GIT Notes

1. **What is git? Give us a brief history on it.** Git is an open source, distributed version control system designed for speed and efficiency
2. **What are the differences between git and svn?** The main difference about git compared to svn is that it is decentralized. There is one repository that is set up that can be referred to as the master repository. From there everybody in the group would be able to clone the repository to their own computer to work on. Unlike svn, git allows us to commit files to our local repository without using the internet. Once an internet connection is reestablished, the user would be able to commit that work to the main repository. Git is considered to be more complex than svn, considered that it adds in the push, checkout, clone and etc., as to where svn is considered to be simple to learn. With that complexity though, git is also said to be faster than svn and have better support for branching and merging, which gets rid of more errors.
3. **What is the purpose of the .getignore file?** If you create a file in your repository named .gitignore, git will understand to ignore the rules in that file for that repository. It will not ignore a file if it had been previously tracked though. Inside the gitignore file, the user is able to list file name extensions that that the user would like git to not track.
4. **Explain the following commands:**



* 1. **Commit –** After adding to the index, commit is used to record changes to the local repository. This allows only the user of the local machine to have access to the changes. Every git commit is a snapshot of the tree along with references to its parent commits.
  2. **Push –** After committing to the local repository, the user may push the changes up to the remote repository. This is how everybody else would be able to access your files.
  3. **Pull –** Git pull is what the user would to do update the user’s local repository with the remote repository. Git pull in simplest terms does a git fetch and a git merge at the same time.
  4. **Merge –** The git merge command is used to merge one branch with another while preserving the ancestry of each commit history. The user should commit any changes before doing the merge to help avoid errors. The merge is done by taking the pointer of each location of the branches being merged, one being the main branch and the other being the branch being merged from, and pointing to a new head in the main branch.
  5. **Rebase –** This can be used to rewrite the history of your commits.
  6. **Fetch** – Git fetch almost does the same thing as git pull. It will pull down from the remote repository and creates a local copy of the remote branch. The only difference is that it does not merge it with the rest of the repository. It is more or less just downloading the files from the remote repository.

1. **What is the stash, and how is it used?** The stash takes the uncommitted, or unfinished work, from the working copy and stashes it away onto a type of stack. If the user wants to stash away work from another branch, that work would be pushed on top of the stack and able to be called later by popping it off.
2. **What GUI programs are available for Windows? Mac?** 
   1. **Windows**
      1. GitHub for Windows – Free
      2. Git Extensions – Free
      3. SourceTree – Free
      4. Git-cola – Free
      5. SmartGit - $79 / user / Free for non-commercial use
      6. GitEye – Free
   2. **Mac**
      1. GitHub for Mac – Free
      2. Tower – $59 / user (Free 30 day trial)
      3. Gitbox - $14.99
      4. GitX-dev – Free
      5. SourceTree – Free
      6. Git-cola – Free
      7. SmartGit - $79 / user / Free for non-commercial use
      8. GitEye – Free
3. **What repository providers are available online?** 
   1. Assembla
   2. Beanstalk
   3. Bettercodes.org
   4. Bitbucket.org
   5. CloudHost
   6. Codebase
   7. CloudForge
   8. Deveo
   9. GitEnterprise
   10. GitHub
   11. GitLab Cloud
   12. Gitorious
   13. Pikacode
   14. ProjectLocker
   15. Repo.or.cz
   16. SourceForge.net
   17. Unfuddle
4. **What are the best practices for branching when developing several features at once?** 
   1. Try to avoid committing unfinished work to your repository’s master working branch.
   2. Create a branch any time you begin non-trivial work including features and complex bug-fixes.
   3. Don’t forget to delete feature branches once they were merged into stable branch. This will keep your repository clean.
   4. Use your repository’s default working branch as your “development” branch.
   5. Create a branch for each environment, including staging and production.
   6. Never merge into an environment branch unless you are ready to deploy to that environment.
   7. Perform a diff between branches before merging – this can help keep debug and dirty code from being deployed, and can also help with writing release notes.
   8. Merges should only flow in one direction: from feature and bug-fixes done in their own branch or in development into staging for testing. Once tested, you can merge those changes from development into production.
5. **What is and how do you make a pull request?** I think we answered this one in defining what pull is?

**The 3 examples are after this, but I don’t know how everybody wants to go about doing these.**