section{Motivation for using Git}
10 Git is a program that allows you to keep track of several versions of your coding files. It keeps a history of all the changes and you can revert to any point in the
   history. Git can be used locally on your own computer, or you can also host a remote repository that can be used to collaborate with others and keep back ups of
   your script files.
11
12 You actually don't need to know a lot about coding to use Git if use a GUI such as Source Tree or GitKraken.
13
14 \section{Getting started with Git}
15 We can get started with Git by installing the program, configuring the settings, and initializing a new folder with Git. In this tutorial we will get started with tracking
   versions of a TeX file using Git.
```

Check older version of the TeX document

- git checkout *commit_id*
  - Choose the *commit_id* for "`getting started with Git`"
- You should see the highlighted line above disappear in your TeX editor.
- This may not occur in all TeX editors, close and reopen your file if this is the case.

```
9  \section{Motivation for using Git}
10 Git is a program that allows you to keep track of several versions of your coding files. It keeps a history of all the changes and you can revert to any point in the
   history. Git can be used locally on your own computer, or you can also host a remote repository that can be used to collaborate with others and keep back ups of
   your script files.
11
12 \section{Getting started with Git}
13 We can get started with Git by installing the program, configuring the settings, and initializing a new folder with Git. In this tutorial we will get started with tracking
   versions of a TeX file using Git.
```

- git status: noticed the HEAD is de-attached
- git checkout master: this will re-attach the HEAD
    - This works here since we didn't make any changes. Please see: https://stackoverflow.com/questions/5772192/how-can-i-reconcile-detached-head-with-master-origin for discussion of re-attaching the HEAD in more complex scenerios.

Create a new section e.g. "Using Git without any code"

- git add .
- git commit -m "Completed draft of Git without code section"

```
\section{Motivation for using Git}
Git is a program that allows you to keep track of several versions of your coding files. It keeps a history of all the changes and you can revert to any point in the history. Git can be used locally on your own computer, or you can also host a remote repository that can be used to collaborate with others and keep back ups of your script files.

You actually don't need to know a lot about coding to use Git if use a GUI such as Source Tree or GitKraken.

\section{Getting started with Git}
We can get started with Git by installing the program, configuring the settings, and initializing a new folder with Git. In this tutorial we will get started with tracking versions of a TeX file using Git.

\section{Using Git without any code}
You don't necessarily need to be an expert in the Git terminal commands to do version control for your projects. There are graphical user interfaces available for Git that you can use to add commits, push to remote repository, create branches, etc using just a few clicks. The following Git GUI are fairly popular 1) Sourcetree, 2) GitKraken, and 3) GitHub desktop.
```

Add a new section e.g. "Using Git with Bitbucket"

- git add .
- git commit -m "Completed draft of Git with Bitbucket"

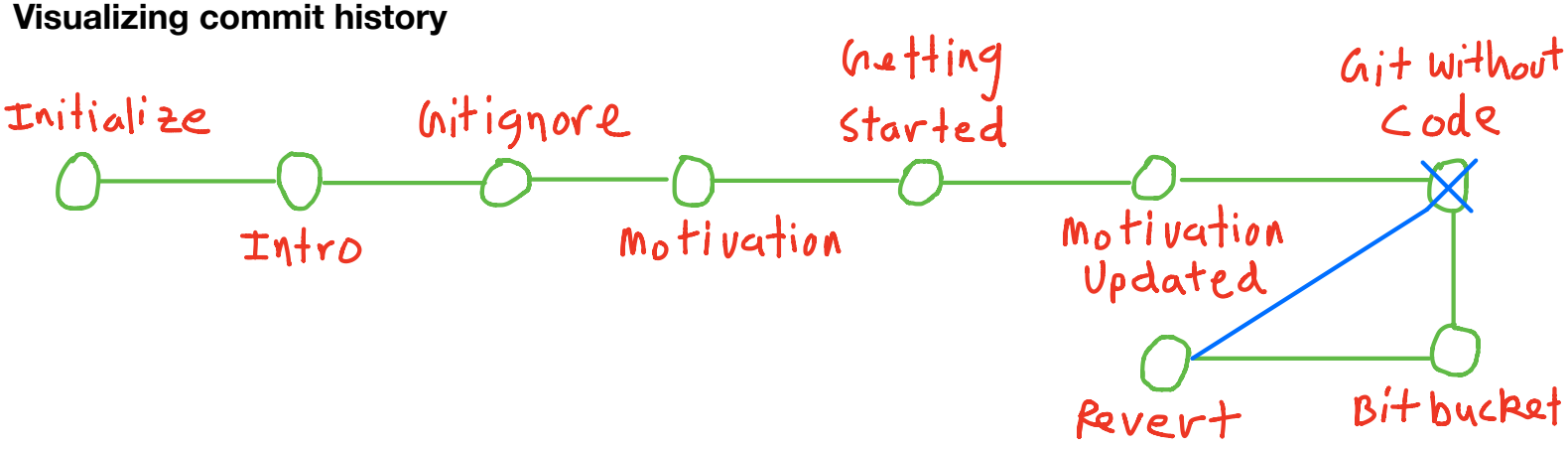
```
\section{Using Git without any code}
You don't necessarily need to be an expert in the Git terminal commands to do version control for your projects. There are graphical user interfaces available for Git that you can use to add commits, push to remote repository, create branches, etc using just a few clicks. The following Git GUI are fairly popular 1) Sourcetree, 2) GitKraken, and 3) GitHub desktop.

\section{Using Git with Bitbucket}
Bitbucket allows you to host your local repository on a online collaborative platform.

\end{document}
```

Revert all the edits made to the most recent "Using Git with Bitbucket" section

- Suppose we don't like the edits made in this new section, and would like to revert back to a previous commit without this section.
- `git revert` *commit_id*
- You should notice the TeX file remove the "Using Git with Bitbucket" section entirely
- `git log --oneline`

```
Hammads-MacBook-Air:git TeX hammadshaikh$ git revert aadee13
[master c0e33f6] Revert "Completed draft of Git with Bitbucket"
 1 file changed, 3 deletions(-)
Hammads-MacBook-Air:git TeX hammadshaikh$ git log --oneline
c0e33f6 Revert "Completed draft of Git with Bitbucket"
aadee13 Completed draft of Git with Bitbucket
```

Visualizing commit history



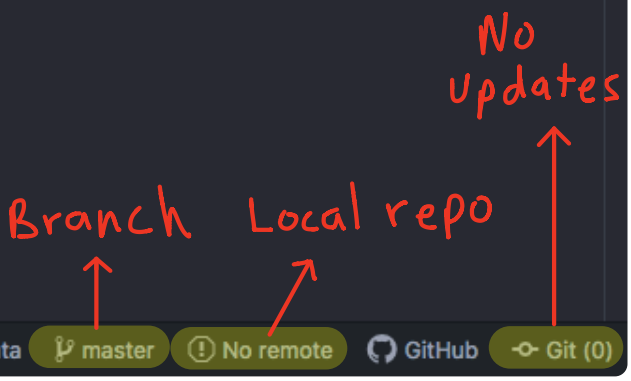
Example of using Git locally with Stata and Atom

- Please read "Configuring Git Locally" section above before going through this example

Beginning wage_determinants.do script for "master" branch

- `git add .`
- `git commit -m "Determinants of wages"`

```
/*

Purpose: Examine the determinants of wages using the NLSY data

*/

* Setup
clear all
set more off

* Open Data
sysuse nlsw88.dta

* Summary statistics
sum wage tenure ttl_exp union collgrad
```



Create "figure" branch to include Figures

- `git branch figure`
- `git checkout figure`
- `git add .`
- `git commit -m "Scatter of marital status vs wage"`

```
* Summary statistics
sum wage tenure ttl_exp union collgrad

* Scatter Plot: Wage and tenure
twoway (scatter wage tenure)
```

LF UTF-8 Stata figure No remote GitHub Git (0)

Create "regression" branch to include Tables

- `git branch regression`
- `git checkout regression`
- `git add .`
- `git commit -m "Wage determinant regressions"`

```
* Summary statistics
sum wage tenure ttl_exp union collgrad

* Wage and tenure
reg wage tenure

* Wage and all other factors
reg wage tenure ttl_exp union collgrad
```

LF UTF-8 Stata regression No remote GitHub Git (0)

Merge "master" with "figure" branch (no merge conflict since figure extends from master)

- `git checkout master`
- `git merge figure`
- `git add .`
- `git commit -m "Include scatter plot from merge"`

```
14  * Summary statistics
15  sum wage tenure ttl_exp union collgrad
16
17  * Scatter Plot: Wage and tenure
18  twoway (scatter wage tenure)
```

LF UTF-8 Stata master No remote GitHub Git (0)

Merge "master" with "regression" branch (merge conflict since new commit to master)

- `git merge regression`

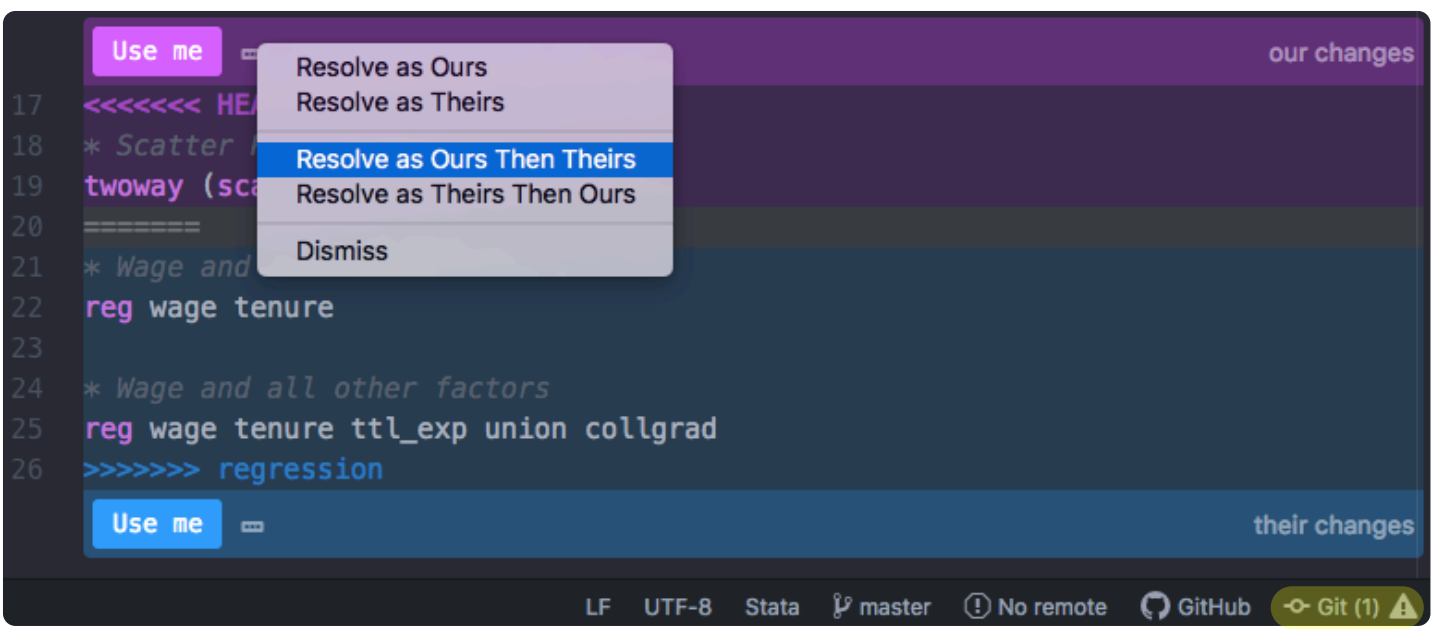
```
14  * Summary statistics
15  sum wage tenure ttl_exp union collgrad
16
    Use me ... our changes
17  <<<<<<< HEAD
18  * Scatter Plot: Wage and tenure
19  twoway (scatter wage tenure)
20  =======
21  * Wage and tenure
22  reg wage tenure
23
24  * Wage and all other factors
25  reg wage tenure ttl_exp union collgrad
26  >>>>>>> regression
    Use me ... their changes
```

Need to resolve merge conflict

LF UTF-8 Stata master No remote GitHub Git (1)

Resolve merge conflict by keeping both changes

- Merge conflict occurs since master branch was changed after "regression" branch was created



- To resolve the conflict, we need to decide what combination of changes to keep.

Stata do file view of the current "master" branch with unresolved conflict

- If you are not using atom, the merge conflict on the Stata editor is shown below
- Just remove the highlighted lines below to perform the "keep both" merge

