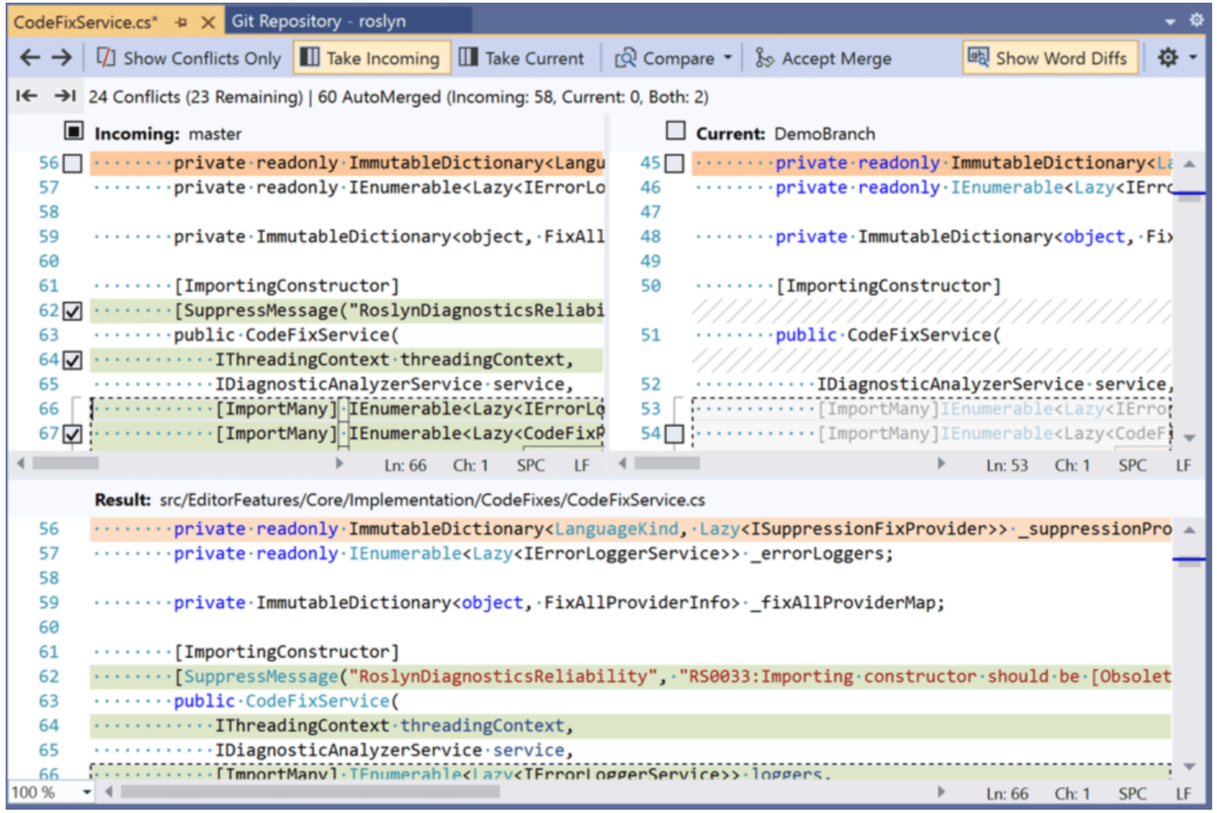
# Outline

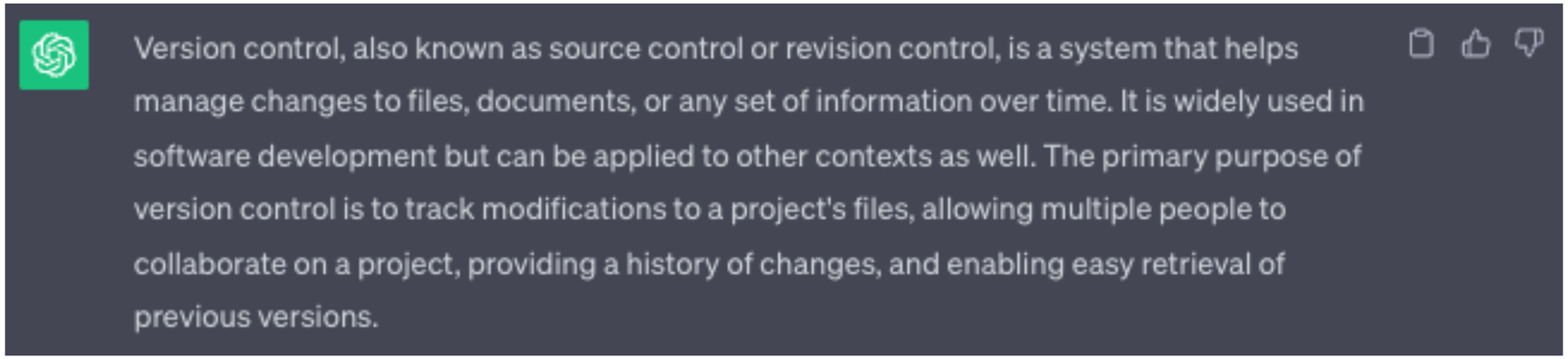
**Planned topics for this lesson:**

- Version (source) control
  *Why we need it? How should we use it?*
- Types of version control
  *Centralised and distributed*
- Version control software
- Lab: Fun with GitHub
  *Get your student account, and git "away"*



# What is version control?

- **A system that records changes to a file or set of files over time, allowing you to recall or access specific versions later.**
- Also known as revision control or source control
- Keep track of changes, by whom and when
- Fundamental tools for developing software projects



# What is version control software?


SOME OF THE POPULAR VERSION CONTROL SOFTWARES [L - R] CVS, SUBVERSION, git (GitHub), Team Foundation Server, Mercurial





BUILT IN VERSION CONTROL AVAILABLE IN A VARIETY OF SOFTWARE PRODUCT

# Why do we need it?

- Backup software source
  - Rollback to previous version
- Keeping a record of who did what and when
  - Know who to praise and who to fire
- Collaborating with the team
  - Know who to praise and who to fire
- Troubleshooting
  - Analyse the change history to figure out what cause the problem
- Statistic
  - Find out who is the most productive

