CS 35L

Software Construction Laboratory

Lecture 9.2

7th March, 2019

Logistics

- Final Exam
  - Date: 17th March, 2019 (Sunday)
  - Time: 3pm to 6pm
  - Location: Franz 1178
- Presentations for Assignment 10
  - https://docs.google.com/spreadsheets/d/1o6r6CKCaB2du3klPflHiquymhBvbn7oP0wkHHMz_q1E/edit?usp=sharing
- Assignment 8 is due on *12th March, 2018* at 11:55pm
- Instructor Evaluation

Review - Previous Lab

- Digital Signature
  - Message Digest
  - Steps to generate a digital Signature
- BeagleBone Assignment

Change Management

Software Development Process

- Involves making a lot of changes to code
  - Addition of features
  - Bug fixing
  - Performance enhancements
- A software team typically has many people working on the same/different parts of code
- Different versions of software released
  - Ubuntu 16, Ubuntu 18
  - Should have ability to fix bugs for users using Ubuntu 16 even after Ubuntu 18 is launched

Source/Version Control

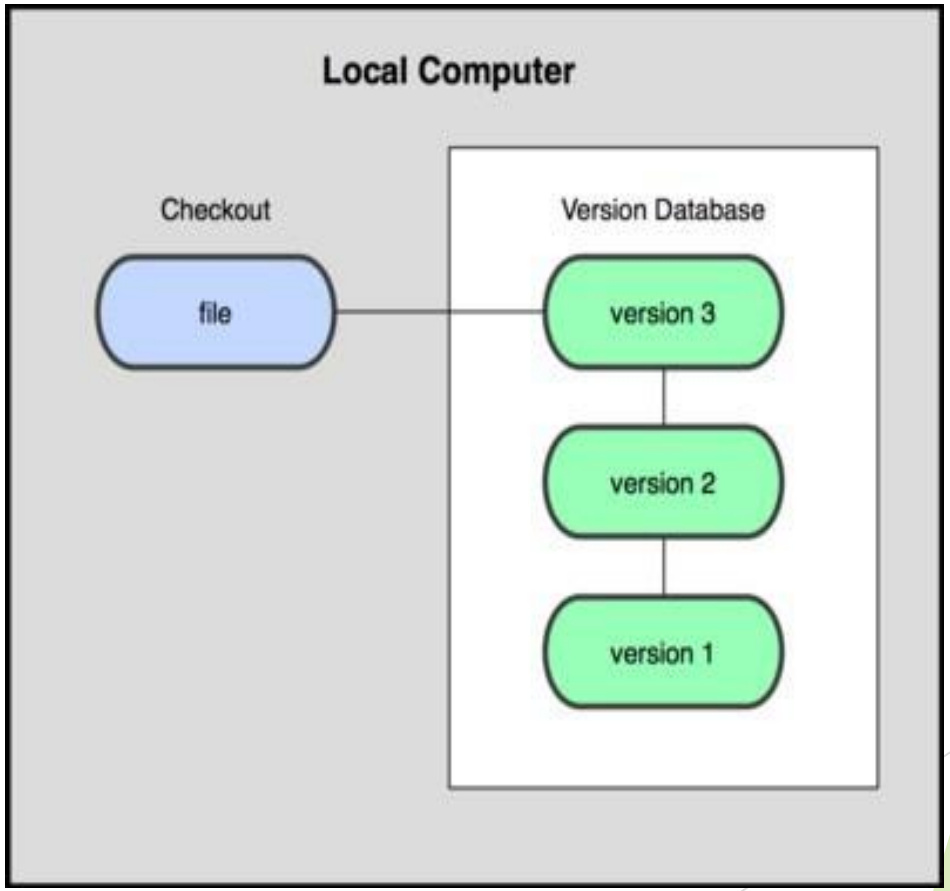
- Track changes to code and other files related to the software
  - What new files were added?
  - What changes were made to files?
  - Which version had what changes?
  - Who made the changes?
- Track entire history of the software
- Version control system (VCS)

Version Control System (VCS)

- Version Control System records changes to a file or set of files over time so that you can recall specific versions later.
- Advantage
  - Everybody on the team is able to work freely on any file at any time
  - Later merge changes into a common version
- Three Models of VCS:
  - Local
  - Centralized
  - Distributed
- Examples:
  - Git, Perforce, Subversion

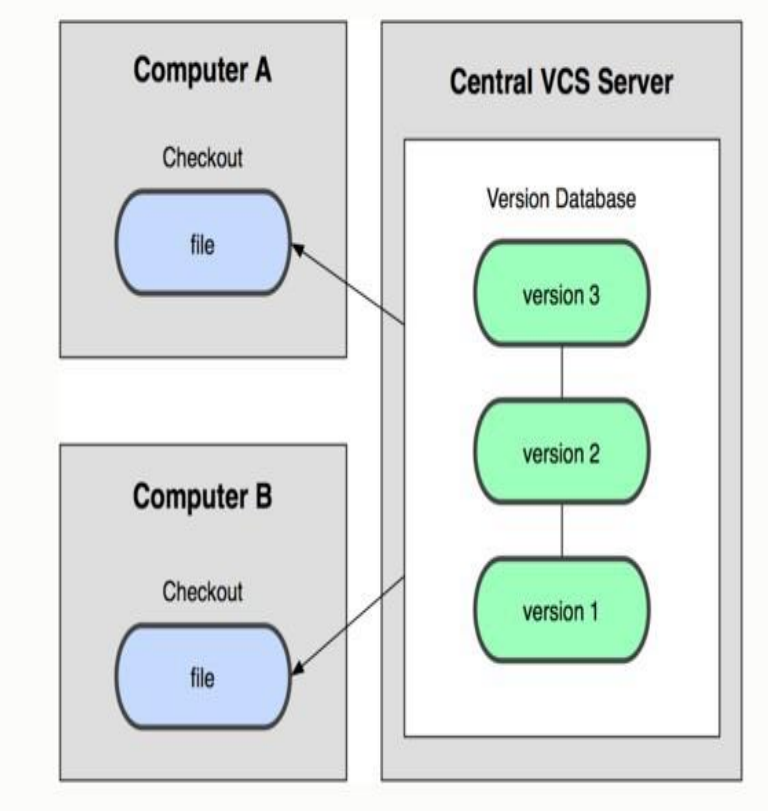
Local VCS

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



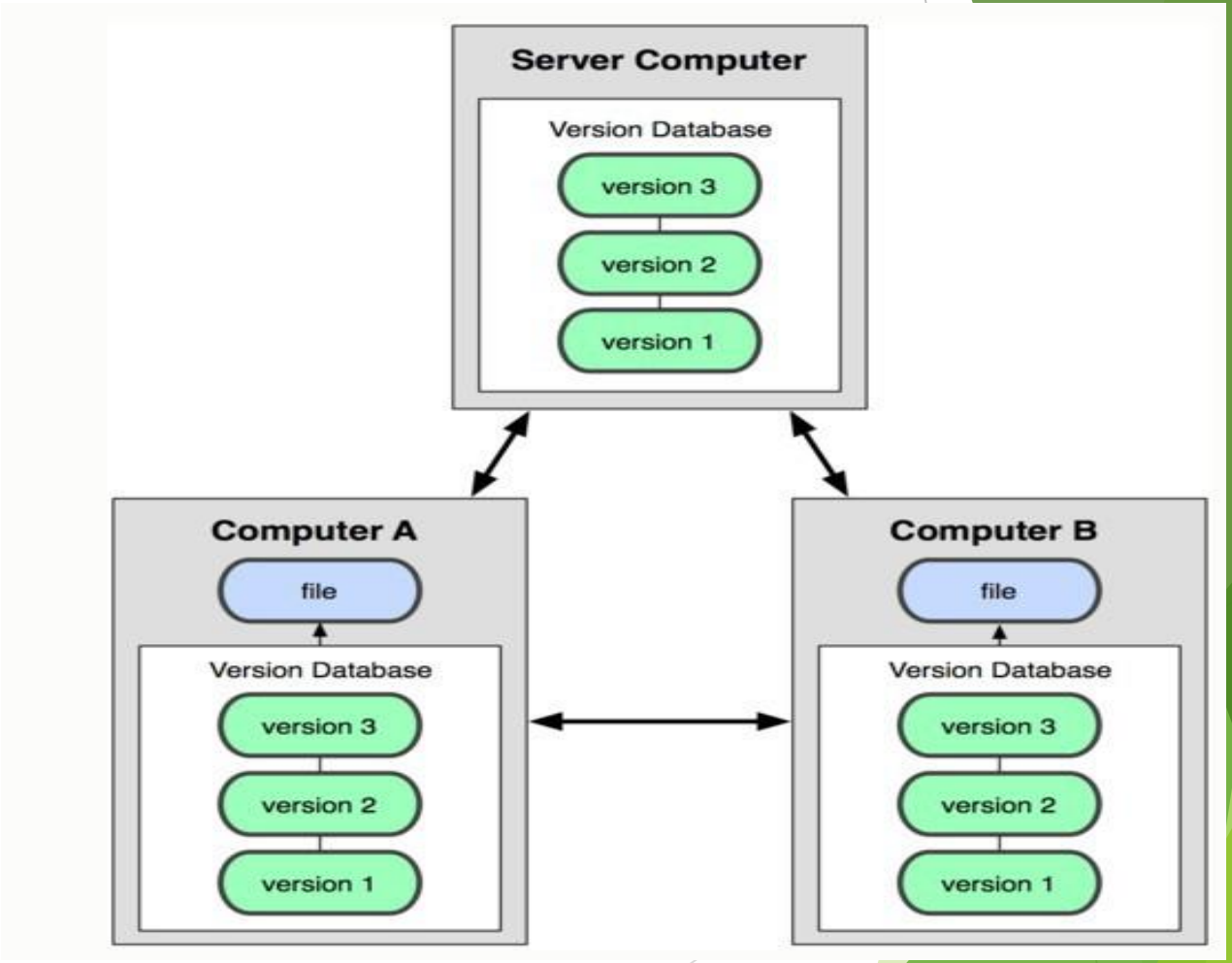
Centralized VCS

- 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 VCS

- 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

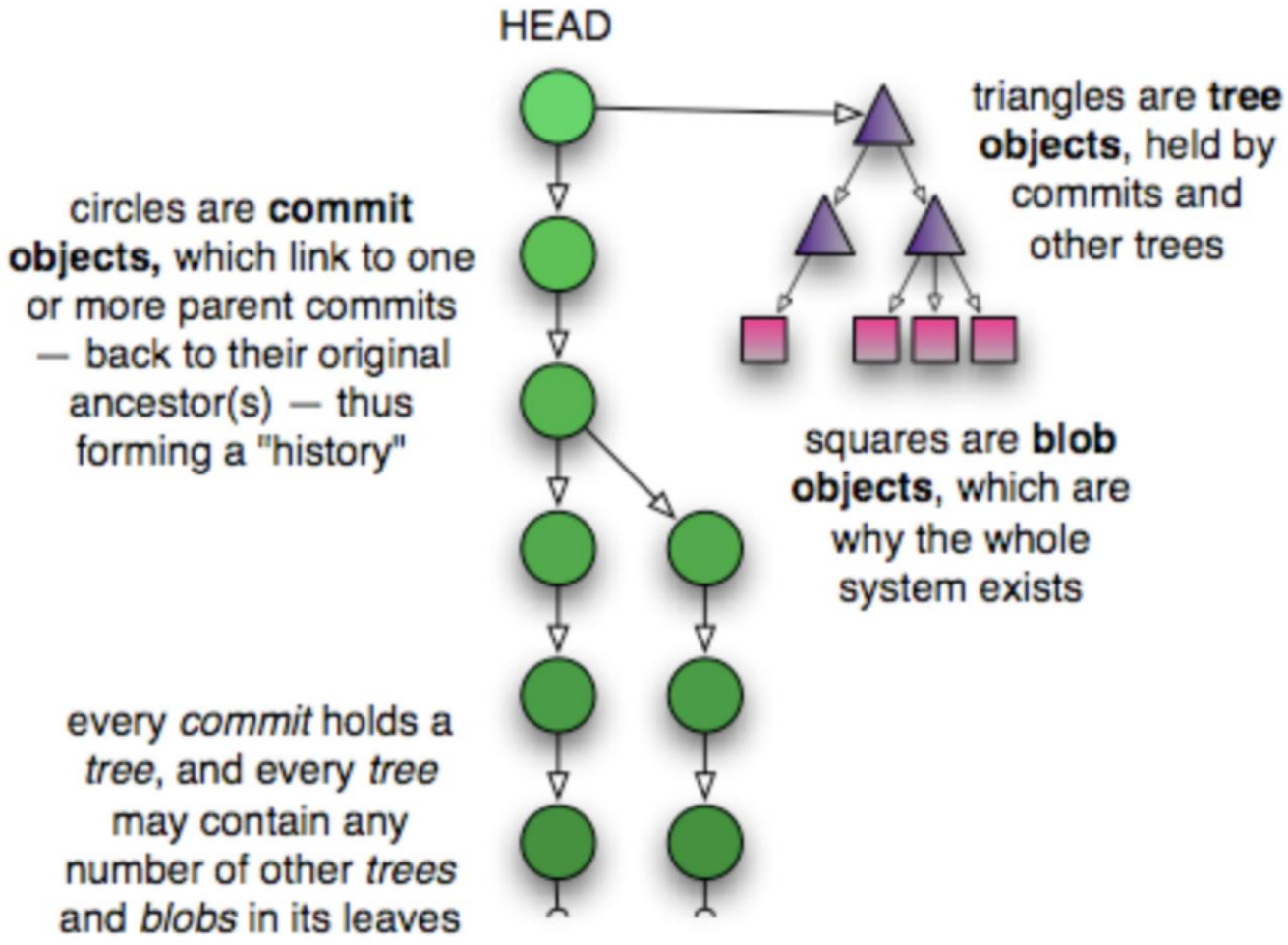


Terms used

- Repository
  - Files and folder related to the software code
  - Full History of the software
- Working Copy
  - Copy of software files in the local repository
