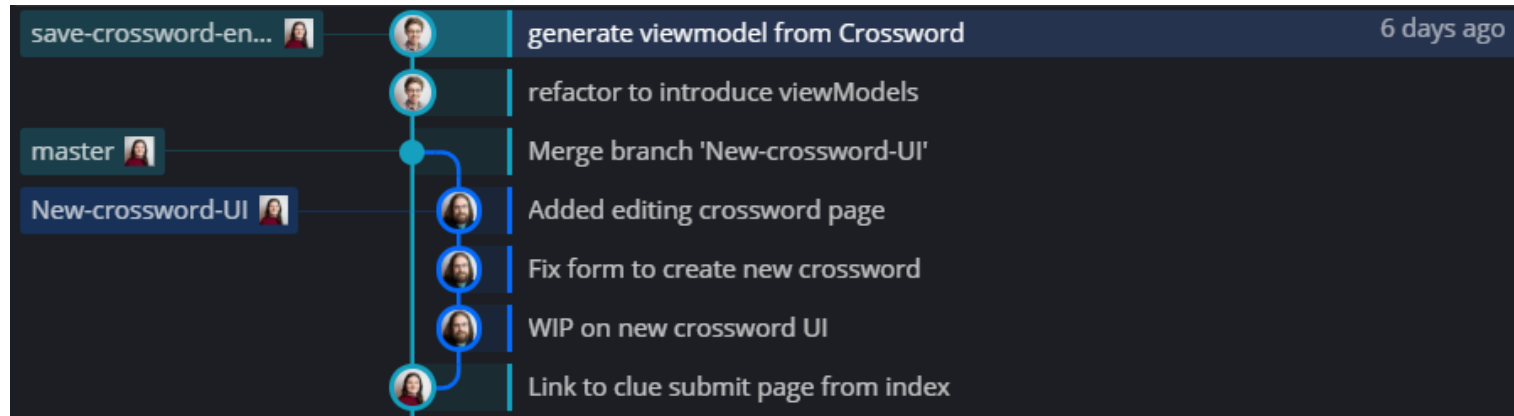


Part 2: Version Control In Git

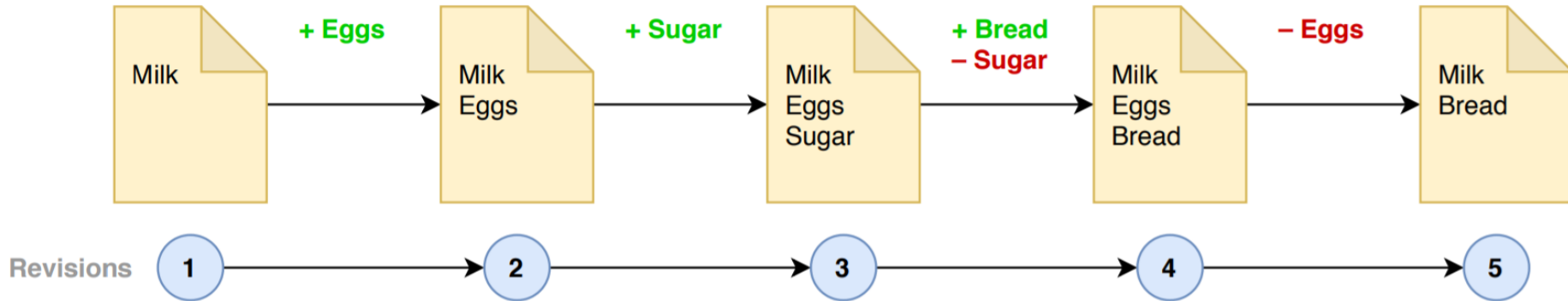
Purpose of Git

Git is a version control system that allows you to track changes to your codebase over time, examine the differences between the current state of your codebase and the last known working version, as well as check out your codebase as it existed at various points in time:



Commits

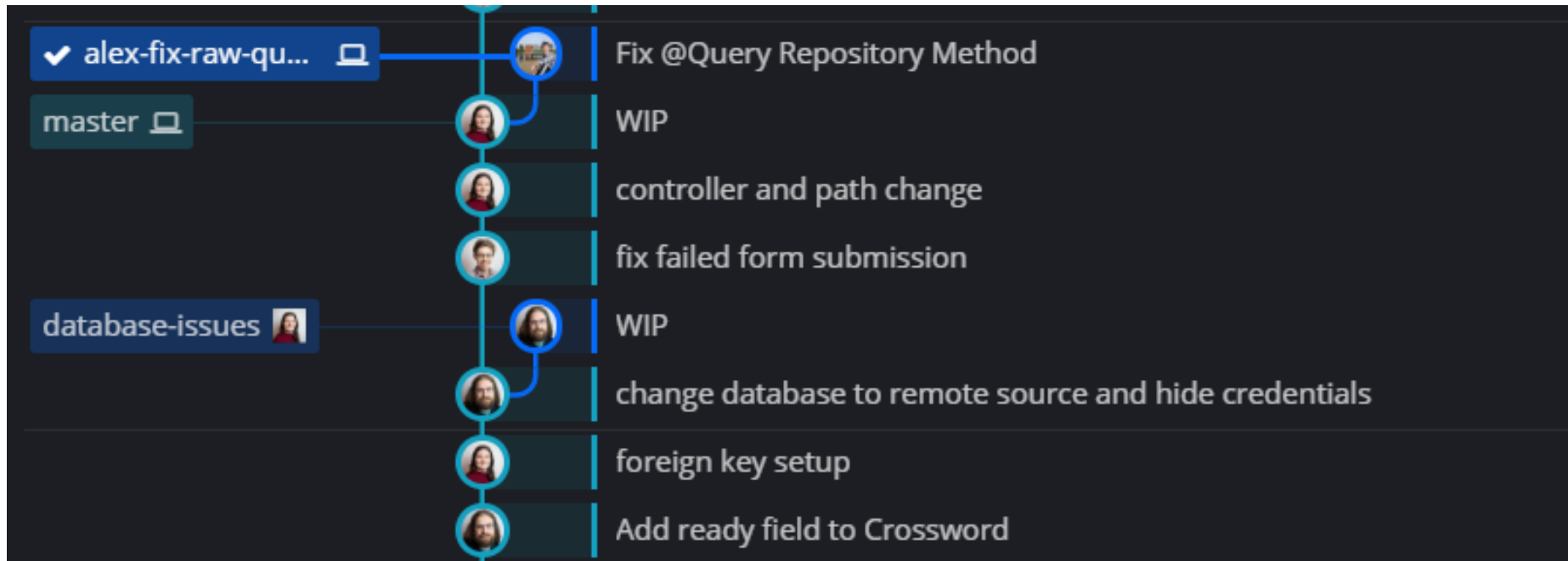
"Commits" are snapshots of your code that you've saved (like working on an important document and having copies called "My doc V1.docx", "My doc V2.docx", "My doc V3.docx", etc).



Every time you make a commit, you build on the previous one, making a chain of commits in a sort of timeline, retaining the ability to "time travel" to any point in time.

