GitHub

SIAM Knights Python Workshops
Utilizing GitHub

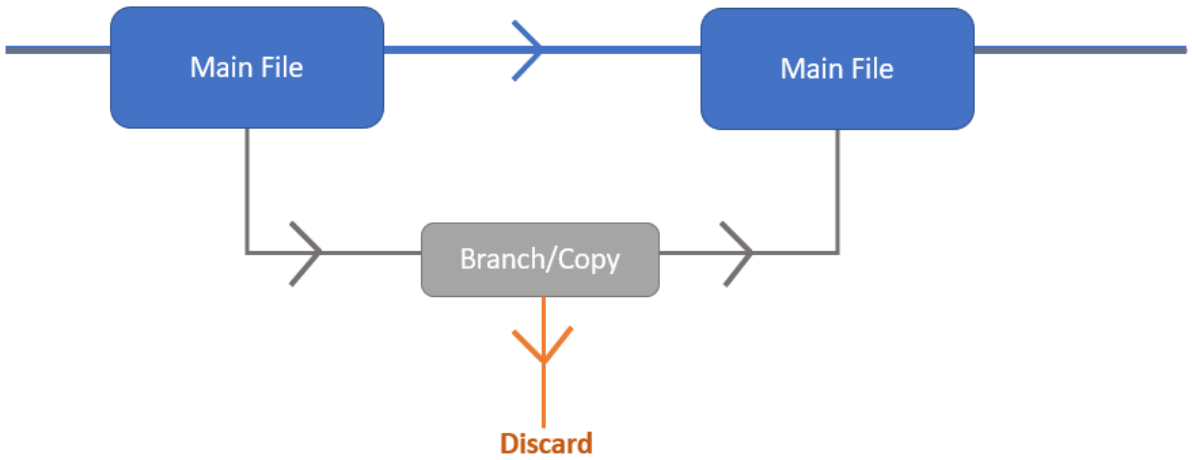
04/04/2021

Hanna Reed

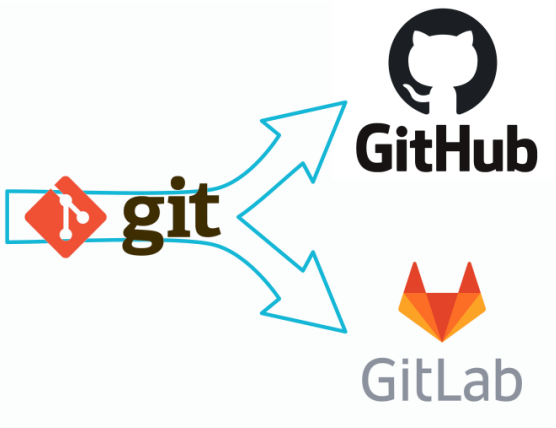
An introduction to using GitHub for personal use and collaboration.

Why would I use this?

- Provides **version history** with easy navigation
- Allows **code synchronization** between machines
- Create **branches** (a temporary copy that can either be merged or discarded)
- Easy collaboration on shared projects (reason for creation)



Some things to keep in mind



- Git vs GitHub
  - **Git** – software that preforms the actions
  - **GitHub** – a place to store your code and provides a web interface to preform git functions (add file, push, pull, merge, etc.)
    - Git is to GitHub as Python is to Spyder
- This talk will contain A LOT of information, some of which may be rather confusing
  - Ask questions, follow along, and practice afterwards
  - This presentation will be made available to you!
- **WARNING**: if you like to work ahead, do so at your own risk. I may not have time to answer your questions if something goes wrong while working ahead.

