Git Starter Doc

Quick Notes

In this document we’re going to be using the command line for interfacing with git (instead of using a GUI). This should help you focus on learning the basics and not having to learn a whole GUI in the process as well. In this section we’re going to cover the minimal command line commands we’re going to use and what they do.

1. **CD:** Change Directory command is used to move directories while using the command line.

```
$ cd "Directory-name"
```

2. **Touch:** The **touch command** is used to create a new file in the current directory.

```
$ touch "Directory-name"
```

3. **DIR:** The **dir command** is used to display all contents of the current directory.

```
$ dir
```

Overview

Git is a DVCS (Distributed version control system) which is used in both single and team oriented projects, it is generally used for tracking changes to files. It is used most commonly for source code management in the software development lifecycle. Because of its distributed version control tool nature git is commonly used by many developers so they can all work on the same project simultaneously.

Starting

Firstly we will need to install git if you don’t already have it. However, some machines come with git pre installed. Before we go through the installation process go ahead and check if you already have git installed on this system. In order to check if git is already installed on the system open command line and run the following command (For windows users).

```
git -- version
```

If git is already installed it should return `git version x.xx.x` if you already have git installed, otherwise you need to download git.

Download git here: Git - Downloads

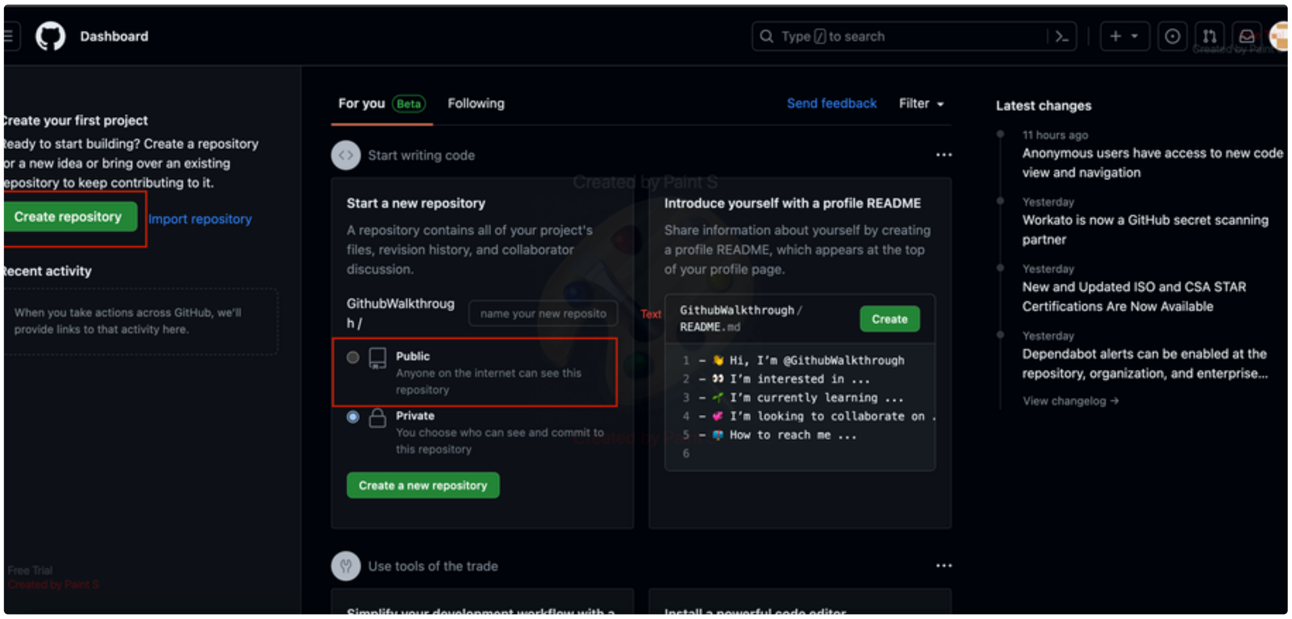
Git Repository Hosting Service

Now you will need to choose where you want to host your remote repository. There are several options for this purpose, GitHub, Azure DevOps, and Bitbucket server. However, for this walkthrough we will be using GitHub. Let's go through some steps to set up your remote repository hosting service (Github). **Note that on GitHub is that all accounts/Repos are public by default, meaning that anyone can see your code/clone your Repositories.** You can get around this by paying in order to have private repositories.

