

# Software Engineering COMP1035

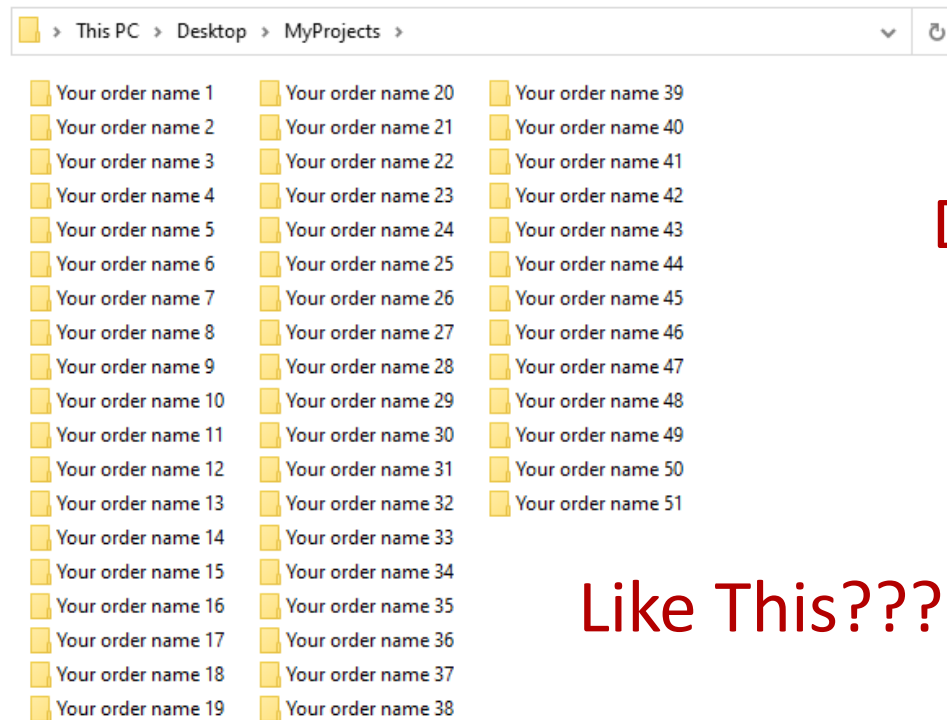
## Lecture 02

### *Git Projects*



# Work As A Group?

How to work on the same project for multi developers?



Like This???

