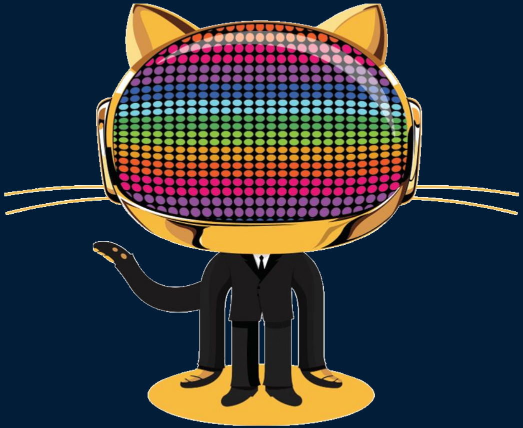
WORKSHOP STRUCTURE

- WHAT IS GIT AND GITHUB
- INSTALLATION AND SETUP
- CREATING A PULL REQUEST
- BRANCHING
- ADVANCE GIT COMMANDS
- MORE ABOUT GITHUB



INTRODUCTION

Git is a Version Control System. Now, what's a **VCS**?

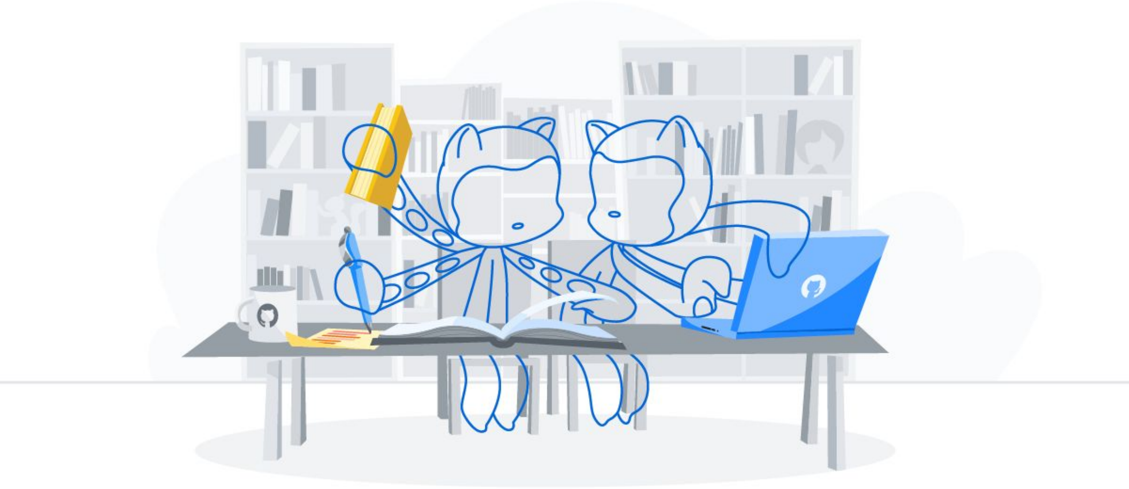
It is a system that tracks changes made in any set of files, usually used for coordinating work among programmers.

Git shows what changes were made, when and by whom to a project.

INTRODUCTION

GitHub is an online software development assistance and version control service.

It is a **cloud-based** hosting service that lets you manage Git repositories.



INTRODUCTION

- We will start this module by downloading **Git Bash,** from the official website (https://git-scm.com/), and follow along for installation.
- Launch the **Git Bash** setup. The very first page contains the important info regarding the *licenses* and *term & conditions*. After reading it click on the next button.
- The next page consists of the components that needs to be installed. Please confirm that the check box regarding *Additional icons* has been marked for it to create a Desktop shortcut.
- The next page contain the info about the **Default editor** (eg. VS Code, Sublime Text, Vim etc.)that we want to set for **Git** and in which you are comfortable.
- The next page consist of the info about **adjusting the name of the initial branch in new repository**. Let it to be the default option i.e "let git decide". Click on next.

INSTALLATION

- After that it contains the info about the adjustment of **Path environment**. Keep this to be the recommended option (i.e Git from the command line and also from 3rd-party software). Click on next.

- In the next page choose the **Use the OpenSSL Library** option(default option) and click on next.
- In the next page choose the option (**Checkout Windows-style, commit Unix-style line endings**) and click on next.
- Again click on next after checking ***Use MinTTY*** option. For all further pages stick with the default options, hence click on next till you reach *Installation* option. Get it installed and click on *Finish.*

--- We have Successfully Installed **GIT BASH---**

INSTALLATION

Git Installation on MacOS

- Go to Git official site and head over to downloads section for macOS
- Either use the homebrew method or the package installer →

1. HomeBrew method -
   - ❑ Go to homebrew official site and copy the bash command and paste it in your terminal and enter to install brew for mac
   - ❑ Now run the command ‘brew install git’ to install git for your mac

2. Package Installer Method
   - ❑ Download the package installer for mac from the Git official site
   - ❑ Run it like a usual application installer on MacOS

Now , you’re all set to push/pull, commit,fork etc with the terminal on mac

- Launch the installed **Git Bash** application and type ***git config*** and we will get all the functions related to git configuration will appear. We can read through them.
- Now type **git config --global user.name “Your UserName”** and then press enter. This would configure your Username.
- Now type **git config --global user.email “Your email-id”** and then press enter. Now this would configure your email-id.
- Now if we want to check whether our name and email-id has been configured successfully or not, we can simply type **git config user.name** and **git config user.email** to check our configured name and email respectively.
- If we want to see all the details, type **git config --list** and all the details would be available on the screen. After that if we want to get out of that simply press **q**-button and that’s all.

----- We are now ready to move ahead with other functions -----

CONFIGURATION

BASIC BASH COMMANDS AND TOOLS

- cd command: Allows you to traverse to other directories.
- pwd: displays present working directory
- mkdir command: Allows you to make directory.
- touch command: Allows you to make file.
- nano: A Command Line Text Editor for editing files in bash. (You can also use vim, but for beginners nano is user friendly)
- cat: displays the contents of the file

Further Commands will be told in the tutorial ahead.

Initializing git repo

- Inside your Project Folder type: “git init” to initialize git repository.
- It will make a .git folder inside your Projects folder.
- .git folder is where, all your tracking data goes.
- To check the current status of git repo type: “git status”

```
richardkalehoff new-git-project
$ 
```

- The Staging Area is when git starts tracking and saving changes that occur in files.
- You tell git that I want to track these specific files, then git says okay and moves them from you Working Tree to the Staging Area and says “Cool, I know about this file in its entirety.”
- Type: “git add <filename>” (For adding a single file to staging area)
- Type: “git add .” (For adding all the files which you changed)

  (Here . means the current directory)

STAGING AREA

- Now, in order to insure if the correct file(s) has been staged, we use the command:

  “git status”

- This command shows you which files are being tracked, which have been modified but are not being tracked and which are not being tracked.

CHECKING STATUS

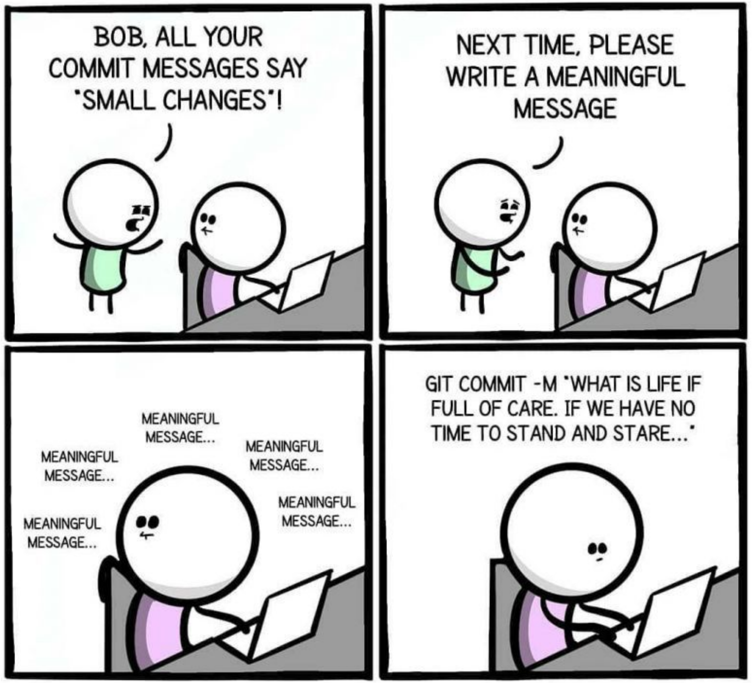
- “git show” is a command line utility that is used to view expanded details on Git objects such as blobs, trees, tags, and commits
- For example:

git show

- There are two ways of viewing history logs:
- “git log” : displays all of the commits in the repository’s history.
- “git reflog” : displays the record when the tips of branches and other references were updated.

Viewing Log

- Now comes the committing part.
- If all things are good you may want commit changes to release the new version of your project.
- To do this we will use: “git commit” command
- Type: “git commit -m <message>”
- Here take care of the message part as it should be concise, clear and short.

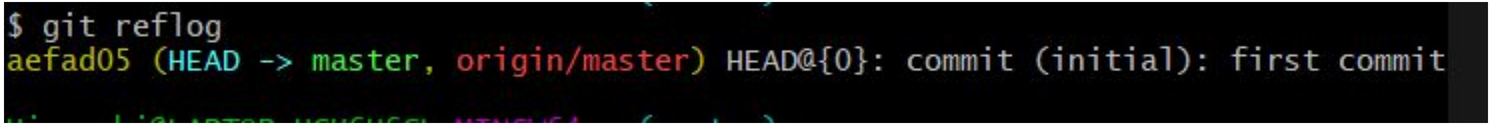

BOB, ALL YOUR COMMIT MESSAGES SAY "SMALL CHANGES"!
NEXT TIME, PLEASE WRITE A MEANINGFUL MESSAGE
MEANINGFUL MESSAGE...
MEANINGFUL MESSAGE...
MEANINGFUL MESSAGE...
MEANINGFUL MESSAGE...
MEANINGFUL MESSAGE...
GIT COMMIT -M "WHAT IS LIFE IF FULL OF CARE. IF WE HAVE NO TIME TO STAND AND STARE..."

COMMIT CHANGES

REVERTING BACK TO PREVIOUS VERSION

- Let's suppose you want to revert back to last version of the file as the current version is buggy or you made some serious mistake.
- So first type: "git diff HEAD"   (allows you to compare the file version in your working directory with the file version last committed in your remote repository)
- To move back to last version ,first check the commit id using the command:

  "git reflog"



```
$ git reflog
aefad05 (HEAD -> master, origin/master) HEAD@{0}: commit (initial): first commit
```

- And now to move back to last version type:

  “git revert <commit id>”

  And the file will revert back to the last version.

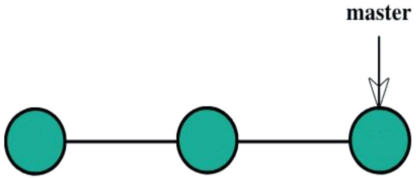
REVERTING BACK TO PREVIOUS VERSION

- Let's suppose you are developing a webapp which employs various APIs. For accessing those APIs you will surely be using some Private Access Keys.
- Let's say you kept all these API Keys in a file. You surely will not want git to track that file. Here's where .gitignore comes into play.
- .gitignore tells git about the files which it should not track.
- Then add the files name (with extension) which you don't want to get tracked. And git will not track those files.
- https://github.com/github/.gitignore (because some secrets shall remain secrets :))

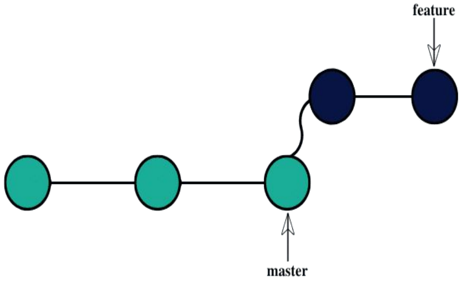
KEEPING YOUR SECRETS PRIVATE (.gitignore)

Why Branching? Let's see an example!

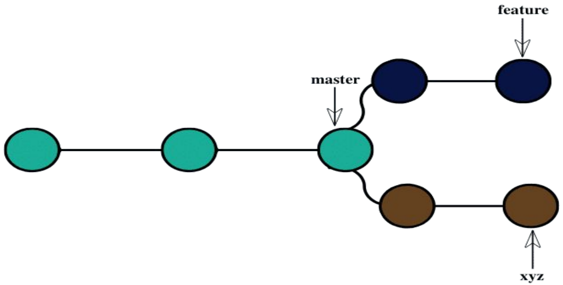
Developing a Project through branching:-

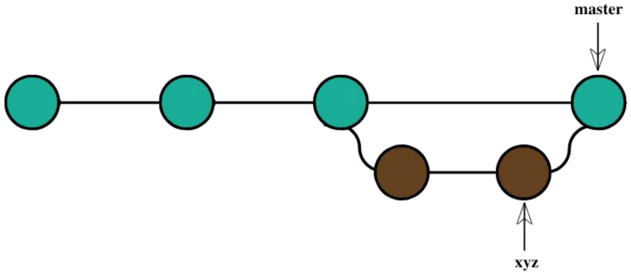


Suppose you are working on a project for a client After this you decide to develop a feature and create a New branch feature and start working.



Since you had some time left you decided to work on one more feature names "xyz". You now show it to client And he disproves the "feature" but allows the "xyz" . Now as you used branching you just need to remove the "feature" branch and merge the "xyz" branch into master.





Branches give you the freedom to independently work on different modules (*not necessarily though*) and merge the modules when you finish developing them. It might sound a cumbersome process, but git branches are swift to be created and destroyed. Just a simple command can perform these processes, and they are very cheap, considering the size they take. Branches in Git help the team, which are in different parts of the world, work independently on independent features that would ultimately combine to produce a great project. Moreover, the branches are very flexible. Using branches does not mean you are using them for different features.

BRANCHING

Operations in Git Branches

- The git branch Command
  - To list all branch name in the repository
  - Create a branch
    - git branch <branchname>
- The git checkout Command
  - To switch between different branches
    - git checkout <branchname>
- Delete a branch
  - git branch -d <branchname> (will show error if you are on that branch)
  - git branch -D <branchname> (force deletes the branch)

Some more Operations in Git Branches

- To clone a specific branch
  - `git clone -b <branchname> <repo_url>`
- The create + checkout Command
  - `git checkout  -b <branchname>`
- Push to a specific branch
  - `git push -u origin <branchname>`
- Merging two branches (Note : you should be on the branch in which you want to merge a specific branch)
  - `git merge <branchname>` (here branchname is the branch you want to merge)

Some more Operations in Git Branches

- To clone a specific branch

  -
    ```
    git clone -b <branchname> <repo_url>
    ```
- The create + checkout Command

  -
    ```
    git checkout  -b <branchname>
    ```
- Push to a specific branch

  -
    ```
    git push -u origin <branchname>
    ```
- Merging two branches (Note : you should be on the branch in which you want to merge a specific branch)

  -
    ```
    git merge <branchname>
    ```
    (here branchname is the branch you want to merge)

Reversing Branch Merge

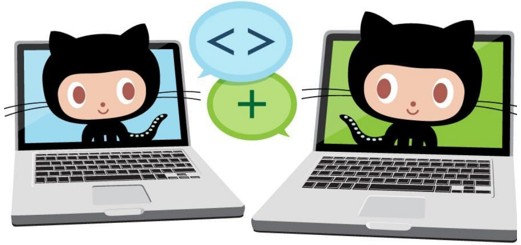
What is a Git merge conflict?

- A merge conflict is an event that occurs when Git is unable to automatically resolve differences in code between two commits.

- When all the changes in the code occur on different lines or in different files, Git will successfully merge commits without your help.

- However, when there are conflicting changes on the same lines, a “merge conflict” occurs because Git doesn’t know which code to keep and which to discard.

GITHUB



MORE ABOUT GITHUB

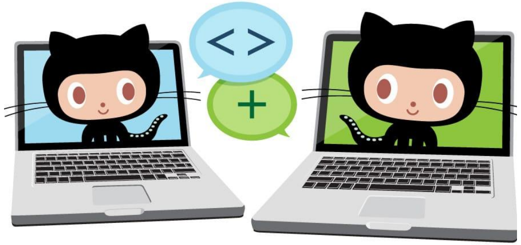
GITHUB

MORE ABOUT GITHUB

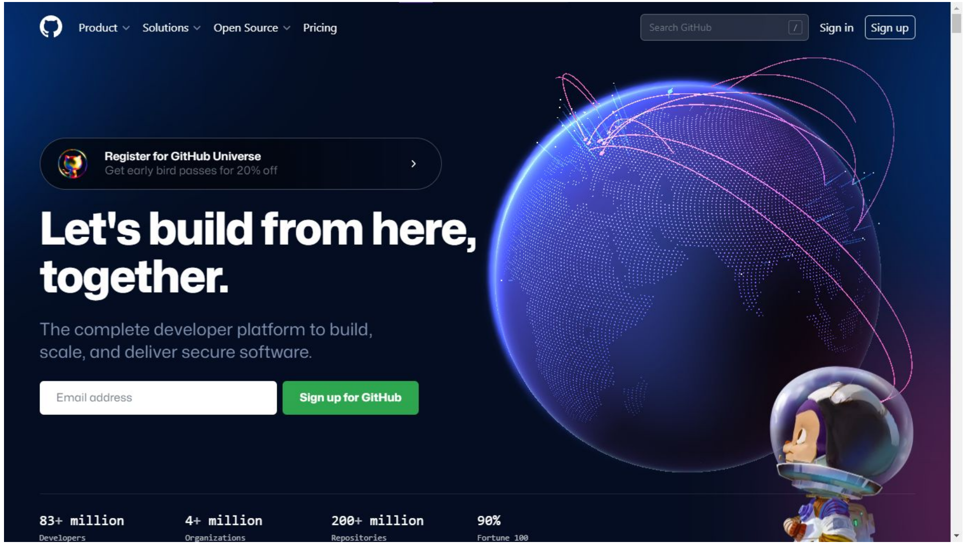
1. **GITHUB is not GIT.** Both are different Things.

2. **Git** is a VCS or Version Control System which tracks of code changes over a period of time.

3. While, **Github** is a hosting platform service layered over GIT (i.e., uses GIT) for tracking the code changes but on cloud.



Git and GitHub are not the Same !!



Register for GitHub Universe
Get early bird passes for 20% off

Let's build from here, together.

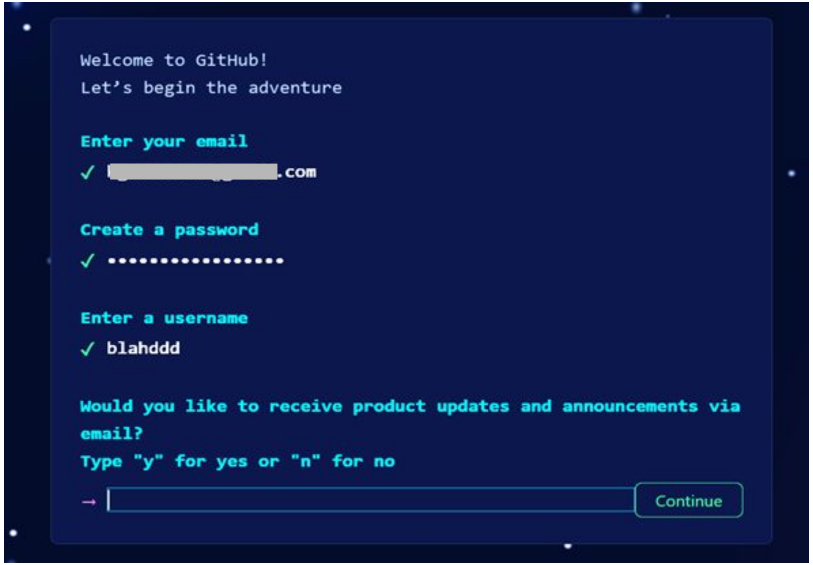
The complete developer platform to build, scale, and deliver secure software.

83+ million
Developers

4+ million
Organizations

200+ million
Repositories

90%
Fortune 100



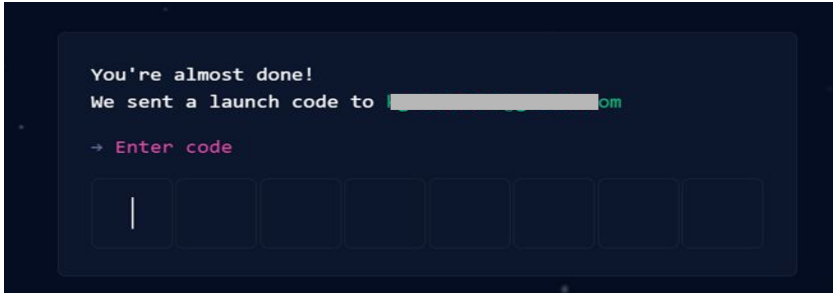
```
Welcome to GitHub!
Let's begin the adventure

Enter your email
✓ [redacted].com

Create a password
✓ ••••••••••••••••

Enter a username
✓ blahddd

Would you like to receive product updates and announcements via
email?
Type "y" for yes or "n" for no
→ |
```



```
You're almost done!
We sent a launch code to [redacted]om

→ Enter code
```

Creating GitHub account

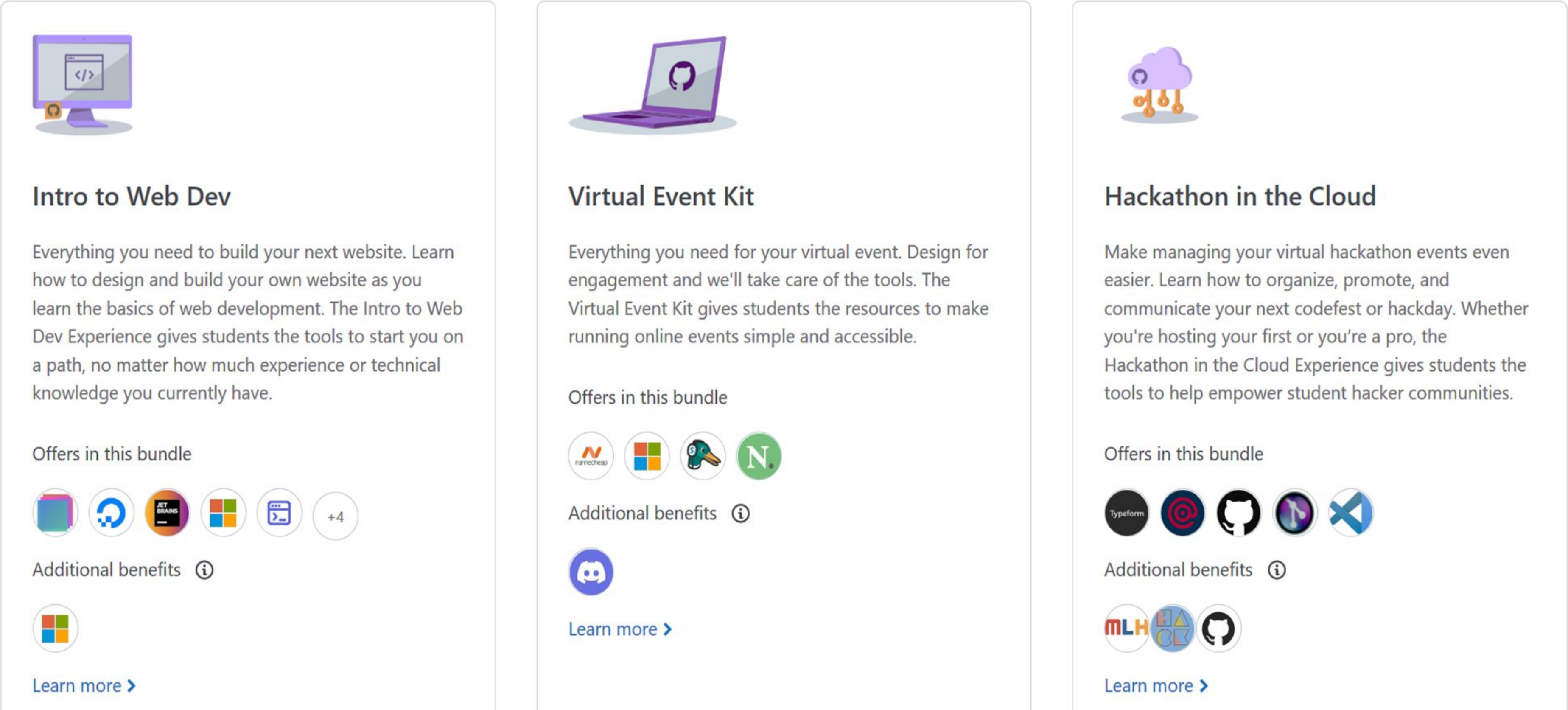
GitHub Student Developer Pack...

- Provides 100+ apps and softwares and various developer tools for free.
- Learning tools and courses can be availed for free.
- Bundles to conduct your own events like hackathons, codefests.
- Discover Virtual Events and Online Talks.

educative

namecheap

Microsoft Azure



Intro to Web Dev

Everything you need to build your next website. Learn how to design and build your own website as you learn the basics of web development. The Intro to Web Dev Experience gives students the tools to start you on a path, no matter how much experience or technical knowledge you currently have.

Offers in this bundle

+4

Additional benefits ⓘ

Learn more ›

Virtual Event Kit

Everything you need for your virtual event. Design for engagement and we'll take care of the tools. The Virtual Event Kit gives students the resources to make running online events simple and accessible.

Offers in this bundle

Additional benefits ⓘ

Learn more ›

Hackathon in the Cloud

Make managing your virtual hackathon events even easier. Learn how to organize, promote, and communicate your next codefest or hackday. Whether you're hosting your first or you're a pro, the Hackathon in the Cloud Experience gives students the tools to help empower student hacker communities.

Offers in this bundle

Additional benefits ⓘ

Learn more ›

GitHub Student Developer Pack...

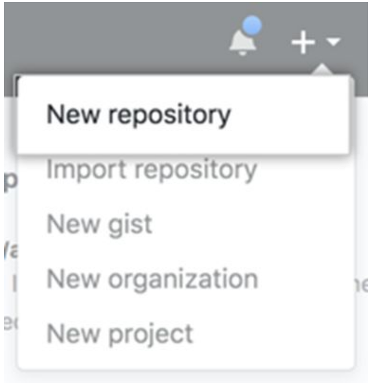
What Is a repository?

A repository **contains all of your project's files and each file's revision history**. You can discuss and manage your project's work within the repository.

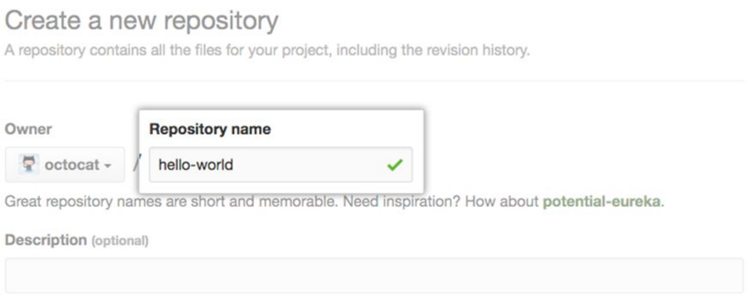
Features Of GitHuB...

CREATING A GITHUB REPO...

1. Optionally, add a description of your repository. For example, "My first repository on GitHub."



2. Type a short, memorable name for your repository. For example, "hello-world".



Create a new repository

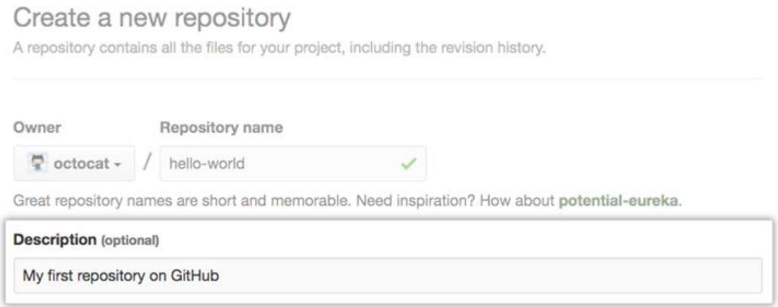
A repository contains all the files for your project, including the revision history.

Owner: octocat / Repository name: hello-world ✓

Great repository names are short and memorable. Need inspiration? How about potential-eureka.

Description (optional)

3. Optionally, add a description of your repository. For example, "My first repository on GitHub."



Create a new repository

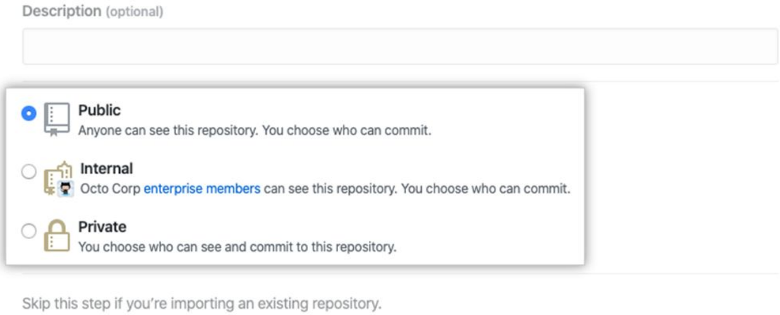
A repository contains all the files for your project, including the revision history.

Owner: octocat / Repository name: hello-world ✓

Great repository names are short and memorable. Need inspiration? How about potential-eureka.

Description (optional): My first repository on GitHub

4. Choose a repository visibility.



Description (optional)

● Public
Anyone can see this repository. You choose who can commit.

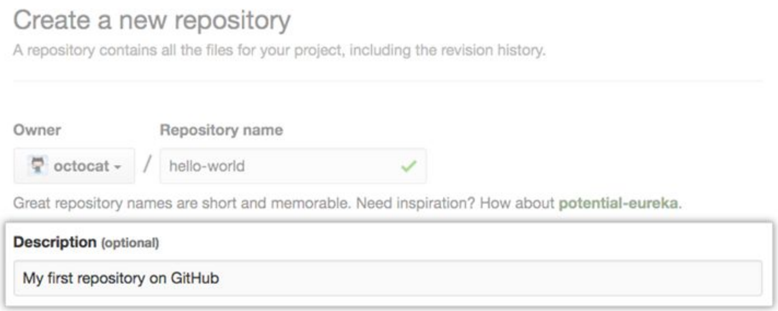
○ Internal
Octo Corp enterprise members can see this repository. You choose who can commit.

○ Private
You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

CREATING A GITHUB REPO...

5. Select **Initialize this repository with a README**.



Create a new repository

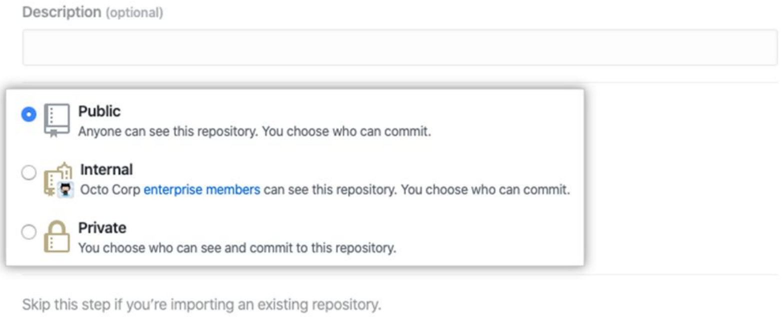
A repository contains all the files for your project, including the revision history.

Owner: octocat / Repository name: hello-world ✓

Great repository names are short and memorable. Need inspiration? How about potential-eureka.

Description (optional): My first repository on GitHub

6. Click **Create repository**.



Description (optional)

● Public
Anyone can see this repository. You choose who can commit.

○ Internal
Octo Corp enterprise members can see this repository. You choose who can commit.

○ Private
You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

CREATING A GITHUB REPO...

- The main idea about remote repo is that you can collaborate to other users, more backup safety, etc.

- As a starting point you can create a Github Account.

- Get yourself familiarise by exploring some of its features.

WORKING WITH REMOTE REPOSITORIES

- Go to your project folder(where you have git initialised)
- Add your github repo link as remote for pushing files to github repo
- By convention the name of the remote normally should be “origin” (Although you can name it anything)
- Type the command: “git remote add origin <your github url>”
- Now to push all your commits to the repo type:

  “git push -u origin <your branch name>”

ADDING FILES TO REMOTE REPO

ADDING FILES TO REMOTE REPO

- But what if you want to work on another project, using new repo afterwards? Simple, you change the origin.
- Type the command: “git remote set-url origin <your github url>”
- Now as before, to push all your commits to the repo type:

  “git push -u origin <your branch name>”

README

- The most important part of a git repository is a README.md file
- This file summarises about the project and helps other user to understand about your code
- .md refers to Markdown File
- Some basics of Markdown:

# Text -> Main Heading Level

## Text -> 2nd Heading Level

Google for detail cheatsheet

Starring and Following:

a. You can star the repositories you find interesting; so you may come back to them later for contributing or just using them in your own projects.

b. Follow the people whose progress and activity you want to follow on GitHub. You will see the repos they starred, people they followed and you get to learn a lot.

GITHUB IS THE SOCIAL NETWORK FOR CODERS!

GitHub Profile README:

a. It is a special README file that shows whenever someone opens your GitHub profile.

b. An awesome place to tell people about yourself, your interests, projects and just basically leave an impression onto the visitor.

c. Can include a variety of stuff like Badges, Stats, various cards like your latest blogs, the songs you listen to, your email-id and other cool things to show-off!

How to create Profile README?

a. Create a repository with the exact same name as your GitHub Username. Eg: grgkaran03

b. Then create a README.md file inside that repo and start editing it in GitHub itself.

c. https://rahuldkjain.github.io/gh-profile-readme-generator/ to create beautiful README's very quickly. Then edit as per your likings.

THE README IS YOUR CANVAS!!

Cool things to add to Profile README!

a. Add your Medium blogs: https://github.com/bxcodec/github-readme-medium-recent-article/

b. Show your music taste: https://github.com/kittinan/spotify-github-profile

c. Flex your GitHub stats (my favourite part about Profile README ;)): https://github.com/anuraghazra/github-readme-stats

d. Many more cool stuff to add (emojis, GIFs, skill badges, social media handles): https://towardsdatascience.com/build-a-stunning-readme-for-your-github-profile-9b80434fe5d7/

Portfolio website using GitHub Pages:

a. You get your own domain to host your personal website with your GitHub account. All for free!

b. Very easy to deploy and it supports everything from basic HTML, CSS based websites to full-fledged React Apps.

c. You have your own personal website that re-deploys whenever you make any changes in the code and push it to the repo. No hassles!

Create personal website on GitHub Pages:

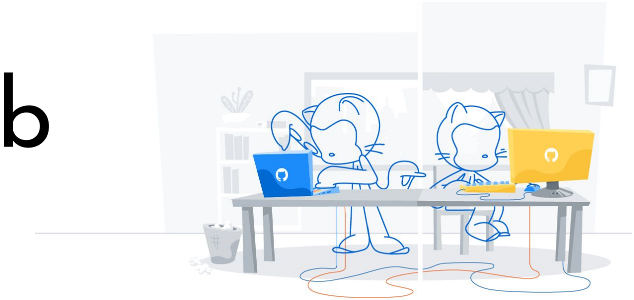
a. Create a repository with the name: **[Your Git User Name].github.io.** For ex: grgkaran03.github.io

b. **Download a template for your website and make changes to it or push your own files to the above repository. (***The entire process of cloning, committing and pushing to remote origin has been shown earlier in the workshop!***).**

c. **As soon as you push, you will be able to see the deployment in a few minutes. Also you can deploy your own projects as well, for ex:** https://grgkaran03.github.io/. Just push your React repository to GitHub and then go to settings and deploy it at your preferred domain-name. Make sure to make a production build of the app and push it to the repo on GitHub. More on this: https://github.com/gitname/react-gh-pages/

MORE ABOUT GITHUB

Collaborating with GitHub



To collaborate with multiple people working on the same project, there are two major options:

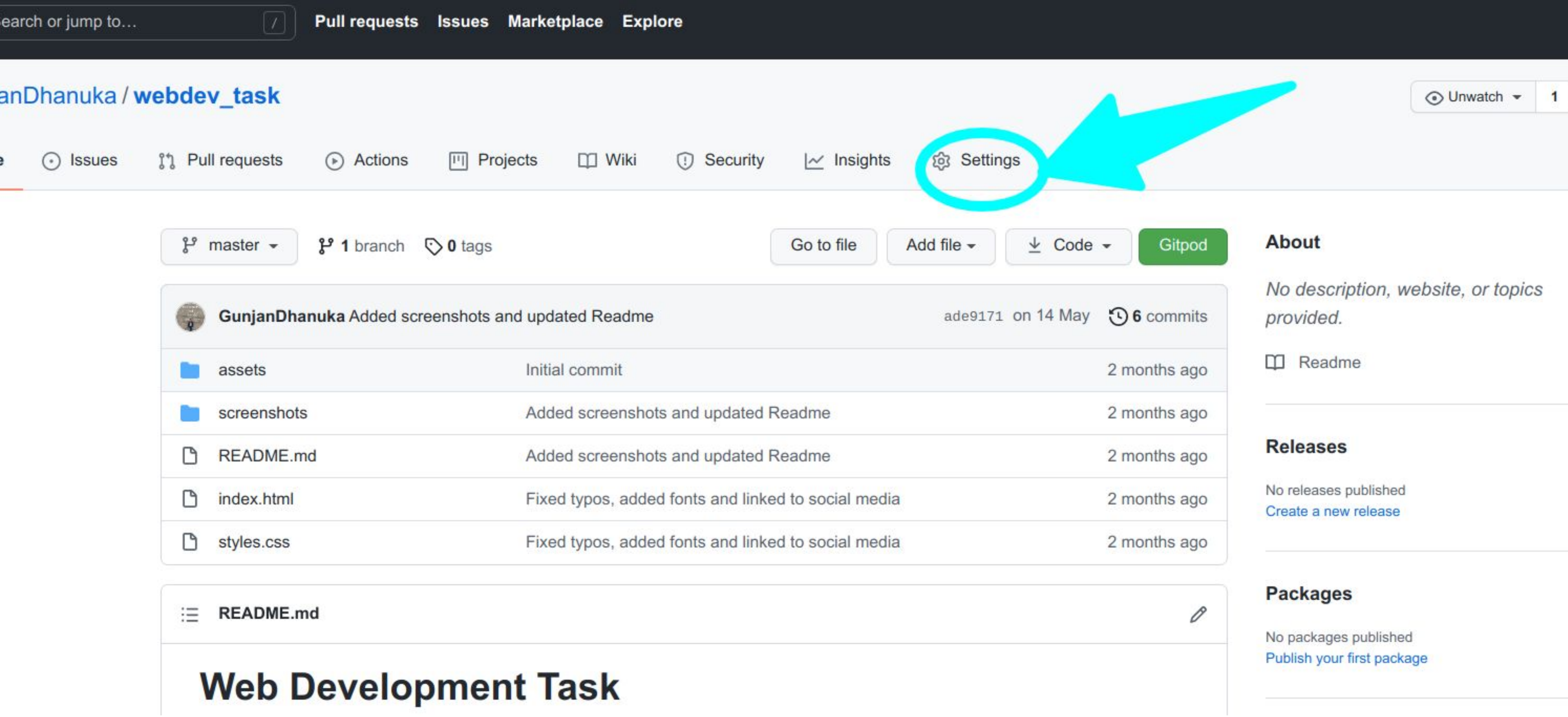
1. Add COLLABORATORS (If you are the owner of the repo)
2. Fork + Pull Request (If you want to someone else's repo)

1. As the owner of a repo you:

- add people as collaborators
- each collaborator can read/write files in the repo
- each collaborator is adding files and other content → making branches → and either directly merging changes in or via pull requests

YOU ADD COLLABORATORS IF THEY ARE A CORE PART OF THE TEAM!

MORE ABOUT GITHUB



GunjanDhanuka Added screenshots and updated Readme ade9171 on 14 May 6 commits

assets	Initial commit	2 months ago
screenshots	Added screenshots and updated Readme	2 months ago
README.md	Added screenshots and updated Readme	2 months ago
index.html	Fixed typos, added fonts and linked to social media	2 months ago
styles.css	Fixed typos, added fonts and linked to social media	2 months ago

README.md

Web Development Task

About

No description, website, or topics provided.

Readme

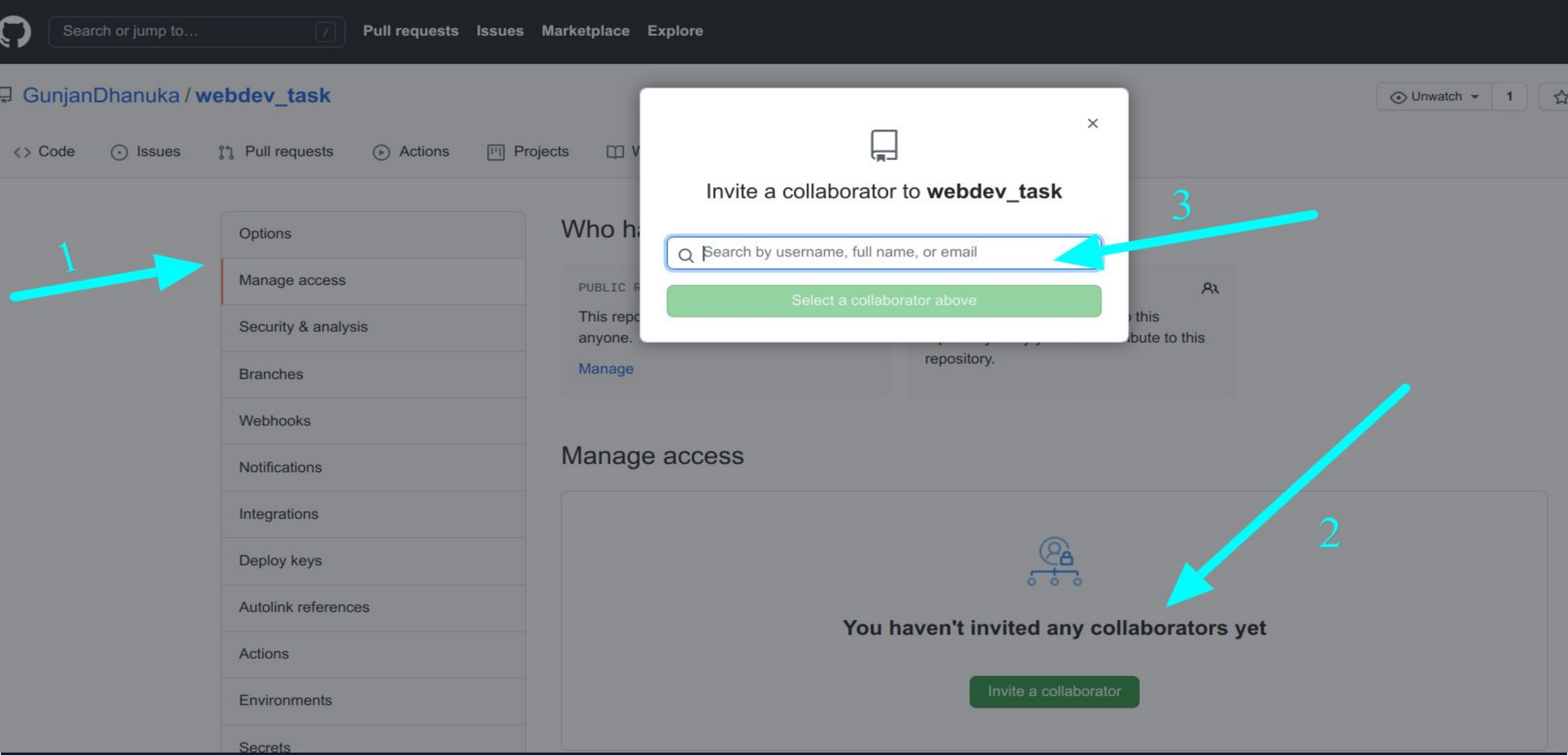
Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

