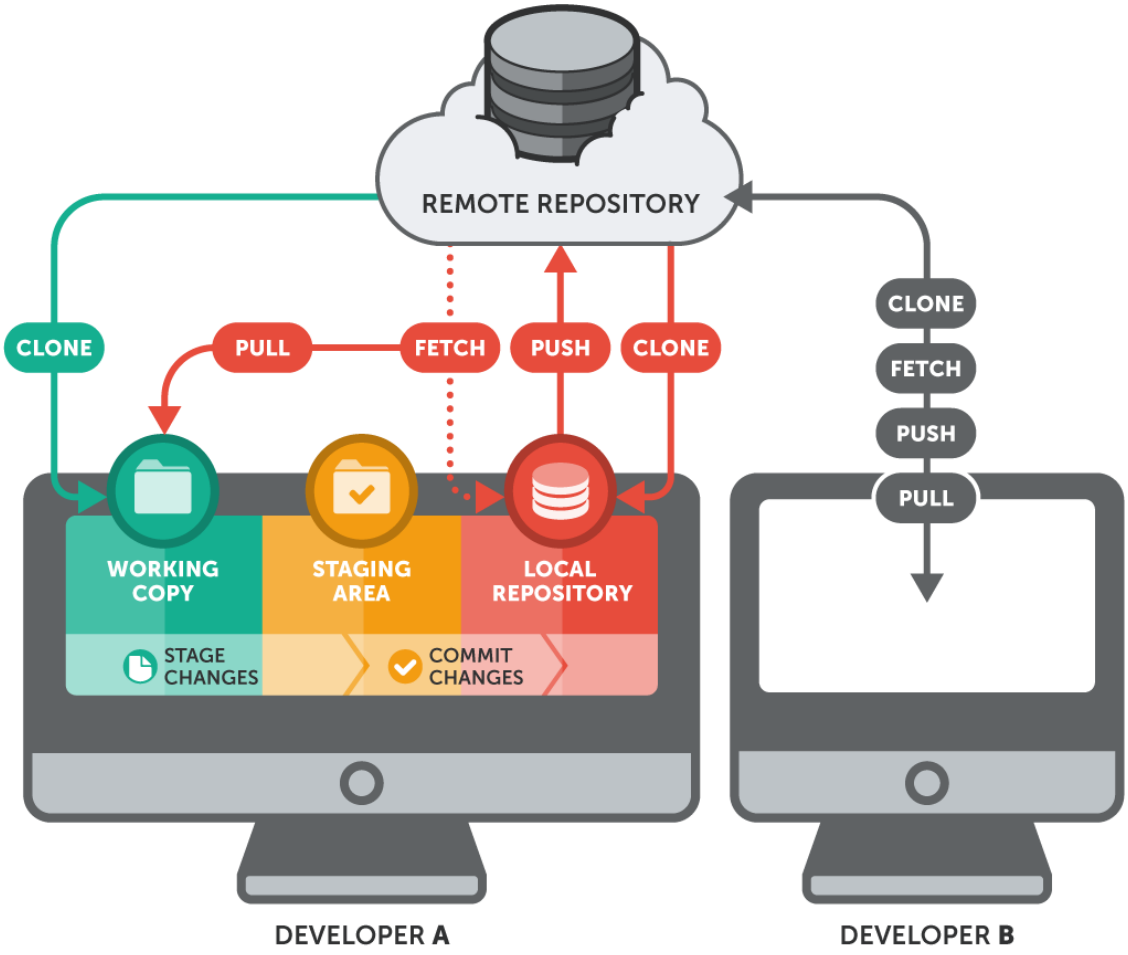
03 Git

April 28, 2022

1 Git Basics: Adding Git Remotes, Pushing and Pulling

What good is keeping tracks of all these versions if we just lose the entire repo by accident, or if we can't put it in a place where others can access it? In this section, we'll practice telling git where it can send updates to backup repositories *(git remotes)*, sending *(pushing)* new commits to them, and even receiving *(pulling)* new commits on those remotes! We'll see that GitHub is a useful place to put a git remote, but that we can put a git remote pretty much anywhere, even on other folders in our local computer.



1.1 Terminal Commands

- `git remote`: List all the names of the remotes for this git repository
  - `git remote -v`: List all the names and locations of the remotes ("verbose mode")
- `git remote add <remotename> <url>`: Add a new remote with a name and a web address
- `git remote add <remotename> <path>`: Add a new remote with a name and a folder location
- `git pull remotename branchname`: Receive the new changes from a given branch on a remote.
- `git push remotename branchname`: Send new changes to a given branch on a remote
- `git clone --bare . <path>`: Clone the current repo to a certain path (make sure the path isn't inside the current repository)

1.2 Exercise

Sequence

Current State: You should start this exercise with a git repo with a couple of files and some commits in the history.

1: Check your git remotes. How many are there?

Current State: There should be no remotes listed.

2: Create an **empty** (really, no files at all, or this won't work) GitHub repo, note its URL.

