CS 35L- Software Construction Laboratory

Winter 19

TA: Guangyu Zhou

Change Management

Week 10

Announcement

- 2019-03-15 is the due date for:
- 9. Change management *(No late submissions allowed for this assignment)*
- 10. Research and development in computing *(No late submissions allowed for this assignment.)*
- Final Report:
    - One team submit one and include the UID and names in your report.
    - Around 1000 words
    - No strict format

Software development process

- Involves making a lot of changes to code
    - Add new features
    - Bugs fixed
    - Performance Improvement
- Many people working on the same project
- Many versions of software released
    - Ubuntu 14, 16 etc.
    - Need to fix bugs in version 14 even most users shipped to 16

Shortcomings of diff and patch

- Conflict: Two people may edit the same file on the same date
  - 2 patches need to be sent and merged
- Changes to one file might affect other files (.h and .c)
  - Need to make sure those versions are stored together as a group

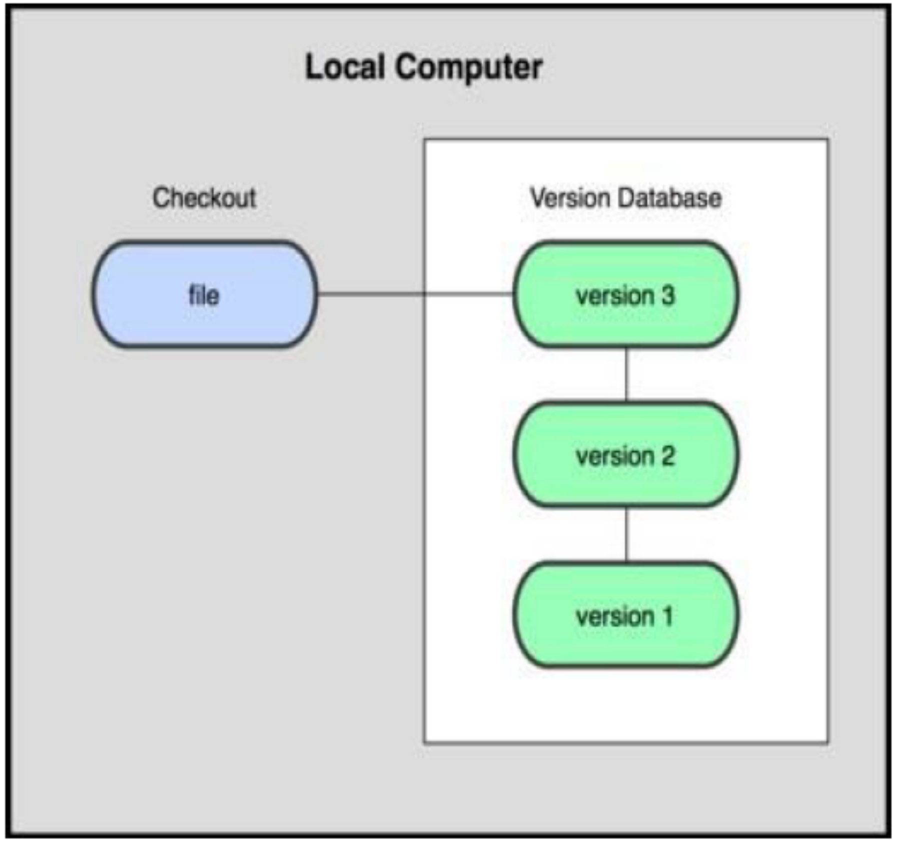
Source/Version Control

- Track changes to code and other files related to software
  - What new files are added?
  - What changes made to files?
  - Which version had what changes?
  - Which user made the changes?
  - Revert to previous version
- Track entire history of software
- Source control software
  - Git, Subversion (SVN), CVS, and others

Architectures of Source/Version Control

- Local Source Control System
- Centralized Source Control System
- Distributed Source Control System

Local Source Control System



- Organize different versions as folders on the local machine
- No server involved
- Other users copy with disk/network

Image Source: git-scm.com

Centralized Source Control System

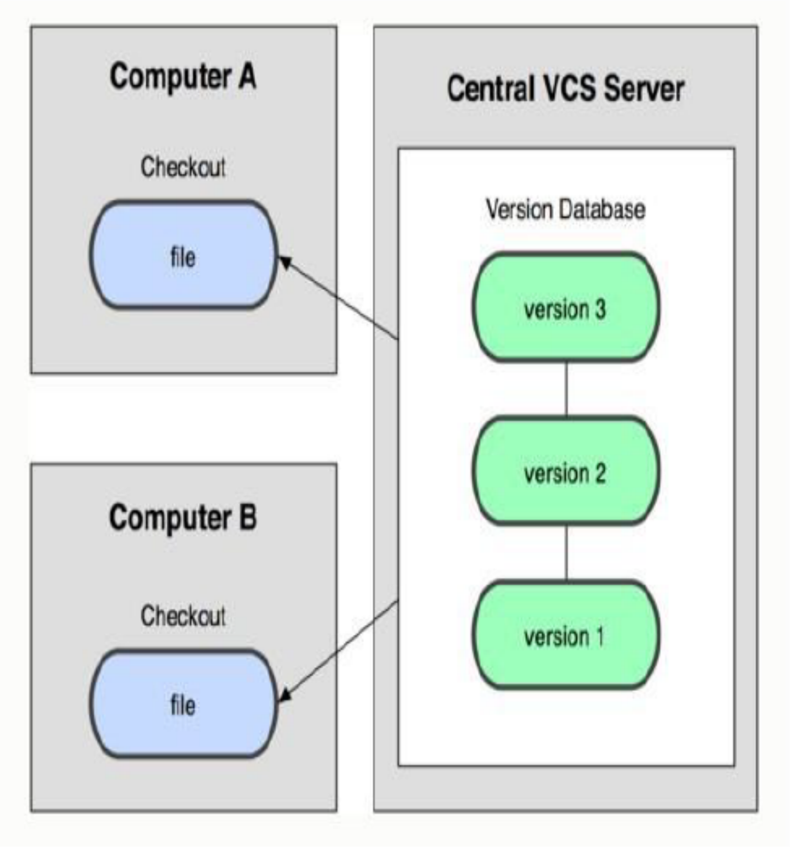


Image Source: git-scm.com

- Version history sits on a central server
- Users will get a working copy of the files
- Changes have to be committed to the server
- All users can get the changes

Distributed Source Control System

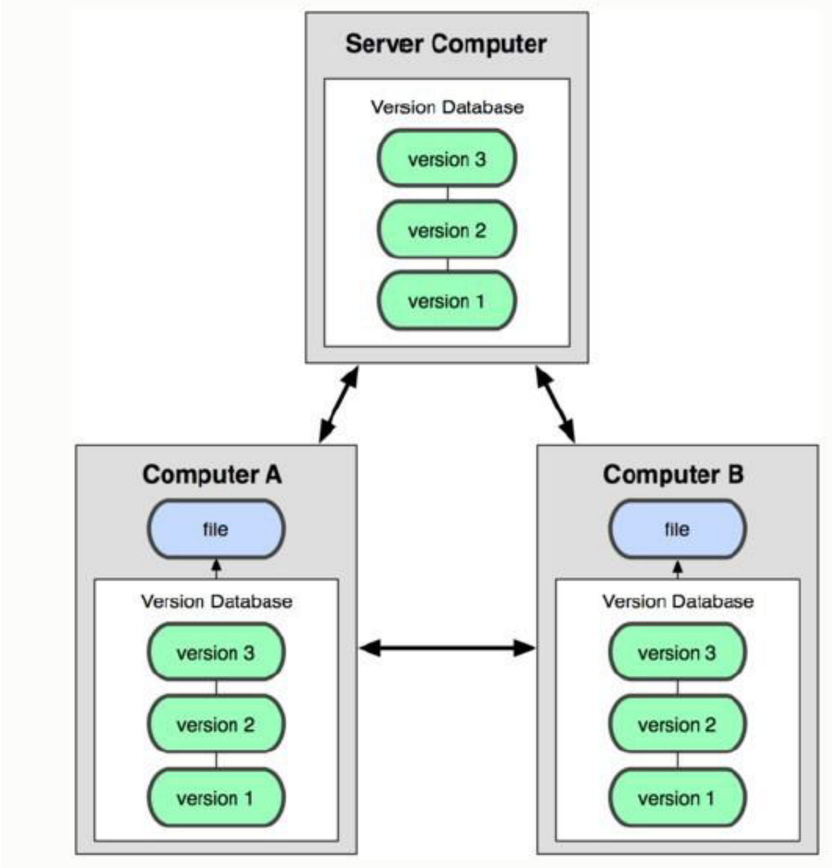


Image Source: git-scm.com

- Version history is replicated on every user’s machine
- Users have version control all the time
- Changes can be communicated between users
- Git is distributed

Terms used

- Repository
    - Files and folders related to the software code
    - Full history of the software
- Working copy
    - Copy of software’s files in the repository
- Check-out
    - To create a working copy of the repository
- Check-in/Commit
    - Write the changes made in the working copy to the repository
    - Commits are recorded by the SCS

Centralized: pros and cons

- Pros
  - Everyone can see changes at any time
  - Simple to design

- Cons
  - Single point of failure (no backup!)