```
17  <<<<<<< HEAD
18  * Scatter Plot: Wage and tenure
19  twoway (scatter wage tenure)
20  =======
21  * Wage and tenure
22  reg wage tenure
23
24  * Wage and all other factors
25  reg wage tenure ttl_exp union collgrad
26  >>>>>>> regression
```

Resolve merge conflict and keep both changes

Commit the finalized script after the merge

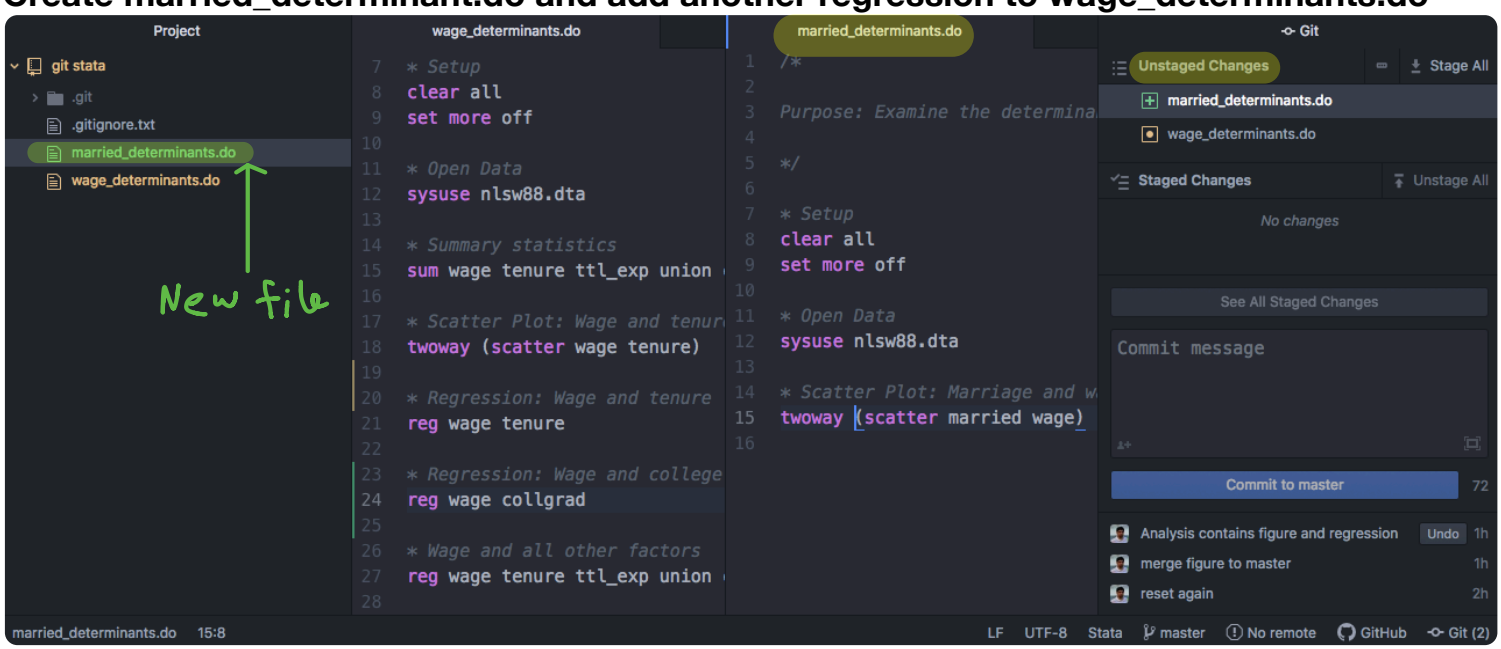
- `git add .`
- `git commit -m "Keep both reg and figure"`

