Week 1: Git and GitHub

This document will introduce you to the basic concepts of the `Git` versioning system, storing your work on GitHub and interacting with those two from within the JupyterHub environment. `Git` is a very powerful tool - while its full potential can only be realized when using it directly from the command line, we will try here to limit our interaction with the terminal to the minimum and use available user interfaces whenever possible. It is very important, though, to understand the basic concepts of how `Git` works and how we can use it to our advantage. Let's dive in!

> **Why Git/GitHub?** Git is a powerful and free and open source distributed version control system. GitHub is an online hosting service for sharing and collaborating on code using Git. In other words, these tools allow you to collaborate on code and coding projects. This is not a tool reserved for software engineers. Scientists use Git/GitHub every day for organizing and collaborating on research projects. In this class, you will use Git/GitHub to collaborate on group projects, share code (analysis workflows), and maybe even to install some extra software.

1. Git setup

1.1 GitHub account

If you don't have a GitHub account yet, please navigate to GitHub.com and create one (click on the "Sign up" button and follow the instructions).

1.2 Git user details

Before you can start using `Git`, you need to tell it who you are. Before committing your first changes you should tell `Git` your name and email address - those will be attached to every of your commits and will identify you as their author. To configure those, open the terminal within JupyterHub (from the *Launcher* or the *File* menu) and run the following commands:

- user name:

  ```
  git config --global user.name "your name"
  ```

- email (enter the one you used to create your GitHub account):

  ```
  git config --global user.email "your email"
  ```

1.3 GitHub authentication on JupyterHub

Additionally, we need to authenticate with GitHub.com in order to push changes to remote repositories. You only need to do this once - it's a very important step, though, so **do not skip this section**!

1. Create a *personal access token* - follow the instruction in this tutorial. Set the expiration date to the end of this semester. In the "Select scope" section select the entire "*repo*" subsection and "*read:org*" in the "*admin:org*" subsection - these are the only permissions we will need.
2. Once the token is generated, copy it - after you close the page you won't be able to recover that token anymore and will need to re-create it, should you lose it.
3. In your JupyterHub workspace, go to the *Terminal* (from *Launcher* or the *File* menu) and execute the following command to begin authentication process:

```
gh auth login
```

Follow the authentication steps as shown below:

- What account do you want to log into? → **GitHub.com**
- What is your preferred protocol for Git operations? → **HTTPS**
- How would you like to authenticate GitHub CLI? → **Paste an authentication token**

Paste the token you copied before on the next line and confirm by hitting Enter/Return. You should see a message similar to this one:

```
✓ Configured git protocol
✓ Logged in as someone
```

4. That's it! You should be set up for using `Git` and interacting with GitHub.com. When your token expires (you will be notified by email), you will need to repeat the steps above.

2. Git cookbook

Before we talk about the actual `Git` workflow, let's look at some basic terminology when referring to branches and repositories located in different places:

Alias	Meaning
remote	any repository located in the cloud (here: GitHub)
origin	repository located in **your** GitHub account
upstream	repository from which a fork was generated (we won't be doing that here)
main	name of the first and "default" branch in any repository (also known as "master" in older repos)

Now, let's see how you can use `Git` to track your work.

2.1 Git repository

There are a couple of ways to create Git repositories:

1) from scratch, directly in the GitHub interface
2) from scratch, on your machine
3) from an existing template, using the GitHub interface

