

Quantitative Economics with Julia

Jesse Perla Thomas J. Sargent John Stachurski

[Home](#) [Python](#) [Julia](#) [PDF](#)

Total coverage 100%

[Org](#) • [Lectures](#) » [Julia 1.1.0](#) » Git, GitHub, and Version Control

jupyter notebook [download](#)

pdf [download](#)

execution test not available

▼ [How to read this lecture...](#)

Git, GitHub, and Version Control

- [Setup](#)
- [Basic Objects](#)
- [Individual Workflow](#)
- [Collaborative Work](#)
- [Collaboration via Pull Request](#)
- [Additional Resources and Troubleshooting](#)
- [Exercises](#)

Co-authored with Arnav Sood

An essential part of modern software engineering is using version control

We use version control because

- Not all iterations on a file are perfect, and you may want to revert changes
- We want to be able to see who has changed what and how
- We want a uniform version scheme to do this between people and machines
- Concurrent editing on code is necessary for collaboration
- Version control is an essential part of creating reproducible research

In this lecture, we'll discuss how to use Git and GitHub

Setup

1. Make sure you create an account on [GitHub.com](#)
 - If you are a student, be sure to use the GitHub [Student Developer Pack](#)
 - Otherwise, see if you qualify for a free [Non-Profit/Academic Plan](#)
 - These come with things like unlimited private repositories, testing support, etc.
2. Install `git` and the GitHub Desktop application
 1. Install `git`

2. Install the [GitHub Desktop](#) application
3. Optionally (but strongly recommended): On Windows, change the default line-ending by:
4. Opening a Windows/Powershell console, or the “Git Bash” installed in the previous step
5. Running the following

```
git config --global core.eol lf
git config --global core.autocrlf false
```

Git vs. GitHub vs. GitHub Desktop

To understand the relationship

- Git is an infrastructure for versioning and merging files (it is not specific to GitHub and does not even require an online server)
- GitHub provides an online service to coordinate working with Git repositories, and adds some additional features for managing projects
- GitHub Desktop is just one of many GUI-based clients to make Git and GitHub easier to use

Later, you may find yourself using alternatives

- GitHub is the market leader for open source projects and Julia, but there are other options, e.g. [GitLab](#) and [Bitbucket](#)
- Instead of the GitHub Desktop, you may directly use the Git command line, [GitKraken](#), or use the Git functionality built into editors such as [Atom](#) or [VS Code](#)

Since these lecture notes are intended to provide a minimal path to using the technologies, here we will conflate the workflow of these distinct products

Basic Objects

Repositories

The fundamental object in GitHub is a repository (or “repo”) – this is the master directory for a project

One example of a repo is the QuantEcon [Expectations.jl](#) package

On the machine, a repo is a normal directory, along with a subdirectory called `.git` which contains the history of changes

Commits

GitHub stores history as a sequence of changes to text, called commits

[Here](#) is an example of a commit, which revises the style guide in a QuantEcon repo

In particular, commits have the following features

- An ID (formally, an “SHA-1 hash”)
- Content (i.e., a before and after state)
- Metadata (author, timestamp, commit message, etc.)

Note: It’s crucial to remember that what’s stored in a commit is only the actual changes you make to text

This is a key reason why git can store long and complicated histories without consuming massive amounts of memory

Common Files

In addition, each GitHub repository typically comes with a few standard text files

- A `.gitignore` file, which lists files/extensions/directories that GitHub shouldn’t try to track (e.g., LaTeX compilation byproducts)

- A `README.md` file, which is a Markdown file which GitHub puts on the repository website
- A `LICENSE.txt` file, which describes the terms under which the repository's contents are made available

For an example of all three, see the [Expectations.jl](#) repo

Of these, the `README.md` is the most important, as GitHub will display it as Markdown when accessing the repository online

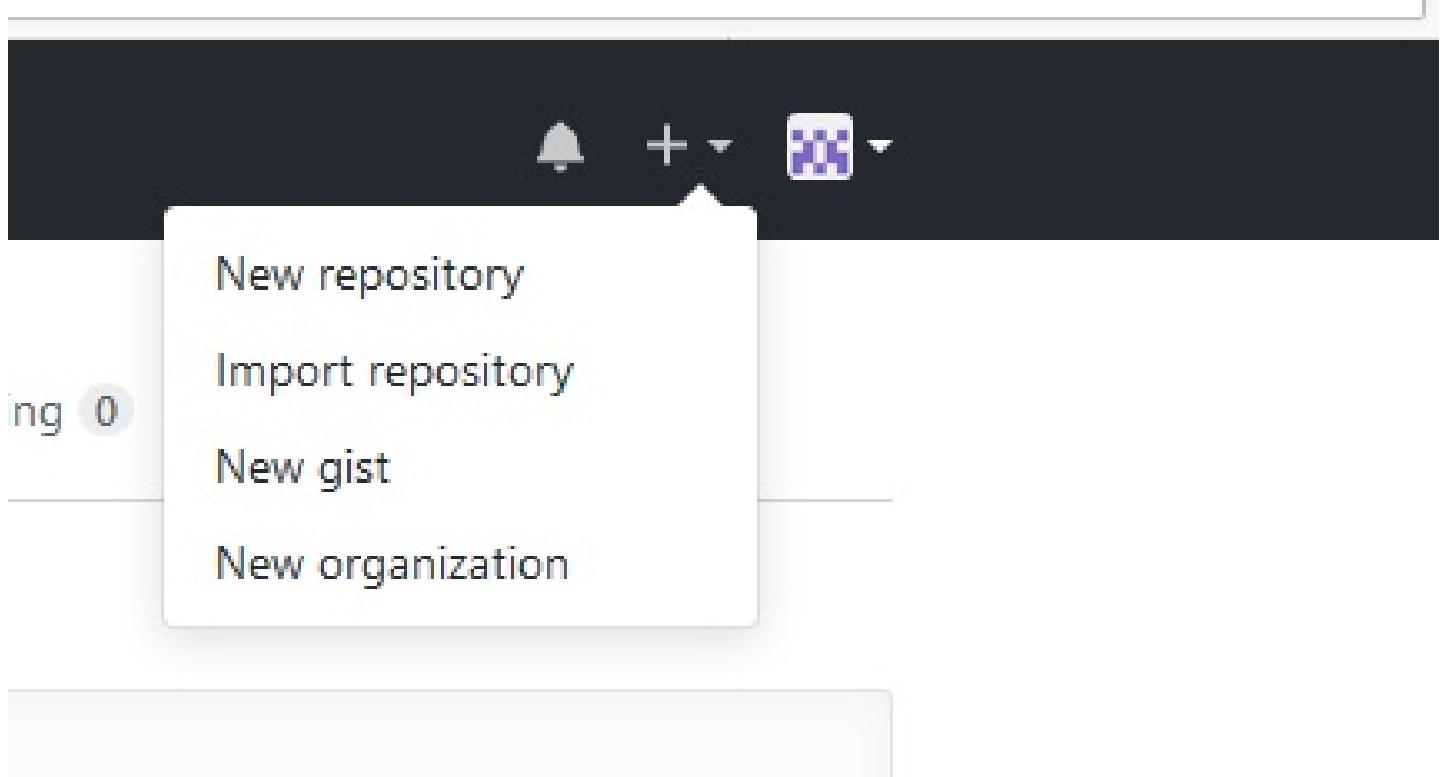
Individual Workflow

In this section, we'll describe how to use GitHub to version your own projects

Much of this will carry over to the collaborative section

Creating a Repository

In general, we will always want to repos for new projects using the following dropdown



