git	git is a code management software. So when you work with a team creating a software, you can all work on the same code and it will not corrupt. It is also used in open source community to contribute together.		All code is stored on a server somewhere. Then on your local machine, you have a directory of your choice store the code for the project. You make git run on this directory to communicate with the server		
	in your local directory, git consists of three "trees" that is maintained by git. That is really how the whole system works. The first is your "Working directory" which holds the actual files. It is really just a folder with your code. Then you have the "index" which acts as a "staging area". Files end up here after you "git add" them. They are flies that are ready to become committed and become part of repo. Then we have the "HEAD" which points to the last commit you've made. It is the structure of the project.				
	Flow of git. When to do what step:				
	There is add, push, pull, branch, etc. When do I do what?				
	add	you add files when you create them and are ready to make them part of your repo.			
	branch	when you want to create a new feature or fix a bug, branch. It creates a copy of the project so you do not mess it up at all. It remains untouched until you "merge" these changes in.			
	commit	You always commit when you get a "task" done. This keeps everything organized. So if you add a new little feature but you're not done coding for the day, commit. It keeps everything organized to take that "snapshot" of the project at every step along the way of the project.			
	merge	This is after you are done with your branch. When you have branched and you want to put that code back into the main project.			
	pull	when you sit down to your project and you want to update your local directory with the main repo on the server. you "pull" in all of the changes other developers have done.			
	push	you push when you are ready to contribute to the project. When you have made some commits, done some branching and merging, whatever. When you are ready for others to see your changes, push it.			
	sites to learn git from:				
	https://help.github.com/	more to learn github, but it has some concepts on there			
	http://learn.github.com/p/intro.html	more in depth of git. Not beginner			
	http://gitimmersion.com/	walkthrough of git. Starts at beginner level	GETS Very Advanced		
	http://rogerdudler.github.com/git-guide/	great if you forget something. Quick great overview			
	http://try.github.com/levels/1/challenges/1	great interactive approach. What I began learning git from. Recommended for beginner			
	TIPS:				
	When you commit changes to a repo, make those changes related to each other. So if you want to update the GUI and database of project, do them in seperate commits. Since commits are "snapshots" if we want to go back in time, we can neatly. So make all progress different commits if they are not related				
	GETTING STARTED:				
	If you want to start a git repo, first I need to ask:				
	Are you starting from scratch? Or is there already a repo you have somewhere you want to use				
	from Scratch-----	1. go to directory on local machine you are going to have all code.			
		2. command: git init			
		3. Now from here, you can either clone a repo that is already started or you can start writing code and using the commands below along with way to build your repo			
	Already started repo----	1. Create copy of local repository (git clone /path/to/repository) or remote server (git clone username@host: /path/to/repository)			
	commands:				
	gitk	GUI for git. run command "gitk" when in a directory git is installed			
	git add fileName	this adds files to your git project's staging area. File is now being tracked by git and is now part of the git project!	You can use wild cards here like git add *.txt *.txt will add all files in CURRENT DIRECTORY ending in txt. If you use '*.txt' (single quotes around) then it adds all files ending in .txt in current directory AND subdirectories		
		git add .	this period will add all files and directories in it to the repository.		
		git add -i	interactive adding. asks you about files while it adds to make sure you are not doing something you shouldn't		
	git branch snapShotName	When you want to fix a bug or add a feature, make a branch of the code. This is a copy of the code where you can make seperate commits to that does not mess with the main code project. Then when you are done with bug or feature, you can merge it back into the project.			
		snapShotName = a name you are giving to your branch. Name is what you are doing to it. So "clean_up" or "bug_database_query" something.			
		git branch = running this by itself will list all of the branches created to you. You can then switch back and forth between your master and other branches with "git checkout branchName" command		You can run "git checkout -b newBranchName" to do the branch and checkout command all at once so it creates a branch and checks out to it.	
		git branch -d branchName	After you merge, you do not really need your branch anymore so this command deletes the branch. This command only works if you have merged the branch with master. If you want to delete a branch before you merge, you must run: git branch -d -f branchName		
	git checkout -- <target>	git checkout will be like a time machine. Target is a filename. That file will be made into the way it was at the last commit that was made.			
	git clean -f	will clear all changes done in your local working set since last pull.			
		-f is an argument			
	git clone /path/to/repo	if you already have a git project on your local machine or a drive, use this command and it will copy it to the directory you are in now. I believe you need "git init" first			
	git clone username@host:/path/to/repo	if you already have a git project on a remote server, use this command and it will copy it to the directory you are in now. I believe you need "git init" first			
	git commit -m "Type some comments here for what you did to project"	This adds your changes of the project that is currently in the staging area (use "git status" to see changes to be committed) and puts those files into the project repository			
		-a = autoremoves files. If you forget to use "git rm" command and instead use bash "rm" then you will either need to run "git rm" for each of those files you bash "rm" or use the -a argument.			
		-m = allows you to do commits from command line. "git commit" will launch your system set editor. From there, on the first line, type your commit comment. Then save file and exit			
	git diff HEAD	usually you run this after doing a "git pull". It displays what has changed to the repo since your last commit.	HEAD is a pointer so optional, but recommended. It holds your position within all your commits. It is used to display changes since your last commit		
		you may also use "git diff --staged" to see what you have done locally to project since your last commit. It helps keep you organized on when you should commit again			
		git diff sourceBranchName targetBranchName	preview the difference between branches		
	git fetch origin git reset --hard origin/master	you want to drop all local changes and commits done and fetch the latest history from the server and point local master branch at it		So if you have really screwed up and want to start over from the most current commit, run these commands	
	git init	run this in the directory that you want to store the project code in. it makes a hidden directory there called ".git". This "initializes" your project			
	git log	prints all the changes committed in project.			
		git log --summary	will give more info about each commit		
		there are lots of more options you can do to see only commits you want to see.			
			git log --pretty=oneline --max-count=2		
			git log --pretty=oneline --since='5 minutes ago'		
			git log --pretty=oneline --until='5 minutes ago'		
			git log --pretty=oneline --author=<your name>		
			git log --pretty=oneline --all		
			git log --all --pretty=format:"%h %cd %s (%an)" --since='7 days ago'		all changes made in last week
			git log --pretty=format:"%h %ad | %s%d [%an]" --graph --date=short		pretty way of displaying commits
			For even more...do "man git-log"		
	git merge branchName	after you branch, make all your commits, etc, and your ready to add your changes to the main project, use "git merge" command.			
		Make sure that when you run this command, you use "git checkout" command to go to your "master" branch (the main project) before you run this command. Merge will merge branch you classify for "branchName" argument with the branch you are currently "checked out" in.			
		When you merge, yes it can make conflicts. That is you and someone else merge the same file and git cannot just pick one of them randomly so it prompts you and makes you take care of it.	for now, I will put this link here to read about conflicts. If you want to add notes about it, go ahead: http://git-scm.com/docs/git-merge#_how_conflicts_are_presented		
	git pull origin master	when you push changes to the git repo, other people may add some code, change some things, so git pull will bring in those changes made to the repo for you. It updates your local repo to be in sync with git server repo		you may want to run "git diff HEAD" to see what has changed after running this command	
	git push -u origin master	push tells git where to put our commits. It means to upload our code progress to the server. Do this when you have made a bunch of good, bug free changes to project.			
		origin = this is the name of our remote. You set this I believe in the "git remote add" command			
		master = branch name			
		-u = tells git to remember the parameters you set so next time simply run "git push" and git will know what to do.			
	git remote add origin git@github.com:userName/my_repo.git		When you create a new repo on some git server like github, you can upload your existing local repo onto the server to get started faster.		
		So lets say you wanted to make a new project. Just jump right into it. On you local machine, go ahead and run all the commands to make the project. Then once you are ready to upload it to a git server like github, then you can use this command to add your code there. Then your project is now a repo for everyone!			
		the only thing you should have to change is the web address after the @ symbol, userName is your username. Then the path to your repo after your username. Your server should tell you all of this.			
	git reset fileName	removies file(s) from staging area. (so not part of project anymore). Files are still on local machine and NOT deleted, simply removed from staging area. Run "git rm" for that			
		git reset --hard origin/master	do this after you run command: "git fetch origin" to erase all of your local changes and commits and start off where last commit was done on repo		
	git rm	removes files from your project. This is NOT like "git reset" command. This will delete code from local machine and repo.			
		If you delete a file with bash "rm" command, you will still have to use "git rm" to remove the file from the project. Try to always use "git rm" but if you forget, then use "git commit -a" argument which will remove all deleted files with the commit. So a full commit would be: "git commit -am "Delete stuff"			
		git rm fileName = removes file			
		git rm *.txt = removes all txt files in current directory			
		git rm '*.txt' = removes all txt files in current directory and all subdirectories			
		git rm -r directoryName = removes directory with all files inside it			
	git status	tells us our current status our git project/setting up is in.	files that git notices are new, it will read them as "Untracked files:". You need to run "git add fileName" to add to staging area so git notices it		
		You may see one or more of the following key words with files listed under them:			
		deleted	file has been deleted and is waiting to be removed from git		
		staged	files are ready to be committed		
		unstaged	files with changes that have not been prepared to be commited		
		untracked	files aren't tracked by git yet. This usually means a new file was created. (run "git add")		
	git tag 1.0.0 1b2e1d63ff	tags are labeling your commits as software release numbers. So 1.0, 2.0, etc.			
		1b2e1d63ff stands for the first 10 characters of the commit id you want to put the tag on. (you can get commit id with: git log)			
	terms:				
	branch	when developers are working on a feature or a bug they will create a copy (branch) of the code project so they can make seperate commits to it without messing with the main project really bad. Then when they are done, they can merge the branch back into the main "master" branch		a branch is NOT available to others unless you push the branch to the remote repo: git push origin branchName	
	commit	A "commit" is a snapshot of our repository. This way if we ever need to look back at the changes we've made (or if someone else does), we will see a nice timeline of all changes.			

	deleted	file has been deleted and is waiting to be removed from git			
	pull request	in github (not sure if others use it too) you can do what is called a pull request. It allows the boss of the project to look through your changes you made and make comments before deciding to merge it with the main project. So when you branch and merge, then a pull request may be made			
	staged	files are ready to be committed. All files listed as "staged" will be included in the next commit. Find out what files are staged with: "git status". (they will be in green. red ones mean they are not added to staging area yet) if there are any files here you do not want staged then use: "git reset fileName" command			
	staging area	A place where we can group files together before we "commit" them to Git. It is on local machine and not part of main repo yet.			
	unstaged	files with changes that have not been prepared to be commited			
	untracked	files aren't tracked by git yet. This usually means a new file was created. (run "git add")			
	configuring:				
	**anytime you want to make these settings a system wide setting, add "--global" after "config": "git config --global user.name "first last"				
	git config color.ui true	use colorful git output			
	git config --global credential.helper cache	saves your repository password in memory so you do not have to type it in	Only works with HTTPS. NOT ssh. That requires keys	default length of time to save password is 15 minutes.	
	git config --global credential.helper 'cache --timeout=3600'		sets how long you want your password saved. 3600 is in seconds. So that is 1 hour		
	git config format.pretty oneline	show log on just one line per commit			
	git config user.name "firstName lastName"	sets your name that will show on your commits			
	git config user.email "emailAddressHere"	sets your email that will show on your commits			
	tools:				
	gitk	GUI for git. run command "gitk" when in a directory git is installed			