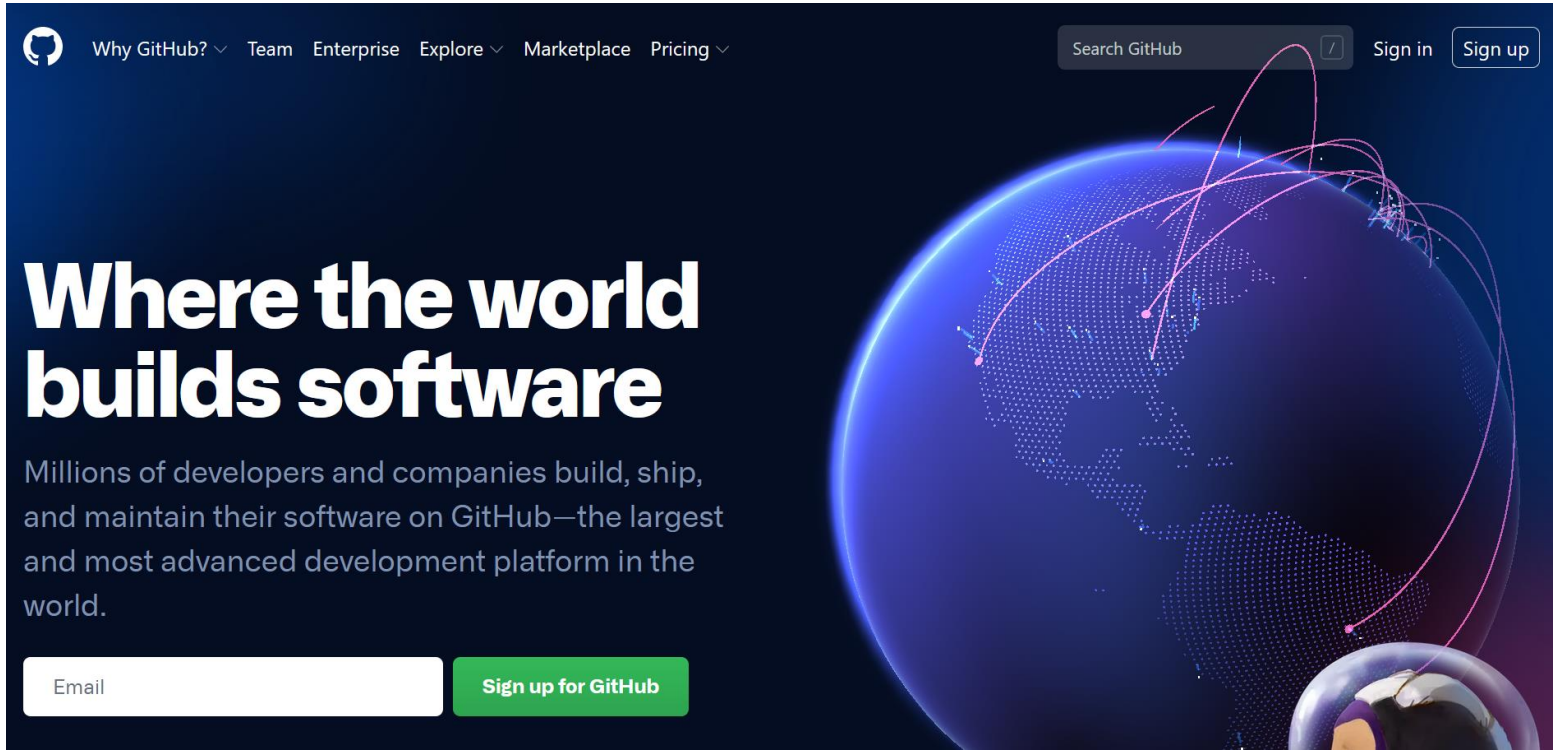
Order of Events:

1. Setting up *GitHub* account
2. Create your first repository
3. Intro to *Sublime Merge* (my choice, you can choose something else) - update as you work
4. Taking advantage of other work on GitHub – forking
5. Collaboration on GitHub – shared repositories, branches, pull requests, issues

Set-Up

- This will be a follow along workshop
  - Please make a GitHub account or login to your current account if you want to follow along!

    (recommended)

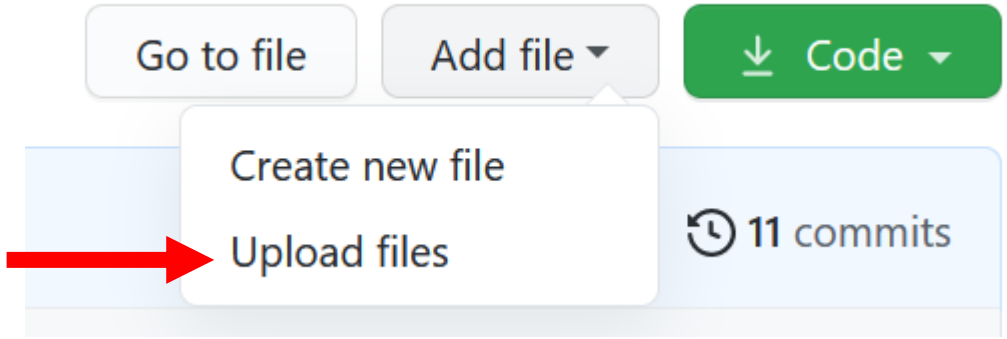


Creating your first repository!!

