# Contents



# 1 GIT

## 1.1 1

1. Mac : http://sourceforge.net/projects/git-osx-installer/
2. Windows : http://msysgit.github.io/
3. Linux : apt-get install git-core OR yum install git-core

ABOUT GIT

1. Git is a version control tool that saves changes to groups of files so you can revert back if needed.
2. There are different types of version control tools

a. Local Version Control saves changes to files in a database

b. Centralized Version Control saves changes to a shared server

c. Distributed Version Control allows for easier sharing of files then LVC and also eliminates problems that could occur if access to the server is lost under a CVC system.

d. DVC clients have a complete backup of the files on their computer. If the server is lost the client just waits to regain contact and then uploads changes.

3. When you commit changes to files Git stores a reference of what the files look like at that moment. If a file isn't changed it isn't stored again.
4. Each client has a complete history of all changes stored locally. The client can also access all changes made to the files historically with a simple command. Also those files cannot be changed without Git knowing and changes are difficult to lose.
5. Files transition between 3 states with Git

a. Modified Files are files that have been recently changed

b. Staged Files have been marked to be saved

c. Committed Files are those that have been saved

6. Git saves all file changes to a directory as a compressed database.

a. You modify files in Working Directory

b. You notify that want to save changes in your Staging Area

c. After you Commit the file changes are saved in the Git directory

## 1.2 USING GIT

1. git config –global user.name "Derek Banas"
2. git config –global user.email derekbanas@verizon.net
3. git config –global core.editor "vim" # Set editor as vim
4. git config –global core.editor "edit -w" # Set editor as Text Wrangler Mac

5. git config –list # Show settings
6. git help OR git help [COMMAND] OR git help add
7. ————————- Track a directory ————————-

a. Go to directory
b. ls -a shows all files
c. git init # Creates the .git directory

8. ————————- Start tracking files ————————-

a. By type : git add *.java
b. By name : git add AndroidManifest.xml

9. ————————- Ignore Files ————————-

a. Create a .gitignore file
b. https://github.com/github/gitignore

10. ————————- git commit -m ‘Initial project version’

a. Commits the changes and sets an abbreviated commit message

11. ————————- git status ————————-

a. Shows the state of your files meaning if they are tracked, have been modified and the branch your on.

12. ————————- Stage A Modified File ————————-

a. Change the file and save
b. git diff # Shows what you changed, but haven’t staged
c. git add AndroidManifest.xml # Stage file
d. git diff –cached # Shows what has been staged, but not committed

13. ————————- Commit The Changes ————————-

a. commit # Opens the editor we defined above or vi
b. In vi click [ESC] i to enter insert mode
c. Type a heading that briefly explains the changes in 50 characters or less
d. Describes the original problem that is being addressed
e. Describes the specific change being made
f. Describes the result of the change
g. Describes any future improvements
h. Post a closes bug notation Closes-Bug: #1291621
i. Hit [ESC] and type wq to save and exit
j. git commit -a -m ‘Changed comment’ # Skips staging and commit message

14. ————————- Remove a File ————————-

a. rm DeleteMe.txt # If you remove a file it shows as “Changed but not updated”
b. git status # If you remove a file it shows as “Changed but not updated”
c. git rm DeleteMe.txt
d. git status # Shows that the file was deleted
e. If you have committed a file to be removed you must add the -f option

f. git rm –cached DeleteMe.txt # Keep file, but remove from staging area

g. git mv DeleteMe.txt Delete.txt # Renames a file

15. ————————- Log Commit History ————————-

a. git log # Shows all of the previous commit messages in reverse order

b. git log –pretty=oneline # Shows commits on one line

c. git log –pretty=format:“%h : %an : %ar : %s”