```
14  * Summary statistics
15  sum wage tenure ttl_exp union collgrad
16
17  * Scatter Plot: Wage and tenure
18  twoway (scatter wage tenure)
19  * Wage and tenure
20  reg wage tenure
21
22  * Wage and all other factors
23  reg wage tenure ttl_exp union collgrad
```

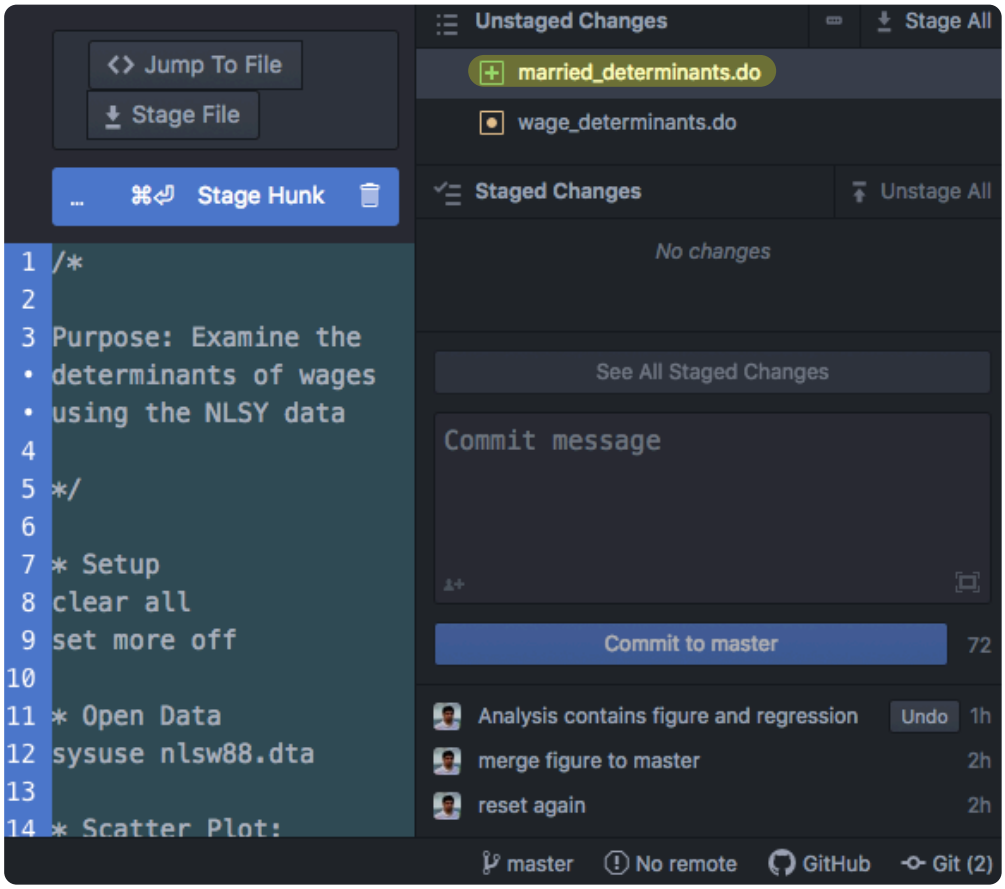
LF UTF-8 Stata master No remote GitHub Git (0)

Let's try Git by using the user interface from Atom rather than terminal

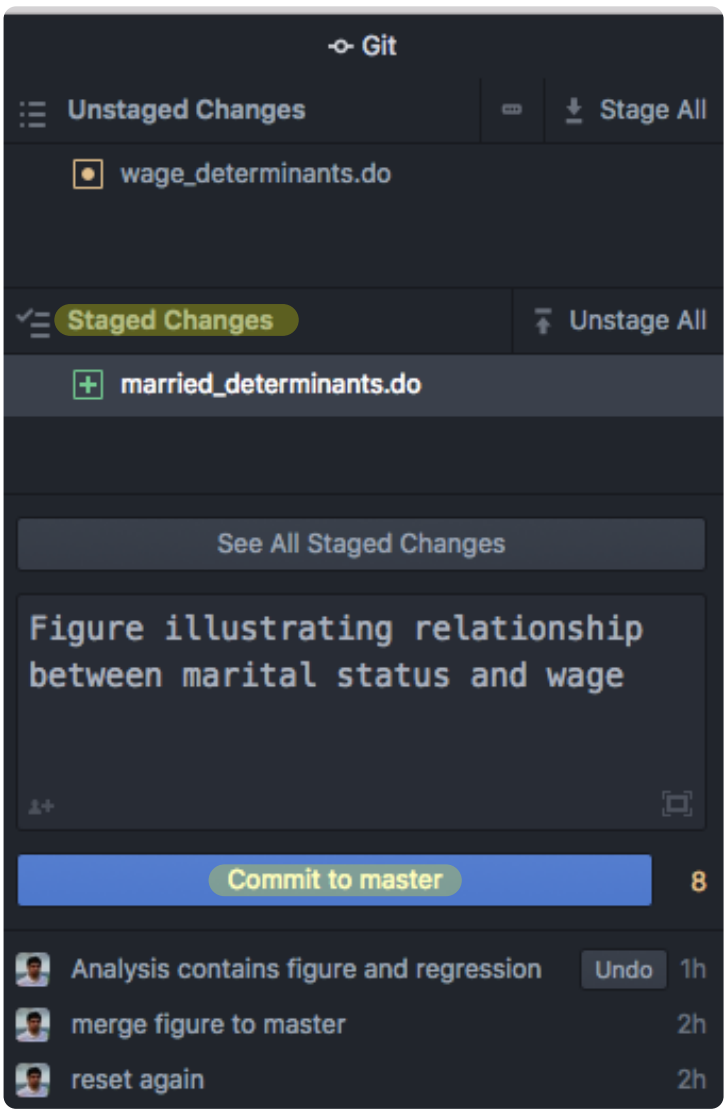
Create married_determinant.do and add another regression to wage_determinants.do

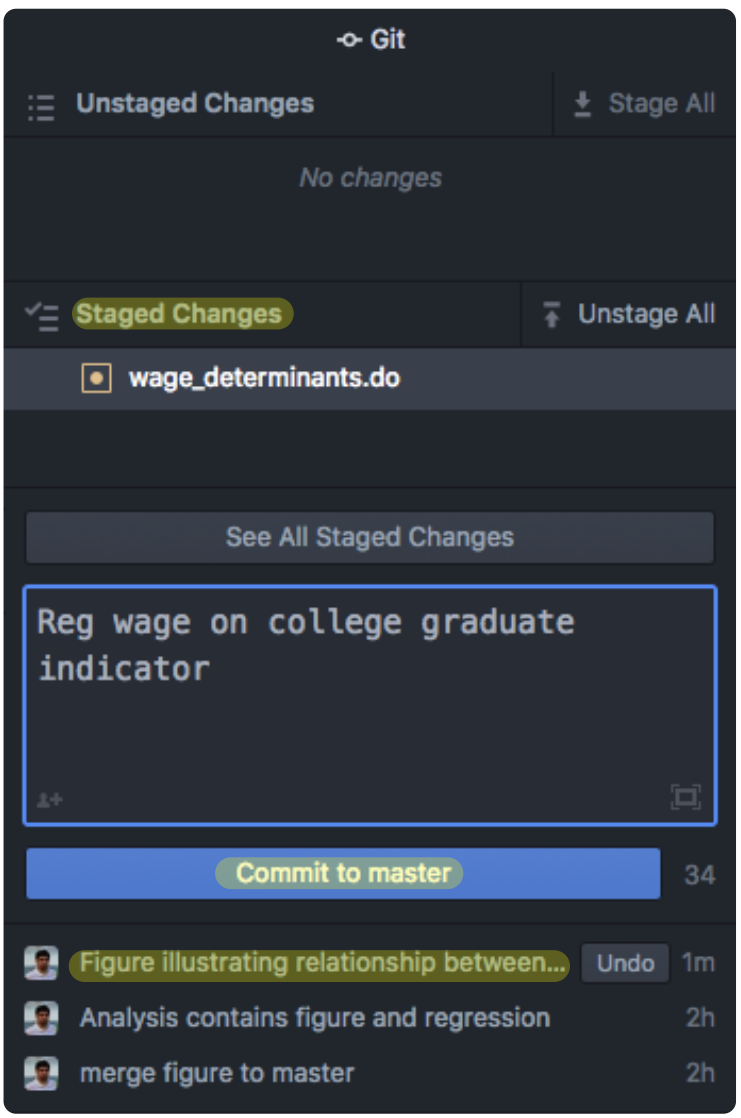


Double Click married_determinants.do --> Click "Stage File" for married_determinants.do

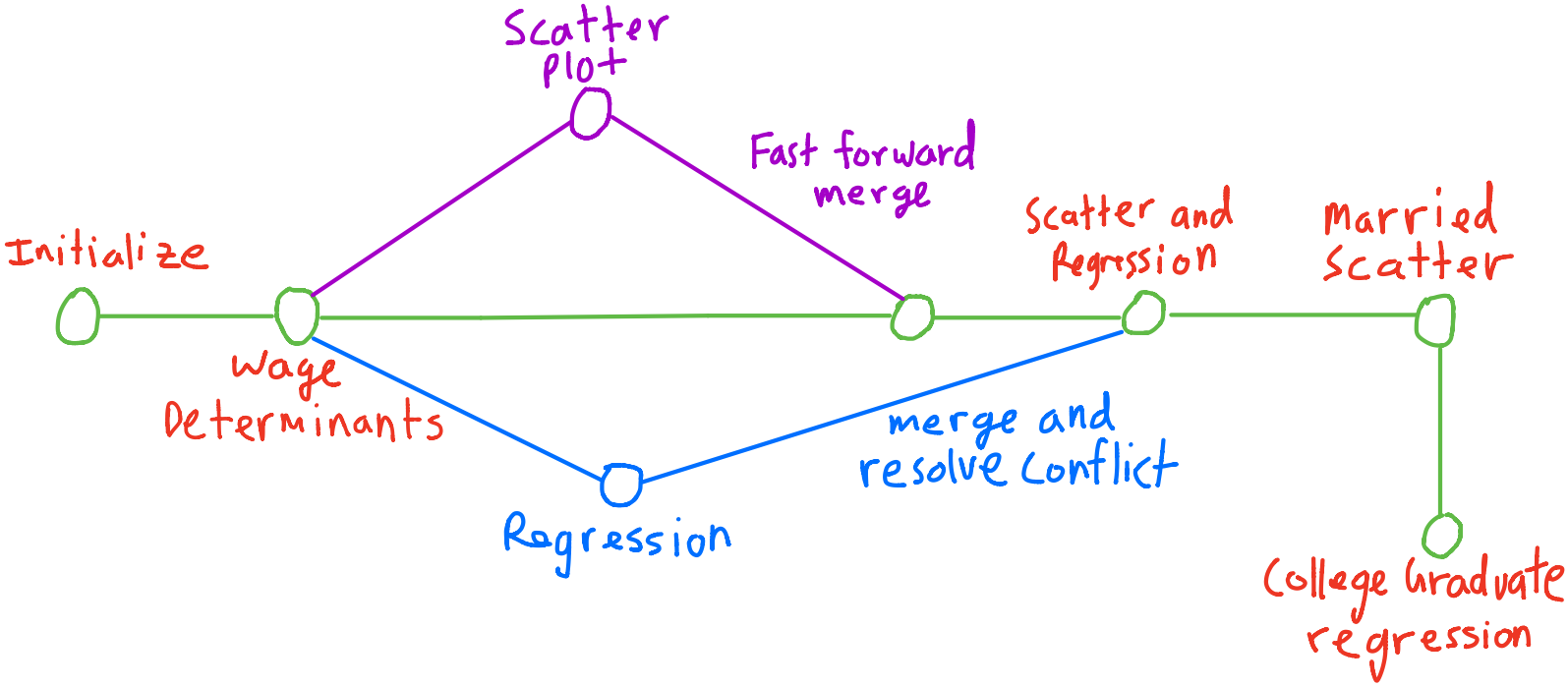


Write commit message --> Click "Commit to master" --> repeat for wage_determinants





Visualizing commit history

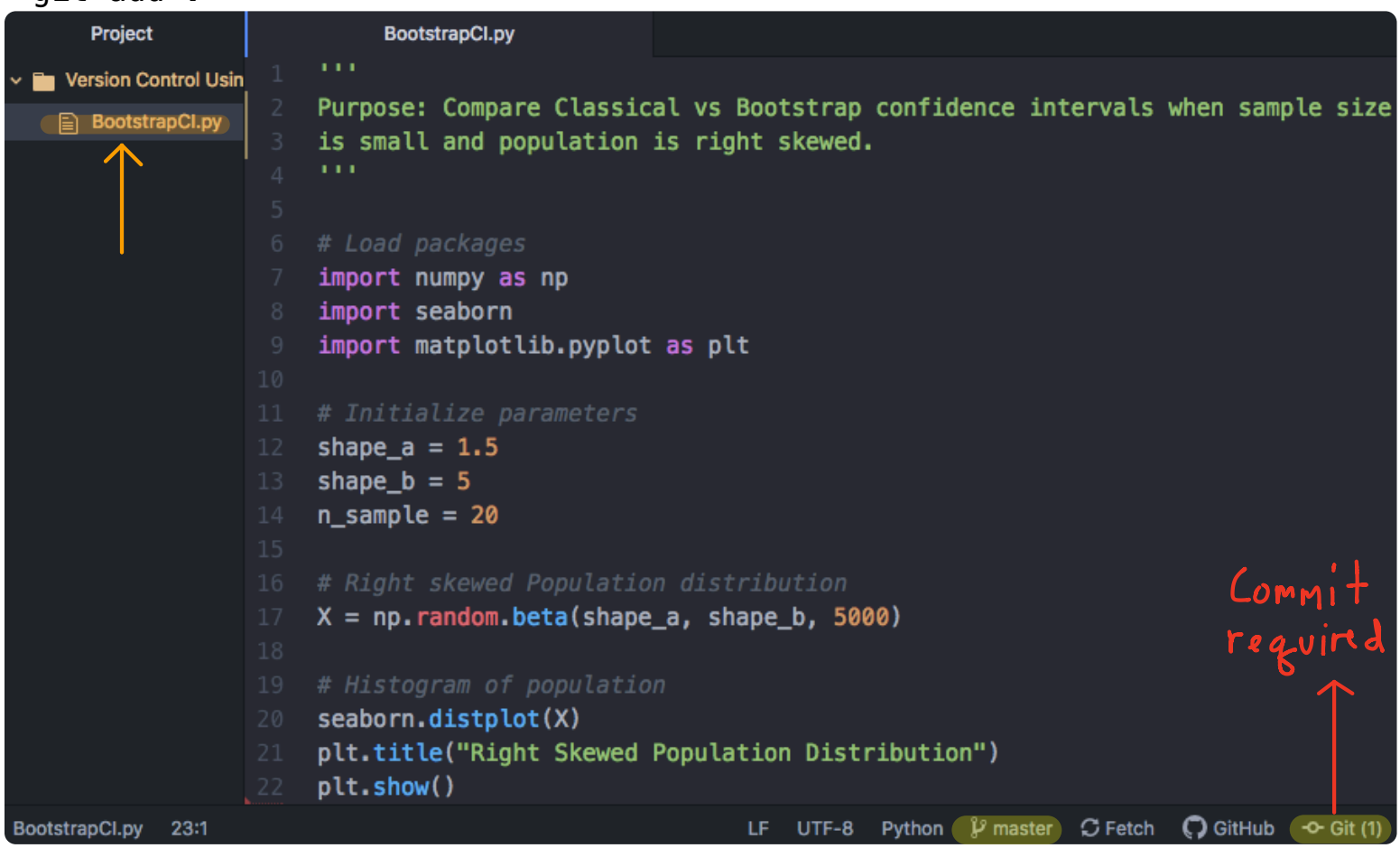


Example of using Git Remotely on GitHub with Python

- Please see the "Connecting Git to a Remote Repository" section before this example.
- Suppose there are two researchers Hammad and Daniel. Their goal is to construct the a confidence interval for the population mean.

Right skewed population distribution

- `git status`
- `git add .`



- `git commit -m "Define right skewed population distribution"`

- Hammad believes that they should use the classical 95% student t CI which assumes the sample mean is normally distributed.

Compute student t 95% confidence intervals

- `git add .`

