# Git

# What is Git

**Version Control**

Software that keeps track of changes, esp text changes.

You make changes, move, etc. Git lets you look at versions, go back and forth, etc.

Git is a Version Control System.

Git is also A Source Code Manager

Examples of version control:

* Word track changes
* Undo CmdZ

**Git History**

SCSS – Source Code Control System. Bundled with Unix, very popular

* Keeps original version, and saved snapshot of set of changes.

RCS – Revision Control system

* Cross platform and open source
* Stored changed document, and worked with set of changes in result (faster)

Both only let you work with one version at a time

CVS – Concurrent version sustems

* Multiple files, entire project
* Put on remote server, let you have multi-version repos

CVS impvoed by Apache Subversion (SVN)

* Tracked text and images…
* Tracked changes to directory as a collective.
* Snapshot bigger than indiivudal files, it had entire sets of hanges at one time.

BitKeeper SCM

* Closed source and proprietary
  + Community version was free, used for source code management for lix kernel from 2002-205, controversial about being owned by company.
* Distributed version control
* Git was born bc of this

Git is deeply connected to linux

* Created by Linus Torvalds
* Distributed version control
* Open-source and free
* Open source project
* Faster than SCMs
* Better safeguards against data corruption

**Distributed Version Control**

* Git is distributed version control
* Different users maintain their own repos
* Changes are stored as change sets
  + Track changes, not versions
* Change sets can be exchanged between repos
  + You might say you merge in change sets, or apply patches
* Imagine changes to doc as sets ABCDEF

Repo 1: AbCDEF

REPO 2: ABCD -> just doesn’t have all change sets

Repo 3: Sets ABCE

Repo 4: ABEF

No repo is master repo, no one is out of sync, all repos are different

We often do make a repo a master repo, but this is convention, we can really do whatever

Bc distributed, no need to communicate w central server, no network access required, no single point of failure

Encourages participation and forking of projects – devs can work indepdnetly, make changes, submit changes for either inclusion or rejection. You can fork version of open source project and take it in your own direction. Well srted in teams.

All repos are considered equal

# Configuring Git

* Git stores configuration at system level, default configs
* Or user configurations
* In your home directory ~/.gitconfig
* Can also store in project by project basis
* Probably gonna want in all of your projects
* Git config –-system – system
* Git config –-global – user
* Git config – project

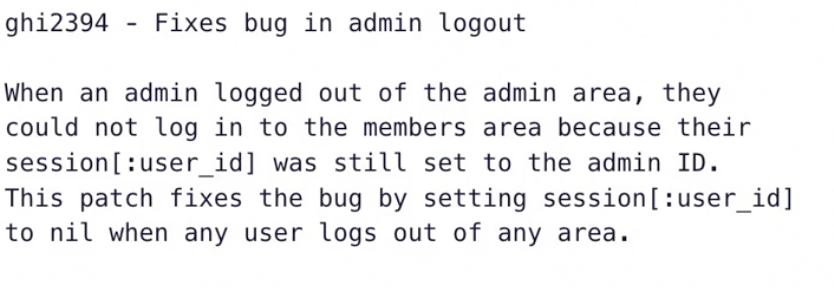
Autocomplete should be added to get rest of command

Git help is very useful

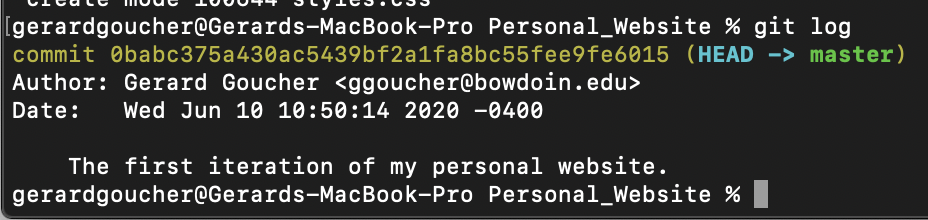
Where git files are stored – git uses many files to track changes, file tracking, etc.

Git tracks by making changes, git add . (. Shorthand for current directory) not tracking yet, we need to commit

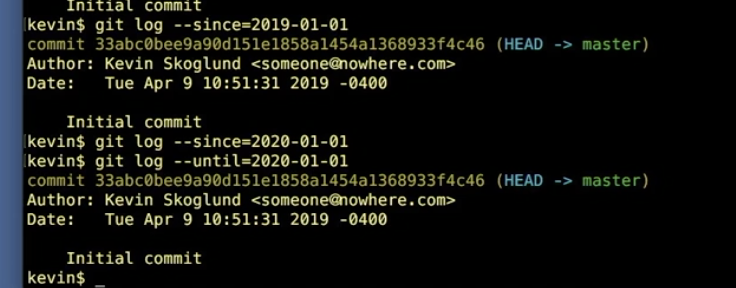
**Commit messages best practices**

* Write short, single line summary
* Optionally follow by a blank line and more complete description
* Keep each line <72 chars. Write in present tense, not past tense
* Clear and descriptive
* Good example

**Viewing commit log**



Author, date, unique commit id, message.

Git log -n 5 limits the number of commits you can see., or you can do commits since dates

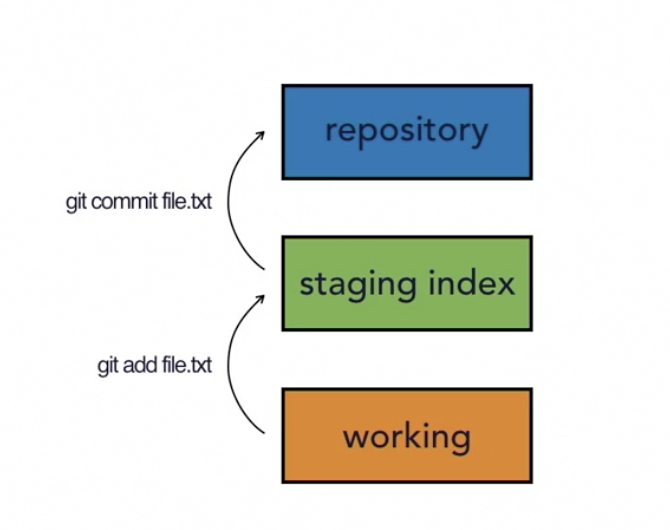
Can also do authors, too

--grep will show any string item when going through your work

**Three tree architecture**

* Repository – Has set of files working between it
* Working copy – same thing too

When we want to move files from repo to working, we checkout copies, when we finish changes, we commit to repo

* Git also has a staging index, you add to the staging index, and from there, you commit to the repo.
* 
* Lets us add what we want first, then commit, so we can choose selectively what we update

**Git Workflow**

Commits are snapshots of hchanges to our projects

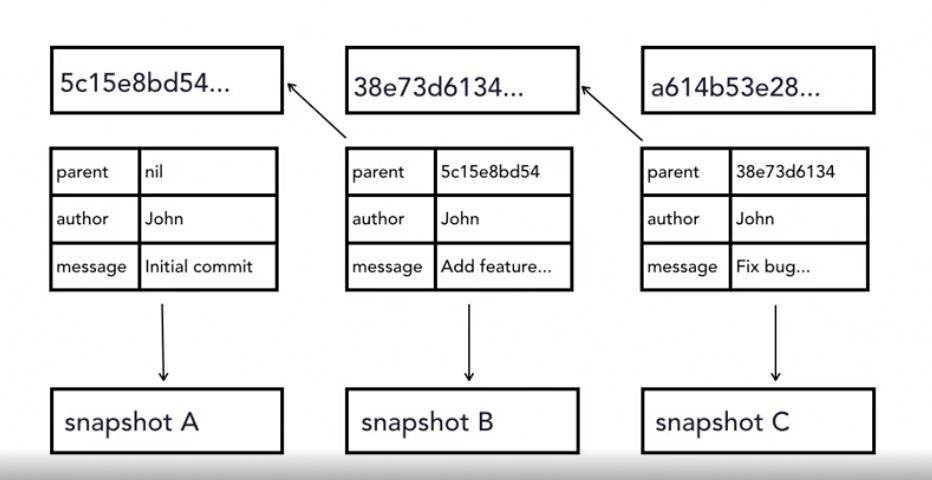
Git generates a checksum for each change set (hash value)

Checksum algos convert data into simple number

Same data has same checksum (checks to see if the same, checks data integrity)

SHA-1 hash algorithm creates checksums (SHA value)

40 char hexadecimal string (0-9, a-f)

It uses metadata as well (so also considers hash value for author and message) too, things are linked

**Head**

Pointer to tip of current branch in repo, points at current branch, last state of repo. Points where we’ve left off.

You always have to git commit -m

Git status shows you that you have nothing to commit. But what if we want to make some changes?

Modified shows you the file that are not staged for commit

When using git add, we’re really just adding sets of changes…

You can view changes in a file with diff.

Diff lets you view changes

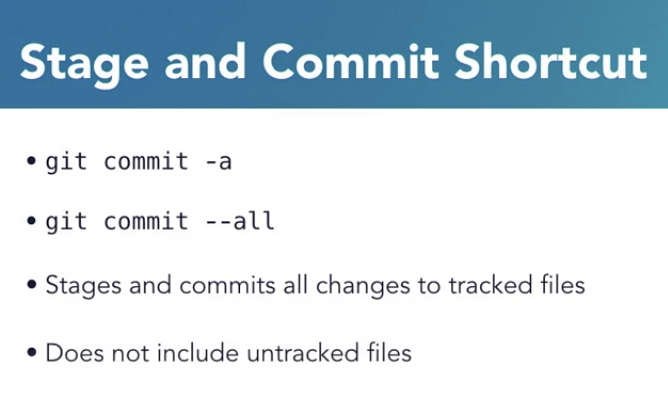
Git diff –staged lets you only look at set of changes in staging (lets you look at difference between repo and stage) - - cache means the same thing

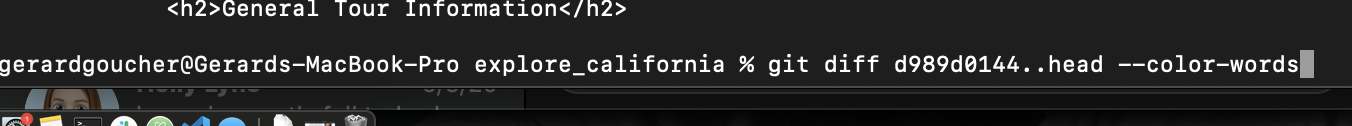
Git rm (remove) – permanently removed, not even moved to the trash

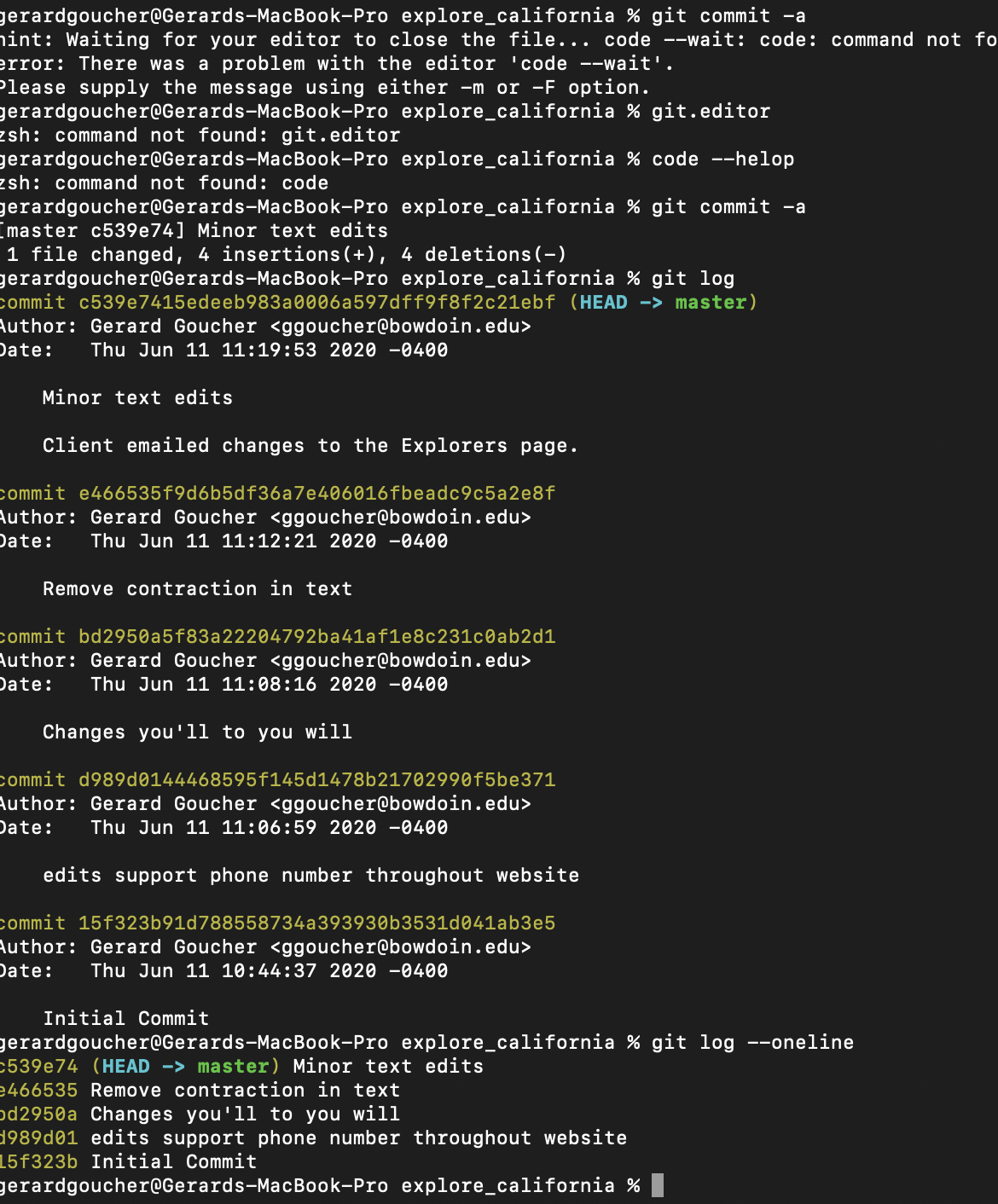
Moving and renaming files in git.

If you change filename in finder, irt will delete first, and then adduntracked for another… it’ll view files as second. Whne you put into staging, you can see this quickly.

Moving a file and renaming a file is the same thing (as you’re moving to a different path. Git mv).

Basically more beneficial to move via git…

Git show sha really does a great job of showing us what changed in each commit

Lets you see code in different levels of time

Git commit -a lets you provide multiline messages in your default text editor

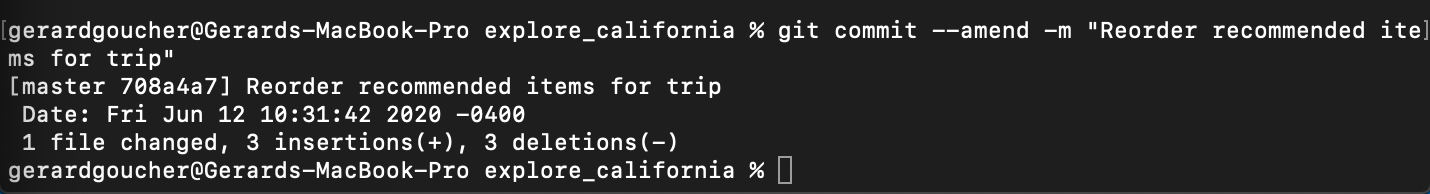
**Atomic Commits**

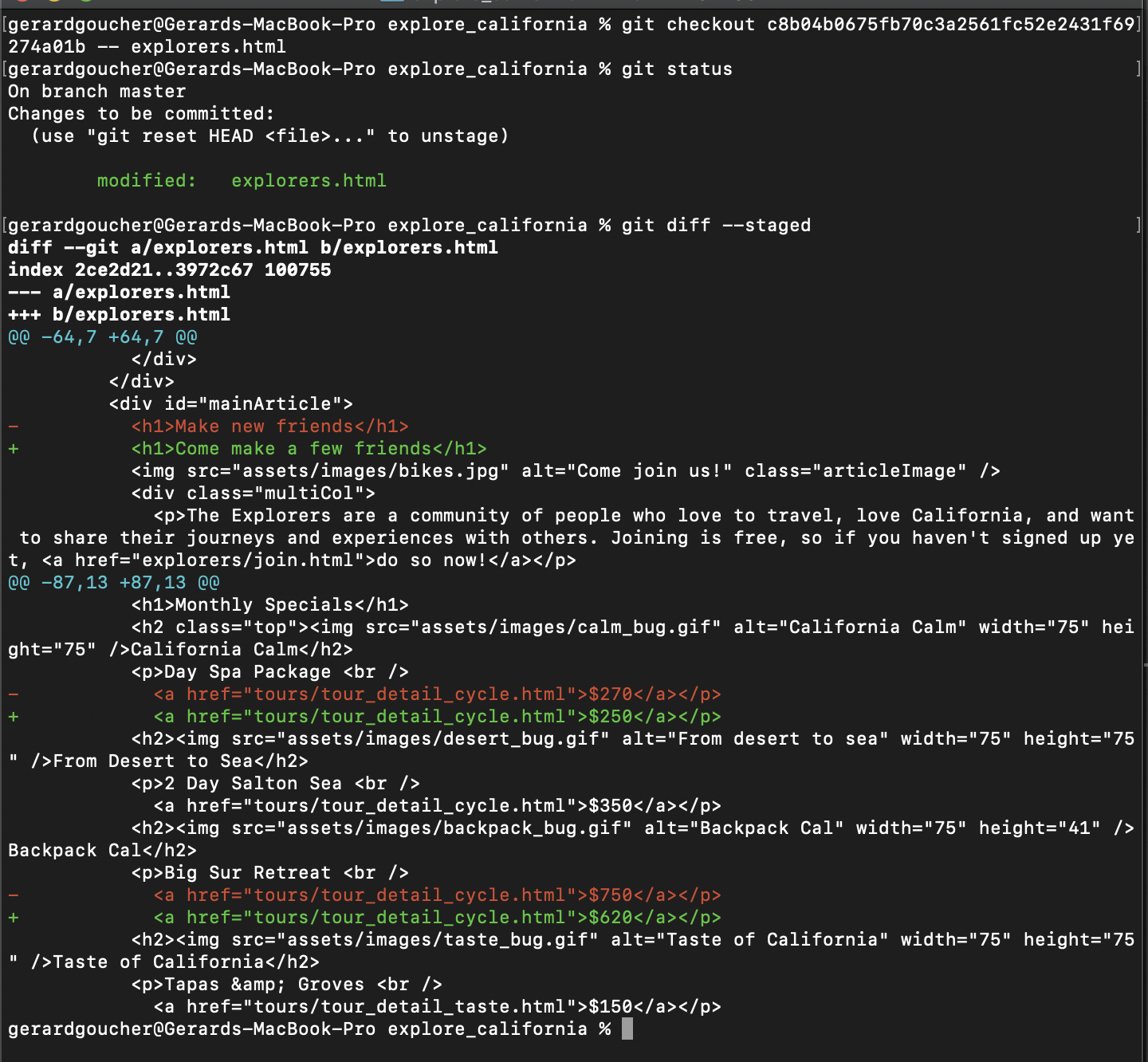
Small commits, make them as small as you can – only affect a single aspect (relate to one thing). Makes them easier to understand, to work with, and to find bugs. Improves collaboration. Atomic commits grabs certain features.

Sometimes you want to edit separately – so why would you add contact page change and also phone number changes

Git checkout – file discards changing in working directory

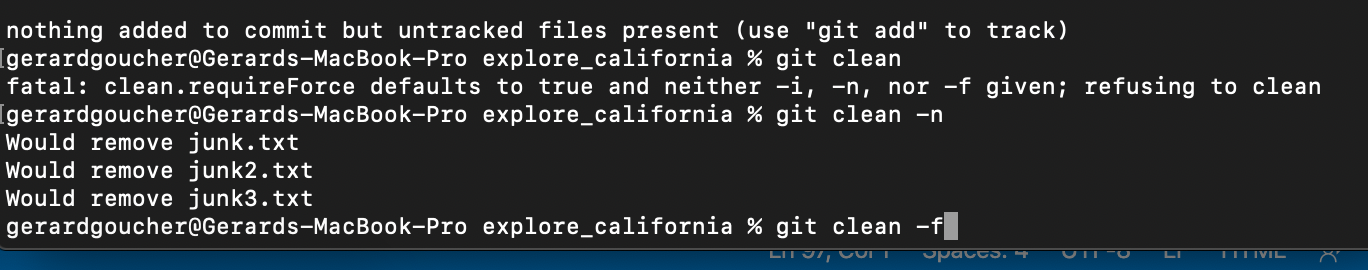
PREVIOUS SHAS STORED IN METADATA – THUS YOU CAN’T CHANGE THE SHA VALUES AND THE GIT METADATA IS VERY STRONG.

You can amend most recent commit, though, as this doesn’t break the change bc you’re at the head. You can also use this to just edit commit message. Changes the SHA, though

Make edits in new commits (change from singular to plural, go back from plural to singular, use new commit)

If you checkout the version before the version you changed (SHA shown above), you can see what’s different)

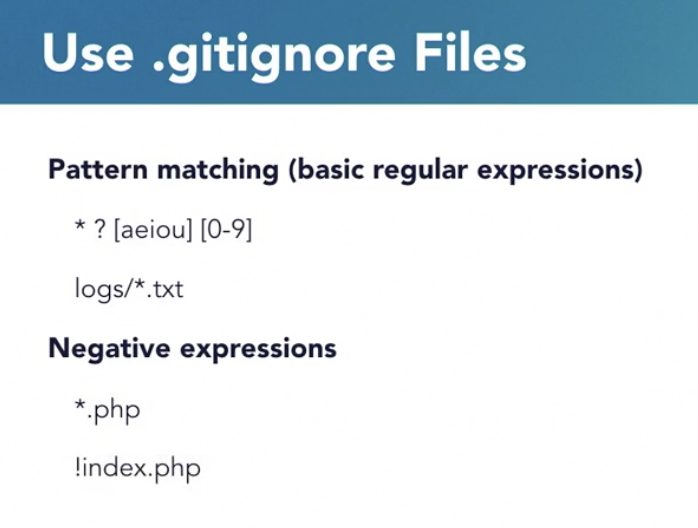
**Removing untracked files**

Removes compiled code, log files, extraneous stuff, etc. We could remove one by one, or we could remove all at once

Git clean -n – removes everything untracked from repo and staging tree (just in working directory) git clean -f removes everying

**Git Ignore**

.gitignore allows you to ignore files

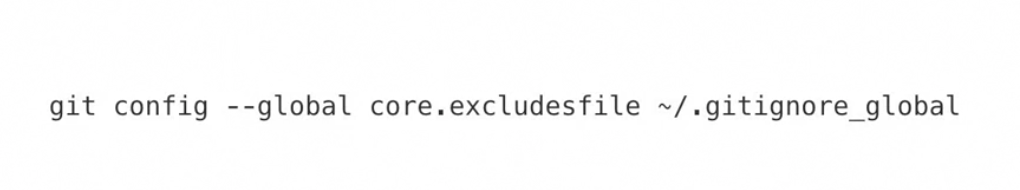
If you put a file in .gitignore, will follow changes and won’t ignore. 

^doesn’t ignore index.php

You can also use trailing slash…

**What to ignore in Git Ignore**

* Compiled source code
* Packages and compressed files
* Logs and databases – changes frequently
* OS generated files – not assential to project
* User uploaded assets – change too frequetnyl
* Github has useful gitignroe templates

You can set ignore file globally

^tells you what file to issue for exclusions (gitignore). Good place is in user directoru. You maybe should just make the .gitignore every time

Ignore files already tracked

What if you want to start ignoring? You could rm db\_config.txt. remove it form the stagin index.

Git rm –cached db\_config.txt, your file will still be there, nothing to commit, totally fine. File still exists in Git, but future changes not changed.

**Track empty directories** – ignores these normally, but what if you want to track for future changes?

Just make a .gitkeep file in this – and it’ll start tracking,

Or you can just do touch temp/.gitkeep