Cengage

INSTALLING GIT

Please note that the installers options changes from version to version. Please follow this guide tentatively and it is not necessarily a step-by-step guide.

Installing Git on Windows

1. Visit the official Git website and download the latest version of Git for Windows: https://git-scm.com/download/win
2. Double-click the downloaded file to start the installation process.
3. Follow the on-screen instructions to complete the installation.
   Note: *Make sure to select the option to add Git to your system's PATH environment variable if it is not selected by default.* This will allow you to run Git from the command line. You can leave everything else as default.
4. Once the installation is complete, open the command prompt or PowerShell by clicking on the Windows icon and type in/search for “command prompt” or “powershell” respectively and open the application, then enter the following command to verify that Git was installed successfully:

   ```
   git --version
   ```

   This should display the version number of Git that you installed.

Installing Git on Mac

1. You can download Command Line Tools for Xcode from the Apple Developer Website: https://developer.apple.com/xcode/
2. Select Download -> Website -> Command Line Tools for Xcode. Or you can try download Homebrew and install Git directly.
3. Once Command Line Tools for Xcode is installed, open the Terminal app on your Mac and enter:

   ```
   git --version
   ```

   This should display the version number of Git that you installed.

If you don't have Homebrew installed on your Mac, you can download it from the official Homebrew website. Please follow the instruction on their website to download and install it.
Once the installation is complete, enter the following command to install Git (make sure to follow the instruction at the end):

```
brew install git
```

Cengage

LEARN GIT IN AN HOUR (OR LESS)!

Trust us, installing Git is probably harder than learning how to use it! You are more than half-way there.

Let's say you're working on a website called "MyWebsite" and you want to use Git to track changes to your files as well as potentially working with other people on this project. Please type these git commands in your terminal or command line. Here's an overview of how you could do it:

1. Setting your identity (for first usage only)
2. Initializing a repository (git init)
3. Adding files to the repository (git add)
4. Committing changes (git commit)
5. Pushing changes (git push)
6. Pulling changes (git pull)
7. Creating a new branch and add changes (git branch, git checkout, and repeat step 3,4,5)

1. Setting Your Identity

The first thing you should do when you install Git is to set your username and email address. This is important because every Git commit uses this information.

```
git config --global user.email your.email@email.com
```

Note: *Replace your.email@email.com with your real email address.*

```
git config --global user.name "Your Username"
```

Note: *Replace your username with your name. The quotes must be included.*

This only needs to be completed once after install.

2. Initializing a Repository

A repository is a centralized location where you store all the files and code related to a particular project. It is like a virtual closet or storage space for all your project files. Git tracks changes made to the files and stores those changes in the repository, allowing you to view or revert to previous versions of your project at any time. Additionally, you can use Git to collaborate with others by sharing the repository with them and allowing them to make their own changes and commit them to the repository.

Open your terminal or command prompt, navigate to the directory where you want to create your repository, and enter the following command:

```
git init
```

This initializes a new Git repository in the current directory.

Tips: to quickly open the terminal or command prompt click on the link for MacOS. For Windows type "cmd" in the address bar of your explorer.

3. Adding Files to the Repository

You can use any text editor that you want, but we suggest Sublime Text for this task. Create a new file called `index.html` in the current directory, and add the following code to the file using your text editor:

```
<html>                      This is if you are building web sites.
  <head>
    <title>MyWebsite</title> For programming source code, create the
  </head>                    source file, e.g. main.cpp
  <body>
    <h1>Welcome to MyWebsite</h1>
    <p>This is a sample website.</p>
  </body>
</html>
```

Then, enter the following command to add the file to Git:

```
git add index.html          use the source you created, e.g. main.cpp
```

4. Committing Changes

Once you've added the file, you need to commit your changes. In Git, a commit is like a snapshot of your files at a particular point in time. When you make changes to a file that is being tracked by Git, Git creates a new snapshot of the file and stores it as a commit. Each commit has a unique identifier (a long string of letters and numbers) that allows you to refer to it later.

Tips: Think of a commit like taking a picture of your files. You can take as many pictures as you want, and each picture captures the state of your files at a particular moment in time.

Enter the following command:

```
git commit -m "Added index.html"      in our case: "Added main.cpp"
```

This creates a new commit with your changes and a message describing what you did.

5. Pushing Changes to a Remote Repository

To share your changes with others who are working on the same project, you can push your changes to a remote repository (like GitHub). First, you will need to create a new repository on GitHub.

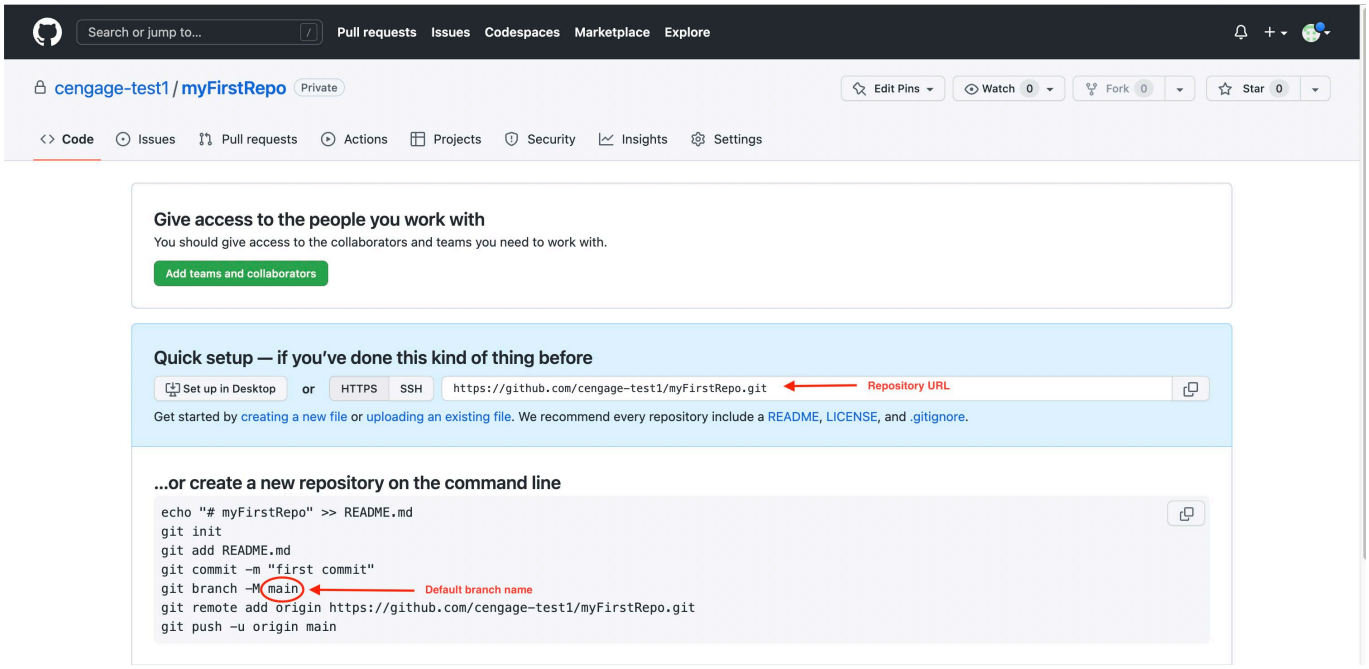
Creating a new repository on GitHub:

To create a new repository on GitHub, follow these steps:

1. Sign in (create an account if you don't have one) to your GitHub account and click on the "+" icon in the top-right corner of the screen.
2. Select "New repository" from the dropdown menu.
3. Choose a name for your repository and add a brief description if you like.
4. Select ~~whether you want your repository to be public (visible to everyone) or~~ private (visible only to you and those you give access to).

Keep this private as it is individual work

5. Choose whether to initialize your repository with a README file (for the purpose of this tutorial only**, do not create a README file**).
6. Click "Create repository" to create your new repository.



Note: Most account will have the default branch name as “main”. If you have an older GitHub account, it maybe different. Please replace “main” with the default branch name you have the following steps.

Once you've created the Github repository, copy the URL of the repository. Then, enter the following command:

```
git remote add origin <URL>
```

This adds the remote repository as the "origin". Use the URL of the repository you just created on GitHub for this example.

```
git push -u origin main
```

This pushes your changes to the "main" branch. The “-u” flag stands for “upstream” ; it tells Git to default the upstream branch to this branch that we are working on. So later on you won’t have to retype it, you can just do "`git push`" instead.

Note: You might be ask to enter your username and password. However, this **is not your Github’s password**. It is your personal access token. Please follow this instruction to create one and use it as your password.

6. Pulling Changes from a Remote Repository

Let's say someone else on your team made changes to the `index.html` file and pushed their changes to the remote repository. To download their changes and merge them with your local copy of the repository, enter the following command:

```
git pull origin main
```

This downloads the changes made by others and merges them with your local copy of the repository.

