Remote Git

Since we started this course, we've only worked in our local repository. Still, Git is a great team working tool; it would be a shame to use it only on a local repository. We are going to see in this section what a remote Git is, and why anyone would want to use it.

Using a remote server is just having a computer holding a copy of your project, and its history. You don't have to push all your commits into it, you can just push the commits you want to share. Your coworkers then pull the commits that interest them and apply them to their own repositories. That does it! You work with a remote server to copy repositories and to push and pull changes. Let's see in detail how it all works.

To set up a remote repository, you will first need a server capable of running the Git software. Any computer worth its salt can run Git as it is a very small software. You also won't need a lot of firepower to run it properly. Even a very small computer like the Raspberry Pi is more than enough for Git.

Guess what? There is a tool that takes care of all those things for us! It is called GitHub! GitHub is the tool of choice when dealing with remote repositories; you can think of GitHub as a code hosting server for projects using Git. It works just like your own Git server, but with less headaches.

It was created in 2008 to host Git projects and is now a subsidiary of Microsoft, which has been investing a lot in Open Source Communities.

GitHub and Open Source

GitHub has always been a close ally of Open Source projects; in fact, GitHub is home to the largest Open Source community in the world. Since developers need a convenient place to build and share their projects, GitHub is an obvious choice. That way, all of the decisions and discussions concerning the projects can be consulted and joined by anyone; and that is the beauty of Open Source.

With GitHub, the best thing you can do to an Open Source project is now easier than ever: contributing. When you spot a project that you like, you can follow it, like on social media, and see its progress. If you want to work on a new feature or fix a "bug", you just have to make a clone of the project and work on it. That process is called "Forking," and it's the backbone of Open Source projects. When you've made all the changes to your copy of the project, you can submit a "Pull Request" (PR) to the maintainer of the project. That means that you are requesting that the changes that you made be pulled and merged into the project. Other contributors will then review your changes and may request some additional changes. Instead of communicating by email or instant messaging, all of this is done on GitHub. After all the parties are in agreement about the changes, the Pull Request is accepted, and your changes are now part of the project!

Of course, Open Source projects are more than code; they need docs, translators, community managers, maintainers, and so much more. You can contribute to projects by writing documentations and providing translations, or even reviewing the changes that other contributors made. Projects also need testers and people that can provide insights about the final products. They are projects that have millions of contributors, so community managers

are needed. They are responsible for the wellbeing of the community, and are expected to enforce the internal code of conduct of the community.

Some contributors are tasked with welcoming and tutoring beginners, which is difficult but very necessary for any project.

GitHub was chosen by millions of Open Source projects because the workflow from idea to release is so easy and accessible. The concept of forking a project to contribute to it is the main driving force of any Open Source project. If you like a project but don't like the direction it's going; you can fork it and start your own flavor of the project. You will then be the maintainer of the new project, and others can submit Pull Requests to you if they want to contribute. Thus, everyone is happy! As previously established, Open Source projects need documentation and tutorials for beginners. For small projects, a text file (called README by convention) is enough.

The "README" file should present the project and convey the problems it solves.

Remote Repositories

Do you remember the github account you created earlier? Click here to login with your username and password [GitHub: Where the world builds software · GitHub]

Creating a remote repository

You GitHub homepage is pretty empty but we're working on filling it with cool projects. At the right side of the page, you'll see some trending repositories or news story; but we won't go there yet

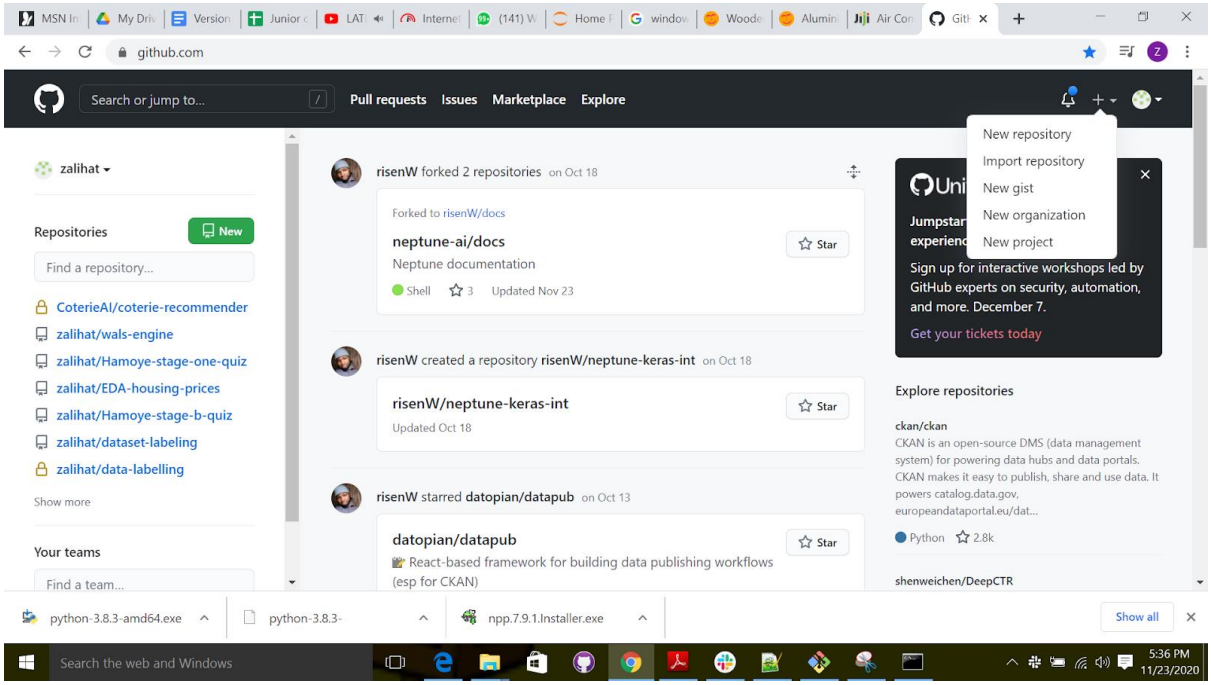

Figure 13: creating a remote repository
Click on new repository and you will see a page like the one below;

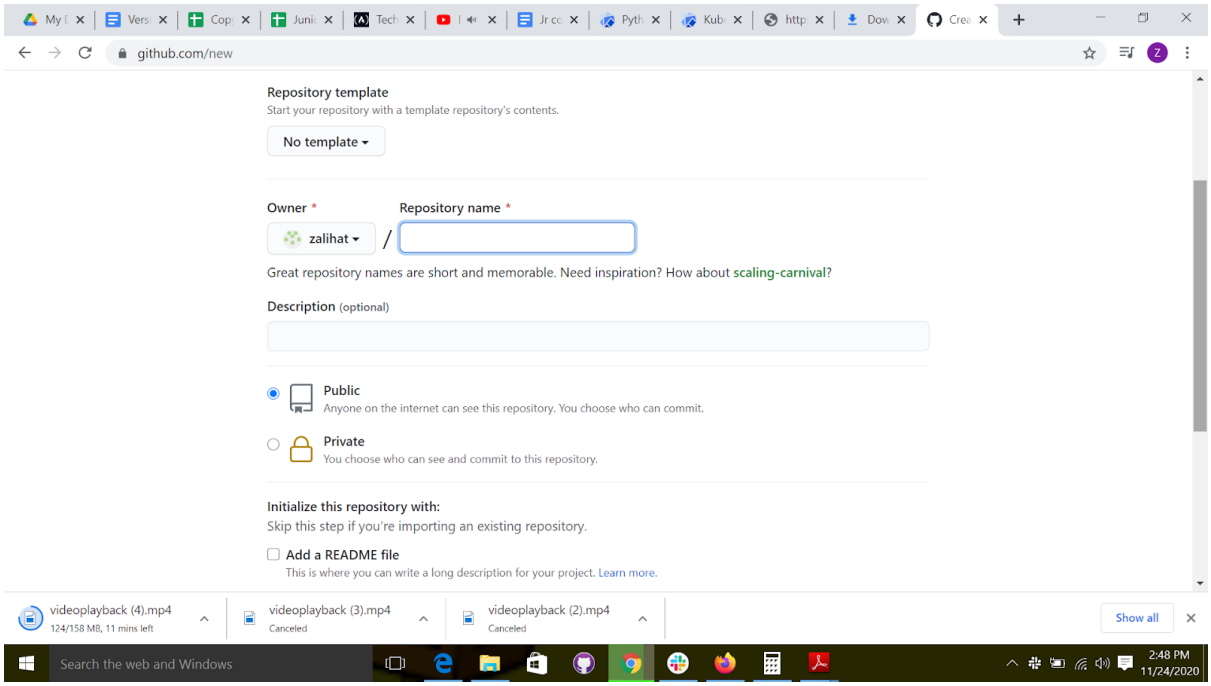

Figure 14: Create repository page
The repository for this example will be named “to-do”.

You can choose to make the repository private if you like, nobody but you will have access to it. A public repository doesn’t mean that anyone can edit it, it just means that anyone can read it and logged in users can propose changes to it. You will still be the maintainer of the project and the owner of the repository.

Then, you have the choice to initialize the repository with a README file. Ignore this for now because we are aiming to create a repository from scratch; and we will add

README, .gitignore, and license files later.

After all is done, click the Submit button to create your first GitHub repository! It’s that simple! You will then be redirected to your project page, which is a unique link to your repository. The link looks like this: https://github.com/your_username/your_repository; For example, the new repository I created is accessible through the following link: https://github.com/zalihat/to-do . Thus, you can’t create two repositories with the same name. Your project page should be similar to the one shown below.

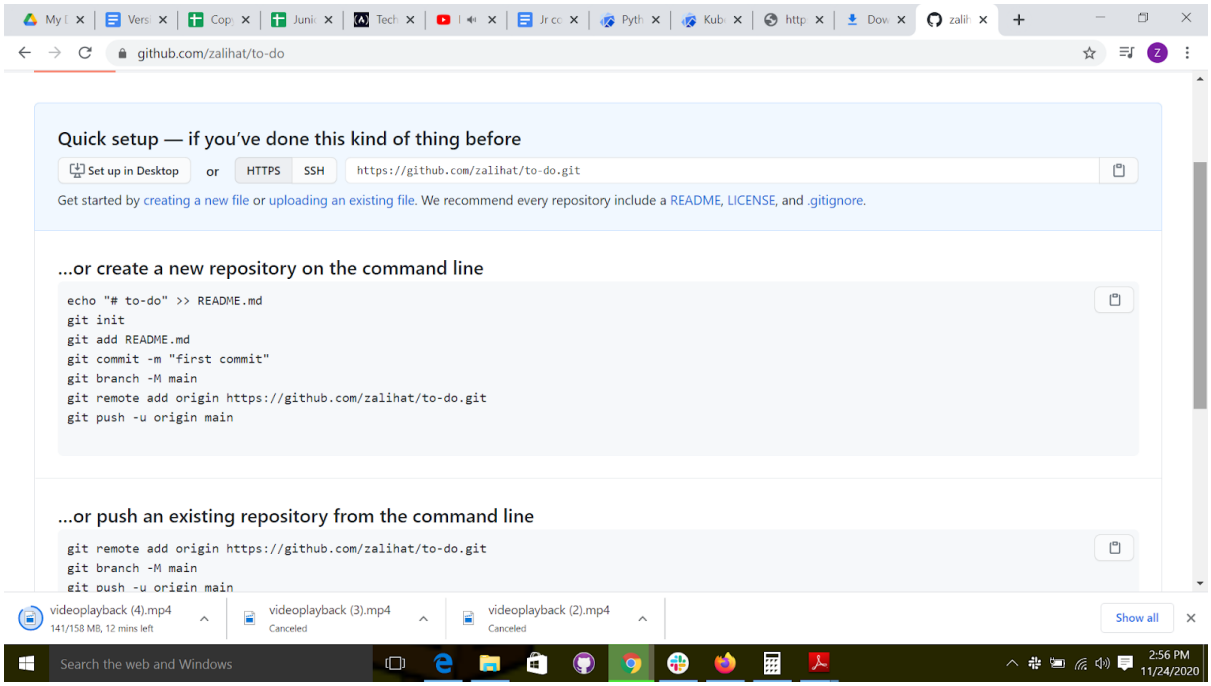

Figure 14 : new repo page

As you can see in the Figure above, there are some instructions on how to get started; whether you want to create a new repository or push an existing one. Since we are building our repository from scratch, we will go with the first option. The second option would have worked for us too because we already have a local repository, but we are going to ignore that from now on.

Thus, we created our first repository and are ready to push our project on it. Still, let's look into the magic box to see what exactly has just happened.

Remember we earlier talked about remote Git, and how we decided to use GitHub as a remote

repository store? This section is a logical extension of that section because we are going

to learn how remote repositories managed with GitHub works.

When we created our repository using the GitHub web site, we gave instructions to GitHub servers and asked them to initialize an empty repository. Also, if you can remember, initializing a repository is very simple: go to any directory and execute git init. That's exactly what happened here, except that this is not to your computer, but to a server hosted by GitHub.

It is therefore as if we executed the following commands on a faraway server which has git installed

1. Create a directory to do
2. Inside the folder "git bash" and run the "git init" command

It's the same commands that we will use to create our local repository. So now, there

is a remote repository in GitHub's servers that we will use to share our project.

Remote repositories are used so you don't have to use your own computer to share your project. In the case of GitHub, the remote repositories are accessible by anyone, but

only the owner can edit them. We will discuss teamwork in a later section.

The main takeaway is that a remote repository is where you can publish your project and make it available to everyone. Also, anyone can clone your repository such that your advancements can be followed to get the latest changes.

Publishing your local repository to a remote one is called “pushing,” and getting the latest commits from a remote repository to a local one is called “pulling.” Push and pull are maybe the most used commands you’ll use in Git.

Even so, how can I tell GitHub which remote repository I want to be linked with my local one? That’s where the unique link to your repository is needed. You’ll use the link to push your local changes or pull the commits you don’t already have.

In conclusion, GitHub created an empty remote repository which can only be modified by you, but can be seen by everyone. What we need to do now is create a local repository, and link it to the remote one.

Linking repositories

Now that we have our local repository, it’s time to link it to the remote! To list, add, or remove remotes, we will use the git remote command. For example, let’s link our current

remotes using this command:

git remote

You shouldn’t get any result because it’s a brand-new repository, and we haven’t linked any remote to it. Let’s add one now.

You will need the unique link to your repository to be able to link a local repository

to it; so, grab yours from the previous section. Mine is https://github.com/zalihat/to-do.git . Don’t forget the “.git“ at the end!

You will also need to create a name for your remote repository. That way, you can

have multiple remotes within a single project. It may be necessary in the case where the test and production remotes are different for each other. The default name is “origin” per

convention. Although you can choose any name, it is recommended to use origin as the

name of the remote where teammates share their work.

The command to add a link to a remote is simple. It’s

git remote add [name] [link]

Hence, to add a link to the newly created repository, you’ll have to execute this command:

git remote add origin https://github.com/zalihat/to-do.git

That’s it! You can check if the remote has been added by executing git remote or git

remote -v to get more information.

