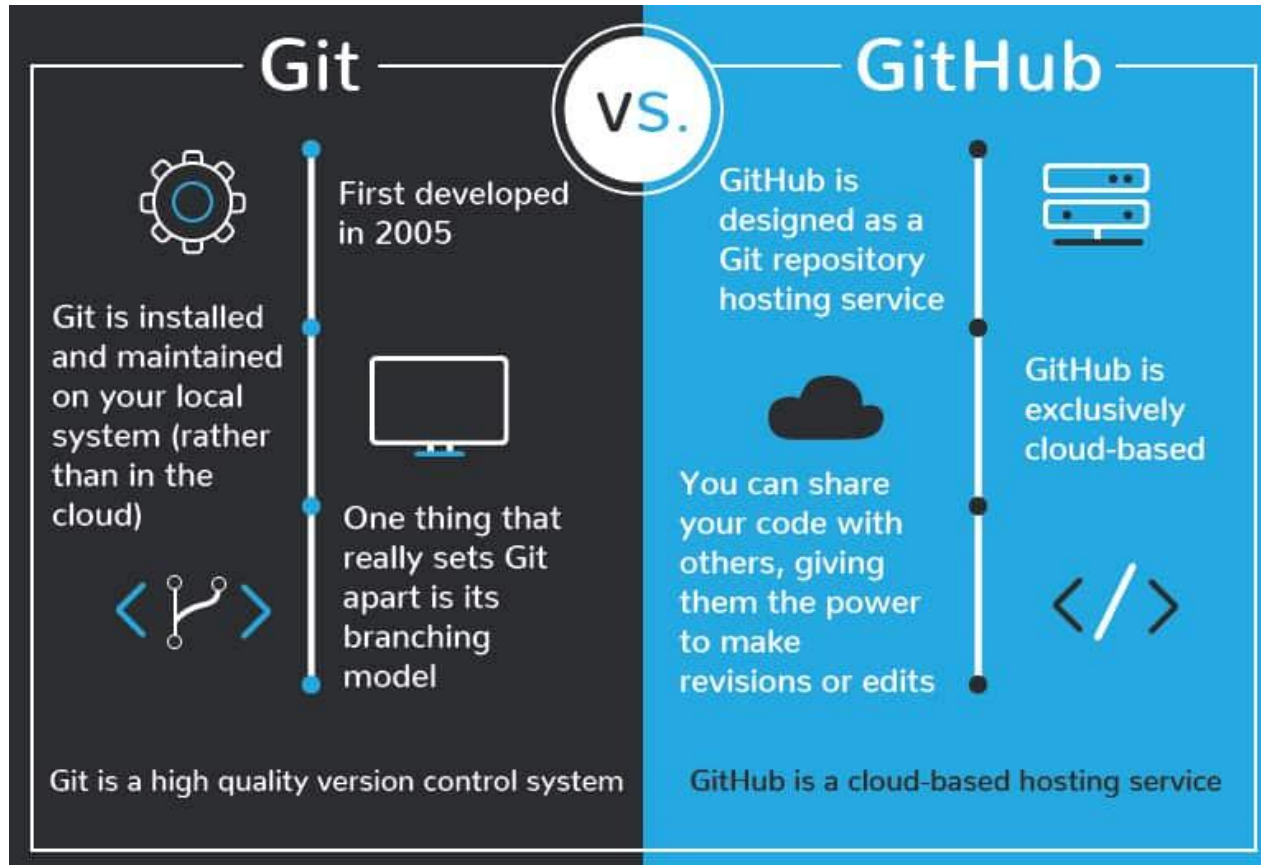


Intro to Git

12/2/2024
Yunti (Anna) Xu

- *What is code version control? Why should we care about it?*



- Where is our lab's [GitHub repository](#)?
- How is our GitHub repository organized?
 - Morimoto lab members have read access to all the repositories, this means you can clone and pull our repositories (public and private) -> should restrict only to PhDs + PostDoc?
 - Aedan and Anna are have admin access -> if you need to be added to your GitHub these two are who you should go to
 - Each project should have an associated team, members should have write access to the team's repositories (well, this is really up to the PhD/PostDoc leading that particular team)

The image displays three screenshots of the GitHub interface for the UCSDMorimotoLab organization, illustrating its structure and repository access.

