Lecture 3

Git and GitHub

Ivan Rudik
AEM 7130

Software and stuff

Necessary things to do:

- Install Git
- Create an account on GitHub
- Install GitHub Desktop if you want a GUI for Git
- Accept invite to the AEM 7130 classroom repository on GitHub

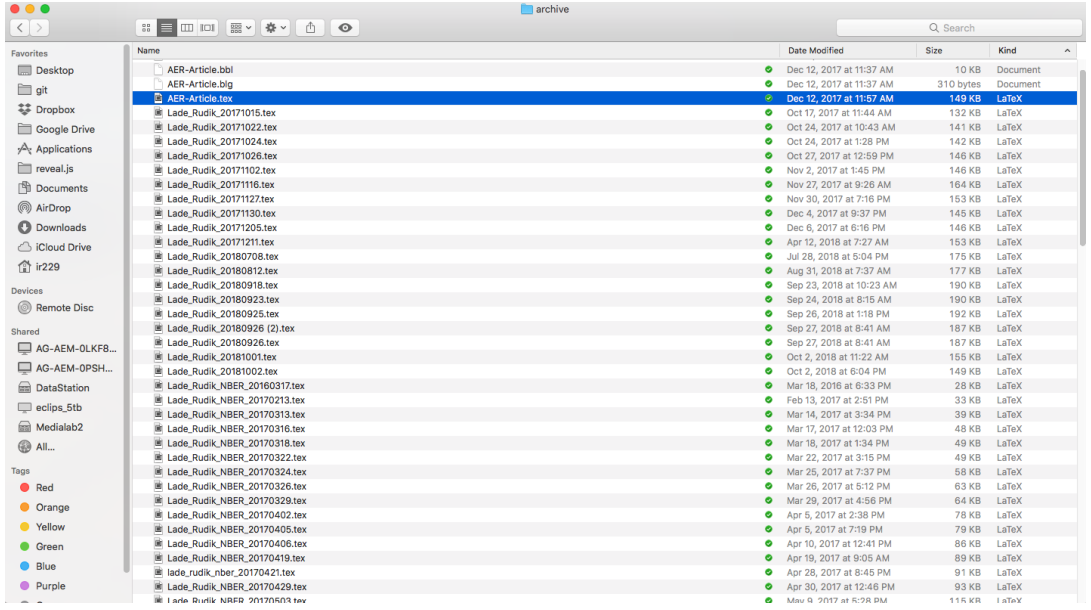
Quick note

A decent chunk of these slides is inspired by Grant McDermott's data science course

If you're interested in more data science-y (similar to Ariel's class) and less numerical/structural material, check his material out

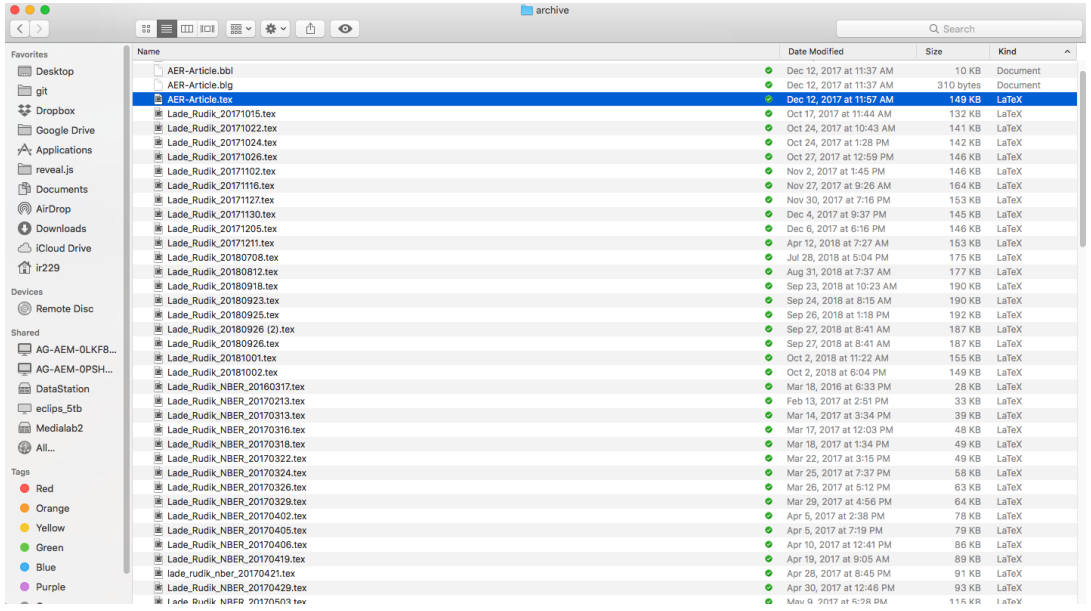
Why bother with this new fangled Git stuff?

The classic **date your file name method** is not good



Why bother with this new fangled Git stuff?

The classic **date your file name method** is not good

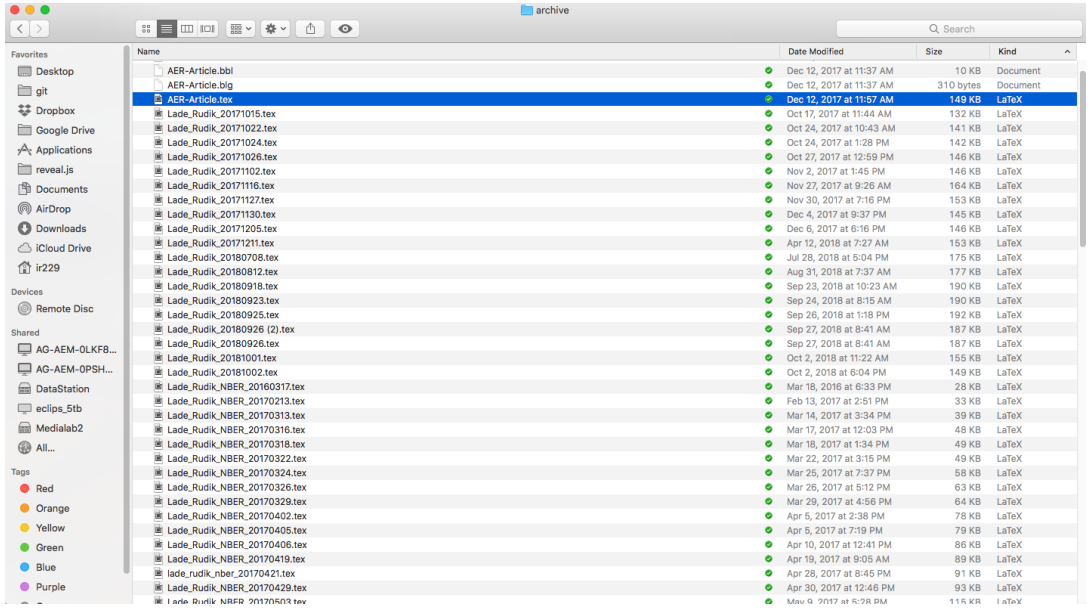


When did you make changes? Who made them?

How do you undo only **some** changes from one update to the next?

Why bother with this new fangled Git stuff?

The classic **date your file name method** is not good



When did you make changes? Who made them?

How do you undo only **some** changes from one update to the next?

If you've ever had a disaster managing code changes (you will), Git can help

Git is the smart way to handle code

What is git?

> Git is a distributed version control system for tracking changes in source code during software development. It is designed for coordinating work among programmers, but it can be used to track changes in any set of files. Its goals include speed, data integrity, and support for distributed, non-linear workflows.

Git is the smart way to handle code

Okay, so what?

Git is the smart way to handle code

Okay, so what?

Git combines a bunch of very useful features:

Git is the smart way to handle code

Okay, so what?

Git combines a bunch of very useful features:

- Remote storage of code on a host like GitHub/GitLab/Bitbucket/etc, **just like Dropbox**

Git is the smart way to handle code

Okay, so what?

Git combines a bunch of very useful features:

- Remote storage of code on a host like GitHub/GitLab/Bitbucket/etc, **just like Dropbox**
- Tracking of changes to files in a very clean way

Git is the smart way to handle code

Okay, so what?

Git combines a bunch of very useful features:

- Remote storage of code on a host like GitHub/GitLab/Bitbucket/etc, **just like Dropbox**
- Tracking of changes to files in a very clean way
- Easy ways to test out experimental changes (e.g. new specifications, additional model states) and not have them mess with your main code

Git is the smart way to handle code

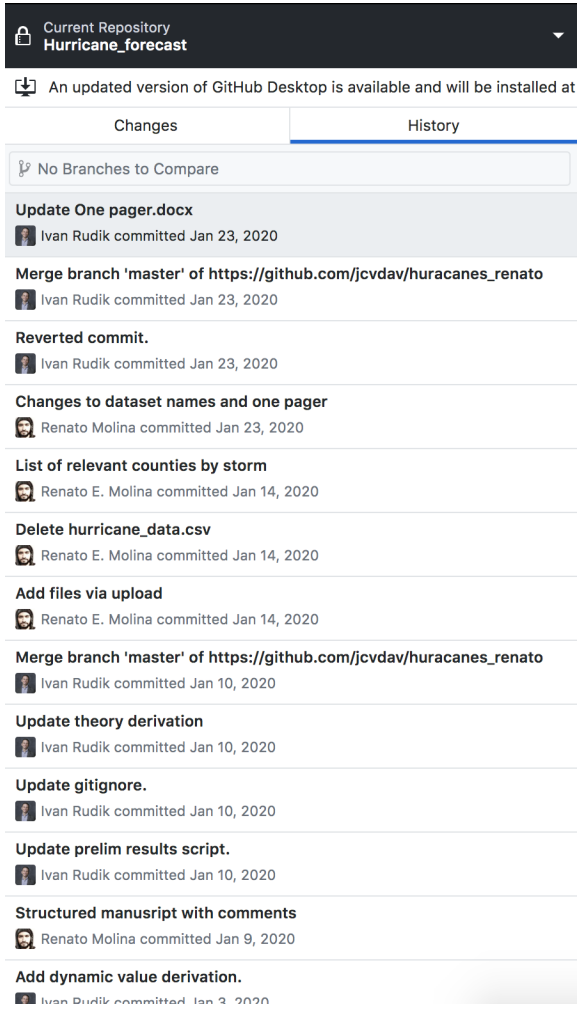
Okay, so what?

