2018

Git Recap : One Million Arab Coder

Reference : [Full-Stack Track - One Million Arab Coders]

Ola S. Darwish

Udacity

3/7/2018

Table of Contents

Git basics

Git Init Recap

Use the git init command to create a new, empty repository in the current directory.

$ git init

Running this command creates a hidden .git directory. This .git directory is the brain/storage center for the repository. It holds all of the configuration files and directories and is where all of the commits are stored.

Git Clone Recap

The git clone command is used to create an identical copy of an existing repository.

$ git clone <path-to-repository-to-clone>

This command:

- takes the path to an existing repository
- by default will create a directory with the same name as the repository that's being cloned
- can be given a second argument that will be used as the name of the directory
- will create the new repository inside of the current working directory

---

Git Status Recap

The git status command will display the current status of the repository.

$ git status

I can't stress enough how important it is to use this command *all the time* as you're first learning Git. This command will:

- tell us about new files that have been created in the Working Directory that Git hasn't started tracking, yet
- files that Git *is* tracking that have been modified
- a whole bunch of other things that we'll be learning about throughout the rest of the course ;-)

Git Log Recap

Fantastic job! Do you feel your Git-power growing?

Let's do a quick recap of the git log command. The git log command is used to display all of the commits of a repository.

$ git log

By *default*, this command displays:

- the SHA
- the author
- the date
- and the message

...of every commit in the repository. I stress the "By default" part of what Git displays because the git logcommand can display a lot more information than just this.

Git uses the command line pager, Less, to page through all of the information. The important keys for Less are:

- to scroll down by a line, use j or ↓
- to scroll up by a line, use k or ↑
- to scroll down by a page, use the spacebar or the Page Down button
- to scroll up by a page, use b or the Page Up button
- to quit, use q

We'll increase our git log-wielding abilities in the next lesson when we look at displaying more info.

---

git log --oneline Recap

To recap, the --oneline flag is used to alter how git log displays information:

$ git log --oneline

This command:

- lists one commit per line
- shows the first 7 characters of the commit's SHA
- shows the commit's message

---

git log --stat Recap

To recap, the --stat flag is used to alter how git log displays information:

$ git log --stat

This command:

- displays the file(s) that have been modified
- displays the number of lines that have been added/removed
- displays a summary line with the total number of modified files and lines that have been added/removed

---

git log -p Recap

To recap, the -p flag (which is the same as the --patch flag) is used to alter how git log displays information:

$ git log -p

This command adds the following to the default output:

- displays the files that have been modified
- displays the location of the lines that have been added/removed
- displays the actual changes that have been made

What does git show do?

The git show command will show *only one commit*. So don't get alarmed when you can't find any other commits - it only shows one. The output of the git show command is exactly the same as the git log -pcommand. So by default, git show displays:

- the commit
- the author
- the date
- the commit message
- the patch information

However, git show can be combined with most of the other flags we've looked at:

- --stat - to show the how many files were changed and the number of lines that were added/removed

- -p or --patch - this the default, but if --stat is used, the patch won't display, so pass -p to add it again
- -w - to ignore changes to whitespace

Git Add Recap

The git add command is used to move files from the Working Directory to the Staging Index.

$ git add <file1> <file2> … <fileN>

This command:

- takes a space-separated list of file names
- alternatively, the period . can be used in place of a list of files to tell Git to add the current directory (and all nested files)

Git Commit Recap

The git commit command takes files from the Staging Index and saves them in the repository.

$ git commit

This command:

- will open the code editor that is specified in your configuration
    - (check out the Git configuration step from the first lesson to configure your editor)

Inside the code editor:

- a commit message must be supplied
- lines that start with a # are comments and will not be recorded
- save the file after adding a commit message
- close the editor to make the commit

Then, use git log to review the commit you just made!

Git Diff Recap

To recap, the git diff command is used to see changes that have been made but haven't been committed, yet:

$ git diff

This command displays:

- the files that have been modified
- the location of the lines that have been added/removed
- the actual changes that have been made

Git Ignore Recap

To recap, the .gitignore file is used to tell Git about the files that Git should not track. This file should be placed in the same directory that the .git directory is in.

Merge Conflict Recap

A merge conflict happens when the same line or lines have been changed on different branches that are being merged. Git will pause mid-merge telling you that there is a conflict and will tell you in what file or files the conflict occurred. To resolve the conflict in a file:

- locate and remove all lines with merge conflict indicators
- determine what to keep
- save the file(s)
- stage the file(s)
- make a commit

Be careful that a file might have merge conflicts in multiple parts of the file, so make sure you check the entire file for merge conflict indicators - a quick search for <<< should help you locate all of them.