Top Left Screenshot: Organization Overview

- Organization: UCSDMorimotoLab
- Repositories: 33
- Projects: 1
- Packages: 0
- Teams: 6** (highlighted)
- People: 14
- Insights
- Settings

Bottom Left Screenshot: Team Overview (CTR)

- Team: CTR
- Members: 4
- Repositories: 7** (highlighted)
- Projects: 0
- Organization roles
- Settings

Right Screenshot: Repositories with direct access (CTR team)

| Repository Name | Visibility | Last Updated |
|-------------------------------------|------------|-------------------------|
| UCSDMorimotoLab/CTR-ApproximateFTL | Private | Updated on Dec 15, 2023 |
| UCSDMorimotoLab/CTR1.0_hardwareDemo | Public | Updated on Aug 4 |
| UCSDMorimotoLab/CTR2.0-Firmware | Private | Updated on Oct 19 |
| UCSDMorimotoLab/CTR_kinematics_tc | Private | Updated on Sep 20, 2021 |

- How to create your own GitHub repository?

This screenshot shows the GitHub profile of a user named YuntiXu. The profile includes a circular profile picture of a woman, the name 'Anna (Yunti) Xu', and the GitHub handle 'YuntiXu'. Below the name, it shows 'UCSD' and 'La Jolla, California'. The 'Repositories' tab is selected, showing a list of repositories: 'cr_stiffness_control' (Private, MATLAB, updated last week), 'CTR_gui' (Private, MATLAB, updated last week), and 'IROS2019-CTR-Manipulability' (Public, Forked from RVMLab/IROS2019-CTR-Manipulability, 'The code for modelling of concentric tube robot and'). A green 'New' button is highlighted with a white box and an arrow pointing towards the 'Create a new repository' page.

This screenshot shows the 'Create a new repository' page on GitHub. The page has a dark theme. At the top, it says 'Create a new repository' and 'A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)'. Below this, it says 'Required fields are marked with an asterisk (*)'. The 'Owner' is 'YuntiXu' and the 'Repository name' is 'Intro to Git'. A green checkmark indicates 'Your new repository will be created as Intro-to-Git. The repository name can only contain ASCII letters, digits, and the characters -, ., and _'. Below this, it says 'Great repository names are short and memorable. Need inspiration? How about shiny-guacamole?'. The 'Description (optional)' field contains 'contains some introductory material to get your started on git'. The 'Initialize this repository with:' section has 'Add a README file' checked. Below this, it says 'Add .gitignore' and 'Choose a license'. At the bottom, there is a green 'Create repository' button highlighted with a white box and an arrow pointing towards the repository page.

This screenshot shows the 'Intro-to-Git' repository page on GitHub. The page has a dark theme. At the top, it says 'YuntiXu / Intro-to-Git'. Below this, it says 'Intro-to-Git' and 'Public'. The 'main' branch is selected. The 'Code' button is highlighted with a green box. Below this, it shows the repository's contents: 'YuntiXu Initial commit' (094c309 - now) and 'README.md Initial commit' (now). The 'README' file is open, showing the title 'Intro-to-Git' and the description 'contains some introductory material to get your started on git'. The right sidebar shows 'About' (contains some introductory material to get your started on git), 'Releases' (No releases published, [Create a new release](#)), and 'Packages' (No packages published, [Publish your first package](#)).

