Post02: Additional Git Commands

Jackson Leisure

12/03/2017

Git used to Git Commands

Contents

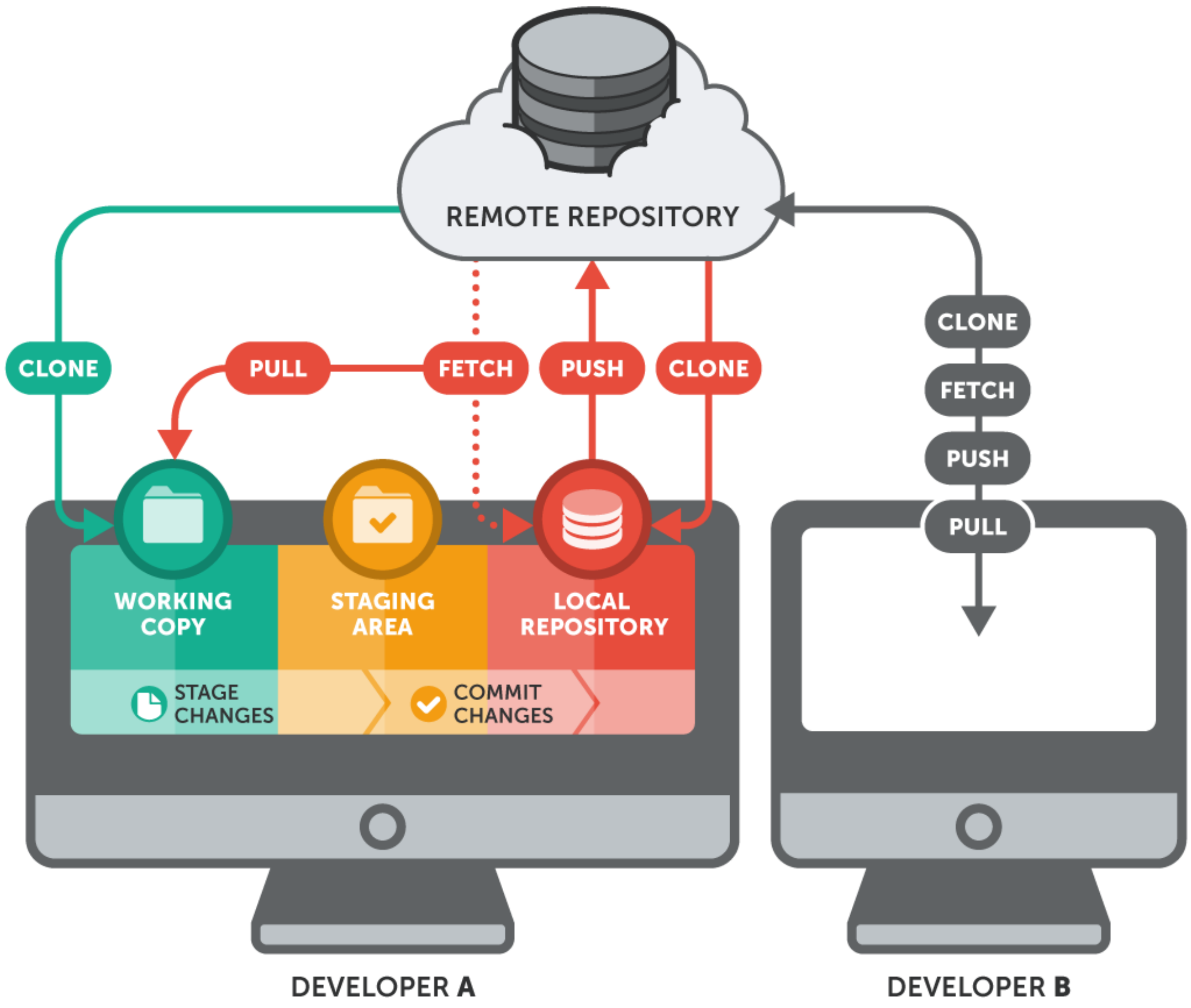
Introduction

As we learned in the beginning of class, Git is a version control system which allows us to keep track of multiple changes within our files and file structure. Additionally, Git allows us to update our files remotely, such as when we submit assignments such as homeworks and posts. The beauty of Git is its simplicity, speed, and ability to store multiple versions of a single project simultaneously, while allowing these versions to eventually be merged into one. Due to these capabilities, Git is an extremely useful piece of software, not only for individual projects, but also for groups who would like to work on various parts of a single project, without disrupting others' work or causing difficulty when trying to merge the group's work into a single working project.

In this class, we have learned a little about Git's capabilities, but we have not covered most of the commands Git has to offer. Here, I will discuss what has already been covered in class, before expanding and discussing other things which can be accomplished with Git. Feel free to follow along as I create and modify a Git repo.

What we learned in class

Lecture material



In Lecture, we received a general overview of what Git is used for, as well as some basic commands. To begin, let us recall some key concepts:

- Repository: A database which contains information describing all of the changes which have been made to the files contained within. A repository is the backbone of your project; you send changes to the repository and you can recover old changes via this structure.