Git combines a bunch of very useful features:

- Remote storage of code on a host like GitHub/GitLab/Bitbucket/etc, **just like Dropbox**
- Tracking of changes to files in a very clean way
- Easy ways to test out experimental changes (e.g. new specifications, additional model states) and not have them mess with your main code
- Built for versioning **code** like R, Julia, LaTeX, etc

Git histories in GitHub Desktop

Some apps can give you a pretty visual of the history of changes to your code (shell can too, but not as nice)



GitHub

Git $\neq$ GitHub

GitHub

Git $\neq$ GitHub

GitHub hosts a bunch of online services we want when using Git

GitHub

Git $\neq$ GitHub

GitHub hosts a bunch of online services we want when using Git

- Allows for people to suggest code changes to existing code

GitHub

Git $\neq$ GitHub

GitHub hosts a bunch of online services we want when using Git

- Allows for people to suggest code changes to existing code
- It's the main location for non-base Julia packages (and **tons** of other stuff) to be stored and developed

GitHub

Git $\neq$ GitHub

GitHub hosts a bunch of online services we want when using Git

- Allows for people to suggest code changes to existing code
- It's the main location for non-base Julia packages (and **tons** of other stuff) to be stored and developed
- It has services that I used to set up this class, etc

The differences

The differences

Git is the infrastructure for versioning and merging files

The differences

Git is the infrastructure for versioning and merging files

GitHub provides an online service to coordinate working with Git repositories, and adds some additional features for managing projects

The differences

Git is the infrastructure for versioning and merging files

GitHub provides an online service to coordinate working with Git repositories, and adds some additional features for managing projects

GitHub stores the project on the cloud, allows for task management, creation of groups, etc

Why Git and GitHub?

Selfish reasons

Why Git and GitHub?

Selfish reasons

The private benefits of having well-versioned code in case you need to go back to previous stages

Why Git and GitHub?

Selfish reasons

The private benefits of having well-versioned code in case you need to go back to previous stages

Your directories will be super clean

Why Git and GitHub?

Selfish reasons

The private benefits of having well-versioned code in case you need to go back to previous stages

Your directories will be super clean

It is **MUCH** easier to collaborate on projects

Why Git and GitHub?

Semi-altruistic reasons

Why Git and GitHub?

Semi-altruistic reasons

The external benefits of open science, collaboration, etc

Why Git and GitHub?

Semi-altruistic reasons

The external benefits of open science, collaboration, etc

These external benefits also generate some downstream private reputational benefits (must be confident in your code to make it public) and can improve future social efficiency (commitment device to post future code)

Why Git and GitHub?

Semi-altruistic reasons

The external benefits of open science, collaboration, etc

These external benefits also generate some downstream private reputational benefits (must be confident in your code to make it public) and can improve future social efficiency (commitment device to post future code)

My code for **everything** I've ever published is on my GitHub (I'll look real shady if I don't post code in the future)

Why Git and GitHub?

Semi-altruistic reasons

The external benefits of open science, collaboration, etc

These external benefits also generate some downstream private reputational benefits (must be confident in your code to make it public) and can improve future social efficiency (commitment device to post future code)

My code for **everything** I've ever published is on my GitHub (I'll look real shady if I don't post code in the future)

Ideally yours will be too

Git basics

Everything on Git is stored in something called a **repository** or *repo* for short

Git basics

Everything on Git is stored in something called a **repository** or *repo* for short

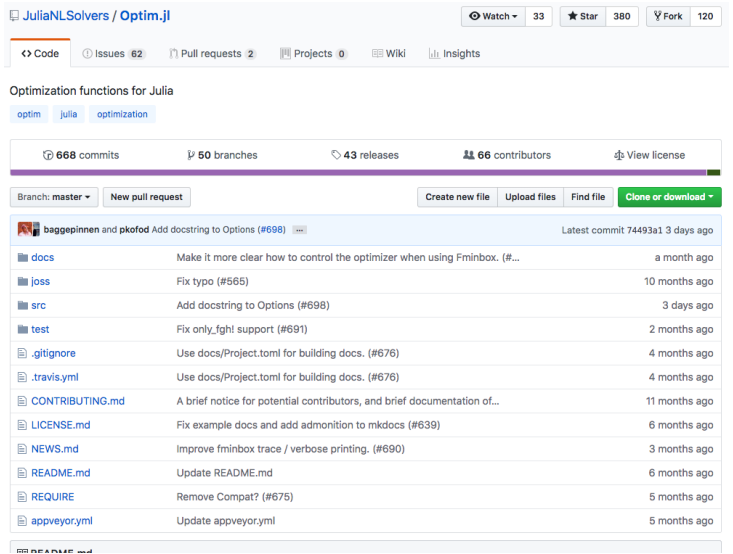
This is the directory for a project

Git basics

Everything on Git is stored in something called a **repository** or *repo* for short

This is the directory for a project

- **Local:** a directory with a `.git` subdirectory that stores the history of changes to the repository
- **Remote:** a website, e.g. see the GitHub repo for the Optim package in Julia



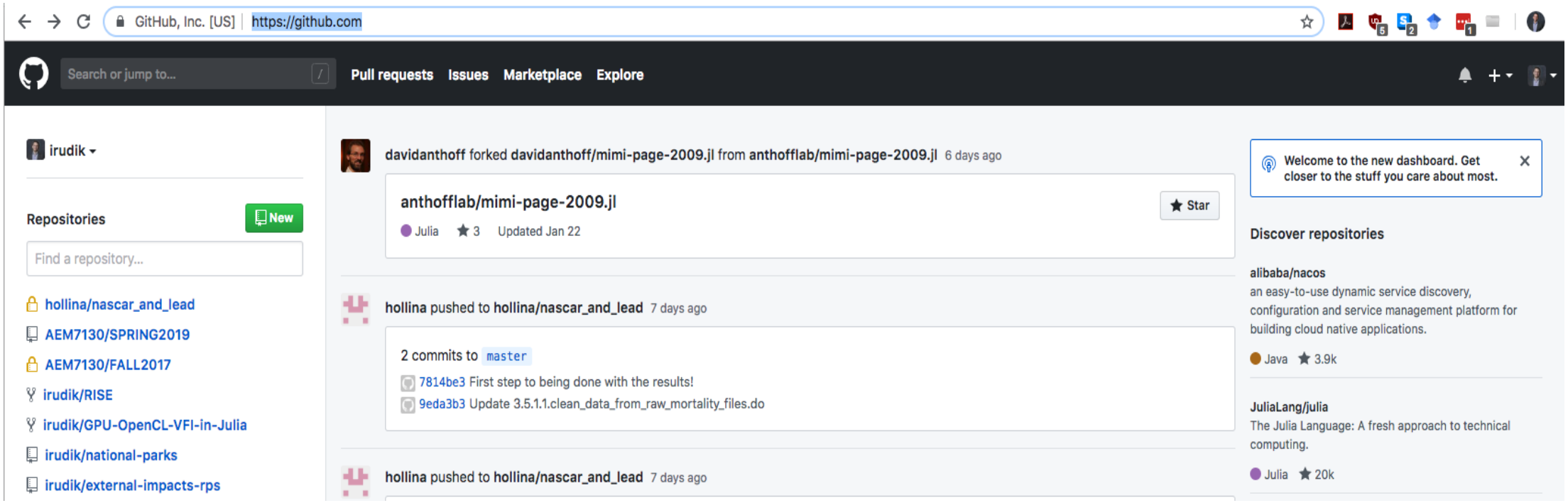
Creating a new repo on GitHub

Let's create a new repo

Creating a new repo on GitHub

Let's create a new repo

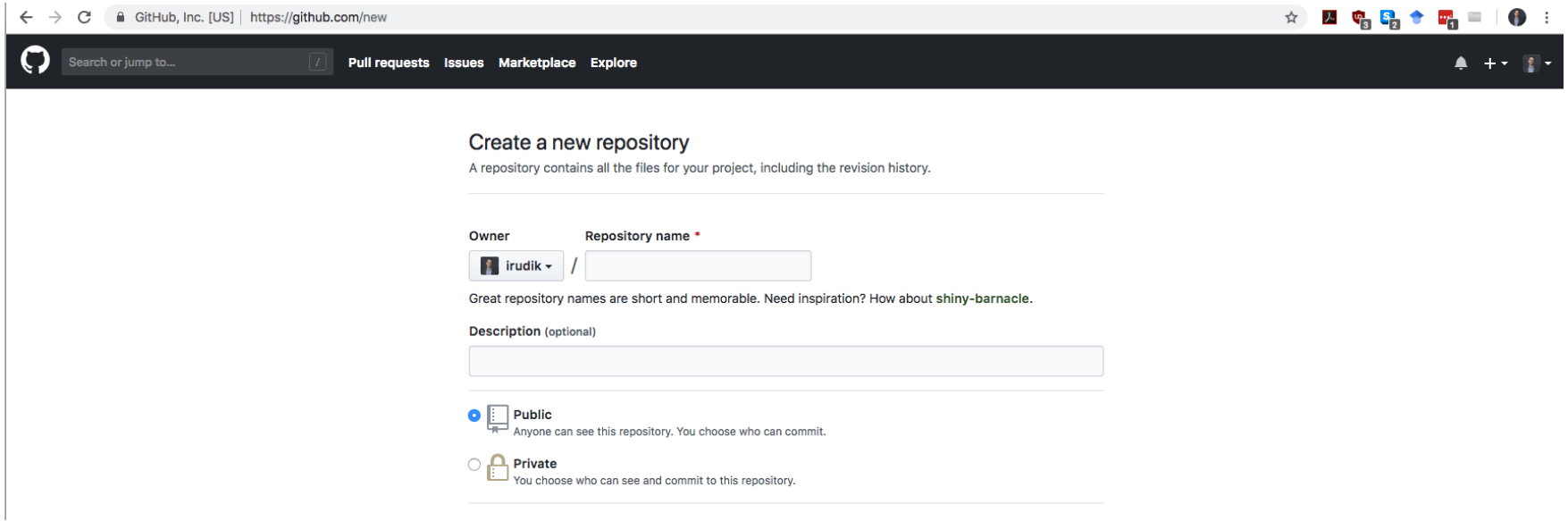
This is pretty easy from the GitHub website: just click on that green `new` button from the launch page



Creating a new repo on GitHub

Next steps:

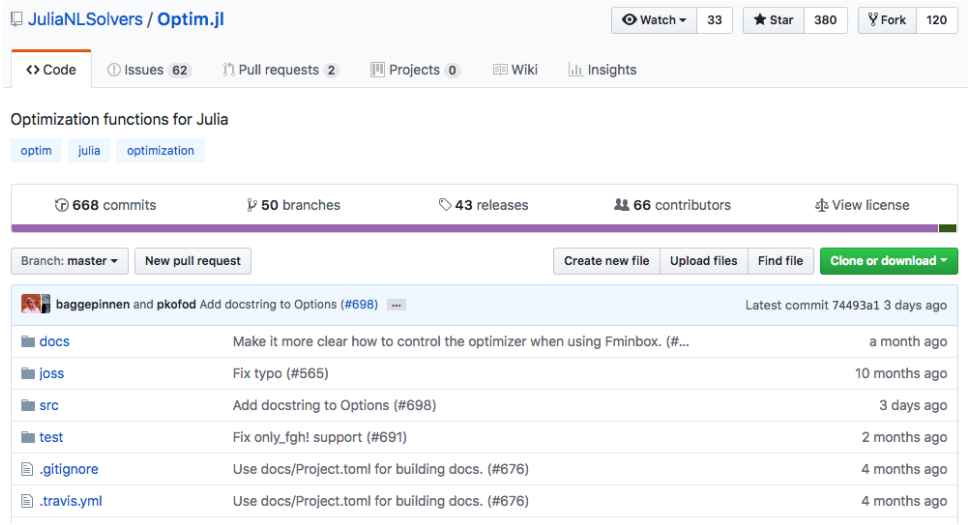
1. Choose a name
2. Choose a description
3. Choose whether the repo is public or private
4. Choose whether you want to add a `README.md` (yes), or a `.gitignore` or a `LICENSE.md` file (more next slide)



Git basics

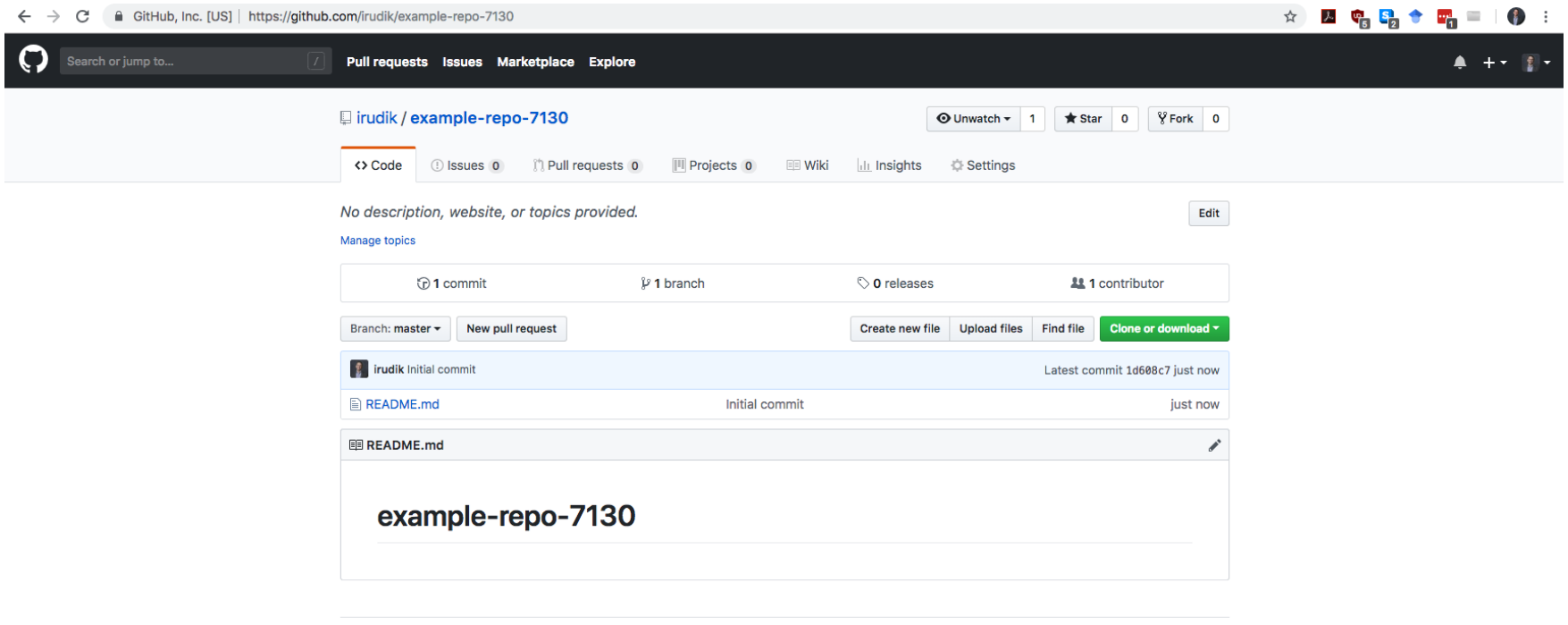
Repos come with some common files in them

- `.gitignore`: lists files/directories/extensions that Git shouldnt track (raw data, restricted data, those weird LaTeX files); this is usually a good idea
- `README.md`: a Markdown file that is basically the welcome content on repo's GitHub website, you should generally initialize a repo with one of these
- `LICENSE.md`: describes the license agreement for the repository



Creating a new repo on GitHub

You can find the repo at https://github.com/irudik/example-repo-7130



How do I get a repo on GitHub onto on my computer?

Clone

To get the repository on your local machine you need to **clone** the repo, you can do this in a few ways from the repo site

How do I get a repo on GitHub onto on my computer?

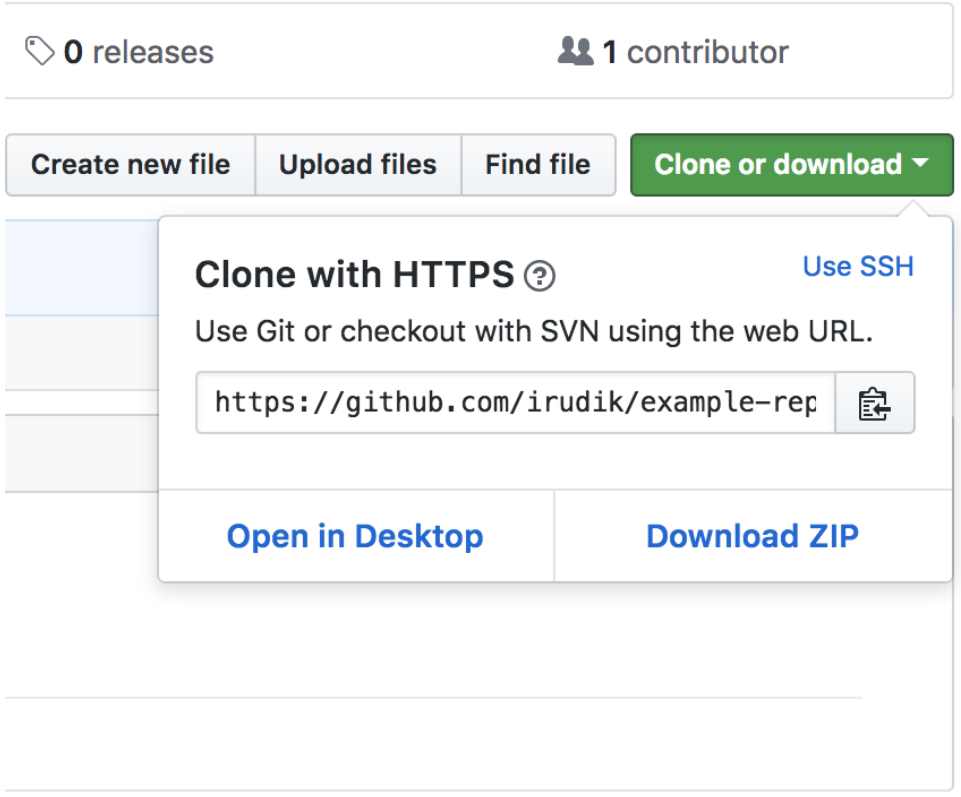
Clone

To get the repository on your local machine you need to **clone** the repo, you can do this in a few ways from the repo site

Key thing: this will **link** your local repository to the remote, you'll be able to update your local when the remote is changed

Cloning

1. If you want to use the GitHub desktop app instead of command line, click on "Open in Desktop"
2. You can use command line `git clone https://github.com/irudik/example-repo-7130.git`