```
30  ## Standard 95% CI
31  # CI components
32  sample_mean = np.mean(data)
33  sample_std = np.std(data)
34  margin_error = stats.t.ppf(1-alpha/2, df)*(sample_std/np.sqrt(n_sample))
35
36  # Lower and upper bounds
37  CI_lower = sample_mean - margin_error
38  CI_upper = sample_mean + margin_error
39  print("The 95% CI is: [" + str(CI_lower) + "," + str(CI_upper) + "]")
40
```

LF UTF-8 Python master Fetch GitHub Git (1)

- `git commit -m "Includes 95% student t CI"`
- Daniel is a bit concerned the CLT may not imply in a right skewed population with a small sample size. So he create a "bootstrap" branch to do his own investigation.
- `git branch bootstrap`
- `git checkout bootstrap`
- `git add .`
- `git commit -m "Compute 95% student t CI coverage prob"`
- `git add .`

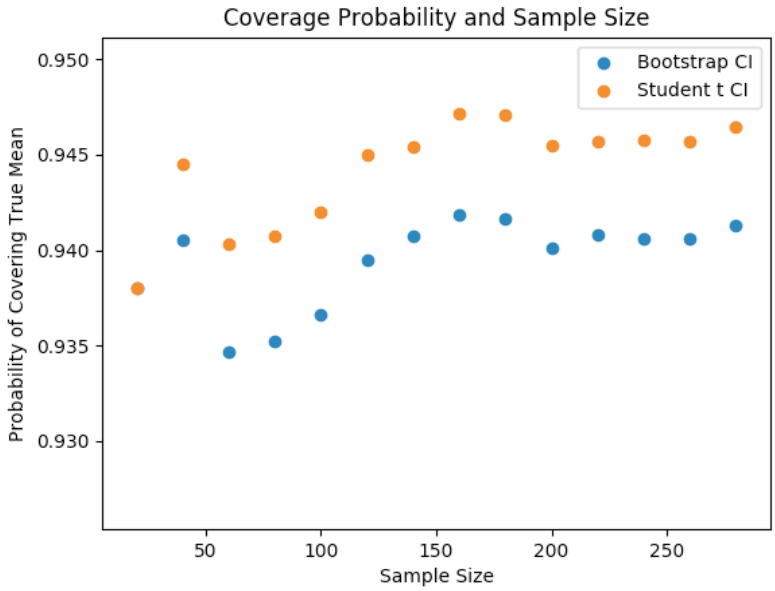
```
34  for sim in range(n_sim):
35
36      # Draw a sample from Population
37      data = np.random.beta(shape_a, shape_b, n_sample)
38
39      ## Standard 95% CI
40      # CI components
41      sample_mean = np.mean(data)
42      sample_std = np.std(data)
43      margin_error = stats.t.ppf(1-alpha/2, df)*(sample_std/np.sqrt(n_sample))
44
45      # Lower and upper bounds
46      CI_lower = sample_mean - margin_error
47      CI_upper = sample_mean + margin_error
48      #print("The 95% CI is: [" + str(CI_lower) + "," + str(CI_upper) + "]")
49
50      # Check whether CI covers true population mean
51      t_coverage.append(pop_mean <= CI_upper and pop_mean >= CI_lower)
52
53  # Print coverage probability for student t 95% CI
54  t_coverage_prob = np.mean(t_coverage)
55  print("Student t 95% CI coverage probability: " + str(t_coverage_prob))
```

Not Master

LF UTF-8 Python bootstrap Publish GitHub Git (1)

- `git commit -m "Compare 95% student t and bootstrap CI coverage prob"`
- Daniel finds in small sample size the student t 95% CI still seems appropriate as the coverage probability is still fairly close to 95%.
- `git add .`

- `git commit -m "Figure Sample Size and stud t CI Coverage Prob"`

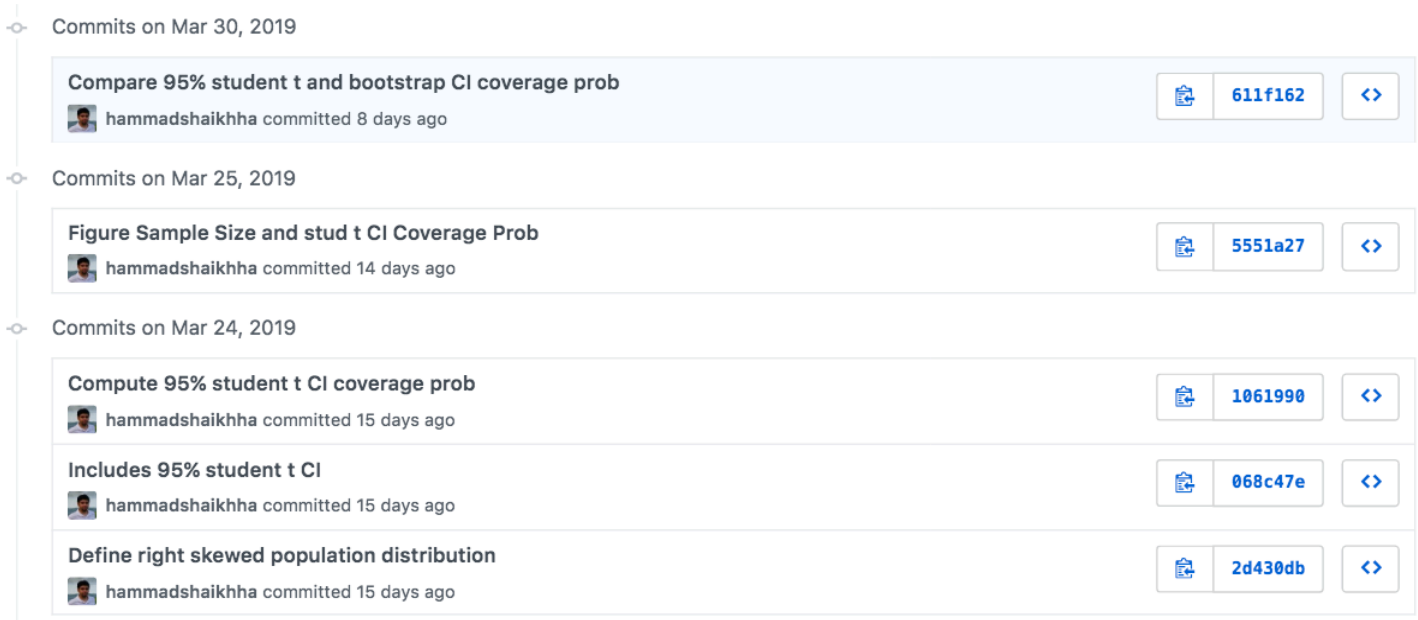


- After seeing the evidence, Hammad believes it is a good idea to include the above figure to justify the use of the student t CI. Hammad and asks Daniel to merge his branch to the master.
- `git checkout master`
- `git merge bootstrap`
- There will no merge conflicts since no commits were created on the master after the branch was created, this is known as a "fast forward merge"
- Hammad and Daniel both agree that their code is suitable for their current object, so tag it as version 1.
- `git tag -a v1 -m "version 1 - student t and bootstrap CI"`