- Commit: Submitting selected files to the repository is called ‘committing’. A commit is, in essence, a ‘snapshot’ of the changes you have made, which is then stored in the repository tree. If you need to recover an earlier version of your project, you can revert to a previous commit, and your project will be restored to your ‘snapshot’.
- Working Directory: The area in which you make changes to your project. This folder is the ‘root’ of the repository and all changes to the repository occur here.
- Staging Area: Directory which contains the files you would like to include in your ‘snapshot’. Sometimes, you do not want to commit certain changes you have made, so you only stage the desirable changes when you make a commit. This area keeps track of those desirable changes.
- HEAD: A file which contains a reference to the ‘current’ commit. This allows Git to keep track of which commit you are currently working on, so the most up to date commit is known.

In the lecture introducting Git. We learned two important commands:

- `git config` allows us to configure out user profiles. Try it now:

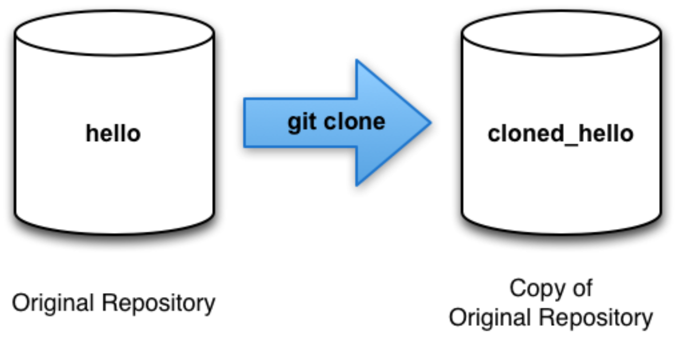
```
# configure your name
git config --global user.name "[YOUR NAME]"
# configure your email
git config --global user.email "[YOUR EMAIL]"
#list the current user profile information
git config --list
```

- `git init` allows us to create a new Git repository in the the current directory. Try it now:

```
# create a directory
mkdir myRepo
cd myRepo/
# initialize a Git repository
git init
```

These two commands are enough to get us started with Git. The rest of our introduction to Git was covered in lab.

Lab material



In Lab, we went over a few more Git commands. These new commands gave us the ability to upload files to our github accounts, in order to allow for assignment submission. Let's go over these commands again:

- `git clone [url]` allows us to make a local copy of a remote repository. This way, we can have our own version of any repository which we can use as a basis for creating changes to a project. As an example, I found it easier to clone the entirety of the repository for this class (rather than just the hws, as we were instructed to do in lab). This way, I could hold all of the data necessary for lab in one easy place, without having to manually download each time. Let's try this now:

```
cd ..
# clone the repository from the class
git clone https://github.com/ucb-stat133/stat133-fall-2017.git
cd stat133-fall-2017/
ls
```

- `git status` informs us as to what changes have taken place since the last commit. Try it now:

```
cd ..
cd myRepo/
ls
#create a text file
> file1.txt
ls
# check the status
git status
```

- `git add [file]` allows us to stage a file. Since we have created a new file in this repository, `git status` has informed us that our new file is not being tracked (it was not present in our last commit). Let's fix this by adding the file to the staging area, so we will be able to include it in our next commit. Try it now:

```
# stage the file
git add file1.txt
git status
```

- `git commit -m [message]` takes a snapshot of our staged files, and stores this in the history of the project. We can recover past commits, as every commit contains references to the commit it came from. It is important to have meaningful commit messages, so that another person (or future you) will be able to understand exactly what changes each commit contains. Try it now:

```
# track the file
git commit -m "initialized repository and created our first text file!"
git status
```

- `git remote add [alias] [url]` allows you to associate this repository with a remote repository (you can also disassociate this repository from a remote repository with `git remote rm [alias]`). This allows you to make changes on one device, push changes to a remote repository, and pull the changes on another device, allowing for a project to be edited from multiple locations. To execute this next step, you'll have to create a repository in your github account, so things will line up. Go ahead and create a new private repository called "myRepo". Next, Try out adding a remote and verifying that it worked properly:

```
# add the remote with the alias 'origin'
git remote add origin https://github.com/[YOUR GITHUB USERNAME]/myRepo.git
git remote -v
# remove the remote with the alias 'origin'
git remote rm origin
git remote -v
# add the remote with the alias 'origin' again
git remote add origin https://github.com/[YOUR GITHUB USERNAME]/myRepo.git
```

- `git push` allows you to send your local changes to the remote repository. This way, your changes can be pulled in the future from another local repository. After executing this command, your github repository should be updated with the new text file we created. Try it it now:

```
git status
# send the changes to the remote
git push origin master
git status
```

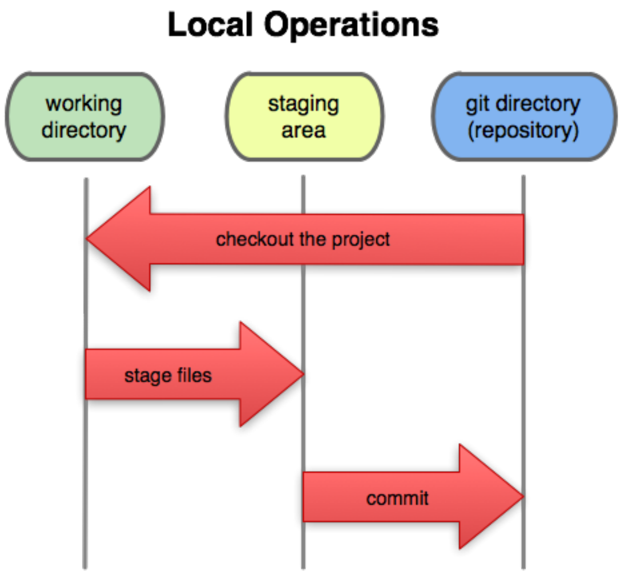
End of material covered in class

This concludes the section on what we learned in class. It doesn't feel like very many commands, but these commands allowed us to update our github repository with the necessary changes to submit our assignments for this class.

Useful Git commands not covered in class

Now that we've recalled the basics, we can expand on these ideas. Git is much more powerful than what we have covered in class, and a solid understanding will allow you to manipulate different versions of your project with ease. I will go through some categories of actions, and explain the relevant Git commands in each section.

Staging and Snapshots



These commands relate to the staging area and working with snapshots.