3: Add a new git remote, called "origin", at the URL of the new GitHub repo. Check the remotes–was it successfully created?

4: Push the current branch to your "origin" remote (you might have to sign in with username and password). Refresh the GitHub page–your files should be there!

Current State: Your repo should have one remote, called "origin", that references the GitHub repo, and the GitHub repo should have all of your files!

5: Let's make a change on our local git repostory. Add a new line to the run.py file, and have it print your name after it prints Hello World. Then commit your changes, and check your GitHub repo by refreshing the page. Has it automatically updated with your new changes?

6: Push the current branch to your "origin" remote (you might have to sign in with username and password). Refresh the GitHub page–your files should be there!

Current State: Your GitHub repo's run.py file should have the new line in it

7: This time, let's make changes on our GitHub repo and **pull** them to our local repository. In the web browser, edit the readme file adding the line "Made by *your first name*", and commit the change. Verify that the change isn't there in your local repository, then pull it to your local machine.

Current State: Your local repository should have the new line in your readme in it

1.2.1 (Extra) Making a Headless Clone Remote

How about we create a second backup for our repo, this time in a different place on our local machine? This is useful, for example, if you want to be able to push backups on Dropbox, Box, or

external hard drives.

The command is `git clone --bare . <path-to-backup-repo>`

For example: `git clone --bare . ../my-backup`

"bare" means that the repo will have no HEAD; this makes it possible to push to it, the same as GitHub.

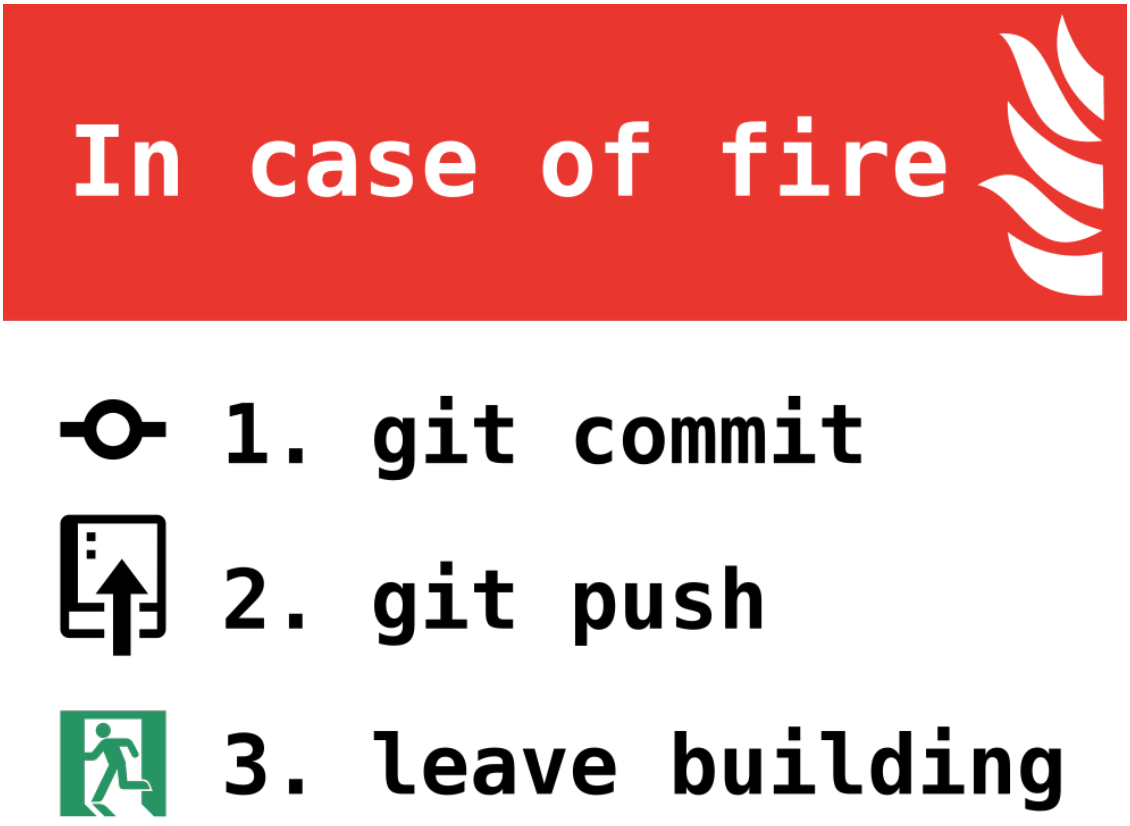
Exercise

1: Make a backup by cloning the project to the folder "my-backup" in the parent directory (`../my-backup`), and open up the folder in your file explorer. Notice that this folder doesn't look the same as the original; it's only the contents of the .git folder.

2: Add this folder as a remote. Let's call it "local"

3: Make a change, commit, then push it to both "local" and to "origin". Now you have two backup locations!

Done!



[ ]: