

CS35L – Winter 2019

Slide set:	9.1
Slide topics:	Source control, Git
Assignment:	9

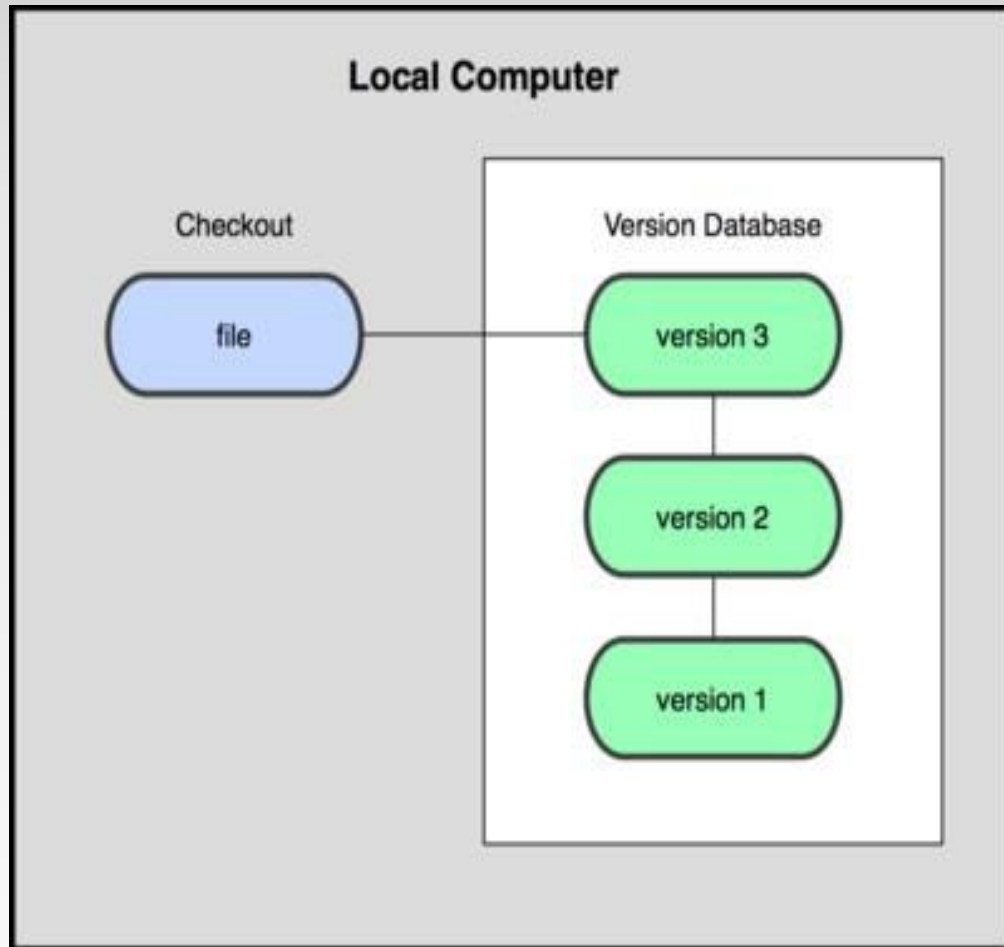
Software Development Process

- 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
- Version control software
 - GIT, Subversion, Perforce

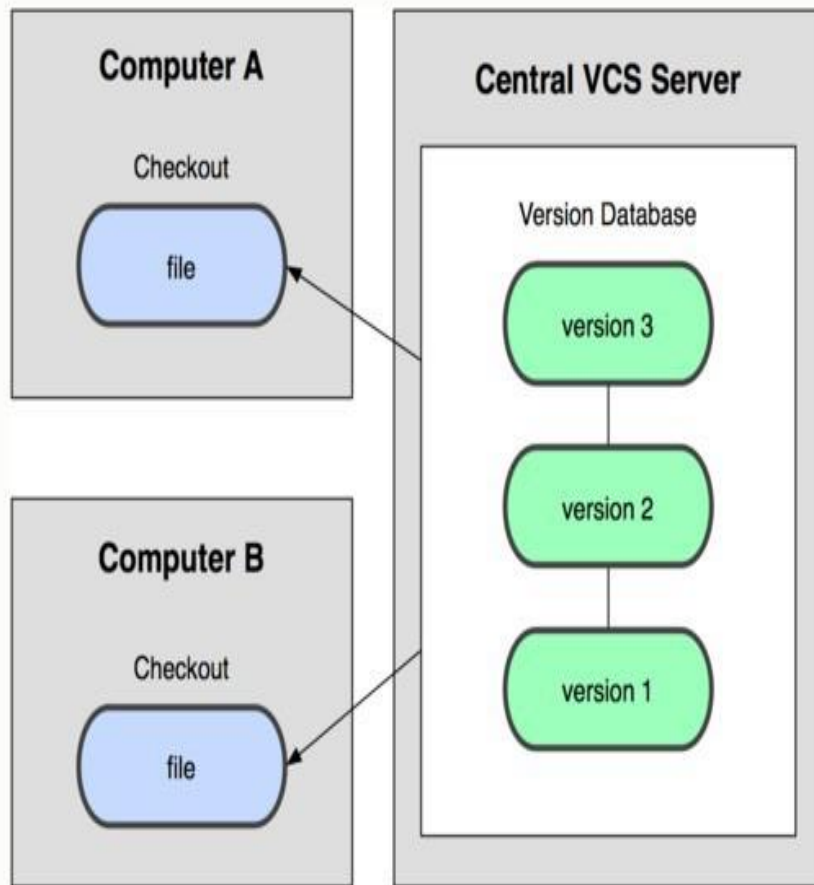
Local VCS



- 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

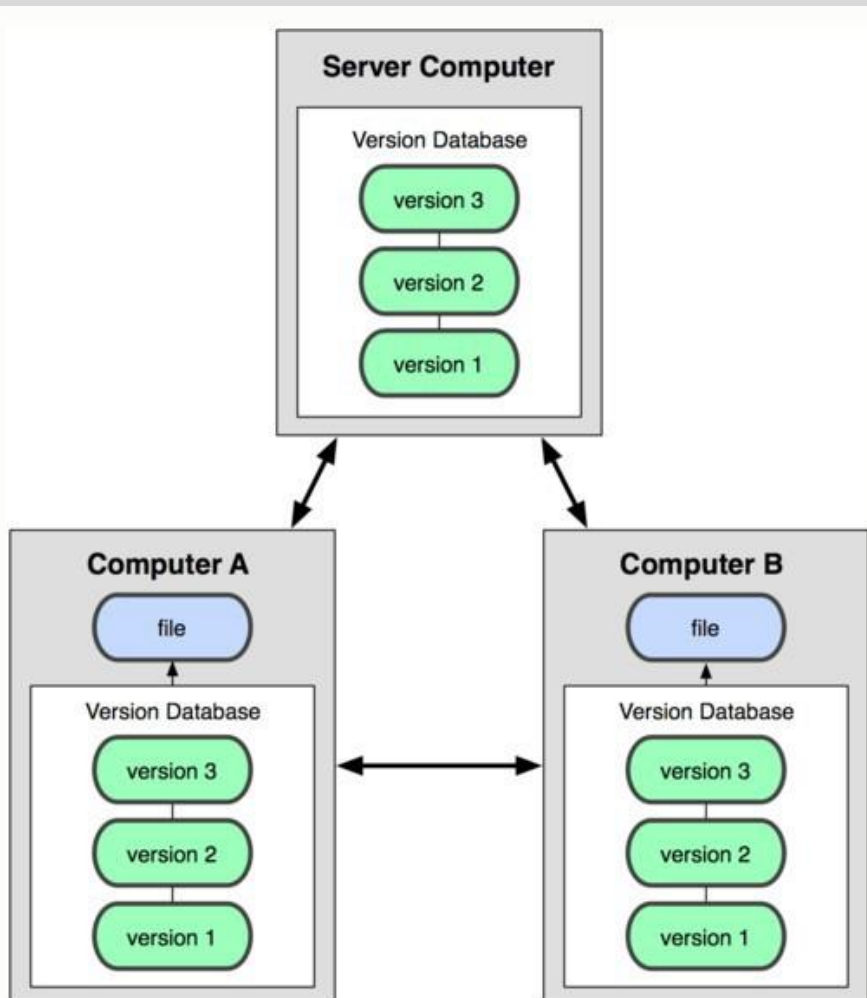
Centralized VCS



- 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

Image Source: git-scm.com

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

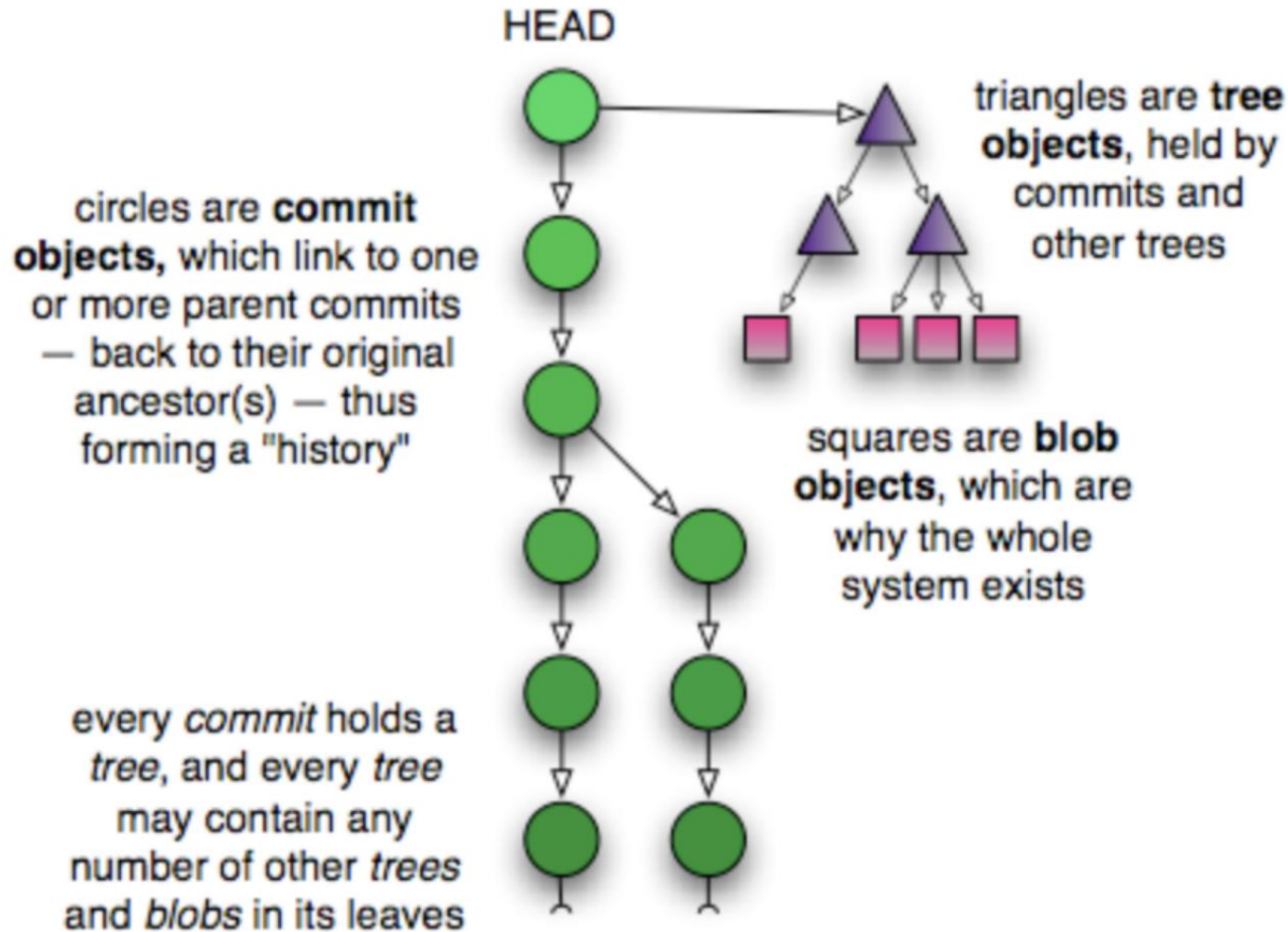
Image Source: git-scm.com

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 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.
- 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

Git States

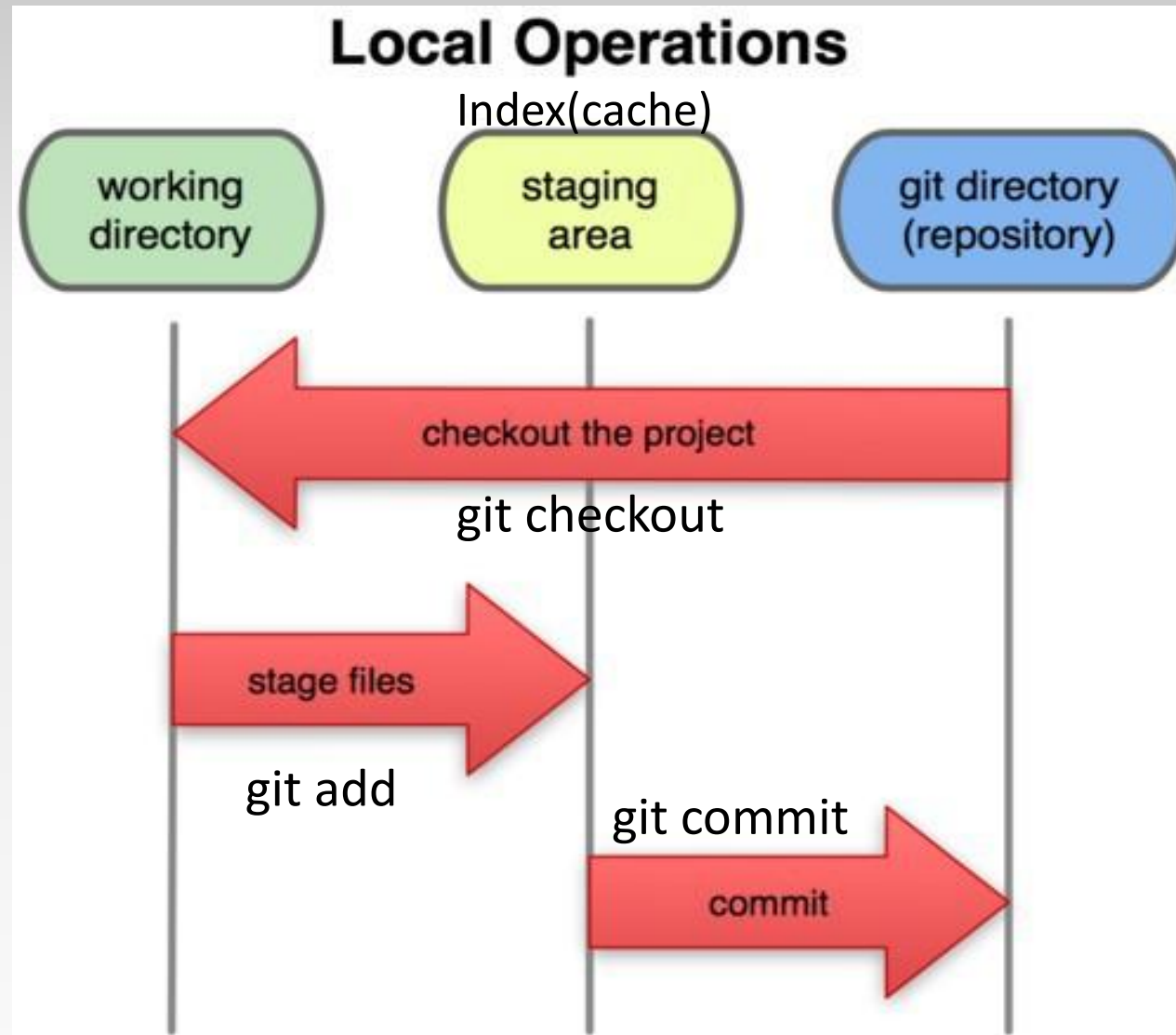


Image Source: git-scm.com

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

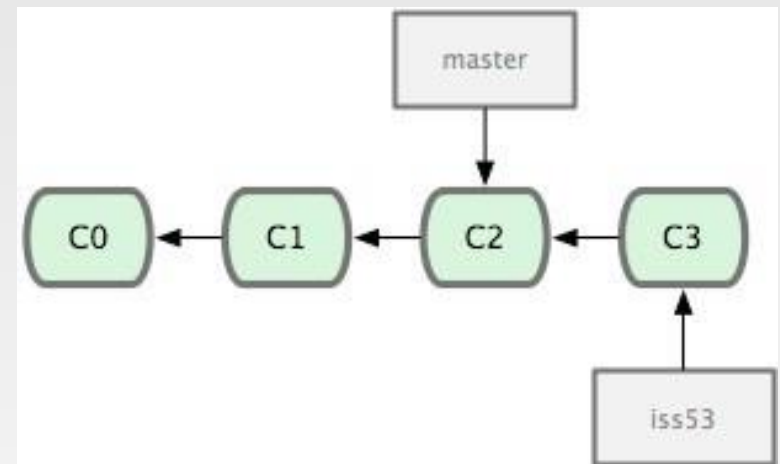


Image Source: git-scm.com

Git Commands

Repository Creation

\$ git init (Start 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)

Commits

\$ git add (Stage modified/new 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

First Git Repository

```
$ mkdir gittest
```

```
$ cd gittest
```

```
$ 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 1"
```