- Clone
  - 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 VCS

GIT Source Control

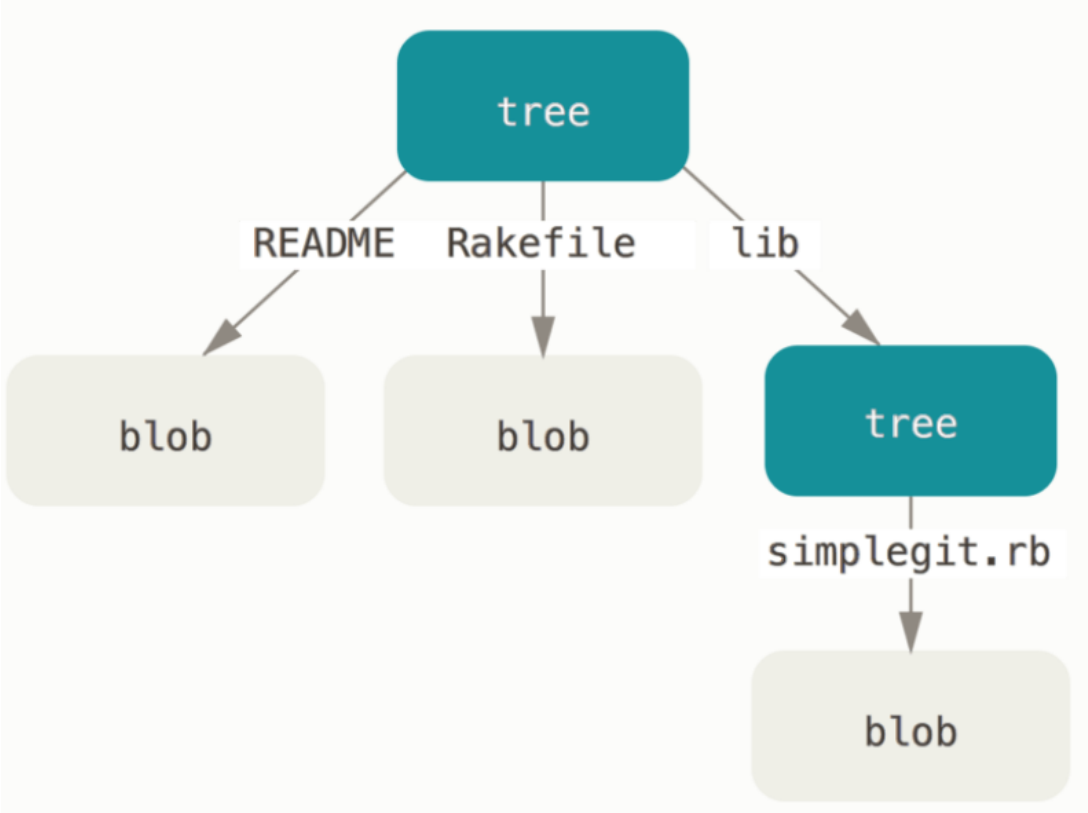
Git Repository Objects



Git Repository Objects

- Blobs (Binary Large Objects):
  - 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.
  - The file's SHA-1 hash is computed and stored
- Trees
  - The tree object contains one line per file or subdirectory, with each line giving file permissions, object type, object hash and filename. Object type is usually one of “blob” for a file or “tree” for a subdirectory
- Commit
  - The commit object contains the directory tree object hash, parent commit hash, author, committer, date and message.
- Tags
  - The tag object type contains the hash of the tagged object, the type of tagged object (usually a commit), the tag name, author, date and message
- Objects uniquely identified with hashes
- https://matthew-brett.github.io/curious-git/git_object_types.html

Example - Trees




```
100644 blob a906cb2a4a904a152e80877d4088654daad0c859      README
100644 blob 8f94139338f9404f26296befa88755fc2598c289      Rakefile
040000 tree 99f1a6d12cb4b6f19c8655fca46c3ecf317074e0      lib
```

Git States

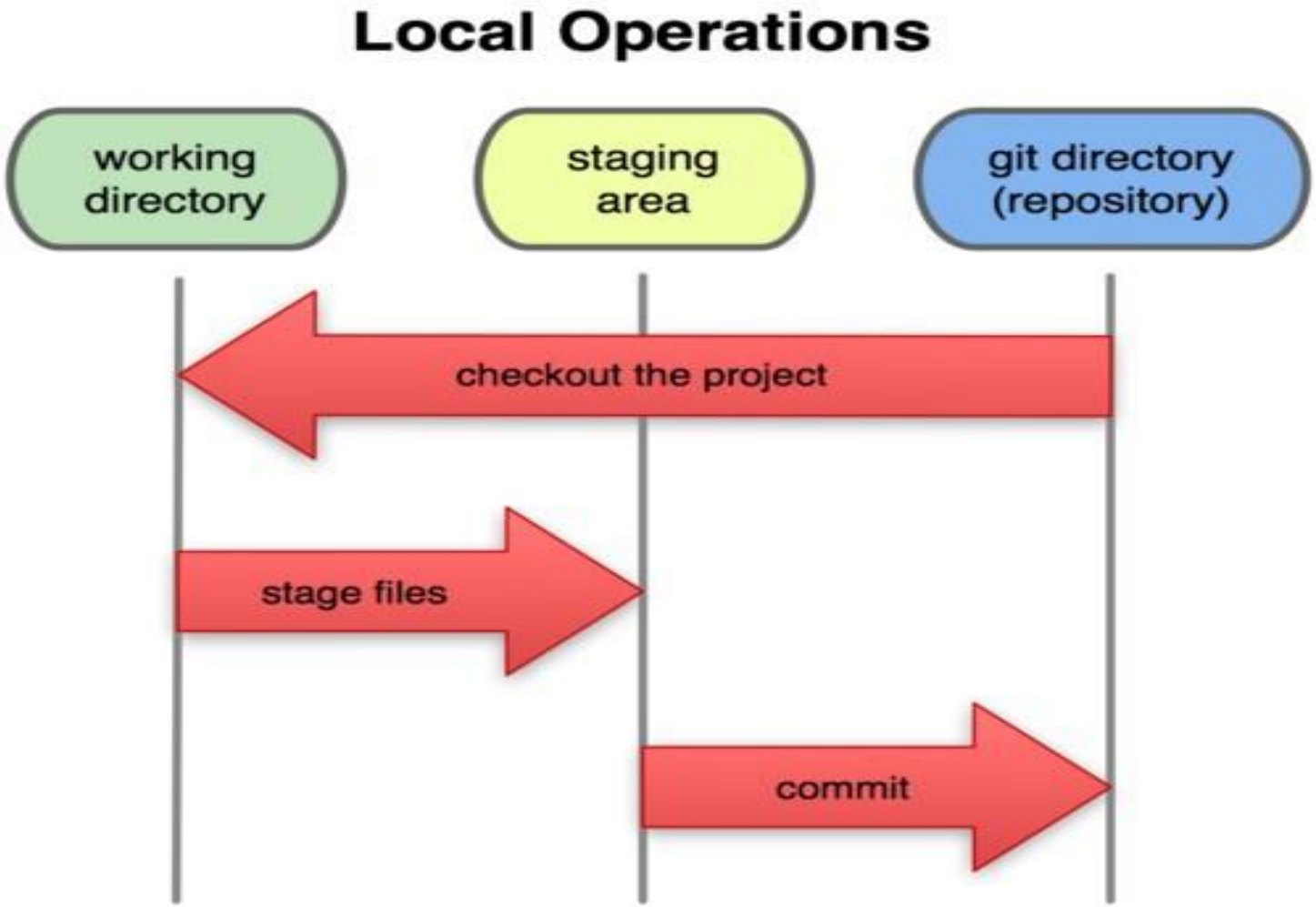


Image Source: git-scm.com

Terms used

- HEAD
  - Refers to the currently active head
  - Refers to a commit object
- Branch
  - Refers to a head and its entire set of ancestor commits
- Master
  - Default branch

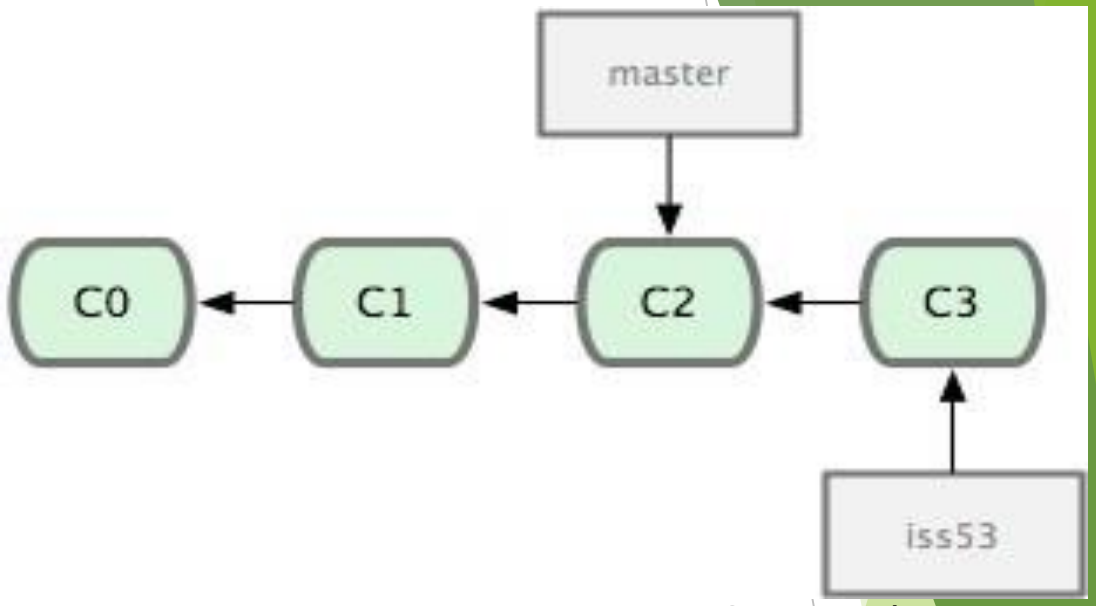


Image Source: git-scm.com

Git Commands

- Repository Creation
  - git init (Create a new repository)
  - git clone (Create a copy of an existing repository)
- Branching
  - git branch <new branch name>
  - git checkout <tag/commit> -b <new branch name> (Creates a new branch)
- Commit
  - git add (Stage modified/new files)
  - git commit (Check in the changes to the repository)

Git Commands

- 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

First Git Repository

- Mkdir gittest
- Cd gittest
- Git init
  - Creates an empty git repo (.git directory with all necessary directories)
- Echo “Hello World” > hello.txt
- Git add .
  - Adds content to the index
  - Must be run prior to a commit
- Git commit –m “First Commit”

Working with git

- Echo “I love git!” >> hello.txt
- Git status
  - Shows list of modified files
- Git diff
  - Shows changes we made compared to the original index
- Git add hello.txt
- Git diff
- Git diff HEAD
- Git commit –m “Second commit”

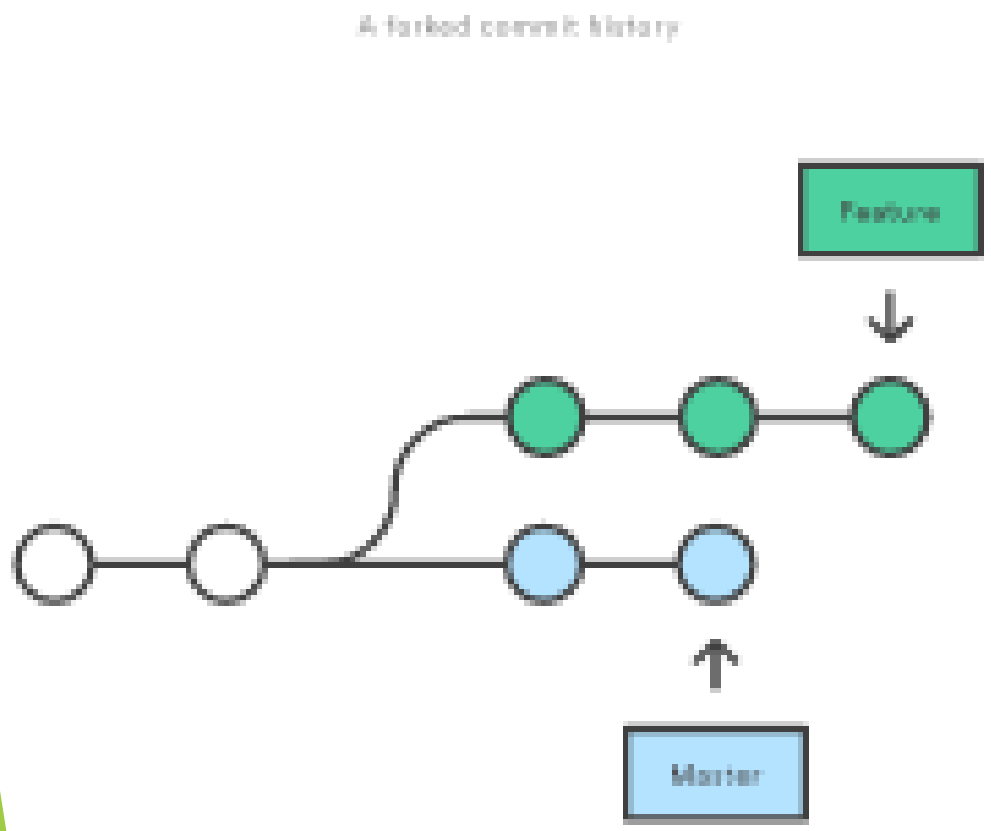
Working with branches

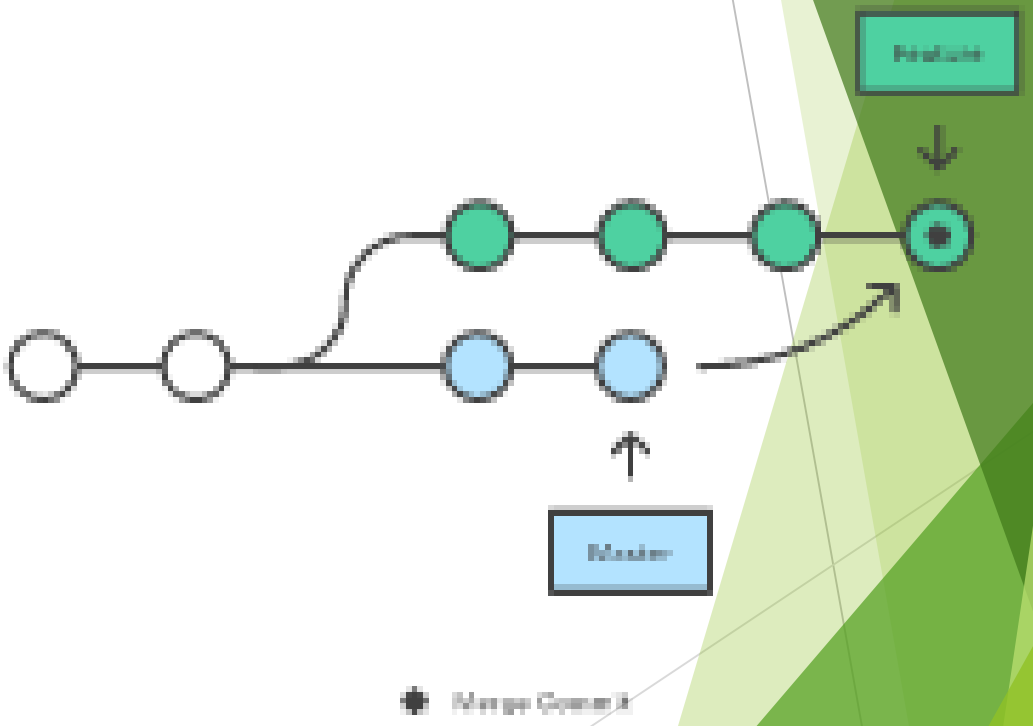
- Git branch test
  - Create new branch
- Git branch
  - List all branches
- Git checkout test
  - Switch to test branch
- Echo “Hello World!” > hw
- Commit the change in new branch
- Git checkout master
  - Back to master branch
- Git log
- Git merge test
  - Merge commits from test branch to current branch

Git integrating changes

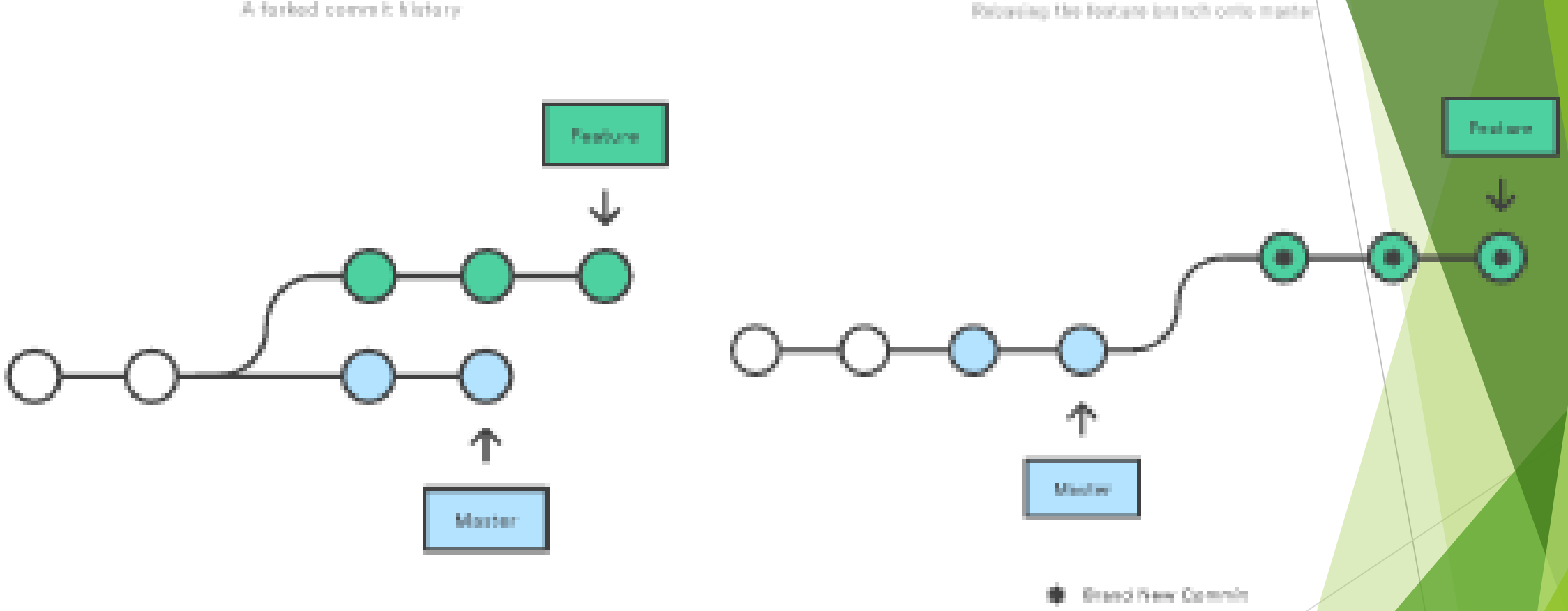
- Required when there are changes in multiple branches
- Two main ways to integrate changes from one branch to another
  - merge
  - rebase
- Merge is simple and straightforward
- Rebase is much cleaner

Git merge





Git rebase



Merge Conflicts

- Usually git will do merge automatically
- Conflict arises when you changed the same part of the same file differently in the two branches you're merging together
- The new commit object will not be created
- You need to resolve conflicts manually by selecting which parts of the file you want to keep

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 9

- Deadline
  - 16th March, 2018, 11:55pm
  - NO late submissions accepted

Assignment 9 - Laboratory

- Fix an issue with diff diagnostic
  - Apply a patch to a previous version
- Installing Git
  - Ubuntu: sudo apt-get install git
  - SEASNet: git is installed in /usr/local/cs/bin
  - Add it to PATH variable or use whole path
    - Export PATH=/usr/localcs/bin:$PATH
- Make a directry ‘gitroot’ and get a copy of the diffutils git repository
  - Mkdir gitroot
  - Cd gitroot
  - Git clone <url>
  - Follow steps given in the specs and use man git to find commands

Assignment 9 - Laboratory

- Hints
  - Git clone
  - Git log
  - Git tag
  - Git show <hash>
  - Git checkout v3.0 –b <branchname>

Presentations

- Today's Presentation:
  - Junhong Wang
  - Don
- Next up:
  - Junting Luo
  - Jefferson Lee

Questions?