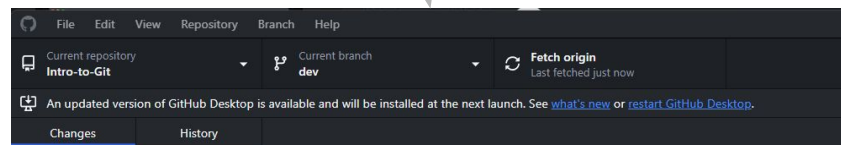
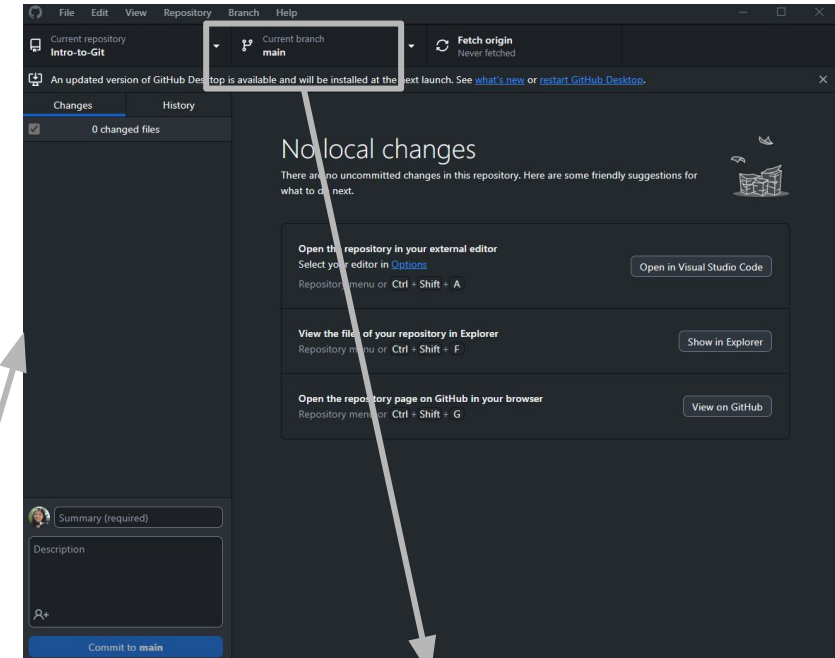
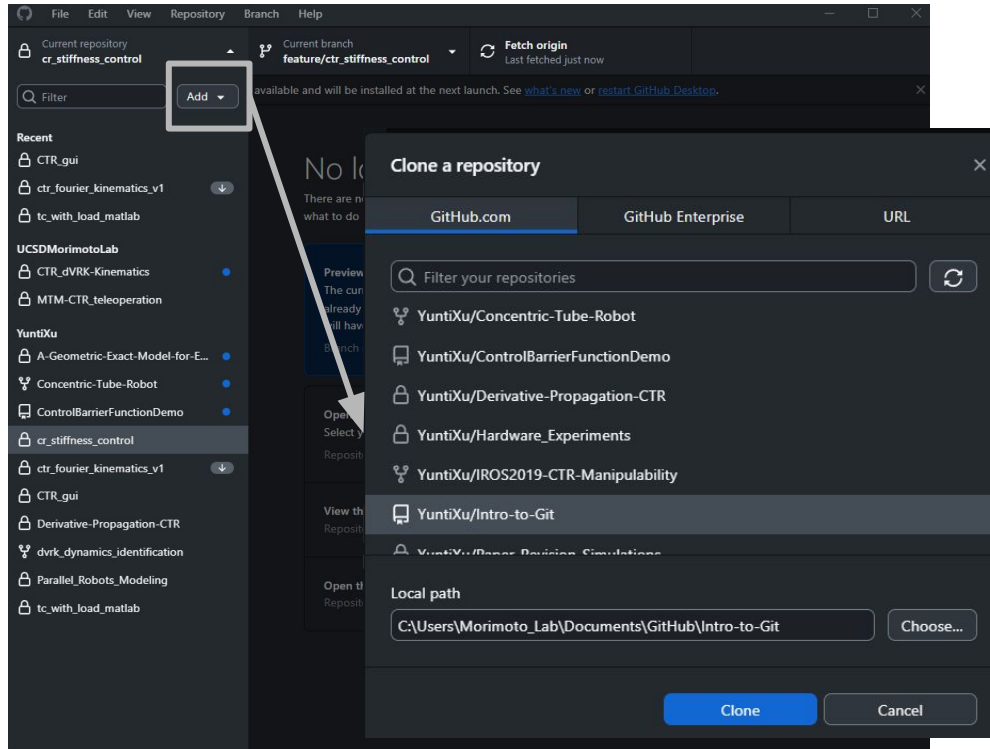
- *How to organize your GitHub repository?*
 - Ultimately it is up to you, it depends on who else might be collaborating with you on the repository. Typically, each repo has at least two branches: **main** and **dev**.
 - The **main** branch contains the “stable” version of your code, this could be the final version of the code used for your paper for instance
 - The **dev** branch is the branch you commit to (ie. update/backup) day-to-day, you will be changing the contents in this branch very frequently!

The image illustrates the process of creating a new branch in a GitHub repository through three sequential screenshots:

- Switch branches/tags:** The first screenshot shows the GitHub repository page for 'YuntXu / Intro-to-Git'. A dropdown menu is open, showing the 'main' branch as the current selection. The 'default' branch is also visible.
- New branch:** The second screenshot shows the 'Branches' section of the repository. A green 'New branch' button is highlighted in the top right corner.
- Create a branch:** The third screenshot shows the 'Create a branch' dialog box. The 'New branch name' field is set to 'dev', and the 'Source' is set to 'main'. The 'Create new branch' button is highlighted in green.

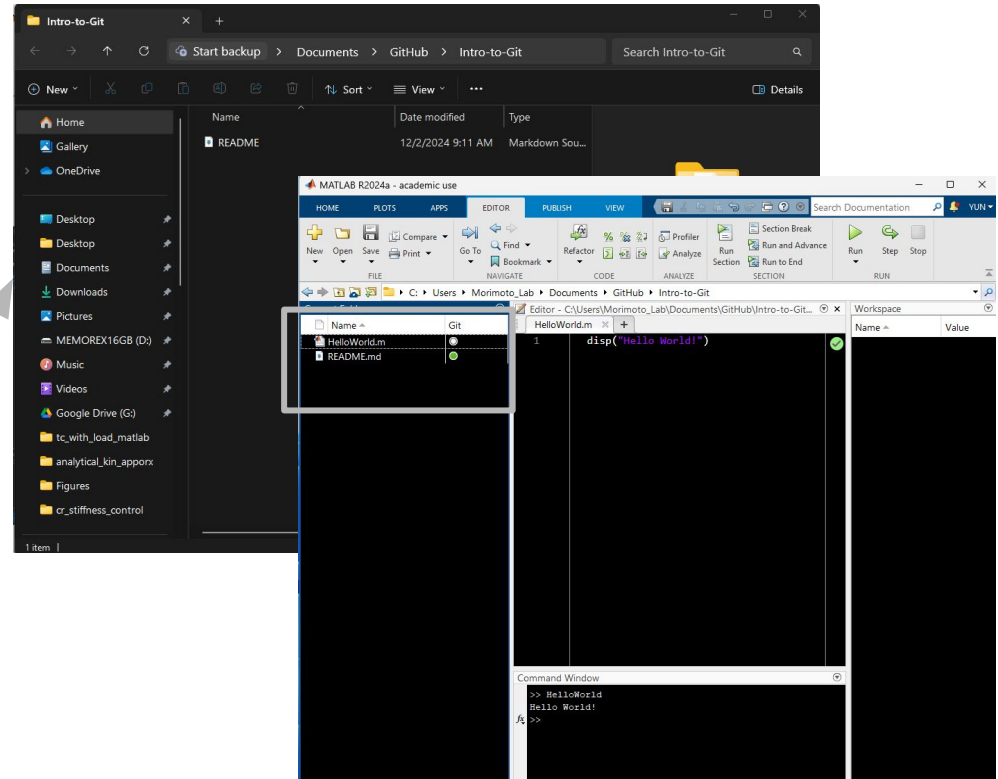
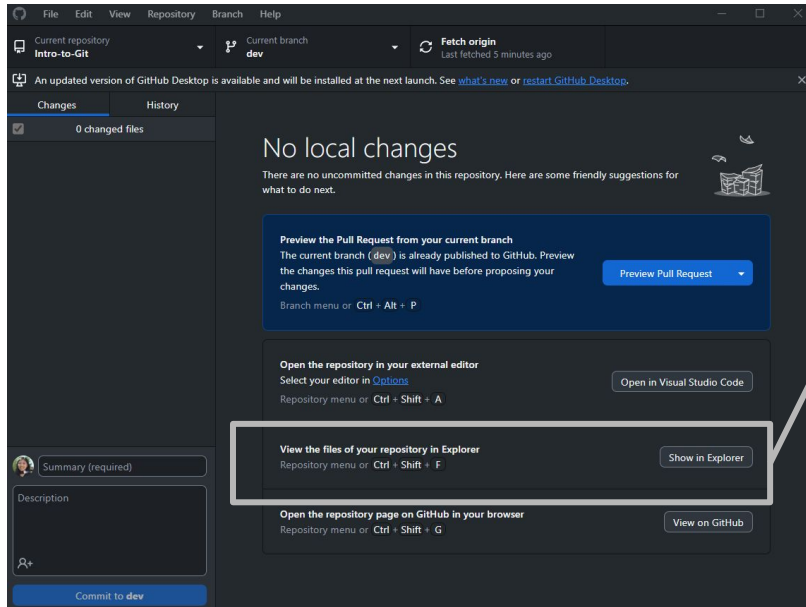
- *How to upload your code to your GitHub repository? (Windows & Mac)*

- The easiest way is to download [GitHub desktop](#), this way you interact with a GUI and don't have to deal with the inner workings of git.
- Step #1: find and clone your repository from GitHub



