An Introduction to:

GIT

GIT

- Overview
  - GIT is a version management system which allows you to divide any file into versions
  - GIT allows others to work on the same set of files as you do keeping track of who does what
  - GIT is a collaboration tool which can reside on a server where multiple people can access at the same time

GIT - Installing

- Windows
  - Windows does not include GIT, so you must download and install it
  - Download it at: https://git-scm.com/downloads
  - Start the installation program and accept all defaults
  - You might want to check the Add Icon to desktop option during installation

GIT - Installing

- Linux
  - From a terminal, issue these commands:
    - sudo apt install git
    - sudo apt install gitg
    - sudo apt install gitk

GIT - Installing

- Mac
  - Mac Osx automatically comes with GIT installed
  - You may want to download and install xCode from the Apple store to install all development tools.
    - It’s free and it’s cool!

GIT

Using

GIT - Using

- Once installed, GIT is pretty easy to start up:
- Windows
  - Open the GIT Bash application
- Linux
  - Open a Terminal
- Mac Osx
  - Open a Terminal from Applications->Utilities

GIT - Using

- GIT needs to know who you are, so you need to issue a couple of commands (you only do this once)
  - ***git config --global user.name "Your user name"***
  - ***git config --global user.email your@emailaddress.com***
    - Replace “Your user name” and your@emailaddress.com with appropriate values
  - This will let other GIT users know who you are and how to contact you if you are working collaboratively with them
  - None of this data will be shared

GIT - Using

- You need to create a folder which will contain your files and your GIT repository
  - ***mkdir Documents/gittest***
  - This will create a folder named gittest inside of your Documents folder/directory
    - Folder and directory can be used interchangeably
  - ***cd Documents/gittest***
  - This will take you inside of the gittest folder which is inside of your Documents folder/directory

GIT - Using

- Next, you need to initialize your GIT repository
  - ***git init***
  - This will create a .git folder inside of your Documents/gittest folder/directory
  - GIT uses this folder to hold everything it needs to operator
  - Don’t ever mess with this folder, let GIT deal with it

GIT - Using

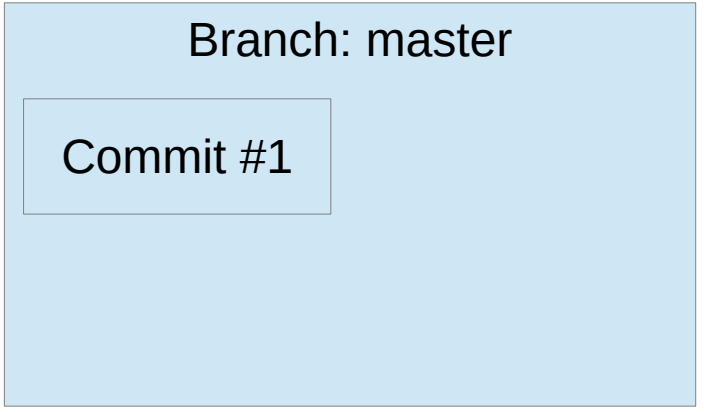
- Create your files
  - Using any type of text editor, create the files you wish to exist in your ***Documents/gittest*** folder
- Add those files to your GIT repository
  - ***git add .***
  - Once added, you will not need to add them again. Their changes shall be tracked by GIT

GIT - Using

- Commit your files
  - Once your files are the way you want them, you need to commit them to the GIT repository
  - ***git commit -a -m “commit message”***
  - -a means commit all changed files
  - -m means the string after it should be used as the comment indicating what is within this commit
- GIT now, not only knows what files it is suppose to manage, but it also contains those files in its repository

GIT - Using

- So, what do we have after the commit?



GIT - Using

- Display your Repository
  - To see your repository and what commits have been done, issue the log command:
    - ***git log -p***
  - This will show what branch you are in and what commits have been done upon this branch
  - The default/first branch name is named master