1. If you already have a GitHub account you can skip this section, but if you don't you will need to go to the link below.

Access GitHub here: GitHub: Let's build from here

2. You will want to follow the instructions on the website in order to sign up and create a Github account. In order to create an account you will need an email, password and username.
3. In order to create your first repository you will need to click on the **"Create Repository"** button and then select the **"Public"** radio button, as shown below.



Dashboard

Type / to search

Create your first project

Ready to start building? Create a repository for a new idea or bring over an existing repository to keep contributing to it.

Create repository | Import repository

Recent activity

When you take actions across GitHub, we'll provide links to that activity here.

For you Beta | Following | Send feedback | Filter

Start writing code

Start a new repository

A repository contains all of your project's files, revision history, and collaborator discussion.

GithubWalkthrough / name your new reposito

Public
Anyone on the internet can see this repository

Private
You choose who can see and commit to this repository

Create a new repository

Introduce yourself with a profile README

Share information about yourself by creating a profile README, which appears at the top of your profile page.

GithubWalkthrough / README.md Create

```
- 👋 Hi, I'm @GithubWalkthrough
- 👀 I'm interested in ...
- 🌱 I'm currently learning ...
- 💞 I'm looking to collaborate on .
- 📫 How to reach me ...
```

Use tools of the trade

Simplify your development workflow with a

Install a powerful code editor

Latest changes

11 hours ago
Anonymous users have access to new code view and navigation

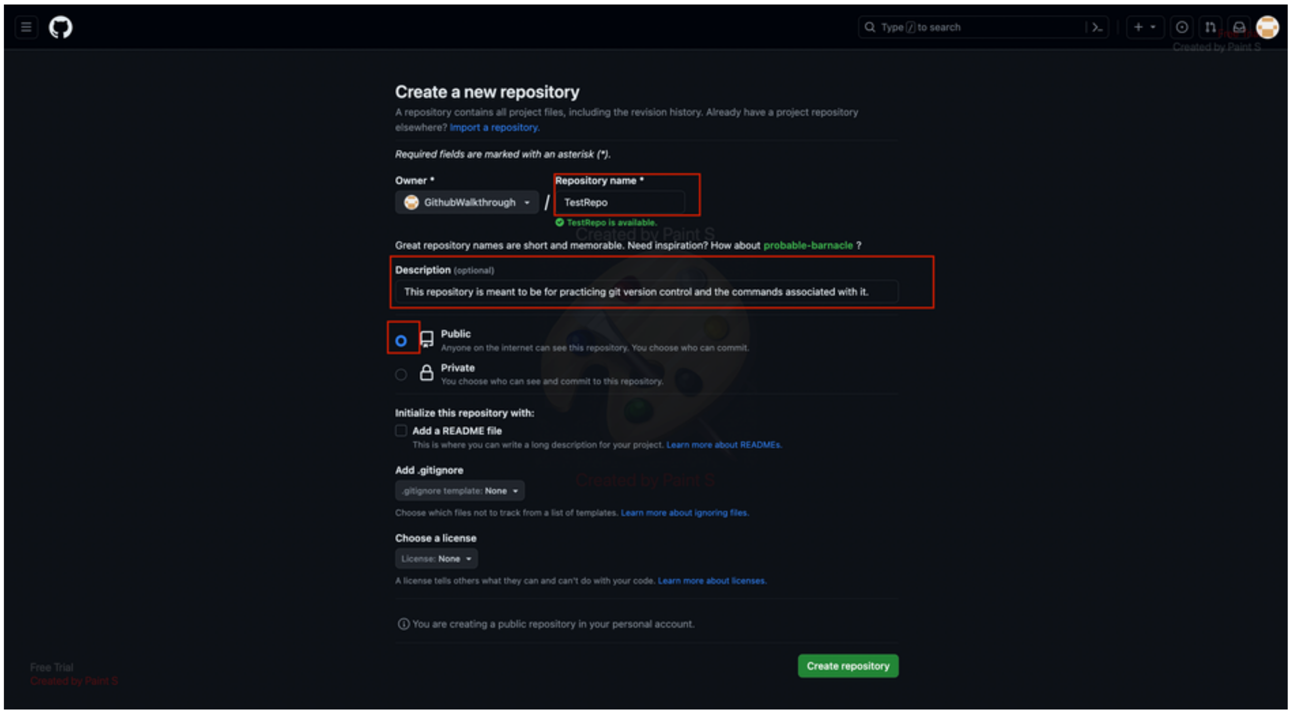
Yesterday
Workato is now a GitHub secret scanning partner