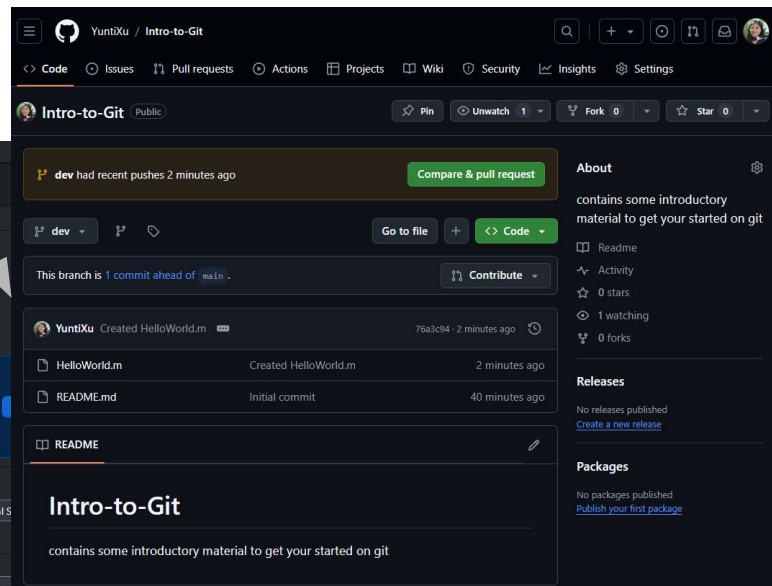
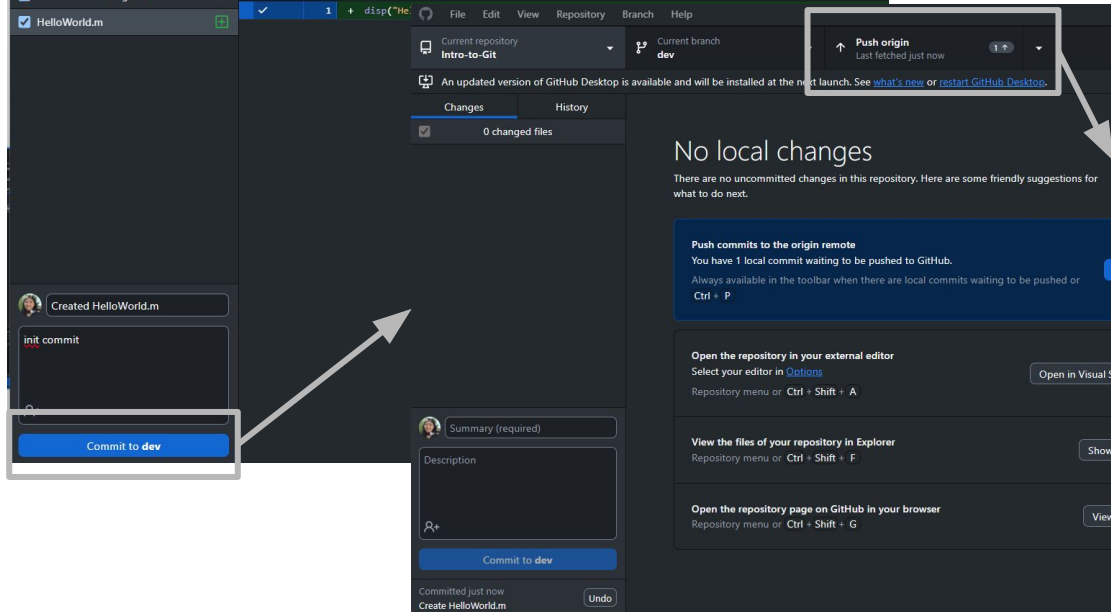
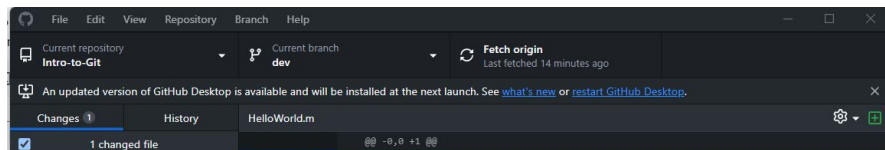
- *How to upload your code to your GitHub repository? (Windows & Mac)*

- The easiest way is to download [GitHub desktop](#), this way you interact with a GUI and don't have to deal with the inner workings of git.
- Step #1: find and clone your repository from GitHub
- Step #2: edit your code in your local folder (a clone of the remote GitHub repository)



● *How to upload your code to your GitHub repository? (Windows & Mac)*

- The easiest way is to download [GitHub desktop](#), this way you interact with a GUI and don't have to deal with the inner workings of git.
- Step #1: find and clone your repository from GitHub
- Step #2: edit your code in your local folder (a clone of the remote GitHub repository)
- Step #3: commit (ie. update) the repository and then push (ie. sync) with the GitHub website

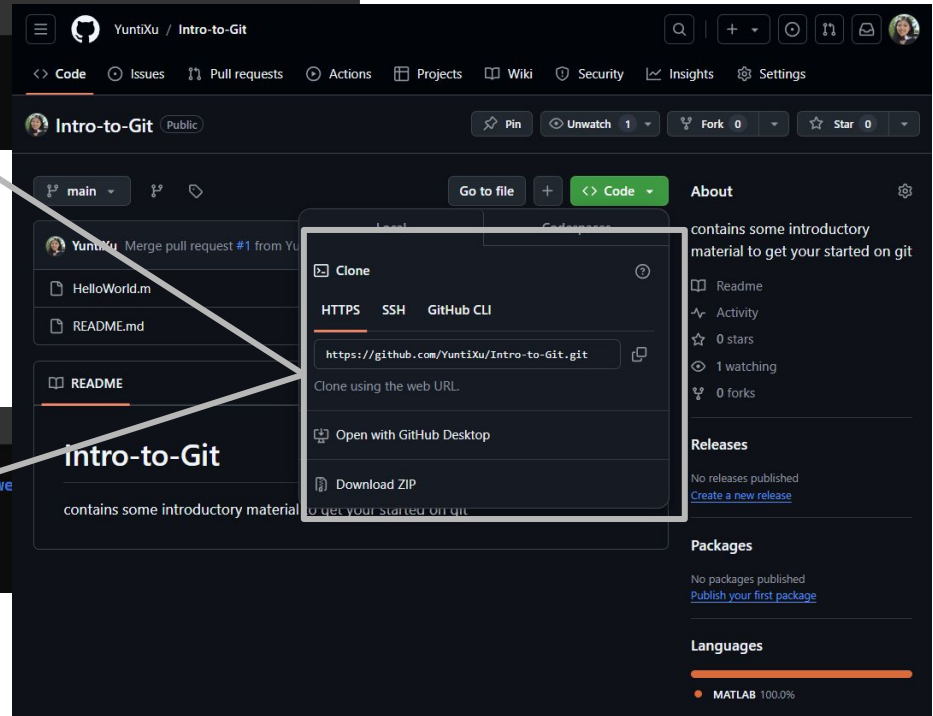


- *How to upload your code to your GitHub repository? (Linux)*

- There are two ways depending on how hardcore you are as a person (well, using Linux is already making a statement)
- The easier approach would be to use an [extension](#) in VSCode, if you are using it as your code editor anyways (those of you who are using Python should already be familiar with this IDE)
 - Step #1: clone your git repository via command line

