ECE 30864
Software Engineering Tools Lab

Lecture 02

Distributed Version Control using Git

Outline

- Version Control Systems
- SVN vs. Git
- What is Git?
- Git Basics
- Git Branching
- Additional Notes

Version Control System (VCS)

The Need For Version Control

- Source code for programs may contain multiple directories with many files
  - A small program may have 1 file
  - A large program may have ***thousands*** of files

- How would you track changes between files?
  - Make backups or copies after every change?
  - Maintain a single “CHANGES” file that lists what was done?

- What would happen if you made a mistake last week and now just found out?

Subversion (SVN)

- Is a *centralized* version control system
  - Manages changes to files and directories
  - Can handle multiple users concurrently
  - Supports local or remote storage of repository data
  - However, we use GIT for this course

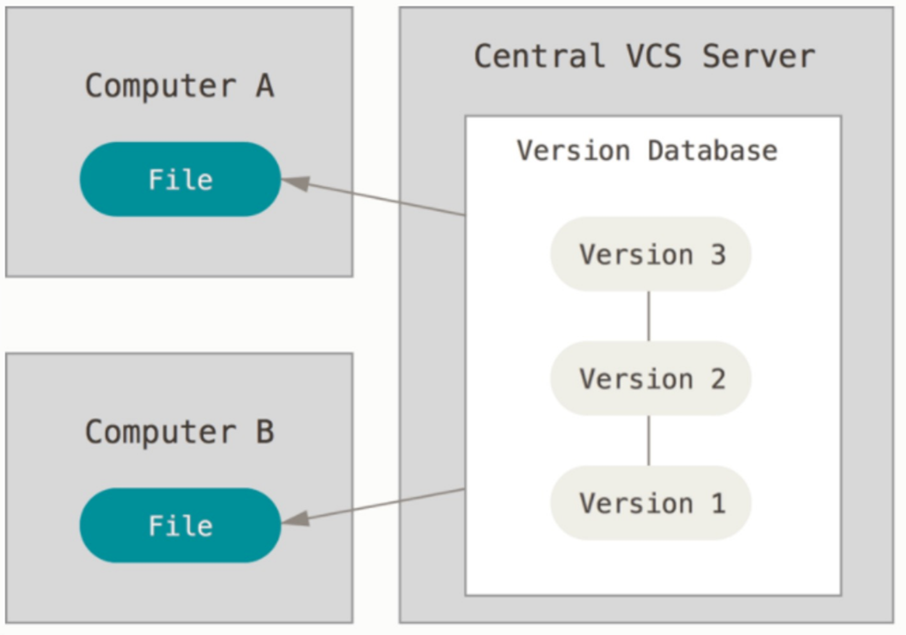
- Lab 0 covers the basics of GIT

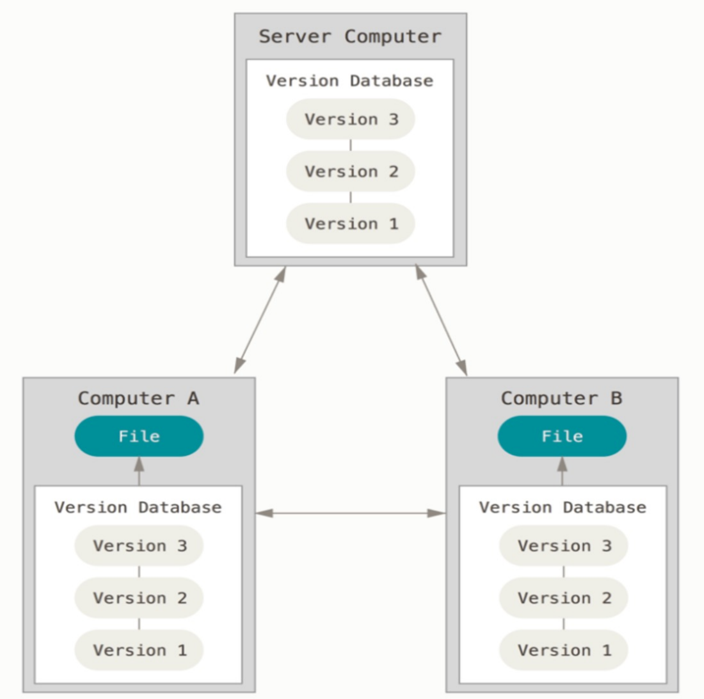
GIT

- *Distributed* Version Control System
- Created in 2005 by Linus Torvalds (creator of Linux)
- Data stored in the form a stream of snapshots
- Uses SHA-1 hash to prevent data loss in transit or file corruption.
- https://git-scm.com/

