常用命令:
git config --global user.name Tao Peng
git config --global user.email "tpeng38@gatech.edu"

mkdir myproject
cd myproject
git status
git init
touch README (即新建 README 這個文件)

GIT_SSL_NO_VERIFY=true git clone https://github.gatech.edu/gt-omscs-softeng/6300Spring16tpeng....git
用以下這句是不是就可以設置為以後不必寫 GIT_SSL_NO_VERIFY=true 了?
git config http.sslVerify false

git add filename (add 後也可以跟一個文件夾或 .)
git commit -m “comment”

git diff README
git log
git show 27c0

git remote -v

GIT_SSL_NO_VERIFY=true git push
或 GIT_SSL_NO_VERIFY=true git push --all
(詳見下頁的 About git push)

如果在 push 之前, 已經有其它人改動了 master, 則在 push 之前, 要先 pull:
GIT_SSL_NO_VERIFY=true git pull
或 GIT_SSL_NO_VERIFY=true git pull origin master

git branch (看有哪些 branch)
git branch newBranch (新建一個 branch)
git checkout newBranch (將 newBranch 設為 current branch)
git checkout -b testing (新建一個叫 testing 的 branch, 並將其設為 current branch)
注意現在已經在 testing branch 裡了

git checkout master
git merge testing
用 git branch -d testing 即可刪除 testing 這個 branch.

git tag Version1
GIT_SSL_NO_VERIFY=true git push repo Version1

Submitting the commit ID for your submission on T-Square. You can get your commit ID by running
git log -1

Remove file:

```
git rm file1.txt
git commit -m "remove file1.txt"
git push origin branch_name
```

Remove folder:

```
git rm -r one-of-the-directories
git commit -m "Remove duplicated directory"
git push origin master
```

About git push:

From stackoverflow:

git push assumes that you already have a remote repository defined for that branch. In this case, the default remote origin is used.

git push origin master indicates that you are pushing to a *specific* remote, in this case, origin. This would only matter if you created multiple remote repositories in your code base. If you're only committing to one remote repository (in this case, *just* your GitHub repository), then there isn't any difference between the two.

From another stackoverflow:

The default action of git push and git push origin has changed since git version 1.7.11:

- •Before 1.7.11, git push by default pushes all branches that also exist remotely with the same name.
- •Since 1.7.11, git push by default pushes the current branch to a remote branch with the same name.

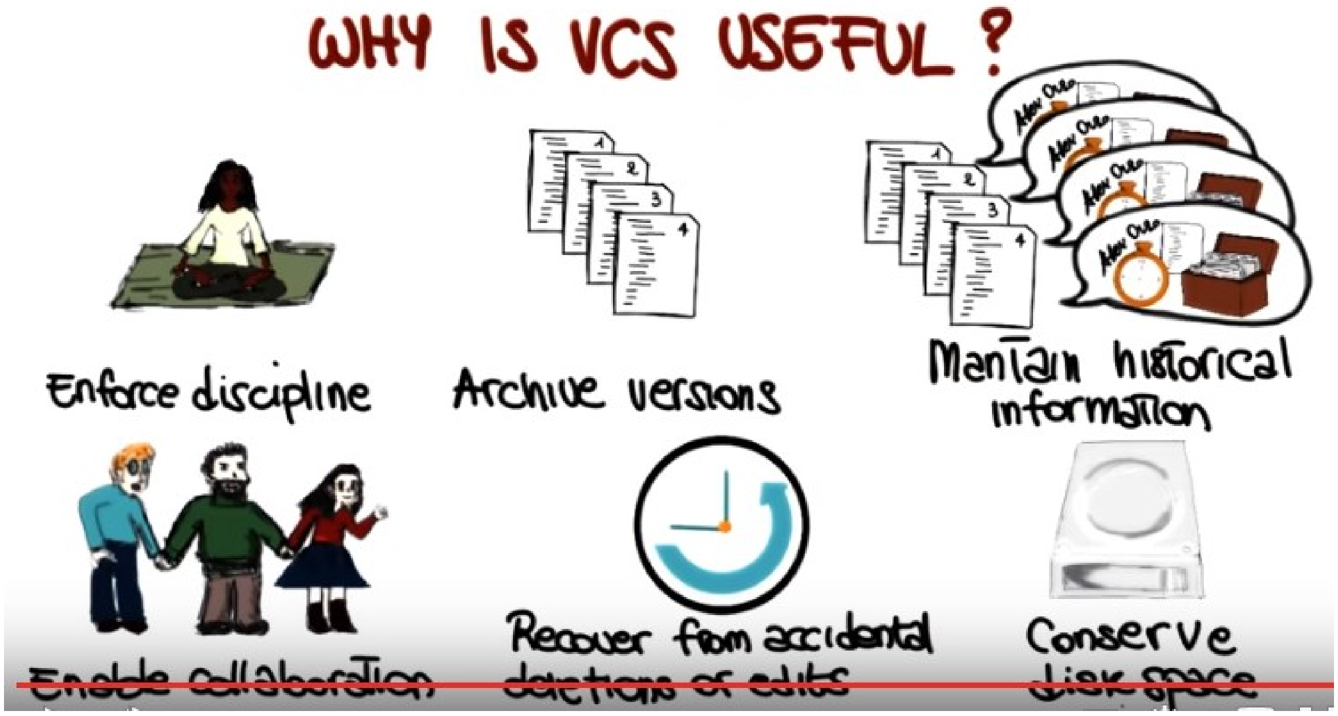
Before and after version 1.7.11, the default behavior can be configured with the push.defaultconfiguration option. This configuration option has been introduced in git version 1.6.3.

1. Hi and welcome to the second lesson on tools of the trade. In the previous lesson we talked about IDs. Integrated Development envrionments and in particular we discussed the eclipse ID. Today we're going to talk about another fundamental type of tools in the software engineering arena. Version control systems. And these are also called, revision or source control systems. In particular, we will focus on a specific version control system called git (g 唸哥, 不唸吉). And as we did for eclipse, we will first present git from a conceptual standpoint. And then we will do a demo. To get some hands-on experience with GIT.

2. >> And I thought that the best way to break the ice on version control systems and Git and some other related concepts was to interview John Britton who works with GitHub. So let's go and see what John has to say about Git, about version control systems in general, and about GitHub. John is in Tapei, [INAUDIBLE]. >> That's correct. >> Okay so we're, you know we couldn't go there so we're interviewing him remotely. And I want, I just want to thank you so much and John for agreeing to talk

to us. >> Thank you very much for having me it was my pleasure. >> And, I'm just going to ask, a few general questions because John is an expert on, Git and GitHub. John is a developer and a community builder is active in both the open source and the open education areas. And as an educational liaison we have, is working to improve Computer Science education by bringing the principles of open source into the classroom. And I'm going to start with an original question, which is what is a version control system? >> So, a version control system is a tool that software developers use. Anybody who's doing you know, working with digital assets Digital projects can also use for keeping track of, you know, revisions of your project, and when I say revises, I mean essentially snapshots of your project over time. So you can image 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. >> Why is this useful? I understand that it is nice to take a snapshot of your project, but what did you do with the snapshot afterwards? I think the most immediately obvious benefit to having snapshots of your project to keeping revisions is that you can go back. If you have ever worked on a project and got to a point where you solved a bunch of your problems, and there is just one more step to do. And you start working on trying to solve that last step, and you break things, you make it worse then it was an hour ago. At that point its easier to just go back to what you had then trying to figure out what you broke. So you can always go back in time, and the other big one is being able to collaborate with multiple people, so its pretty seldom these days that you. Work on a production totally on your own. It's most common to work in, you know, in teams and small groups. And so, 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. >> Alright, that's great, because those are exactly some of the topics that we're going to. Cover in the lesson and so since we're going to talk about the specifics of version control system which is Git and you're definitely an expert in, in Git. So what would you say is specifically special about Git? What characterizes it and And how does it compare to other version control systems. >> So if any of you have used version control systems before, you may have heard of something like subversion, CVS, or maybe a commercial solution like ProForce. I think the main important characteristics of Git are first that it's open source. And the second, that it's a distributed version control system. So what that means, the distributed version control system is essentially a system for tracking revisions of your software that doesn't have any central repository. So the 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 Central system. >> Okay, great. And so now that we saw what Git is, what is GitHub and how does it fit into this picture of the distributed, revision control system? >> So 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. There's two ways you can use GitHub. You can use it publicly for open source and you can use it in private within your team, or your company, or within your class. And, Git Hub started out just as a way to host your, your Git repositories. But it's actually grown into quite a bit more. It's an entire collaboration system around, around your code. >> How many users do you have? >> I would say that we're approaching five million. I don't know the exact number. We're definitely more than four million right now. But yeah, I'd say somewhere, somewhere close to between four and five million. >> So that's a lot space I,'d guess. Terabytes of disk space, I would imagine. >> There are a lot of GIT repositories on, on our servers. >> Something else you want to say? I guess that the when taking about GitHub there's one thing that you kind of can't leave out and that's that's a feature that's called a polar quest. So when you're using GitHub, you can share your Gits repository, do some work, and actually do do a code review. Of proposed changes which is what we call a polar quest on github.com. Essentially what it lets you do is 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 Totally integrate some proposed changes. You have, kind of a conversation about what your code. And I think that's really valuable when you are working in a team. >> Thank you, John, that was very informative and thanks again for taking the time to talk to us. >> No problem, thanks for having me. I'll talk to you soon. >> Let's thank again John for enlightening us on some aspects of version control systems, Git and GitHub. And now, let's go over some of the topics that we discussed with John to recap them.

3. So first of all, what is a version control system? A version control system or VCS, is a system that allows you to manage multiple revisions of the same unit of information. For example of documents, of 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. Now, let's drill into this concept in a little more detail. And let's do that by discussing why is VCS useful, especially in the context of software engineering and of software development. So first of all, using a version control system enforces discipline, because it manages the process by which the control of items passes from one person to another. Another important aspect of VCS is that it allows you for archiving versions. So you can store subsequent versions of source controlled items into a VCS. And not only you can store versions, you can also maintain a lot of interesting and important historical information about these versions. For example, a VCL will store information such as, who is the author for this specific version stored in the system. Or, for another example, on what day and what time that version was stored. And a lot of other interesting information about the specific version of the item. Information that you can then retrieve and for example, use to compare versions. Obviously, the fact of 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. And I'm sure that you all had the experience of deleting a file by mistake or modifying a file in the wrong way, or in the most common case of changing something in your code for instance. And breaking something and not being able to go back to a version that was working. Not remembering, for example, what is that you changed that broke the code. In all these cases a version control system can be extremely useful because it will allow you to recover from this accidental deletions or edits. And for example, to go back of yesterdays version that was working perfectly, and also to compare, for example, yesterdays version with today version and see what is that you changed. Finally, a version control system will normally also allow you to conserve and save disk space on both the source control client and on the server. Why? Well, for instance because it's centralizing the management of the version. So 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. In addition, version control system often uses efficient algorithms to store these changes. And therefore, you can keep many versions without taking up too much space.



4. Now before we continue, and we look at more details of version control systems, I want to ask you a quick question about VCS. I want to know whether you have used a version control system before, and if so, which one or which ones. I'm going to list in here some of the most commonly used version control systems, like CVS, Subversion, GIT, and I'm also allowing you to specify other VCS in case you have used different ones.

5. And of course there's no right answer for this. I just wanted to collect some statistics. To see what kind of previous experience you have with this kind of systems.

6. What I want to do next, is to look at how version control systems actually work. We saw what they are. We saw why they are useful. But how do they actually work? And we're going to do that by starting from some essential actions that version control systems perform. The first one is the addition of files. So, when you use a version control system, you can add a file to the repository. And 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, 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 then to commit your changes to the central repositories, so they can become visible to all of the other users, other repositories. Finally, 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. And these are just three of the basic actions, but there are many, many more. And we'll see several of those.

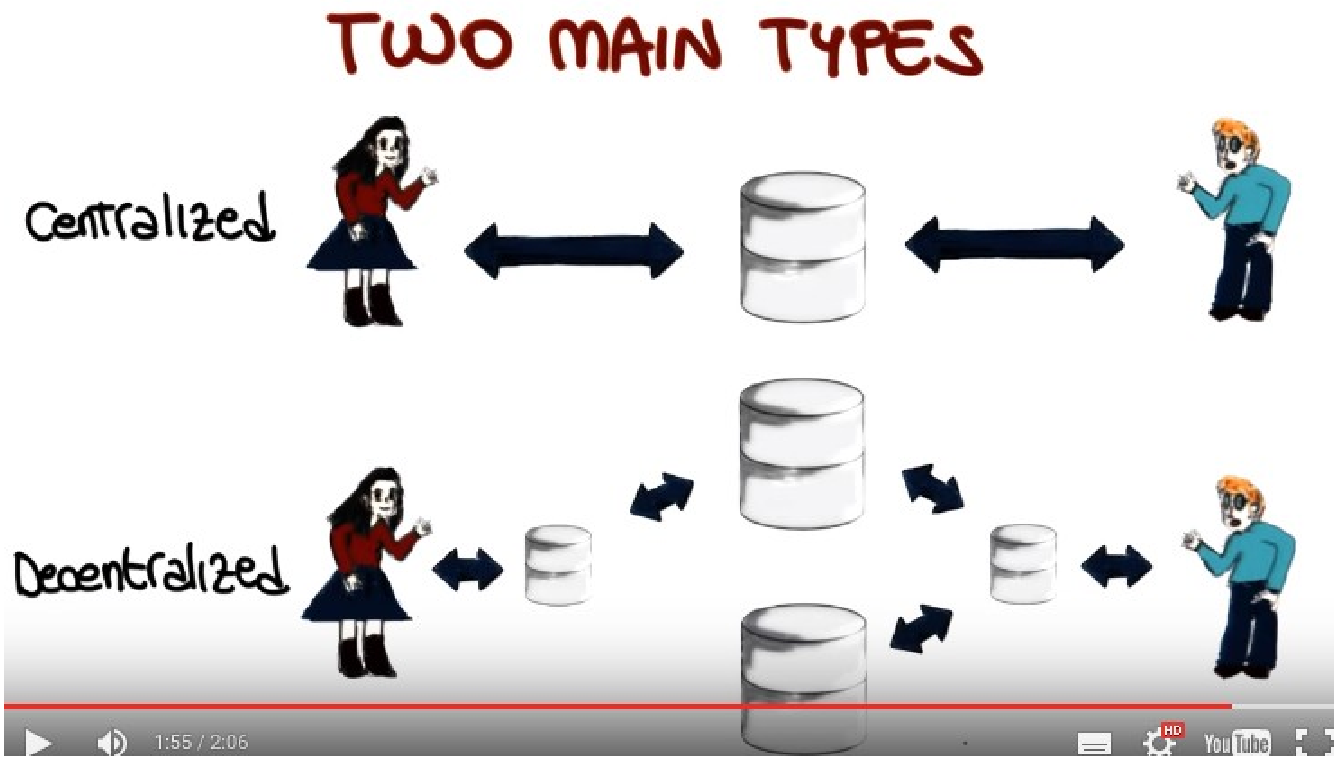
7. Before looking at additional actions, though, I would like to see what is the basic workflow in a version control system using the three actions that we just saw. And to do that I'm going to use two of our friends, Brad and Janet. So we have Janet here, Brad, and a VCS that they are using. Now imagine that 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. And when she adds and commit, that's exactly what will happen, in foo will be come available here, and will be accessible to the other users. In this case it'll be accessible to Brad. If Brett 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. To do that, he will commit the file and the result will be exactly the same of when Janet committed her file. That the updated file will be sent to the repository and the repository will store that information and make it available for other users. So now, if Janet performs an update, she will get the new version of foo.txt with the additional information that was added by Brad. And we will see all of this in action in our next demo in a few minutes.

8. Before getting to the demo, I want to say a few more things. In particular, I discuss the main don'ts in VCS. So, what are some things that you don't want to do, and you should not do, when you're using a version control system? And I'm going to mention two, in particular, because these are two that I witnessed several times when I was teaching this class and also when collaborating with other people. So, 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. So 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. There might be exceptions to these rules, but 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, and that happens all the time, especially to novice users of VCS. Is that 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, no, no 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 ever do that. I have seen that done many times, and it always leads to disasters. First of all it is useless, and second it's risky. Because then what happens is that at the time in which you have to turn in your assignment, in the case you are doing an assignment, but even in more serious situation, when you have to turn in your code, for example to your colleagues. 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. So again, try the VCS.

"DON'TS" IN VCS
EXE
2GB
VCS file
local version file

9. Something else I want to mention is that there are many different version control systems but can classify them normally in two main types centralized VCS's and decentralized VCS's. So what is the difference between these two, let's use a game out friends Janet. And Brett. In the case of a centralized version control system there is a single centralized, as the name says, repository. On which they are commiting their files. So 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. So 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 when they're ready to release the version, they can push it to a central repository. And 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. I'm just going to mention a few, because there are really many. One is the fact of having this local version. If you used VCS before, I'm sure you've been in the situation in which 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, to be looked up. 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. Another big advantage, is that you can use multiple remote repository. In fact, centralized is not the right name for this one. This is just a remote repository, and I can have more than one. For example, Brad might want to push to another remote repository. As well. For instance, 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.

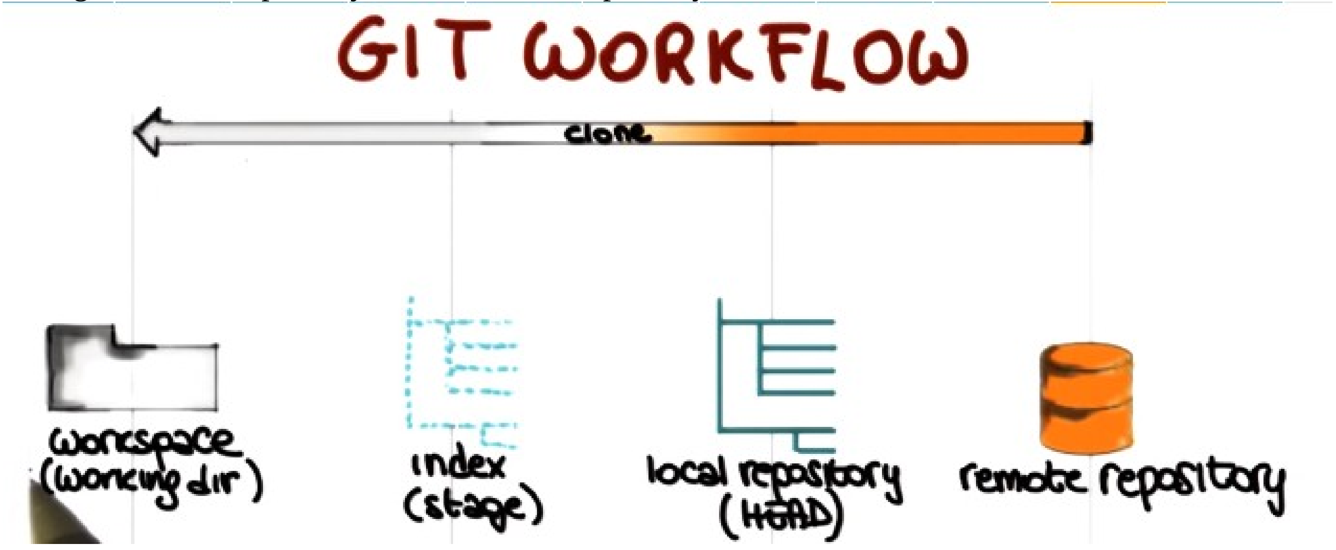


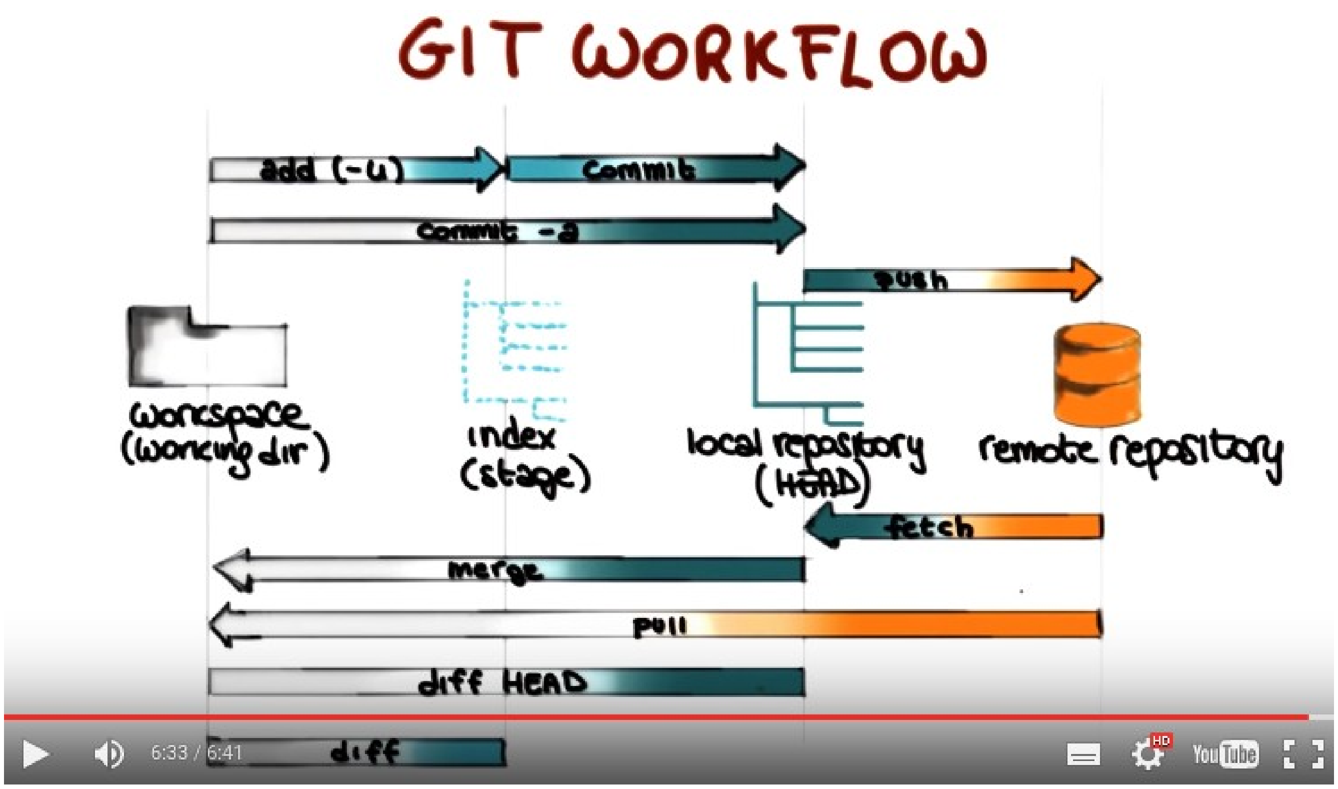
10. One good representative of distributed version control systems, is GIT. A distributive version control system that was initially designed and developed by Linus Torvalds. I'm pretty sure you know who Linus Torvalds is. He's basically this guy who started and created the Linux operating system. And Linux was unhappy with the existing version control systems, and wanted a different one. He wanted to use it for maintaining the Linux kernel. In particular, he wanted one with some kicker characteristics. For example, the fact that it was distributed. He wanted it to be fast. He wanted it to have a simple design. And he wanted to have a strong support for parallel branches, because many people were contributing to the kernel at the same time. And therefore there many different branches of development. And finally, he wanted for the virtual control system to be able to handle large projects. As the Linux kernel is, and to do it in an efficient way. So if you want to get an idea of how popular GIT is today, 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. So the, it had a 30% adoption rate. So we will use a GIT as a version control system for the class.

11. As we did for our clips, and IDs in general, we want to start a GIT in a hands on way. So we're going to start by seeing how to install GIT. And GIT is also multiplatform, so you can install it no matter what operating system you are using, unless of course you are using some arcane operating system. But if you are using Linux, for instance, there should be a package available that can install GIT for your specific distribution. If you're using Mac OS, GIT is also available as part of XCode and also as an independent package. Finally, if you're using Windows, GIT is available as a package with an installer. In general, you can go here to get information about how to get GIT, where to download it, how to install it, and so on. So, now what I'd like for you to do is to go, get GIT, install it, in case you don't have it installed already on your machine. And after that, you should be able to run GIT from the command line. And, that's exactly what we're going to do through a demo.

12. But before jumping into the demo I would like to give a high level overview of the GIT workflow,

which will help you better, following the demo. So let me start by representing four fundamental elements in the GIT workflow which are these four; the workspace which is your local directory. The index, also called the stage, and we'll see in a minute what the index is. Then, we have the local repository. We'll also refer to this as HEAD in the, when we explain the different commands and then, the word flow. And finally, the remote repository. If you consider a file in your work space it can be in three possible states. It can be committed which means that the data, the latest changes to the file are safely stored here. It could be modified, which is the case of the file being changed and no, none of these changes being saved to the local repository so locally modified or it can be staged. And stage means that the file is basically part of this index. And what that means, that it's been tagged to be considered in the next commit. And I know that this is not all 100% intuitive, so let's look at that again by considering the actual workflow and let's see what happens when you issue the different commands in git. So the first command that you normally run in case you, you're getting access to a remote repository, is the git clone command. And the git clone, followed by the url for that repository, will create a local copy of the repository in your workspace. And of course, you don't have to do this step if you're creating the repository yourself. The next command that we already saw is the command add. And what the command add does is to add a file that is in the workspace to this index. And we say that after that, the file is staged. So it's marked to be committed, but not yet committed. And here I'm just mentioning this minus u option. If you specify the minus u option, you will also consider deleted files File, but let's not get there for now, we'll talk about that when we do the demo. As I said, 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, so when you execute git commit, all the files that are staged, that are released it here, their changes will be committed to the local repository. So your files, as I was saying, they can be in three states. They will go from the modified state to the stage state when you execute the app. And then from the stage state to the committed state when you perform a GIT Commit. Okay, so at this point your changes are safely stored in the local repository. Notice that you can also perform these two steps at once by executing a Commit -8. So if you have a set of modified files, and all these files are already part of the repository, so they're already known to diversion control system, you can simple execute a commit minus A. And what the commit minus A command will do, it will stage your file and then commit them. All at once. So it's a convenient shortcut. Of course, as I said, this will not work if the file is in your file. So if a file is your file, you have to manually add it. Otherwise commit minus A will just stage and commit at once. As with this [UNKNOWN] when we looked at the diffence between centralized and decentralized Version console system. We saw that in the case of the decentralized, there is a locker repository which is this one. And then you have to explicitly push your changes to a remote repository, and this is exactly what the git 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 so how does it work when you're actually getting files from the repository instead of committing files to the repository. So the first command I want to mention is the get fetch command and what the get fetch command does is to get files from the remote repositories to your local repository... But not yet to your working directory. And we will see what is the usefullness of doing this operation. Of having the files all in the local respository, but not in your local directory. So, what that means, just to make sure that we're on the same page. Is that you will not see these files when you 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. So at this point your files will be updated. To what is in the remote reposity. Or at least what was in the remote repositoyore at the time of the fetch. SImilarly to what happened for the add and commit. There's a shortcut which is the command, get, pull. So in case you want to get the changes directly. To your work space with a single command, you can issue a git pull command and what will happen, is that the changes will get collected from the remote repository and they will go to your locker repository and to your work space, at once. So this has the same affect as performing a git fetch and a git merge. So if we can do everything in one command, why, why we want to fetch and berch as two separate operations? So one of the reason is because this allows us to compare files before we actually get the latest version of the files. In particular, I can run the command get diff Had to get the difference between my local files, the files in my working directory, and the files in my local repository. So what I can do, I can fetch the files from the remote repository, and once I feth these files. I can run a get diff head and check what the differences are. And based on the differences this side whether I want to merge or not. So what were talking about [INAUDIBLE] diff, there is something else that you can use with the diff command. So what you can do, you can run git diff without further specifying here. In this case, what the command tell you is 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 that what you could still add to the stage for the further commit, and that you haven't already. So what lockout changes will not make it to the next commit, basically. And this you can use, for example, as a sanity check before doing a commit to make sure all the lockiout changes that you have, and that you want to commit, are actually staged and therefore will be considered. So now we will cover all the commands that we saw here. In our practical demo. But please feel free to refer back to this Git Workflow to get a kind of a high level vision. Or maybe you want to keep it next to you, because this really gives you the overall structure and the overall view of what happens when you run the different commands. And it also helps you visualize The different elements that are relevant when you're using GIT. So the workspace, once more, the index or stage, the local repository, and the remote repository.





13. In this first part of the git demo, we will call it the basics of git. So for example, 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, so you can run the command git and, if you just execute git you will get the usage information for git, with the most commonly used git commands. And to find information on any command, you can simply type. Git help, and the name of the command. For example, lets try to write git help init. And that brings up the git manual page for git init, which describes the command, the synopsis, and so on. Now, lets get started with using git by introducing ourselves to git, which is the first thing we need to do. 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 (git config --global user.name “George P. Burdell”). Which means we are telling it our user name. We'll specify our user name which in this case is George P. Burdell. You could also provide your email address in the same way (git config –global user.email “gpbud@gatech.edu”). So you 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.

上面的命令為:
git config --global user.name Tao Peng
git config --global user.email "tpeng38@gatech.edu"

We will now look at some commonly used commands that to create and maintain a locker repository. Let's first create a, a new project and call it my project. So, to do that we are simply going to create a directory and then we're going to move into that directory. Now, if we try to call the git status command at this point to see what's the state of my project, of course git doesn't know anything about this project, right? So, you will get an error. It will tell you that, basically, we're not in a git repository. So how do we create a git repository? How do we make this? A Git repository, but we do it 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. So let's just create a, a new, empty file. Which we're going to call touch REAME. So now if you run git status, as you can see, git will tell you there is a file that's called README, but it's untracked. Now what that means is that the file not staged, if you remember our lesson. So what we need to do, we first need to tell git that, you know, this needs to be considered. And the way we do that, is by calling the Git add command and then we specify README as the argument for the command. If we call again, Git status. Now, as you can see, Git knows that there is a new file called README, because the file is staged. So Git is aware of the fact that this file has to be committed.

以下的命令為:
mkdir myproject
cd myproject
git status
git init
git status
touch README (即新建 README 這個文件)
git status
git add README (此處的的 add 和下面的 commit 是甚麼意思? 見前面(不遠處)的 Git workflow 圖.)
git status

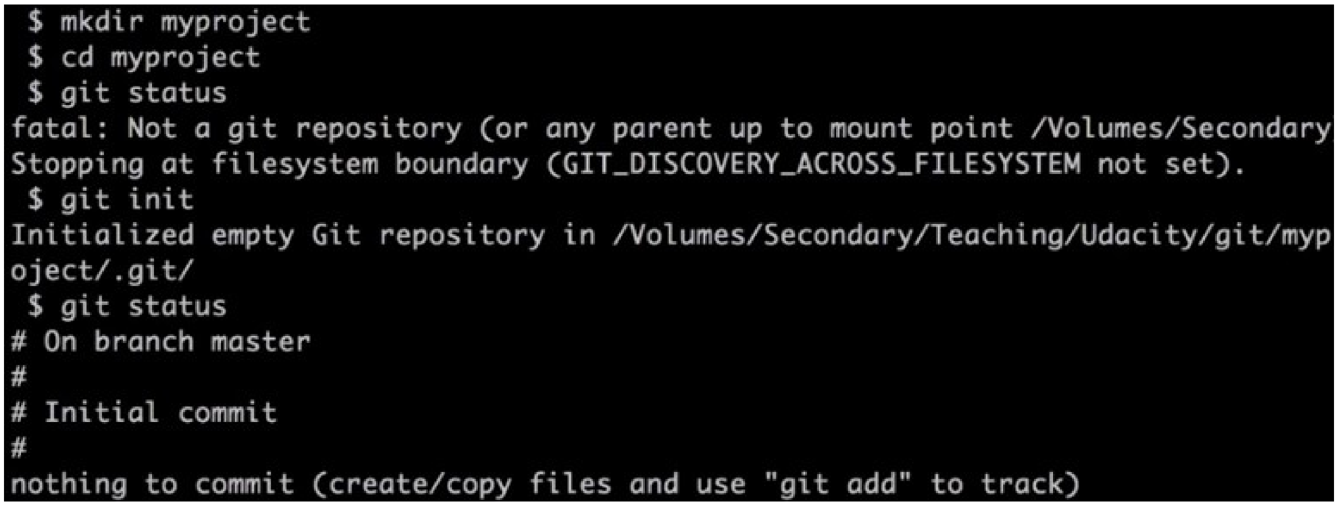
然後:
git commit
然後就是一個 text editor 頁面
然後寫
Added README file (注意這是一個 comment, 實踐表明可以隨便寫, 但這不是我們想寫入 README 的內容)

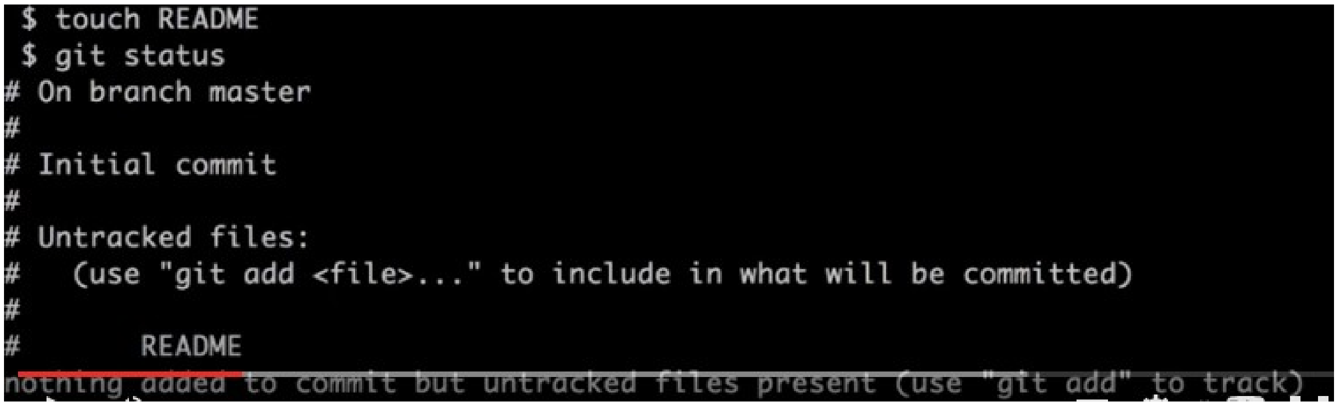
以下是我電腦的 text editor. 這個 text editor 是 GNU nano 2.2.6. 最下面的^G 表示 Ctrl+G. 所以網上說:
If you press Ctrl-X, for exit, it will ask you whether you want to save the file. Ctrl-O is for saving file without exiting the editor.
Ctrl-G is for help on key combinations.

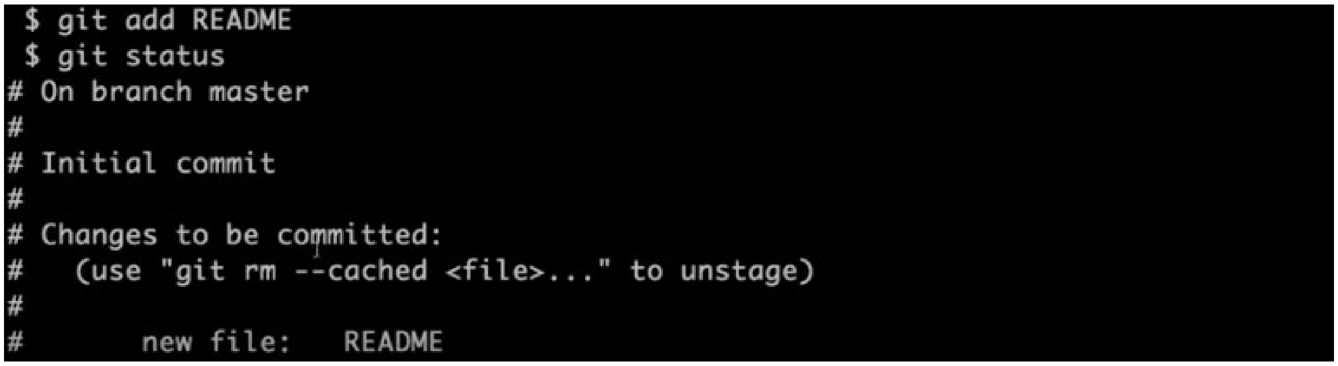
簡單地說: 保存並退出的命令為:
Ctrl+X → Y → 回車

(還有個網上說: Alternatively, you can just do git commit -m "comment" instead of having git open the editor to type the message)

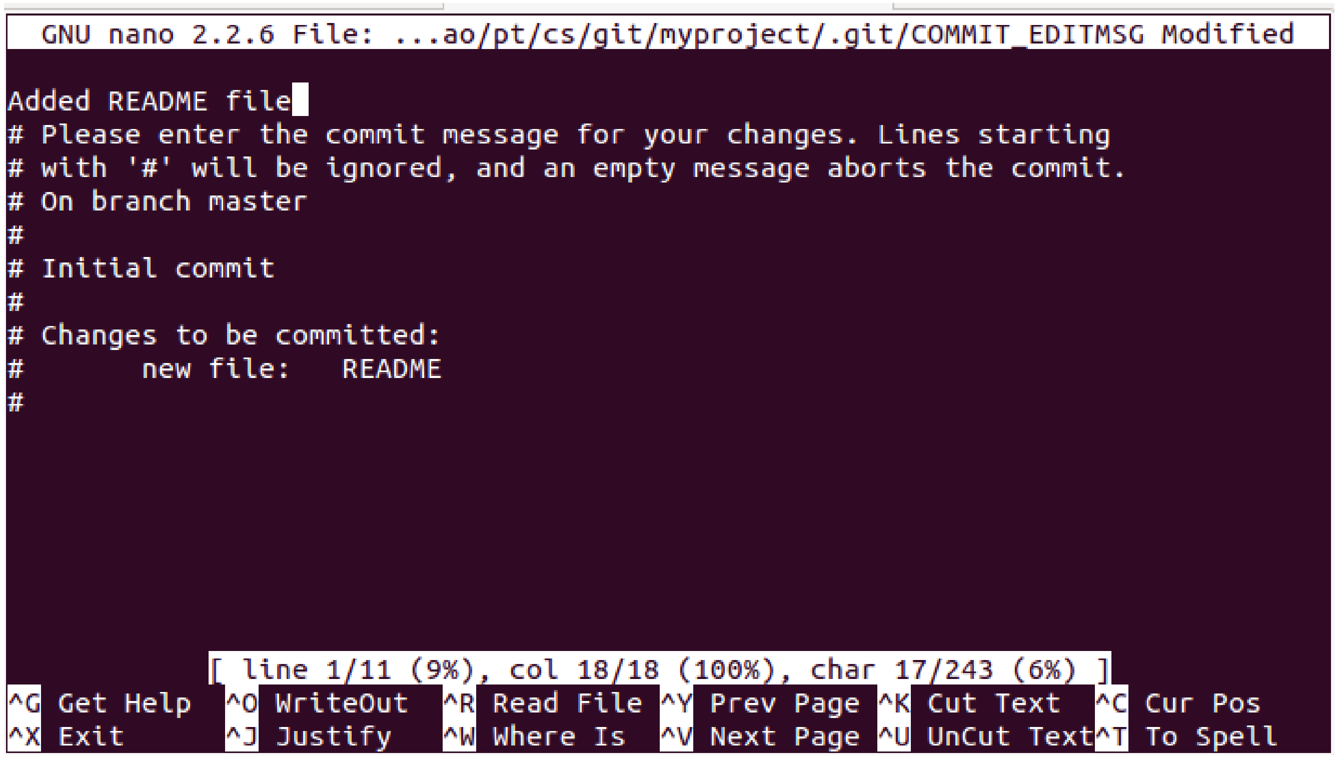
以上 add 和 commit 可以合成一句: git commit -a -m "Added README file" (要避免用這一句, 還是用以上的 git add file 和 git commit -m "comment"兩句, 原因見後面的[注 857928])


```
 $ mkdir myproject
 $ cd myproject
 $ git status
fatal: Not a git repository (or any parent up to mount point /Volumes/Secondary
Stopping at filesystem boundary (GIT_DISCOVERY_ACROSS_FILESYSTEM not set).
 $ git init
Initialized empty Git repository in /Volumes/Secondary/Teaching/Udacity/git/myp
oject/.git/
 $ git status
# On branch master
#
# Initial commit
#
nothing to commit (create/copy files and use "git add" to track)
```


```
 $ touch README
 $ git status
# On branch master
#
# Initial commit
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       README
nothing added to commit but untracked files present (use "git add" to track)
```


```
 $ git add README
 $ git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   README
```

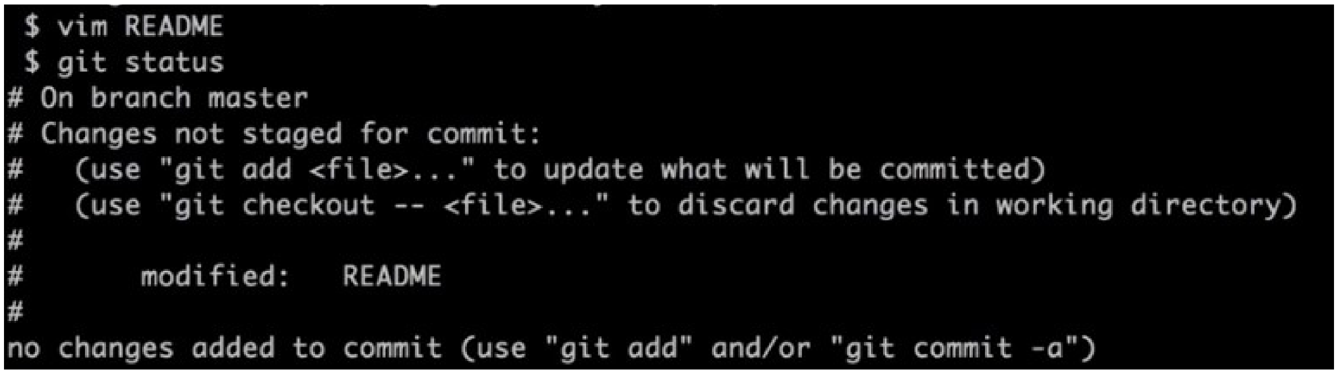
So, to commit a file, we simply execute git commit, which will open a text editor, which can be different, depending on what is your environment, and here we need to add a comment to be added to the commit.



```
  GNU nano 2.2.6 File: ...ao/pt/cs/git/myproject/.git/COMMIT_EDITMSG Modified

Added README file
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
#
# Initial commit
#
# Changes to be committed:
#       new file:   README
#

                      [ line 1/11 (9%), col 18/18 (100%), char 17/243 (6%) ]
^G Get Help    ^O WriteOut    ^R Read File   ^Y Prev Page   ^K Cut Text    ^C Cur Pos
^X Exit        ^J Justify     ^W Where Is    ^V Next Page   ^U UnCut Text^T To Spell
```

So here we simply write in Added README file, then we can close and save and this will add the file to the Git repository. The local Git repository of course. 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. Because of course the only file that we have, is committed to the repository.

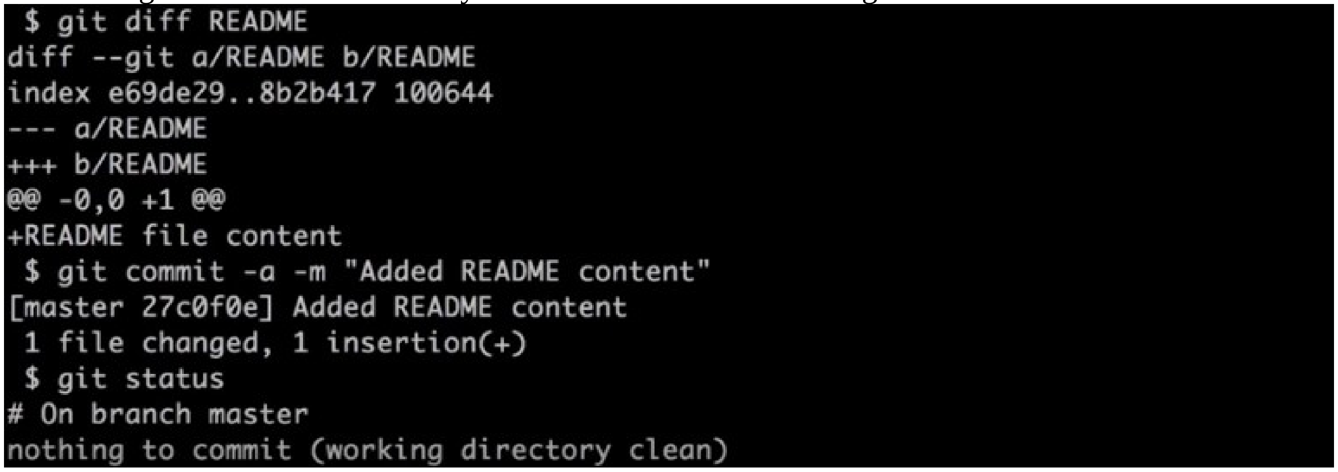
Now, let's make some changes to our README file (即用 vi README). I'm just going to add some text here (隨便甚麼 text). Once more, we can run git status, and at this point, get knows about this file. So, it will know that README file has been modified. Remember that before, it was telling you that it was a new file, now it knows that there was a different version in the repository.



```
 $ vim README
 $ git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   README
#
no changes added to commit (use "git add" and/or "git commit -a")
```

上面的命令為:
vi README
然後修改 RADME 文件
git status

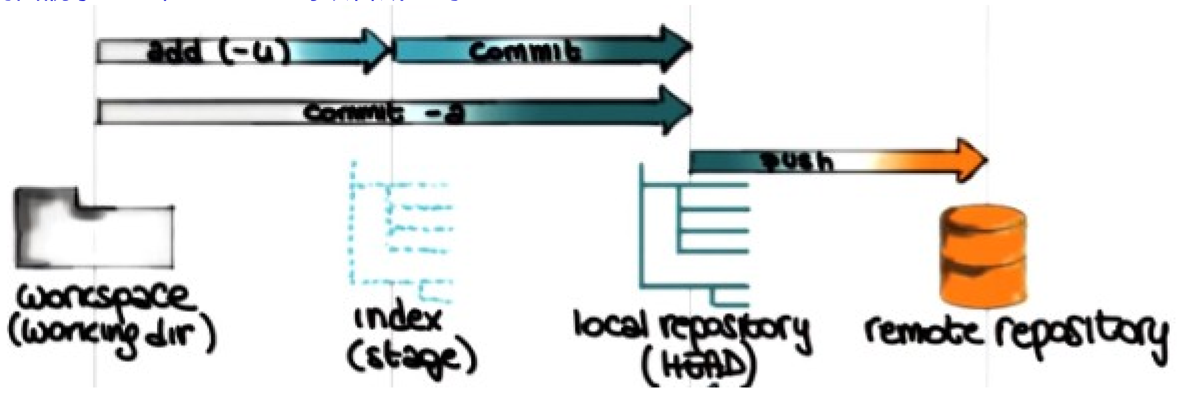
So something we can do, at this point, for example, is to check the differences. Between this file and the committed one by executing get diff readme and if you look at the output of the get diff command here, you can see that this line, readme file content was added and you'll see that it was added because there's a plus sign before that line. In case of deletion of lines, you'll see a minusm sign there. So 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. So to tell git, that the file has to be staged, we will, can use the usual git add command. But if you remember the lesson, we can also use a shortcut. So you, we don't really have to do this in two steps. We can simply say, 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. Something else that we can do, is that we can also provide the right away message for the commit, without having to open an editor. So, to do that we can specify the -m option. And at this point a we can just put a in double quotes our content we 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 a insertion again if we run this status you will see that there is nothing else to commit.



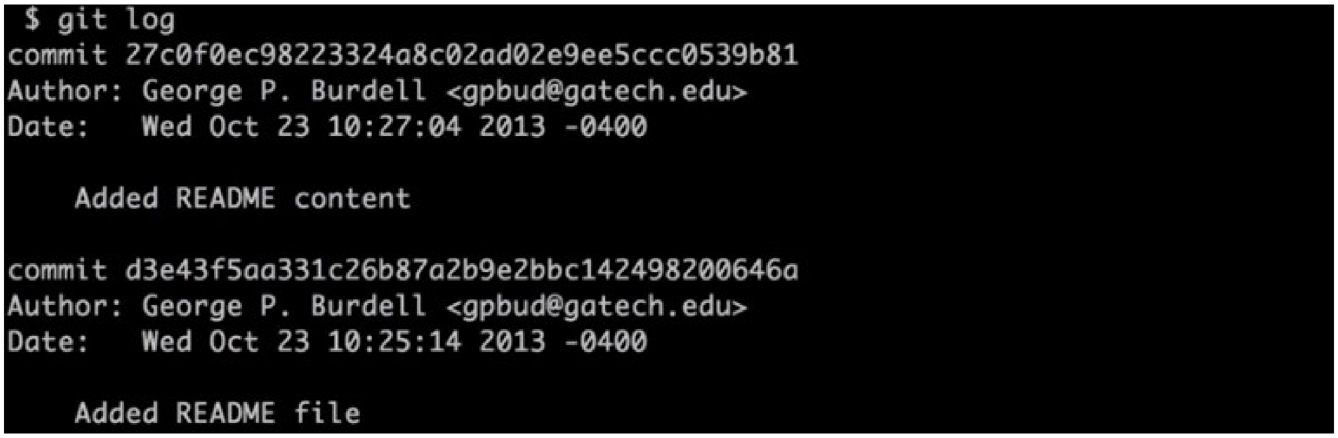
```
 $ git diff README
diff --git a/README b/README
index e69de29..8b2b417 100644
--- a/README
+++ b/README
@@ -0,0 +1 @@
+README file content
 $ git commit -a -m "Added README content"
[master 27c0f0e] Added README content
 1 file changed, 1 insertion(+)
 $ git status
# On branch master
nothing to commit (working directory clean)
```

上面的命令為:
git diff README
git commit -a -m "Added README content" (如果加了這句, 就會像上面窗口那樣說 nothing to commit, 因為已执行了 git commit -a -m...後, 就沒有甚麼要 commit 的了. 如果不加 git commit -a -m...這句, 則會是示 Changes not staged for commit 和 no changes added to commit)

注意前面說了 add 和 commit 可以合成一句: commit -a



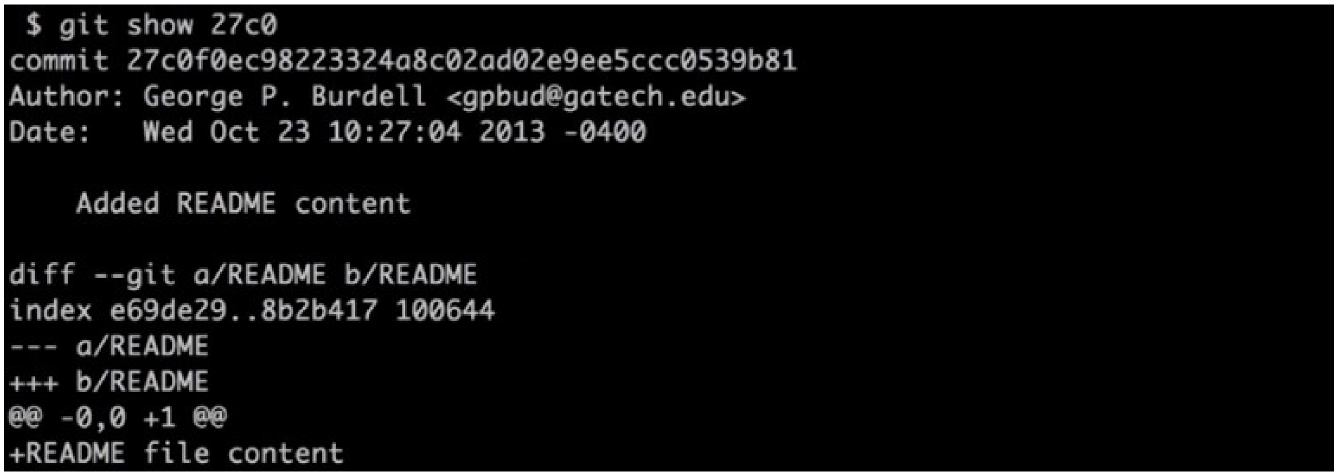
So now lets imagine that you want to see the version history for your repository. You can do that by running the git log command. So if you run data, it will show you all the different commits For your repository. And hcommit is got a commit ID, as you can see here and the one down, down here is the first commit, where as the one above is the second commit. And as you can see, we'll also show you the comments associated with hcommit.



```
 $ git log
commit 27c0f0ec98223324a8c02ad02e9ee5ccc0539b81
Author: George P. Burdell <gpbud@gatech.edu>
Date:   Wed Oct 23 10:27:04 2013 -0400

    Added README content

commit d3e43f5aa331c26b87a2b9e2bbc142498200646a
Author: George P. Burdell <gpbud@gatech.edu>
Date:   Wed Oct 23 10:25:14 2013 -0400

    Added README file
```

上面的命令為:
git log

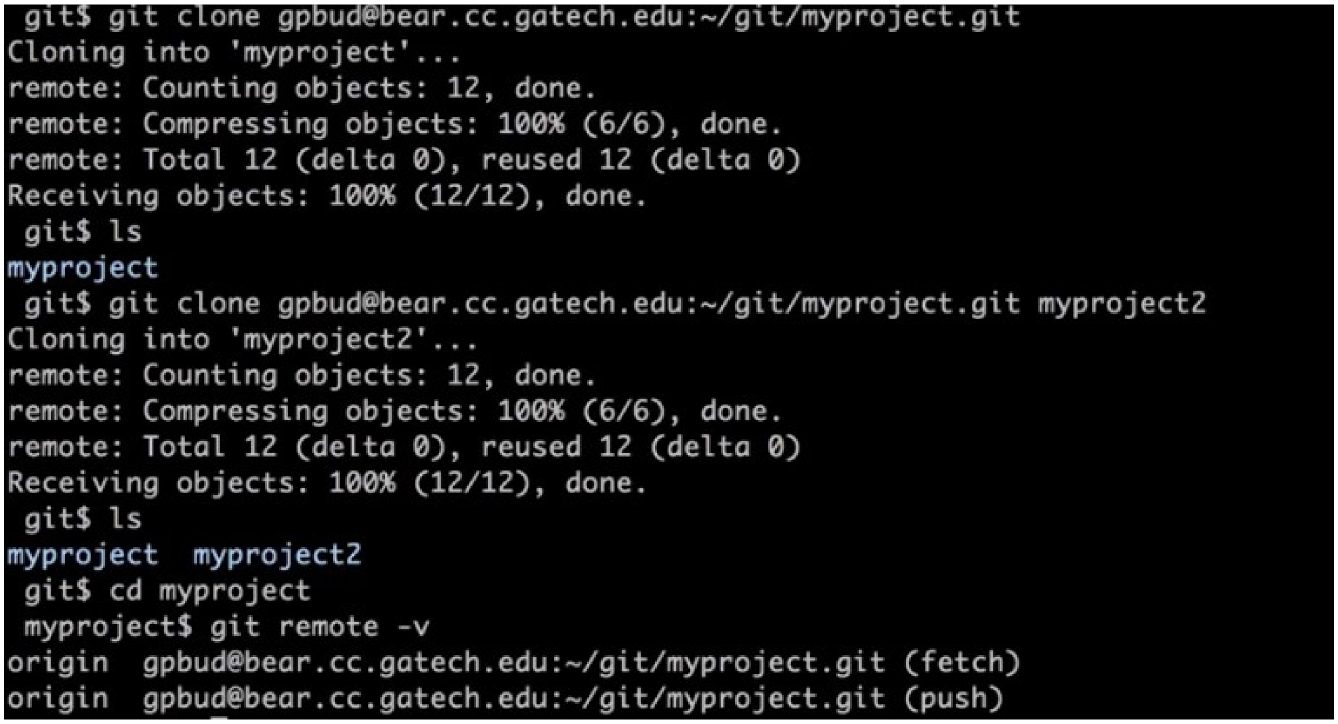
And in case you wanted to see the changes introduced by a commit. You can use that git show command, and you can provide the commit ID for the commit that you're interested in. And you don't really need to provide the whole ID, you can provide the first four or more characters. So that's what we're going to do here. So we're going to specify the second commit, and when we execute the command it will show use the changes introduced by that commit.



```
 $ git show 27c0
commit 27c0f0ec98223324a8c02ad02e9ee5ccc0539b81
Author: George P. Burdell <gpbud@gatech.edu>
Date:   Wed Oct 23 10:27:04 2013 -0400

    Added README content

diff --git a/README b/README
index e69de29..8b2b417 100644
--- a/README
+++ b/README
@@ -0,0 +1 @@
+README file content
```

上面的命令為:
git show 27c0

To fetch a report from a remote server, you can use the git clone command. So you will write git clone and then specify the URL for the remote repository. Here we are using the SSH protocal and there are different protocals that can be used, so the remote repository can be made avaiable in different ways. As you can see, 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. For example, in this case, my project two. And, so here you'll get the project in my local work space with the name that I specified. So, let's go inside one of these two projects that have the same content because they're coming from the repository. If you want to see the details of the server you can use the remote command and specify the flag -v.



```
 git$ git clone gpbud@bear.cc.gatech.edu:~/git/myproject.git
Cloning into 'myproject'...
remote: Counting objects: 12, done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 12 (delta 0), reused 12 (delta 0)
Receiving objects: 100% (12/12), done.
 git$ ls
myproject
 git$ git clone gpbud@bear.cc.gatech.edu:~/git/myproject.git myproject2
Cloning into 'myproject2'...
remote: Counting objects: 12, done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 12 (delta 0), reused 12 (delta 0)
Receiving objects: 100% (12/12), done.
 git$ ls
myproject  myproject2
 git$ cd myproject
 myproject$ git remote -v
origin  gpbud@bear.cc.gatech.edu:~/git/myproject.git (fetch)
origin  gpbud@bear.cc.gatech.edu:~/git/myproject.git (push)
```

以上的命令為:

```
git clone gpbud@bear.cc.gatech.edu:~/git/myproject.git
ls
git clone gpbud@bear.cc.gatech.edu:~/git/myproject.git myproject2
ls
cd myproject.git
git remote -v
```

在我電腦上, 應寫成 git clone tpeng38@gatech.edu:~/home/tao/pt/cs/git/myproject.git myproject2
但提示是:

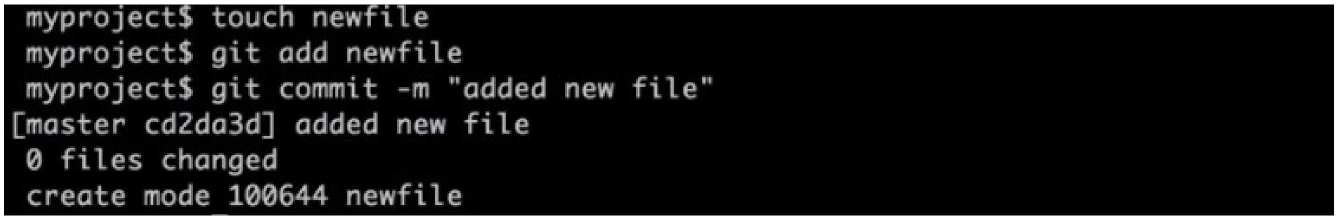
```
Cloning into 'myproject2'...
ssh: connect to host gatech.edu port 22: Connection timed out
fatal: Could not read from remote repository.
Please make sure you have the correct access rights and the repository exists.
```

我在做 assignment2 時, 也遇到了同樣的問題(很多同學也有這個問題), 後來有個網上給出的解決方法是 work 的, 就是在 git clone 之前加一個 GIT_SSL_NO_VERIFY=true, 即:

```
GIT_SSL_NO_VERIFY=true git clone https://github.gatech.edu/gt-omscs-softeng/6300Spring16tpeng38.git
```

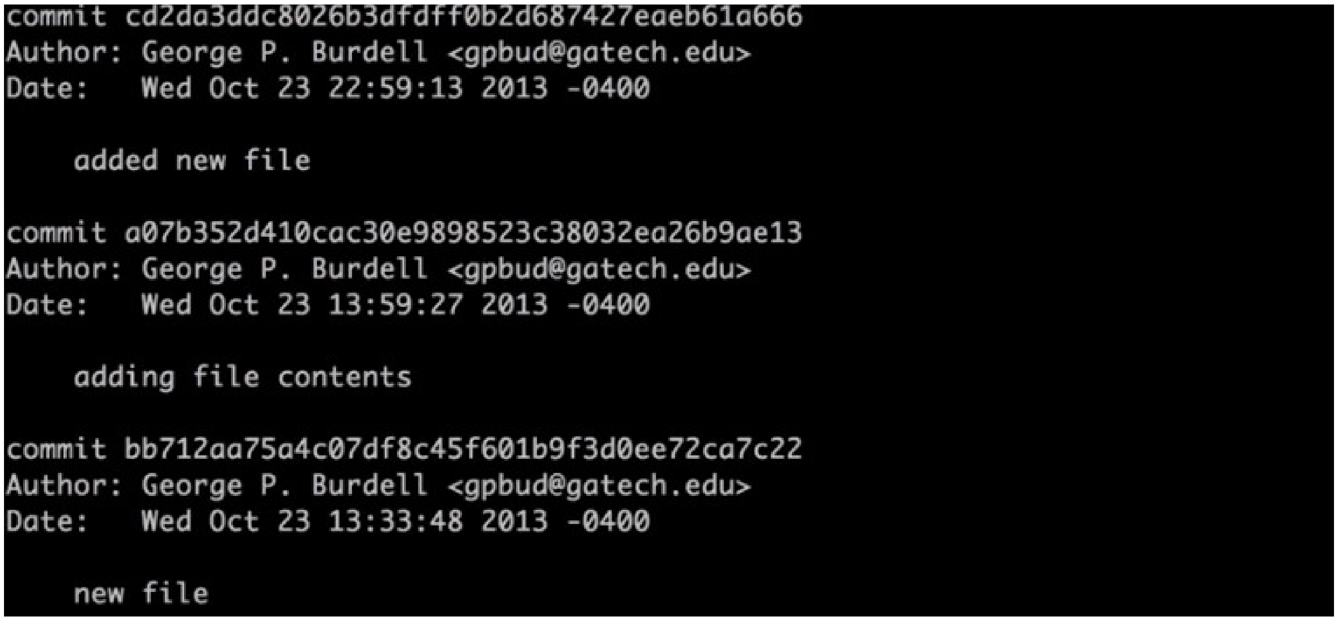
然後 username 是 tpeng38, password 是 gatech 郵箱密碼, 別忘了末尾有個 2.

And here we'll show you what is the remote repository now let's go ahead to make some changes to the project for example let's edit file. So I'm just going to create this empty file which I am going to call new file I'm going to add it to my index so that it gets committed. Later on and then I'm going to run git commit to actually commit it to the local repository. And I'm going to specify the comment for the commit right away here from the command line. So when we do that the file gets added to my local repository. And if we want to double check that, we can run git log. And if you look at the last commit at the top, you can see that it's telling me that the new file was added to the repository, showing the comment that I added. But this is just for the local repository, so I need to use the git push command to push it to the remote repository. And at this point, when I run that, my local changes will be committed to the remote repository.



```
 myproject$ touch newfile
 myproject$ git add newfile
 myproject$ git commit -m "added new file"
[master cd2da3d] added new file
 0 files changed
 create mode 100644 newfile
```

以上的命令為:
先进入 myproject 文件夾
touch newfile
git add newfile
git commit -m "added new file"
然後輸入 git log, 即得到以下的結果(type q to exit the log screen):



```
commit cd2da3ddc8026b3dfdff0b2d687427eaeb61a666
Author: George P. Burdell <gpbud@gatech.edu>
Date:   Wed Oct 23 22:59:13 2013 -0400

    added new file

commit a07b352d410cac30e9898523c38032ea26b9ae13
Author: George P. Burdell <gpbud@gatech.edu>
Date:   Wed Oct 23 13:59:27 2013 -0400

    adding file contents

commit bb712aa75a4c07df8c45f601b9f3d0ee72ca7c22
Author: George P. Burdell <gpbud@gatech.edu>
Date:   Wed Oct 23 13:33:48 2013 -0400

    new file
```

(上圖中要看頂部的 added new file 那個 log)

然後輸入 git push, 即得到以下的結果:

我電腦上直接 git push 不行, 要按以下這樣:

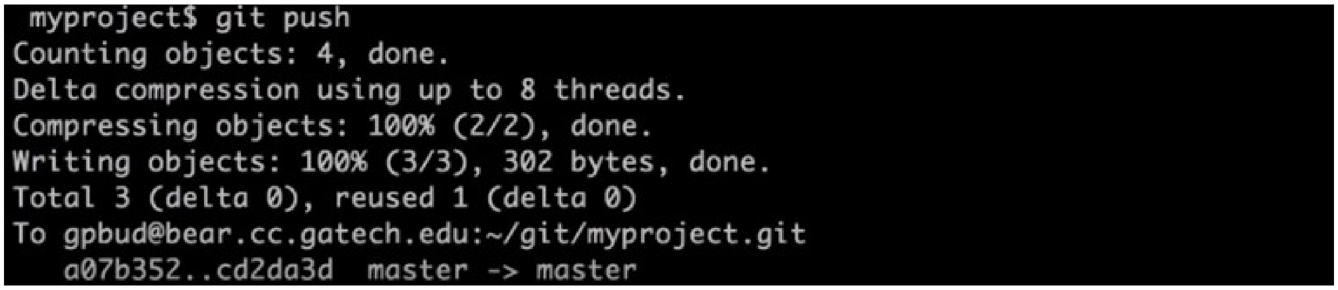
以下這句是不必要的, 直接用 origin 這個 remote 即可.

git remote add repo 'https://github.gatech.edu/gt-omscs-softeng/6300Spring16tpeng38.git'
上句是添加 repo 為一個 remote, 另一個是默認的 remote 叫 origin. 可以用 git remote -v 看所有的 remote.

From online: 在clone完成之后，Git会自动为你将此远程仓库命名为origin. origin只相当于一个别名，运行git remote –v或者查看.git/config可以看到origin的含义.

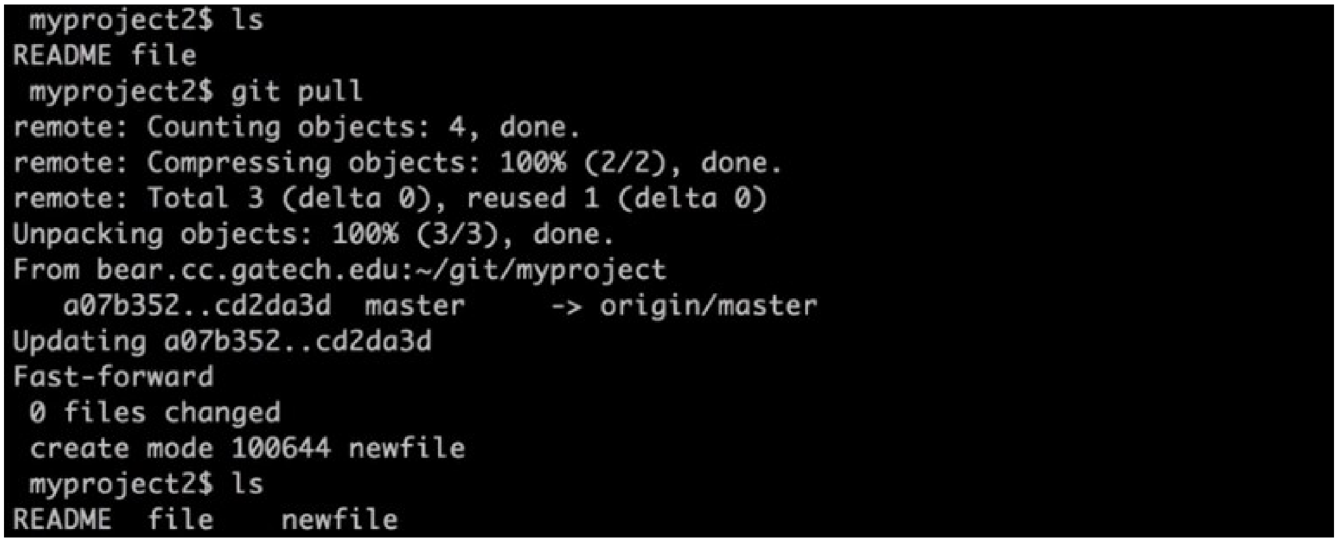
然後(若沒添加 repo 為一個 remote, 就將 repo 換成 origin 即可, 後同):
GIT_SSL_NO_VERIFY=true git push --all repo
也可以不要 repo, 即:
GIT_SSL_NO_VERIFY=true git push --all

如果在 push 之前, 已經有其它人改動了 master, 則在 push 之前, 要先 pull:
GIT_SSL_NO_VERIFY=true git pull repo master



```
 myproject$ git push
Counting objects: 4, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 302 bytes, done.
Total 3 (delta 0), reused 1 (delta 0)
To gpbud@bear.cc.gatech.edu:~/git/myproject.git
   a07b352..cd2da3d  master -> master
```

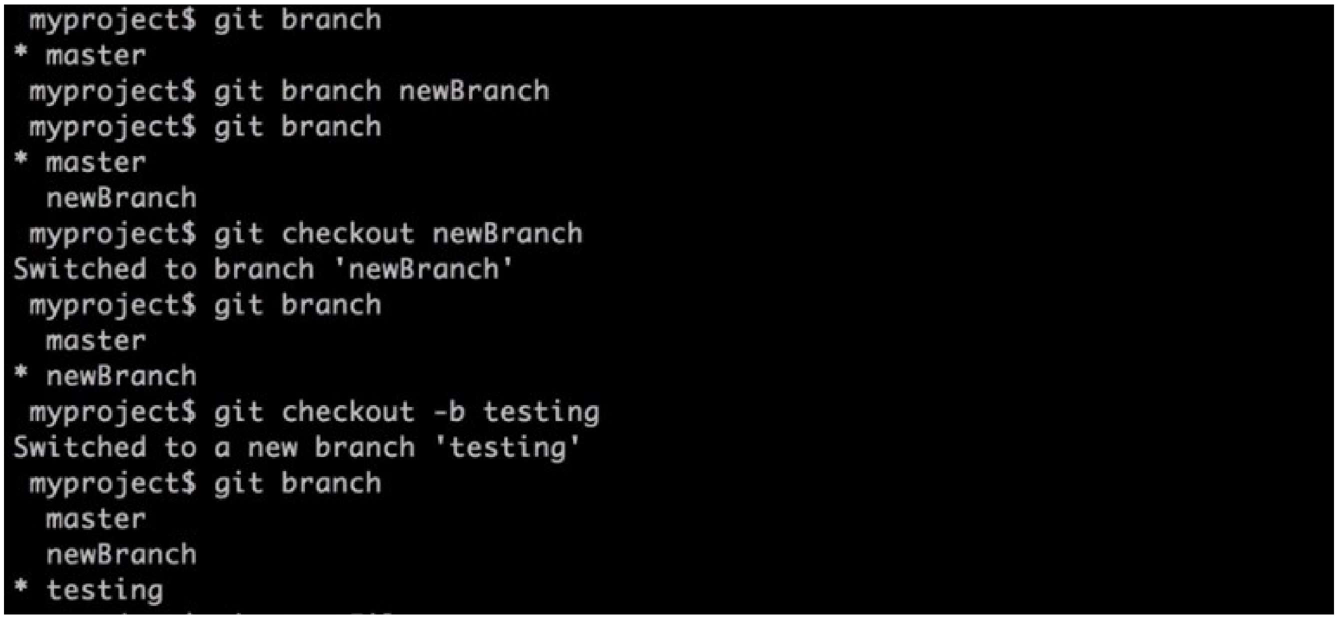
So now let's go to the other copy of the project that we created. The one under directory myproject2. If you remember this project was linked up to the same remote project. But of course, if we run get log here, we don't see this latest change that we made, because we didn't synchronize this local copy with the remote copy. And so we just have these files, the README and ,Five that worked there before. So what we need to do is that we need to pull the changes from the remote repostery 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. And 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 and of course, this is just an example, so we had two copies of the project on the same machine and for the same user, so the normal users scenario for this, it will be that, each user will have their local copy, but this should have given you the idea of how, git allows you to work on some local file. Commit them and push them to a remote repository and other users to get your changes, do further changes push them as well and then, you know, they will allow you to get their changes, and so on and so forth. So really allows this collaboration between different users and keep in track of all the changes made by the different users.



```
 myproject2$ ls
README file
 myproject2$ git pull
remote: Counting objects: 4, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 1 (delta 0)
Unpacking objects: 100% (3/3), done.
From bear.cc.gatech.edu:~/git/myproject
   a07b352..cd2da3d  master     -> origin/master
Updating a07b352..cd2da3d
Fast-forward
 0 files changed
 create mode 100644 newfile
 myproject2$ ls
README  file    newfile
```

以上的命令為:
进入 myproject2
git pull
ls
可以看到 myproject2 中也有 newfile 這個文件了

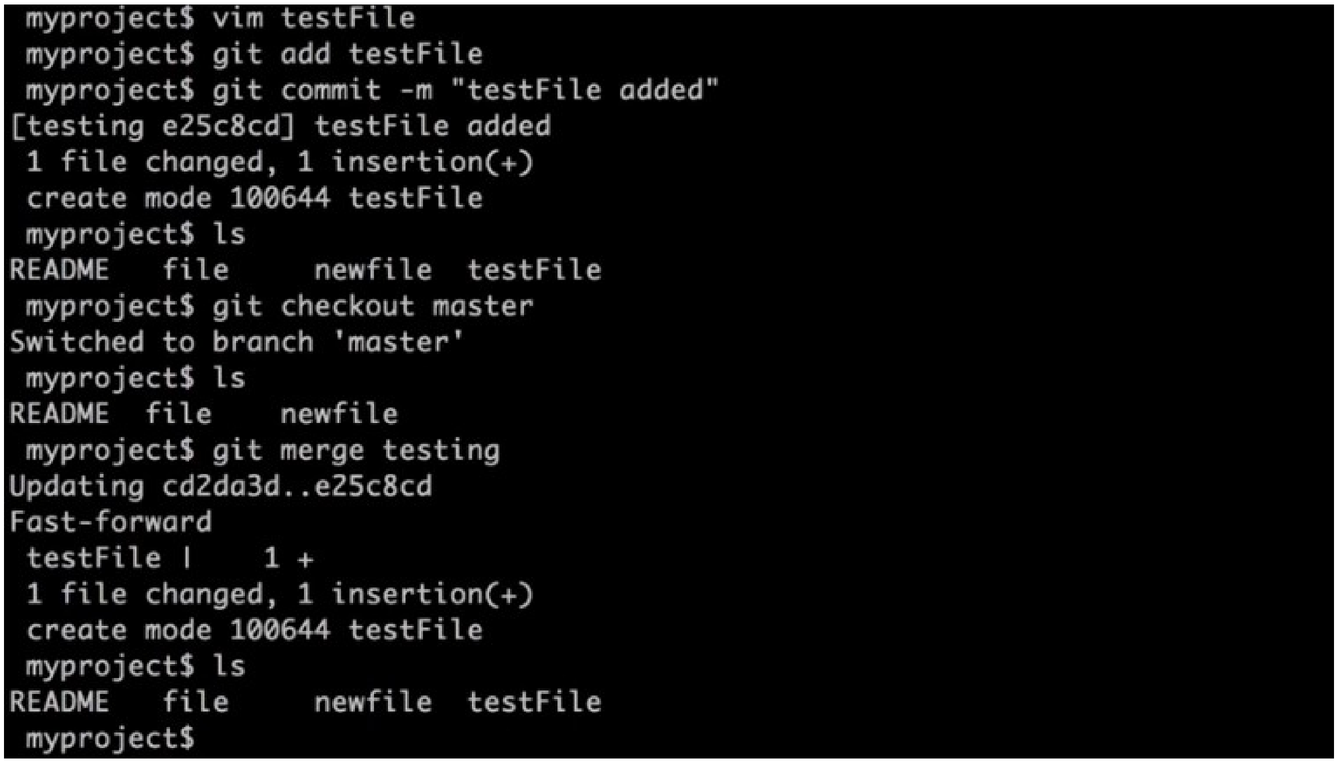
So now let's look at some more advanced concept, which are the concept of branching, and merging. So what branching means is basically is to make a copy, to create a branch of the current project so that we can work on that copy indpendently from the other copy, from the other branch. And then we can decide whether we want to keep, both branches, or we want to merge them at some point. And you can of course have multiple branches, not just two. And the reason why this is particularly useful is because in many cases if you think, about the way, with the valve software in general, we work with artiacts. We might have the need to create kind of a separate copy of your work space...to do some experiments for example. So you want to change something in the code, you're not really sure it's going to work and you don't want to touch your main copy. So that's the perfect application for branching. If you want to do something like that...you want to experiment or do some modifications that you're not sure about, you will branch your code, you will do the changes...and then if you're happy with the changes, you will merge them. That branch with the original one, or worse if you're not happy with the changes you will just throw away that branch. So this is just one possible use of branch but it's one of the main uses of that. So in all let's see how that can be done with kit. So 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 there indicates that this is our current branch. So how do we create a new branch? So we simply run the command git branch and specify a name for the new branch, for example we'll call it New branch, to make it very explicit. At this point, if we run git the branch off course, we will have a new branch plus master will still be our current branch. So if you want to switch to the new branch, we will use the git checkout command and specify the name of the branch that we want to become our current branch. So when we run that, git will tell us that we switched to the new branch. And 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 -b flag and then the name of the new branch. It will do both things at the same time. 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 after executing the command. So now if we look at the GIT branch output, you can see that there is three branches and we are currently on the testing branch.



```
 myproject$ git branch
* master
 myproject$ git branch newBranch
 myproject$ git branch
* master
  newBranch
 myproject$ git checkout newBranch
Switched to branch 'newBranch'
 myproject$ git branch
  master
* newBranch
 myproject$ git checkout -b testing
Switched to a new branch 'testing'
 myproject$ git branch
  master
  newBranch
* testing
```

以上的命令為:
git branch (看有哪些 branch)
git branch newBranch (新建一個 branch)
git branch
git checkout newBranch (將 newBranch 設為 current branch)
git branch
git checkout -b testing (新建一個叫 testing 的 branch, 並將其設為 current branch)
git branch
注意現在已經在 testing branch 裡了

So now let's create a new file and just call it test file, put some content in there, save it, we edit and commit it. And as you can see, now in this current branch, we have our testFile. So now let's switch to a different branch. So let's go back to the master branch using the usual git checkout command. So now if we do an ls, if we check the content of the current directory, we can see that the testFile is not there, because of course, it's not in this branch. so now 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. 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. So we will specify testing in this case. 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, is in my current, Current branch. And if I run the branch, you'll see that the testing branch is obviously still there, so let's assume that we want to delete the testing branch at this point because we don't need it anymore. We could simply execute the branch -d which stands for minus delete, specify the name of the branch and this will eliminate that branch. As confirmed by running the command git branch or the testing branch no longer shows up.



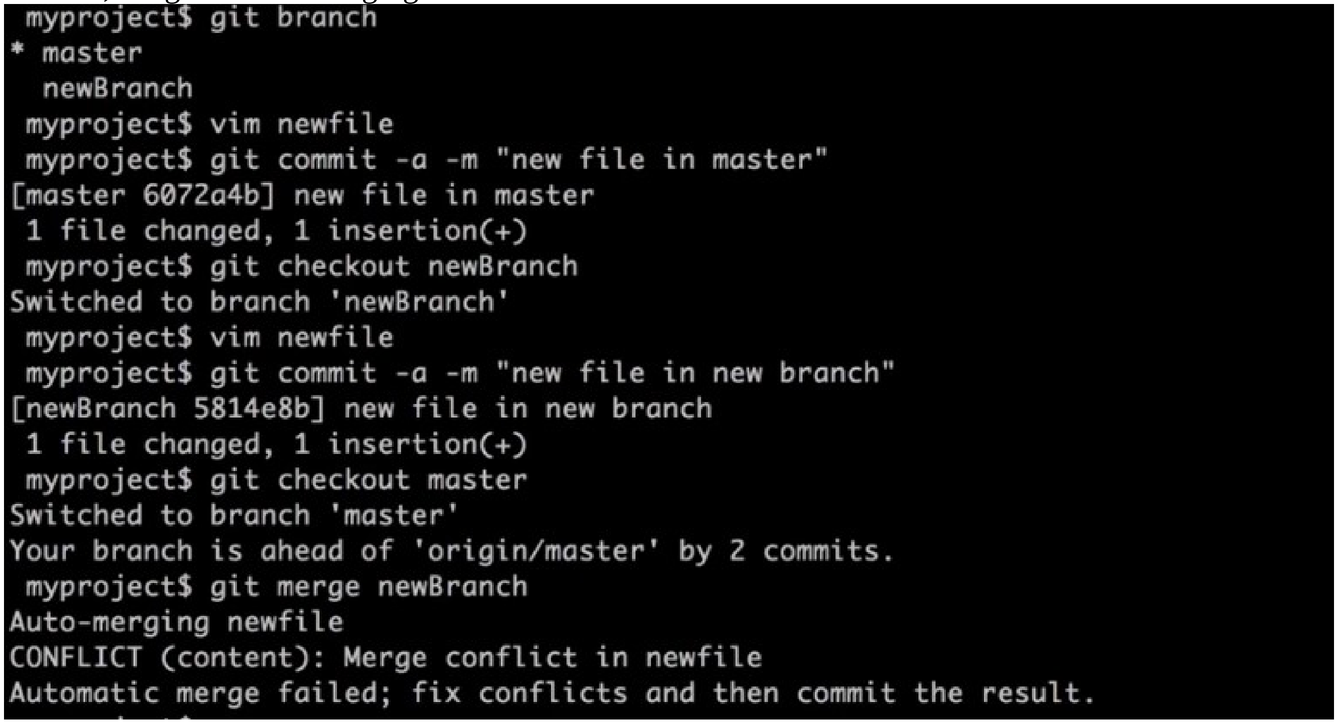
```
 myproject$ vim testFile
 myproject$ git add testFile
 myproject$ git commit -m "testFile added"
[testing e25c8cd] testFile added
 1 file changed, 1 insertion(+)
 create mode 100644 testFile
 myproject$ ls
README    file      newfile  testFile
 myproject$ git checkout master
Switched to branch 'master'
 myproject$ ls
README   file     newfile
 myproject$ git merge testing
Updating cd2da3d..e25c8cd
Fast-forward
 testFile |     1 +
 1 file changed, 1 insertion(+)
 create mode 100644 testFile
 myproject$ ls
README    file      newfile  testFile
 myproject$
```

以上的命令為(注意目前在 testing branch 裡):
vim testFile, 然後在 testFile 加一些內容, 保存並退出
git add testFile
git commit -m "testFile added" (注意本句和上句是必須的, 否則所有 branch 中都會有 testFile)
ls (testing branch 中有 testFile)
git checkout master
ls (master branch 中無 testFile)
git merge testing
ls (master branch 中有 testFile 了)
然後用 git branch -d testing 即可刪除 testing 這個 branch.

[注 857928] 注意, 若將 add 和 commit 合成一句: git commit -a -m "Added README file", 則所有 branch 中都會有 testFile 這個文件, 所以以後要避免用這一句, 還是用以上的 git add file 和 git commit -m "comment"兩句比較好!

實踐表明:
若最開始只有 master branch, 然後在裡面建了一個文件 a 並 commit, 然後建了一個新的 branch 叫 newBranch, 則 newBranch 中也有 a 這個文件. 若再在 newBranch 中建了一個文件 b 並 commit, 則 master branch 中並無 b. 只有 merge 了後, master branch 中才有 b. 這些跟實際應用中的需要是一致的, 想想即知.

So, something that might happen when you merge a branch is, is that you might have conflicts For example, in case you change the, the same file into different branches. So, let's see an example of that. So, we're going to check which branches we have, so we have two branches, in this case, master and new branch. Our current branch is master. Let's open this file called new file and, add some content there. So now let's commit this changes to the get to the locker repository. Now let's reach to the other branch and if you remember we do this by running git checkout and the name of the branch. And at this point we do the same operation here. So we take this file and we change it here to. In this case we have content that reflects the fact that we are. In the new branch just for convenience. At this point, we also can move the file here. The comment here is, of course, that this is the new file in the new branch. So, at this point, what we have here is that we have this file called New File that has been defined independently both in the master branch and in the new branch. So we have a conflict. Right? So, now, let's switch back to the master branch. So 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. And when we run that, we get an auto merging conflict.

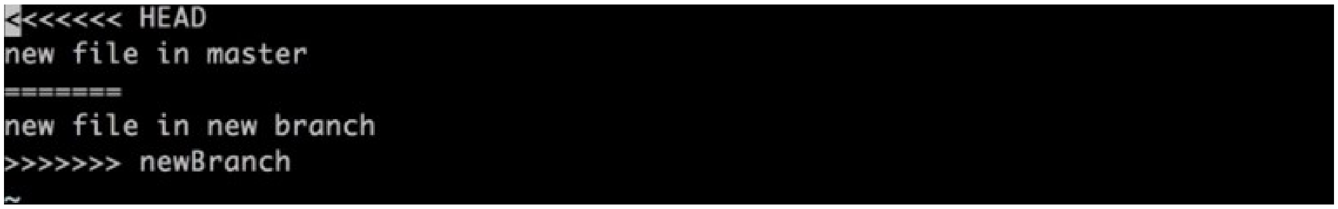


```
 myproject$ git branch
* master
  newBranch
 myproject$ vim newfile
 myproject$ git commit -a -m "new file in master"
[master 6072a4b] new file in master
 1 file changed, 1 insertion(+)
 myproject$ git checkout newBranch
Switched to branch 'newBranch'
 myproject$ vim newfile
 myproject$ git commit -a -m "new file in new branch"
[newBranch 5814e8b] new file in new branch
 1 file changed, 1 insertion(+)
 myproject$ git checkout master
Switched to branch 'master'
Your branch is ahead of 'origin/master' by 2 commits.
 myproject$ git merge newBranch
Auto-merging newfile
CONFLICT (content): Merge conflict in newfile
Automatic merge failed; fix conflicts and then commit the result.
```

以上的命令為:
git branch (此時在 master branch 裡)
vim newfile
git commit -a -m "new file in master"
git checkout newBranch (此時在 newBranch 裡了)
vim newfile
git commit -a -m "new file in new branch"
git checkout master (此時又回到 master branch 裡)
git merge newBranch (有 conflict 了)

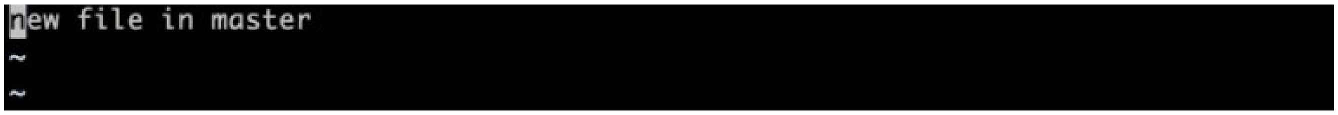
So at this point what we can do, is that we can manually fix the conflict by opening the new file. So the file that was showing. The conflict. So here you can see the kind of of information that you get in the conflicted file. So it's telling you basically that there is in the head which is the, the master this conflict. Which is new file in master. Which is the content that we added of course. And then you know, under, you know, the separator you can see the content that was added in the new branch. Which is the contents in new file, in new branch. So basically, what this is showing you is the parts of the file that are conflicting. In this case, we only have one line, is basically the whole file into two versions and you can decide which version you want to keep or how you want to merge in general, the two pieces. So here, let's assume that we want to keep the,the content from the master. So what we're going to do is we're going to elimate the annotations and we're going to eliminate the additional content. We save this file. So at this point what we need to do is simply commit the modify file the merge file and we do that in the normal way. We call get add, Specifying the file, so git add newfile. Then we run git commit newfile, and we 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.

newfile 中的內容:



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

將不要的(即 newBranch 中的)和注釋刪掉, 然後保存退出:



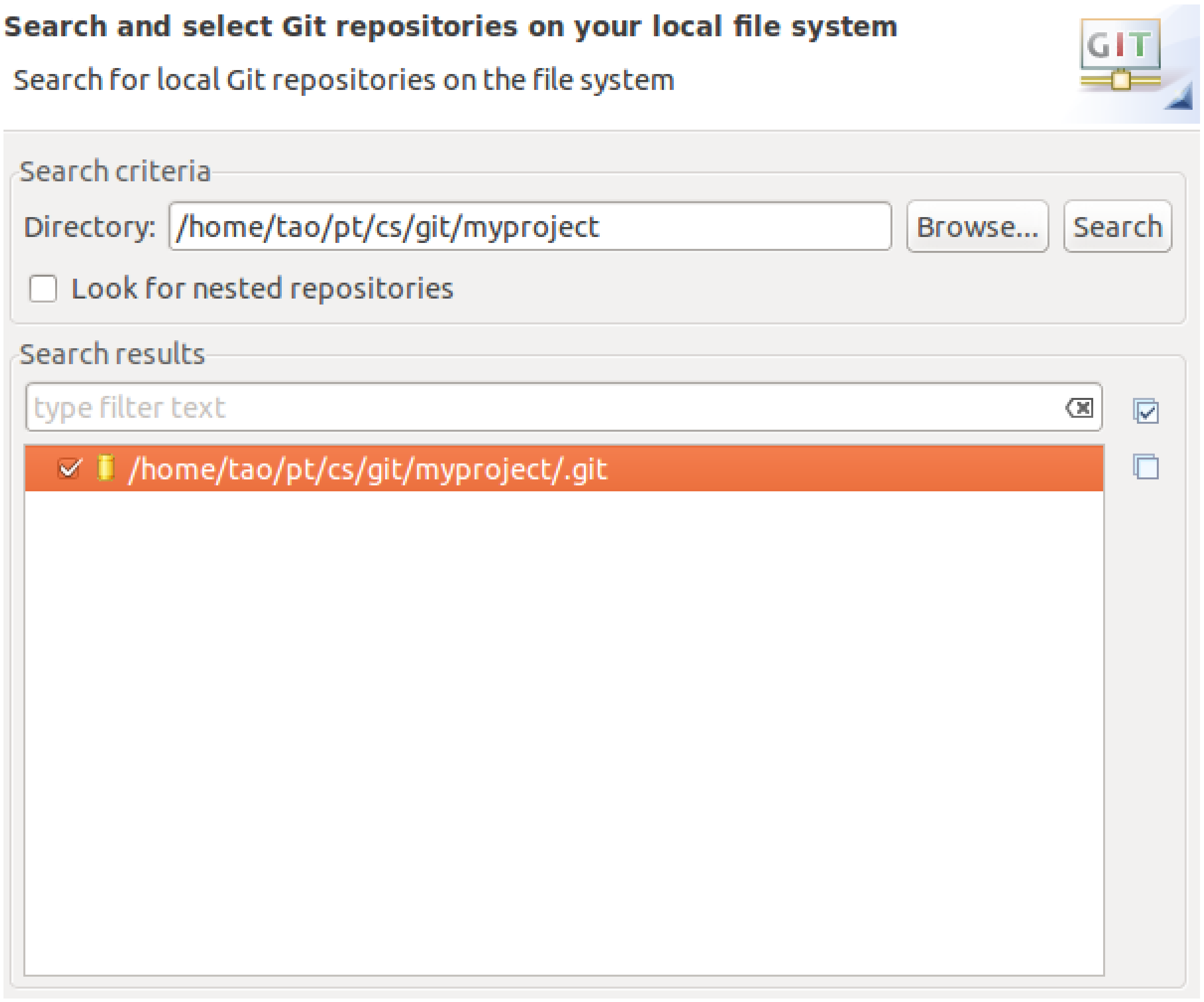
```
new file in master
~
~
```

然後輸入以下命令:
git add newfile
git commit -m "merged newfile"
然後 merge 就完成了

14. Now that we saw some of the git basic functionalities in practice, let's go a step further. If you remember I mentioned before that many of these version control systems are actually integrated into IDEs. So what were going to look at next is what happens if we put together git and eclipse. And the result is egit, or EGit is a plug in for the eclipse IDE that adds git functionality to eclipse. So let's hear how that works in practice. So support for git is available in many ID's including Eclipse. And if you want to get a git hub for Eclipse, you should go to eclipse.github.com and you can download the plugin. So this bring us to the uh,plugin page and you can use their provided URL and directions to install the plugin. In this case we're going to copy this address. So we're going to Eclipse, Help, Install new software. We can 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. In this case I'll just call it eclipse get plugin. Then when I 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. You can accept the terms and conditions for the plug in, and then just finish. And at this point, Eclipse will install the plugin, which might take a little bit of time. So we're just going to speed it up. And 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. And at this point, you want to click ES. And when Eclipse restarts. You'll have your plugin.

(Plugin 裝好後) We're going to go to the git repository perspective that we can select here.

(這個圖標在右上角, 意思是選 perspective, 點了它後, 選 Git) And when we click OK, you can see that our display will change. And 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.
We're going to add an existing local repository (由於我的 git 文件夾沒用默認的路徑, 所以不要選默認的, 要 Browse 自已選, 即如下. 注意 myproject 是 git 中的 project, 它裡面有 README 等文件, 而 MyProject 才是 Eclipse 中的 Java project.). This is the one that we created earlier, so we'll select it and click finish,


Search and select Git repositories on your local file system

Search for local Git repositories on the file system

Search criteria

Directory: /home/tao/pt/cs/git/myproject

Look for nested repositories

Search results

type filter text

/home/tao/pt/cs/git/myproject/.git

and you can see that my project is now added to this set of gift repositories. Now let's check out the project from the repository by selecting import project. And here you can import something as an existing project, you can use a new project wizard, and in this case I chose the option of importing as a general project. Then I click Next and as you can see, I have the project name up there and I can click Finish.
以上的步驟:
右鍵點 myproject->Import Projects->Import as general project->Next->Finish
然後選 Resource perspective(如何選 perspective? 前面講過多次了, 點右上角的那個田字鍵)

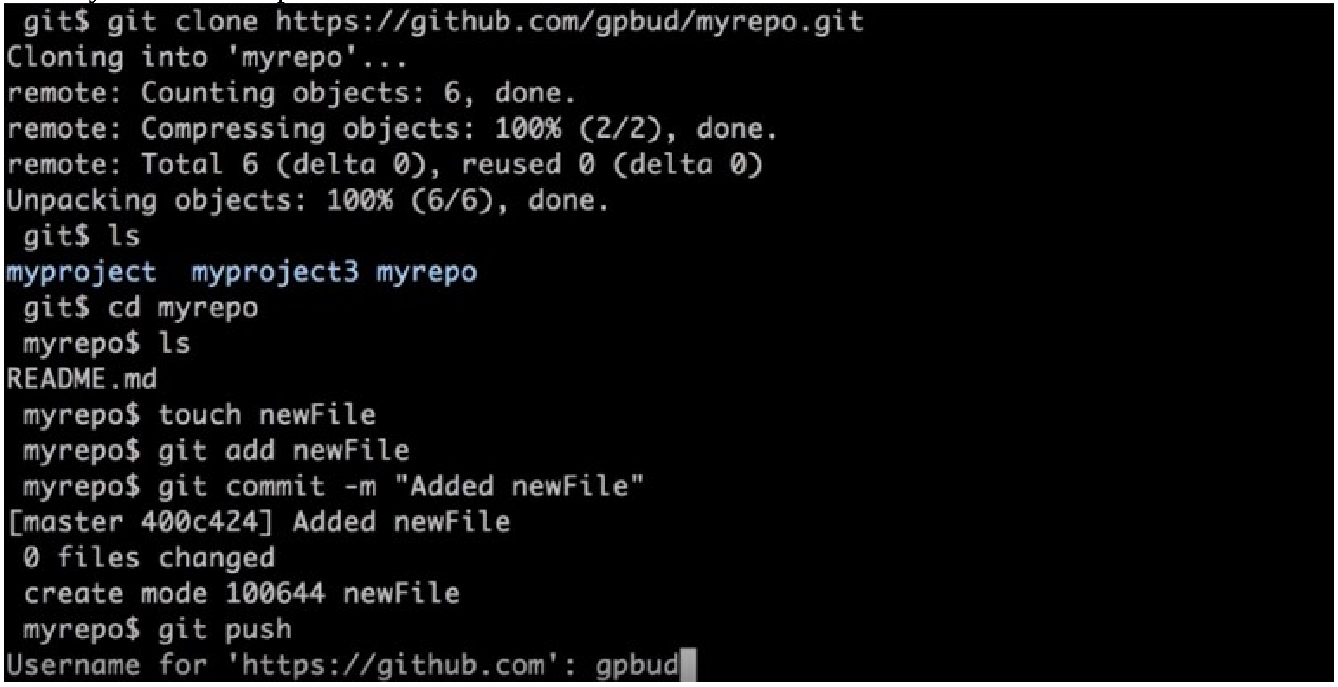
So now, if I go to the resource perspective by clicking here, I can see that the project has been added to my set of projects. And I can see all the files within the project, particularly, if I click on the README, you can see that we have the Readme file that we created before. Same thing for the test file. One thing I can do at this point, it to execute different git commands, perform different git operations by using the team sub manual in the contactual menu. And here there are several things I can do including some advanced commands. And just to give it a shot, I am going to try to click show local history, and this shows the history of the file. For example it shows the author and it shows when he was created, when he was authored.
以上的步驟:
在文件內容中的任何一個地方點右鍵->Team->Show in History

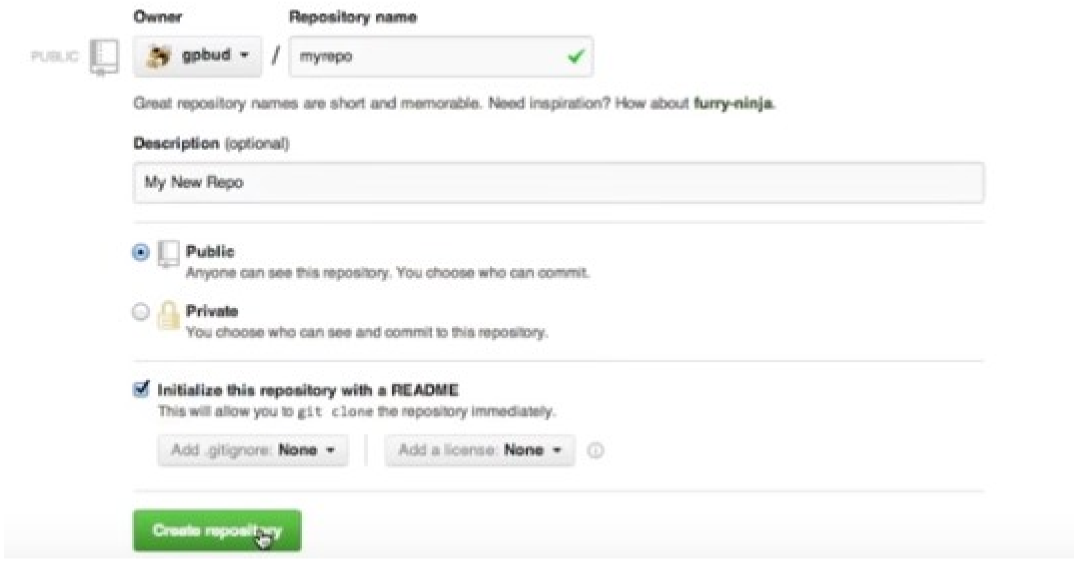
Lets make some changes to this file by adding some new content. Okay. I saved the file and now I can see that error that indicates that my file was locally changed. So now if I go to the team menu, you can see that I have the option to add to the index, to stage the file. And now I got this new label that star that shows the files added to the index. And now at this point, I can go to the team menu again and I can actually commit the file by selecting the corresponding entry. This allows me to enter the commit message, exactly in the same way which I could do that from the command line with the textual editor. And after I put the comment there, I can actually commit. And now if we 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.

以上的步驟:
修改某個文件(如 README)
保存文件(可以按上面的 Save 按鈕)
左上角文件名前出現一個>(即 error, 我覺得他說的應該是 arrow), 表示文件是 locally changed.
在文件內容中的任何一個地方點右鍵->Team->Add to index
左上角文件名前出現一個*(即 star), 表示文件是 added to index.
在文件內容中的任何一個地方點右鍵->Team->Commit
在新窗口中輸入 Commit message(即前面命令行中的 comment, 如 Added more content 等)
然後就可以用 Team menu 就可以 push 到 remote repository 了(如 GitHub 等), 這是下面要講的.

15. In the interview that we did at the beginning of the class, we talked with John about GitHub, where GitHub is a Git hosting website, and John told you all about it. For this class, we will be using GitHub as our Git hosting. Let's see how GitHub works in practice and let's see some of the common features offered by GitHub. This is what we'll do in the third part of this Git demo. What I'm showing here is the GitHub website and as I said, GitHub is a Git hosting website and you can create an account on GitHub by simply signing up on the website. And because we already have an account that we're simply going to sign in to see what kind of functionality GitHub offers. And we're going to specify our username and password. And as you can see on the GitHub website, you can use this menu(右上角那個+號) up on the right to create a new repository or change the account settings. Let's click on our user profile. And here we can see some statistics for our user. For example, we can see statistic about our contributions and our repositories. So now if we go to the Repositories view, we can create a new repository. We give it a name. Let's call it myrepo. We can provide the description for the repository. If we want, we can initialize the repository by adding a README file. And even though we are not doing it right now, if you can see up here, you can also add a license here on the right and it allows you to choose from a set of predefined licenses. And you can also a .gitignore file, which, in case you don't know what that is, it's a very convenient file that will automatically exclude from the repositories file that should not be added. So if you remember in the lesson we said there are things that you should not add to the repositories. For example, derived files. So here, using this menu, you can pick the type of project that you have. For example, Java project or PHP project or many other kinds of projects. And the GitHub will automatically add that file for you. But let's skip that for now and simply create our repository. And that creates a repository that contains the README file because that's what we decided to do. And it also allows you to edit the README file by clicking on it. It will bring up an editor and here you can write, you know, for example, initial readme for your project. Then you can add your commit message up there and then you can commit the changes to your README file. The site also

provides many other features, like, for example, creating issues, pull requests, adding and editing a wiki, and also, you know, defining other characteristics and settings for the repository. Now, if we go to the repository, you can see that we also get the HTTPS link for the repository. So this is the URL that you can use to clone your repository. If you remember, with a git clone command, that's the URL that you can specify. So let's try to do that and clone that repository. So we're going to copy this URL. To do that, we're going to execute git clone and specify the URL that we just copied. And you can see that the project was created, was cloned locally. And if we go under myrepo, which is the name of the repository, you can see that the README file that we created on GitHub is here. So if we create a new file, which we're going to call again, newFile just to be clear. And then we can add it, commit it, specifying as usual a commit message. So at this point, we can push our local changes to the remote GitHub repository. And because the GitHub repository is password protected, we have to specify our login and password. And of course, if you pass the wrong password, GitHub is not going to let you in. So let's try again. Let's try to get the password right this time. I'm going to specify again, my login and my password. At this point, the push is successful and my changes are actually pushed to the master, which is the GitHub repository. To double check that, let's go back to the GitHub repository and as you can see, that the file that we added, newFile, is there as expected. And of course, there's many more things that you can do on the GitHub website, so I strongly encourage you to go and try out things. But the key message here is that the GitHub is a Git hosting website where you can get an account and create your remote repositories.


```
 git$ git clone https://github.com/gpbud/myrepo.git
Cloning into 'myrepo'...
remote: Counting objects: 6, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 6 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (6/6), done.
 git$ ls
myproject  myproject3 myrepo
 git$ cd myrepo
 myrepo$ ls
README.md
 myrepo$ touch newFile
 myrepo$ git add newFile
 myrepo$ git commit -m "Added newFile"
[master 400c424] Added newFile
 0 files changed
 create mode 100644 newFile
 myrepo$ git push
Username for 'https://github.com': gpbud
```


Owner

Repository name

gpbud / myrepo

Great repository names are short and memorable. Need inspiration? How about furry-ninja.

Description (optional)

My New Repo

Public
Anyone can see this repository. You choose who can commit.

Private
You choose who can see and commit to this repository.

Initialize this repository with a README
This will allow you to git clone the repository immediately.

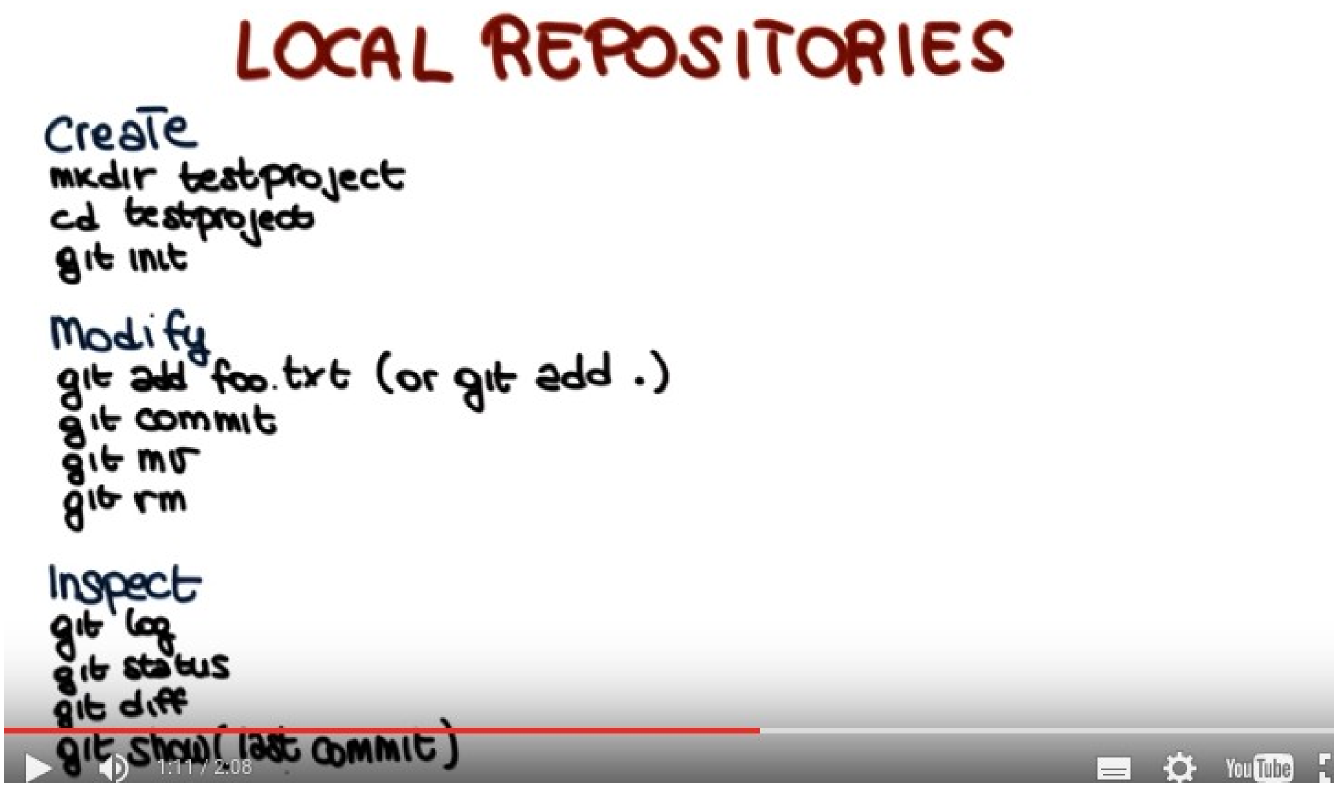
Add .gitignore: None

Add a license: None

Create repository

16. Now that we are done with our demo, I just want to go through a quick GIT recap to remind you of the main commands that we saw, and what they do. And you can also use these as sort of a reference when you work with GIT. And by the way, if you look around and you do a search, you can see that there's tons of examples on the web of GIT type tutorials, videos, examples, manuals. So feel free to explore. And I'm actually going to put some references to tutorials and videos that I found particularly useful in the notes for the class.

17. So, let me start by recapping some of the operations that we can perform on local repositories. I'm just going to list them here and go through them by separating them into three main categories. The first one is commands that, to create a repository and notice that not all of these are git commands, that for example, to create the repository, we would normally want to. Create a directory, which is exactly what we did in our demo. We want to go to that directory and then execute the git init statement, which initializes that directory as a git repository. The second category includes commands that we'll use to modify the content of the repository. We saw that we can use git add to add a specific file or a complete directory to our index. So to the list of files that will be committed, that 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 locker repository, and we can also use git move and git rm or git remove to move files around and to remove files. Finally, the third category is the category of commands that we can use to inspect the concrete repository. And this set includes git log, that we can use to see the log of the repository, git status, that can give us important information about the status of the file center repository. Git diff, that we can use to see the differences between for example, our locker files. And the remote files. And finally git show, that will show us information about our last commit. What we committed, what were the changes and so on. And again, we saw most or all of these commands in our demo. So let me also remind you of a possible workflow. Which again, we already saw but it's always good to go through it once more. And remember that this is just an example. It's just a possible workflow. You can do many different things, you can have many different workflows with git. This is just up to illustrate some of the things that you can do. 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.



LOCAL REPOSITORIES

Create

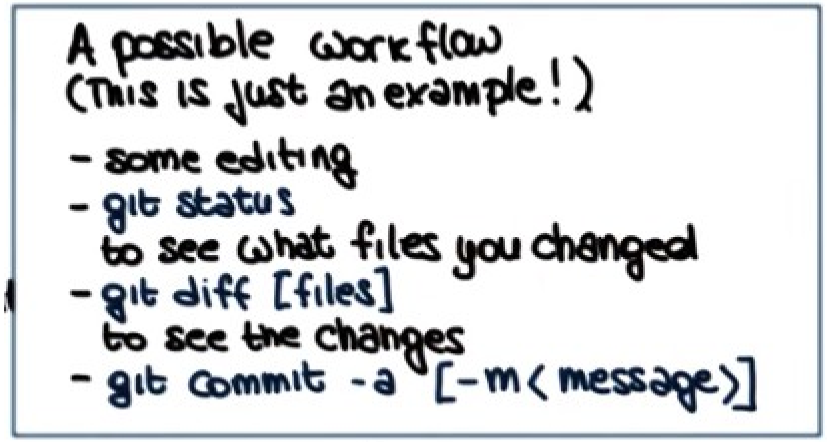
```
mkdir testproject
cd testproject
git init
```

Modify

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

Inspect

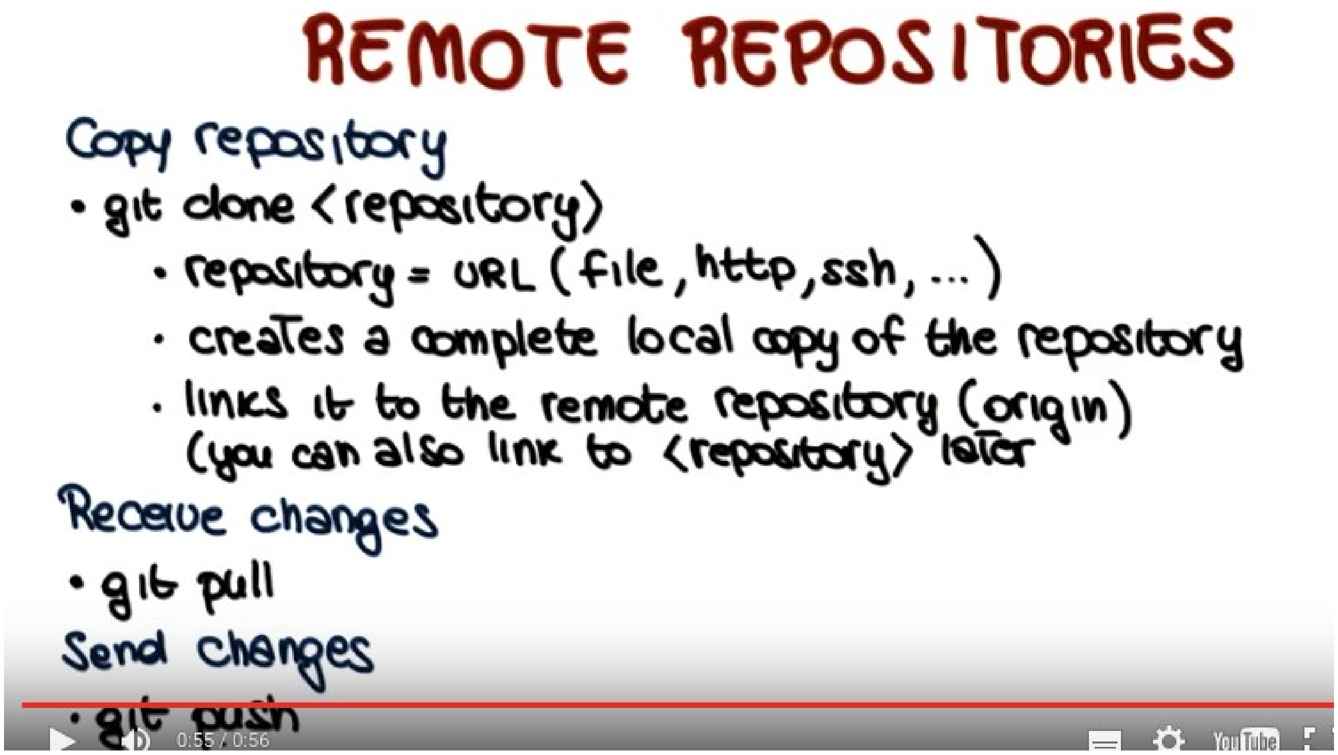
```
git log
git status
git diff
git show (last commit)
```



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>]

18. Similarly, let's go through some commands that you can run on remote repositories. First command is the command to copy a repository, which is git clone in which you get a remote repository and you make a lot of copy in your working directory. The repository can be specified as a URL. It can be a local file, it can be specified 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, as it says, 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. And we saw that you can also perform the same operation through two commands, get fetch and git merge. In case you want to inspect the changes before actually merging them, before actually getting them in your local copy. And if you want to send changes that you have in your local repository to a remote repository, you will use the git push command.



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

以下的來自(節選, 不全)
https://git-scm.com/book/en/v2/Git-Basics-Tagging

Tagging

Like most VCSs, Git has the ability to tag specific points in history as being important. Typically people use this functionality to mark release points (v1.0, and so on). In this section, you'll learn how to list the available tags, how to create new tags, and what the different types of tags are.

Listing Your Tags

Listing the available tags in Git is straightforward. Just type `git tag`:

```
$ git tag
v0.1
v1.3
```

Creating Tags

Git uses two main types of tags: lightweight and annotated.
A lightweight tag is very much like a branch that doesn’t change – it’s just a pointer to a specific commit.
Annotated tags, however, are stored as full objects in the Git database. They’re checksummed; contain the tagger name, email, and date; have a tagging message; and can be signed and verified with GNU Privacy Guard (GPG). It’s generally recommended that you create annotated tags so you can have all this information; but if you want a temporary tag or for some reason don’t want to keep the other information, lightweight tags are available too.

Annotated Tags

Creating an annotated tag in Git is simple. The easiest way is to specify `-a` when you run the `tag` command:

```
$ git tag -a v1.4 -m "my version 1.4"
$ git tag
v0.1
v1.3
v1.4
```

Lightweight Tags

Another way to tag commits is with a lightweight tag. This is basically the commit checksum stored in a file – no other information is kept. To create a lightweight tag, don’t supply the `-a`, `-s`, or `-m` option:

```
$ git tag v1.4-lw
$ git tag
v0.1
v1.3
v1.4
v1.4-lw
v1.5
```

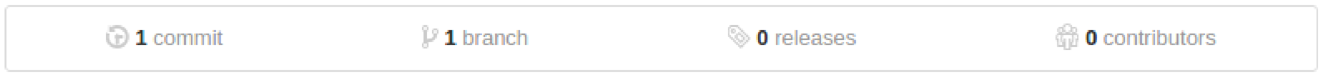
Sharing Tags

By default, the `git push` command doesn’t transfer tags to remote servers. You will have to explicitly push tags to a shared server after you have created them. This process is just like sharing remote branches – you can run `git push origin [tagname]`.

```
$ git push origin v1.5
Counting objects: 14, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (12/12), done.
Writing objects: 100% (14/14), 2.05 KiB | 0 bytes/s, done.
Total 14 (delta 3), reused 0 (delta 0)
To git@github.com:schacon/simplegit.git
 * [new tag]         v1.5 -&gt; v1.5
```

我: 以上的 push 方法在我電腦上不 work, 而要用以下方法:
git tag Version1
GIT_SSL_NO_VERIFY=true git push repo Version1

如何看 tag 是否成功地 push 到 GitHub 上了?
在 GitHub 頁面上:
OMS CS 6300 repo.


1 commit　1 branch　0 releases　0 contributors

如果 releases 變成 1, 則可點进去看 version 數.

如何刪 tag?
到 local root repo(如 6300Spring16tpeng38),
git tag -d Version1
git push origin :Version1 (此前句加 GIT_SSL_NO_VERIFY=true 也不行, 還不如直接在上面那個 GitHub 截圖上點 relases, 进去後有刪的按鈕)

如何刪一個 branch?
git branch -D myBranch
git push origin :myBranch

For assignment 2, from Piazza:

the instructors' answer,
where instructors collectively construct a single answer

You should be able to use either one (我: 指 lightweight or annotated tags.). FYI, we used a lightweight tag for our solution (from which we got the graph).

For assignment 2, from Piazza:

Haven't done the assignment yet, but when you clone a repository, you will only get the master branch locally unless you specify a different one as an argument. "git branch -a" will show you all local and remote branches. The Syncing section here should help - https://www.atlassian.com/git/tutorials/syncing

Even though "$ git branch" only shows the "master", "development" branch is still there. One can see it by using "$ git branch -a" switch. I found the following stack overflow post very helpful in my case.

http://stackoverflow.com/questions/67699/clone-all-remote-branches-with-git

Thank you all.

Andy Parrish 4 days ago
In the User1 folder I did the "git push --all" command, and both branches show up in my Github page. However, when I cloned the REPO for my User2 folder, the development branch is now gone and there's only the master branch. Any suggestions for how to fix this?

Thanks!

Jung-en Wu 4 days ago
Thanks for helping out guys!

Andy Parrish: if you just do $ git checkout development, in Terminal 2 you should be able to switch to development branch. Somehow it is not visible under the command $ git branch

Andy Parrish 4 days ago
Thank you, Jung-en Wu!

```
tao@tao-ThinkPad-T430:~/pt/cs/git/6300/assignment-2/User2/6300Spring16tpeng38/Assignment2$
git checkout development
Branch development set up to track remote branch development from origin.
Switched to a new branch 'development'
```

Part3 #5

1.How to pull changes from the remote repository?
Should I use git pull master?
2. How to deal with the conflicts?

I used a pull. Not sure if it does more than master, but it worked.

Part 3 Error message pushed branch behind its remote

"git updates were rejected because a pushed branch tip is behind its remote"

I checked and all the pulls are up to date

My graph looks like the graph provided by the professor so I know I'm close

shoulld I use Git push master rather than Git push --all?

I had gotten that as well. It looked like it was for the development branch, which is not edited in Part 3.
The master branch was certainly pushed, which the only place that was edited in this part.
 Everything looked correct.

a git push origin master would have been sufficient, it appears, instead of a git push --all origin.

I just did a "git pull", edited the file with conflicts, and then pushed and that worked fine for me.

Alex has a demo in the video lecture on Git (P1L4?) that covers the proper process for resolving conflicts when you git pull.