- `git status` is covered in class.
- `git add [file]` is covered in class.
- `git reset [file]` unstages a previously staged file. This command allows you to undo a `git add [file]` command, if you have changed your mind about prepping a file to be committed. Try it now:

  ```
  # insert text into the file
  echo "Hello, world." >> file1.txt
  git status
  git add file1.txt
  git status
  # unstage the file
  git reset file1.txt
  git status
  ```

- `git diff` allows us to see what changes we have made but have not yet staged. This helps us decide which files we actually want to stage. Try it now:

  ```
  # check the difference
  git diff
  ```

- `git diff --staged` allows us to see what changes we have staged but have not yet committed. This is similar to the above command, but helps us see whether we actually want to commit the files we have staged. Try it now:

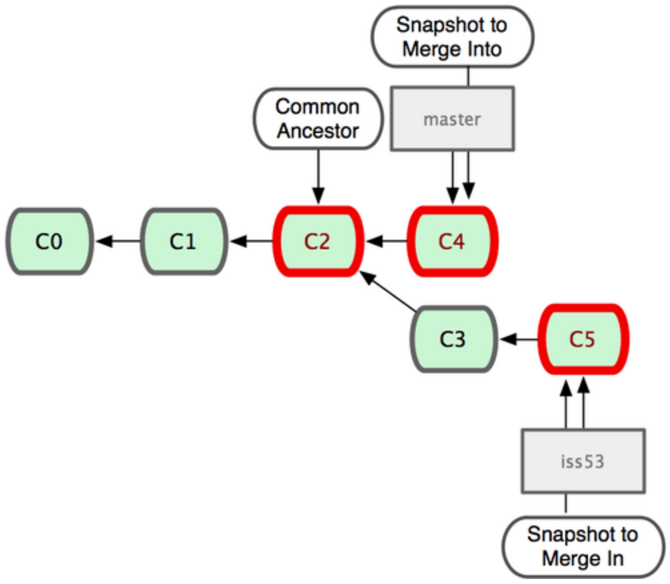
  ```
  git add file1.txt
  # check the difference
  git diff --staged
  ```

- `git commit -m [message]` is covered in class.

Let's commit and push these changes, just to clear everything out:

~~~
```
git status
# add all untracked files
git add -u
git status
git commit -m "hello world in file1.txt"
git status
git push origin master
git status
```
~~~

Branching and Merging



These commands relate to creating new branches in Git and merging changes to files.

Another useful definition: in Git, a “branch” is a node in the repository’s commit tree; branches help you work on different versions of the same project, without having changes from one affect the other. This is one of the core reasons as to why VCS like Git exist.

- `git branch` displays all current branches; the active branch is highlighted with an asterisk. This allows you to see all the different branches currently in your project. Try it now:

  ```
  # display all current branches
  git branch
  ```

- `git branch [branch name]` creates a new branch in your project. This branch stems from your currently active commit, and will allow you to have another parallel version of your project. Try it now:

  ```
  # create a new branch names 'branch1'
  git branch branch1
  git branch
  ```

- `git checkout [branch name]` allows you to switch which branch you are currently working on. Now, all changes made will apply to the new branch, rather than the last branch you switched from. Try it now:

  ```
  # switch to branch1
  git checkout branch1
  git branch
  ```

- `git merge [branch name]` merges the history from the specified branch into the current branch. This allows you to incorporate changes from another branch into your main project branch if you decide that the changes were what you wanted. Try it now:

  ```
  echo "Changes from branch1" >> file1.txt
  git status
  git add file1.txt
  git status
  git commit -m "Changes from branch1 to file1.txt"
  git status
  git checkout master
  # merge the changes from branch1 into master
  git merge branch1
  git status
  ```