SVN vs. GIT

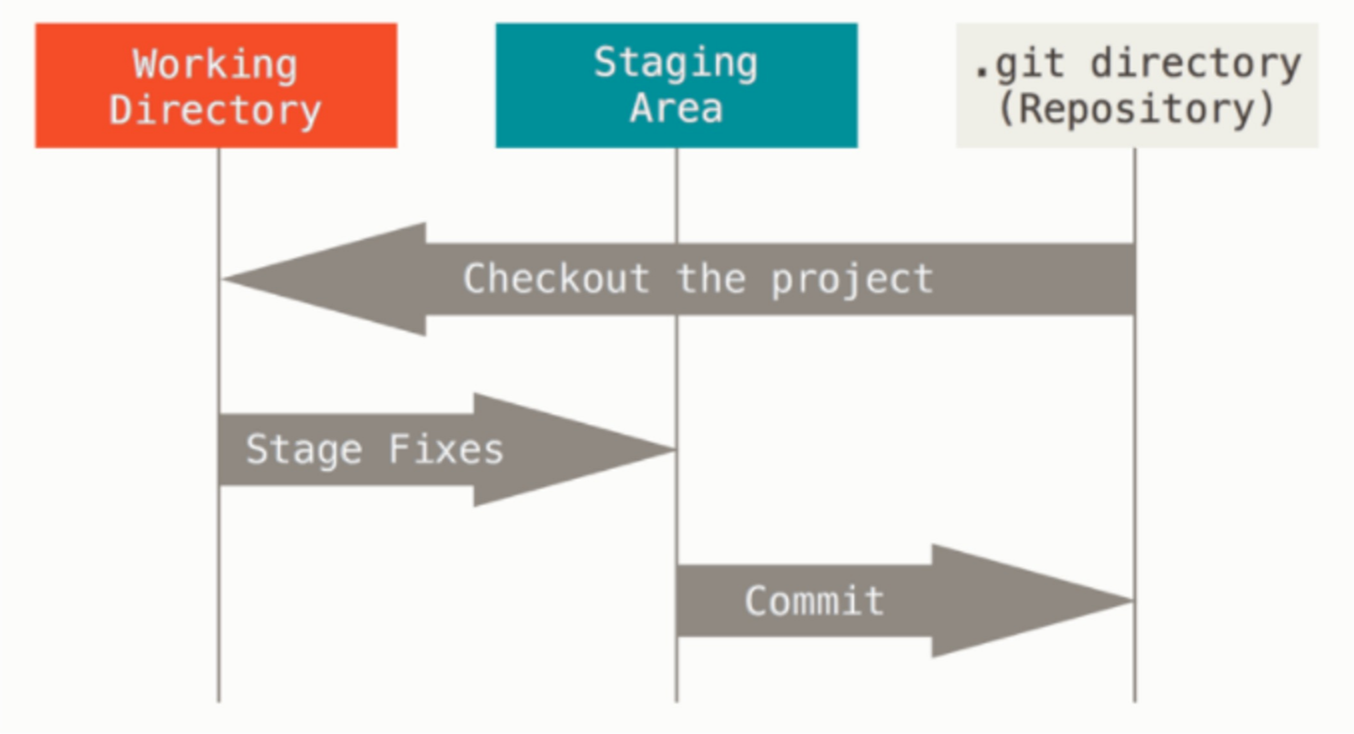
SVN	GIT
Centralized VCS	Distributed VCS
Uses deltas between versions of files to track changes	Uses a stream of snapshots of files to track changes
Every operation requires communication with central repository – network latency bound	Most operations are performed locally





GIT States

- Make changes to your files in the working directory
- Choose changes you want to stage in the staging area
- Commit the changes in the staging area to the repository



GIT Terminology

- Repository - a store that contains all distinct copies (revisions) of your work.
- Revision – a unique snapshot of the repository contents at a specific point in time
  - A revision is identified by a unique number (SHA-1 Hash)
  - A revision represents the state of all files at a point in time
- Working Copy – a local copy of what is stored in the repository.
  - Modifications to file and directories are made to a working copy
  - Includes a ".git" folder containing a snapshot of all repository info
- Staging Area – Holds pending changes before committing

GIT Terminology (2)

- Cloning - the act of creating a new local repository
  - Downloads whatever is stored in the repository including version history
  - You can clone a specific revision, not just most recent

- Committing - the act of uploading changes from the staging area to the (local) repository
  - Creates a new revision in the repository

- Pulling - the act of downloading changes from the remote repository to local repository
  - Your local repository could be several revisions old
  - Synchronizes the local repository to the remote repository

GIT Basic Commands

- `git init`
  - Creates a new local repository
- `git clone username@host:/path/to/repository`
  - Clones a remote repository
- `git add <filename>`
  - Adds the file to the staging area
  - Using * instead of the filename adds all the files in the directory to the staging area
  - Tip: the option ‘-u’ only adds already tracked files (does not add new files)

GIT Basic Commands (2)

- `git commit –m “Commit message”`
  - Commits changes to head of local repository
  - Note: This command does not modify the remote repository
  - Empties the staging area

- `git push`
  - Sends the changes to your remote repository

- `git status`
  - List the files you have changed and those you still need to add or commit

- `git log`
  - Shows the version history of the repository with all commits

GIT Basic Commands (3)

- `git pull`
  - Fetch and merge changes on the remote server into your local repo.

- `git branch`
  - Lists all branches in your repo and indicates the current branch

- `git checkout <branchname>`
  - Switches to the branch mentioned in the command

- `git checkout –b <branchname>`
  - Creates a new branch and switches to it

GIT Branching

- Branching has several advantages:
  - Enables parallel feature development
  - Allows multiple developers to work on the same codebase
  - Helps maintain a cleaner production version of code.

- **However, branching has its own complications**

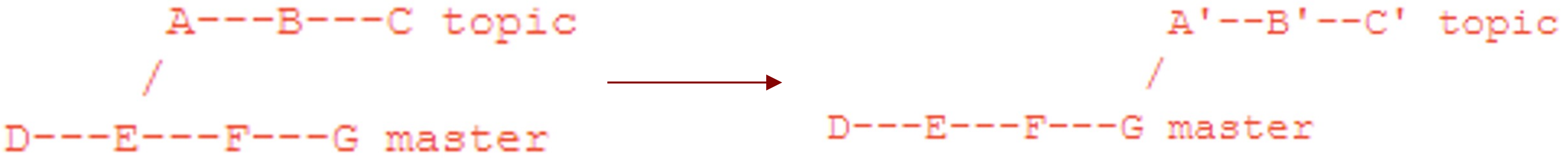
GIT Rebase

- Imagine the following scenario
  - Master branch has the following commits – master_commit1, master_commit2, master_commit3
  - You fork a development branch called ‘dev’ from master
  - You then add a few commits – dev_commit1, dev_commit2 in your branch ‘dev’
  - In the meantime, there occurs a fourth commit on master called master_commit4
  - You need the changes from master_commit4 on your ‘dev’ branch to continue development

- How would you resolve this?

GIT Rebase (2)

- Git Rebase – Reapplies commits on top of another base
- In our scenario,
  - `git rebase master dev`
  - This would rebase the development branch ‘dev’ with the latest master branch commits.



```
      A---B---C topic                        A'--B'--C' topic
     /                      ──────>         /
D---E---F---G master          D---E---F---G master
```

GIT Merge

- Merges changes from one branch to another
- `git merge development`
  - This command merges changes from development branch onto the master branch
  - Useful Flag:
    - `--squash`
    - Squashes all the commits in development branch into one commit when merging with the master branch

Additional Notes

- gitignore file is used to specify untracked files that Git should ignore while committing code.

- git stash is a useful command to stash your local uncommitted changes while pulling changes from your remote repo
  - Stashed changes can then be reapplied with pop/apply

- Git internals are all stored under the .git folder (hidden folder) in the root of your repository.