Yesterday
New and Updated ISO and CSA STAR Certifications Are Now Available

Yesterday
Dependabot alerts can be enabled at the repository, organization, and enterprise...

View changelog →

4. Now you will need to pick a name for your repository, and a meaningful description of what it consists of. For example, I chose **"TestRepo"** as I'm setting this up strictly for this walkthrough. The description describes exactly that. As for the README file and git ignore we can skip these for now but we will touch on them further in this walkthrough. Finally click on the **"Create Repository"** button, this will create your repository which will be hosted on GitHub.


Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Required fields are marked with an asterisk (*).

Owner * GithubWalkthrough / Repository name * TestRepo

TestRepo is available.

Great repository names are short and memorable. Need inspiration? How about probable-barnacle ?

Description (optional): This repository is meant to be for practicing git version control and the commands associated with it.

[✓ 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:

Add a README file — This is where you can write a long description for your project. Learn more about READMEs.

Add .gitignore — .gitignore template: None

Choose which files not to track from a list of templates. Learn more about ignoring files.

Choose a license — License: None

A license tells others what they can and can't do with your code. Learn more about licenses.

You are creating a public repository in your personal account.

GitHub Authentication

Before we set up the local version of this repository we will want to do a few housekeeping tasks in order to make our lives easier further down the line. For this section of the walk through we will be using the command prompt on windows (terminal for mac).

1. First, we will want to add our GitHub Username and email to git through the command prompt. In order to do this you will need to run the following commands in command line.

```
$ git config --global user.name "Your name here"
$ git config --global user.email "your_email@example.com"
```

(Don't type the $; that just indicated that you're doing this at the command line).

Now you don't need to type in your email and username when you want to commit/push changes to your remote repository. However if we stop right now we will still need to type in our password every time we want to push changes for our local machine.

2. One way to get around this is to set up an SSH key private/public key pair, instead of using your password for authentication every time you try to make changes you just need to share your public key with GitHub. If the public key you shared with GitHub matches your private key then you will be granted passwordless authentication. SSH keys work with something called cryptography which essentially ensures that no one can reverse engineer your private key from the shared public one.
3. In order to set up these keys you will need to open command line and enter the following command:

```
$ ssh-keygen -t rsa -C "your_email@example.com"
```

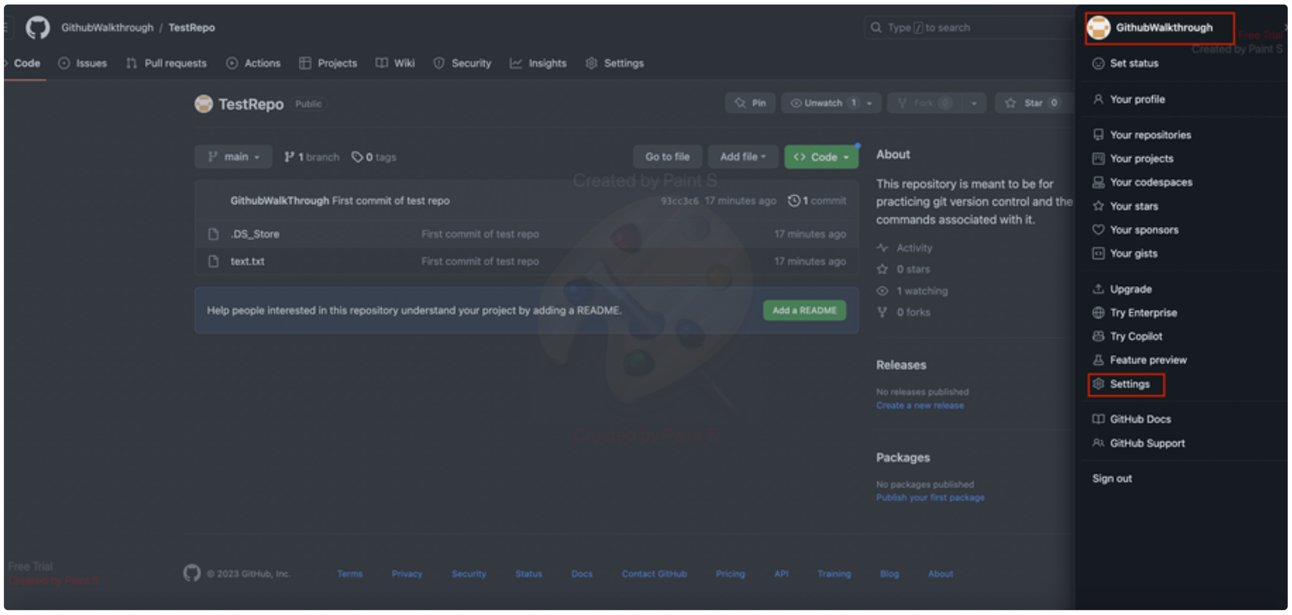
Note: In order to copy the public key from command line, cd to the directory your public key is stored:

```
$ cd .ssh
```

