Introduction to Git and GitHub

—Track, Collaborate, and Elevate Your Code

Git & GitHub

What is Git?

- **Git** is a free, open-source **Version Control System (VCS)** used to track changes in your code and manage project history.
- It helps you save *snapshots* of your files as you work, so you can undo mistakes, review past versions, and collaborate efficiently.

What is GitHub?

- **GitHub** is an online platform/service that hosts Git repositories, making it easy to share your projects, showcase work, and collaborate with others.
- Provides remote storage, project management tools, and easy collaboration for teams.

Difference: Git vs GitHub

Feature	Git	GitHub
Tool Type	Version Control System (software)	Hosting service for Git repositories
Usage Location	Runs locally on your computer	Web-based (remote, accessible anywhere)
Purpose	Tracks and manages code changes/history	Stores, shares, and manages repos online
Collaboration	Possible (with manual file sharing)	Streamlined (multi-user, pull requests, etc.)
Example	`git commit`, `git add`, `git push` (local)	github.com/yourusername/project (online repo)

Why Use Git?

- **Popular and Industry-Standard:** Used in both small projects and large companies
- **Free & Open Source**
- **Fast and Scalable:** Suitable for projects of any size
- **Main Uses:**
    - Tracking code history
    - Collaborating with teams

Version Control Concepts

- **Version Control System:** Tool to track all changes in coding projects
- **History Tracking:** See exactly what changed, when, and by whom
- **Project Collaboration:** Manage who made which changes, avoid overwriting

What is GitHub?

- **Definition:** A website/service for hosting and managing Git repositories online
- **Uses:**
    - Store projects and code
    - Share with others (employers, collaborators)
    - Showcase work in portfolios and resumes

Repositories (Repos)

- **Definition:** A folder that contains a project's code and history (tracked by Git)
- **Types:**
    - Public (anyone can view)
    - Private (only you/selected can view)
- Repository = Project Folder

Creating a GitHub Account

- Step-by-step: Go to github.com, sign up with email
- General advice: Use a personal, long-term email

Setting up Your First Repository

- How to create a new repo (“New” button)
- Naming your repo, adding description
- Choosing public or private
- **Adding a README:** Importance of the README.md file for project details

Basic Git and GitHub Workflow

- Local vs. Remote (your computer vs. GitHub)
- Common Actions:
  - Clone: Copy repo from GitHub to your computer
  - Add: Stage changes/files to include in next commit
  - Commit: Save a snapshot of your changes (with message)
  - Push: Upload changes from your computer to GitHub
- Analogy: “Screenshot” for your project at each commit

Git Initial Configuration

- Configure your name and email for Git (used with your commits)
  - `git config --global user.name "Your Name"`
  - `git config --global user.email "your.email@example.com"`
- Check configuration with `git config --list`
- **Think of setting user.name and user.email like writing your signature on every document you submit — it's how everyone knows who contributed what in a shared project

