[**Assignment 1 - 1 3**](#_n1omi3mr2aut)

[**Assignment 1 - 2 3**](#_oq6aq9jq1lnk)

[**Assignment 1 - 3 3**](#_4u9jcyp3ggy6)

[**Assignment 1 - 1 Column Names 3**](#_4bishn1y4pfn)

[**Assignment 1 - 2 Dashboard and Business Ideas Summary 3**](#_gn07r9i4a3h)

[Referral Source Optimization: 3](#_yihd4g97xc4w)

[Geographic Demand Mapping: 3](#_ts756q4dwuvb)

[Age Group Focused Dashboard: 4](#_n4vhkcx7sibw)

[Gender Balance Analysis: 4](#_l4b3erm1xjvz)

[Popular Courses Analysis: 4](#_p1waqos0nk9q)

[Classroom Utilization and Teacher Performance: 4](#_2laojh70opxz)

[Seasonal Enrollment Patterns: 4](#_wr12jiy1ty3d)

[**Assignment 1 - 3 Choosing a Back-end Platform 4**](#_oulijaxerhvc)

[Starting with REST API 4](#_vq6qzzdmw4hh)

[Different options: 5](#_12bjo3hvkwx3)

[Node.js 5](#_ywv6r45a4s1)

[Django 5](#_ppnqwch93hkm)

[Flask 5](#_mdt7dl9rvwkq)

[Choosing a Service for 24/7 Operation 6](#_ta9aeekryigu)

[Google Cloud Platform (GCP) 6](#_2dhc9tsdaxqa)

[Google App Engine 6](#_nznov7hegbhz)

[Google Compute Engine 6](#_f6cnlafjxere)

[Amazon Web Services (AWS) 6](#_3d6qxjj72z1d)

[Elastic Beanstalk 6](#_rm6e87ddxphf)

[EC2 (Elastic Compute Cloud) 6](#_n6vc2a5o8b9q)

[Replit 7](#_yppn359u6fai)

[Replit 7](#_q1ta1hd7okeu)

[Conclusion 7](#_pv9plbei4i39)

[**Assignment 2-1 8**](#_uajzs4ubh7wq)

[JSON File Description 8](#_jxb7ngjx0ss)

[**Assignment 2-2 8**](#_haa9sdnfm0v7)

[**Assignment 2-1 JSON Data Types 8**](#_1uegzmqviiik)

[**Assignment 2-2 JSON Populated Prototype 8**](#_9gzpn1ydasi5)

[**Assignment 3-1 9**](#_uwnh77nb3w0)

[**Assignment 3-2 9**](#_ilh85r7fsaw4)

[**Assignment 3-1 The Words 9**](#_gieyztkxt2m8)

[What is git? 9](#_6kd06pbrz15h)

[Repository: 9](#_aobw85ap03tk)

[● Local repository 9](#_6v9s6b7dxkay)

[● Remote repository 9](#_m6e4axd2fw4s)

[Git clone: 10](#_wkrzehdjgomt)

[Git add: 10](#_knqz7e4kncgx)

[Git commit: 11](#_ige86bqrans3)

[Git push: 12](#_ppm5gue1kl79)

[Git pull: 13](#_kz8dcp2r7ud1)

[Git branch: 14](#_l5x8lq6rzhtk)

[Git checkout: 15](#_sr9bvjlonkg9)

[Git merge: 16](#_eyet3bd6cr4k)

[Conflict: 16](#_lky070fddklv)

[Git cherry-pick: 17](#_30aptey9frj1)

[Git stash: 18](#_3xmka5a4ewbt)

[Git fetch: 18](#_fbzlygbcc84t)

[**Assignment 3-2 Sourcetree 19**](#_1w65czb8kbgw)

[Git push with Sourcetree: 21](#_9zvy5hywuatj)

[Git fetch with Sourcetree 21](#_xvyowlpa9d4g)

[Git branch with Sourcetree: 23](#_wv9oh9gsc439)

[Git merge with Sourcetree: 23](#_c93eov2vuih3)

[Git stash with Sourcetree: 23](#_w5z4hlg3drm0)

[**User Story 25**](#_qx1nf6byywsd)

[Git commit 25](#_jj6abxl045jz)

[Git push 27](#_2xk4po3e5xuz)

[Git branch 27](#_jc96c4rtu5st)

[Git fetch and pull 28](#_inmvivoph2uo)

[Git merge](#_qy8y781vbvm)

Assignment 5

[**Waterfall and Agile: Comparison 31**](#_i1gbzp55lq8y)

[Waterfall System 31](#_trtov1c3tfq1)

[Requirements 31](#_h6ize8rn8b3m)

[Design 32](#_bkqzh84jvgy2)

[Implementation 32](#_glv9f6malo5h)

[Verification 32](#_cyvoyroz2qs7)

[Deployment 32](#_tasjn4qltxaj)

[Maintenance 33](#_su5xod64kz8t)

[Conclusion 33](#_iqox0f23hckq)

[Advantages 33](#_fazszxxpqg1l)

[Disadvantages 33](#_3bymhngztog9)

[**What is Agile? 34**](#_jmdwuxa6hpl)

[**Why do we need Agile? 34**](#_eivpqzgcdju5)

BEKIR INAL | ASSIGNMENT 1

# Assignment 1 - 1

The first part of the project involved selecting potential column names for the website's database. The selected column names are to be listed in a Google Sheets document for easy reference and organization.

# Assignment 1 - 2

This phase focused on conceptualizing various dashboard ideas for the website, aimed at providing the business owner with valuable insights. These ideas include visualizations and analyses of course popularity, demographic data, financial overviews, and more.

# Assignment 1 - 3

The third phase involves selecting a platform to host and run the backend code continuously. This platform needs to be reliable, capable of handling requests from the front-end seamlessly, and should support 24/7 operation without downtime.

# Assignment 1 - 1 Column Names

The column names are listed in a Google Sheets document for easy reference and organization.

[possible\_columns](https://docs.google.com/spreadsheets/u/0/d/1qECsJKSLYxroAj0yeKhEmZtyAE9Sq7R82vDnZXrODsU/edit)

# Assignment 1 - 2 Dashboard and Business Ideas Summary

Various dashboard ideas for the website, aimed at providing the business owner with valuable insights.

## Referral Source Optimization:

Analyze the most effective referral sources based on family input. Use this data to increase advertising in the most productive channels. Pie charts or bar graphs are great for showing referral source proportions

## Geographic Demand Mapping:

Create a map view based on students' postal codes to help the business owner decide on new locations based on population density. Heat maps or geographic maps with postal code data can visualize student concentration for location planning.

## Age Group Focused Dashboard:

Develop a dashboard highlighting the age groups with the highest and lowest enrollment. This information could be used to tailor specific courses or events for popular age groups and adjust

offerings for less popular ones or it could also be used to combine classes with less students. Histograms or bar charts segmented by age groups can display the age distribution and identify key demographics.

## Gender Balance Analysis:

A dashboard to analyze gender distribution. Mixed classes could be considered if there's a significant imbalance, or separate classes if the numbers are more even.(Girls Only Class)

Pie charts or segmented bar graphs for gender distribution can inform class structuring based on gender balance.

## Popular Courses Analysis:

A dashboard showing the most enrolled courses, allowing for adjustments in course offerings based on popularity. Bar charts that rank courses by enrollment numbers can highlight popular courses for potential expansion.

## Classroom Utilization and Teacher Performance:

Examine class attendance rates to assess teacher performance and classroom utilization. This could lead to investigations into low attendance classes or the merging of classes. Line graphs or scatter plots can show attendance trends and help to correlate them with teacher performance.

## Seasonal Enrollment Patterns:

Examine how enrollment numbers fluctuate throughout the year. Use time-series analysis to identify peak enrollment periods, which can help in planning marketing campaigns and staffing. Time-series analysis with line graphs can identify peak enrollment periods, aiding in campaign planning.

# Assignment 1 - 3 Choosing a Back-end Platform

## Starting with REST API

REST API, or Representational State Transfer API, is a set of protocols and standards used for exchanging data and functionality between web services. It usually operates over HTTP, allowing access to resources using simple URLs. In REST, each web service or data object is represented by a URL. Clients interact with these resources using standard HTTP methods:

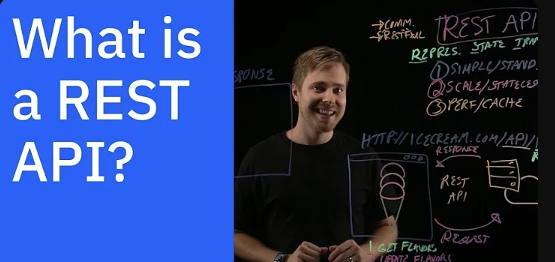
GET: Retrieves data from a server. It's used for reading information.

POST: Sends data to a server to create a new resource. It's often used for submitting form data.

PUT: Updates existing data on a server. This method is typically used to modify an existing resource.

DELETE: Removes data from a server. As the name suggests, it's used to delete a resource.

**Source:**



[What is a REST API?](https://www.youtube.com/watch?v=lsMQRaeKNDk&ab_channel=IBMTechnology)

Now we need to take a look at REST API tools. The main contenders are Node.js, Django and Flask. Each of these has its strengths and is suited for different types of projects.

## Different options:

### Node.js

It's great for building fast, scalable network applications. It's JavaScript-based, which makes it a good choice if you're looking to maintain a consistent language

between the front-end and back-end.

However, it would be very difficult for me to use it since I have never written any code in JS.

### Django

A high-level Python web framework that encourages rapid development and clean, pragmatic design. It's a 'batteries-included' framework, meaning it comes with a lot of built-in features, including an ORM, admin panel, and authentication support. It's well-suited for larger applications or when you need a lot of built-in functionalities.

Since we are not working on a large and complicated project I also won’t use this framework.

### Flask

A micro web framework for Python. It's lightweight and flexible, giving you more control

over the components you use. Flask is ideal for smaller applications or when you want simplicity and the ability to customize your setup extensively. Flask is able to respond to requests from the front-end. It also works very well with WordPress.

In the end, I decided to use Flask due to its ease of use and my little back-end experience. To learn it, I will watch the youtube playlist “**Create a Flask Blog”** from Codemy.com starting with the video below.



[Before You Install Flask...Watch This! Flask Fridays #1](https://www.youtube.com/watch?v=0Qxtt4veJIc&list=PLCC34OHNcOtolz2Vd9ZSeSXWc8Bq23yEz&ab_channel=Codemy.com)

## Choosing a Service for 24/7 Operation

After settling on Flask as our backend framework, the next crucial step is to find a hosting service that allows our application to run continuously, 24/7. We need a platform that's reliable, can handle our traffic demands, and integrates well with Flask. Here's a look at some popular options:

## Google Cloud Platform (GCP)

### Google App Engine

Offers a fully managed platform that can scale your Flask application automatically. It's a good choice for continuous operation due to its robust infrastructure and auto-scaling capabilities.

### Google Compute Engine

Provides customizable virtual machines. While it requires more setup and management than App Engine, it gives you more control over the environment.

## Amazon Web Services (AWS)

### Elastic Beanstalk

Ideal for easy deployment and management of applications. AWS handles the details of capacity provisioning, load balancing, auto-scaling, and application health monitoring.

### EC2 (Elastic Compute Cloud)

Offers scalable virtual servers. It's suitable if you're looking for more control and are comfortable managing server configurations and deployments.

## Replit

### Replit

Known for its ease of use and real-time collaboration features, Replit can host and run Flask applications. However, it's typically more suited to development and prototyping rather than long-term production deployment.

**Always-On:** For applications that need to run continuously, Replit offers an "Always-On" feature, which keeps your server running 24/7. This feature is part of their Hacker plan, a paid subscription service.

## Conclusion

For a Flask application with our current needs, Google App Engine or AWS Elastic Beanstalk are strong contenders for their managed environments and auto-scaling features. Replit's Always-On feature is also an option for a hassle-free setup, especially while the application's scope remains small to medium. As we expect to grow, choosing a platform that balances ease of use with scalability will be key. We'll need to consider where our application will be a year from now to make the best choice for the long term.

But the most important thing for me is that it is an easy and understandable service to see how things are progressing. Replit seems like the most logical choice for this.

About always on feature:

[Replit Hosting Update. Replit Deployments (NO Free plan)](https://www.youtube.com/watch?v=nzbthLDpW4M&ab_channel=Gunther)

BEKIR INAL | ASSIGNMENT 2

# Assignment 2-1

For this part of the project, we will make a prototype of the actual [JSON](https://www.youtube.com/watch?v=7mj-p1Os6QA&t=1s) file that we will use for the project. However, instead of populating the prototype, we only need to specify the data types in the JSON structure.

## JSON File Description

JSON, an acronym for JavaScript Object Notation, is a data structuring format. It is widely used in web applications for data transmission between a server and a client. JSON is notable for its lightweight nature, making it easy for humans to read and write, as well as for machines to parse and generate. It is frequently utilized in APIs and web services to offer a portable, text-based method for representing structured data.

# Assignment 2-2

In the second part of the project, we will populate the prototype JSON file with random data. This step is to provide an illustrative example of what the final JSON file will look like. By doing this, we aim to understand how the actual JSON will be structured and formatted when it is filled with real data.

# Assignment 2-1 JSON Data Types

For sharing the project structure I’ve made in a broncode-editor, I’ve provided a GitHub link below. This allows others to access and view my JSON file prototype and its structure conveniently. GitHub is an excellent platform for version control and collaboration, making it easy to share your work and collaborate with others.

[assignment/json\_types.json at main · inalbekir/assignment (github.com)](https://github.com/inalbekir/assignment/blob/main/json_types.json)

# Assignment 2-2 JSON Populated Prototype

Here, we fill in the data structures we think and make in order to get an idea of how the data we will work with in the future may look like.

[assignment/example\_data.json at main · inalbekir/assignment (github.com)](https://github.com/inalbekir/assignment/blob/main/example_data.json)

BEKIR INAL | ASSIGNMENT 3

# Assignment 3-1

In this part of the assignment we should learn the basic git terms such as commit, pull, merge etc. and we should explain each word with examples.

# Assignment 3-2

For this segment of the assignment, we are tasked with understanding what SourceTree is, exploring its functionality, and employing the words given above.

# Assignment 3-1 The Words

I will be explaining key Git terms such as 'commit', 'pull', 'push', 'branch', 'merge', 'conflict', 'stash', 'cherry-pick', 'checkout', and 'fetch.' Additionally, for better insight and understanding, I've included a few extra terms that are relevant to my research.

## What is git?

Git is a version control system used by developers to track changes in source code during software development, allowing for efficient collaboration and management of project history.

## Repository:

When working with Git, it's important to be familiar with the term **repository**. A Git repository is a container for a project that is tracked by Git.

We can single out two major types of Git repositories:

### Local repository

An isolated repository stored on your own computer, where you can work on the local version of your project.

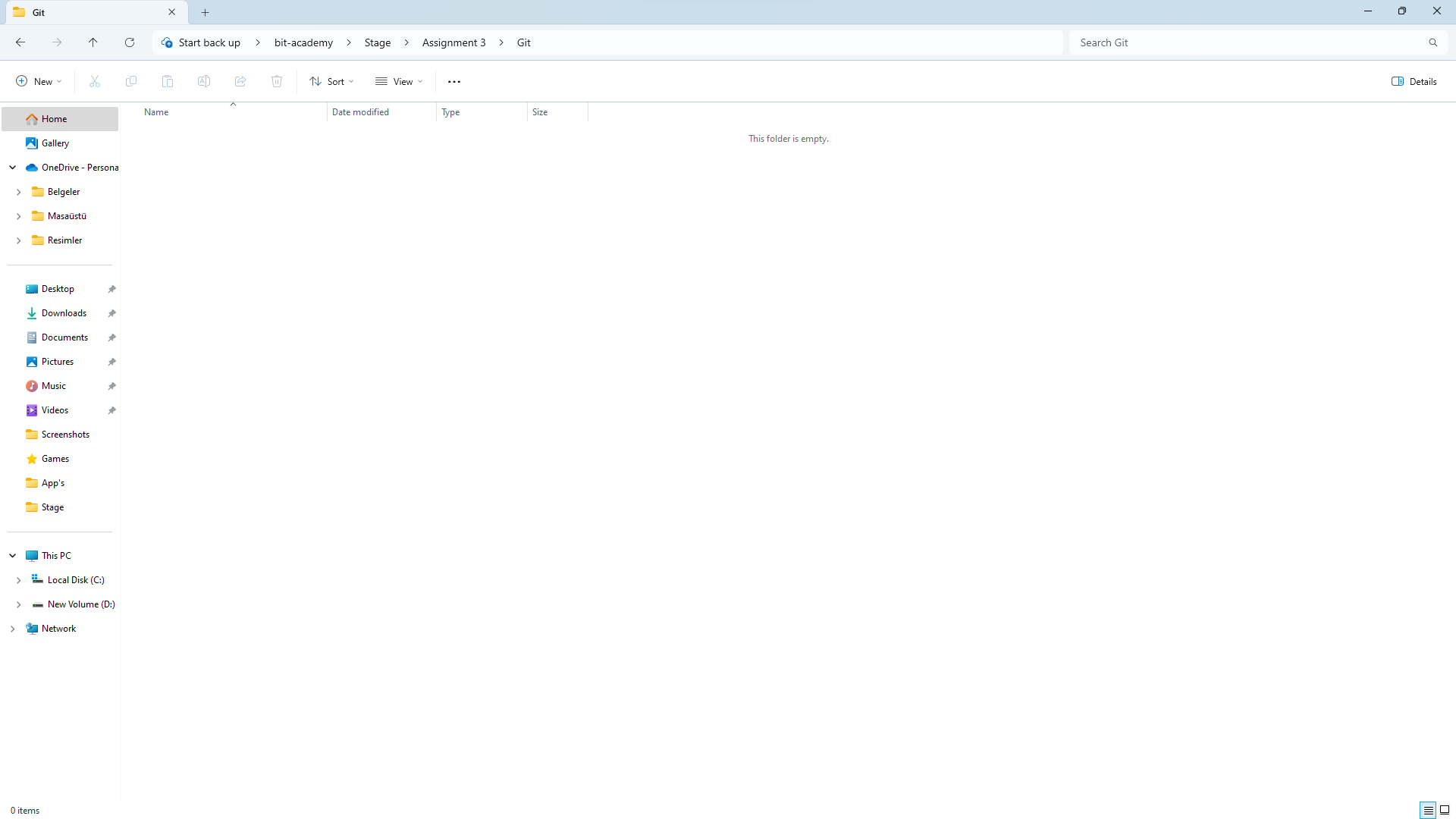
### Remote repository

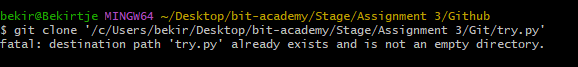
Generally stored outside of your isolated local system, usually on a remote server. It's especially useful when working in teams - this is the place where you can share your project code, see other people's code and integrate it into your local version of the project, and also push your changes to the remote repository.

### Git clone:

To clone a remote repository and start tracking your project with Git, use your terminal software and navigate to the main folder of your project, then type **“git clone (repository)”**.

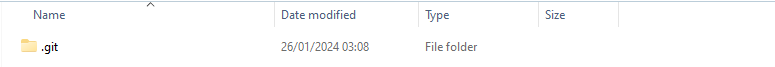
**Before git clone:**





## 

After git clone:



### Git add:

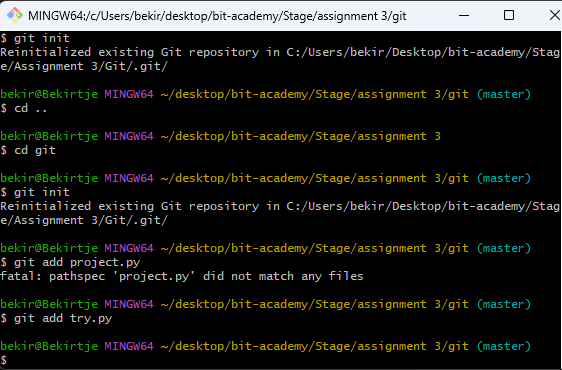
From the project folder, we can use the **git add** command to add our files to the staging area, which allows them to be tracked.

We can add a specific file to the staging area with the following command:

| git add file.js |
| --- |

Instead of having to add the files individually, we can also add all the files inside the project folder to the staging area:

| git add . |
| --- |

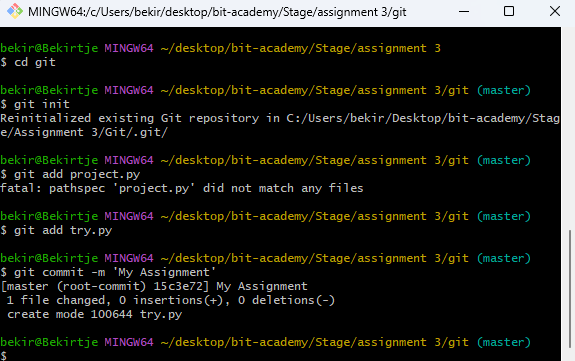


### Git commit:

A **commit** is a snapshot of our code at a particular time, which we are saving to the commit history of our repository. After adding all the files that we want to track to the staging area with the `**git add`** command, we are ready to make a commit.

To commit the files from the staging area, we use the following command:

| git commit -m "Commit message" |
| --- |

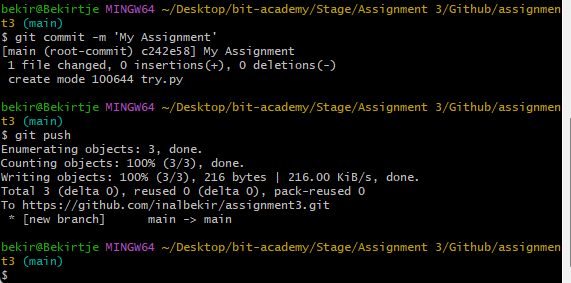


### Git push:

The ’**git push**’ command uploads your local branch commits to the remote repository branch. If you're working with others, this allows them to see the changes you've made. If you're using a remote repository as a backup, git push ensures that your remote repository matches your local one.

To push the files that you already committed, we use the following command:

| git push |
| --- |



Now the changes that I’ve made are also uploaded on the main branch.

### Git pull:

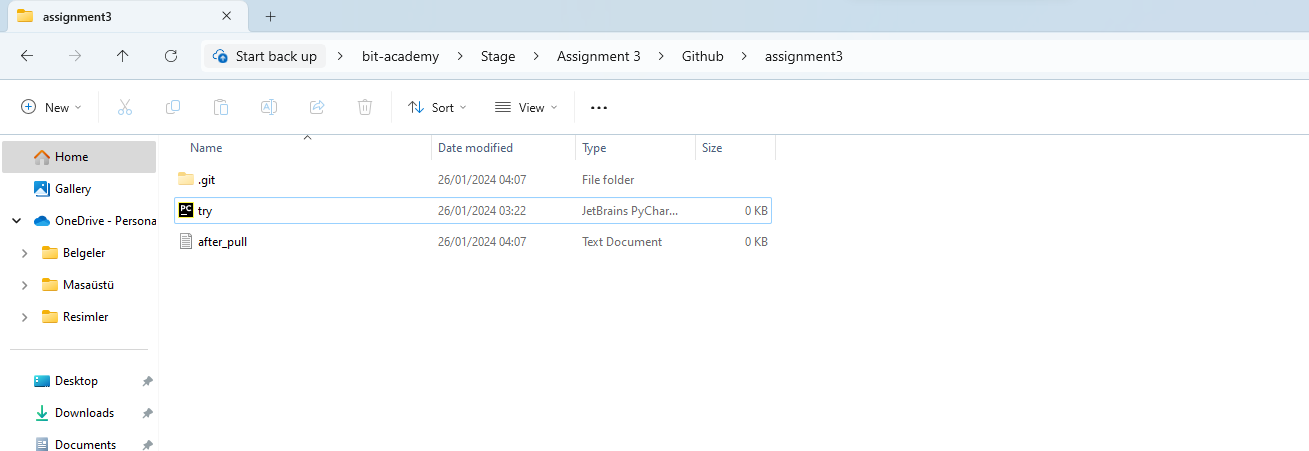
After pushing changes to the remote repository, or to synchronize with contributions from others, we use the git pull command. It fetches changes from the remote repository and updates the local repository to match those changes.

For example, if I upload a file directly to my repository on GitHub—without using command-line tools—I can then use git pull in my local repository to retrieve that new file.

To pull the files that are pushed into the repository from others, we use the following command:

| git pull origin |
| --- |

My folder before updating the local repository

After updating the local repository using pull command:

As you can see after updating the local repository using **git pull,** I’ve got an extra txt file from the repository which I didn’t make myself.

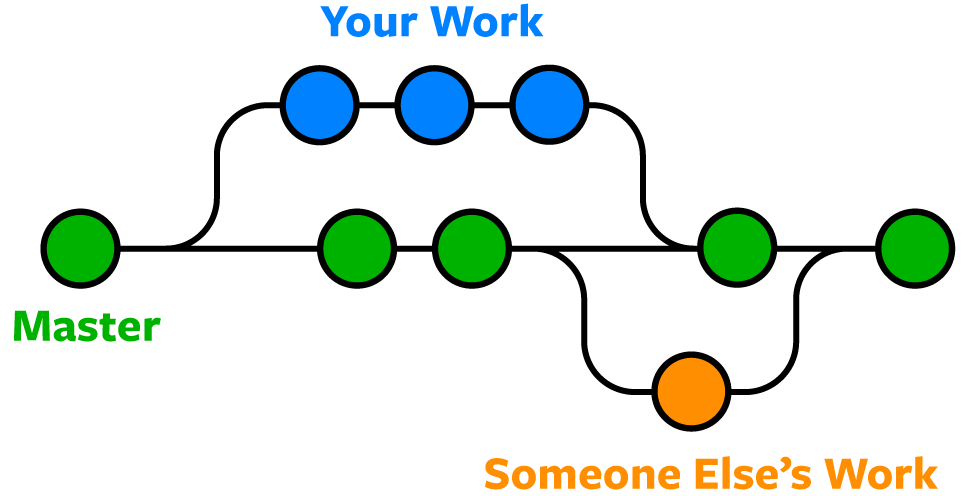
### Git branch:

**Branches:** A branch could be interpreted as an individual timeline of our project commits.

With Git, we can create many of these alternative environments (i.e. we can create different branches) so other versions of our project code can exist and be tracked in parallel.

That allows us to add new (experimental, unfinished, and potentially buggy) features in separate branches, without touching the 'official' stable version of our project code (which is usually kept on the master branch).

When we initialize a repository and start making commits, they are saved to the \*\*master\*\* branch by default.

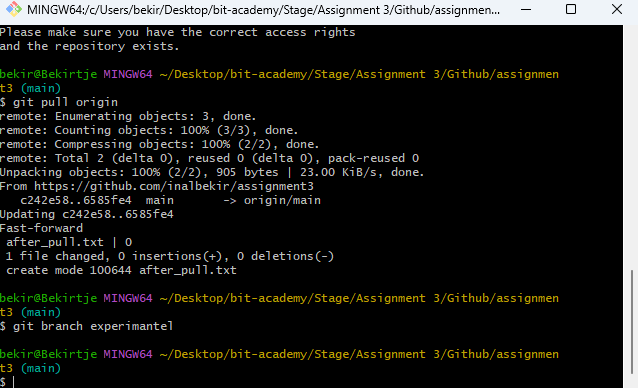
****

**Making a new branch:** You can make a new branch using the following command:

| git branch <new-branch-name> |
| --- |

firstly you need to use the following command to be able to push any files:

| git push --set-upstream origin <branch-name> |
| --- |

****

### Git checkout:

To switch to a different branch, you use the git checkout command:

| git checkout <branch-name> |
| --- |

To create a new branch and change to it at the same time, you can use the -b flag:

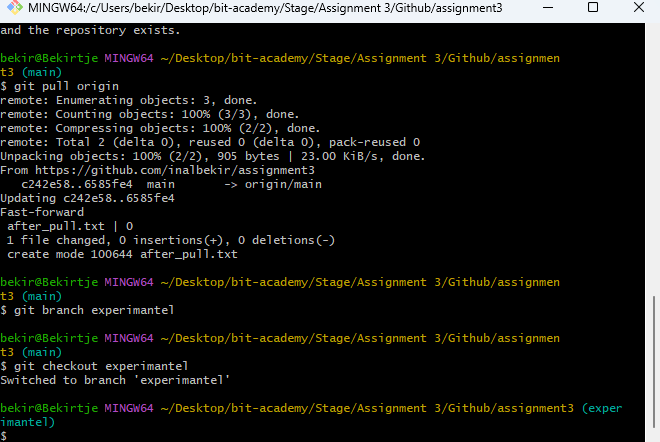
| git checkout -b <new-branch-name> |
| --- |

To go back to the master branch, use this command:

| git checkout master |
| --- |

To delete a branch, you can run the git branch command with the -d flag:

| git branch -d <branch-name> |
| --- |



First I was working in the **main** branch and I switched to experimental using **git checkout.**

### Git merge:

This command is used locally to combine the changes from one branch (like a feature branch) into another branch (like the main or master branch). It’s a way to integrate work from different branches without sharing changes with a remote repository.

To combine the changes from one branch into another branch, you can use following command:

| git merge |
| --- |

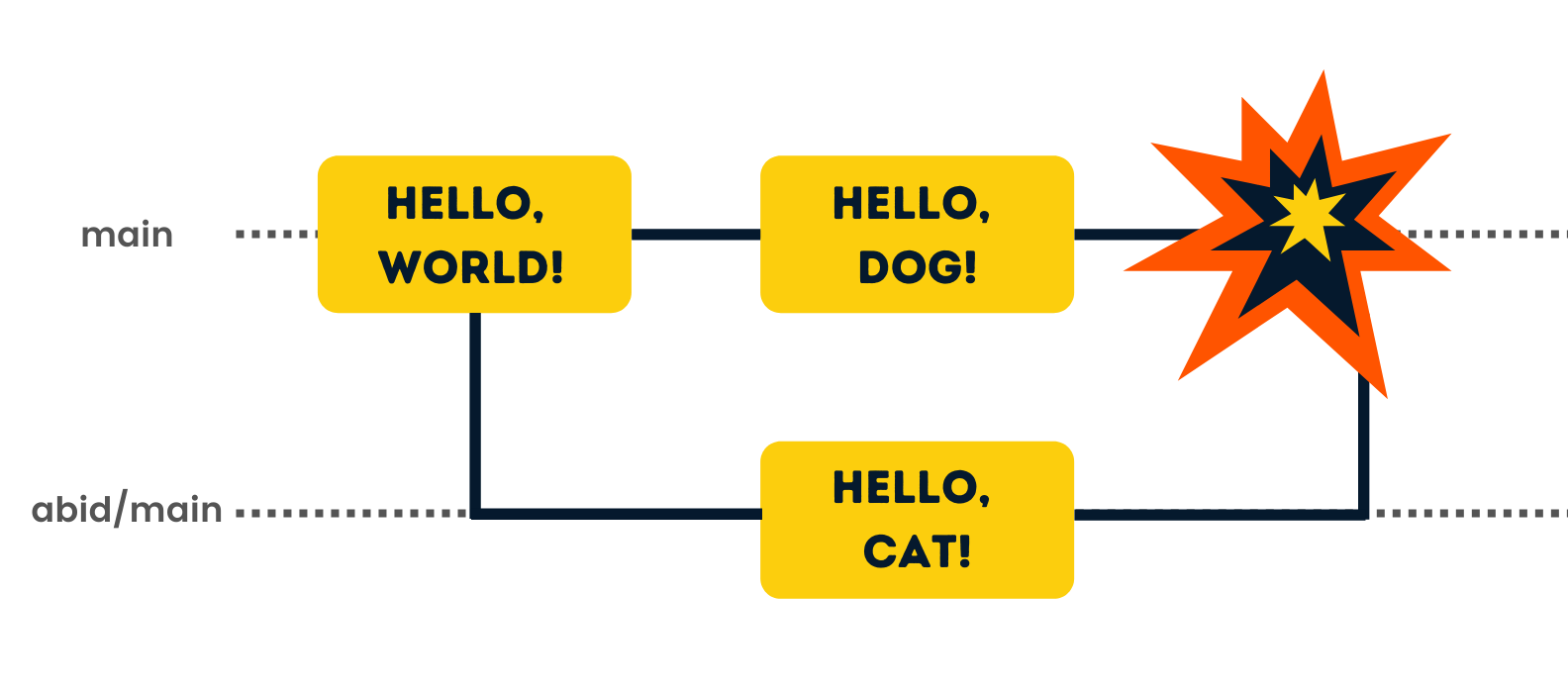
**But if someone also merges to the same line in the same file from other branches, you might get a conflict.**

### Conflict:

A conflict in Git occurs when two branches have changes that cannot be automatically merged. This usually happens when commits are made to the same line of a file in different branches.

When a conflict occurs, you have to choose manually the code that you’re going to be using.

An example of a conflict

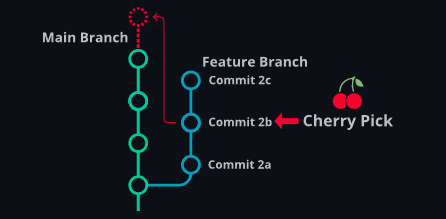
****

### Git cherry-pick:

This is a command in Git that allows you to choose a specific commit from one branch and apply it onto another branch. It's a useful tool for selectively transferring commits without needing to merge or rebase an entire branch.

To only choose 1 commit from feature branch and upload it to the main branch, you can use the following command:

| git cherry-pick |
| --- |



For example, instead of merging and updating the main branch with the whole feature branch, you can only choose one specific commit and continue working on other commits.

### Git stash:

**git stash** is a command in Git that temporarily shelves (or stashes) changes you've made to your working directory, allowing you to work on something else, and then come back and re-apply them later on. Essentially, it takes the modifications you've made to tracked files, saves them away in a stack of unfinished changes that you can reapply at any time, and then reverts them from your working directory. This is especially useful for maintaining a clean working directory without committing incomplete work.

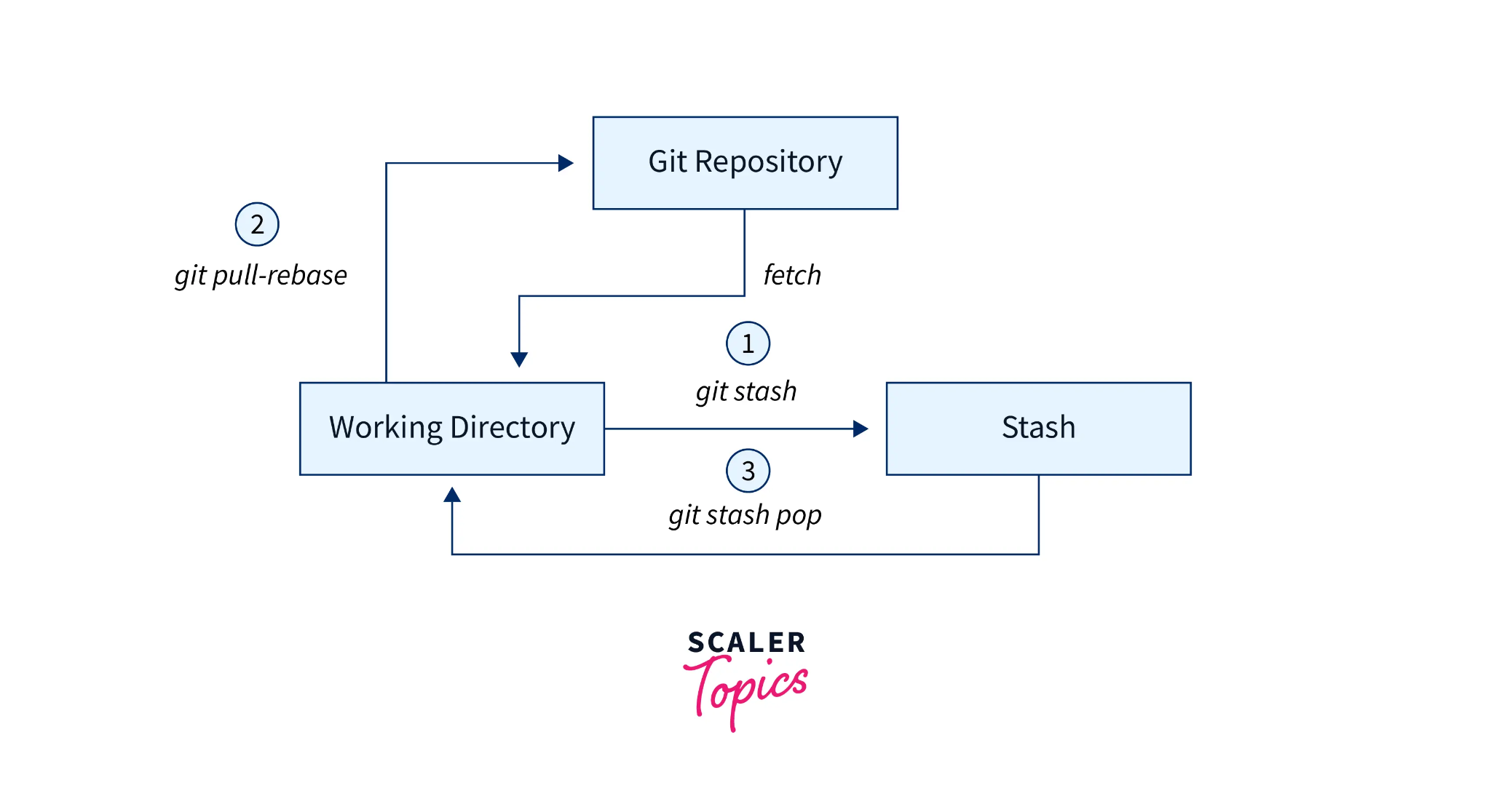
To quickly switch between branches and save your work, you can use the following command:

| git stash |
| --- |

One might prefer using git stash over committing because it allows for quickly switching between branches without the need to commit incomplete work. Stashing is particularly useful and more appropriate when your code isn't ready for a formal commit, offering a cleaner and more understandable way to manage ongoing changes.

To get all the changes back, use the following command:

| git stash pop |
| --- |



### Git fetch:

This is a command used to update your local repository with changes from a remote repository, without merging those changes into your current working branch. This allows you to review updates before deciding to merge them, making it a safe way to see what others have been working on, without affecting your own local development."

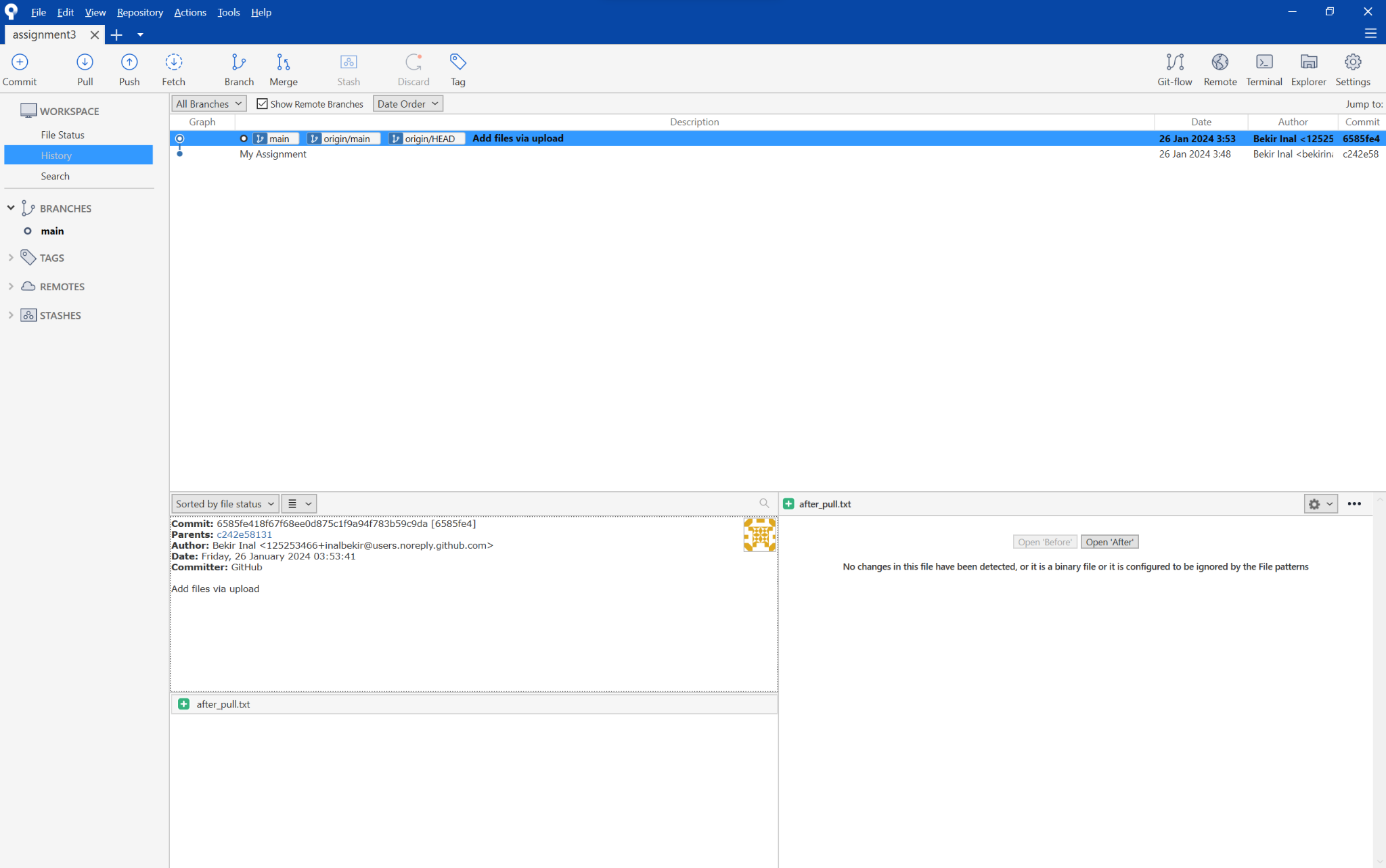
To review the updates before merging, you can use the following command:

| git fetch |
| --- |

The primary difference between git fetch and git pull is that fetch updates your local repository with changes from the remote without merging them into your current branch, while pull updates and also merges those changes into your current branch.

# Assignment 3-2 Sourcetree

First of all, once you have cloned your remote repository in sourcetree, you will get a empty screen like this:

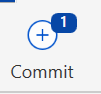
****

As you can see, there is nothing to commit. To be able to commit something we need to make some changes in the folder or in the files in the folder.

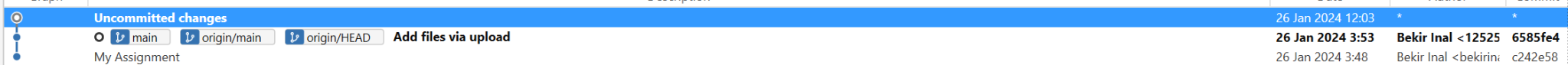
If you look at the top of the screen, you can see buttons to clone, pull, push, fetch, branch, merge and stash. These are the buttons we will be using to do our work. We will start with a commit.

**Git commit with Sourcetree**

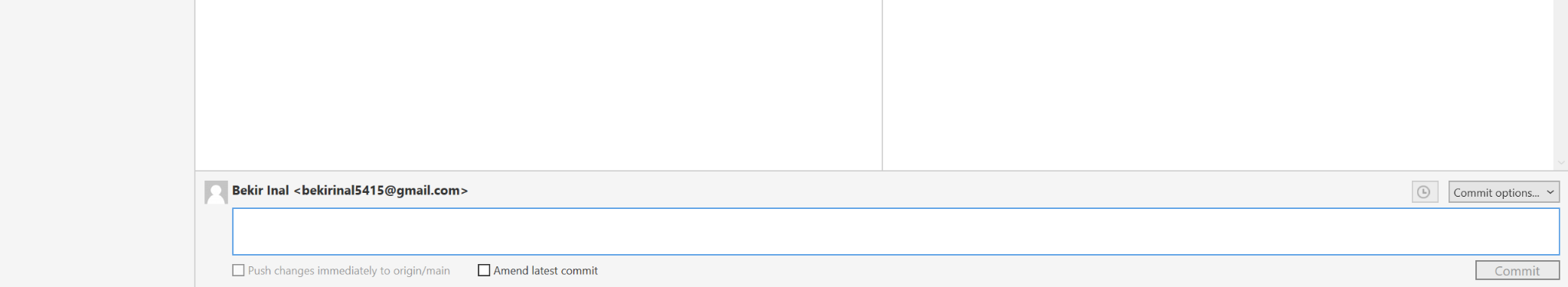
When you have made the changes you want, and you are ready to commit those changes. You have to click on the commit button. Whenever I make a change to the files, a number appears above the commit button. This number indicates both the changes made and how many files were changed.



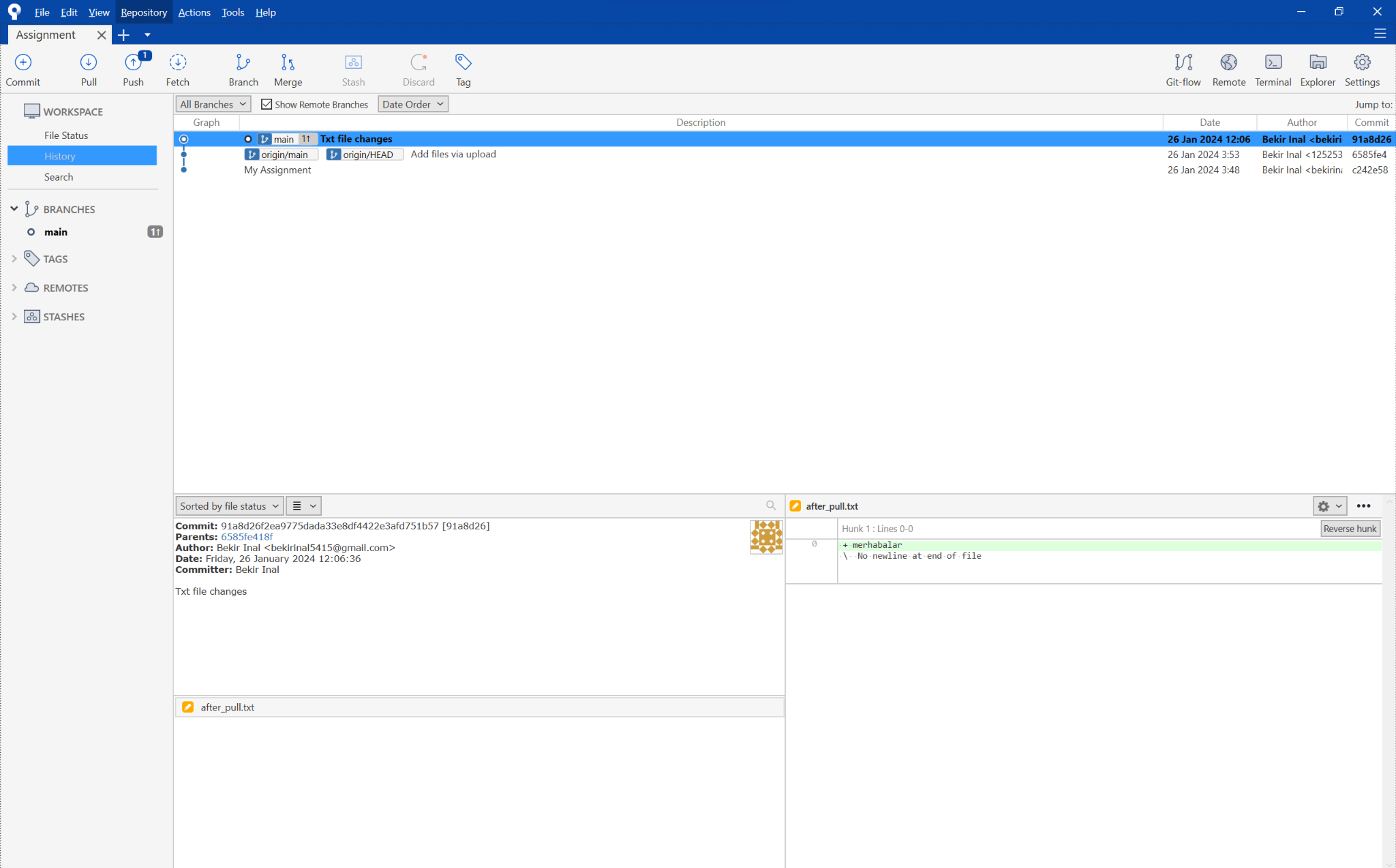
And you can also get a message that says ‘Uncommitted changes’ and you can also see the ID of the commit on the right side of the screen.



When I try to commit these changes, it asks me to add a message. This is actually the same thing as the -m operation in the terminal.



Now the commit appears as a dot on the line. You can also see what you have committed at the bottom right as you can see below.

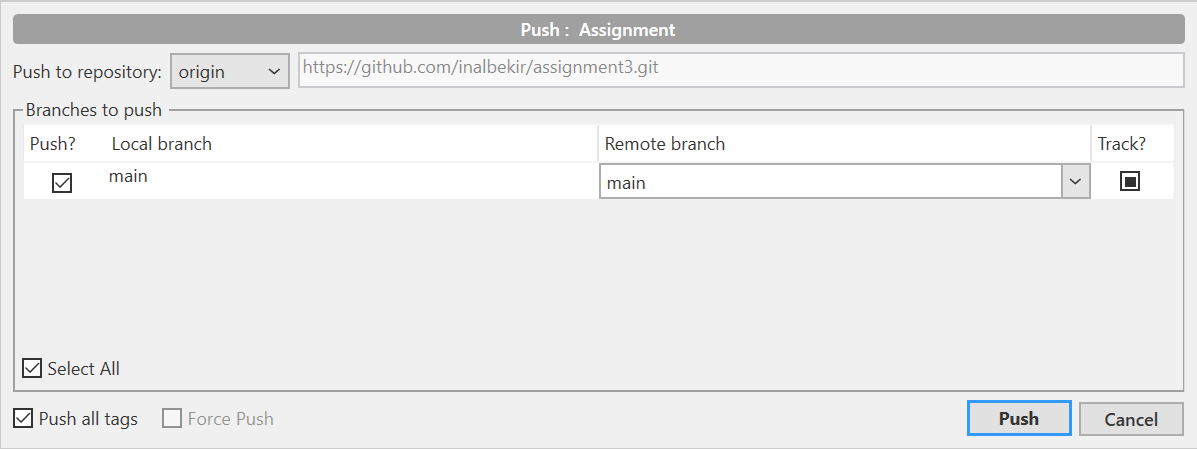


## Git push with Sourcetree:

After you commit the changes you see a number appears on push button.



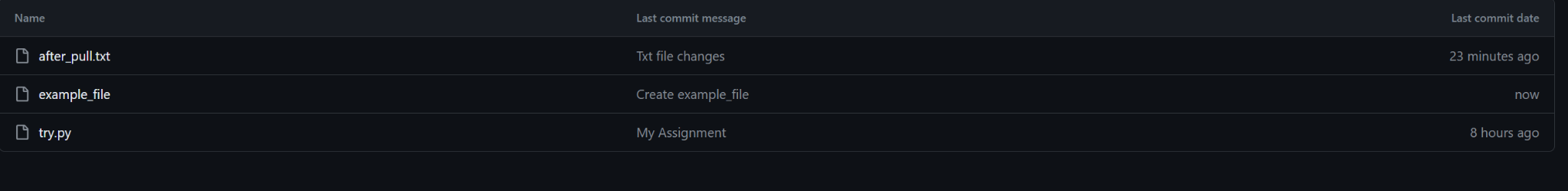
When you try to push the changes



You see this screen. You can choose which branch you want to push the changed files and simply push it.

## Git fetch with Sourcetree

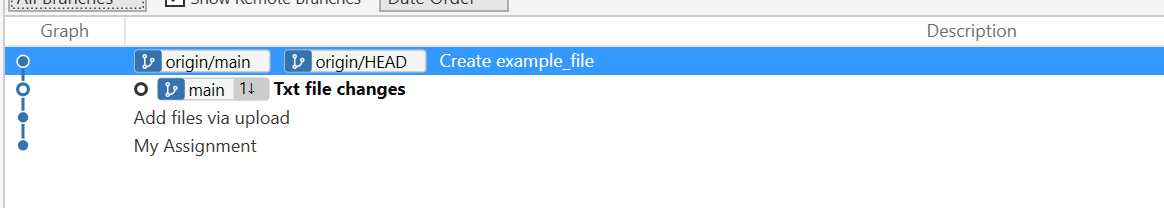
I added one more file to my git repository without using the terminal so I don’t have it locally.



Now I will try to fetch this file to my local repository.



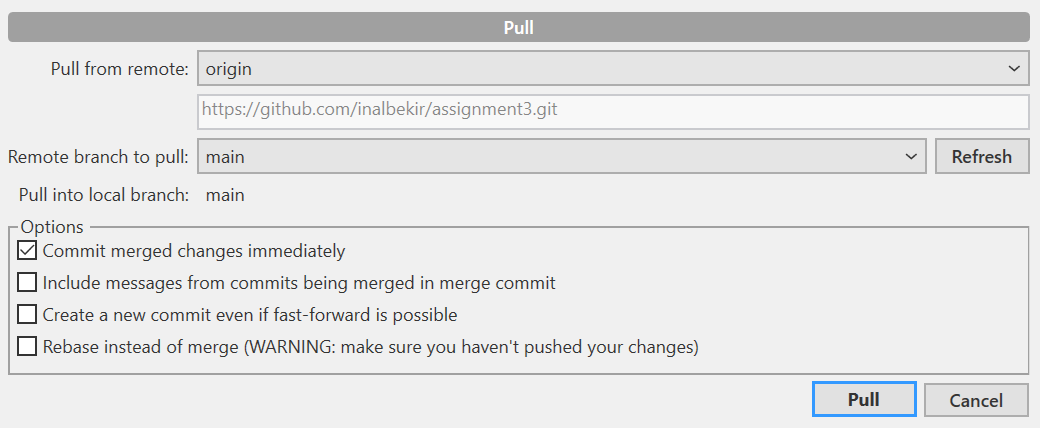
Now I can review the changes made like this



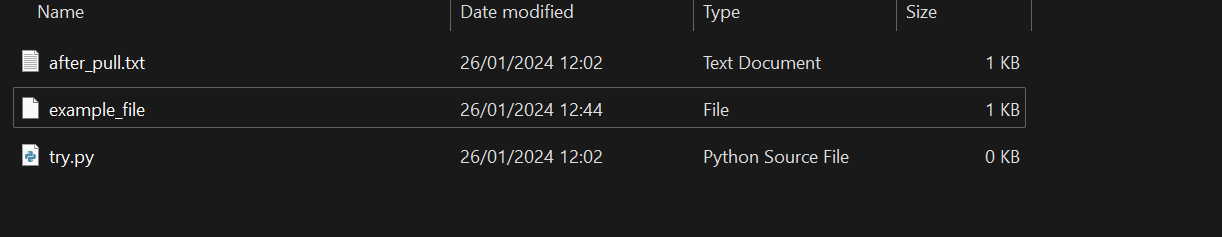
Since I can view the changes made, it's my decision whether to incorporate those files into my local repository.



If I want it, I can simply upload my repository using the pull button.

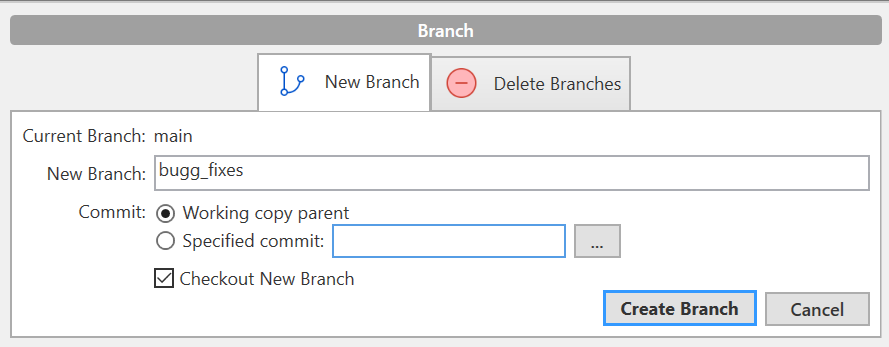


Now I can see it locally

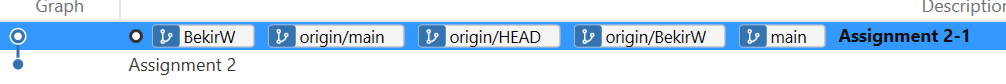


## Git branch with Sourcetree:

When I need to create a new branch, I simply click the 'branch' button and assign it a name.

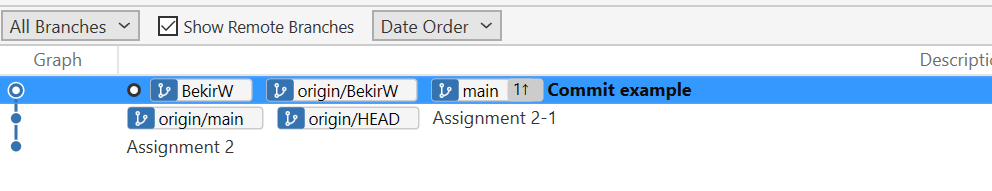


When you make a new branch, you also have to git push it so you have the same branch on local and on remote branch.



## Git merge with Sourcetree:

Whenever I want to merge the files, I first need to checkout to the main branch. Then I can simply click on the merge button and it’s done.

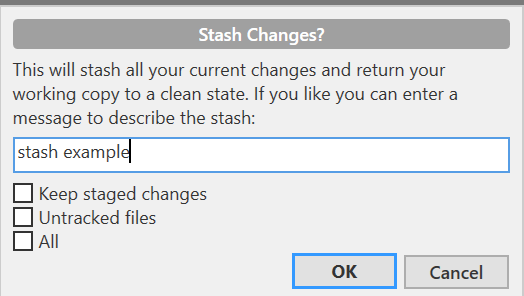


When you see the lines next to the main branch it means that you successfully merged the files.

## Git stash with Sourcetree:

I changed my file a little bit and I want to go to another branch for help and I don’t want to commit the changes because it would be unnecessary. Therefore I will use stash.





Now I can work on another branch without committing the changes on the other branch.

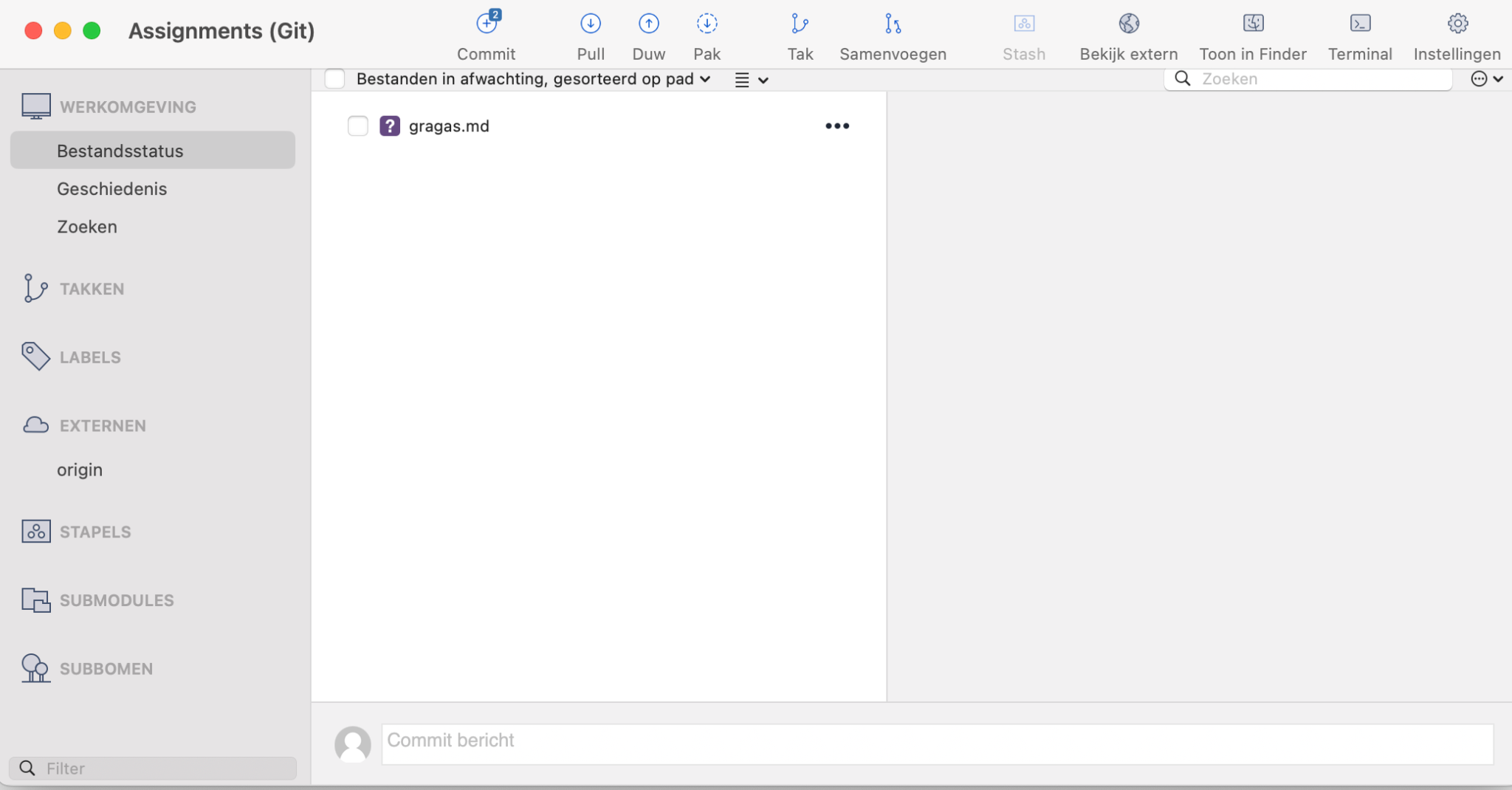
BEKIR INAL | ASSIGNMENT 4

# User Story

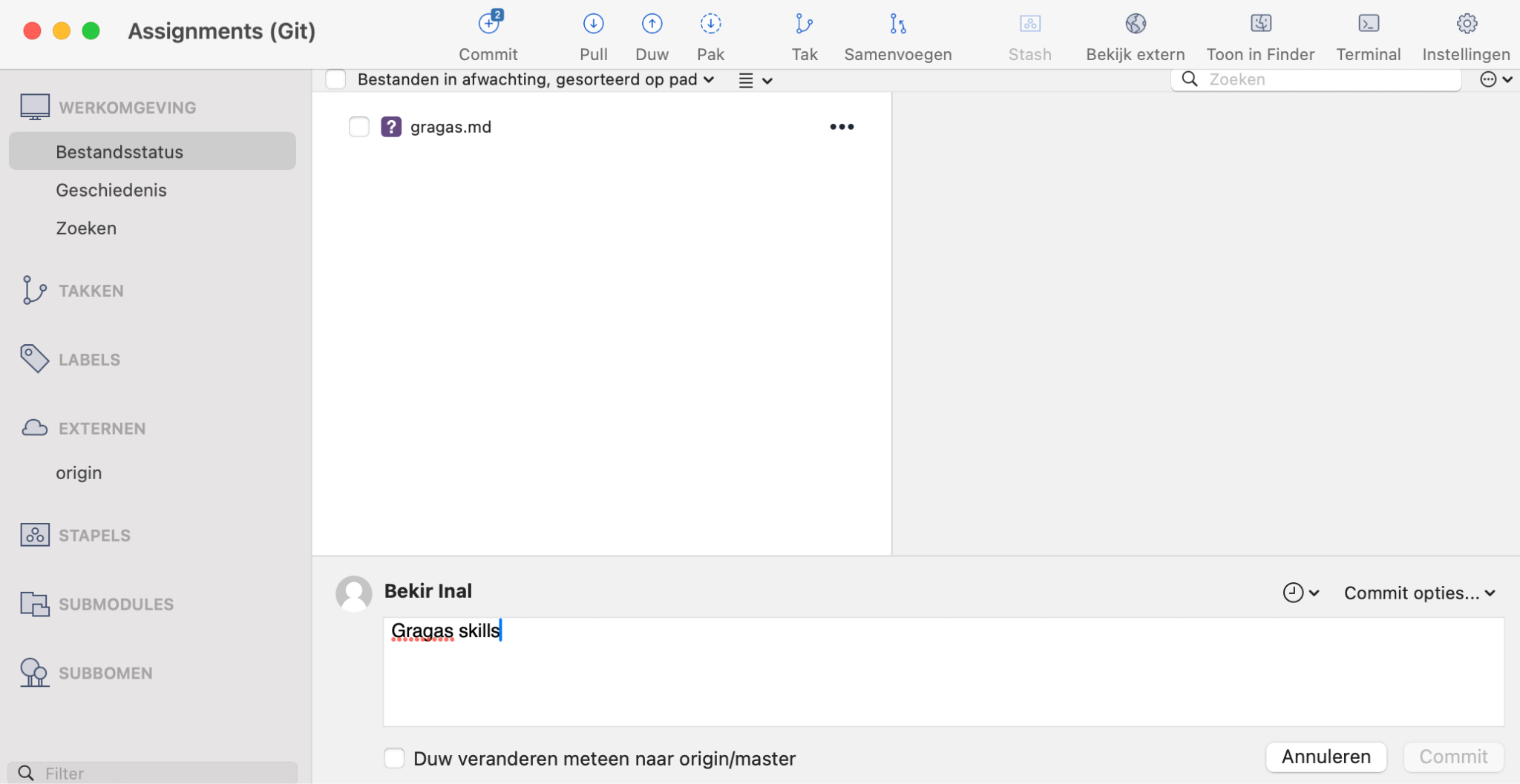
In this assignment, we will take on the roles of game developers at Riot Games, tasked with creating the skill sets for a new character in League of Legends: Gragas. But this champion needs some bug fixes. To efficiently collaborate on this project, we will be using Git and SourceTree to manage our work. This document will guide you through the process, from cloning the repository to handling conflicts during the merge.

## Git commit

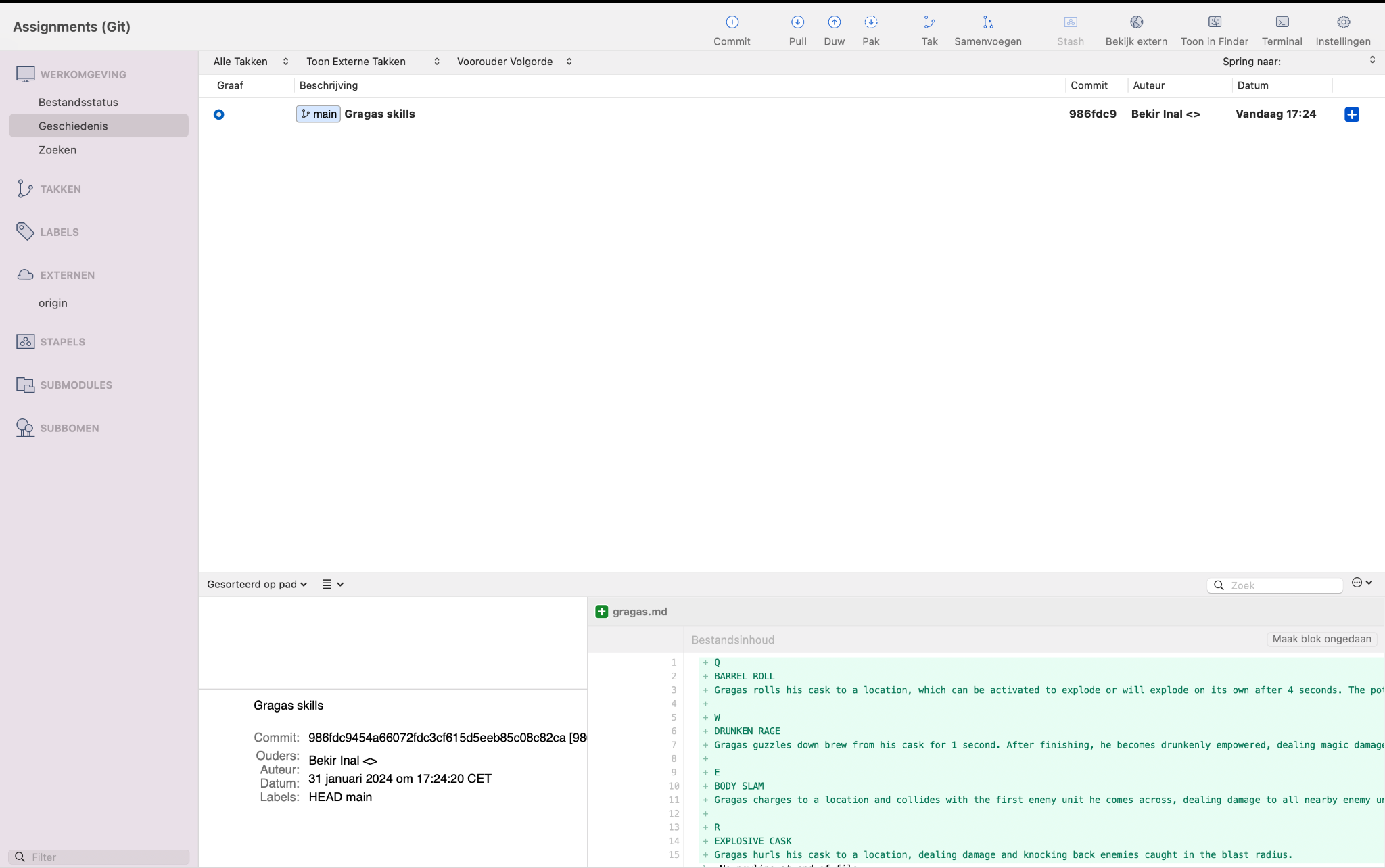
When I add my file that has information of the new champion skill, I can clearly see a number popping up on the commit button.



When I try to commit it asks me if I am sure. You can see it on the following screenshot.



Now as you can see, I can see the commit that I made.



Now the champion I made is committed to the main branch.

## Git push

I committed my file but I still don’t have it in my repository. Let’s add it to my repository as well.

Therefore I just click on the push button and I see this screen.

## 

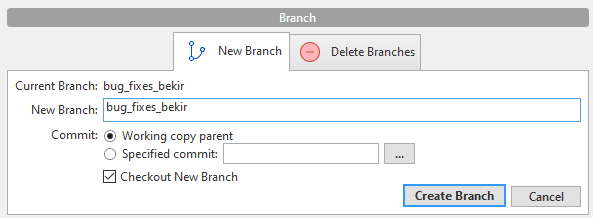
Now I can see the skills file pushed to my repository.



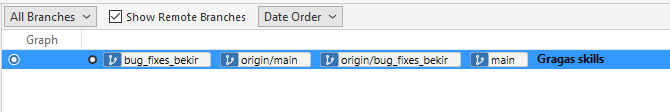
## Git branch

Gragas is coded and can be played in the game but it needs some bug fixes. I and one of my colleagues will work on this task. Therefore we both should make a new branch.

We simply click on the branch button and you can name your branch as you want. Then you simply click on the ‘Create Branch’ . Now I can work on my own branch for the bug fixes.



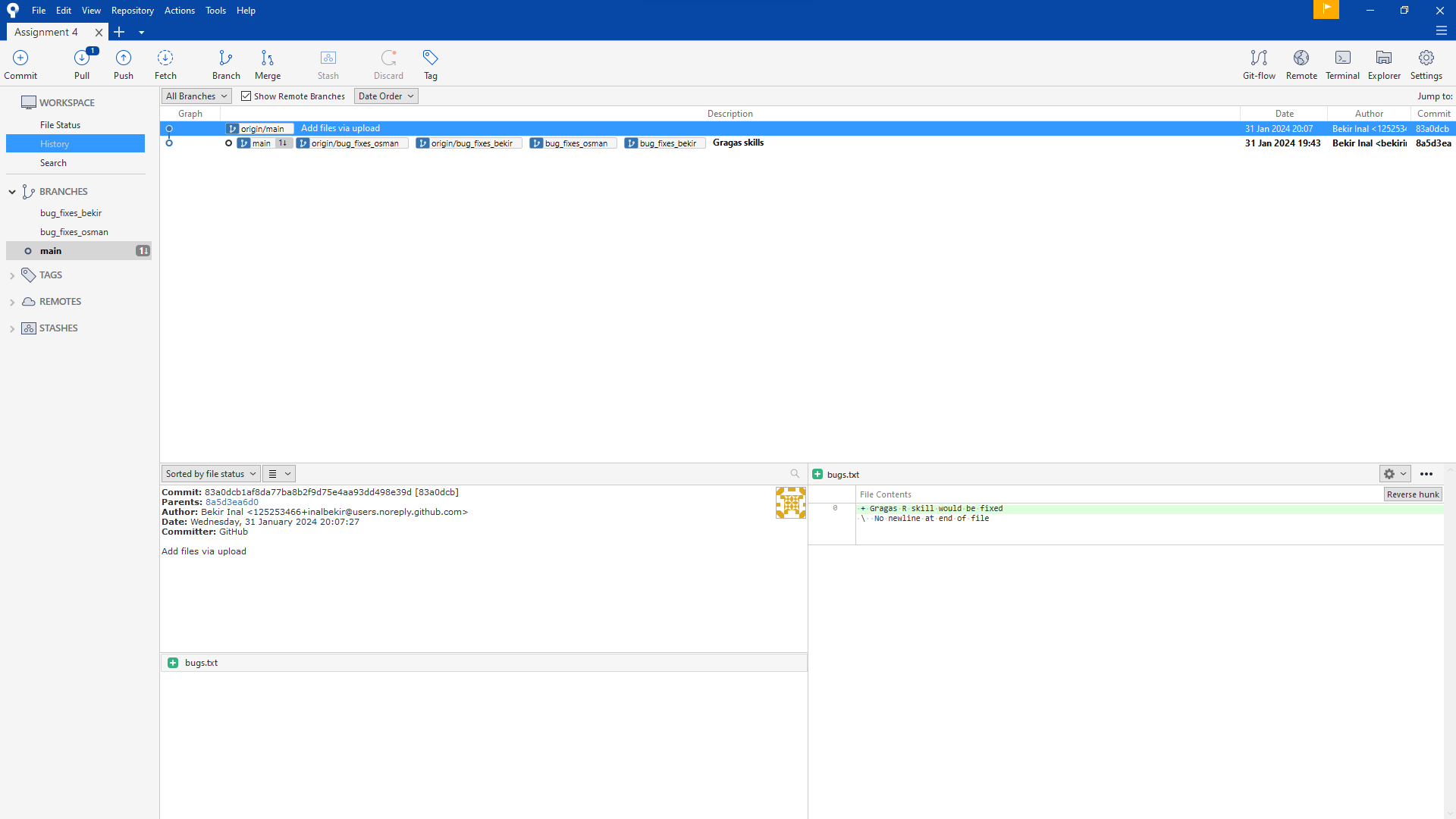
You should push your branch once more so that you have it locally and also remote.



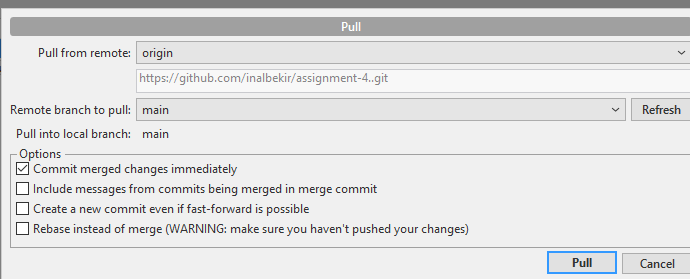
And he makes his own branch.

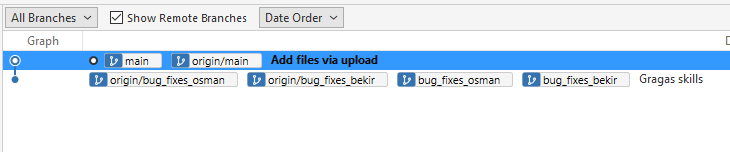
## Git fetch and pull

We don’t know what should be fixed yet, so someone should have pushed it to the repository. Let's check it.



As you can see, information about the bugs are uploaded to the repository. Now when I fetch the file I can see a file is uploaded to the repository. To have it locally I need to pull it.



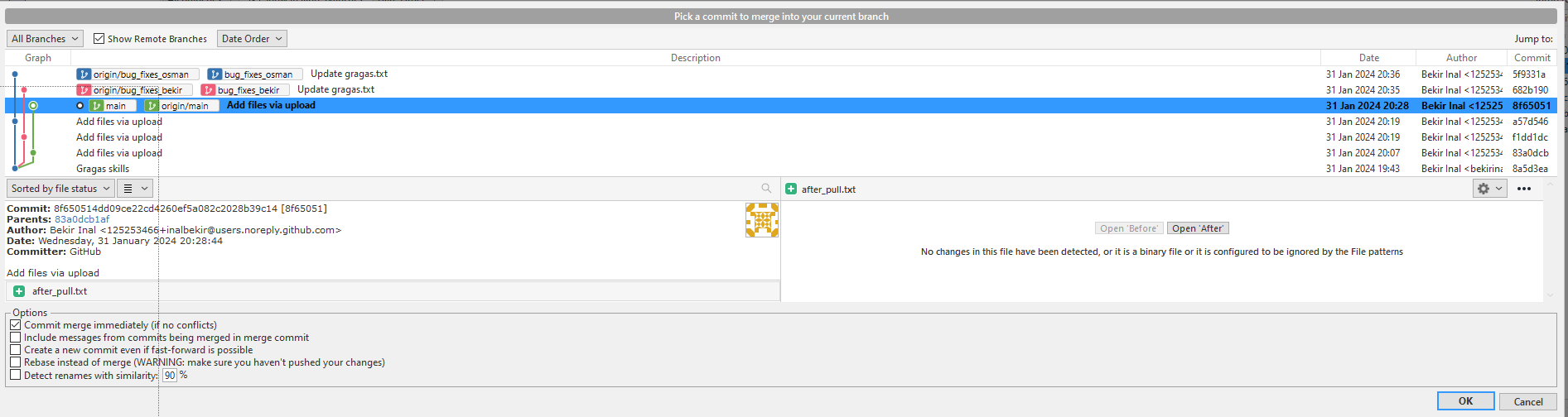


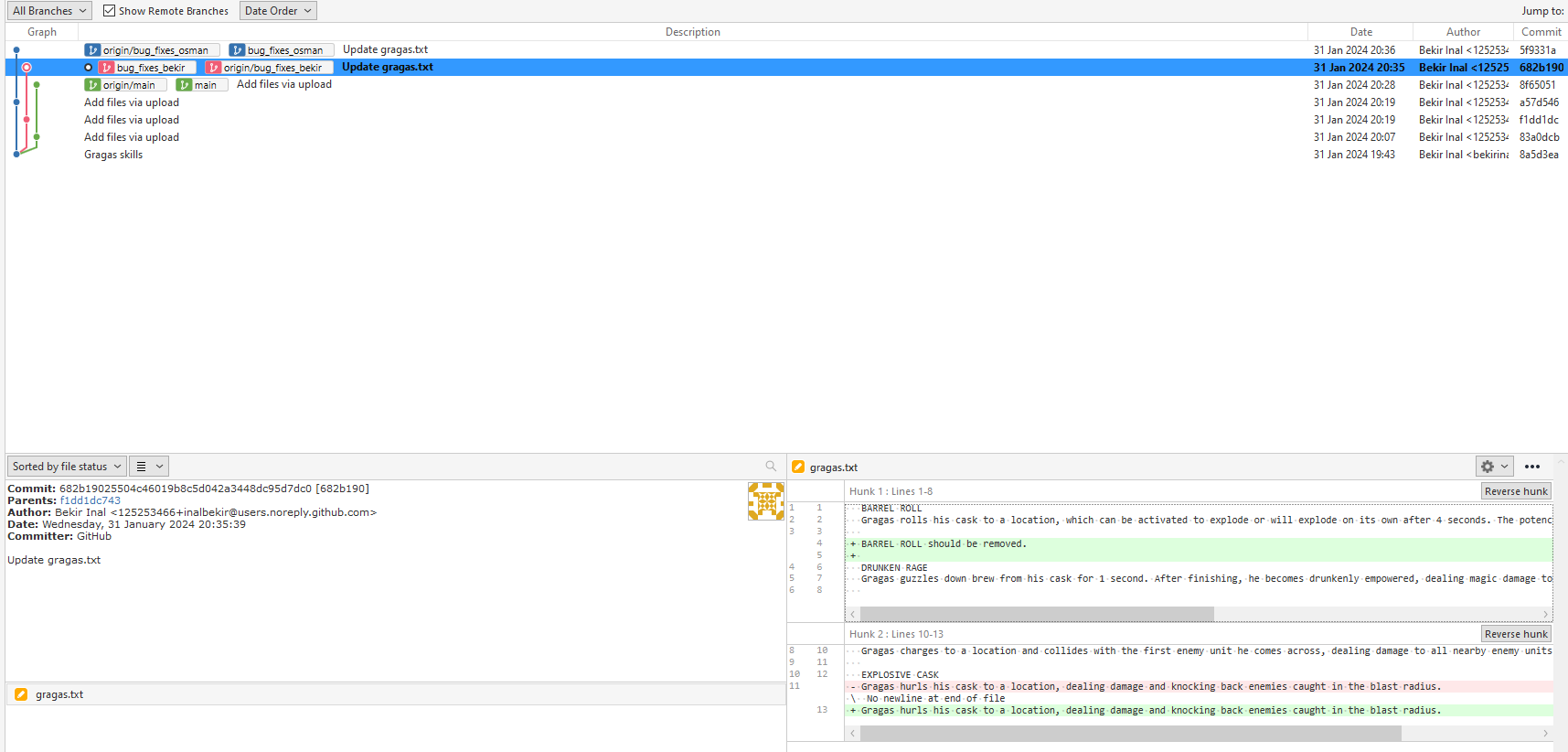
Now the files are uploaded locally and I can start working on the bug fixes.

## Git merge

Now that we are done with the bug fixes, we should push the file to the main branch from our own branches.

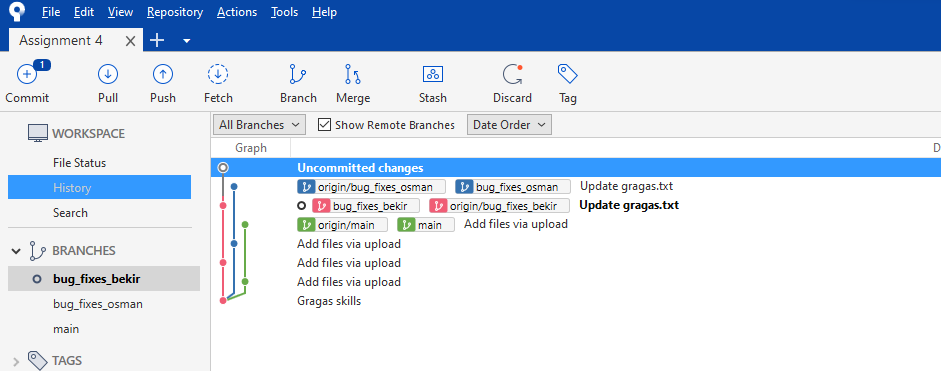
To merge the files, we simply click on the merge file. When you click on the merge file you can see this screen



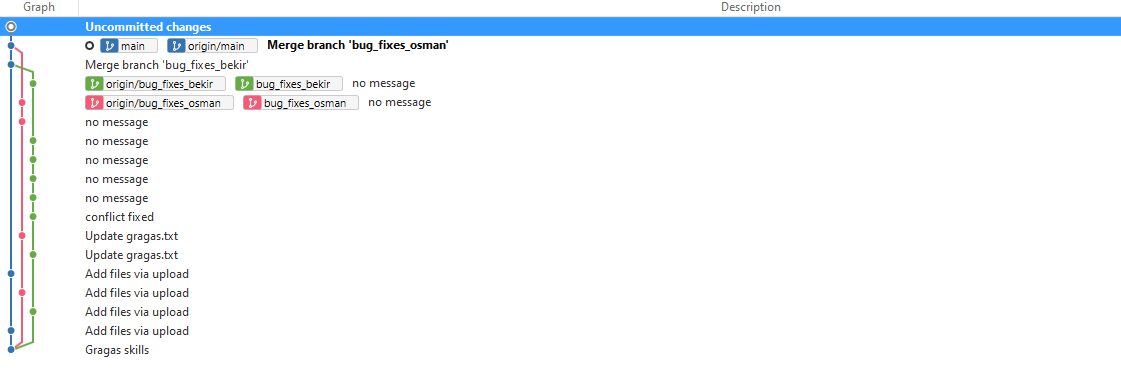


As you can see on the right corner, you can choose which one to use for the project.

When you choose one, you need to commit the one you will use for the project.



After that, you need to push it and you are done!



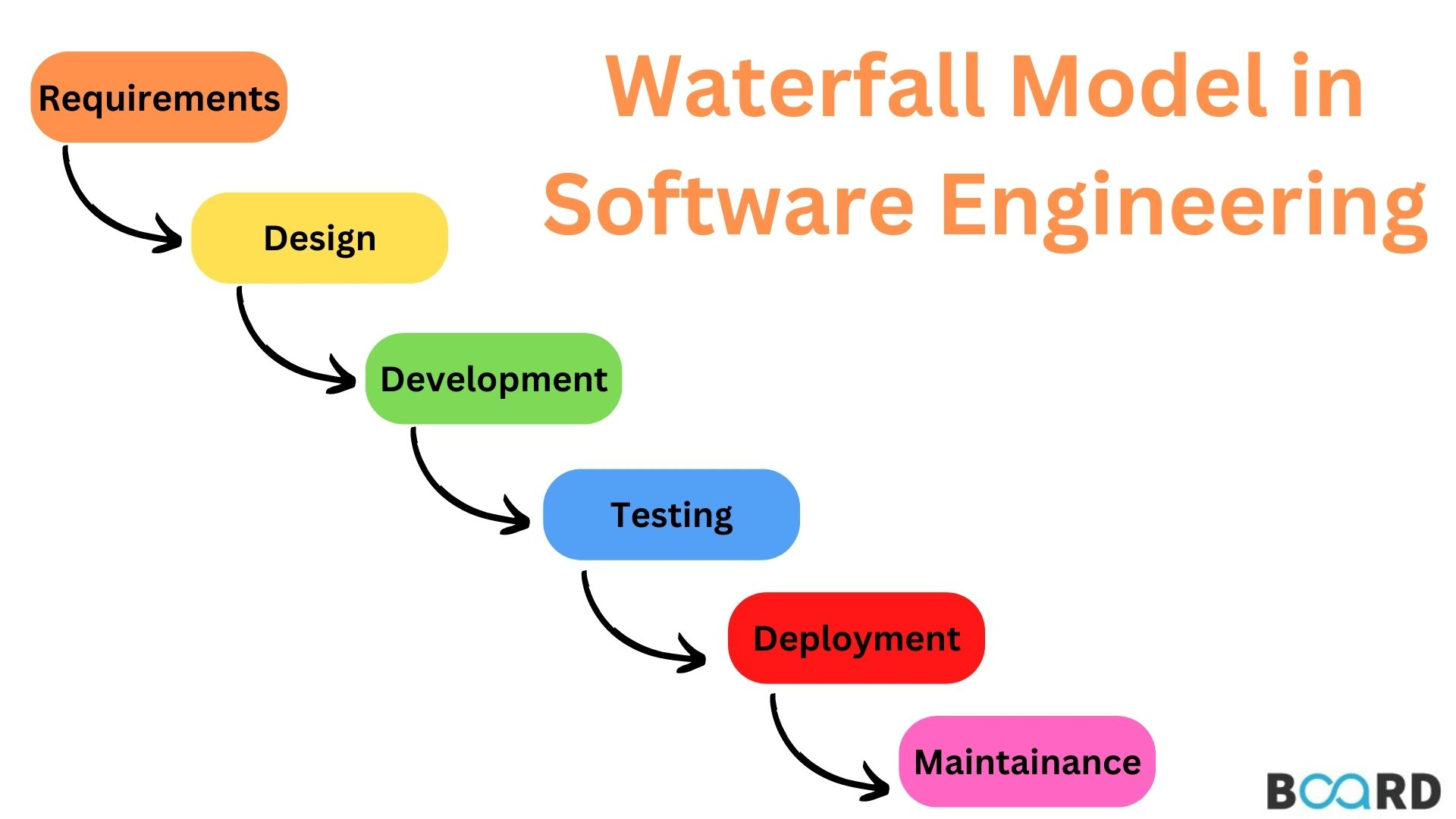
BEKIR INAL | ASSIGNMENT 5

In this assignment, I need to explain what the Agile system is and how it works. To get to know the Agile system, we need to know the old systems as well. That's why one of the systems we will take a look at is the waterfall system.

# Waterfall and Agile: Comparison

## Waterfall System

Back in the day developers used to work with the waterfall system. When you hear about the waterfall system, you can think of a banking application, insurance application or some Police department application.



### Requirements

Starts with the concept, or the idea of what the customer wants to do.

* needs the software will address
* problems the software will resolve
* functionalities the customer desires

These requirements are captured in the requirements document and it should be approved before going to the next phase.

### 

### 

### Design

Includes the logical design and the physical design.

* **Logical design** is an abstract representation of how the software data flows the inputs or the outputs. Such as how data moves through the system or data structures.
* **Physical design** determines the hardware such as storage. Programming languages are also being decided in this phase.

Once you’re done with both of these steps and it’s approved, you can go to the next step.

### Implementation

To move to this phase, the design phase should be approved. Implementation phase is building the design into actual software.

Often, software is built in units and integrated into a whole at a later point like a puzzle. As one software unit implementation is complete, another software unit implementation begins. Once you’re done with all the units and it’s approved you can move to the next phase.

### Verification

Testing the software against the requirements. During the verification phase, the project team may select a pilot group of users to test the software’s user friendliness or to provide user feedback.

If the software does not meet the requirements in the requirements documents, It is sent back to the software engineers for further implementation. Once the project verification part is done and the phase is deemed complete, the project moves to the next phase.

### Deployment

The actual release of the new software into the information technology environment. As with the implementation phase this phase may also be broken into units to allow for ongoing deployment and maintenance.



As you can see on the picture above, once you start to deploy the project as units, you start the maintenance phase for each unit. Once A location is deployed, it is going through the maintenance phase while you’re deploying the B location.

### Maintenance

This is the phase of the project in which initial issues are resolved. The customer provides feedback on the new software for the project team to look into.

Additionally the project team made two additional testing in the live environment to verify acceptable deployment. One during deployment and one after the deployment with the customer’s feedback.

## Conclusion

The waterfall system is a good system for managers to use, but it is not suitable for all projects. It has clear advantages, however it does have disadvantages as well.

### Advantages

* Easy to understand
* Easy to manage
* Fewer production issues
* Better budget management

### Disadvantages

* Not flexible
* Doesn’t handle unexpected risks well
* Not a good for complex or long-term project
* Difficult to capture all requirements up front, so if you missed something you should go through all the phases once more.

# What is Agile?

Agile methodology was developed as a response to the limitations of traditional project management (waterfall) approaches in adapting to rapid changes and complex project requirements. It prioritizes flexibility, customer collaboration, and rapid iterations, allowing businesses to respond quickly to market changes and customer needs. Agile's widespread adoption across industries is attributed to its proven effectiveness in improving project outcomes, enhancing team productivity, and ensuring continuous delivery of value to customers. This methodology aligns with modern business practices by emphasizing lean principles, innovation, and a customer-centric approach to product development.

# 

# Why is Agile the most used methodology?

The Agile system offers several significant advantages, making it highly beneficial for companies, especially in dynamic and complex project environments. Some of the best positive sides of the Agile system include:

**Flexibility and Adaptability:**

Agile allows teams to respond quickly to changes in customer needs or market conditions. Its iterative nature enables adjustments to be made during the project lifecycle, rather than at the end, which can significantly reduce costs and time to market.

**Enhanced Customer Satisfaction:**

By involving customers throughout the development process and prioritizing their feedback, Agile ensures that the final product is more closely aligned with customer expectations, thereby increasing satisfaction and the likelihood of project success.

**Improved Product Quality:**

Continuous testing, integration, and feedback within Agile cycles ensure early detection and correction of defects, leading to a higher quality product. Regular reviews and adaptations also mean that quality is a continuous focus throughout the project.

**Increased Project Control:**

Agile provides stakeholders with better visibility into project progress and potential issues through regular meetings (e.g., daily stand-ups, sprint reviews) and the use of visual management tools. This transparency allows for more informed decision-making and adjustments.

**Higher Team Morale and Productivity:**

Agile promotes a collaborative, self-organizing team environment that empowers team members and encourages active participation in the project management process. This empowerment boosts team morale and can lead to increased productivity and innovation.

These advantages highlight why Agile has become a preferred methodology for managing projects across various industries, not just in software development. Its focus on customer value, adaptability, and team collaboration aligns well with the needs of modern businesses facing rapid changes and complex challenges.

# Agile system

# 

There are a few important things in the logic underlying the Agile system.

* Developing a project incrementally allows for the sequential launch of distinct components, such as the front-end interface, shopping cart, payment systems, or email services, in an online retail platform. This methodology facilitates immediate feedback from customers and stakeholders on each discrete segment, enabling rapid modifications and enhancements.
* Moreover, this approach minimizes the risk of system-wide failures triggered by minor coding errors. In the event of a malfunction, the issue can be swiftly rectified without disrupting the overall project continuity, thus ensuring a more resilient and adaptable development process.

# Why do we need Agile?

Scrum is an [agile project management](https://www.atlassian.com/agile/project-management) framework that helps teams structure and manage their work through a set of values, principles, and practices. Much like a rugby team (where it gets its name) training for the big game, scrum encourages teams to learn through experiences, self-organize while working on a problem, and reflect on their wins and losses to continuously improve.

While the scrum I’m talking about is most frequently used by software development teams, its principles and lessons can be applied to all kinds of teamwork. This is one of the reasons scrum is so popular. Often thought of as an agile project management framework, scrum describes a set of meetings, tools, and roles that work in concert to help teams structure and manage their work.

In this article, we’ll discuss how a traditional scrum framework is comprised with the help of the [Scrum Guide](https://www.scrumguides.org/) and David West, CEO of Scrum.org. We’ll also include examples of how we see our customers stray from these fundamentals to fit their specific needs. For that, our own Megan Cook, Group Product Manager for Jira Software and former agile coach, will give tips and tricks in our Agile Coach video series:

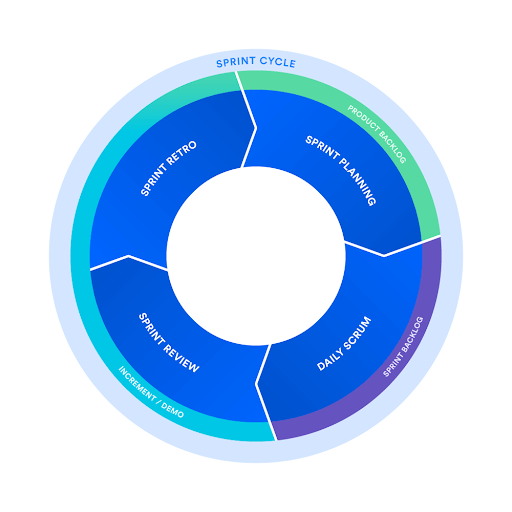
## Agile vs. scrum

People often think scrum and agile are the same thing because scrum is centered around [continuous improvement](https://www.atlassian.com/agile/project-management/continuous-improvement), which is a core principle of agile. However, scrum is a framework for getting work done, whereas [agile](https://www.atlassian.com/agile) is a philosophy. The agile philosophy centers around continuous incremental improvement through small and frequent releases. You can’t really “go agile”, as it takes dedication from the whole team to change the way they think about delivering value to your customers. But you can use a framework like scrum to help you start thinking that way and to practice building agile principles into your everyday communication and work.

The difference between agile and the definition of scrum can be found in the Scrum guide and the [Agile manifesto](https://www.atlassian.com/agile/manifesto). The Agile manifesto outlines four values:

* Individuals and interactions over processes and tools
* Working software over comprehensive documentation
* Customer collaboration over contract negotiation
* Responding to change over following a plan

The definition of scrum is based on empiricism and lean thinking. Empiricism says that knowledge comes from experience and that decisions are made based on what is observed. Lean thinking reduces waste and focuses on essentials. The scrum framework is heuristic; it’s based on continuous learning and adjustment to fluctuating factors. It acknowledges that the team doesn’t know everything at the start of a project and will evolve through experience. Scrum is structured to help teams naturally adapt to changing conditions and user requirements, with re-prioritization built into the process and short release cycles so your team can constantly learn and improve.



While scrum is structured, it is not entirely rigid. Its execution can be tailored to the needs of any organization. There are many theories about how exactly scrum teams must work in order to be successful. However, after more than a decade of helping agile teams get work done at Atlassian, we’ve learned that clear communication, transparency, and a dedication to continuous improvement should always remain at the center of whatever framework you choose. And the rest is up to you.

## 

## 

### The scrum framework

The scrum framework outlines a set of values, principles, and practices that scrum teams follow to deliver a product or service. It details the members of a scrum team and their accountabilities, “artifacts” that define the product and work to create the product, and scrum ceremonies that guide the scrum team through work.

### Members of a scrum team

A scrum team is a small and nimble team dedicated to delivering committed product increments. A scrum team’s size is typically small, at around 10 people, but it’s large enough to complete a substantial amount of work within a sprint. A scrum team needs three specific roles: product owner, [scrum master](https://www.atlassian.com/agile/scrum/scrum-master), and the development team. And because scrum teams are cross-functional, the development team includes testers, designers, UX specialists, and ops engineers in addition to developers. 

[Scrum Roles Explained - Agile Coach (2018)](https://www.youtube.com/watch?v=fIY-yi4ckkE&ab_channel=Atlassian)

### 

### 

### 

### The scrum product owner

Product owners are the champions for their product. They are focused on understanding business, customer, and market requirements, then prioritizing the work to be done by the engineering team accordingly. Effective product owners:

* Build and manage the product backlog.
* Closely partner with the business and the team to ensure everyone understands the work items in the product backlog.
* Give the team clear guidance on which features to deliver next.
* Decide when to ship the product with a predisposition towards more frequent delivery.

The product owner is not always the [product manager](https://www.atlassian.com/agile/product-management/product-manager). Product owners focus on ensuring the development team delivers the most value to the business. Also, it's important that the product owner be an individual. No development team wants mixed guidance from multiple product owners.

### The scrum master

Scrum masters are the champions of scrum within their teams. They coach teams, product owners, and the business on the scrum process, and look for ways to fine-tune their practice of it.

An effective scrum master deeply understands the work being done by the team and can help the team optimize their transparency and delivery flow. As the facilitator-in-chief, he/she schedules the needed resources (both human and logistical) for sprint planning, stand-up, sprint review, and the sprint retrospective.

### The scrum development team

Scrum teams get all the work done. They are the champions for sustainable development practices. The most effective scrum teams are tight-knit, co-located, and usually five to seven members. One way to work out the team size is to use the famous ‘two pizza rule’ coined by Jeff Bezos, the CEO of Amazon (the team should be small enough to share two pizzas).

Team members have differing skill sets, and cross-train each other so no one person becomes a bottleneck in the delivery of work. Strong scrum teams are self-organizing and approach their projects with a clear ‘we’ attitude. All members of the team help one another to ensure a successful sprint completion.

The scrum team drives the plan for each sprint. They forecast how much work they believe they can complete over the iteration using their historical velocity as a guide. Keeping the iteration length fixed gives the development team important feedback on their estimation and delivery process, which in turn makes their forecasts increasingly accurate over time.

### Scrum artifacts

[Scrum artifacts](https://www.atlassian.com/agile/scrum/artifacts) are important information used by the scrum team that helps define the product and what work to be done to create the product. There are three artifacts in scrum: product backlog, a sprint backlog, and an increment with your definition of “done”. They are the three constants a scrum team should reflect on during [sprints](https://www.atlassian.com/agile/scrum/sprints) and over time.

[Scrum Artifacts - Agile Coach (2018)](https://www.youtube.com/watch?v=BkiAMlUF3YI&ab_channel=Atlassian)

* Product Backlog is the primary list of work that needs to get done and maintained by the product owner or product manager. This is a dynamic list of features, requirements, enhancements, and fixes that acts as the input for the sprint backlog. It is, essentially, the team’s “To Do” list. The product backlog is constantly revisited, re-prioritized and maintained by the Product Owner because, as we learn more or as the market changes, items may no longer be relevant or problems may get solved in other ways.
* Sprint Backlog is the list of items, [user stories](https://www.atlassian.com/agile/project-management/user-stories), or bug fixes, selected by the development team for implementation in the current sprint cycle. Before each sprint, in the sprint planning meeting (which we’ll discuss later in the article) the team chooses which items it will work on for the sprint from the product backlog. A sprint backlog may be flexible and can evolve during a sprint. However, the fundamental sprint goal – what the team wants to achieve from the current sprint – cannot be compromised.
* Increment (or Sprint Goal) is the usable end-product from a sprint. At Atlassian, we usually demonstrate the “increment” during the end-of-sprint demo, where the team shows what was completed in the sprint. You may not hear the word “increment” out in the world, as it’s often referred to as the team’s definition of “Done”, a milestone, the sprint goal, or even a full version or a [shipped epic](https://www.atlassian.com/agile/project-management/epics). It just depends on how your teams defines “Done” and how you define your sprint goals. For example, some teams choose to release something to their customers at the end of every sprint. So their definition of ‘done’ would be ‘shipped’. However, this may not be realistic of other types of teams. Say you work on a server-based product that can only ship to your customers every quarter. You may still choose to work in 2-week sprints, but your definition of ‘done’ may be finishing part of a larger version that you plan to ship together. But of course, the longer it takes to release software, the higher the risk that software will miss the mark.

As you can tell, there are lots of variations, even within artifacts, that your team can choose to define. That’s why it’s important to remain open to evolving how you maintain even your artifacts. Perhaps your definition of ‘done’ provides undo stress on your team, and you need to go back and pick a new definition.

### 

### 

### 

### Scrum ceremonies or events

The scrum framework includes scrum practices, ceremonies, and meetings that scrum teams perform on a regular basis. The [agile ceremonies](https://www.atlassian.com/agile/scrum/ceremonies) are where we see the most variations for teams. For example, some teams find doing all of these ceremonies cumbersome and repetitive, while others use them as a necessary check-in. Our advice is to start out using all of the ceremonies for two sprints and see how it feels. You can then perform a quick [retro](https://www.atlassian.com/agile/scrum/retrospectives) and see where you might need to adjust.

Below is a list of all the key ceremonies a scrum team might partake in:

* **Organize the backlog:** Sometimes known as backlog grooming, this event is the responsibility of the product owner. The product owner’s main jobs are to drive the product towards its product vision and have a constant pulse on the market and the customer. Therefore, he/she maintains this list using feedback from users and the development team to help prioritize and keep the list clean and ready to be worked on at any given time. You can read more about [maintaining a healthy backlog here](https://www.atlassian.com/agile/scrum/backlogs).
* **Sprint planning:** The work to be performed (scope) during the current [sprint](https://confluence.atlassian.com/agile/glossary/sprint) is planned during this meeting by the entire development team. This meeting is led by the scrum master and is where the team decides on the sprint goal. Specific user stories are then added to the sprint from the product backlog. These stories always align with the goal and are also agreed upon by the scrum team to be feasible to implement during the sprint.  
    
  At the end of the planning meeting, every scrum member needs to be clear on what can be delivered in the sprint and how the increment can be delivered.
* **Sprint:** A sprint is the actual time period when the scrum team works together to finish an increment. Two weeks is a pretty typical length for a sprint, though some teams find a week to be easier to scope or a month to be easier to deliver a valuable increment. Dave West, from Scrum.org advises that the more complex the work and the more unknowns, the shorter the sprint should be. But it’s really up to your team, and you shouldn’t be afraid to change it if it’s not working! During this period, the scope can be re-negotiated between the product owner and the development team if necessary. This forms the crux of the empirical nature of scrum.  
    
  All the events from planning to retrospective happen during the sprint. Once a certain time interval for a sprint is established, it has to remain consistent throughout the development period. This helps the team learn from past experiences and apply that insight to future sprints.
* Daily scrum or stand up: This is a daily super-short meeting that happens at the same time (usually mornings) and a place to keep it simple. Many teams try to complete the meeting in 15 minutes, but that’s just a guideline. This meeting is also called a ‘daily stand-up’ emphasizing that it needs to be a quick one. The goal of the daily scrum is for everyone on the team to be on the same page, aligned with the sprint goal, and to get a plan out for the next 24 hours.  
    
  The stand up is the time to voice any concerns you have with meeting the sprint goal or any blockers.  
    
  A common way to conduct a stand up is for every team member to answer three questions in the context of achieving the sprint goal:  
    
  • What did I do yesterday?  
  • What do I plan to do today?  
  • Are there any obstacles?  
    
  However, we’ve seen the meeting quickly turn into people reading from their calendars from yesterday and for the next day. The theory behind the stand up is that it keeps distracting chatter to a daily meeting, so the team can focus on the work for the rest of the day. So if it turns into a daily calendar read-out, don’t be afraid to change it up and get creative.
* **Sprint review:** At the end of the sprint, the team gets together for an informal session to view a demo of, or inspect, the increment. The development team showcases the backlog items that are now ‘Done’ to stakeholders and teammates for feedback. The product owner can decide whether or not to release the increment, although in most cases the increment is released.  
    
  This review meeting is also when the product owner reworks the product backlog based on the current sprint, which can feed into the next sprint planning session. For a one-month sprint, consider time-boxing your [sprint review](https://www.atlassian.com/agile/scrum/sprint-reviews) to a maximum of four hours.
* **Sprint retrospective:** The [retrospective](https://www.atlassian.com/agile/scrum/retrospectives) is where the team comes together to document and discuss what worked and what didn’t work in a sprint, a project, people or relationships, tools, or even for certain ceremonies. The idea is to create a place where the team can focus on what went well and what needs to be improved for the next time, and less about what went wrong.