Git for Version Control

About Git

- Create by Linux Torvards, 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

Installing/learning Git

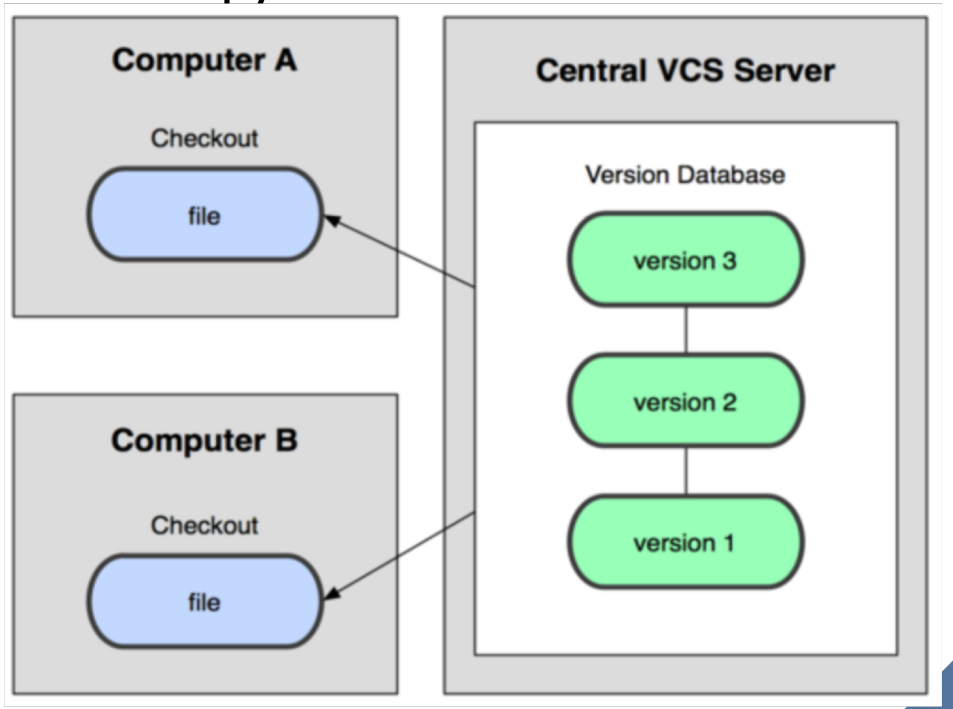
- Git website: http://git-scm.com/
  - Free on-line book: http://git-scm.com/book
  - Reference page for Git: http://gitref.org/index.html
  - Git tutorial: http://schacon.github.com/git/gittutorial.html
  - Git for Computer Scientists:
    - http://eagain.net/articles/git-for-computer-scientists/

