

July 20, 2022

DAY 1: Getting started with version control

What is version Control?

- Version control systems allow you to track changes to your codebase/files over time.
- They allow you to go back to some previous version of the codebase without any issues.
- Also, they help in collaborating with people working on the same code

Why we need
version control?

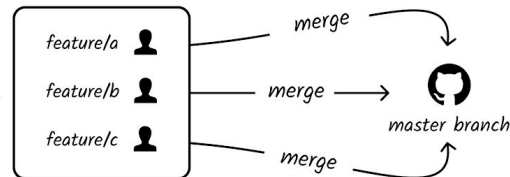
Continuous Integration and Delivery (CI / CD)

[@kamranahmedse](#)

Practices that let you automate certain aspects of your development and deployments.

CI — Continuous Integration

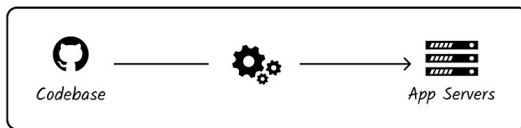
Process of continuously integrating code changes from multiple contributors to a codebase.



Developers working on their feature branches

CD — Continuous Delivery

Extension of continuous integration that allows you to quickly deploy your changes.



Deployments are automated

There is no wait till the release date. Builds and test runs are automated and changes get merged to main branch often.

Deployments are automated and painless. Integrated codebase can be deployed anytime without any issues.

Git and GitHub

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

GitHub is a provider of internet hosting for software development and version control using Git.

It offers the distributed version control and source code management functionality of Git, plus its own features.

Three states of Git

Working Directory

Staging Area

Git Directory

Three states of Git

Three states of Git

Git Directory

Stores the metadata and object database for your project (while cloning the repository)

Three states of Git

Three states of Git

Working Directory

Single checkout of one version of the project.

The files in the directory are pulled out of the compressed database in the Git directory and placed on disk for you to edit and use

Three states of Git

Three states of Git

Staging Area

It is a simple file, generally present in `.git` directory, the stores information about what will go into your next commit

How Git works

Here is a basic overview of how Git works:

1. Create a "repository" (project) with a git hosting tool (like Github)
2. Copy (or clone) the repository to your local machine
3. Add a file to your local repo and "commit" (save) the changes
4. "Push" your changes to your main branch
5. Make a change to your file with a git hosting tool and commit
6. "Pull" the changes to your local machine
7. Create a "branch" (version), make a change, commit the change
8. Open a "pull request" (propose changes to the main branch)
9. "Merge" your branch to the main branch

Steps to get you started with git:

Create a "repository" (project) with a git hosting tool (like Github)



```
git init // initialise your  
local repository with git
```

Copy (or clone) the repository to your local machine



```
git clone url //Cloning the  
remote repository into your  
local machine
```

Git Cheat Sheet

GIT BASICS

```
git init <directory> //Create empty Git repo in specified directory. Run with no arguments to initialize the current directory as a git repository.
```

```
git clone <repo> //Clone repo located at <repo> onto local machine. Original repo can be located on the local filesystem or on a remote machine via HTTP or SSH.
```

```
git config user.name //Define author name to be used for all commits in current repo. Devs commonly use --global flag to set config options for current user.
```

```
git add <directory> // Stage all changes in <directory> for the next commit. Replace <directory> with a <file> to change a specific file.
```

```
git commit -m "<message>" //Commit the staged snapshot, but instead of launching a text editor, use <message> as the commit message.
```

```
git status //List which files are staged, unstaged, and untracked.
```

```
git log //Display the entire commit history using the default format. For customization see additional options.
```

```
git diff //Show unstaged changes between your index and working directory
```

Git Cheat Sheet

UNDOING CHANGES



```
git revert <commit> //Create new commit that undoes all of the changes made in  
<commit>, then apply it to the current branch.
```

```
git reset <file> //Remove <file> from the staging area, but leave the working directory  
unchanged. This unstages a file without overwriting any changes.
```

```
git clean -n //Shows which files would be removed from working directory.  
Use the -f flag in place of the -n flag to execute the clean.
```

Git Cheat Sheet

REWRITING GIT HISTORY



`git commit --amend` //Replace the last commit with the staged changes and last commit combined. Use with nothing staged to edit the last commit's message.

`git rebase <base>` //Rebase the current branch onto <base>. <base> can be a commit ID, branch name, a tag, or a relative reference to HEAD.

`git reflog` //Show a log of changes to the local repository's HEAD.
Add `--relative-date` flag to show date info or `--all` to show all refs.

Git Cheat Sheet

GIT BRANCHES




`git branch` //List all of the branches in your repo. Add a `<branch>` argument to create a new branch with the name `<branch>`.

`git checkout -b <branch>` //Create and check out a new branch named `<branch>`. Drop the `-b` flag to checkout an existing branch.

`git merge <branch>` //Merge `<branch>` into the current branch.

Git Cheat Sheet

REMOTE REPOSITORIES



```
git remote add <name> <url> //Create a new connection to a remote repo. After adding a
remote, you can use <name> as a shortcut for <url> in other commands.

git fetch <remote> <branch> //Fetches a specific <branch>, from the repo. Leave off <branch>
to fetch all remote refs.

git pull <remote> //Fetch the specified remote's copy of current branch and
immediately merge it into the local copy.

git push <remote> <branch> //Push the branch to <remote>, along with necessary commits and
objects. Creates named branch in the remote repo if it doesn't exist.
```


Thank you