

Course Code: SL3001	Course: Software Construction and Development
Instructor(s):	Miss Nida Munawar, Miss Abeeha Sattar

Lab # 09

What is Git?

Git is a very popular version control system.

What is a Version Control System (VCS)?

A version control system records the changes made to code, over time, in a database. This database is referred to as the repository. The project history can be viewed here and we can also see who has made what changes, why the changes were made, and when the changes were made. If the project runs into any issues after the changes, it can be reverted back to an older, more stable version.

Without a version control system, a lot of manual work will be needed to maintain versions of the project. This would become even more difficult to manage if multiple people are working on the project on their own PCs.

The two main benefits of using any VCS are:

- Easier to track project history
- Easier to collaborate on the project

You can have two types of VCS:

- Distributed
- Centralized

Distributed Version Control Systems:

In a distributed VCS, all team members have a copy of their project on their machine. Therefore, each member can save changes to the projects locally.

Git is an example of a Distributed VCS

Centralized Version Control Systems:

A central server is present, and all team members have to connect to the central server to get the latest copy of the project code. They then update their changes as a new version on the central server.

The disadvantage of a centralized server is the single point of failure. In case of server failure, the team members cannot update their changes to the server, nor can they obtain the recent copy of the project.

Now let's try to install, setup and use Git! Follow the following steps

1. Download and install the latest version of Git from <https://git-scm.com/downloads>
2. Make sure it is installed properly in your system by using the following command in your command prompt:

```
git --version
```

```
C:\Users\Fast>git --version  
git version 2.36.0.windows.1
```

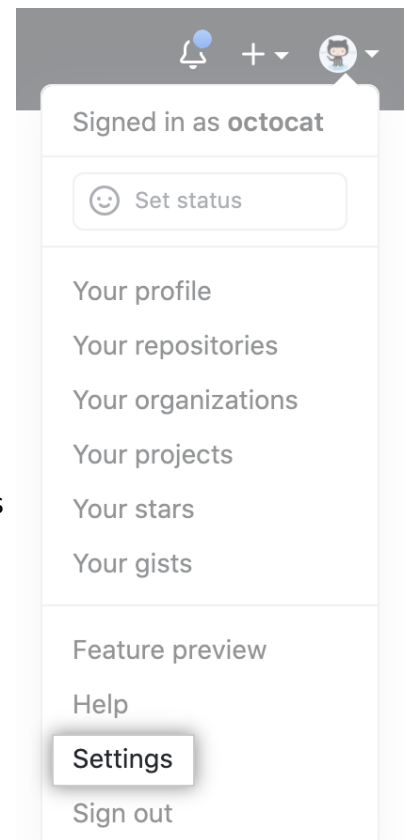
3. Open Git Bash.
4. Set a Git username using the following command:

```
git config --global user.name "Your Name"
```
5. Confirm that you have set the Git username correctly:

```
git config --global user.name
```
6. Set an email address in Git using the following command:

```
git config --global user.email "email@example.com"
```
7. Confirm that you have set the email address correctly:

```
git config --global user.email
```
8. Add the email address to your account on GitHub, so that your commits are attributed to you and appear in your contributions graph.
We will need this later. On your GitHub account:
 - a. In the upper-right corner of any page, click your profile photo, then click **Settings**.
 - b. In the "Access" section of the sidebar, click Emails.
 - c. In "Add email address", type your email address and click **Add**.
 - d. Verify your email address.
 - e. In the "Primary email address" list, select the email address you'd like to associate with your web-based Git operations.



9. In the command prompt (or Git Bash) browse to the directory that you wish to create the repository in. You can do this by typing:

```
cd <directory path here>
```

10. Type the following to create a repository in the location:

```
git init
```

```
C:\Users\Fast>cd C:\Users\Fast\eclipse-workspace
```

```
C:\Users\Fast\eclipse-workspace>git init
```

```
Initialized empty Git repository in C:/Users/Fast/eclipse-workspace/.git/
```

11. You can check the status of your repository by using the following command:

```
git status
```

```
Untracked files:
```

```
(use "git add <file>..." to include in what will be committed)
```

```
.metadata/  
Basic Inheritance/  
ExceptionHandling/  
GenericsExample/  
JavaFXExample/  
Mid2Solution/  
SwingDemo.rar  
SwingDemo/  
WBuilderTest/  
fspecclipse/  
log4j2Example/
```

```
nothing added to commit but untracked files present (use "git add" to track)
```

12. Before we can commit files to Git, we must first stage them. In order to stage the files, we use the following command:

```
git add <name of file/folder you want to add>
```

```
C:\Users\Fast\eclipse-workspace>git add JavaFXExample/
```

Note: If your folder has whitespace, you will get an error, you will need to wrap it in single quotes (''). For example: `git add 'folder with space/'`.

13. You can use `git add .` to add all files and folders to the staging area.
14. A **commit** is a snapshot of our code at a particular time, which we are saving to the commit history of our repository.

```
git commit -m "Commit message"
```

```
C:\Users\Fast\eclipse-workspace>git commit -m "Committing all my projects"
```

```
[master (root-commit) 444a9ae] Committing all my projects
```

```
269 files changed, 21851 insertions(+)
```

```
create mode 100644 .metadata/.bak_0.log
```

```
create mode 100644 .metadata/.lock
```

```
create mode 100644 .metadata/.log
```

```
create mode 100644 .metadata/.plugins/org.eclipse.core.resources/.history/14/b00d5e987a8d001c1f2982340d9a7504
```

```
create mode 100644 .metadata/.plugins/org.eclipse.core.resources/.history/17/80fb859f798d001c1f2982340d9a7504
```

```
create mode 100644 .metadata/.plugins/org.eclipse.core.resources/.history/1a/50ab986920c1001c1647f8a12ef37167
```

```
create mode 100644 .metadata/.plugins/org.eclipse.core.resources/.history/1e/901f5bb1728d001c1f2982340d9a7504
```

```
create mode 100644 .metadata/.plugins/org.eclipse.core.resources/.history/2/207ef16d20c1001c1647f8a12ef37167
```

15. To view the commit, you can use the command: `git log`

```
C:\Users\Fast\eclipse-workspace>git log
commit 444a9aebd7eda9e346c25220f11afbbbad786e7a (HEAD -> master)
Author: abeeha.sattar13 <abeeha.sattar13@outlook.com>
Date: Thu Apr 28 13:08:16 2022 +0500

    Committing all my projects
```






16. You can also set it up so that you can ignore some files. First of all, navigate to the location of your repository (which you should be in currently, if you have followed the steps accurately). Type the following to create a `.gitignore` file. If it is successful, there will be no message.

`touch .gitignore`

17. If you want to ignore a file that is already checked in, you must untrack the file before you add a rule to ignore it. From your terminal, untrack the file using the following command.

`git rm --cached -r FILENAME`

18. Add the files and folders that you want to ignore inside the `.gitignore` file.

	.metadata	2/14/2022 10:39 AM	File folder
<input type="checkbox"/>	FXDemo	3/31/2022 11:01 PM	File folder
	HelloWorld	2/14/2022 10:40 AM	File folder
<input checked="" type="checkbox"/>	Ignored	5/5/2022 10:21 PM	File folder
	JavaFXProject	3/28/2022 10:12 PM	File folder
	Log4j2Demo	3/29/2022 9:41 PM	File folder
	.gitignore	5/5/2022 10:29 PM	Text Document 1 KB

```
#####
#For Comments#
#####
#Folder Name:
Ignored
```

19. Use the `git status` command to check on the status of your repository.

```
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:   .gitignore

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .metadata/
```

20. Commit the updated `gitignore` file.
21. When we initialize a repository and start making commits, they are saved to the master branch by default. We can create a branch using the following command:

`git branch <new-branch-name>`

There will be no message if it is successful.

