



**AIMS** | African Institute for  
Mathematical Sciences  
NEXT EINSTEIN INITIATIVE

# An Introduction to Git and Github



# The what and the why

Version Control, Git, and GitHub

# What is version control, git, and GitHub?

**Version control** is a method for tracking changes to source code over time in a way that facilitates collaboration.

**Git** is an open source system for **version control**.

**GitHub** is a platform that hosts **git** repositories and provides other features and services built around **git**.

# Why use version control, git, and GitHub?

## Tracking changes

Since version control tracks all changes to your codebase, it allows you to easily revert your code to a previous state when something breaks.


## Collaboration

Version control is built for collaboration. It provides a method for multiple people to work on the same files in parallel and merge changes together when ready.

# Why use version control, git, and GitHub?

Git is the most popular tool for version control.


It's free, open source, and scales well from small personal projects to the largest code bases in the world.

 GitHub has become the most popular platform for sharing code and collaborating on open source projects.

# Getting started

Installing Git, using GitHub

# Getting started

**Steps 1:** If you don't already have a GitHub account, head over to [github.com/signup](https://github.com/signup) 

**Step 2:** Then install Git 

## Mac OS X

Ships with xcode.

Otherwise install with homebrew:

```
$ brew install git
```

## Linux

Available through most default package managers.

On Debian/Ubuntu, for example:

```
$ apt-get install git
```

## Windows

Head over to [git-scm.com/download/win](https://git-scm.com/download/win) and run the installer.

# Configure Git

After installing, run the following commands to tell Git who you are:

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

Optionally add your local SSH key to GitHub so you don't have to enter your username and password from the command line all the time: [github.com/settings/ssh/new](https://github.com/settings/ssh/new)

If you haven't already generated an SSH key, run the following:

```
$ ssh-keygen -t ed25519 -C "your_email@example.com"
```

Then copy the contents of the file in the `.ssh` directory that ends with `.pub` to GitHub



# Git and GitHub basics

The basic commands you need to know

# Creating a new repository

In git, a project / code base is called a **repository**. To create a new repository, head over to [github.com/new](https://github.com/new), fill out the details and make sure to check these boxes:

☒ **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: Python ▼

Then hit  

Now **clone** your repository from the command line:

```
$ git clone https://github.com/my-username/my-project.git && cd my-project
```

Or, if you setup SSH following the previous slide:

```
$ git clone git@github.com:my-username/my-project.git && cd my-project
```

**Live demo**

# Git workflow

**Step 1:** *Pull* to sync your local clone with the remote repository on GitHub

```
$ git pull
```

**Step 2:** Make changes, i.e. go about adding, removing, or editing files in the directory of your git repo

**Step 3:** *Stage* changes

```
$ git add -A
```

**Step 4:** *Commit* changes

```
$ git commit -m "Meaningful commit message"
```

**Step 5:** *Push* changes

```
$ git push
```

# More on *staging*, *committing*, and *pushing*

Git uses a two-step commit:

- 1 First you have to add a snapshot of the files you want to commit to the staging area with `git add`. For example:

```
$ git add -A ← stage all files that have been added, removed, or changed
```

```
$ git add -u ← stage only existing files that have changed
```

```
$ git add README.md ← stage only the README.md file
```

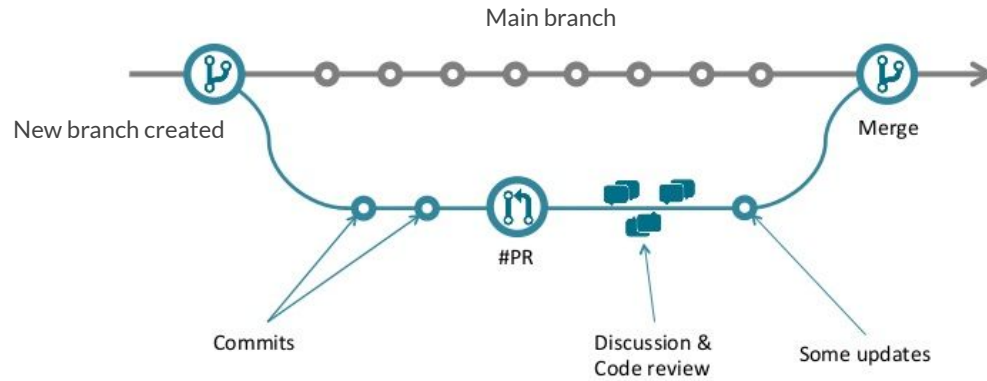
- 2 Then you commit them with `git commit`, which stores the snapshot permanently to your local repository.

Finally,

- 3 you ship 📦 that commit off to the remote repository with `git push`.

**Live demo**

# Collaborate git workflow



# Collaborative git workflow

**Step 1:** *Pull changes* - `$ git pull`

**Step 2:** Create a new branch

`$ git checkout -b patch-1`

**Step 3:** Make changes...

**Step 4:** *Stage changes* - `$ git add -A`

**Step 5:** *Commit changes* - `$ git commit -m "Meaningful commit message"`

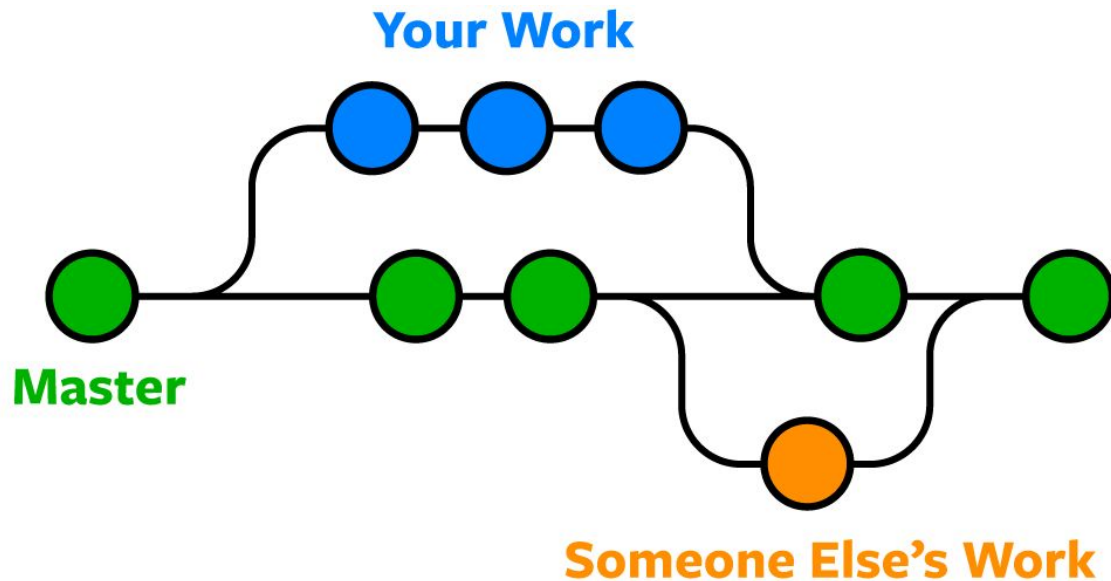
**Step 6:** *Push changes* - `$ git push -u origin patch-1`

**Step 7:** When you're satisfied with your branch, open a *pull request* on GitHub

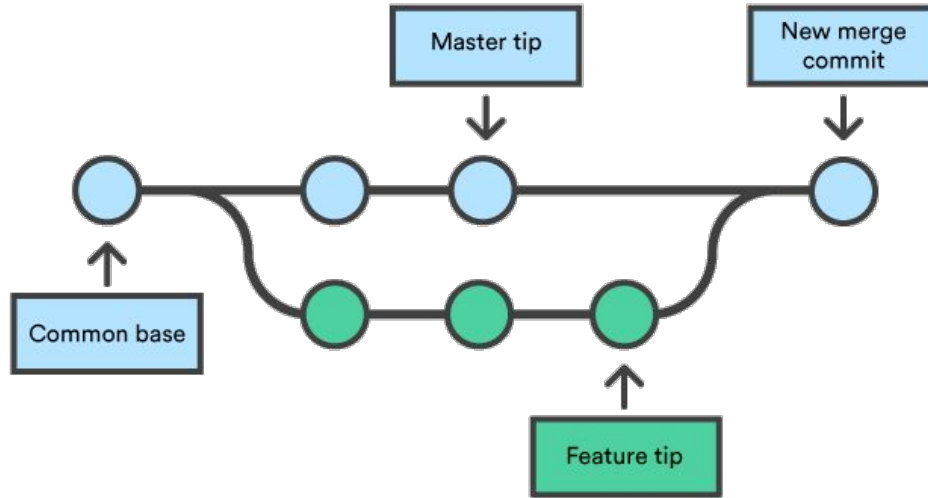


**Live demo**

# Merge Conflicts



# Merge Conflicts



# Merge Conflicts

**Step 1:** Merge new branch on master - `$ git merge patch-1`

**Step 2:** Merge conflict - Find offending files

```
$ CONFLICT(content): Merge conflict in test.txt
```

**Step 3:** Fix merge conflicts - Correct files, keep what is necessary

**Step 5:** Commit changes - `$ git commit -m "Fixed merge conflict in test.txt"`

**Step 6:** Push changes - `$ git push`

**Live demo**

# Best Practices

## Collaboration

Divide work between collaborators.

Each person is responsible for their own file or part of the project.

Create a new branch for each change/addition. Always have a working main.

## Communication

Discuss code structure with all collaborators before starting.

Work together when resolving merge conflicts.

Notify others if you're editing their code.

