Open Source Software Development

Version control - Git and GitHub

Thái Minh Tuấn - Email: minhtuan@ctu.edu.vn

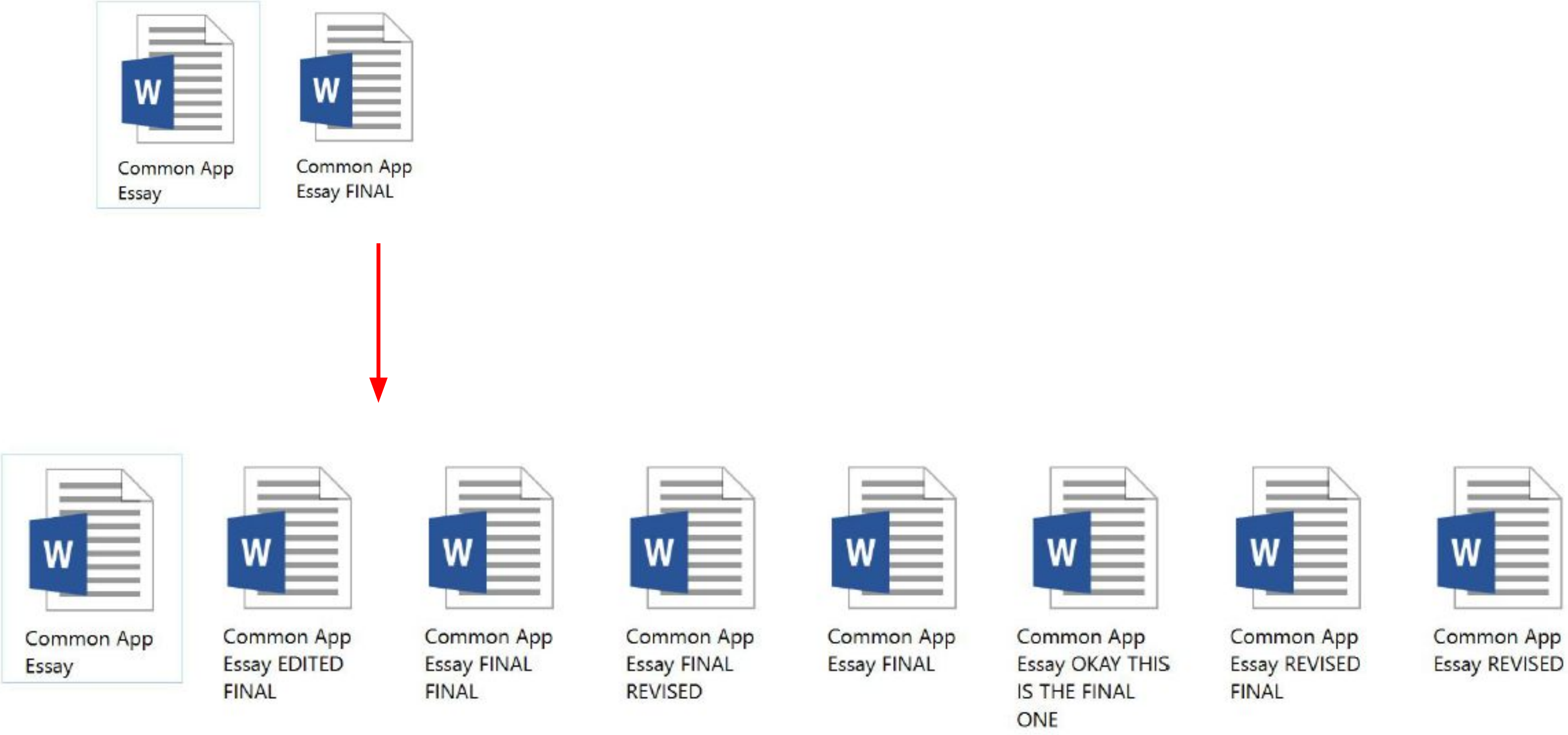
Learning Objectives

- What is version control?
- Why do we need a VCS?
- Centralized version control vs. Distributed version control
- Git: Concept, Terminology, Workflow, etc.
- GitHub

What is version control?

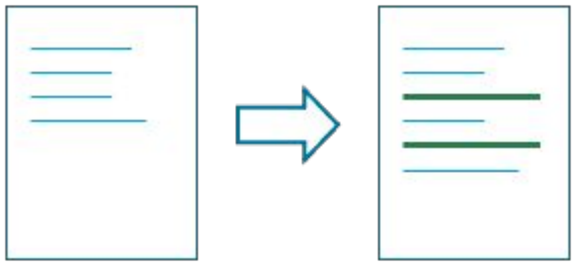
- Version control records changes to a set of files over time
  - This makes it easy to review or obtain a specific version (later)
- Simple example:
  - René writes a paper, using version control: **v1.0**
  - René corrects grammatical mistakes and typos: **v1.1**
  - René discovers new findings and rewrites the paper: **v1.2**
  - René realizes the new findings are wrong: restore **v1.1**
- Who uses version control?
  - Developers (obviously)
  - Researchers
  - Applications (e.g., (cloud-based) word processors)

Why use version control?