“The full project history is only stored in one central place.”

Distributed: pros and cons

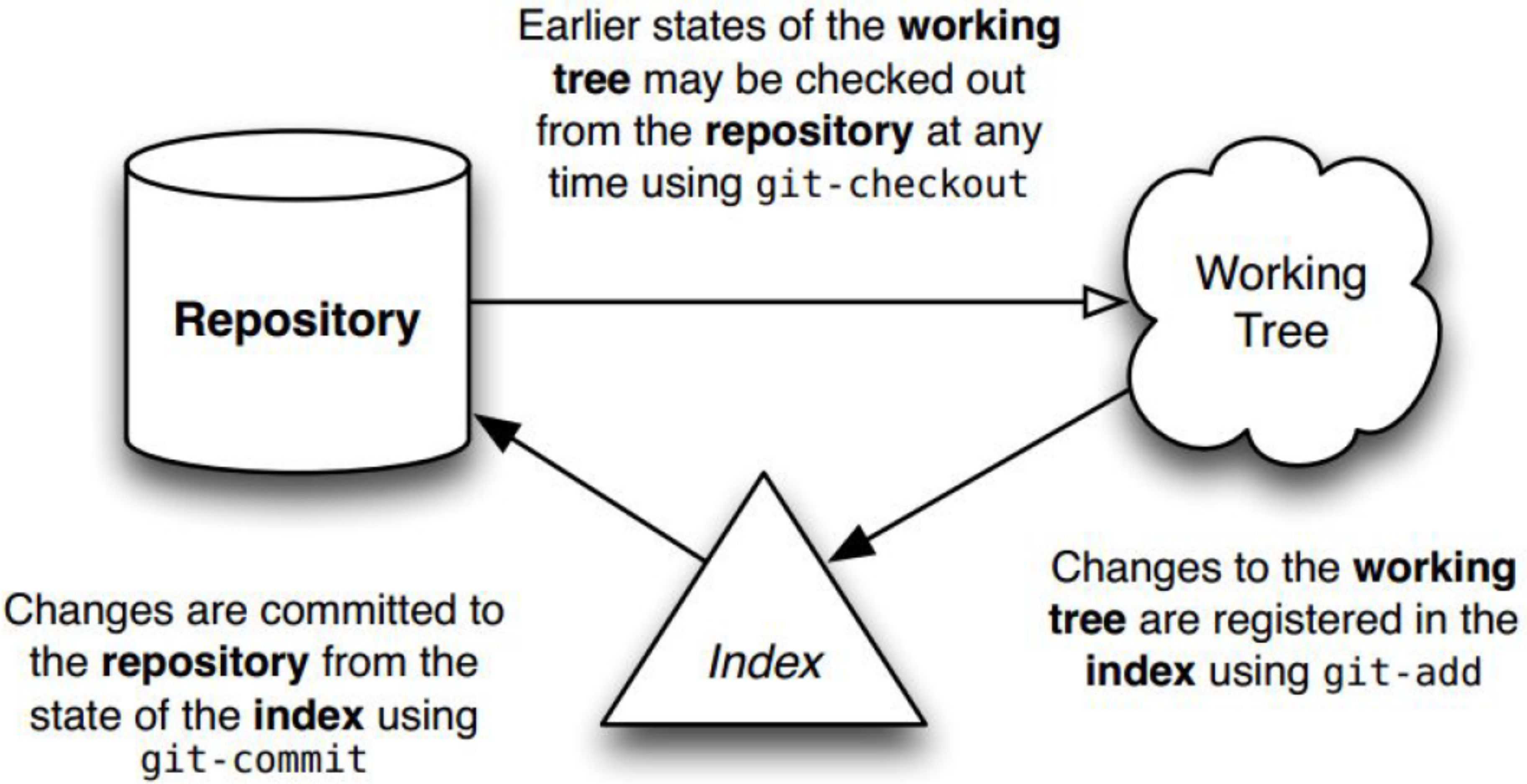
- Pros
  - Commit changes/revert to an old version while offline
  - Commands run extremely fast because tool accesses the hard drive and not a remote server
  - Share changes with a few people before showing changes to everyone
- Cons
  - long time to download
  - Heavy space overhead to store all versions of code

Centralized vs. Distributed VCS

- Single central copy of the project history on a server
- Changes are uploaded to the server
- Other programmers can get changes from the server
- Example: SVN, CVS

- Each developer gets the full history of a project on their own hard drive
- Developers can communicate changes between each other without going through a central server
- Example: **Git**, Mercurial, Bazaar, Bitkeeper

Git Workflow

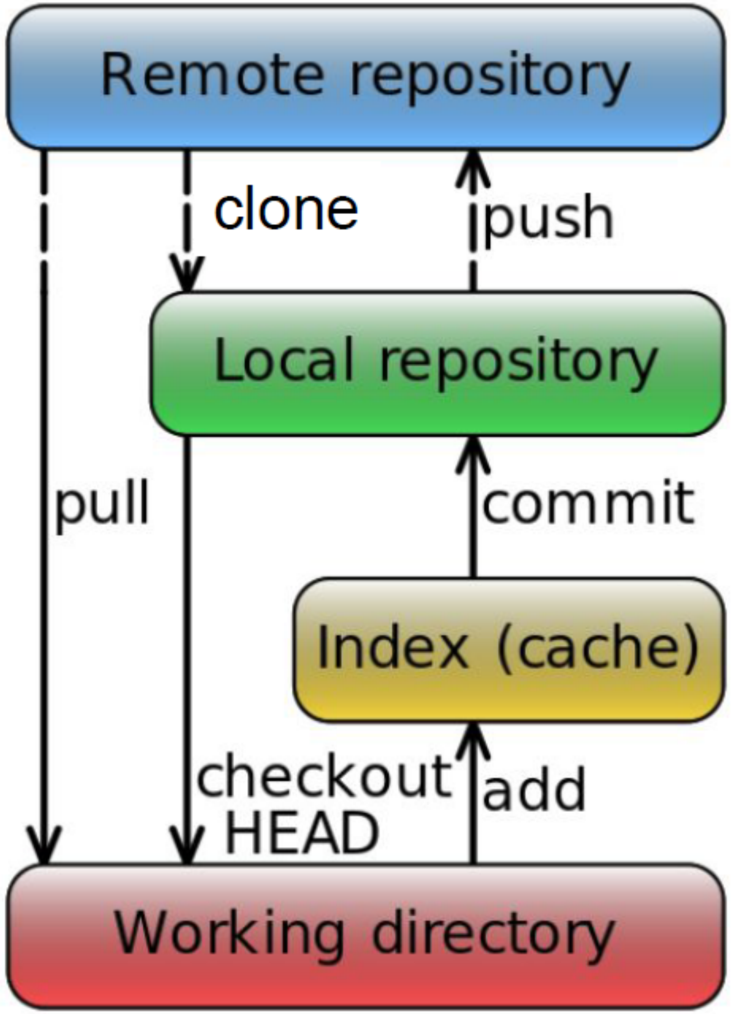


Earlier states of the **working tree** may be checked out from the **repository** at any time using `git-checkout`

Changes are committed to the **repository** from the state of the **index** using `git-commit`

Changes to the **working tree** are registered in the **index** using `git-add`

Overview of git



Git commands (1)

- Repository creation
  - **git init** (start a new repository)
  - **git clone** (create a copy of an existing repository)
- Branching
  - **git checkout <tag/commit> -b <new_branch_name>** (create a new branch)
- Commit
  - **git add** (stage modified files)
  - **git commit** (check-in changes to the repository)

Git commands (2)

- Getting info
  - **git status** (shows modified files, new files, etc)
  - **git diff** (compares working copy with staged files)
  - **git log** (shows history of commits)
  - **git show** (show a certain object in the repository)
- Getting help
  - **git help**

Git Repository Objects

- Objects used by Git to implement source control
  - **Blobs** (binary large object).
    - When we git add a file such as example_file.txt, git creates a *blob* object containing the contents of the file. Blobs are therefore the git object type for storing files.
  - **Trees** (Groups blobs/trees together)
    - The tree object contains one line per file or subdirectory, with each line giving file permissions, *object type*, object hash and filename.
  - **Commit**
    - Refers to a particular "git commit"
    - Contains all information about the commit
  - **Tags** (A named commit object for convenience)
- Objects uniquely identified with **hashes**
- **More information: https://matthew-brett.github.io/curious-git/git_object_types.html**

Git States

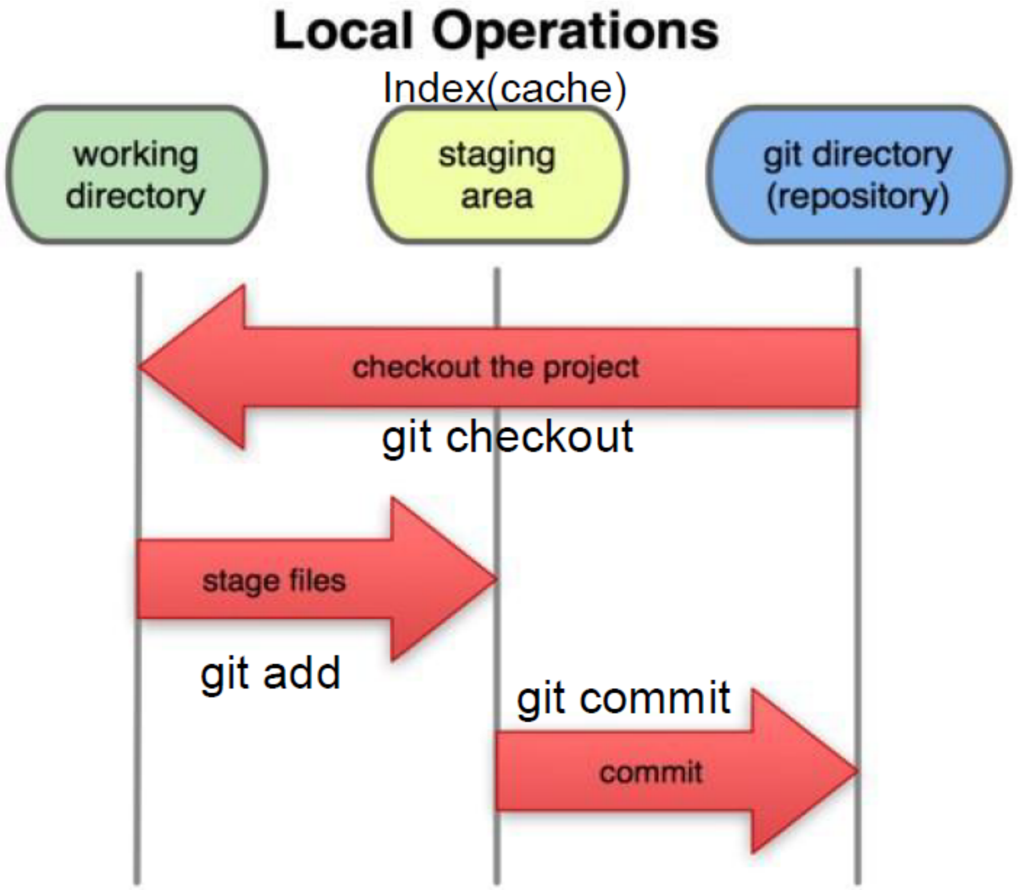


Image Source: git-scm.com

First Git Repository

- $ *mkdir gitroot*
- $ *cd gitroot*
- $ *git init*
  - creates an empty git repo (.git directory)
- $ *echo “Hello World” > hello.txt*
- $ *git add*
  - Add content to the index
  - Run prior to a commit
- $ *git commit –m ‘Check in number one’*

Working with Git

- *$ echo “I love Git” >> hello.txt*
- *$ git status*
  - Show list of modified files (hello.txt here)
- *$ git diff*
  - Show changes made compared to index
- *$ git add hello.txt*
- *$ git diff*
- *$ git diff HEAD*
  - Show changes in the working version
- *$ git commit –m ‘Second commit’*

Lab 9

- GNU Diffutils uses " ` " in diagnostics
  - Example: diff . –
  - Output: diff: cannot compare - to a directory
  - Want to use apostrophes only (instead of backtick)
- Diffutils maintainers have a patch for this problem:

  maint: quote 'like this' or "like this", not `like this'
- Problem: You are using Diffutils version 3.0, and the patch is for a newer version

Lab 9

- Task: Fix an issue with the diff diagnostic
- Crucial Steps: first create a new work directory
  - (4) Generate a patch
    - First get the hash code in the log file
    - Then use git format-patch -1 [hash code] --stdout > [the patch file]
  - (9) learn the detailed usage of vc-diff and vc-revert
  - (10) consider changing ` to ‘