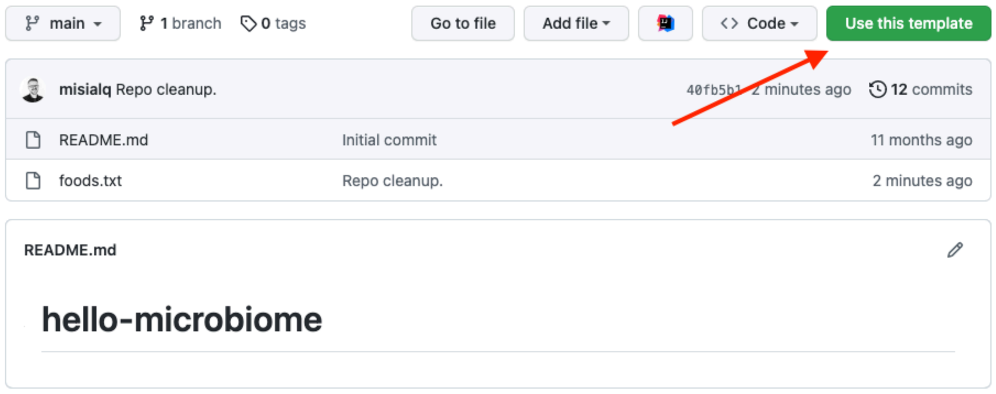
We will use option 3) and create a repo form an already existing template. Please follow the steps below to do that:

1. Navigate to the repo template: https://github.com/fsb-edu/hello-microbiome.

2. Click on the green "Use this template" button to make a copy of that repo in your GitHub account (see figure 1).

3. Fill out the name of the repository that will be created in your account (see figure 2). You can also name it "*hello-microbiome*", just like the original template. Click on "Create repository from template".

4. You should now see the README page of the newly created repo in your GitHub account (figure 3).



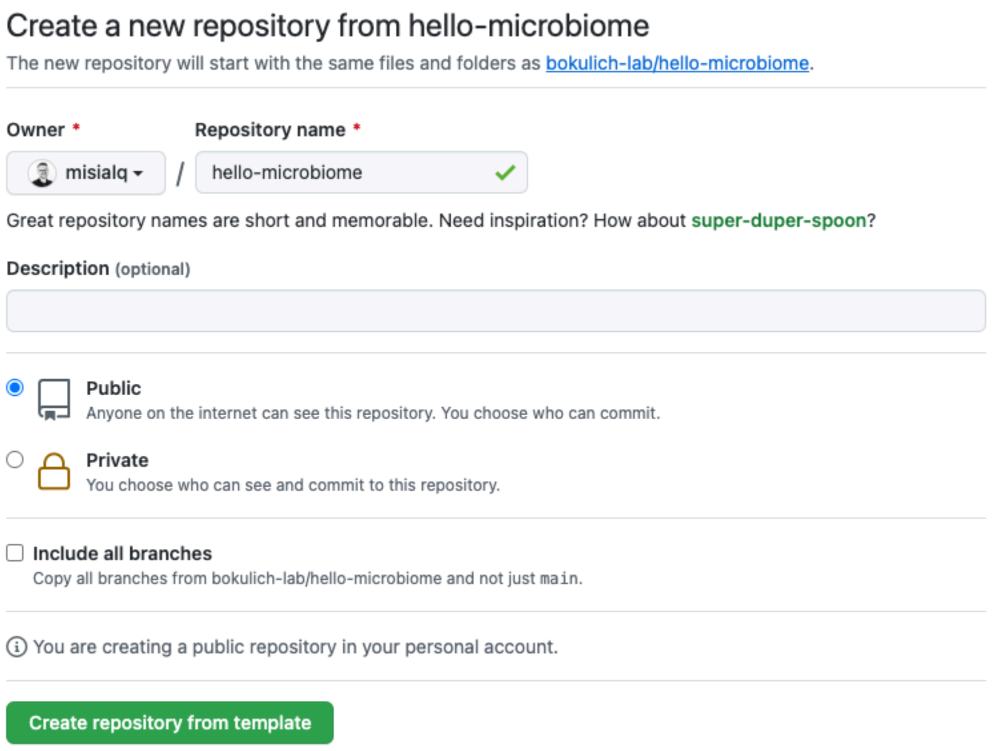
misialq Repo cleanup. 40fb5b1 2 minutes ago 12 commits

README.md	Initial commit	11 months ago
foods.txt	Repo cleanup.	2 minutes ago

README.md

hello-microbiome

Figure 1: GitHub - template repository, step 1


Create a new repository from hello-microbiome

The new repository will start with the same files and folders as bokulich-lab/hello-microbiome.

