BKPK MoH Indonesia

THE WORLD BANK
IBRD • IDA | WORLD BANK GROUP
Development Economics • Impact

DIME
ANALYTICS

Intro to Git and GitHub

María Reyes Retana

July 15, 2024

Development Impact Evaluation (DIME)
The World Bank

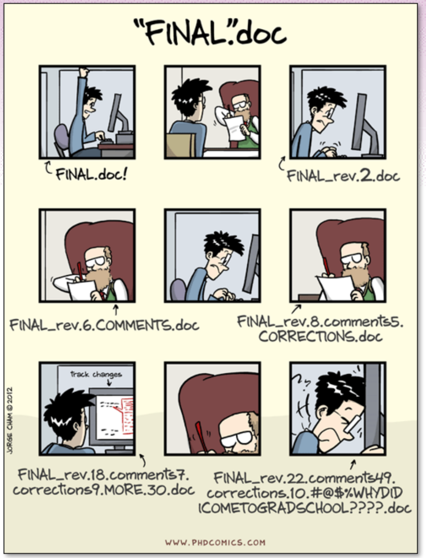
Before the session starts:

1. Do you have a GitHub.com account? If not, go to `https://github.com/join` and sign up
2. Have you shared your GitHub username with the session organizer?
3. Have you installed GitHub Desktop? If not go 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 `https://github.com/dime-wb-trainings/lyrics-jul15`?
6. And have you accepted at `https://github.com/dime-wb-trainings/lyrics-jul15/invitations`?

Introduction

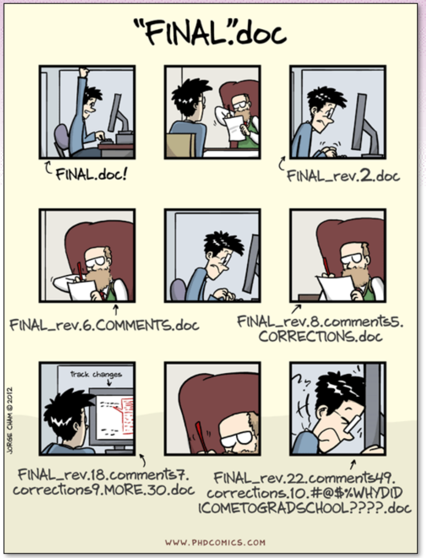
What is Git used for?

- Git solves the *Final.doc* problem


"FINAL".doc
FINAL.doc!
FINAL_rev.2.doc
FINAL_rev.6.COMMENTS.doc
FINAL_rev.8.comments5.CORRECTIONS.doc
track changes
FINAL_rev.18.comments7.corrections9.MORE.30.doc
FINAL_rev.22.comments49.corrections.10.#@$%WHYDID ICOMETOGRADSCHOOL????.doc
JORGE CHAM © 2012
WWW.PHDCOMICS.COM

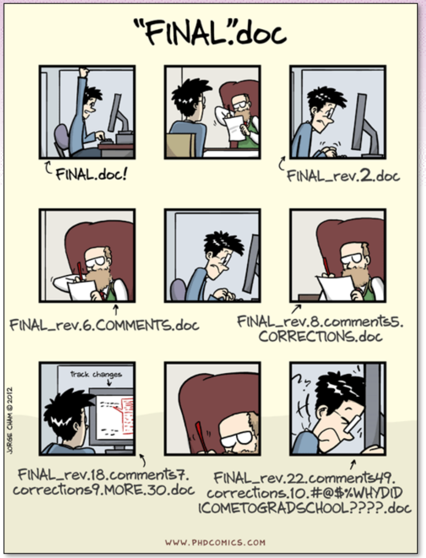
What is Git used for?

- Git solves the *Final.doc* problem
- Common solution to the *Final.doc* problem: Name all your docs like *YYMMDD_docname_INITIALS.doc*


"FINAL".doc
FINAL.doc!
FINAL_rev.2.doc
FINAL_rev.6.COMMENTS.doc
FINAL_rev.8.comments5.CORRECTIONS.doc
track changes
FINAL_rev.18.comments7.corrections9.MORE.30.doc
FINAL_rev.22.comments49.corrections.10.#@$%WHYDID ICOMETOGRADSCHOOL????.doc
JORGE CHAM © 2012
WWW.PHDCOMICS.COM

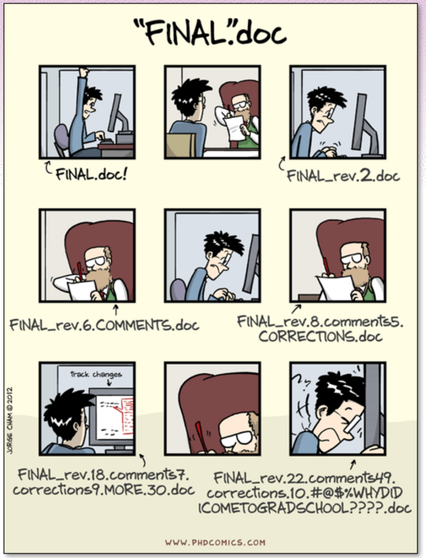
What is Git used for?

- 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

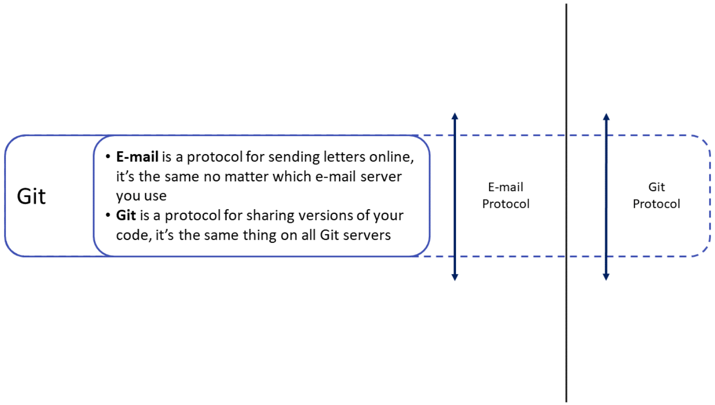

"FINAL".doc
FINAL.doc!
FINAL_rev.2.doc
FINAL_rev.6.COMMENTS.doc
FINAL_rev.8.comments5.CORRECTIONS.doc
track changes
FINAL_rev.18.comments7.corrections9.MORE.30.doc
FINAL_rev.22.comments49.corrections.10.#@$%WHYDID ICOMETOGRADSCHOOL????.doc
JORGE CHAM © 2012
WWW.PHDCOMICS.COM

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...

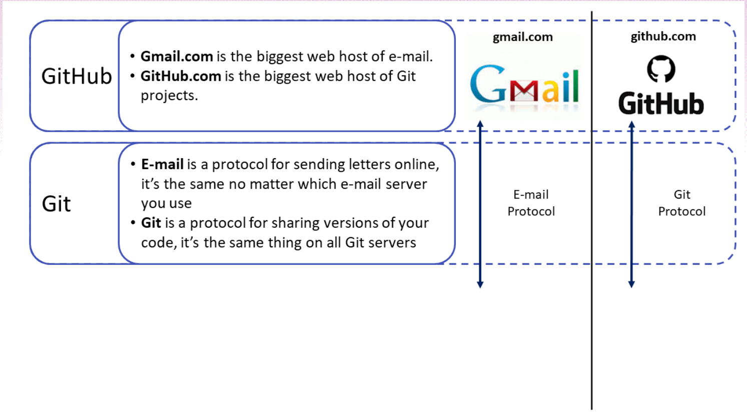

"FINAL".doc
FINAL.doc!
FINAL_rev.2.doc
FINAL_rev.6.COMMENTS.doc
FINAL_rev.8.comments5.CORRECTIONS.doc
track changes
FINAL_rev.18.comments7.corrections9.MORE.30.doc
FINAL_rev.22.comments49.corrections.10.#@$%WHYDID ICOMETOGRADSCHOOL????.doc
JORGE CHAM © 2012
WWW.PHDCOMICS.COM

What is Git, GitHub and GitHub Desktop?



- **E-mail** is a protocol for sending letters online, it's the same no matter which e-mail server you use
- **Git** is a protocol for sharing versions of your code, it's the same thing on all Git servers

What is Git, GitHub and GitHub Desktop?

