CS 35L

Week 4

TA: Tomer Weiss
Jan-26-2016

goo.gl/EGdzrl

Slides

Announcements

- Assignment 10 reminder
  - 5 minute presentation with slides
    - Please sign-up here
    - **Include topic**
  - Brief Research report

- Signups will be **first come, first serve** basis.
- Same for presentation topics
- For reference on presentation, grading, please refer to this rubric.

Assignment 10 guidelines

- Presentation and research report:
  - Discuss the topic you are presenting
    - Assessment of the article you are presenting as opposed to just overview
      - What are the limitations?
      - Any critique?
      - Other approaches
  - Topic must be computer science related
    - 3 recent issues of ACM technews, or consult TA for different topic
  - Link to assignment 10

Change Management

Week 4

What Changes Are We Managing?

Software

- Planned software development
  - team members constantly add new code
- (Un)expected problems
  - bug fixes
- Enhancements
  - Make code more efficient (memory, execution time)

“The only constant in software development is change”

Software development process

- Involves making a lot of changes to code
  - New features added
  - Bugs fixed
  - Performance enhancements
- Software team has many people working on same/different parts of code
- Many versions of software released
  - Ubuntu 10, Ubuntu 12, etc
  - Need to be able to fix bugs for Ubuntu 10 even (some customers still use it) though you shipped Ubuntu 12

Features Required to Manage Change

- Backups
- Timestamps
- Who made the change?
- Where was the change made?
- A way to communicate changes with team

How to achieve that

- Big project with multiple files
  - Bug fix required changing multiple files
  - Bug fix didn’t work
  - How to find the problem
- Figure out which parts changed (diff?)
- Communicate changes with team (patch?)
- But diff and patch are not that good

Disadvantages of diff & patch

- Diff requires keeping a copy of old file before changes
- Work with only 2 versions of a file (old & new)
  - Projects will likely be updated more than once
  - store versions of the file to see how it evolved over time

index.html
  index-2009-04-08.html
  index-2009-06-06.html
  index-2009-08-10.html
  index-2009-11-04.html
  index-2010-01-23.html
  index-2010-09-21.html

- Numbering scheme becomes more complicated if we need to store two versions for the same date

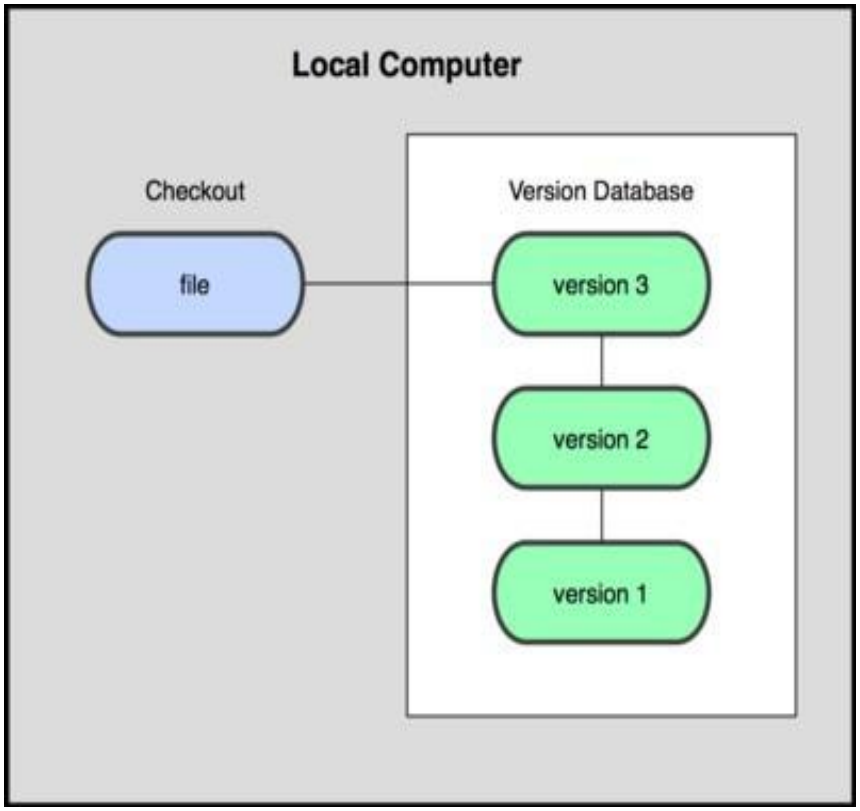
Disadvantages of diff & patch

- 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 & .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 were 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

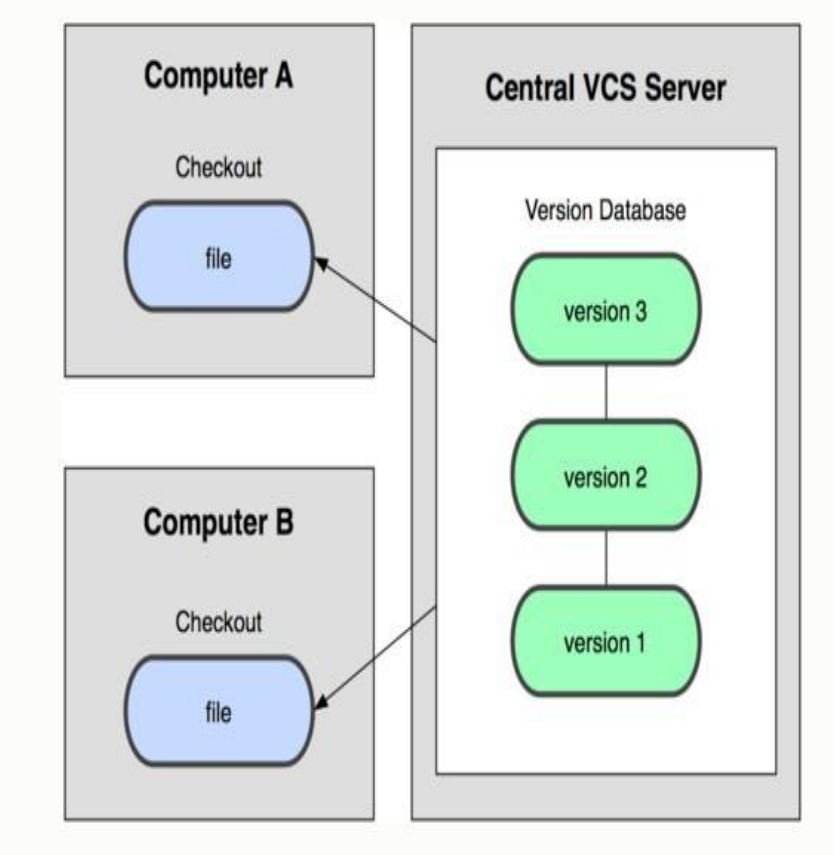
Local SCS



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

Image Source: git-scm.com

Centralized SCS



- 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

Image Source: git-scm.com

Distributed CCS

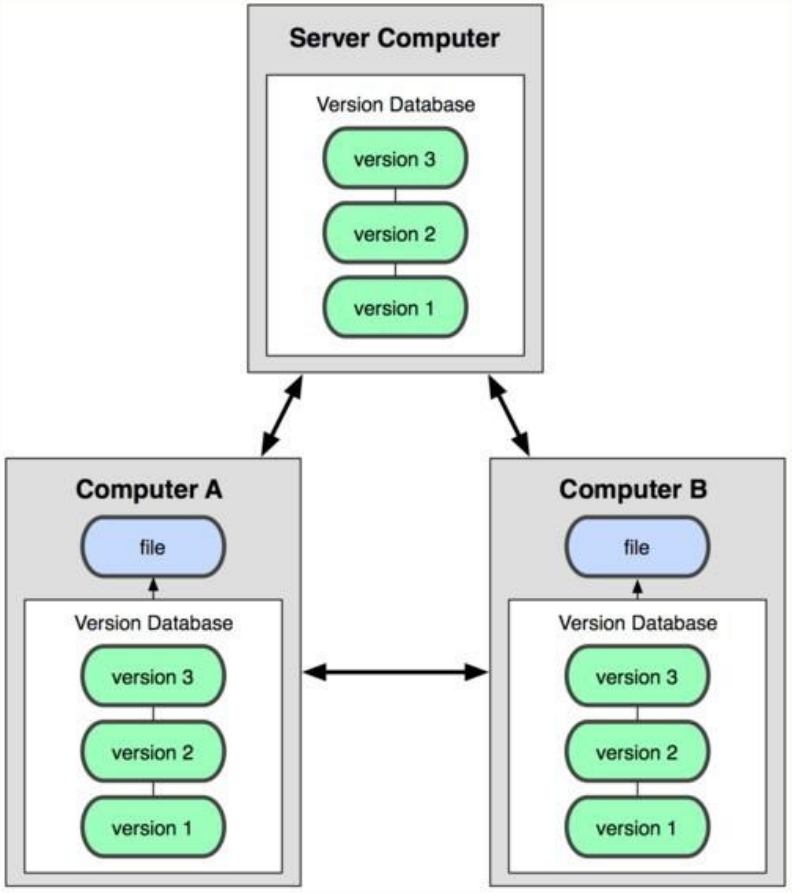


Image Source: git-scm.com

- 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 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
- Examples: 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
- Examples: **Git**, Mercurial, Bazaar, Bitkeeper

Centralized: Pros and Cons

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

Pros

- Everyone can see changes at the same time
- Simple to design

Cons

- Single point of failure (no backups!)