- A **repository** will host all the files for your project (or multiple projects) in one space.
- Main page >> repositories >> new
  - **README.md** : tells others what's going on and provides documentation space for you
  - **.gitignore** : tells git which files to ignore (the hidden, back-end ones) you can choose a template based on the coding language you will be using – choose python for our purposes.
  - **Licenses** : tells other what they can do with your repository https://docs.github.com/en/github/creating-cloning-and-archiving-repositories/licensing

Adding Files – the upload

- You can add files to your repo via upload from your machine
  - Try uploading a file from your computer to the repo you just created
  - If you want an example file, I posted one in the discord *note – you will need to download to your PC and then upload to GitHub*

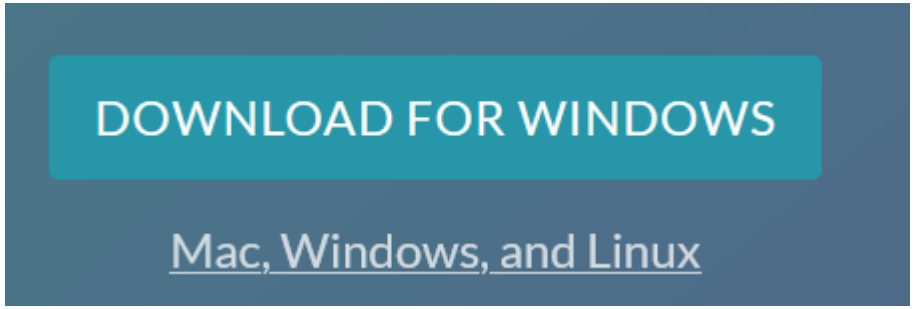


- Okay cool – we can make collections of files … but is this the extent??

Sublime Merge

- This is a **Git Client** which allows you to update your repo's as you edit them on your PC … meaning … no uploading, no copy/pasting, and your GitHub files stay up to date!

- Please download: https://www.sublimemerge.com/
  - There are other options; however, this is the one I know use and will be showing. After this workshop, you are free to browse others



How to use?

1. Open Sublime Merge on your computer

2. Clone the repo you have already created in GitHub **(watch me and then try yourself)** *note, sublime will ask you to sign-in so that it can connect to your GitHub account*
   - **main** and **origin/main**
     - Origin is the first instance of your repo (by default)

3. Let's make some changes! **(watch me and then try yourself)**
   - Which we will do right now

README.md

- Your readme will provide yourself and others some context to the files you have in each repository (and even each folder in the repository)

1. Open the README.md file on your computer
   - You can open in notebook, sublime text, or whatever text editor you like
2. Use the Markdown language to write your readme file
   - # and ## for headers
   - * and ** for italic and bold
   - Lots more – https://guides.github.com/features/mastering-markdown/
   - Let's push our changes to the GitHub using Sublime Merge! (watch me first)
3. Check out your edits on GitHub!

Version History

- Make changes on your local file
- Go to Sublime Merge:
  - Stage the modified file
  - Write a commit message
  - Commit
- Make changes on your local file
- Go to Sublime Merge:
  - Stage the modified file
  - Write a commit message
  - Commit

- Go to Sublime Merge:
  - Checkout previous version
    - Detached head!
  - Checkout current version

Push your modifications to GitHub

- Go to Sublime Merge:
  - Push
- Look at your modifications on GitHub

Pull modifications from GitHub

- Go to GitHub and open your file
- Make some modifications >> commit

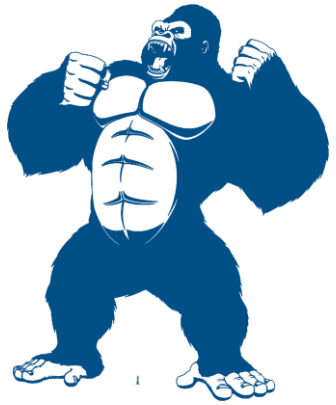
- Go to Sublime Merge:
  - Pull
- Check out your modifications on your local repository

What about conflicts?