Pushing to remote repositories

We finally got our local and remote repositories linked. It’s time to push our project to

GitHub so we can share our work.

Pushing commits to a remote repository is very simple; but first, let's create some commits to push. In your working directory, create a file called "README.md and put in the description of your project in Markdown. For example, here is my README.md file:

TODO list

A simple app to manage your daily tasks

Features

* List of daily tasks

Now, let's add the newly created file to the staging area by using git add.

```
git add README.md
```

Now is the time to commit our project with git commit. As commit message, many developers choose "Initial commit" when it's the first. It's not a rule, and you can change it if you want to.

```
git commit
```

Since we've done these many times already, you should be comfortable with staging and committing by now.

Just like that, we have our first commit! Now, we can push those changes to the remote repository. The command to push changes to remote is simple; you just need the name of the remote repository and the branch to be pushed. Since we haven't created any branch yet (we'll learn about branches in a later section), our only branch is called "master." The git push command is

git push <remote_name> <branch_name>

So, in our case, the command will be git push origin master

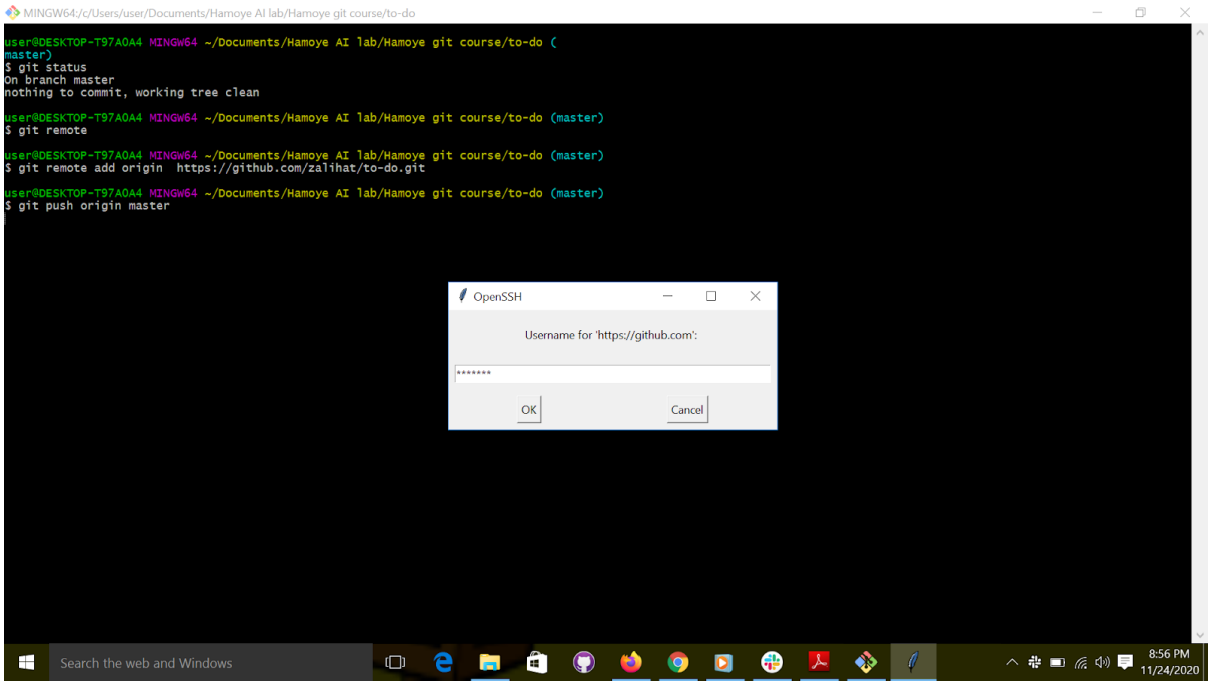

Figure 15: git push

You will be prompted to input your username and password for authentication. This is to ensure that you are the one making changes to the repository.