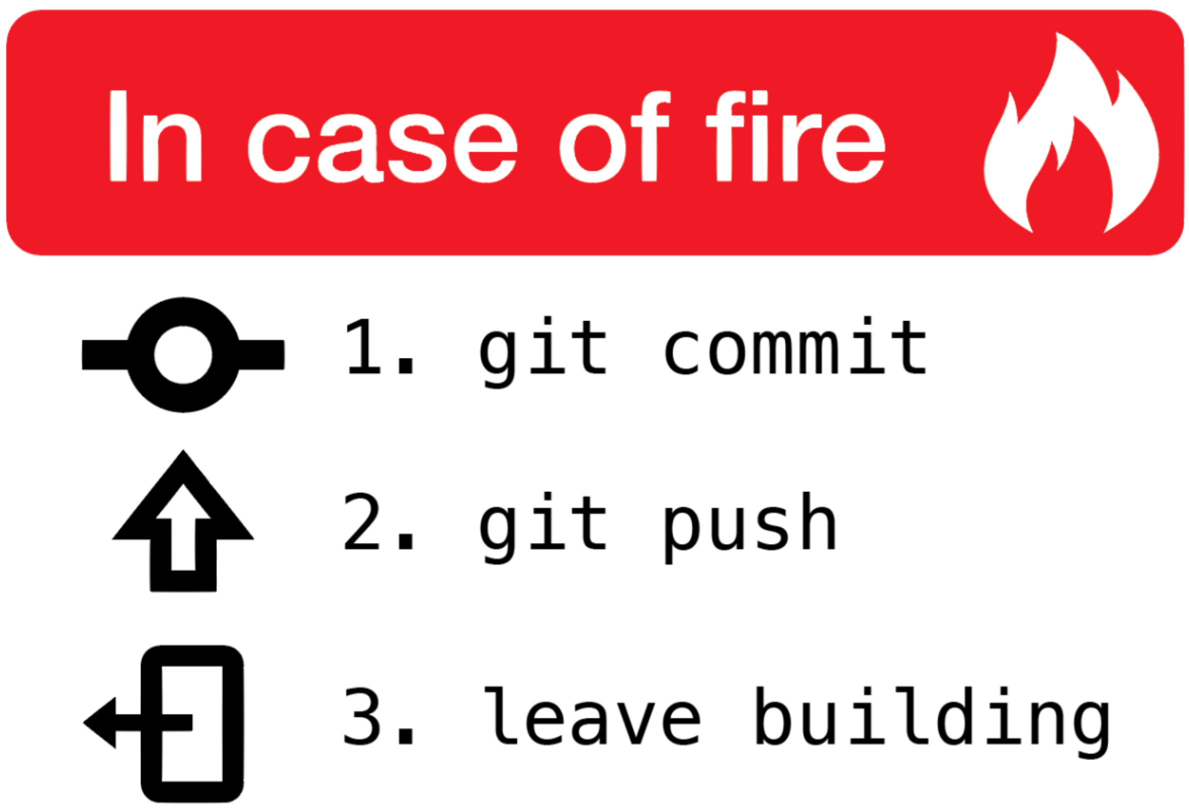
- `git log` shows all commits in this history of the current branch. This allows you to go back and see what changes you have made for each commit in this branch (assuming you have been creating descriptive commit messages, of course). This helps you know what changes occurred at which point in time, and will help you when trying to recover previous commits. Try it now:

  ```
  # look at past commits
  git log
  ```

Let’s commit and push these changes, just to clear everything out:

~~~
```
git add -u
git commit -m "learned about staging & snapshots"
git push origin master
```
~~~

Sharing & Updating



These commands relate to sending and receiving updates from remote repositories.