# What should you check in?

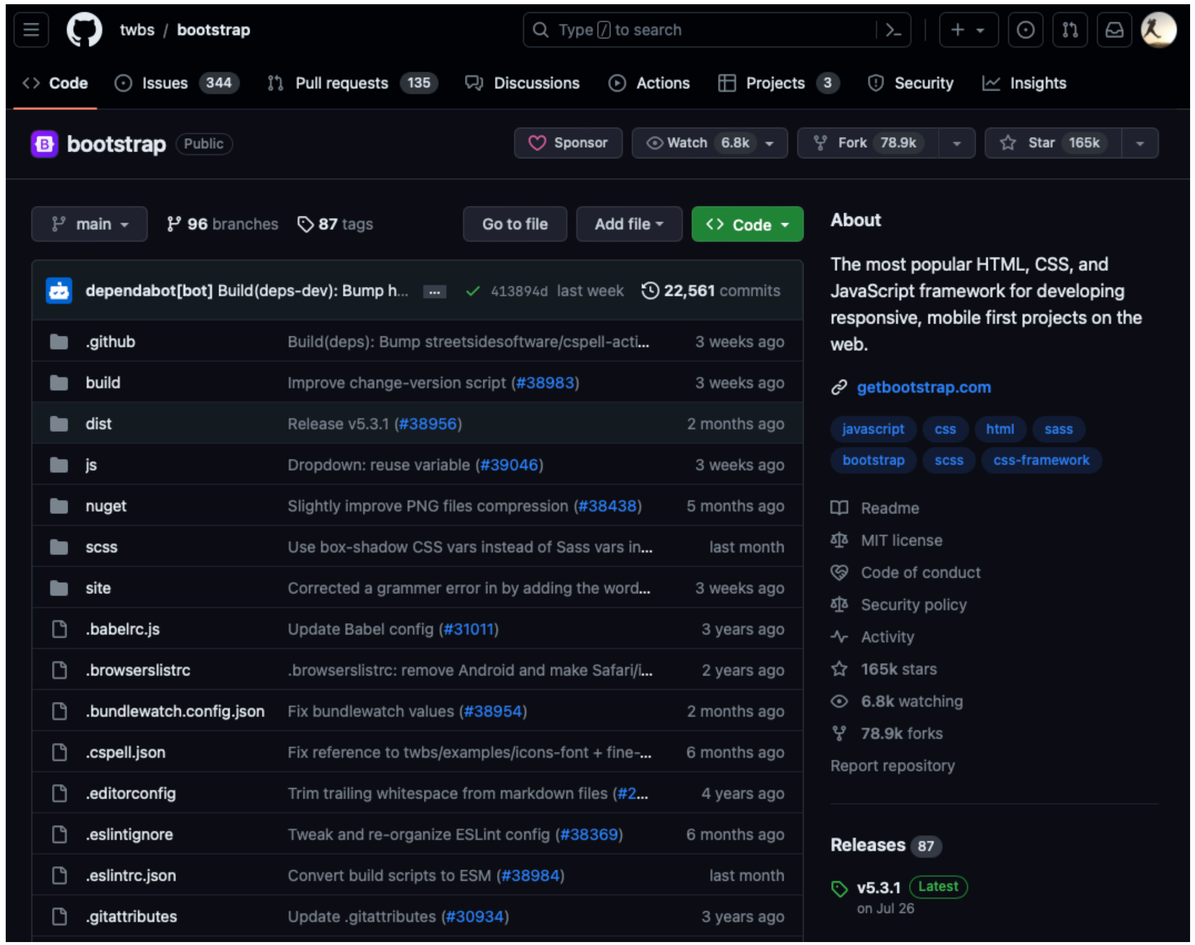
- Everything that influence the build

  CONFIGURATION FILES, FILE ENCODINGS, BINARY SETTINGS, ETC.

- Everything to setup the project from a clean checkout / fork

  SOURCE CODES, DOCUMENTATIONS, MANUALS, IMAGE FILES, DATASETS, ETC.

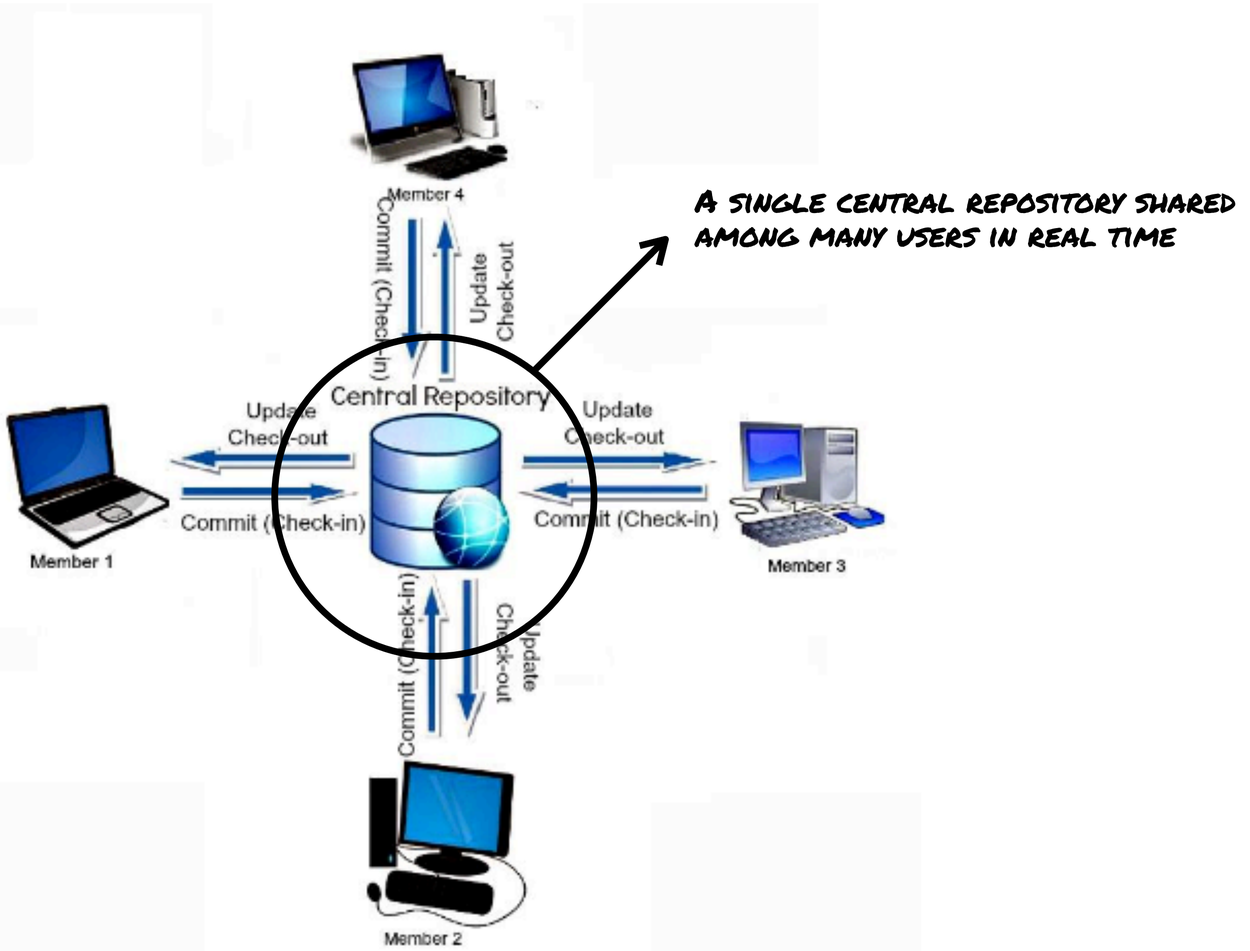
- What about IDE “Junk” files?



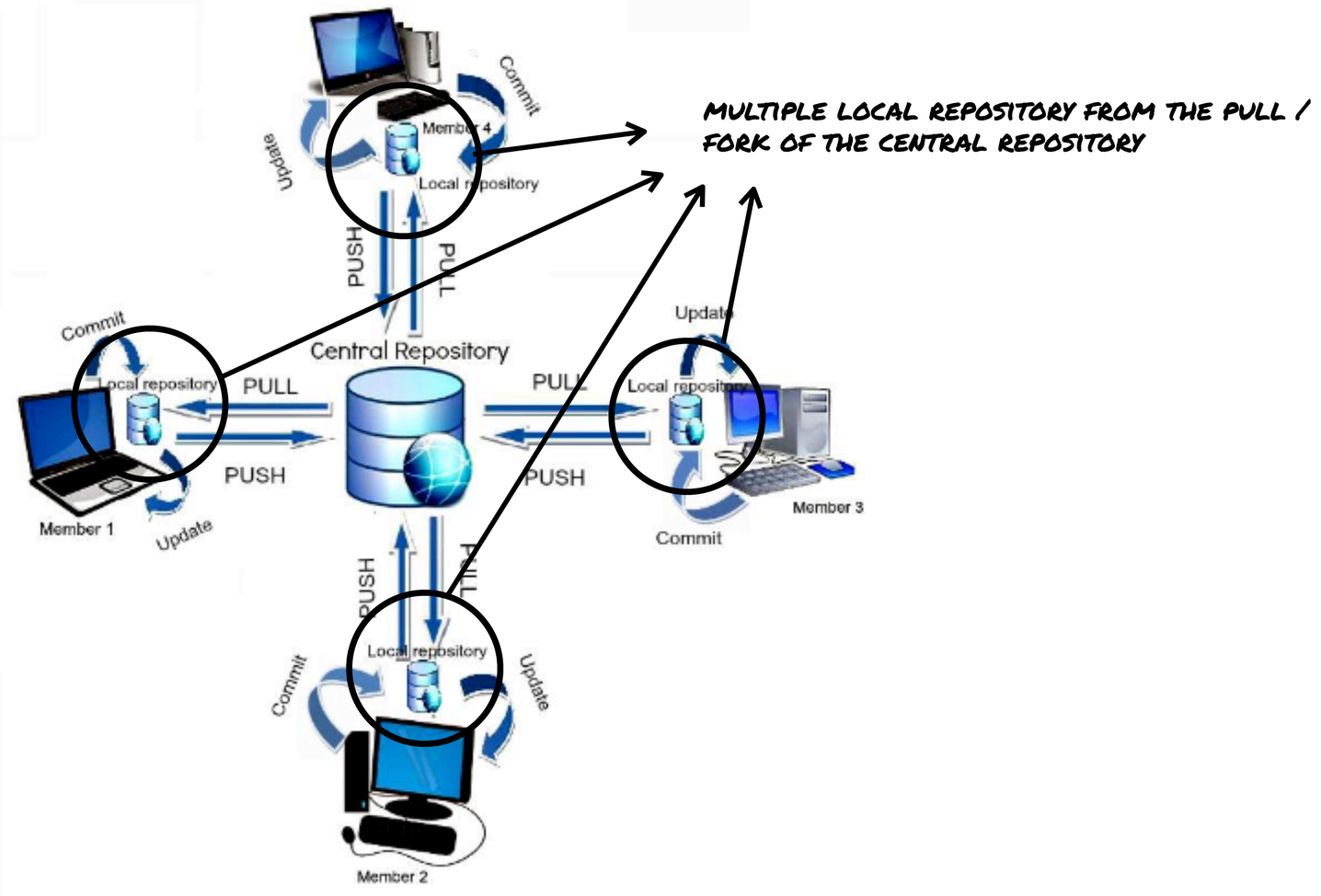
# What should you **NOT** check in?