Owner * misialq / Repository name * hello-microbiome

Great repository names are short and memorable. Need inspiration? How about super-duper-spoon?

Description (optional)

[✓ Public] Anyone on the internet can see this repository. You choose who can commit.

Private You choose who can see and commit to this repository.

Include all branches Copy all branches from bokulich-lab/hello-microbiome and not just `main`.

You are creating a public repository in your personal account.

Figure 2: GitHub - template repository, step 2

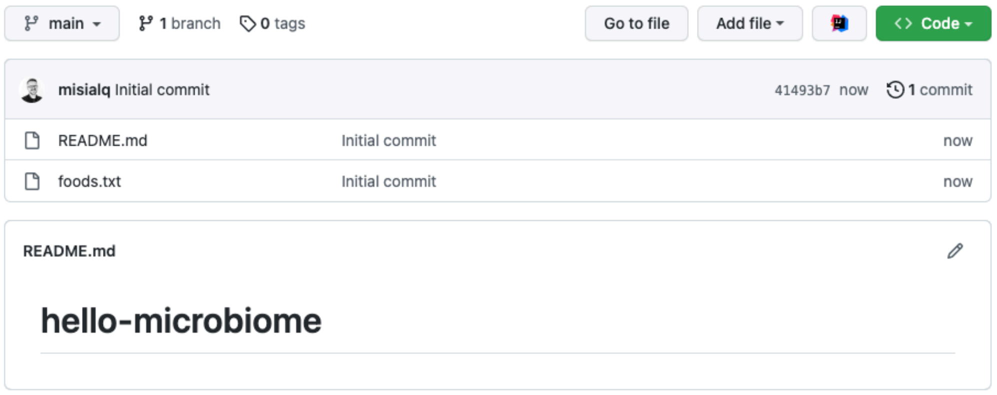


Figure 3: GitHub - template repository, step 3

2.3 Git workflow

Let's spend a moment on understanding the basics of working with `Git`. The typical workflow consists of the following steps (see figure 4 below for a visual representation of those):

1. You create a repository using one of the available hosting tools (e.g., GitHub; that's what we did in the steps above).
2. You **clone** (copy) the repository to your local machine (Fig. 4, step 1).
3. You work on files, make changes, add/remove files. You **add** the modified files to the so-called *staging area* to indicate to `Git` that you want to track (record) changes in those files.
4. You **commit** the changes (to *commit* means to record in the history of the repo; Fig. 4, step 2).
5. You **push** the changes to the repository on the GitHub's server - think about it as a backup of your data (Fig. 4, step 3). Many people can push to the same repository and work together on the same project (you will do that later in this class, too).
6. If some changes were introduced to the files stored on GitHub (e.g., someone else pushed their commits after yours or you changed the files using GitHub's interface; Fig. 4, step 4) you need to **pull** those changes to your local repository before you continue working (Fig. 4, step 5).

Let's try to put the things we've learnt so far together and see them in action in our JupyterHub environment.

2.4 JupyterHub's Git integration

The JupyterHub environment comes equipped with an extension allowing for easy interaction with `Git` and GitHub. You can access the extension from the toolbar on the leftmost side of your workspace (figure 5).

1. Before we continue, go the "*hello-microbiome*" repository that you created in your GitHub account. On its main page, click on the green "Code" button and copy the https link from the "Clone" section (figure 6).

2. Back in the JupyterHub workspace, on the Git extension tab you should see three options (the blue buttons):

   - Open the FileBrowser
   - Initialize a Repository
   - Clone a Repository

   Select the "Clone a Repository" one and paste the link you copied above in the pop-up window. This