the following command will copy the contents to your clipboard.

```
$ clip < id_rsa.pub
```

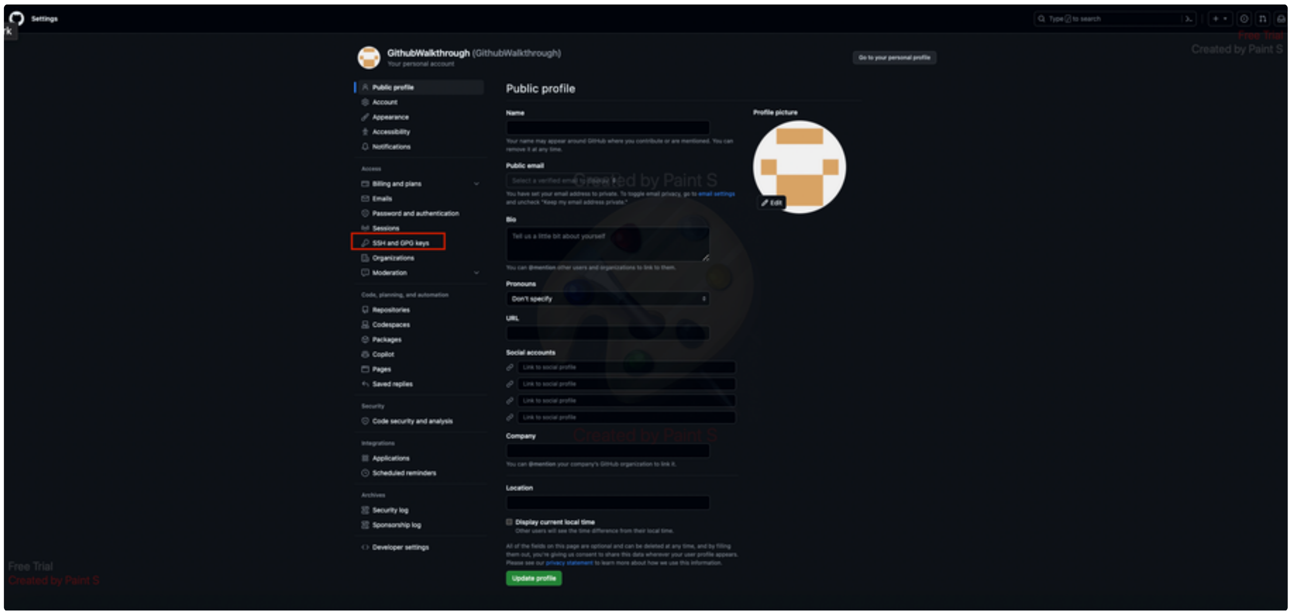
4. After copying your public key you will want to go to GitHub again and follow these steps:
   a. Click on your GitHub profile picture → Settings