GIT - Using

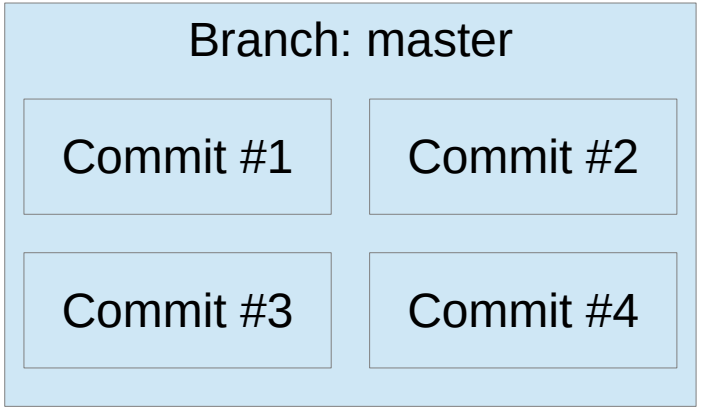
- Tagging your Commits
  - Each commit will have a very large, unusually complex ID
    - Example: **commit 8a99b0cfa065c8b84e84e4477918ab5e5e90fcd5**
  - You can use that ID to reference each commit, or you can tag it with something that is more manageable
    - git tag tag-name 8a99b0cfa065c8b84e84e4477918ab5e5e90fcd5
  - This will give you a nice tag-name to use when referencing your commits

GIT - Using

- More commits
  - You may continue editing your files
  - Each time you are happy with a file and want to ‘save your place’, do a commit
  - Give each commit a valid description so you can tell what you were doing with each version of your code
    - ***git commit -a -m “added user input”***
  - Each time you do a commit, that commit becomes your current and most recent version of your file(s)

GIT - Using

- So, what do we have after the commit?
  - After three more commits

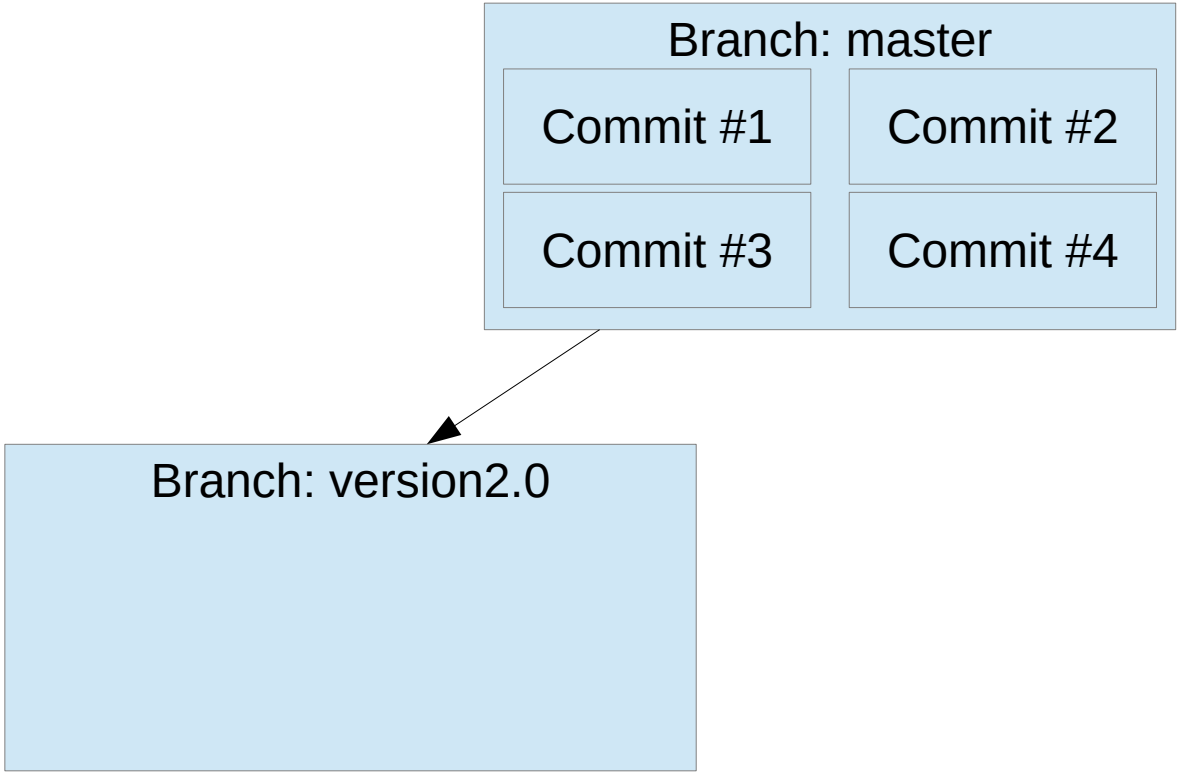


GIT - Using

- When you are done...
  - When you are happy with all of your files in the master branch and would like to start a new version of your files without loosing their current state, you create a branch
    - ***git branch branch-name***
  - This will create a new branch which will contain all of the original files. When you change these new files, the old will stay the same
    - Example: ***git branch version2.0***

GIT - Using

- So, what do we have after the branch?



GIT - Using

- Changing Branches
  - Creating a branch does not mean you are actually using it yet
  - You must ‘check out’ the new branch to begin working on it
    - ***git checkout version2.0***
  - Now, if you issue the command ***git branch***, you will notice your new branch (in this case version2.0) will have an asterisk (*) next to it showing you that it is the current branch
  - After checking out the new branch, you can begin working on it and the master branch will be unaffected

GIT - Using

- Developing your new branch
  - Now you can modify and add new files to this new branch
  - For new files, don’t forget to do a ***git add .*** command to add them to the GIT repository
  - You may do ***git commit -a -m “message”*** commands just as you did with master branch
  - Don’t forget, commit when you get to a point where you want to permanently save your changes to the GIT repository

GIT - Using

- Going back to a previous branch
  - If you wish to go back to a previous branch (like master):
    - Make sure you and and commit all of your current branch's files
    - Issue the ***git branch master*** command to make master the current branch
  - You can move to any branch you wish
  - Now, if you issue the command ***git branch***, you will notice your old branch (in this case master) will have an asterisk (*) next to it showing you that it is the current branch

GIT - Using

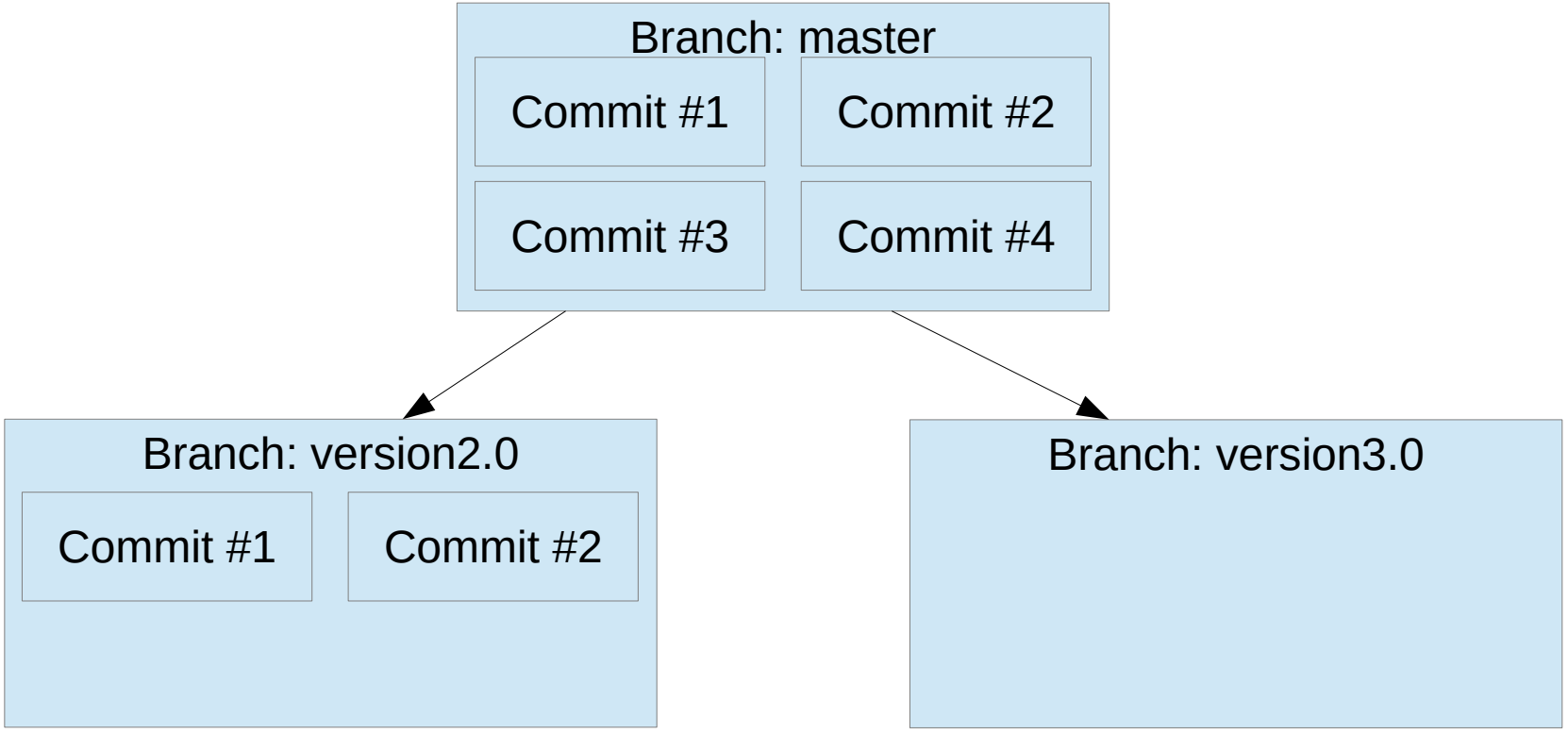
- Going back to a previous branch
  - Also notice that any files you created in your version2.0 branch will no longer be there
  - Any changes you made to the files in your version2.0 branch will also no longer be there
  - This is as it should be. Your master branch won't have the changes you made in your version2.0 branch – that's the magic of GIT

