

P1L4 Version Control Systems

Notes 08/25/2014

We'll talk about a fundamental type of tools in the software engineering arena: Version Control Systems.

- *AKA Revision or Source Control Systems*

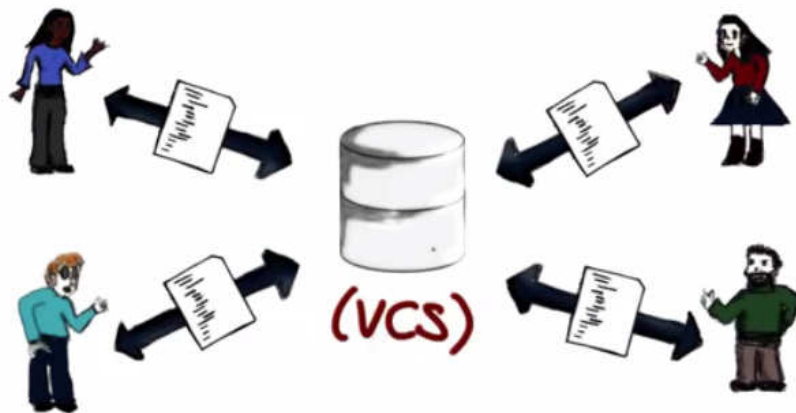
We will focus on a specific version control system called git and present it from a conception perspective and also get some hands on experience.

Interview with John Britton (works with GitHub)

- A version control system is a tool that software developers use but anybody who's working with digital assets/projects can also use for keeping track of revisions of their project
 - By revise we essentially mean snapshots of your project over time.
 - Imagine doing some work and then every so often, be it every couple of hours, every couple of days, saving a permanent snapshot of your project.
- The most immediately obvious benefit to having snapshots of your project (to keeping revisions) is that you can go back.
 - Sometimes you start working on trying to solve that last step, and you break things.
 - You make it worse than it was an hour ago and at that point it's easier to just go back to what you had then trying to figure out what you broke.
- The other big one is being able to collaborate with multiple people
 - It's pretty seldom these days that you work on a production totally on your own. It's most common to work in teams and small groups.
 - Using a revision neutral system allows you to collaborate with other people.
 - And make sure that you don't step on each other's toes as you're working.
- If you have used version control systems before, you may have heard of something like subversion, CVS, or maybe a commercial solution like ProForce.
 - The main important characteristics of Git are
 - It's open source.
 - It's a distributed version control system.
 - The distributed version control system is essentially a system for tracking revisions of your software that doesn't have any central repository.
 - Biggest characteristic is that I can do my work and you can also work on the same project at the same time without communicating with each other and without communicating to a central system.
- GitHub is the world's largest code host, and we essentially have a website where you can collaborate with people when you're writing code.

- Two ways you can use GitHub.
 - You can use it publicly for open source
 - You can use it in private within your team, or your company, or within your class.
- Git Hub started out just as a way to host your Git repositories but it's actually grown into quite a bit more.
 - It's an entire collaboration system around, around your code.
- We're approaching five million users.
- Polar Quest Feature
 - When you're using GitHub, you can share your Git repository, do some work, and actually do a code review off proposed changes which is what we call a polar quest on github.com.
 - It lets you have a discussion about a set of proposed changes and leave feedback in line with the code.
 - You could say, for example, this method needs to be re-factored or I think I found if off by one error here, just different types of feedback so that before you integrate some proposed changes you have a kind of conversation about your code.
 - That's really valuable when you are working in a team.

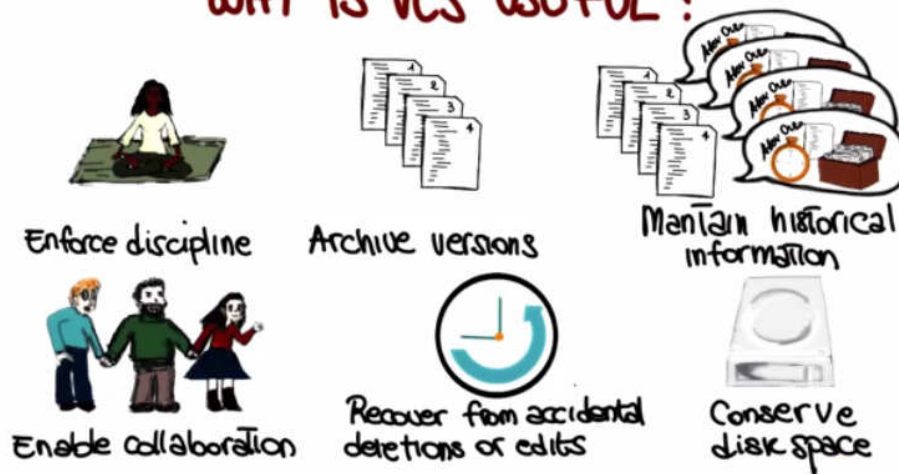
WHAT IS A VERSION CONTROL SYSTEM?



Version Control System Introduction

- A version control system, or VCS, is a system that allows you to manage multiple revisions of the same unit of information such as documents, source files or any other item of that sort.
- And as the graphical depiction shows, a VCS allows a multiple actors. Here we have four, to cooperate and share files.

WHY IS VCS USEFUL ?



- Using a version control system enforces discipline, because it manages the process by which the control of items passes from one person to another.
- It allows you to archive versions, so you can store subsequent versions of source controlled items into a VCS.
- You can also maintain a lot of interesting and important historical information about these versions.
 - Who is the author for this specific version stored in the system.
 - On what day and what time that version was stored.
 - And a lot of other interesting information about the specific version of the item that you can then retrieve and use to compare versions.
- Having a central repository in which all these items are stored enables collaboration, so people can more easily share data, share files, share documents through the use of VCS.
- In the following cases a VCS can be extremely useful because it will allow you to recover from accidental deletions or edits. (You can go back to yesterday's version that was working perfectly and compare yesterday's version with today's version and see what it is that you changed.)
 - Deleting a file by mistake
 - Modifying a file in the wrong way
 - Changing something in your code and breaking something and not being able to go back to a version that was working (not remembering what you changed that broke the code)
- Will typically allow you to conserve and save disk space on both the source control client and on the server because it's centralizing the management of the version.
 - Instead of having many copies spread around, you'll have only one central point where these copies are stored or a few points where these copies are stored.
 - Version control system often uses efficient algorithms to store these changes so you can keep many versions without taking up too much space.

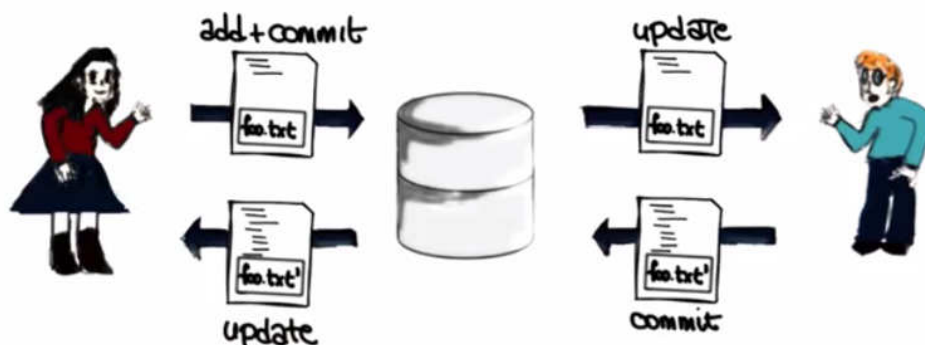
ESSENTIAL ACTIONS



Essential Actions

- Some essential actions that version control systems perform.
 - The addition of files
 - You can add a file to the repository.
 - At that point the file will be accessible to other people who have access to the repository.
 - And now the fundamental action is commit.
 - When you change a file that is already in the repository, when you make some local changes to a file that is already in the repository, you want to commit your changes to the central repositories, so they can become visible to all of the other users and other repositories.
 - Another fundamental action is the action of updating a file.
 - If we have a repository and someone else can modify the files in the repository, I want to be able to get the changes that other people made to the files in the repository.

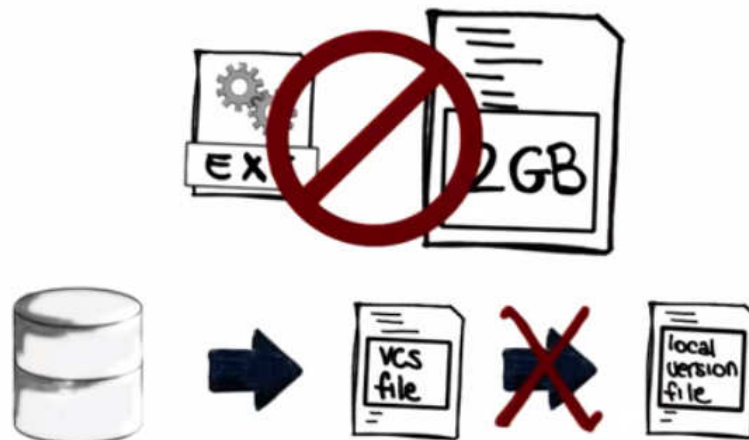
EXAMPLE WORKFLOW



Example Workflow

- Let's look at the basic workflow in a version control system using the three actions that we just saw.
- We'll use our friends, Brad and Janet and a VCS that they are using.
- Janet creates a file called foo.txt and puts some information in the file.
- At that point she might want to add the file to the repository and to commit it so that her changes and the file get to the central repository.
- When she adds and commit, foo.txt will become available and will be accessible to the other users, in this case, Brad.
- If Brad were to run an update command, what will happen is that the file foo.txt will be copied on the local work space of Brad and Brad will be able to access the file.
- At this point Brad might want to modify the file, for example add something to this existing file.
- After doing that, he also may want to share the updated file with Janet.
- He will commit the file and the result will be exactly the same of when Janet committed her file—the updated file will be sent to the repository and the repository will store that information and make it available for other users.
- Now if Janet performs an update, she will get the new version of foo.txt with the additional information that was added by Brad.

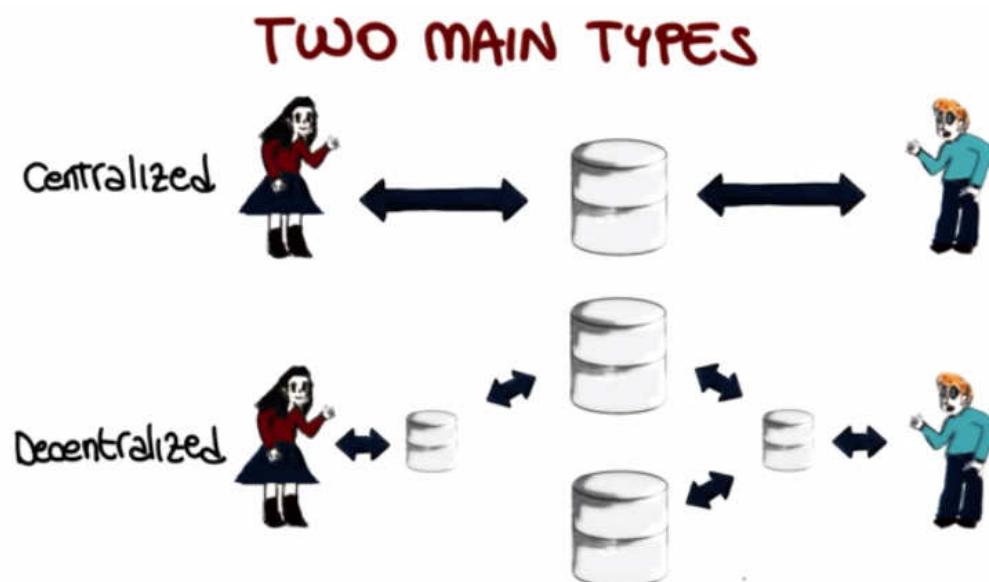
"DON'TS" IN VCS



Don'ts in VCS

- There are two kinds of resources that you don't want to add to a VCS normally.
 - One is derived files.
 - For example an executable that is derived by compiling a set of source files, where the source files all already in the repository.

- At that point, there is no reason to also add the executable file in the repository.
 - In general, any executable file should not be added to repository.
- The second class of files that I want to mention is these bulky binary files.
 - If you have one such file, it is normally not a good idea to store them under a version control system, to store them in the repository.
 - In general, these are the kind of files that you want to keep local, and you don't want to put in the VCS repository.
- Another typical mistake is you get your file from VCS and so you get your local copy of the file that was in the VCS, and you want to make some changes, and before making the changes you decided, let me actually save a local copy of the file, and I'm going to work on that one. Or let me save it before I modify it, or let take a snap shot of a whole tree of files. Just because I don't really trust the fact that VCS is going to be able to help and is going to be able to recover from possible mistakes.
 - Never do this.
 - It always leads to disasters.
 - First of all it is useless, and second it's risky.
 - What happens is that at the time you have to turn in your assignment or code you always end up being confused about which is the version that you're really using.
 - So absolutely no local copies!
 - No local redundancy when you're using a version control system.
 - Trust the version control system, and trust the version control system to be able to manage your versions.
 - You can always save it, commit it, retrieve previous versions, and you'll be able to do everything that you can do by copying the file yourself, and even more.



Two Types of VCS

- There are many different version control systems
- We can classify them normally in two main types
 - Centralized VCS's
 - Decentralized VCS's.
- In the case of a centralized version control system there is a single centralized repository on which you are committing files.
 - When Janet commits a file, the file will go from her local working directory to the repository, and the same will happen to Brett.
- The decentralized system is a little more interesting because in this case, they will both have sort of a local repository in which they can commit their changes
 - They can commit changes without the other users of the VCS being able to see these changes.
 - And when they're happy with the version and ready to release the version, they can push it to a central repository.
 - At that point, it will become available to the other users of the repository—to the other users of the VCS.
 - There are several advantages in a distributive system.
 - Having a local version - you want to kind of take a snapshot of what you have but you don't want that snapshot to be available to the other users because it's still not ready to be released
 - If you're using a centralized system, there's really no way you can do that, unless you make a local copy, which is something we said you don't want to do.
 - With a distributor—with a decentralized VCS—you can commit your local changes here, in your local repository, and you can push them to the central repository only when you're ready
 - You can use multiple remote repositories.
 - For example, Brad might want to push to another remote repository, as well.
 - This could be a repository where the files are accessible for wider distribution.
 - Imagine developing a software system in which a team is sharing internal versions, and then only some of these versions are actually pushed to the repository that is seeable to the whole world.

Introduction to Git

- One good representative of distributed version control systems, is GIT.

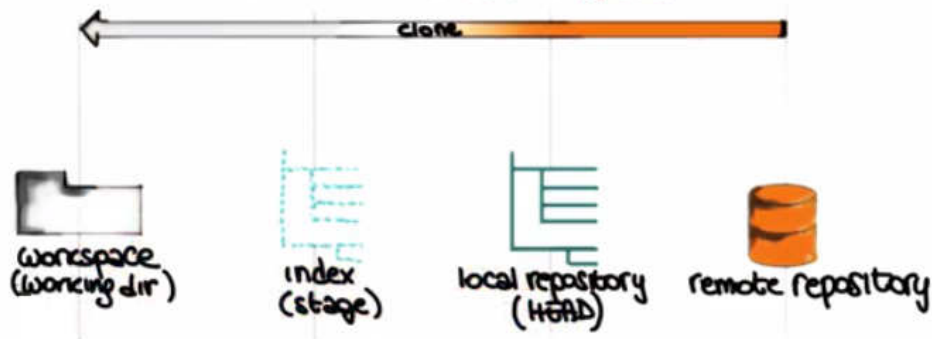
- It was initially designed and developed by Linus Torvalds, the man who started and created the Linux operating system.
- Linus was unhappy with the existing version control systems, and wanted a different one.
- He wanted to use it for maintaining the Linux kernel and wanted one with some key characteristics.
 - The fact that it was distributed.
 - He wanted it to be fast.
 - He wanted it to have a simple design
 - He wanted to have a strong support for parallel branches, because many people were contributing to the kernel at the same time.
 - Finally, he wanted for the virtual control system to be able to handle large projects.
- Git popularity: there was a survey performed across the Eclipse IDE users, and it showed that in 2013 GIT was used by about 30% of the developers.



Git Workflow

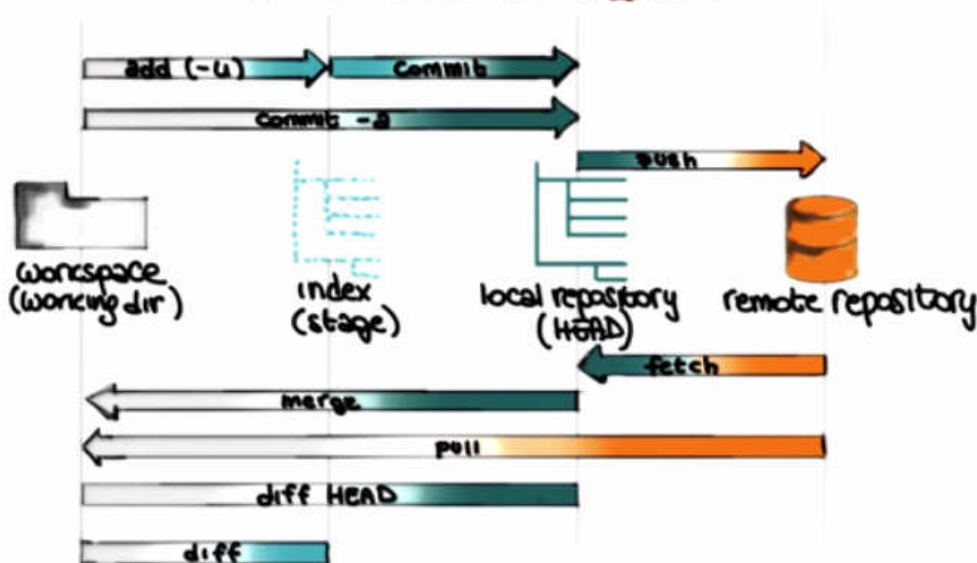
- There are four fundamental elements in the GIT workflow
 - The workspace - your local directory.
 - The index - also called the stage
 - The local repository - we'll also refer to this as HEAD
 - The remote repository
- If you consider a file in your work space it can be in three possible states.
 - Committed - which means that the data, the latest changes to the file are safely stored here.
 - Modified - which is the case of the file being changed and none of these changes have been saved to the local repository (so locally modified)
 - Staged - the file is basically part of this index, i.e., that it's been tagged to be considered in the next commit.

GIT WORKFLOW



- The first command that you normally run when you're getting access to a remote repository, is the git clone command.
- The git clone, followed by the url for that repository, will create a local copy of the repository in your workspace.
- You don't have to do this step if you're creating the repository yourself.

GIT WORKFLOW



- The next command that we already saw is the command add.
 - The add command adds a file that is in the workspace to this index.
 - After that the file is staged, i.e., it's marked to be committed, but not yet committed.
 - If you specify the minus u option, you will also consider deleted files
- If you add the file, it just gets added to this index but is not actually committed, so what you need to do, is to commit the file

- When you execute git commit, for all the files that are staged, their changes will be committed to the local repository.
 - Your files can be in three states. They will go from the modified state to the stage state when you execute the add. And then from the stage state to the committed state when you perform a GIT Commit.
- At this point your changes are safely stored in the local repository.
 - You can also perform these two steps at once by executing a Commit -a.
 - If you have a set of modified files, and all these files are already part of the repository, so they're already known to the VCS, you can simply execute a commit minus A.
 - This will stage your file and then commit them all at once.
 - This will not work if the file is a new file. Then you have to manually add it.
- As we discussed when we looked at the difference between centralized and decentralized Version control system, we saw that in the case of the decentralized, there is a local repository and you have to explicitly push your changes to a remote repository, and this is exactly what the git push command does.
 - It pushes your changes Data into local repository to the remote repository so at this point all of your changes will be visible to anyone who has access to the remote repository.
- Now, let's see the opposite flow, that is when you're actually getting files from the repository instead of committing files to the repository.
- What the fetch command does is get files from the remote repositories to your local repository but not yet to your working directory.
 - This operation is useful
 - What this means is that you will not see these files in your workspace. You will still have your local files here.
 - So this is sort of a physical distinction.
- In order to get your data files from the local repositories to your workspace you have to issue another command which is the command git merge.
 - Git merge will take the changes in local repository and get them to your local workspace.
 - At this point your files will be updated to what is in the remote repository or at least what was in the remote repository at the time of the fetch.
- There's a shortcut which is the command git pull in case you want to get the changes directly to your work space with a single command,
 - The changes will get collected from the remote repository and they will go to your local repository and to your work space, at once.
 - Has the same effect as performing a git fetch and then a git merge.
- So if we can do everything in one command, why, why we want to fetch and merge as two separate operations?
 - This allows us to compare files before we actually get the latest version of the files.

- In particular, I can run the command `git diff HEAD` to get the difference between my local files (the files in my working directory), and the files in my local repository.
 - And based on the differences decide whether I want to merge or not.
- You can also run `git diff` without further specifying head.
 - This command tells you the difference between the files that you have in your work space and the ones that are staged for a commit.
 - So basically what it will be telling you is what you could still add to the stage for the further commit, and that you haven't already.
 - So what local changes will not make it to the next commit
 - This can be used as a sanity check before doing a commit to make sure all the local changes that you have, and that you want to commit, are actually staged and therefore will be considered.

Intro to Git Demo

- First Part: the basics of git.
 - How to introduce yourself to git
 - How to create a repository
 - How to commit changes and get changes from the repository, and so on.
- So after you installed git you should have the git tool available on the command line
 - If you just execute `git` you will get the usage information for git, with the most commonly used git commands.
 - To find information on any command, you can simply type: `git help`, and the name of the command.
 - Example: try to write `git help init` and it will bring up the git manual page for `git init`, which describes the command, the synopsis, and so on.
- The first thing we need to do is introduce ourselves to git
 - To do that we use the "`git config`" command, in particular we are going to write to the `git config` minus, minus, global user dot name (which means we are telling it our user name) and specify our name in quotes
 - You could also provide your email address in the same way.
 - Still use the `git config --global` command. But in this case you will write `user.email` as the property and then you'll specify a suitable email address. In this case, the email address of George P. Burdell.

```
rm          Remove files from the working tree and from the index
show        Show various types of objects
status      Show the working tree status
tag         Create, list, delete or verify a tag object signed with GPG

See 'git help <command>' for more information on a specific command.
$ git help init
$ git config --global user.name "George P. Burdell"
$ git config --global user.email "gpbud@gate
```

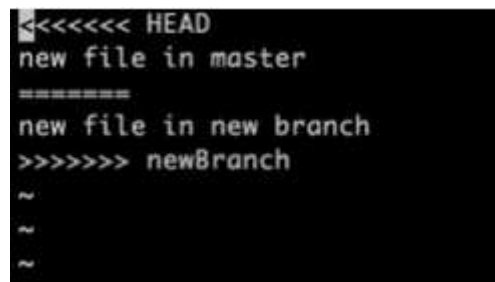
- We'll now look at some commonly used commands that to create and maintain a local repository.
- Let's first create a new project and call it my project. So, to do that we are simply going to create a directory (`mkdir myproject`) and then we're going to move into that directory (`cd myproject`).
- If we try to call the "git status" command at this point to see what's the state of my project, git doesn't know anything about this project so you will get an error. It will tell you that, basically, we're not in a git repository.
- Make a git repository by calling `Git init` and the output will tell you that the repository was initialized.
- If we check the status again, you will see that now Git recognizes the repository and will tell you that there is nothing to commit because, of course, the repository is completely empty.
- Let's just create a new, empty file which we're going to call README (`touch README`).
- Now if you run "git status", git will tell you there is a file that's called README, but it's untracked.
 - What that means is that the file not staged
 - We need to tell git that this needs to be considered by calling the "git add" command and then we specify README as the argument for the command.
- Again call "git status" and Git knows that there is a new file called README, because the file is staged.
- To commit a file, we simply execute "git commit", which will open a text editor, which can be different, depending on what is your environment, and here we need to add a comment to be added to the commit.
 - Here we simply write in Added README file, then we can close and save (hit ESC and then type `":wq"`) and this will add the file to the local Git repository.
 - At this point, if we ran Git status again to see where we are. You can see that Git tells you that there is nothing to commit.
- Now, let's make some changes to our README file.
 - Add text
 - VIM README
 - Type "README file content"
 - ESC, `:wq`
- Now run git status and git will know that README file has been modified.
 - Remember that before, it was telling you that it was a new file but now it knows that there was a different version in the repository.
 - At this point we can check the differences between this file and the committed one by executing "git diff readme"
 - If you look at the output of the get diff command here, you can see that this line, readme file content was added and you'll see that it was added because there's a plus sign before that line.
 - In case of deletion of lines, you'll see a minus sign there.

- At this point, if we want to commit our file, remember that we'll always have to tell git that we want to stage the file before committing it. Otherwise, it will be ignored by the commit operation.
- To tell git that the file has to be staged we can use the usual "git add" command but we can also use a shortcut with simply "git commit -a," and this will tell git to commit all of the files that git knows about, which in this case is only the written file of course.
- We can also right away provide a message for the commit without having to open an editor
 - To do that we can specify the -m option. And at this point we can just put our content in double quotes and press enter
 - "git commit -a -m "Added README content"
 - Press enter and as you can see it will notify us that one file was changed and in particular it will also tell you that there was an insertion
- Run git status and you will see that there is nothing else to commit.
- Imagine that you want to see the version history for your repository. You can do that by running the git log command and it will show you all the different commits for your repository
 - Each commit has a commit ID and will show you the comments associated with the commit
- In case you want to see the changes introduced by a commit you can use that git show command, and you can provide the commit ID for the commit that you're interested in.
 - You don't really need to provide the whole ID just the first four or more characters.
 - Execute the command it will show us the changes introduced by that commit.
- To fetch a report from a remote server, you can use the git clone command.
 - Write "git clone" and then specify the URL for the remote repository
 - Here we are using the SSH protocol and there are different protocols that can be used, so the remote repository can be made available in different ways.
 - When you clone the project, the project is cloned into the local directory.
 - If you wanted to import the project under a different name you could just specify the name that you want for the local directory as an argument
 - git clone [gpdub@bear.cc.gatech.edu:~/git/myproject.git](https://gpdub@bear.cc.gatech.edu/~git/myproject.git) myproject2
 - Then you'll get the project in your local work space with the name that you specified, here myproject2 (type "ls" to see the contents of your work space")
- So, let's go inside one of these two projects that have the same content because they're coming from the repository (cd myproject).
- If you want to see the details of the server you can use the git remote command and specify the flag -v.
- Let's go ahead to make some changes to the project; let's add a file
 - Create an empty file and call it new file (touch newfile) and add it to your index so that it gets committed.

- Then run `git commit` to actually commit it to the local repository and specify the comment from the command line.
- So the file gets added to my local repository and to double check we can run `git log`.
- If you look at the last commit at the top, you can see that it's saying new file was added to the repository, showing the comment that was added.
 - But this is just for the local repository
 - Need to use the `git push` command to push it to the remote repository.
 - When you run that the local changes will be committed to the remote repository.
- Now let's go to the other copy of the project that we created, the one under directory `myproject2`.
 - This project was linked up to the same remote project but if we run `"git log"` here, we don't see the latest change that we made, because we didn't synchronize this local copy with the remote copy. And so we just have the README and the files that were there before
 - What we need to do is pull the changes from the remote repository using `"git pull"`, and when we do that, that will actually pull these changes and therefore, create the new files that we created in the other directory.
 - If we run `git log` now, you can see that now we have the new entry. The comment at the top, that says this new file was added
 - The normal user scenario for this will be that each user will have their local copy, work on some local file, commit them and push them to a remote repository where others can get changes, do further changes, and push them as well so on and so forth.
- Now let's look at some more advanced concepts of branching and merging.
- Branching basically means make a copy, to create a branch of the current project, so that we can work on that copy independently from the other copy (from the other branch)
 - Then we can decide whether we want to keep both branches or we want to merge them at some point.
 - You can of course have multiple branches, not just two.
 - This is particularly useful because if you think about the way we develop software in general, we work with artifacts. We might have the need to create kind of a separate copy of your work space...to do some experiments for example.
 - You want to change something in the code, you're not really sure it's going to work out and you don't want to touch your main copy.
 - So that's the perfect application for branching.
 - If you're happy with the changes, you will merge that branch with the original one,
 - If you're not happy with the changes you will just throw away that branch.

- Let's see how that can be done with git.
- First of all if you want to see which branches are currently present in your project, you can simply execute "git branch", and in this case, you can see that there's only one branch, which is called master, and the star indicates that this is our current branch.
- To create a new branch we simply run the command "git branch" and specify a name for the new branch as an argument
 - git branch newBranch
- If we run git branch now we will have a new branch plus master will still be our current branch.
- If you want to switch to the new branch, use the "git checkout" command and specify the name of the branch that we want to become our current branch.
 - git checkout newBranch
 - git will tell us that we switched to the new branch.
- If we run "git branch" you will see that now the star is next to "newBranch" because that's our current branch.
- There is a shortcut for these two commands.
 - If you run the command "git checkout" specify the minus b flag and then the name of the new branch, it will do both things at the same time.
 - git checkout -b testing
 - It will create the new branch called testing in this case, and then it will switch to new branch and then it will tell you this after executing the command.
- Now if we look at the GIT branch output, you can see that there are three branches and we are currently on the testing branch.
- So now let's create a new file
 - Vim testFile
 - Add content "This is a test" and save it
 - Add it and commit it.
 - and just call it test file, put some content in there, save it, we edit and commit it.
 - Type "ls" and now in this current branch, we have our testFile.
- Now go back to the master branch using the usual "git checkout command."
 - If we do an ls, we can see that the testFile is not there, because of course, it's not in this branch.
 - Let's assume that we are happy with the testFile that we created, with the modification that we made on the branch, and so we want to merge the Dev branch with our master branch.
 - To do that we can call the git merge command and we'll specify the branch that we want to merge with the current one.
 - git merge testing
 - That will merge the testing branch with the current branch, which is the master, which means that now the testFile is in my current working directory

- Now let's assume that we want to delete the testing branch at this point because we don't need it anymore.
 - Simply execute the branch minus d which stands for minus delete, specify the name of the branch and this will eliminate that branch.
 - `git branch -d testing`
 - As confirmed by running the command `git branch` or the testing branch no longer shows up.
- So, something that might happen when you merge a branch is, is that you might have conflicts such as changing the same file in two different branches.
- Let's see an example of that.
 - Check which branches we have with "`git branch`" and we'll have two branches, master and newBranch
 - Our current branch is master.
 - Open a file called new file and add some content there. So now let's commit this changes so they get to the local repository.
 - Now let's reach to the other branch and by running `git checkout` and the name of the branch.
 - Do the same operation (create newfile) here.
 - And at this point we do the same operation here
 - At this point, what we have here is this file called newfile that has been defined independently both in the master branch and in the newBranch.
 - So we have a conflict.
 - Switch back to the master branch.
 - Now, let's say we want to merge the two branches. So since we are in master, we want to say that when I merge the new branch into the current one.
 - `git merge newBranch`
 - When we run that, we get an auto merging conflict.
 - At this point we can manually fix the conflict by opening the file that was showing the conflict
 - Here you can see the kind of information that you get in the conflicted file.



```

<<<<<<< HEAD
new file in master
=====
new file in new branch
>>>>>>> newBranch
~
~
~
  
```

- It's telling you that in the Head, the master we have the content "newfile in master" and under the separator the content added to newBranch.
- Since we only have one line, basically the whole file is two versions and you can decide which version you want to keep.
- Let's assume that we want to keep the content from the master.

- So we eliminate the annotations and eliminate the additional content.
- We save this file and commit the modified file in the normal way
- git add newfile and specify in the comment for clarity that this is the merged file, so that we performed a merge.
- And at this point we are done with our merge.



Git + Eclipse Demo

- Many of these version control systems are actually integrated into IDE's.
- If we put together git and eclipse the result is Egit
- EGit is a plug in for the eclipse IDE that adds git functionality to eclipse.
- If you want to get a github for Eclipse, you should go to eclipse.github.com and you can download the plugin.
- Use the provided URL and directions to install the plugin.
- In this case we're going to copy this address.
 - Go to Eclipse, Help, Install new software.
 - Click on "Add" to add a new site from which to get software.
 - We paste the location that we just copied here and we can give it a descriptive name (eclipse git plugin).
 - Then when you click okay, Eclipse will go and look for plugins.
 - And as you can see, there are two options. We can select both of them, and click on next.
 - You can see that the Eclipse identified a few dependencies.
 - You can click next and accept them.
 - Accept the terms and conditions for the plug in, and then just finish.
 - Then Eclipse will install the plugin, which might take a little bit of time.
 - When Eclipse is done, you will get this prompt that will tell you that you need to restart Eclipse for the plugin to be actually installed.
 - Click yes and when Eclipse restarts, you'll have your plugin.
- Now go to the git repository perspective and when we click OK, you can see that the display will change.
 - Since we don't have any repository yet, we are provided with the possibility of adding an existing local git repository, cloning a git repository or creating a new local git repository.

- Add an existing local repository. Pick the one created earlier and click finish, and you can see that my project is now added to this set of git repositories.
- Now let's check out the project from the repository by selecting import project.
 - You can import something as an existing project
 - You can use a new project wizard
 - Choose the option of importing as a general project and click next.
 - You will see the project name and can click Finish.
 - Now, if you go to the resource perspective, you can see that the project has been added to your set of projects.
 - You can see all the files within the project
 - One thing you can do now is execute different git commands/perform different git operations by using the team sub menu in the contextual menu.
 - There are several things you can do including some advanced commands.
 - Click show local history, and this shows the history of the file such as author and when it was created
- Let's make some changes to this file by adding some new content.
 - Save the file and you'll see an arrow that indicates that your file was locally changed.
 - Now go to the team menu and you'll have the option to add to the index, to stage the file.
 - And now I got this new label with a star that shows the file was added to the index.
 - Go to the team menu again and commit the file by selecting the corresponding entry. It allows you to enter the commit message exactly in the same way as the command line with the textual editor.
 - Now if you look at the history view, we can see here that we have a new version for the file that we just modified. And we can also see the commit comment.
 - And, at this point, if we had remote repository we could push our changes to that remote repository as well, again, using the team sub manual and the contextual manual.
 - And, speaking of remote repositories, what we are going to see next is how to use GitHub repositories which are remote repositories that are hosted on GitHub.

Github Demo

- GitHub is a Git hosting website
- We will be using GitHub as our git hosting and give you GitHub repositories that you can use for your projects.
- Let's see some of the common features offered by GitHub.
- (Create an account and) sign in to see what kind of functionality GitHub offers.
- On the GitHub website, you can use the menu up on the right to create a new repository, or change the account settings.

- Click on the user profile, and you can see some of the statistics for the user such as contributions and repositories
- If we go to the Repositories view, we can create a new repository.
 - We can give it a name.
 - Let's call it My Repo.
 - We can provide a description for the repository.
 - If we want, we can initialize the repository by adding a README file.
 - You can also add a license, here on the right and it allows you to choose from a set of pre-defined licenses.
 - You can also add a .gitignore file.
 - It's a very convenient file that will automatically exclude from the repository, those files that should not be added.
 - There are things that you should not add to the repositories, such as write files.
 - So here using this menu you can take the type of project that you have including java, PHP, or many other kinds of projects and GitHub will automatically add that file for you.
- Create the repository and it will create a repository that contains the readme file if that's what you decided to do
 - It also allows you to edit the readme file by clicking on it and it will bring up an editor.
 - Then you can add a comment and commit the changes to your readme file.
- The site also provides many other features like
 - Creating an issue
 - Poll requests
 - Adding and editing a Wiki
 - Defining other characteristics and settings for the repository.
- If you go to the repository, you can see that we also get the HTTPS link for the repository.
 - This is the URL that you can use to clone your repository.
 - Try it.
 - Execute git clone and specify the URL
 - The project created was cloned locally, and if you go under myrepo, which is the name of the repository, you can see that the readme file that we created on github is there.
 - Create a new file and call it newFile
 - Add it, commit it, and specify a commit message.
 - Push the local changes to the remote GitHub repository, and because the GitHub repository is password protected, you have to specify your login and password.
 - If you pass the wrong password, github is not going to let you in.
 - With correct credentials the push is successful and the changes are actually pushed to the master, which is the GitHub repository.

- To double check that, go back to the GitHub repository and you'll see newFile is there as expected.
- There's many more things that you can do on the GitHub website.
- Key message here is that GitHub is a git hosting website, where you can get an account and create your remote repositories.

LOCAL REPOSITORIES

Create
 mkdir testproject
 cd testproject
 git init

Modify
 git add foo.txt (or git add .)
 git commit
 git mv
 git rm

Inspect
 git log
 git status
 git diff
 git show (last commit)

A possible workflow
 (This is just an example!)

- some editing
- git status
to see what files you changed
- git diff [files]
to see the changes
- git commit -a [-m <message>]

Recap Local Git Repositories

- Commands that create
 - Notice that not all of these are git commands
 - For example, to create the repository, we would normally want to create a directory
 - We want to go to that directory and then execute the git init statement, which initializes that directory as a git repository.
- Commands that we'll use to modify the content of the repository.
 - We can use git add to add a specific file or a complete directory to our index or the list of files that will be committed and will be considered in the next commit.
 - Then we can use commit to actually commit the changes that we made to those files to our local repository
 - We can also use git move (mv) and git remove (rm) to move files around and to remove files.
- Commands that we can use to inspect the concrete repository.
 - Use git log to see the log of the repository
 - Git status can give us important information about the status of the file center repository.
 - Git diff can be used to see the differences between our local files and the remote files.
 - Git show will show us information about our last commit

- What we committed
 - What were the changes and so on
- You can do many different things and can have many different workflows with git.
 - So, you might do some local editing. Execute git status to see what files you changed.
 - Then you might run a git diff on the files to see what are these changes.
 - And then you can run git commit minus a to commit your changes.
 - And in case you want to specify the commit message right away without having to go through an editor, you can also add the minus m parameter and specify the message here on the same line.

REMOTE REPOSITORIES

Copy repository

- git clone <repository>
 - repository = URL (file, http, ssh, ...)
 - creates a complete local copy of the repository
 - links it to the remote repository (origin)
(you can also link to <repository> later)

Receive changes

- git pull

Send changes

- git push

Recap Remote Git Repositories

- The command to copy a repository is git clone in which you get a remote repository and you make a local copy in your working directory.
 - The repository can be specified as the following:
 - A URL
 - A local file
 - Using the http or the ssh protocol
 - And there's also other ways to do it.
 - This creates a complete local copy of the repository and links it to the remote repository, which is what is called the origin.
 - And if you want, you could also actually link to the repository later.
- Then the normal way of receiving changes from a repository is to perform a git pull command

- We saw that you can also perform the same operation through two commands, `git fetch` and `git merge` in case you want to inspect the changes before actually merging them
- If you want to send changes that you have in your local repository to a remote repository, you will use the `git push` command.