- Binaries
  JAR FILES, OR ANY OTHER "BUILD" FILES FROM THE SOURCE
- Intermediate files from build / compilation
  .PYC (PYTHON) OR .O (FOR C), ETC
- Files which contain “absolute path”
- Personal Preferences / Settings
  SPECIFIC SETTING ONLY FOR YOUR MACHINE

# Centralised Version Control System



# Distributed Version Control System

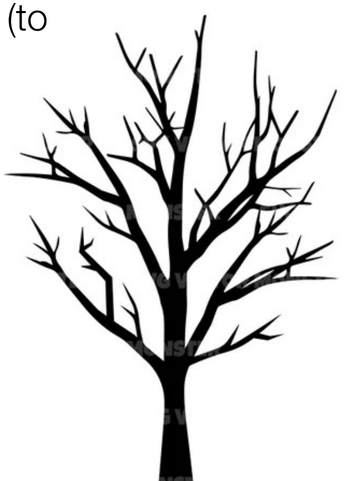


# Subversion


- Code centralised in a repository
- Check out a working copy onto your machine
  - General, you don't have the entire repository checked out in subversion, you only check out a specific branch
- Make changes to it
- Changes committed back to a central repository - "normally" with useful comments
- Maintain a change log (Who, When, What)

# Subversion

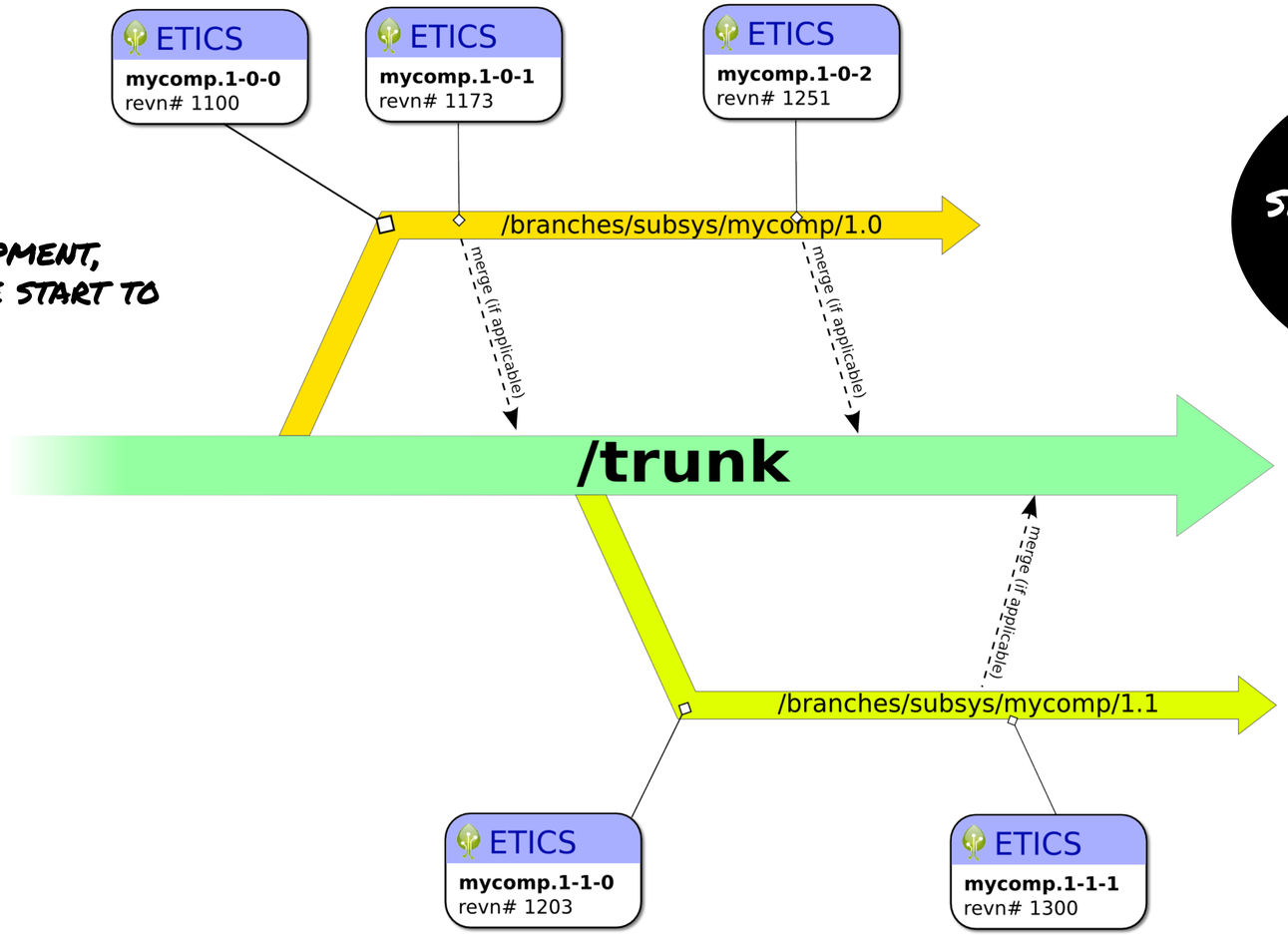

- Subversion is like a tree
- A tree has a trunk and some branches
  - Branches grow from the trunk, and thinner branches grow from thicker branches
- If the trunk is sick, so are the branches, and eventually the whole tree can die
- If a branch is sick, you can cut it, another one may grow
- If a branch grows too much, it may become too heavy for the trunk, and the tree might fall down
- When you feel your tree, your trunk, or a branch is nice looking, you can take a picture of it (to remember how nice it was that day)