We can then configure repository options as such

 A screenshot of a web browser displaying the GitHub 'Create a new repository' page. The URL is https://github.com/new. The page has a dark header with the GitHub logo and navigation links like 'Pull requests', 'Issues', 'Marketplace', and 'Explore'. Below the header is a search bar and a button to 'Search or jump to...'. The main content area is titled 'Create a new repository' and contains the following fields:

- Owner:** A dropdown menu shows 'quanteconuser' selected. Next to it is a repository name input field containing 'example_repository'.
- Description (optional):** A text input field containing 'A repository to model the GitHub workflow'.
- Visibility:** Radio buttons for 'Public' (selected) and 'Private'. The 'Public' option is described as 'Anyone can see this repository. You choose who can commit.' The 'Private' option is described as 'You choose who can see and commit to this repository.'
- Initialization:** A checked checkbox for 'Initialize this repository with a README'. A small note below says 'This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.'
- Additional Options:** Buttons for 'Add .gitignore: Julia' and 'Add a license: MIT License'.
- Create repository:** A large green button at the bottom.

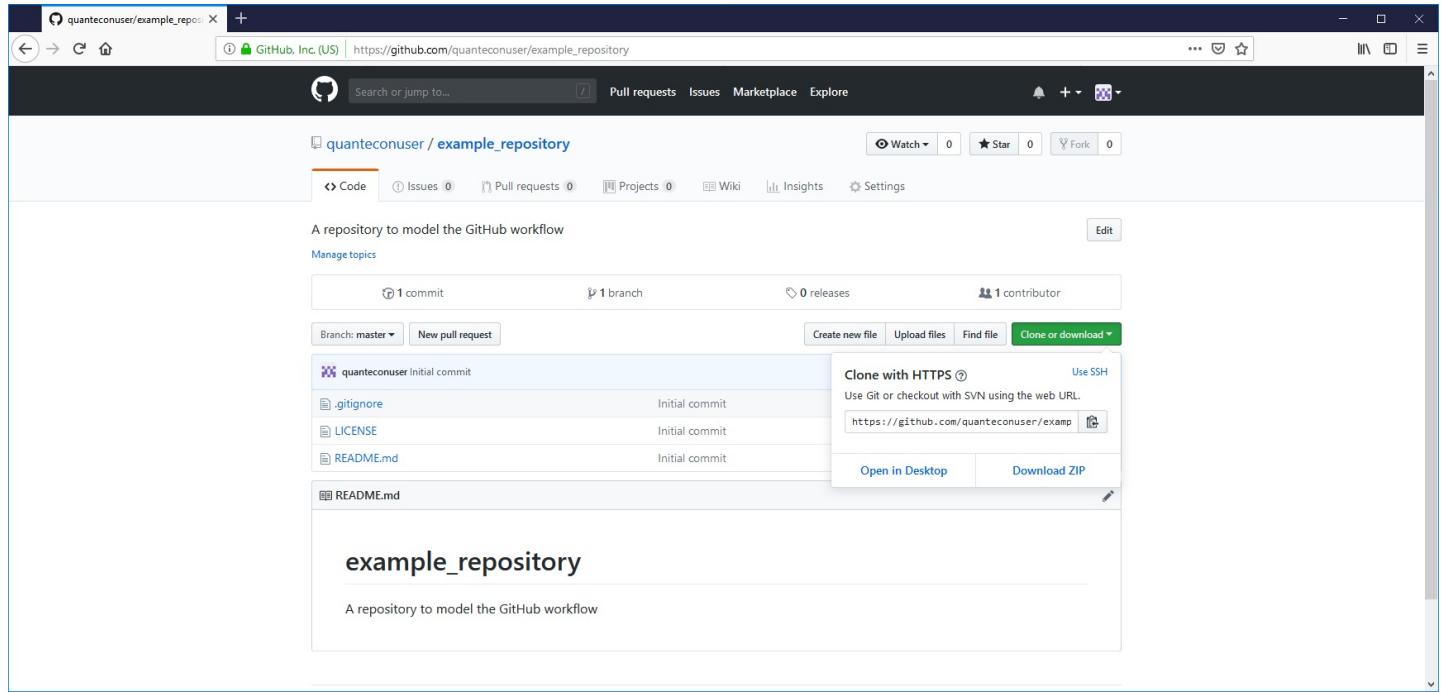
In this case, we're making a public repo `github.com/quantecon_user/example_repository`, which will come with a `README.md`, is licensed under the MIT License, and will ignore Julia compilation byproducts

Note This workflow is for creating projects de novo; the process for turning existing directories into git repos is a bit more complicated

In particular, in that case we recommend that you create a new repo via this method, then copy in and commit your files (see below), and then delete the old directory

Cloning a Repository

The next step is to get this to our local machine



This dropdown gives us a few options

- “Open in Desktop” will call to the GitHub Desktop application that we’ve installed
- “Download Zip” will download the directory without the .git subdirectory (avoid this option)
- The copy/paste button next to the link lets us use the command line, i.e. `git clone https://github.com/quanteconuser/example_repository.git`

Making and Managing Changes

Now that we have the repository, we can start working with it

For example, let’s say that we’ve amended the `README.md` (using our editor of choice), and also added a new file `economics.jl` which we’re still working on

Returning to GitHub Desktop, we should see something like

The screenshot shows the GitHub desktop application interface. At the top, the menu bar includes File, Edit, View, Repository, Branch, Help, and a repository dropdown for 'example_repository'. Below the menu, there are three main sections: 'Changes' (2), 'History', and 'README.md'. The 'Changes' tab shows two files: 'economics.jl' and 'README.md'. The 'History' tab shows a single commit for 'README.md'. The 'README.md' tab displays the file's content with a diff view:

```

@@ -1,2 +1,4 @@
 1 1 # example_repository
 2 2 A repository to model the GitHub workflow
 3 +
 4 +This is a change that we want to turn into a commit. ⚡

```

In the bottom right corner of the diff view, there is a small red icon with a white 'H'.