- `git log --oneline`

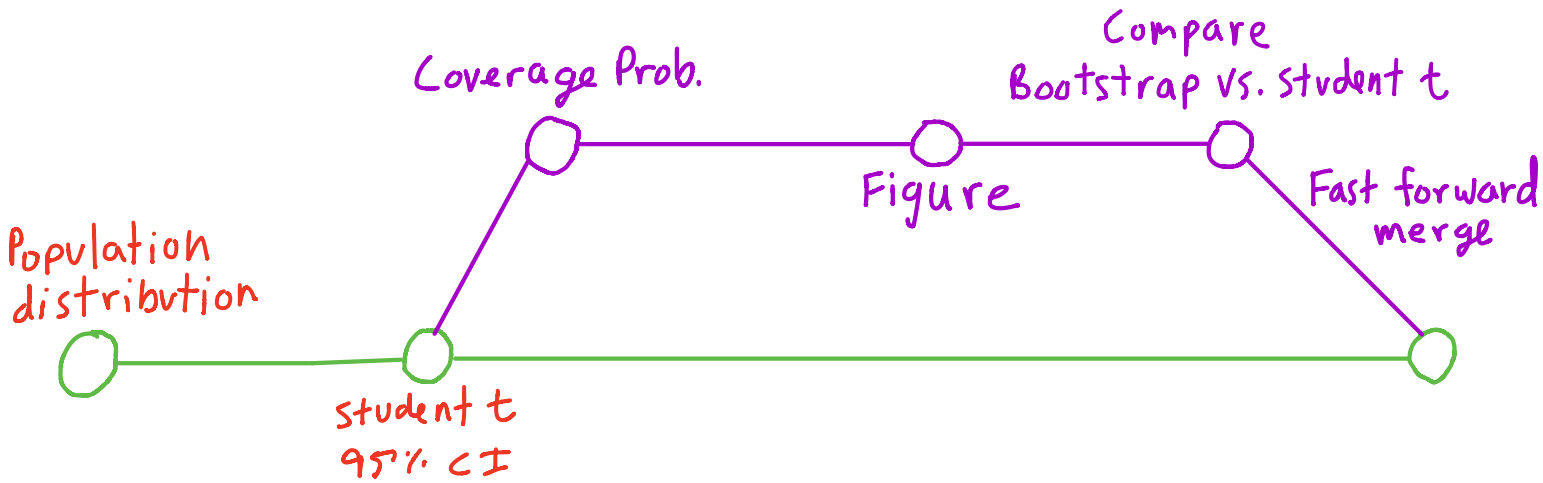
```
Hammads-MacBook-Air:Version Control Using Git hammadshaikh$ git log --oneline
611f162 Compare 95% student t and bootstrap CI coverage prob
5551a27 Figure Sample Size and stud t CI Coverage Prob
1061990 Compute 95% student t CI coverage prob
068c47e Includes 95% student t CI
2d430db Define right skewed population distribution
```

Push the finalized script to GitHub

- `git push origin`
- Note the pushes to the remote repository were made over time in the history below.



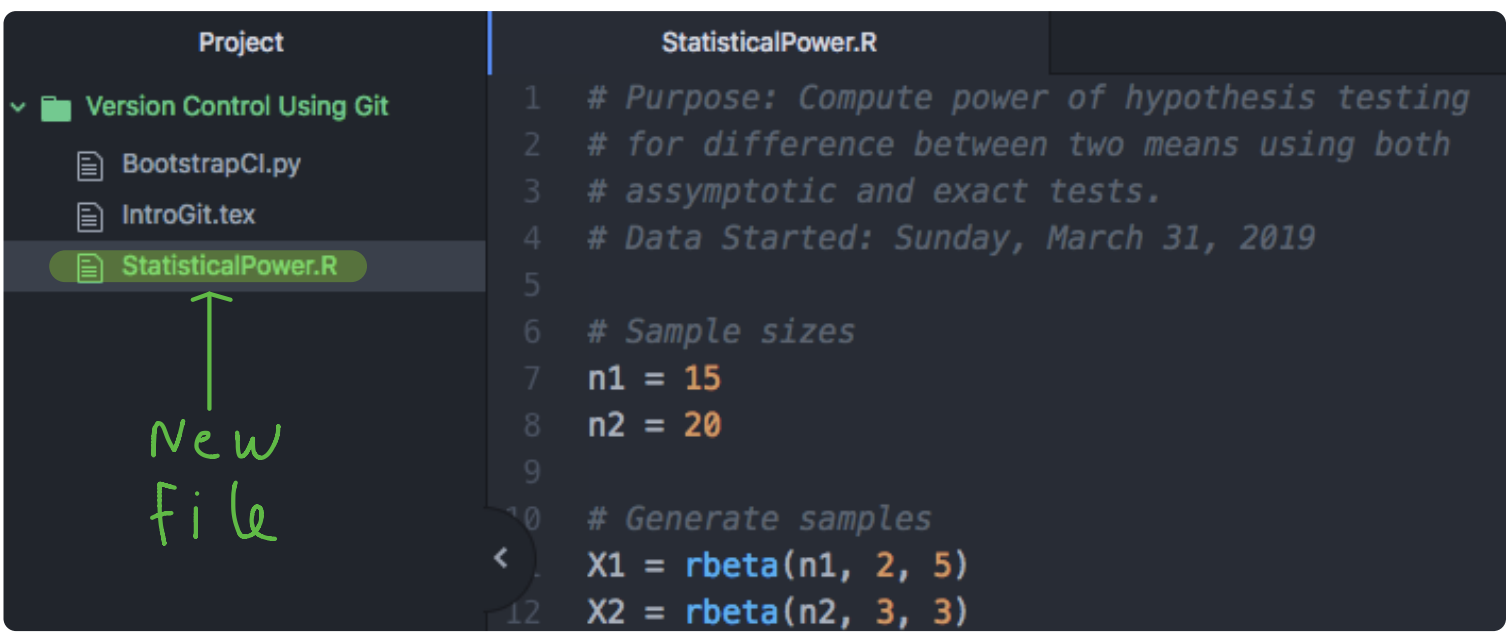
Visualizing commit history



- Note that since there were no commits to the master branch after the bootstrap branch was created, there will be no merge conflict when merging the master branch with the bootstrap branch (fast forward merge).

Example of using Git Remotely on GitHub with R

- Please see the "Connecting Git to a Remote Repository" section before this example.
- Suppose two researchers Hammad and Catherine want to compute the power of a hypothesis test for comparing two population means.
- `git status`
- `git add .`
- `git commit -m "Generate two samples from different beta distributions"`

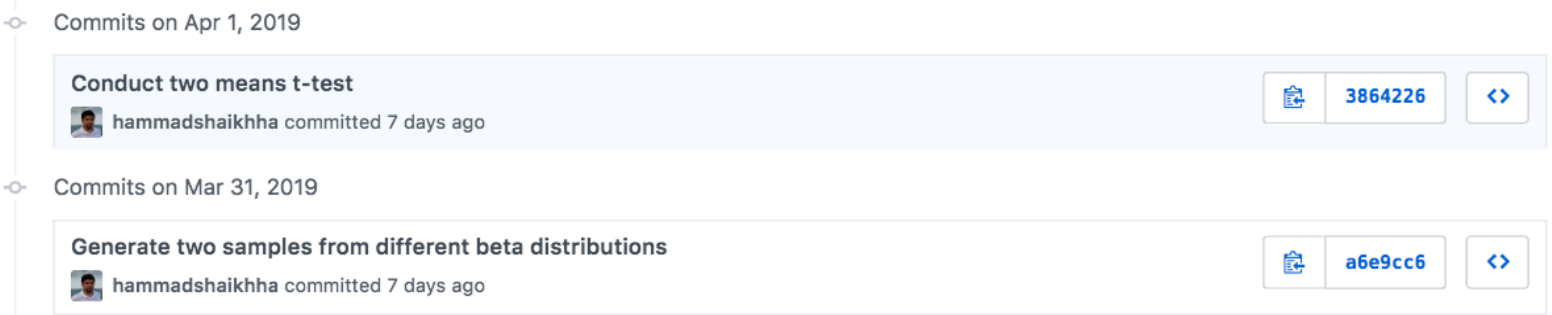


Test for statistical significance in the difference between the two means

- Hammad computes hypothesis test using rejection region approach.
- `git add .`
- `git commit -m "Conduct two means t-test`

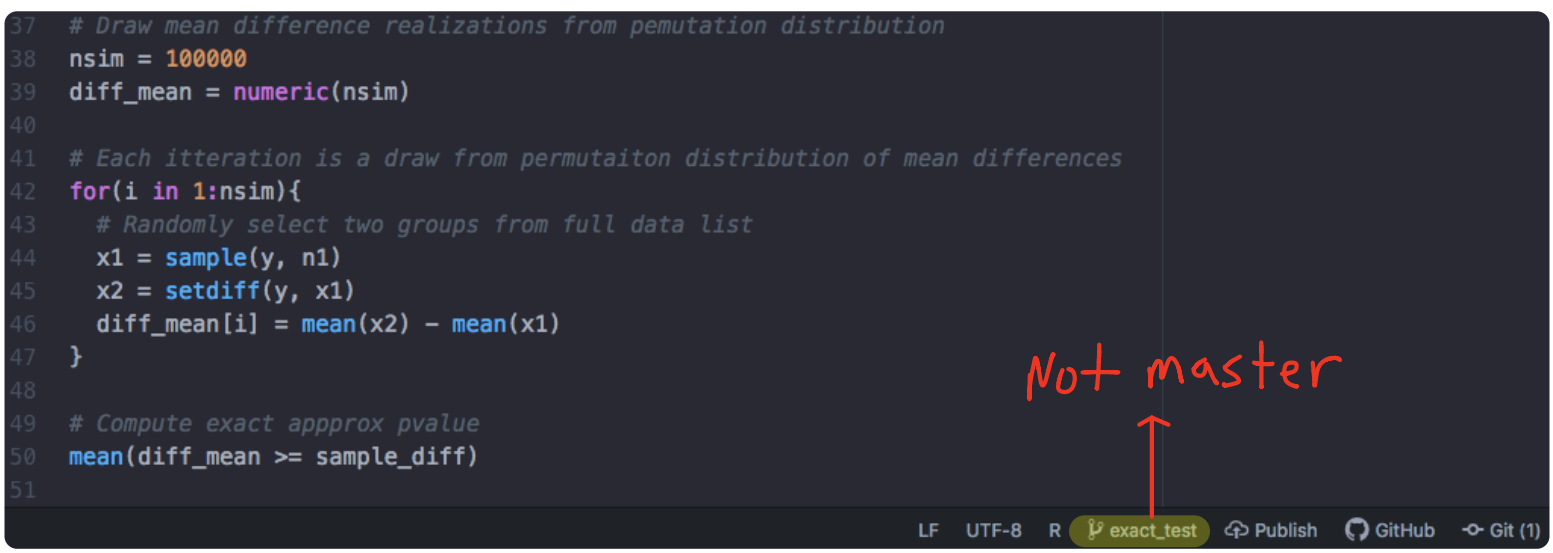
Push commit to remote repository

- `git push origin`

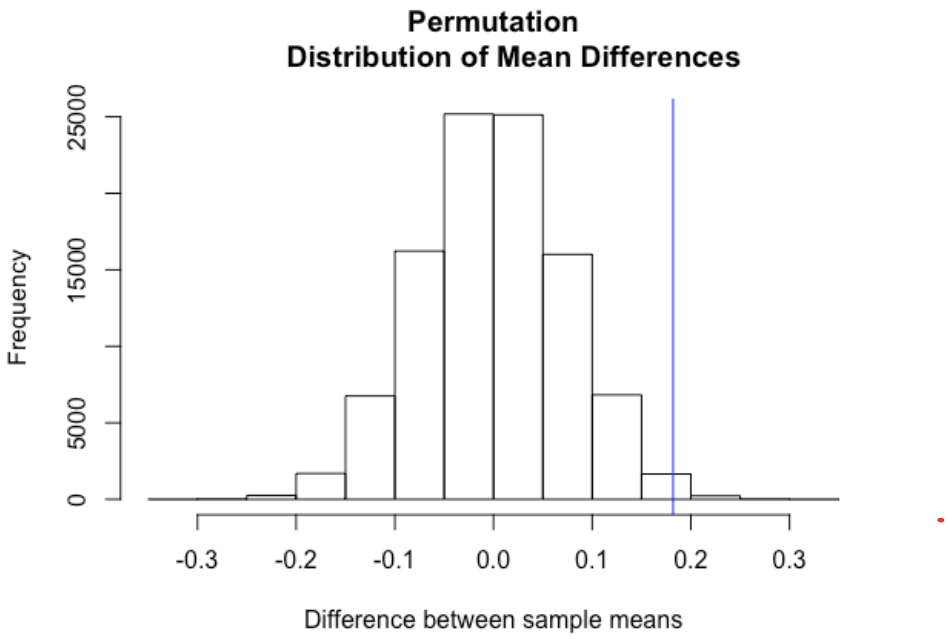


- Catherine is not too confident in the hypothesis test since the sample sizes are small. She makes an "exact_test" branch and does her own investigation using randomized inference.
- `git branch exact_test`
- `git checkout exact_test`

Compute approximate of exact pvalue



- After Catherine's branch, Hammad continues to work on the masters and compute the p-value using the typical approach.
- `git checkout master`
- Catherine informs Hammad she has set a seed, Hammad should try to use the same seed. To see Catherine's branch, he needs to stash his changes first.
- `git stash`
- `git checkout master`
- `git checkout exact_test`
- `git stash pop`
- `git add .`
- `git commit –m "Compute approx exact pvalue"`



- Hammad think its a good idea to report both the typical and approximated exact pvalue, so he recommends to merge Catherine's branch into master.
- `git checkout master`

Check difference between master and exact_test branch before the merge

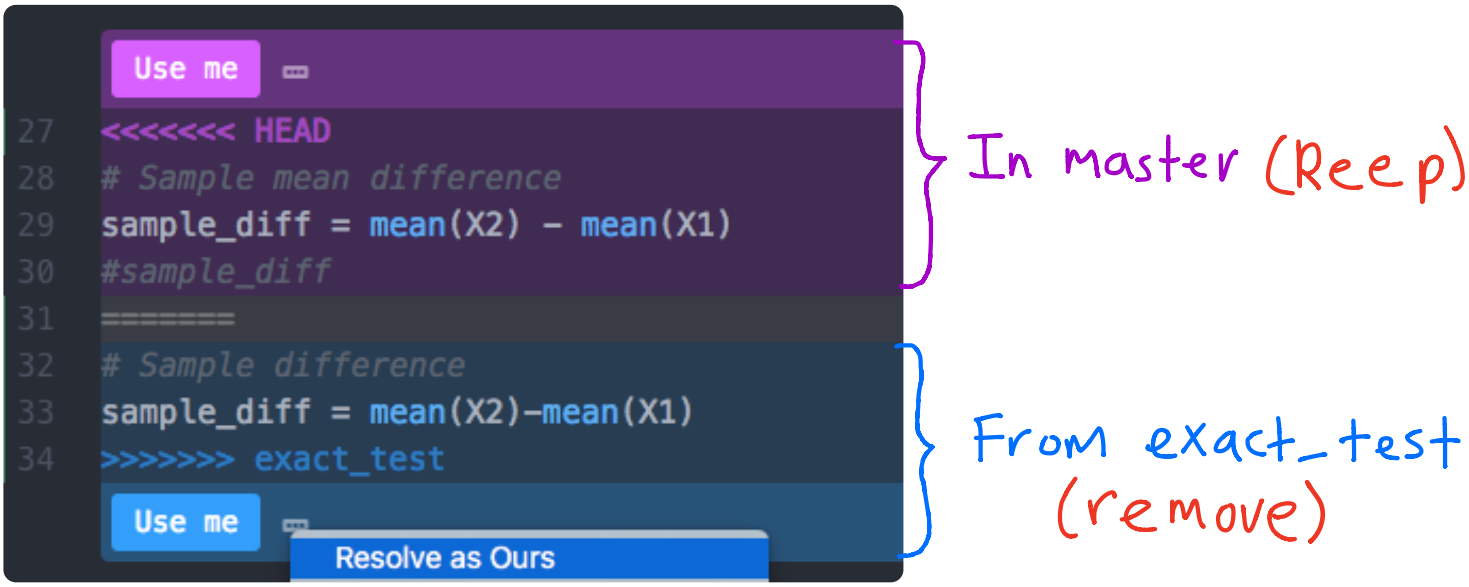
- `git difftool origin/master exact_test`

```r
# Seed
set.seed(21283)

# Generate samples
X1 = rbeta(n1, 2, 5)
X2 = rbeta(n2, 3, 3)



# Degrees of freedom
df = n1 + n2 - 2

# Pooled variance
pool_var = ((n1-1)*var(X1) + (n2-1)*var(X2))/(n1 + n2 - 2)

# Sample mean difference
sample_diff = mean(X2) - mean(X1)
#sample_diff

# t - statistics and critical vaue
tstat = (sample_diff)/sqrt(pool_var*(1/n1 + 1/n2))
tcv = abs(qt(alpha/2, df))

# Rejection region
reject = tstat > tcv

# Compute p-value

pvalue = pt(-tstat, df)*2
```

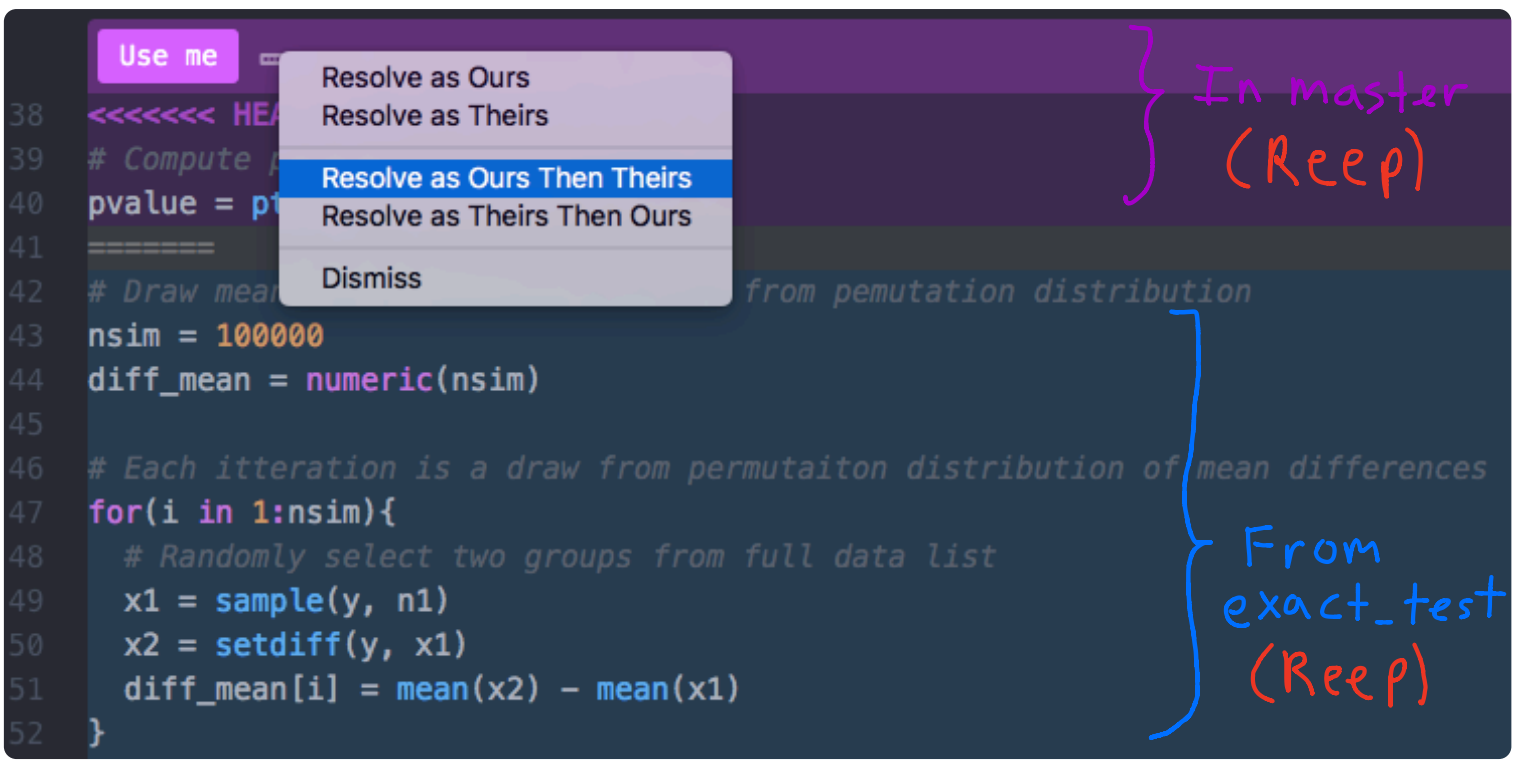
```r
# Seed
set.seed(21283)

# Generate samples
X1 = rbeta(n1, 2, 5)
X2 = rbeta(n2, 3, 3)

# Store full data in vector
y = c(X1,X2)

# Degrees of freedom
df = n1 + n2 - 2

# Pooled variance
pool_var = ((n1-1)*var(X1) + (n2-1)*var(X2))/(n1 + n2 - 2)

# Sample difference
sample_diff = mean(X2)-mean(X1)


# t - statistics and critical vaue
tstat = (sample_diff)/sqrt(pool_var*(1/n1 + 1/n2))
tcv = abs(qt(alpha/2, df))

# Rejection region
reject = tstat > tcv

# Draw mean difference realizations from pemutation
distribution
nsim = 100000
diff_mean = numeric(nsim)

# Each itteration is a draw from permutaiton distribution of
mean differences
for(i in 1:nsim){
  # Randomly select two groups from full data list
  x1 = sample(y, n1)
  x2 = setdiff(y, x1)
  diff_mean[i] = mean(x2) - mean(x1)
}

# Compute exact appprox pvalue
mean(diff_mean >= sample_diff)
```

- `git merge exact_test`
- `git add .`
- `git commit -m "Merge exact pvalue approx to master"`

Resolve merge conflict by keeping changes in master



Resolve conflict by keeping both typical pvalue and approximating exact pvalue



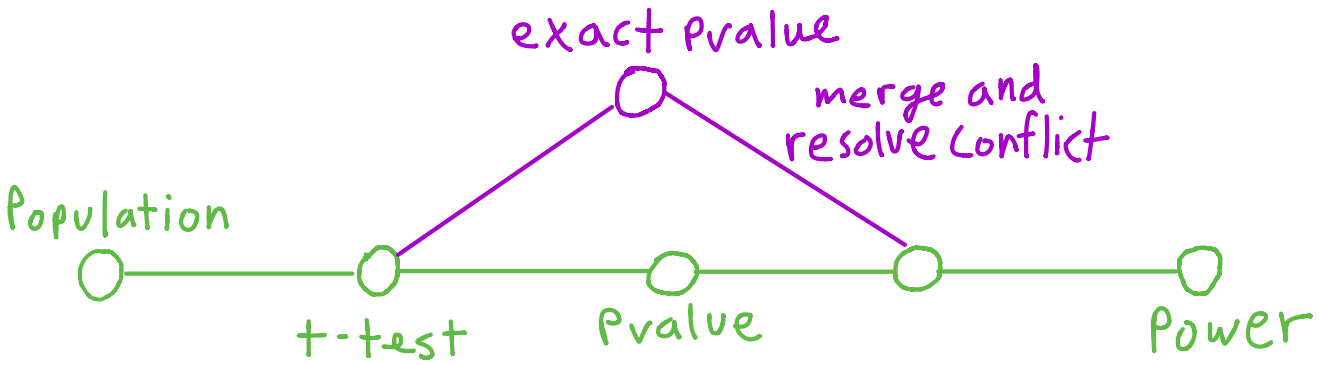
- Hammad completes the code to include power calculation. He calculated power by looking at the proportion of correctly rejected null H0 across simulations.
- `git add .`
- `git commit -m "Compute power using t test and exact method"`
- `git log --oneline`

