Introduction to GitHub

Koen Leuveld

EDI

k.leuveld@surveybe.com

March 13, 2019

Overview

Introduction

What is git?

- Git is a Version Control System;
- It was developed by the creator of Linux to keep track of the work of all the people helping out on developing Linux;
- It allows you to download the project you're working on from a central server, make all the edits you want offline, and upload those back to the server;
- It works as a set of arcane commands you type from the command line. But there are programs that make it easier, like GitHub Desktop.

Why Git?

- It handles conflicts really nicely (in plain text files, like CSV, .do and .ado);
- It allows you to revert any change (any commit, in git-speak);
- See who's responsible for each edit to a file;
- It allows you to download any old version of any file (through the website or the command line);
- It allows you to have multiple versions of the same files, so everyone can work on their own version: these versions (branches) can then be merged together.

What we'll do today

- The main problem with learning git is the vast amount of terms and concepts there are;
- Today we'll only look at the concepts you'll need to use it, none of the more advanced stuff;
- You won't be a git wizard, but you'll know enough key terms to be able to google for help, and understand the help being offered.

Check that you've done the following

- Download and install GitHub Desktop: `https://desktop.github.com/`
- Optional, if you'd like to give the command line a go, download Git: `https://git-scm.com/downloads`
- Make an account on GitHub.com, and share it with Koen, so he can invite you to the training repository.

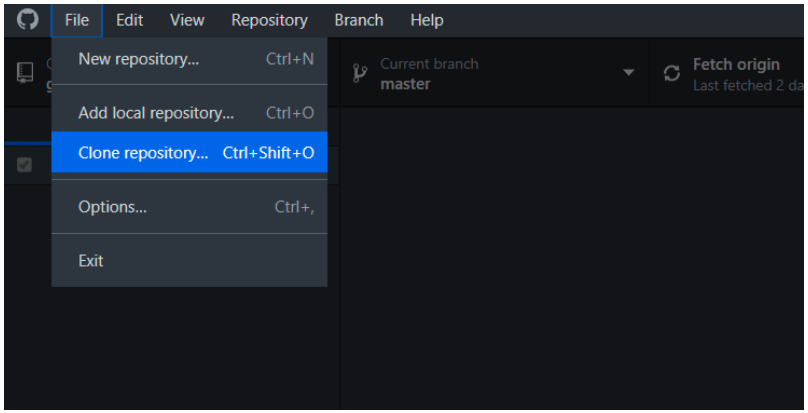
Working with GitHub Desktop

Overview

- With git, your files are held in a **repository** (repo). There's two types of copies of this repository:
  - The **remote** repository, or **origin**: this is a server somewhere. In our case today, it's GitHub. EDI adofiles and Surveybe code are hosted on Codebase.
  - Your (or your collaborator's) **local** repository: this is on your computer, and you make edits here.
- GitHub Desktop will handle the syncing between the remote and local repos, but you will need to manually tell it when. GitHub Desktop also works for git repositories not hosted on GitHub itself.
- You can choose any folder to put your local repo in. It's best not to use a shared Dropbox folder, as updates made by other Dropbox users will confuse Git.

Downloading the repo

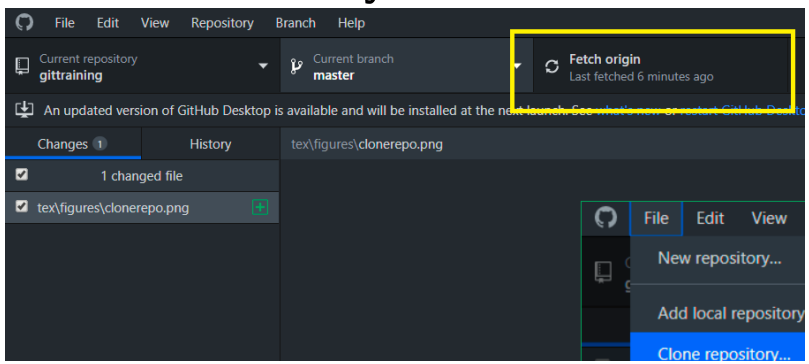
- Clone (i.e make a local copy of) the following repository: `https://github.com/kleuveld-edi/gittraining`



- You're ready to go!

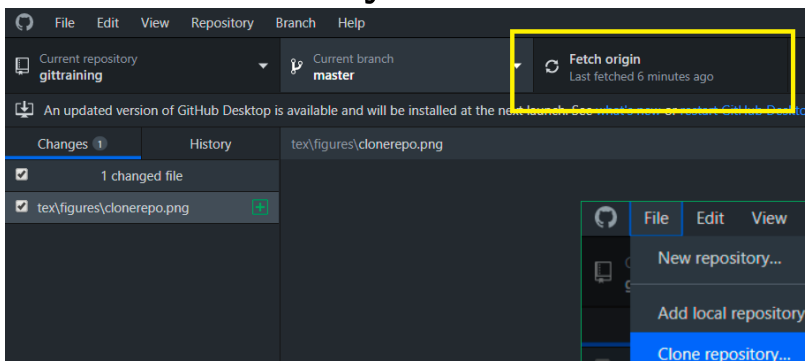
Git workflow; getting other people's edits

The workflow to get other people's changes is straightforward, and uses the sync button on the top:



Git workflow; getting other people's edits

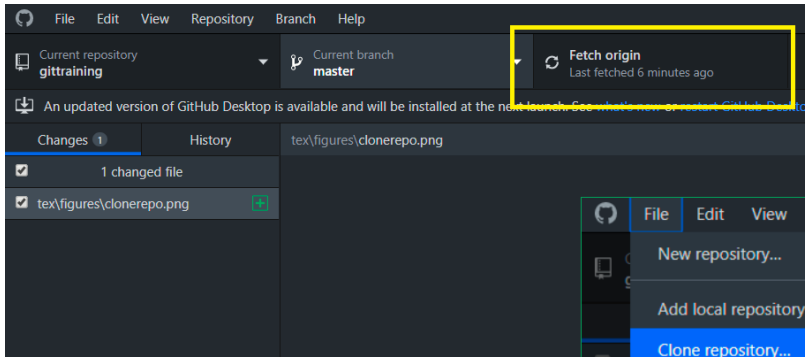
The workflow to get other people's changes is straightforward, and uses the sync button on the top:



1. **Fetch** changes: this tells git to check if there's any changes on the server. Click it, and the button will change to...
2. **Pull** changes: if there are changes, this tells git to download them

Git workflow; getting other people's edits

The workflow to get other people's changes is straightforward, and uses the sync button on the top:



1. **Fetch** changes: this tells git to check if there's any changes on the server. Click it, and the button will change to...
2. **Pull** changes: if there are changes, this tells git to download them

I will now create a file called example.do which we will all edit later on. Take the above steps to download it: you only need to press the sync button twice.

Git workflow: making edits

The general workflow to get your changes onto the git server is:

1. Change or add File
2. **Stage** changed file
3. **Commit** change
4. **Push** Commit(s)

Git workflow: making edits

Expanding on this:

1. Change or add File
2. **Stage** changed file
   - This tells Git that you've changed the file.
   - GitHub Desktop does this automatically!
3. **Commit** change
4. **Push** Commit(s)

Git workflow: making edits

So forget about staging!

1. Change or add File
2. **Stage** changed file
3. **Commit** change
   - This adds the changes you've staged to your **local** repository.
   - Git now saves a snapshot of the state of your project.
   - This means you can undo all the changes you've made since the last commit.
4. **Push** Commit(s)

Git workflow: making edits

So forget about staging!

1. Change or add File
2. **Stage** changed file
3. **Commit** change
4. **Push** Commit(s)
   - Commits will be uploaded to the remote repository (the github server for now).
   - Others can now **Pull** your changes
   - Your repo must be up-to-date if you're going to Push (else you may push outdated things!), so you may have to pull first (git will tell you)

Git workflow: making edits

All things together:

1. Change or add File
2. **Stage** changed file
3. **Commit** change
   - This adds the changes you've staged to your **local** repository.
   - Git now saves a snapshot of the state of your project.
   - This means you can undo all the changes you've made since the last commit.
4. **Push** Commit(s)
   - Commits will be uploaded to the remote repository (the github server for now).
   - Others can now **Pull** your changes
   - Your repo must be up-to-date if you're going to Push (else you may push outdated things!), so you may have to pull first (git will tell you)

Now you!

One by one:

1. Add a file to the folder where you've put your repository. For example: Koen.txt
2. Commit the change (press "Commit to Master" in bottom left). GitHub Desktop may prefill a summary for you. These are always in present tense, as it describes what is done when the commit is applied.
3. Push the change to the central repository: Repository - Push
4. If the repo has been changed before you try to push, you need to pull first: Repository - Pull

Notes:

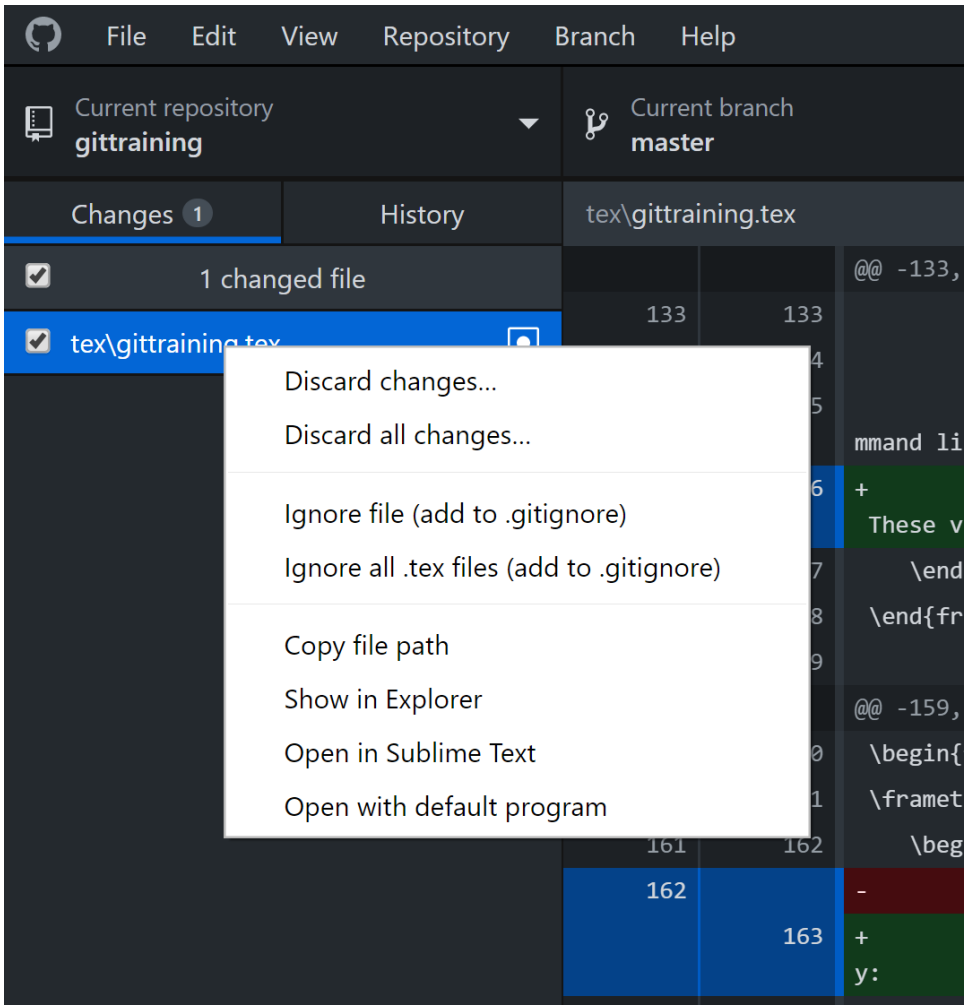
1. It all works fairly similar to Dropbox, but you have to do everything manually. This may or may not be a good thing.
2. To prevent having to pull other people's work everytime you want to push, you can make sure to work in your own branch (more on that later).
3. You can do all we've done from the command line, but it'd be more trouble than it's worth.

Undoing Things

There's a number of ways to undo things.

