

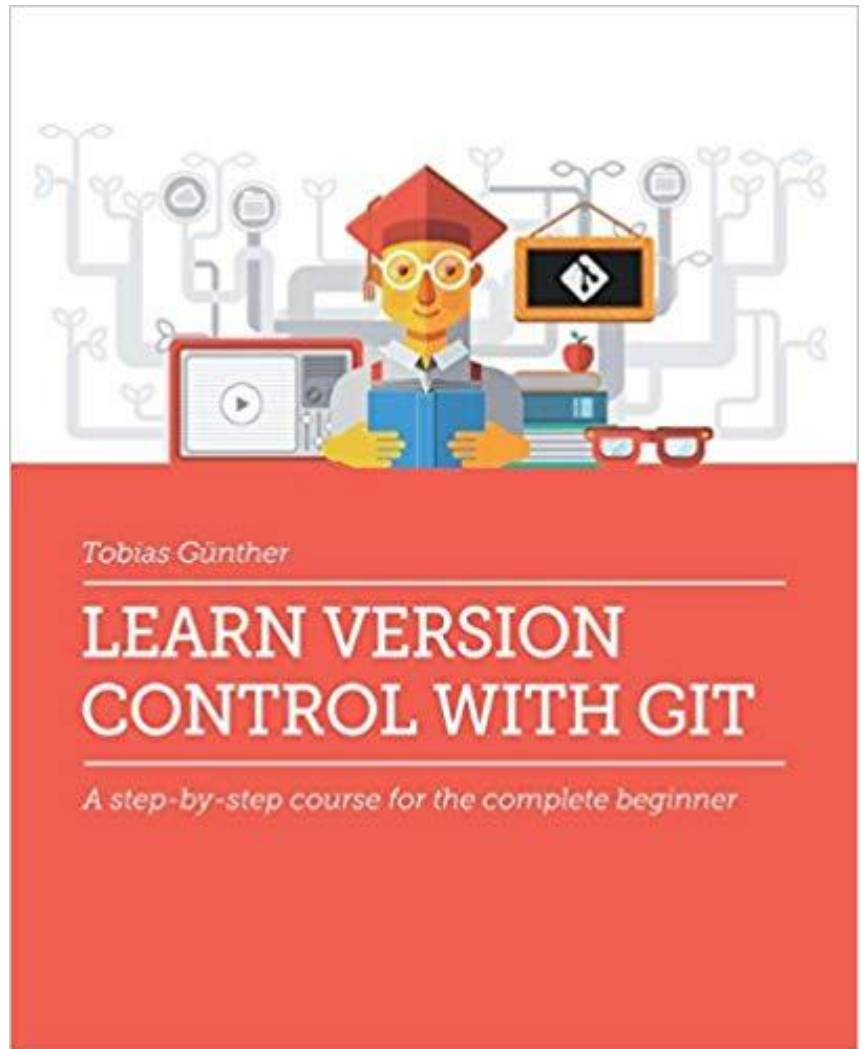


# git

Zeeshan Hanif

Director/CTO Panacloud  
PIAIC

Book we will  
follow



# What is Version Control System?

- ❖ A software utility that tracks and manages changes to a filesystem.
- ❖ Also known as revision control or source control system
- ❖ It's a management of changes to documents, computer programs, large websites and other collection of information

# What is Version Control System?

- ❖ A VCS also offers collaborative utilities to share and integrate these filesystem changes to other VCS users.
- ❖ VCS will track the addition, deletion, and modification actions applied to files and directories
- ❖ A repository is a VCS term which describes when VCS is tracking a filesystem.

# What is Version Control System?

- ❖ VCS options include Git, Mercurial, SVN and preforce.
- ❖ You can think of a VCS as a kind of “database”.
- ❖ It lets you save a snapshot of your complete project at any time you want
- ❖ When you later take a look at an older snapshot your VCS shows you exactly how it differed from the previous one



Time



Your  
Project



VCS



# What is Version Control System?

- ❖ Version control is independent of the kind of project / technology / framework you're working with
- ❖ It works just as well for an HTML website as it does for a design project or an iPhone app
- ❖ It lets you work with any tool you like; it doesn't care what kind of text editor, graphics program, file manager or other tool you use

# Why Use a Version Control System?

## ❖ Collaboration

- Without a VCS in place, you're probably working together in a shared folder on the same set of files.
- And you have to coordinate with others so that they don't work on same file, it will be very difficult to manage
- With a VCS, everybody on the team is able to work absolutely freely - on any file at any time.
- The VCS will later allow you to merge all the changes into a common version.



# Why Use a Version Control System?

## ❖ Storing Versions (Properly)

- Saving a version of your project after making changes is an essential habit. But without a VCS, this becomes tedious and confusing very quickly:
  - How much do you save? Only the changed files or the complete project?
  - How do you name these versions? “myapp\_2013-11-12\_v23”
  - The most important question: How do you know what exactly is different in these versions?

# Why Use a Version Control System?

## ❖ Storing Versions (Properly) (cont...)

- Version control system acknowledges that there is only one project
- Therefore, there's only the one version on your disk that you're currently working on
- Everything else - all the past versions and variants - are neatly packed up inside the VCS
- When you need it, you can request any version at any time and you'll have a snapshot of the complete project right at hand.

# Why Use a Version Control System?

## ❖ Restoring Previous Versions

- Being able to restore older versions of a file or whole project
- If the changes you've made
- lately prove to be garbage, you can simply undo them in a few clicks

# Why Use a Version Control System?

## ❖ Understanding What Happened

- Every time you save a new version of your project, your VCS requires you to provide a short description of what was changed
- Additionally (if it's a code / text file), you can see what exactly was changed in the file's content

# Why Use a Version Control System?

## ❖ Backup

- A side-effect of using a distributed VCS like Git is that it can act as a backup
- every team member has a full-blown repository of the project on his disk
- If central server break down (and your backup drives fail), all you need for recovery is one of your teammates' local Git repository.

# Different Types Of Version Control Systems

❖ VSS

❖ SVN

❖ GIT

# VSS

Microsoft Visual SourceSafe (VSS) is a source control program, oriented towards small software development projects.

# SVN

SVN is the abbreviated form of “Apache Subversion” and is a popular version control system tool. It is a centralized version control system



# GIT

Git is a distributed version-control system for tracking changes in source code during software development. It is designed for coordinating work among programmers, but it can be used to track changes in any set of files.

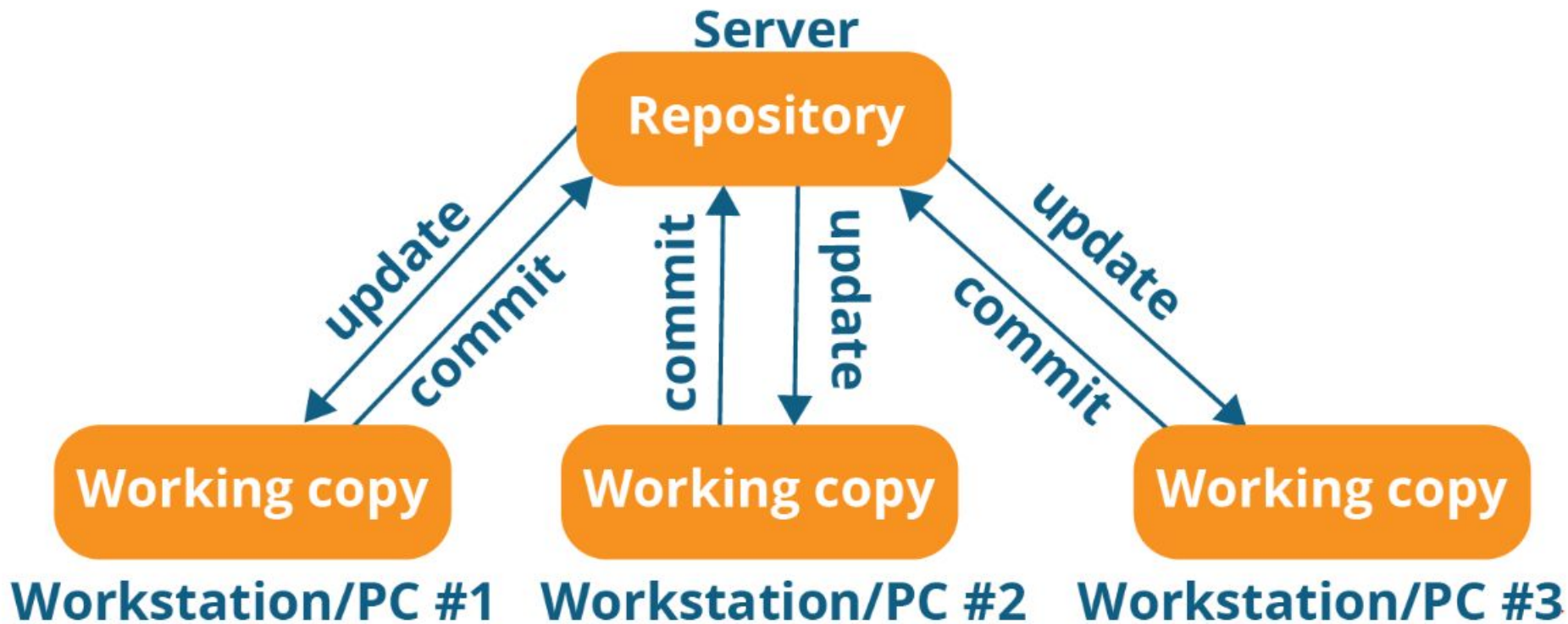
# Types of Version Control Systems

- ❖ Centralized Version Control System (CVCS)
- ❖ Distributed Version Control System (DVCS)

# Centralized Version Control System (CVCS)

Centralized version control system (CVCS) uses a central server to store all files and enables team collaboration. It works on a single repository to which users can directly access a central server.

## Centralized version control system



# Centralized Version Control System (CVCS)

Every programmer can extract or update their workstations with the data present in the repository or can make changes to the data or commit in the repository. Every operation is performed directly on the repository.

# CVCS Drawbacks

- ❖ It is not locally available; meaning you always need to be connected to a network to perform any action.
- ❖ Since everything is centralized, in any case of the central server getting crashed or corrupted will result in losing the entire data of the project.

# Distributed Version Control System (DVCS)

These systems do not necessarily rely on a central server to store all the versions of a project file.

# Distributed Version Control System (DVCS)

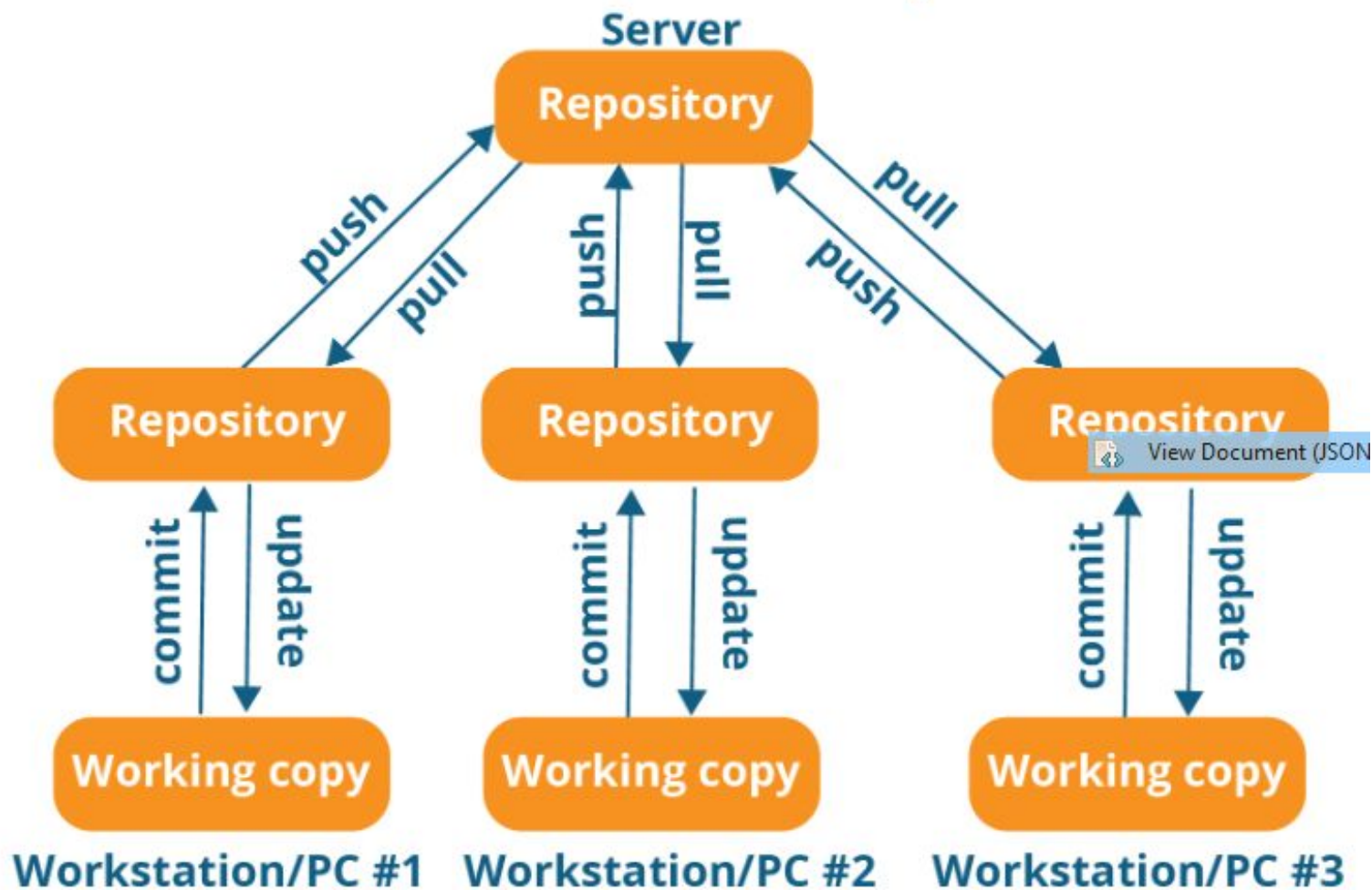
In Distributed VCS, every contributor has a local copy or “clone” of the main repository i.e. everyone maintains a local repository of their own which contains all the files and metadata present in the main repository.



# Distributed Version Control System (DVCS)

GIT is Distributed version  
Control System

# Distributed version control system



# Distributed Version Control System (DVCS)

Every programmer maintains a local repository on its own, which is actually the copy or clone of the central repository on their hard drive. They can commit and update their local repository without any interference

# Distributed Version Control System (DVCS)

Programmer can update their local repositories with new data from the central server by an operation called “**pull**” and affect changes to the main repository by an operation called “**push**” from their local repository.

# Distributed Version Control System (DVCS)

## Advantages

All operations (except push & pull) are very fast because the tool only needs to access the hard drive, not a remote server. Hence, you do not always need an internet connection.

# Distributed Version Control System (DVCS)

## Advantages

Committing new change-sets can be done locally without manipulating the data on the main repository. Once you have a group of change-sets ready, you can push them all at once.

# Distributed Version Control System (DVCS)

## Advantages

Since every contributor has a full copy of the project repository, they can share changes with one another if they want to get some feedback before affecting changes in the main repository.

# Distributed Version Control System (DVCS)

## Advantages

If the central server gets crashed at any point of time, the lost data can be easily recovered from any one of the contributor's local repositories.



# Git

Git is a distributed version-control system for tracking changes in source code during software development

Everything we learned about Distributed version control system is applicable on GIT

# Installing and Setting Up Git

- ❖ Go to <https://git-scm.com/downloads>
- ❖ Installation Guide:
  - <https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>
- ❖ For Linux:
  - <https://git-scm.com/download/linux>
- ❖ For Windows:
  - <https://git-scm.com/download/win>

# Installing and Setting Up Git

- ❖ For windows after download
  - Double click on exe file and follow the installation wizard
  - Do not change anything just keep the default setting and press next on every step
  - It will install git on your machine

# Installing and Setting Up Git

- ❖ Open Terminal and run following commands to setup your name and email id
  - `git config --global user.name "Your Name"`
  - `git config --global user.email "Your Email"`

# Installing and Setting Up SmartGit GUI Tool

- ❖ Go to <https://www.syntevo.com/smartgit/download>
- ❖ Download SmartGit
- ❖ Install SmartGit with all default settings

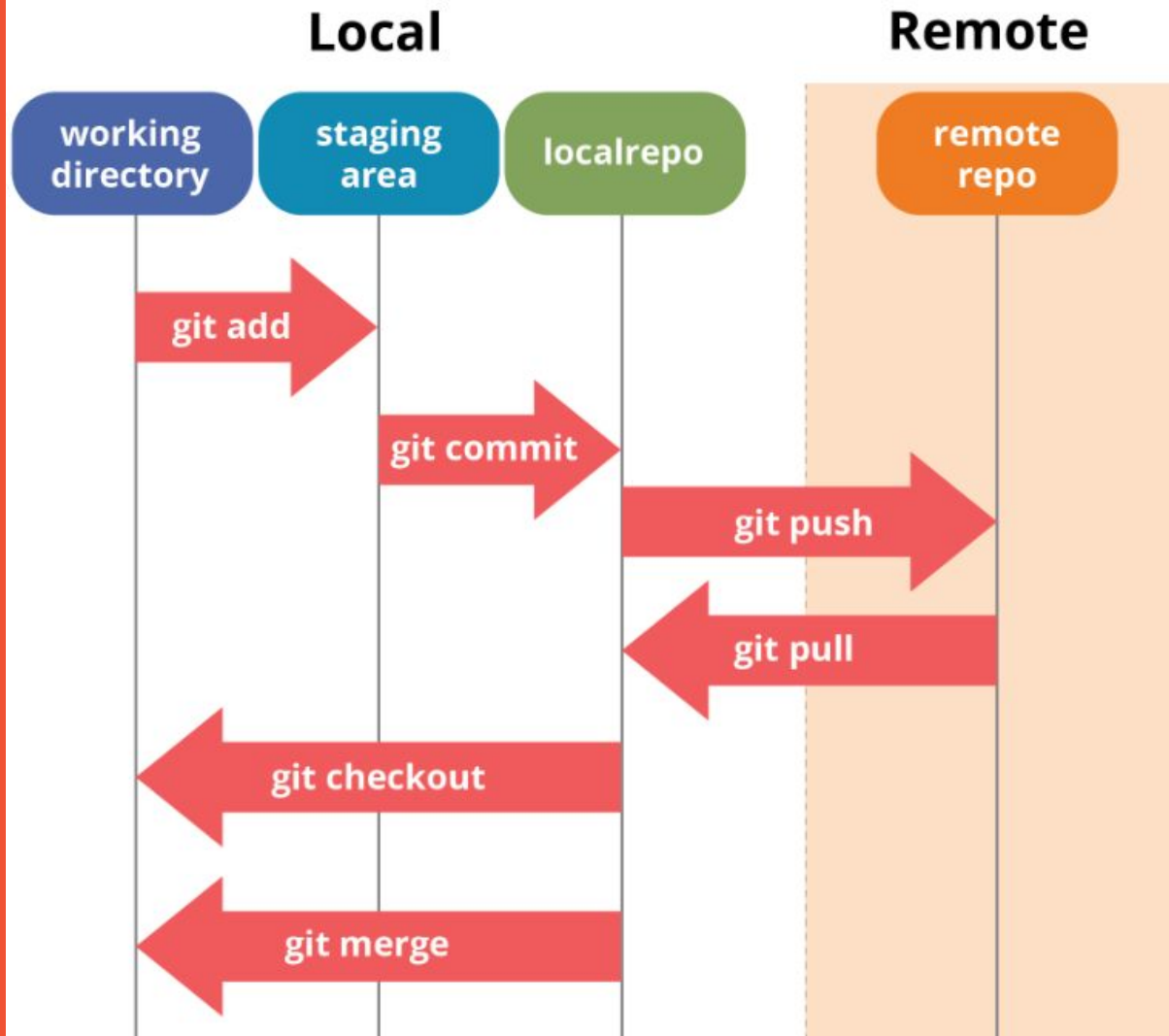
# Basic operations in Git are

- ❖ Initialize
- ❖ Add
- ❖ Commit
- ❖ Pull
- ❖ Push

# Advanced Git operations

- ❖ Branching
- ❖ Merging
- ❖ Rebasing

# Git Operations





# Important Terms

## 1. Repository

- ❖ Think of a repository as a kind of database where your VCS stores all the versions and metadata that accumulate in the course of your project.
- ❖ In Git, the repository is just a simple hidden folder named “.git” in the root directory of your project

# Important Terms

## 2. Working Directory

The root folder of your project is often called the “working copy” (or “working directory”). It’s the directory on your local computer that contains your project’s files.

# Important Terms

## 3. File Status

# File Status Lifecycle

untracked

unmodified

modified

staged

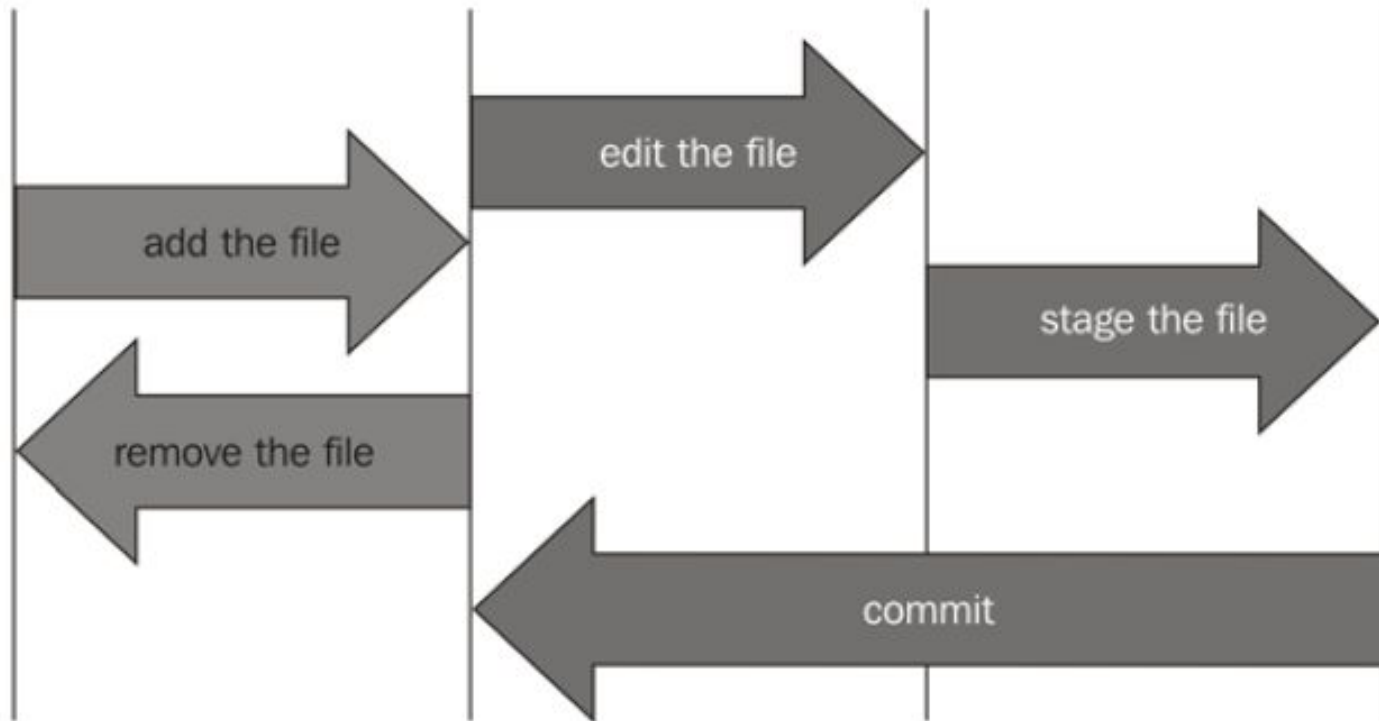
add the file

edit the file

stage the file

remove the file

commit



# Important Terms

## 4. Staging Area

Staging area is a virtual place that collects all the files you want to include in the next commit

In Git, simply making some changes doesn't mean they're automatically committed.

# Important Terms

## 4. Staging Area

Every commit is “hand-crafted”: each change that you want to include in the next commit has to be marked explicitly (“added to the Staging Area” or, simply put, “staged”)

## Working Copy

Your Project's Files



Git watches tracked files  
for new local modifications...

## Staging Area

Changes included in  
the Next Commit

## Local Repository

The ".git" Folder

### Tracked (and modified)



If a file was modified since it  
was last committed, you can  
stage & commit these changes

stage



Changes that were added to  
the Staging Area will be  
included in the next commit

commit



All changes contained in a  
commit are saved in the local  
repository as a new revision



Changes that are **not staged** will  
not be committed & remain as  
local changes until you stage &  
commit or discard them

### Untracked



Changes in untracked files aren't  
watched. If you want them included  
in version control, you have to tell  
Git to start tracking them. If not, you  
should consider ignoring them.

# Basic Workflow



# Starting with an Unversioned, Local Project

1. Open terminal and create directory on your machine
  - a. `C:\Repo\myproject`
2. Go into directory in terminal
3. Initialize repository in this directory
  - a. `git init`
  - b. This will create `.git` hidden folder in your directory which will make your current folder, a git repository

# Starting with an Unversioned, Local Project

4. Create first.txt and second.txt
5. Check status, it will show you two untracked files
  - a. `git status`
6. Add these files to staging area
  - a. Two ways to add files in staging
    - i. `git add first.txt second.txt` OR
    - ii. `git add .`

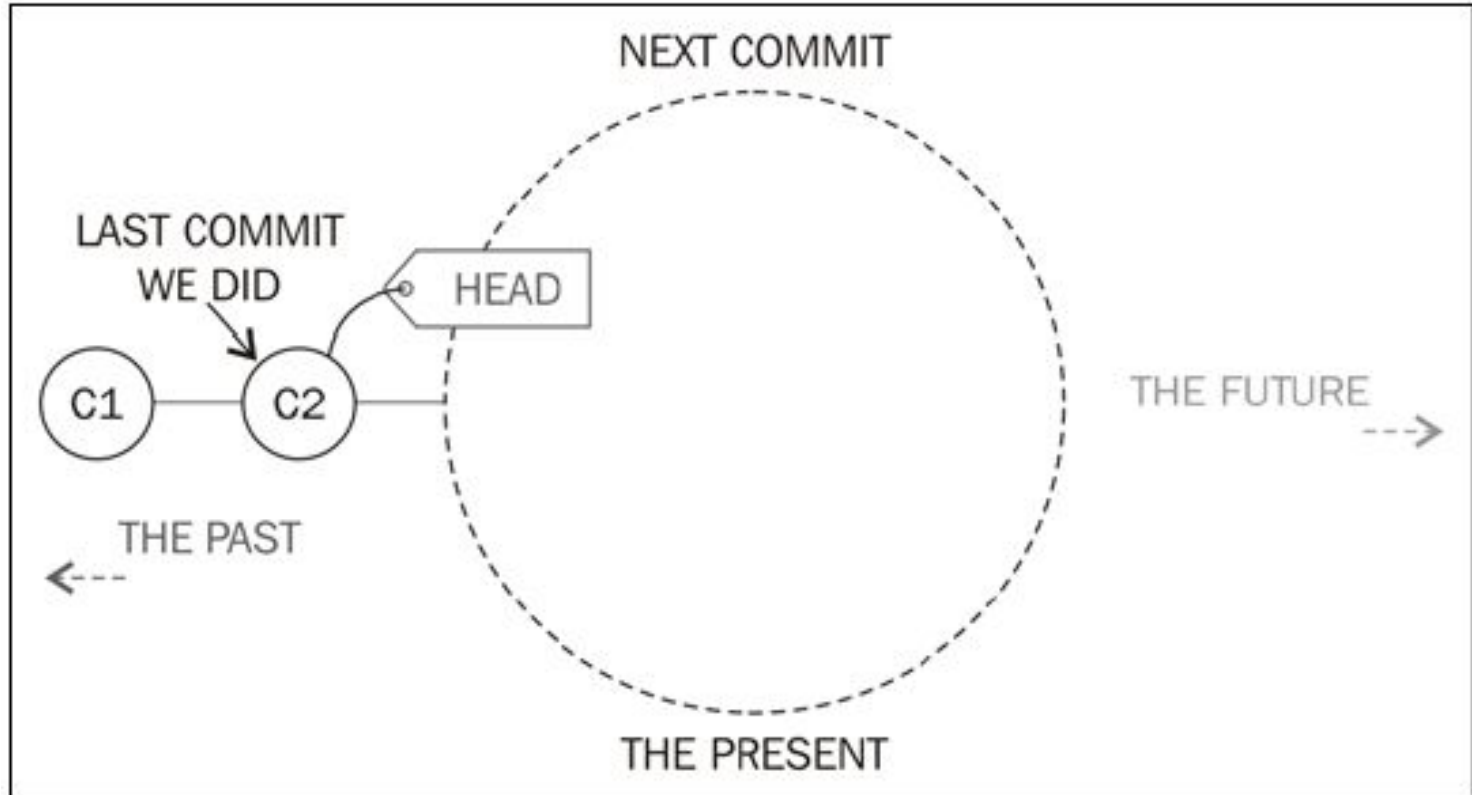
# Starting with an Unversioned, Local Project

4. Commit your files to VCS with commit message
  - a. `git commit -m "implemented new feature"`
5. Commit message is very important, you should provide proper commit message so that it can be refer back to identify what was added in that commit
6. Check logs to see commit history
  - a. `git log`

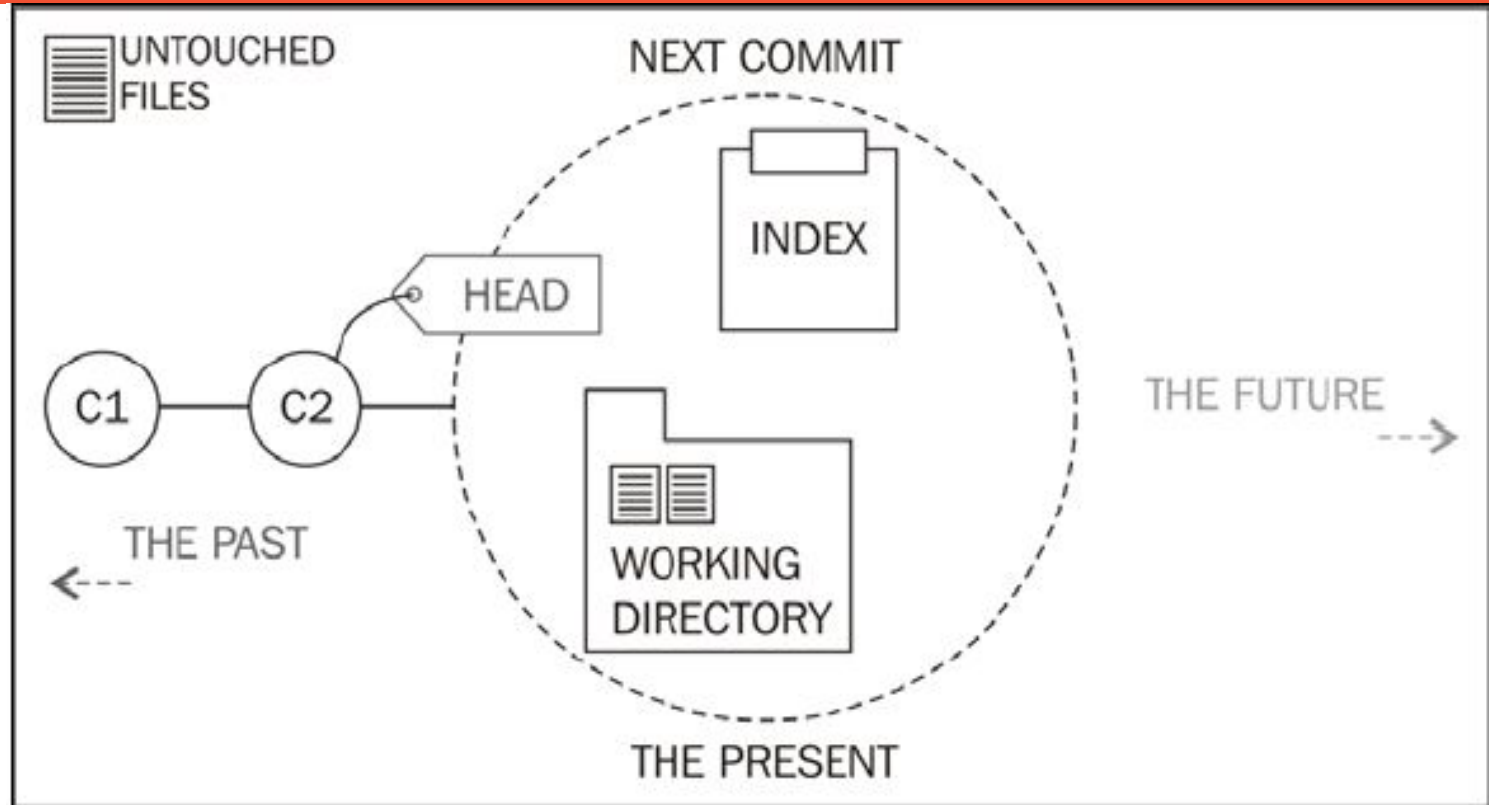
# Demo with Terminal

# Demo with Smartgit UI

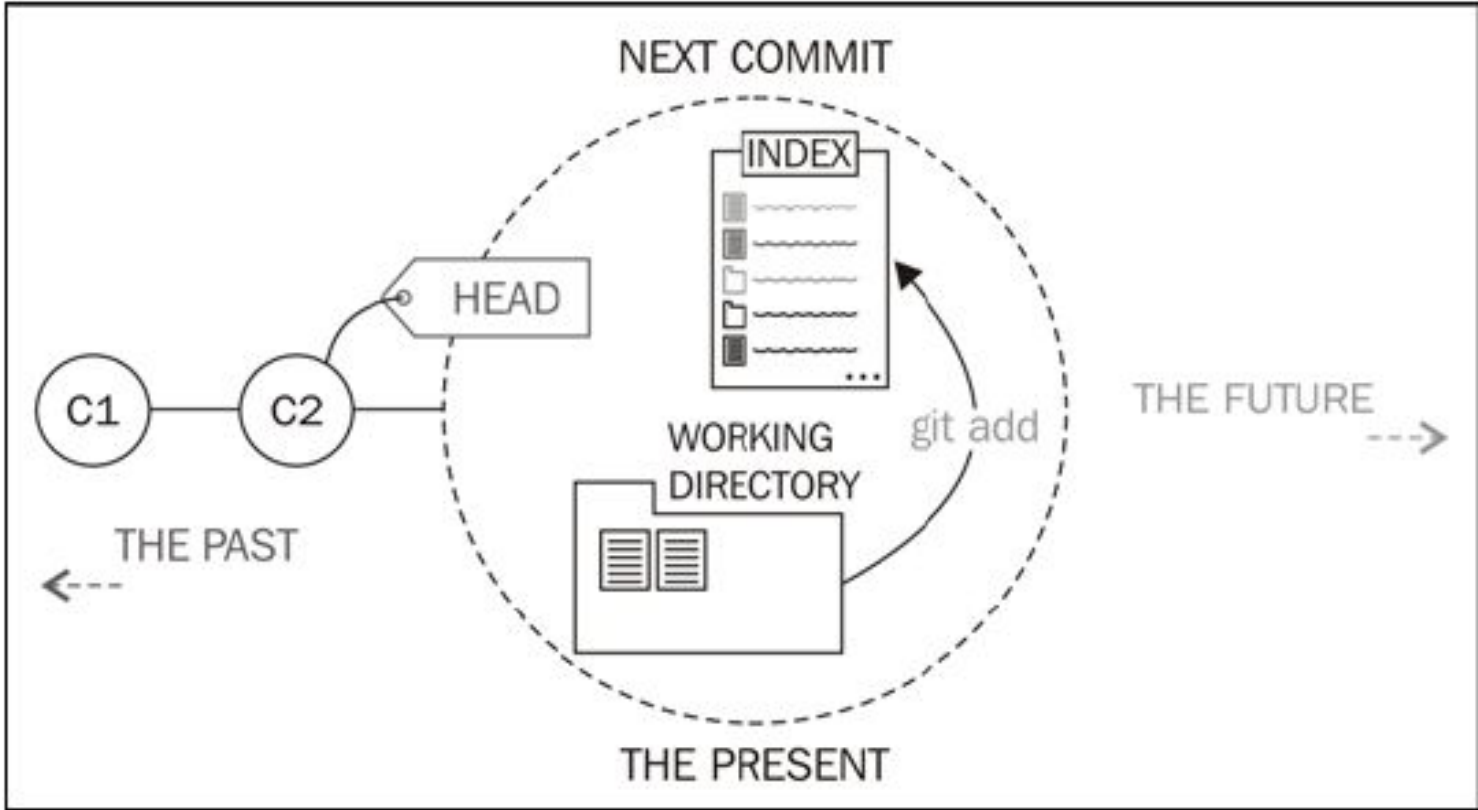
# HEAD shows last commit



# New and changed files will be part of next commit

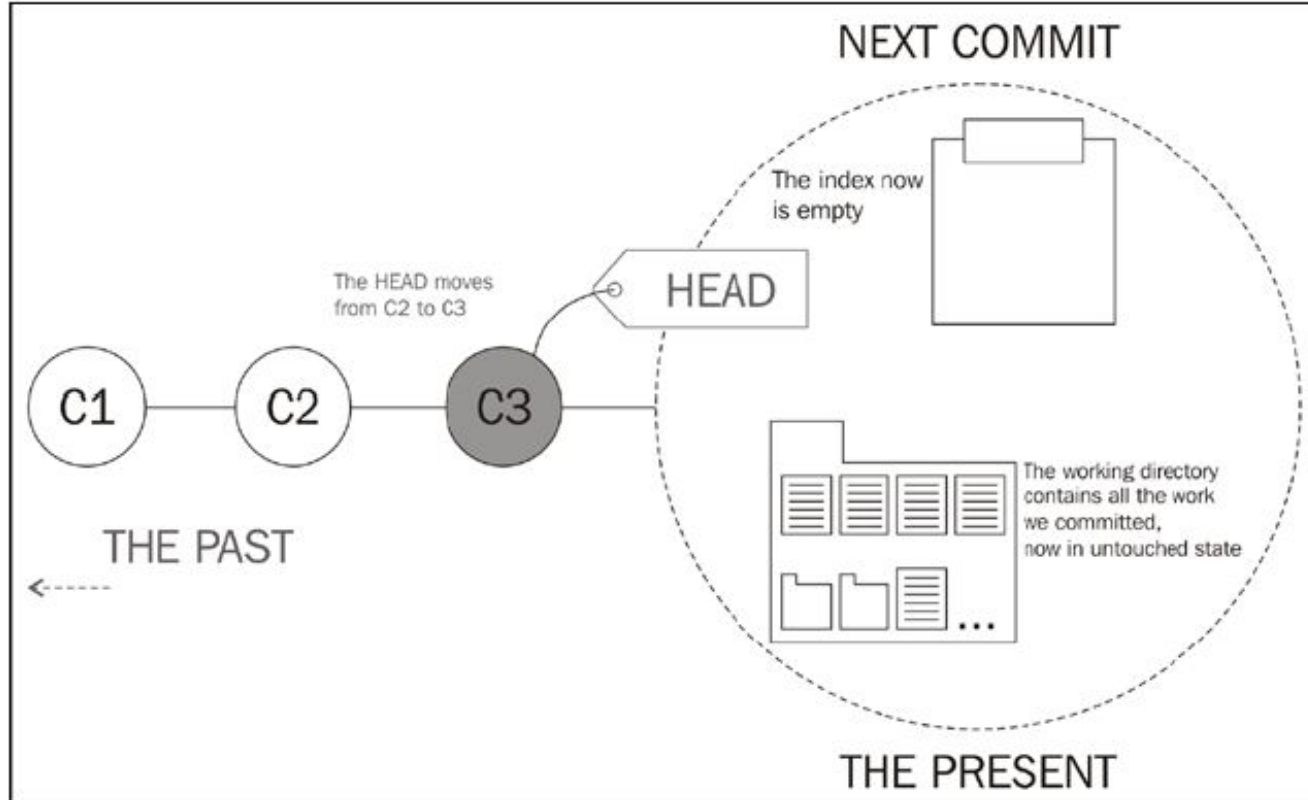


# git add command add files in staging area





# After commit, our new commit will be HEAD



# Commit Hash

- ❖ Every commit has a unique identifier: a 40-character checksum called the “commit hash”.

# Commit Hash

- ❖ As multiple people can work in parallel, committing their work offline, without being connected to a shared repository
- ❖ Therefore commit Hash helps in identifying which user made specific commit

# git reset

Unstage files or  
remove change

- ❖ git reset command will remove file from staging area
- ❖ git reset can remove changes in files if they are not committed

# git reset

Unstage files or  
remove change

- ❖ git rest
- ❖ git reset --hard

# Demo with Terminal

# Demo with Smartgit UI

# Ignoring Files

- ❖ Typically, in every project and on every platform, there are a couple of files that you don't want to be version controlled
- ❖ E.g
  - .DS\_Store
  - Node\_modules
  - Build
  - logs



# Ignoring Files

- ❖ Create an empty file in your favorite editor and save it as “.gitignore” in your project’s root folder.
- ❖ You can define rules in “.gitignore” file to ignore files
- ❖ Add file or directory path or extension or name

# Ignoring Files -- Examples

- ❖ Ignore one specific file
  - `path/to/file.ext`
- ❖ Ignore all files with a certain name (anywhere in the project)
  - `filename.ext`
- ❖ Ignore all files of a certain type (anywhere in the project)
  - `*.ext`
- ❖ Ignore all files in a certain folder:
  - `path/to/folder/*`

# Demo with Terminal

# Demo with Smartgit UI

# Branching and Merging

# Why branches are important

- ❖ In every project, there are always multiple different contexts where work happens
- ❖ Each feature, bugfix, experiment, or alternative of your product is actually a context of its own
- ❖ There can be unlimited amount of different contexts
- ❖ Most likely, you'll have at least one context for your "main" or "production" state, and another context for each feature, bugfix, experiment, etc.

# Why branches are important

- ❖ In real-world projects, work always happens in multiple of these contexts in parallel:
  - While you're preparing two new variations of your website's design (context 1 & 2)
  - you're also trying to fix an annoying bug (context 3).
  - On the side, you also update some content on your FAQ pages (context 4)
  - one of your teammates is working on a new feature for your shopping cart (context 5)
  - and another colleague is experimenting with a whole new login functionality (context 6).

# Why branches are important

## ❖ Another example

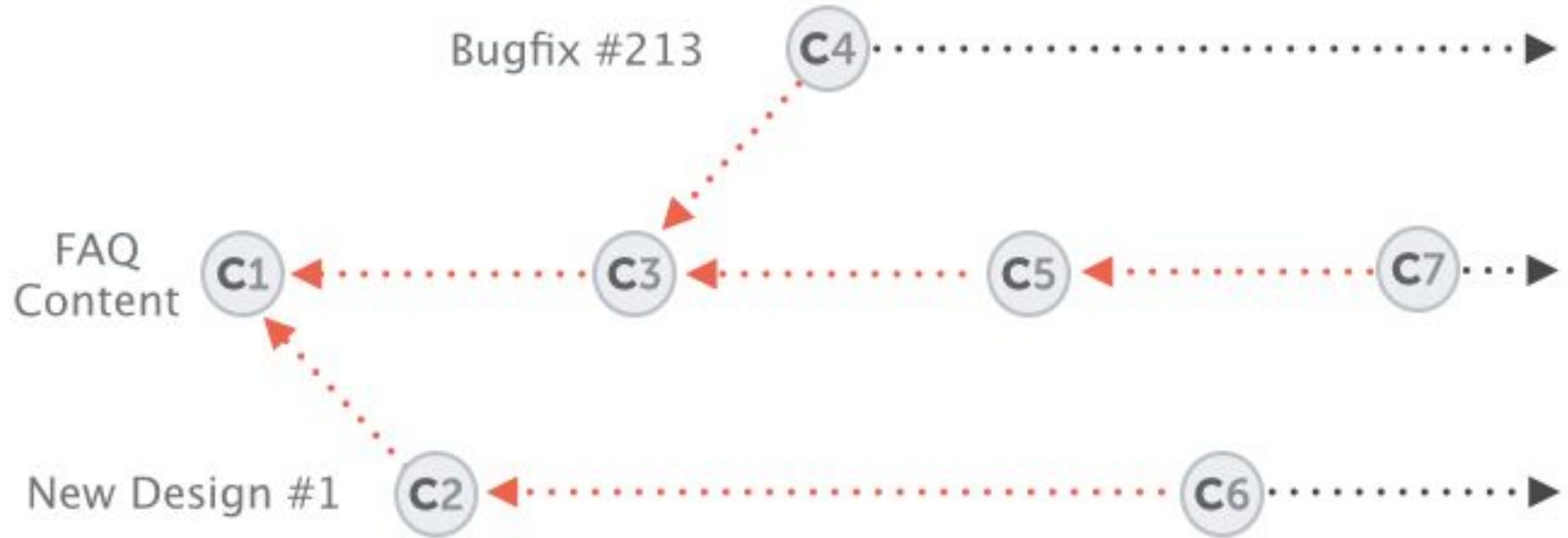
- Production, development and feature context
- Production code is tested and deployed and we don't want any problem in that
- Development team is working on new feature and we don't want that feature to effect production code until it is properly tested
- All feature combines in development to have final testing before deployment



# Branches to the Rescue

- ❖ Branches are what we need to solve context problems.
- ❖ Because a branch represents exactly such a context in a project and helps you keep it separate from all other contexts.

# Branches to the Rescue



# Branches to the Rescue

- ❖ All the changes you make at any time will only apply to the currently active branch; all other branches are left untouched
- ❖ This gives you the freedom to both work on different things in parallel and, above all, to experiment - because you can't mess up!
- ❖ In case things go wrong you can always go back / undo / start fresh / switch contexts

# Working with Branches

- ❖ Without knowing, we were already working on a branch.
- ❖ This is because branches aren't optional in Git: you are always working on a certain branch (the currently active, or “checked out”, or “HEAD” branch).
- ❖ Check ‘git status’ and it will show you current branch
- ❖ The “master” branch was created by Git automatically for us when we started the project.

# Working with Branches

- ❖ 'master' branch does not mean anything special
- ❖ It is initially created when we start project
- ❖ 'HEAD' always represent current branch

# Branch Commands

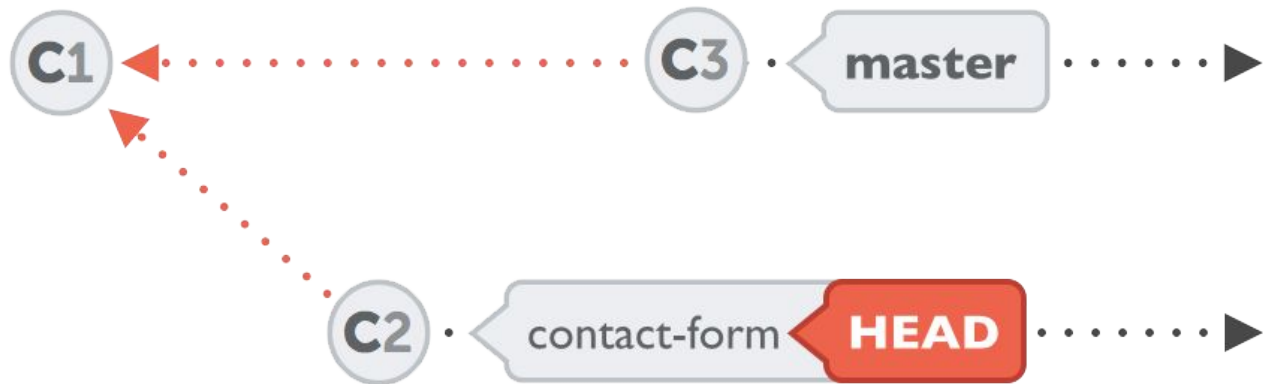
- ❖ `git branch`
  - Show list of branches
- ❖ `git branch -v`
  - Show list of branches with some details
- ❖ `git branch new-dev`
  - Creates new branch with name 'new-dev'
- ❖ `Git checkout new-dev`
  - Switch to 'new-dev' branch

# Branch Commands

- ❖ `git merge new-dev`
  - Merge 'new-dev' branch into current active branch
- ❖ `git log new-dev..master`
  - Show commit difference in two branches

# git checkout contact-form

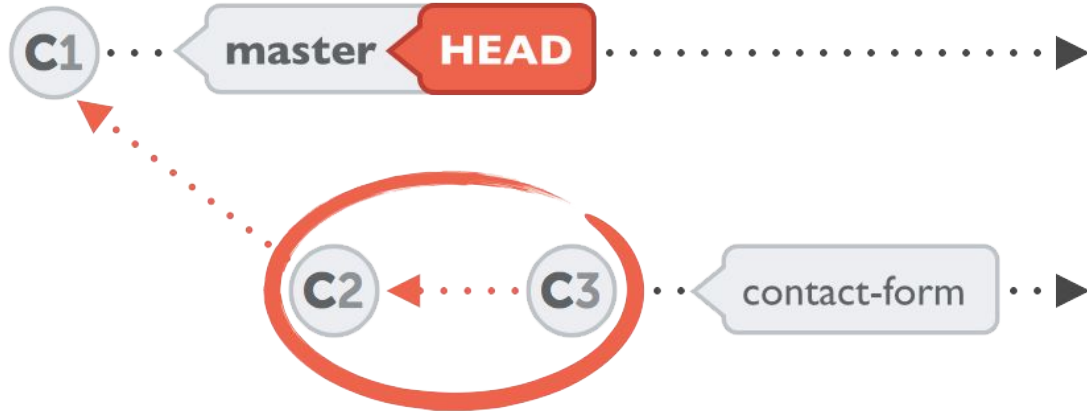
- ❖ git checkout move you to other branch and HEAD represent that branch





# Merging Changes

- ❖ You can merge all your commits in your branch into another branch by git merge command



# Demo with Terminal

# Demo with Smartgit UI

# Stash

- ❖ Commit wraps up changes and saves them permanently in the repository
- ❖ Stash save changes **temporarily**

# Stash

- ❖ If you want to switch to another branch without committing changes in current branch
- ❖ You will need to save your changes somewhere so that you have clean working directory
- ❖ Stash will work in this case

# Stash

- ❖ Later, at any time, you can restore the changes from stash in your working copy - and continue working where you left off.
- ❖ You can create as many Stashes as you want

# Stash Commands

- ❖ `git stash`
  - Save local changes  
stash clipboard
- ❖ `git stash save <name>`
  - Save local changes in  
stash clipboard with  
the name provided in  
command
- ❖ `git stash list`
  - Show list of stashes

# Stash Commands

- ❖ `git stash pop`
  - Apply latest stash and remove it from clipboard
- ❖ `git stash apply stashname`
  - Apply specific stash and that stash will remain saved in clipboard



# Demo with Terminal

# Demo with Smartgit UI

# Short-Lived / Topic Branches

- ❖ They are about a single topic or feature and once work done with them they are removed

# Long-Running Branches

- ❖ Branch that are higher level and independent of any feature, remain for longer time
- ❖ They represent states in your project lifecycle - like a “production”, “testing”, or “development” state

# Remote Repositories

# Remote Repositories

- ❖ About 90% of version control related work happens in the local repository: staging, committing, viewing the status or the log/history, etc.
- ❖ If you're the only person working on your project, chances are you'll never need to set up a remote repository.
- ❖ Only when it comes to sharing data with your teammates, a remote repo comes into play.

# Remote Repositories

- ❖ Think of it like a "file server" that you use to exchange data with your colleagues.
- ❖ Local repositories reside on the computers of team members. In contrast, remote repositories are hosted on a server that is accessible for all team members - most likely on the internet or on a local network.

# Remote Repositories

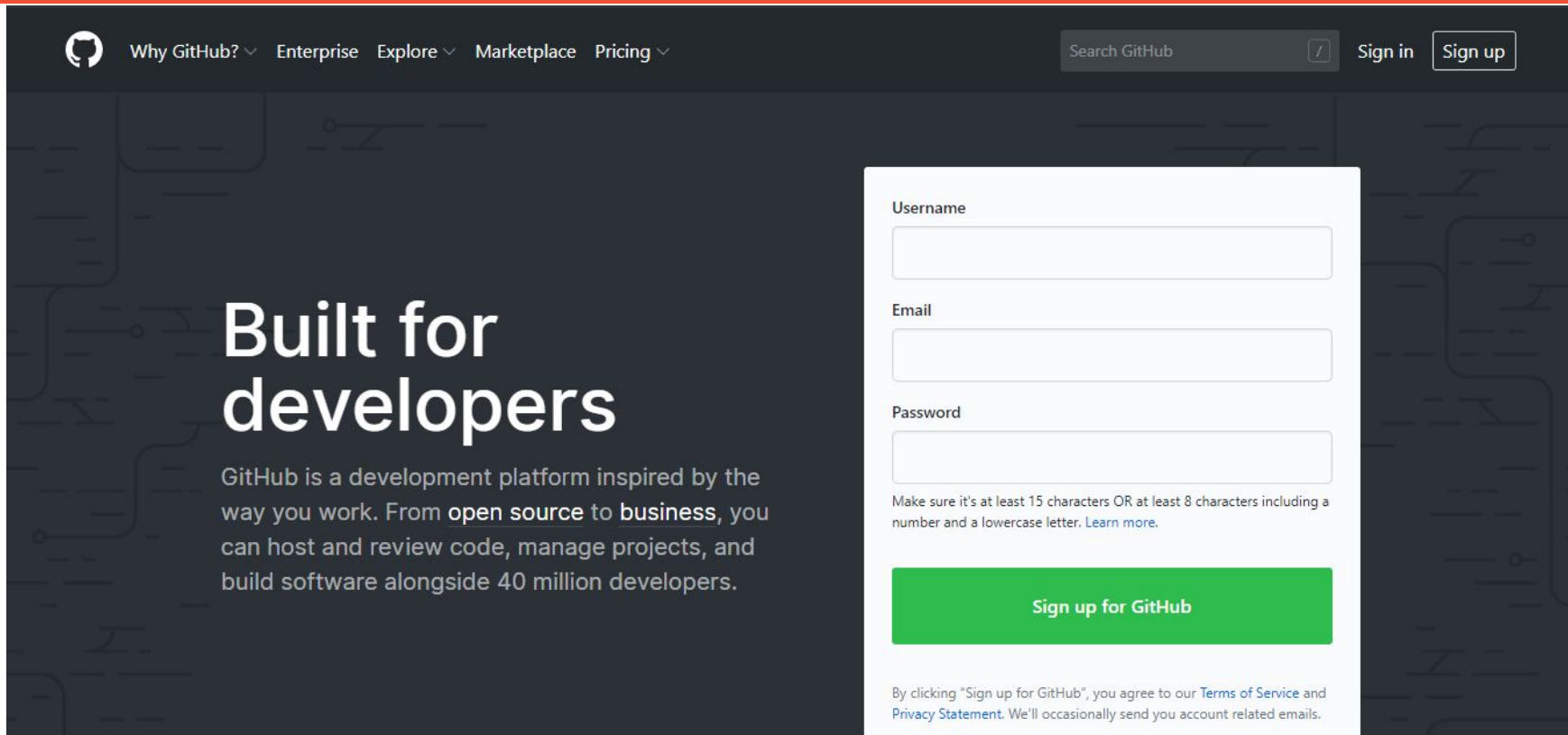
- ❖ For remote repository you have to first select the online service that offer git and allow you to create remote repository
- ❖ Following are few online services:
  - GitHub (<https://github.com>)
  - BitBucket (<https://bitbucket.org>)
  - GitLab (<https://gitlab.com>)
  - Many others







- ❖ Github is online service for git to create remote repositories on github's server and collaborate with teammates/colleague

# Setting up a new GitHub account

A screenshot of the GitHub website's sign-up page. The background is dark with a faint, light-colored circuit-like pattern. On the left, the GitHub logo is in the top left corner. Navigation links include 'Why GitHub?', 'Enterprise', 'Explore', 'Marketplace', and 'Pricing'. A search bar is in the top right, followed by 'Sign in' and 'Sign up' buttons. The main heading 'Built for developers' is in large white text. Below it, a paragraph describes GitHub as a development platform. On the right, a white sign-up form contains fields for 'Username', 'Email', and 'Password'. A green button labeled 'Sign up for GitHub' is below the form. At the bottom, a disclaimer states that clicking the button agrees to the Terms of Service and Privacy Statement.

 Why GitHub? ▾ Enterprise Explore ▾ Marketplace Pricing ▾

Search GitHub 

Sign in [Sign up](#)

## Built for developers

GitHub is a development platform inspired by the way you work. From [open source](#) to [business](#), you can host and review code, manage projects, and build software alongside 40 million developers.

**Username**

**Email**

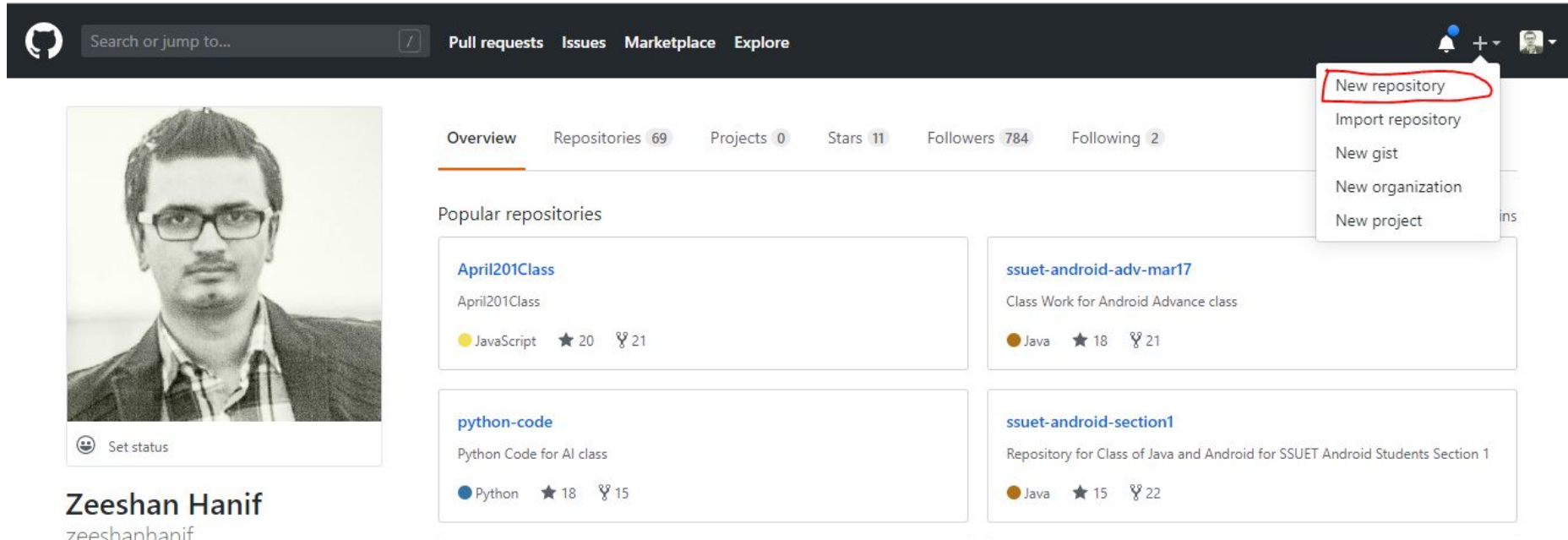
**Password**

Make sure it's at least 15 characters OR at least 8 characters including a number and a lowercase letter. [Learn more](#).

[Sign up for GitHub](#)

By clicking "Sign up for GitHub", you agree to our [Terms of Service](#) and [Privacy Statement](#). We'll occasionally send you account related emails.

# Create New Repository



The screenshot shows the GitHub profile of Zeeshan Hanif. The profile includes a profile picture, a bio, and a list of popular repositories. A dropdown menu is open in the top right corner, showing options to create a new repository, import a repository, create a new gist, create a new organization, or create a new project. The 'New repository' option is highlighted with a red circle.

**Search or jump to...** **Pull requests** **Issues** **Marketplace** **Explore**

**Overview** **Repositories** 69 **Projects** 0 **Stars** 11 **Followers** 784 **Following** 2

**Popular repositories**

- April201Class**  
April201Class  
JavaScript ★ 20 🍴 21
- ssuet-android-adv-mar17**  
Class Work for Android Advance class  
Java ★ 18 🍴 21
- python-code**  
Python Code for AI class  
Python ★ 18 🍴 15
- ssuet-android-section1**  
Repository for Class of Java and Android for SSUET Android Students Section 1  
Java ★ 15 🍴 22

**Zeeshan Hanif**  
zeeshanhanif

**New repository**  
Import repository  
New gist  
New organization  
New project

# Create New Repository

## Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner

 zeeshanhanif ▾

Repository name \*

MyProject ✓

Great repository names are short and memorable. Need inspiration? How about **cautious-invention**?

Description (optional)

My First Project on Github



Public

Anyone 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.

☒ Initialize this repository with a README

This will let you immediately clone the repository to your computer.

Add .gitignore: None ▾

Add a license: None ▾



Create repository

&lt;&gt; Code

! Issues 0

🔗 Pull requests 0

📁 Projects 0

📖 Wiki

🛡 Security

📊 Insights

⚙ Settings

## My First Project on Github

Edit

[Manage topics](#)

🕒 1 commit

🌿 1 branch

📦 0 releases

👤 1 contributor

Branch: master ▾

New pull request

Create new file

Upload files

Find File

Clone or download ▾



zeeshanhanif Initial commit

Latest commit 7f5a02a now



README.md

Initial commit

now



README.md

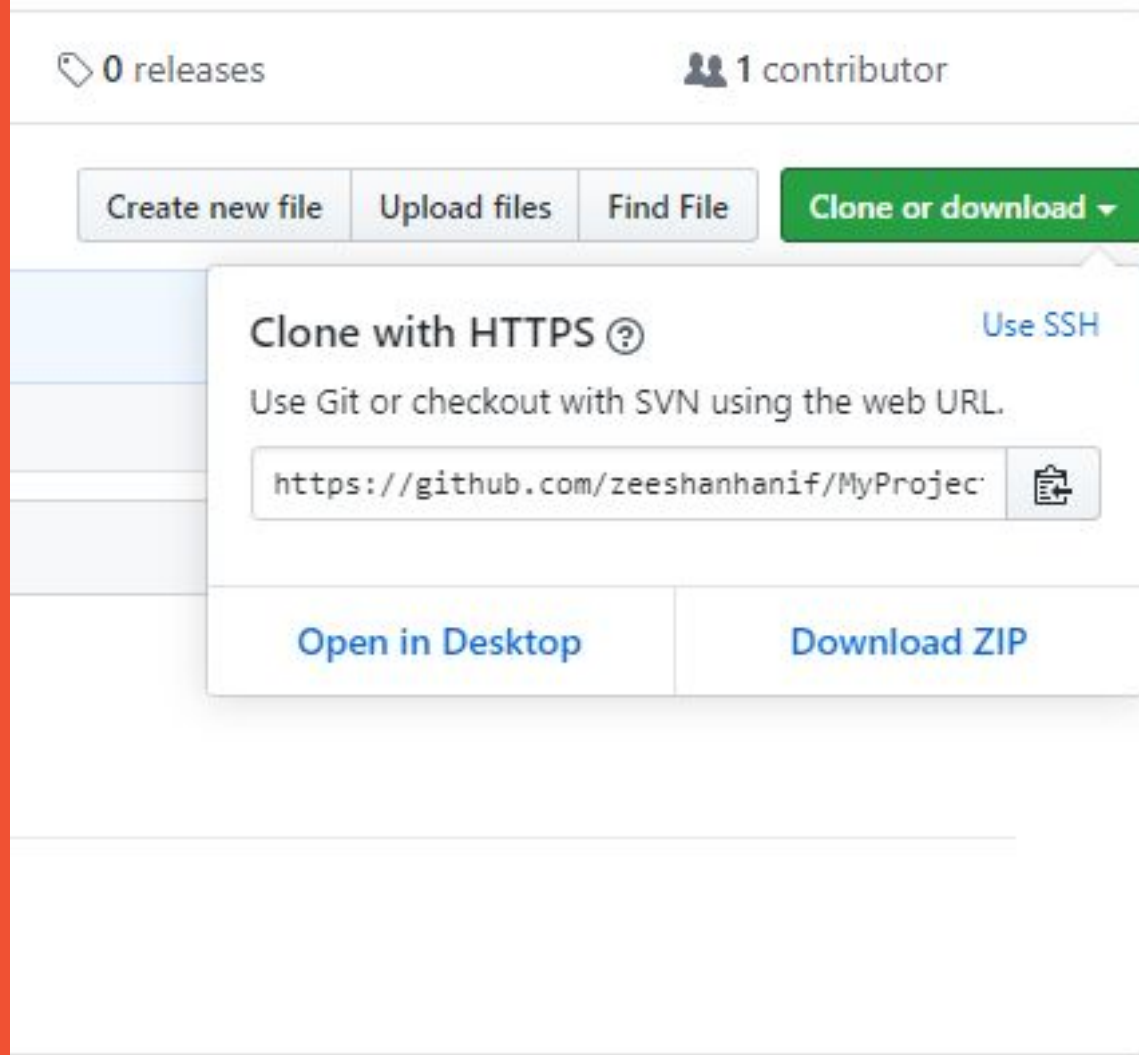


# MyProject

My First Project on Github

# Cloning a repository

Copy the url in popup



# Cloning a repository

- ❖ Go to the local root folder, C:\Repos, for the repositories.
- ❖ Open a terminal within it.
- ❖ Type git clone  
`https://github.com/zeeshanhanif/MyProject.git`

# Work on remote repository

- ❖ After clone, make changes to your project on your local machine
- ❖ Add file or change a file
- ❖ Add file to staging area by `git add .`
- ❖ Commit file `git commit -m "updated feature"`
- ❖ Run command `"git push"` to push your change to remote repository on server



# Commands to interact remote repository

- ❖ git push
  - Push changes to remote repository
- ❖ git fetch
  - Fetch changes from remote repository
- ❖ git merge
  - Merge changes that was fetch by 'git fetch' command

# Commands to interact remote repository

- ❖ `git pull`
  - Fetch and merge changes from remote repository
- ❖ `git remote -v`
  - Show remote urls
- ❖ `git remote show origin`
  - Show details of origin

# Commands to interact remote repository

- ❖ “git remote add myremote <https://github.com/zeeshanhanif/MyProject.git>”

This command will add remote repo in local repository

# Demo Basic clone and push pull with Terminal

Demo Basic clone  
and push pull with  
Smartgit UI

# Publish a local repository to GitHub

- ❖ If you already have local repository on your machine and want to connect it with remote repository then:
  - Create repository on github
  - Open terminal in your local repository
  - Run following command
  - `git remote add origin`  
<https://github.com/zeeshanhanif/MyProject.git>
  - `git push -u origin master`

# Publish a local repository to GitHub

- ❖ `git push -u origin master`
  - The “-u” flag establishes a tracking connection between remote and our local

# Commands to interact remote repository

- ❖ git push 'remote' 'branch'
  - git push origin master
- ❖ Git push command require which remote repository you want to push and in which branch of remote repository



# Commands to interact remote repository

- ❖ `git log`
  - Show logs for current branch
- ❖ `git log remote/branch`
  - `git log origin/master`
- ❖ In `git log` command you specify remote repository and branch so it will show logs from that branch of remote.

# Demo with Terminal

# Demo with Smartgit UI

# Pushing a local branch to a remote repository

- ❖ git branch work
  - This will create new branch 'work' on local repo
- ❖ git checkout work
  - Switch to 'work' branch
- ❖ Make changes
- ❖ git push -u origin work

# Demo with Terminal

# Demo with Smartgit UI

# Git workflow

## git-flow

- ❖ When working in team we want follow some rules and style to work on project.
- ❖ For this we should follow branching model in where there should be specific branches for specific purpose

# Git workflow

## git-flow

- ❖ Branches (example)
  - master
  - development
  - feature/rss-feed
  - hotfix/missing-link



# Git workflow

## git-flow

- ❖ When working in team we want to review code of team members and merge code in main branch only after that.
- ❖ For this purpose github provide pull request feature.
- ❖ In professional projects you will see the very often

Demo

# Social Coding

–

## Forking a repository

- ❖ First, remember that fork is not a Git feature, but a GitHub invention
- ❖ When you fork on GitHub, you get a server-side clone of the repository on your GitHub account

# Social Coding

–

## Forking a repository

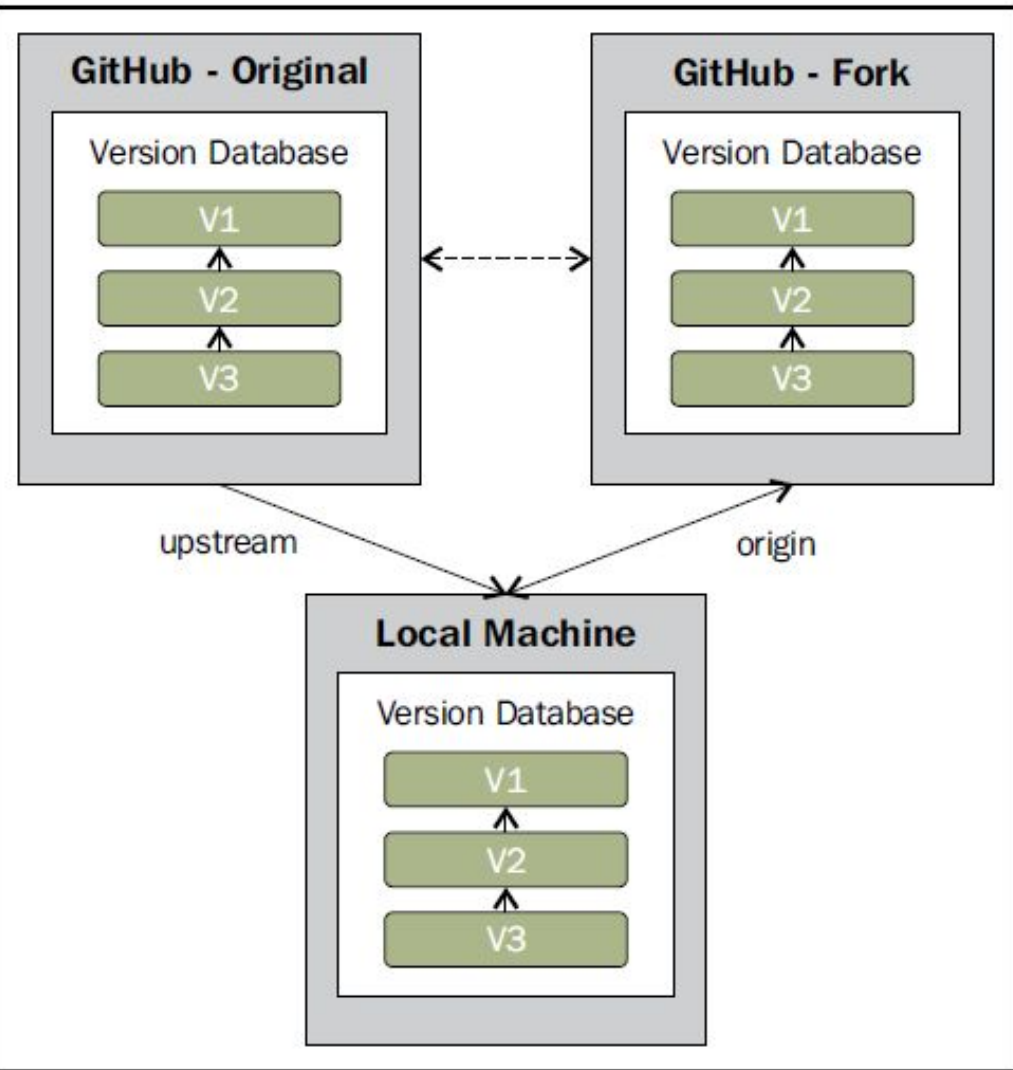
- ❖ When you want to work on someone else's repository on which you don't have write access, you create fork of it
- ❖ This is mostly done with public projects, where community contributes

# Social Coding

–

## Forking a repository

- ❖ When you fork that means you are creating copy of someone else's repo on github into your github account.
- ❖ This happens on server side directly on github
- ❖ Then you can clone the repo from your account



# Social Coding

–

## Forking a repository

- ❖ After clone you can work on repo as it is your repo
- ❖ Make changes, commit push/pull all happens in your repo
- ❖ Only when you want to contribute your changes to original repo you can create pull request

# Social Coding

–

## Forking a repository

- ❖ A pull request is a way to tell the original author, "Hey! I did something interesting using your original code. Do you want to take a look and integrate my work, if you find it good enough?"



# Social Coding

–

## Forking a repository

- ❖ As you are not contributor to the project you can only submit your work to original repo with pull request
- ❖ Author of original repo has rights to accept or reject or you changes

Demo

# Deleting Branches

- ❖ When you are done with a branch and it is no longer needed then you can delete the branch
  - `git branch -d contact-form`
- ❖ Deleting remote branch, add “r” flag
  - `git branch -dr origin/contact-form`

Demo

# Undoing Local Changes

- ❖ If you have local changes that are not committed and want discard change then you can use following commands
- ❖ Discard changes in single file
  - `git checkout HEAD <file/to/restore>`
- ❖ Discard all changes that are not committed (already learned previously)
  - `git reset --hard HEAD`

Demo

# Undoing Committed Changes

# Undoing Committed Changes

- ❖ Once code is committed and you feel your changes were wrong and you want to undo commits
- ❖ You can undo committed changes using following commands:
  - `git revert <commitHash>`
  - `git reset --hard <commitHash>`

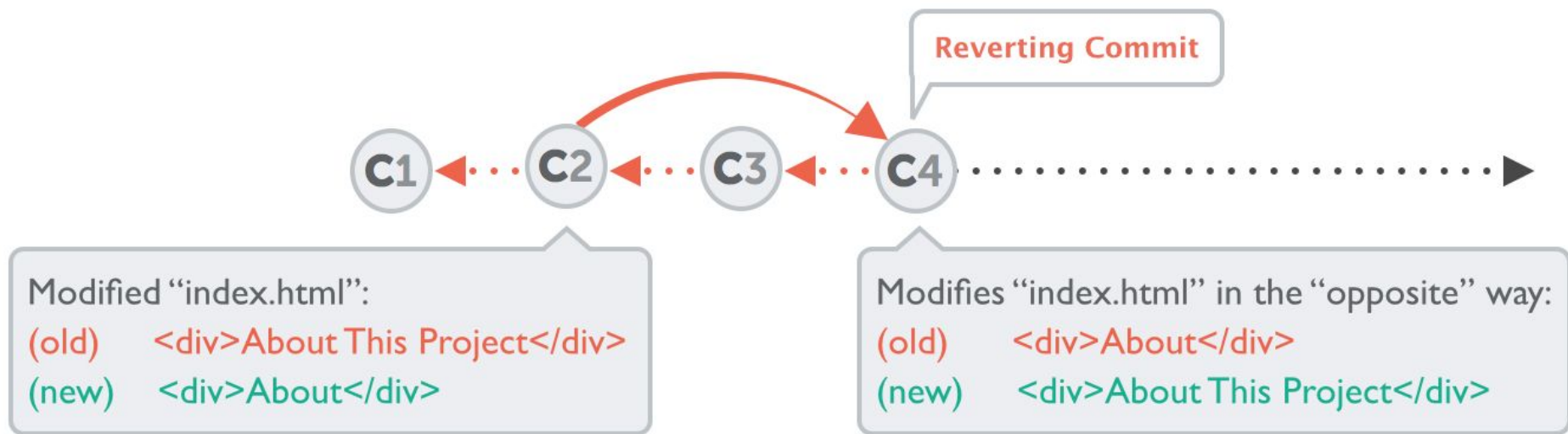


# git revert <commitHash>

- ❖ This command does not actually delete any commit.
- ❖ Instead it reverts the effects of a certain commit, effectively undoing it.
- ❖ It does this by producing a new commit with changes that revert each of the changes in that unwanted commit.
- ❖ For example, if your original commit added a word in a certain place, the reverting commit will remove exactly this word, again.

# git revert <commitHash>

❖ For example: `git revert 2b504be`



# git reset --hard <commitHash>

- ❖ It neither produces any new commits nor does it delete any old ones.
- ❖ It works by resetting your current HEAD branch to an older revision (also called “rolling back” to that older revision):
- ❖ If you call it with “--keep” instead of “--hard”, all changes from rolled back revisions will be preserved as local changes in your working directory.

# git reset --hard <commitHash>

- ❖ For example: `git reset --hard 2be18d9`
- ❖ Preserved as local changes : `git reset --keep 2be18d9`



Demo

# Rebase

# Rebase as an Alternative to Merge

- ❖ Merging is definitely the easiest and most common way to integrate changes.
- ❖ But merging is not the only one: “Rebase” is an alternative means of integration.
- ❖ Rebasing is quite a bit more complex than merging

# Rebase

--

## Understand merge first

Two possibilities

- ❖ Fast-Forward
- ❖ Merge Commit

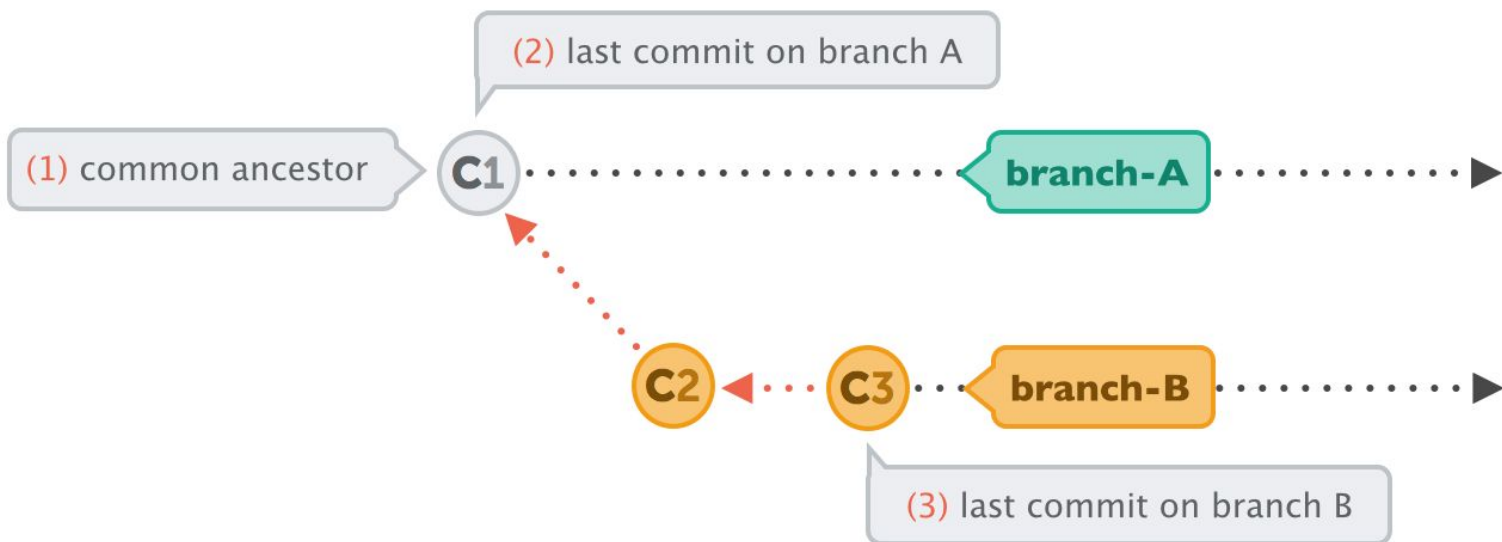


# Understand merge first : Fast-Forward

- ❖ In very simple cases, one of the two branches doesn't have any new commits since the branching happened - its latest commit is still the common ancestor.

# Understand merge first : Fast-Forward

- ❖ Only one branch has new commits

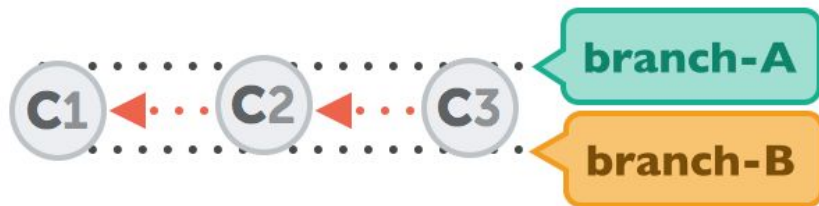


# Understand merge first : Fast-Forward

- ❖ In this case, performing the integration is dead simple
- ❖ Git can just add all the commits of the other branch on top of the common ancestor commit.
- ❖ In Git, this simplest form of integration is called a “fast-forward” merge. Both branches then share the exact same history.

# Understand merge first : Fast-Forward

- ❖ Both branch have same history after fast-forward



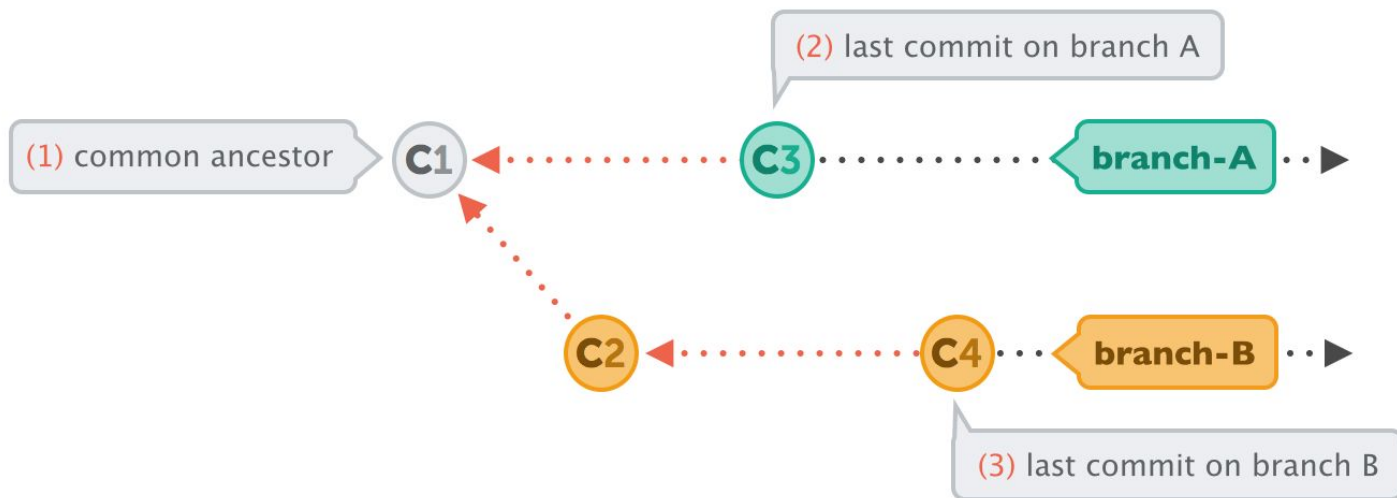
Demo

# Understand merge first : Merge Commit

- ❖ In a lot of cases, however, both branches moved forward individually.
- ❖ And can have different commits

# Understand merge first : Merge Commit

- ❖ Both branches have commits that are done after branch created



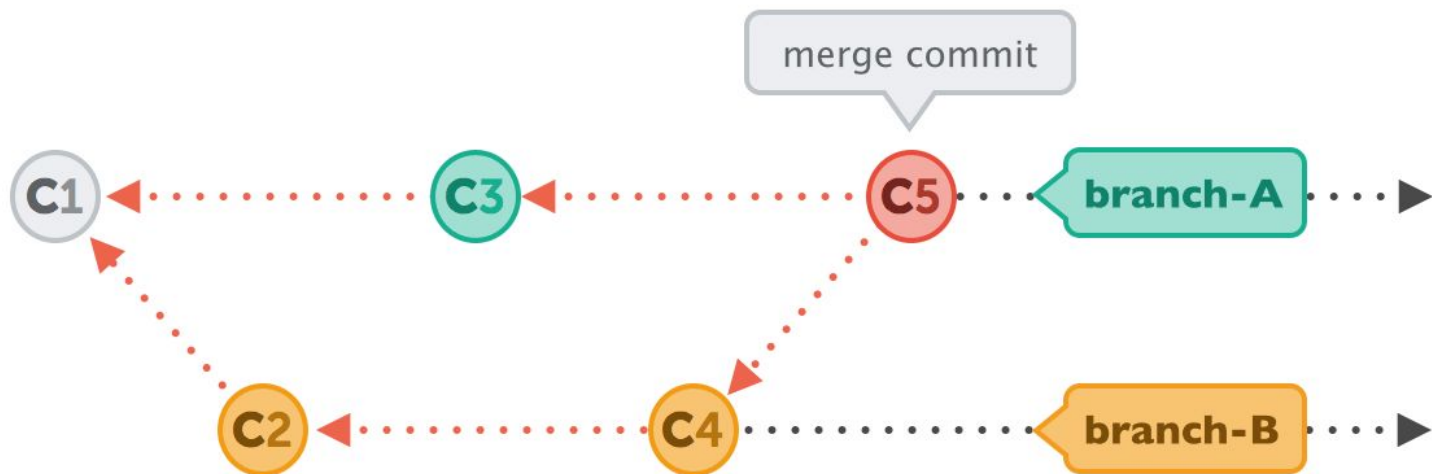
# Understand merge first : Merge Commit

- ❖ To make an integration, Git will have to create a new commit that contains the differences between them - the merge commit.



# Understand merge first : Merge Commit

- ❖ Git automatically created merge commit “C5”

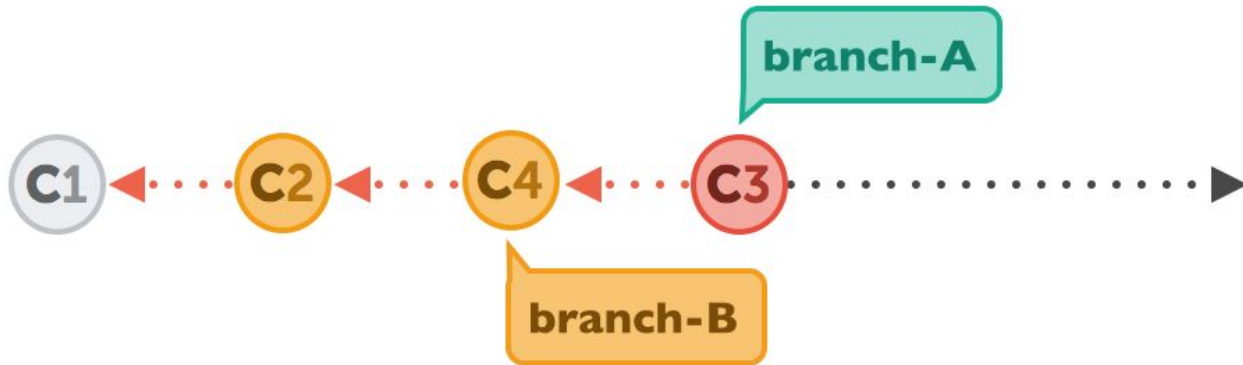


Demo

# Rebase

- ❖ Sometimes we prefer to go without such automatic merge commits.
- ❖ We want the project's history to look as if it had evolved in a single, straight line.
- ❖ No indication remains that it had been split into multiple branches at some point.

# Rebase

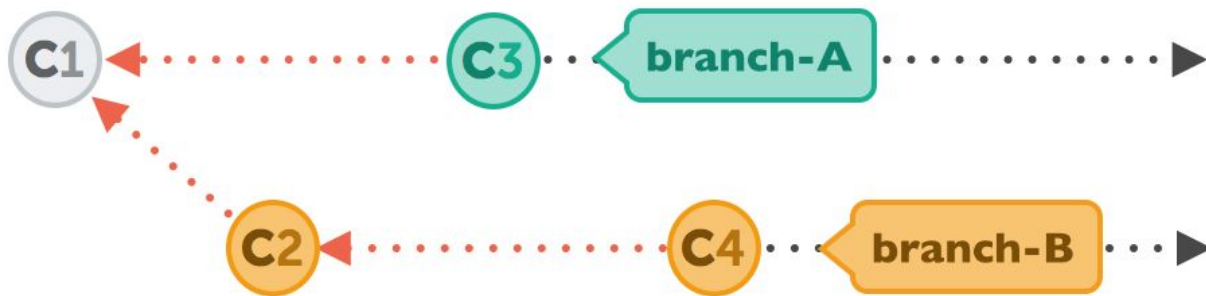


# Rebase

- ❖ Let's walk through a rebase operation step by step.
- ❖ The scenario is the same as in the previous examples: we want to integrate the changes from branch-B into branch-A, but now by using rebase.

# Rebase

- ❖ Same scenario as we did with Merge



# Rebase Command

- ❖ `git rebase <BranchName>`
- ❖ `git rebase branch-B`

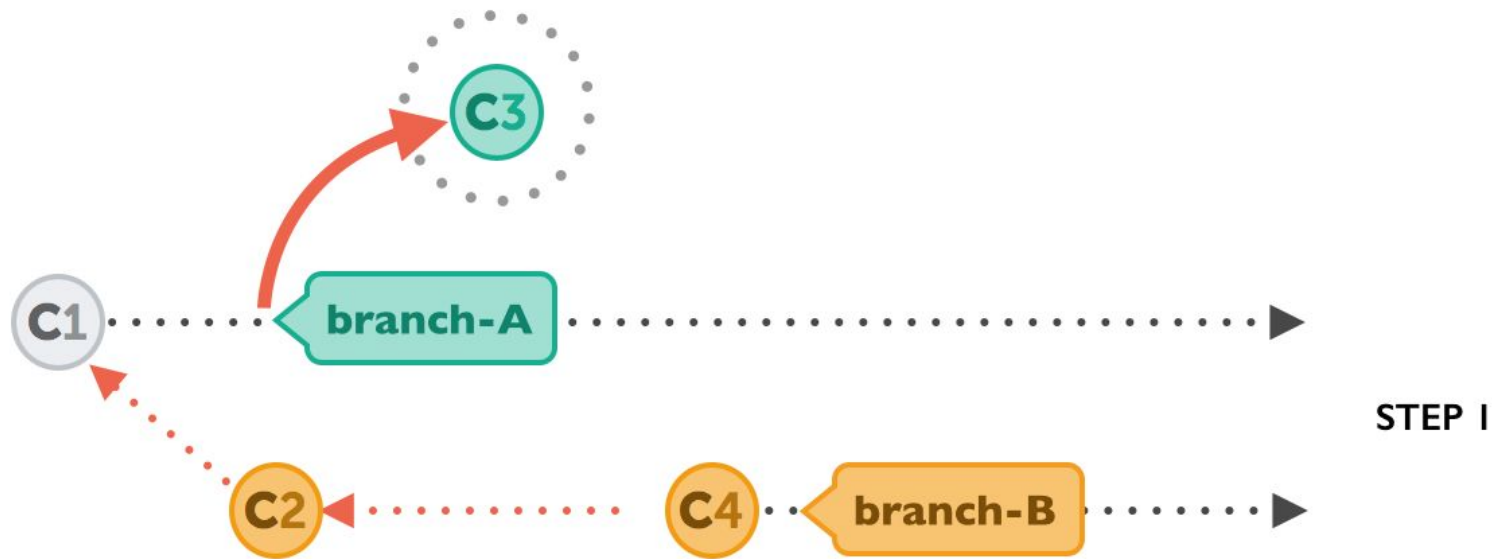
# Rebase -- Step 1

- ❖ First, Git will “undo” all commits on branch-A that happened after the lines began to branch out (after the common ancestor commit).
- ❖ However, of course, it won’t discard them: instead you can think of those commits as being “saved away temporarily”.



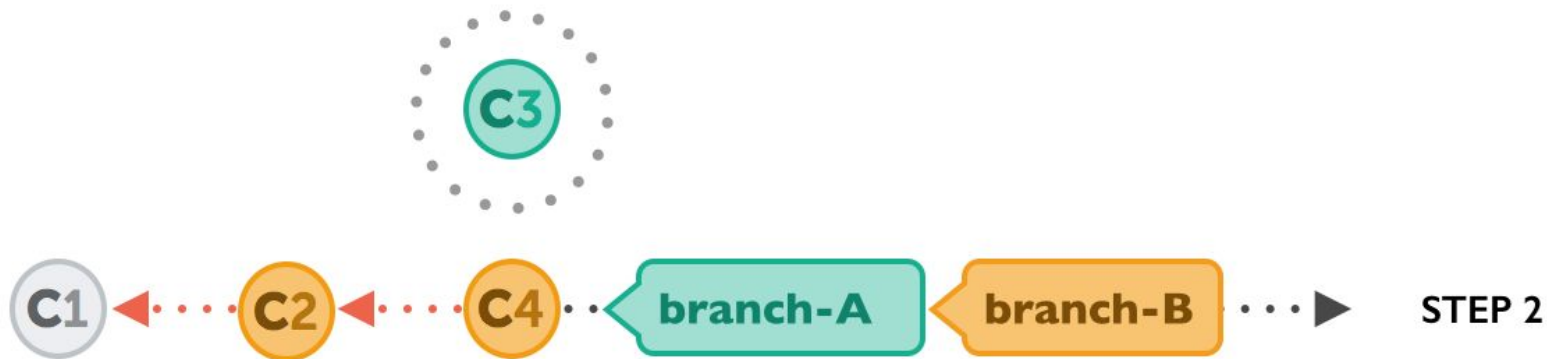
# Rebase -- Step 1

- ❖ Undo all commits on branch-A after common ancestor



## Rebase -- Step 2

- ❖ Next, it applies the commits from branch-B that we want to integrate. At this point, both branches look exactly the same.

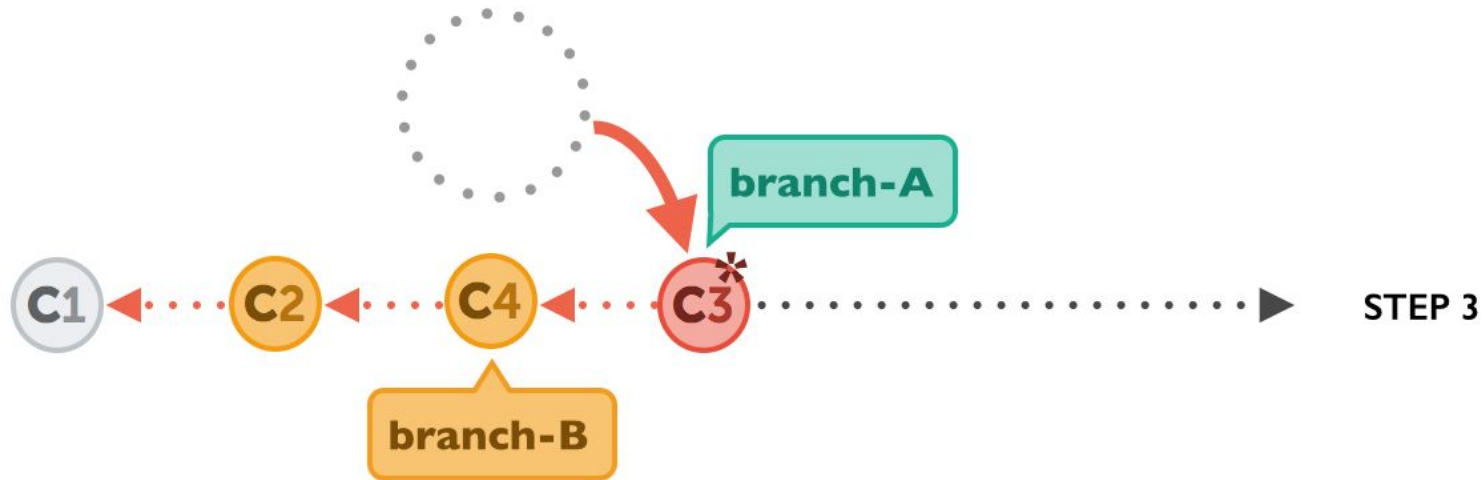


## Rebase -- Step 3

- ❖ In the final step, the new commits on branch-A are now reapplied - but on a new position, on top of the integrated commits from branch-B (they are re-based).
- ❖ The result looks like development had happened in a straight line.
- ❖ Instead of a merge commit that contains all the combined changes, the original commit structure was preserved.
- ❖

# Rebase -- Step 3

- ❖ Applying Branch A commits in the end



# The Pitfalls of Rebase

- ❖ Of course, using rebase isn't just sunshine and roses. You can easily shoot yourself in the foot if you don't mind an important fact: **rebase rewrites history**.
- ❖ As you might have noticed in the last diagram above, commit "C3\*" has an asterisk symbol added.
- ❖ This is because, although it has the same contents as "C3", it's effectively a different commit.

# The Pitfalls of Rebase

- ❖ The reason for this is that it now has a new parent commit (C4, which it was rebased onto, compared to C1, when it was originally created).
- ❖ Rewriting history in such a way is unproblematic as long as it only affects commits that haven't been published, yet
- ❖ If it is published then, some other developer might have based his work on on original C3, this will make it more and more complex

# The Pitfalls of Rebase

- ❖ Therefore, you should use rebase only for cleaning up your local work - but never to rebase commits that have already been published.

Demo



# Markdown

# Markdown

- ❖ Markdown is a lightweight markup language that you can use to add formatting elements to plaintext text documents.
- ❖ Created by John Gruber in 2004, Markdown is now one of the world's most popular markup languages.
- ❖ Markdown is often used to format readme files, for writing messages in online discussion forums, and to create rich text using a plain text editor.

# GitHub Flavored Markdown (GFM)

- ❖ Github release its own markdown language based on original markdown
- ❖ So to format readme file we can use markdown syntax

Online Markdown Editor - ...

dillinger.io

1

≡

DILLINGER

SAVE TO ▾

IMPORT FROM ▾

⚙

DOCUMENT NAME

WORDS: 596

Untitled Document.md

MARKDOWN

↕

PREVIEW

1 ▾

2

3

4

5

6

7

8

9

10

11

12

13

## # Dillinger

Dillinger is a cloud-enabled, mobile-ready, offline-storage, AngularJS powered HTML5 Markdown editor.

- Type some Markdown on the left
- See HTML in the right
- Magic

You can also:

- Import and save files from GitHub, Dropbox, Google Drive and One Drive
- Drag and drop files into Dillinger
- Export documents as Markdown, HTML and PDF

## Dillinger

Dillinger is a cloud-enabled, mobile-ready, offline-storage, AngularJS powered HTML5 Markdown editor.

- Type some Markdown on the left
- See HTML in the right
- Magic

You can also:

- Import and save files from GitHub, Dropbox, Google Drive and One Drive
- Drag and drop files into Dillinger
- Export documents as Markdown, HTML and PDF

# Basic writing and formatting syntax

- ❖ Heading
- ❖ Styling text
- ❖ Quoting text
- ❖ Quoting code
- ❖ Links
- ❖ List
- ❖ Task List

# Heading

- ❖ To create a heading, add one to six # symbols before your heading text.

```
# The largest heading  
## The second largest heading  
##### The smallest heading
```

# The largest heading

---

## The second largest heading

---

The smallest heading

---

# Styling text

- ❖ You can indicate emphasis with bold, italic, or strikethrough text.



Style	Syntax	Keyboard shortcut	Example	Output
Bold	<b>** **</b> or <b>_ _</b>	command/control + b	<b>**This is bold text**</b>	<b>This is bold text</b>
Italic	<i>* *</i> or <i>_ _</i>	command/control + i	<i>*This text is italicized*</i>	<i>This text is italicized</i>
Strikethrough	<del>~ ~</del>		<del>~This was mistaken text~</del>	<del>This was mistaken text</del>
Bold and nested italic	<b><i>** **</i></b> and <b><i>_ _</i></b>		<b><i>**This text is extremely important**</i></b>	<b><i>This text is extremely important</i></b>
All bold and italic	<b><i>*** **</i></b>		<b><i>***All this text is important***</i></b>	<b><i>All this text is important</i></b>

# Quoting text

- ❖ You can quote text with a >

In the words of Abraham Lincoln:

> Pardon my French

In the words of Abraham Lincoln:

Pardon my French

# Quoting code

- ❖ You can call out code or a command within a sentence with single backticks. The text within the backticks will not be formatted.

Use ``git status`` to list all new or modified files that haven't yet been committed.

Use `git status` to list all new or modified files that haven't yet been committed.

To format code or text into its own distinct block, use triple backticks.

Some basic Git commands are:

```

`git status`

`git add`

`git commit`

```

Some basic Git commands are:

`git status`

`git add`

`git commit`

# Links

- ❖ You can create an inline link by wrapping link text in brackets [ ], and then wrapping the URL in parentheses ( ). You can also use the keyboard shortcut command + k to create a link.

This site was built using [GitHub Pages](https://pages.github.com/).

This site was built using [GitHub Pages](#).

# Lists

- ❖ You can make an unordered list by preceding one or more lines of text with - or \*



- George Washington
- John Adams
- Thomas Jefferson

- George Washington
- John Adams
- Thomas Jefferson

To order your list, precede each line with a number.

1. James Madison
2. James Monroe
3. John Quincy Adams

1. James Madison
2. James Monroe
3. John Quincy Adams

# Nested Lists

- ❖ You can create a nested list by indenting one or more list items below another item.

1. First list item
  - First nested list item
  - Second nested list item

1. First list item
  - First nested list item
  - Second nested list item

1. First list item
  - First nested list item
    - Second nested list item

# Task lists

- ❖ To create a task list, preface list items with a regular space character followed by `[ ]`. To mark a task as complete, use `[x]`

- [x] Finish my changes
- [ ] Push my commits to GitHub
- [ ] Open a pull request

- ☒ Finish my changes
- ☐ Push my commits to GitHub
- ☐ Open a pull request

Demo

# Resources

- ❖ <https://www.amazon.com/Learn-Version-Control-step-step/dp/1520786506>
- ❖ <https://www.git-tower.com/learn/git/ebook>
- ❖ <https://git-scm.com/book/en/v2/Git-Branching-Rebasing>
- ❖ <http://git-scm.com/book/en/v2/Git-Branching-Remote-Branche#Tracking-Branches>
- ❖ <https://help.github.com/en/articles/basic-writing-and-formatting-syntax>



# git

Zeeshan Hanif

Director/CTO Panacloud  
PIAIC