1. Preface

This tutorial is a slimmed-down version of a git tutorial. If you need a more in-depth tutorial, google it.

Please read every line carefully, and don't skip any steps.

1.1. How to Read this Tutorial

- Optional steps are indicated with (Optional) at the front of the header line.
- User input is indicated with <angle brackets>. For example, if I wanted you to input your name, I would write:

```
some command "<your name>"
```

- Note that the quotes are mandatory wherever you see them.
- Coloured text should be inputted into your terminal exactly as shown, except for the specified inputs.

1.2. Notes for Windows Users

- Use **PowerShell**
- Forward slashes and backslashes are interchangeable

2. Git, from Scratch

2.1. Install Git on:

2.1.1. MacOS:

Some versions of MacOS come with git preinstalled. Check with:

```
git --version
```

If you get an error, you need to install git. You can do this with:

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/
install.sh)"
brew install git
```

2.1.2. Windows:

You can install git for Windows from here: https://git-scm.com/download/win

2.1.3. Linux:

If you're on Linux, you already have git installed. Check with:

```
git --version
```

If you get an error, you need to install git. Since you're on Linux, you can figure this out yourself. Hint: Google "install git on <distro> linux"

2.2. Setting up Git, locally

```
git config --global user.name "<your name>"
git config --global user.email "<your email>"
ssh-keygen -t ed25519 -C "<your email>"
```

3. GitHub

3.1. Creating a GitHub Account

Go to https://github.com/join. Use your **personal email address**.

Fill in the form, and you're done. They'll probably send you an email asking you for verification or something.

3.2. Creating a GitHub Repository

3.2.1. Side Track: Create a Directory for your Code

The first thing you'll need to do is standardize where you store your code. You may choose any directory on your computer, but I recommend you create it in your Documents directory under `~/Documents/code`.

3.2.1.1. MacOS:

```
cd
mkdir -p Documents/code
```

3.2.1.2. Windows:

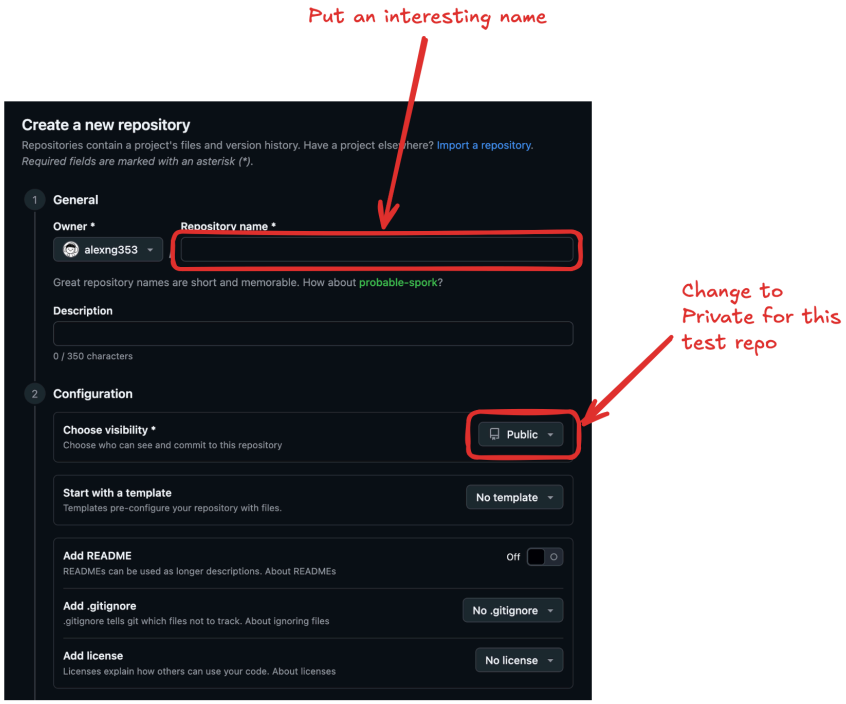
```
cd
mkdir Documents\code
```

3.2.1.3. Linux:

```
cd
mkdir -p Documents/code
```

3.2.2. Creating the Repository

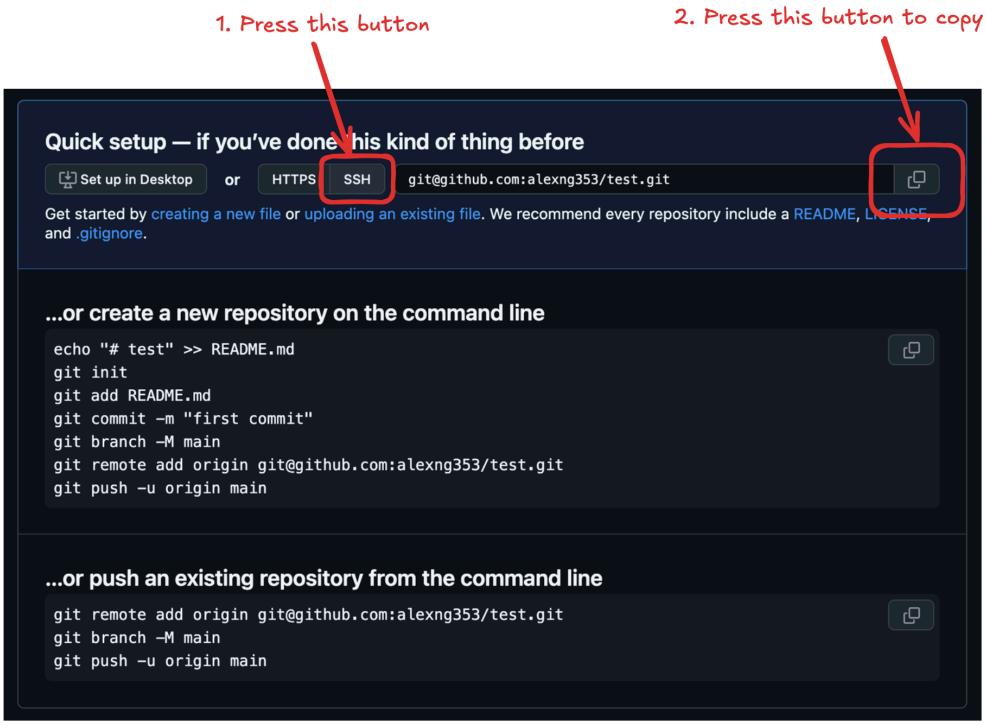
Go to https://github.com/new. You'll see a form like this:



Fill it in, then click "Create repository". Hint: It's at the bottom of the page, and it's **very green**.

3.2.3. Setting up the Repository on your Computer

Copy the URL of your repository. It'll look something like this:



Next, go to your terminal and run:

```
cd ~/Documents/code # or wherever you want to store your code
git clone <URL>
ls # this should show you all the files in your current directory
cd <repo name>
```

3.2.4. Making your first commit

Now that you have the repository set up, you can make your first commit. This part is kind of important, so I'll explain it in detail.

First, make sure you're in the directory of your repository. You can do this by running

```
git rev-parse --is-inside-work-tree
```

If you're not in a repository, you'll get an error. If it doesn't say error, you're in a repository.

Next, you'll create a file in your repository.

```
echo "# test" >> README.md
```

You can check if it's there by running:

```
ls
```

Now, you can add your file to the staging area. This indicates to git that you want to commit this file **at the current revision**: if you make more changes to this file after you've added it to the staging area, **you'll need to add it again**.

```
git status
git add README.md
git status
```

```
➜  test git:(main) echo "# test" >> README.md
➜  test git:(main) ✗ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

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

nothing added to commit but untracked files present (use "git add" to track)
➜  test git:(main) ✗ git add README.md
➜  test git:(main) ✗ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   README.md
```

Notice how the first `git status` showed README.md was red, which indicates that it's **not staged**. The second `git status` showed it was green, which indicates that it's **staged**.

You then need to commit your changes. This is like saving your changes to git. The syntax for the commit command is `git commit -m "<message>"`. The `-m` flag indicates that you're going to give a message to describe the changes you made.

```
git commit -m "first commit"
```

And you can push it to GitHub with:

```
git push
```

3.2.5. Branches

Branches are how you work with multiple people on the same project. You can create a branch (and switch to it at the same time) with:

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

The checkout command is how you switch between branches. You can switch back to the main branch with:

```
git checkout main
```

You can list all the branches with:

```
git branch -a
```

3.2.6. Recap

3.2.6.1. Important Commands

- `git status`: shows the status of your repository
- `git add`: adds a file to the staging area
- `git commit`: commits the changes in the staging area
- `git push`: pushes your changes to GitHub
- `git pull`: pulls changes from GitHub
- `git log`: shows the commit history

- `git diff`: shows the changes between commits
- `git checkout <branch>`: switches to a branch
- `git branch -a`: lists all branches

3.2.6.2. Simple Git Workflow

Every time you want to make changes, especially when you're working with a team, you can follow this workflow:

1. `git checkout main` — Switch to main branch
2. `git pull` — Pull other's changes
3. `git checkout -b <branch name>` — Create a new branch
4. Make changes
5. `git add -A` — Add all changes to staging area
6. `git commit -m "<message>"` — Commit changes
7. `git push` — Push changes to GitHub