- Make a local edit
- Go to Sublime Merge:
  - Stage >> comment >> commit (don't push)
- Go to GitHub:
  - Modify file
  - Commit

- Go to Sublime Merge:
  - Pull
  - CONFLICT!!!
  - Resolve ☺
  - Push

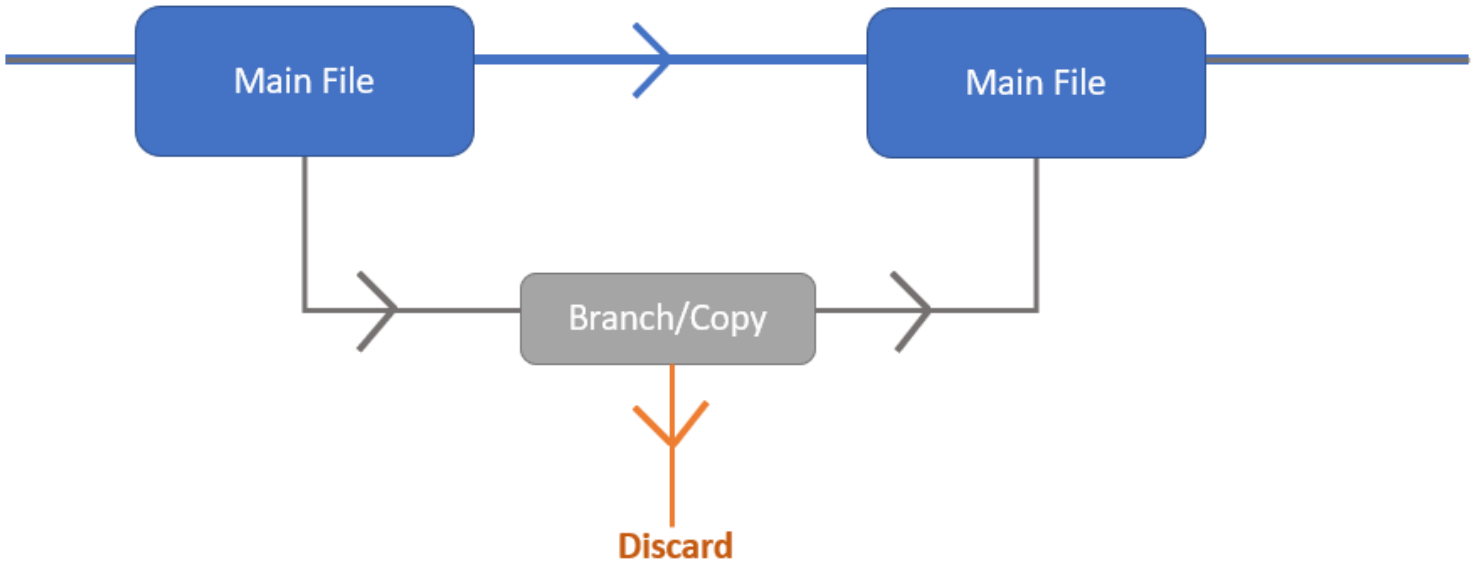
Local Version



GitHub Version

Creating Branches

- Experiment without ruining your already awesome work
- How can we make a branch?
  - Got to Sublime Merge:
    - Create a branch
    - Notice the pane on the left – the branch you are on is bolded!
- Managing conflicts will be the same as before!



