TAM Bootcamp - git

Login to github.com : Create your account and use your personal account credentials

Getting Started (Install git)

On Windows (if you are a windows user)

- Open a terminal session
- Install git via the procedure here
    - http://git-scm.com/download/win

On a Mac (if you are a mac user)

- On your laptop (this example is on the mac)
    - Open a terminal session

- Install homebrew (if you don't have it installed) on your mac
    - You can get the instructions here:
        - https://brew.sh/
    - These are also the commands for brew:

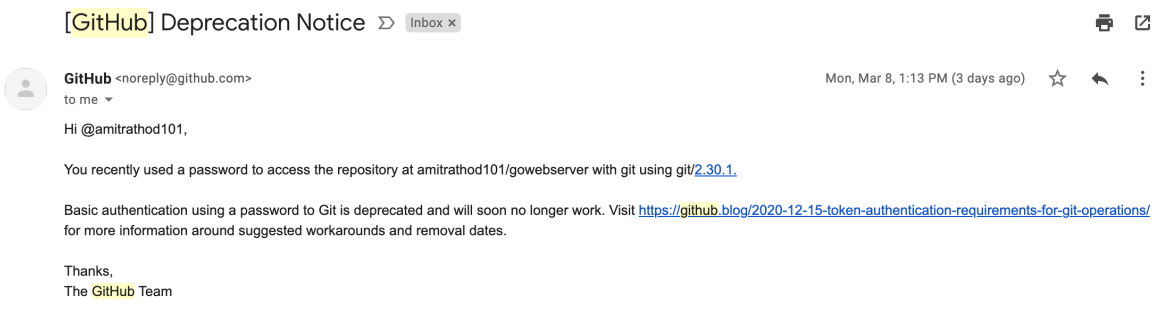
      % brew update
      % brew doctor

- If this is the first time you are using git on a mac, remember Apple comes with its own version of git, let's take a look:
    - Enter this command

      % git –version
    - You should see something similar to:

      Git version 2.24.1 (Apple Git-126)

    - If you see this, you are running the apple version of git, not the official distribution
    - Check your primary email used on github and you will see a mail like this.



[GitHub] Deprecation Notice Inbox ×

GitHub <noreply@github.com> Mon, Mar 8, 1:13 PM (3 days ago)
to me

Hi @amitrathod101,

You recently used a password to access the repository at amitrathod101/gowebserver with git using git/2.30.1.

Basic authentication using a password to Git is deprecated and will soon no longer work. Visit https://github.blog/2020-12-15-token-authentication-requirements-for-git-operations/ for more information around suggested workarounds and removal dates.

Thanks,
The GitHub Team

    - Next, install the git client with this command:

      % brew install git
    - Brew installs the official git distro in /usr/local/bin

- So far, so good. Now you have to change your path to use the official git distribution
- Now enter this command:
  % export PATH=/usr/local/bin:$PATH
- Confirm that the git version is correct by entering this command:
  % git –version
- You should see that the version of git has been upgraded
  git version 2.30.0 or maybe better.

- In order to upgrade git (if you need to in the future)
  % brew upgrade git

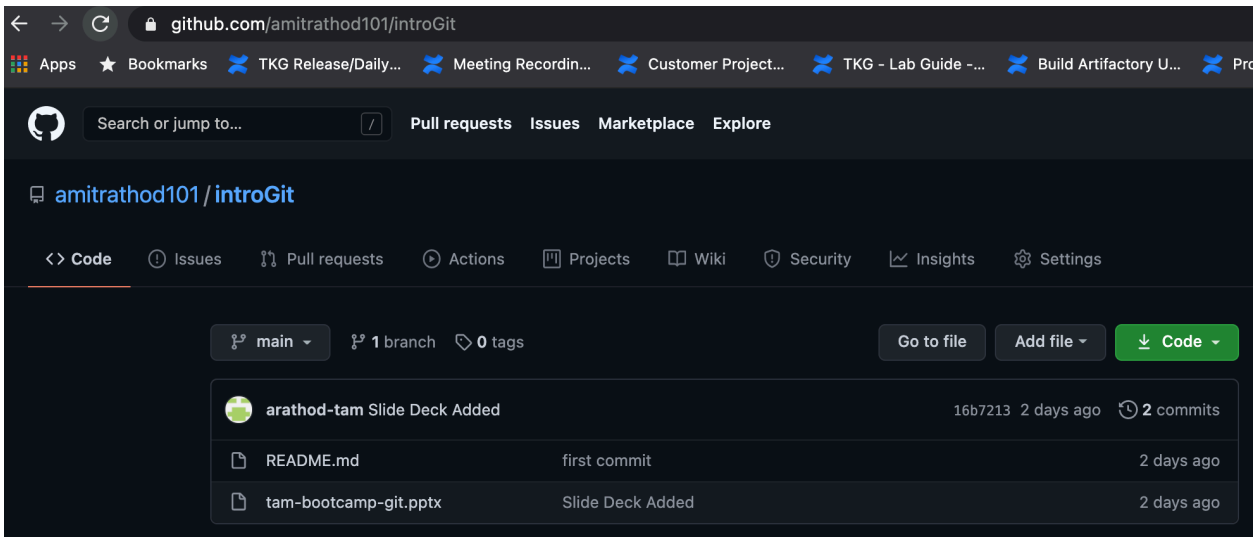
In a browser, go to https://github.com/amitrathod101/introGit

Basic Commands

Fork the introGit repository

(Fork creates your own copy of the repository that you can use for the rest of the lab)

- Login to your github.com repository
- Go to amitrathod101/introGit



- In the upper right hand corner, select “Fork”
- When it asks you where to fork introGit, select your personal repository
- Now you should see “introGit” under your repositories

Clone the sample project in your repository onto your laptop

- On your laptop, create a directory and cd into the directory
  % cd /users/<your home directory>
  % mkdir github.com
  % cd github.com
  % mkdir <the name of your github account>
  % cd <the name of your github account>

- Now you will make a local copy of the introGit repository
    - Enter the git clone command (your command will vary)

      % git clone https://github.com/<the name of your github account>/introGit.git

    - You should see output similar to:

      Cloning into 'introGit'...
      remote: Enumerating objects: 6, done.
      remote: Counting objects: 100% (6/6), done.
      remote: Compressing objects: 100% (5/5), done.
      remote: Total 6 (delta 0), reused 0 (delta 0), pack-reused 0
      Receiving objects: 100% (6/6), done.

    - Change directory into the newly created introGit

      % cd introGit

    - Look at the contents of the folder, notice the hidden .git.

      % ls -lah

    - Do a git status. git status will be used often to provide information on your current state

      % git status

      On branch main
      Your branch is up to date with 'origin/main'.

      nothing to commit, working tree clean

Modify/Checking a file

(The following few commands (status/add/commit/push) show you how to make changes to a file and check them into your repository in git)

- Let's edit a file using the "vi" editor (if installed) or any other editor of your choice

  % vi README.md
  OR
- make a change (any change you want) in the file like:
  % echo "# learnGit - my first change" >> README.md

git status

- "git status" provides you with the state of the current git commands
- Enter the following command:

  % git status

- git then gives you the following output:

```
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
modified:   README.md

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

- The output tells you that the file, README.md, has been modified. It also tells you that you can use “git add README.md” to start tracking the file.

git add

- The “git add” command prepares your file to be tracked.
- Enter the following git commands:

```
% git add README.md
% git status
```

- You should see output similar to:

```
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
modified:   README.md
```

- This tells you that your changes need to be “committed”

git commit

- Now that you have added your file, you need to “commit” the file
- Enter the following command

```
% git commit -m "added notes  to README.md"
```

- You will see output similar to:

```
[main b7150cf] added notes to README.md
 1 file changed, 3 insertions(+)
```

- Next, let’s get the status again. Enter the following command:

```
% git status
```

- You will see output similar to:

```
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

- This is telling you that you need to use “git push” to publish your changes to README.md

git push

- git push takes your changes to README.md and pushes them up to the remote repository
- Enter the following command:

```
% git push
```

- You will see output similar to the following, which tells you that the changes to the file have been published to your repository:

```
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 411 bytes | 411.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/amitrathod101/introGit.git
   818263e..b7150cf  main -> main
```

- Go to your github.com account and look at the changes you made to the Markdown.

- You should see the line(s) that you had changed on your laptop are now pushed to the repository

git restore

- git restore can be used to recover a file if you accidentally delete a file in your local directory
- First, delete the README.md file from your laptop

```
% rm README.md
```

- Now, let’s use git to recover the file using the following command:

```
% git restore README.md
```

- List the files in the directory to show that README.md has been restored

```
% ls
```

- If you want to restore a previous version of a file from the master branch, you can use the following command

  % git restore README.md

git log

- The git log command gives you detailed log information on the commands that you have used
- Enter the following command:

  % git log

- You should see output similar to:

```
commit 8247ce3acc4618df5ff80de76783f0a8370bf5d8 (HEAD -> main, origin/main)
Author: Amit Rathod <amitrathod101@gmail.com>
Date:   Thu Mar 11 12:47:45 2021 +1300

    verifying git user commit

commit 3caf769693afa3a01f0d37a994e4677f632fc9dc
Author: Amit Rathod <arathod@vmware.com>
Date:   Thu Mar 11 12:32:11 2021 +1300

    removed unnecessary comment

commit 24c97414380938a7063ca0487768aae1be262533
Author: Amit Rathod <arathod@vmware.com>
Date:   Thu Mar 11 12:22:07 2021 +1300

    first take on kubebuilder

commit c7b1b883a26d5a4eab1e3a72b06e8939a384c369
Author: Amit Rathod <arathod@vmware.com>
Date:   Thu Mar 11 11:54:25 2021 +1300

    guestbook crd with kubebuilder
(END)
```

- You can also try “git log --decorate --graph --all” OR “git log --oneline --decorate --graph --all” OR “git log –oneline --pretty” → Whatever gives you a kick!

git help

- The git help command give you help on the various git commands
- Try entering the following commands

  % git help git

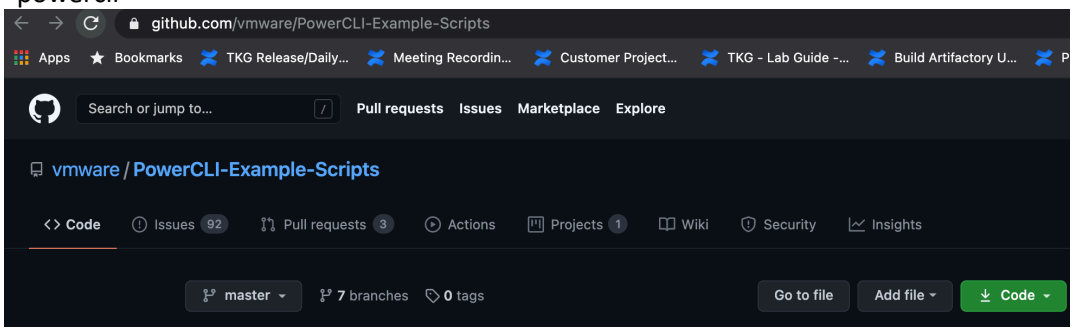
  % git help -a

  % git commit --help | more

Advanced Topics

git fork

- The git fork command makes a copy of another users repository and places it in your repository. The fork is a completely separate copy (i.e. you do not get any changes made in the original repository)
- Go to github.com and search for anything you like, in this case, I am simply searching for “powercli”



- Search for the "powercli" repository
- In the upper right hand corner select "Fork"
- Look at your repositories to view the new fork
- You can now do a “git clone” from your forked repository onto your laptop

git pull

- When two or more developers are working in the same code base, they are adding/deleting/modifying the same or different files. You will want to pull down these changes during the day to make sure everyone on the team is using the same codebase for development.
- In this example, Developer 1 adds a new file to the introGit repository. Let’s simulate adding a new file into the repository:
    - Use the "Add File"->Create new file in the github web screen
    - Name the file “test-pull”
        - Add a comment to the file
        - Save the file (at the bottom of the page)
    - Look at repository in github and you should see that the file “test-pull” has been created:

- Now, you as Developer 2(it’s yourself actually, just imagine another person), want to “pull” the latest changes from the repository into your local directory
- Do the “git pull” command below:

  % git pull

- If you do an “ls -al”, you should have pulled down the file that dev 1(yourself again) created.
- Best practices for using git pull
    - Pull every morning
    - On very heavily used repositories, pull multiple times during the day

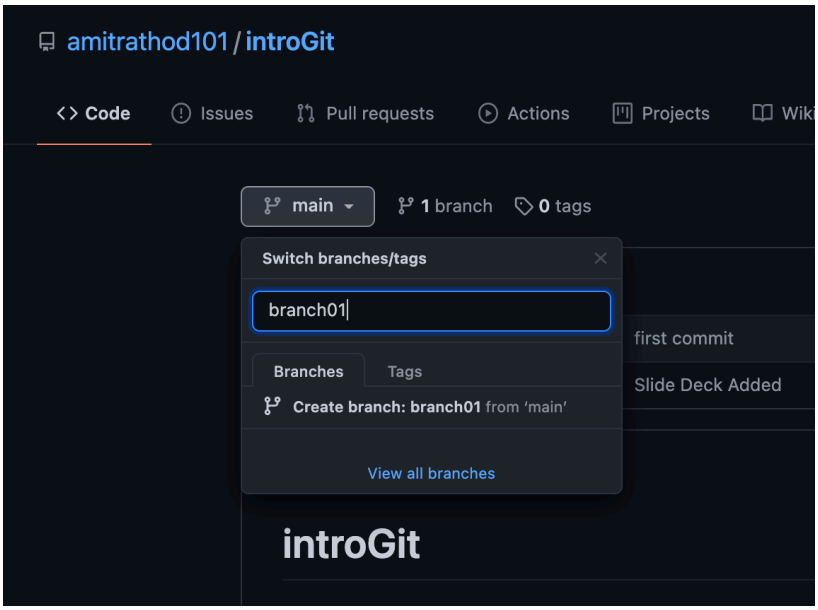
Branching

- When you create a branch, you create an identical copy of the project at that point in time
- First, let's explore the concept of remote branching and then local branching

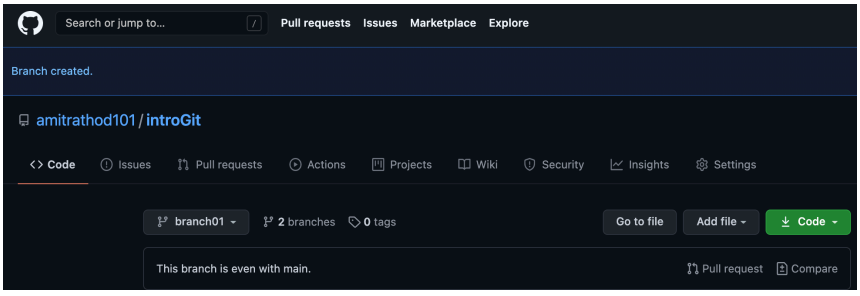
Remote Branching

- Remote branching occurs in the github.com repository (hence, being "remote" from your local laptop)
- You would use this for a project team...all team members work in this branch

  - Navigate to your repository in the github.com web page
  - Select the dropdown under the main branch



  - Create a branch called "branch01".
  - Now the number of branches changed to (2) and the default branch is "version-1"
    - The branches you have are "main" and "branch01"
    - You are automatically switched to the new branch, branch01
    - Any changes you make to the files in the repository will be applied to this branch



- Go to your local repository on your laptop and get any changes using git fetch:
- The fetch command will just get the metadata, you could also do a git pull too!
  % git fetch

- This updates the remote branch
- Take a look at your available branches. The current branch is noted by an "*"

  % git branch -a

- The output will be similar to:

Local Branching

- This is a branch that only "you" can see. It only exists on your local machine
- Let's create a local branch:

  % git branch laptop-branch

- Let's view the current branches:
-

  % git branch

  - Note *main is the current context

- You are still on the main branch. Let's switch to the new local branch:

  % git switch laptop-branch

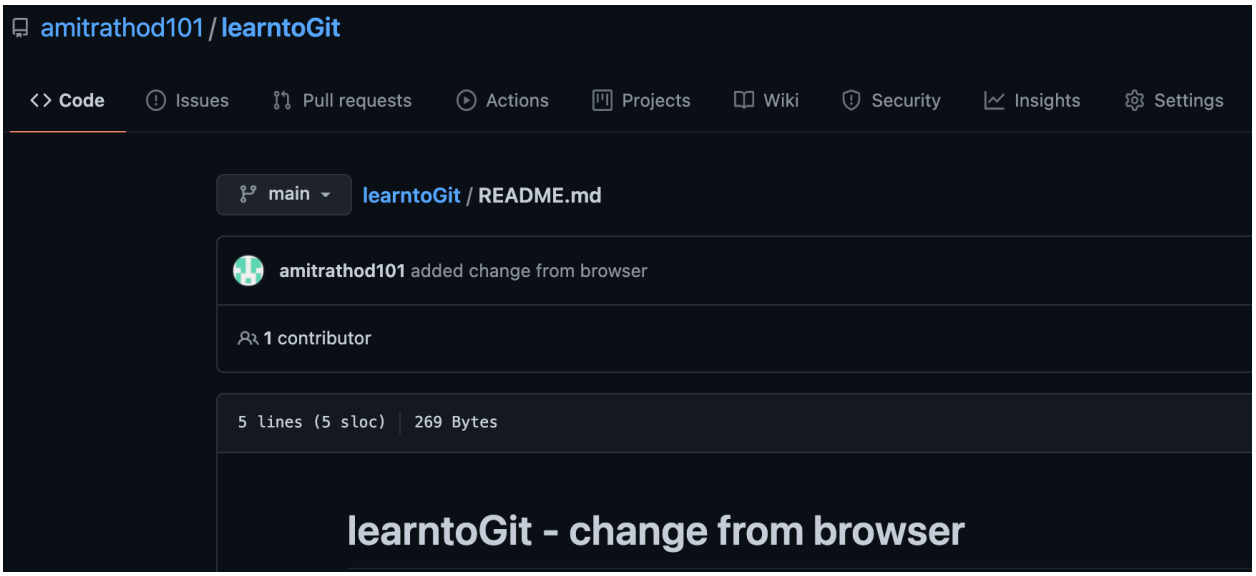
  - Note *laptop-branch is the current context
  - *Note: git checkout -b <name> will create the branch and switch contexts*
  - In our case we created the branch using the command git branch so that is why we choose git switch.

- You can compare branches using the following command:

  % git diff <local-branch> <origin/remote-branch>

Handling Conflicts

- **Conflict from browser**
  - Go to github.com/<your_github_id>/<introGit> , click on the README.md file and then Edit using the pencil. Add a line there, put in a commit message and then commit.
  - Now on the browser, your README.md file should show the change you made a step earlier.



  - Now on the local machine, we don't have those changes yet, until we do a git pull. We want to simulate a Conflict and hence we wont do git pull or git fetch yet.
  - Edit the README.md file now , make sure you add a line(or edit the line) to the same line which you added in the first step of this section. For the sake of brevity, I have just shown the first line keeping other lines as is.

```
~/go/src/github.com/amitrathod101/learntoGit  main ±  cat README.md
# learntoGit - change from laptop
```

  - Try to checkin the changes using the following commands:

```
% git add README.md
% git commit  -m "simulate a conflict"
% git push
```

- You should see output similar to:

```
~/go/src/github.com/amitrathod101/learntoGit  main ±  git add README.md
~/go/src/github.com/amitrathod101/learntoGit  main +  git commit  -m "simulate a conflict"
[main 707a878] simulate a conflict
 1 file changed, 1 insertion(+), 1 deletion(-)
~/go/src/github.com/amitrathod101/learntoGit  main  git push -u origin main
To github.com:amitrathod101/learntoGit.git
 ! [rejected]        main -> main (fetch first)
error: failed to push some refs to 'github.com:amitrathod101/learntoGit.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.
```

- Note that the output is giving you a "hint" to do a git pull…Let's try that:

  % git pull

```
✘ ~/go/src/github.com/amitrathod101/learntoGit  main  git pull
hint: Pulling without specifying how to reconcile divergent branches is
hint: discouraged. You can squelch this message by running one of the following
hint: commands sometime before your next pull:
hint:
hint:   git config pull.rebase false  # merge (the default strategy)
hint:   git config pull.rebase true   # rebase
hint:   git config pull.ff only       # fast-forward only
hint:
hint: You can replace "git config" with "git config --global" to set a default
hint: preference for all repositories. You can also pass --rebase, --no-rebase,
hint: or --ff-only on the command line to override the configured default per
hint: invocation.
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 700 bytes | 175.00 KiB/s, done.
From github.com:amitrathod101/learntoGit
   00c15e7..bd29557  main       -> origin/main
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
Automatic merge failed; fix conflicts and then commit the result.
```

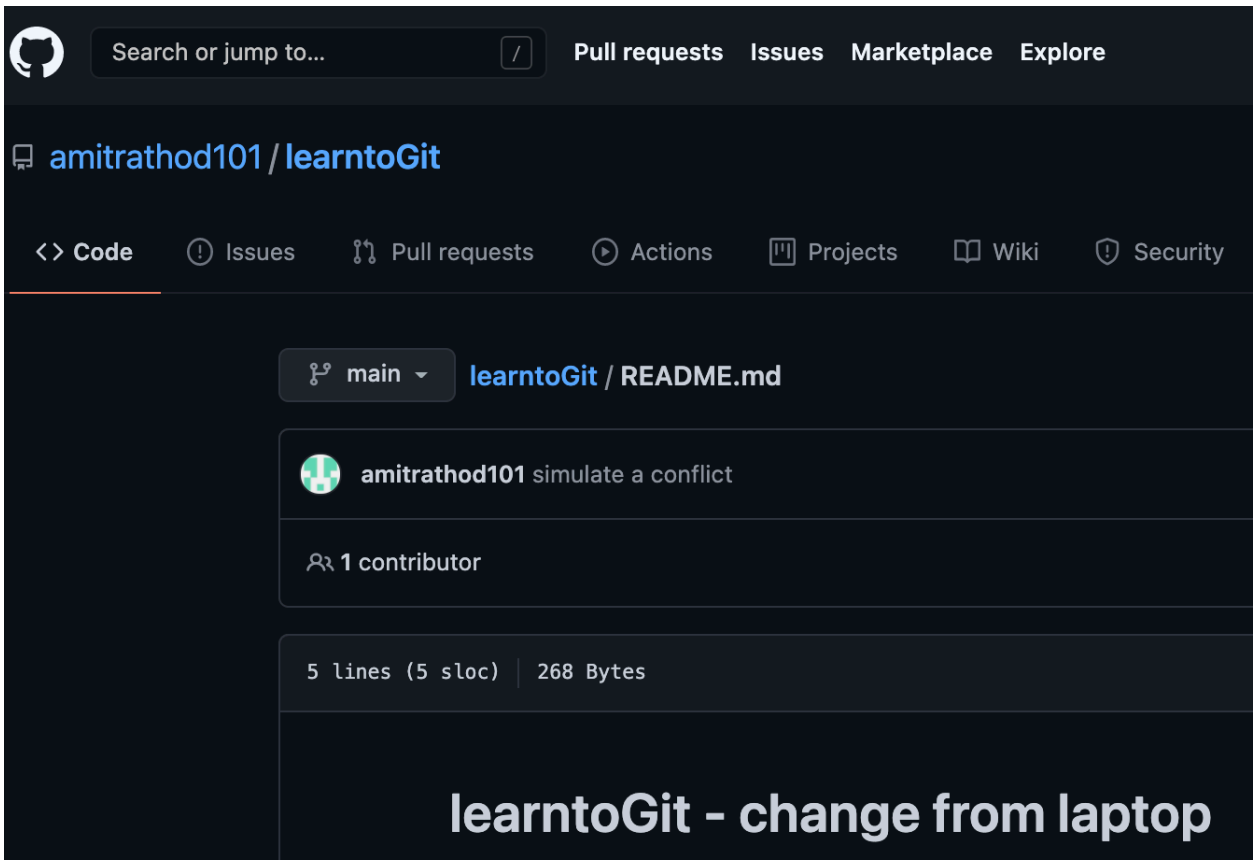
- The next part is not as obvious. Edit the file using the editor of your choice and remove the parts which you think should NOT be a part of and save the file. At the end the file should look like how you would want the file to look like.

```
 ✘  ~/go/src/github.com/amitrathod101/learntoGit  main ±+ >M<  cat README.md
<<<<<<< HEAD
# learntoGit - change from laptop
=======
# learntoGit - change from browser
>>>>>>> bd295579a34953a75b67cf350f4b74f275086b47
# learntoGit - adding 1st change - hotfix - adding a change for main branch merge conflict
# learntoGit - adding 1st change - creating merge conflict in testing
# learntoGit - adding more changes
# learntoGit - my first branch change
```