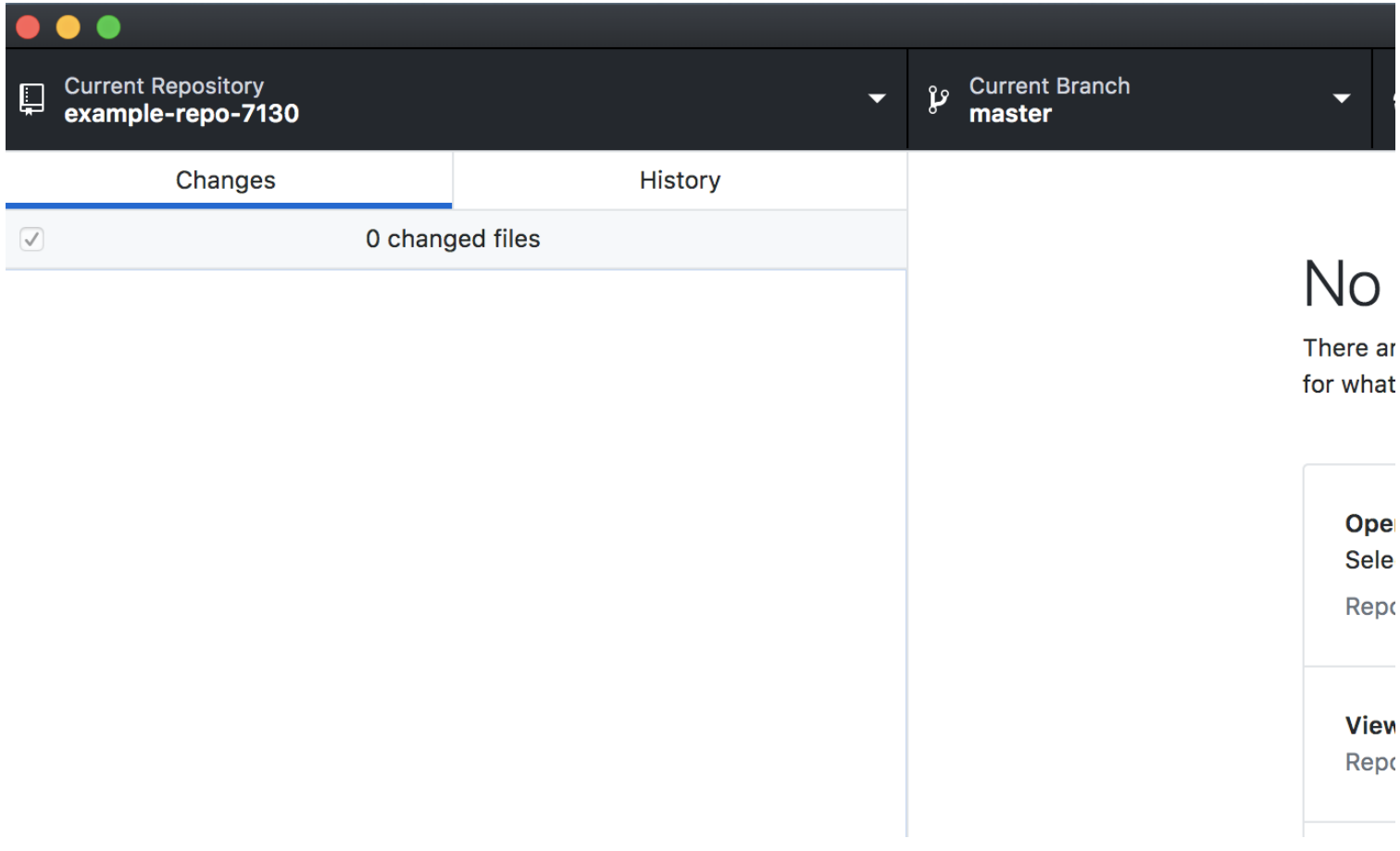
Cloning

You're done! Now create and clone your own repository, initialized with a `README.md`, and follow along.

```
15:04  $ git clone https://github.com/irudik/example-repo-7130.git
Cloning into 'example-repo-7130'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 16 (delta 0), reused 0 (delta 0), pack-reused 13
Unpacking objects: 100% (16/16), done.
~/Desktop/git
15:05  $ ls
ND-Flaring                drillinginfo-data-import    external-impacts-rps       irudik.github.io           optimal-climate-policy-aej  robust-control-jl
aem-7xxx                  dynamic-stochastic-dice     growth-versus-levels       lrr-mcmc-dice              purple_air                  seere-lab
aem7130                   ed_rubin_class              hr_recs                    nascar_and_lead            rc_paper                    steering-the-climate-system
climate-learning-hpc      example-repo-7130           hurricane_forecasts        optimal-climate-policy     rec_markets                 workflow
~/Desktop/git
15:05  $
```

Cloning

You're done! Now create and clone your own repository, initialized with a `README.md`, and follow along.

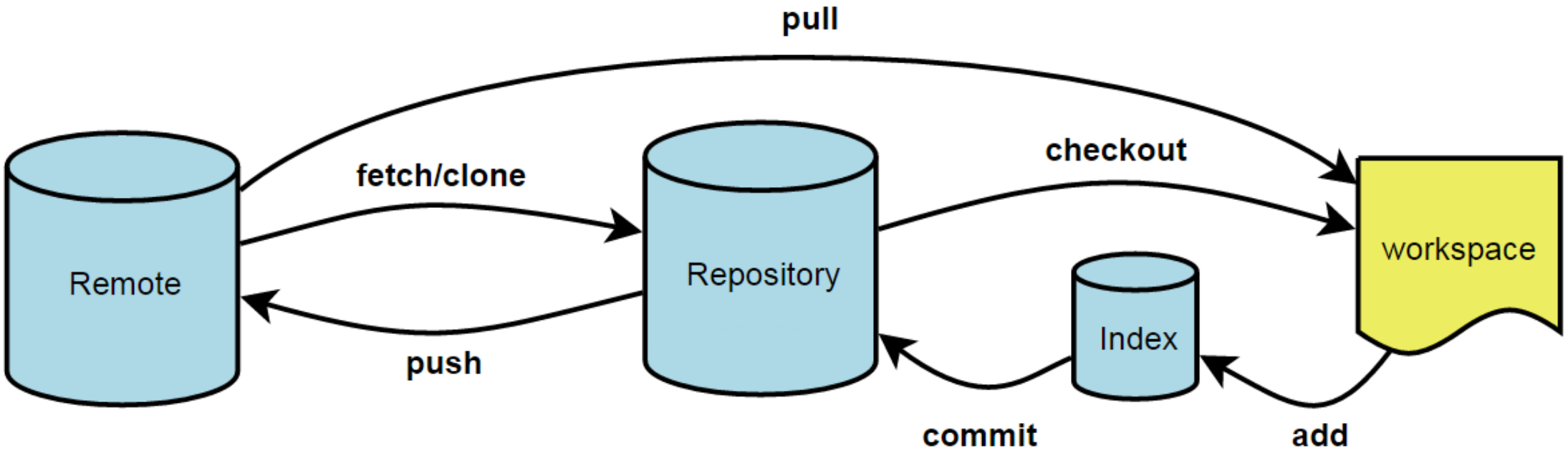


The flow of Git

Workspace: the actual files on your computer

Repository: your saved local history of changes to the files in the repository

Remote: The remote repository on GitHub that allows for sharing across collaborators



Using Git

There are only a few basic Git operations you need to know for versioning solo economics research efficiently

Using Git

There are only a few basic Git operations you need to know for versioning solo economics research efficiently

Add/Stage: This adds files to the index, in other words, it takes a snapshot of the changes you want updated/saved in your local repository (i.e. your computer)

- `git add -A` Adds all files to the index

Using Git

There are only a few basic Git operations you need to know for versioning solo economics research efficiently

Add/Stage: This adds files to the index, in other words, it takes a snapshot of the changes you want updated/saved in your local repository (i.e. your computer)

- `git add -A` Adds all files to the index

Commit: This records the changes to your local repository

- `git commit -m "Updated some files"` Commits the changes added to the index with the commit message in quotations

Using Git

Push: This sends the changes to the remote repository (i.e. GitHub)

- `git push origin master` Pushes changes on your local repo to a **branch** called `master` on your remote, typically named `origin` (can often omit `origin master`)

Using Git

Push: This sends the changes to the remote repository (i.e. GitHub)

- `git push origin master` Pushes changes on your local repo to a **branch** called `master` on your remote, typically named `origin` (can often omit `origin master`)

Pull: This takes changes on the remote and integrates them with the local repository (technically two operations are going on: fetch and merge)

- `git pull origin master` Integrates the changes on the `master` branch of your remote `origin` into your local repo (again, can often omit `origin master`)

Using Git

In your own repository do the following using either shell or GitHub Desktop:

Using Git

In your own repository do the following using either shell or GitHub Desktop:

1. Open `README.md` in some text editor and insert the following code: `# Hello World!`
2. Save `README.md`
3. Add the changes to `README.md` to the index
4. Commit the changes to your local repo with the message: "First README.md edit."
5. Push the changes to your remote

Using Git

In your own repository do the following using either shell or GitHub Desktop:

1. Open `README.md` in some text editor and insert the following code: `# Hello World!`
2. Save `README.md`
3. Add the changes to `README.md` to the index
4. Commit the changes to your local repo with the message: "First README.md edit."
5. Push the changes to your remote

Did the changes show up your repo's GitHub page?

Using Git: branching

Some more (but not very) advanced operations relate to **branching**

Branching creates different, but parallel, versions of your code

e.g. If you want to test out a new feature of your model but don't want to contaminate your `master` branch, create a new branch and add the feature there

If it works out, you can bring the changes back into `master`

If it doesn't, just delete it

Using Git: branching

Branch: This adds/deletes/merges different **branches** of your repository

- `git branch` Lists all local branches
- `git branch -a` Lists all remote branches
- `git branch solar-panels` Creates a new branch called `solar-panels`
- `git branch -d solar-panels` Deletes the local `solar-panels` branch

Using Git: branching

Checkout: This switches you between different commits or branches

- `git checkout solar-panels` Switches you to branch `solar-panels`
- `git checkout -b wind-turbines` Creates a new branch called `wind-turbines` and checks it out

Using Git: branching

Merge: This merges two separate histories together (e.g. merges a separate branch back into the master)

- `git checkout master`

  `git merge wind-turbines`

  Checks out `master` and then merges `wind-turbines` back into the master

This brings the changes from `wind-turbines` since the initial branch back into the `master` branch

Using Git

In your own repository do the following using either shell or GitHub Desktop:

Using Git

In your own repository do the following using either shell or GitHub Desktop:

1. Create and checkout a new branch called `test-branch`
2. Edit `README.md` and add the following code: `## your_name_here`
3. Save `README.md`
4. Add the changes to `README.md` to the index
5. Commit the changes to your local repo with the message: "Test change to README.md."
6. Merge the changes back into the `master` branch
7. Push the changes to your remote

Using Git

In your own repository do the following using either shell or GitHub Desktop:

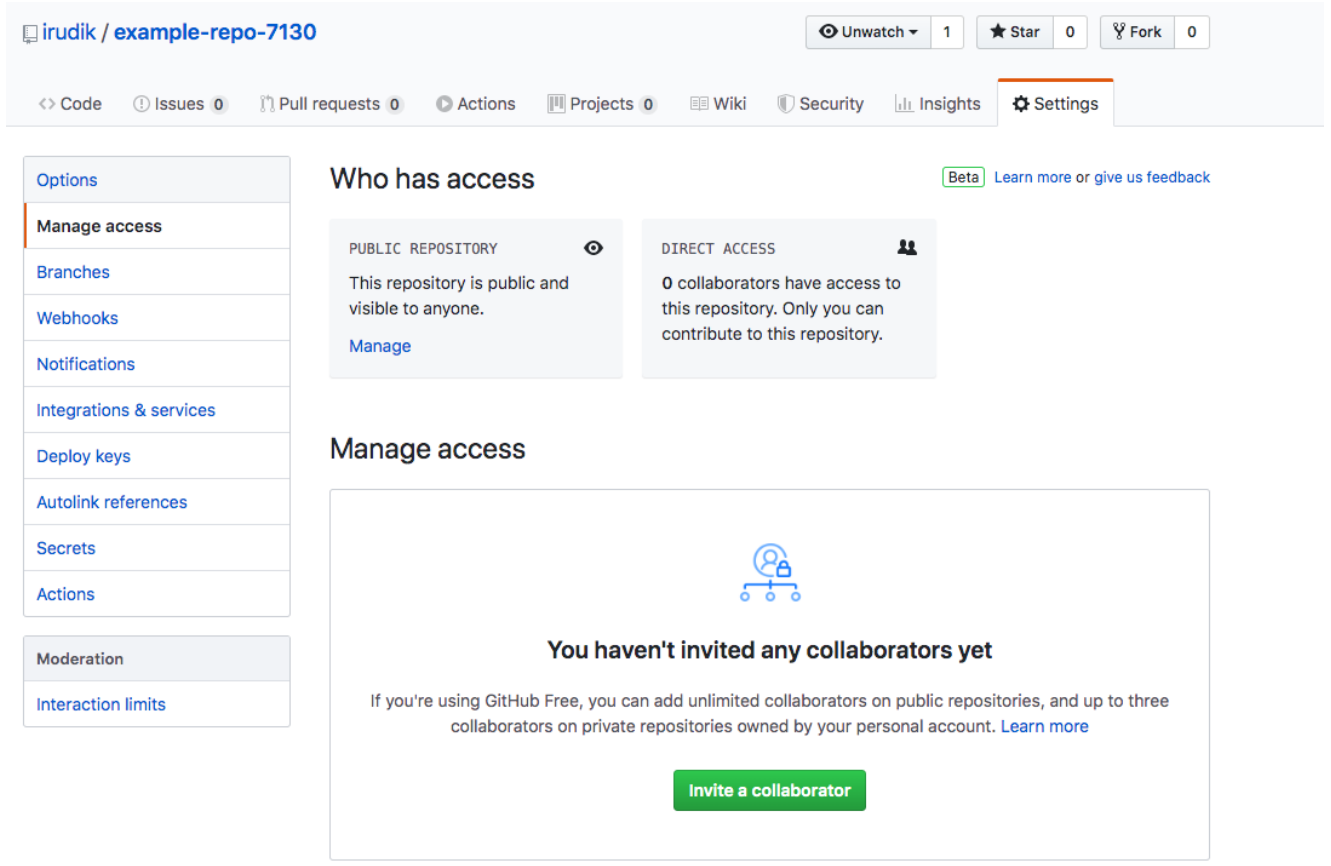
1. Create and checkout a new branch called `test-branch`
2. Edit `README.md` and add the following code: `## your_name_here`
3. Save `README.md`
4. Add the changes to `README.md` to the index
5. Commit the changes to your local repo with the message: "Test change to README.md."
6. Merge the changes back into the `master` branch
7. Push the changes to your remote

Did the changes show up your repo's GitHub page?

Teaming up

Find a partner for this next piece:

One of you invite the other to collaborate on the project (GitHub page → Settings → Manage access → invite a collaborator)



Teaming up

If you were the one being invited, accept the invite, and clone the repo to your local

Teaming up

If you were the one being invited, accept the invite, and clone the repo to your local

Now do the following:

1. Each of you edit the `# Hello World!` line of code to be something else and different from each other

Teaming up

If you were the one being invited, accept the invite, and clone the repo to your local

Now do the following:

1. Each of you edit the `# Hello World!` line of code to be something else and different from each other
2. Commit the changes to your local

Teaming up

If you were the one being invited, accept the invite, and clone the repo to your local

Now do the following:

1. Each of you edit the `# Hello World!` line of code to be something else and different from each other
2. Commit the changes to your local
3. Have the repo creator push their changes

Teaming up

If you were the one being invited, accept the invite, and clone the repo to your local

Now do the following:

1. Each of you edit the `# Hello World!` line of code to be something else and different from each other
2. Commit the changes to your local
3. Have the repo creator push their changes
4. Have the collaborator push their changes

Can't push changes when you aren't updated

Shell

```
ag-aem-1bhhv2r:ivan_git ir229$ git push
To https://github.com/hollina/nascar_and_lead.git
 ! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'https://github.com/hollina/nascar_and_lead.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
ag-aem-1bhhv2r:ivan_git ir229$
```

It turns out that the second person can't push their local changes to the remote

The second person is pushing their history of changes

But the remote is already one commit ahead because of the first person, so the second person's changes can't be pushed

Update by pulling after you commit local changes

You need to pull the remote changes first, but then you get the following message:

```
ag-aem-1bhhv2r:ivan_git ir229$ git pull origin master
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), 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.
From https://github.com/hollina/nascar_and_lead
 * branch            master     -> FETCH_HEAD
   3ec89b2..03c774b  master     -> origin/master
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
Automatic merge failed; fix conflicts and then commit the result.
ag-aem-1bhhv2r:ivan_git ir229$
```

And we got a **merge conflict** in `README.md`

This means there were differences between the remote and your local that conflicted

Merge conflicts

Sometimes there will be conflicts between two separate histories

- e.g. if you and your collaborator edited the same chunk of code separately on your local repos

Merge conflicts

Sometimes there will be conflicts between two separate histories

- e.g. if you and your collaborator edited the same chunk of code separately on your local repos

When you try to merge these histories by pushing to the remote, Git will throw a **merge conflict**

Merge conflicts

Sometimes there will be conflicts between two separate histories

- e.g. if you and your collaborator edited the same chunk of code separately on your local repos

When you try to merge these histories by pushing to the remote, Git will throw a **merge conflict**

When you get a merge conflict, the conflicted part of the code in your file will look like:

```
$ <<<<<<< HEAD
$ # nascar_and_unleaded ⟵ my local version
$ =======
$ # nascar_and_leaded ⟵ the remote version
$ >>>>>>> 03c774b0e9baff0230855822a11e6ed24a0aa6b2
```

Merge conflicts

```
$ <<<<<<< HEAD
$ # nascar_and_unleaded ⟵ my local version
$ =======
$ # nascar_and_leaded ⟵ the remote version
$ >>>>>>> 03c774b0e9baff0230855822a11e6ed24a0aa6b2
```

`<<<<<<< HEAD` indicates the start of the conflicted code

Merge conflicts

```
$ <<<<<<< HEAD
$ # nascar_and_unleaded ⟵ my local version
$ =======
$ # nascar_and_leaded ⟵ the remote version
$ >>>>>>> 03c774b0e9baff0230855822a11e6ed24a0aa6b2
```

`<<<<<<< HEAD` indicates the start of the conflicted code

`=======` separates the two different conflicting histories

Merge conflicts

```
$ <<<<<<< HEAD
$ # nascar_and_unleaded ⟵ my local version
$ =======
$ # nascar_and_leaded ⟵ the remote version
$ >>>>>>> 03c774b0e9baff0230855822a11e6ed24a0aa6b2
```

`<<<<<<< HEAD` indicates the start of the conflicted code

`=======` separates the two different conflicting histories

`>>>>>>> lots of numbers and letters` indicates the end of the conflicted code and the hash (don't worry about it) for the specific commit

Fixing the merge conflict

Merge conflicts can be fixed by directly editing the file, then doing an `add` of the conflicted file, a `commit`, and then a `push` to the remote

```
[ag-aem-1bhhv2r:ivan_git ir229$ git commit -am "Fixed merge conflict for README.md title."
 [master f697e22] Fixed merge conflict for README.md title.
[ag-aem-1bhhv2r:ivan_git ir229$ git push origin master
Counting objects: 6, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (6/6), 784 bytes | 784.00 KiB/s, done.
Total 6 (delta 0), reused 0 (delta 0)
To https://github.com/hollina/nascar_and_lead.git
   03c774b..f697e22  master -> master
ag-aem-1bhhv2r:ivan_git ir229$
```

Fixing the merge conflict

Merge conflicts can be fixed by directly editing the file, then doing an `add` of the conflicted file, a `commit`, and then a `push` to the remote

```
[ag-aem-1bhhv2r:ivan_git ir229$ git commit -am "Fixed merge conflict for README.md title."
 [master f697e22] Fixed merge conflict for README.md title.
[ag-aem-1bhhv2r:ivan_git ir229$ git push origin master
Counting objects: 6, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (6/6), 784 bytes | 784.00 KiB/s, done.
Total 6 (delta 0), reused 0 (delta 0)
To https://github.com/hollina/nascar_and_lead.git
   03c774b..f697e22  master -> master
ag-aem-1bhhv2r:ivan_git ir229$
```

Fixed!

Git help pages are excellent, so is StackExchange

```
$ git help add
```

```
GIT-ADD(1)                                                        Git Manual                                                        GIT-ADD(1)

NAME
       git-add - Add file contents to the index

SYNOPSIS
       git add [--verbose | -v] [--dry-run | -n] [--force | -f] [--interactive | -i] [--patch | -p]
                 [--edit | -e] [--[no-]all | --[no-]ignore-removal | [--update | -u]]
                 [--intent-to-add | -N] [--refresh] [--ignore-errors] [--ignore-missing]
                 [--chmod=(+|-)x] [--] [<pathspec>...]

DESCRIPTION
       This command updates the index using the current content found in the working tree, to prepare the content staged for the next commit. It typically adds the current content of
       existing paths as a whole, but with some options it can also be used to add content with only part of the changes made to the working tree files applied, or remove paths that do not
       exist in the working tree anymore.

       The "index" holds a snapshot of the content of the working tree, and it is this snapshot that is taken as the contents of the next commit. Thus after making any changes to the working
       tree, and before running the commit command, you must use the add command to add any new or modified files to the index.

       This command can be performed multiple times before a commit. It only adds the content of the specified file(s) at the time the add command is run; if you want subsequent changes
       included in the next commit, then you must run git add again to add the new content to the index.

       The git status command can be used to obtain a summary of which files have changes that are staged for the next commit.

       The git add command will not add ignored files by default. If any ignored files were explicitly specified on the command line, git add will fail with a list of ignored files. Ignored
       files reached by directory recursion or filename globbing performed by Git (quote your globs before the shell) will be silently ignored. The git add command can be used to add ignored
       files with the -f (force) option.

       Please see git-commit(1) for alternative ways to add content to a commit.

OPTIONS
       <pathspec>...
           Files to add content from. Fileglobs (e.g.  *.c) can be given to add all matching files. Also a leading directory name (e.g.  dir to add dir/file1 and dir/file2) can be given to
           update the index to match the current state of the directory as a whole (e.g. specifying dir will record not just a file dir/file1 modified in the working tree, a file dir/file2
           added to the working tree, but also a file dir/file3 removed from the working tree. Note that older versions of Git used to ignore removed files; use --no-all option if you want
           to add modified or new files but ignore removed ones.

           For more details about the <pathspec> syntax, see the pathspec entry in gitglossary(7).

       -n, --dry-run
           Don't actually add the file(s), just show if they exist and/or will be ignored.

       -v, --verbose
           Be verbose.

       -f, --force
           Allow adding otherwise ignored files.

       -i, --interactive
           Add modified contents in the working tree interactively to the index. Optional path arguments may be supplied to limit operation to a subset of the working tree. See "Interactive
           mode" for details.

       -p, --patch
           Interactively choose hunks of patch between the index and the work tree and add them to the index. This gives the user a chance to review the difference before adding modified
           contents to the index.

           This effectively runs add --interactive, but bypasses the initial command menu and directly jumps to the patch subcommand. See "Interactive mode" for details.

       -e, --edit
           Open the diff vs. the index in an editor and let the user edit it. After the editor was closed, adjust the hunk headers and apply the patch to the index.

           The intent of this option is to pick and choose lines of the patch to apply, or even to modify the contents of lines to be staged. This can be quicker and more flexible than using
           the interactive hunk selector. However, it is easy to confuse oneself and create a patch that does not apply to the index. See EDITING PATCHES below.

       -u, --update
           Update the index just where it already has an entry matching <pathspec>. This removes as well as modifies index entries to match the working tree, but adds no new files.

           If no <pathspec> is given when -u option is used, all tracked files in the entire working tree are updated (old versions of Git used to limit the update to the current directory
           and its subdirectories).

       -A, --all, --no-ignore-removal
```