I. %h - Abbreviated Hash

II. %an - Authors Name

III. %ar - Date Changed

IV. %s - First Line of Comment

d. git log -p -2 # Shows the last 2 commit changes

e. git log –stat # Prints abbreviated stats

f. git log –since=1.weeks # Show only changes in the last week

g. git log –since=“2014-04-12” # Show changes since this date

h. git log –author=“Derek Banas” # Changes made by author

i. git log –before=“2014-04-13” # Changes made before this date

16. ————————- Undoing a Commit ————————-

a. git commit –amend # If you want to change your previous commit

b. Normally done if you forgot to stage a file, or to change the commit message

17. ————————- Unstage a File ————————-

a. git reset HEAD AndroidManifest.xml

### 1.3 2

————————- GIT PART 2 ————————-

1. GitHub allows you to host your code repositories online. I’ll set up everything for it in this video. I’ll also cover remote depositories in general.

2. Remote repositories are normally read only, or read write only to those who are authorized

3. You either push or pull updates from these remote repositories.

4. To push your directory to GitHub ————————-

a. git init

b. git add . # Stages all new and modified files and directories

c. git commit -m ‘Initial Project Version’

d. git remote add origin https://github.com/derekbanas/SimpleFragment.git

e. git push origin master

5. git remote -v # Lists all remotes and their URLs

6. git fetch origin # Gets data from the remote, but it doesn’t merge changes with your work

7. git pull https://github.com/derekbanas/SimpleFragment.git

a. Pulls all changes and saves them to your directory

8. How to push changes to GitHub ————————-

a. I add .gitignore for Android on my local machine

b. In the terminal type

I. git add .gitignore # Stage .gitignore

II. git commit -m 'Added .gitignore for Android' # Commit

III. git push # Push the changes to GitHub

9. git remote rename origin sf # Renames remote to sf

10. Tagging ————————-

a. Tags are used to tag files at important points in history

b. git tag -a v0.1 -m 'version 0.1' # Creates an annotated tag

c. git tag # Shows all the tags

d. git show v0.1 # Shows details about the commit that was tagged

e. git tag v0.4-lw

I. Creates a lightweight tag on a commit that stores the hash for the commit

f. git tag -a v0.01 c930a8

I. You can tag commits after the event also. When you enter this command an editor opens for you to leave a comment. The final part is the hash for the commit you want to tag.

g. git push sf v0.1 # You can also push tags

I. The tag shows up under releases on GitHub

II. git push sf –tags # You can also push all tags at once

h. You can set aliases to save time

I. git config –global alias.co commit

II. Now you can type git co to commit

i. Clone a GitHub Repository

I. Go to the directory you want to use

II. git clone https://github.com/derekbanas/google-api-nodejs-client.git

### 1.4 3

————————- GIT PART 3 ————————-

1. Branching allows you to take a project in your own direction without effecting the main code. You use them to make sure you don't introduce unstable code to the master branch.

2. When you commit a project :

a. Each file is given a hash code

b. A tree object that contains those files and the associated hash codes receives a hash code

c. A commit object stores a reference to the tree and other data like the author, commit comment, a reference to the previous commit and other data.

3. The default branch is called the master. As you make additional commits the current newest version is referred to as the master.

4. If you create a branch you can go in a different direction with the project without effecting the master until you merge. Each branch creates a new pointer to a committed version of files and doesn't make another copy of the files.

5. We can actually create many branches, but be careful while doing this because it can get to be hard to merge multiple branches. A pointer known as HEAD can be pointed at any committed version or to any branch with the checkout command.
6. When you are finished with your branch you can merge back into the master commit and move on. You can also do all of this locally or on a remote.
7. ————————- Simple Branch Example ————————-

a. git checkout -b fix20 # Create a branch and switch to it

I. Same as : git branch fix20 git checkout fix20

b. Change AndroidManifest.xml in vim
c. git commit -a -m 'Added Branch fix20' # Commit the change to the branch, but not to master
d. git checkout master # Switch to master
e. git push origin fix20 # Push the branch to GitHub
f. git fetch origin # If someone else fetches from the server they get a reference to the branch on the server but not all the files
g. git checkout -b fix20 origin/fix20 # Retrieves the branch fix20
h. git branch # Shows all branches
i. git branch –merged # Shows all merged branches
j. git branch –no-merged # Shows unmerged branches
k. git branch -v # Shows all branches and their last commits
    - – Points out the branch currently checked out
l. git merge fix20 # Merge the branch version with the master git push # Push the change to GitHub
m. git branch -d fix20 # You can delete merged branches with this
n. git branch -D fix22 # Deletes unmerged branches
o. git push origin :fix20 # Deletes the branch on GitHub
p. git branch -m newBranchName # Renames a branch

8. ————————- Multiple Branch Example ————————-

a. a. git checkout -b fix21 # Create a branch and switch to it
b. Edit AndroidManifest.xml
c. git commit -a -m 'Added Branch fix21'
d. git checkout master # Switch to master
e. Look at Manifest to see that nothing changed
f. git checkout -b 'hotfix' # Create a new branch
g. git commit -a -m 'Added Hot Fix' # Commit the Hot Fix
h. git checkout master # Switch to master
i. git merge hotfix # Merge the hotfix version with the master
j. git branch -d hotfix # Delete the hotfix branch
k. git checkout fix21 # Switch to fix21 branch
l. git checkout master # Make sure you are in master
m. git merge fix21 # Merges the branch and master if there are no conflicts

n. If there is a conflict resolve it

o. git branch -d fix21 # Delete the unneeded branch git branch -D fix21 # To force delete

p. git mergetool # You can merge with a graphical tool

I. Backup : Contents of the file before calling the merge tool

II. Base : The common ancestor of the files being merged

III. Local : Version being pointed at by HEAD

IV. Remote : The branch being merged into head

9. ————————- Rebasing Example ————————-

a. Rebasing moves a branch to a new ( master / base ) commit. This is also referred to as a fast forward merge. Just never rebase commits that have been pushed to a public repository

b. git checkout -b fix22

c. Edit AndroidManifest.xml

d. git commit -a -m 'Changed the comment to 10'

e. git checkout -b hotfix

f. Edit another file other then AndroidManifest

g. git commit -a -m 'Edited file. . . .'

h. git checkout master

i. git merge hotfix

j. git branch -d hotfix

k. git checkout fix22

l. git rebase master # Move branch to new master commit

m. git checkout master

n. git merge fix22

10. ————————- Reverting Vs. Resetting Example ————————-

a. Some times you want to eliminate a previous commit, but you still want to keep the commit for integrity reasons. Revert undoes changes made in that commit and makes a new commit. Reset actually deletes the commit which can cause problems.

b. Do something that will be undone

c. git commit -m 'Made a change that I will undo'

d. git revert HEAD # You are back to where you started, but the commit was made

e. Reset eliminates previous commits and you can never get them back. You really should never use it actually.

f. git reset someFile # Removes a file from the staging area, but leave the working directory unchanged

g. git reset # Reset the staging area to match the most recent commit while leaving the working directory unchanged

h. git reset aCommit # Move back to this previous commit, reseting the staging area, but not the working directory

i. git reset –hard # Reset both the staging area and working directory to match the most recent commit

j. git reset –hard aCommit # Move back to the commit listed and change staging and working directory

11. ————————- Clean Example ————————-

a. Clean removes untracked files from your directory and is undoable.

b. git clean -n # Shows which files will be removed

c. git clean -f # Remove untracked files

d. gir clean -df # Remove untracked files and untracked directories in the current directory

e. git reset –hard # Undoes changes on all tracked files git clean -df # Removes all untracked files

### 1.5 4

————————- GIT PART 4 ————————-

1. Here I'll demonstrate a workflow option with git called Fork 7 Pull. I'll also answer some questions I've received like how to work with multiple GitHub accounts on the same computer.

————————- GENERATING SSH KEYS ————————-

2. SSH keys allow you to identify trusted computers without the need for passwords and here I'll show you how to generate multiple codes for multiple GitHub accounts.

a. ssh-keygen -t rsa -C "Your Email Address" # Generates the key

3. Then you have to define the name of the file you want to save the key in

4. A public key and a randomart image are generated. The randomart image is provided because it is easier to recognize then a random string of numbers.

