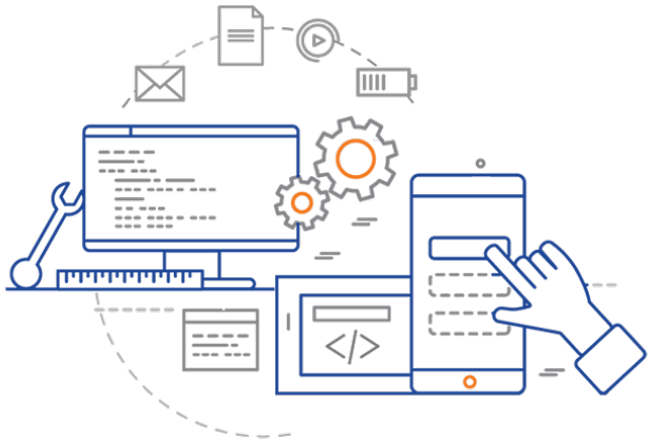


CS 398: Application Development

Week 05 Lecture: Using Git

Benefits; Git commands; Branching; Workflows.

Administrative

Design Reviews from last week

- Return later today or tomorrow morning.
- You will be provided with our marking sheets/comments, and your grade.

Quizzes

- Open Mon 10 AM every week.
- Close the following Sun 11:59 PM.

Return to Class

- Mon Feb 7th in-class (!)

This Week

Goal: Get everything setup for Sprint 1 kickoff (next Mon Feb 7th in-class)

Lectures This Week

- **Mon:** Git, Branching, Collaboration
- **Wed:** Gradle, Build Systems
- **Fri:** Building desktop applications OR Building mobile applications **<— NOT on Quiz**

Lectures Next Week

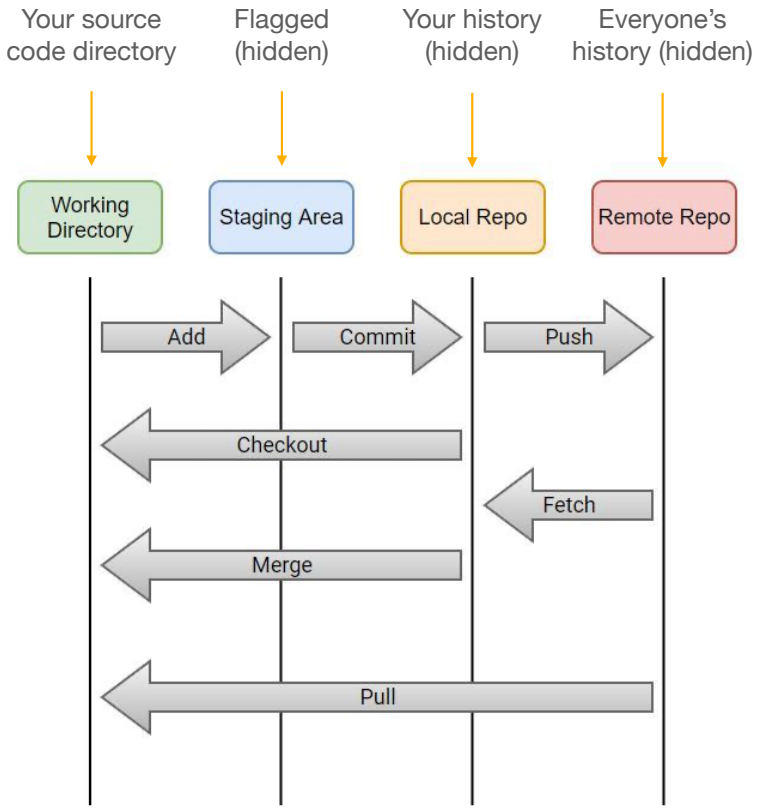
- Unit testing, Refactoring

Review

How does it work?

Git is designed around these core concepts:

- **Working Directory**: A copy of your repository, where you will make your changes before saving them in the repository.
- **Staging Area**: A logical collection of changes from the working directory that you want to collect and work on together (e.g. it might be a feature that resulted in changes to multiple files that you want to save as a single change).
- **Repository**: The location of the canonical version of your source code ("Local Repo" in this diagram).



Local Git Commands

These commands use a **local** staging area and repository.

Command	Description	Example

`git init`	Create a new repository in the current directory.	`$ mkdir repo; cd repo`
		`$ git init`
		`Initialized empty Git repository in /repo/.git/`

`git add`	Add a file to the staging area	`$ vim readme.md` `$ git add readme.md`

`git commit`	Commit all staged files to the repo	`$ git commit -m “Added readme”`
		`[master (root-commit) d3c834b] Added readme`
		` 1 file changed, 0 insertions(+), 0 deletions(-)`
		` create mode 100644 readme.md`

`git status`	Display the status of the current staging area.	`$ git status`
		`On branch master`
		`nothing to commit, working tree clean`

`git checkout`	Checkout a specific commit to this working area. Can use to revert a file.	`$ git checkout main.kt` `Updated 1 path from index.`

Remote Git Commands

Command	Description	Example

`git clone`	Clone the remote repository to a local directory.	`$ git clone https://git.uwaterloo.ca/j2avery/cs349-public.git repo`
		`Cloning into 'repo'...`
		`remote: Enumerating objects: 531, done.`
		`remote: Counting objects: 100% (531/531), done.`
		`remote: Compressing objects: 100% (280/280), done.`
		`remote: Total 2702 (delta 209), reused 320 (delta 100), pack-reused 2171`
		`Receiving objects: 100% (2702/2702), 7.30 MiB | 13.00 MiB/s, done.`
		`Resolving deltas: 100% (939/939), done.`

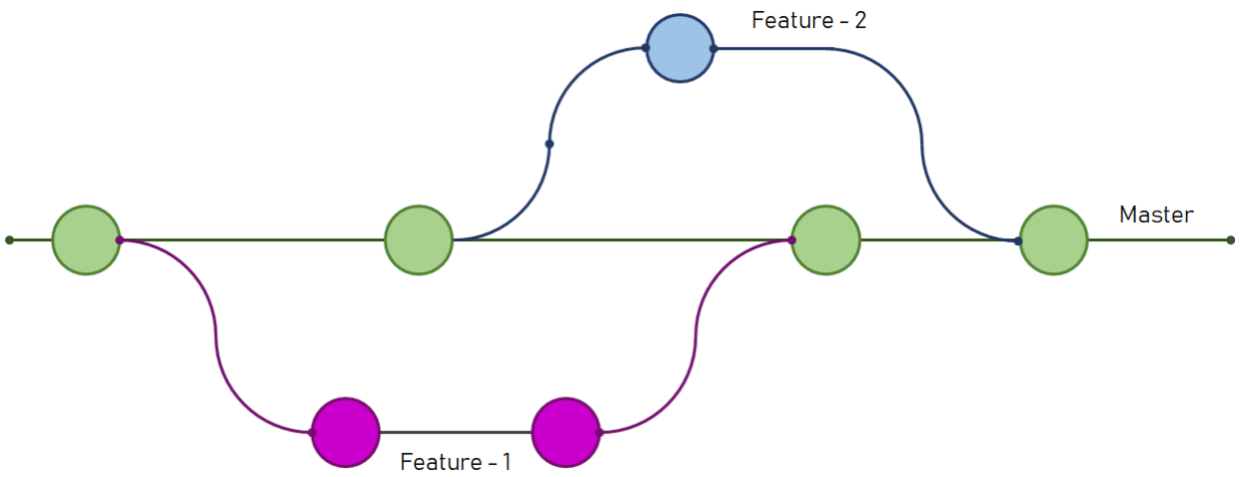
`git pull`	Merge changes into the local repo.	`$ cd repo`
		`$ git pull`
		`Already up to date.`

`git remote`	Modify the remote connection.	`$ git remote` `origin`

Concept: Branching

Think of a repository as a set of commits, all in a line. The main set of commits is like a trunk of a tree. The trunk (also called Master or Main) is where commits are stored by default.

A **branch** is a fork in the tree, where we “split off” work and diverge from one of the commits. Branches diverge from a specific commit, and do not include changed that happened on the trunk after the branch occurred.



Feature branches, merged back into Main once the feature is complete and tested.