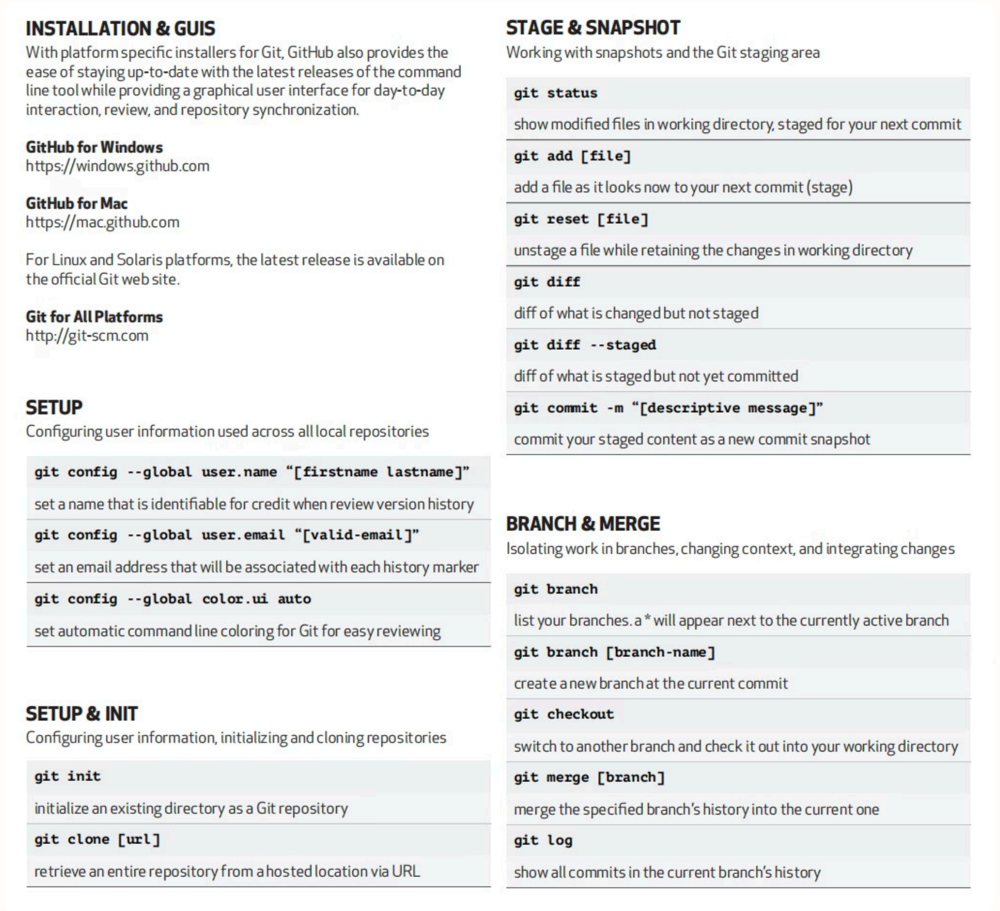
First Git Commands

- **git clone [repo link]** – Copy GitHub repository to your system
- **cd [folder]** – Navigate into the cloned folder
- **ls / ls -a** – List files, including hidden '.git' folder

Core Git Commands

- **git status:** Shows current status of files (tracked/untracked/modified)
- **git add [file]:** Moves file to staging area
- **git commit -m "message":** Creates a snapshot with a message
- **git push:** Sync changes to GitHub
- **git pull:** Get updates from GitHub to your computer

GitHub Cheat sheet-



INSTALLATION & GUIS

With platform specific installers for Git, GitHub also provides the ease of staying up-to-date with the latest releases of the command line tool while providing a graphical user interface for day-to-day interaction, review, and repository synchronization.

GitHub for Windows
https://windows.github.com

GitHub for Mac
https://mac.github.com

For Linux and Solaris platforms, the latest release is available on the official Git web site.

Git for All Platforms
http://git-scm.com

SETUP

Configuring user information used across all local repositories

`git config --global user.name "[firstname lastname]"`	set a name that is identifiable for credit when review version history
`git config --global user.email "[valid-email]"`	set an email address that will be associated with each history marker
`git config --global color.ui auto`	set automatic command line coloring for Git for easy reviewing

SETUP & INIT

Configuring user information, initializing and cloning repositories

`git init`	initialize an existing directory as a Git repository
`git clone [url]`	retrieve an entire repository from a hosted location via URL

