New Test Version 3????

1. What is a version control system?

A system that allows you to revert selected files back to a previous state, revert the entire project back to a previous state, compare changes over time, see who last modified something that might be causing a problem, who introduced an issue and when, and more.

1. What is Git?

Git is a Distributed Version Control System that helps you manage your project. Every time you commit, or save the state of your project, Git basically takes a picture of what all your files look like at that moment and stores a reference to that snapshot. To be efficient, if files have not changed, Git doesn’t store the file again, just a link to the previous identical file it has already stored.

1. Reasons to love Git
2. **History**

* Git keeps track of every change we made to our project. It tells us exactly on which day we changed which parts. Even it shows us which line of code we added, which line of code we removed.
* Most operations in Git need only local files and resources to operate. You have the entire history of the project right there on your local disk. For example, to browse the history of the project, Git doesn’t need to go out to the server to get the history and display it for you — it simply reads it directly from your local database. This means you see the project history almost instantly. If you want to see the changes introduced between the

current version of a file and the file a month ago, Git can look up the file a month ago and do a local difference calculation, instead of having to either ask a remote server to do it or pull an older version of the file from the remote server to do it locally.

1. **Collaboration**

More productive when working in a team. You don’t need to wait until others finish theirs work. You can continue to work on your changes. It merges everyone change, gives everyone the most updated copy of the files.

1. **Feature branches**

* Git tracks branches separatedly from one another
* Git push all files into a hiden system folder locally

1. The Three States

Git has three main states that your files can reside in: *modified*, *staged*, and *committed*:

• Modified means that you have changed the file but have not committed it to your database yet.

• Staged means that you have marked a modified file in its current version to go into your next

commit snapshot.

• Committed means that the data is safely stored in your local database.

A close up of a logo

Description automatically generated

* The working directory is a single checkout of one version of the project. These files are pulled out of the compressed database in the Git directory and placed on disk for you to use or modify.
* The staging area is a file, generally contained in your Git directory, that stores information about what will go into your next commit.
* The Git directory is where Git stores the metadata and object database for your project. This is the most important part of Git, and it is what is copied when you *clone* a repository from another computer.

1. The basic Git workflow

1) You modify files in your working tree.

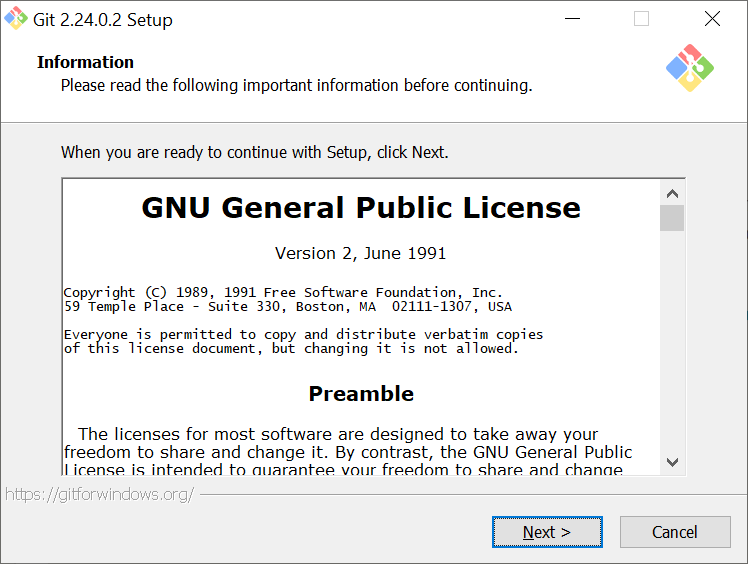
2) You selectively stage just those changes you want to be part of your next commit, which adds

*only* those changes to the staging area.