- Then follow the standard add/commit/push procedure to check in the file. Make sure to add a descriptive comment to tell others which lines you changed and why

```
 ~/go/src/github.com/amitrathod101/learntoGit  main ±+ >M<  vi README.md
 ~/go/src/github.com/amitrathod101/learntoGit  main ±+ >M<  cat README.md
# learntoGit - change from laptop
# learntoGit - adding 1st change - hotfix - adding a change for main branch merge conflict
# learntoGit - adding 1st change - creating merge conflict in testing
# learntoGit - adding more changes
# learntoGit - my first branch change
 ~/go/src/github.com/amitrathod101/learntoGit  main ±+ >M<  git add README.md
 ~/go/src/github.com/amitrathod101/learntoGit  main >M<  cat README.md
# learntoGit - change from laptop
# learntoGit - adding 1st change - hotfix - adding a change for main branch merge conflict
# learntoGit - adding 1st change - creating merge conflict in testing
# learntoGit - adding more changes
# learntoGit - my first branch change
 ~/go/src/github.com/amitrathod101/learntoGit  main >M<  git commit  -m "lqptop change persists"
[main 8f79538] lqptop change persists
 ~/go/src/github.com/amitrathod101/learntoGit  main  git push -u origin main
Enumerating objects: 8, done.
Counting objects: 100% (8/8), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 422 bytes | 422.00 KiB/s, done.
Total 4 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 1 local object.
To github.com:amitrathod101/learntoGit.git
   bd29557..8f79538  main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
```

- Look at file on github



This ends the scenario of conflict arising from changes on the browser.

- **Conflicts from branch merging – for screenshots see ppt**
  - Let's simulate a scenario where a developer is working on a piece of code where he needs to do some testing and hence naturally he/she will create a new branch and call it "testing". While he/she is going about in the day, he/she is being told by his/her boss that an urgent hotfix is required by a customer and he/she is asked to leave whatever he/she was doing and start working on the hotfix immediately.
  - First creates a branch first using
    % git branch hotfix
  - Then moves from the main branch to the hotfix branch using
    % git checkout hotfix
  - Makes changes to the code, in our case README.md
  - Checks the status using
    % git status
  - Does git add README.md
  - Does the git commit -m "hotfix change"
  - Now the fix is ready in his/her "hotfix" branch. The next steps could be rounds of code review, E2E tests or submitting a pull request. Assuming all this went well. Maybe he is the only developer or the maintainer of that code.
  - Now he/she checks out to the main branch using
    % git checkout main
  - Now he/she does the merge using
    % git merge hotfix
  - Now he/she looks at the code to see if the changes from the hotfix made it to the main branch.
  - If all is well, now he can get back to his work which he/she started earlier in the day while working on the testing branch.

git ignore

- As we have already seen, git constantly looks at the files in your directory to determine what needs to be pushed up to the remote repository. If it sees a new file, it tells you it is "untracked". The problem is that not all files should be pushed (i.e. log files, executables, credentials files". The .gitignore file allows you to specify which files to ignore

- Let's go back to your local repository

  % cd introGit

- Create a file you do not want to check in and put in some random text

  % vi top-secret-file

- If you now do a "git status", git will show the file as "U" - untracked

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

- In that same directory create a .gitignore file

  % vi .gitignore
  top-secret-file

- Save the file
- Now do a “git status”. The output should no longer show “my-secret-file” as untracked.

This ends the scenario of branch conflicts.

End