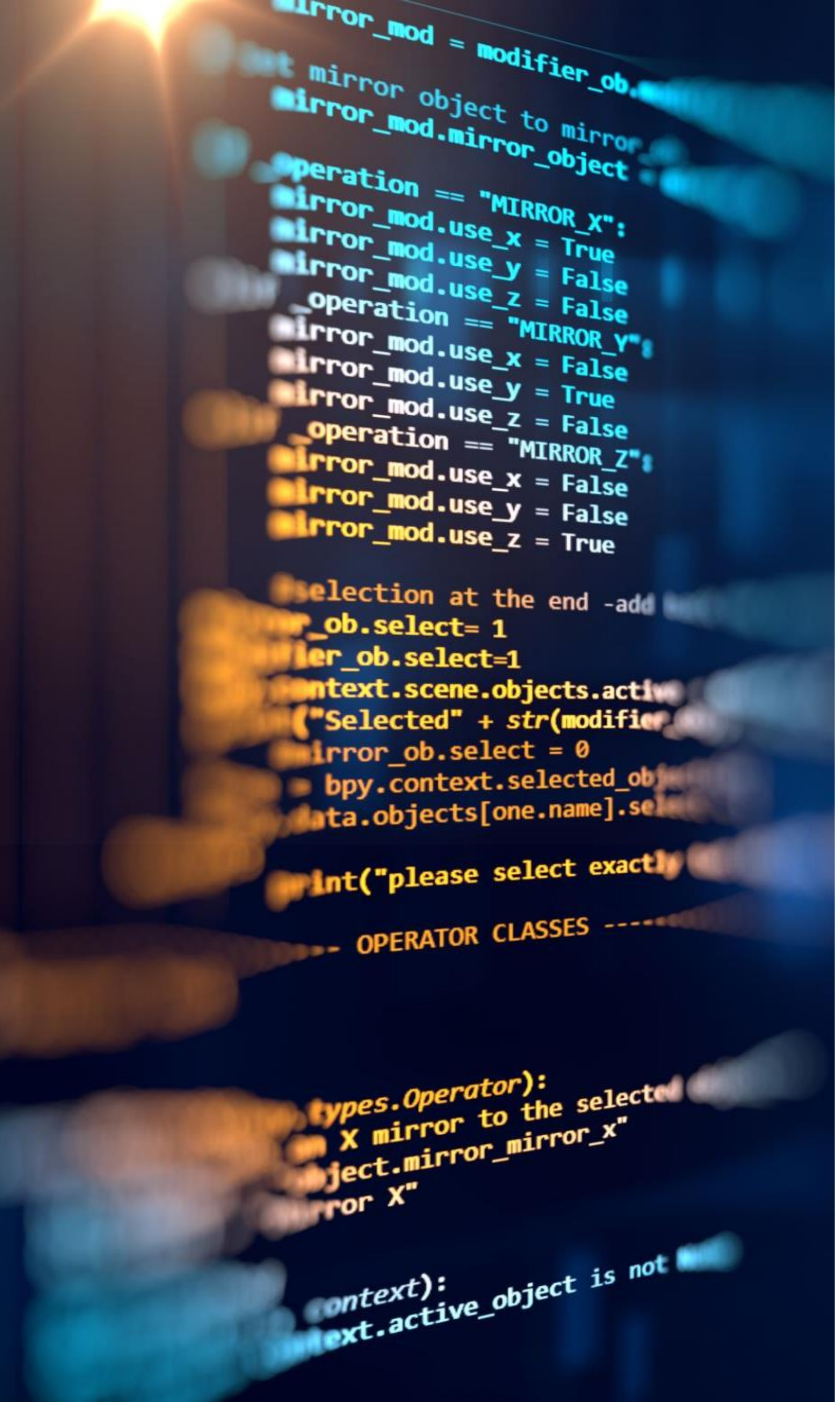
Code Management

(Version Control)



Version Control

- Version control or the source code control is a practice of tracking and managing the changes to the software code.
- As the environment grows, the SVC helps team to work together effectively and efficiently.
- SVC keeps track of every changes happens to the code in a special kind of database and the developers can go back to their previous code state if they need.
- When a lot of developers work together on a code, there are good chances to mix the code, and a lot of chaos can happen due to changes made by many people. Version control help us to mitigate this problem.

Git

- Git is the most common and widely used version control system in the world.
- It is an opensource system.
- Git was developed by Linus Torvalds in 2005.
- Git is an example of Distributed version control system. (DVCS).
- DVCS means rather than having one place for full version history of the software, you can have separate copy of the code in your local machines as well with the full history of changes.
- Git works on the branching strategy, which means you can have many branches from your code just like a tree having branches connected to its trunk.

What is GitHub/Gitlab/Bitbucket

- All these are code hosting platform for version control and collaboration.
- GitHub is a cloud-based solution for code versioning whereas you can use cloud-based as well as install the Bitbucket/GitLab in your environment.
- All these platforms use git command line as the interactive program.

Git in Action

Install Git on your local machine

Clone a git repository

```
# git clone <repo URL>
```

Checking status of repository

```
# git status
```

Adding or Removing files

```
# git add <filename>
# git add . (it will add all newly created file in the project)
```

Committing the changes locally

```
# git commit -m “commit message”
```

Checking the commit history

```
# git log
```

Pushing the local changes to the remote Repo

```
# git push
```

Create local git repository

```
# git init path/to/folder (initialize a directory)
```

Git Configurations

- Git configurations are used to keep a track of all changes and who made those changes.
- You can set the config values globally or on project basis.

```
# git config -global user.name “username”
# git config -global user.email “email”
```

Omit the global flag if you want to set the configs on project basis.

- To list all configs, use the command `# git config -all`
- The global details are saved at `~/.giconfig`

Git Stages

There are four stages in which we can understand the git lifecycle, although there can be more or less stages based on individual understanding.

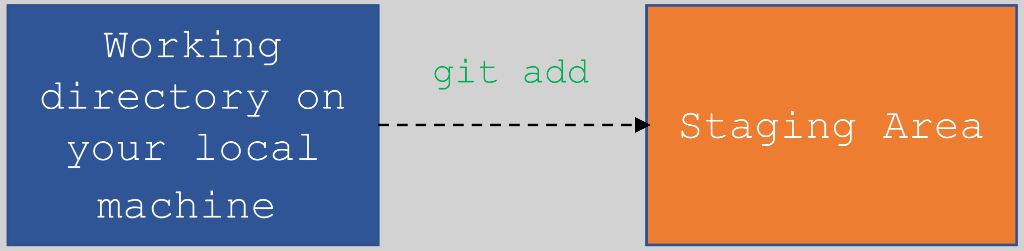
- Local Working directory
- Staging Area or staged
- Local Git Directory
- Remote syncing

Consider a project residing in your local system. This project may or may not be tracked by Git. In either case, this project directory is called your Working directory.

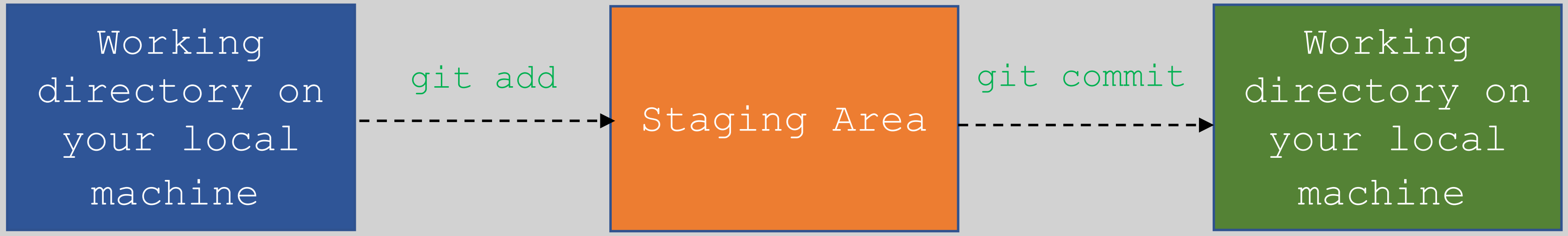
Working directory on your local machine

Git Stages…

While we're in the working directory, we select the files that have to be tracked by Git, needs to be added to the staging area.

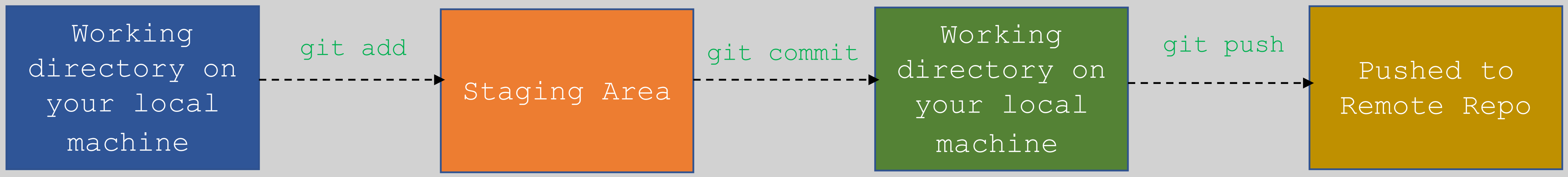


Once you have changed the files, commit your changes locally by using git commit command. It tells the git to keep track of the files and keeps the information of who made the commit and when


Working directory on your local machine
git add
Staging Area
git commit
Working directory on your local machine

Git Stages…

Push the code to the hosting platform (Remote repository)


Working directory on your local machine
git add
Staging Area
git commit
Working directory on your local machine
git push
Pushed to Remote Repo

Hands On LAB-1

- **Task 1:**
  - Install git in your Virtual machine or local machine (laptop)
  - Create your own repository on GitHub.
  - Clone the repository in your local machine.
  - Add some test files in this repository and commit the changes locally.
  - Push the changes to the remote repo.

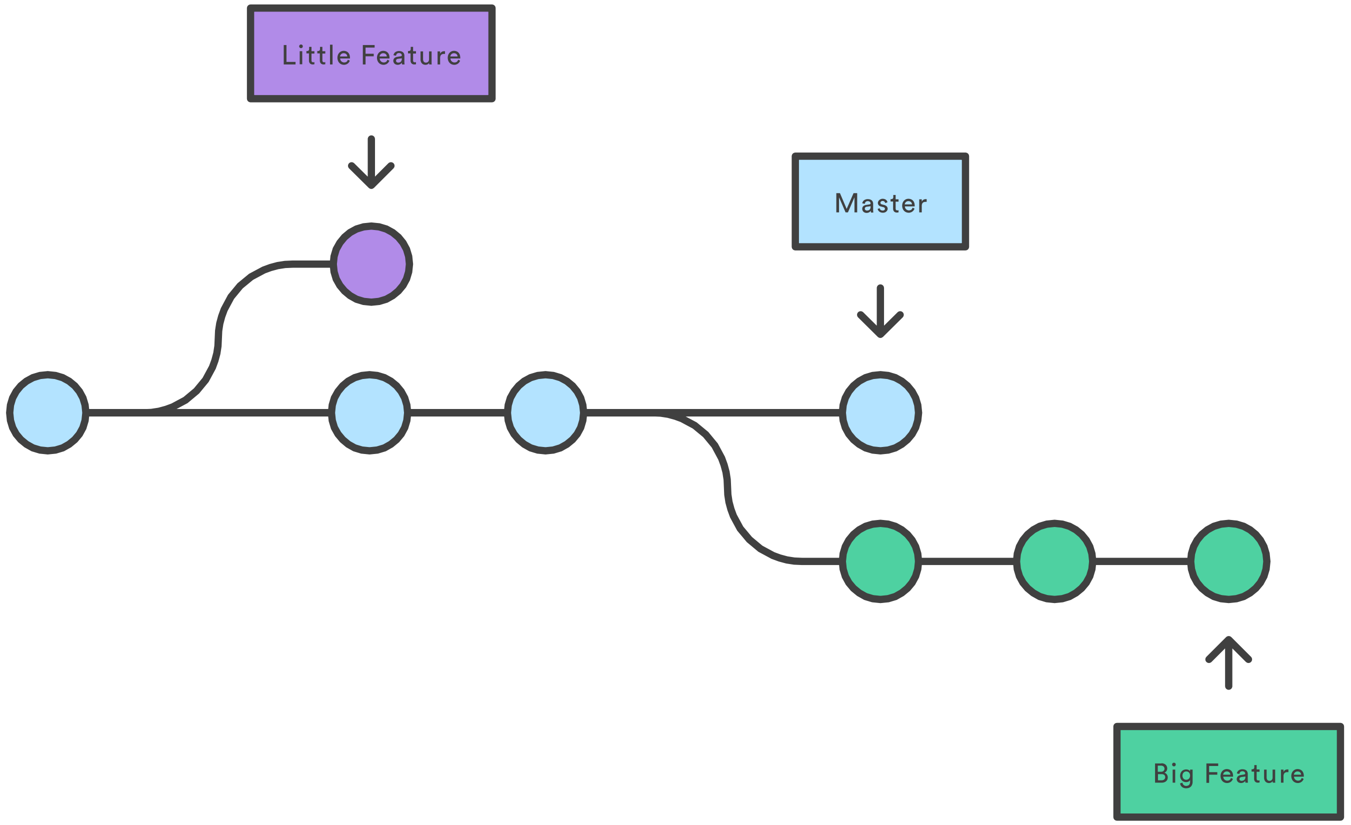
- **Task 2:**
  - Create a local repository and initialize it.
  - Put some test files in it.
  - Add the files to git track and commit them locally.
  - Push the repository to the remote server (GitHub or GitLab)

Git Branch

- Git branches are basically a pointer to a snapshot of your changes.
- When you want to work on some features or issues, it is always recommended (always) to create a new branch and work on the new branch itself. Do not make any changes to the master branch.
- Master is the main branch by default for all your repositories (you can consider any other branch as your main branch such as production/develop etc.)
- Branching help the developers to work on their individual tasks separately without effecting other's workloads.
- You can think of a branch as a new working directory, staging area and project history.

Git Branch...

- In this diagram we can see two different isolated lines, one for small feature and other for big feature.
- By developing the feature using branches, it is possible to work on multiple features parallelly also it keeps the main code (master branch) out of any garbage.
- Once you are done with your feature, you can merge the branch into the master.



Check the branches

```
# git branch (local branches)
# git branch -a (All remote branches)
```

Create and delete a branch

```
# git branch <branch name>
# git branch -d <branch name> (to delete)
```

Checkout to another branch

```
# git checkout <branch-name>
```

Push a new local branch to Remote

```
# git push -u origin <branch-name>
```

Check the difference between two commits

```
#git diff <filename>
```

Ignore certain files

- *Create a git ignore file and write the name of files/directory you want to ignore*

```
# touch .gitignore
```

Merge branches

```
# git checkout baseBranch (master)
# git merge <featureBranch>
```