Below the main interface, a 'Summary (required)' dialog is open. It has a 'Description' field containing the text 'This is a change that we want to turn into a commit.' and a large blue 'Commit to master' button.

To select individual files for commit, we can use the check boxes to the left of each file

Let's say you select only the README to commit. Going to the history tab should show you our change

The screenshot shows the GitHub desktop application interface. At the top, the menu bar includes File, Edit, View, Repository, Branch, Help, and a repository dropdown for 'example_repository'. Below the menu, there are three main sections: 'Changes' (1), 'History', and 'README.md'. The 'Changes' tab shows one file: 'README.md'. The 'History' tab shows a single commit for 'README.md'. The 'README.md' tab displays the file's content with a diff view:

```

@@ -1,2 +1,4 @@
 1 1 # example_repository
 2 2 A repository to model the GitHub workflow
 3 +
 4 +This is a change that we want to turn into a commit. ⚡

```

In the bottom right corner of the diff view, there is a small red icon with a white 'H'.

On the left side of the interface, a sidebar shows a commit history for 'README.md' with the message 'QuantEcon User committed just now'.

The Julia file is unchanged

Pushing to the Server

As of now, this commit lives only on our local machine

To upload it to the server, you can simply click the "Push Origin" button at the top the screen

The small "1^" to the right of the text indicates we have one commit to upload

Reading and Reverting History

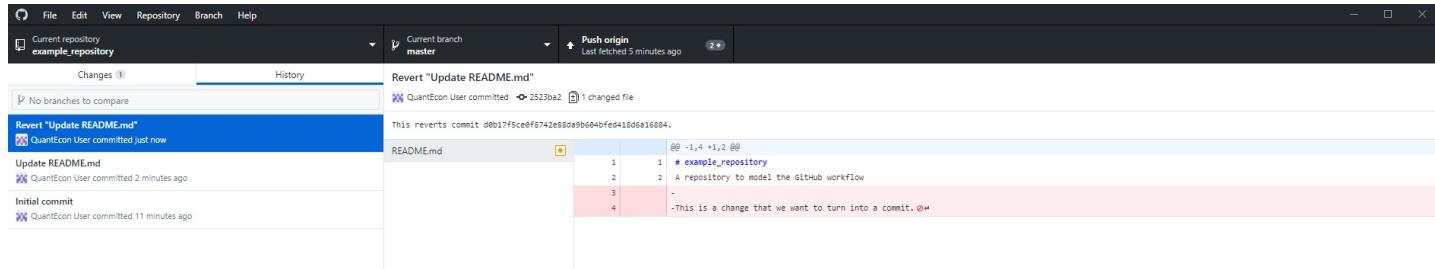
As mentioned, one of the key features of GitHub is the ability to scan through history

By clicking the "commits" tab on the repo front page, we see [this page](#) (as an example)

Clicking an individual commit gives us the difference view, (e.g., [example commit](#))

Sometimes, however, we want to not only inspect what happened before, but reverse the commit

- If you haven't made the commit yet, just right-click the file in the "changes" tab and hit "discard changes" to reset the file to the last known commit
- If you have made the commit but haven't pushed to the server yet, go to the "history" tab as above, right click the commit and click "revert this commit." This will create the inverse commit, shown below



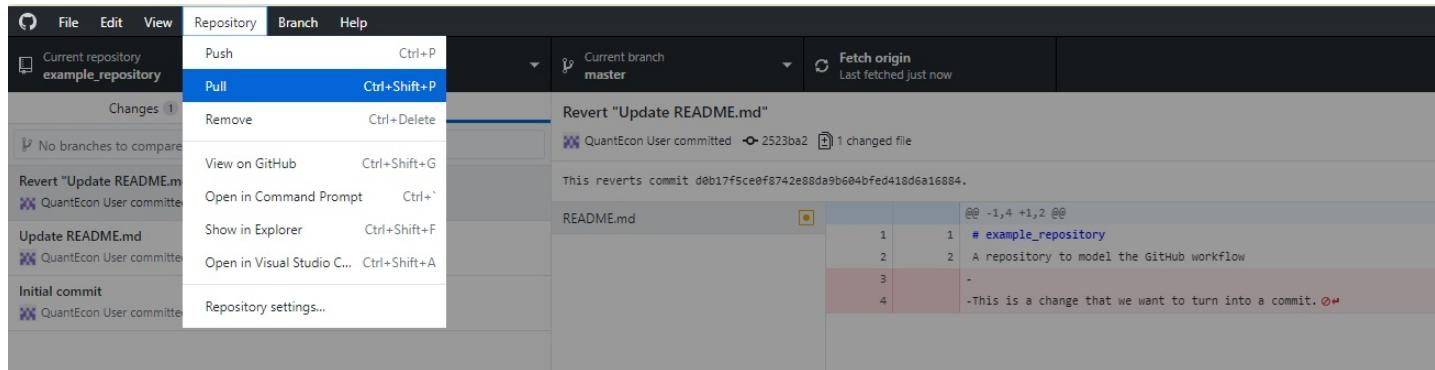
Working across Machines

Generally, you want to work on the same project but across multiple machines (e.g., a home laptop and a lab workstation)

The key is to push changes from one machine, and then to pull changes from the other machine

Pushing can be done as above

To pull, simply click pull under the "repository" dropdown at the top of the screen



Collaborative Work

Adding Collaborators

First, let's add a collaborator to the [quanteconuser/example_repository](#) lecture we created earlier

We can do this by clicking "settings => collaborators," as follows

The screenshot shows the GitHub repository settings page for 'quanteconuser / example_repository'. The 'Collaborators' tab is selected. On the left, there's a sidebar with options like 'Code', 'Issues 0', 'Pull requests 0', 'Projects 0', 'Wiki', 'Insights', and 'Settings'. The main area has a heading 'Collaborators' and a sub-section 'Push access to the repository'. It says 'This repository doesn't have any collaborators yet. Use the form below to add a collaborator.' Below this is a search bar for 'Search by username, full name or email address' with the note 'You'll only be able to find a GitHub user by their email address if they've chosen to list it publicly. Otherwise, use their username instead.' At the bottom right of the search bar is a 'Add collaborator' button.

Project Management

GitHub's website also comes with project management tools to coordinate work between people

The main one is an issue, which we can create from the issues tab

You should see something like this

The screenshot shows the GitHub 'New Issue' creation interface for 'quanteconuser / example_repository'. The 'Issues' tab is selected in the top navigation. The main form has fields for 'Title' (with 'Write' and 'Preview' tabs), 'Leave a comment', and a 'Submit new issue' button. To the right, there are dropdown menus for 'Assignees' (set to 'No one—assign yourself'), 'Labels' (set to 'None yet'), 'Projects' (set to 'None yet'), and 'Milestone' (set to 'No milestone'). At the bottom of the form, it says 'Styling with Markdown is supported'.