Managing tasks and workflow

GitHub is also very useful for task management in solo or group projects using **issues** and **pull requests**

Issues: task management for you and your collaborators, should be able to completely replace email

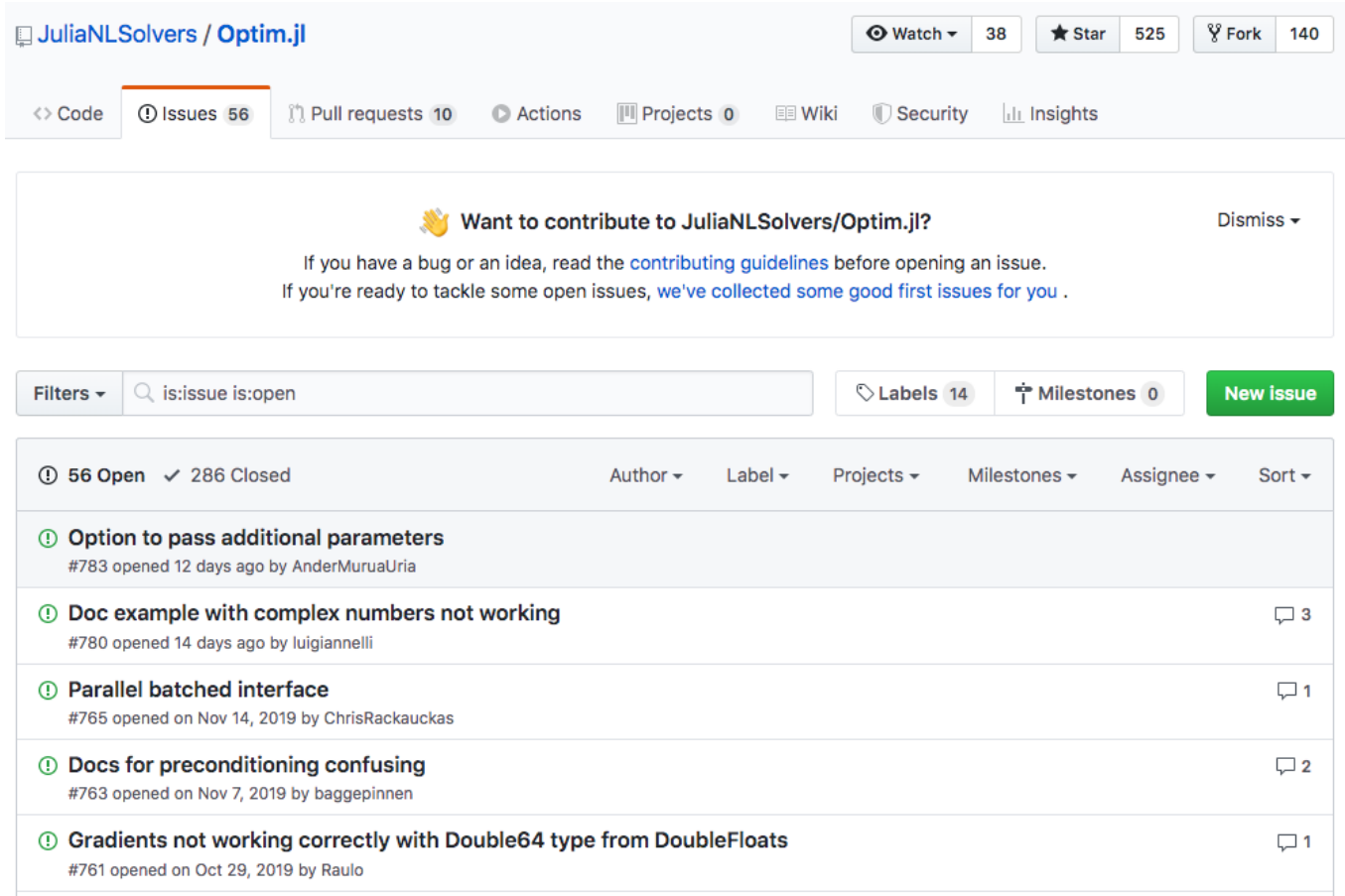
Let's look at the issues for the `Optim` package in Julia

Issues

The issues tab reports a list of 56 open issues (286 closed, meaning the task or problem has been solved)

Each issue has its own title

Lets check out the issue about the `Double64` type



Issues

The issue is because one person has found an error with the package where it doesn't seem to work correctly with a certain type of variable `Double64`

Someone else has responded with some feedback

JuliaNLSolvers / Optim.jl

Watch 38 · Star 525 · Fork 140

Code · Issues 56 · Pull requests 10 · Actions · Projects 0 · Wiki · Security · Insights

Gradients not working correctly with Double64 type from DoubleFloats #761

New issue

Open · Raulo opened this issue on Oct 29, 2019 · 1 comment

Raulo commented on Oct 29, 2019

Any gradient code would not work correctly with Double64 float

```
function f(x)
       return (1.0 - x[1])^2 + 100.0 * (x[2] - x[1]^2)^2
end
function g!(G, x)
       G[1] = -2.0 * (1.0 - x[1]) - 400.0 * (x[2] - x[1]^2) * x[1]
       G[2] = 200.0 * (x[2] - x[1]^2)
end
using Optim
using DoubleFloats
initial_x=[Double64(0.0),Double64(0.0)]
optimize(f, g!, initial_x, method=BFGS(),show_trace=true)
```

It will crash with `ERROR: ArgumentError: Value and slope at step length = 0 must be finite.` after not realizing the procedure converged and showing NaNs as Gradient norm: `24 3.391382e-204 NaN` . Using Double64 type constants in f(x) and g(x) does not help. It will work without explicit gradient:

```
optimize(f, initial_x, method=BFGS(),show_trace=true)
```

but the gradient norm will be printed as NaN, too.

The code works correctly with BigFloat, ArbFloat from ArbNumerics, and Float128 from Quadmath, so it may be an obscure bug in DoubleFloats.

pkofod commented on Dec 20, 2019 · Collaborator

That's interesting... So it does seem to work, but something is weird with the tracing (I adde print statements in the g code)

```
julia> optimize(f, g!, initial_x, method=BFGS(),show_trace=true)
```

Assignees: No one assigned

Labels: None yet

Projects: None yet

Milestone: No milestone

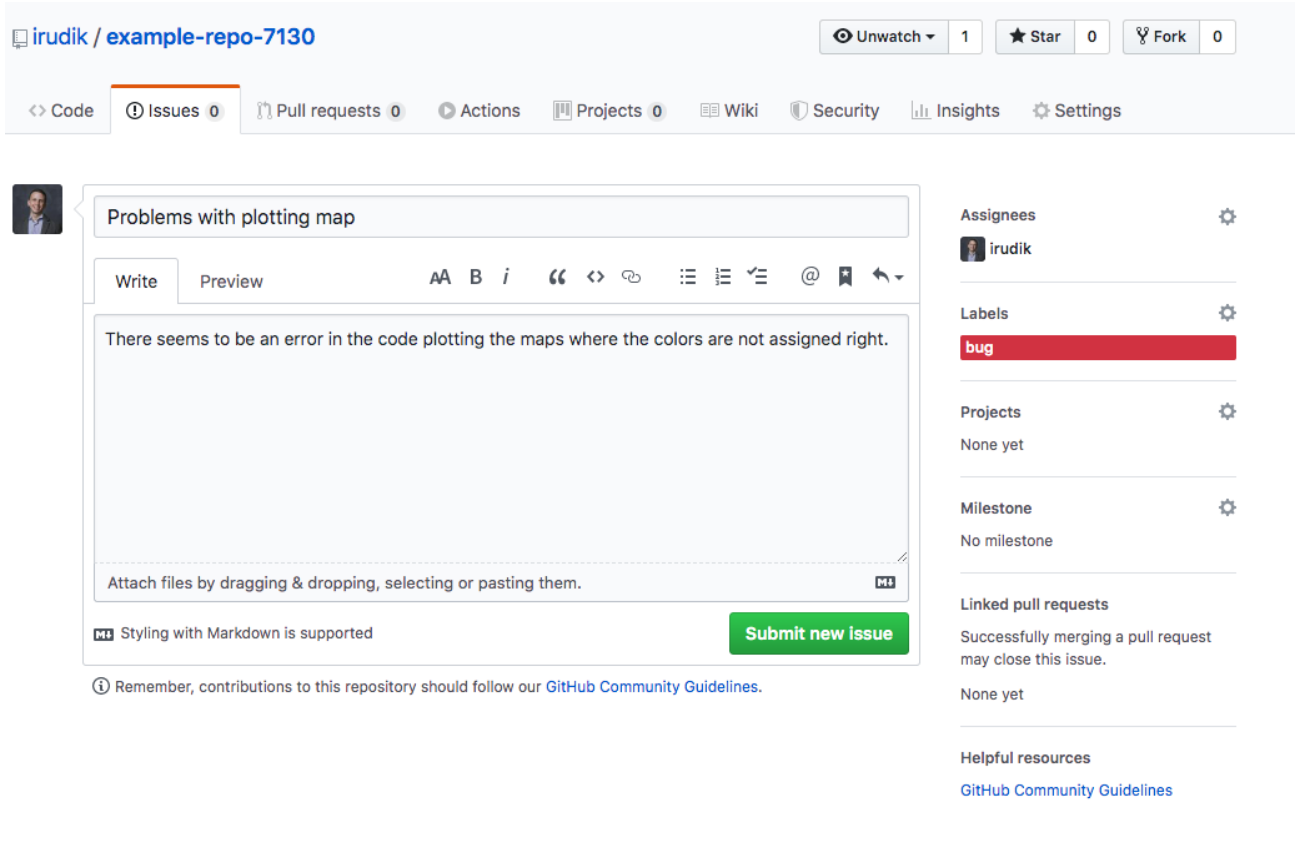
Linked pull requests: Successfully merging a pull request may close this issue. None yet

Notifications: Customize · Subscribe · You're not receiving notifications from this thread.