Definitely NOT through WeChat



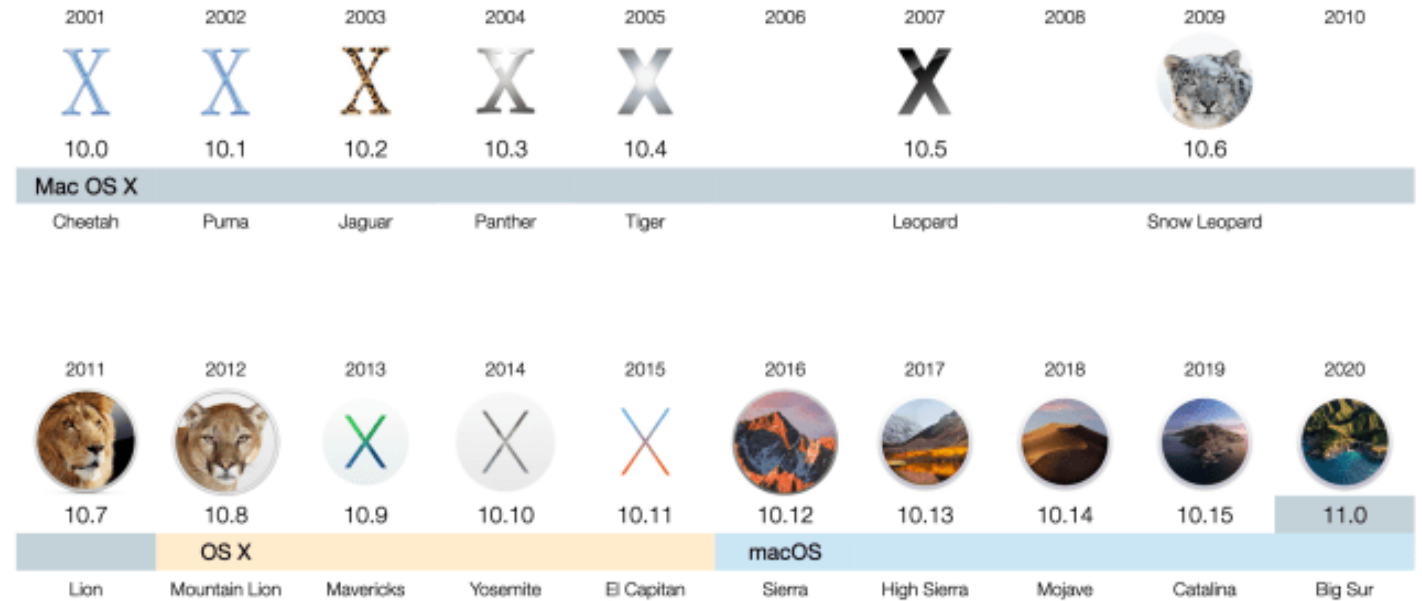


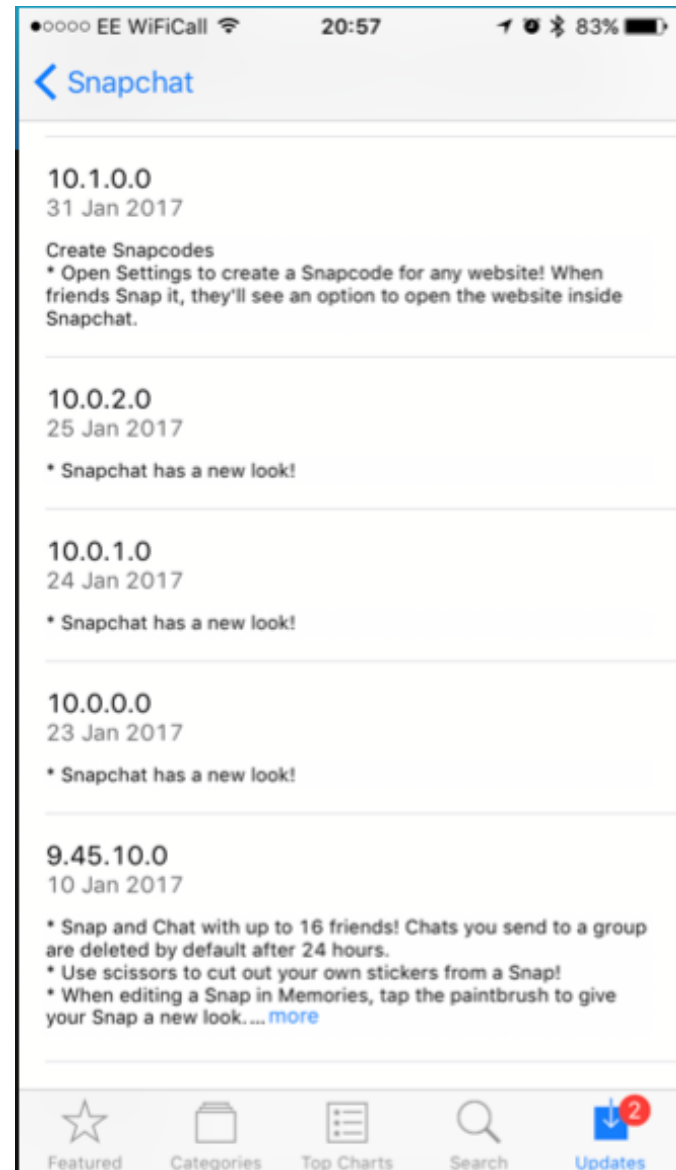
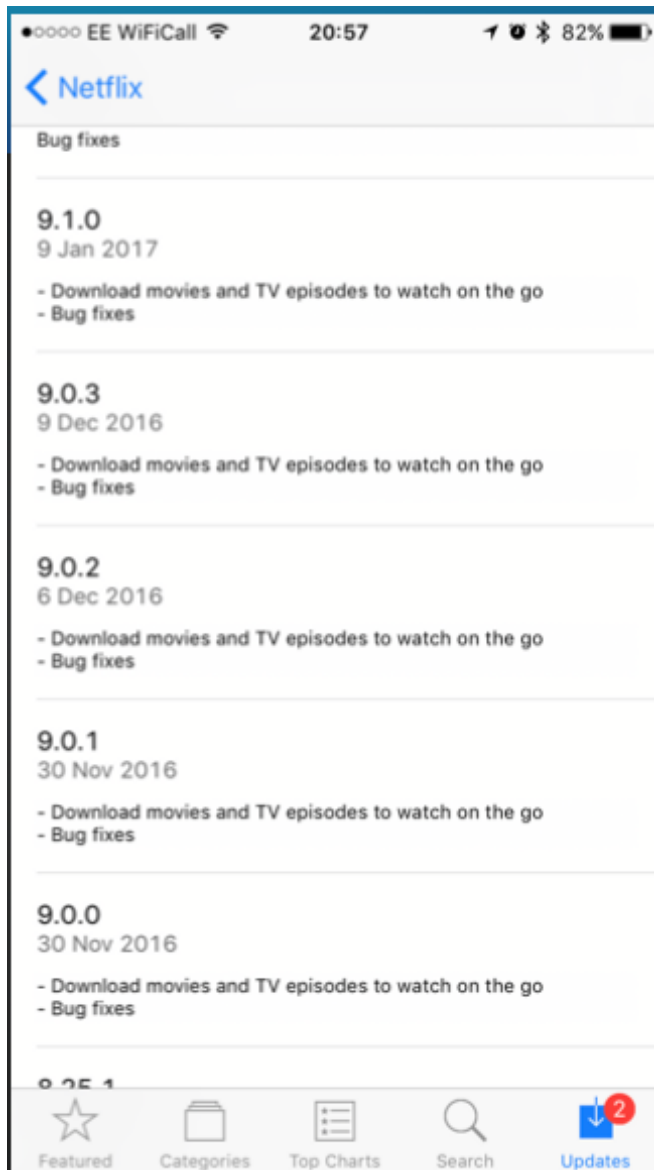
# How Projects Use Git

# Updated Releases (MacOS)

- <https://en.wikipedia.org/wiki/MacOS>

## macOS History

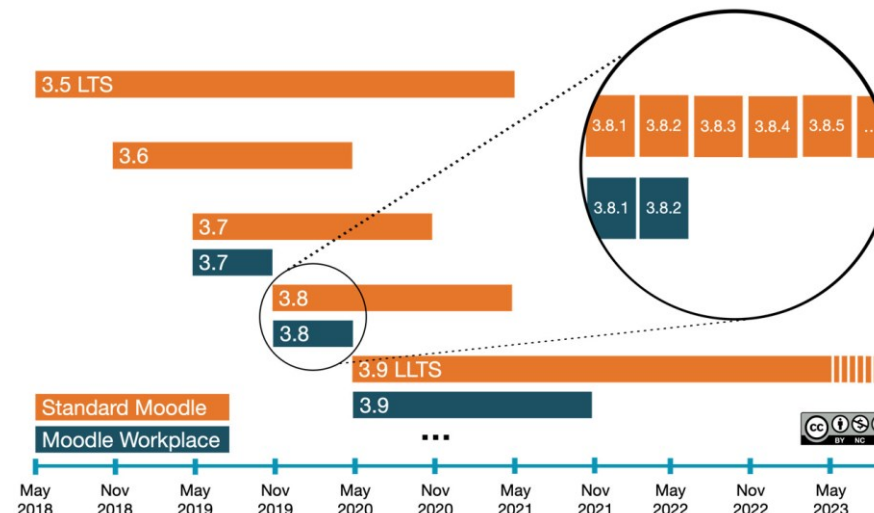




# Moodle Versions

*“Since version 2.0, Moodle aims to release the new major version every six months or so”.*

Minor versions are released every two months, including both bugs and security issues fixed.

