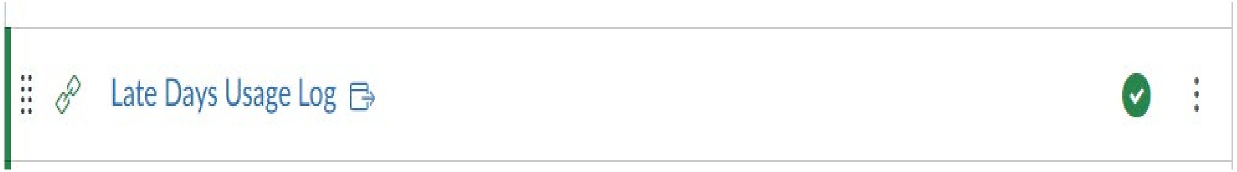
Git & GitHub for Beginners: Understanding Version Control

STAT201 Winter 2025

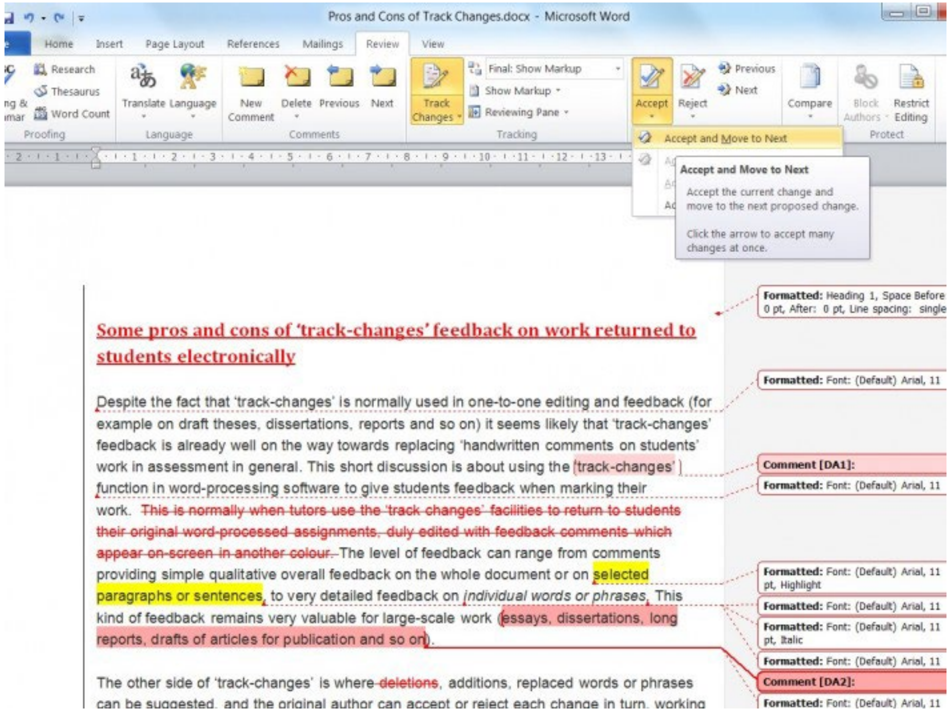
1 Motivation

How can you collaborate with others?


Late Days Usage Log

What if a student intentionally deletes all the content in this file?

Track changes from Word



Accept and Move to Next

Accept the current change and move to the next proposed change.

Click the arrow to accept many changes at once.

Some pros and cons of 'track-changes' feedback on work returned to students electronically

Despite the fact that 'track-changes' is normally used in one-to-one editing and feedback (for example on draft theses, dissertations, reports and so on) it seems likely that 'track-changes' feedback is already well on the way towards replacing 'handwritten comments on students' work in assessment in general. This short discussion is about using the 'track-changes' function in word-processing software to give students feedback when marking their work. ~~This is normally when tutors use the 'track changes' facilities to return to students their original word-processed assignments, duly edited with feedback comments which appear on-screen in another colour.~~ The level of feedback can range from comments providing simple qualitative overall feedback on the whole document or on selected paragraphs or sentences, to very detailed feedback on *individual words or phrases*. This kind of feedback remains very valuable for large-scale work (essays, dissertations, long reports, drafts of articles for publication and so on).

The other side of 'track-changes' is where ~~deletions~~, additions, replaced words or phrases can be suggested, and the original author can accept or reject each change in turn, working

Formatted: Heading 1, Space Before 0 pt, After: 0 pt, Line spacing: single

Formatted: Font: (Default) Arial, 11

Comment [DA1]:

Formatted: Font: (Default) Arial, 11

Formatted: Font: (Default) Arial, 11 pt, Highlight

Formatted: Font: (Default) Arial, 11

Formatted: Font: (Default) Arial, 11 pt, Italic

Formatted: Font: (Default) Arial, 11

Comment [DA2]:

Formatted: Font: (Default) Arial, 11

Limitations of Track Changes

- Can Be Bypassed or Turned Off
- Difficult to Track Deletions in a **Fully Deleted** Document
- Loss of History After Acceptance
- Messy in Large Collaborations
- Limited File Type Support
- Limited Conflict Resolution

About Version Control System

- Compared with the “Track Changes” feature from Word, it is more rigorous, powerful, and scaled up to multiple files.
- A system that keeps records of your changes
- Allows for collaborative development
- Allows you to know **who** made **what** changes and **when**
- Allows you to **revert** any changes and go back to a previous state

Git: the best VCS (most popular)

- Git isn’t the only version control system

mercurial

SUBVERSION®

PERFORCE

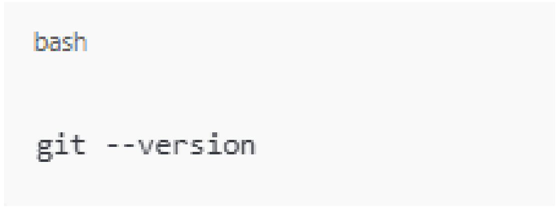
2 How to use git

Git Installation & Setup

Install Git

- Windows download from git-scm.com
- Mac: brew install git (if using Homebrew)

Check if Git installed:



bash

```bash
git --version
```

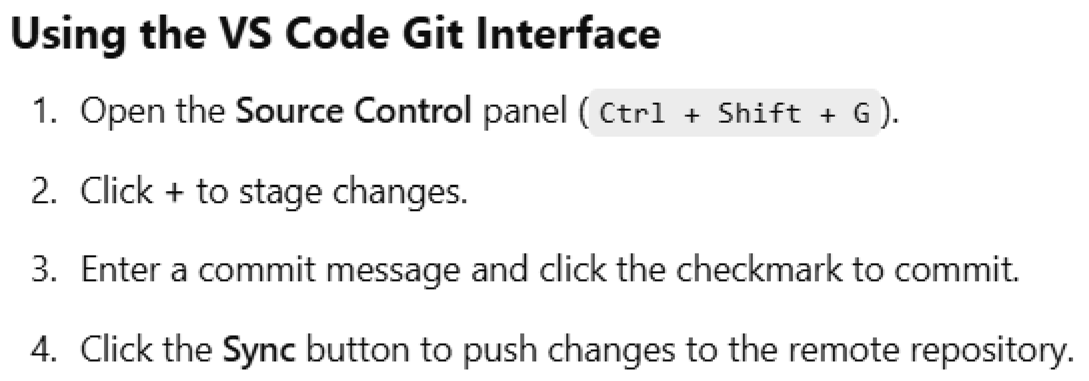
VS Code Git Integration

- Once Git is installed on your system, VS Code automatically detects it.
- You don’t need to install any additional Git extensions for basic functionality, as VS Code includes a built-in Git source control feature.

https://code.visualstudio.com/docs/sourcecontrol/overview

GUI or Command line

You can interact with Git using the **Graphical User Interface (GUI)**, or through the **Command Line Interface (Recommended!).**



Using the VS Code Git Interface

1. Open the **Source Control** panel (`Ctrl + Shift + G`).
2. Click + to stage changes.
3. Enter a commit message and click the checkmark to commit.
4. Click the **Sync** button to push changes to the remote repository.

Git configuration

- Open terminal in vs code
- Configure the username and email (First-Time Setup)

```
git config --global user.name "Your Name"
git config --global user.email "your.email@example.com"
```

- Check if config is correct?

```
git config --list
```

You only need to do this once!

RStudio + Git

Initializing a Repository in RStudio

1. Open RStudio and create a new project.
2. In the **Create Project** window, choose **Version Control** > **Git**.
3. Select an existing Git repository or clone a repository by providing a URL.

Basic Git Workflow in RStudio

1. **Stage Changes**:
   - Open the **Git** pane and check the files you want to stage.
2. **Commit Changes**:
   - Click **Commit**, enter a commit message, and press **Commit**.
3. **Push to Remote Repository**:
   - Click **Push** to upload changes to GitHub.
4. **Pull Changes**:
   - Click **Pull** to fetch updates from the remote repository.