Distributed: Pros and Cons

"The entire project history is downloaded to the hard drive"

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

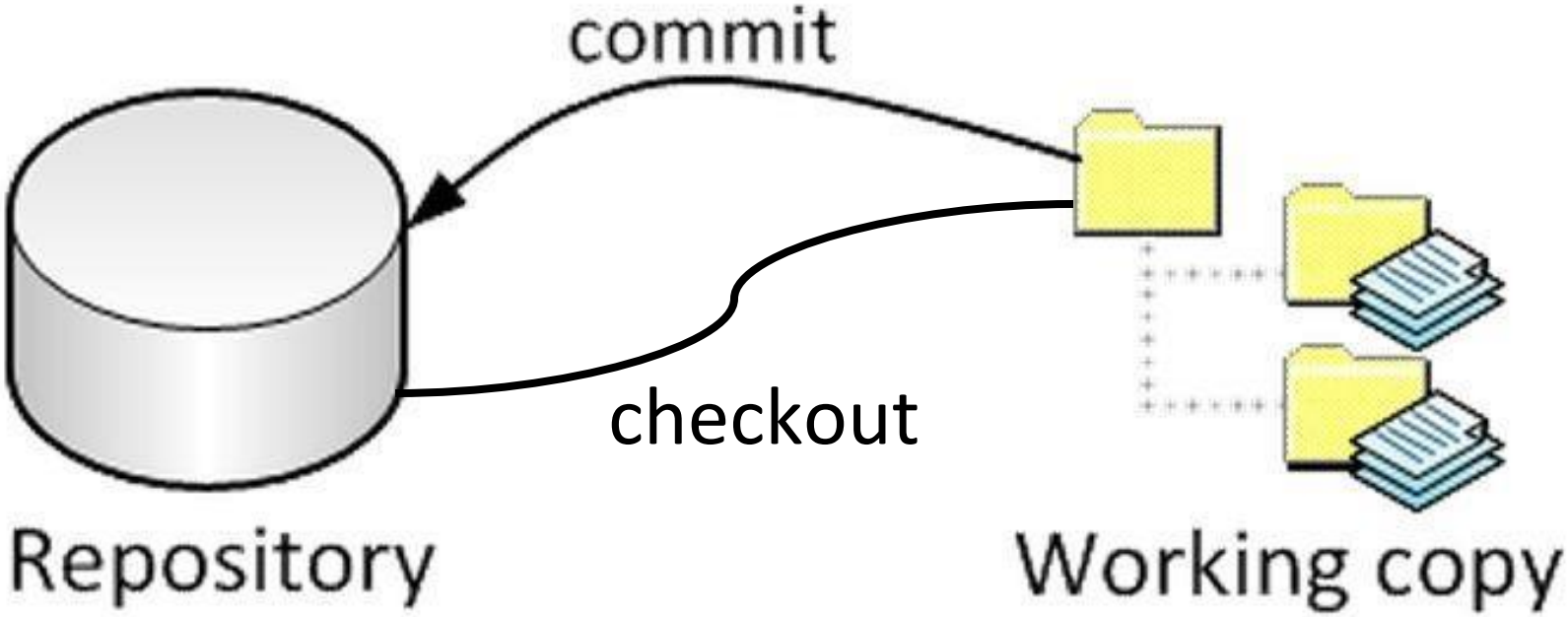
Cons

- long time to download
- A lot of disk space to store all versions

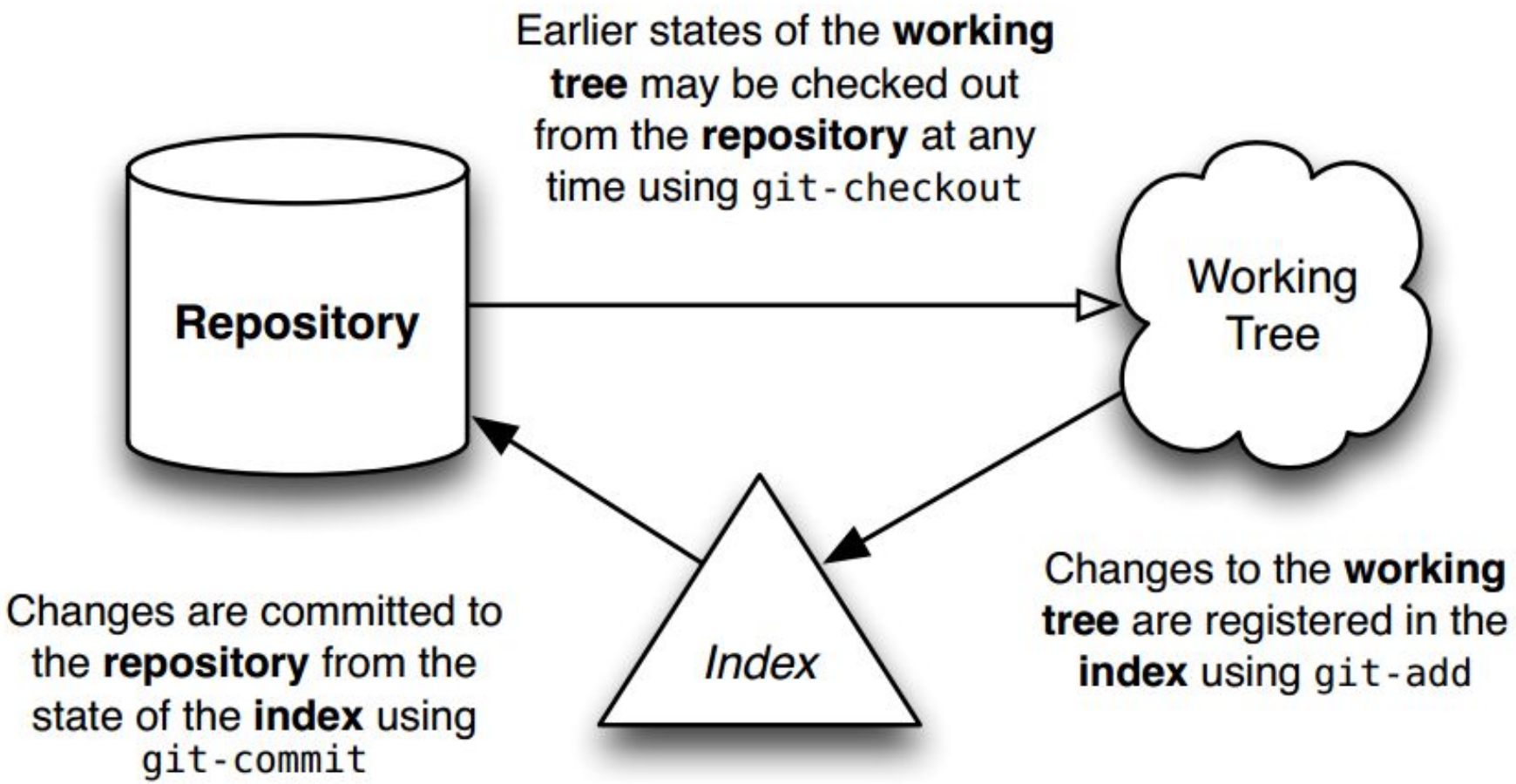
Git source control

Week 4

Big Picture



Git Workflow



Git commands

- 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>`** (creates a new branch)
- Commits
  - **`git add`** (stage modified files)
  - **`git commit`** (check-in changes to the repository)
- 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**
    - Sequence of bytes
  - **Trees**
    - Groups blobs/trees together
  - **Commit**
    - Refers to a particular “git commit”
    - Contains all information about the commit
  - **Tags**
    - A named commit object for convenience (e.g. versions of software)
- Objects uniquely identified with **hashes**

Terms used

- Head
  - Refers to a commit object
  - There can be many heads in a repository
- HEAD
  - Refers to the currently active head
- Detached HEAD
  - If a commit is not pointed to by a branch
  - This is okay if you want to just take a look at the code and if you don't commit any new changes
  - If the new commits have to be preserved then a new branch has to be created
    - `git checkout v3.0 -b BranchVersion3.1`
- Branch
  - Refers to a head and its entire set of ancestor commits
- Master
  - Default branch

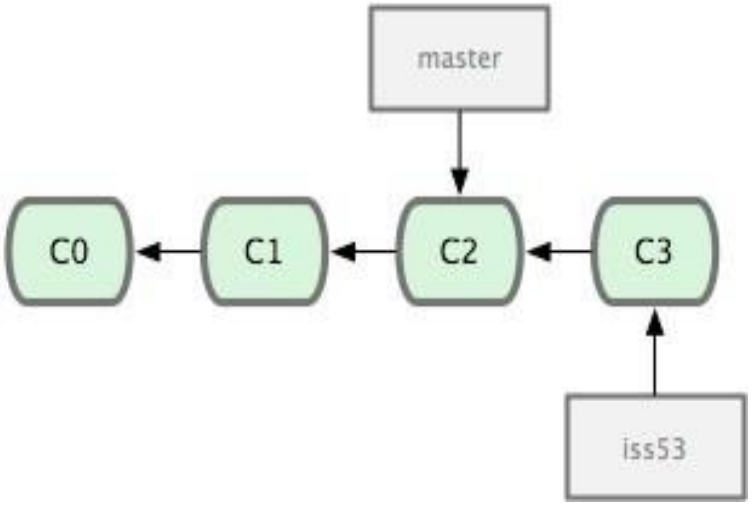


Image Source: git-scm.com

Git States

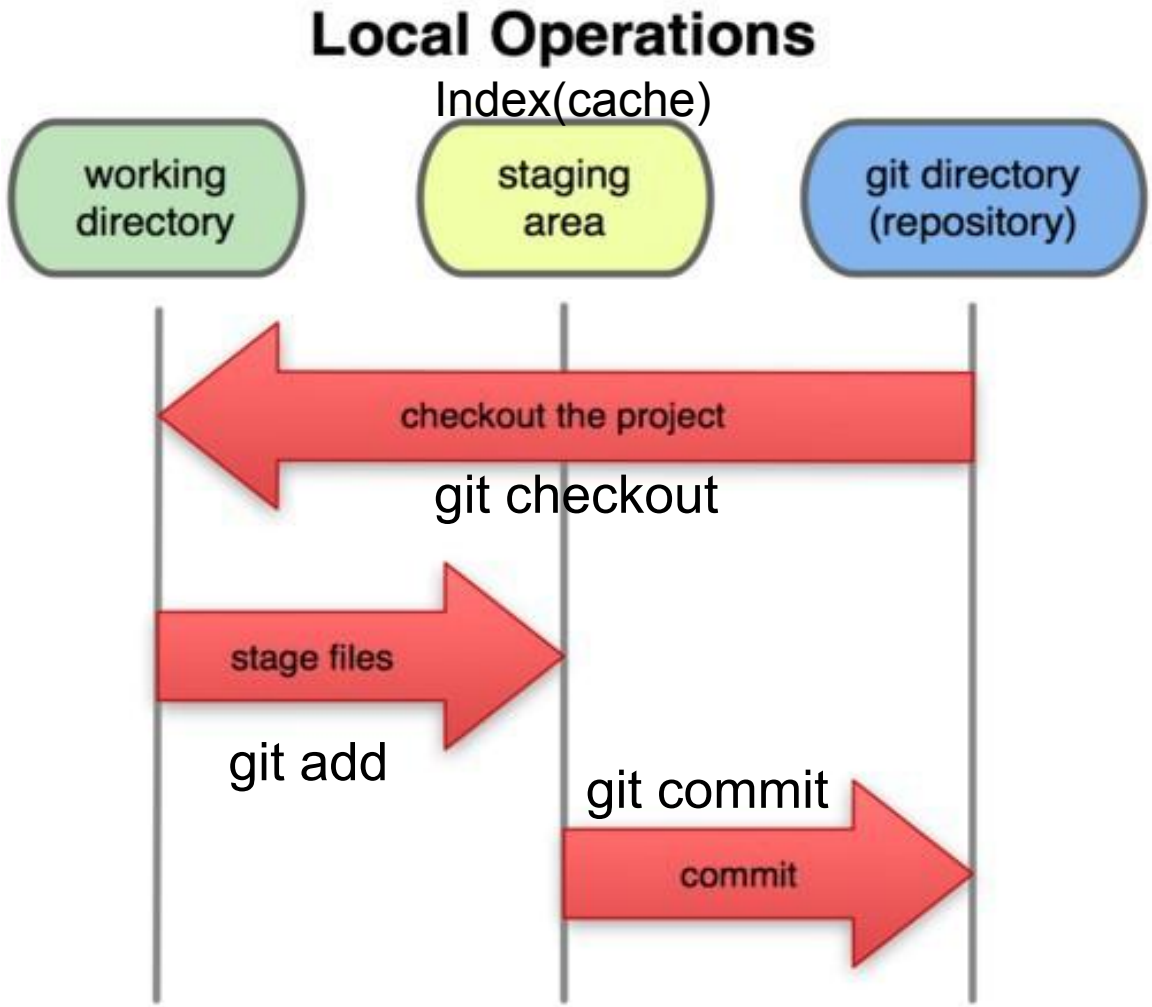


Image Source: git-scm.com

First Git Repository

- `$ mkdir gitroot`
- `$ cd gitroot`
- `$ git init`
  - creates an empty git repo (.git directory with all necessary subdirectories)
- `$ echo "Hello World" > hello.txt`
- `$ git add .`
  - Adds content to the index
  - Must be run prior to a commit
- `$ git commit –m 'Check in number one'`

Working With Git

- `$ echo "I love Git" >> hello.txt`
- `$ git status`
  - Shows list of modified files
  - hello.txt
- `$ git diff`
  - Shows changes we made compared to index
- `$ git add hello.txt`
- `$ git diff`
  - No changes shown as diff compares to the index
- `$ git diff HEAD`
  - Now we can see changes in working version
- `$git commit –m 'Second commit'`

Undoing What Is Done

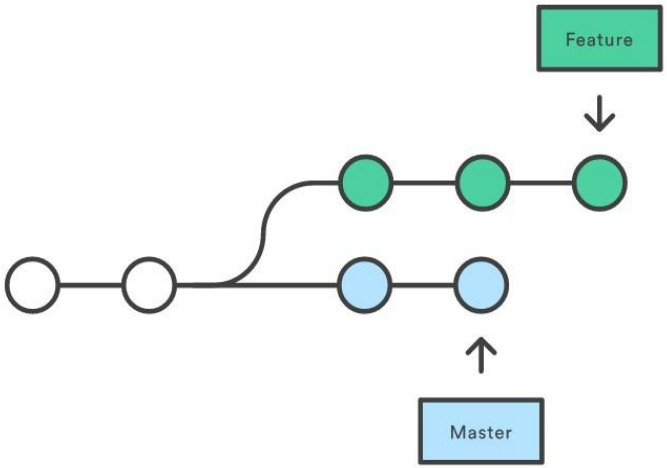
- **git checkout**
  - Used to checkout a specific version/branch of the tree
  - `git rebase master` (returns to current working version)
- **git revert**
  - Reverts a commit
  - Does not delete the commit object, just applies a patch
  - Reverts can themselves be reverted!
- **Git never deletes a commit object**
  - It is very hard to lose data