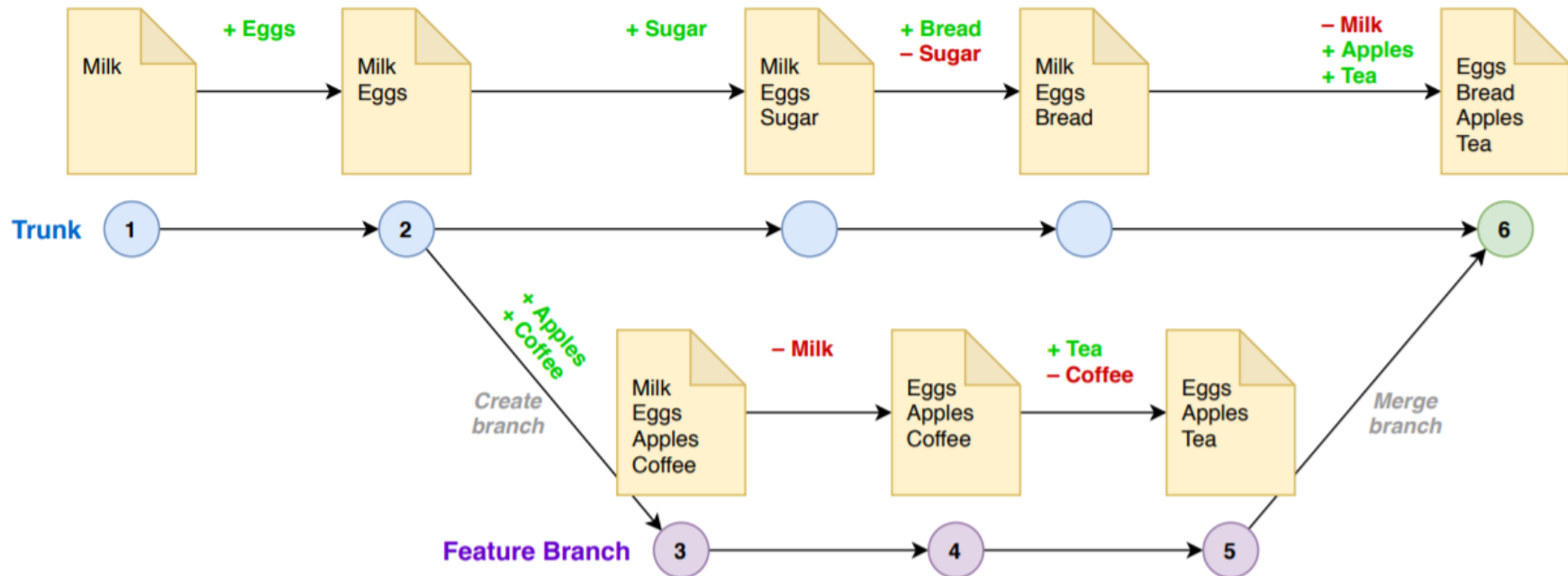
Branches

You can also make "Branches" - parallel universes in which your codebase looks slightly different. You can travel to different branches too.



Merging Of Branches

Branches can be merged into each other (typically creating a “merge commit”)



Basic Git Commands

Here is a summary of all the git commands you need to know about right now:

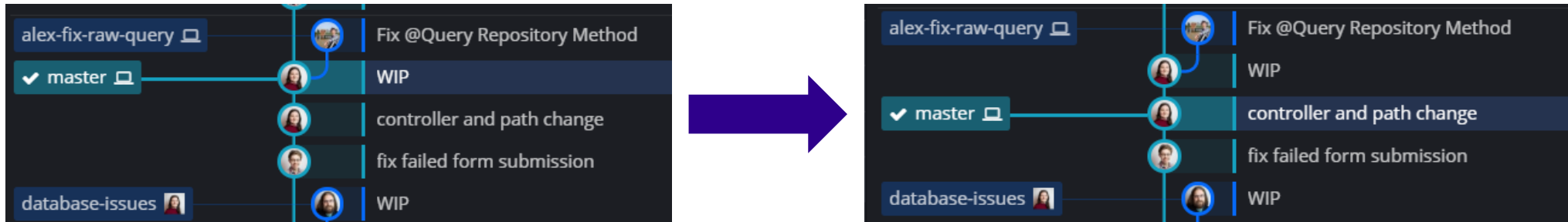
```
$ git status
$ git init
$ git add <path to file>
$ git commit -m "<commit message>"
$ git branch <new branch name>
$ git checkout <target branch>
$ git remote add <remote name> <remote url>
$ git fetch
$ git pull
$ git push
$ git merge <branch to be merged into current branch>
```

Part 2: New Concepts

Resetting

In Git, branches are merely a movable pointer/label on a commit. The label moves when you commit to that branch.

Resetting is a branch operation that moves the label directly:



Resetting (Continued)

The command for resetting is

```
$ git reset [--soft] [--hard] <location (can be branch, commit hash, etc.)>
```

A **soft** reset (--soft) will only move the position of branch label (meaning that git will now notice a lot of staged changes)

Using a **hard** reset (--hard) will change all tracked files to match the new commit along with removing any new files that were added since.