Git Tag Recap

To recap, the git tag command is used to add a marker on a specific commit. The tag does not move around as new commits are added.

$ git tag -a beta

This command will:

- add a tag to the most recent commit
- add a tag to a specific commit *if a SHA is passed*

Git Branch Recap

To recap, the git branch command is used to manage branches in Git:

 # to list all branches

```
$ git branch


# to create a new "footer-fix" branch

$ git branch footer-fix


# to delete the "footer-fix" branch

$ git branch -d footer-fix
```

This command is used to:

- list out local branches
- create new branches
- remove branches

Recap Of Changes

We've made the following changes:

1. on the master branch, we added a default color to the page
2. we created a sidebar branch and added code for a sidebar
3. on the master branch, we changed the heading of the page
4. on the sidebar branch, we added more content to the sidebar
5. we created a footer branch and added social links to the footer

These changes are all on their own, separate branches. Let's have Git combine these changes together. Combining branches together is called **merging**.

Merge Recap

To recap, the git merge command is used to combine branches in Git:

```
$ git merge <other-branch>
```

There are two types of merges:

- Fast-forward merge – the branch being merged in must be *ahead* of the checked out branch. The checked out branch's pointer will just be moved forward to point to the same commit as the other branch.
- the regular type of merge
    - two divergent branches are combined
    - a merge commit is created

Merge Conflict Recap

A merge conflict happens when the same line or lines have been changed on different branches that are being merged. Git will pause mid-merge telling you that there is a conflict and will tell you in what file or files the conflict occurred. To resolve the conflict in a file:

- locate and remove all lines with merge conflict indicators
- determine what to keep
- save the file(s)
- stage the file(s)
- make a commit

Be careful that a file might have merge conflicts in multiple parts of the file, so make sure you check the entire file for merge conflict indicators - a quick search for <<< should help you locate all of them.

Changing The Last Commit

You've already made plenty of commits with the git commit command. Now with the --amend flag, you can alter the *most-recent* commit.

```
$ git commit --amend
```

Revert Recap

To recap, the git revert command is used to reverse a previously made commit:

```
$ git revert <SHA-of-commit-to-revert>
```

This command:

- will undo the changes that were made by the provided commit
- creates a new commit to record the change

Reset Recap

To recap, the git reset command is used erase commits:

$ git reset <reference-to-commit>

It can be used to:

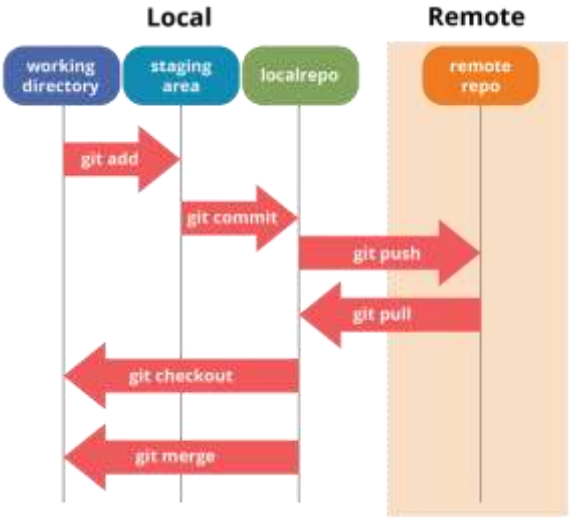
- move the HEAD and current branch pointer to the referenced commit
- erase commits with the --hard flag
- moves committed changes to the staging index with the --soft flag
- unstages committed changes --mixed flag

Typically, ancestry references are used to indicate previous commits. The ancestry references are:

- ^ – indicates the parent commit
- ~ – indicates the first parent commit

Recap :

Expected Knowledge



- ***creating repositories with git init and git clone***
- ***reviewing repos with git status***
- ***using git log and git show to review past commits***
- ***being able to make commits with git add***
- ***commit them to the repo with git commit***
- ***you need to know about branching, merging branches together, and resolving merge conflicts***
- ***and being able to undo things in Git:***
    - ***git commit --amend to undo the most recent commit or to change the wording of the commit message***
    - ***and git reset If you're comfortable with all of these, then you'll be good to go for this course.***

Collaborative work with other

Remote repository

Recap

A remote repository is a repository that's just like the one you're using but it's just stored at a different location. To manage a remote repository, use the git remote command:

$ git remote

- It's possible to have links to multiple different remote repositories.
- A shortname is the name that's used to refer to a remote repository's location. Typically the location is a URL, but it could be a file path on the same computer.
- git remote add is used to add a connection to a new remote repository.
- git remote -v is used to see the details about a connection to a remote.

