GIT

Rob Hayward

November 4, 2014

1 Notes from the Github team

These are notes taken from the You Tube videos Github and Git foundations: training

- git diff HEAD will show all the changes since the last commit.
- git diff −−stat
- git log −−oneline just records the commits
- git log −−patch gives an full details of the commits. Can be combined with −−oneline
- git rm will remove the file from the file system. However, it does not remove the file from history.
- git add −u will add any deleted files to the staging system
- git rm −−cached filename will tell git not to track the filename file any more. This is useful if a file is tracked that you don't want tracked.
- git mv will move files and stage.
- git add −A . will identify all files that have been moved. The dot tells git to start at the current working directory and move all the way down.
- git log −M −−follow tells the log to follow the file across moves.
- git ls−files −−others −−ignored −−exlude−standard will shows all the files that git is ignoring.

Similarity index (which comes at the end of the log) shows how similar a file is before and after a move takes place. The 50% level if a cut off to determine whether git considers it to be a move or a new file.

Git ignore will allow files to be ignored. Details are in the .gitignore file. It contains the details of the files to be ignored. Ignored directories will also ignore the subdirectories.

branch

When working on a project, new features should be carried out on a branch. The master branch is the main branch.

- git branch <name> give the branch a new name.
- git branch −d <name> to delete the branch. You will be warned if it has not been merged in.
- git branch −D <name> will not give the warning.
- git checkout <name> will switch to a new branch. When you switch branches, any work in the staging area of the commit area will come across. Cannot switch to a new branch if any of the files in the staging area or working directory would be overwritten.

Files may appear to disappear and reappear as you change branches. These files will remain in their respective branches.

1.1 checkout

Used for changing branches.

- git branch will tell you what branch you are on.
- git status also states the name of the branch
- git checkout <commit ref> will re-write the working tree from that commit. This is a 'detatched HEAD'. This is a picture of what the working directory looked like at that time. Make sure you checkout back to a branch.
- To discard edits, git checkout −−<filename> will wipe that files contents from the last commit.
- git checkout −b <newbranchname> will create the branch and check out to it.

1.2 merge

This will bring branches and multiple lines of history together.

- git merge will being the history of two or more branches together.
- Move to the branch that you want to merge to and git merge <branchfilename>
- To resolve a conflict, find the file from the message or find it from git status. Open file with texteditor and find the conflict.

- {<<<<}HEAD shows the file in the branch that is currently checked out to.
  \item ======== is the divide with the branch that is being merged in
  \item \verbatim{>>>>>>} <nameofbranchthatisbeingmerged> The end of the conflict.
  \item Remove the conflict, run status, stage the file, commit the file.
  \item \lstinline{git merge --abort} if you do not want to deal wth the conflict. Ths
  \item Remove the branch after the merge.

2 Other items of interest

The git stash command will save all the changes that have been made on the local directory and will update the working directory to match the head. git show will show what has been saved.

3 Learn Git

This is a walk through Try Git. From the required directory,

```
git init
```

This will initialise a git repository. This means creating an empty repository in git. This is a hidden directory where git operates.

```
git status
```

Will tell us the current state of the project.

```
git add octocat.txt
```

Will tell git to start tracking the changes in the file octocat.txt. It will send the file to the staging area.

There are a number of different levels:

1. staged: files are ready to be commited
2. unstaged: files with changes that have not been prepared to be commited
3. untracked: files that are not yet tracked by git
4. deleted: files that have been deleted and waiting to be removed from git

```
git status
```

Will show the changes that are now ready to be commited. The files listed are in the staging area.

```
git commit –m "Add cute octocat story"
```

This will commit the file and add the comment.

Wildcards * can be used. For example

```
git add '*.txt'
```

Quotes are important. This will also take files from the root directory as well as the new directory. All the text files are added to the staging area. Now commit

```
git commit –m "Add all the octocat files"
```

Use

```
git log ––summary
```

to see more information on each commit. This includes information on when files are added and deleted.

It is possible to add to the existing commit if there is some mistake and there is a small ammendment.

```
git commit ––amend
```

This is a good way of keeping changes that are related to each other together in one place.