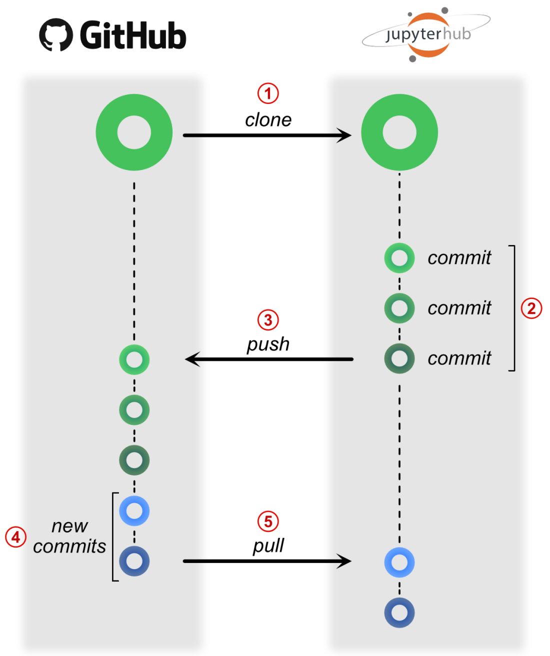
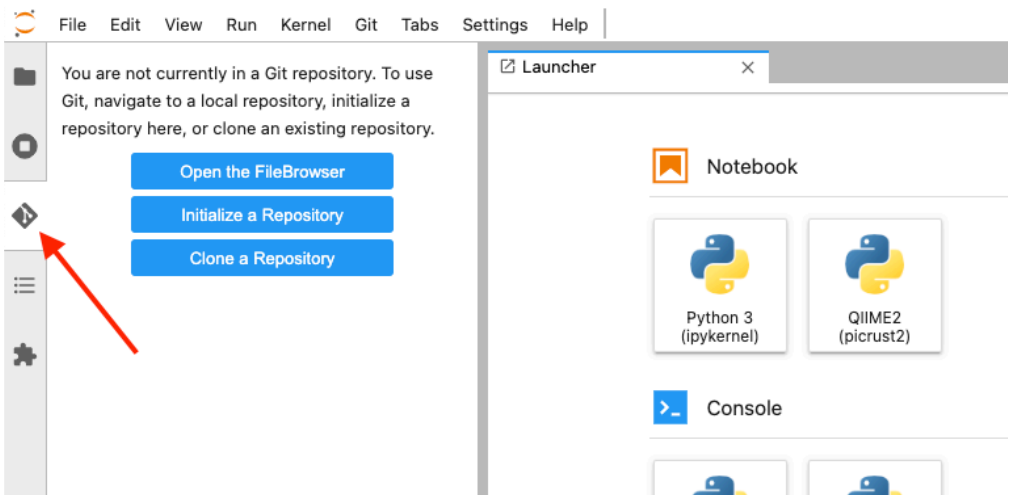


Figure 4: Git workflow



You are not currently in a Git repository. To use Git, navigate to a local repository, initialize a repository here, or clone an existing repository.

Figure 5: JupyterHub - Git extension

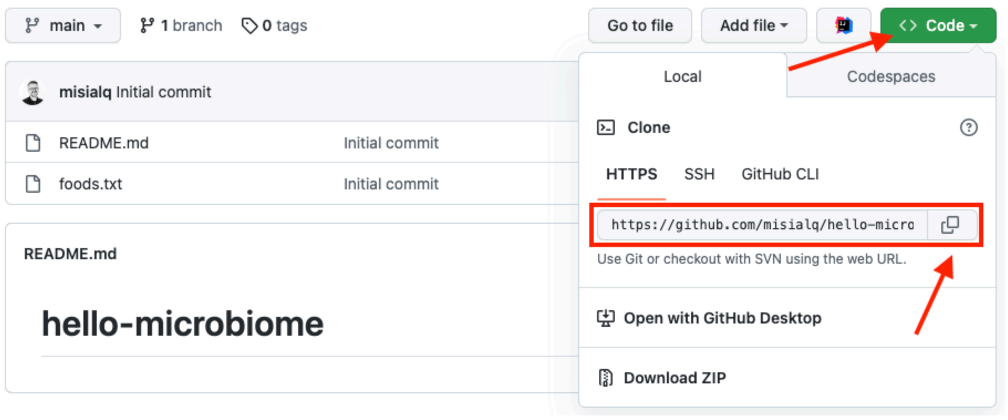


Figure 6: Cloning a Git repository

should have copied the repository from GitHub to you workspace - you should now see a new folder in the directory tree of your workspace (named the same as the repository you cloned).

3. Navigate to this folder by double-clicking on its name and open the Git extension tab (if not already open). You should now see something similar to figure 7:

   You should see the name of your repository on the top and the name of the current branch (we will not discuss branches here, but you can think of them as different versions of your work - Git allows you to switch between different versions of your project and work on those simultaneously). Moreover, on the "Changes" tab you will later be able to see which files were changed and what will be committed (saved in history) - right now this section is empty as we have not changed anything yet.

4. Speaking of changes - let's make some! Navigate to the repository overview by clicking on the little folder icon at the top of the leftmost toolbar. Open the "foods.txt" file by double-clicking on it. Add your favourite fermented food to the list and save the file.

5. Go back to the Git tab. In the "Changes" section you should now see that the "foods.txt" file appeared in the "Changed" section. This indicates that the file was changed, but is not yet *staged* for the next commit, i.e. those changes would not be saved in history. Click on the little "plus" sign next to the file to *stage* it (see the top panel of figure 8). The file should get moved to the "Staged" section (middle panel in figure 8)

6. We are ready to finally *commit* our changes! Enter a descriptive message in the box at the bottom of the Git panel (see the bottom panel of figure 8) and click "COMMIT". Congratulations, you just committed your first changes! Try to add more foods to the list and repeat the process to practice.

7. To see the history of your changes (in a form of commit list) move to the "History" tab. Let's say you want to find out what was changed between two commits. We can easily compare two commits here and see the differences between the modified files. To compare two commits do the following:

   a) next to the first commit, click on the little page with the arrow pointing right (figure 9, step 1)
   b) next to the second commit, click on the little page with the arrow pointing left (figure 9, step 2)
   c) under the commit list, click on the little +/- icon next to the file you want to compare changes for (figure 9, step 3)
   d) compare the differences in the new tab that opened (figure 9, step 4) - lines that were added are highlighted in green and the ones that were removed are red
   e) close the comparison tab when done