2 participants

Issues

From the issues tab, click the green **new issue** button which takes you here

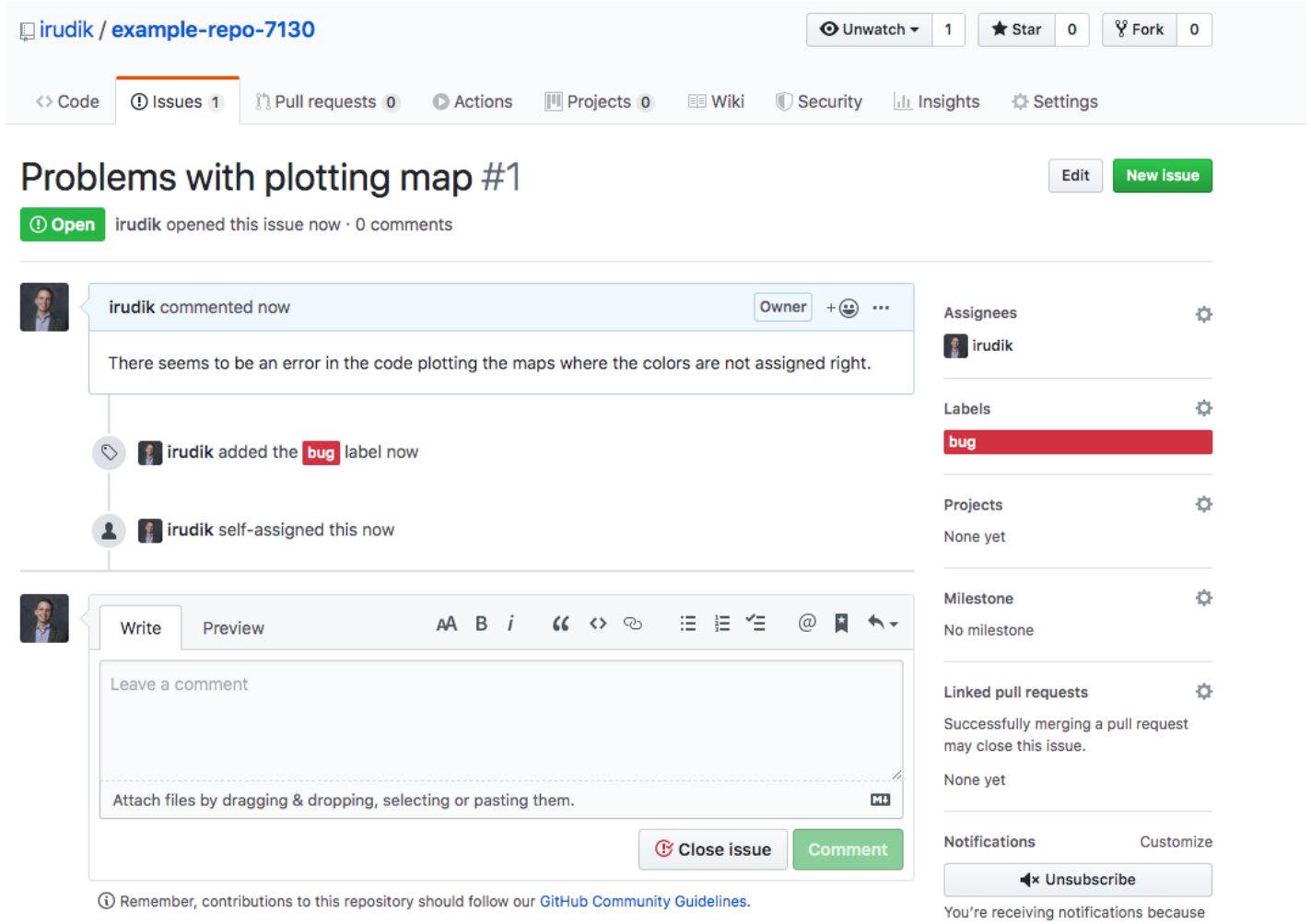


You can:

- create a title
- add some text for the body of the issue
- select people to assign the issue to
- add some labels

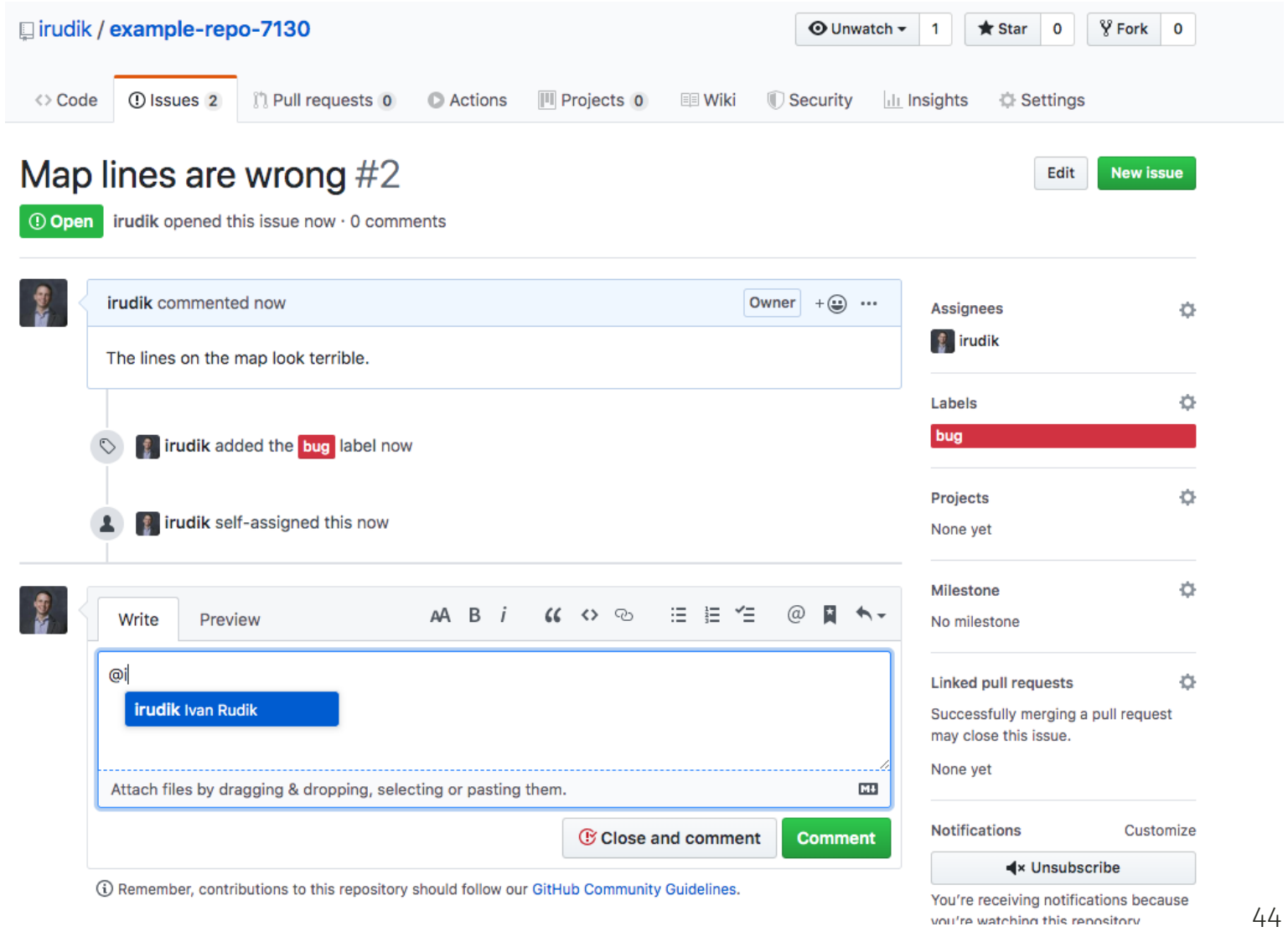
Issues

The issue keeps track of the history of everything that's happened to it



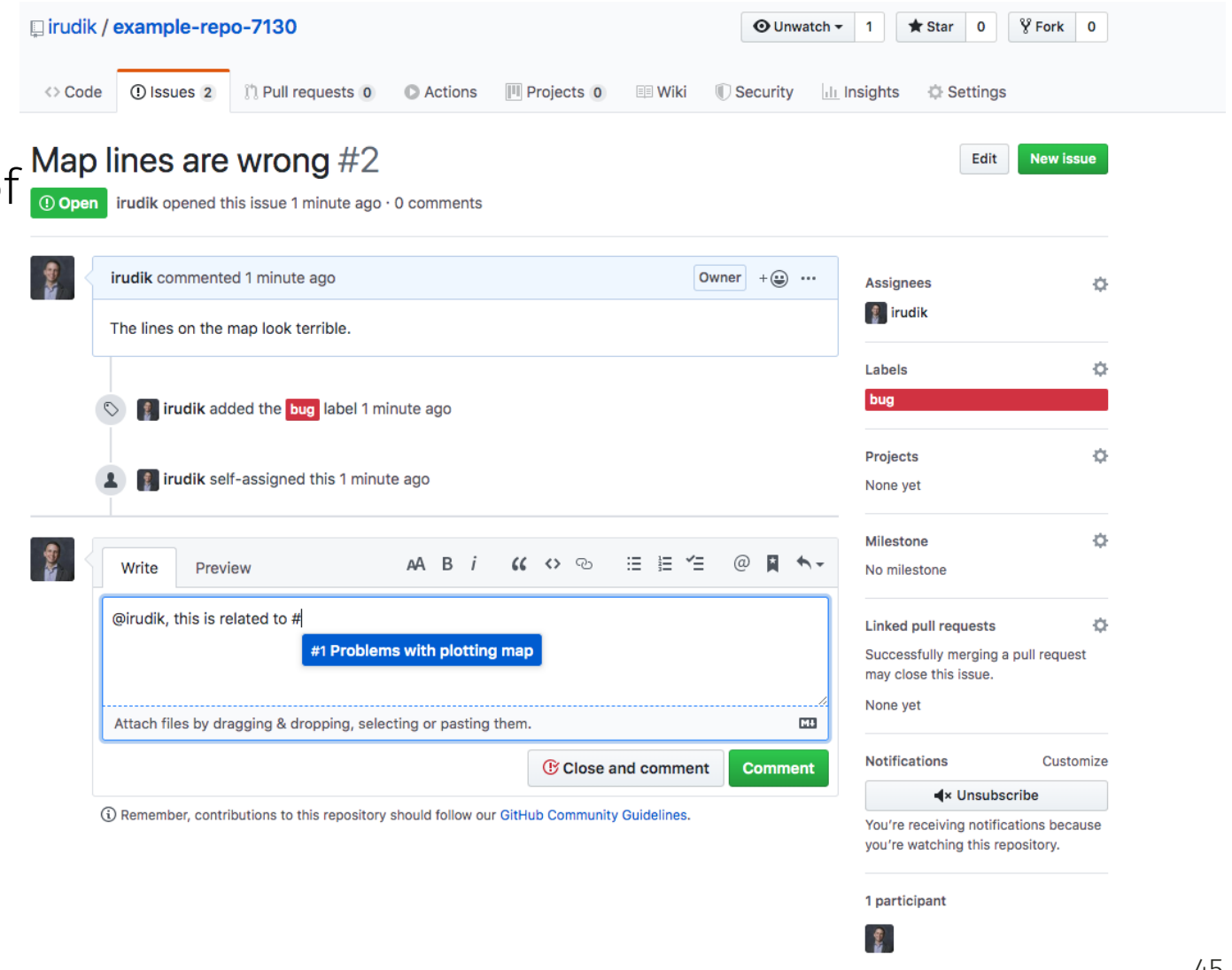
Issues

You can reference people with `@` which brings up a dropdown menu of all collaborators on the project



Issues

You can also reference *other issues* if they're related by using `#` which brings up a dropdown of all issues for your repository



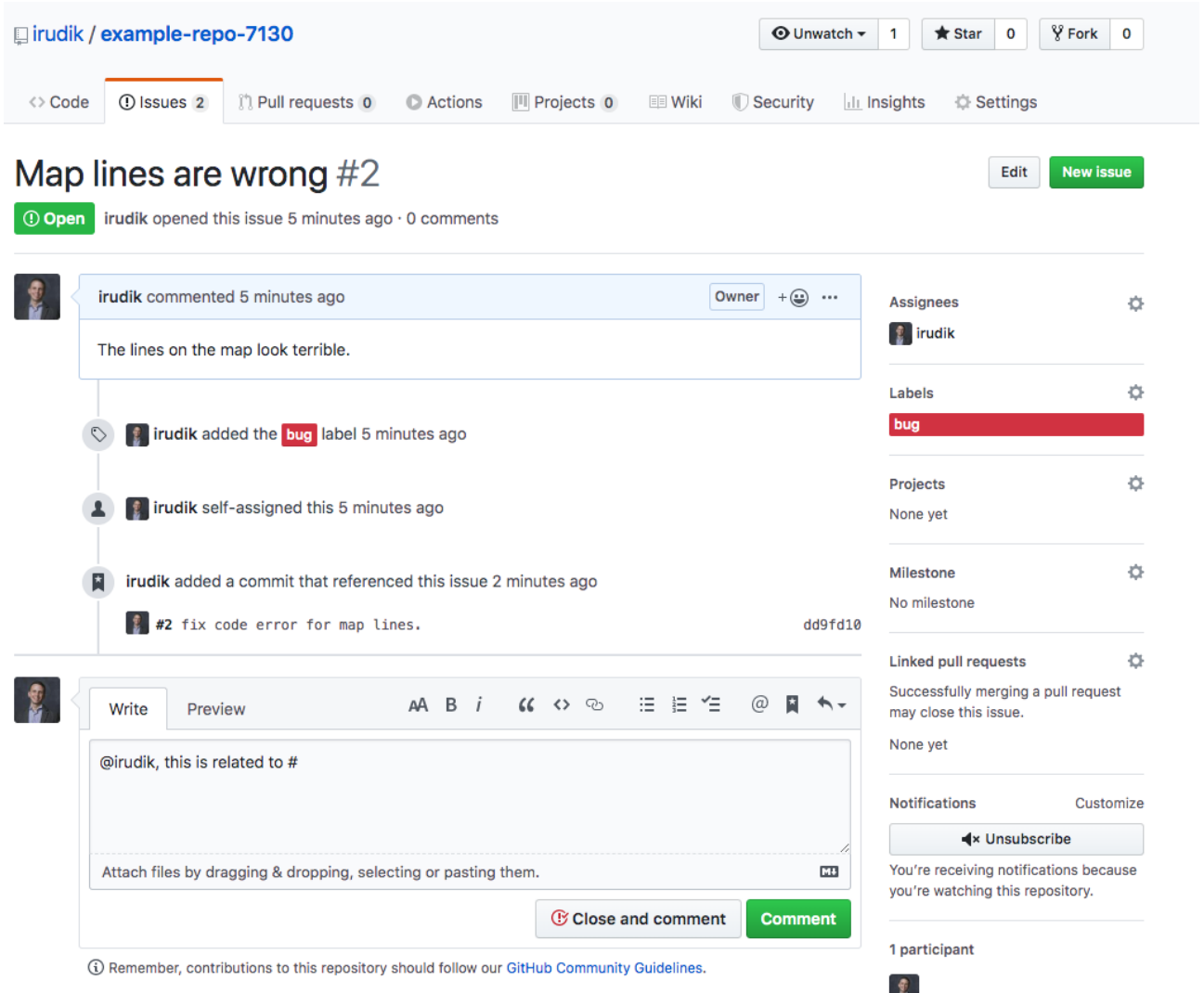
Issues

Issues can also be referenced in your commits to your project by adding `#issue_number_here` to the commit message

```
~/Desktop/git/example-repo-7130 [master|● 1]
16:21  $ git cm "#2 fix code error for map lines."
[master dd9fd10] #2 fix code error for map lines.
 1 file changed, 2 insertions(+)
~/Desktop/git/example-repo-7130 [master ↑·1|✔]
16:21  $ git push
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 315 bytes | 315.00 KiB,
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/irudik/example-repo-7130.git
   06e2f7a..dd9fd10  master -> master
```

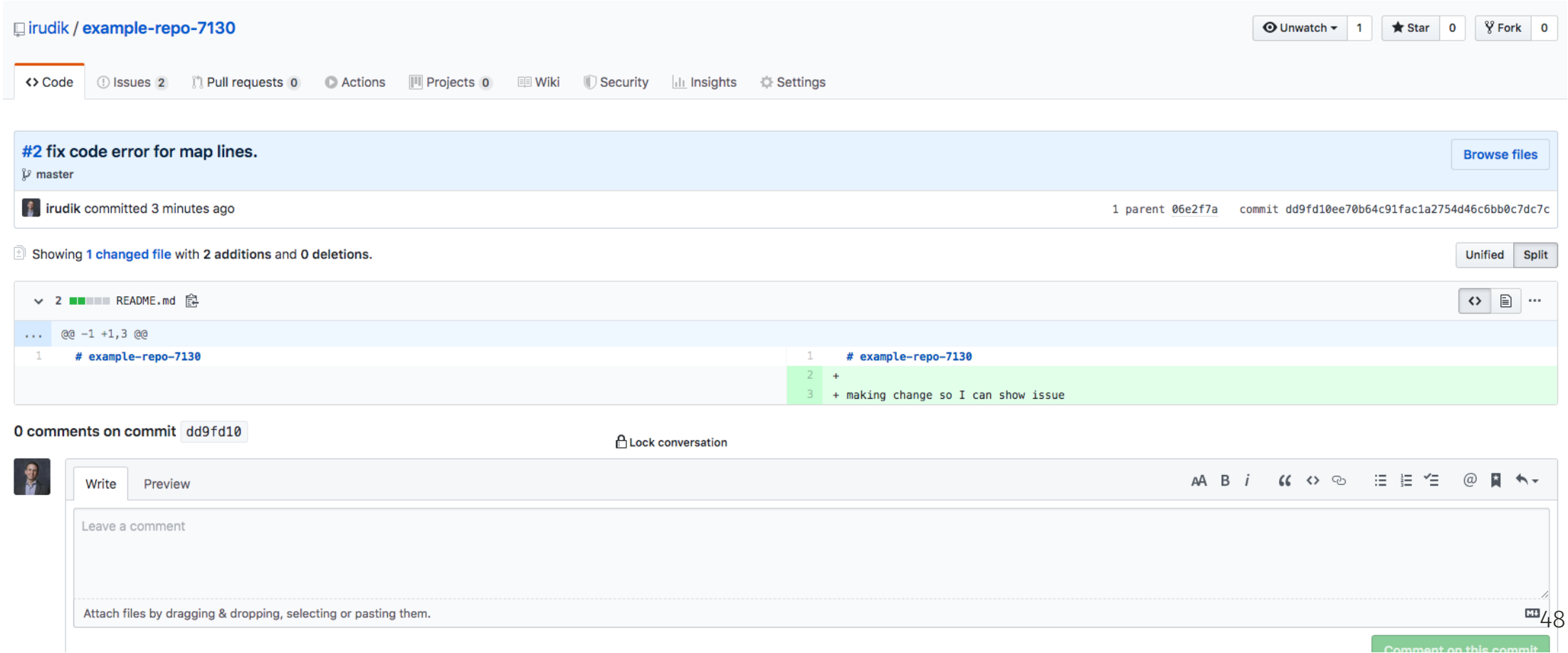
Issues

Then those commits show up in your issue so you have a history of what code changes have been made.



Issues

If you click on the commit, it takes you to the `git diff` which shows you any changes to files made in that commit



Windows users!!!!!

Do the following:

- Open up a command prompt or Git Bash (recommend Bash from here on out)
- Run the following commands:

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

Windows users!!!!!

Do the following:

- Open up a command prompt or Git Bash (recommend Bash from here on out)
- Run the following commands:

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

Git tracks changes by specific characters at the end of each line, it does this differently across Windows/Unix by default

Next up:

Optimization: root-finding and maximization/minimization