7. Creating a New Branch and Adding New Changes

Let's say you want to add a new feature to your website without affecting the original version. To create a new branch called "new-feature", enter the following commands:

```
git branch new-feature
git checkout new-feature
```

The first command creates a new branch call “new-feature”, the second command will switch/change your current branch to said branch.

Now, add this to the end of your `index.html` file

```
to main.cpp
```

```
<style>
  h1 {
    color: blue;
  }
  p {
    color: green;
  }
</style>
```

```
add a new function instead
```

This adds some CSS code to your website that makes the heading blue and the paragraph text green.

Save the file and enter the following commands:

```
          main.cpp
git add index.html
git commit -m "Added color to the new-feature branch"
```

```
change the red text to describe what the function does
```

This creates a new commit with your changes and a message describing what you did. You already did this in step 2 and 3. It is really just the same thing again!

After this, Git will save your recent change. If you want your collaborators to see it, you will need to push it though. Please refer back to Step 4 for it, remember that you are working on a different branch this time, so the branch name should no longer be “master".

Tips: Remember, you can switch back to the original version (the master branch) at any time by entering the following command:

```
git checkout main
```

Well done, young wizard! You have now acquired the magical powers of Git, the tool that allows you to track changes to your code and work with your fellow wizards on a project. With a bit of practice and dedication, you'll be able to use Git like a true wizard and create multiple versions of your projects with ease. So, grab your wand (or keyboards) and hop on your broomstick (maybe computers) - the world of Git awaits you!

Chapters 1 and 2 of this ebook will cover most of what you would be doing

https://git-scm.com/book/en/v2

cheat sheet is on next page below

Cengage

GIT CHEAT SHEET

Basic Commands:

- **git init**: Initializes a new Git repository.
- **git clone <repository>**: Clones a remote Git repository to your local machine.
- **git add <file>**: Adds a file to the staging area for the next commit.
- **git commit -m "<message>"**: Commits changes to the repository with a message describing the changes made.
- **git status**: Displays the status of the repository, including which files have been modified or added.
- **git log**: Displays a history of commits made to the repository.
- **git push**: Pushes changes from your local repository to a remote repository.
- **git pull**: Pulls changes from a remote repository to your local repository.
- **git branch**: Lists all branches in the repository.
- **git checkout <branch>**: Switches to the specified branch.

Advanced Commands:

- **git add .**: Add all files to the staging area for the next commit. Useful when you have a lot of changes and don't want to add one file at a time.
- **git merge <branch>**: Merges the specified branch into the current branch.
- **git rebase <branch>**: Rebase the current branch on top of the specified branch.
- **git stash**: Stashes changes made to the repository without committing them.
- **git cherry-pick <commit>**: Applies the changes made in the specified commit to the current branch.
- **git reset**: Resets the repository to a previous commit, discarding any changes made since that commit.

Terminologies:

- **Branch:** A copy of your repository that you can make changes to without affecting the original version.
- **Commit:** A snapshot of your files at a particular point in time.
- **Merge:** Combining two or more branches into a single branch.
- **Pull:** Downloading changes made by others and merging them with your local copy of the repository.
- **Push:** Sending your changes to a central server (like GitHub) so others can access them.

- **Rebase:** Moving the current branch to a new base commit and replaying all of the changes made on the current branch on top of the new base.
- **Remote Repository:** A Git repository hosted on a central server (like GitHub) that can be accessed by others.
- **Repository:** A centralized location where you store all the files and code related to a particular project.
- **Staging Area:** An intermediate step between making changes to your files and committing those changes to the repository. Files added to the staging area are ready to be committed.
- **Stash:** Saving changes made to the repository without committing them, allowing you to switch to a different branch or work on a different task without losing your changes.

Basic Git Workflow

1. **Initialize a Git repository:** Start by creating a new Git repository using `git init` command. This creates a new repository and sets up the necessary files for Git to track changes.
2. **Make changes to your code:** Make changes to your code and files as needed.
3. **Stage changes:** Use the `git add` command to stage changes for the next commit. This adds modified files to the staging area, ready to be committed.
4. **Commit changes:** Use the `git commit` command to commit your changes to the repository with a descriptive message.
5. **Push changes:** Use the `git push` command to send your local changes to a remote repository, like GitHub or Bitbucket. This is essential when working on a team or collaborating with others on a project.
6. **Pull changes:** Use the `git pull` command to download changes made by others from the remote repository and merge them with your local copy of the repository.
7. **Create and switch branches:** Use the `git branch` and `git checkout` commands to create and switch between branches. Branches allow you to create a copy of your repository that you can make changes to without affecting the original version.