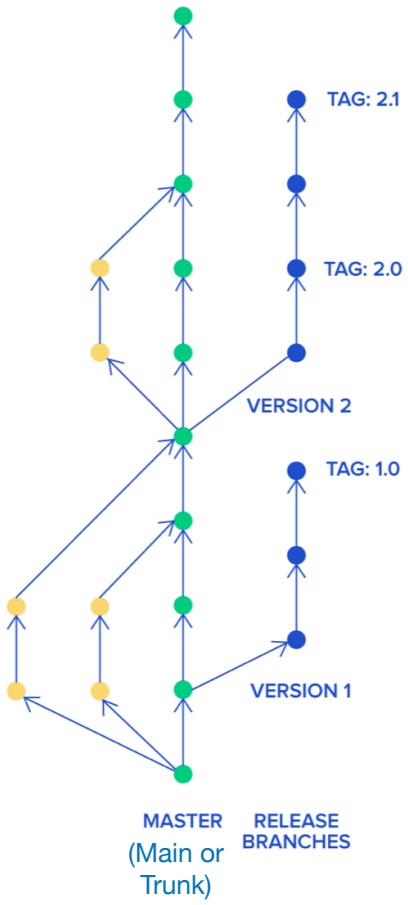
Branching Models: Trunk-Based Dev

In the trunk-based development model, all developers work on a single branch with open access to it. Often it's simply the main or trunk branch. They commit code to it and run it.

It's also common to create short-lived feature branches. Once code on their branch compiles and passes all tests, you merge straight to master.

Characteristics

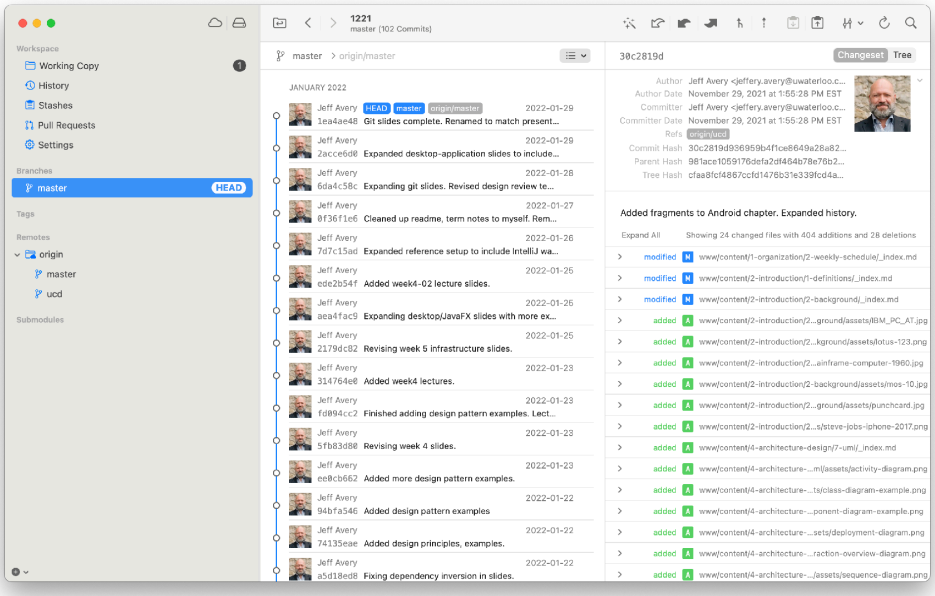
- Feature branches are short-lived.
- Development is continuous so merges are more frequent and easier to resolve.
- Integration testing becomes more important!



https://www.toptal.com/software/trunk-based-development-git-flow

Git Clients

Numerous git clients exist that provide an easier alternative to the command-line.



https://www.git-tower.com

https://github.com/jesseduffield/lazygit

Activities

Activities This Week

Setup GitLab

- All requirements logged in Git and unassigned.
- Milestones (sprints) setup.
- Infrastructure tasks moved to Sprint 0, closed as appropriate.

Source code

- Starting project committed to Git repo.
- Git works across all machines. Everyone has a git client, and knows how to git pull/push.
- IntelliJ is setup for everyone, and the starting project builds.

Technical Investigation

- Choose toolkits; investigate libraries
- Think about data format! How will you store, represent this data?