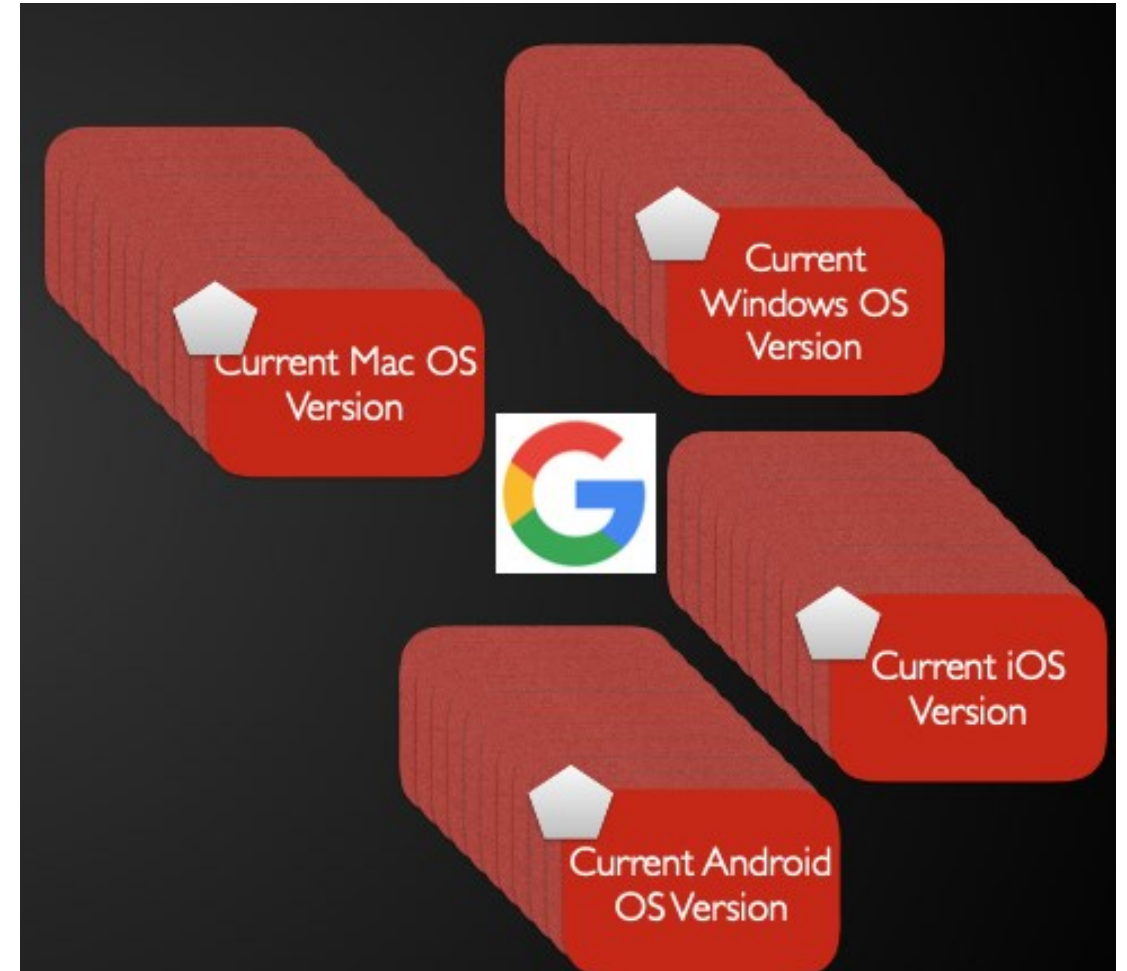


[https://docs.moodle.org/dev/Moodle\\_versions](https://docs.moodle.org/dev/Moodle_versions)

# Versions of Google Chrome

---

- Issues new versions on a regular basis.
- Has trial-test releases before each public release.
- Need to build each release to multiple platforms.



# Chromium Dev Channel

---

- Stable versions – currently version 80.0.3987.87.
- Beta versions – version 66.0.3359.33 (Official Build) beta.
- Dev versions – more often than weekly.
- Canary builds – are the bleeding edge. Released daily, this build has not been tested or used, it's released as soon as it's built.
- Other builds: “**If you're absolutely crazy**, you can download the latest working (and that's a very loose definition of working) build from [download-chromium.appspot.com](https://download-chromium.appspot.com).”

<https://www.chromium.org/getting-involved/dev-channel>

<https://www.thewindowsclub.com/chrome-stable-beta-dev-canary>



# Overview

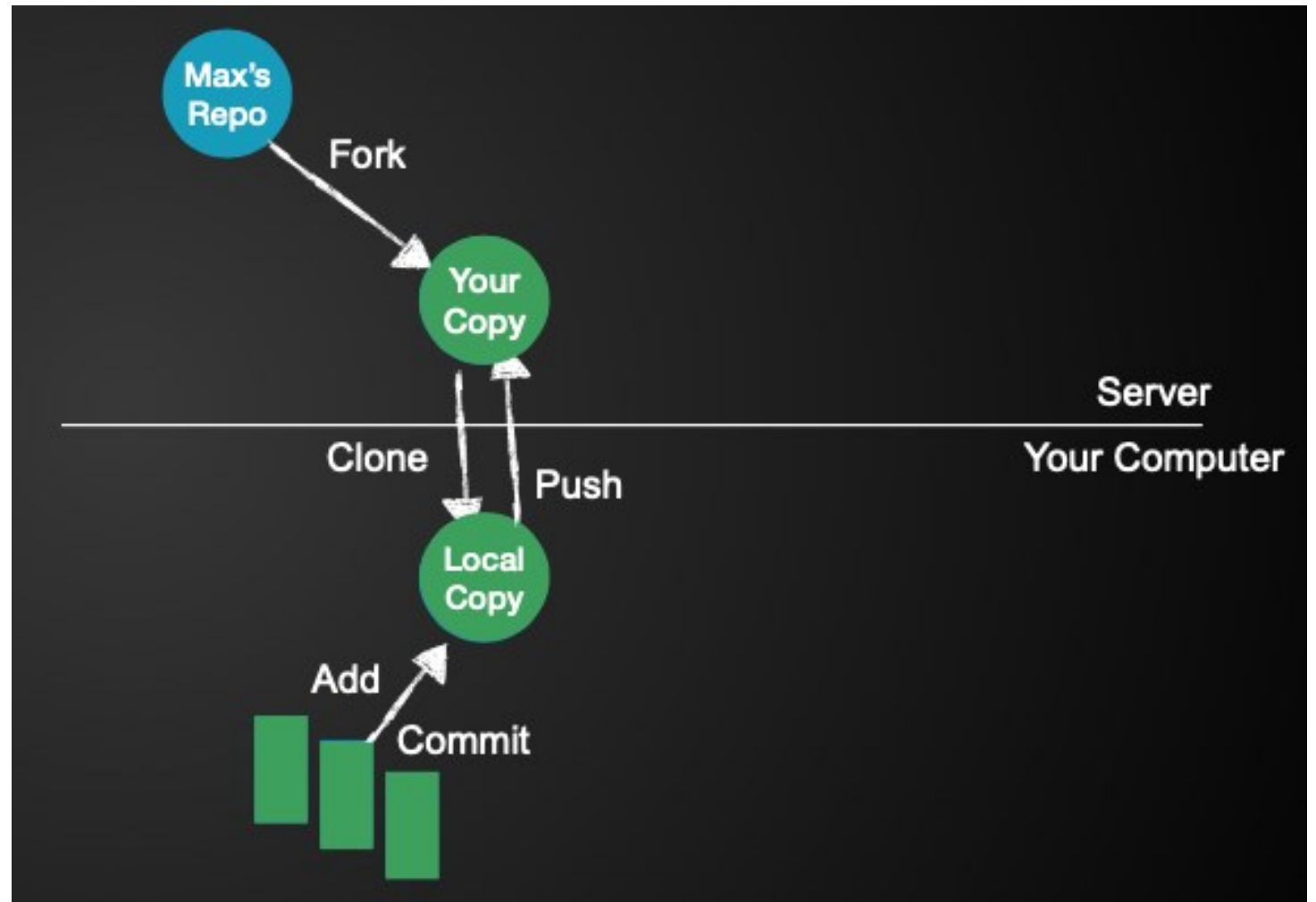


<https://csprojects.nottingham.edu.cn/>

# Semester 2

---

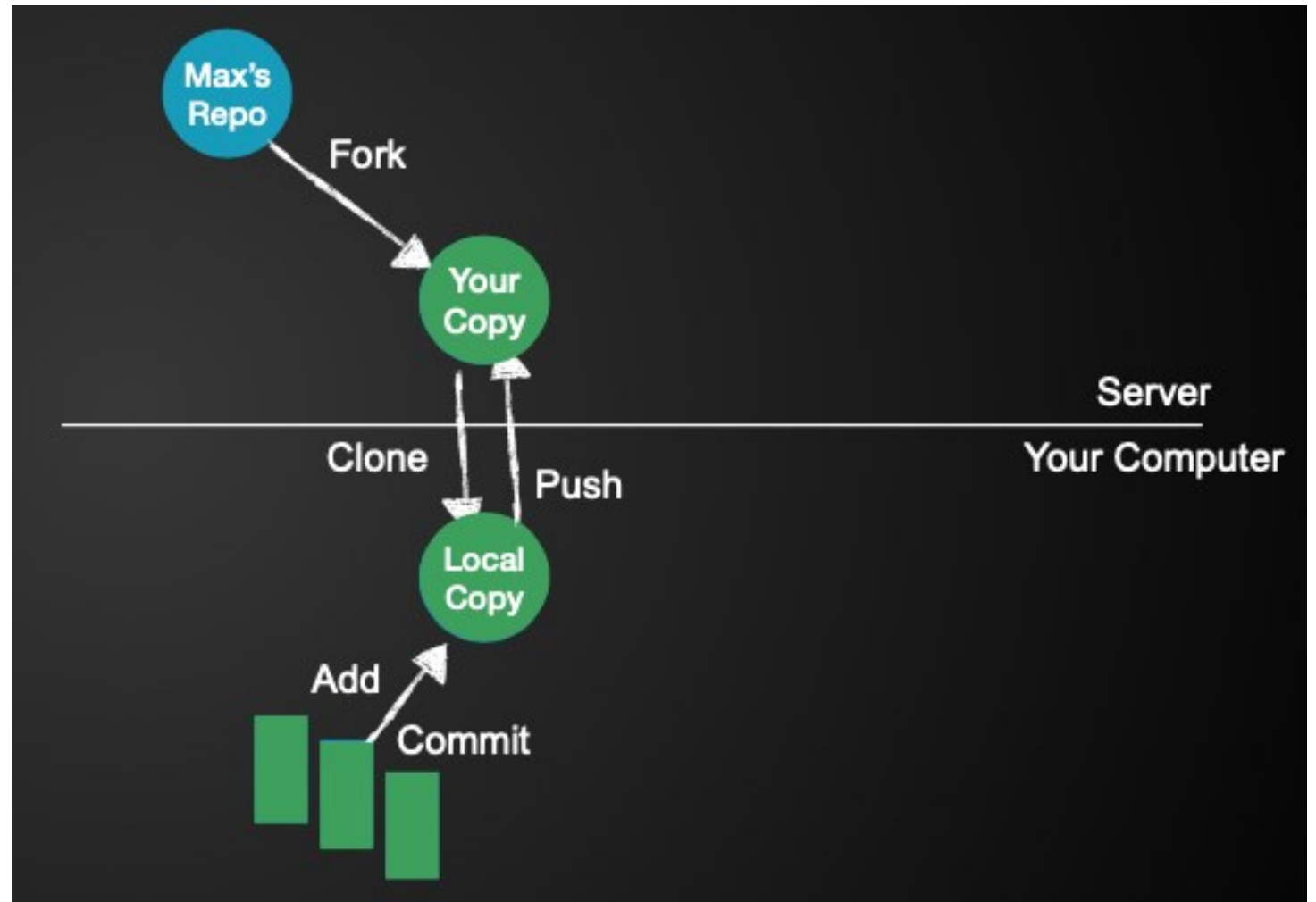
- You will be learning:
  - Fork
  - Clone
  - Status
  - Commit
  - Push
- This was an individual repository.