Let's unpack the different components

- The assignees dropdown lets you select people tasked to work on the issue
- The labels dropdown lets you tag the issue with labels visible from the issues page, such as "high priority" or "feature request"
- It's possible to tag other issues and collaborators (including in different repos) by linking to them in the comments – this is part of what's called GitHub-Flavored Markdown

For an example of an issue, see [here](#)

You can see open issues at a glance from the general issues tab

The screenshot shows the GitHub Issues page for a repository named 'quanteconuser / example_repository'. At the top, there are tabs for 'Code', 'Issues 1', 'Pull requests 0', 'Projects 0', 'Wiki', 'Insights', and 'Settings'. A prominent message box at the top says 'Label issues and pull requests for new contributors' and 'Now, GitHub will help potential first-time contributors discover issues labeled with `help wanted` or `good first issue`'. Below the message are filters for 'is:issue is:open', 'Labels', 'Milestones', and a 'New issue' button. A search bar contains the query 'is:issue is:open'. The main list shows one open issue: '#1 Learn GitHub good first issue help wanted' by quanteconuser, which was opened 43 seconds ago. The issue has a status of '2 of 4' and is labeled with 'good first issue' and 'help wanted'. At the bottom, there's a 'ProTip!' message about mentions and a footer with copyright information and links to Contact GitHub, Pricing, API, Training, Blog, and About.

The checkboxes are common in GitHub to manage project tasks

Reviewing Code

There are a few different ways to review people's code in GitHub

- Whenever people push to a project you're working on, you'll receive an email notification
- You can also review individual line items or commits by opening commits in the difference view as [above](#)

The screenshot shows the GitHub commit history for a file named 'Update README.md'. It shows a commit from 'quanteconuser' that reverts a previous commit. The commit message is 'Revert "Update README.md"'. The commit details show it reverts commit 'd0b17f5'. The commit was made 8 minutes ago and has 1 parent commit '2523ba270e1455773968bab73aed42b06b86f380'. Below the commit details, it says 'Showing 1 changed file with 0 additions and 2 deletions.' The diff view shows two changes: one addition and one deletion. In the 'Unified' view, the addition is '# example_repository' and the deletion is 'A repository to model the GitHub workflow'. Below the diff, there are '0 comments on commit 2523ba2'. A comment box is open, showing a message 'I don't think we should revert this.' and a 'Comment on this commit' button. There are also 'Write' and 'Preview' buttons, and a toolbar with various icons for styling and file operations.

Merge Conflicts

Any project management tool needs to figure out how to reconcile conflicting changes between people

In GitHub, this event is called a “merge conflict,” and occurs whenever people make conflicting changes to the same line of code

Note that this means that two people touching the same file is OK, so long as the differences are compatible

A common use case is when we try to push changes to the server, but someone else has pushed conflicting changes

GitHub will give us the following window

Merge branch 'master' of https://github.com/quanteconuser/example_repository

Description

Commit to master

Committed just now
Update README.md Undo

- The warning symbol next to the file indicates the existence of a merge conflict
- The viewer tries to show us the discrepancy (I changed the word repository to repo, but someone else tried to change it to “repo” with quotes)

To fix the conflict, we can go into a text editor (such as Atom)

README.md — C:\Users\Arnav Sood\Desktop\example_repository — Atom

File Edit View Julia Selection Find Packages Help

Project README.md

example_repository

use me

our changes

their changes

Let's say we click the first “use me” (to indicate that my changes should win out), and then save the file

Returning to GitHub Desktop gives us a pre-formed commit to accept

Changes 2 History README.md

economics.jl README.md ✓

No content changes found

Merge branch 'master' of https://github.com/quanteconuser/example_repository

Description

Commit to master

Committed 2 minutes ago
Update README.md Undo

Clicking “commit to master” will let us push and pull from the server as normal

Collaboration via Pull Request

One of the defining features of GitHub is that it is the dominant platform for open source code, which anyone can access and use

However, while anyone can make a copy of the source code, not everyone has access to modify the particular version stored on GitHub

A maintainer (i.e. someone with “write” access to directly modify a repository) might consider different contributions and “merge” the changes into the main repository if the changes meet their criteria

A pull request (“PR”) allows **any** outsiders to suggest changes to open source repositories

A PR requests the project maintainer to merge (“pull”) changes you’ve worked on into their repository

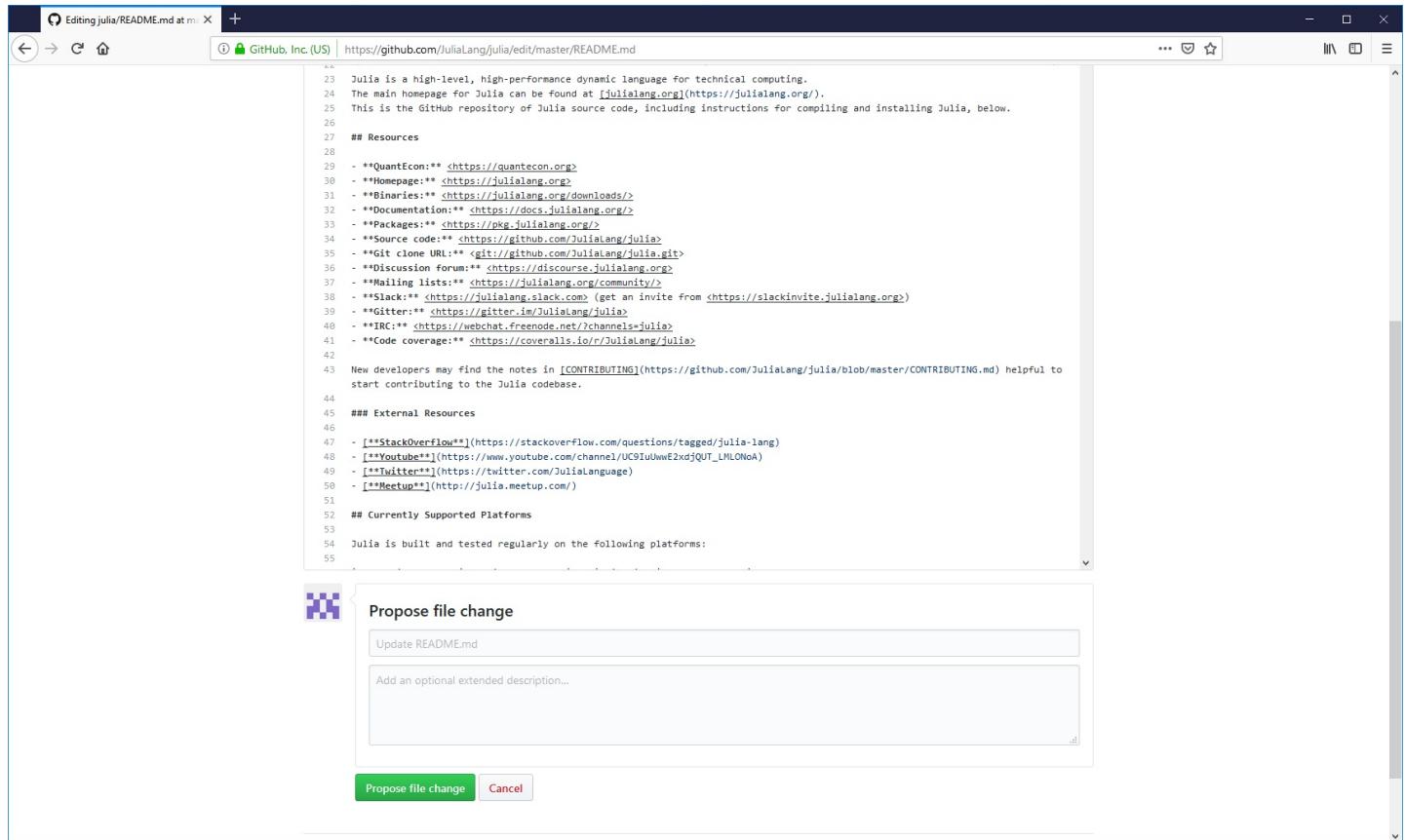
There are a few different workflows for creating and handling PRs, which we’ll walk through below