Git Rebase

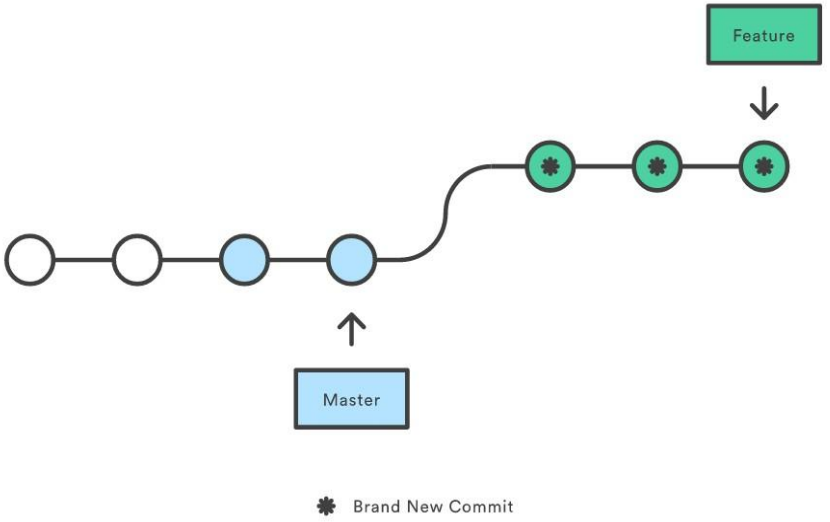
- Rewrites commit history.
- Loses context
- Never use this on public branches!
- How to rebase?

```
$ git checkout feature
$ git rebase master
```

A forked commit history

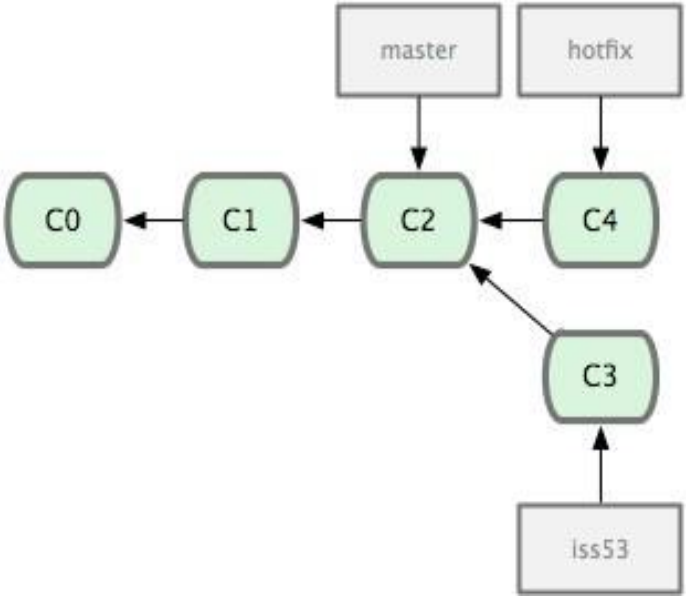


Rebasing the feature branch onto master



Source - https://www.atlassian.com/git/tutorials

Merging



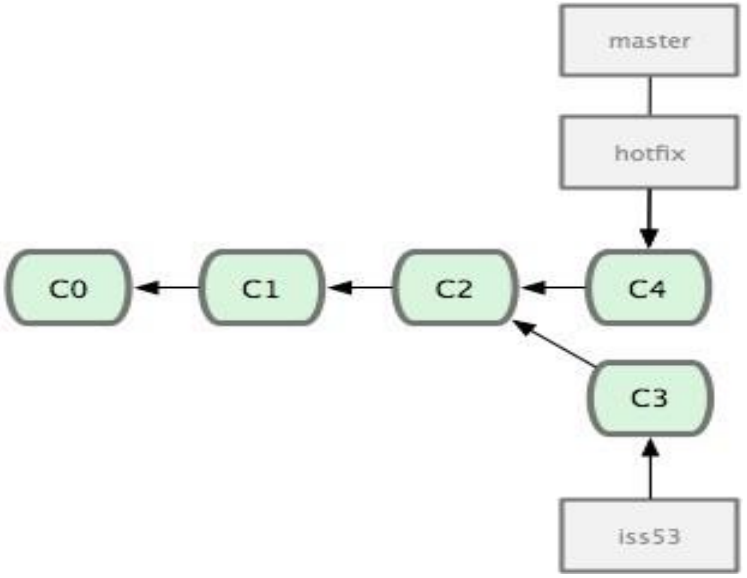


Image Source: git-scm.com

- Merging hotfix branch into master
  - git checkout master
  - git merge hotfix
- Git tries to merge automatically
  - Simple if it is a forward merge
  - Otherwise, you have to manually resolve conflicts

Merging

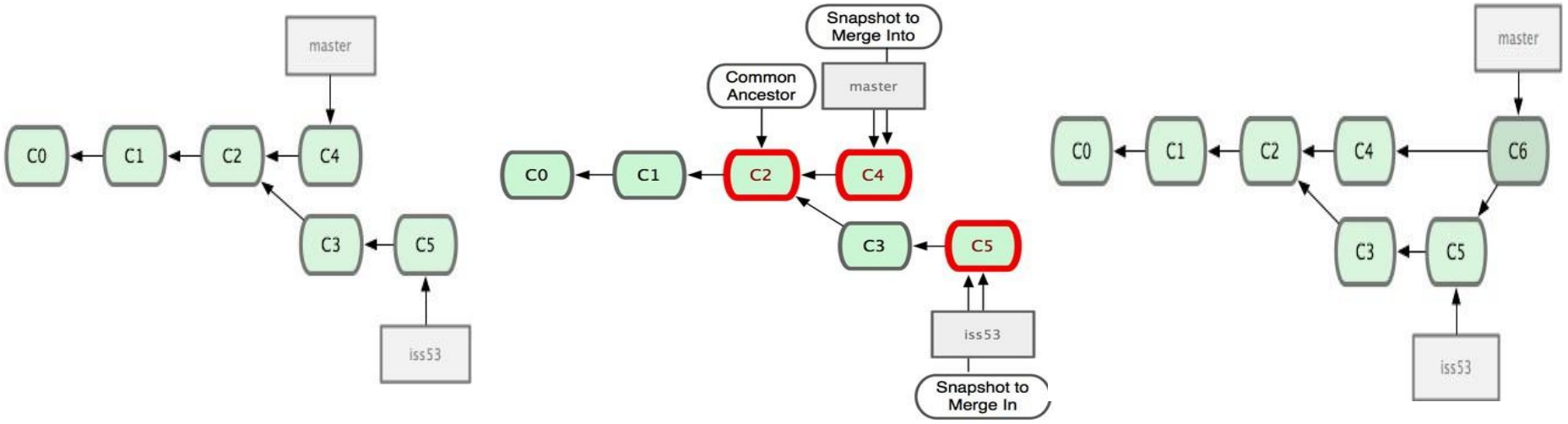


Image Source: git-scm.com

- Merge iss53 into master
- Git tries to merge automatically by looking at the changes since the common ancestor commit
- Manually merge using 3-way merge or 2-way merge
  - Merge conflicts - Same part of the file was changed differently

Merging

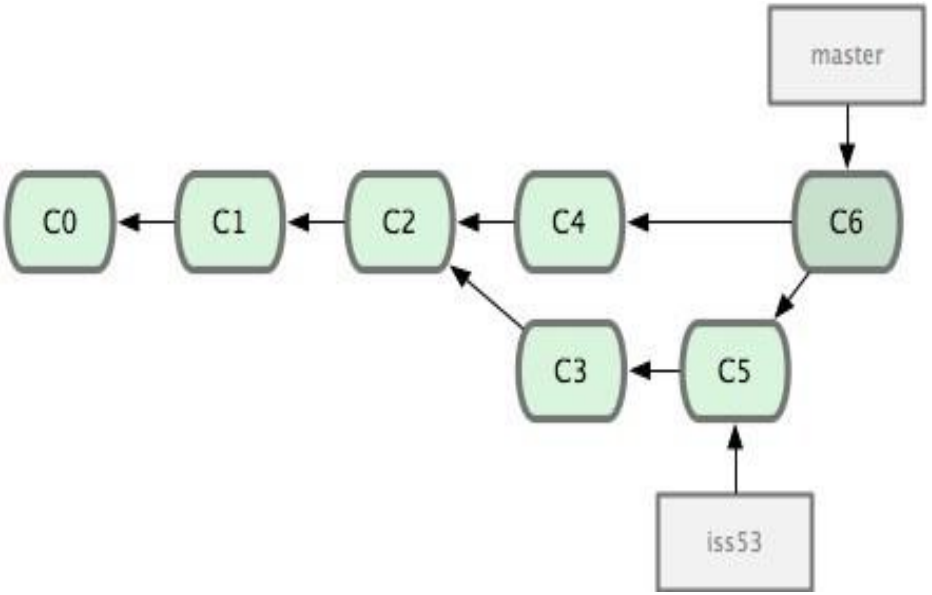


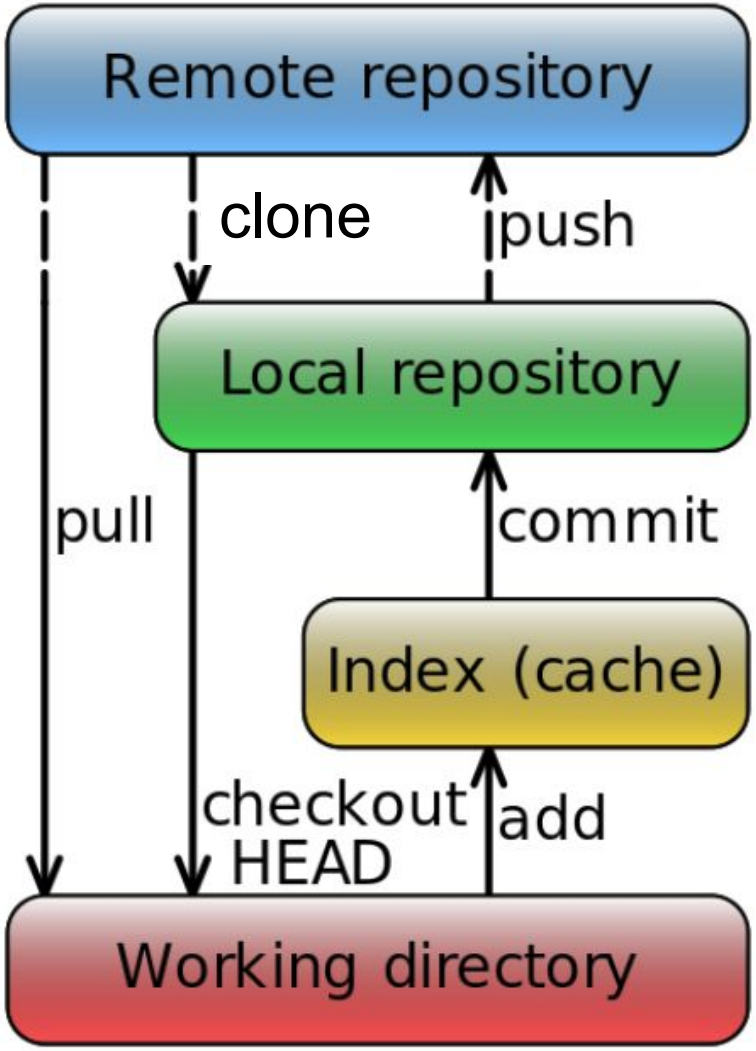
Image Source: git-scm.com

- Refer to multiple parents
  - git show hash
  - git show hash^2 (shows second parent)
- HEAD^^ == HEAD~2

More Git commands

- Reverting
  - **`git checkout HEAD main.cpp`**
    - Gets the HEAD revision for the working copy
  - **`git checkout -- main.cpp`**
    - Reverts changes in the working directory
  - **`git revert`**
    - Reverts commits (this creates new commits)
- Cleaning up untracked files
  - **`git clean`**
- Tagging
  - Human readable pointers to specific commits
  - **`git tag -a v1.0 -m ‘Version 1.0’`**
    - This will name the HEAD commit as v1.0

Overview


Remote repository
clone
push
Local repository
pull
commit
Index (cache)
checkout
HEAD
add
Working directory

Assignment 4

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

Backporting

Taking a certain software modification (patch) and **applying it to an older version** of the software than it was initially created for.

Assignment 4

- Fix an issue with the diff diagnostic
- Hints for the first few steps
  - **`git clone`**

- Homework
  - Patch file in a particular format(email)
  - **`git format -patch -[num] --stdout`**
    - **`man git format-patch`** to find out what -[num] means
  - **`git am patchfile`**
  - For running gitk, you will have to enable X forwarding
    - **`ssh -x username@lnxsrv07.seas.ucla.edu`**
    - Need x11 installed on your local machine

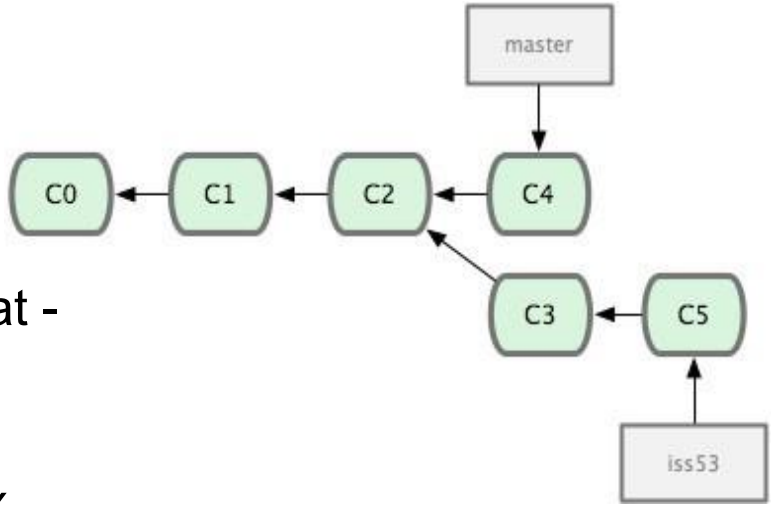


Image Courtesy: git-scm.com

Useful Links

- Git Tutorial
  - By topic
- Git Beginner's Tutorial (alternative)
  - Step by step tutorial + testing terminal
- Git Visual Guide
  - For visualizing what each command does
- Git From The Bottom Up
  - For understanding how Git is structured and the details of how it tracks changes

More Git hints

- Git beginner's tutorial (highly recommended):
  - Click here
- Git cheat sheet:
  - Click here
- Putty X11 forwarding (you'll need this for gitk):
  - Click here
- gitk introduction/tutorial:
  - Click here

Lab

web.cs.ucla.edu/classes/winter16/cs35L/assign/assign4.html