# Subversion

**Trunk:**

*Main body of development, originating from the start to the finish*

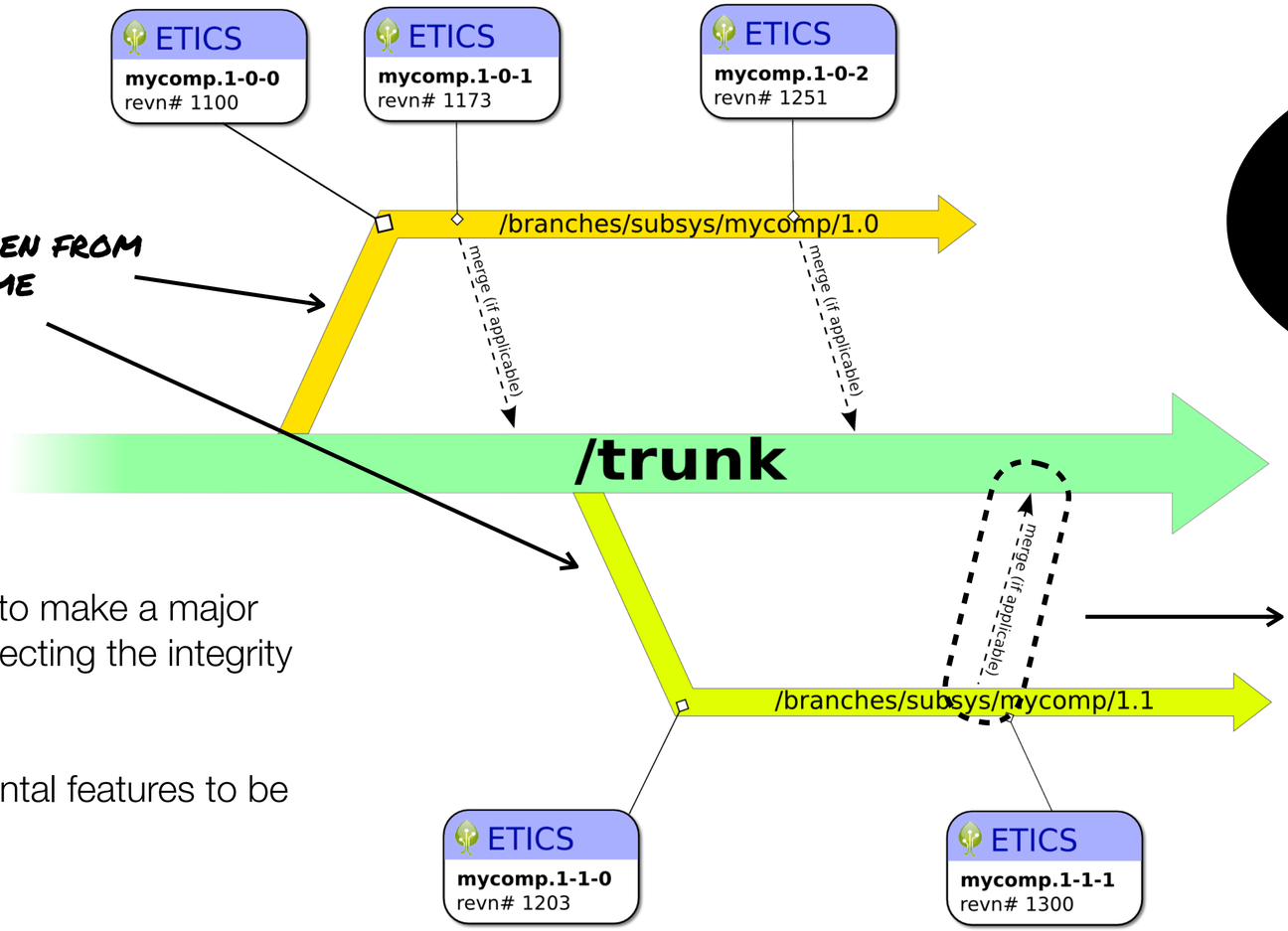


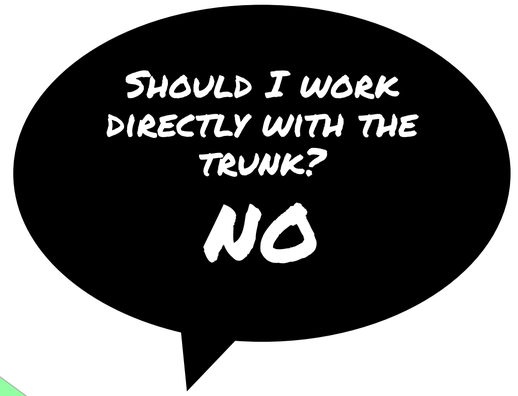

*Should I work directly with the trunk?*

# Subversion

**Branch:**

COPY OF THE CODE, TAKEN FROM A SPECIFIC POINT IN TIME

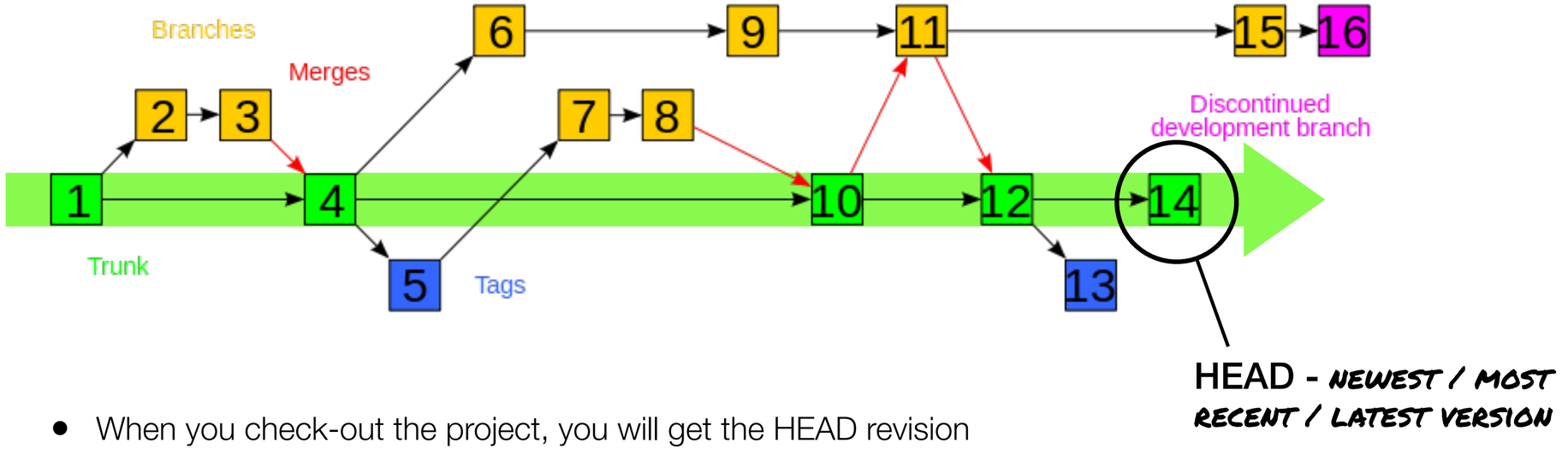



- Allows a developer to make a major changes without affecting the integrity of the trunk
- Allows for experimental features to be tried and tested

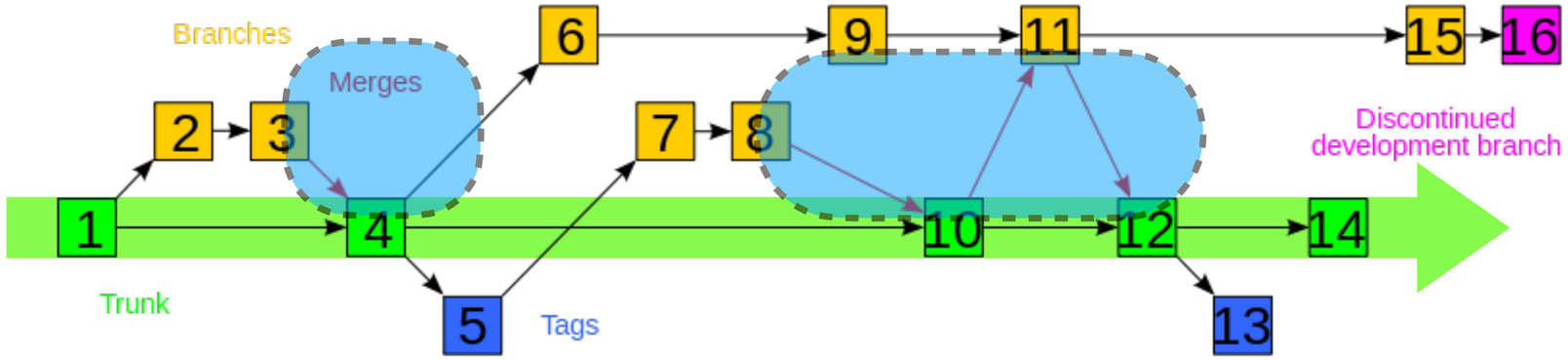
If everything works as planned, then merge the branch back into the trunk

# Subversion



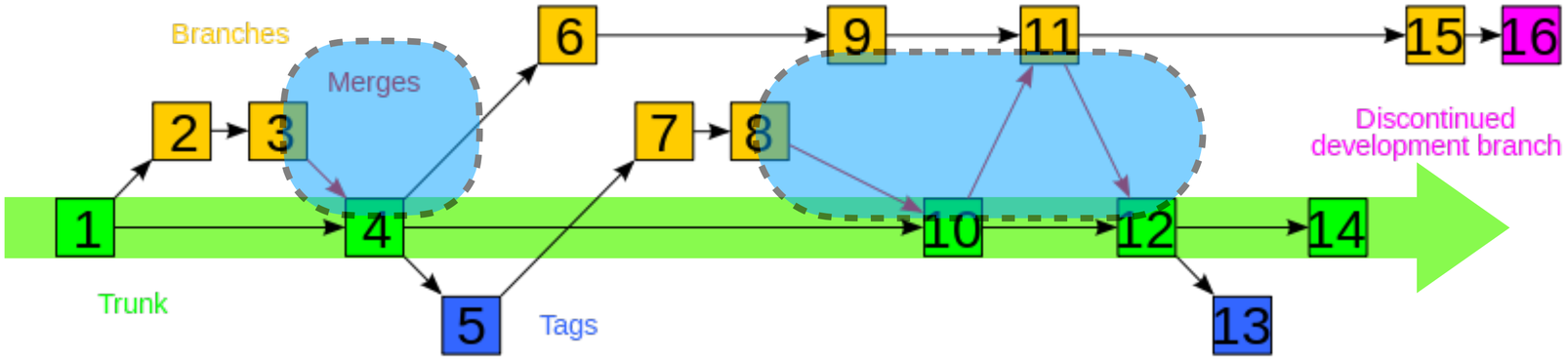

- When you check-out the project, you will get the HEAD revision
- When you invoke the command `svn update`, you are updating your local copy to the HEAD version as well.

# Subversion



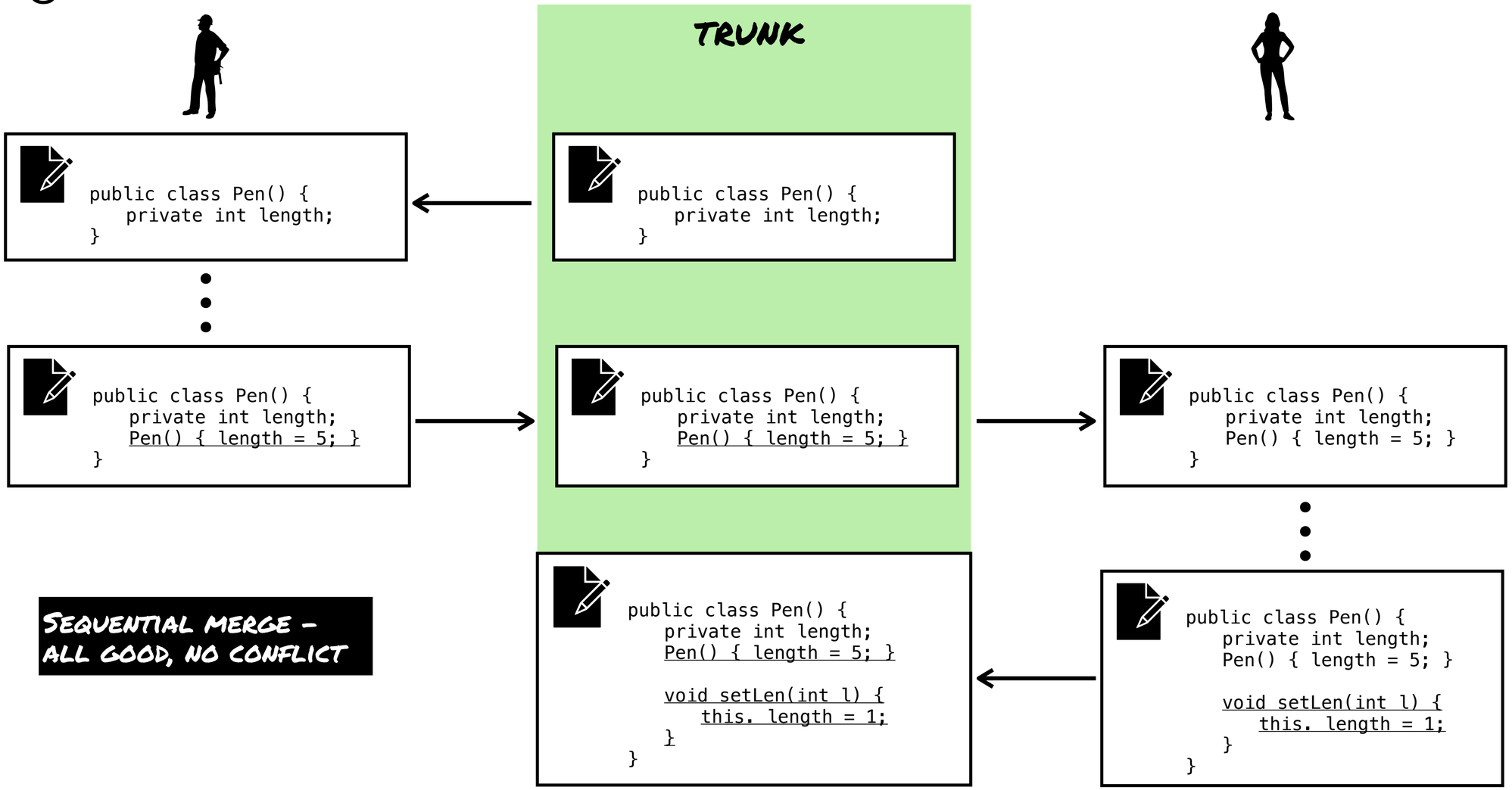

- Branches should be eventually merged back into the trunk with `svn commit`
- The trunk must build afterwards

# Subversion

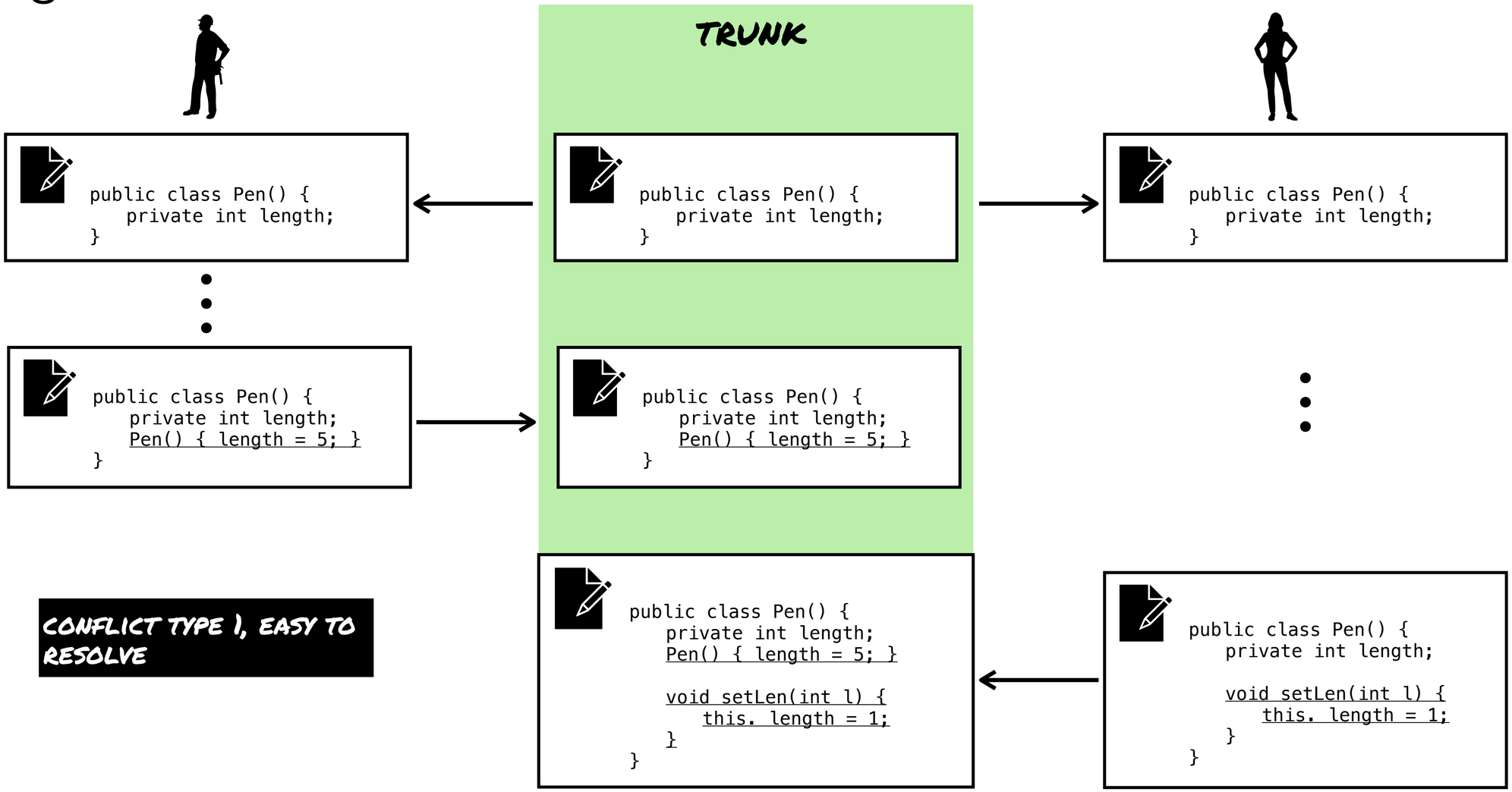



- Commit is a process of storing changes from private workplace to central server. After commit, changes are made available to all the team.
- Other developers can retrieve these changes by updating their working copy.
- **Commit is an atomic operation**. Either the whole commit succeeds or is rolled back. Users never see half finished commit.

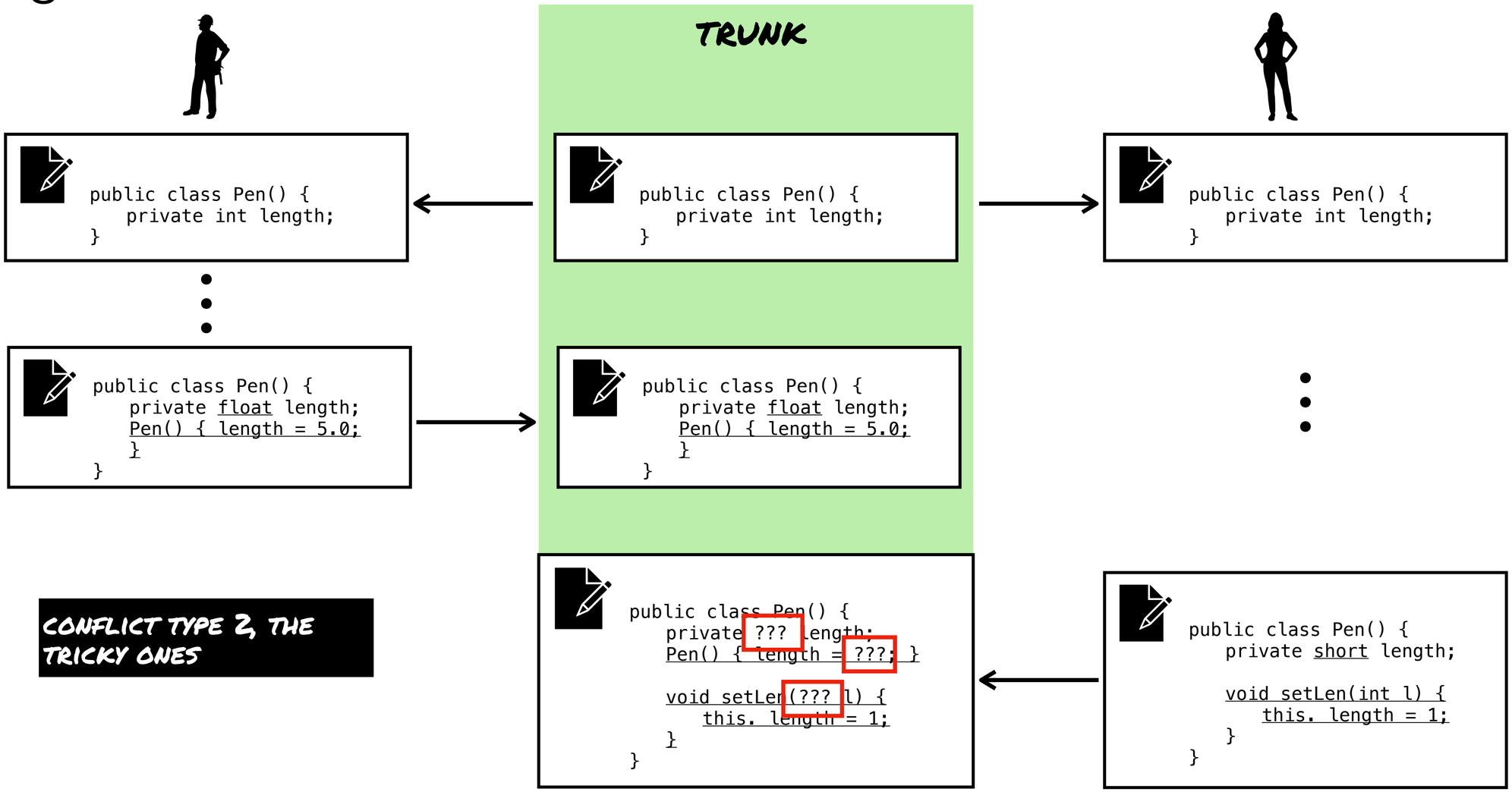
# Merge Conflicts



# Merge Conflicts



# Merge Conflicts



# Git

Git is like a Swiss Army knife for version control, empowering developers with its distributed nature and powerful branching capabilities. It's a game-changer in collaborative coding. SVN, on the other hand, offers simplicity and stability, like a reliable old friend. Choose wisely based on your project's needs and team dynamics.

**Linus Torvalds**

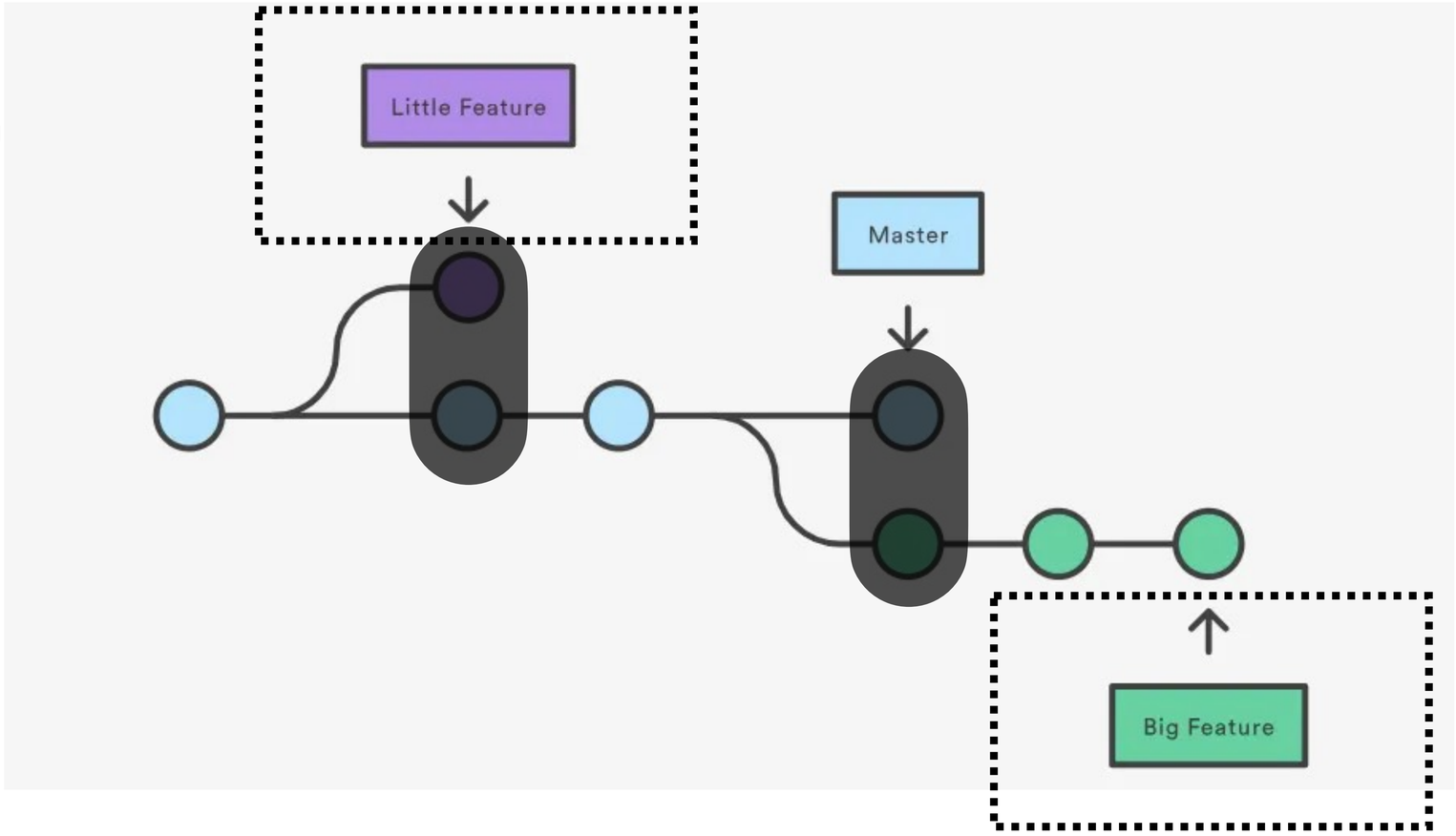

# Git


**How was it created?**

- Torvalds wasn't using any version control for Linux Kernel circa 1991 - 2002
- Changes were passed around as patches and archived files
- In 2002, they began using BitKeeper for managing the source for the Linux Kernel
- In 2005 the relationship broke down and BitKeeper revoked their license
- In no time April 2005, we were blessed with Git

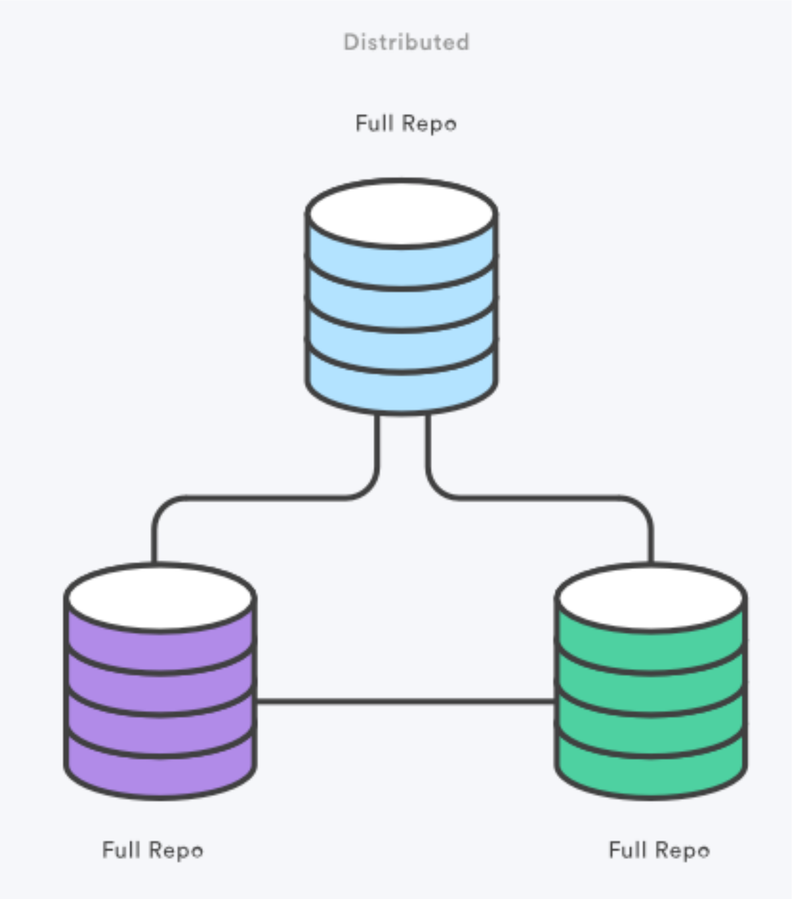
# Feature Branch Workflow

- Git encourages branching for every feature - regardless of the size, a branch can easily be created
- After successful completion of the new feature, branch is merged into trunk



# Distributed Development

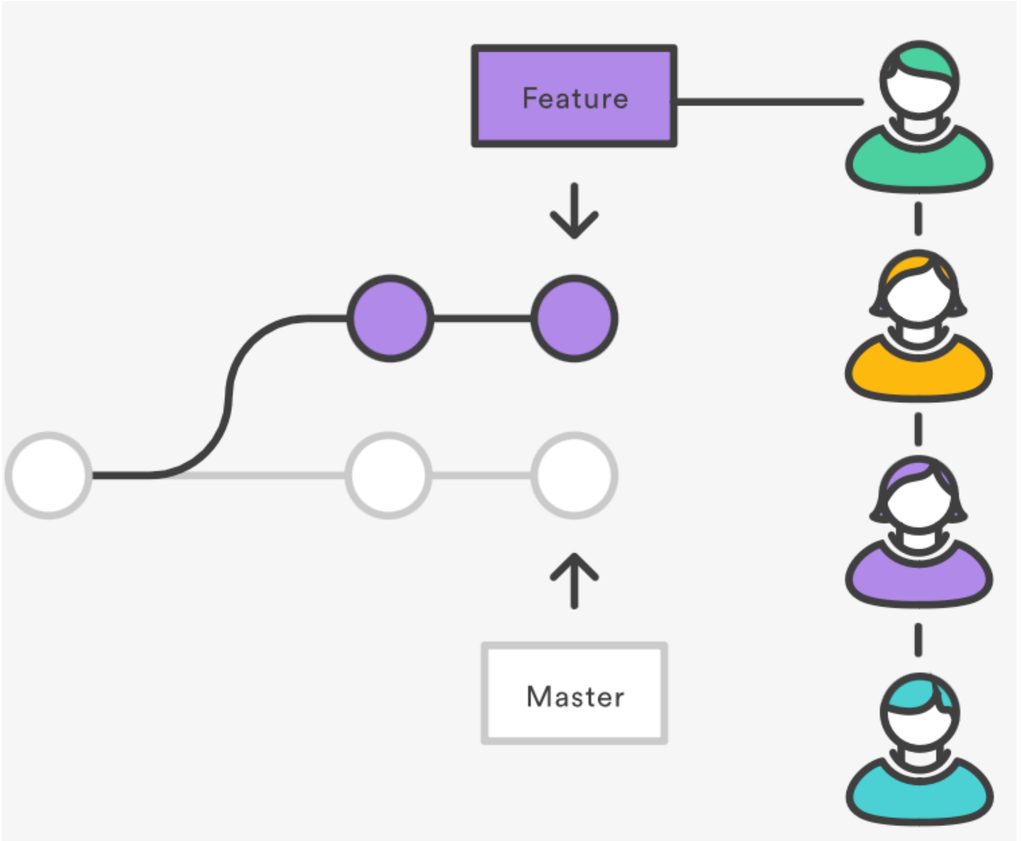
- Each developer gets their local repository
- Git is extremely fast, you don't need a network connection to commit changes (push), inspect previous version and perform diffs
- If someone breaks the production trunk / branch, you can continue working



