Git - Github

'Global Information Tracker'

Git - Github

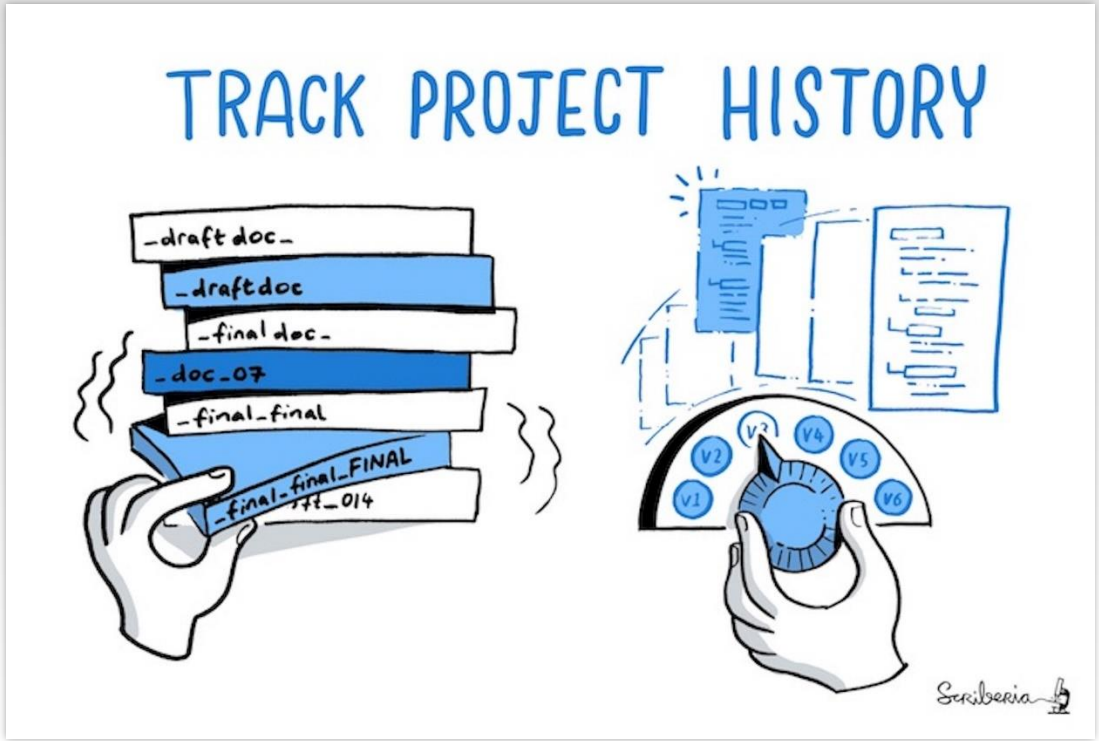
Introduction

- What is Git – Github?
- What is VCS?
- What is it used for?

Git-Github

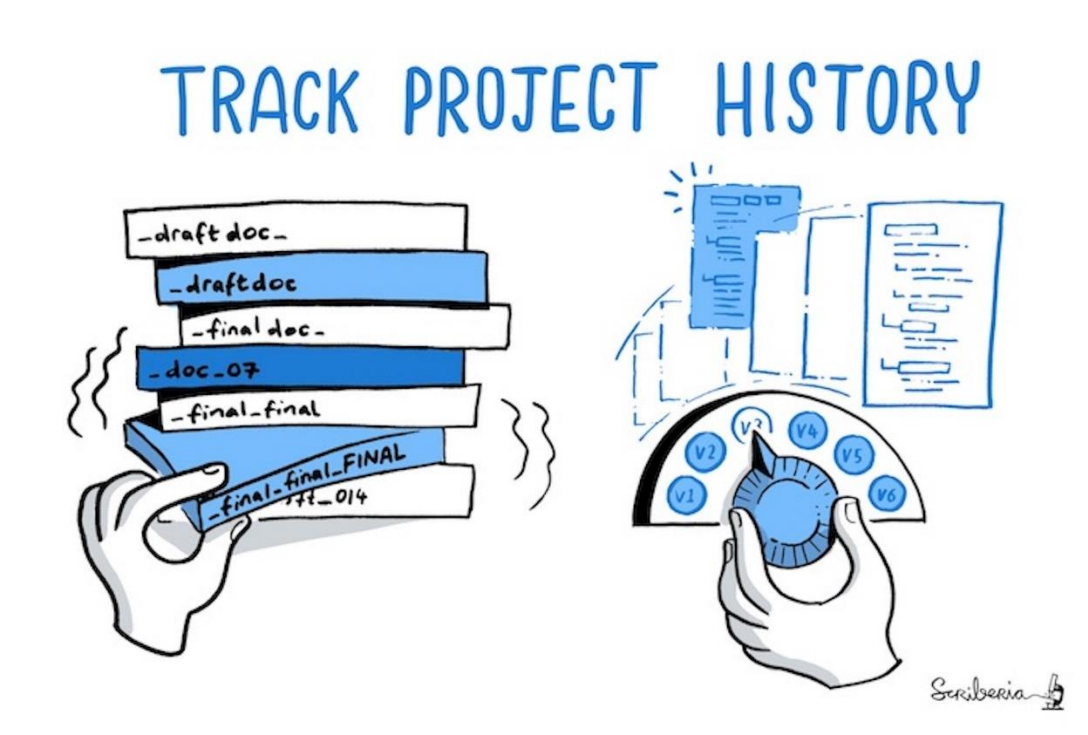
Git-Github is a version control system

VCS



A version control system is a system that records changes in file and folder structure in the project.

VCS



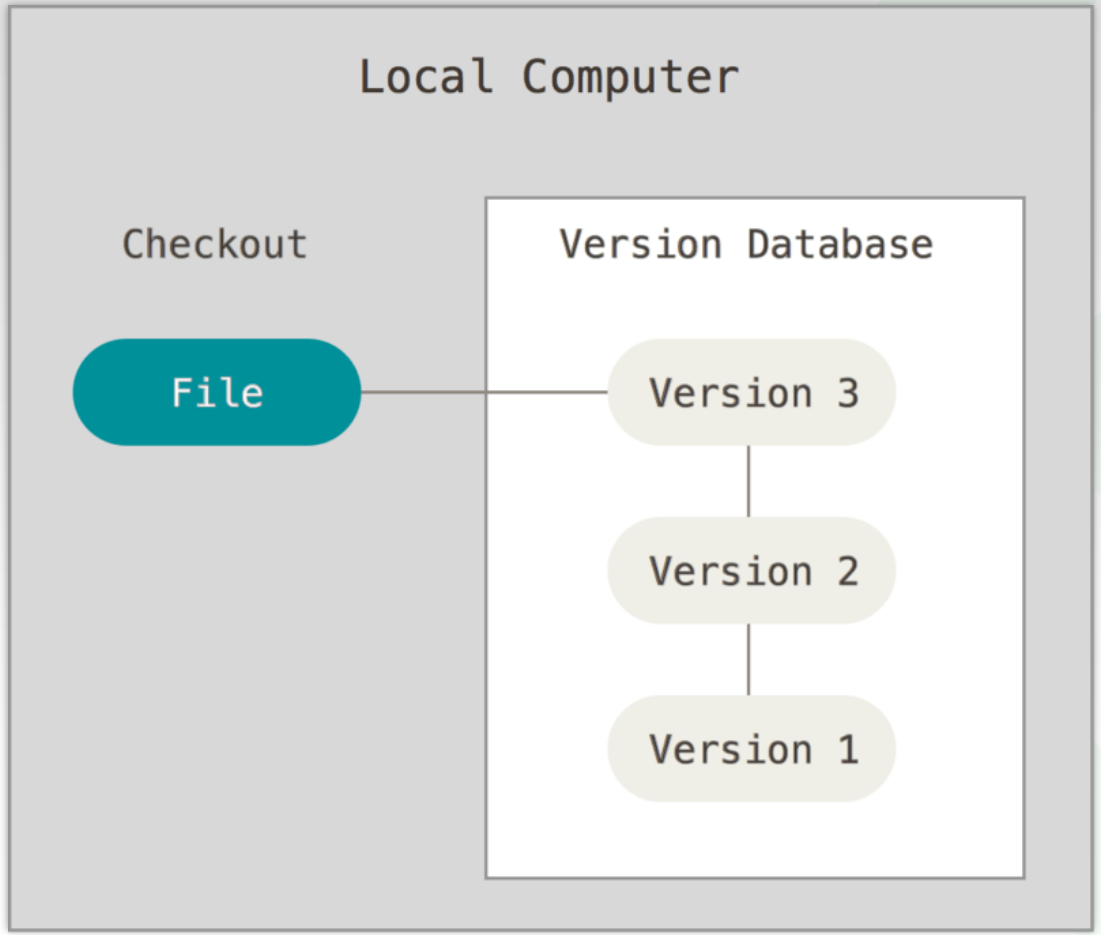
- Reverting some files or the entire project to the previous version,
- Comparison of changes made over time,
- Identifying who made the last changes that cause problems

Types of VCS

There are 3 types of Version Control Systems.

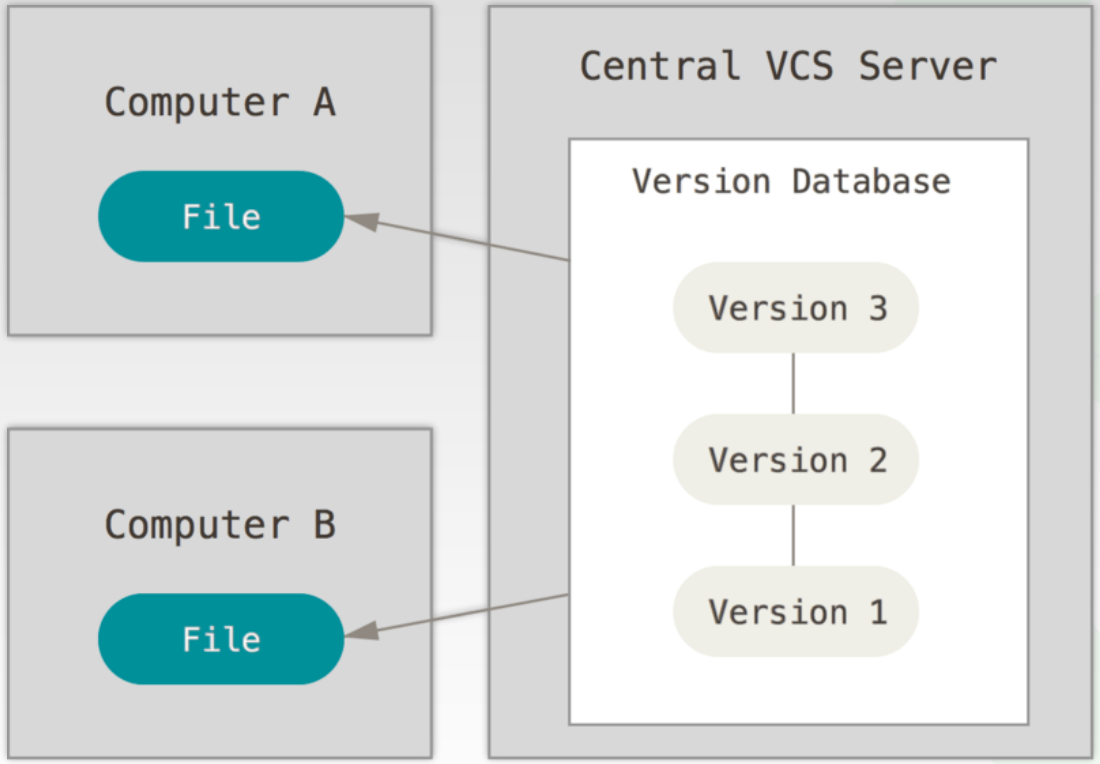
- LOCAL
- CENTRALIZED
- DISTRIBUTED

Local Version Control System



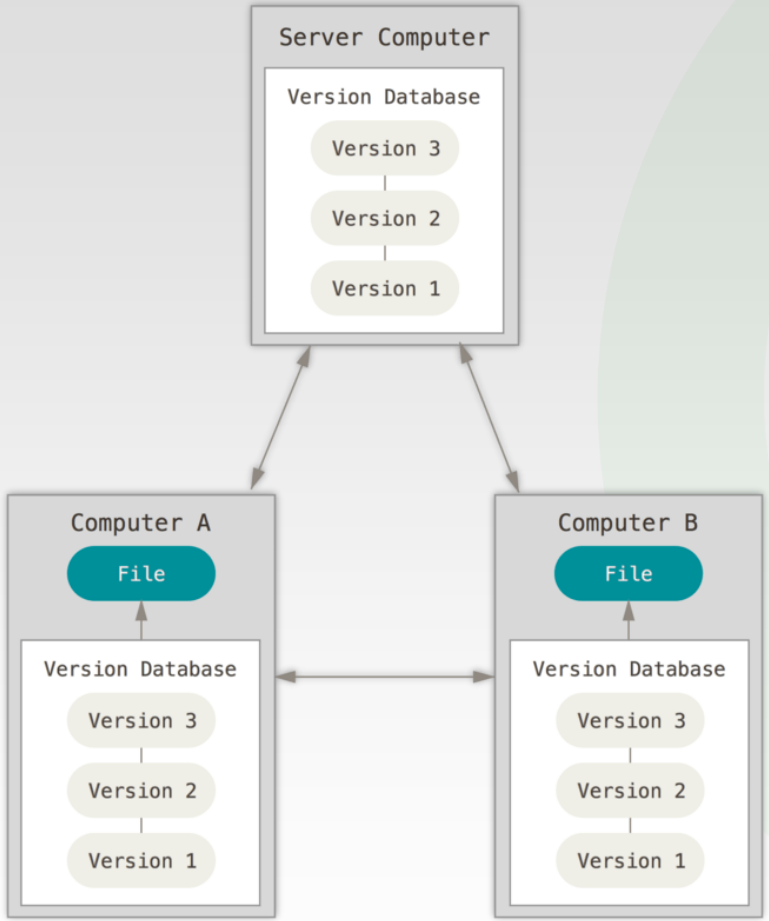
https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

Centralized Version Control System



https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

Distributed Version Control System



https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

What is Git-Github Used for?

git LOCALE

- Local version management
- To be able to work offline
- Undo errors
- Ability to switch between versions

GitHub REMOTE

- Backup
- Share
- Deploy
- Collobration

Git - Github

- Installation and initial settings
- Repository
- Creating a local repo
- Working directory, staging area, commit changes
- Cancel changes
- Reverting to previous versions
- Branches

Installation and Initial Settings

Version Control System

- In order to create the Git infrastructure and use git commands, the Git library must be installed

  [https://git-scm.com/downloads]

- *git --version*

Installation and Initial Settings

Git configuration

```
git config --global user.name "John Doe"
git config --global user.email "john@doe.com"
```

It associates the commits with the name and e-mail specified here. Other people working in the repo will see this name and email.

```
git config --global color.ui true
```

Provides coloring of commands in terminal

- **System** parameter takes effect on all users and all repositories.
- **Global** parameter takes effect on all repositories of the current user
- **Local** parameter is only effective on the current repo.

Repository

Repository is like a data structure used by VCS to store metadata for a set of files and directories. It contains the collection of the files as well as the history of changes made to those files.

Creating a local repo

git init

Git init command is used to import a project into the version system on our local computer. Using this command creates a **.git** folder in the project folder. This will store our local repo.

General Concepts

Working Space

It is the workspace where the .git folder is located. Changes to folders and files are made here.

Staging Area

It is the place where the files or folders to be versioned are temporarily collected.

After the version (commit) is created, the staging area is automatically emptied.

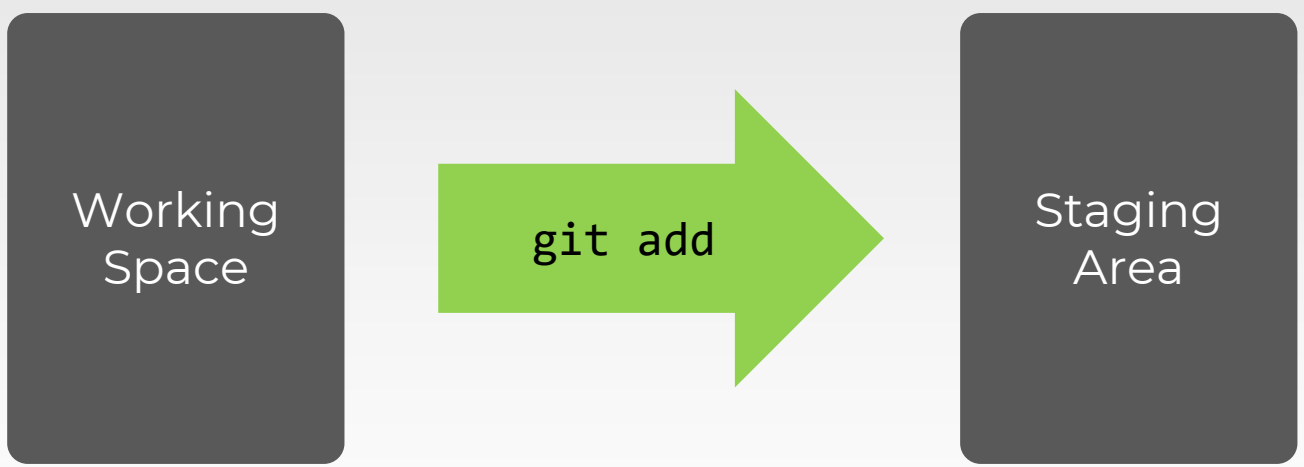
Commit Store

Git keeps each commit as a separate version. Thus, if problems arise in the project after various changes, it is possible to return to the previous commit.

Creating local versions

It is used to see the status of Working Space or Staging area.

git status

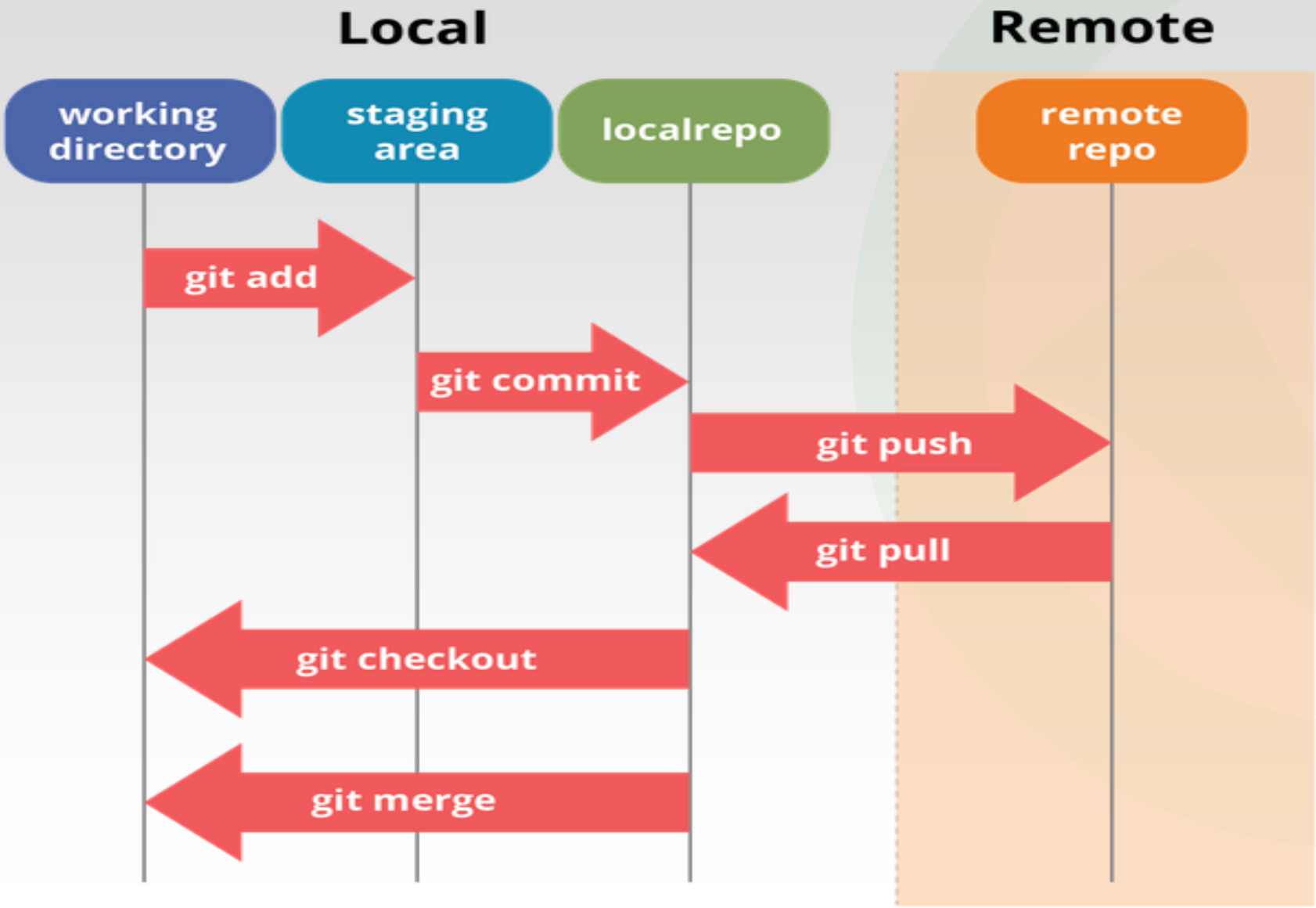


git log

This command is used to see the versions created

```
git add file_name
or
git add .
```

```
git commit –m"version name"
```


Local
Remote
working directory
staging area
localrepo
remote repo
git add
git commit
git push
git pull
git checkout
git merge

Version details

git show

It is used to see what changes have occurred within a version.

git show *[first 7 characters of the hash code]*

git log

```
C:\Users\TechProEd\Desktop\MyProject>git show 9b4be5a
commit 9b4be5ac81aa3fba354a79006318d856406c2093 (HEAD -> master)
Author: techproed <techproeducation@gmail.com>
Date:   Fri Mar 10 19:06:38 2023 +0300

    version 2

diff --git a/Codes.java b/Codes.java
index e69de29..c6568ea 100644
--- a/Codes.java
+++ b/Codes.java
@@ -0,0 +1 @@
+new line
\ No newline at end of file

C:\Users\TechProEd\Desktop\MyProject>
```

Codes to create versions

Main commands

```
git init
git add .
git commit -m "version text"
```

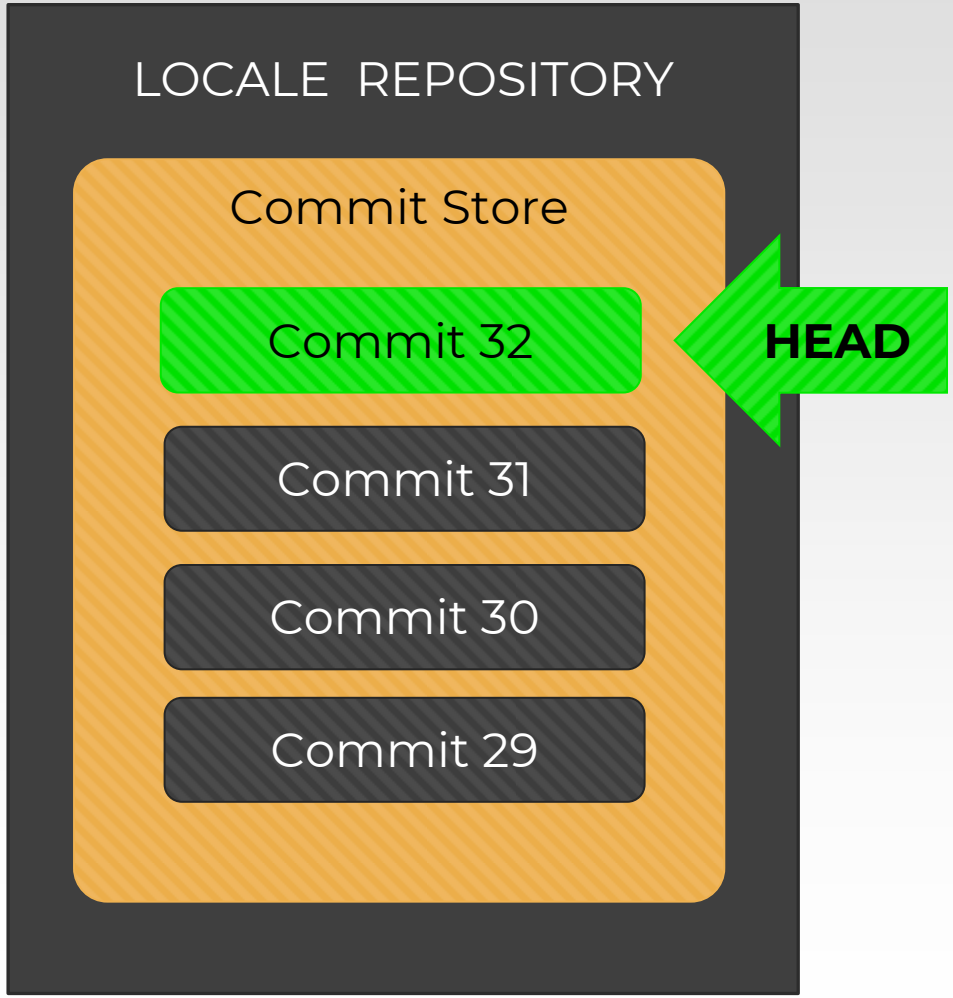
- `git init` — Creates a repo. Used only once in each project.
- `git add .` — Sends files to the staging area
- `git commit -m "version text"` — Creates a version

Auxiliary commands

```
git status
git log
git show [hash_kodu]
```

- `git status` — Provides information about the general situation
- `git log` — Returns a list of versions
- `git show [hash_kodu]` — Shows changes in version

Commit Store & Head



- There can be more than one commit in a repo. The most recent commit is called **HEAD**.
- Changing this **HEAD** can revert to previous versions.

```
C:\Users\TechProEd\Desktop\MyProject>git show 9b4be5a
commit 9b4be5ac81aa3fba354a79006318d856406c2093 (HEAD -> master)
Author: techproed <techproeducation@gmail.com>
Date:   Fri Mar 10 19:06:38 2023 +0300

    version 2

diff --git a/Codes.java b/Codes.java
index e69de29..c6568ea 100644
--- a/Codes.java
+++ b/Codes.java
@@ -0,0 +1 @@
+new line
\ No newline at end of file

C:\Users\TechProEd\Desktop\MyProject>
```

HEAD

Canceling the changes

Working space

git restore *[file]*

Cancels a single file

git restore .

Cancels all files

Stage Area

git restore --staged *[file]*

Cancels a single file

git restore --staged .

Cancels all files

Commit Store

git checkout [hash] [file]

The file returns to the version specified with the hash.

git checkout [hash] .

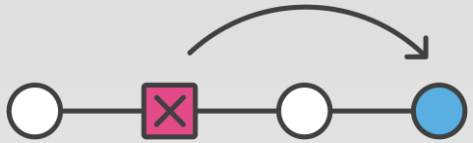
Returns to the version hash value given

git reset --hard

It cancels the changes in the working space and empties the staging area.

Revert to previous versions

1.Way: CHECKOUT



To view the previous version

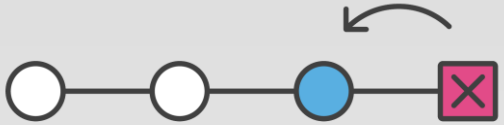
git checkout *[hash]* .

To make this action permanent

git commit –m"..."

Revert to previous versions

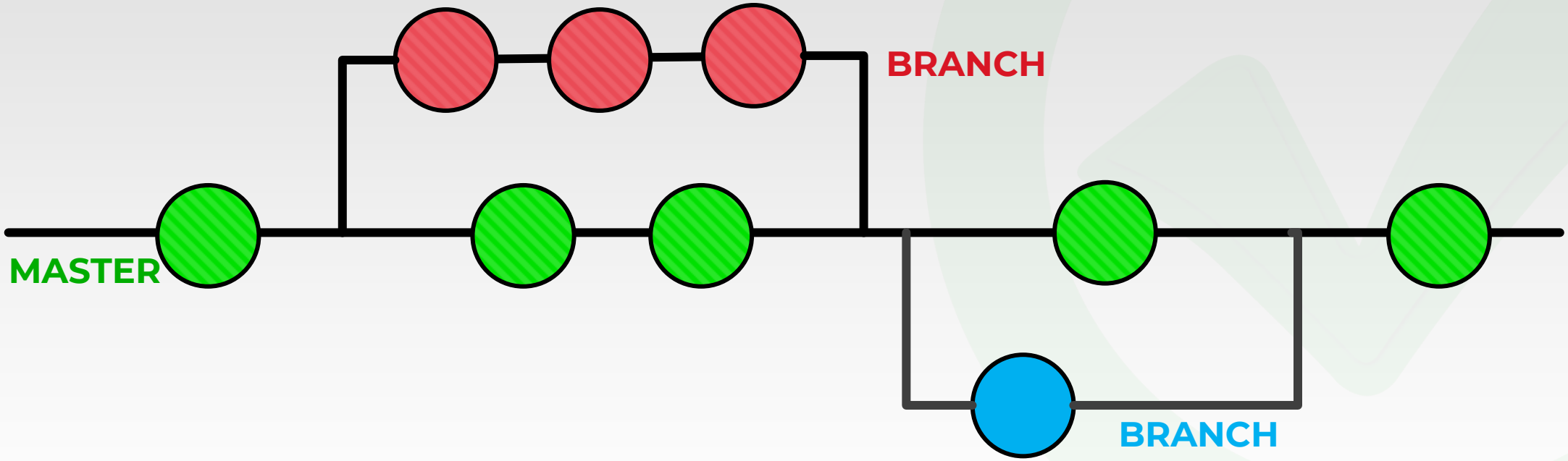
2.Way: RESET



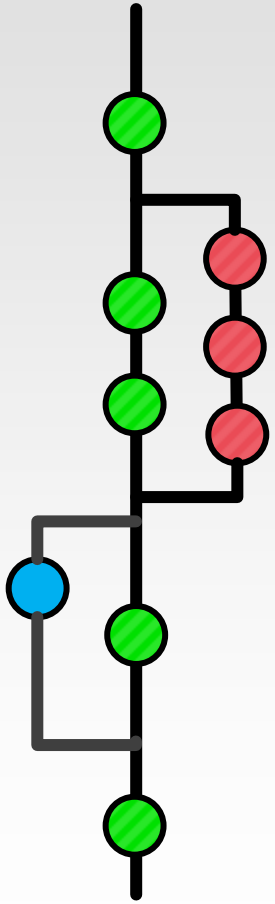
Revert to previous version **irreversibly**

git reset --hard *[hash]*

Branch



Benefits of branches



- The security of the original codes is ensured
- Each developer is responsible for his own section
- Faster development
- Fewer errors
- Problems are fixed faster.
- Organized code structure
- There will be no chaos

Branch Commands

git branch *[name]*	git checkout *[name]*	git branch -m *[name]*
Creates a new branch	**Branch becomes active**	**Changes the branch name**
git branch	git branch –d *[name]*	git merge *[name]*
Lists existing branches	**Deletes branch**	**Merges branches**

Stashing

Stashing is done to temporarily undo the changes in the working directory and the stage area, which have not yet been committed.

git stash

Stores the changes made after commit to a temporary memory and cleans working space and staging area

git stash list

Used to see stored changes

git stash pop

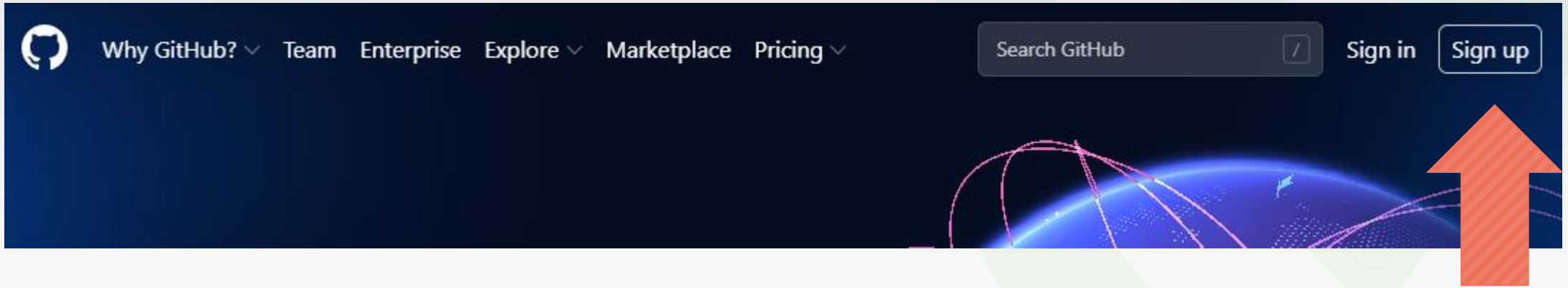
Used to restore stored changes.

Git - Github

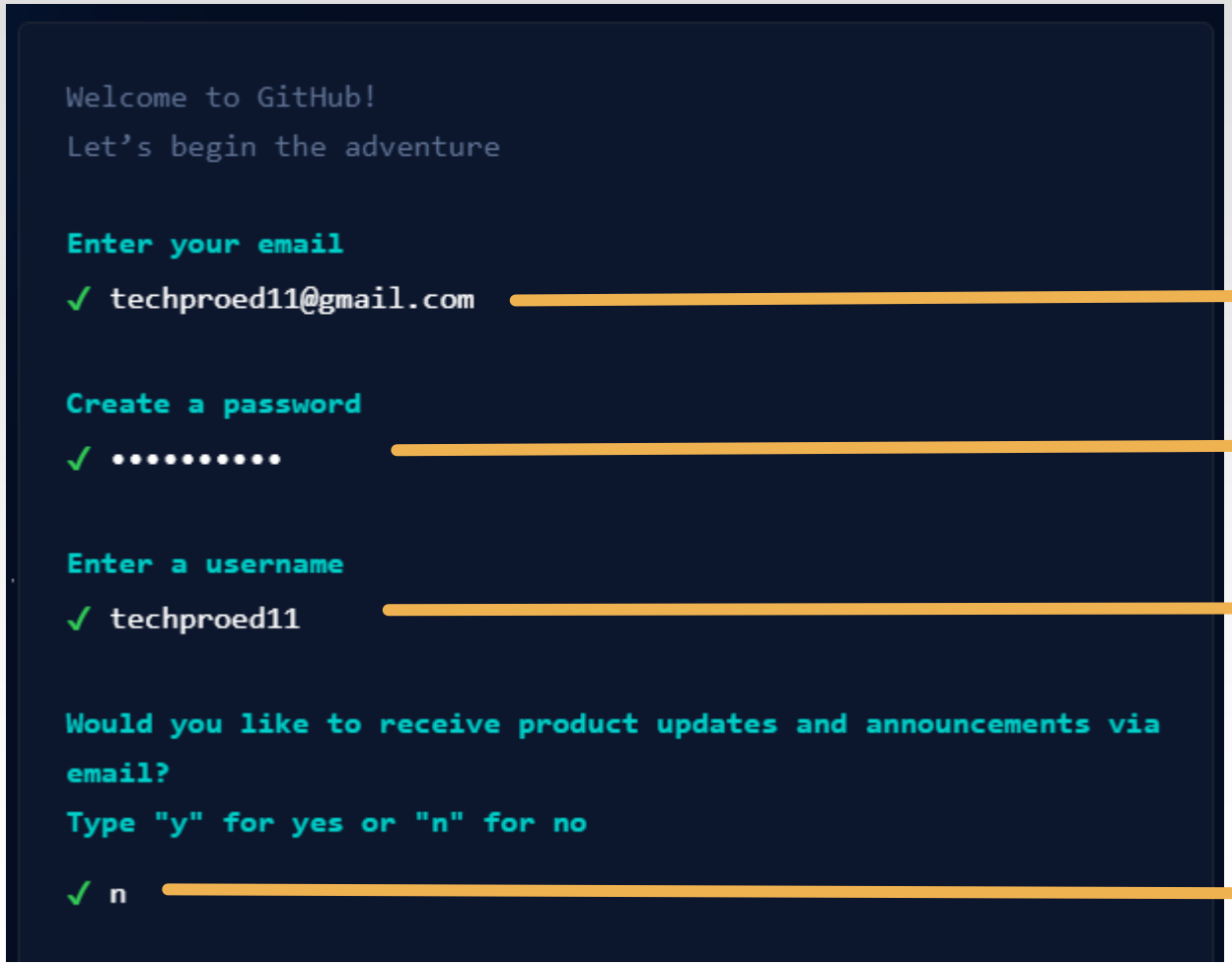
- Account creation
- Creating a repo
- General concepts
- Github working principle
- Clone
- Push & Pull
- Gitignore
- Merge & Conflicts

Account creation

Github.com



Account creation



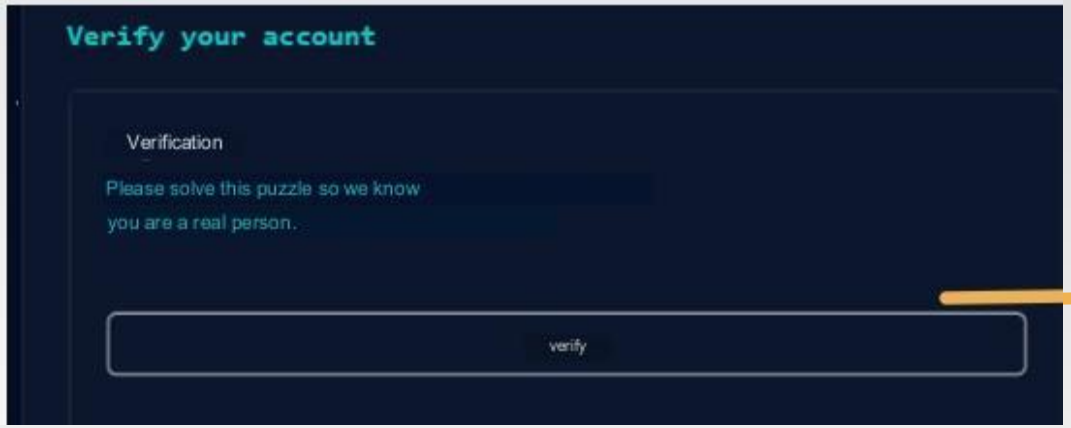
Enter your e-mail address

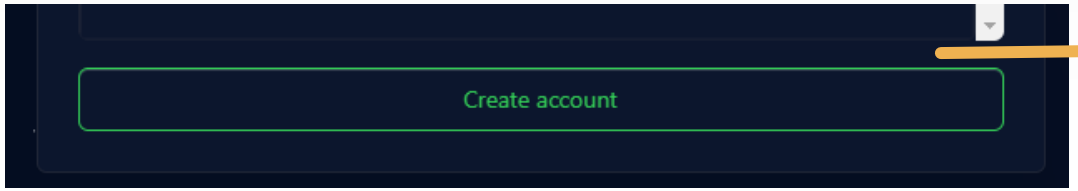
Set password

Set a username

We write **"n"** if we do not want to be notified of product updates and promotions via email.

Account creation



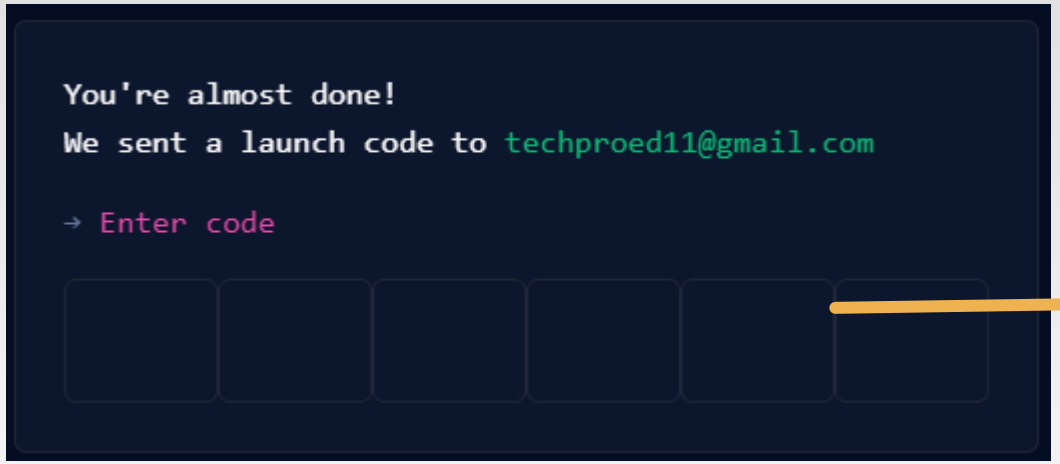


Press the verify button

Process the verification steps

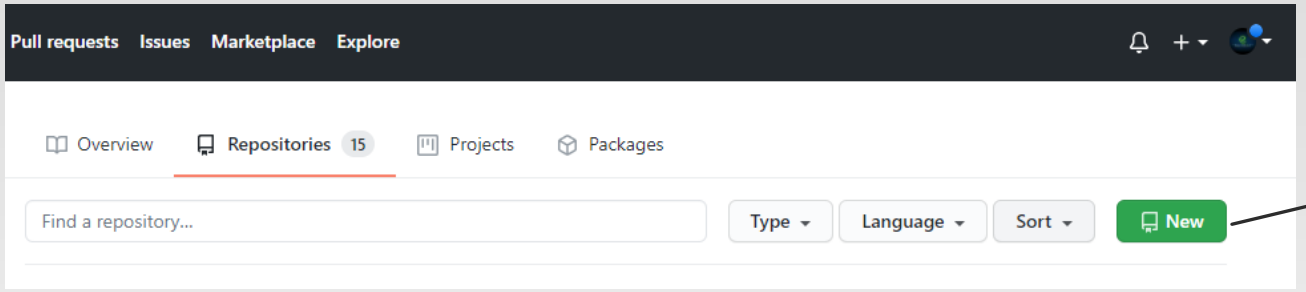
Press the Create Account button

Account creation

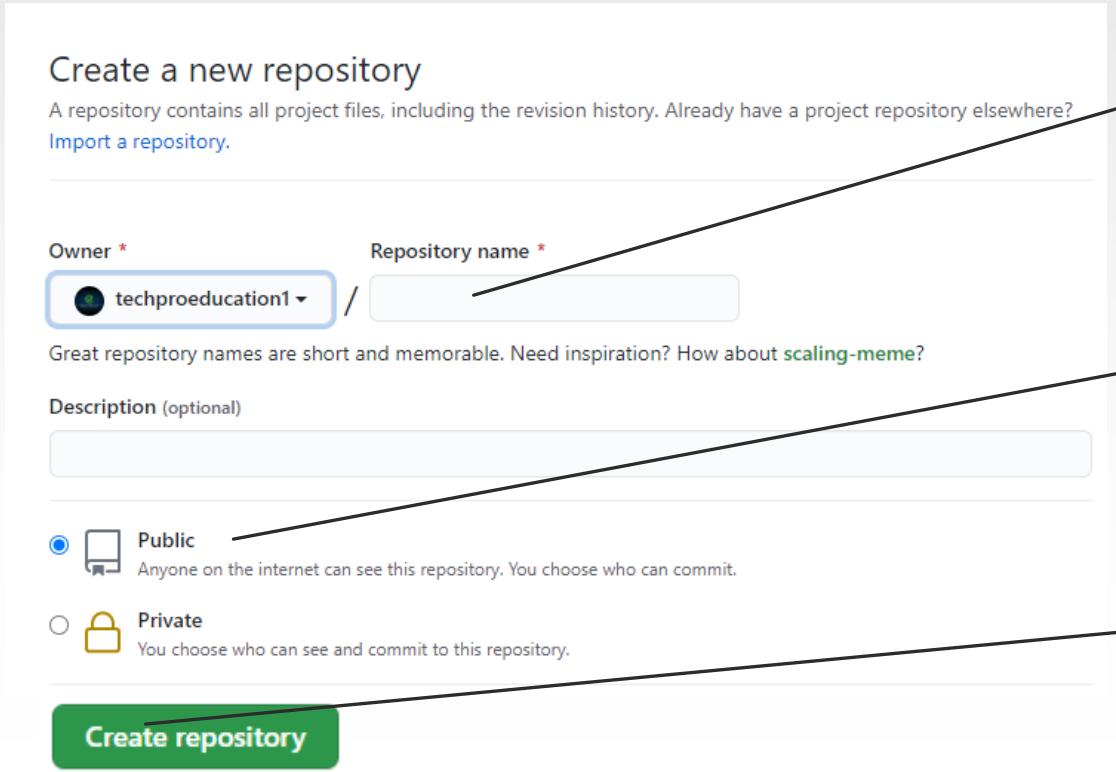


Complete the process by entering the code sent to the e-mail address.

Creating a Github repo



1 Press "New" button



2 Give a name for the repository

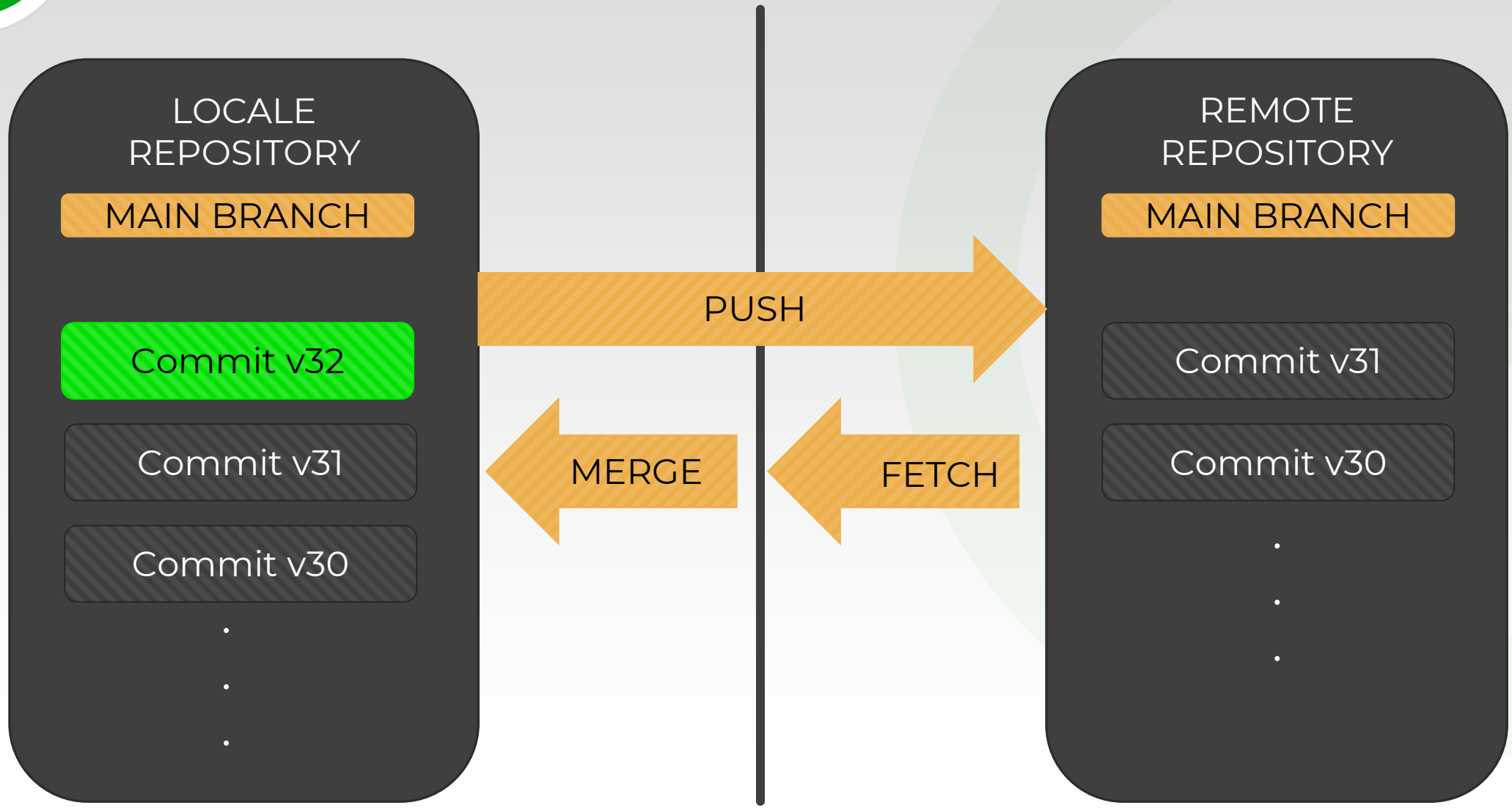
3 Is it accessible to everyone, or only to users we designate?

4 Press "Create repository" button

Concepts

Clone	The process of downloading a repo from Github to the local
Push	The process of sending locally created commits to Github.
Fetch	The process of downloading the changes, if any, by comparing the latest version in Github with the local one.
Merge	The process of applying the downloaded changes to the local
Pull	Makes the Fetch and Merge processes alone

Github Working Principle



Cloning

git clone

The process of downloading a repo from Github to the local is called cloning.
Private repos that have the necessary permissions or public repos can be cloned.

The **git clone** command is used for this.

Git Push

If the local repo was created by **cloning**

```
git add .
git commit –m "version name"
git push
```

Used to associate local git repo with remote github repo

If the local repo was created with **git init**

```
git add .
git commit –m "version name"
git remote add origin [remote url]
git push -u origin [branch name]
```

This is how the first push is done. Then just **git push** is enough.

Pulling a commit from Github

If you want to update the local repo via Github, the following commands are used

git fetch

Downloads changes from remote to local

git merge

Applies the downloaded changes to the local repo

OR

git pull

fetch & merge

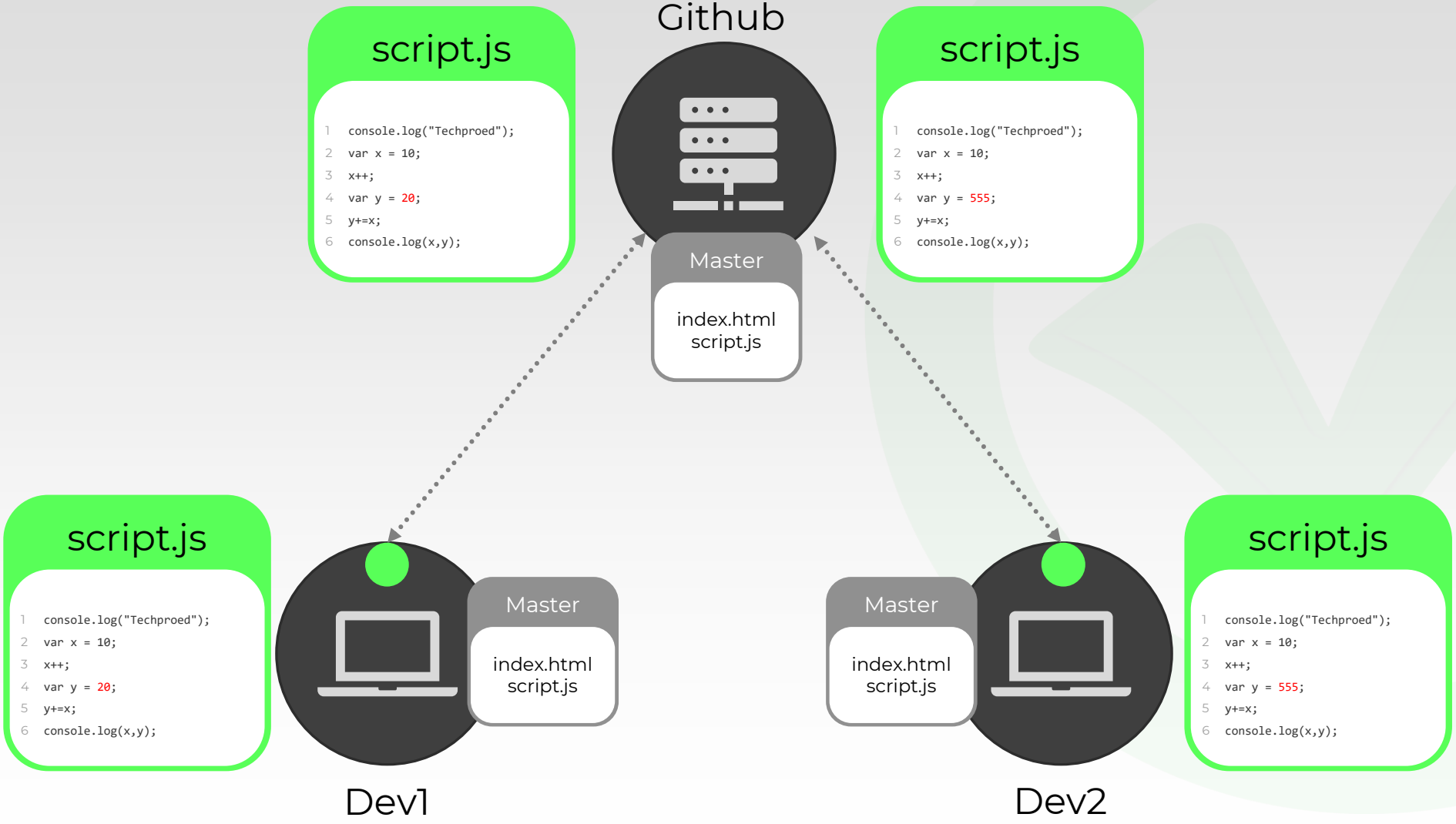
.gitignore

The **.gitignore** file specifies intentionally untracked files that Git should ignore.

```
out/
.idea/
.idea_modules/
*.iml
*.ipr
*.iws
```

.gitignore

Merge Conflict



Merge Conflict

script.js

```
1  console.log("Techproed");
2  var x = 10;
3  x++;
4  var y = 20;
4  var y = 555;
5  y+=x;
6  console.log(x,y);
```

```
Auto-merging script.js
CONFLICT (content): Merge conflict in script.js
Automatic merge failed; fix conflicts and then
commit result.
```

```
<<<<<<<<<<<<<<<  HEAD (Current Change)
 var y = 20;
==================
>>>>>>>>>>>>>>>> Dev2 (Incoming change)
 var y = 555;
==================
```

Git – Github Work Cycle

BEST PRACTISE