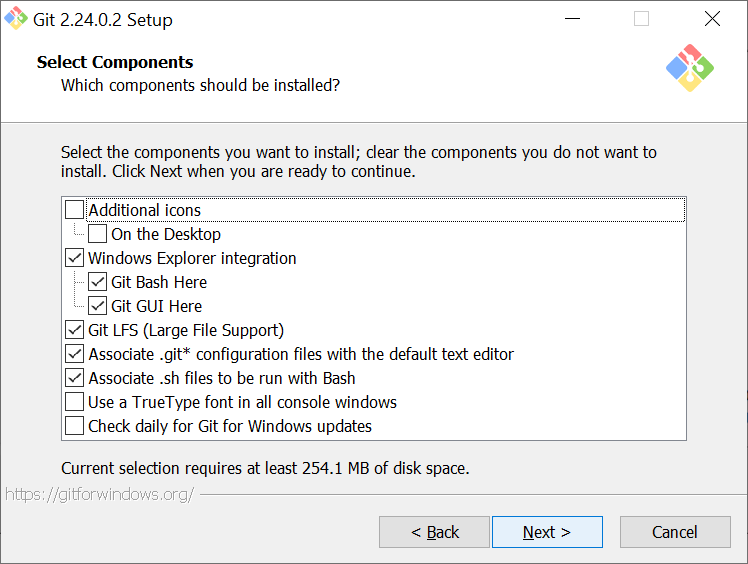
3) You do a commit, which takes the files as they are in the staging area and stores that snapshot permanently to your Git directory.

1. Installing Git in Windows

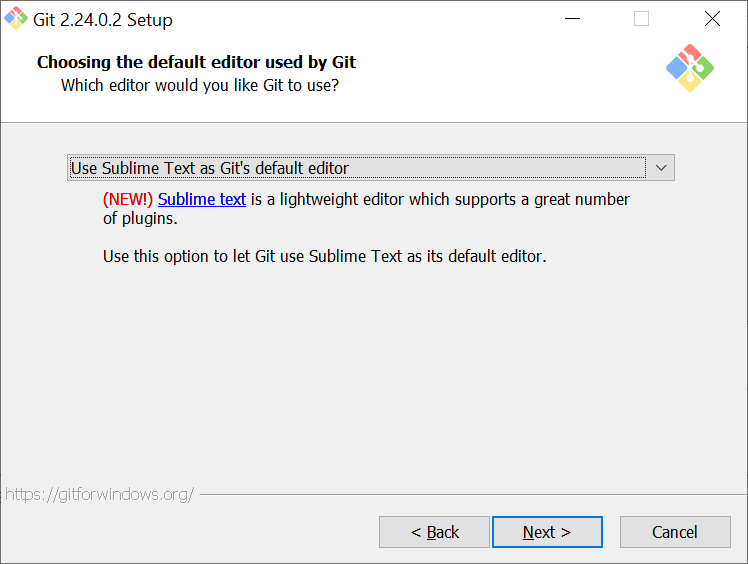
* Go to https://git-scm.com/download/win and the download will start automatically. Note that this is a project called Git for Windows, which is separate from Git.
* Double click on downloaded file to start installation process



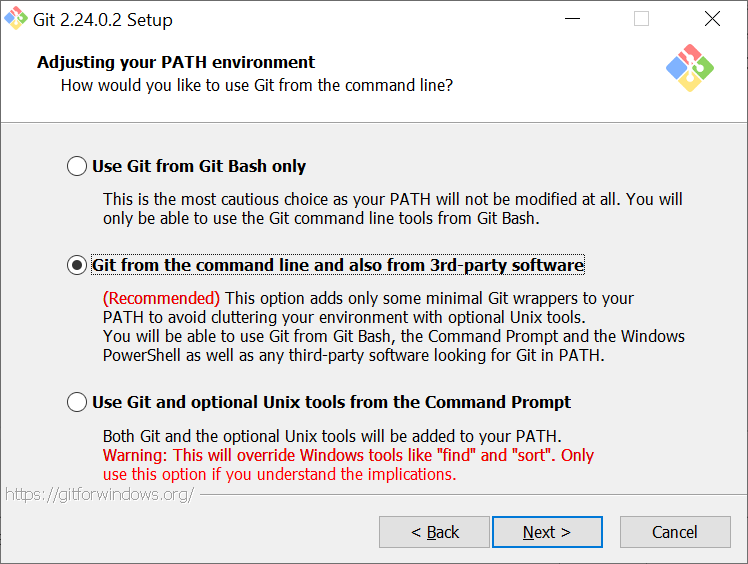
* Select component you want to install by check it then click “Next” button.



* Select a default editor used by Git then click “Next” button.



* Select an option how you would like to use Git from the command line then click “Next” button.



* For the next dialogues just leave default options and click “Next” buttons until you see the “Finish” button.

