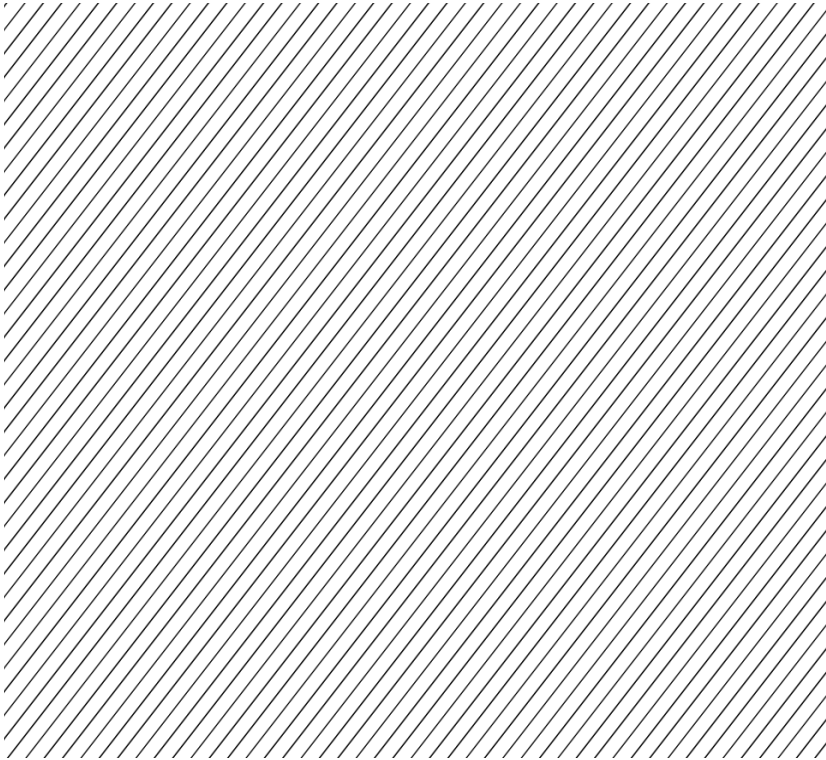
Cognizant
Softvision

Version control systems

Contents

Version Control systems (VCS)



Cognizant Softvision

Version control systems

Version control (or revision control, or source control) is a system that records changes to a file or set of files over time so that you can recall specific versions later.

- it’s all about managing multiple versions of documents, programs, web sites, etc.

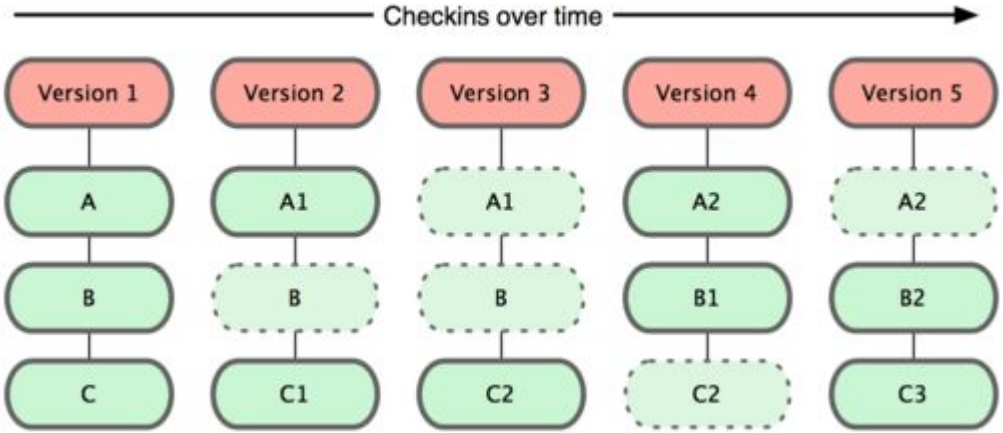
Essential for team projects, but also very useful for individual projects

Other VCS: CVS, Subversion, Mercurial, Perforce, Bazaar, and so on

Version control systems - differences

(Most) other systems store information as a list of file-based changes - set of files and the changes made to each file over time

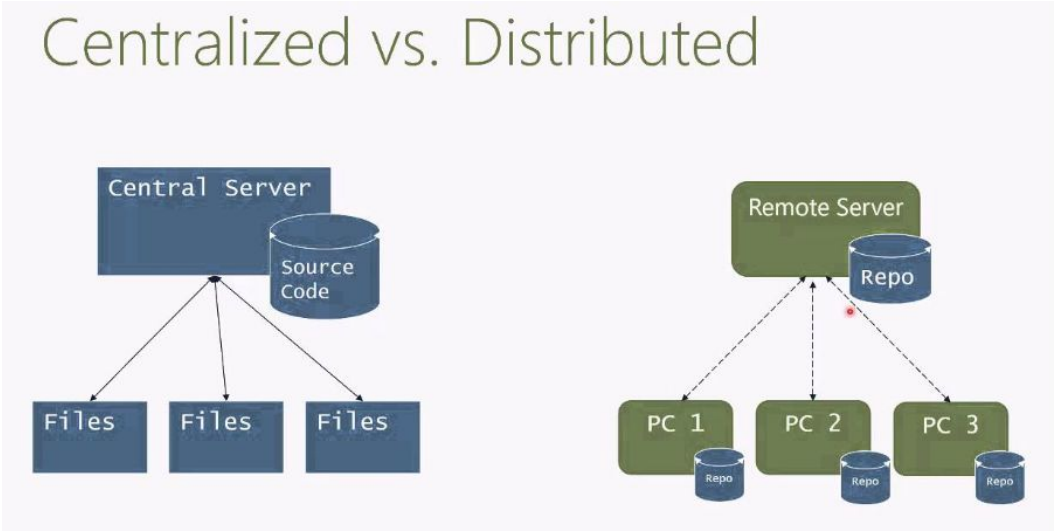
Git - set of snapshots of a mini filesystem



Centralized vs Distributed

Centralized Version Control

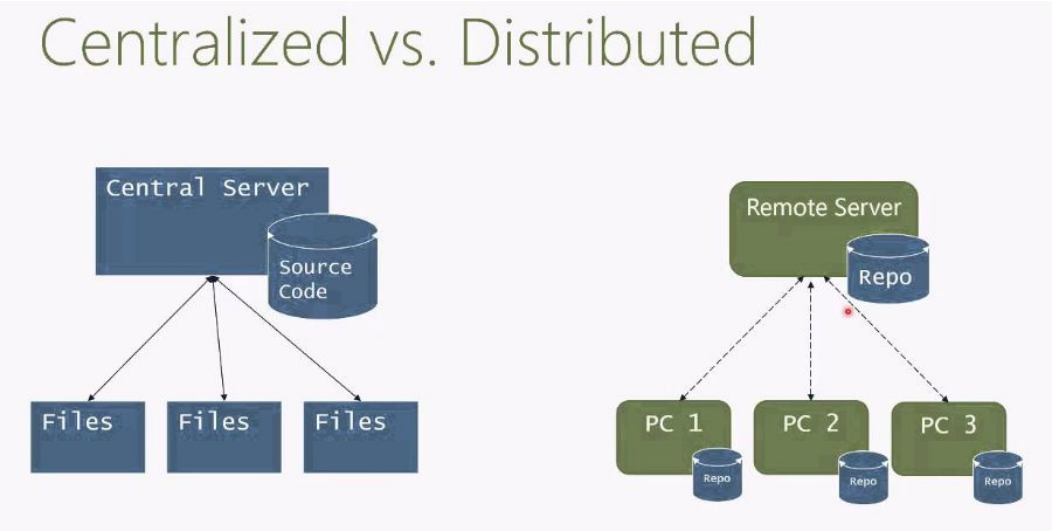
- works on a client-server model
- a single, (centralized) master copy of the code base, and pieces of the code that are being worked on are typically locked, (or “checked out”)
- only one developer is allowed to work on that part of the code at any one time.



Centralized vs Distributed

Distributed Version Control

- peer-to-peer model: the code base is distributed amongst the individual developers' computers
- no locking of parts of the code
- developers make changes in their local copy and then issue a request to merge their changes into the master copy



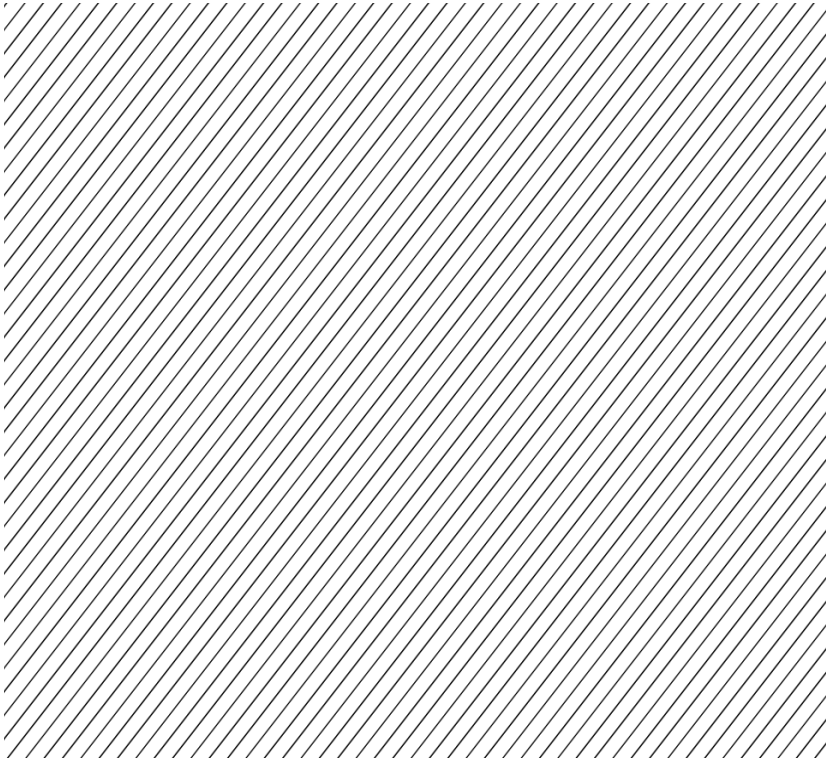
Cognizant
Softvision

GIT

Distributed version control system

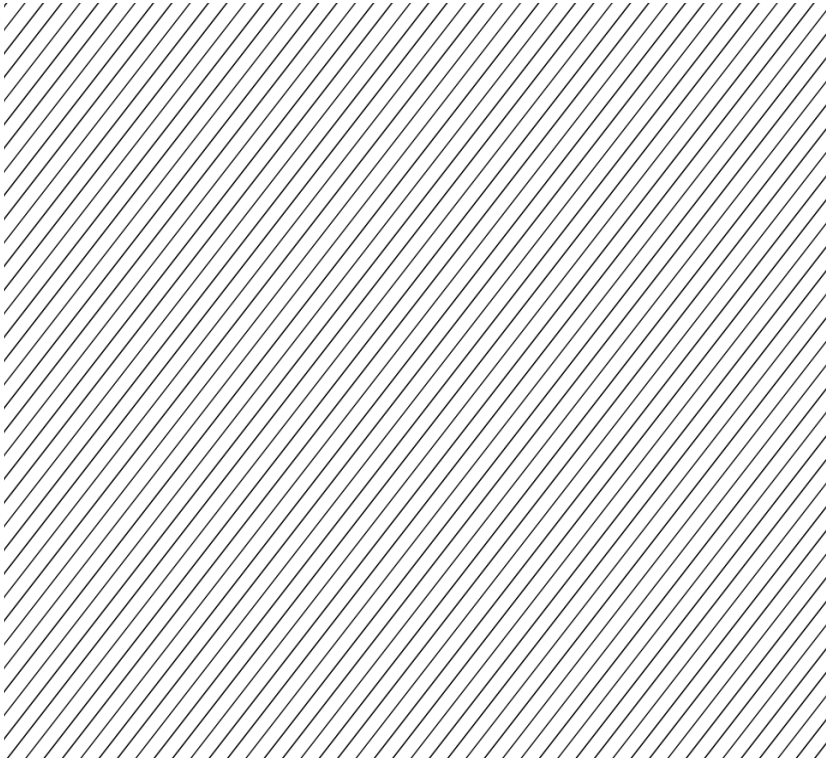
Contents

GIT - Distributed Version Control System



Contents

GIT - Distributed Version Control System



Why git ?

- **Save Time**: speed
- **Work Offline**: most operations are possible simply on your local machine: make a commit, browse your project's complete history, merge or create branches...
- **Undo Mistakes**: there's a little "undo" command for almost every situation. Correct your last commit because you forgot to include that small change. Revert a whole commit because that feature isn't necessary, anymore.
- **Don't Worry**: It is very difficult to get the system to do anything that is not undoable or to make it erase data in any way. Also every clone of a project that one of your teammates might have on his local computer is a fully usable backup

Why  ?

- **Make Useful Commits**: helps you create granular commits with its unique "staging area" concept: you can determine exactly which changes shall be included in your next commits, even down to single lines.
- **Work in Your Own Way**: you can connect with multiple remote repositories, rebase instead of merge, and work with submodules when you need it. But you can just as easily work with one central remote repository.
- **Don’t Mix Things Up**: while you’re working on feature A, nothing (and no-one) else should be affected by your unfinished code. Branching is the answer for these problems.
- **Go With the Flow**: Git is used by more and more well-known companies and Open Source projects: Ruby On Rails, jQuery, Perl, Debian, the Linux Kernel and many more.

Install git

Installing on Windows
http://git-scm.com/download/win

Check version

```
git --version
```

Create a github account and a git repository

Sign-up: https://github.com/join

Create repository: https://github.com/new

- with readme
- with .gitignore for Visual Studio Code

Configure git

Enter these lines (with appropriate changes):

```
git config --global user.name "John Smith"
git config --global user.email jsmith@example.com
```

You only need to do this once

Check settings:

```
git config --list
```

Getting help with git

If you ever need help while using Git:

```
git help <verb>
git <verb> --help
man git-<verb>
```

Example:

```
git help config
```

Initializing a git Repository in an Existing Directory

`git init`

- creates a new subdirectory named .git that contains all of your necessary repository files — a Git repository skeleton

Begin tracking those files and do an initial commit:

```
git add --all
git add README
git commit -m 'initial project version'
```

Cloning a git repository

```
git clone URL
git clone URL mypath
```

These make an exact copy of the repository at the given path

Via HTTPS

```
git clone https://github.com/svekozak/myreponame.git
```

Via SSH (you need a SSH key)

```
git clone git@github.com:svekozak/myreponame.git
```

Generating SSH keys

```
cd ~
mkdir .ssh
cd .ssh
Ssh-keygen.exe
```

When prompted for a passphrase, type a password to complete the process. When finished, the output looks similar to:

```
+---[RSA 2048]----+
|*= =+.           |
|O*=.B            |
|+*o* +           |
|o +o.  .         |
| ooo  + S        |
| .o.ooo* o       |
|  .+o+*oo .      |
|   .=+..         |
|   Eo            |
+----[SHA256]-----+
```

Check generated files by typing:

```
dir
```

Add SSH key to account

```
clip < ~/.ssh/id_rsa.pub
```

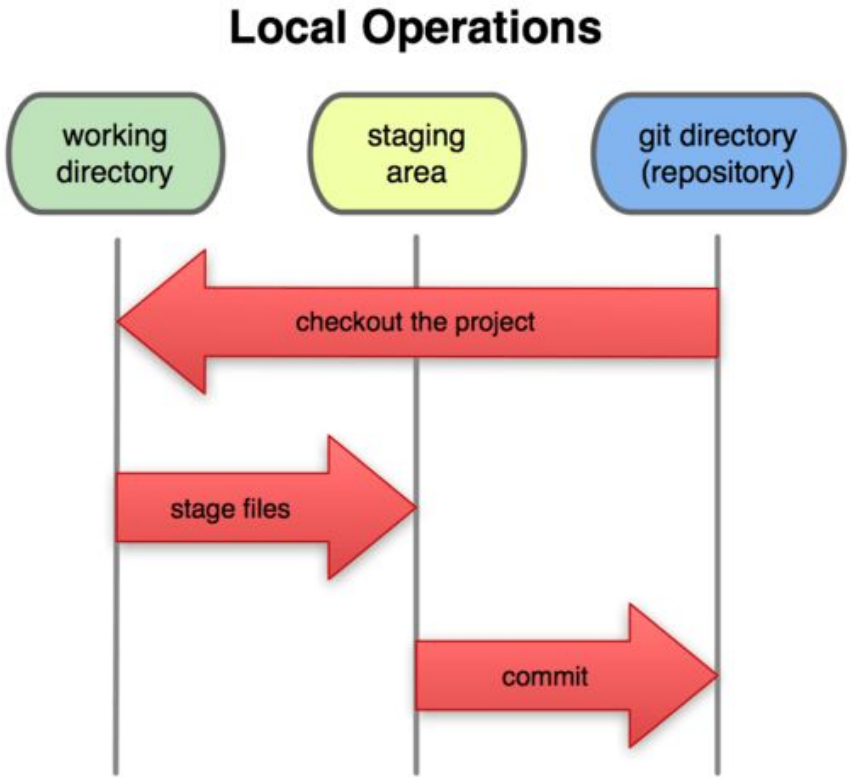
Go to: https://github.com/settings/ssh/new

git - The Three States

Committed - safely stored in your local database

Modified - changed the file but have not committed it to your database yet

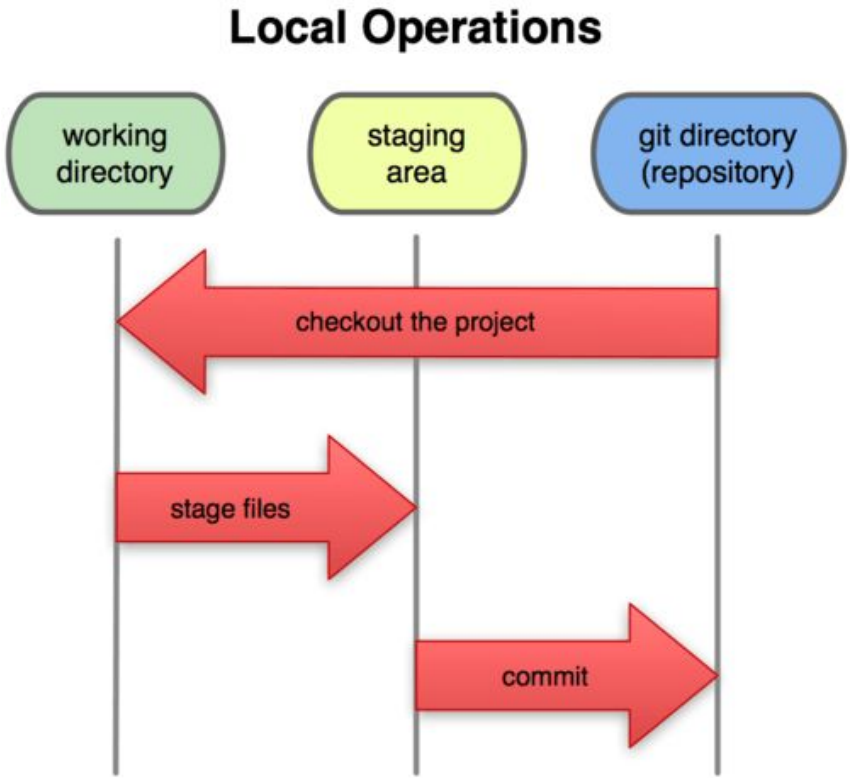
Staged - marked a modified file in its current version to go into your next commit snapshot



git - The Three Sections

Git directory - where Git stores the metadata and object database for your project (copied when cloning)

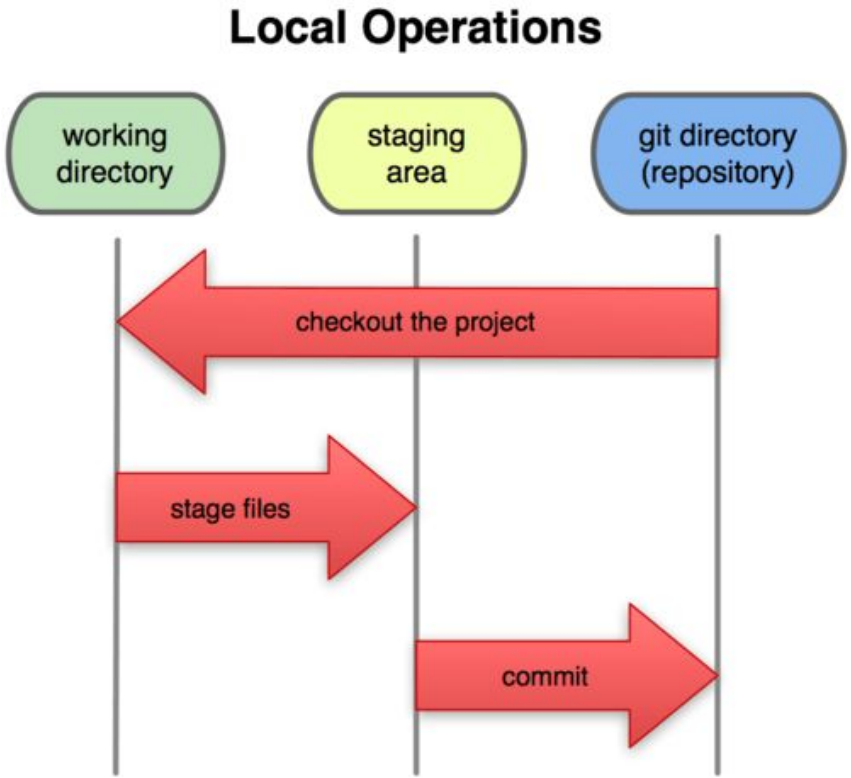
Working directory - a single checkout of one version of the project. These files are pulled out of the compressed database in the Git directory and placed on disk for you to use or modify.



git - The Three Sections

Staging area - a simple file, generally contained in your Git directory, that stores information about what will go into your next commit

- sometimes referred to as the index

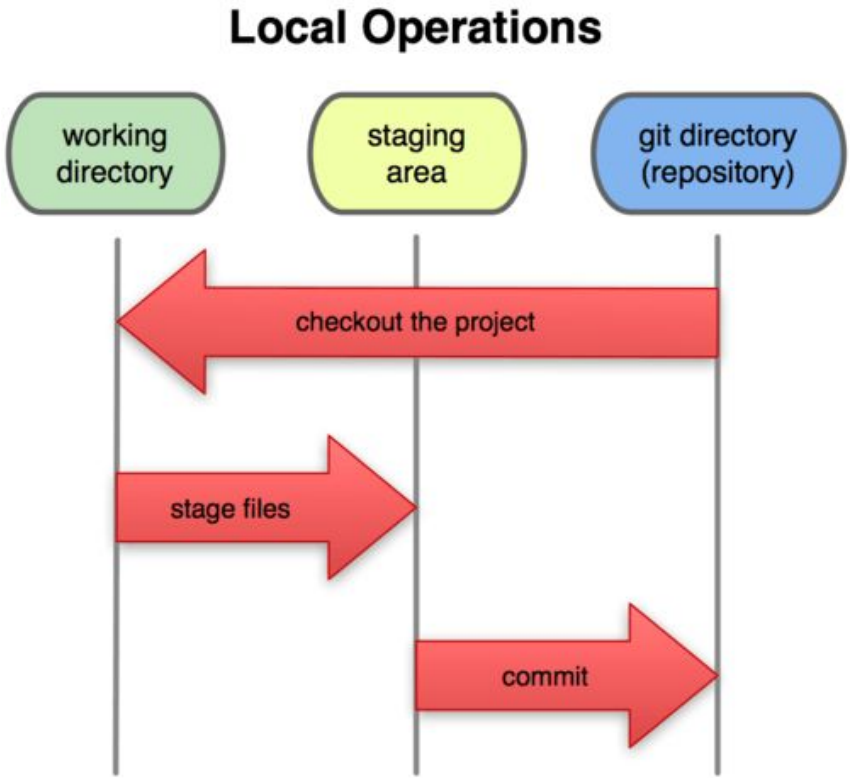


git - local workflow

1. You modify files in your **working directory**.

2. You stage the files, adding snapshots of them to your **staging area**.

3. You do a commit, which takes the files as they are in the staging area and stores that snapshot permanently to your **Git directory**.



Checking the Status of Your Files

```
git status
```

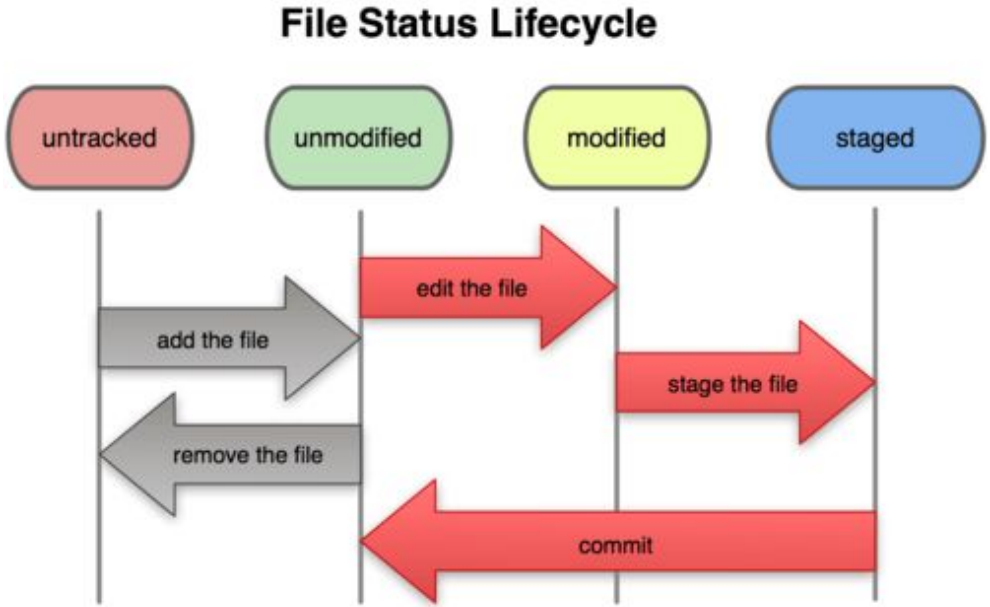
Tells you:

- which branch you're on
- your untracked files
- changes not staged for commit
- changes to be committed

```
git diff
```

- shows you the exact lines added and removed

```
git diff --staged
```
(`--cached`)



Tracking New Files / Staging Modified Files

```
git add (files)
```

Committing Your Changes

`git commit`

- launches your editor of choice (**`git config --global core.editor`**)

`git commit -m "PROJX-1978: Remove logo from footer"`

- typing your commit message inline

Viewing the Commit History

`git log`

- lists the commits made in that repository in reverse chronological order
- SHA-1 checksum, the author's name and e-mail, th date written, and the commit message

`git log -2`

- limits the output to only the last two entries

```
$ git log
commit bcb792dcc7dfbfcfd620ee73ed7422295f3d50ca (HEAD -> computer_player, origin/computer_player)
Author: lpenzey <lucaspenzeymoog@gmail.com>
Date:   Fri Jul 27 15:19:27 2018 -0500

    cleaned formating with rubocop

commit e953f0fdbfcf8038afec2a50f72c9d65601d346c
Author: lpenzey <lucaspenzeymoog@gmail.com>
Date:   Fri Jul 27 14:55:41 2018 -0500

    updated script

commit d443cc147cf543bc2892a82143e3b0ab016f7847
Author: lpenzey <lucaspenzeymoog@gmail.com>
Date:   Fri Jul 27 14:53:12 2018 -0500

    added travisci

commit bf0c6b5362ea8e675a6c866d388aee7867b816ea
Author: lpenzey <lucaspenzeymoog@gmail.com>
Date:   Fri Jul 27 14:49:45 2018 -0500

    added travisci

commit ed75abbca061c2c93834d1ee606a1b665175e10d
Merge: 08aa658 a098936
Author: lpenzey <lucaspenzeymoog@gmail.com>
Date:   Thu Jul 26 10:15:16 2018 -0500

    Merge branch 'save_resume_game' of https://github.com/lpenzey/Mastermind into save_resume_game
```

Working with Remotes

Showing your remotes:

`git remote -v`

Adding remotes:

`git remote add [shortname] [url]`

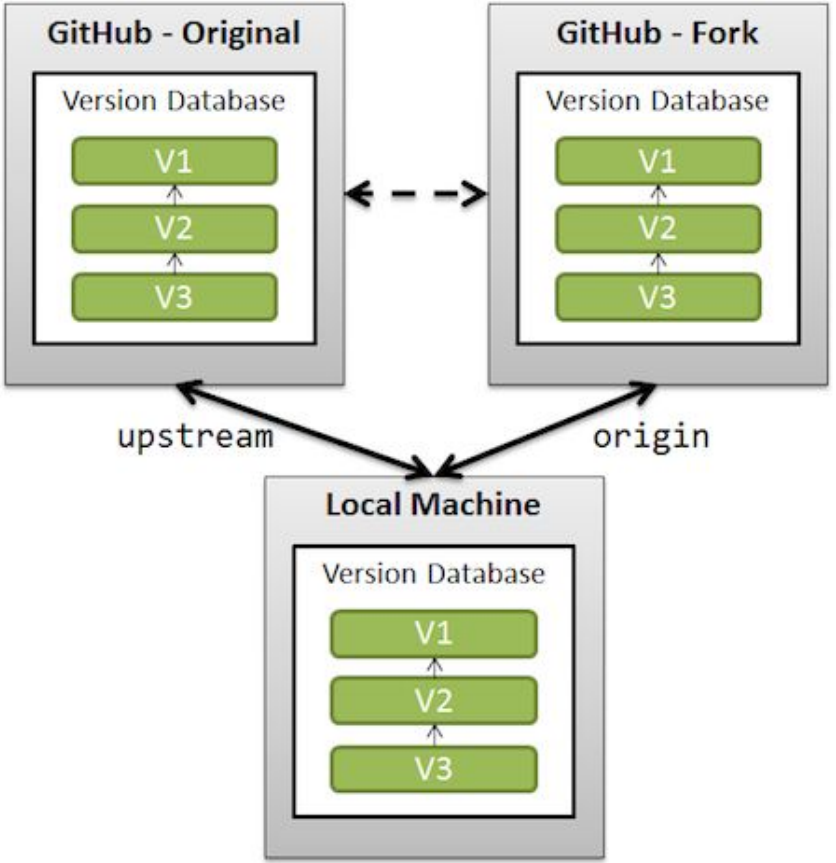
Fetching and Pulling from Your Remotes

`git fetch [remote-name]`

- fetches any new work that has been pushed to that server since you cloned (or last fetched from) it

`git pull`

- automatically **fetches** and then **merges** a remote branch into your current branch



Working with Remotes

Pushing to Your Remotes:

`git push [remote-name] [branch-name]`

- only if you cloned from a server to which you have write access and if nobody has pushed in the meantime

Removing and Renaming Remotes

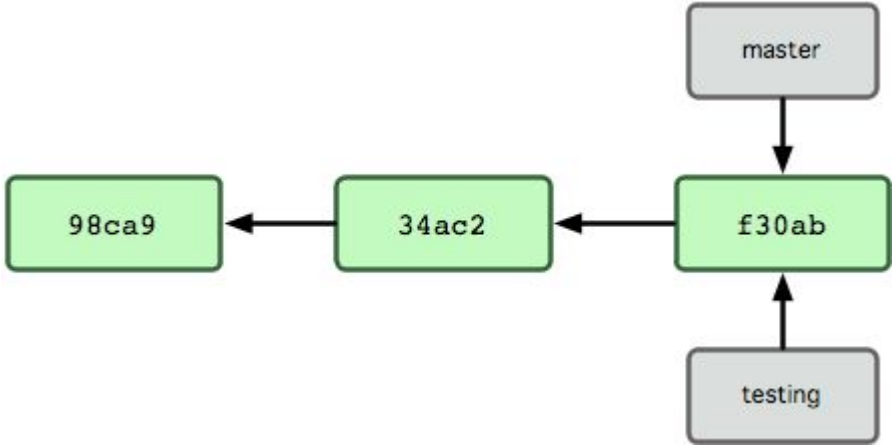
`git remote rename pb paul`

- this changes your remote branch names too: *pb/master* is now at *paul/master*
-

`git remote rm paul`

git Branching

Branching means you diverge from the main line of development and continue to do work without messing with that main line



Create a branch:

```
git branch testing
```

Move to a branch:

```
git checkout testing
```

Do both:

```
git checkout -b testing
```

git Branching

List branches:

`git branch`

- to see the last commit on each branch, you can run it with **`-v`**

Basic Merging:

```
git checkout master
git merge testing
```

Deleting a branch:

```
git branch -d testing
```

Cognizant
Softvision

Thank you!