Note: If the changes are for a Julia Package, you will need to follow a different workflow – described in the [testing lecture](#)

Quick Fixes

GitHub’s website provides an online editor for quick and dirty changes, such as fixing typos in documentation

To use it, open a file in GitHub and click the small pencil to the upper right



The screenshot shows a GitHub browser window with the URL <https://github.com/JuliaLang/julia/edit/master/README.md>. The page displays the Julia README file, which includes links to various resources like QuantEcon, Homepage, Binaries, Documentation, Packages, Source code, Git clone URL, Discussion forum, Mailing lists, Slack, Gitter, IRC, and Code coverage. Below the file content, there's a note about contributing to the Julia codebase. A modal dialog box titled "Propose file change" is overlaid on the page, containing fields for updating the README and adding an optional extended description. At the bottom of the modal are "Propose file change" and "Cancel" buttons.

Here, we’re trying to add the QuantEcon link to the Julia project’s `README` file

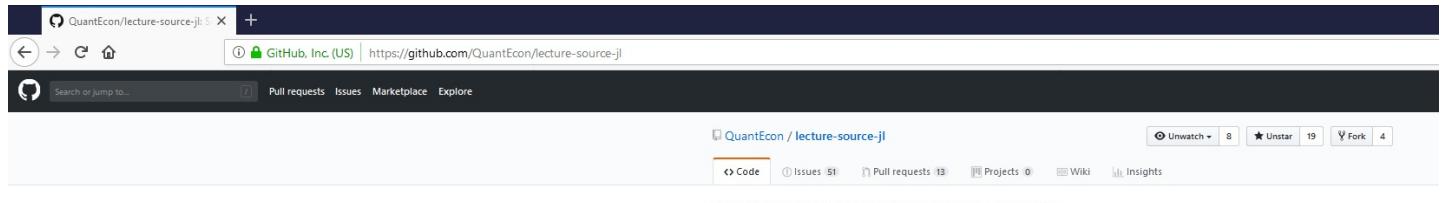
After making our changes, we can then describe and propose them for review by maintainers

But what if we want to make more in-depth changes?

No-Access Case

A common problem is when we don’t have write access (i.e. we can’t directly modify) the repo in question

In that case, click the “Fork” button that lives in the top-right of every repo’s main page



The screenshot shows the GitHub main page for the repository [QuantEcon/lecture-source-jl](https://github.com/QuantEcon/lecture-source-jl). The page includes navigation links for Pull requests, Issues, Marketplace, and Explore. In the top right corner, there are buttons for Unwatch, Star, Fork, and a link to Insights. The repository name "QuantEcon / lecture-source-jl" is displayed above the main content area. Below the header, there are links for Code, Issues, Pull requests, Projects, Wiki, and Insights. A note at the bottom states "Source files for 'Lectures in Quantitative Economics' -- Julia version".

This will copy the repo into your own GitHub account

For example, [this repo](#) is a fork of our original git setup

Clone this fork to our desktop and work with it in exactly the same way as we would a repo we own (as the fork is in your account, you now have write access)

That is, click the “clone” button on our fork

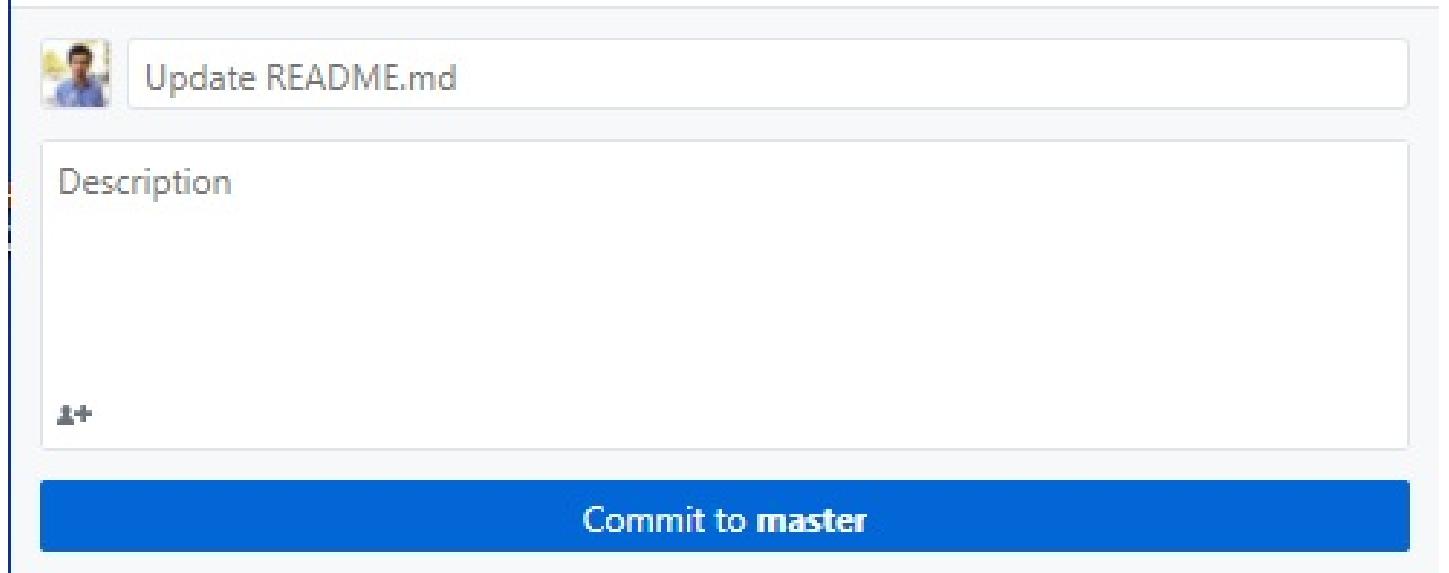
A screenshot of a GitHub repository page for 'ubcecon / example_repository'. The page shows basic statistics: 5 commits, 1 branch, 0 releases, 2 contributors, and an MIT license. A prominent 'Clone or download' button is highlighted in green. The repository description is 'A repository to model the GitHub workflow'. Below the stats, there's a list of files: .gitignore (Initial commit), LICENSE (Initial commit), and README.md (Update README.md). A commit by 'arnav' is shown: 'Update README.md'. A tooltip for the 'Clone with HTTPS' button provides the URL: https://github.com/ubcecon/example_repo.