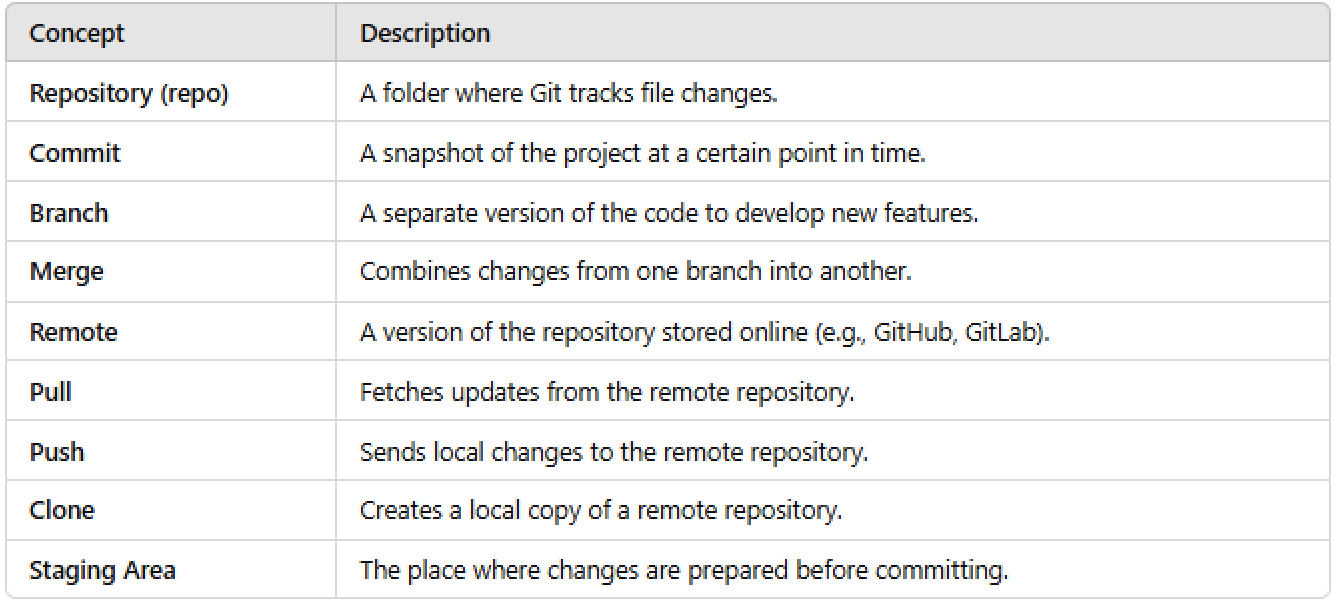
Using the RStudio Terminal for Git Commands

You can also use the terminal in RStudio to run Git commands, just like in VS Code.

3 How does git work?

How does git work?

Can be complicated at first, but there are a few key concepts



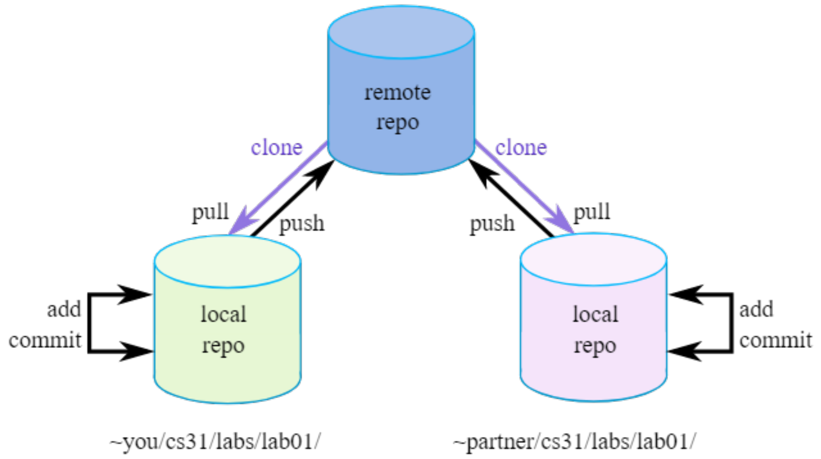
Concept	Description
Repository (repo)	A folder where Git tracks file changes.
Commit	A snapshot of the project at a certain point in time.
Branch	A separate version of the code to develop new features.
Merge	Combines changes from one branch into another.
Remote	A version of the repository stored online (e.g., GitHub, GitLab).
Pull	Fetches updates from the remote repository.
Push	Sends local changes to the remote repository.
Clone	Creates a local copy of a remote repository.
Staging Area	The place where changes are prepared before committing.

Key Concepts: Repositories

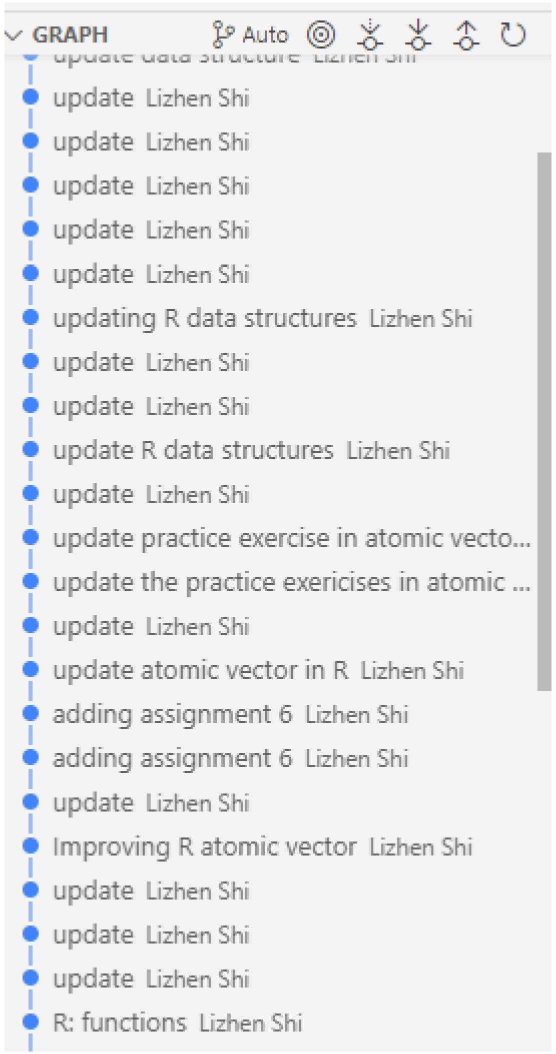
- Often shortened to ‘repo’
- a **storage location** for project files and version history.
- It allows you to **track changes, collaborate, and restore previous versions.**
- Can be **local** (on your computer) or **remote** (hosted on GitHub, GitLab, Bitbucket, etc.).

Repos: Local Repo and Remote

- Can live on a local machine (local repo) or on a remote server (Remote repo, GitHub, bitbucket)



a repo is made up of a bunch of commits history



update data structure Lizhen Shi
update Lizhen Shi
update Lizhen Shi
update Lizhen Shi
update Lizhen Shi
update Lizhen Shi
updating R data structures Lizhen Shi
update Lizhen Shi
update Lizhen Shi
update R data structures Lizhen Shi
update Lizhen Shi
update practice exercise in atomic vecto...
update the practice exericises in atomic ...
update Lizhen Shi
update atomic vector in R Lizhen Shi
adding assignment 6 Lizhen Shi
adding assignment 6 Lizhen Shi
update Lizhen Shi
Improving R atomic vector Lizhen Shi
update Lizhen Shi
update Lizhen Shi
update Lizhen Shi
R: functions Lizhen Shi

Key Concepts: Commit

- A snapshot of the project at a specific point in time.
- Can be a noun or verb
  - “I commited code”
  - “I just made a new commit”
- Essentially, a project is made up of a bunch of commits

Key Concepts: Snapshots

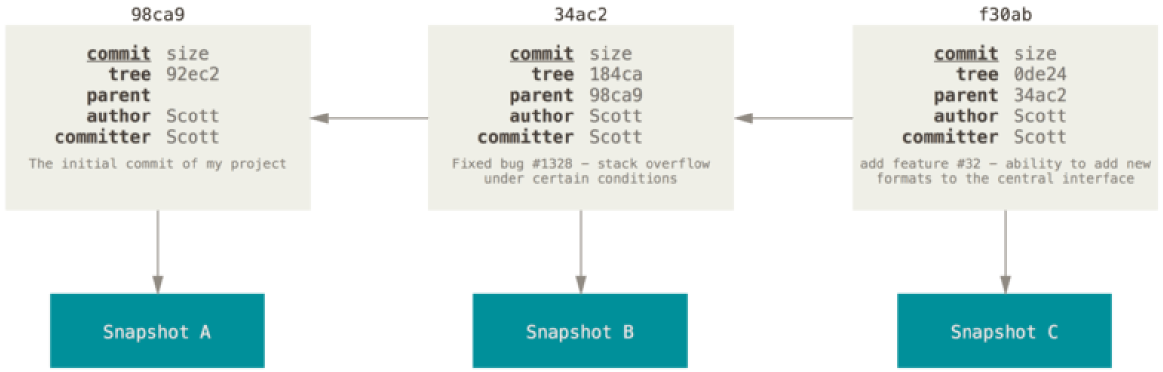
- The way git keeps track of your code history
- Essentially records what all your files look like at a given point in time
- You decide when to take a snapshot, and of what files
- Have the ability to go back to visit any snapshot

Key Concepts: Commit

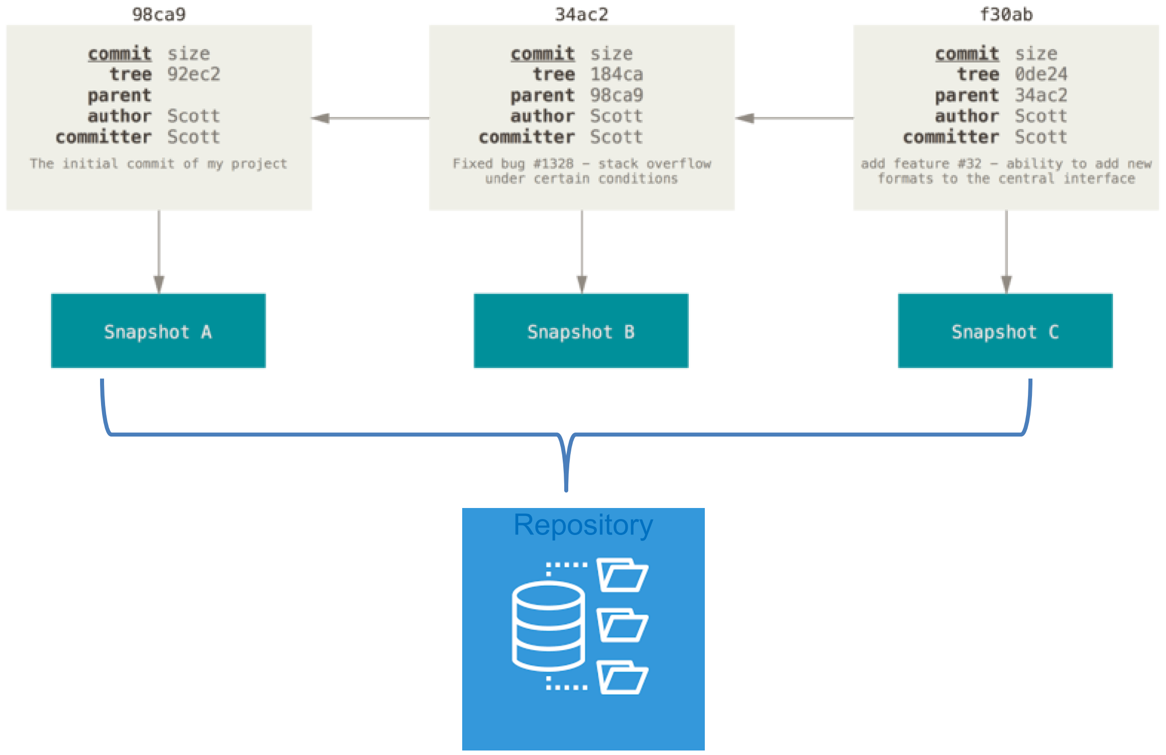
Each commit contains three pieces of information:

1. Information about how the files changed from previously
2. A reference to the commit that came before it
   - Called the "parent commit"
3. A hash code name
   - Will look something like: fb2d2ec5069fc6776c80b3ad6b7cbde3cade4e

Git commit history graph



Git commit history graph with repo



4 How do we make a commit?

Breaking down the Git commit Workflow

Working Directory:

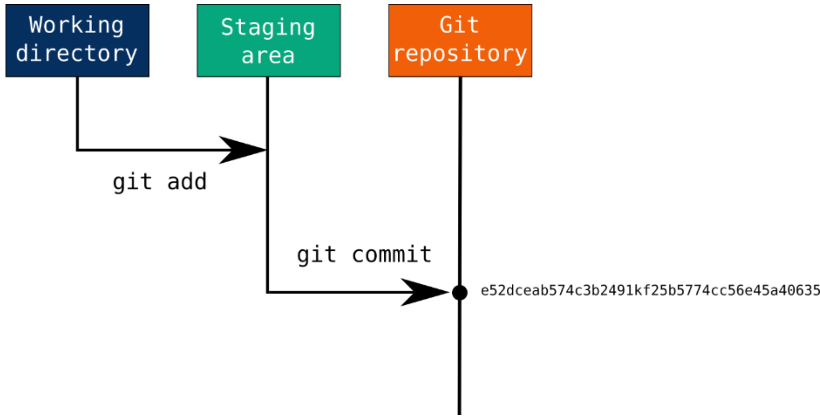
- *Your local workspace where you create, edit, delete, and organize project files.*

Staging Area:

- *A tracked space to prepare and review changes before committing them to the repository.*

Local git Repo:

where Git permanently stores project versions, tracked as commits

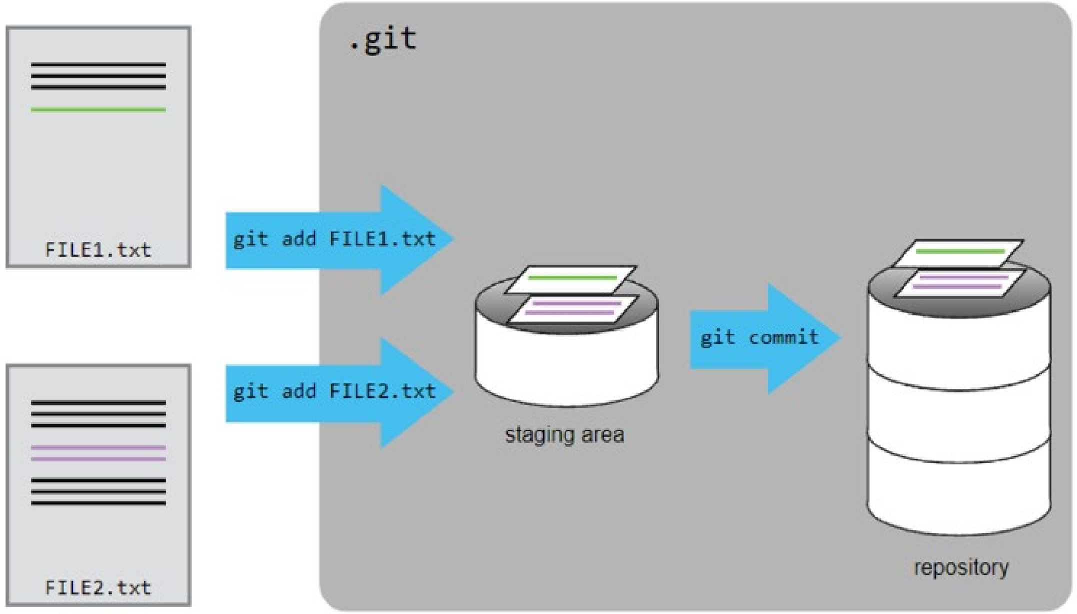


Commit process

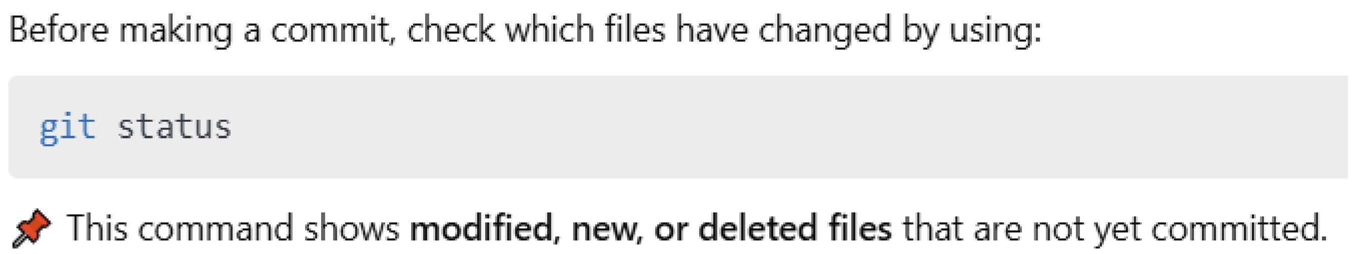
The process:

- From your local working directory, make some changes to a file
- Use the ‘git add’ command to put the file onto the staging area
- Use the ‘git commit’ command to create a new commit in the local repo

Adding your files to git repository



Step 1: Check the Repo status



Before making a commit, check which files have changed by using:

```
git status
```

📌 This command shows **modified, new, or deleted files** that are not yet committed.

The git status command displays **the state of the working directory and the staging area**. It lets you see which changes have been staged, which haven't, and which files aren't being tracked by Git

git status

Use this to check at what stage of the workflow you are at.

This happens when you have added the file.py file in your working directory but haven’t staged your changes yet.
$ git status

```
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)
         modified:   file.py
no changes added to commit (use "git add" and/or "git commit -a")
```

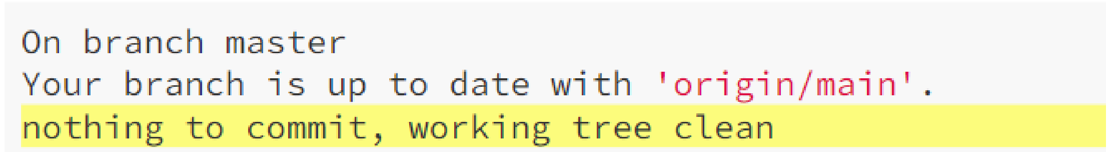
After you stage the file.py file, but before you commit the changes.
$ git add file.py
$ git status

```
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)
        modified:   file.py
```

git status

Use this to check at what stage of the workflow you are at.

This happens when your local repository is in sync with the remote
$ git status



```
On branch master
Your branch is up to date with 'origin/main'.
nothing to commit, working tree clean
```

Step 2: Stage the Changes

Before committing, you need to **stage** the files using `git add`.

- **Add a specific file:**

```
git add filename.txt
```

- **Add all modified files:**

```
git add .
```

📌 Staging **prepares the files** for commit but does not save them yet.

Adding and ignoring files

To stage specific files in your repository, you can name them directly

$ git add file.py other-script.py

Or you can add all of them at one

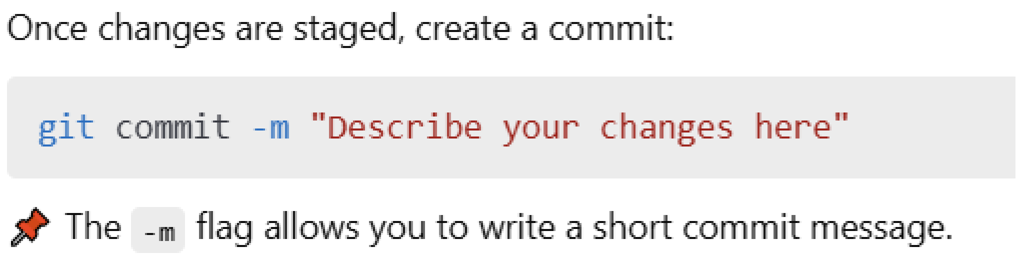
$git add .

You might want to not track certain files in your local repository, e.g., sensitive files such as credentials. But it might get tedious to type out each file that you do want to include by name.

Use a .gitignore file to specify files to always ignore.

Create a file called .gitignore and place it in your repo. The content of the file should include the names of the files that you want Git to **not track**.

Step 3: Make a commit



Once changes are staged, create a commit:

```
git commit -m "Describe your changes here"
```

📌 The `-m` flag allows you to write a short commit message.

Commit your changes

Use an informative commit message

- (Not great) "Analyze data" 😔
- (Better) "Estimate logistic regression" 🎉

Have a consistent style

- Start with an action verb
- Capitalize message

Commits are *cheap*, use them often!

Step 4: Verify the Commit

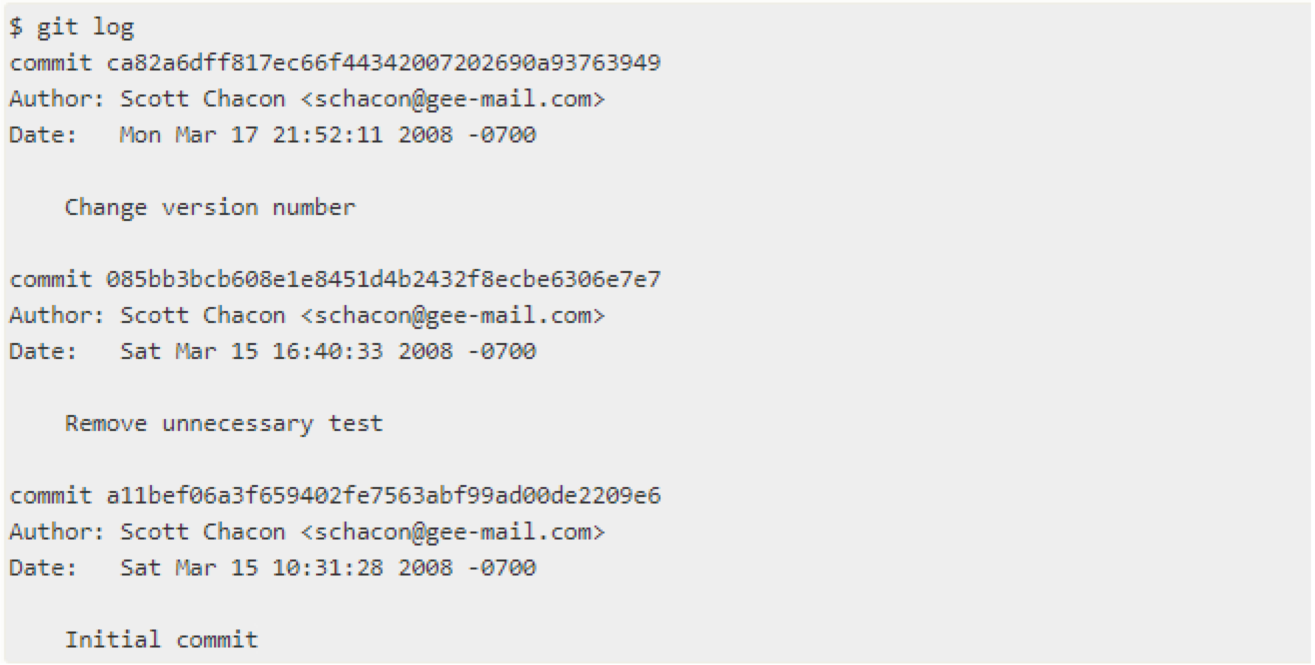
Check the commit history to confirm the changes were saved:

```
git log --oneline --graph
```

📌 This shows a compact **history of commits** in your repository.

git log

When you run git log in this project, you should get output that looks something like this:



```
$ git log
commit ca82a6dff817ec66f44342007202690a93763949
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Mon Mar 17 21:52:11 2008 -0700

    Change version number

commit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Sat Mar 15 16:40:33 2008 -0700

    Remove unnecessary test

commit a11bef06a3f659402fe7563abf99ad00de2209e6
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Sat Mar 15 10:31:28 2008 -0700

    Initial commit
```

git log --oneline # showing every commit in one line

git revert: Undoing mistakes

*Imagine you did some work, **committed** the changes, and **pushed** them to the remote repo. But you'd like to undo those changes.*

Running git revert is a "soft undo".

Say you added some plain text by mistake to file.py. Running git revert will do the opposite of what you just did (i.e., remove the plain text) and create a new commit. You can then git push this to the remote.

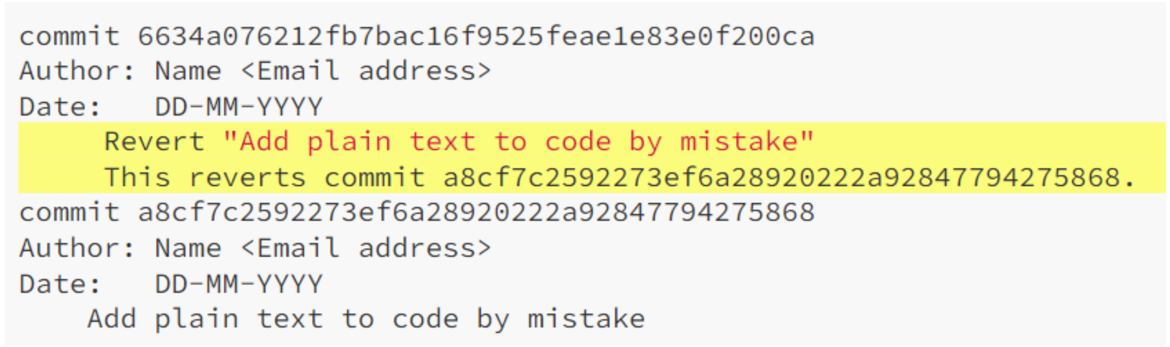
```
$ git revert <hash-of-the-commit-you-want-to-undo>
$ git push
```

Undoing mistakes

git revert *is the safest option to use*

It will preserve the history of your commits

$ git log



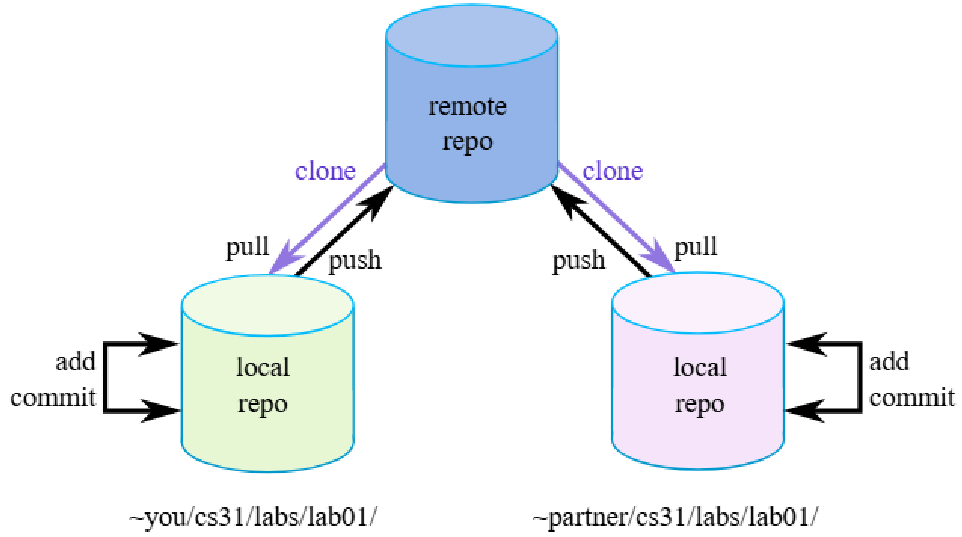
```
commit 6634a076212fb7bac16f9525feae1e83e0f200ca
Author: Name <Email address>
Date:   DD-MM-YYYY
     Revert "Add plain text to code by mistake"
     This reverts commit a8cf7c2592273ef6a28920222a92847794275868.
commit a8cf7c2592273ef6a28920222a92847794275868
Author: Name <Email address>
Date:   DD-MM-YYYY
    Add plain text to code by mistake
```

After commit, where are the changes?

- The changes are saved only in your local repo
- How can you share with anyone else?

Local Repository: Think of it as your private workspace. Commits here are stored on your machine and are not visible to others until you push them to a remote repository.

Git repos: Local repo and remote repo



To share your changes or collaborate with others, you need to interact with a **remote repository**.

Things to note

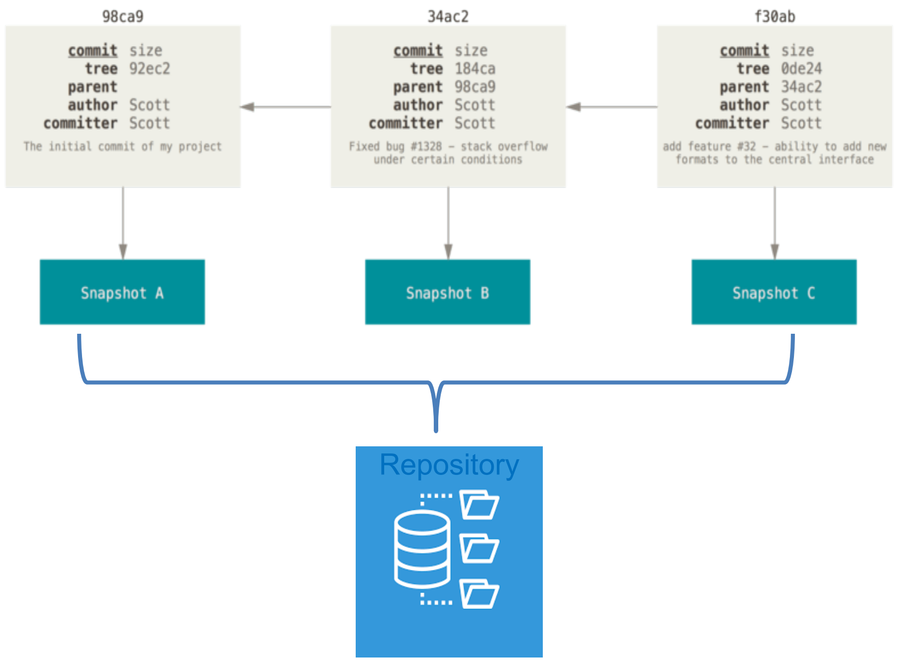
- Final exam: Thursday, 03/20/2025: 3PM-5PM
- Focus on materials after midterm
- Review the quizzes, assignments, and practice exercises in the textbook
- Office hour: Friday (1-4pm)

Why introducing Git and GitHub

- Version control (Tracking Changes)
- Collaboration skills
- Portfolio building
- Industry Standard Tool
    - ❑ Learning Git early gives students a **head start in their careers**.

Last Class: Key concepts on Git

- Repo
- Commit



Last class: Breaking down the Git commit Workflow

Working Directory:

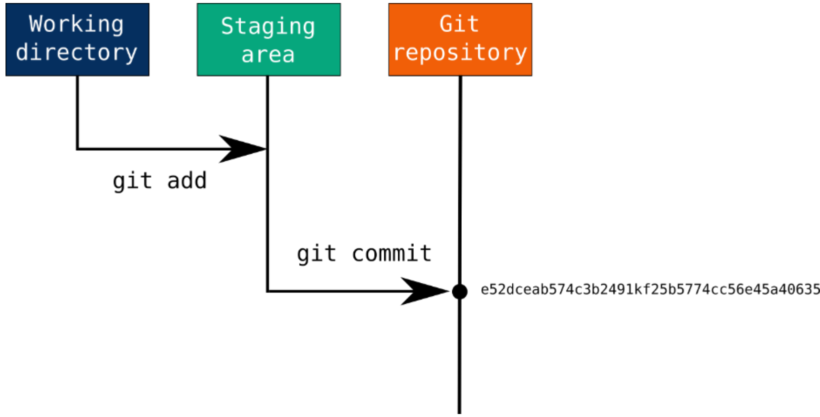
- *Your local workspace where you create, edit, delete, and organize project files.*

Staging Area:

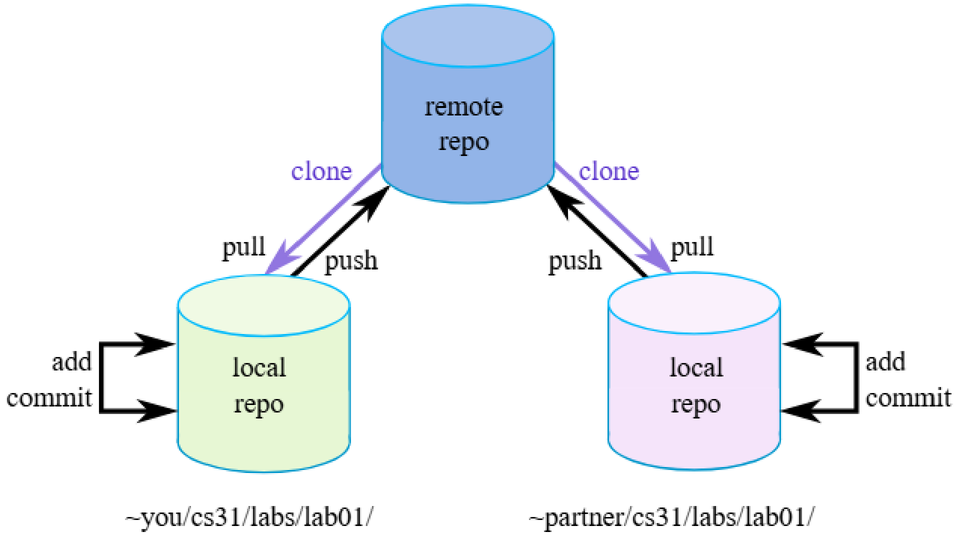
- *A tracked space to prepare and review changes before committing them to the repository.*

Local git Repo:

where Git permanently stores project versions, tracked as commits



Last Class: Local repo and remote repo



To share your changes or collaborate with others, you need to interact with a **remote repository**.

5 What is GitHub?

About GitHub?

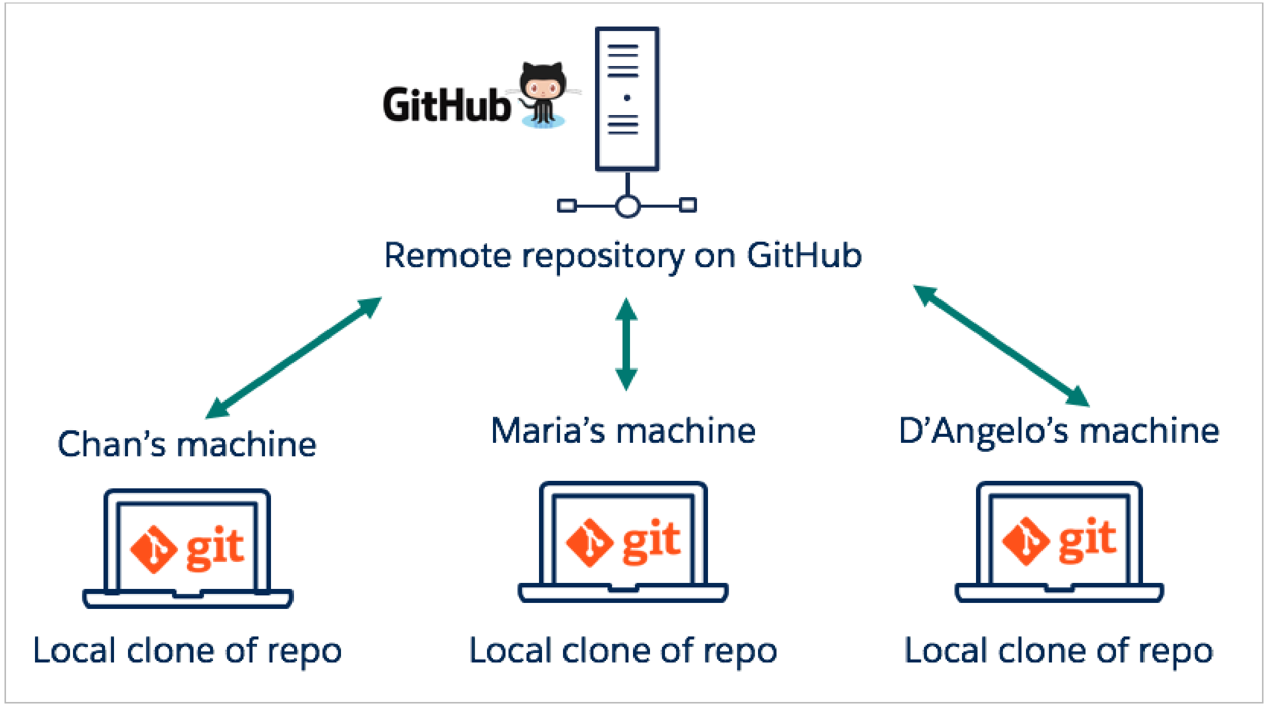
- It is one of the most popular remote repositories used for hosting Git projects.
- GitHub is a cloud-based platform for version control and collaboration using Git.
- It allows multiple users to work on a project by managing code changes efficiently.
- GitHub provides features like pull requests, issues, and actions to streamline software development.

Setting up a GitHub Repo

Steps:

1. **Create a GitHub Account** at https://github.com
2. **Create a New Repository**
    1. Click on **New Repository**.
    2. Enter a repository name, description (optional), and choose visibility (public/private).
    3. Initialize with a README (optional) and choose a license (if applicable).
    4. Click **Create Repository**.

Cloning from a remote server allows teams to work together



Cloning a Remote Rep to Local

Steps:

1.Copy the Repository URL from GitHub.
2.Open a terminal and navigate to the desired directory.
3.Run the command:

```
git clone <repository_url>
```

4.This downloads the repository and sets up a connection to the remote repository.

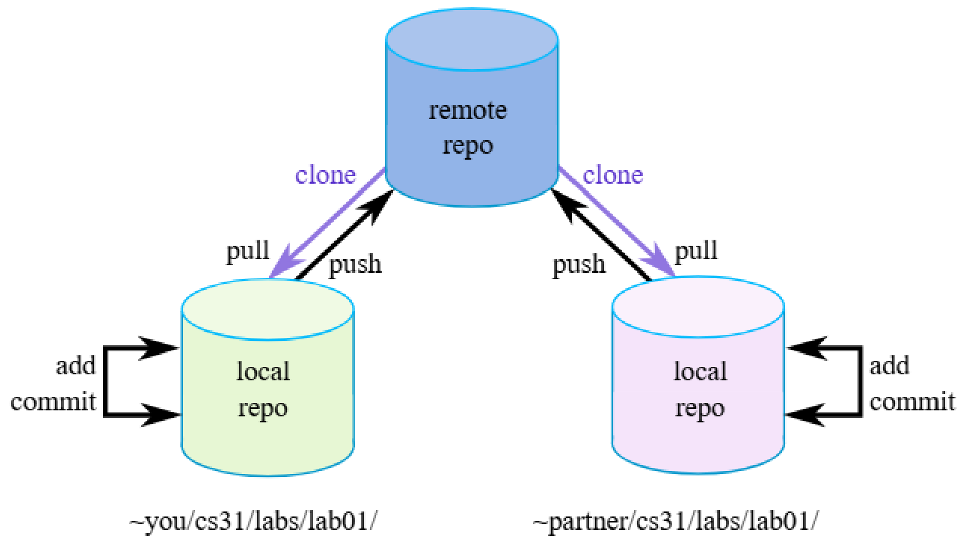
Connecting git with Github

- **Push** your local changes to the remote repo.
- **Pull** from remote repo to get most recent changes.
  - (fix conflicts if necessary, add/commit them to your local repo)

- To fetch the most recent updates from the remote repo into your local repo, and put them into your working directory:
  - `git pull origin main`

- To put your changes from your local repo in the remote repo:
  - `git push origin main`

Git interaction with GitHub



Authentication Issues

When you push for the first time, Git will prompt for a **username** and **password**:

- ❑ **Username:** Your GitHub username
- ❑ **Password:** Your **Personal Access Token**

Since GitHub **no longer supports password authentication**, you must use a **Personal Access Token (PAT)** when pushing to a remote repository via HTTPS.

Generate a Personal Access Token (PAT)

1. Go to **GitHub** → **Settings**: GitHub Settings
2. Navigate to **Developer settings** → **Personal access tokens** → **Tokens (classic)**
   Or visit directly: GitHub PATs
3. Click "**Generate new token (classic)**"
4. Set **Expiration** (choose "**No expiration**" for long-term use, but note that GitHub recommends setting an expiration date for security reasons).
5. Select **Scopes** (Permissions):
   - ✅ `repo` → **Full control of private repositories**
   - ✅ `workflow` → **Access GitHub Actions**
   - ✅ `write:packages` → (If working with packages)
6. Click **Generate token** and **copy it immediately** (you won't see it again).

The whole process

Clone the repository

```
$ git clone URL
```

For example

```
$ git clone https://github.com/javedali99/git-tutorial.git
```

Stage your files

```
$ git add .
```

Commit your changes

```
$ git commit –m “Add example code”
```

Push your changes to GitHub

```
$ git push
```

git pull

Use this to fetch changes from the remote and to merge them into your local repository.

Your collaborators have been adding some awesome content to the repository, and you want to fetch their changes from the remote and update your local repository.

```
$ git pull
```

What this is doing under-the-hood is running a git fetch and then git merge

Getting help on Git Commands

This slide focuses on fundamental Git commands and concepts. Git is a highly robust tool, so don't hesitate to delve deeper and explore its full capabilities.

Getting help:

```
$ git help <verb>
$ git <verb> --help
```

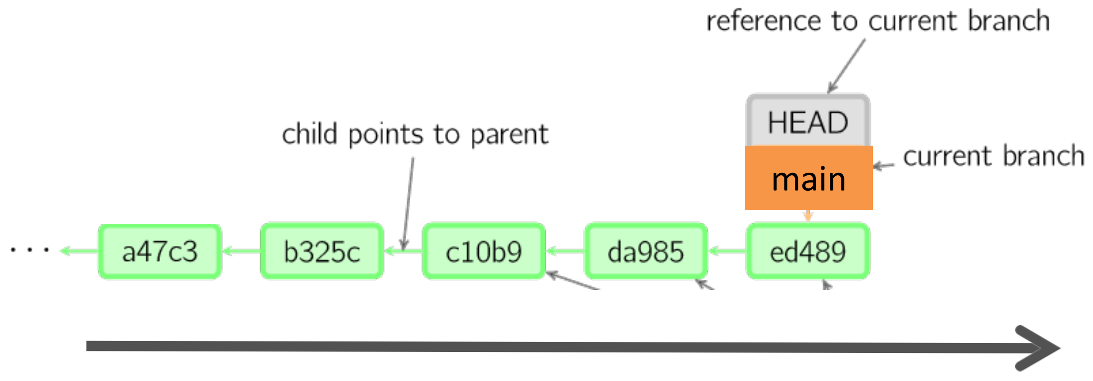
We Will Stop Here

- To keep things simple for beginners, we will not cover advanced topics like branching, conflict resolution, and rebasing.
- However, if you're interested in exploring further, I’ve included additional slides for you to review on your own.