22. To display a list of all the branches, you can use the following command:

```
git branch
khan@DESKTOP-PTFQS28 MINGW64 ~/eclipse-workspace (master)
$ git branch
  br1.1
* master
```

23. To switch branches, you have to “checkout” a branch by using the following command:

```
git checkout <branch-name>
khan@DESKTOP-PTFQS28 MINGW64 ~/eclipse-workspace (master)
$ git checkout br1.1
Switched to branch 'br1.1'
```

24. To create a new branch and change to it at the same time, you can use the -b flag:

```
git checkout -b <branch-name>
khan@DESKTOP-PTFQS28 MINGW64 ~/eclipse-workspace (br1.1)
$ git checkout -b br1.2
Switched to a new branch 'br1.2'
```

25. To merge the changes from a different branch into your current branch, you can use this command:

```
git merge <branch-name>
khan@DESKTOP-PTFQS28 MINGW64 ~/eclipse-workspace (br1.2)
$ git merge master
Already up to date.
```

26. Finally, if you want to delete a branch, you can use this command:

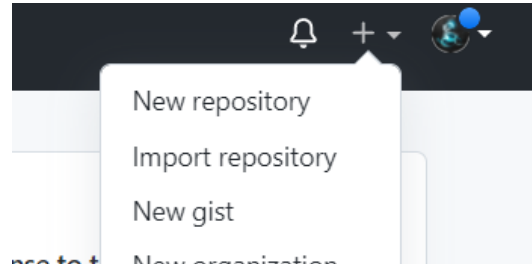
```
git branch -d <branch-name>
khan@DESKTOP-PTFQS28 MINGW64 ~/eclipse-workspace (br1.2)
$ git branch -d br1.1
Deleted branch br1.1 (was c9b5c8e).
```

GitHub

GitHub provides a web based interface for git. It also provides an online server where you can store the files. (Remember that Git stores the files locally and not on a server)

First of all, create an account on GitHub, and proceed with the following steps:

1. Create a repository by clicking on the plus icon:



2. Fill out the repository name (do not use spaces), and create a repository with default options:

Owner * Repository name *

Abeeha-Sattar / SCD

Great repository names are short and memorable. Need inspiration? How about [cuddly-guide?](#)

Description (optional)

☒ **Public**
Anyone on the internet can see this repository. You choose who can commit.

☐ **Private**
You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.

☐ **Add a README file**
This is where you can write a long description for your project. [Learn more.](#)

Add .gitignore
Choose which files not to track from a list of templates. [Learn more.](#)

.gitignore template: **None** ▼

Choose a license
A license tells others what they can and can't do with your code. [Learn more.](#)

License: **None** ▼

You are creating a public repository in your personal account.

Create repository

3. After you have created your repository, let's push some changes to it. First of all we have create an origin to which we can push to. Use the following lines to create an origin:

```
git remote add origin https://github.com/Abeeha-Sattar/SCD.git
```

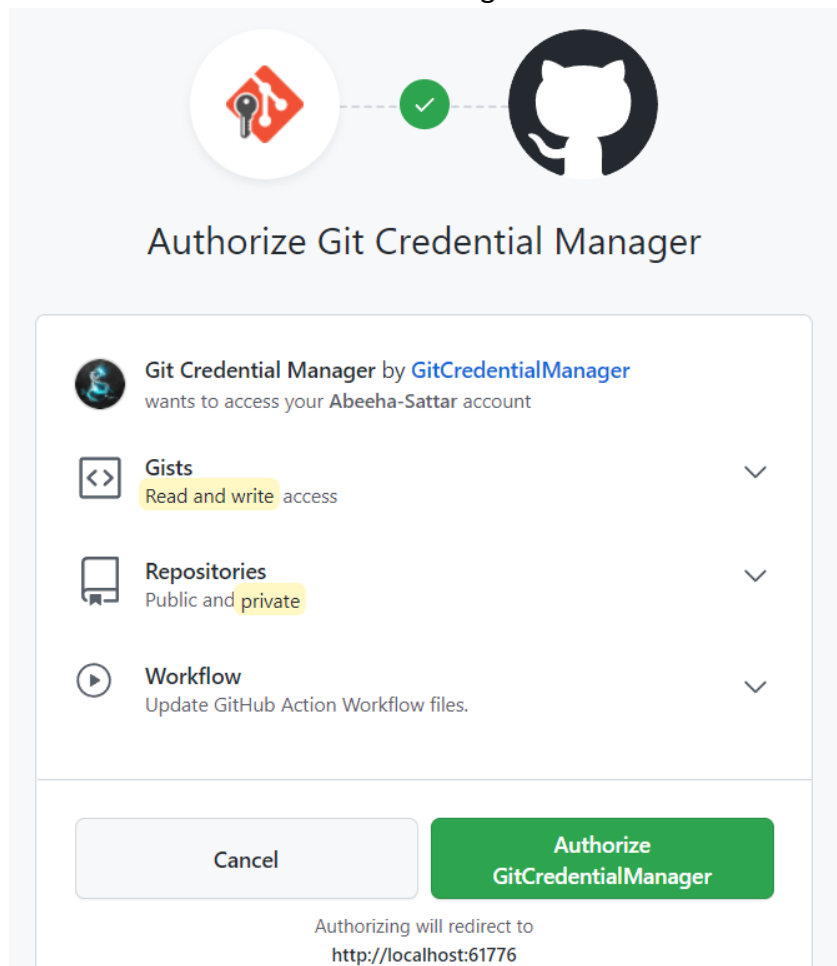
```
khan@DESKTOP-PTFQS28 MINGW64 ~/eclipse-workspace (br1.2)
$ git remote add origin https://github.com/Abeeha-Sattar/SCD.git
```

4. To push to origin, use the following command:

```
git push -u origin <branchname>
```

```
khan@DESKTOP-PTFQS28 MINGW64 ~/eclipse-workspace (br1.2)
$ git push -u origin master
```

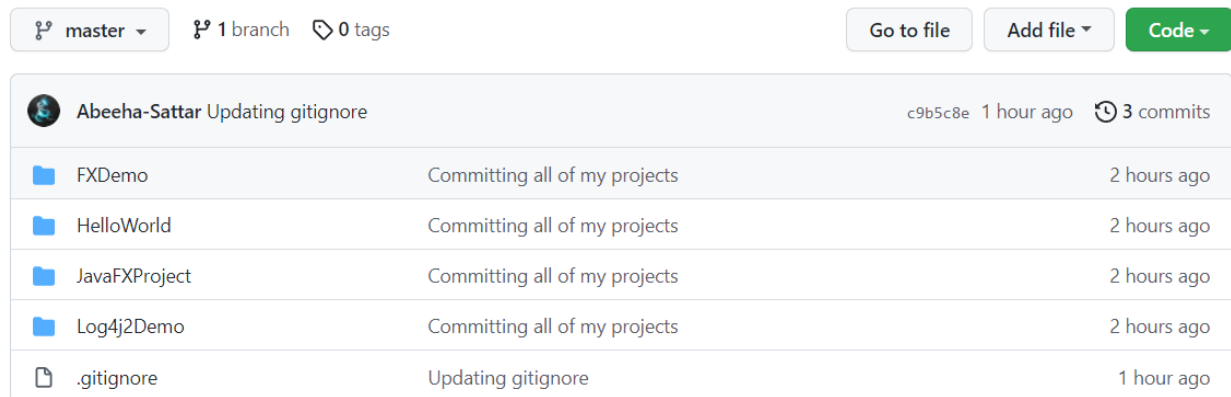
5. You will be asked to authorize the login. Authorize via browser.



6. You will be able to view the progress on your git bash/cmd

```
khan@DESKTOP-PTFQS28 MINGW64 ~/eclipse-workspace (br1.2)
$ git push -u origin master
Enumerating objects: 237, done.
Counting objects: 100% (237/237), done.
Delta compression using up to 4 threads
Compressing objects: 100% (175/175), done.
Writing objects: 61% (146/237), 2.76 MiB | 124.00 KiB/s
```

7. Refresh your repository page after you have pushed your files successfully.



The screenshot shows a GitHub repository interface. At the top, there are buttons for "Go to file", "Add file", and "Code". Below this, a commit by "Abeeha-Sattar" is shown with the message "Updating gitignore". A list of files is displayed below the commit:

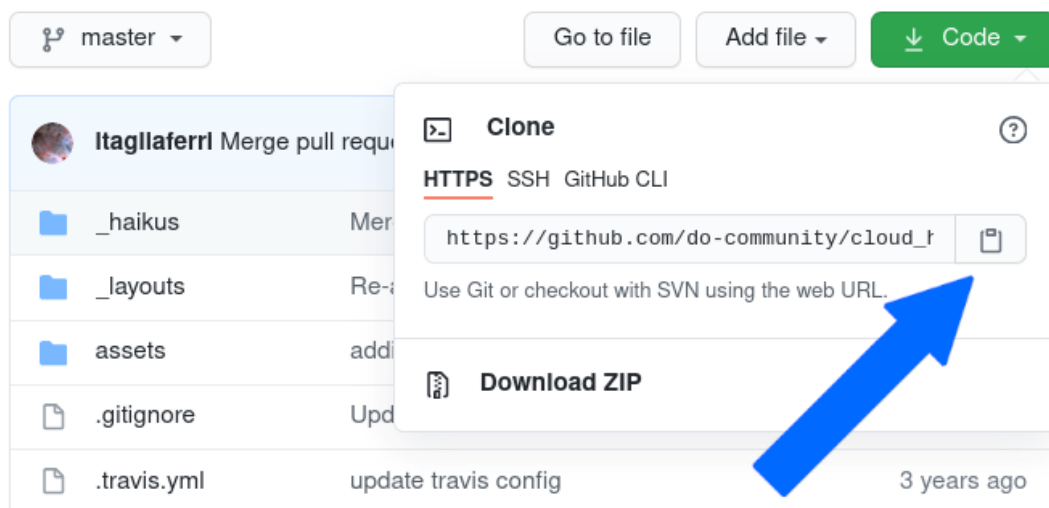
File	Commit Message	Time
FXDemo	Committing all of my projects	2 hours ago
HelloWorld	Committing all of my projects	2 hours ago
JavaFXProject	Committing all of my projects	2 hours ago
Log4j2Demo	Committing all of my projects	2 hours ago
.gitignore	Updating gitignore	1 hour ago

8. Create a new branch where you will get the recent copy of the project from the repository – this is called cloning (using bash or cmd)

```
git checkout -b <branch-name>
```

9. Clone the repository using the following command:

```
git clone https://github.com/your-username/repository.git
```



The screenshot shows a GitHub repository page for "Itagllaferri". A "Clone" dialog box is open, displaying the repository URL: "https://github.com/do-community/cloud_". A blue arrow points to the "Code" button in the top right corner of the repository page.

10. You can view the progress as it is being cloned:

```
khan@DESKTOP-PTFQS28 MINGW64 ~/eclipse-workspace (1.3)
$ git clone https://github.com/Abeeha-Sattar/SCD.git
Cloning into 'SCD'...
remote: Enumerating objects: 237, done.
remote: Counting objects: 100% (237/237), done.
remote: Compressing objects: 100% (146/146), done.
Receiving objects: 57% (136/237), 1.81 MiB | 187.00 KiB/s
```

Git and GitHub with Eclipse

1. Make sure that EGit is installed on your Eclipse, via the Eclipse Marketplace.


Eclipse Marketplace

Select solutions to install. Press Install Now to proceed with installation.
Press the "more info" link to learn more about a solution.



Search Recent Popular Favorites Installed Giving IoT an Edge

Find: All Markets All Categories Go




EGit - Git Integration for Eclipse 6.0.0

EGit is the Git integration for Eclipse. Git is a distributed versioning system, which means every developer has a full copy of all history of every revision of...

[more info](#)

by [Eclipse.org](#), EPL

[egit](#) [jgit](#) [git](#) [dvcs](#) [scm](#)

★ 1729  Installs: **615K** (1,123 last month) Installed

2. Click on the 'Window' menu bar option, then choose 'Preferences'.
3. Type "git" in the search bar, then choose that path 'Team > Git > Configuration'. Click 'Add Entry...'.
4. Add your email and name as entries:

Configuration

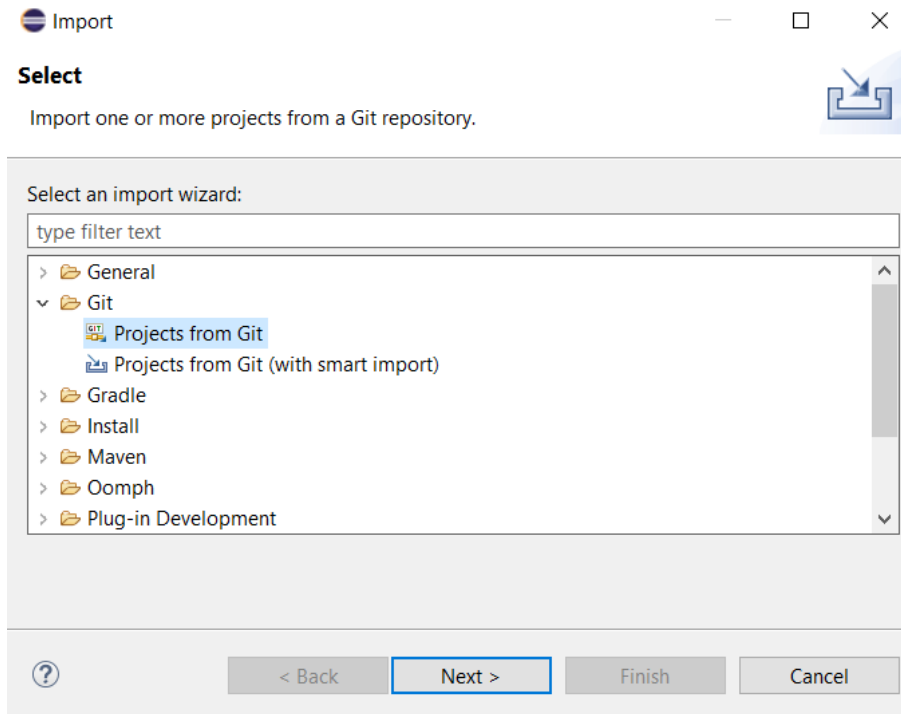
Repository Settings User Settings System Settings

Location: Open

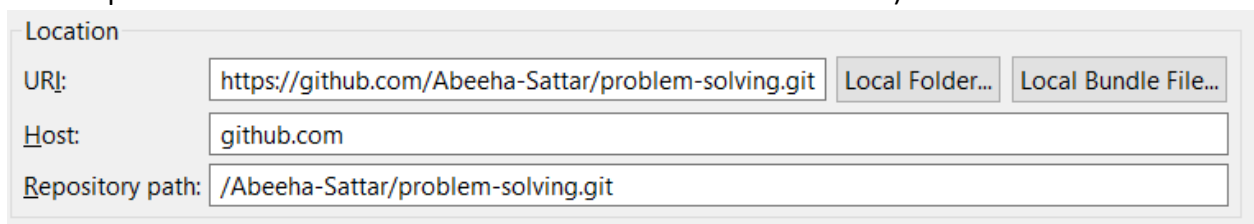
Key	Value
▼ core	
excludesfile	C:/Users/khan/.gitignore_global
▼ user	
email	abeeha.sattar13@gmail.com
name	Abeeha Sattar

Add Entry...
Remove

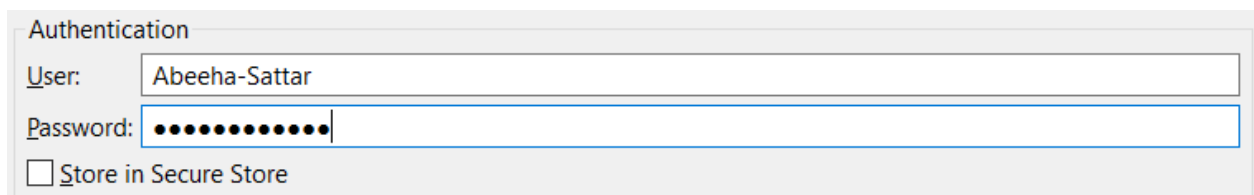
5. Next, we will create a project with GitHub and Eclipse.
6. Create a new repository on GitHub for your project.
7. Click the green 'Clone or download' button
8. A menu appears, which should say 'Clone with HTTPS'; if it instead says 'Clone with SSH', click on 'Use HTTPS'
9. Click on the copy-to-clipboard button
10. In Eclipse, choose 'File' (top left), then 'Import...'
11. In the dialog that opens, choose 'Git > Projects from Git' and click 'Next'



12. Click on 'Clone URI', then click 'Next'. It will autofill some fields for you. (Make sure that you have copied the link otherwise the information will not be autofilled)



13. Ensure that your GitHub username and password are entered under 'Authentication', and click Next.



14. Select the main branch, and click 'Next'.
15. Choose a directory where the local copy of the repo should live, and click 'Next'.

Local Destination

Configure the local storage location for problem-solving.



Destination

Directory:

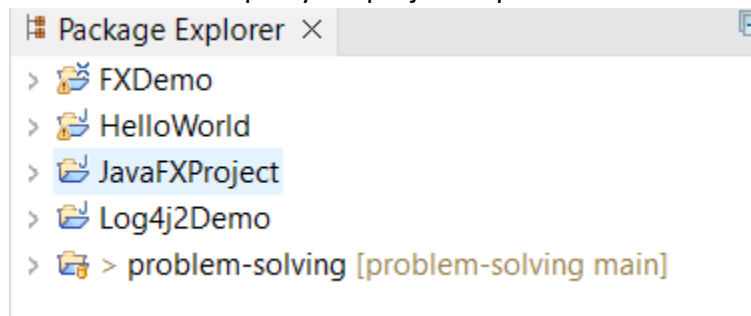
Initial branch:

☐ Clone submodules

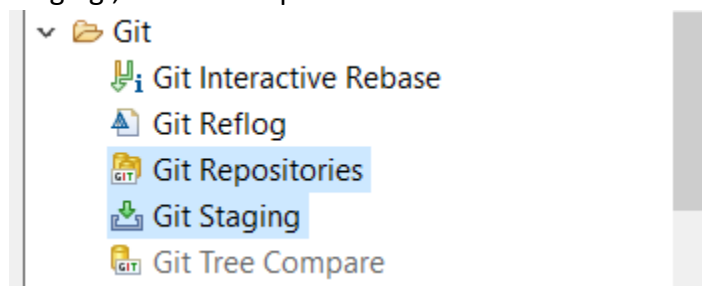
Configuration

Remote name:

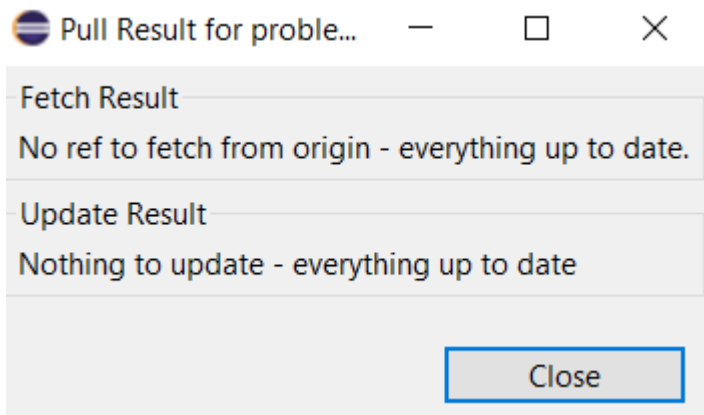
16. On the next screen choose 'Import as general project', click 'Next'
17. On the final screen, keep the default Project name, which should match the repo name, and click 'Finish'
18. It will now show up in your project explorer.



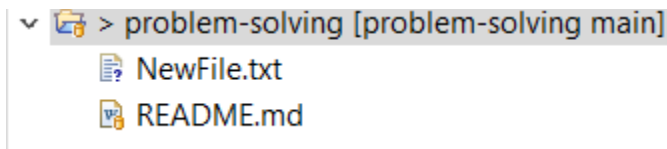
19. To make the 'Git Repositories' and the 'Git Staging' windows visible go to 'Window > Show View > Other...'
20. Expand 'Git' (click on the little arrow on the left) and choose 'Git Repositories' and 'Git Staging', then click Open.



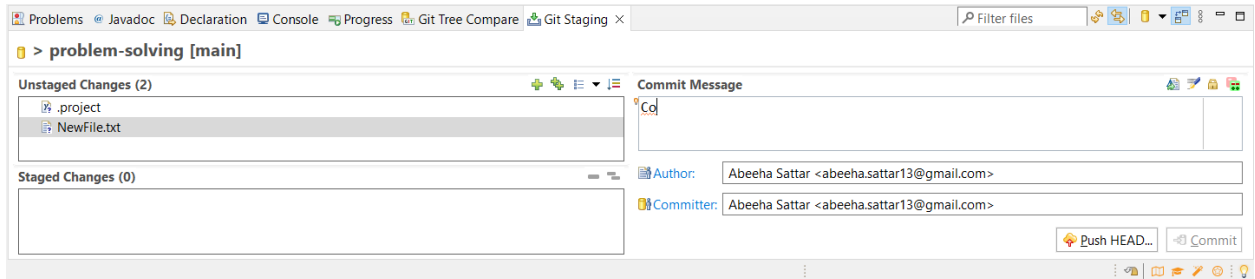
21. In the 'Git Repositories' window, right-click on the project and select 'Pull'. Click ok the status message that appears, telling you whether anything was downloaded from the remote repository to your local machine.



22. In your project explorer, go to your project and create a new text file in it. Write anything within the text file.



23. Now go to the 'Git Staging' window, press Refresh, and drag the 'NewFile.txt' file (only this file) from the 'Unstaged Changes' area to the 'Staged Changes' area.



24. In the 'Commit Message' area, write a short descriptive summary of your changes, like "Committed NewFile.txt"
25. Ensure your 'Author' and 'Committer' fields are filled with your username and e-mail.
26. Before we can commit, first we need to go back to GitHub > Settings > Developer Settings > Personal Access Tokens. Click on Generate New Token. Fill out the Note, set the Expiration to No Expiration, and then select "repo", and generate the token. Make sure to copy your personal access token now. You won't be able to see it again!

Note

Eclipse Repo Token

What's this token for?

Expiration *

No expiration  The token will never expire!

GitHub strongly recommends that you set an expiration date for your token to help keep your account secure. [Learn more](#)

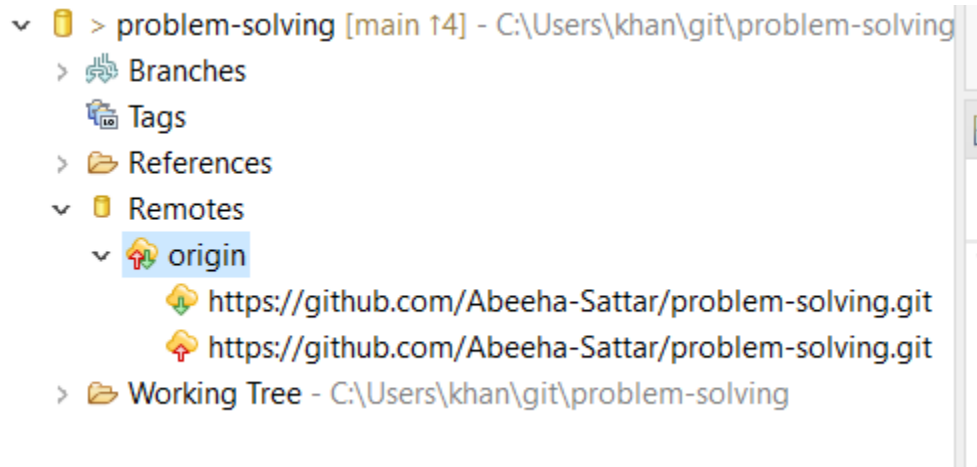
Select scopes

Scopes define the access for personal tokens. [Read more about OAuth scopes.](#)

<input checked="" type="checkbox"/> repo	Full control of private repositories
<input checked="" type="checkbox"/> repo:status	Access commit status
<input checked="" type="checkbox"/> repo_deployment	Access deployment status
<input checked="" type="checkbox"/> public_repo	Access public repositories
<input checked="" type="checkbox"/> repo:invite	Access repository invitations
<input checked="" type="checkbox"/> security_events	Read and write security events

27. On Eclipse, in the Git Repositories view:

28. Right-click the Remotes sub-node for GitHub (origin or the name you have chosen when you have cloned the repository) and choose Configure Push...



29. Click the Change... button to change the URI in the upper right

30. Replace the password with the copied generated GitHub token

31. Click Finish and Save to apply the changes

32. Select Commit and Push on the Git Staging Window.

33. Go to GitHub and view the changes!