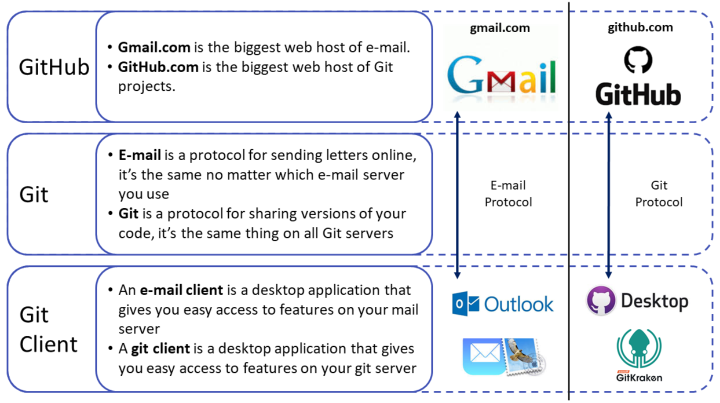

GitHub

- **Gmail.com** is the biggest web host of e-mail.
- **GitHub.com** is the biggest web host of Git projects.

Git

- **E-mail** is a protocol for sending letters online, it's the same no matter which e-mail server you use
- **Git** is a protocol for sharing versions of your code, it's the same thing on all Git servers

What is Git, GitHub and GitHub Desktop?



GitHub	• **Gmail.com** is the biggest web host of e-mail. • **GitHub.com** is the biggest web host of Git projects.
Git	• **E-mail** is a protocol for sending letters online, it's the same no matter which e-mail server you use • **Git** is a protocol for sharing versions of your code, it's the same thing on all Git servers
Git Client	• An **e-mail client** is a desktop application that gives you easy access to features on your mail server • A **git client** is a desktop application that gives you easy access to features on your git server

What will we learn?

In **An intro to Git and GitHub - Contributor Role** 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

MVO

Three Git concepts needed to do this:

- Clone
- Commit
- Branch

Code free training!

No code today!

We will not work with code today.

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, often **repo** for short. When you enter `https://github.com/dime-wb-trainings/lyrics-jul15` you get to what we will call the **repository landing page**.

Go from **GitHub.com** to **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 landing page of that repo.

Go from **repo** to **landing page**

1. Click the repo name in `dime-wb-trainings/lyrics-jul15` at the top of any page within the repo

Clone

What is cloning?

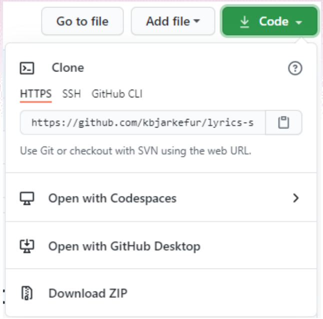
Cloning is similar to downloading a **repository** to your computer.

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 send any edits you make to the files when sharing them with your team.

How do I clone a repository from GitHub?

How to clone a repository:

1. Go to the **landing page** of `https://github.com/dime-wb-trainings/lyrics-jul15`
2. Click the green *Code* button (see image)
3. Click *Open with GitHub Desktop*
4. Select where on your computer to clone the repository. Do **NOT** clone in a shared folder, like a network drive or in DropBox. Create a *GitHub* folder in non-synced location and clone there. Read more about this in our guide here.



https://github.com/kbjarkefur/lyrics-s

Use Git or checkout with SVN using the web URL.

Explore the clone

Explore the clone!

Compare the files and folders you cloned to your computer with those in the repository on `https://github.com/dime-wb-trainings/lyrics-jul15`

Collaboration on a repository

Collaboration needs two more concepts

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

Commits

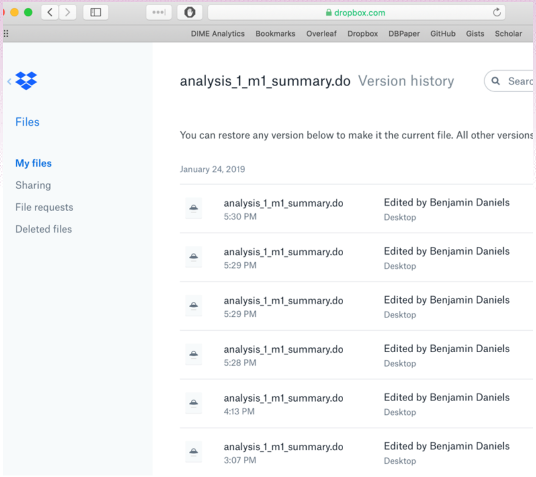
Branches

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?