- At command line: (where verb = config, add, commit, etc.)
  - git help *verb*

Centralized VCS

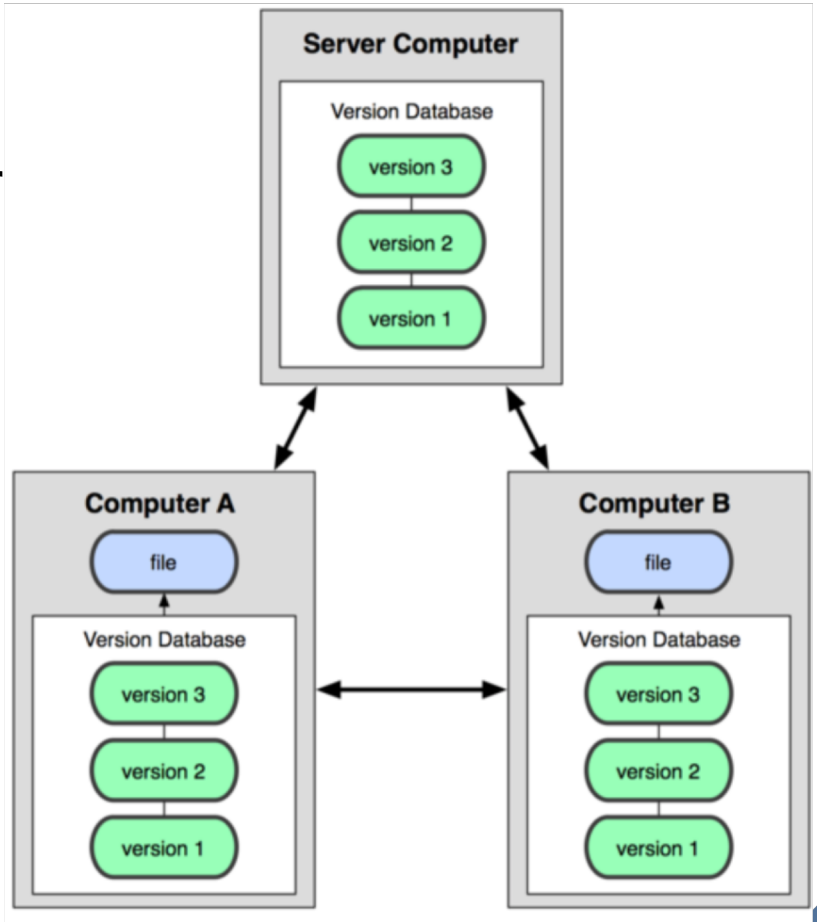
- In Subversion, CVS, Perforce, etc. A central server repository (repo) holds the "official copy" of the code
  - the server maintains the sole version history of the repo
- You make "checkouts" of it to your local copy
  - you make local modifications
  - your changes are not versioned
- When you're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

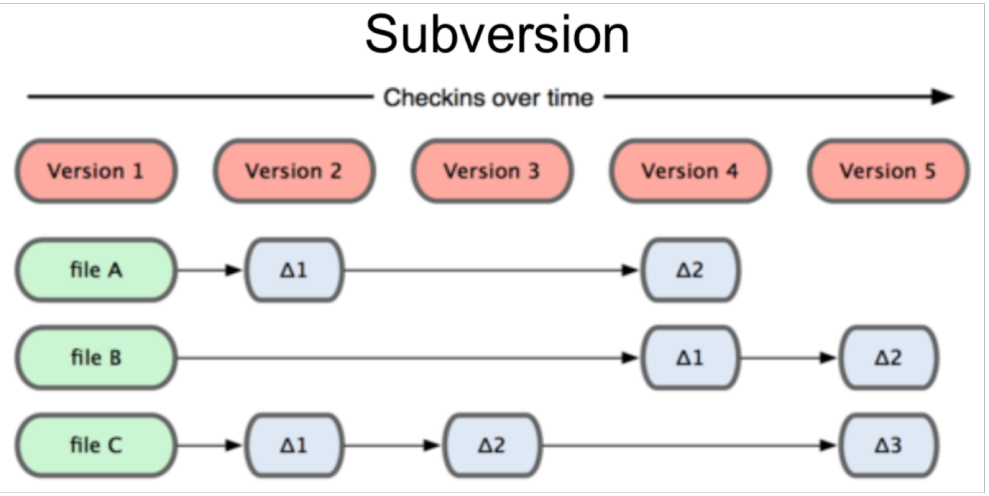


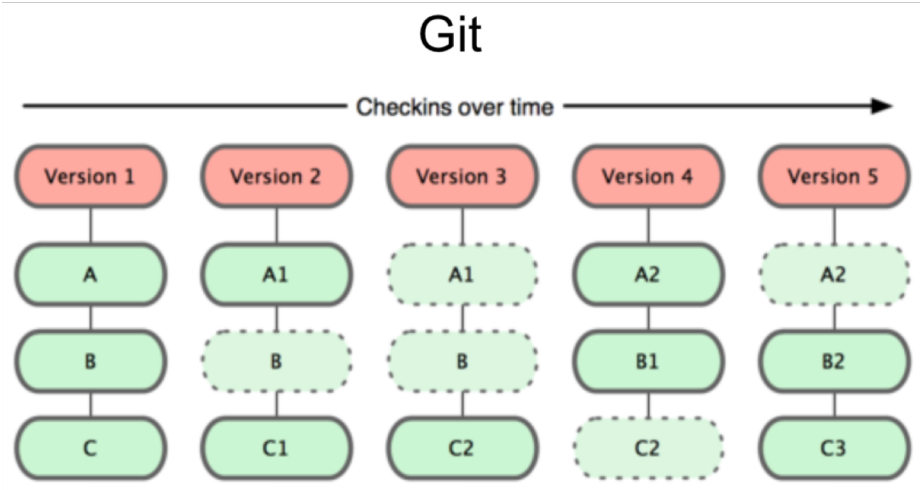
- When you're ready, you can "push" changes back to server

Git snapshots

- Centralized VCS like Subversion track version data on each individual file.

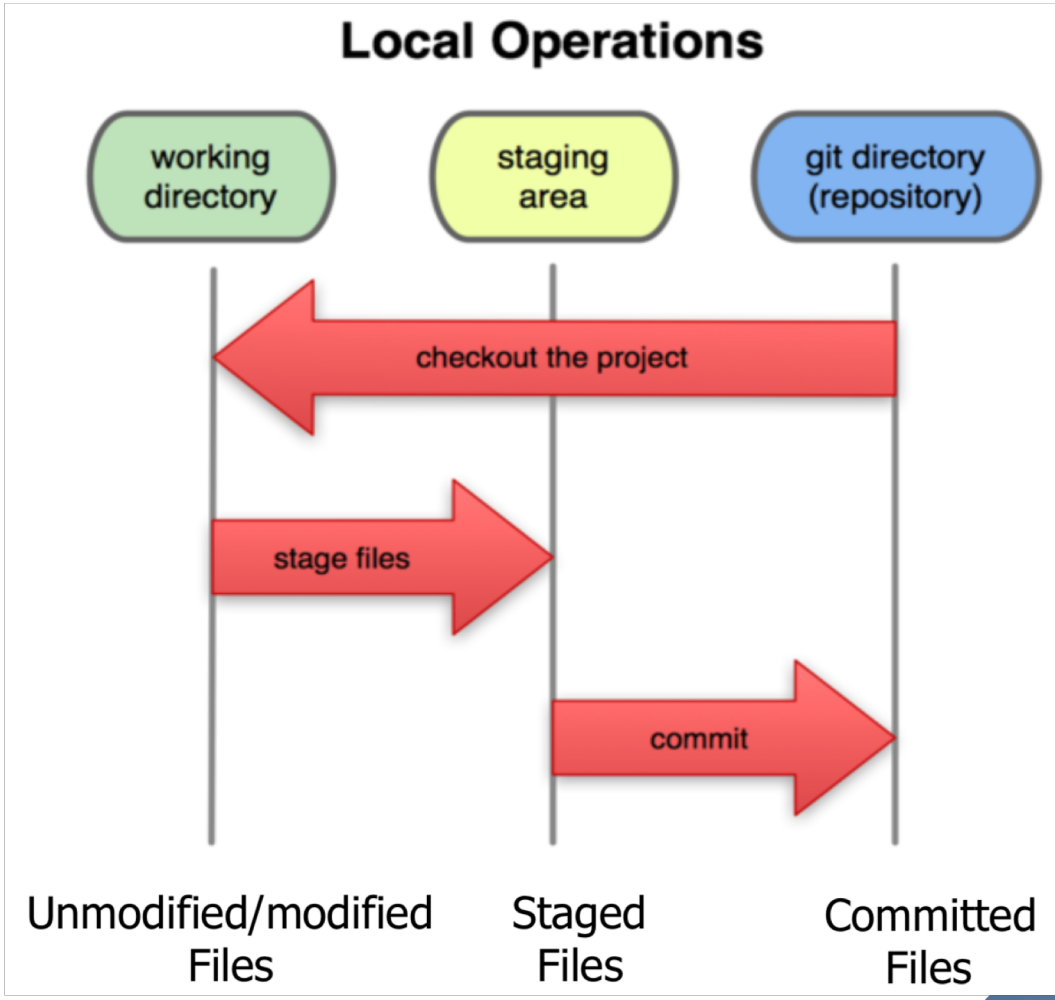
- Git keeps "snapshots" of the entire state of the project.
  - Each checkin version of the overall code has a copy of each file in it.
  - Some files change on a given checkin, some do not.
  - More redundancy, but faster.





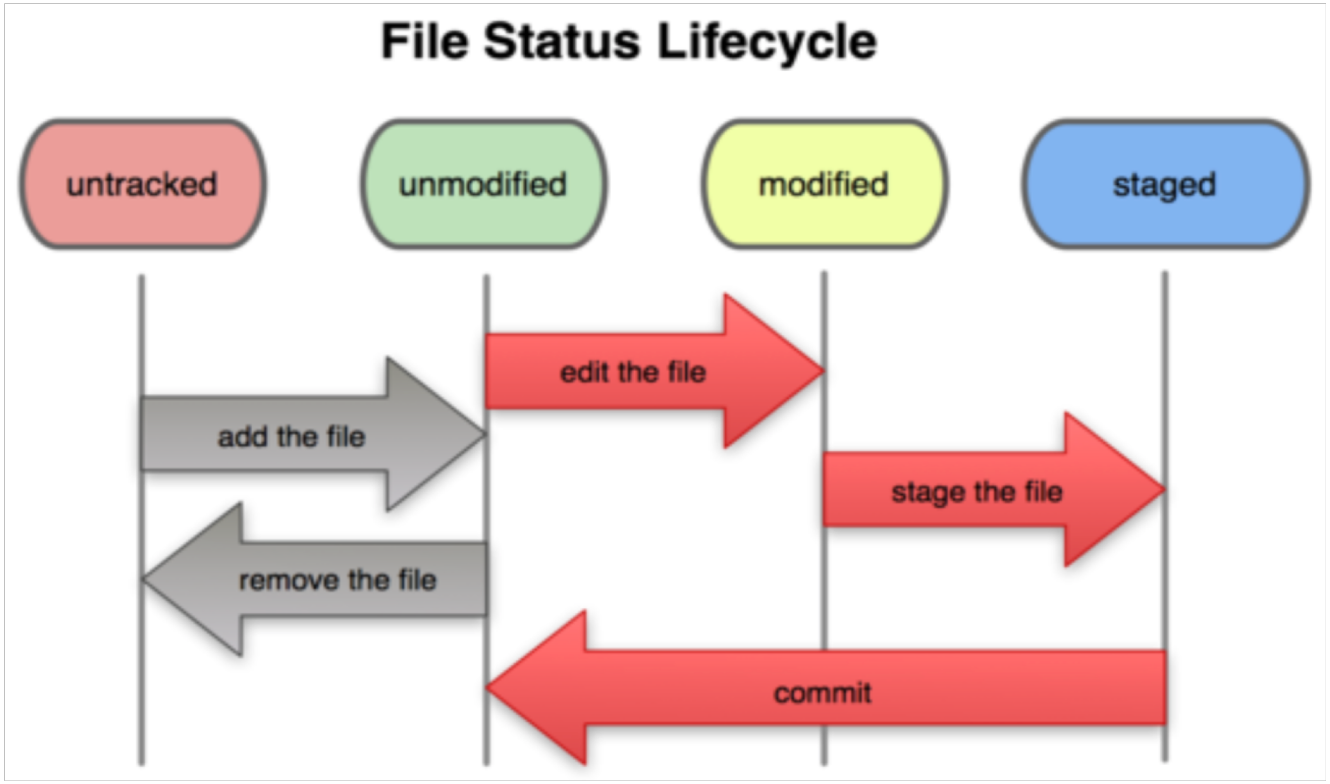
Local git areas

- 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
    - Staged files are ready
    - 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

```
[jerry@localhost project]$ git log
commit 3b92fd8883585585a4eb55a3313da284c8931c2e
Author: Tom <tom@disney.com>
Date:   Sun Dec 9 07:58:01 2018 -0500

    Added about.html

commit 23b6e76ed2e1086779c8ed447c1f485186873916
Author: Jerry <jerry@disney.com>
Date:   Sun Dec 9 07:44:34 2018 -0500

    Added index.html

commit 5eaba2da3292086d005f2f1fed5e37017d1da544
Author: Tom <tom@disney.com>
Date:   Sun Dec 9 07:14:32 2018 -0500

    Initital project
```

Git installation

- [CentOS ~]# yum -y install git-core
- [ubuntu ~]$ sudo apt-get install git-core

```
[jerry@localhost project]$ git --version
git version 1.8.3.1
```

Initial Git configuration

- Set the name and email for Git to use when you commit:
  - git config --global user.name "Jerry"
  - git config --global user.email jerry@disney.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)
  - git config --global merge.tool vimdiff

- Color highlighting
  - git config --global color.ui true
  - git config --global color.status auto
  - git config --global color.branch auto

```
[jerry@localhost project]$ git config --list
user.name=Jerry
user.email=jerry@disney.com
color.ui=true
color.status=auto
color.branch=auto
core.editor=vim
merge.tool=vimdiff
branch.autosetuprebase=always
core.repositoryformatversion=0
core.filemode=true
core.bare=false
core.logallrefupdates=true
remote.origin.url=gituser@192.168.100.1:project.git
remote.origin.fetch=+refs/heads/*:refs/remotes/origin/*
branch.master.remote=origin
branch.master.merge=refs/heads/master
branch.master.rebase=true
branch.version-1.0.remote=origin
branch.version-1.0.merge=refs/heads/version-1.0
branch.version-1.0.rebase=true
```

Create a Bare Repository on server

- Create New User
  - [root@CentOS ~]# groupadd dev
  - [root@CentOS ~]# useradd -G devs -d /home/gituser -m -s /bin/bash gituser
  - [root@CentOS ~]# passwd gituser

- Initialize a new repository by using **init** command followed by **--bare** option. It initializes the repository without a working directory. By convention, the bare repository must be named as **.git**.
  - [gituser@CentOS ~]$ pwd

```
/home/gituser
```

  - [gituser@CentOS ~]$ mkdir project.git
  - [gituser@CentOS ~]$ cd project.git/
  - [gituser@CentOS project.git]$ ls
  - [gituser@CentOS project.git]$ git --bare init

```
Initialized empty Git repository in /home/gituser-
m/project.git/
```

  - [gituser@CentOS project.git]$ ls

```
branches config description HEAD hooks info objects refs
```

Setup SSH

- Generate Public/Private RSA Key Pair
  - [tom@CentOS ~]$ pwd

```
/home/tom
```

  - [tom@CentOS ~]$ ssh-keygen

  - [jerry@CentOS ~]$ pwd

```
/home/jerry
```

  - [tom@CentOS ~]$ ssh-keygen

- Adding Keys to authorized_keys
  - [tom@CentOS ~]$ pwd

```
/home/tom
```

  - [tom@CentOS ~]$ ssh-copy-id -i ~/.ssh/id_rsa.pub gituser@git.server.com

```
gituser@git.server.com's password: Now try logging into the
machine, with "ssh 'gituser@git.server.com'", and check in:
.ssh/authorized_keys to make sure we haven't added extra
keys that you weren't expecting.
```

Create a local Git repo

- [tom@CentOS ~]$ pwd /home/tom
- [tom@CentOS ~]$ mkdir tom_repo
- [tom@CentOS ~]$ cd tom_repo/
- [tom@CentOS tom_repo]$ git init

```
Initialized empty Git repository in
/home/tom/tom_repo/.git/
```

- [tom@CentOS tom_repo]$ echo 'TODO: Add contents for README' > README
- [tom@CentOS tom_repo]$ git status -s

```
?? README
```

- [tom@CentOS tom_repo]$ git add .
- [tom@CentOS tom_repo]$ git status -s

```
A README
```

- [tom@CentOS tom_repo]$ git commit -m 'Initial repo'

Push changes to the server repo

- [tom@CentOS tom_repo]$ git remote add origin gituser@git.server.com:project.git
- [tom@CentOS tom_repo]$ git push origin master

```
Counting objects: 3, done.
Writing objects: 100% (3/3), 242 bytes, done.
Total 3 (delta 0), reused 0 (delta 0)
To gituser@git.server.com:project.git
* [new branch]
master -> master
```

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.

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

branch 1's version

branch 2's version

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

Interaction with 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

More

- Stash operation
  - git stash
  - git stash list
  - git stash pop
- Cherry pick
- Review change
  - git log

Online repo

- Github
- Bitbucket