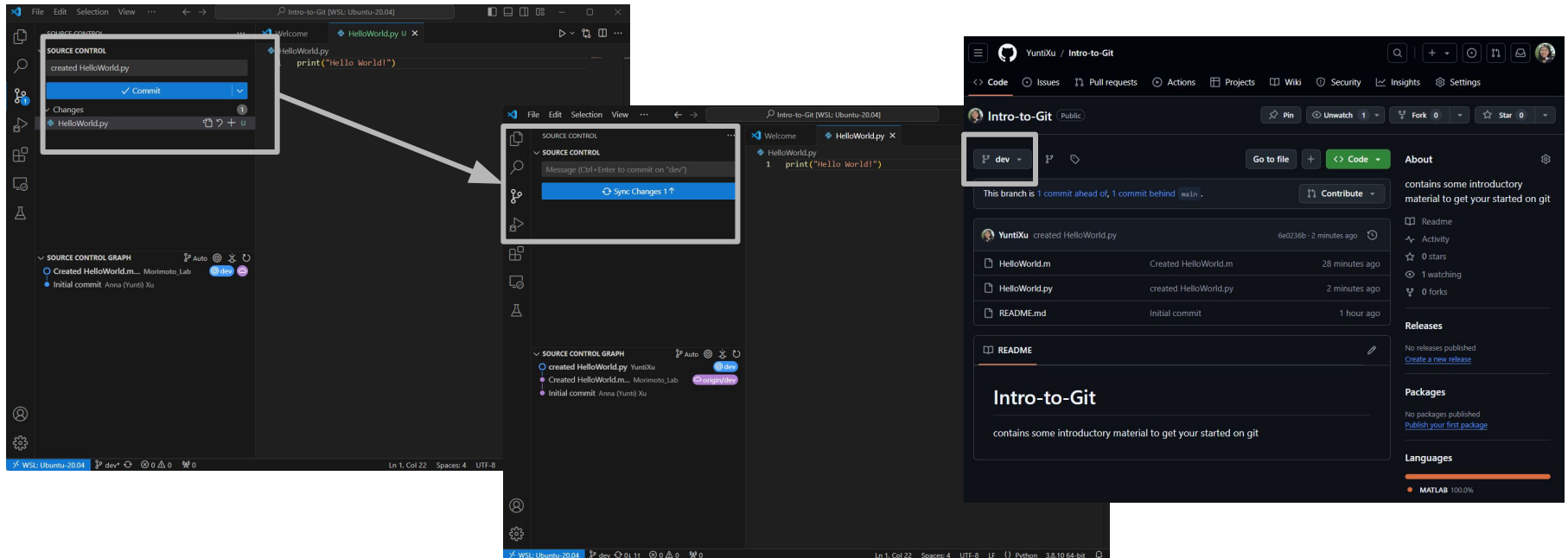
```
ytxu@DESKTOP-4ITQVEH:~$ git clone https://github.com/YuntiXu/Intro-to-Git.git
```

```
ytxu@DESKTOP-4ITQVEH:~$ ls
IROS2019-CTR-Manipulability  Modeling-and-Control-of-Concentric-Tube-Continuum-Robots  we
Intro-to-Git                  ctr_fourier_kinematics_v1
ytxu@DESKTOP-4ITQVEH:~$ cd Intro-to-Git/
ytxu@DESKTOP-4ITQVEH:~/Intro-to-Git$ code .
```



- *How to upload your code to your GitHub repository? (Linux)*

- There are two ways depending on how hardcore you are as a person (well, using Linux is already making a statement)
- The easier approach would be to use an [extension](#) in VSCode, if you are using it as your code editor anyways (those of you who are using Python should already be familiar with this IDE)
 - Step #1: clone your git repository via command line
 - Step #2: edit your code in your local folder (a clone of the remote GitHub repository) + commit and push to origin directly in VSCode



