

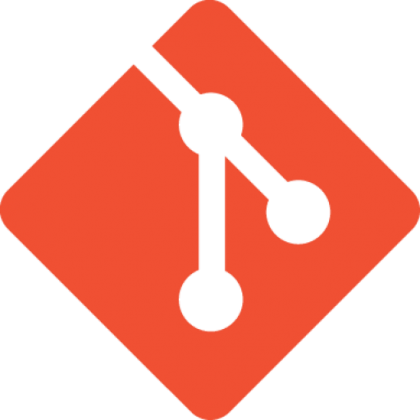
**Git for Version Control**

These slides are heavily based on slides created by Ruth Anderson for CSE 390a. Thanks, Ruth!

images taken from http://git-scm.com/book/en/

http://www.cs.washington.edu/403/

# About Git

* Created by Linus Torvalds, creator of Linux, in 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)

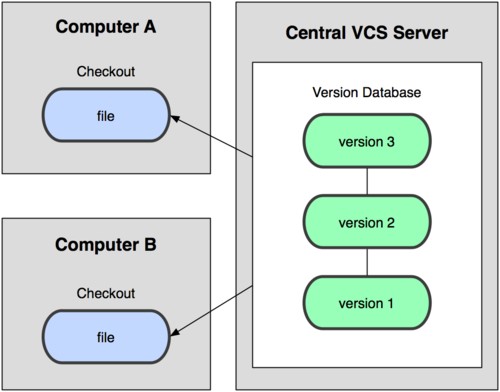
* + Fully distributed
  + Able to handle large projects efficiently
  + (A "git" is a cranky old man. Linus meant himself.)

# Installing/learning Git

* Git website: http://git-scm.com/
  + Free on-line 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*

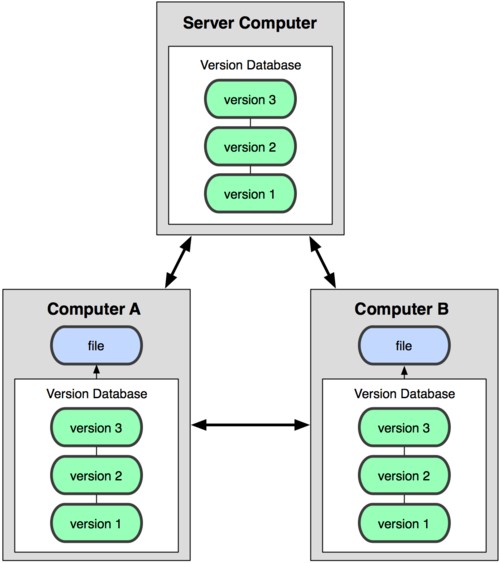
# Centralized VCS

* In Subversion, CVS, Perforce, etc. A central server repository (repo) holds the "official copy" of the code
  + the server maintains the sole version history of the repo
* You make "checkouts" of it to your local copy
  + you make local modifications
  + your changes are not versioned
* When you're done, you

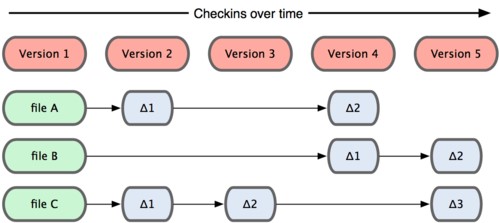
"check in" back to the server

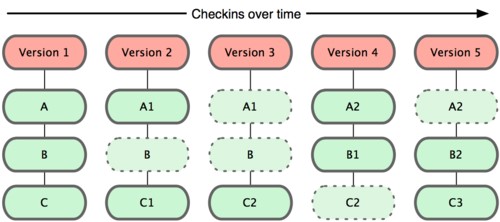
* + your checkin increments the repo's version

# Distributed VCS (Git)

* In git, mercurial, etc., you don't "checkout" from a central repo
  + you "clone" it and "pull" changes from it
* Your local repo is a complete copy of everything on the remote server
  + yours is "just as good" as theirs
* Many operations are local:
  + check in/out from local repo
  + commit changes to local repo
  + local repo keeps version history
* When you're ready, you can "push" changes back to server

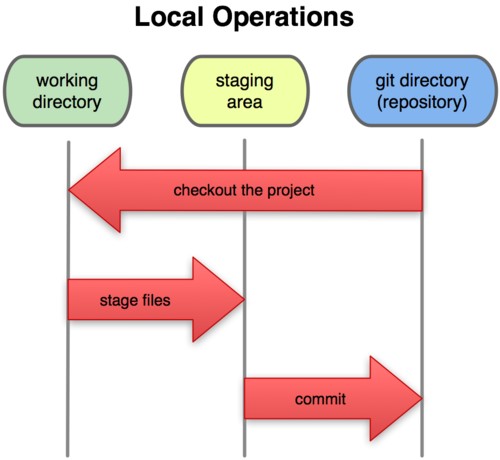
# Git snapshots

* Centralized VCS like Subversion Subversion track version data on each individual file.
* Git keeps "snapshots" of the entire state of the project.
  + Each checkin version of the

overall code has a copy of Git each file in it.

* + Some files change on a given checkin, some do not.
  + More redundancy, but faster.

# Local git areas

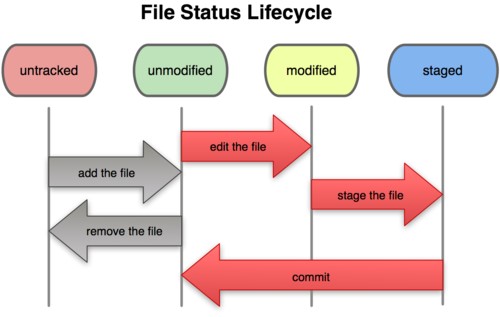
* In your local copy on git, files can be:

–  In your local repo

* (committed)
* Checked out and modified, but not yet committed
  + (working copy)
* Or, in-between, in a **"staging" area** Unmodified/modified Staged Committed
  + Staged files are ready Files Files Files to be committed.
  + A commit saves a snapshot of all staged state.

# Basic Git workflow

* **Modify** files in your working directory.
* **Stage** files, adding snapshots of them to your staging area.
* **Commit**, which takes the files in the staging area and stores that snapshot permanently to your Git directory.



# Git commit checksums

* In Subversion each modification to the central repo increments the version # of the overall repo.
  + In Git, each user has their own copy of the repo, and commits changes to their local copy of the repo before pushing to the central server.
  + So Git generates a unique **SHA-1 hash** (40 character string of hex digits) for every commit.
  + Refers to commits by this ID rather than a version number.
  + Often we only see the first 7 characters:
* 1677b2d Edited first line of readme
* 258efa7 Added line to readme
* 0e52da7 Initial commit

# 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)

# Git commands

|  |  |
| --- | --- |
| **command** | **description** |
| 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)

1677b2d Edited first line of readme

258efa7 Added line to readme

0e52da7 Initial commit

* 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

* 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> branch 1's version

**=======**

<div id="footer">

thanks for visiting our site branch 2's version

</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).

# 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

# GitHub

* 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, such as the Linux kernel.
  + You can get free space for open source projects, or you can pay for private projects.
* Free private repos for educational use: github.com/edu
* Question: Do I always have to use GitHub to use Git?
* Answer: No! You can use Git locally for your own purposes.
* Or you or someone else could set up a server to share files.
* Or you could share a repo with users on the same file system, as long everyone has the needed file permissions).