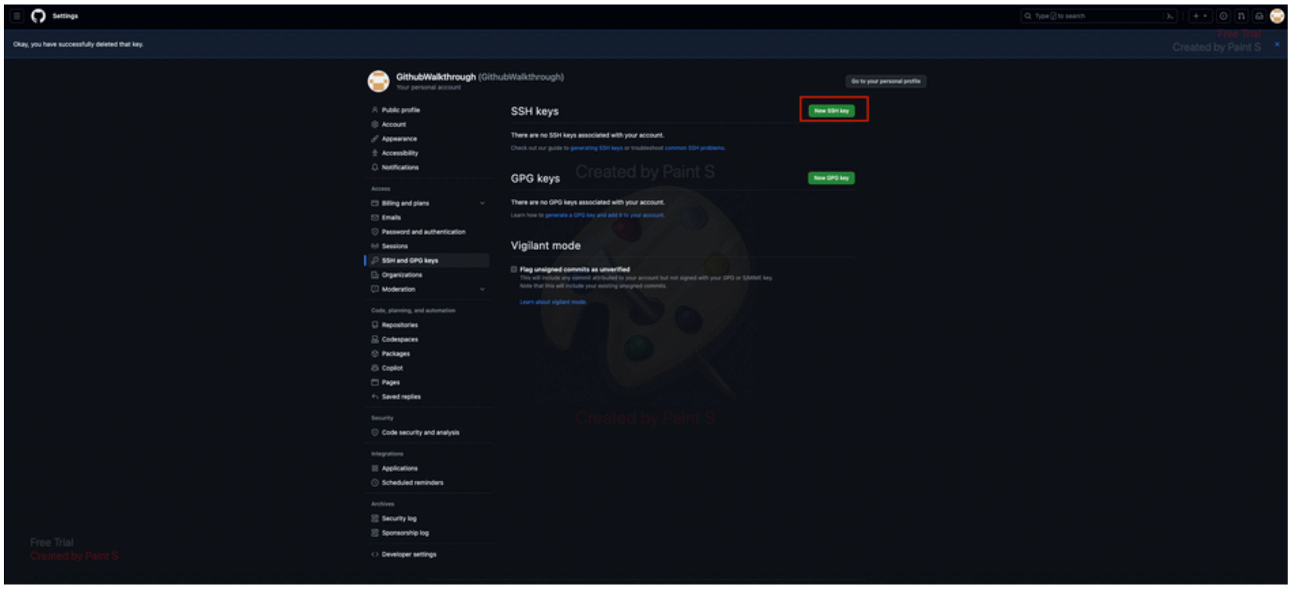
This repository is meant to be for practicing git version control and the commands associated with it.

Help people interested in this repository understand your project by adding a README.

   b. Click on the “SSH and GPG Keys” setting



   c. Click on “New SSH Key”



Okay, you have successfully deleted that key.

SSH keys

There are no SSH keys associated with your account.

Check out our guide to generating SSH keys or troubleshoot common SSH problems.

GPG keys

There are no GPG keys associated with your account.