You'll see a new repo with the same name but different URL in your GitHub Desktop repo list, along with a special icon to indicate that it's a fork

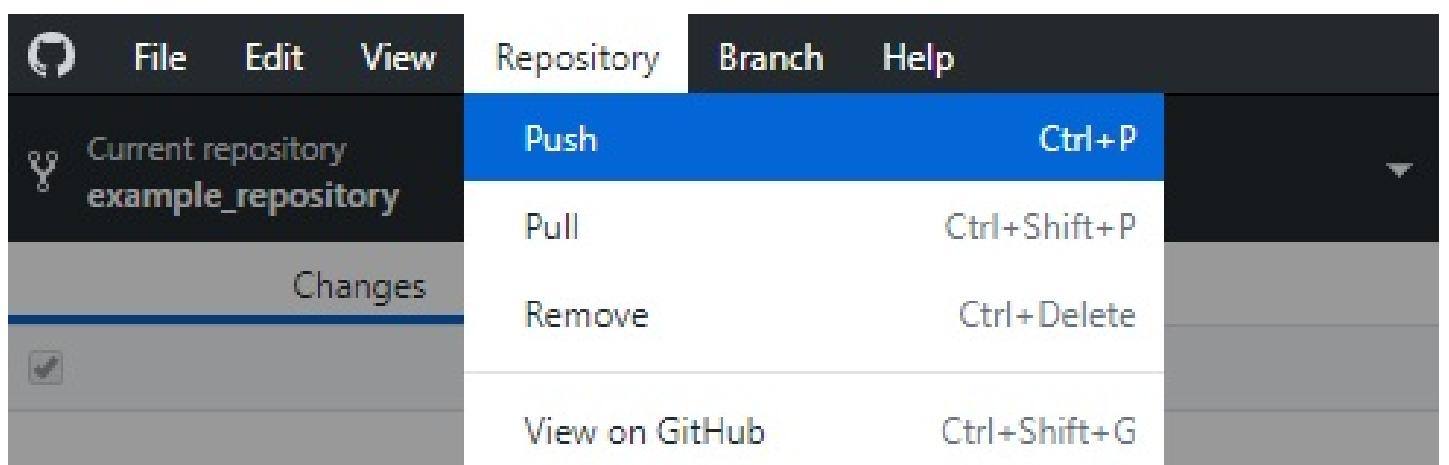
A screenshot of GitHub Desktop showing two repositories in the list. The first is 'ubcecon/example_repository' and the second is 'quanteconuser/example_repository'. The 'quanteconuser' repository is marked with a yellow fork icon, indicating it is a fork.

Commit some changes by selecting the files and writing a commit message

A screenshot of GitHub Desktop in 'Changes' mode. A commit message '1 changed file' is visible above a list of files. The file 'README.md' has a checked checkbox next to it, indicating it is selected for commit. The commit message field contains the text 'Update README.md'.



And push by using the dropdown



Below, for example, we've committed and pushed some changes to the fork that we want to upstream into the main repo

The screenshot shows the GitHub desktop application interface. On the left, there's a sidebar with a dropdown for 'Current repository' set to 'example_repository'. Below it are sections for 'Changes' and 'History'. The main area shows a pull request between 'Current branch' (set to 'master') and 'Fetch origin' (set to 'Last fetched just now'). A specific commit, 'Update README.md' by 'Arnav Sood committed Nov 2, 2018', is highlighted. The commit details show it added a file named '# example_repository' and updated the workflow. The commit message includes: 'A "repo" to model the GitHub workflow' and '+Here are some changes we want to merge into the main repository.'

We should make sure that these changes are on the server (which we can get to by going to the [fork](#) and clicking “commits”)

The screenshot shows the GitHub web interface for the repository 'ubcecon/example_repository'. At the top, there are buttons for 'Unwatch', 'Star', 'Fork', and a count of '3' issues. Below that is a navigation bar with tabs for 'Code', 'Pull requests 0', 'Projects 0', 'Wiki', 'Insights', and 'Settings'. A dropdown menu shows the current branch is 'master'. The main content area is titled 'Commits on Nov 2, 2018' and lists five commits:

- 'Update README.md' by 'arnav' (committed 11 days ago) - Verified, SHA: 8406bb2
- 'Update README.md' by 'quanteconuser' (committed 11 days ago) - Verified, SHA: edd69cf
- 'Revert "Update README.md"' by 'quanteconuser' (committed 11 days ago) - SHA: 2523ba2
- 'Update README.md' by 'quanteconuser' (committed 11 days ago) - SHA: d0b17f5
- 'Initial commit' by 'quanteconuser' (committed 11 days ago) - Verified, SHA: 7438796

At the bottom, there are 'Newer' and 'Older' buttons.

Next, go to the pull requests menu and click “New Pull Request”

You'll see something like this

The screenshot shows the GitHub web interface comparing two branches. The URL is 'https://github.com/quanteconuser/example_repository/compare/master...ubcecon:master'. The top navigation bar includes 'Issues 1', 'Pull requests 0', 'Projects 0', 'Wiki', and 'Insights'. The main content is titled 'Comparing changes' and says 'Choose two branches to see what's changed or to start a new pull request. If you need to, you can also compare across forks.' It shows a comparison between 'base fork: quanteconuser/example_repository' and 'head fork: ubcecon/example_repository' with 'compare: master'. A green checkmark indicates 'Able to merge'. Below this is a summary: '1 commit', '1 file changed', '0 commit comments', and '1 contributor'. It lists a single commit by 'arnav' on Nov 02, 2018, titled 'Update README.md' with SHA 8406bb2. At the bottom, it shows 'Showing 1 changed file with 2 additions and 0 deletions.' and a diff view of the README.md file.