5. cd ~/.ssh # Takes us to the location of our keys

6. I'll open the public key with vim

7. Now I'll copy the entire key from ssh-rsa till the end with my email

8. Got to GitHub and sign in

9. Click on account settings

10. Click on SSH

11. Give the key a name, paste in the key and click add key

12. Your public key is then listed

————————- CREATING MULTIPLE GITHUB ACCOUNTS ————————-

13. Now I'll create another ssh key for a completely new account on GitHub ssh-keygen -t rsa -C "Your Email Address" # Generates the key

14. Give it a name

15. A public key and randomart are generated again

16. cd ~/.ssh # Takes me to the location for the keys ls # Lists everything in the directory

17. Go to GitHub again using a different GitHub account and click Account Settings

18. vim newthinktank.pub # Get the new key that was generated

19. Copy the key

20. After you click SSH Keys in the sidebar on GitHub, paste the key in, give it a title and click Add Key

21. We used a unique name for our keys so we have to tell ssh about them. ssh-add ~/.ssh/derekbanas ssh-add ~/.ssh/newthinktank

22. touch ~/.ssh/config # Creates the empty file config vim config # Open config

23. We are defining which account we want to work with by associating a keyword to our 2 different hosts.

Host github.com HostName github.com User git IdentityFile ~/.ssh/derekbanas

Host github-ntt HostName github.com User git IdentityFile ~/.ssh/newthinktank

24. Change to the directory you want to use on GitHub

25. This is the account on GitHub that holds the original master files

26 - 28. After I edit some files I stage them and commit them

29. git remote add myorigin git@github-ntt:newthinktank/CrazyTipCalc.git

    a. Create an alias for our remote directory

    b. github-ntt # I identify myself using the newthinktank ssh key

    c. newthinktank/CrazyTipCalc.git # The specific files I want on GitHub

30. git push myorigin master When you try to push to GitHub you may see this warning. You can verify you are pushing to GitHub by comparing the public keys

31 - 34. You can see GitHubs public keys here https://help.github.com/articles/what-are-github-s-ssh-key-fingerprints

35 - 37. Log into GitHub using my derekbanas account and search for the directory I have associated with my newthinktank account

38 - 41. Get the URL from GitHub and I can clone it git clone https://github.com/newthinktank/CrazyTipCalc.git

———————- FORK & PULL WORKFLOW ———————-

The way the Fork & Pull works is that anyone can Fork a repository and make changes locally. They don't have the ability to push their potentially damaging code. They can however request that the host repository pull their changes if they would like using a Pull Request. ( This is a very common workflow in the open source community )

42. Find a repository you'd like to work on and click Fork in GitHub

43 - 45. git clone https://github.com/derekbanas/CrazyTipCalc.git - Get the URL for the fork on GitHub and clone it on your local computer

46. git remote add upstream https://github.com/derekbanas/CrazyTipCalc.git
    - Assigns the original remote and not the fork to the keyword upstream

47. git fetch upstream
    - Pull in changes made in the original repository with effecting the local files

48 - 53. I can change a file locally, stage and commit it. I can then push it to more Fork on GitHub.

54. git merge upstream/master # Merges files on GitHub with my local files

55. If I think my changes should be merged with the original repository I can make a Pull Request. Click on Compare, review, create a pull request on GitHub.

56. Changes are listed as well as other data associated with your Forked version. Click the button labeled Create Pull Request if you think you're ready.

57 - 58. Leave a detailed reason why the Pull Request should be excepted and click Send Pull Request

59. The owner of the original repository can see how many Pull Requests they have received on the right side of the screen.

60 - 63. They can go through the Pull Requests and decide what to do.

64 - 65. To merge click Merge Pull Request in GitHub and then leave a detailed explanation why it was done.