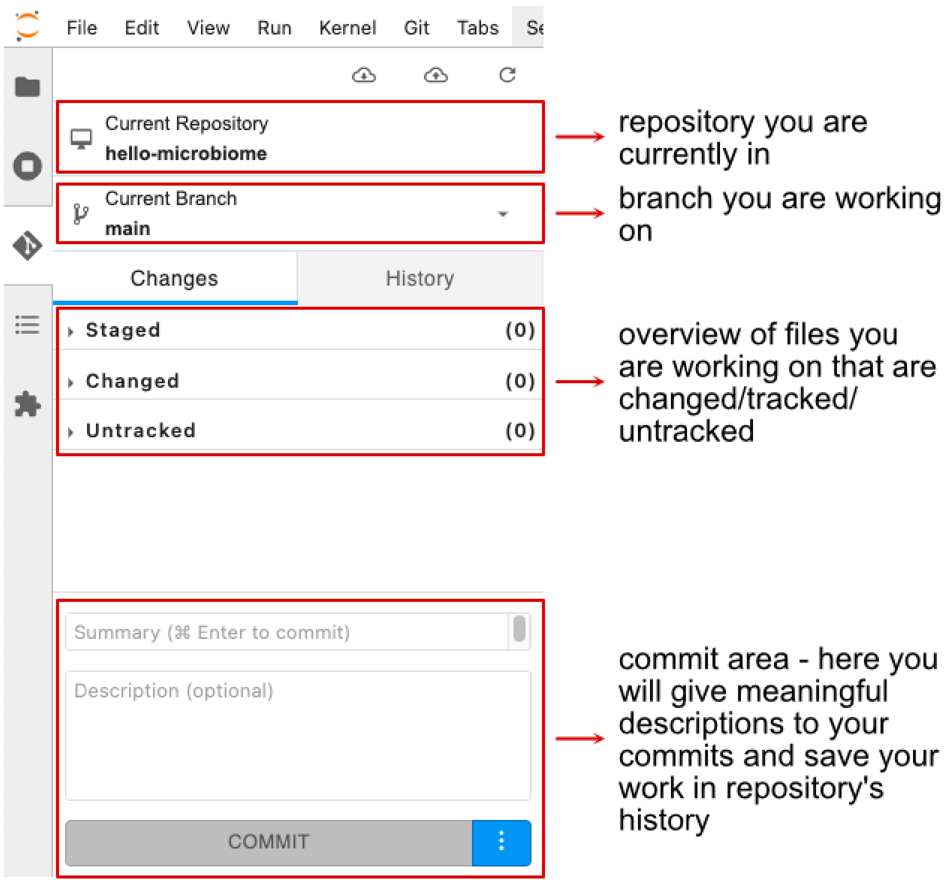

File
Edit
View
Run
Kernel
Git
Tabs
Se
Current Repository
hello-microbiome
Current Branch
main
Changes
History
Staged (0)
Changed (0)
Untracked (0)
Summary (⌘ Enter to commit)
Description (optional)
COMMIT
repository you are currently in
branch you are working on
overview of files you are working on that are changed/tracked/ untracked
commit area - here you will give meaningful descriptions to your commits and save your work in repository's history

Figure 7: Git extension - overview

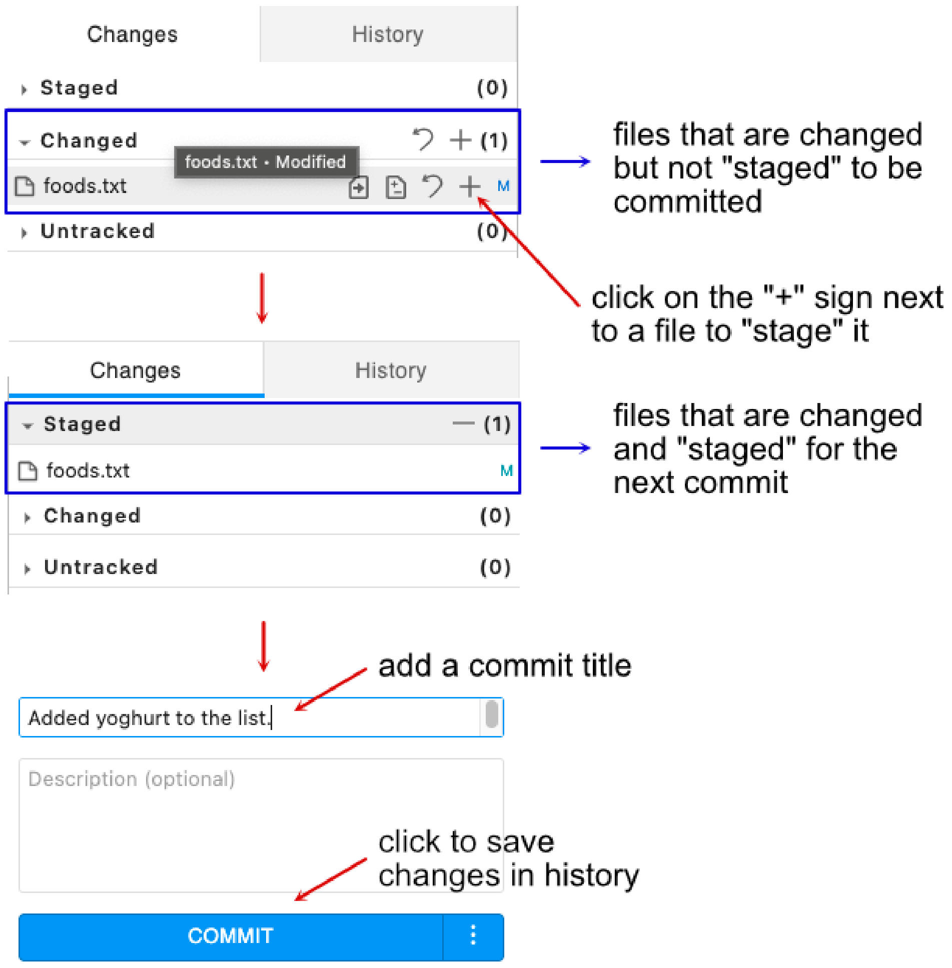
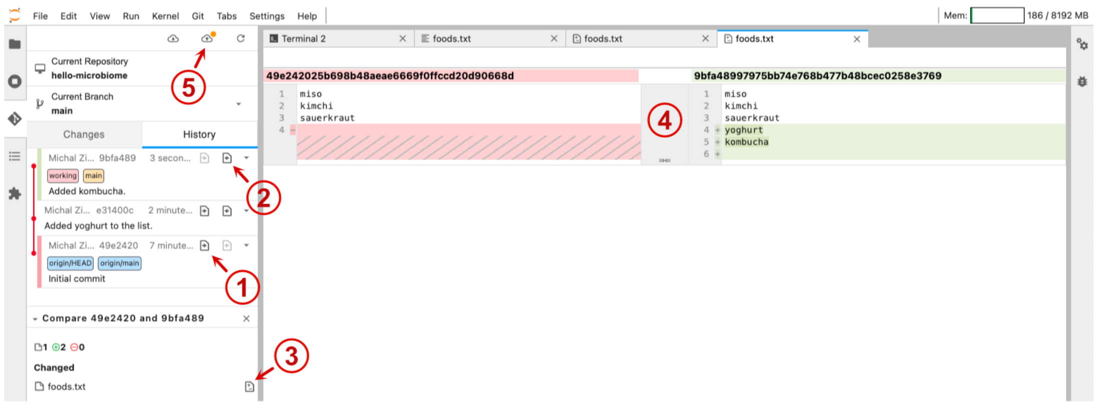

Changes
History
Staged (0)
Changed (1)
foods.txt • Modified
foods.txt
M
Untracked (0)
files that are changed but not "staged" to be committed
click on the "+" sign next to a file to "stage" it
Changes
History
Staged (1)
foods.txt
M
Changed (0)
Untracked (0)
files that are changed and "staged" for the next commit
add a commit title
Added yoghurt to the list.
Description (optional)
click to save changes in history
COMMIT

Figure 8: Git extension - staging files



```
49e242025b698b48aeae6669f0ffccd20d90668d        9bfa48997975bb74e768b477b48bcec0258e3769
1  miso                                          1  miso
2  kimchi                                        2  kimchi
3  sauerkraut                                    3  sauerkraut
4 -                                              4 + yoghurt
                                                 5 + kombucha
                                                 6 +
```

Figure 9: Git extension - comparing files

8. Finally, let's push our changes to GitHub. Click on the little cloud icon with the arrow pointing up to upload the changes to your repository on GitHub (figure 9, step 5).

9. To check that it all worked, go to your repository's main page on GitHub and open the "foods.txt" file - it should now contain the new foods you added in this tutorial.

For more information and tutorials on how to use `Git` feel free to check out the Atlassian Git resources.

3. Cheatsheet

This section is optional. Here you will find some commands that are useful when working with `Git` on a daily basis using command line. You may need to use these if you want more control over the actions you are performing when using `Git`, as the Git extension in the JupyterHub workspace only allows us to use the very basic `Git` functionalities.

- list all local branches:

  ```
  git branch
  ```

- create a *new* branch:

  ```
  git checkout -b <branch name>
  ```

- switch to an already *existing* branch:

  ```
  git checkout <branch name>
  ```

- check status of the working tree/staging area:

  ```
  git status
  ```

- pull the latest changes from the upstream:

  ```
  git pull upstream main
  ```

- add new and/or modified files to the staging area (prepare them for committing):

  ```
  git add <list files here>
  ```

- compare current changes with the last commit:

  ```
  git diff
  ```

- commit current changes (save them to version control "history"):

  ```
  git commit -m <your descriptive message>
  ```

- undo one commit (move changes back to staging area):

  ```
  git reset --soft HEAD~1
  ```

- push your work to your remote repository:

  ```
  git push origin <branch name>
  ```