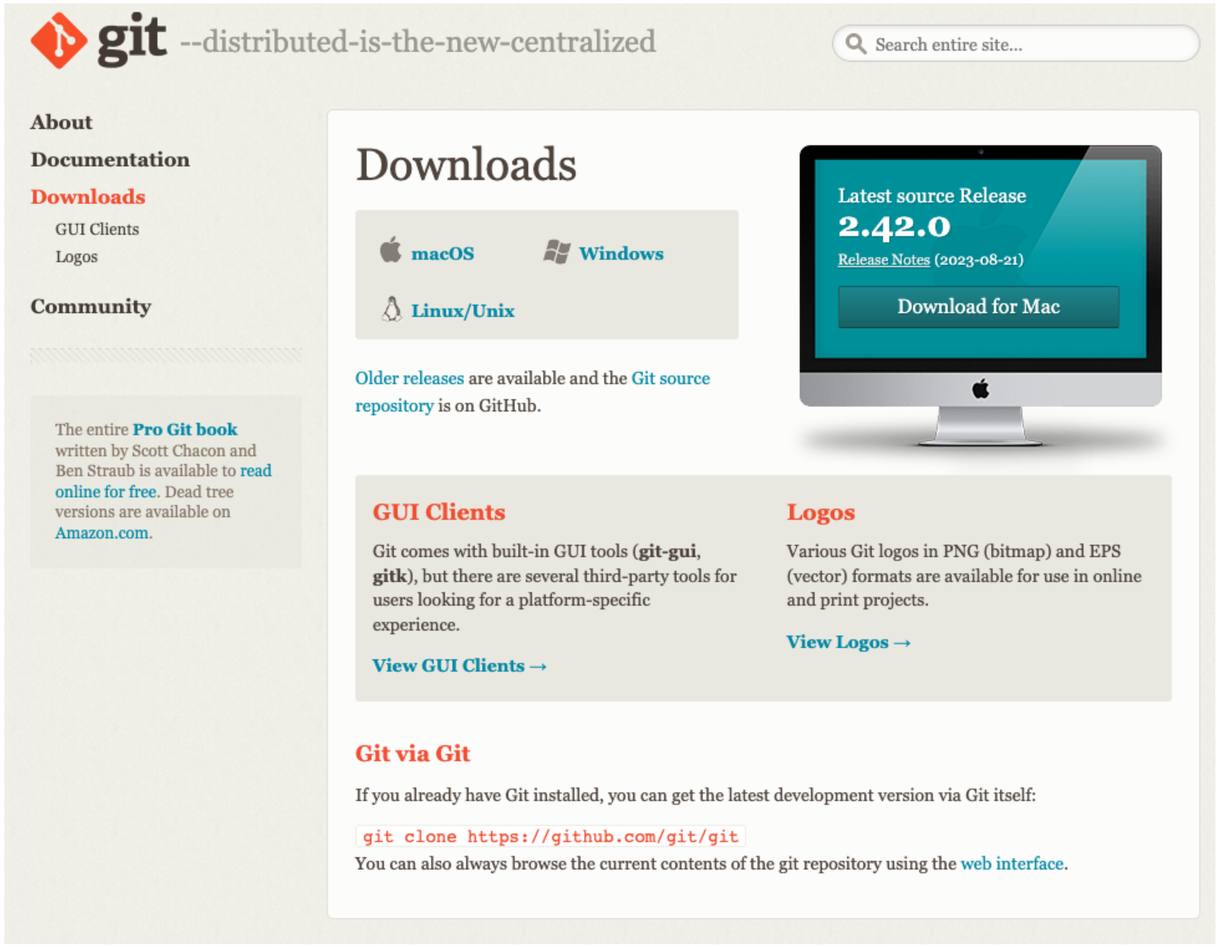
# Pull Requests

- A pull request is where you ask another developer to merge your feature into their repository

- Project leads can keep track of changes
- Project leads can merge it with their repository



# Git - Demo




1. GET GIT ON YOUR MACHINE - GO HERE:
   https://git-scm.com/downloads
   https://git-scm.com/book/en/v2/Getting-Started-Installing-Git

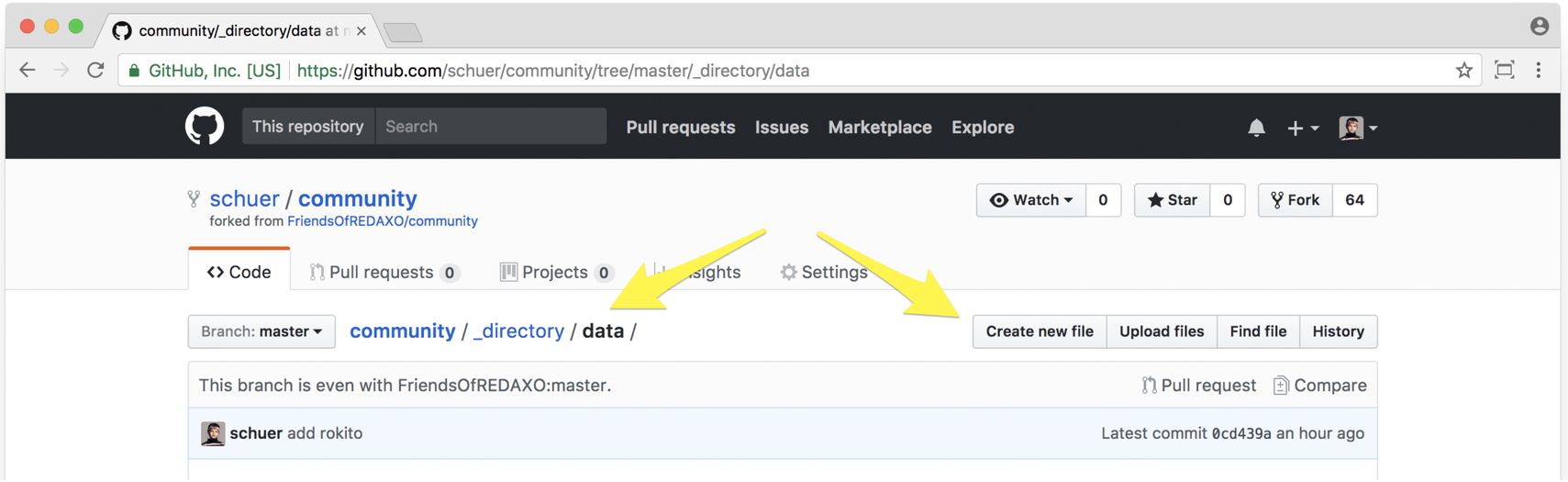
2. USE DEFAULT SETTING FOR NOW - YOU CAN RECONFIGURE LATER

3. ON YOUR TERMINAL / PROMPT, RUN
   `git help git`

4. CREATE REPOSITORY, COMMIT, AND DO SOME STUFFS

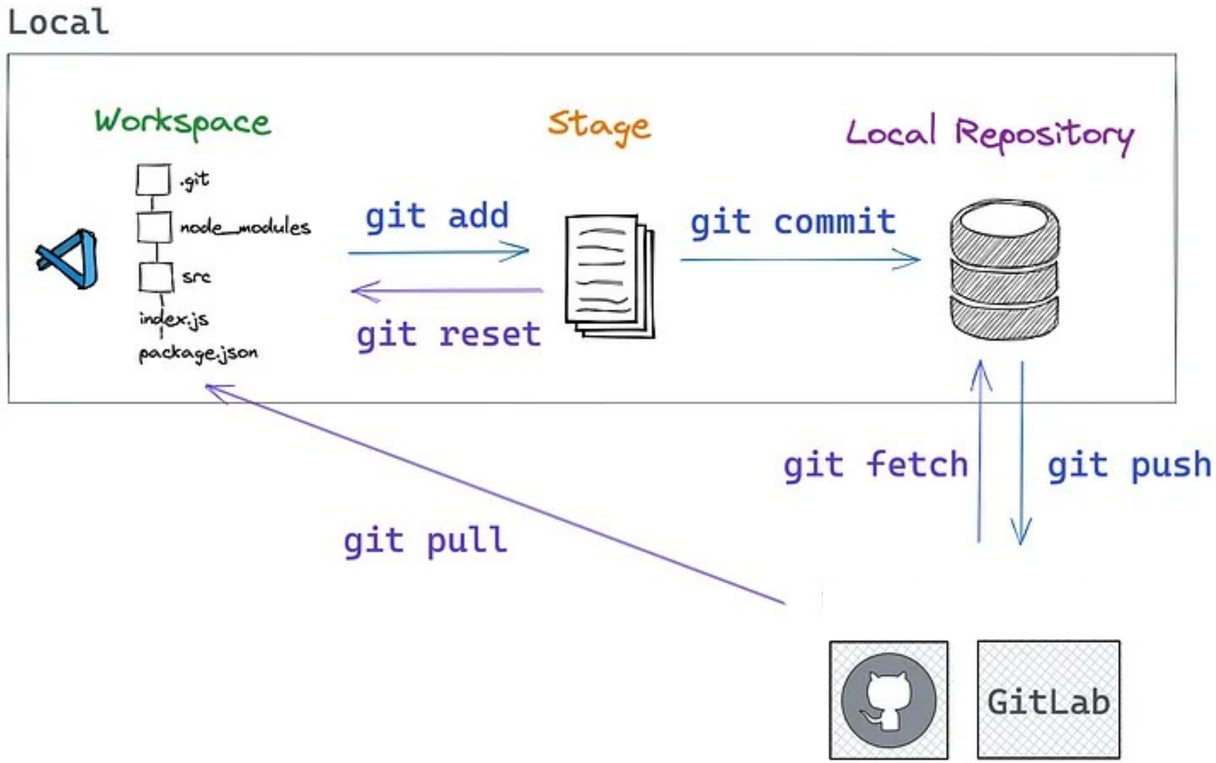
# GitHub

- GitHub is a web based hosted service for Git repositories.
- Git allows you to host remote Git repositories and has a wealth of community based service that makes it ideal for open source projects
- It is a publishing tool, a version control system and a collaboration platform



# GitHub - Commands

- `git pull`
  - pull the latest version from the repository you clones
  - synchronise will all commits in the repository
- `git fetch` and `git merge`
  - pull is actually a combination of the two, and you can run these individually
- `git push`
  - once you have committed changes locally, you must push them to a remote repository
- `git clone`
  - many project don't require you to create your own repository, instead you clone it from a remote location



# Pull requests

- If you make change to the repository, you can create a pull request
- Everyone can review the code and decide whether or not it should be included in the master branch
- It's a forum for discussing the changes.