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"
```

Working With Branches

\$ git branch test (Create new branch)
\$ git branch (Lists all branches)
\$ git checkout test (Switch to test branch)

\$ echo "hello world!" > hw

\$ **Commit the change in new branch**

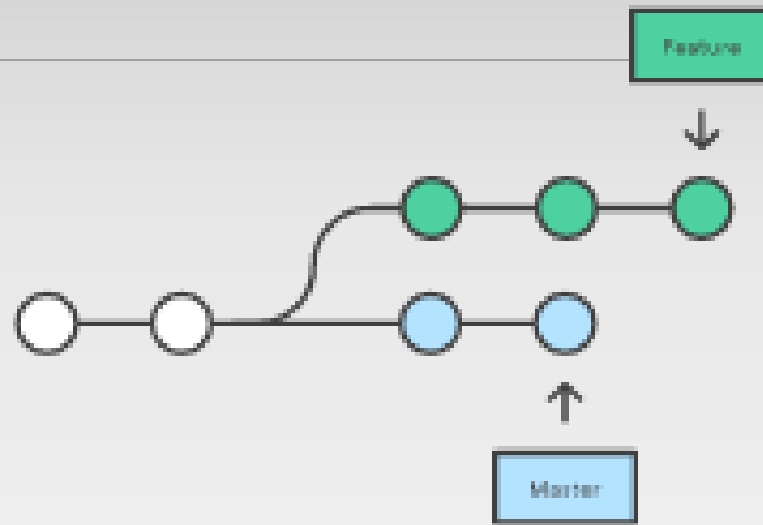
\$ git checkout master (Back to master branch)
\$ git log
\$ git merge test (Merges commits from test branch to
current branch - here master)

Git Integrating Changes

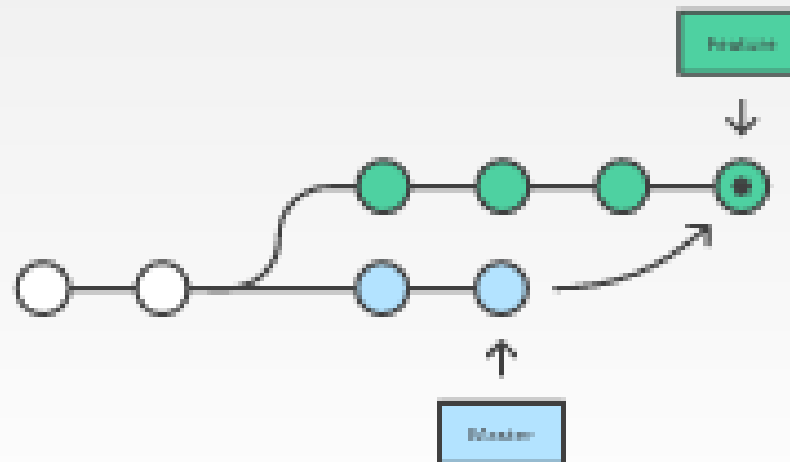
- Required when there are changes in multiple branches
- Two main ways to integrate changes from one branch to another – merge and rebase
- Merge is simple and straightforward
- Rebase is much cleaner

Git merge

A forked commit history

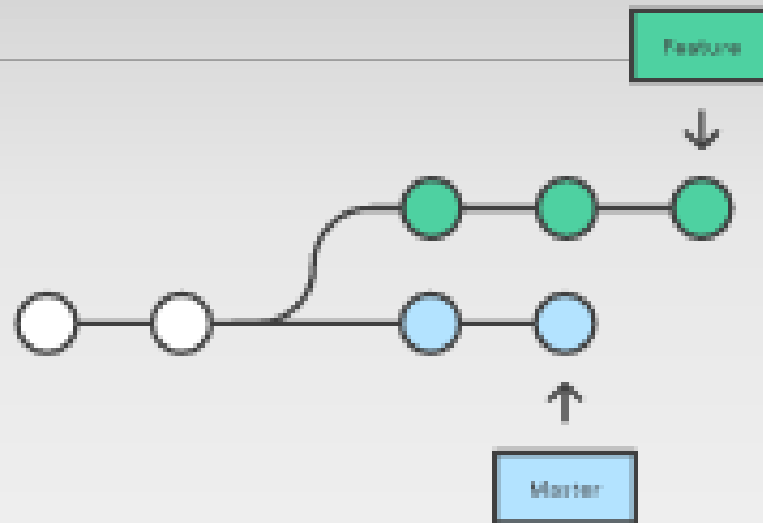


Merging master into the feature branch

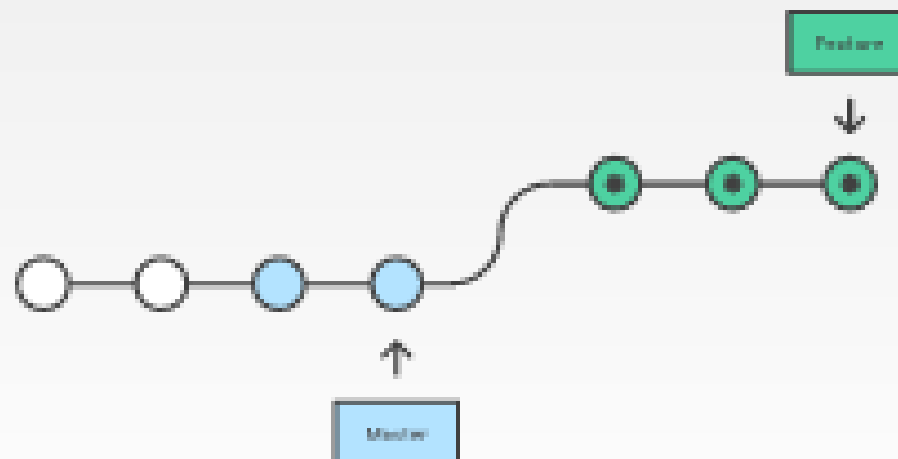


Git rebase

A forked commit history



Rebasing the feature branch onto master



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 (No late Submission!)

Friday March 15, 11:55 PM

Lab 9

- 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/local/cs/bin:$PATH`
- Make a directory 'gitroot' and get a copy of the Diffutils Git repository
 - `$ mkdir gitroot`
 - `$ cd gitroot`
 - `$ git clone <URL>`
 - Follow steps in lab and use `man git` to find commands

Hints

1. `git clone`
2. `git log`
3. `git tag`
4. `git show <hash>`
5. `git checkout v3.0 -b <branchname>`