Push changes to a remote:

Recap

The git push command is used to send commits from a local repository to a remote repository.

$ git push origin master

The git push command takes:

- the shortname of the remote repository you want to send commits to
- the name of the branch that has the commits you want to send

---

pull changes from a remote:

Recap

If there are changes in a remote repository that you'd like to include in your local repository, then you want to *pull* in those changes. To do that with Git, you'd use the git pull command. You tell Git the shortname of the remote you want to get the changes from and then the branch that has the changes you want:

$ git pull origin master

When git pull is run, the following things happen:

- the commit(s) on the remote branch are copied to the local repository
- the local tracking branch (origin/master) is moved to point to the most recent commit
- the local tracking branch (origin/master) is merged into the local branch (master)

Also, changes can be manually added on GitHub (but this is not recommended, so don't do it).

Recap: git pull vs git fetch

You can think of the git pull command as doing two things:

1. fetching remote changes (which adds the commits to the local repository and moves the tracking branch to point to them)
2. merging the local branch with the tracking branch

The git fetch command is just the first step. It just retrieves the commits and moves the tracking branch. It *does not* merge the local branch with the tracking branch. The same information provided to git pull is passed to git fetch:

- the shorname of the remote repository
- the branch with commits to retrieve

$ git fetch origin master

Forking a repo

Recap

Forking is an action that's done on a hosting service, like GitHub. Forking a repository creates an identical copy of the original repository and moves this copy to your account. You have total control over this forked repository. Modifying your forked repository does not alter the original repository in any way.

Reviewing existing work

Recap

The git log command is extremely powerful, and you can use it to discover a lot about a repository. But it can be especially helpful to discover information about a repository that you're collaborating on with others. You can use git log to:

- group commits by author with git shortlog
- $ git shortlog
- filter commits with the --author flag

- $ git log --author="Richard Kalehoff"
- filter commits using the --grep flag
- $ git log --grep="border radius issue in Safari"

Determining what to work on

Recap

Before you start doing any work, make sure to look for the project's CONTRIBUTING.md file.

Next, it's a good idea to look at the GitHub issues for the project

- **look at the existing issues to see if one is similar to the change you want to contribute**
- **if necessary create a new issue**
- **communicate the changes you'd like to make to the project maintainer in the issue**

When you start developing, commit all of your work on a topic branch:

- **do not work on the master branch**
- **make sure to give the topic branch clear, descriptive name**

As a general best practice for writing commits:

- **make frequent, smaller commits**
- **use clear and descriptive commit messages**
- **update the README file, if necessary**

NEXT

Pull request

A pull request is a *request* for the source repository to pull in your commits and merge them with their project. To create a pull request, a couple of things need to happen:

- you must *fork* the source repository
- clone your fork down to your machine
- make some commits (ideally on a topic branch!)

- push the commits back to *your fork*
- create a new pull request and choose the branch that has your new commits

---

stay in sync with source project

 # to make sure I'm on the correct branch for merging

$ git checkout master

 # merge in Lam's changes

$ git merge upstream/master

*# send Lam's changes to *my* remote*

$ git push origin master

Recap

When working with a project that you've forked. The original project's maintainer will continue adding changes to their project. You'll want to keep your fork of their project in sync with theirs so that you can include any changes they make.

To get commits from a source repository into your forked repository on GitHub you need to:

- get the cloneable URL of the source repository
- create a new remote with the git remote add command
    - use the shortname upstream to point to the source repository
    - provide the URL of the source repository
- fetch the new upstream remote
- merge the upstream's branch into a local branch
- push the newly updated local branch to your origin repo

---

managing active pull request PR

Recap

As simple as at may seem, working on an active pull request is mostly about communication!

If the project's maintainer is requesting changes to the pull request, then:

- make any necessary commits on the same branch in your local repository that your pull request is based on
- push the branch to the *your* fork of the source repository

The commits will then show up on the pull request page.

NEXT

---

squashing commits

Recap

The git rebase command is used to do a great many things.

 # interactive rebase

$ git rebase -i <base>

 # interactively rebase the commits to the one that's 3 before the one we're on

$ git rebase -i HEAD~3

Inside the interactive list of commits, all commits start out as pick, but you can swap that out with one of the other commands (reword, edit, squash, fixup, exec, and drop).

I recommend that you create a backup branch *before* rebasing, so that it's easy to return to your previous state. If you're happy with the rebase, then you can just delete the backup branch!