STAGE & SNAPSHOT

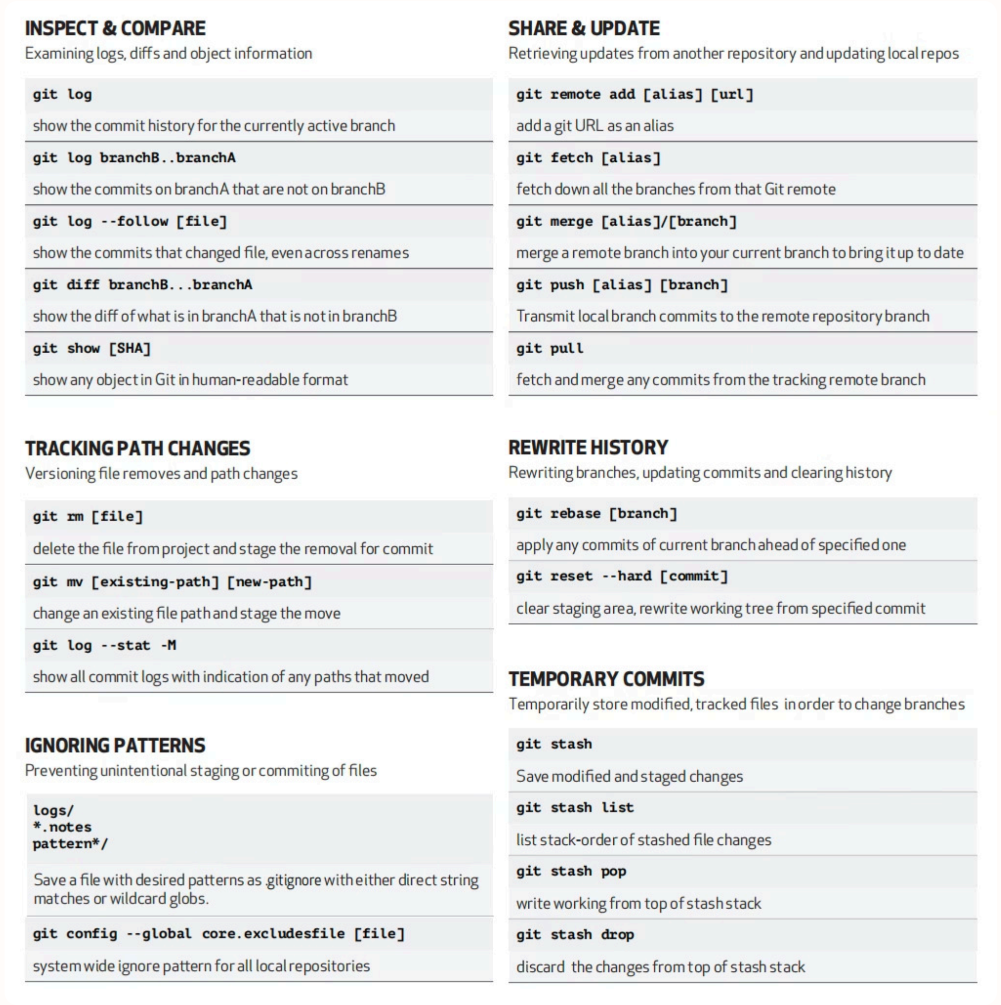
Working with snapshots and the Git staging area

`git status`	show modified files in working directory, staged for your next commit
`git add [file]`	add a file as it looks now to your next commit (stage)
`git reset [file]`	unstage a file while retaining the changes in working directory
`git diff`	diff of what is changed but not staged
`git diff --staged`	diff of what is staged but not yet committed
`git commit -m "[descriptive message]"`	commit your staged content as a new commit snapshot

BRANCH & MERGE

Isolating work in branches, changing context, and integrating changes

`git branch`	list your branches. a * will appear next to the currently active branch
`git branch [branch-name]`	create a new branch at the current commit
`git checkout`	switch to another branch and check it out into your working directory
`git merge [branch]`	merge the specified branch's history into the current one
`git log`	show all commits in the current branch's history



INSPECT & COMPARE

Examining logs, diffs and object information

`git log`	show the commit history for the currently active branch
`git log branchB..branchA`	show the commits on branchA that are not on branchB
`git log --follow [file]`	show the commits that changed file, even across renames
`git diff branchB...branchA`	show the diff of what is in branchA that is not in branchB
`git show [SHA]`	show any object in Git in human-readable format

TRACKING PATH CHANGES

Versioning file removes and path changes

`git rm [file]`	delete the file from project and stage the removal for commit
`git mv [existing-path] [new-path]`	change an existing file path and stage the move
`git log --stat -M`	show all commit logs with indication of any paths that moved

IGNORING PATTERNS

Preventing unintentional staging or commiting of files

```
logs/
*.notes
pattern*/
```

Save a file with desired patterns as .gitignore with either direct string matches or wildcard globs.