Common App Essay
Common App Essay FINAL
Common App Essay
Common App Essay EDITED FINAL
Common App Essay FINAL FINAL
Common App Essay FINAL REVISED
Common App Essay FINAL
Common App Essay OKAY THIS IS THE FINAL ONE
Common App Essay REVISED FINAL
Common App Essay REVISED

Why Do We Need a VCS?

VCSs Track File Changes



Code is organized within a repository.

- VCSs tell Us:
    - Who made the change?
        - So you know whom to blame
    - What has changed (added, removed, moved)?
        - Changes within a file
        - Addition, removal, or moving of files/directories
    - Where is the change applied?
        - Not just which file, but which version or branch
    - When was the change made?
        - Timestamp
    - Why was the change made?
        - Commit messages

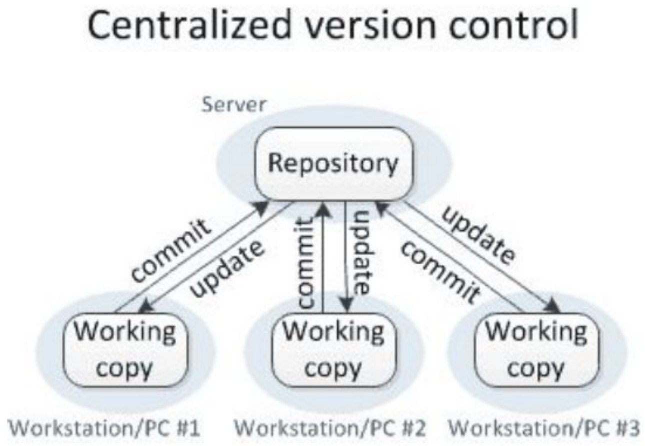
- Basically, the Five W’s

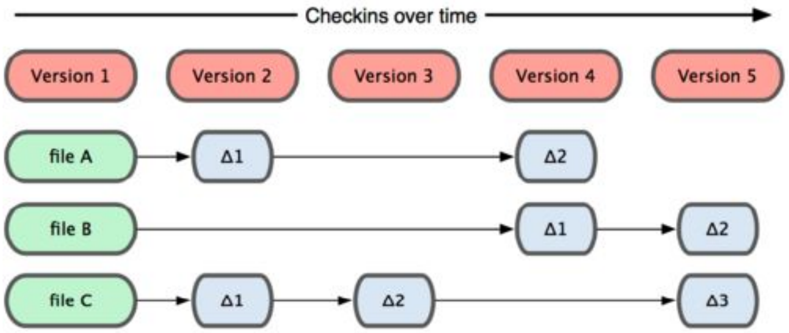
Brief History Of Version Control Software

- First Generation – Local Only
    - SCCS – 1972: Only option for a LONG time
    - RCS – 1982: For comparison with SCCS, see this 1992 forum link
- Second Generation – Centralized
    - CVS – 1986: Basically a front end for RCS
    - SVN – 2000: Tried to be a successor to CVS
    - Perforce – 1995: Proprietary, but very popular for a long time
    - Team Foundation Server – 2005:
        - Microsoft product, proprietary
        - Good Visual Studio integration
- Third Generation – Decentralized
    - BitKeeper – 2000
    - GNU Bazaar – 2005
    - Canonical/Ubuntu
    - Mercurial – 2005
    - Git – 2005
    - Team Foundation Server – 2013

Centralized version control

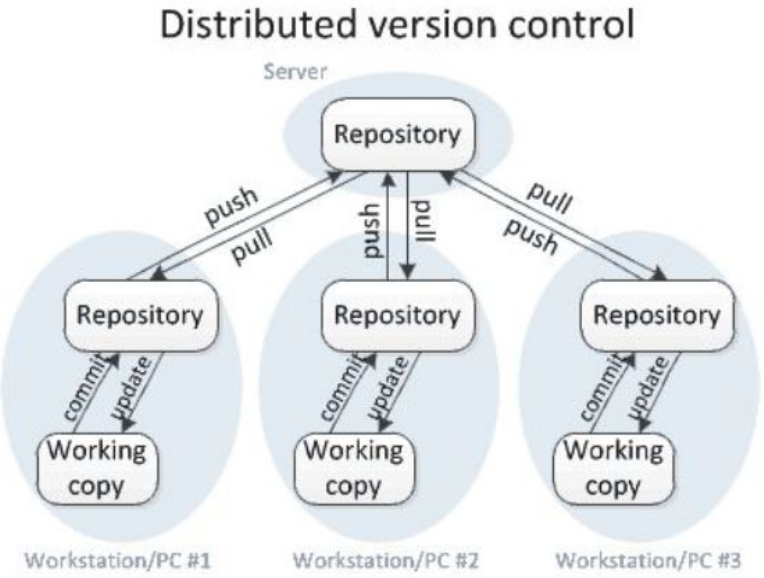
- **One central repository**
- All users commit their changes to the central repository.
    - Each user has a working copy
    - As soon as they commit, the repository gets updated
    - Track version data on each individual file
    - Examples: SVN (Subversion), CVS