analysis_1_m1_summary.do Version history

You can restore any version below to make it the current file. All other versions

January 24, 2019

analysis_1_m1_summary.do 5:30 PM	Edited by Benjamin Daniels Desktop
analysis_1_m1_summary.do 5:29 PM	Edited by Benjamin Daniels Desktop
analysis_1_m1_summary.do 5:29 PM	Edited by Benjamin Daniels Desktop
analysis_1_m1_summary.do 5:28 PM	Edited by Benjamin Daniels Desktop
analysis_1_m1_summary.do 4:13 PM	Edited by Benjamin Daniels Desktop
analysis_1_m1_summary.do 3:07 PM	Edited by Benjamin Daniels Desktop

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 snapshot of all files in the project folder and lists how that snapshot differs from the previous snapshot (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 .txt 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 that we know what a commit is, we can explore how the `https://github.com/dime-wb-trainings/lyrics-jul15` 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.**

- `https://github.com/dime-wb-trainings/lyrics-jul15/commits`
- This list can also be found in GitHub Desktop in the *History* tab

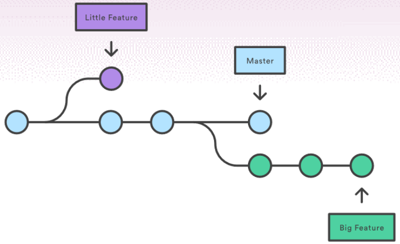
Branches

Introducing branches

Branches is the ”killer feature” of Git. This is where Git becomes really powerful as a collaboration tool and not just 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:

- Linear progression -> https://github.com/dime-wb-trainings/lyrics-jul15/commits
- Non-linear progression -> https://github.com/dime-wb-trainings/lyrics-jul15/network

Exploring branches

- You can change branch in */commits*. What happens then?
- Go to the landing page, what happens if you change the 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 files 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 `https://github.com/dime-wb-trainings/lyrics-jul15` and click the button where it says *Branch: main*.
- Write your name in the field and click *Create branch: your_name*. Make sure it says *from 'main'*.
- See how the button now says *Branch: your_name*

Combining Commit & Branch

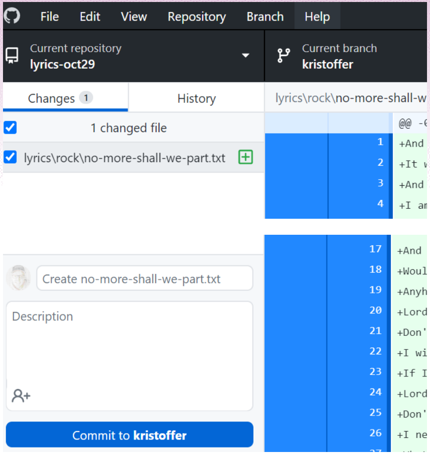
Now it is time to collaborate

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
   - 3.3 Click *Commit to your_name* - check out your branch if it says *main*
   - 3.4 Click the sync button



lyrics\rock\no-more-shall-w

1 changed file

lyrics\rock\no-more-shall-we-part.txt

Create no-more-shall-we-part.txt

```
@@ -
+And
+It w
+And
+I ar
+And
+Woul
+Anyh
+Lord
+Don'
+I wi
+If I
+Lord
+Don'
+I ne
```

Check your contribution

Check your commit on GitHub:

- Can you find your commit at:
  https://github.com/dime-wb-trainings/lyrics-jul15/network
- Can you find your commit at:
  https://github.com/dime-wb-trainings/lyrics-jul15/commits

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 main branch.

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

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

Merge your contribution

To make a pull request for your branch:

- Go to `https://github.com/dime-wb-trainings/lyrics-jul15/pulls` and click *New pull request*
- Make sure that the *main* 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 main 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 main branch now? Why not?

Your contribution will not be included until the branch is merged. We have more trainings on the details of merging branches but for now, this is all you need to know:

- Always have someone to review your PR before merging it
- Always delete your branch after it is merged - you can always recreate a new branch with the same name if you want

What have we learned?

In **An intro to Git and GitHub - Contributor Role** 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 **An intro to Git and GitHub - Contributor Role** 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 **main branch**

Setting up a Good Workflow

Importance of structuring data work

- Research projects are often long, have multiple team members, and involve many different data-sets
- During the typical project lifetime, many team members will make crucial contributions – and it is likely that not all of them were part of the initial planning
- Therefore, it is important to both organize all of our data work and set up an effective ”inter-generational” data work communication
- This is best done before any code is written – however, edits and additions will be required during the course of a project

Importance of structuring data work

- Taking time for these tasks at the beginning of project will save time over the project’s full life-cycle
- This helps senior team members to explain tasks they delegate, as well as helping junior team members to confirm and communicate their understanding of those tasks to their supervisors
- Having a good structure from the beginning makes it easier to collaborate and save valuable time on communicating and explaining the different moving parts of a project
- Will help to make your work clearer and less error-prone. If errors or changes are required, it will be easier to implement them.

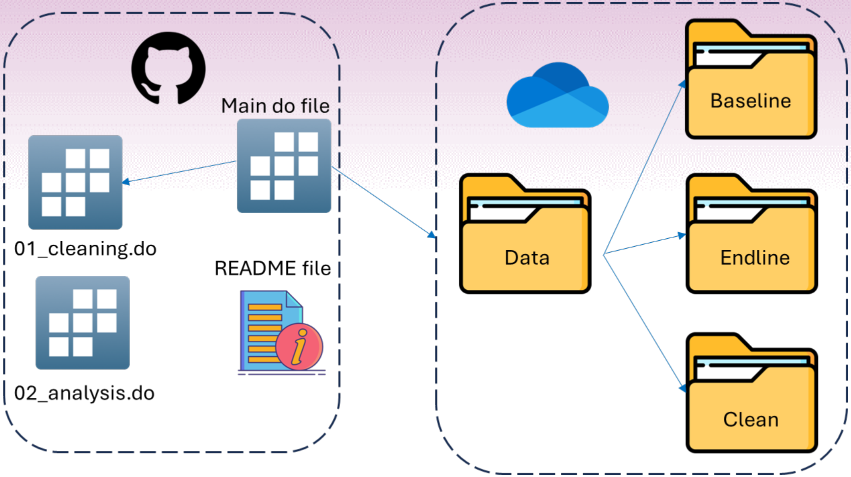
Key takeaways

In this part of the session, we will cover

- `GitHub Repository`: Start your project by including version control in GitHub to improve collaboration.
- `Folder structure`: Organizing your project folder for effective collaboration.
- `Main scripts`: One script that run all project code.
- `Documentation`: README file containing relevant project information, decisions, instructions, etc.

Organizing your data work using these principles will make your research more effective, efficient, and reproducible!

Structure of a good workflow



Example of a Good Workflow

GitHub: repository creation

- **Steps for Repository Creation**
  1. Go to GitHub and log in to your account.
  2. Click on the ”New” button to create a new repository.
  3. Enter a name for your repository.
  4. Set the repository visibility to private if necessary.
  5. Click ”Create repository”.
- Note: In this case, the repository has already been created, as it normally will in a regular project. This demonstration is for reference.

Hands-on: cloning the mock project

Once your project has a repository.

- **Clone the repository**
  1. Go to the GitHub repository: `https://github.com/dime-wb-trainings/GitHub-MockProject`.
  2. Click on the green ”Code” button and select ”Open with GitHub Desktop”.
  3. Follow the prompts to clone the repository to your local machine.

Hands-on: create a branch in the mock project

Once you have the repository on your computer:

- **Create a branch**
  1. As we will be working collaboratively, create a branch named ”workflow_” followed by your initials.
  2. Switch to that branch to start making changes to the project.
- When working on a real project, follow the principle: **branch often, merge often.** In practice:
  - Create a branch for each task.
  - After completing the task, merge the branch back to the main branch by creating a Pull Request (PR).
  - Repeat this process for subsequent tasks to ensure smooth and collaborative development.

Folder structure

- At the beginning of a research project, organize your data work into a folder structure that is easy to follow
- Think about all the sources and rounds of data that your project is anticipated to have
- Have one separate sub-folder for each source of data, and keep sub-folders for each data source as similar as possible
- Avoid creating individual folders in an ad-hoc manner as the need arises - be methodological

Hands-on: folder structure

- Download the mock data from the provided link.
- Save the data files in the `data/` folder on your local machine.
- Arrange the folder structure intuitively:
  - `code/` - Folder for all Stata scripts.
  - `data/` - Folder for data files (not tracked by Git).
  - `outputs/` - Folder for your outputs, if relevant.
  - `README.md` - Project documentation.
  - `.gitignore` - Specify files and folders to ignore in Git.

Main script

- The most important purpose of the **main script** is to have one script that runs all other scripts in the intended order
- It should never be required that that a human needs to know in which order scripts need to run in order for the project code to execute correctly
- Having a text-file or file-naming convention that communicate the run order is marginally better, but still error prone
- A main file that automatically run all files in correct order is the best practice and a requirement in DIME Analytics' reproducibility verification – we call this ”One-click Reproducibility”

Main script - additional benefits

- The main script becomes an ordered table of contents for all data work
- Specify settings and other requirements in one place – such as:
  - Check and install software packages needed in the project code
  - Set any setting needed to make random processes reproducible
  - Set project-wide variables such as conversion rates or list of control variables
- **Collaboration**
  - Use globals (Stata) dynamically set root paths.
  - Other can set-up the project on their computer by just setting the root paths – useful for collaboration and essential for one-click reproducibility.

Hands-on: main script setup

- **Create a `main.do` basic**
  - Open a new Stata do-file and save it as `main.do` in the `code/` folder. (in this exercise the main.do is already in your cloned folder, but is empty)
  - Set the global paths and the instructions to run the codes for your project in `main.do` (level one):

```
* Main do file for GitHub-MockProject

* Set global paths (user should change accordingly)

global data-path "path/to/your/data/folder"

global code-path "path/to/your/code/folder"

* Run Scripts

do "${code-path}/01-cleaning-script.do"
```

Hands-on: main script setup

- **Create a `main.do` advanced**
  - For a more advanced way to set up your global paths you can do the following.
  - If plenty of people are collaborating this could help not to change the file paths every time you download the main file.

```
 * Main do file for GitHub-MockProject

 * Set global paths (user should change accordingly)

 * Research Assistant folder paths
 if "‘c(username)’" == "ResearchAssistant" {
 global data-path "path/to/your/data/folder"
 global code-path "path/to/your/code/folder"
}

 * Run Scripts

 do "${code-path}/01-cleaning-script.do"
```

README file

- **Quick Project Overview**
  - Provides a summary of the project’s purpose and objectives.
  - Helps new contributors quickly understand the project’s scope and goals.
- **Setup Instructions**
  - Guides new team members on how to set up the project locally.
  - Lists necessary software, dependencies, and installation steps.
- **Documentation of Key Decisions**
  - Records important decisions made during the project.
  - Provides context for the project’s development and structure.
- **Instructions for Use**
  - Details how to run and use the project.
  - Includes examples and explanations of key functionalities.
- **Improves Collaboration**
  - Facilitates onboarding of new team members.
  - Ensures consistency and clarity in communication.

Hands-on: README File

- **Add a** `README.md`
  - Provide an overview of the project.
  - Include instructions on how to set up the project locally.
  - Note: The `README.md` is already there, but it is just a skeleton; participants should complete it.
  - Example of a good README can be found here: Template README.md

Other Elements: .gitignore

- **Usefulness of `.gitignore` file**
  - Prevents sensitive or unnecessary files from being tracked in the repository.
  - Keeps the repository clean and focused on relevant files.
  - Avoids conflicts by ignoring files that are specific to individual environments.
  - By default, our template should include data and ‘.xlsx‘ files in ‘.gitignore‘ for several reasons:
    - Binary files are not well tracked in GitHub.
    - Ensures privacy and data security by not including sensitive data files.
- **Explore `.gitignore` file**
  - Note: The `.gitignore` file is already there, you don’t need to create it, but in a project, you might.

Useful links: GitHub

- All DIME Analytics GitHub trainings: https://osf.io/e54gy/
- Other DIME Analytics GitHub resources: https://github.com/worldbank/dime-github-trainings. For example:
  - DIME Analytics GitHub Templates (for example .gitignore): https://github.com/worldbank/dime-github-trainings/tree/master/GitHub-resources/DIME-GitHub-Templates
- Markdown cheat sheet (how to format text on GitHub.com): https://www.markdownguide.org/cheat-sheet/

Useful links: Workflow

- Template Main do file: https://github.com/worldbank/dime-data-handbook/blob/main/code/box-2-4-stata-master-dofile.do
- Template README https://github.com/social-science-data-editors/template_README/blob/release-candidate/templates/README.md. For example: