# Git Basics

## Agenda

1 INTRODUCING *GIT*

2 GIT BASICS

3 BRANCH

4 REMOTE GIT

5 GIT HOOK

6 USEFUL RESOURCES

## INTRODUCING *GIT*

## What is Git

- 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
- Used for source code management, tracking changes in the source code
- Enabling multiple developers to work together on non-linear development.
- Linus Torvalds created Git in 2005 for the development of the Linux kernel.

## Features of Git

- Tracks history
- Free and open source
- Supports non-linear development
- Creates backups
- Scalable
- Supports collaboration
- Branching is easier
- Distributed development

## Team working before Version Control Systems

- Developers copied their changes onto the server.
- Any changes made to the source code were unknown to the other developers.
- No transparency or history about changes.
- There was no communication between the developers.
- There was a chance to lose other’s changes.

## Team working after Git

- Every developer has an entire copy of the code on their local systems.
- Any changes made to the source code can be tracked by others.
- There is transparency, history about changes.
- There is regular communication between the developers.
- No data lost.

## GIT BASICS

# Configure Git

- There are levels of Git config
  - Project
  - Global
  - System

```
git config user.name "John Doe"
```

```
git config --global user.name "John Doe"
```

```
git config --system user.name "John Doe"
```

- Print config
  - Specific config
  - All configs

```
git config --global user.name
```

```
git config -l
```

## Getting a Git Repository

- Take a local directory that is not under version control, and turn it into a Git repository

```
git init
```

- Clone an existing Git repository from elsewhere

```
git clone https://github.com/libgit2/libgit2
```

or

```
git clone https://github.com/libgit2/libgit2 lib_git_project
```

  - supported protocols:
    - https://
    - SSH:   `git://` or `user@server:path/to/repo.git`

## `.git` folder

- Inner folders
    - config
    - Description
    - HEAD
    - hooks/
    - logs/
    - info/
    - objects/
    - refs/

# Recording changes to the Repository

- Each file in the working directory can be in one of two states:
  - Tracked
    - Unmodified
    - Modified
    - Staged
  - Untracked

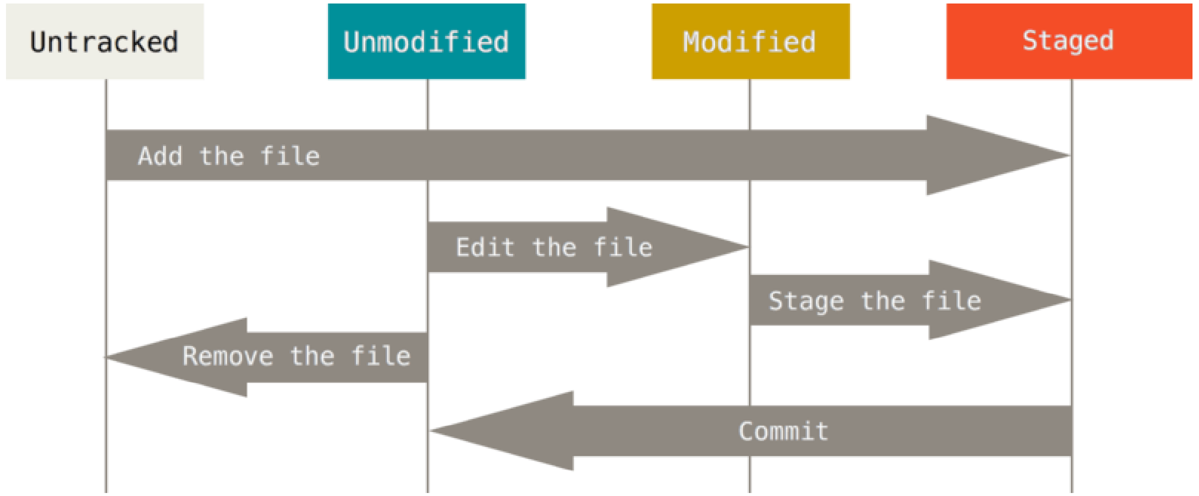

Figure 8. The lifecycle of the status of your files

## Recording changes to the Repository

- Tracking new files
- Status check `git status` , short status `git status -s`
- Staging modified files `git add .` or `git add <file-name>`
- Ignoring files
- Viewing your unstaged and staged changes `git diff` vs `git diff --staged`
- To unstage all the staged files `git reset`
- Committing your changes `git commit` vs `git commit –m “commit message”`
- Skipping the staging area `git commit -a` vs `git commit -a –m “commit message”`
- Removing files `git rm <file-name>`
- Moving files `git mv file_from file_to` vs `git mv file_from file_to -f`

## Viewing the Commit History

- Check history

```
git log
```

  - Format output

```
git log --pretty=<format-style>
```

    - Format Styes
      - Full,
      - Fuller,
      - Format,
      - OneLine

```
git log --pretty=format:"%h - %an, %ar : %s"
```

# Viewing the Commit History - Format Specifiers

| Specifier | Description of Output |
|---|---|
| %H | Commit hash |
| %h | Abbreviated commit hash |
| %T | Tree hash |
| %t | Abbreviated tree hash |
| %P | Parent hashes |
| %p | Abbreviated parent hashes |
| %an | Author name |
| %ae | Author email |
| %ad | Author date (format respects the --date=option) |
| %ar | Author date, relative |
| %cn | Committer name |
| %ce | Committer email |
| %cd | Committer date |
| %cr | Committer date, relative |
| %s | Subject |

## Branch

- Branch is a named, lightweight movable pointer/reference to commits.
- Creating branch

```
git branch <new-unique-branch-name>
```

- Check outing branch

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

- Creating and check outing

```
git checkout -b <new-unique-branch-name>
```

- Naming strategy/conventions

```
<group-name>/<{ticket-id}_{short-summary}>
```

  - Group name
    - feature, bugFix, hotFix, release
  - Id of the ticket
  - Short summary of the feature or bug, usually it matches with title of the ticket

## Branch

- List branches

```
git branch –l
```

- Removing branch

```
git branch –d <branch-name>
```

  - Deleting a branch does not mean the commits will be deleted too!

- Renaming branch

```
git branch --move <bad-branch-name> <corrected-branch-name>
```

- Merging branch

```
git merge <branch-name>
```

  - Fast-forward
  - Non-fast forward
- Long run Branch strategy
- Git Stash
  - Create stash
  - List Stashes
  - Git stash

```
git stash save <name>
```

```
git stash list
```

```
git stash apply <stash-id>
```

## REMOTE GIT

## Branches

- *Remote branches*
    - are in .git/refs/remotes
    - *can be fetched/rebased/merged*
    - *Can be checked out*
    - *Can be tracked by a local branch*

- *Local branches*
    - *can be: commited/pushed/rebased/merged*
    - *Local and remote branches are independent*

- Fetching `git fetch`
- Pulling `git pull`
- Push `git push`

## GIT HOOK

## Git hook

- A way to fire off custom scripts when certain important actions occur.
  - ***Client-side***
  - ***Server-side***
- Triggered by operations such as committing and merging, while server-side hooks run on network operations such as receiving pushed commits.

## USEFUL RESOURCES

## Useful resources

- Git documentation
- https://git-school.github.io/visualizing-git/#free
- https://learngitbranching.js.org/

## DEMO