Local Branches

- Make sure you are on the branch and make some edits

  stage >> commit >> **DON’T PUSH!**

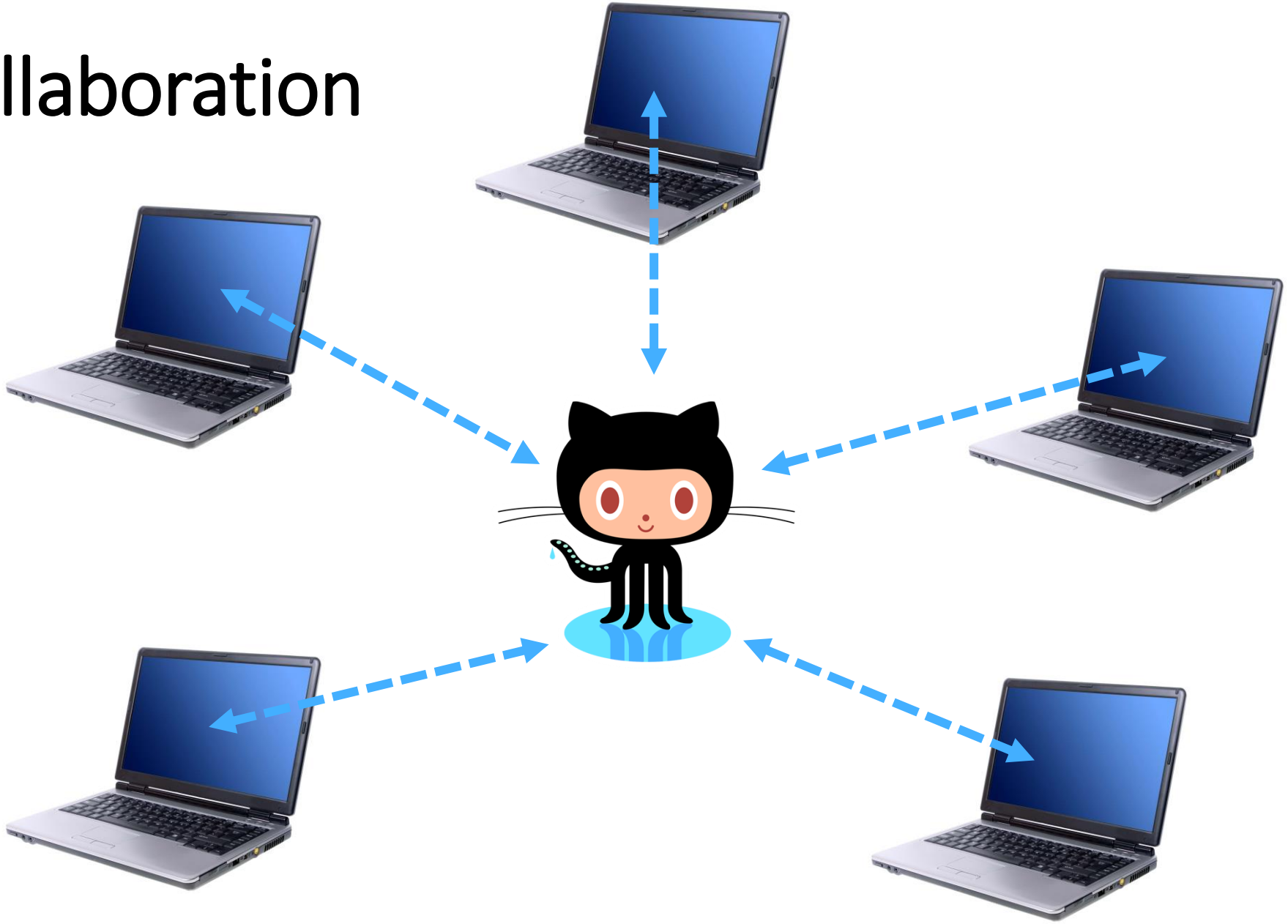
- To update main, we must MERGE the branch!
  1. Checkout main
  2. Merge branch to main
  3. Push main and delete branch

Pushed Branches

- What if we push the branch to GitHub?
  - Pushing to GitHub will push your branch to GitHub (it will not update main)
  - You will have a local branch and the branch in GitHub (just like main!)
  - Collaborators can see your work and may even work on the same branch!
  - While you work on the branch, you will push/pull as you did before

    Note: main will remain unchanged
  - Once you are ready, you will merge the branch to main and delete the branch (unless you want to keep it for more experiments)

Collaboration



Collaboration on the same repository

- Repository >> settings >> manage access >> invite collaborators
- Good practice:
  - Main is reserved for the “truth” (convention)
  - Make edits on branches – merge into main once complete
    - Deal with conflicts – same as before

Collaboration Between Repositories

- **Fork** a repository:
  - Go to GitHub:
    - Find the repository: Hannarea, SIAM-Knights-Python-Workshops
    - Fork it
  - Clone this repository onto your computer
- Collaborate:
  - Edit something on the repository (add a file, edit the readme, add a comment to the file, delete a file, whatever you want!)
  - Stage, commit, and push these changes to YOUR copy
  - Create a pull request from your repository
    - This will give me the option of updating my repository with your edits!

Review:

- ✓ Store projects in repositories
- ✓ Create branches for risk-free editing
- ✓ Version history
- ✓ Code synchronization
- ✓ Easy collaboration on coding projects
- ✓ Utilize others work via forking