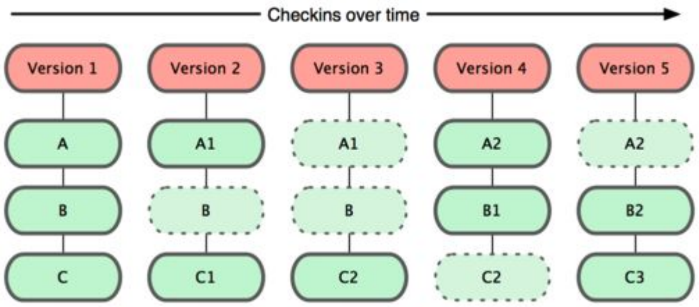




Distributed version control

- **Multiple clones of a repository**
- Each user commits to a local (private) repository
  - All committed changes remain local unless pushed to another repository
  - No external changes are visible unless pulled from another repository
  - Each check-in version of the overall code has a copy of each file in it.
    - Some files change on a given check-in, some do not
    - More redundancy, but faster.
  - Examples: Git, Hg (Mercurial)





About GIT

- Started by Linus Torvalds – 2005
  - Came out of Linux development community
  - Designed to do version control on Linux kernel
- Goals of Git:
  - Speed
  - Support for non-linear development
    - Thousands of parallel branches
    - Branching is cheap and easy!!!!
  - Fully distributed
  - Able to handle large projects efficiently
- Official Site
- Wikipedia
- Initial README commit

(A "git" is a cranky old man. Linus meant himself)

Installing/learning Git

- Git website: http://git-scm.com/
    - Free online book: http://git-scm.com/book
    - Reference page for Git: http://gitref.org/index.html
    - Git tutorial: http://schacon.github.com/git/gittutorial.html
    - Git for Computer Scientists:
        - http://eagain.net/articles/git-for-computer-scientists/
- At command line: (where `verb = config, add, commit`, etc.)

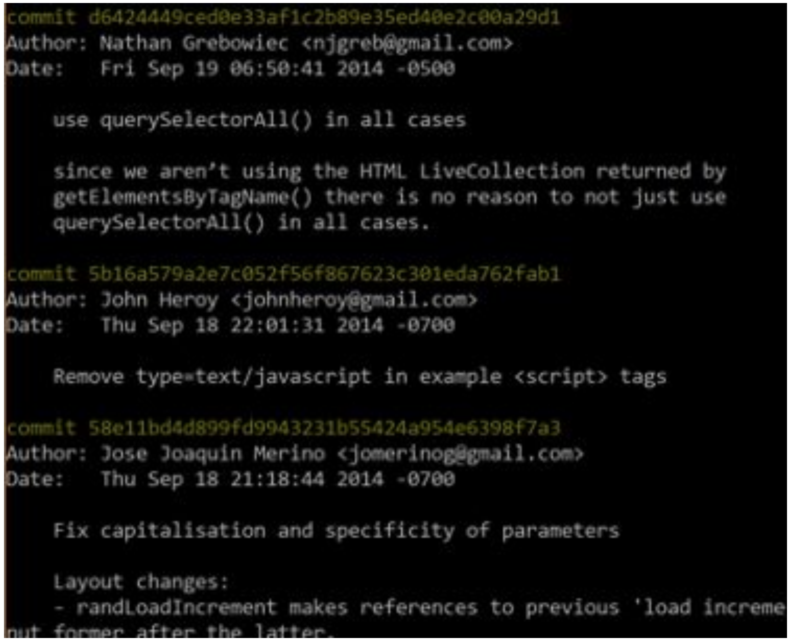
```
git help verb
```

Terminology

- Branch
  - A history of successive changes to code
  - A new branch may be created at any time, from any existing commit
  - May represent versions of code
    - Version 1.x, 2.x, 3.x, etc.
  - May Represent small bug-fixes/feature development
  - Branches are cheap
    - Fast switching
    - Easy to “merge” into other branches
    - Easy to create, easy to destroy
  - See this guide for best practices
- Commit
  - Set of changes to a repository’s files
  - More on this later
- Tag
  - Represents a single commit
  - Often human-friendly
    - Version Numbers
- A Repository may be created by:
  - Cloning an existing one (git clone)
  - Creating a new one locally (git init)