This gives us a quick overview of the commits we want to merge in, as well as the overall differences

Hit create and then click through the following form

This opens a page like this on the main repo

The key pieces are

- A list of the commits we're proposing
- A list of reviewers, who can approve or modify our changes
- Labels, Markdown space, assignees, and the ability to tag other git issues and PRs, just as with issues

Here's an [example pull request](#)

To edit a PR, simply push changes to the fork you cloned to your desktop

For example, let's say we commit a new change to the README after we create the PR

After pushing to the server, the change is reflected on the [PR page](#)

That is, creating a pull request is not like bundling up your changes and delivering them, but rather like opening an ongoing connection between two repositories, that is only severed when the PR is closed or merged

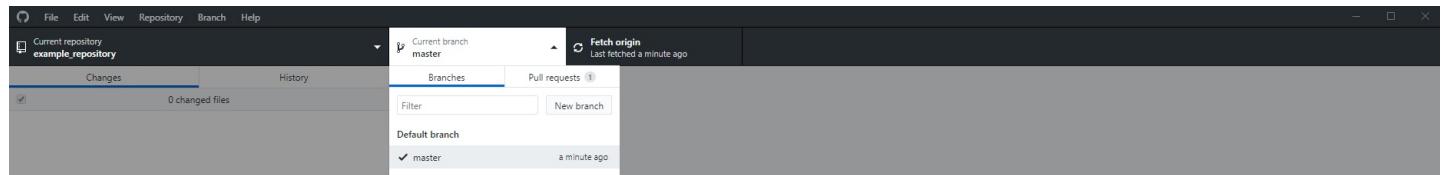
Write Access Case

As you become more familiar with GitHub, and work on larger projects, you will find yourself making PRs even when it isn't strictly required

If you are a maintainer of the repo (e.g. you created it or are a collaborator) then you don't need to create a fork, but will rather work with a git branch

Branches in git represent parallel development streams (i.e., sequences of commits) that the PR is trying to merge

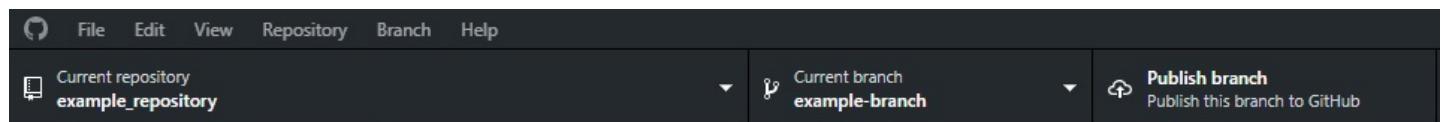
First, load the repo in GitHub Desktop and use the branch dropdown



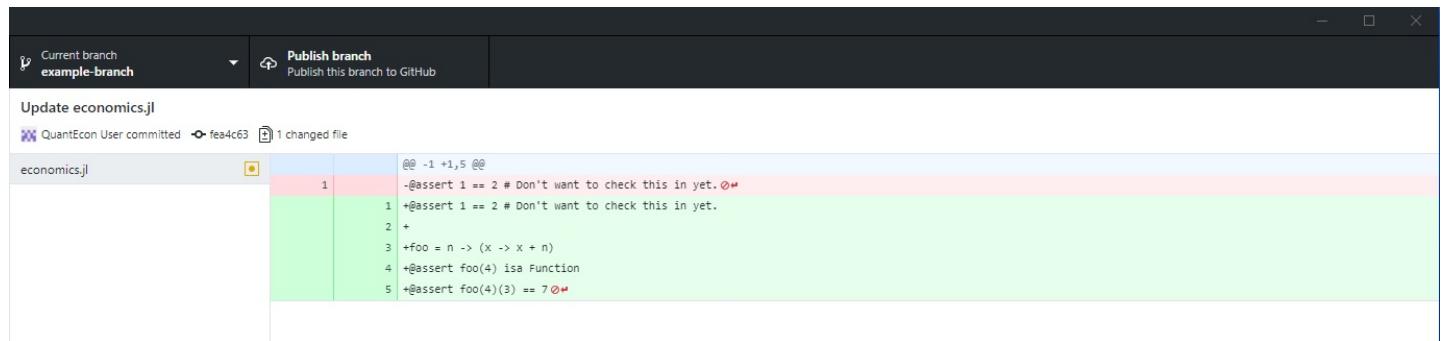
Click "New Branch" and choose an instructive name (make sure there are no spaces or special characters)

This will "check out" a new branch with the same history as the old one (but new commits will be added only to this branch)

We can see the active branch in the top dropdown



For example, let's say we add some stuff to the Julia code file and commit it



To put this branch (with changes) on the server, we simply need to click "Publish Branch"

Navigating to the [repo page](#), we will see a suggestion about a new branch

ls

The screenshot shows a GitHub repository page for 'quanteconuser/example_repository'. At the top, there are navigation links for 'Pull requests', 'Issues', 'Marketplace', and 'Explore'. Below the header, the repository name 'quanteconuser / example_repository' is displayed, along with 'Watch 0', 'Star 0', 'Fork 1' buttons. A modal window titled 'Label issues and pull requests for new contributors' is open, explaining that GitHub will help potential first-time contributors discover issues labeled with 'help wanted' or 'good first issue'. Below the modal, a message says 'Your recently pushed branches:' followed by 'example-branch (less than a minute ago)'. There is a 'Compare & pull request' button. The main content area shows a search bar with 'is:pr is:open' and filters for '1 Open' and '1 Closed'. A specific pull request is listed: '#3 opened 11 days ago by arnavs' with the title 'Update README.md'. At the bottom, a pro tip says 'Type g p on any issue or pull request to go back to the pull request listing page.'