`git config --global core.excludesfile [file]`	system wide ignore pattern for all local repositories

SHARE & UPDATE

Retrieving updates from another repository and updating local repos

`git remote add [alias] [url]`	add a git URL as an alias
`git fetch [alias]`	fetch down all the branches from that Git remote
`git merge [alias]/[branch]`	merge a remote branch into your current branch to bring it up to date
`git push [alias] [branch]`	Transmit local branch commits to the remote repository branch
`git pull`	fetch and merge any commits from the tracking remote branch

REWRITE HISTORY

Rewriting branches, updating commits and clearing history

`git rebase [branch]`	apply any commits of current branch ahead of specified one
`git reset --hard [commit]`	clear staging area, rewrite working tree from specified commit

TEMPORARY COMMITS

Temporarily store modified, tracked files in order to change branches

`git stash`	Save modified and staged changes
`git stash list`	list stack-order of stashed file changes
`git stash pop`	write working from top of stash stack
`git stash drop`	discard the changes from top of stash stack

Common File States in Git

- **Untracked:** New files, not being tracked yet by Git
- **Modified:** Tracked files, with changes not yet staged/committed
- **Staged:** Files prepared for commit

Using .gitignore

- Files/folders to exclude from tracking
- Typical examples: node_modules, temp files

Collaborating with Teams

- Multiple people can work on the same repo
- Git tracks who made which changes
- Avoid overwriting work

GitHub Branches Introduction

- Main: Default branch (base version)
- Purpose: Parallel development, features, fixes

What Are Branches in Git & GitHub?

- **Branch:** An independent line of development—like creating a "parallel workspace" to try out new features or fixes without affecting the main code.
- **Default Branch:** Usually called `main` or `master`; this is your production/primary code.
- **Feature Branches:** For new features, bug fixes, experiments—keeps changes separate.

Main Points	**Description**
Isolate Work	Create branches for features, fixes, or experiments
Multiple Versions	Work on several tasks in parallel
Safe Playground	Main branch stays stable

Why Use Branches?

- Develop new features *without* affecting live (main) code.
- Safely fix bugs, experiment, or review other's code.
- Team members or contributors can work independently.

Analogy: A branch is like “Save As”—you can make changes in a copied file and merge good changes back later.

Creating and Switching Branches (Commands & UI)

- **Create branch:** `git branch new-feature`
- **Switch branch:** `git checkout new-feature` or modern: `git switch new-feature`
- **View branches:** `git branch` (lists all local branches).
- **Graphical:** On GitHub, click “Branches” tab.

Merging Branches

- **Merge:** Takes the changes from one branch and applies them to another (commonly feature branch → main).
- **Git Merge Command:** git checkout main,
    - git merge new-feature
- **Pull Request:** On GitHub, open a Pull Request to merge branches—team can review and approve changes before they’re merged online.

Resolving Merge Conflicts

- **Conflict:** Happens if the same part of a file changed in both branches.
- **Git will mark conflict zones:**

```
<<<<<<< HEAD
Your changes
=======
Their changes
>>>>>>> branch-name
```

- **Resolution Steps:**
  - Open conflicted file, look for conflict markers.
  - Edit to keep the correct code.
  - `git add <file>` (after resolving)
  - `git commit -m "Resolved conflict"`

Best Practices for Branches & Merges

- Name branches clearly, e.g., `feature/login-form`
- Keep branches short-lived, merge often
- Delete branches after merging to keep the repo tidy
- Pull latest changes to minimize conflicts before merging

Visual Summary

- **Branches:** Isolate work →
- **Merges:** Bring changes together →
- **Conflicts:** Happen if same line/code area is changed →
- **Solution:** Edit and resolve, always review!