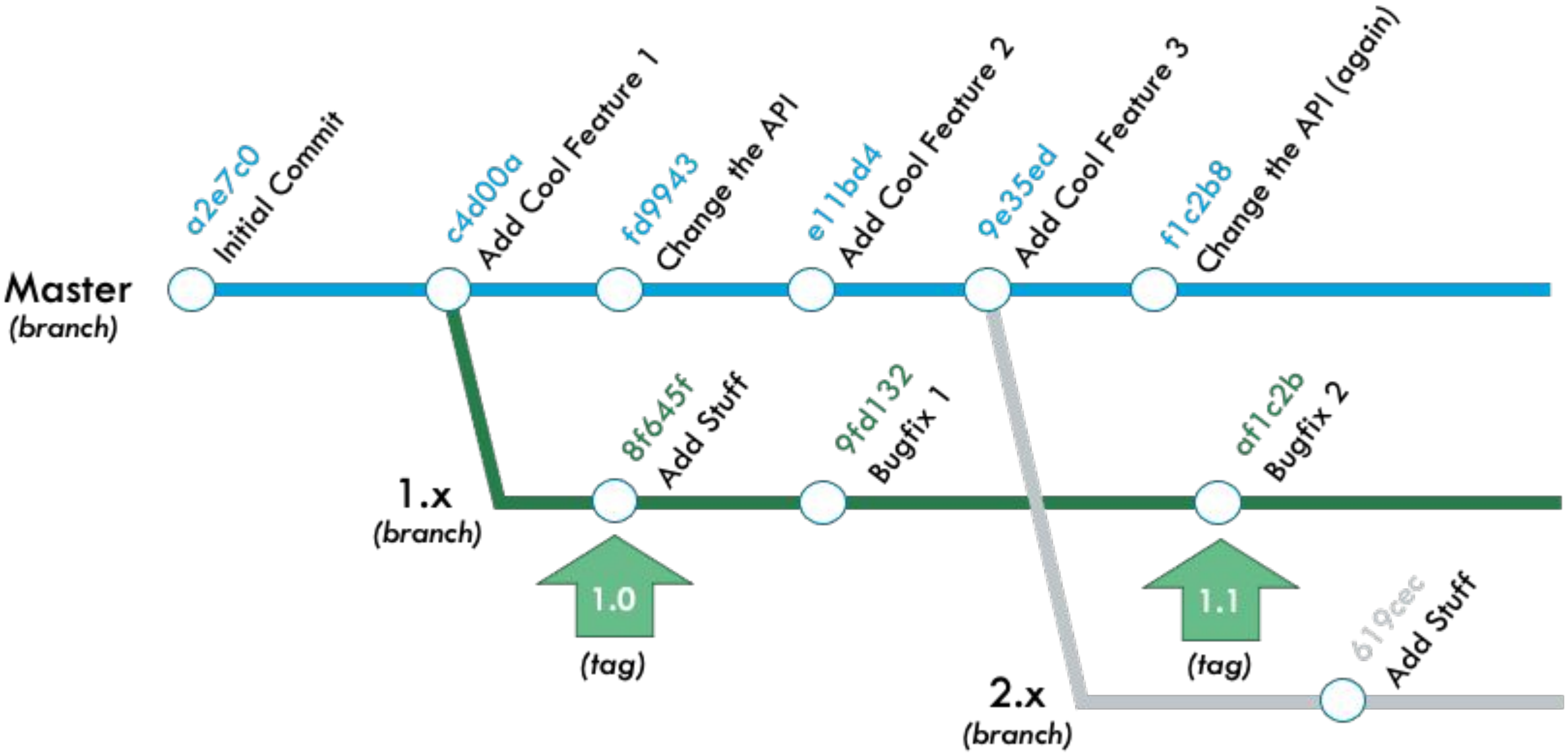
What is a Commit?

- Specific snapshot within the development tree
- Collection of changes applied to a project’s files
    - Text changes, file and directory addition/removal, chmod
    - Metadata about the change
- Identified by a SHA-1 Hash
    - Can be shortened to approx. 6 characters for CLI use
        - (e.g., “git show 5b16a5”)
    - HEAD – most recent commit
    - ORIG_HEAD – after a merge, the previous HEAD
    - <commit>~n – the nth commit before <commit>
    - e.g., 5b16a5~2 or HEAD~5
    - master@{01-Jan-2018} – last commit on master branch before January 1, 2018



```
commit d6424449ced0e33af1c2b89e35ed40e2c00a29d1
Author: Nathan Grebowiec <njgreb@gmail.com>
Date:   Fri Sep 19 06:50:41 2014 -0500

    use querySelectorAll() in all cases

    since we aren’t using the HTML LiveCollection returned by
    getElementsByTagName() there is no reason to not just use
    querySelectorAll() in all cases.

commit 5b16a579a2e7c052f56f867623c301eda762fab1
Author: John Heroy <johnheroy@gmail.com>
Date:   Thu Sep 18 22:01:31 2014 -0700

    Remove type=text/javascript in example <script> tags

commit 58e11bd4d899fd9943231b55424a954e6398f7a3
Author: Jose Joaquin Merino <jomerinog@gmail.com>
Date:   Thu Sep 18 21:18:44 2014 -0700

    Fix capitalisation and specificity of parameters

    Layout changes:
    - randLoadIncrement makes references to previous 'load increme
put former after the latter.
```

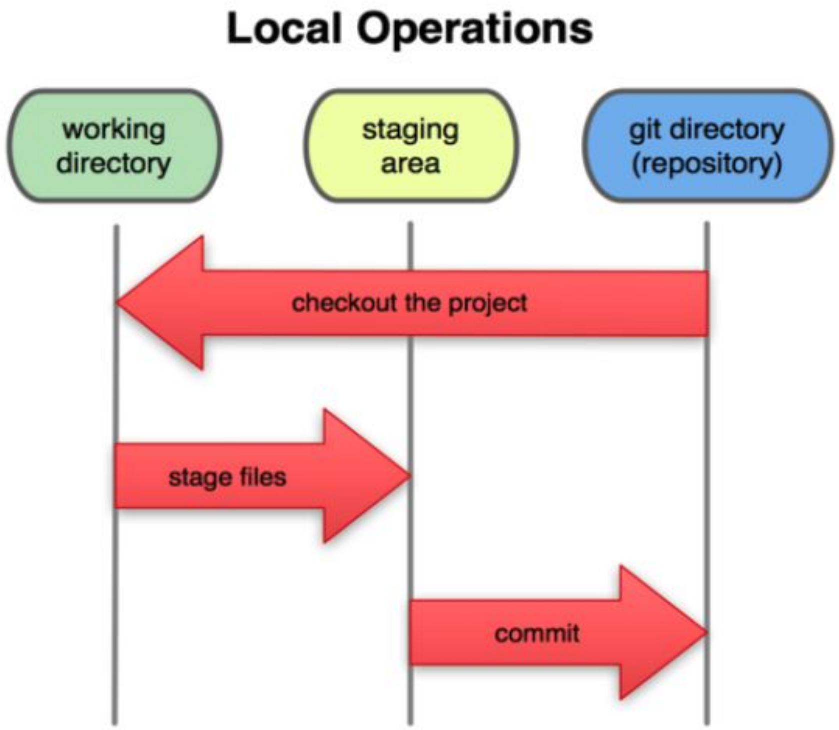
Branches, Commits, and Tags



Terminology

- Working Files
    - Files that are currently on your File System
- The Stage (also called the “index”)
    - Staging is the first step to creating a commit
    - The stage is what you use to tell Git which changes to include in the commit
    - File changes must be “added” to the stage explicitly
    - Only changes that have been staged will be committed
- Checkout
    - Replace the current working files with those from a specific branch or commit

Git Workflow - Local operations



Unmodified/modified Files

Staged Files

Committed Files

Initial Git configuration

- Set the name and email for Git to use when you commit:
  - `git config --global user.name "Bugs Bunny"`
  - `git config --global user.email bugs@gmail.com`
  - You can call `git config –list` to verify these are set.
- Set the editor that is used for writing commit messages:
  - `git config --global core.editor nano`
  - (it is vim by default)

Creating a Git repo

- Two common scenarios: (only do one of these)
- To **create a new local Git repo** in your current directory:
  - `git init`
    - This will create a .git directory in your current directory.
    - Then you can commit files in that directory into the repo
  - `git add filename`
  - `git commit -m "commit message"`
- To **clone a remote repo** to your current directory:
  - `git clone url localDirectoryName`
  - This will create the given local directory, containing a working copy of the files from the repo, and a .git directory (used to hold the staging area and your actual local repo)
- Fork (clone + metadata)
  - Create a completely independent copy of Git repository

Git commands

`git clone url [dir]`	copy a Git repository so you can add to it
`git add file`	adds file contents to the staging area
`git commit`	records a snapshot of the staging area
`git status`	view the status of your files in the working directory and staging area
`git diff`	shows diff of what is staged and what is modified but unstaged
`git help [command]`	get help info about a particular command
`git pull`	fetch from a remote repo and try to merge into the current branch
`git push`	push your new branches and data to a remote repository
Others: `init, reset, branch, checkout, merge, log, tag`	

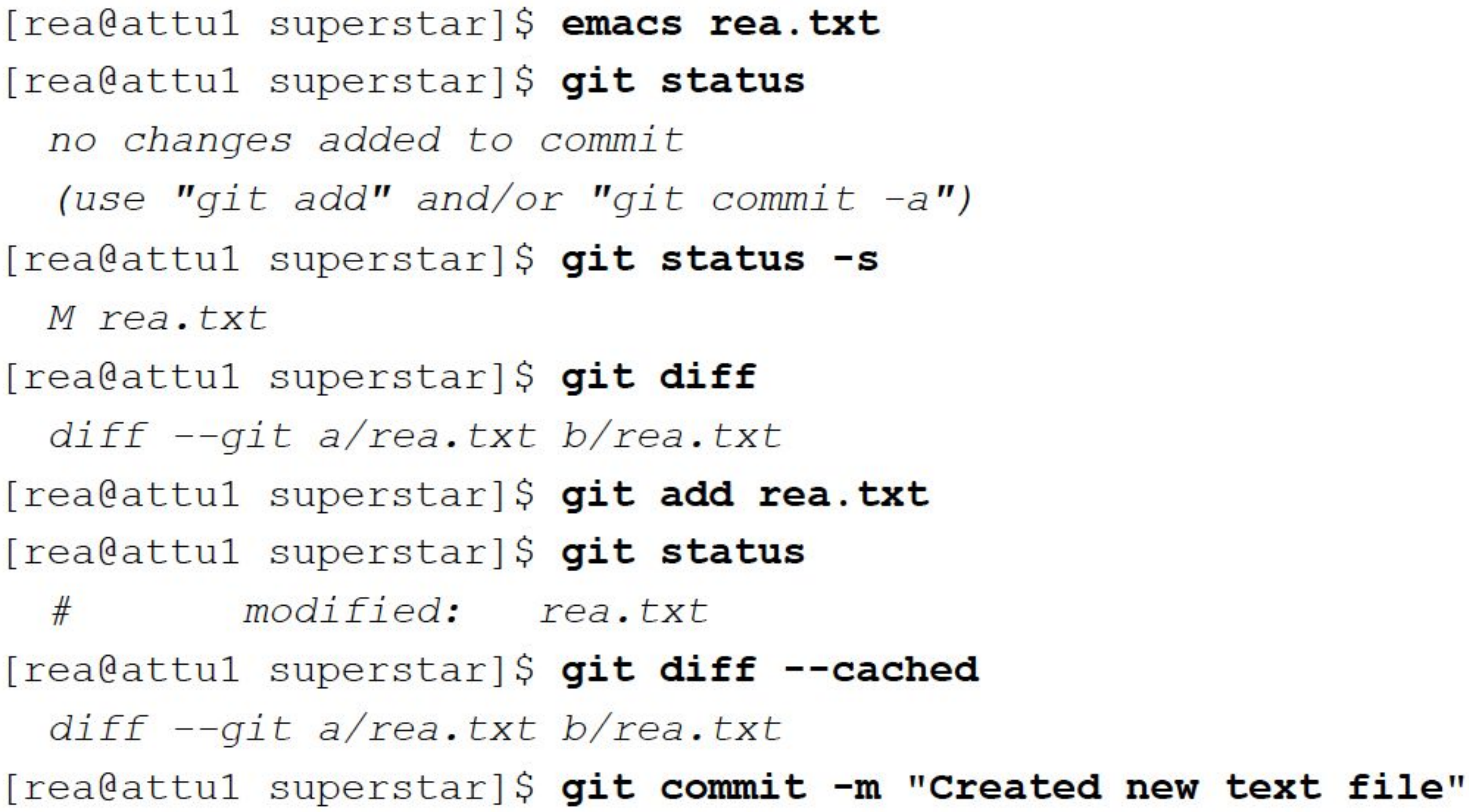
Add and commit a file

- The first time we ask a file to be tracked, and *every time before we commit a file*, we must add it to the staging area:
  - `git add Hello.java Goodbye.java`
  - Takes a snapshot of these files, adds them to the staging area.
  - In older VCS, "add" means "start tracking this file." In Git, "add" means "add to staging area" so it will be part of the next commit.
- To move staged changes into the repo, we commit:
  - `git commit -m "Fixing bug #22"`
- To undo changes on a file before you have committed it:
  - `git reset HEAD -- filename` #(unstages the file)
  - `git checkout -- filename` #(undoes your changes)
- All these commands are acting on your local version of repo

Viewing/undoing changes

- To view status of files in working directory and staging area:
  - `git status` or `git status -s` (short version)
- To see what is modified but unstaged:
  - `git diff`
- To see a list of staged changes:
  - `git diff --cached`
- To see a log of all changes in your local repo:
  - `git log` or `git log --oneline` (shorter version)
  - `git log -5` (to show only the 5 most recent updates), etc.

An example workflow



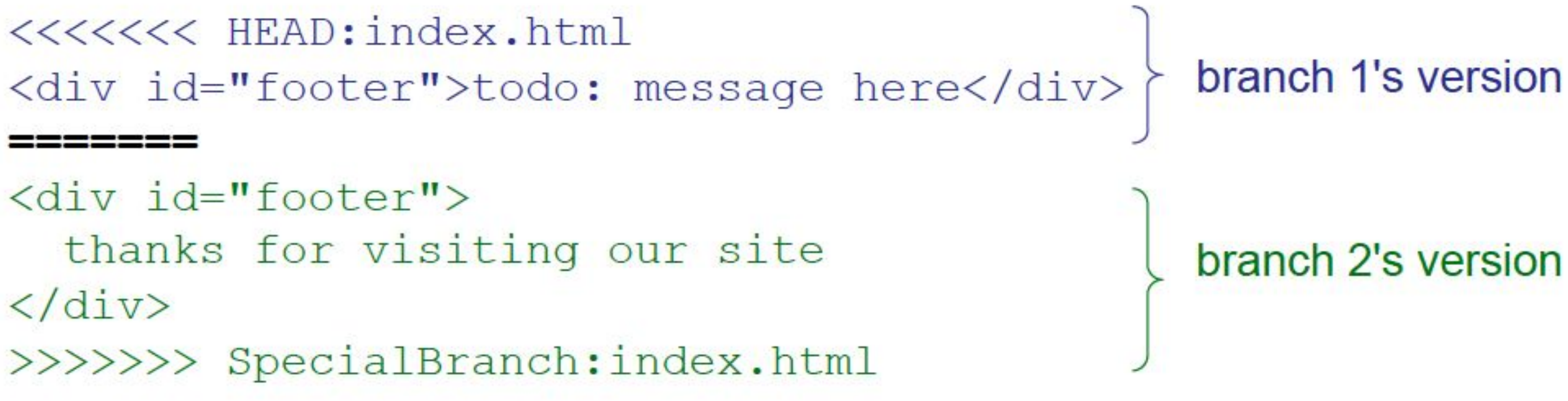
```
[rea@attu1 superstar]$ emacs rea.txt
[rea@attu1 superstar]$ git status
  no changes added to commit
   (use "git add" and/or "git commit -a")
[rea@attu1 superstar]$ git status -s
  M rea.txt
[rea@attu1 superstar]$ git diff
  diff --git a/rea.txt b/rea.txt
[rea@attu1 superstar]$ git add rea.txt
[rea@attu1 superstar]$ git status
  #       modified:   rea.txt
[rea@attu1 superstar]$ git diff --cached
  diff --git a/rea.txt b/rea.txt
[rea@attu1 superstar]$ git commit -m "Created new text file"
```

Branching and merging

- Git uses branching heavily to switch between multiple tasks
- To create a new local branch:
  - `git branch name`
- To list all local branches: (* = current branch)
  - `git branch`
- To switch to a given local branch:
  - `git checkout branchname`
- To merge changes from a branch into the local master:
  - `git checkout master`
  - `git merge branchname`

Merge conflicts

- 2 branches modify the same place in the same file
- The conflicting file will contain <<< and >>> sections to indicate where Git was unable to resolve a conflict:



```
<<<<<<< HEAD:index.html
<div id="footer">todo: message here</div>
=======
<div id="footer">
  thanks for visiting our site
</div>
>>>>>>> SpecialBranch:index.html
```

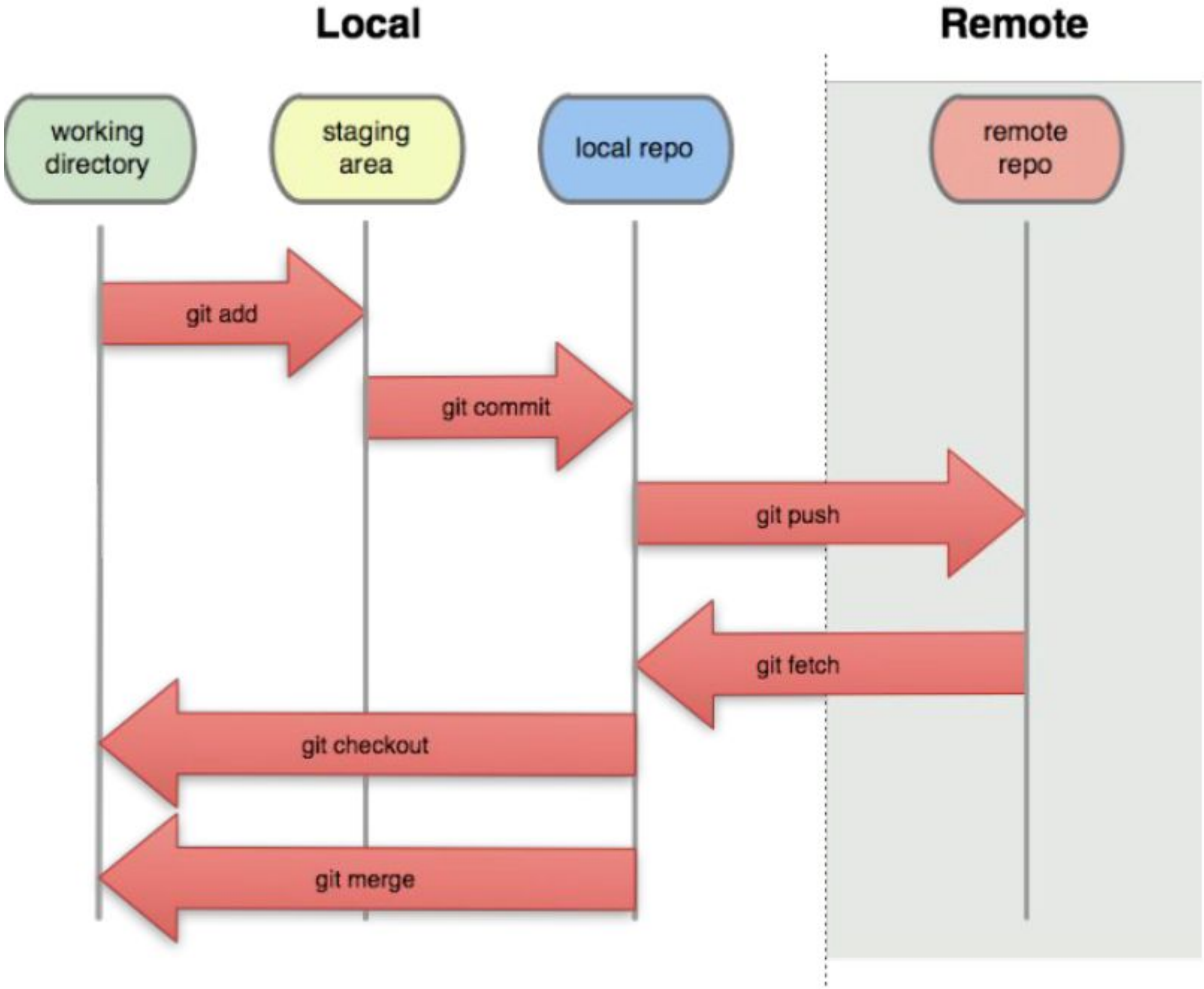
- Find all such sections, and edit them to the proper state (whichever of the two versions is newer / better / more correct)
  - Then, `add` and `commit`

Interaction w/ remote repo

- **Push** your local changes to the remote repo
- **Pull** from remote repo to get most recent changes
  - (fix conflicts if necessary, add/commit them to your local repo)

- To fetch the most recent updates from the remote repo into your local repo, and put them into your working directory:
  - `git pull origin master`
- To put your changes from your local repo in the remote repo:
  - `git push origin master`

Git Workflow



Common Industry Programmer Workflow

1. Choose a bug/feature to work on
2. Checkout the appropriate branch
3. Pull the latest additions from the main repository
4. Create a branch for the bug/feature request
5. Do the coding necessary & commit
6. Push the changes to your remote
7. Create a pull request to main repo

The pull request is where other programmers review your code & make comments/suggestions

If changes are needed to your code, then repeat the process.

Never commit directly to the main repo

Practical Guidelines - DO's

- Make small, incremental commits (within reason) are good
  Avoid monolithic commits at all cost
- Use a separate branch for each new bug/feature request
- Write nice commit messages
  - Otherwise, the commit log is useless
- Use a .gitignore to keep cruft out of your repo
- Search engines are your friend
  - Someone else has had the same question/mistake/situation as you, and they have already asked the question online. It's probably on StackOverflow

Practical Guidelines - Dont's

- Do not commit commented-out debug code.
  - It’s messy. It’s ugly. It’s unprofessional.
- Do not mix your commits.
  - E.g., Don’t commit two bug-fixes at the same time
- Do not commit sensitive information
  - Passwords, database dumps, etc.)
- Do not commit whitespace differences, unless it is specifically needed
- Do not commit large binaries

Practical Guidelines - Dont's

- Pro Git – free Apress ebook
- Visualizing Git histories
- Wikipedia on Version Control, with definitions
- Git Cheat Sheet
- Git Commit Etiquette for Junior Devs
- https://www.codeschool.com/learn/git
- Free course to walk you through basic Git concepts

GitHub (1/2)

- Git began as an offshoot of the Linux kernel development community
  - Initially created by Linus Torvalds himself
- However, it was quickly realized it could be used for any project that had similar needs:
  - A large group of contributors
  - A widely dispersed community of contributors
  - A very open development method with frequent releases
- Use of Git grew explosively after the founding of GitHub in 2008
  - 3 years after the creation of Git.
- GitHub.com is a site for online storage of Git repositories
  - You can create a remote repo there and push code to it
  - Many open source projects use it

GitHub (2/2)

- Before GitHub projects needed to have their own servers to host repositories
  - Need good amount of knowledge to setup, administer and secure and protect the integrity of repositories.
  - It is now pretty easy to get a project rolling quickly.
- In 2018, GitHub was acquired by Microsoft. In early 2020, GitHub had over 40 million users and 100 million repositories!
- Question: Do I always have to use GitHub to use Git?
  - No! You can use Git locally for your own purposes.
  - Or you or someone else could set up a server to share files
  - There are other sites that offer similar services, including:
    - GitLab, GitKraken, Launchpad

Public vs Private Repositories

- Private Repositories
    - Only selected collaborators can see the repository, or clone it (make a local copy), or download its contents in a variety of forms
    - The owner must specifically authorize each collaborator
- Public Repositories
    - Anyone given the proper link, can copy, clone or fork the repository, or download its contents
    - However, unless the owner authorizes them as a collaborator, one does not have permission to upload or make modifications