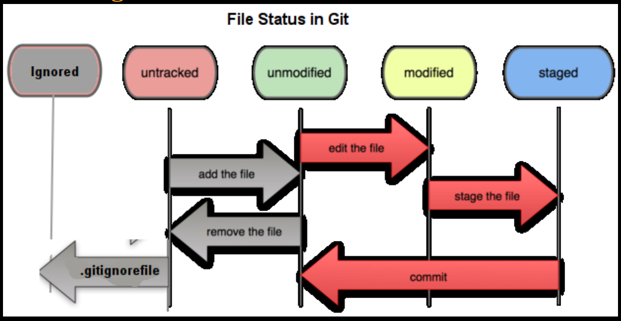
GIT Notes:

Sean al-Baroudi

October 18, 2017

1 Intro to GIT and GITHub [?]:

- GIT is a Version control system that saves snapshots of files (as opposed to differences in files such as CVS and SVN), that runs on a local computer or server.
- GITHub is an online set of repositories for the world community to share code and projects. It is on the internet.
- The basics of GIT can be seen in the diagram below:
- **ADD Diagram**



- Files themselves (that are tracked by GIT) have a number of statuses: Including:
  1. **Ignored:** Use the *.gitignore* file to exclude files from version control.
  2. *Untracked:* This is a file that GIT does not maintain; a default new file added to the project.
  3. *Unmodified:* A file that is tracked, but not changed.
  4. *Modified:* A file that is tracked, and changed compared to whats in the last commit of the project.
  5. *Staged:* **A copy of a modified file** that will be added to the commit. **Note:** Changing a tracked file outside of the staged area will not update the staged file; you need to re-add the new change.
- In addition to these statuses, there is one more state a file can be in: **committed** (a local copy stored in the hashed project database).

Summary of Basic Git Commands:

- **git init**: In a folder of your choosing, run this to start GIT.
- **git add:** A multi-use command that adds files to the tracked set, or adds file to the Staged area.
- **git clone:** this pulls a git repo from a given source. Use it to work on someone elses project. Usage:

```
git clone <repo url>
```

- **git status:** indicates what files have been changed, and the status of various files in the directory. *Use frequently to assess file stages and merge problems.*
- **git diff:** on its own indicates differences between staged and modified files; in other words, **unstaged differences**.
- **git commit -m:** pushes current staged files to the repository. To make amends, and not make a new snapshot (such as adding a forgotten file:

```
git add <forgotten files>
git commit --amend
```

- **git mv:** Renames files; **this is assumed to change files in the staged area.**
- **git log:** shows the log of commits that have occured.
- **git rm [filenames...]** Do this to remove a file. If done without git's knowledge, checkouts can cause deleted files to return again. This removes a file from the staged area. **Note:** Deleting a file from the local folder does not affect the staged file. Also: to untrack, you need the –cached option.
- **gitk:** Gives you a graphical representation of the history of the project, and shows all branches. Use this to be spatially aware of project merges and changes.

Git Branching:

- Branching is the process of traversing or creating another "line of development" - where LOD refers to a particular direction, or version of code a project follows.
- You can branch code to develop your own version, try out new ideas, or experiment with code.
- **Usage Patterns:**
  1. Developers have a **main production branch, and development branch**. For on the fly fixes and development, they switch back and forth between them.
  2. A hotfix branch is made to diagnose a problem in code. It is then merged to the production code.
- Branches allow you to completely change your code base: as GIT is just snapshots, new directories of code can easily just be switched in and out by playing with inodes on the file system.
- Branch History is stored in a linked list of Snapshots. Snapshots contain a:
  - **Commit Data Structure**, which links to:
  - **Hash Tree Data Structure:** which points to the project directory folder structure and
  - **Hashed Items**, which are referenced by the Hash Tree fields.
- A git branch then is a pointer that references a particular snapshots directory tree
- Default Git Branch: "MASTER".
- Concept Distinction: Commit History and Branches. The tree of historical commits is always present in the .git folder. Each running "branch" is just a pointer variable to a particular snapshot along the commit tree.
- There can be multiple branch pointers. There is one HEAD meta-pointer, that points to a particular branch pointer. Analogously, this is *dereferenced*, and its particular directory structure is loaded into your work space
- To create a new branch pointer:

```
git branch <name>
```

- To switch to a new branch:

```
git checkout <name>
```

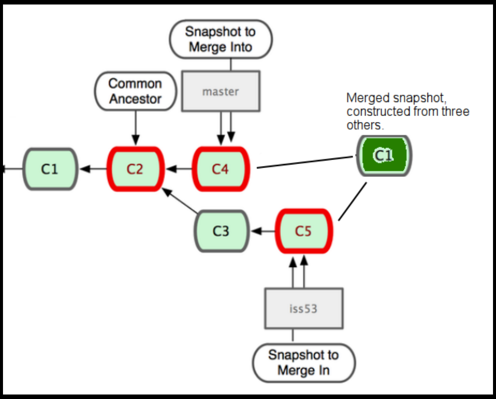
- Delete a branch pointer:

```
git branch -d <name>
```

- **Local vs Remote:** The local branch is whatever you have last checked out. Remote refers to any other branch not checked out (on server, in history for merge, etc).

Merging:

- Merging is the process of combining two or more snapshots into one.
- Types of Merge (Topological Description):
  1. **Fast-Forwarding:** If one snapshot is upstream of another snapshot, git just has to move the "Master" or "Deployed" pointer forward.
  2. Forked Branch Merge: When merging two branches that are forked, **Git uses the two current branch pointer snapshots, and the common anscestor** to create a new snapshot, which points to both of the merged snapshots.

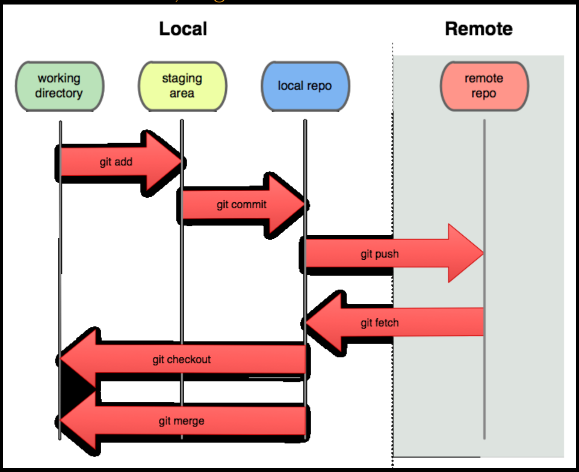
- 

- Merging (without File Conflict): When work on Branch X is done, to merge with branch Y, do the following:

```
git checkout Y
git merge X
```

- **Merging with File Conflict:** The commands are the same, but now differences have to be resolved [?]. Do the following:

```
git checkout Y
git merge X
git status [lists file conflicts]
git mergetool [kdiff3]
rm -f *.orig [if any are lying around]
git add <fixed files>
git commit -m "message"
```

- **Note:** The snapshot that is not checked out is considered to be the "remote" one, in github.
- 

-
-
-
-
-
-
-
-

Remote Repos:

- To get a copy of a remote repo, use the following:

```
git clone <url to repo>
```

- When this is done an **Origin:Master** pointer will be created, that refers to the snapshot last seen on the remote server.
- As you do work on your copy, the origin and your copy will fall out of synch. To join (not merge the two) and create a branch about a common ancestor, do the following:

```
git fetch origin
```

- **Pushing:** Involves taking your local copy of code, and overwriting code on a remote server. To push, execute the following:

```
git push origin <branch name>
```

2 Corrections:

None