```
[Hammads-MacBook-Air:Version Control Using Git hammadshaikh$ git log --oneline
e48824e Compute power using t test and exact method
3135299 Merge exact pvalue approx to master
fd43a3e Compute approx exact pvalue
7498d62 Compute pvalue using t distribution
3864226 Conduct two means t-test
a6e9cc6 Generate two samples from different beta distributions
```

Push commits on remote repository

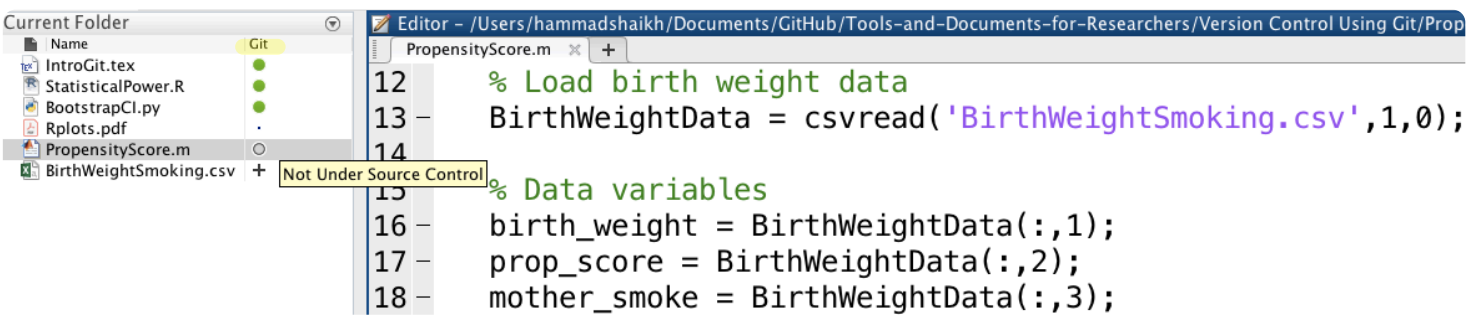
- `git push origin`

Visualizing commit history



Example of using Git Remotely on GitHub with Matlab

- Suppose Hammad and Scott are working on determining the relationship between a infants birthweight and the smoking status of the mother.



OLS regression of infant birth weight on mother smoking status

- `git add .`
- `git commit -m "OLS reg of birth weight on smoke status"`

```
21
22    % Descriptive analysis
23 -  X = [ones(nsize)  mother_smoke];
24 -  y = birth_weight;
25
26    % OLS regression
27 -  beta_hat = inv(X'*X)*X'*y;
```

Compute standard errors

- `git add .`
- `git commit -m "Compute homoscedastic standard errors"`

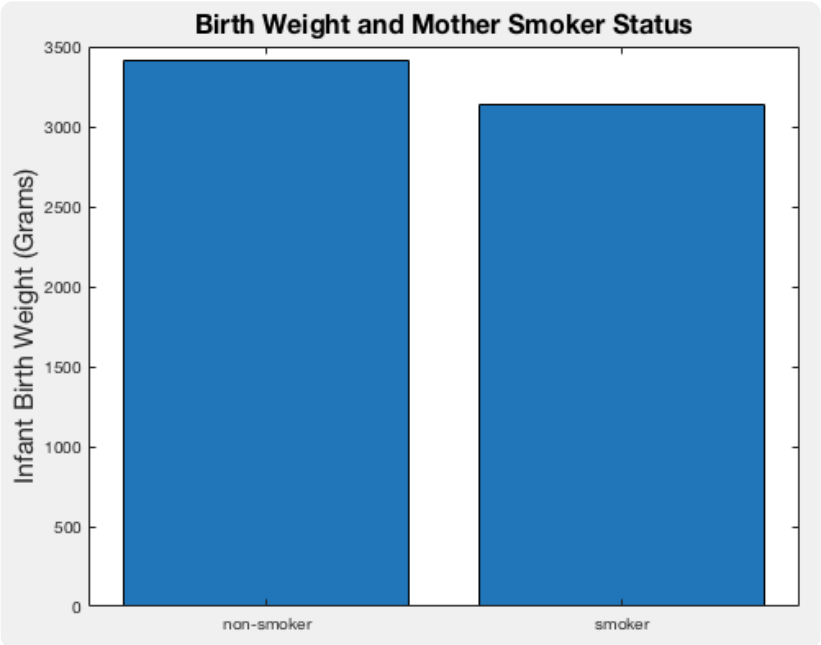
```
36    % Homoscedastic standard errors
37    % Generate residuals
38 -  e = y - X*beta_hat;
39
40    % Covariance matrix
41 -  sigma_sq = (1/(nsize(1)-2))*sum(e.^2);
42 -  VCov = sigma_sq*inv(X'*X);
43
44    % Standard errors
45 -  b0_se = sqrt(VCov(1,1));
46 -  b1_se = sqrt(VCov(2,2));
```

- `git push origin`

Compute homoscedastic standard errors
hammadshaikhha committed a day ago — ae004ba

OLS reg of birth weight on smoke status
hammadshaikhha committed a day ago — ee8bf47

- Hammad begins by conducting a descriptive analysis and checking the difference between baby birth weights of mothers who smoke compared to those that don't.



Compute weighted difference between means (note: incorrect implementation)

- `git add .`
- `git commit -m "Outcome weighted by inv prop score ATE"`

```
48      % Propensity score weighting
49 -    y_weighted_smoke = y(mother_smoke == 1)./prop_score(mother_smoke == 1);
50 -    y_weighted_notsmoke = y(mother_smoke == 0)./(1-prop_score(mother_smoke == 0));
51
52      % Compute ATE
53 -    ATE_prop_weight = mean(y_weighted_smoke) - mean(y_weighted_notsmoke);
```

- Hammad and Scott have a disagreement on how to implement propensity score weighting. Hammad wants to weight the outcome directly, whereas Scott prefers to weight the regression. Scott creates a separate branch for his own implementation.
- `git branch prop_score`
- `git checkout prop_score`

Inverse propensity score weighted OLS regression (note: correct implementation)

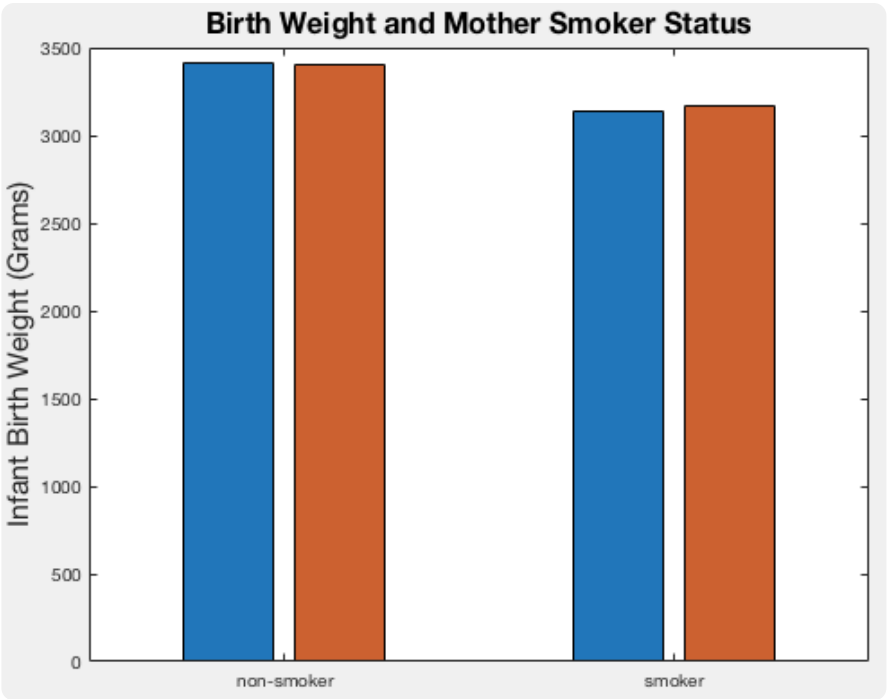
- `git add .`
- `git commit -m "Inverse propensity score weighted OLS"`

PropensityScore.m | OLSWeighted.m*

```
1   function S = OLSWeighted(birth_weight, X, inv_prob_weight, beta)
2
3       % Weighted sum of square errors
4 -     S = sum(inv_prob_weight.*((birth_weight - X*beta).^2));
5
6 -   end
```

Update histogram showing both OLS and inv prop. weighted results

- `git commit -m "Update histogram with IPW results"`



Outcome weighted OLS in master branch (note: incorrect implementation)

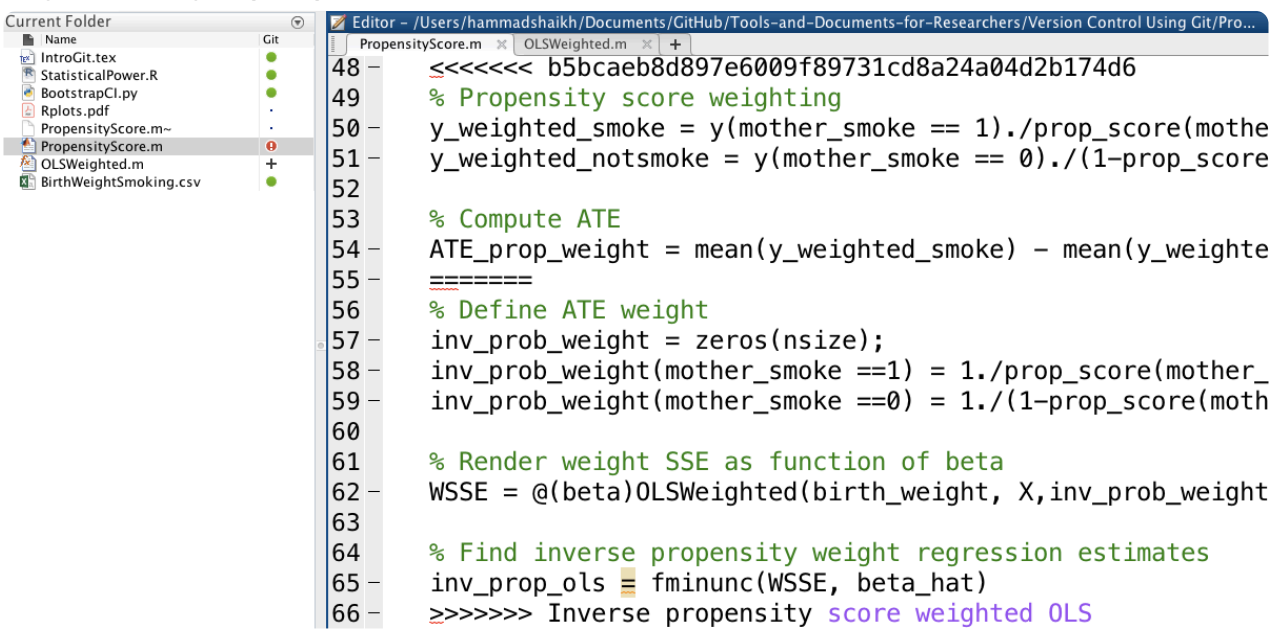
- `git checkout master`
- `git add .`
- `git commit -m "OLS with IPW outcome"`

```
55      % OLS with weighted outcome
56 -    y_weighted = zeros(nsize);
57 -    y_weighted(mother_smoke == 1) = y_weighted_smoke;
58 -    y_weighted(mother_smoke == 0) = y_weighted_notsmoke;
59 -    beta_hat = inv(X'*X)*X'*y_weighted;
```

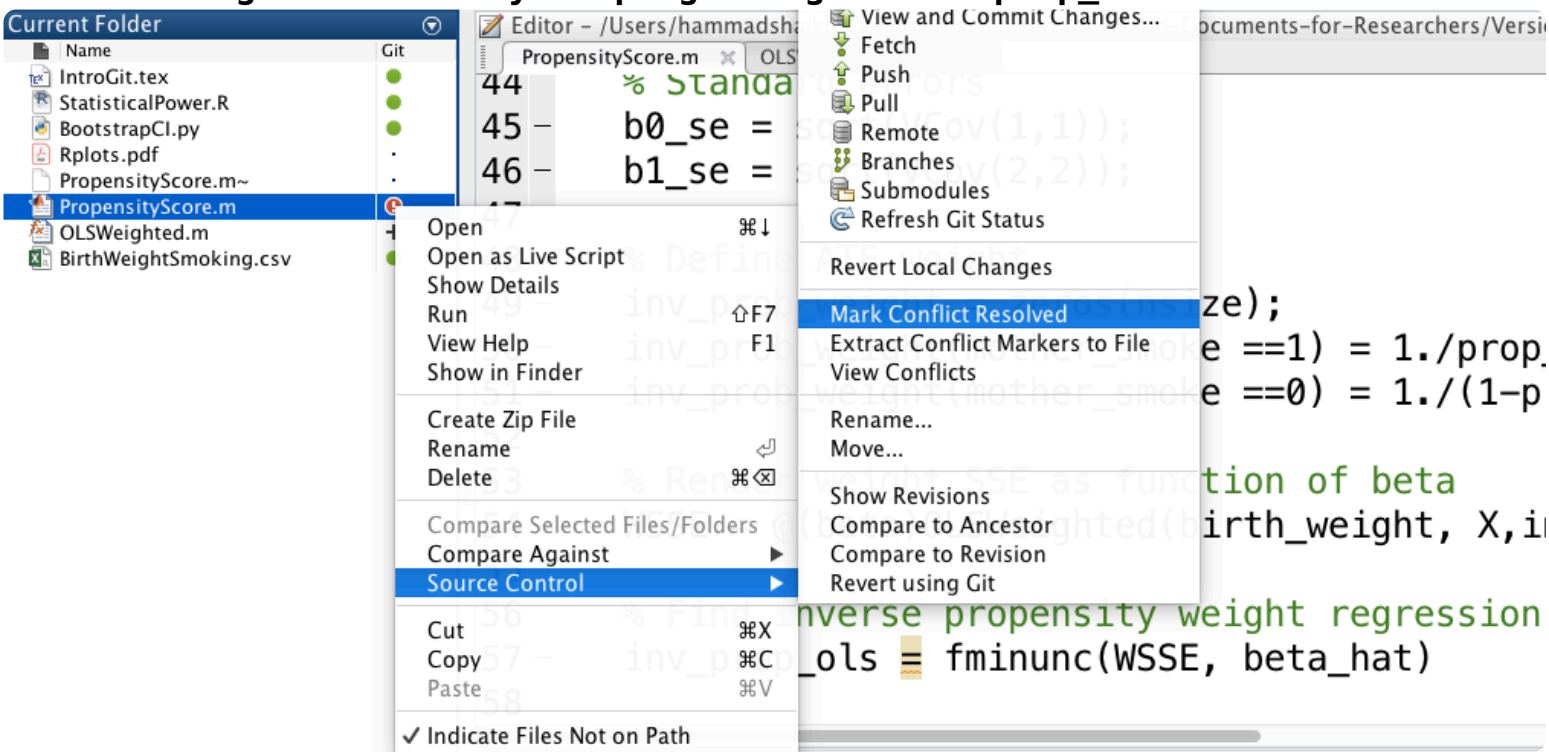
- Hammad weights the outcome directly and notices the sign changes, that is mothers who smoke have heavier infants than mothers who don't smoke. After some reading, Hammad realizes he has made a mistake. Hence Hammad merges master with prop_score branch.

Merge master with prop_score branch for correct implementation of IPW

- `git merge prop_score`



Resolve merge conflict by keeping changes from prop_score branch



- git add .
- git commit -m "Correct IPW estimates from merge"

Compute hetroscedastic robust standard errors for the IPW estimates

- git add .
- git commit -m "Compute hetroscedastic robust standard errors"

```
59      % Hetroscedastic robust standard errors
60      % Generate residuals
61 -    X_weighted = X.*sqrt(inv_prob_weight);
62 -    e_robust = y.*sqrt(inv_prob_weight) - X_weighted*inv_prop_ols;
63
64      % Hetro. robust cov-matrix
65 -    V_robust = inv(X_weighted'*X_weighted)*(X_weighted'*diag(e_robust.^2)
66
67      % Robust standard errors
68 -    b0_se_robust = sqrt(V_robust(1,1));
69 -    b1_se_robust = sqrt(V_robust(2,2));
```

- git log --oneline

```
Hammads-MacBook-Air:Version Control Using Git hammadshaikh$ git log --oneline
6a1bc0a Compute hetroscedastic robust standard errors
b050280 Correct IPW estimates from merge
85d1828 Update histogram with IPW results
2f46c6c Inverse propensity score weighted OLS
516b641 OLS with IPW outcome
b5bcaeb Outcome weighted by inv prop score ATE
ae004ba Compute homoscedastic standard errors
ee8bf47 OLS reg of birth weight on smoke status
```

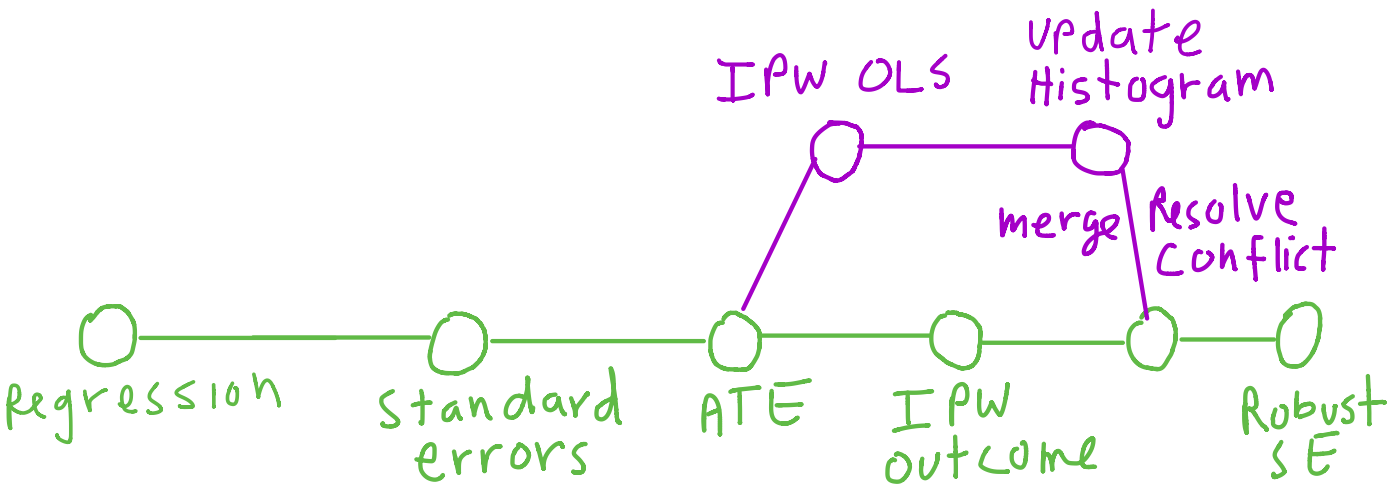
Push commits on remote repository

- git push origin

Commits on Apr 6, 2019

Correct IPW estimates from merge hammadshaikhha committed a day ago	b050280
Update histogram with IPW results hammadshaikhha committed a day ago	85d1828
Inverse propensity score weighted OLS hammadshaikhha committed a day ago	2f46c6c
OLS with IPW outcome hammadshaikhha committed a day ago	516b641
Outcome weighted by inv prop score ATE hammadshaikhha committed a day ago	b5bcaeb

Visualizing commit history



Conclusion

- Git is a version control program that is capable of tracking the evolution of script files over time.
  - Allows researchers to track the evolution of their project.
  - Works well with all the common programming languages (e.g. R, python, Matlab, Stata, etc)
  - Only a small increase in coding time required to perform version control.
    - Mainly need to commit major code changes
- Creating branches in Git allows you to experiment without affecting the primary code.
- Version control using Git allows us to view or revert to any old commit.
  - We can also view differences in a script file across two different commits.
- The merging feature allows multiple researchers to collaborate on the same script file and finalize a version which keeps some combination of suggested changes.
- Version control using Git can be done locally or on a online collaborative platform such as GitHub or Bitbucket.
  - Private repositories useful for researchers not wanting to make their code public.
    - Easy to publicize the code for replication purposes after research is published.
  - Provides a back up of all your script files
    - Available both locally and also on the remote server
- Git commands can be executed through terminal or some graphic user interface.

Resources for learning Git:

https://raw.githack.com/uo-ec607/lectures/master/02-git/02-Git.html#9
https://web.stanford.edu/~gentzkow/research/CodeAndData.pdf
https://www.sas.upenn.edu/~jesusfv/Chapter_HPC_5_Git.pdf
https://www.atlassian.com/git/tutorials/learn-git-with-bitbucket-cloud
https://www.frankpinter.com/git/
https://michaelstepner.com/blog/git-vs-dropbox/
https://www.youtube.com/watch?v=Loav1kbA640
https://www.youtube.com/watch?v=MFtsLRphqDM&list=PL4cUxeGkcC9goXbgTDQ0n_4TBzOO0ocPR&index=2