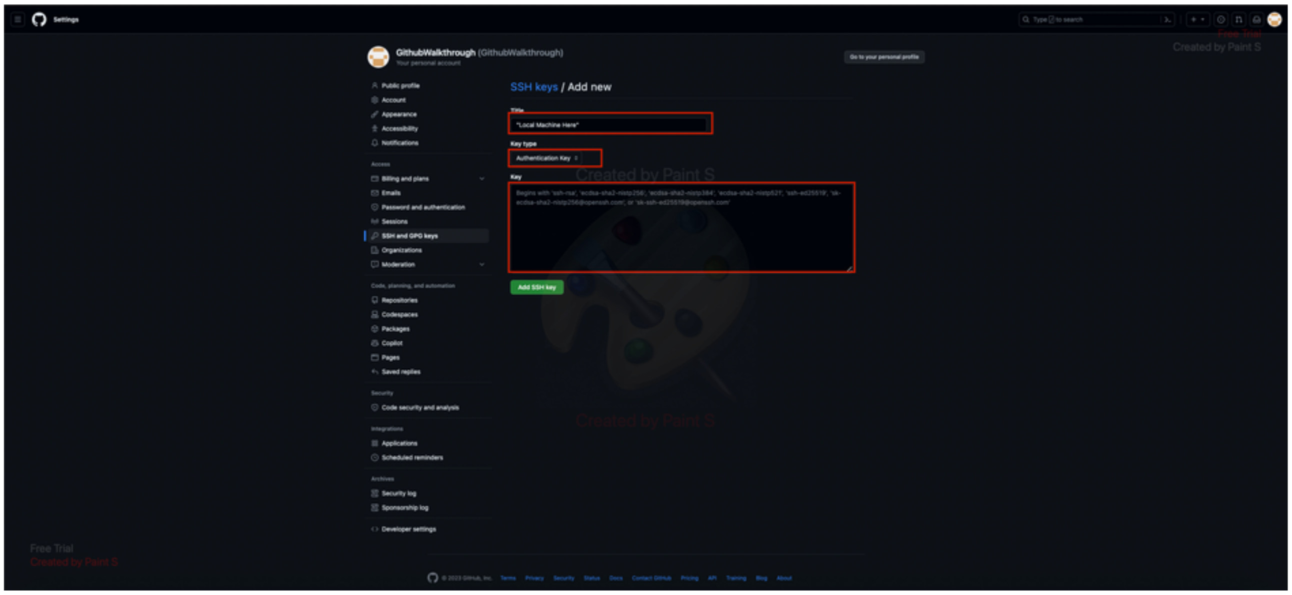
Learn how to generate a GPG key and add it to your account.

Vigilant mode

Flag unsigned commits as unverified
This will include any commit attributed to your account but not signed with your GPG or S/MIME key.
Note that this will include your existing unsigned commits.

Learn about vigilant mode.

d. Make sure to enter a meaningful name in the title, it's good practice to set it as your local machine i.e. **"Riley's Mac"**. Make sure to have **"Authentication Key"** selected, and then paste the public SSH Key that we copied a little bit ago. Finally, click on the **"Add SSH Key"** button and that's it. You now have SSH Key authentication set up. You can push and pull the latest code from your remote repository without having to enter your username or password ever again.



SSH keys / Add new

Title: "Local Machine Here"

Key type: Authentication Key

Key: Begins with 'ssh-rsa', 'ecdsa-sha2-nistp256', 'ecdsa-sha2-nistp384', 'ecdsa-sha2-nistp521', 'ssh-ed25519', 'sk-ecdsa-sha2-nistp256@openssh.com', or 'sk-ssh-ed25519@openssh.com'

The Repository

In order to start working with our Remote repository we created on GitHub we will need to first create a copy on our local machine. The cool thing about git is when you clone a repository it copies the whole repository which means if something happens to your remote copy you have an exact backup on your local machine.

1. In order to clone our remote repository we will first want to move to a directory where we want to store our repository locally, then we will need to run the following command on command line:

```
$ git clone "git@github.com:YourUser_name/YourRepository_name.git"
```

a. Running this will create a copy of our remote repository in the directory we specified on our local machine.

2. In order to make sure your repo is up to date you should run the following command:

```
$ git pull
```

b. The `git pull` command is used to fetch and download content from a remote repository and immediately update the local repository to match that content. Merging remote upstream changes into your local repository is a common task in Git-based collaboration work flows. The `git pull` command is actually a combination of two other commands, `git fetch` followed by `git merge`. In the first stage of operation `git pull` will execute a `git fetch` scoped to the local branch that `HEAD` is pointed at. Once the content is downloaded, `git pull` will enter a merge workflow. A new merge commit will be-created and `HEAD` updated to point at the new commit.

Common Commands

In this section we're going to discuss some common git commands that everyone should know and how to best use them with some examples.

