

An intro to Git and GitHub

DIME Analytics

April 18, 2019

DECIE - The World Bank

Before the session starts:

1. Do you have a GitHub.com account? If not, got to <https://github.com/join> and sign up
2. Have you sent your GitHub username to the organizer or to kbjarkefur@worldbank.org ?
3. Have you installed GitHub Desktop? If not got to <https://desktop.github.com/> and download it.
4. Have you logged in at least once on GitHub Desktop? If not open GitHub Desktop and log in using your GitHub account.
5. Have you been invited to the github.com/kbjarkefur/lyrics repository? And have you accepted the invitation?

An intro to Git and GitHub

DIME Analytics

April 18, 2019

DECIE - The World Bank

Intro to the intro session

Today's session will not make you an expert in Git and GitHub, but it will teach you how to use the fundamental building blocks of Git. This may or may not be enough depending on which role you will have.

The Git Roles

Non-Coding PI

- A PI that is only involved in discussing results, but wants to have access to all code and see what people are working on
- This person will know what they need to know after this training!

Coding PI

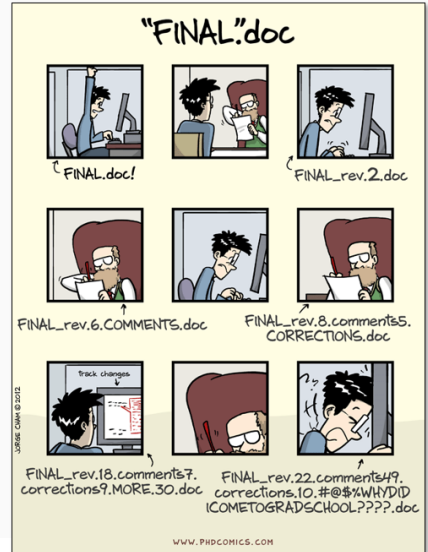
- A PI that is involved in writing code but does not have time to be the maintainer of the Git folder
- Some more learning-by-doing is needed, but for now no more training after this, as long as there is a repo maintainer

Repo Maintainer

- Typically an RA. This person spends a lot of time with the Git folder, and will ensure that the Git work flow is followed
- This person will need more training than this training, and/or support from DIME Analytics

What is Git used for?

- Git solves the *Final.doc* problem

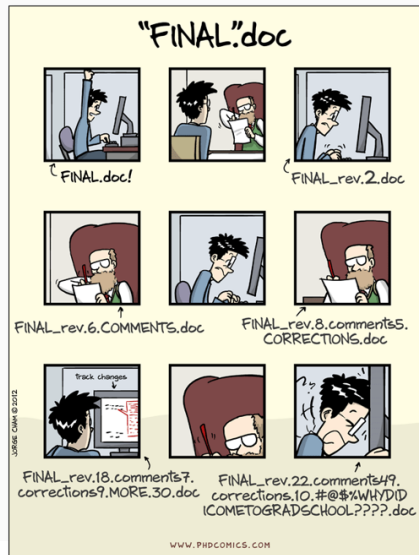


What is Git used for?

- Git solves the *Final.doc* problem
- Common solution to the *Final.doc* problem.

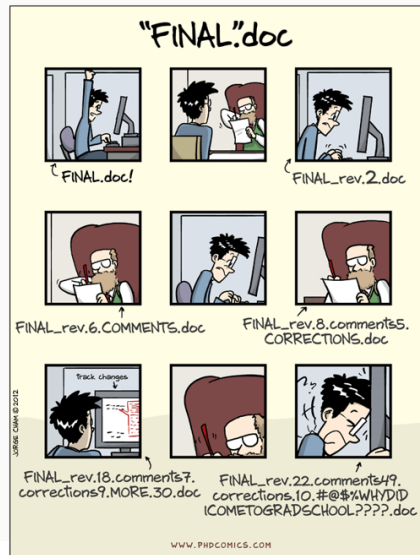
Name all your docs like

YYMMDD_docname_INITIALS.doc



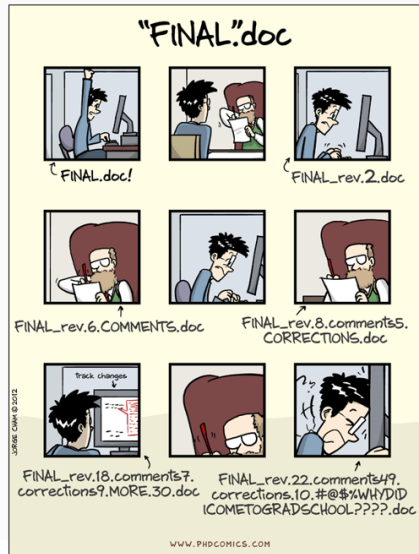
What is Git used for?

- Git solves the *Final.doc* problem
- Common solution to the *Final.doc* problem.
Name all your docs like
YYMMDD_docname_INITIALS.doc
- Git tracks *YYMMDD* and *INITIALS* for all edits without the user having to remember it

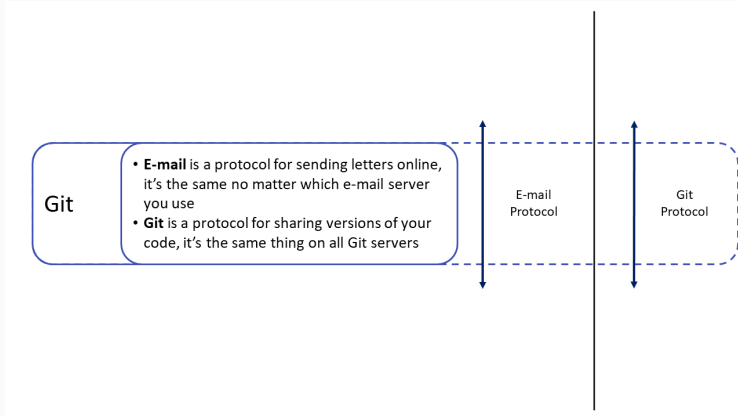


What is Git used for?

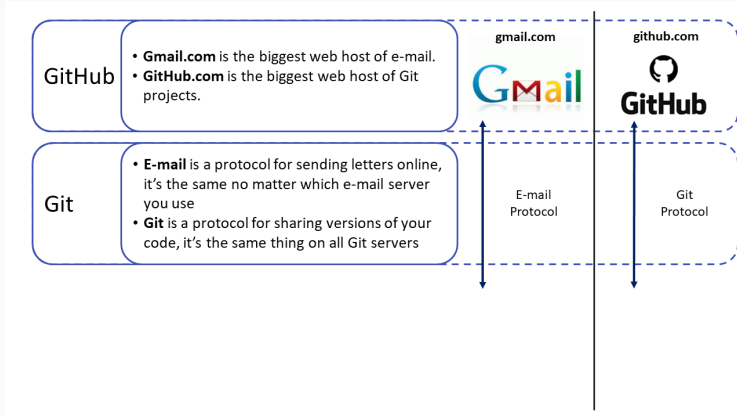
- Git solves the *Final.doc* problem
- Common solution to the *Final.doc* problem.
Name all your docs like
YYMMDD_docname_INITIALS.doc
- Git tracks *YYMMDD* and *INITIALS* for all edits without the user having to remember it
- That's far from everything, Git also solves:
 - Conflicting copy problem (DropBox etc.)
 - I can't re-produce my Baseline report problem
 - Who wrote this code 4 years ago and why?
 - And much much more...



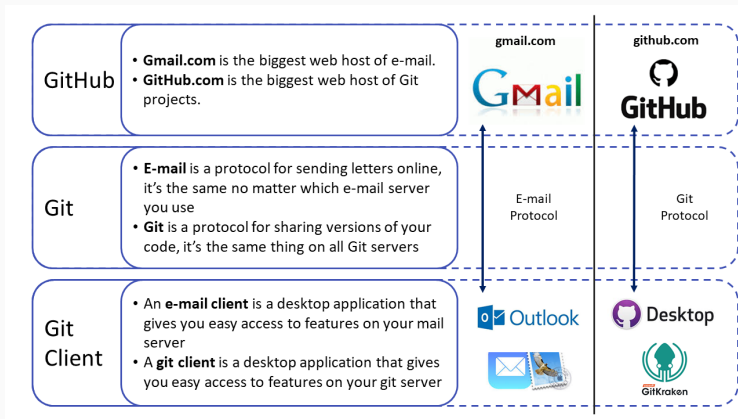
What is Git, GitHub and GitHub Desktop?



What is Git, GitHub and GitHub Desktop?



What is Git, GitHub and GitHub Desktop?



What will we learn?

In the **An intro to Git and GitHub** you will learn to:

- Explore the history of a project folder in GitHub and see what different team members are currently working on
- Download a project folder from GitHub so you can work on it
- Create a space in the project folder where you can make your edits
- Make edits and share those versions with your team. When you are ready, request that your edits are included in the main version

Three Git concepts needed to do this:

- Clone
- Commit
- Branch

Code free training!

No code today!

We will not work with code today in our repository.

Code tends to distract people if, for example, they see a command they do not understand.

Instead we will work with lyrics in .txt files that works exactly the same as code files in Git.

How to browse GitHub.Com

Your project folder is called a **repository** in Git. We will call the page you land on when you enter `github.com/kbjarkefur/lyrics` the **main page**.

GitHub.com -> Repo

1. From anywhere on `github.com` click the *octocat* icon in the top left corner.
2. In the menu to your left you see the repositories you are invited to
3. Click any repo to get to the main page of that repo.

Repo -> Main page

1. Click the repo name in [kbjarkefur/lyrics](#) at the top of any page within the repo
2. Click the tab that says **Code** below the repo name at any page in your repo

Clone

What is cloning?

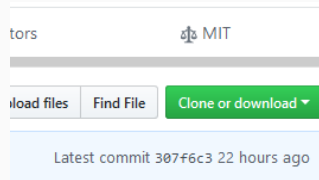
Cloning is similar to downloading a **repository** to your computer.

The difference between cloning and downloading is that **when Git clones a repository it remembers where you downloaded it from**. This is necessary so that Git knows where to share the edits you make to the files in the repository.

How do I clone a repository from GitHub?

How to clone a repository:

1. Go to the **main page** of `github.com/kbjarkefur/lyrics`
2. Click the green *Clone or download* button (see image)
3. Click *Open in Desktop*
4. Select where on your computer to clone the repository. Usually in your *Documents* folder. Do **NOT** clone in a shared folder, like a network drive or in DropBox.



Explore the clone!

Compare the files and folders you cloned to your computer with those in the repository on `github.com/kbjarkefur/lyrics`

Collaboration on a Repository

In order to collaborate on a repository we need to introduce two topics:

Commits

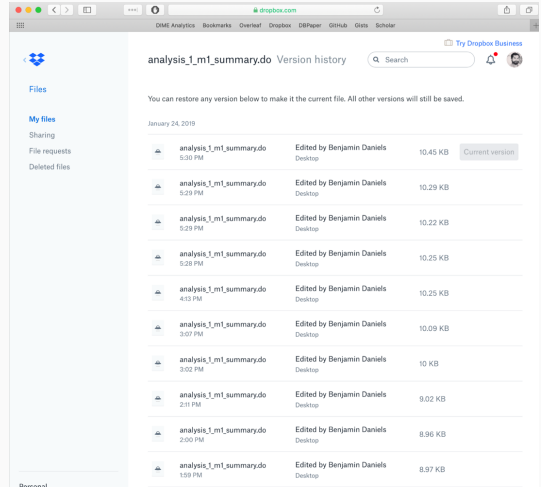
Branches

Commit

What is a version in version control?

First look at how version control works in another software you might be familiar with

In DropBox each saved version of a file is saved to the version history. This is the only way to do it automatically, but are all these versions meaningful differences?



The screenshot shows the Dropbox web interface. On the left is a sidebar with navigation options: Files, My files, Sharing, File requests, and Deleted files. The main area displays the 'Version history' for the file 'analysis_1_m1_summary.do'. A message states: 'You can restore any version below to make it the current file. All other versions will still be saved.' The version history table lists 10 versions, all edited by Benjamin Daniels on the Desktop. The top version is the 'Current version'.

Version	Edited by	Location	Size	Status
analysis_1_m1_summary.do 5:30 PM	Benjamin Daniels	Desktop	10.45 KB	Current version
analysis_1_m1_summary.do 5:29 PM	Benjamin Daniels	Desktop	10.29 KB	
analysis_1_m1_summary.do 5:29 PM	Benjamin Daniels	Desktop	10.22 KB	
analysis_1_m1_summary.do 5:28 PM	Benjamin Daniels	Desktop	10.25 KB	
analysis_1_m1_summary.do 4:13 PM	Benjamin Daniels	Desktop	10.25 KB	
analysis_1_m1_summary.do 3:07 PM	Benjamin Daniels	Desktop	10.09 KB	
analysis_1_m1_summary.do 3:02 PM	Benjamin Daniels	Desktop	10 KB	
analysis_1_m1_summary.do 2:11 PM	Benjamin Daniels	Desktop	9.02 KB	
analysis_1_m1_summary.do 2:00 PM	Benjamin Daniels	Desktop	8.96 KB	
analysis_1_m1_summary.do 1:59 PM	Benjamin Daniels	Desktop	8.97 KB	

What is a commit?

Instead of having a list of each saved version of a file, in Git we use **commits** to **indicate what is each meaningful difference between two versions of our project folder.**

Each commit is a snap shot of all files in the project folder, and lists how that snap shot differ from the previous snap shot (the previous commit).

Each commit has a time stamp and tracks who did the commit. This is very similar to the *YYMMDD_docname_INITIALS.doc* solution to the *Final.doc* problem.

How to make a commit

We need to introduce *branches* before we can all commit to the same repository, so for now, let me show you how to make a commit:

1. I add a new lyrics file in the clone
2. I use GitHub desktop to commit the new file to the repository
3. Can you see the new file on your computer?
4. Can you see it if you sync in GitHub Desktop?

Exploring commits

Now when we know what a commit is, we can start exploring how the `github.com/kbjarkefur/lyrics` repository was created.

We will see a list of commits, that at first sight is similar to the the version history in DropBox, but **in Git the version list is more meaningful, as it is a list of only meaningful differences.**

- `github.com/kbjarkefur/lyrics/commits`
- This list can also be found in GitHub Desktop

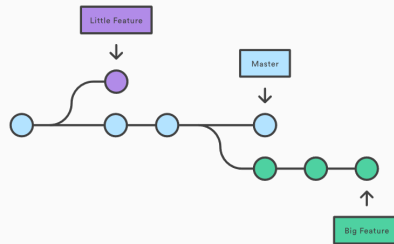
Branch

Introducing branches

Branches is the "killer feature" of Git. This is where Git becomes really powerful as a collaboration tool and as version control.

Branches allows you to **create a copy of the code where you can experiment**, if you like the result, **you can very easily merge your experiment with the main version of your code.**

This non-linear version control is much more similar to how we actually work than the strictly linear version control in, for example, DropBox



Looking at branches

One more way to explore the repository

- `github.com/kbjarkefur/lyrics/commits` <- Linear progression
- `github.com/kbjarkefur/lyrics/network` <- Non-linear progression

Exploring branches

- You can change branch in `/commits`. What happens when you change branch?
- Go to the main page, what happens if you change branch here?
- Which version is in the clone on your computer? They are all actually in your clone, but only one is shown - **checked out** - at the time
- What happens to the content of the folder on your computer when you check out another branch in GitHub Desktop?

Working with branches

A typical Git work flow involves multiple branches and there are tools in GitHub to makes that work flow easy, but that is not within today's scope. Although, what you should know after this training is only how to create your own branch and how to commit to it.

Create a branch:

- Go to `github.com/kbjarkefur/lyrics` and click the button where it says *Branch: master*.
- Write your name in the field and click *Create branch: your_name*. Make sure it says *from 'master'*.
- See how the button now says *Branch: your_name*
- Go to `github.com/kbjarkefur/lyrics/network` to check that your branch is there.

Combining Commit & Branch

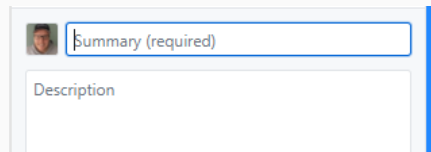
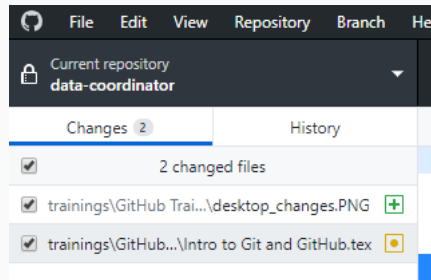
Now it is time to collaborate

Now it is time for you to collaborate:

1. Make sure your branch is checked out in GitHub Desktop.
2. Open a text editor. It could be *Notepad* if you are using Windows, *TextEdit* if you are using Mac, or any other code editor like *Atom*, or *Notepad++* etc.
3. Google the lyrics of your favorite song, and copy the lyrics to a new file in the text editor you just opened
4. Save the lyrics in your local clone according to these instructions:
 - Save the file in *.txt* format (Especially Mac users, this is not always the default)
 - Name the file after the title of the song
 - Save it in the appropriate genre folder (create a new genre folder if needed)

Do your first commit

1. Open the changes tab in GitHub Desktop
2. GitHub Desktop tracks your clone and has noticed that you changed something in it
3. Then you need to do the three steps required to commit a file to the repository:
 - 3.1 Make sure the file you want to add is checked
 - 3.2 Write a commit message and click
Commit to master
 - 3.3 Click the sync button



Check your commit on GitHub:

- Go to `github.com/kbjarkefur/lyrics/network`
 - Can you find your commit?
- Go to `github.com/kbjarkefur/lyrics/commits`
 - Can you find your commit?
 - If you cannot see your commit, make sure that you are looking in the correct branch

Pull Requests

A feature related to branches is a **pull request**. When the edits you have done in your branch are ready to be merged with the main version of the code, you make a pull request, i.e. you are requesting that your edits are pulled into the master branch.

A pull request can be made either by the person that edited the branch or the repo maintainer.

It is very common in GitHub repositories that only the repo maintainer have access to the master branch, and pull requests are then the only way to contribute to the master branch.

To make a pull request for your branch:

- Go to `github.com/kbjarkefur/lyrics/pulls` and click *New pull request*
- Make sure that the *master* branch is selected as the *base:* branch, and then select your branch as the *compare:* branch
- Scroll down to check the edits you are requesting to be pulled in to the master branch. If it looks ok, then click *Create pull request*
- Then you have the chance to add more instructions if you want, then click *Create pull request* again

Merge your contribution

Can you see your lyrics file in the master branch now? Why not?

Future trainings will teach you about merging pull requests. So I will do that for you now. Can you see your lyrics file in the master branch now?

What have we learned?

In the **An intro to Git and GitHub** you have learned to:

- Explore the history of a project folder in GitHub and see what different team members are currently working on
- Download a project folder from GitHub so you can work on it
- Create a space in the project folder where you can make your edits
- Make edits and share those versions with your team. When you are ready, request that your edits are included in the main version

What have we learned?

In the **An intro to Git and GitHub** you have learned to:

- Explore the history of **repository** and see what different team members are currently working on
- **Clone** a **repository** so you can work on it
- Create a **branch** in the **repository** where you can make your edits
- Make edits and share **commits** with your team. When you are ready, make a **pull request** to the **master branch**

Magic Trick

Magic Trick

All great training ends with a magic trick! This is a pedagogical magic trick, don't worry, just bear with me!

I need a volunteer for a simple task from the audience!

Next steps for the research team

What will the roles do:

The Git Roles

Non-Coding PI

- A PI that is only involved in discussing results, but wants to have access to all code and see what people are working on
- This person will know what they need to know after this training!

Coding PI

- A PI that is involved in writing code but does not have time to be the maintainer of the Git folder
- Some more learning-by-doing is needed, but for now no more training after this, as long as there is a repo maintainer

Repo Maintainer

- Typically an RA. This person spends a lot of time with the Git folder, and will ensure that the Git work flow is followed
- This person will need more training than this training, and/or support from DIME Analytics

What will the roles do:

Non-Coding PI

- Clone the repository
- Pull (sync) to get updates
- See commit history and network graph

Coding PI

- Everything in Non-Coding PI
- Commit to a single branch
- Eventually adopt a more advanced Git work flow

Repo Maintainer

- Everything in Coding PI
- Make branches are up to date, and that commits are included in master branch
- Other things not included in this training

Next steps for the research team

Before adopting Git the research team should discuss the following things:

- Who will be repo maintainer
- Agree on a work flow
- Public/private repository
- External collaborators
- Where to put data and where to put code
- Request a World Bank repo to be set up