1. Create a GitHub account

* Go to <https://github.com/> to create an account
* Enter your username, email and password, then click on “Sign up for GitHub button”
* Pick a plan
* Open your email to verify your email in GitHub account by clicking on a link in a sent email.

1. Basic about GitHub
   * What is GitHub
   * [Create a Repository](https://guides.github.com/activities/hello-world/#repository)
   * [Create a Branch](https://guides.github.com/activities/hello-world/#branch)
   * [Make a Commit](https://guides.github.com/activities/hello-world/#commit)
   * [Open a Pull Request](https://guides.github.com/activities/hello-world/#pr)
   * [Merge Pull Request](https://guides.github.com/activities/hello-world/#merge)
   1. Create a new repository

A repository is a container that contains all the files for your project, including the revision history.

Do following steps to create a new repository:

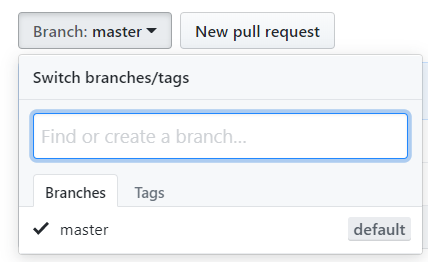
* After loging in GitHub, navigate to the top-right of the page, click on little cross beside your name, and choose on “Create a new repository”.
* Give it a name and a description. The given name will ultimately form the URL under which your repository will be registered. The description is optional but is recommended.
* Select Public vs Private option for your repository. Generally, you go with public, unless you do not want your files to be seen by everybody. However, the private repositories come with a price.
* Check the box “Initialize this repository with a README” to add a README.md file to your project. The file usually includes comprehensive information about the project you are hosting under your repository, such as installation guides, and build and usage instructions, as well as guidelines on how you can contribute. You can always add a README file later if you leave this option unchecked for the time being.
* Choose a programming language and type of license on the two boxes right above the “Create repository” button.

Bellow snapshot shows the “Create a new repository” page.

A screenshot of a social media post

Description automatically generated

* 1. Create a Branch
* Branching is the way to work on different versions of a repository at one time.
* By default, your repository has one branch named master which is considered to be the definitive branch. We use branches to experiment and make edits before committing them to master.
* When you create a branch off the master branch, you’re making a copy, or snapshot, of master as it was at that point in time. If someone else made changes to the master branch while you were working on your branch, you could pull in those updates.
* At GitHub, developers, writers, and designers use branches for keeping bug fixes and feature work separate from our master (production) branch. When a change is ready, they merge their branch into master.
* To create a branch, click on “Branch” combo box, type name for a new branch, then hit Enter



* 1. Make and commit changes
* On GitHub, saved changes are called *commits*. Each commit has an associated *commit message*, which is a description explaining why a particular change was made. Commit messages capture the history of your changes, so other contributors can understand what you’ve done and why.
* To make and commit a change on a branch:
  + Click the README.md file.
  + Click the pencil icon in the upper right corner of the file view to edit.
  + In the editor, write a bit about yourself.
  + Write a commit message that describes your changes.
  + Click “Commit changes” button.
* These changes will be made to just the README file on your current branch, so now this branch contains content that’s different from master.
  1. Open a Pull Request
* Now that you have changes in a branch off of master, you can open a pull request.
* Pull Requests are the heart of collaboration on GitHub. When you open a pull request, you’re proposing your changes and requesting that someone review and pull in your contribution and merge them into their branch. Pull requests show diffs, or differences, of the content from both branches. The changes, additions, and subtractions are shown in green and red.
* As soon as you make a commit, you can open a pull request and start a discussion, even before the code is finished.
* By using GitHub’s [@mention system](https://help.github.com/articles/about-writing-and-formatting-on-github/#text-formatting-toolbar) in your pull request message, you can ask for feedback from specific people or teams, whether they’re down the hall or 10 time zones away.
* You can even open pull requests in your own repository and merge them yourself. It’s a great way to learn the GitHub flow before working on larger projects.
* To open a Pull Request:
  + Click the **Pull Request** tab, then from the Pull Request page, click the green **New pull request** button.
  + In the **Example Comparisons** box, select the branch you made, readme-edits, to compare with master (the original).
  + Look over your changes in the diffs on the Compare page, make sure they’re what you want to submit.
  + When you’re satisfied that these are the changes you want to submit, click the big green **Create Pull Request** button.
  + Give your pull request a title and write a brief description of your changes. When you’re done with your message, click **Create pull request**!
  1. Merge your Pull Request
* To merging your branch into the master branch
  + Click the green Merge pull request button to merge the changes into master.
  + Click Confirm merge.
  + Go ahead and delete the branch, since its changes have been incorporated, with the Delete branch button in the purple box.
  1. The Raw, Blame, and History buttons
* The Raw, Blame, and History buttons appear when viewing a single file of a repository. For example, let's visit the README.md file by clicking on it:

A screenshot of a cell phone

Description automatically generated

* The Raw button, like the name suggests, opens the file in a raw form, meaning that any HTML formatting disappears. This is particularly useful when you want to download a single file. You will notice that many guides on the internet use this raw file format when they tell you to download something using command-line tools, such as *wget* or *curl*. If you have ever tried to download a file from GitHub and all you got was an HTML file, remember the usage of raw.
* The Blame button makes use of Git's blame function. Basically, for each line of a file, Git informs you about who modified that line and when that line was modified. If you want to know more, visit <https://git-scm.com/docs/git-blame>.
* The History button is nothing more than what git log does for a particular file.
  1. The Watch, Star, and Fork buttons
* You've probably spotted the three buttons sitting at the top-right corner of your repository page. These appear for every public repository.
  + The Watch button manages the level of subscription in a repository. GitHub notifies you with an email whenever an action takes place in a repository you follow and, at the same time, it lists them in the Notifications area (<https://github.com/notifications>) where you can later mark them as read, as shown in the following screenshot:

A screenshot of a social media post

Description automatically generated

* + There are four levels of subscription as can be seen in the above screenshot. The default behaviour when you create a new repository is Watching.
  + The Star button is a way to show your appreciation to a repository and its creator. It depicts the popularity of a project. Whenever you star a repository, it gets added to your list of starred repositories. You can see all your starred repositories at <https://github.com/stars>.
  + The Fork button is to use when you want to contribute to a project. When you fork a repository, it gets copied in your own namespace, and that way you have full ownership of that copy; thus, you are able to modify anything you want.
  1. Creating a new issue
* An issue tracker is that of a bug tracker, since reporting and discussing bugs play a vital role in the growth of your project. It can also be used to make feature requests, serve as a discussion board of a blog or a project, and even be used as a notepad for repairing your house!
* To create a new issue, go to https://github.com/<username>/<repository>/issues for an overview of all issue activity. If no one has ever opened an issue in your project, you will be presented with a blank page with GitHub prompting you to open a new issue. Let's go ahead and do this. Click on the big green button that says New issue.
* An issue can be created when you provide the minimum of a title. Look at the following screenshot carefully where the Submit new issue button is grayed out and cannot be clicked. The title should be as descriptive as possible of the message you are trying to pass when creating an issue.
* Below, under the Write tab, you can provide the details and essentially start a discussion with everyone who wants to participate (if the repository is public, that is). That's why GitHub cleverly suggests to Leave a comment.

A screenshot of a social media post

Description automatically generated

* Besides writing, you can also attach images by simply dragging and dropping, or by selecting them using folder navigation.
* Next to the Write tab is the Preview tab. Preview shows what the result will be when you submit the issue. It will style the regular text of the Write tab into a meaningful text accordingly, with URLs properly formatted, images shown, emojis displayed, and so on.
* Feeling ready to submit it? Hit Submit new issue at the bottom of the page.
* Each created issue is assigned a unique number that we can later use in other issues for reference. In our example, since this was the very first issue, it was assigned the number #1. Some useful information is provided in the title area. You can see that the issue is marked as Open, the username of the person who created it, the time it was created, and how many comments there are.
* If you later realize that you made a mistake, don't panic—you can always edit the issue you created. The Edit button allows you to edit the title, and the pencil icon is used for editing the description. Close the issues using the Close issue button.
  1. Assigning issues to users
* A repository can have more than one collaborator. A collaborator is a person who has push access to the repository and, in our case, can also edit and close issues.
* User assignment works well in repositories with large traffic where a team is involved and is responsible for bug fixes, enhancements, and so on.
* There is an Assignee section inside each issue. Click on a gear icon next to the “Assignees” button and type or choose a name for a user whom you want to assign the issue to as shown in the following screenshot:

A screenshot of a social media post

Description automatically generated