1. **Git init**, command allows us to create a new local repository when starting or even at any point in a project on our local machine. In order to use this command move to the correct directory where your project is located and run the following command. This will add a hidden subdirectory ".git" which will allow git to track our project for any and all changes. (You don't need to specify a repo name when running this command).
   a.
   ```
   $ git init <repository name>
   ```
2. **Git clone** (which we previously touched) creates a local working copy of the remote repository.
   a.
   ```
   $ git clone <git-repo-url>
   ```
3. **Git branch**, this command lets us add a new branch locally, see all existing branches or even delete a branch.
   a. The following command creates a new branch locally
      i.
      ```
      $ git branch <branch-name>
      ```
   b. The following command lists all remote and local branches
      i.
      ```
      $ git branch -a
      ```
   c. The following command will delete a branch
      i.
      ```
      $ git branch -d <branch-name>
      ```
4. **Git checkout**, this command allows you to switch to an existing branch or create a new branch and switch to it. Before executing this command you need to make sure that all changes are either committed or stashed.
   a. Switching to a new branch
      i.
      ```
      $ git checkout <branch-name>
      ```
   b. create and switch to a new branch
      i.
      ```
      $ git checkout -b <new-branch-name>
      ```
5. **Git add**, this command adds your changes to the staging area where you can compare them to your local and remote copy. Before committing a new modified file, it should be added to the staging area using the git add command.
   a. Adding specific files
      i.
      ```
      $ git add <file-name-1> <file-name-2>
      ```
   b. Adding all new, modified, and deleted files:

      i.
      ```
      $ git add -A
      ```
   c. Adding modified and deleted files:
      i.
      ```
      $ git add -u
      ```

6. **git commit**, this command saves the changes in your local repository. Along with this you need to include a descriptive message. this message helps other developers understand what was changed.
   a. Committing any files added with the git add command and files that have been changed since then as well.
      i.
      ```
      $ git commit -a
      ```
   b. Commit files with a message
      i.
      ```
      $ git commit -m "<commit-message>"
      ```
   c. You can replace the two previous commands with the following single command
      i.
      ```
      $ git commit -am "<commit-message>"
      ```
   d. Modify the last commit with the latest changes as a new commit
      i.
      ```
      $ git commit --amend -m "<commit-message>"
      ```
7. **Git Push**, this command pushes the committed file changes from the local repository to the remote repository, it will also create a named branch in the. remote repository if it does not exist.
   a.
      ```
      $ git push or $git push <remote> <branch-name>
      ```
   b. If you created a new branch on your local repository, then you will need to push the branch with the following command:

```
$ git push --set-upstream <remote> <branch-name> or $ git push -u origin <branch-name>
```

8. **Git Pull**, fetches the latest pushed changes from the remote server and copies them into your local repository so you can get the latest updates from your teammates.
   a.
      ```
      $ git pull or $ git pull <remote> or $ git pull <remote> <branch-name>
      ```
9. **Git Merge**, this command merges your branch with the parent branch. The parent branch can either be a development or master branch depending on your workflow. This command will also create a new commit if there are no conflicts. Before running this command you should be on the same branch locally that you wish to update on your remote repository side.
   a. First we need to switch the branch that we want to merge into:
      i.
      ```
      $ git checkout <branch>
      ```
   b. Secondly we need to make sure we have the latest updates by running a Git Fetch command:
      i.
      ```
      $git pull or $git fetch
      ```
   c. Lastly we will finally merge:
      i.
      ```
      $ git merge <branch-name>
      ```
10. **Git Status**, this command provides you with an overview of the current status of your repository
   a.
      ```
      $ git status
      ```
      The Git status command returns the following information about your branch:
      i. Your current branch name
      ii. If your branch is up to date or not
      iii. How many commits your local branch is behind from the remote copy
      iv. changes that are staged and to be committed

v. changes that are not staged for commit

vi. un-tracked files

Additional Resources

In this section we’re going to cover some useful information about git which can save you from some headaches down the road. we’re going to cover, stashing, popping, reverting commits, git ignore, and git readme file basics.

1. Git ReadMe file is meant to make it easier for someone to understand and engage with the project. It should consist of a project name, table of contents, Installation instructions, what its purpose is, who the contributors are, and any licensing.

More info on Readme: About READMEs - GitHub Docs

2. Stashing/Popping is used in tandem when the contents of your current working directory aren’t meant for the current branch you’re on.

More info on Stash/Pop:

Git Stash Pop Command - Scaler Topics

Example of stashing changes from one branch and stash pop them into a different branch.

3. Git Ignore is used when you intentionally don’t want git to track certain files.

More info on GitIgnore: Git - gitignore Documentation

Preconfigured GitIgnore Files: gitignore.io