Source control - GIT

Lab 4
Sharath Gopal

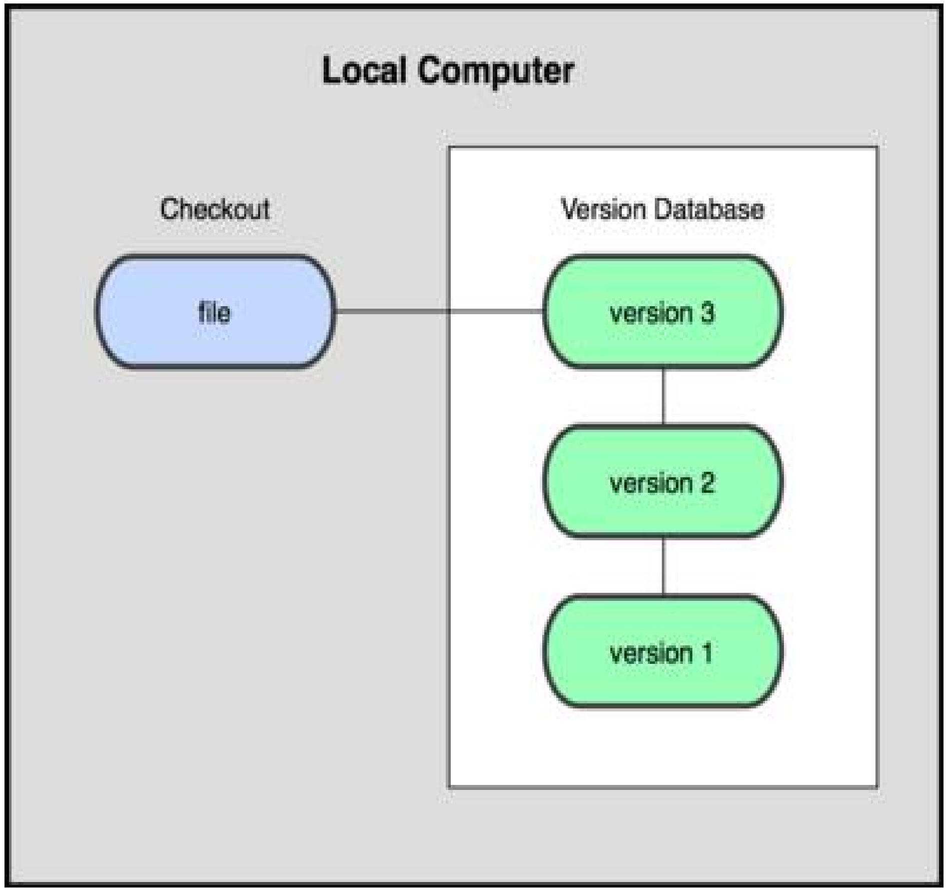
Software development process

- Involves making a lot of changes to code
  - New features added
  - Bugs fixed
  - Performance enhancements
- Software team has many people working on the 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 for customers using it, even though you have shipped Ubuntu 12.

Source/Version Control

- Track changes to code and other files related to the software
  - What new files were added?
  - What changes made to files?
  - Which version had what changes?
  - Which user made the changes?
- Track entire history of the software
- Source control softwares
  - GIT, Subversion, Perforce

Local SCS



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

Image Source: git-scm.com

Centralized SCS

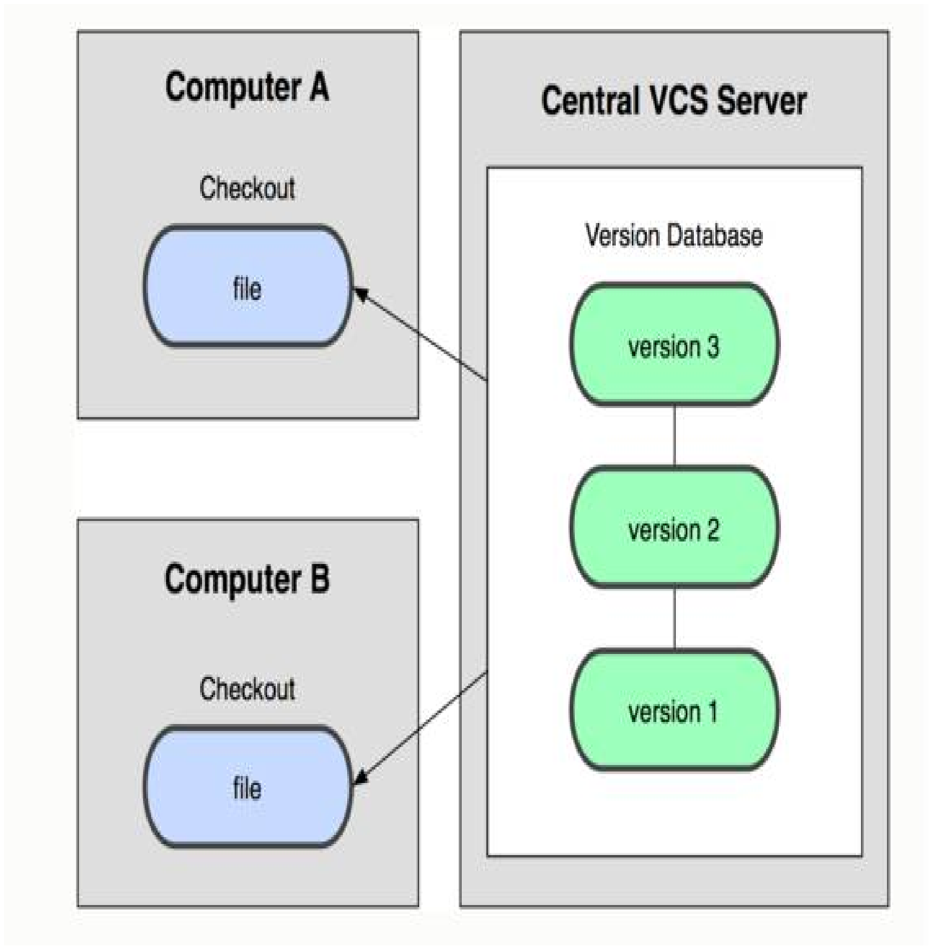


Image Source: git-scm.com

- 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 SCS

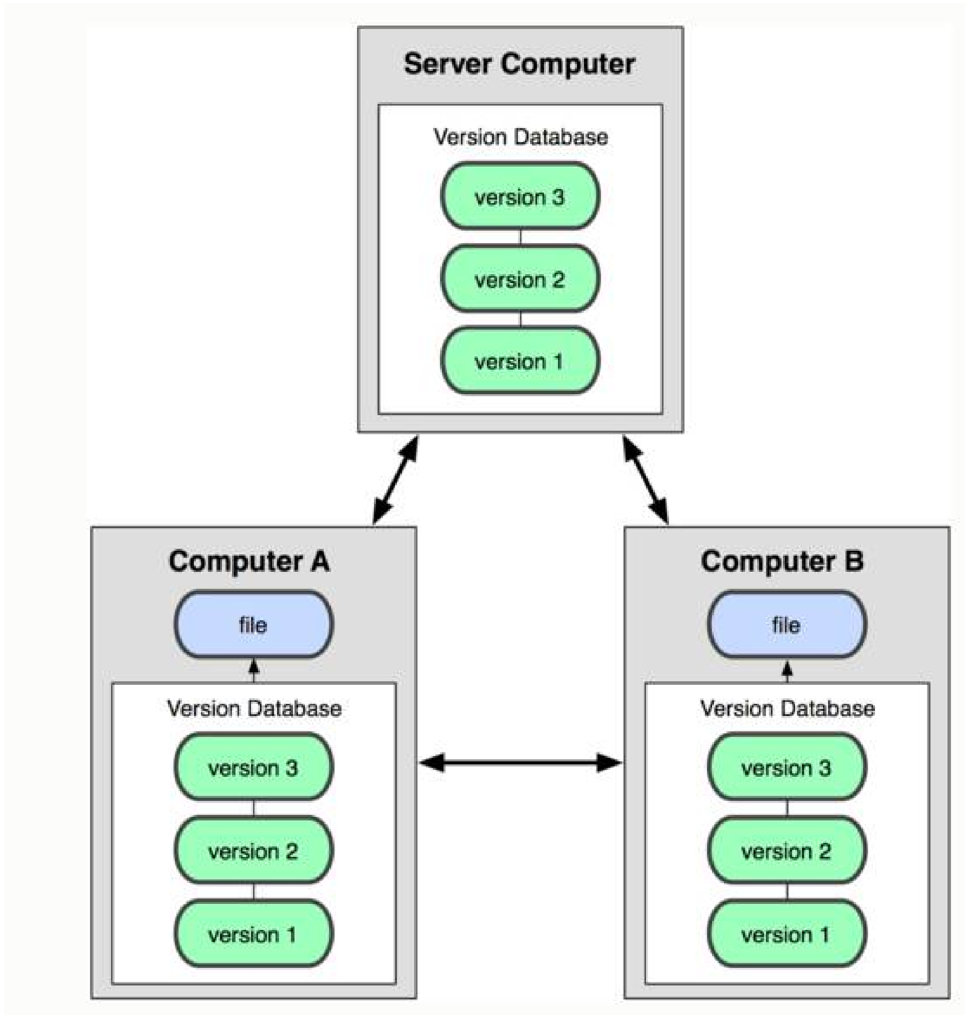


Image Source: git-scm.com

- Version history is replicated at 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 folder 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

GIT Source control

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**
    - Just a named commit object for convenience (example: versions of the software)
- Objects uniquely identified with **hashes**

Git States

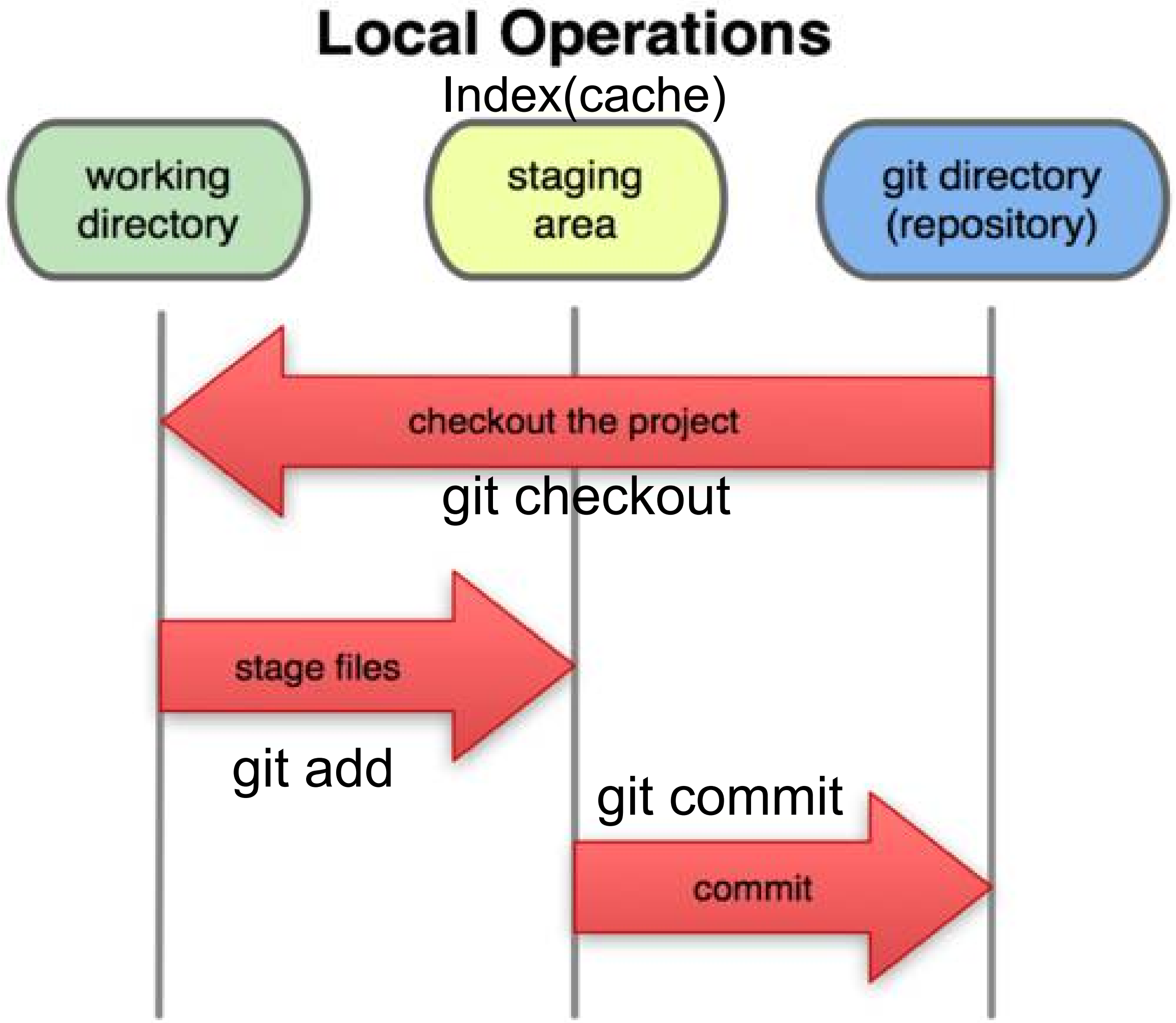


Image Source: git-scm.com

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

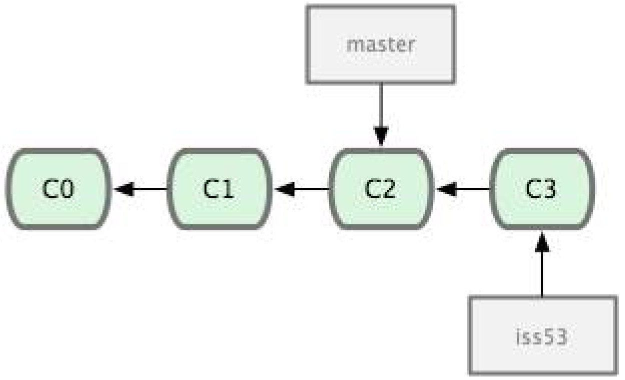


Image Source: git-scm.com

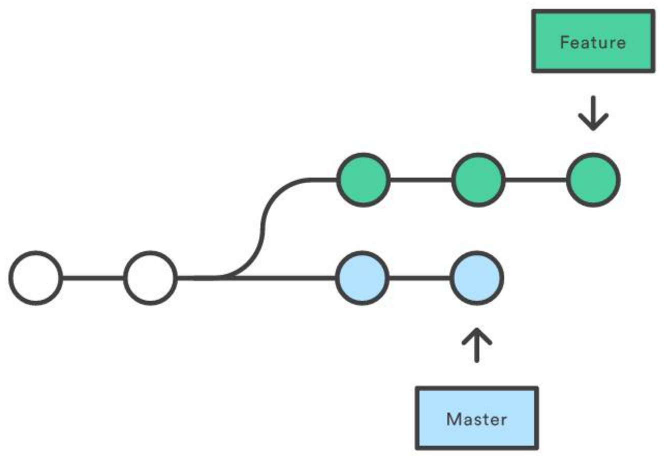
Git commands

- Repository creation
  - git init (Start a new repository)
  - git clone (Create a copy of an exisiting repository)
- Branching
  - git checkout <tag/commit> -b <new_branch_name> (creates a new branch)
- Commits
  - git add (Stage modified files)
  - git commit (check-in the 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

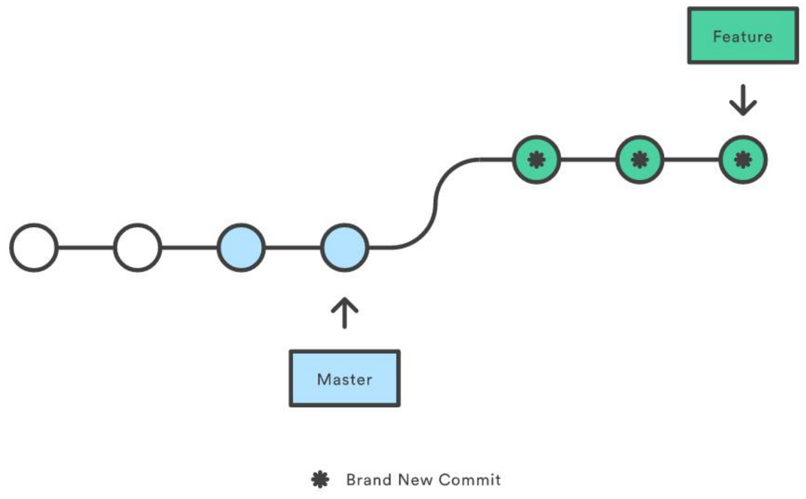
Git Rebase

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

A forked commit history

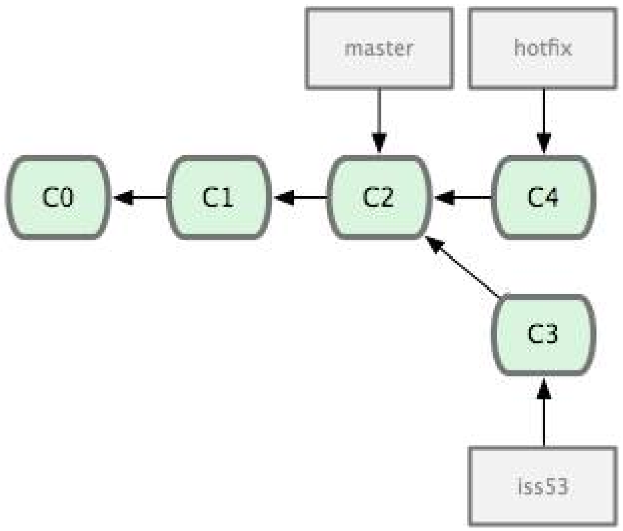


Rebasing the feature branch onto master



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

Merging



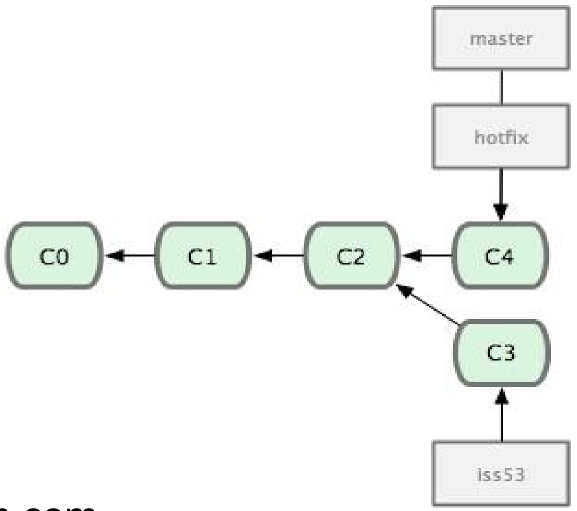


Image Source: git-scm.com

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

Merging

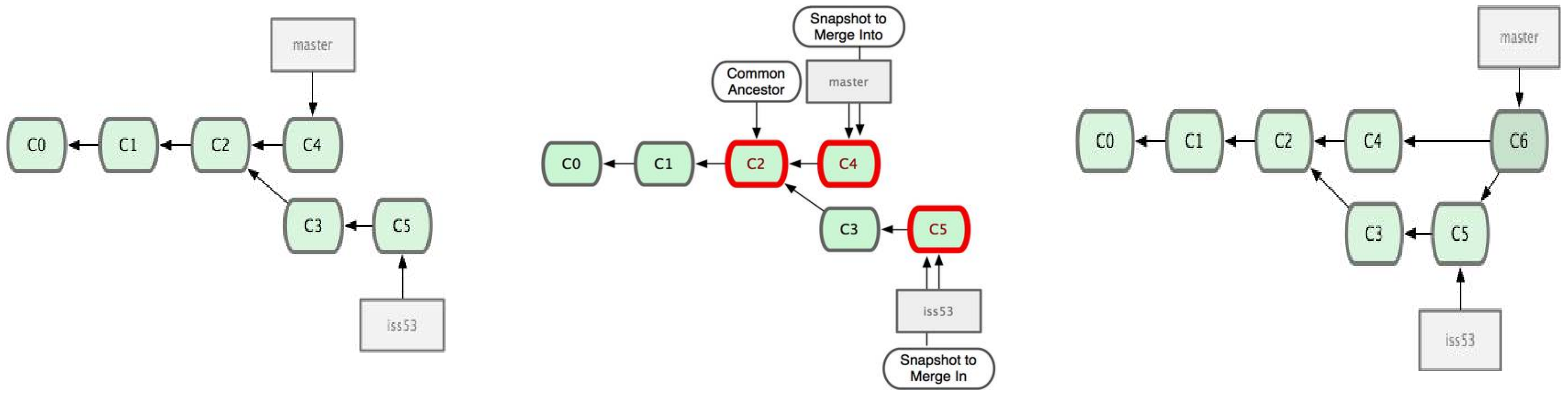


Image Source: git-scm.com

- 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

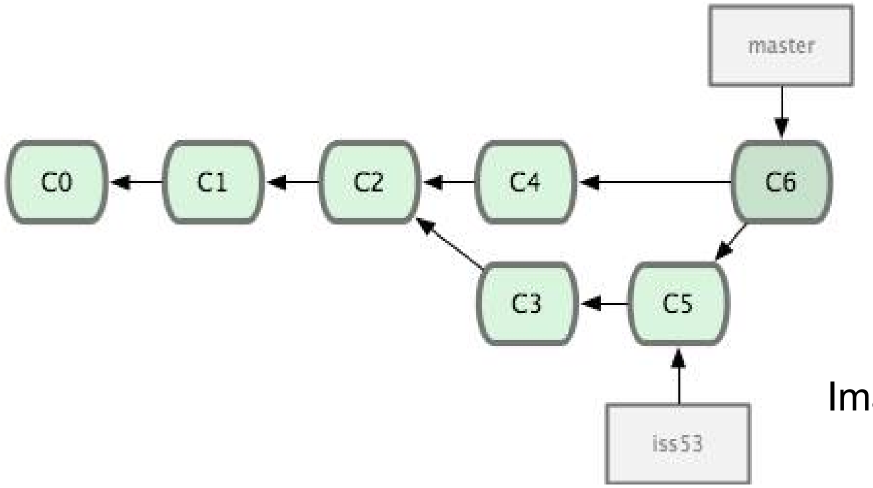


Image Source: git-scm.com

- 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
    - Reverting 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

Assignment 4

- Backporting
  - Apply a patch to a previous version
- Fix an issue with the diff diagnostic
- Hints for the first few steps
  - 1) git clone
  - 2) git log
  - 3) git tag
  - 4) git show <hash_value>
  - 5) git checkout v3.0 -b <BranchName>
  - ...
- 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@lnxsrv.seas.ucla.edu
    - You have to install X11 on your local machine

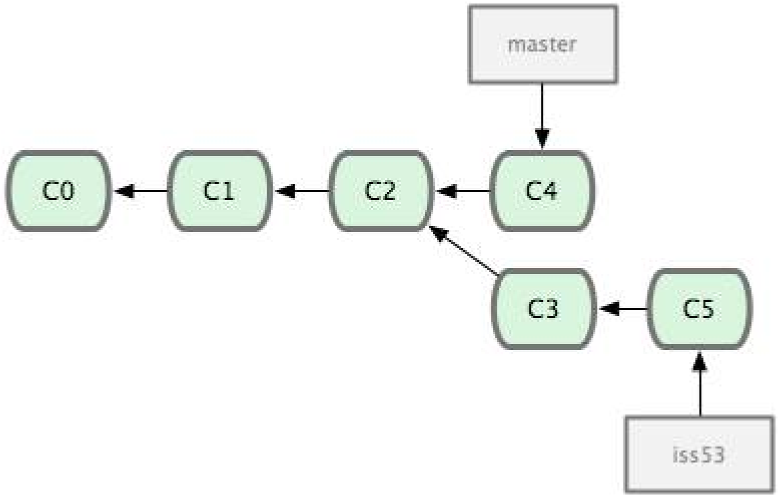


Image Courtesy: git-scm.com

GITK – Git repo browser

To run gitk on seasnet, you have to setup X11 on your local machine and use X forwarding

- `ssh -X username@lnxsrv.seas.ucla.edu`
- X11 for different operating systems
  - Windows - You can explore one of these links
    - http://www.straightrunning.com/XmingNotes/
    - http://sourceforge.net/projects/xming/
    - (Instructions for PUTTY) https://wiki.utdallas.edu/wiki/display/FAQ/X11+Forwarding+using+Xming+and+PuTTY
  - Linux
    - X11 should already be installed.
  - Mac
    - Install http://xquartz.macosforge.org/landing/