The location is very flexible and you can even specify commits relative to the position of the current branch:

```
$ git reset --hard HEAD~2 # Move the current branch back 2 commits
```

Rebasing Branches

Rebasing is where a branch's commits are replayed on top of another branch. It can be used as an alternative to merging branches.

This can be useful for simplifying a repository's commit history but does change that history permanently (so is not without risk).

The most common use for this is where you want to replay a feature branch on top of the latest state of the master branch:



Rebasing Branches (Continued)

The command for rebasing is:

```
$ git rebase <target branch> # Assumes checked-out branch is the branch to be rebased  
$ git rebase <branch to rebase> <target branch> # Alternative to checking out branch beforehand
```

Interactive rebases provide customisation of how the branch's commits should be replayed (typically an editor will open providing you with instructions on your customisation options):

```
$ git rebase -i <target branch> # After checking out branch to be rebase  
$ git rebase -i <branch to rebase> <target branch> # Alternative to checking out branch beforehand
```

Warning: Like merging, rebasing can lead to merge conflicts!

Part 2: Exercise

The Goal

Setting up a GitHub repository with your code from part 1.

You will:

- Create a local repository, with some initial commits and branches
- Create an online repository for your code and push up code from your local repository
- (Extension) Create some pull requests for your repository
- (Extension) Create (!) and resolve a merge conflict

Step 1

From the directory of your code from part 1, initialise a git repository and commit everything you have as an initial commit.

- If you don't have git on your machine at this point please download Git from <https://git-scm.com/downloads>
 - a) Add a readme file (called README.md) that describes how to use your code (in simple plaintext) and make a new commit with a message "Added README"
 - b) Add a text file with the name NewFile.txt to the directory of your local repository and create an appropriate commit.
 - c) Create a new branch called "reset", checkout this branch and then reset the branch to before the file in point b) was added (try a soft and then a hard reset).
 - d) Checkout the master branch. Next create and checkout a branch called "revert". Finally create a revert commit for the file addition (hint: use `git revert`)

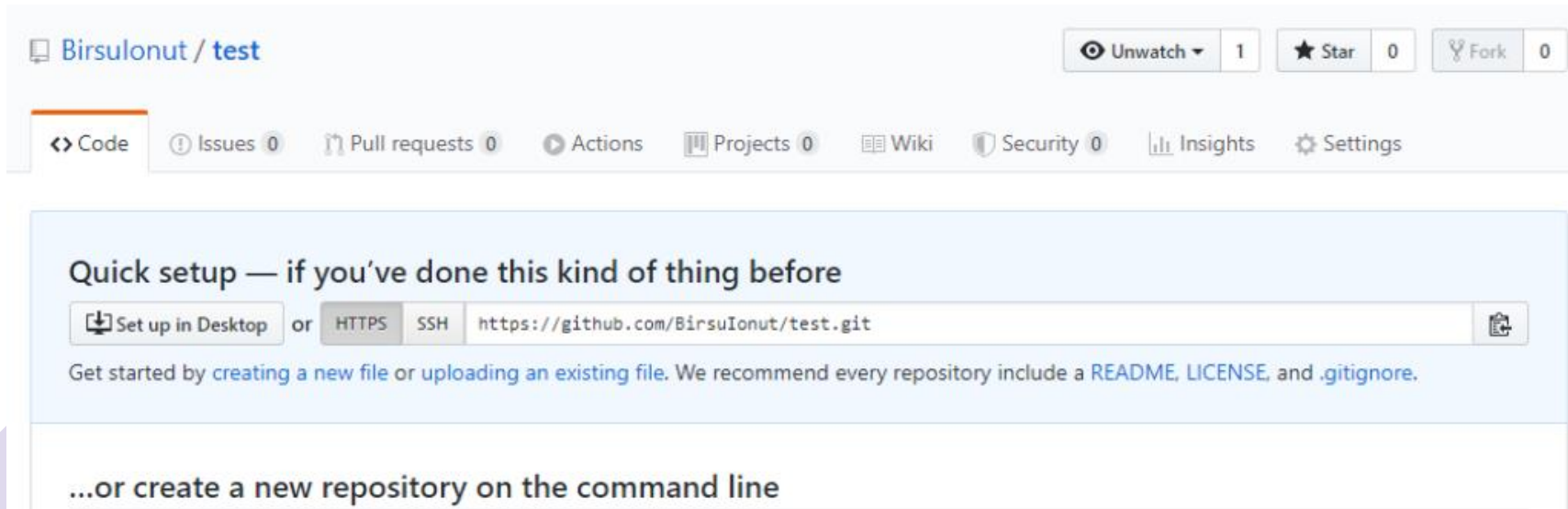
Before moving to step 2, please allow your trainer to review the state of your git repository

Tip: You may want to run a git visualiser tool like gitk to track your commits/branches

Step 2 (Page 1 of 2)

Next, let's push up the code from parts 1 & 2 to a remote repository:

- a) If you don't have an account already, create one on <https://github.com/>
 - You may want to create a personal one if you only have a work account
- b) Create a new repository, giving it a name and leaving everything else untouched. You should now see this page:



The screenshot shows the GitHub interface for a repository named 'test' by user 'Birsulonut'. At the top, there are buttons for 'Unwatch' (1), 'Star' (0), and 'Fork' (0). Below this is a navigation bar with links for 'Code', 'Issues' (0), 'Pull requests' (0), 'Actions', 'Projects' (0), 'Wiki', 'Security' (0), 'Insights', and 'Settings'. The main content area has a heading 'Quick setup — if you've done this kind of thing before'. Below the heading are two buttons: 'Set up in Desktop' and 'HTTPS'. To the right of these buttons is a text input field containing the SSH URL 'https://github.com/BirsuIonut/test.git'. Below the input field is a text prompt: 'Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).' At the bottom of the section is a link: '...or create a new repository on the command line'.

Step 2 (Page 2 of 2)

- d) Follow the instructions under "**...or push an existing repository from the command line**"
- e) You will be prompted to provide your GitHub username and password in the terminal
 - If you prefer not to enter your credentials over a terminal you can create a SSH key to use with your GitHub account. This will involve running:

```
$ ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
```

And uploading the key to GitHub as described [here](#)
(Ask a tutor if you want more details)

Refresh the page and now you should be able to see your pushed code. Please show the results to your tutor.

Step 3

Next, let's create a pull request:

- a) Create a new branch (you can call it something like **pow-operator**)
- b) Change the code from part 1 by adding a new operator, **pow**

```
calc ^ 2 5
```

- c) Afterwards, make a commit and push the new branch to origin
- d) On the repository page, go to branches and create a **New pull request** (you can also add a comment about the changes you made). For now, **do not merge it**.

Step 4 (Page 1 of 2)

Finally, let's resolve a merge conflict:

- a) Checkout master and create a new branch, this time called **mod-operator**.
- b) Change the code from part 1 by adding a new operator, **mod**.

```
calc % 2 5
```

For the purposes of this exercise, you should have some overlapping work with the **pow-operator** branch (e.g. new work inside the same if-clauses, or adding functions at the end of the files, etc).

- c) Make a commit and push the new branch to origin.

Step 4 (Page 2 of 2)

- d) Merge the **pow-operator** branch (you can also safely delete the branch, as github will suggest you).
- e) Create a new pull request, this time for **mod-operator** branch. This time though, you will get some warnings about some existing conflicts.
- f) Go ahead and press **Resolve conflicts** button and resolve the conflicts GitHub couldn't solve on his own. For each file, when you finished, press **Mark as resolved**.
- g) After resolving all the conflicts, commit merge and merge the branch into master.

Step 5

Finally, let's try an interactive rebase:

- a) Create a branch called pre-merge that 2 commits behind master (i.e. before the 2 merge commits that were created in step 4).
- b) Create a new branch from master called ***squashed-merge-commits***.
- c) Interactively rebase ***squashed-merge-commits*** onto pre-merge, combining the 2 merge commits into 1 and changing the commit message to reflect both commits.