MORE ABOUT GITHUB



Invite a collaborator to **webdev_task**

Who h

This repo
anyone.

this
bute to this
repository.

Manage access

You haven't invited any collaborators yet

MORE ABOUT GITHUB

2. If you don't own a repo/ not an official contributor:

a. You will FORK a repo.

b. Work and make changes on the forked copy of the repo.

c. Make a PULL REQUEST in order to make your changes accepted into the original repo.

d. The owner of the original repo will check if your changes are helpful - and MERGE them in!

YAY YOU NOW HAVE SOME COOL INTERNET FRIENDS!

MORE ABOUT GITHUB

To fork and clone a repository

a. Open a repository.

b. Click on Fork button.

c. Click on Create Fork.

d. Open the forked repo, click on Code and copy the url.

e. Open git bash and type: git clone <your-remote-git-repo-url>

f. Type: cd <filename>

g. Type: git remote add upstream <your-remote-git-repo-url>

FORKING AND CLONING

To create a pull request

a. Open the source repository.

b. Click on Contribute.

c. Click on Open Pull Request.

d. Type a comment and then click Pull Request.

CREATE A PULL REQUEST

Contribution guidelines

- Don’t commit code as an unrecognized author.
- Define code owners for faster code reviews.
- Keep branches up to date.
- Enable security alerts.

Licensing

- **APACHE**

  Permissions-commercial use,modification,distribution,patent use,private use
  Limitations-trademark use,liability,warranty

- **MIT**

  Permissions-Commercial use,Modification,Distribution, Private use
  Limitations-liability,warranty

- **GNU**

  Permissions-commercial use,modification,distribution,patent use,private use
  Limitations-liability,warranty

MORE ABOUT GITHUB

Git Merge Conflicts

And how to resolve them

A merge conflict is an event that occurs when Git cannot automatically resolve the differences between two commits.

What are merge conflicts?

Conflicts generally arise when

- Two people have changed the same line in the same file.
- If a person deleted the file whilst other person was modifying it
- There are changes in either the working directory or staging area.

In these cases, Git cannot automatically determine which of the changes to keep and which to discard.

When do they occur...

➔ Git fails to start the merge `(Changes in working directory)`

A merge will fail to start when Git sees there are changes in either the working directory or staging area of the current project.

Git fails to start the merge because these pending changes could be written over by the commits that are being merged in.

When this happens, it is not because of conflicts with other developers', but conflicts with pending local changes.

To solve this type of conflict, the changes made need to be either committed or discarded, which can be done using **`git stash`**, **`git checkout`**, **`git commit`** or **`git reset.`**

Types of merge conflicts

➔ Git fails during the merge `(Changes in staging area)`

A failure DURING a merge indicates a conflict between the current local branch and the branch being merged.

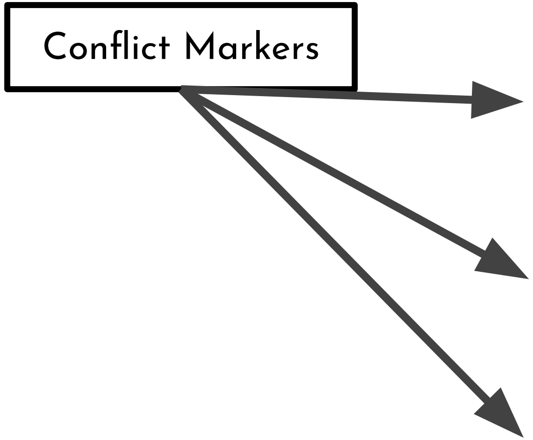
This indicates a conflict with another developers code.

Git will do its best to merge the files but will leave things for you to resolve manually in the conflicted files

Let us see how it happens and resolve it

Types of merge conflicts

WHO WOULD WIN?
A TEAM OF DEVELOPERS
ONE MERGY BOI



Conflict Markers

```
<<<<<<< HEAD

//conflict in current branch

=======

//conflict in merging branch

>>>>>>> “Branch Name”
```

What do we have here…

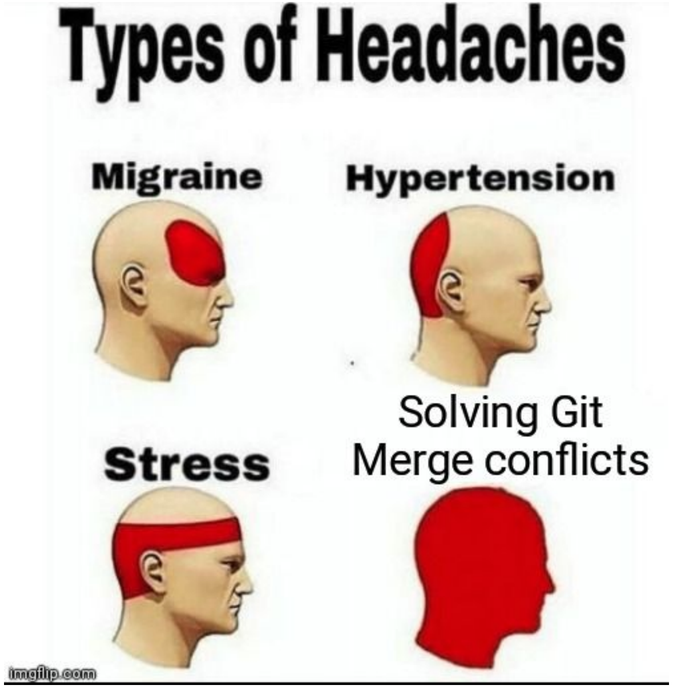
Resolving through command line is done on local repository, and then pushed to remote repository.

- You can either save the changes from both the files by just removing the conflict markers i.e “<<<<<<< HEAD”, “========” and “>>>>>>> “Branch Name””
- Or save either of the change by just keeping the content within the either pair of conflict markers i.e. “<<<<<<< HEAD” and “========” or “=======” and “>>>>>>> “Branch Name””

Resolve using command line

You can resolve simple merge conflicts that involve competing line changes on GitHub, using the conflict editor.

For more complex conflicts, you'll have to resolve them locally and then push to remote



Resolving on GitHub

So, How do we exactly do this on GitHub?

- Frequently Pull/Push code to the mainline branch, which makes it less likely to have conflicts and they'll be smaller when they happen
- Make devs work on independent parts of the code.
- Frequent Communication with your fellow team mates

Some tips to avoid/minimize merge conflicts

15 Min Break

GitHub Actions

Introduction

GitHub Actions allows you to automate your build, test, and deployment pipeline. You can create **`workflows`** that build and test every pull request to your repository, or deploy merged pull requests to production.

Workflows

A workflow is a configurable automated process that will run one or more jobs.
Eg: You can have a workflow that adds a label every time someone opens a new issue

A repository can have multiple workflows, each of which can perform a different set of tasks.

- Triggering a workflow

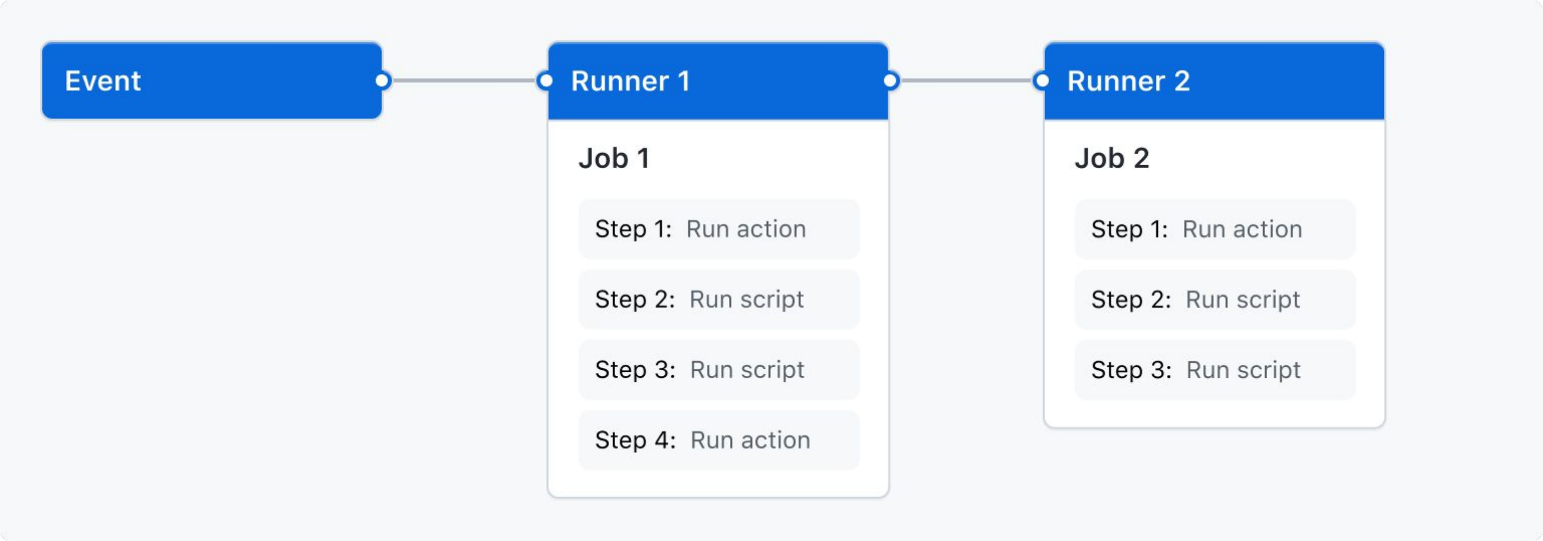
Workflow triggers are events that cause a workflow to run. These events can be:

- Events that occur in your workflow's repository
- Events that occur outside of GitHub and trigger a repository_dispatch event on GitHub
- Scheduled times
- Manual

GitHub Actions

Your workflow contains one or more *jobs* which can run in sequential order or in parallel.

Each job will run inside its own virtual machine *runner*, or inside a container, and has one or more *steps* that either run a script that you define or run an action.



Event

Runner 1

Job 1

Step 1: Run action

Step 2: Run script

Step 3: Run script

Step 4: Run action

Runner 2

Job 2

Step 1: Run action

Step 2: Run script

Step 3: Run script

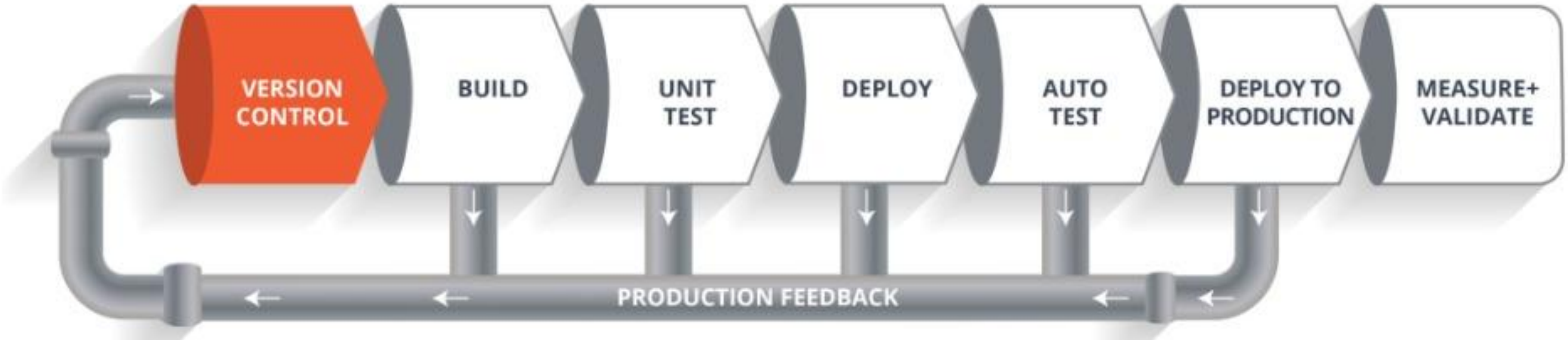
Components of GitHub actions

- **Events**
  A specific activity in a repository that triggers a workflow run.
  Eg: Activity can originate from GitHub when someone creates a pull request, opens an issue or pushes a commit to a repository.

- **Jobs**
  A job is a set of steps in a workflow that execute on the same runner. Each step is either a shell script that will be executed, or an action that will be run. Steps are executed in order and are dependent on each other.

- **Actions**
  An action is a custom application for the GitHub Actions platform that performs a complex but frequently repeated task.

Life Cycle of an Application !



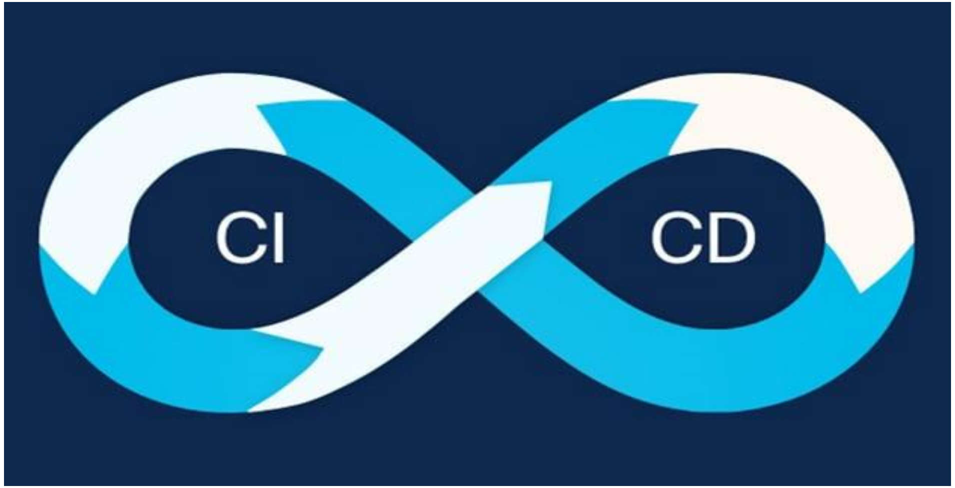
At

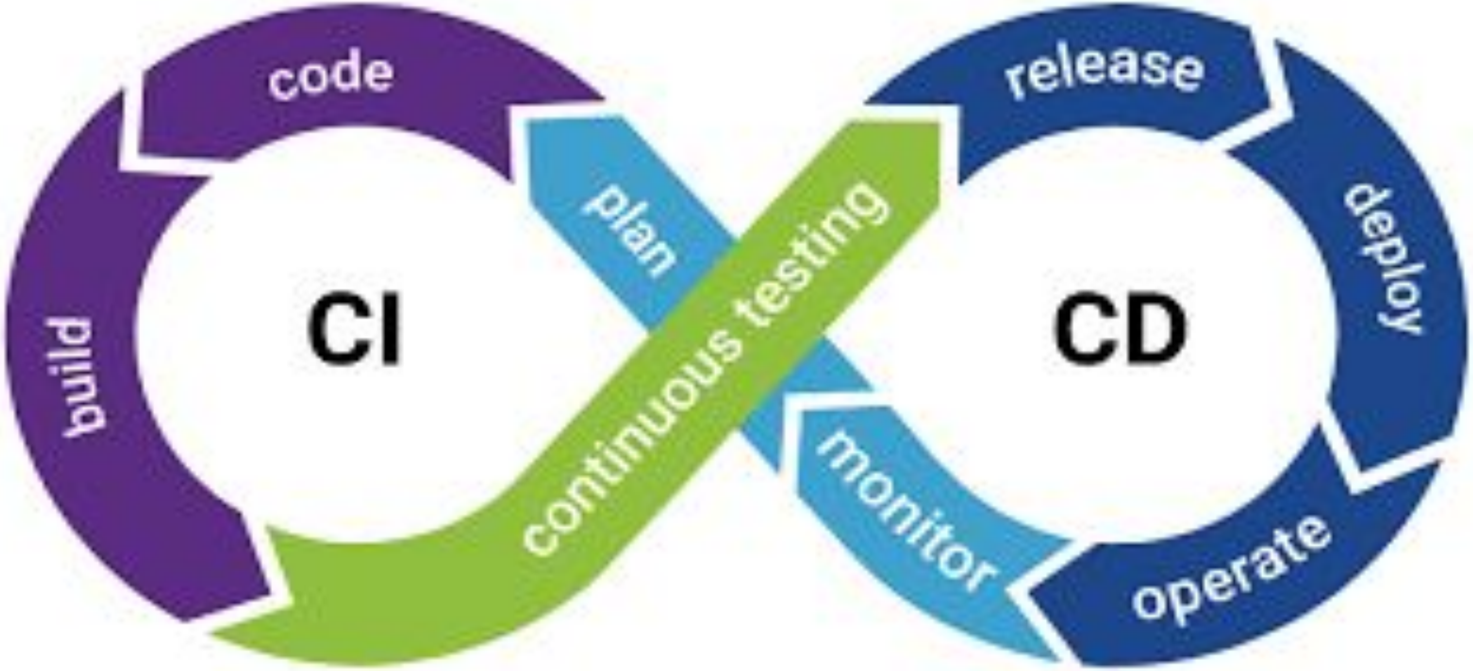
Can we do something to automate the above mentioned complicated method ???

At

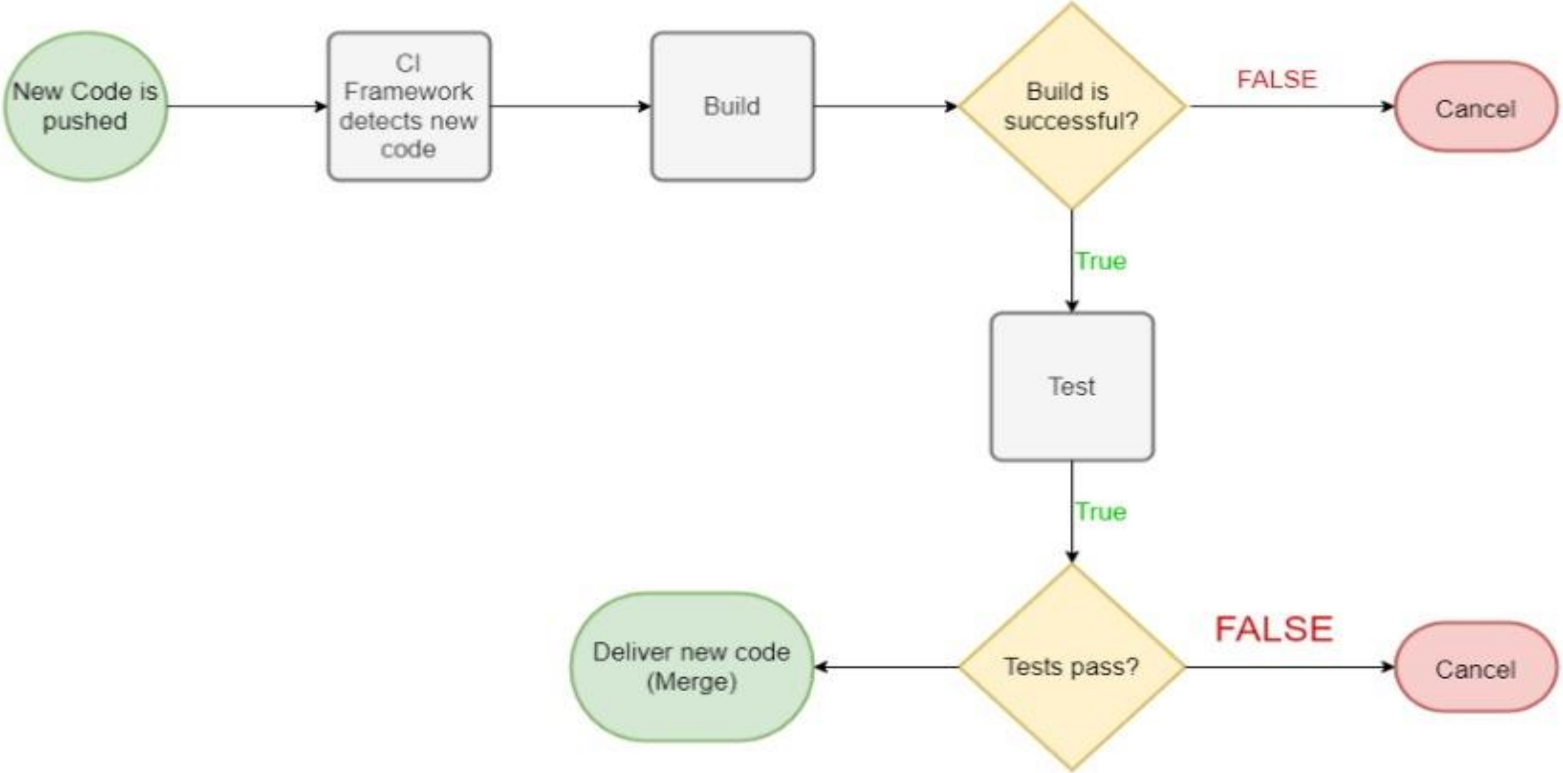
Introduction to CI/CD

a. What is CI (Continuous Integration)
b. What is CD (Continuous Deployment / Continuous Delivery)



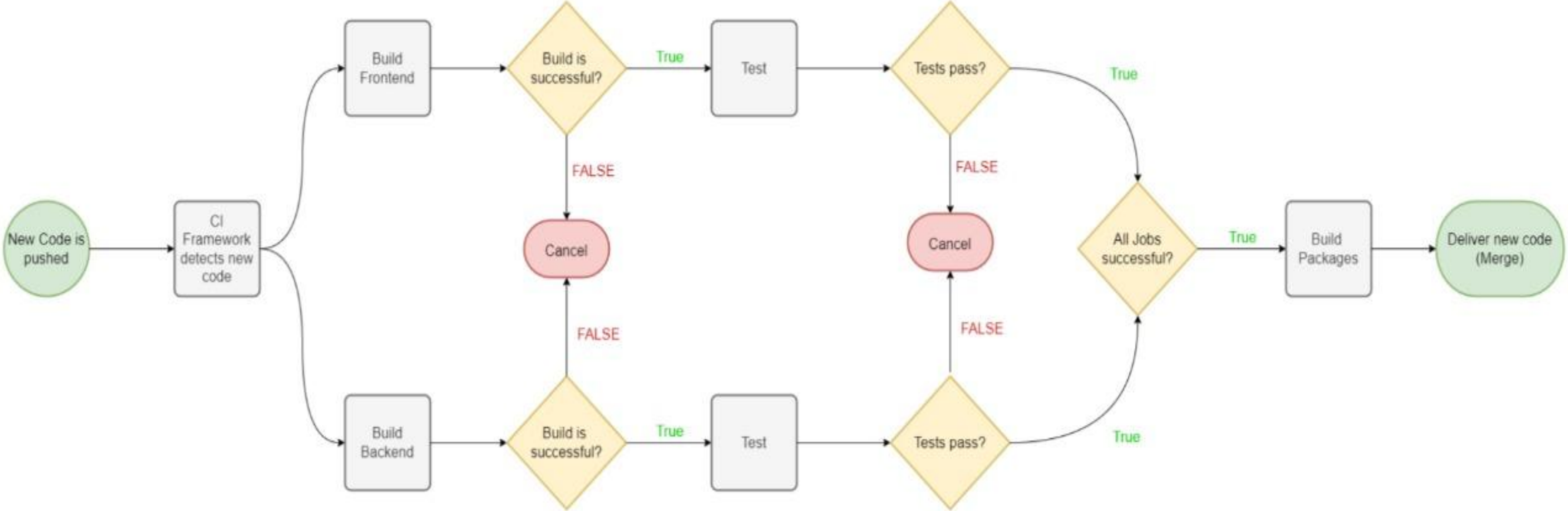

code
build
plan
continuous testing
CI
release
deploy
CD
operate
monitor

At



Simple CICD pipeline

The example given is of course very straight forward but CI/CD pipelines can get very complex very quickly, take for example this next pipeline.



of your project and its requirements you can decide what is most suitable for you. If you have a small project and want to get your toes wet in some CI/CD action you can give GitHub Acti EE! It is free for public GitHub

Migration Easier

Debugging as a Superpower