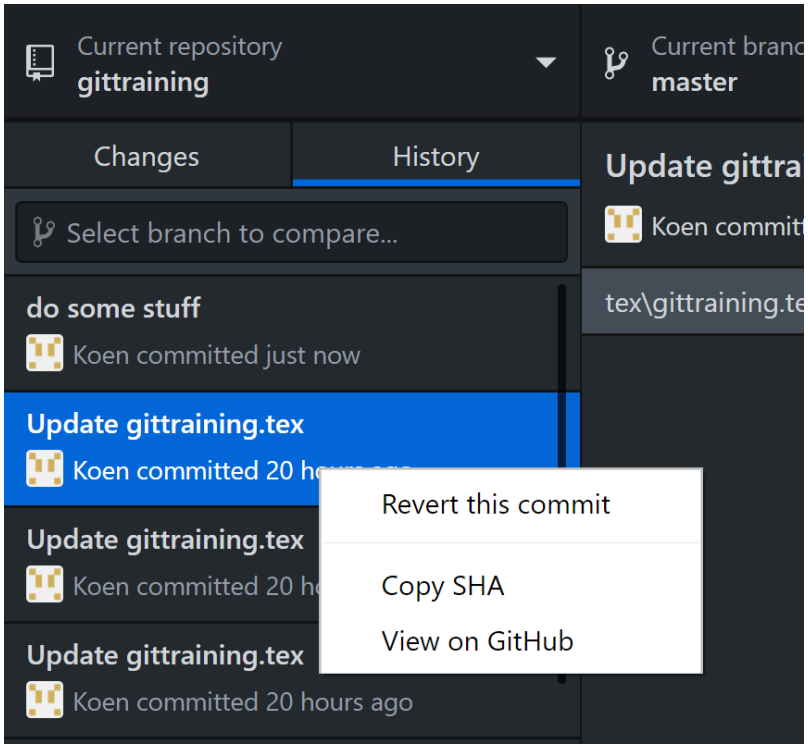
Discard Changes

- Press "Discard Changes" in the "Changes" tab of GitHub desktop to throw away all changes since the last commit:



Reverting a commit

- Revert a previous commit: In the "History" tab of GitHub Desktop you can see all the commits. You can right-click any commit to revert it.
- Reverting creates a new commit, which you can then revert as well!



More

- You can also just browse the history tab, find old text that you've deleted, and copy it back into the current version of your file.
- There is a very detailed history of each file on the GitHub / Codebase website. This allows you to see who's made which edit etc.
- There's many other ways of undoing things, here: `https://github.blog/2015-06-08-how-to-undo-almost-anything-with-git/`

Dealing with other people

Git shines in collaborating with other people and dealing with the eventual conflicts. A big part of this is also making sure everyone works on their own version of the files: their own Branch.

Editing the same file

Editing different lines in one file without conflicts:

Editing the same line to create conflicts:

Editing the same file

Editing different lines in one file without conflicts:

1. In the file example.do, everyone add something to their **own line**, and commit your changes;
2. Then we'll have to sync it all by pulling and then pushing;
3. At the end of all this, we'll have one file that everyone edited, simultaneously.

Editing the same line to create conflicts:

Editing the same file

Editing different lines in one file without conflicts:

1. In the file example.do, everyone add something to their **own line**, and commit your changes;
2. Then we'll have to sync it all by pulling and then pushing;
3. At the end of all this, we'll have one file that everyone edited, simultaneously.

Editing the same line to create conflicts:

1. Now, two people your name after "Add your name here:";
2. Both commit, and one pushes their commit;

Editing the same file

Editing different lines in one file without conflicts:

1. In the file example.do, everyone add something to their **own line**, and commit your changes;
2. Then we'll have to sync it all by pulling and then pushing;
3. At the end of all this, we'll have one file that everyone edited, simultaneously.

Editing the same line to create conflicts:

1. Now, two people your name after "Add your name here:";
2. Both commit, and one pushes their commit;
3. The other person must now pull the first's commit before pushing his;
4. The pulled commit will conflict with a local commit, and Git will allow you to address the conflict yourself by telling you where the conlfict is:

Resolving a conflict

```
4
5   <<<<<<< HEAD
6   We wanted to say this in this line
7   =======
8   But the others want to say this instead!
9   >>>>>>> 241e4140b3c404b0f53d80c94a505ca9e72eb1b5
10
11
12
13
```

Notes:

1. HEAD is us. To be precise, it's the most recent version of files that we have.
2. The long string of letters and numbers is the SHA (unique ID) of the commit that is conflicting with our work.
3. We can now just edit the file by picking and choosing what we want.

Branches

- **Branches** allow you to have your multiple versions of all the files in the repo, at the same time;
- You can edit things in your branch, while leaving the main branch (**MASTER**) untouched;
- Of course, this means that you are also not bothered by other people's changes!
- Once you're happy with the changes you've made and feel they're ready to be shared, you can **merge** your branch back into master;
- Merging works like pulling all changes in one go: meaning you'll have to resolve conflicts;
- MASTER is always the main branch, and it's good to discuss with your colleagues before merging your changes into MASTER. GitHub and Codebase offer tools for this, but you can discuss however you want.

Now you

Now just play around. Try:

- Creating a branch.
- Making changes to files in your own branch and commiting them;
- Publishing and pulling your branch to the server;
- Looking at other people's branches;
- Merging other branches into yours;
- Merging your branch back into the MASTER branch;
- Browsing all files on the GitHub website.

Final thoughts

Final thoughts

- Your collobarators (DPs) technically don't need to use git for it to be useful. Just make a branch where you keep their changes, and a branch where you keep yours and merge this from time to time, and share the results back to the collaborator.
- Explore the github website to find all versions of all files. Note how much information there is to process, not quite user-friendly!