Bonus!!

1. **Forking & Contributing to Other Repositories**
   - What is a fork (copy of someone else's repo to your account)?
   - How to contribute via Pull Requests to open-source projects.
2. **Git Pull vs Git Fetch (Quick Note)**
   - `pull` = fetch + merge automatically.
   - `fetch` = check updates without merging immediately.
3. **Git Log & History Viewing**
   - `git log` → shows commit history.
   - `git log --oneline --graph` for a visual history.
4. **Syncing a Local Branch with Remote**
   - `git pull origin branch-name` to keep up-to-date before merging.
5. **Reverting Changes (Undo Features)**
   - `git restore` for files not committed.
   - `git reset` for staging area.

Class Flow:

1)Clone & status→→Remote(github) , Local(laptop, PC)

2)status→→untracked(new file), modified(changed any code), staged(add), unmodified(commit)

3)Code changes———>add (git add <- file name->) staged———-> commit (git commit -m "some msg")

4)git add .

5)Your branch is ahead of origin/main——>Git push command (local repo content to remote repo)

6)git init—-> new repo

7)what Is git push -u origin main

Init Commands

init - used to create a new git repo

git init

git remote add origin <- link ->

git remote -v

(to verify remote)

git branch

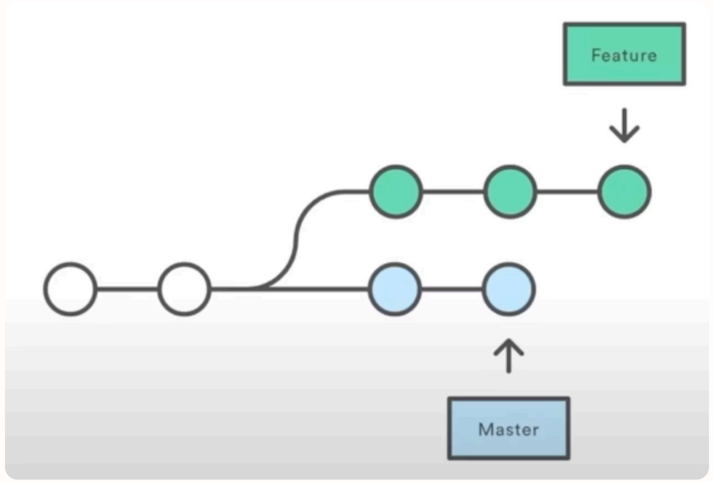
(to check branch)

git branch -M main (to rename branch)

git push origin main

8) workflow:- github repo→clone→changes→add(staging)→commit→push

9)Git Branches



Branch Commands

git branch (to check branch)

git branch -M main (to rename branch)

git checkout ‹- branch name -> (to navigate)

git checkout -b <- new branch name -> (to create new branch)

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

Merge code

Way 1

git diff <- branch name-> (to compare commits, branches, files & more)

git merge <- branch name-> (to merge 2 branches)

Way 2

Create a PR (Pull Request)

Pull Request

It lets you tell others about changes you've pushed to a branch in a repository on GitHub.

PR review ——> Lead or Senior Dev

10) git pull origin main

Merge Conflits

11) Resolving Merge Conflicts

An event that takes place when Git is unable to automatically resolve differences in code between two commits.

- main—change one line and commit
- new branch—change the same line and commit
- —>you can see the diff—>git diff main

Two ways Again—>1)PR 2)git merge

Undoing changes

—git log

Case 1: staged changes

git reset ‹- file name ->

git reset

Case 2: commited changes (for one commit)

git reset HEAD~1

Case 3: commited changes (for many commits)

git reset ‹- commit hash ->

git reset --hard ‹- commit hash ->

Fork

A fork is a new repository that shares code and visibility settings with the original "upstream" repository.

Fork is a rough copy.

Q&A / Thank You

Thank you for joining us on this journey into Collaborative Coding.

We look forward to seeing what you'll create your valuable projects with GitHub!