# Semester 2

---

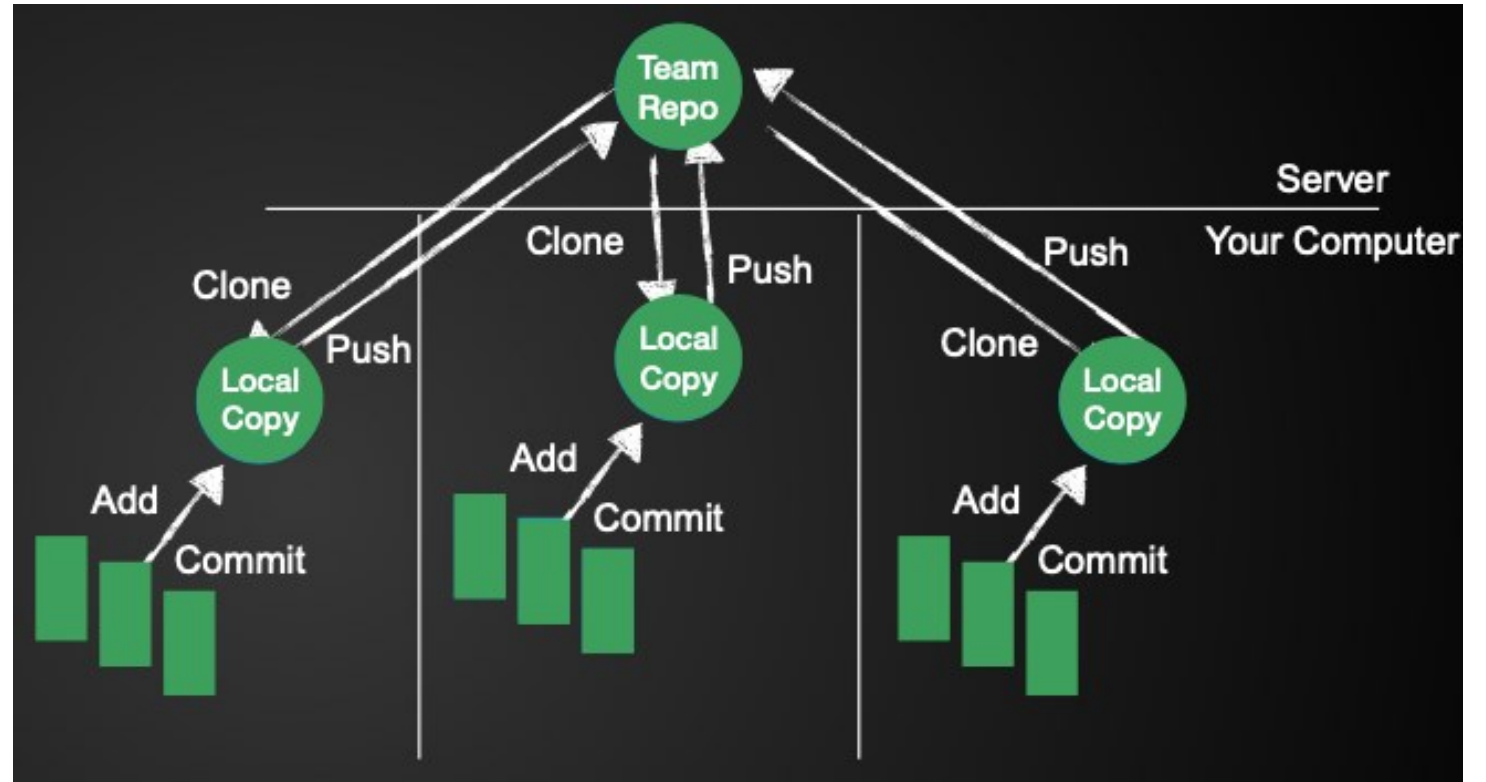
- A fork is a copy of a repository.
- You'll learn in Lab-01.