GIT - Using

- Creating yet another branch
  - Now that we're back on the master branch, you can create yet another branch, then check out that new branch
  - As you saw with the version2.0 branch, you will see the master branch's files within this new branch.
  - Any changes or additions you make to files at this point will be part of this third branch affecting no other files in any other branch

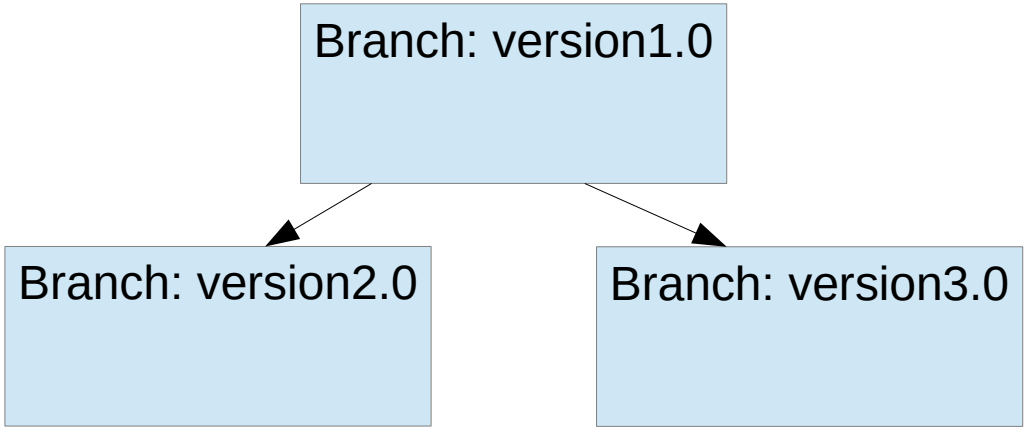
GIT - Using

- So, what do we have after creating the third branch?



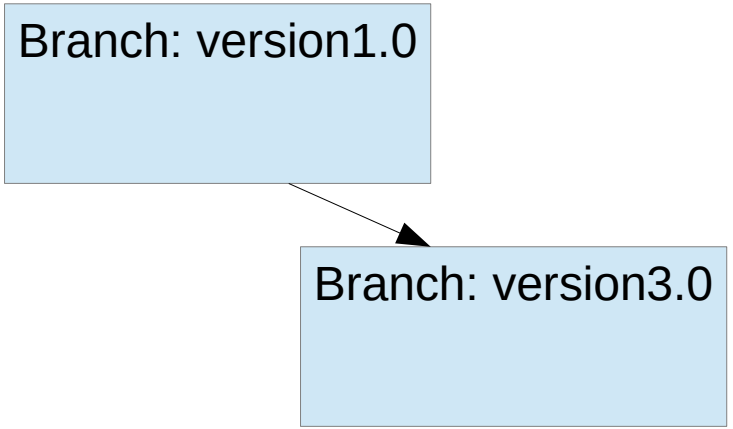
GIT - Using

- Renaming Branches
  - You may also rename a branch
  - For instance, master may not be a good branch name, perhaps version1.0 would be better
    - ***git branch master version1.0***



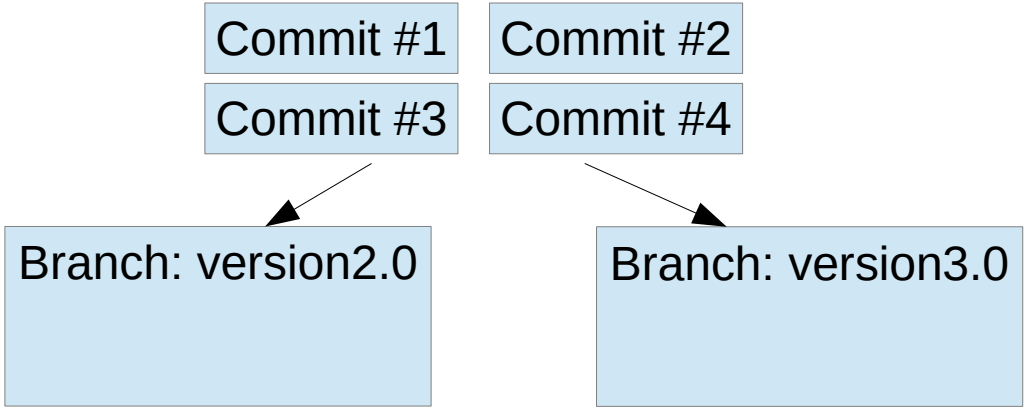
GIT - Using

- Deleting Branches
  - You may also delete a branch
  - For instance, version2.0 is no longer needed
    - ***git branch -d version2.0***



GIT - Using

- Deleting Branches
  - If you delete parent branches (like version1.0), the branch is deleted but the commits are not – because other branches depend on those commits
    - ***git branch -d version1.0***



GIT - Using

- Checking out a Commit
  - If you need to access files from a specific commit, you can also use the git checkout command
  - If you were tagging your commits, this should be easy, if not, you'll need to use that huge commit number
    - ***git checkout tagname***
  - It's best not to change anything in a specific commit, but you can look at and use files from within this commit by copying them

GIT - Using

- More useful displays of your repository
  - The command-line/terminal view of things can leave you confused as to what is going on
  - GIT has a graph argument you can use in the log command to see a much more graphic-like display of your repository
    - ***git log --graph --decorate --all***
  - A more compact version:
    - ***git log --graph --oneline --decorate --all***

GIT

Remote GIT

Remote GIT

- You can use GIT locally (on your computer), or you can transfer your work to a server or both.
- This is basically what GitHub does. It is a set of servers who have GIT repositories enabled and you sync/pull/push data with those servers
- You can do this yourself

Remote GIT

- Getting Started

  1) You need to have access to a userid on a server
  2) You need to create a GIT repository on the server
  3) You add a GIT connection to your local GIT repository
  4) You push your local GIT repository to the server???

Remote GIT

- Creating a Remote Repository
  - Make sure you have an ssh login to the remote server
  - Log onto the server where your ssh ID is located
    - Example: ***ssh -p portnumber userid@servername.com***
  - Make a directory to hold your git repository
    - Example: ***mkdir ~/Documents/gittest***
  - Create the GIT repository
    - ***cd ~/Documents/gittest***
    - ***git init***
    - ***git config --bool core.bare true***

Remote GIT

- Create the GIT connection
  - On your local computer, in your local GIT repository directory, create a connection document
  - ***git remote add connectionName ssh://userid@servername.com:port/repository/location***
    - Example: ***git remote add remoteGittest ssh://userid@servername.com:12345/home/id/Documents/gittest***
  - This document will identify that your local repository is connected to the remote repository
  - You may make as many of these as you like if you need to synchronize with multiple servers

Remote GIT

- Pushing local repository to the remote repository
  - When pushing a branch to your remote repository, make sure it is a branch that is NOT checked out on the remote repository
  - Because our remote is ‘bare’, we don’t have this problem
    - ***git push repository-name branch-name***
    - Example: ***git push remoteGittest Version3.0***
  - To push all branches to the remote repository
    - ***git push repository-name –all***
  - To push all branches and tags to the remote repository
    - ***git push repository-name --mirror***

Remote GIT

- Pushing local repository to the remote repository
  - From here you can make continue to make changes to your local repository and when you’re ready, push back to the remote repository
    - ***git push repository-name***
  - Note: If you created multiple connection documents within your repository, and you wish to update all remote servers with your local changes, don’t specify a repository name in your push
    - ***git push***

GIT - Remote

Cloning from a Remote Server

Remote GIT

- If you want to copy a GIT repository from a server which will be new to you, you can use the git clone statement
  - ***git clone ssh://userid@servername.com :port/server/git/repository/location***
- This statement will create a new directory for you and all of your GIT data
- If you go into the new directory and type ***git branch***, you will not see all branches. You will need to checkout each branch to expose them within the repository

Remote GIT

- As you make changes to this newly cloned repository locally, you can then just push them to the server
  - ***git push***
- All changes shall be synchronized to the remote server

GIT

The End