3.1 Remote server

An empty GitHub repository has been created. To push the local repo to the GitHub server a remote repository is added. This command takes the remote name and the repository URL.

```
git remote add origin https://github.com/try-git/try_git.git
```

The push command tells Git where to put the commit when it is ready. Now the local changes are pushed to the **origin** repo on GitHub. The name of the remote is **origin** and the default local branch name is **master**. The **-u** item tells Git to remember the parameters so that the next time we can simply run **git push** and Git will know what to do.

```
git push –u origin master
```

The main remote is usually called **origin**

To check for changes on the GITHub repository and pull down any changes run

```
git pull origin master
```

Sometimes when you go to pull you may have changes you don't want to commit just yet. One option is to *stash* the changes. Use the command

```
git stash
```

to stash the changes and

```
git stash apply
```

to reapply the changes after the pull.

3.2 changes

If there are changes since the last commit, these can be identified with 'git diff' command.

```
git  diff HEAD
```

In this case, we are looking for the difference since the last commit. This is identified with the HEAD command.

The head is the pointer that holds your position within all the different commits. By default, HEAD points to your most recent commit, so it can be used as a quick way to reference that commit without having to look a the SHA.

The diff function can also be used to look at differences within files that have already been staged. Remember that staged files are files that we have told git are ready to be committed.

it is a good idea to keep related changes together in separate commits. Using 'git diff' gives you a good overview of the changes that you have made and lets you add files or directories one at a time and commit them separately.

```
git  diff --staged
```

The –staged option will let you see changes that you just staged.

You can also type git add .. The dot represents the current directory, so everything in it, and everything beneath it gets added.

3.3 Unstage

Files can be unstaged by using the 'reset' function.

```
git reset octofamily/octodog.txt
```

Will take octodog.tex out of the staging area. If you want to go back to how things were before octodog emerged you need to use checkout. For example,

```
git checkout --octocat.tex
```

will get rid of all changes since the last commit for octocat.tex

The '–' pointer tells git that there are no more options.

4 Branching

When developers are working on a feature, they often create a copy (a branch) of their code so that they can make separate commits to it. When this is completed, it can be merged back into the master branch.

For example, to remove all the octocats, create a branch called 'clean up' for the work.

```
git branch clean_up
```

Branches are what naturally happens when you want to work on multiple features at the same time. You don't want the master to have feature A half done and feature B half done.

```
git branch
```

will show two branches: master and clean up. Switch can be made to new branch

```
git checkout clean_up
```

```
git checkout -b new_branch
```

Can be used to checkout and create a branch at the same time.

5 Clean up

Now in the clean up branch, it is possible to clean up what has been done. Using the 'git rm' command, the files can be removed from the disk and stage the removal of the files.

```
git rm '*.txt'
```

The wildcard will get rid of all the files in one go.

To remove an entire folder, use the recursive option on 'git rm'

```
git rm -r folder_of_cats
```

This will recursively remove all folders and files from the given directory.

Now files (and directories) have been removed, they can be committed.

```
git commit -m "Remove the cats"
```

It is probably best to run 'git status' first to check the changes that are to be committed.

If you delete a file without 'git rm', you will have to use 'git rm' to get rid of the files from the working tree. The -a option on the 'git commit' command will auto remove deleted files with the commit.

```
git commit –am "Delete stuff"
```

Now that the problem of the cats has been resolved, switch back to the master so that the changes in the branch can be copied or merged to the master.

A pull request can all someone to look through changes before allowing the merger. There is more here on GitHub Help.

```
git checkout master
```

6 Merge

Now the changes from the clean up will be merged with the master branch. We are already in the master branch.

```
git merge clean_up
```

If there is a conflict, a decision has to be make about which code takes priority. There is more about this in the Pro Git Book in the section about how conflicts are presented.

To complete, delete the clean up branch using

```
git branch –d clean_up
```

If you want to get rid of a feature the 'git branch -d bad-feature' command may not work if you were deleting something that has not been merged. In this case, add –force (-f) option or -D which combines -d and -f together into one command.

7 Git Push

The final step is to push what you have been working on to the remote repository.

```
git push
```

8 More Information

- Git Documentation
- Pro Git Book
- Git Hub Training

9 Notes from Pro Git Book

10 Other Notes

This is an answer to a question about pulling the origin/master when the local is not up to date.

```
git fetch --all
git reset --hard origin/master
```

The first line will fetch everything from the origin and the second will overwrite all local files. Note on Stack Overflow