6 Git branching model for collaboration (Skipped)

Main branch

- All commits in git live on some branch
- When creating a new project in Github, the default branch name is main

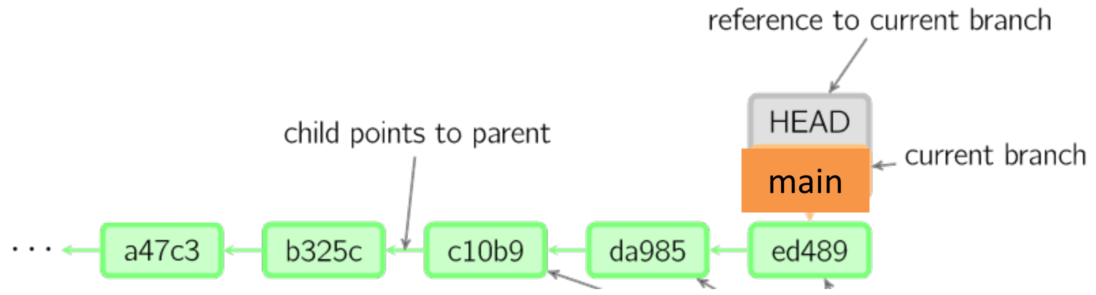


Time going forward

A bunch of commits linked together that live on some branch, contained in a repository

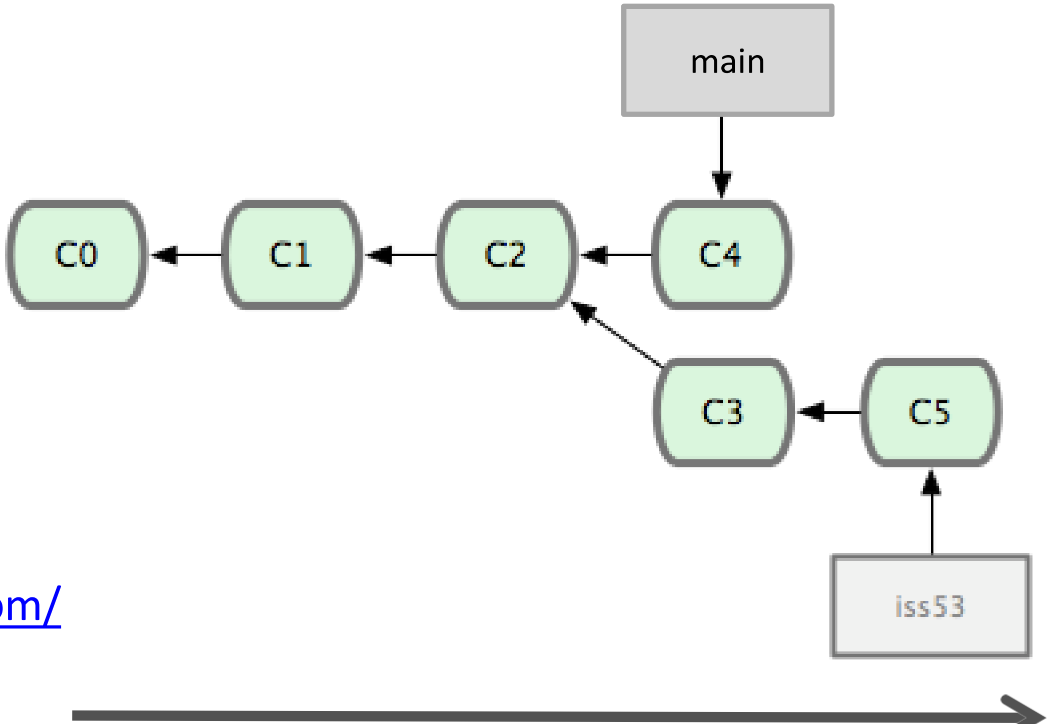
what is HEAD?

- HEAD as the "current branch". When you switch branches, the HEAD revision changes to point to the tip of the new branch.



Time going forward

Key Concepts: Branching off the main branch

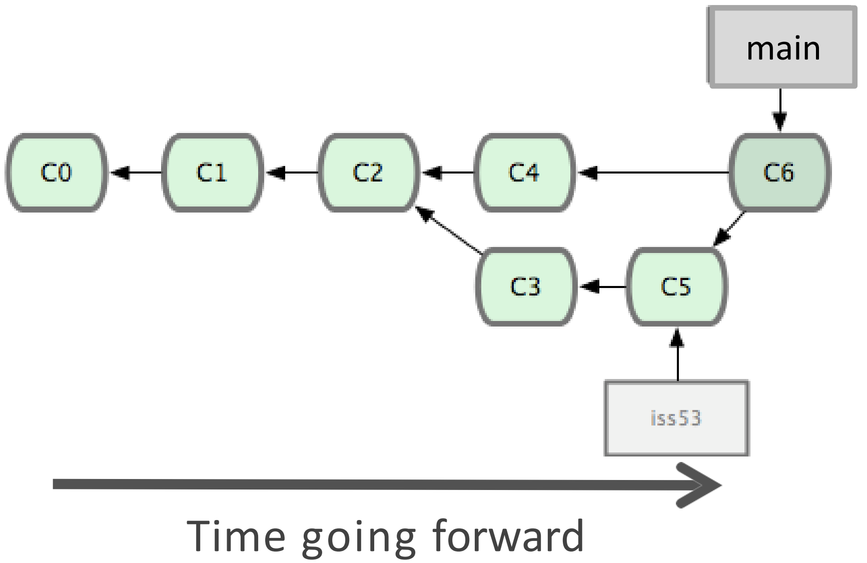


Time going forward

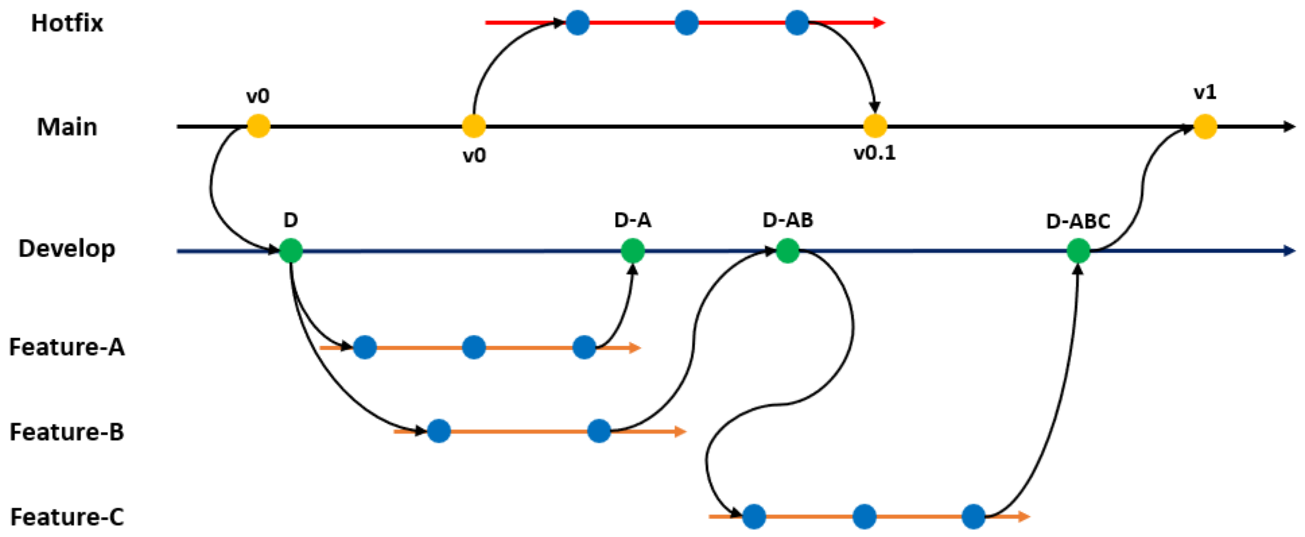
Images from: http://codingdomain.com/ git/merging/

Key Concepts: Merging

- Once you're done with your feature, you merge it back into main



Branching Strategy



How to Collaborate?

1. Preserve the Main Branch:
Avoid direct modifications to files on the main branch. Keep it clean and stable for seamless collaboration.

2. Branch for Your Changes:
Create your dedicated branch when making alterations to your project. This ensures a systematic and organized development process.

3. Commit to Your Branch First:
Before integrating changes, commit them to your personal branch. This allows for individual development without impacting the main branch.

4. Collaborative Code Review:
Engage in thorough code reviews with your group members to ensure the compatibility of your changes. This step is crucial for preventing any adverse effects on your colleagues' work.

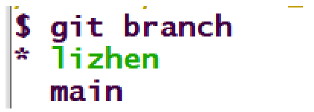
5. Merge with Main After Review:
Once the code review is successfully completed, merge your changes into the main branch. This systematic approach maintains code integrity and enhances overall project stability.

5. Coordinate with Your Colleagues:
Before modifying any files, ensure your colleagues have pulled the latest changes. This proactive step minimizes conflicts and fosters a smoother collaborative development environment.

By adhering to these practices, you contribute to a more organized and collaborative development environment.

Git commands for branching

$ git branch # it will list all the branches in the repo, the branch name in green is the current working one



```
$ git branch
* lizhen
  main
```

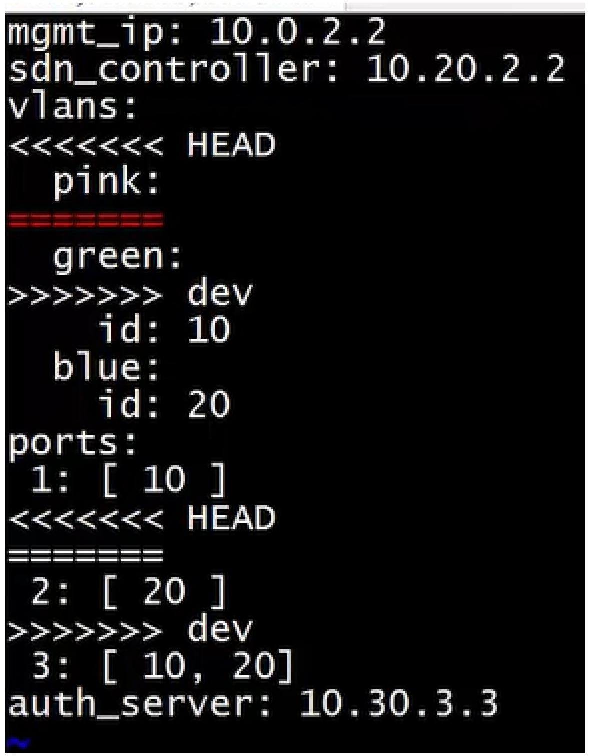
$ *git branch <new-branch>* #Specifying a new branch to be created.

$ git switch <branch name> # switch to a specific branch by pointing HEAD at the branch

$ *git merge <branch name>* #merge the changes in <branch name> to the current working branch

$ *git branch -d <branch name >* # delete a branch

Merge conflict

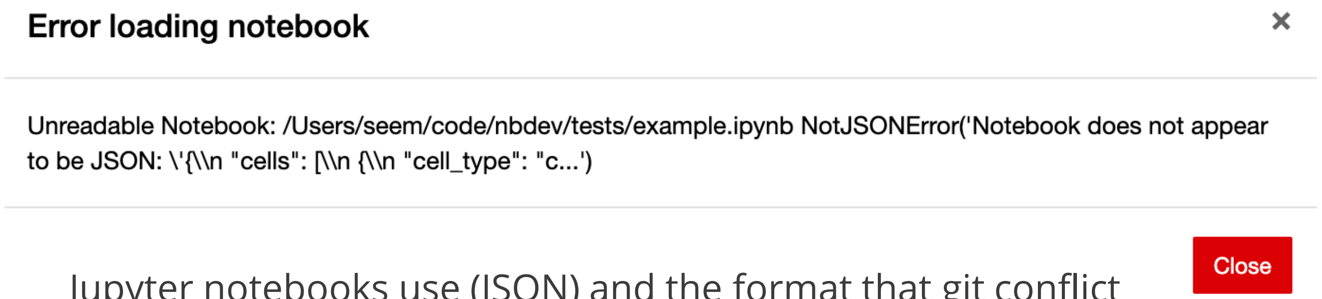


```
mgmt_ip: 10.0.2.2
sdn_controller: 10.20.2.2
vlans:
<<<<<<< HEAD
  pink:
=======
  green:
>>>>>>> dev
    id: 10
  blue:
    id: 20
ports:
 1: [ 10 ]
<<<<<<< HEAD
=======
 2: [ 20 ]
>>>>>>> dev
 3: [ 10, 20]
auth_server: 10.30.3.3
~
```

- The set of equals signs separates out the state of the file in the two branches.
- We need to decide how we want our merged file look like
- When we decide, we delete the Git markers: the equals, and the brackets as well as the text we want to drop

The Jupyter+git merge conflict

if you and your teammate both modify a notebook cell (including, in many cases, simply executing a cell withuout changing it), and then try to open that notebook later:



Error loading notebook ×

Unreadable Notebook: /Users/seem/code/nbdev/tests/example.ipynb NotJSONError('Notebook does not appear to be JSON: \'{\\n "cells": [\\n {\\n "cell_type": "c...')

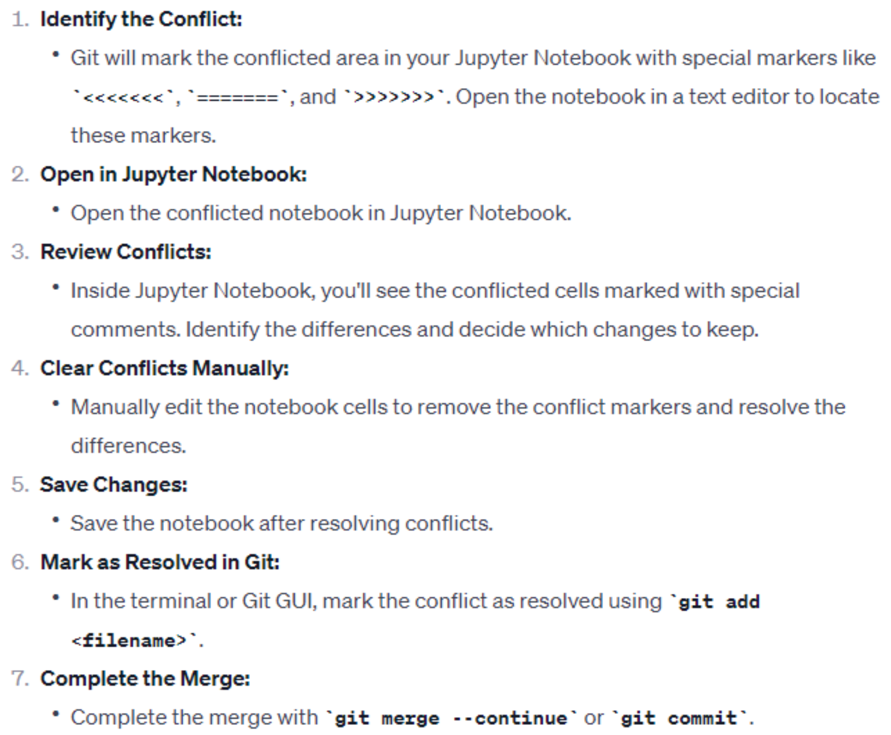
Jupyter notebooks use (JSON) and the format that git conflict markers assume by default (plain lines of text). This is what it looks like when git adds its conflict markers to a notebook:

```
   "source": [
<<<<<< HEAD
    "z=3\n",
======
    "z=2\n",
>>>>>> a7ec1b0bfb8e23b05fd0a2e6cafcb41cd0fb1c35
    "z"
   ]
```

That's not valid JSON, and therefore Jupyter can't open it.

Handling merge conflicts in Jupyter Notebooks

Different compared to plain text files due to their JSON-based structure. Here are steps to resolve merge conflicts in Jupyter Notebooks:



1. **Identify the Conflict:**
   - Git will mark the conflicted area in your Jupyter Notebook with special markers like `<<<<<<<`, `=======`, and `>>>>>>>`. Open the notebook in a text editor to locate these markers.
2. **Open in Jupyter Notebook:**
   - Open the conflicted notebook in Jupyter Notebook.
3. **Review Conflicts:**
   - Inside Jupyter Notebook, you'll see the conflicted cells marked with special comments. Identify the differences and decide which changes to keep.
4. **Clear Conflicts Manually:**
   - Manually edit the notebook cells to remove the conflict markers and resolve the differences.
5. **Save Changes:**
   - Save the notebook after resolving conflicts.
6. **Mark as Resolved in Git:**
   - In the terminal or Git GUI, mark the conflict as resolved using `git add <filename>`.
7. **Complete the Merge:**
   - Complete the merge with `git merge --continue` or `git commit`.

Handling merge conflicts in Jupyter Notebooks

Consider using tools like nbdime or ReviewNB to visualize and resolve notebook diffs during the merge process. It can provide a clearer view of changes and simplify conflict resolution.

Additional Resources

- Official git site and tutorial:
  https://git-scm.com/
- GitHub guides:
  https://guides.github.com/
- Command cheatsheet:
  https://education.github.com/git-cheat-sheet-education.pdf
- Interactive git tutorial:
  https://try.github.io/levels/1/challenges/1
- Visual/interactive cheatsheet:
  http://ndpsoftware.com/git-cheatsheet.html