At which point the process of creating a PR is identical to the previous case

Julia Package Case

One special case is when the repo in question is actually a Julia project or package

We cover that (along with package workflow in general) in the [testing lecture](#)

Additional Resources and Troubleshooting

You may want to go beyond the scope of this tutorial when working with GitHub

For example, perhaps you run into a bug, or you're working with a setup that doesn't have GitHub Desktop installed

Here are some resources to help

- Kate Hudson's excellent [git flight rules](#), which is a near-exhaustive list of situations you could encounter, and command-line fixes
- The GitHub [Learning Lab](#), an interactive sandbox environment for git
- The docs for forking on [GitHub Desktop](#) and [the GitHub Website](#)

Command-Line Basics

Git also comes with a set of command-line tools

They're optional, but many people like using them

Furthermore, in some environments (e.g. JupyterHub installations) you may only have access to the command line

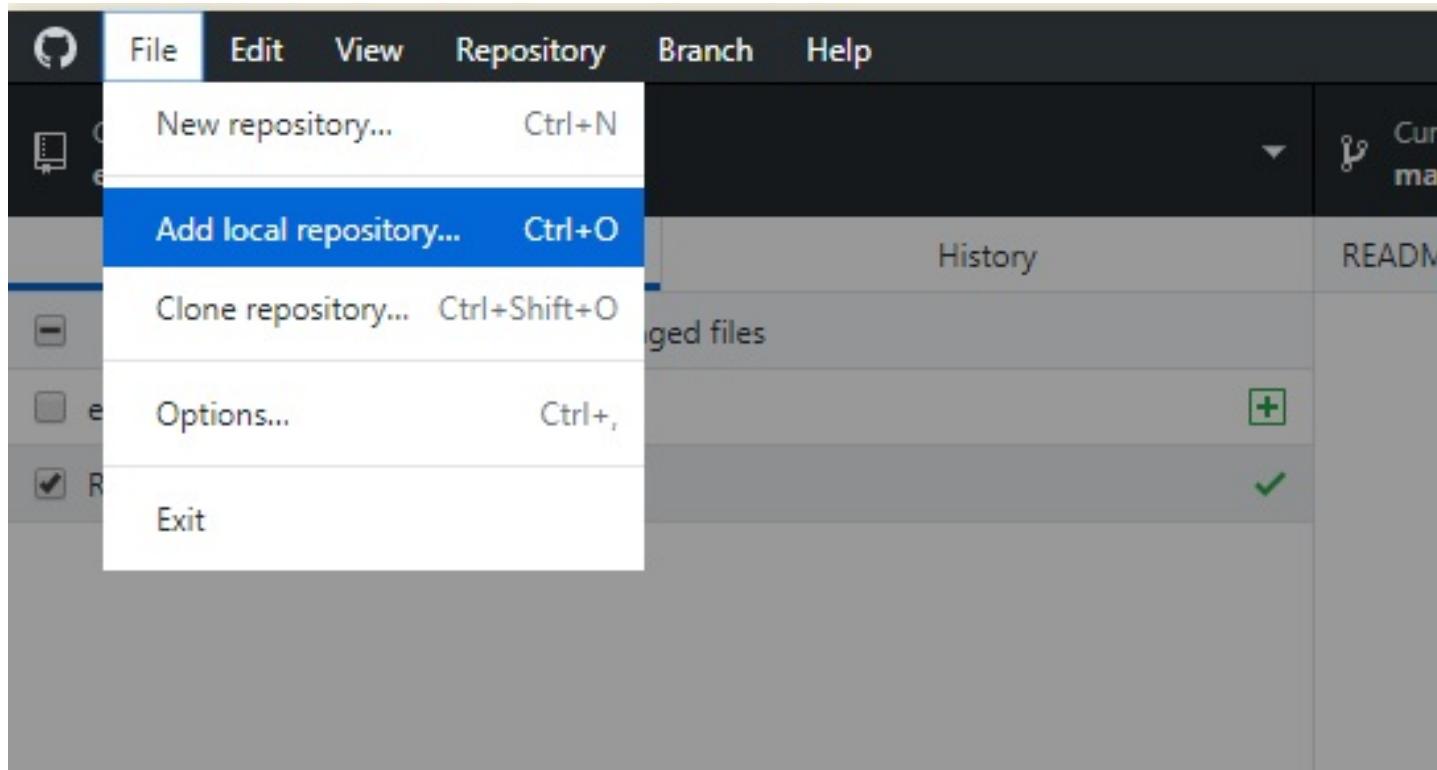
- On Windows, downloading `git` will have installed a program called `git bash`, which installs these tools along with a general Linux-style shell
- On Linux/MacOS, these tools are integrated into your usual terminal

To open the terminal in a directory, either right click and hit “open git bash” (in Windows), or use Linux commands like `cd` and `ls` to navigate

See [here](#) for a short introduction to the command line

As above, you can clone by grabbing the repo URL (say, GitHub’s [site-policy repo](#)) and running `git clone https://github.com/github/site-policy.git`

This won’t be connected to your GitHub Desktop, so you’d need to use it manually (`File => Add Local Repository`) or drag-and-drop from the file explorer onto the GitHub Desktop



From here, you can get the latest files on the server by `cd`-ing into the directory and running `git pull`

When you `pull` from the server, it will never overwrite your modified files, so it is impossible to lose local changes

Instead, to do a hard reset of all files and overwrite any of your local changes, you can run `git reset --hard origin/master`

Exercises

Exercise 1a

Follow the instructions to create a [new repository](#) for one of your GitHub accounts In this repository

- Take the code from one of your previous assignments, such as [Newton’s method in Introductory Examples](#) (either as a `.jl` file or a Jupyter notebook)
- Put in a `README.md` with some text
- Put in a `.gitignore` file, ignoring the Jupyter files `.ipynb_checkpoints` and the project files, `.projects`

Exercise 1b

Pair-up with another student who has done Exercise 1a and find out their GitHub ID, and each do the following

- Add the GitHub ID as a collaborators on your repository
- Clone the repositories to your local desktop

- Assign each other an issue
- Submit a commit from GitHub Desktop which references the issue by number
- Comment on the commits
- Ensure you can run their code without any modifications

Exercise 1c

Pair-wise with the results of Exercise 1b examine a merge-conflict by editing the `README.md` file for your repository that you have both setup as collaborators

Start by ensuring there are multiple lines in the file so that some changes may have conflicts, and some may not

- Clone the repository to your local desktops
- Modify **different** lines of code in the file and both commit and push to the server (prior to pulling from each other)–and see how it merges things “automatically”
- Modify **the same** line of code in the file, and deal with the merge conflict

Exercise 2a

Just using GitHub’s web interface, submit a Pull Request for a simple change of documentation to a public repository

The easiest may be to submit a PR for a typo in the source repository for these notes, i.e.

<https://github.com/QuantEcon/lecture-source-jl>

Note: The source for that repository is in `.rst` files, but you should be able to find spelling mistakes/etc. without much effort

Exercise 2b

Following the instructions for forking and cloning a public repository to your local desktop, submit a Pull Request to a public repository

Again, you could submit it for a typo in the source repository for these notes, i.e. <https://github.com/QuantEcon/lecture-source-jl>, but you are also encouraged to instead look for a small change that could help the documentation in another repository.

If you are ambitious, then go to the Exercise Solutions for one of the Exercises in these lecture notes and submit a PR for your own modified version (if you think it is an improvement!)



This work is licensed under a [Creative Commons Attribution-NoDerivatives 4.0 International License](#).

© Copyright 2017, Thomas J. Sargent and John Stachurski. Created using [Sphinx](#), hosted with [AWS](#).