# Semester 2

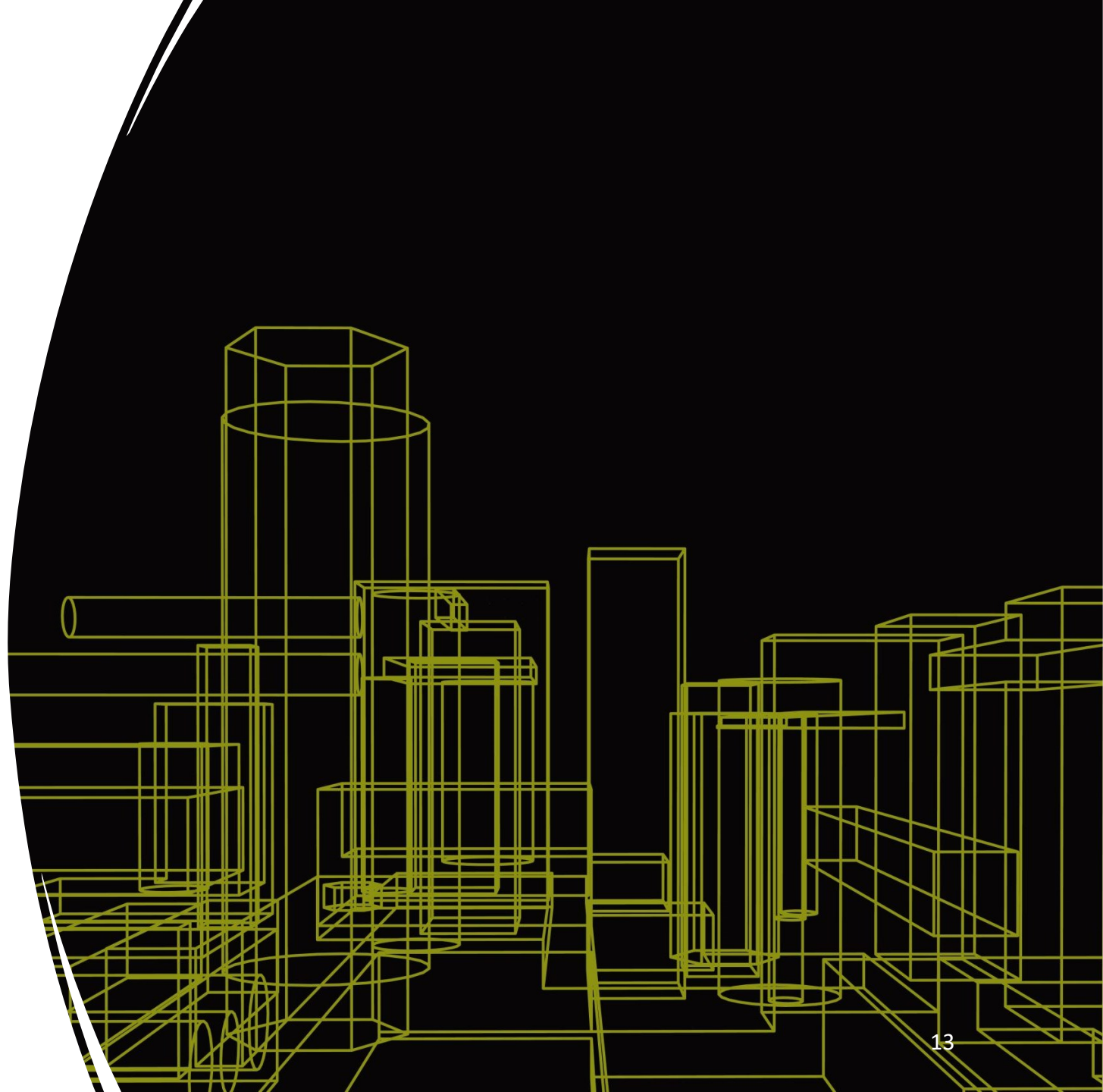
---

- Team actions
  - Pull before Push.
  - Conflict resolution.



# Some Fundamentals

---



# Git is Designed to Protect Code

(hence **Version Control**)

---

- It will keep a history of everything.
  - It's hard to lose a file.
  - You can always find an old copy.
- It's designed to stop multiple people overwriting changes by mistake.
  - It will tell you when someone else has changed things.
  - And ask you integrate them before you submit yours.
- It lets you work on copies (branches).
  - So that you aren't 'messing with the main version.
    - We're going to do branching later in this term.

# You See a 'View' (called HEAD)

---

- A git folder has a complex structure of files behind it.
  - See the `.git` hidden folder!
  - This may include many versions of the same code.
  - Could have many 'folders', which could have different or additional files in it.
- If you never branch or anything, it's not that complex.

# You See a 'View' (called HEAD)

---

- In abstracted terms
  - It has the main code folder and a staged/working folder.
  - `git add`: you are saying 'I want to move this file to the main code folder'.
  - `git commit`: you commit it into the main code folder.
  - It's a bit more complex than that because it remembers every commit you made ever too.
- In your 'view' – you see whichever version you are 'working on'.
  - Based on which branch the `git checkout`.
  - With the updated files you've edited before you add/commit.



# Git Folders Have a History

---

- At any time, you can step backwards to a previous commit.
  - Perhaps to before you broke it.
- You could roll back an entire git folder if you really broke it.
- You can extract a single file out of an old commit too.
- Or you can checkout an old commit >> a new branch.
  - You can play with it there, if that helps figure something out.
    - And then just delete the branch later (for cleanup).
- It's designed to keep your code-base safe.

# Git Folders Clone a Remote Git Folder

---

- When `git push` and `git pull`:
  - Git is doing a comparison to see what the difference is between yours.
- This means you can have an official source on e.g., a server.
- And you have a working copy, which you can dump if you want.
- Or when you think your copy is safely correct.
  - You can update the official source.

# Different Group Members

---



# Designed to Support Multiple People

---

- Multiple people have their own copy, and **git push** updates to server.
- Git will automatically handle anything that's easy.
  - Like you added a file – so it'll just push it.
- Even when someone else has edited a different bit of the same file.
  - Git will say – those are different bits, so I can merge those easily.
- And when it can't - it'll ask you to do it “intelligently”.
  - Before it pushes, it grabs the remote changes.
  - It combines both changes into one file, that you must edit before you can push.
  - When you have resolved it, then you can **git push**.

# What's Going On

---

Git Push	"I'd like to push my changes please."
	"Sorry Mary – cannot do. Jeff changed a few things since you pulled. Have a look first."
Git Pull	"Let's have a look then".
	"OK, here's the things Jeff changed, mushed with yours. Look at that file in particular and let me know when it's all working at your end".
[edit file]	"Ah I see, I'll combine these bits. Like this ...".
Git Push	"OK try that".
	"Cheers Mary, I'll make sure other people pull this shiny code before they push".



# .gitignore

---



# .gitignore

---

- This is a list of types of files that git will ignore.
- This means you can get git to only manage source code files.
  - And ignore all build files, .exe's etc.
  - Can also ignore any private files – perhaps a txt file that has an authentication token in it.
- Very useful for projects.
- You can find all sorts of .gitignore files for different programming languages online.

# Git tag

**git tag -a v1.2 -m "version 1.4"**

---

- Basically, gives a friendly name to a commit.
- This can be a working version that is ready to release.
- This can be a version you want to show someone (like a supervisor).
- Use `git push -follow-tags` to make sure the server gets that tag.



Why Doesn't E.g., Dropbox Work  
Instead?

Why Don't Collaborative Editors (Like Google Docs) Work for This?



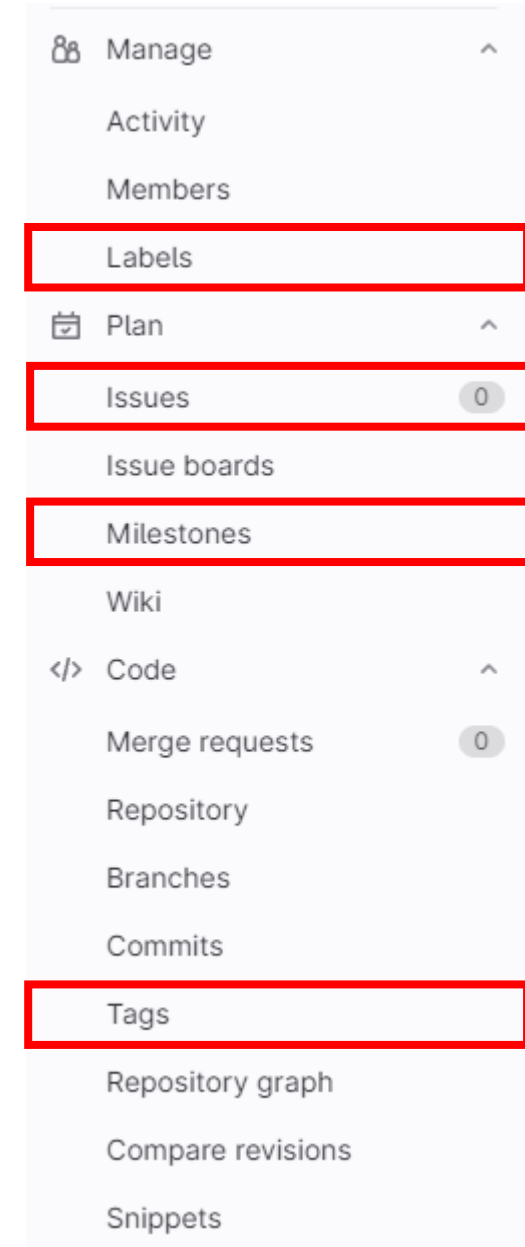
# Milestones, Issues, and Labels


---

# Milestones, Issues, Labels

---

- Can use milestones to track progress.
- You can associate a milestone with issues and pull requests.
- Label can help you to organize and tag your work so you can track and find work items.





# Multi-Users Demonstration Video

---

Available at Moodle page under  
Week 1: Lecture 02



# Summary

- Introducing Git as a project management tool to enhance transparency and collaboration within the group.
- Exploring version control - understanding conflict resolution and its solutions.
- Introduction to fundamental Git commands such as pull, push, commit, add, status, etc.
- In-class demonstration of Git usage – to be further practised during the computing lab session.