- *How to upload your code to your GitHub repository? (Linux)*

- There are two ways depending on how hardcore you are as a person (well, using Linux is already making a statement)
- If you using MATLAB unfortunately, you could attempt the above...but I would suggest directly do your git commands from terminal
 - Remember, you are basically doing the same thing, except instead of clicking buttons on a GUI you are writing the commands in terminal...

```
ytxu@DESKTOP-4ITQVEH: ~/Intro-to-Git$ git status
On branch dev
Your branch is up to date with 'origin/dev'.

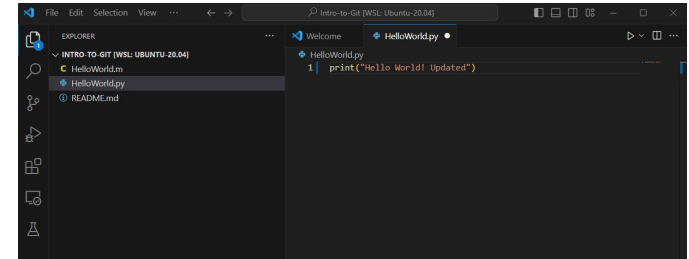
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   HelloWorld.py

no changes added to commit (use "git add" and/or "git commit -a")
ytxu@DESKTOP-4ITQVEH: ~/Intro-to-Git$ git commit -a
```

```
ytxu@DESKTOP-4ITQVEH: ~/Intro-to-Git$ git status
On branch dev
Your branch is up to date with 'origin/dev'.

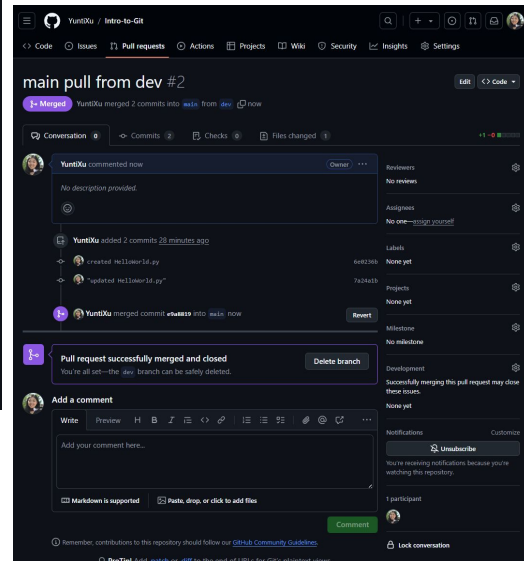
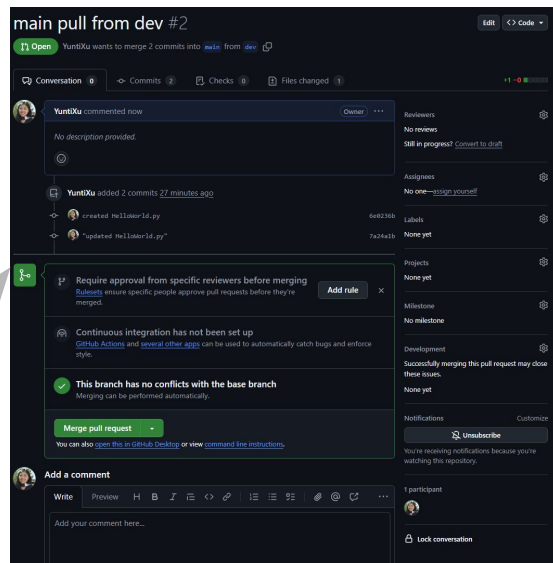
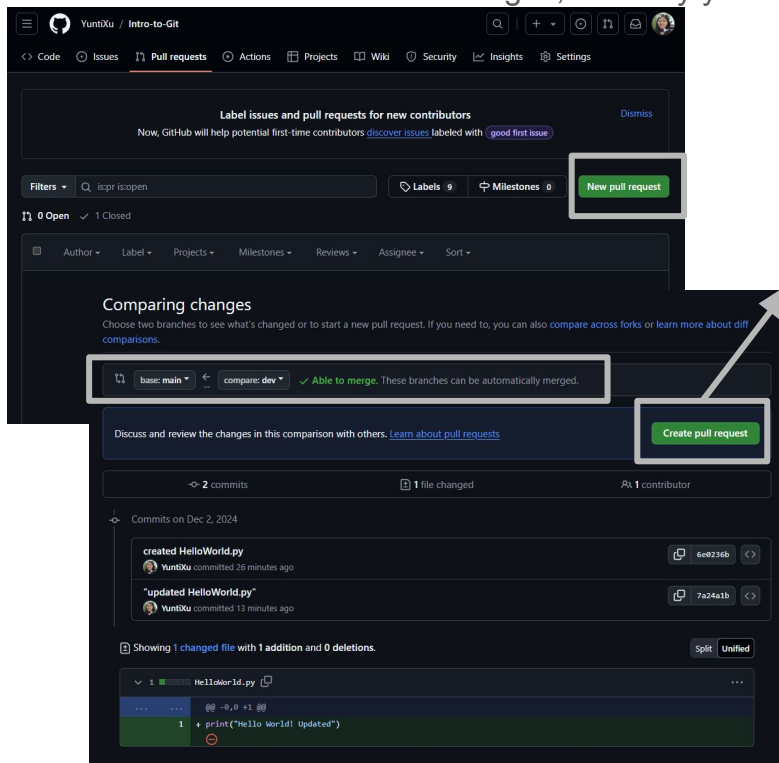
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   HelloWorld.py

no changes added to commit (use "git add" and/or "git commit -a")
ytxu@DESKTOP-4ITQVEH: ~/Intro-to-Git$ git commit -a
[dev 7a24a1b] "updated HelloWorld.py"
1 file changed, 1 insertion(+), 1 deletion(-)
ytxu@DESKTOP-4ITQVEH: ~/Intro-to-Git$
```



```
ytxu@DESKTOP-4ITQVEH: ~/Intro-to-Git$ git push
Username for 'https://github.com': Yuntixu
Password for 'https://Yuntixu@github.com':
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 12 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 340 bytes | 340.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/Yuntixu/Intro-to-Git.git
6e0236b..7a24a1b dev -> dev
ytxu@DESKTOP-4ITQVEH: ~/Intro-to-Git$
```

- *What happens when you are done with code development?*
 - So far, we have been working with the **dev** branch, when we are satisfied with our code, we should merge all the changes into the **main** branch
 - It is good practice to do especially in scenarios where you are developing your code in components, once you have tested a “subsystem”, have a stable version of it saved somewhere before making any more changes, that way you can always fall back on a working version!



Questions?