- `git remote add [alias] [url]` is covered in class.
- `git push [alias] [branch]` sends local repository branches to the remote repository. This allows the local branch to be fetched from another local repository linked to the same remote. Try it now:

```
# send the changes in this branch to the remote repository
git push origin branch1
```

- `git pull` fetches and merges any commits from the remote branch. This allows for pushed commits to easily be pulled back down from (usually) the master branch. Try it now (it won't do much at the moment):

```
# fetch and pull the changes from the remote repository
git pull
```

- `git fetch [alias]` fetches all the branches present in the remote designated by [alias]. This allows additional branches to be created in local repositories, pushed to a remote repository, and fetched from another local repository, allowing those branches to be merged in the local repository. Let's try it out:

```
cd ..
mkdir otherRepo
cd otherRepo/
git clone https://github.com/[YOUR GITHUB USERNAME]/myRepo.git
cd myRepo/
git pull
# fetch the branches from remote names 'origin'
git fetch origin
cd ../../myRepo/
git checkout branch1
echo "More changes from branch1" >> file1.txt
git add -u
git commit -m "More changes from branch1"
git push origin branch1
cd ../otherRepo/myRepo/
git pull
git fetch origin
```

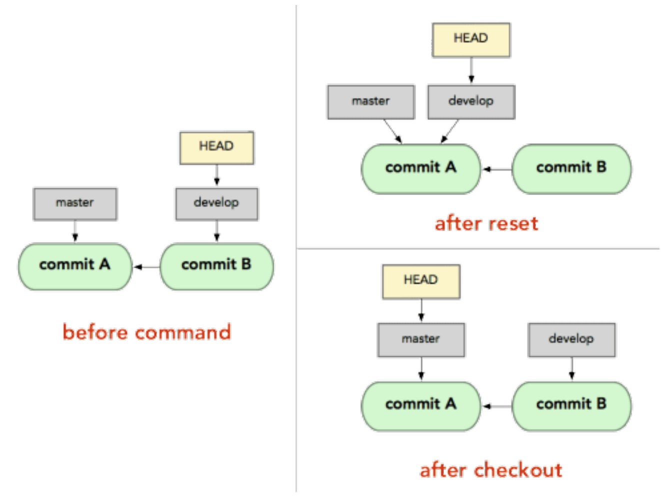
- `git merge[alias]/branch` merges the changes from a fetched branch into the current branch. This is how we incorporate the changes we fetched down from the remote repository. Try it now:

```
# check the contents of the file
cat file1.txt
git merge origin/branch1
# check the changed contents of the file
cat file1.txt
git push
```

Let's commit and push these changes, just to clear everything out:

~~~
```
git add -u
git commit -m "learned about sharing & updating"
git push
cd ../../myRepo/
git pull origin branch1
git checkout master
git pull origin master
```
~~~

Commit History



These commands relate to changing the history of the commits in the working tree.

- `git rebase [branch]` applies commits of the current branch ahead of the specified branch. This allows you to have a cleaner project history and creates a more linear working tree. Try it now:

  ```
  git checkout branch1
  # sets master as the base of branch1
  git rebase master
  ```

- `git reset --hard [commit]` rewrites the working tree from the specified commit. This resets the working directory and the current commit to the specified one, allowing you to undo commits and work from that version of the project. Try it now:

  ```
  cat file1.txt
  # undoes all commits after the specified commit
  git reset --hard [commit ID from inserting 'Hello, world.']
  cat file1.txt
  ```

End of material not covered in class

These four sections cover the Git commands I commonly use in my projects. Git has many other commands which I did not cover here, but they can be found in the Git documentation. Hopefully the commands covered in this document provide you with a deeper understanding of how Git works and why we use it for our projects.

References

```
- Professor Sanchez; 'Git Basic Concepts'
- Professor Sanchez; 'Lab 3: Github repo for Stat 133'
- Professor Sanchez; 'Git Cheat Sheet'
- Professor Hilfinger; CS61A & CS61B Lectures/Assignments
- Wikipedia - Git; 'https://en.wikipedia.org/wiki/Git'
- git-scm - Documentation; 'https://git-scm.com/doc'
- atlassian - Tutorials; 'https://www.atlassian.com/git/tutorials'
- Juri Strumpflohner - Git Explained; 'https://juristr.com/blog/2013/04/git-explained/'
```