

Session 2 - Introduction to Version Control and basics of Git

Time estimated: 3.5 hours

Kiril Vasilev and Riccardo Taormina

September 19, 2023

Contents

1	Introduction	3
1.1	What is Version Control?	3
1.2	What is Git?	4
1.3	What is GitLab?	4
2	Main concepts and terminology	4
3	Basics of Git	5
3.1	Installing Git	5
3.2	Setting up Git	5
3.3	Creating a repository (from scratch)	6
3.4	Your first commit	7
3.5	Comparing commits – git log and git diff	10
4	Branching and merging	12
4.1	What are branches	12
4.2	Creating a branch	13
4.3	Merging branches	15
4.4	Deleting branches	19
4.5	Creating a branch from an arbitrary commit (Optional)	19
5	Merge conflicts	21
6	Tracking changes	25
6.1	Git log	25
6.2	Git checkout – recover old versions of a file (Optional)	27
6.3	Git checkout – cancelling staged and unstaged changes (Optional)	27
6.4	Git blame (Optional)	28
7	Reverting changes	29
8	Resetting changes/branches	31
9	Stashing changes (Optional)	32
10	GitIgnore	34
11	Interactive commits (Optional)	36
12	Common mistakes using Git	40

13 Interacting with a remote repository	40
13.1 Creating a remote repository	40
13.2 Setting up SSH agent	44
13.3 Pushing to a repository	47
13.4 Force pushing changes (Dangerous)	51
13.5 Fetching/Pulling from a repository	52
13.6 Cloning existing repository	53
14 GitLab videos	56
15 Aliasing commands (Optional)	56
16 Conclusion	57
17 References and used resources	58
18 Appendices	58
18.1 Appendix A: Commonly used Git commands	59
18.2 Appendix B: Commonly used bash commands	60

1 Introduction

Welcome to our introductory tutorial on version control systems, Git and GitLab. We will start with teaching you how to use Git via command line interface and next we will cover the interaction with GitLab. In our Appendices, you can find an overview of all commonly used commands in this tutorial. We wish you good luck and hope that you enjoy learning about Git!

Note: some of the sections in this document have “Optional” next to them. These sections cover bonus material, which is very interesting to go over, but it is not mandatory.

We strongly recommend that you complete the steps in this tutorial as well to familiarise yourself how this tool works instead of simply reading over the document. In addition, try to type the commands instead of copy-pasting them, so that you could memorise them and be more efficient in your work later.

1.1 What is Version Control?

While working on personal or university projects, without a doubt you have come across the following situation: You have finished drafting a report and believe you are done with it and save the file as “report.doc”. However, later you decide to experiment and make some changes, but you still want to keep your old working version, so you make a new file called “report-final.doc”. Now imagine that you send the file to a friend of yours to proofread and make some comments on it and they send it back. Next, you incorporate their feedback and end up naming the new one “report-final-2.doc”.

What you have been doing is called version control. Version control systems start with a base version of the document and then record changes you make each step of the way. You can think of it as a recording of your progress: you can rewind to start at the base document and play back each change you made, eventually arriving at your most recent version, as shown in figure 1.1:



Figure 1.1: Consecutive file changes

Once you think of changes as separate from the document itself, you can then think about “playing back” different sets of changes on the base document, resulting in different versions of that document. For example, two users can make independent sets of changes on the same document, resulting in 2 independent versions (figure 1.2).

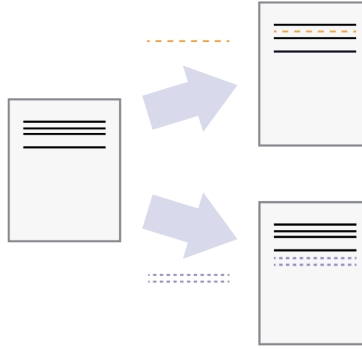


Figure 1.2: Parallel file changes

Unless multiple users make changes to the same section of the document - a conflict - you can incorporate two sets of changes into the same base document (figure 1.3).

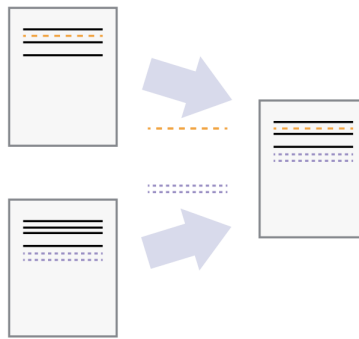


Figure 1.3: Merge of changes

1.2 What is Git?

Git is a version control system (VCS), used by software developers (and not only!) to work on projects in parallel together. It provides multiple benefits such as tracking changes to files, working side by side with other people, and the ability to rollback to previous versions of files without losing track of newer changes.

1.3 What is GitLab?

GitLab is a cloud-based version control system built around Git. It provides a lot more features such as Issues, Merge Requests, CI/CD pipelines, etc.

2 Main concepts and terminology

Here we present a list of the terminology we will use while going over the tutorial. Do not panic if you do not understand what each of the following means. Later, we will provide a more elaborate explanation with examples. Bear in mind that the list below is not exhaustive, and more terms may show up.

1. Repository – Storage, where VCS store their history of changes and information about who made them.
2. Remote (of repository) – a version control repository stored somewhere else and the changes between the two are usually synchronized. We will refer to the Gitlab repository as a *remote*.
3. Commit – Snapshot of the current state of the project. If a commit contains changes to multiple files, all the changes are recorded together.

4. Staging – preparation of files to be committed. During the staging we propose files to be committed.
5. Branch – development (time) line. The main development line is called "main" (previously it was called "master" on git).
6. Cloning – copying (downloading) an existing project on your laptop. Usually, it is done only during the first time of getting the remote repository.
7. Pushing – uploading new commits (changes) to the remote server.
8. Pulling – retrieving new commits from the remote repository.
9. Fetching – check for new changes on the remote repository without pulling them yet.
10. Conflict – when changes made by multiple users to the same file are incompatible, you can get into a conflict. Helping users resolve those conflicts is one of the key advantages of VCS.
11. Tracked (files) – files that Git knows about – they are either in the staging area or were previously added to the repository.
12. Untracked (files) – files that Git does not know about – they are likely new files that have not been staged yet.
13. Snapshot – copy of the current version of the entire repository.

3 Basics of Git

3.1 Installing Git

The steps to install Git and Git bash are shown here <https://kirilvasilev16.github.io/mude-website/>.

3.2 Setting up Git

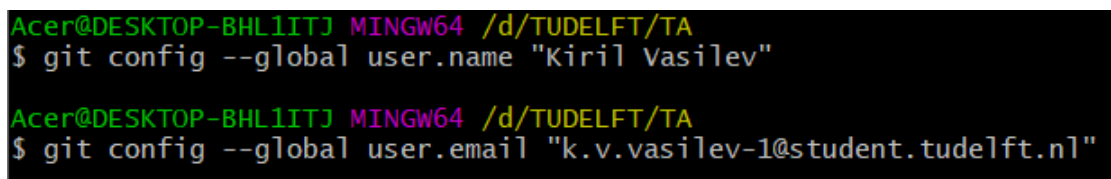
Git commands have the structure of `git <verb> <options>`, where `<verb>` denotes what action we want to take and `<options>` are the arguments, which we pass to the command.

Before we start, we first need to locally set up the git environment, so that when we make changes, we can be identified as the author of those commits. Open git-bash on your computer and type the following lines in the command line interface and use your name and email address between the quotation marks.

Since you will be uploading your progress on a university instance of Gitlab, it is recommended to use your student email when committing.

```
git config --global user.name "Kiril Vasilev"
git config --global user.email "k.v.vasilev-1@student.tudelft.nl"
```

Console example in figure 3.4:



```
Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/TA
$ git config --global user.name "Kiril Vasilev"

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/TA
$ git config --global user.email "k.v.vasilev-1@student.tudelft.nl"
```

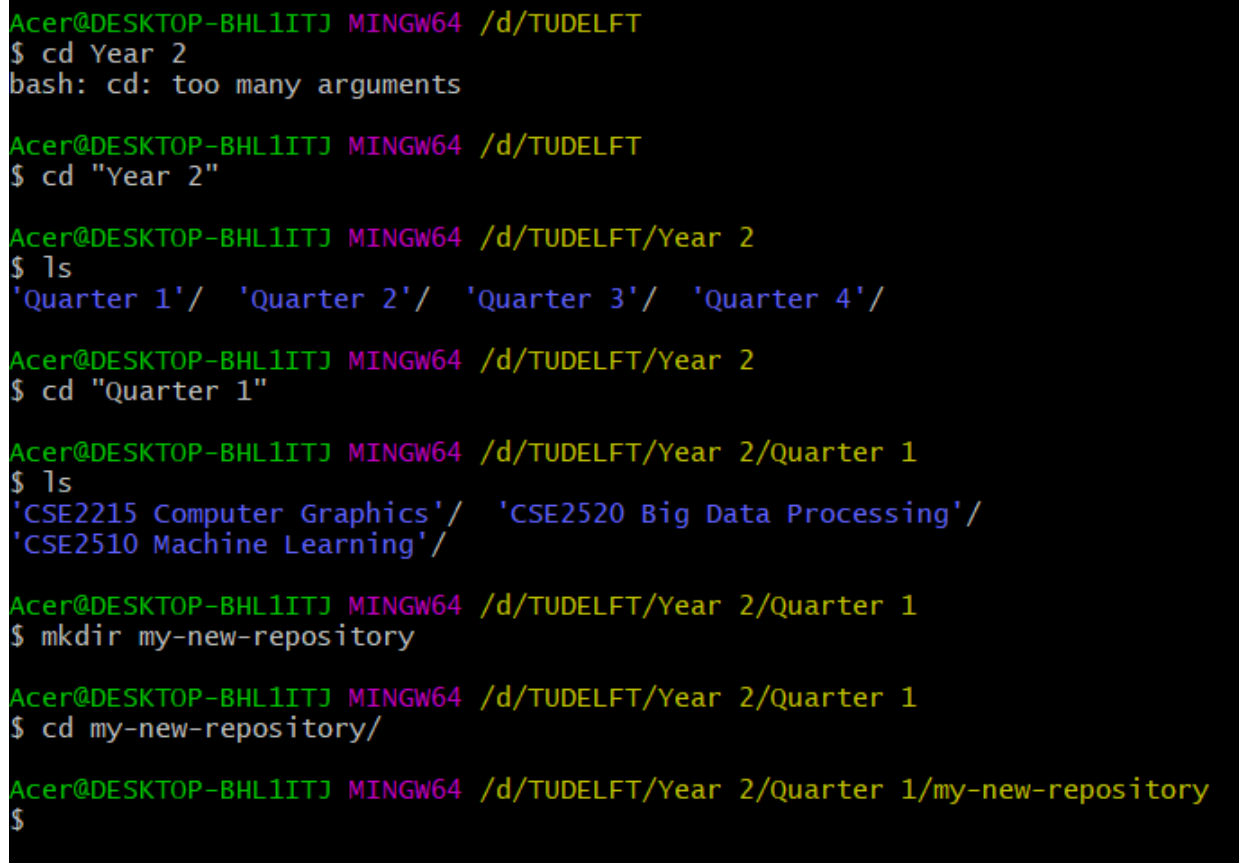
Figure 3.4: Setting up name and email to use for git

Note that in the commands above you **do not need** to type the \$ sign.

3.3 Creating a repository (from scratch)

Open git bash and go to the desired directory, where you wish to set up your repository. You can use the commands `cd` to move through directories and `ls` to list files in the current directory. Have a look at figure 3.5, where we move to the location where we want to set up our new repository. Namely, we wish to set up our repository in a new folder called `my-new-repository`, which is inside the folders `TUDELFT/Year 2/Quarter 1`.

Note that it is not necessary to create a new folder before creating a new repository. You can make a repository in an existing folder as well.



```
Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT
$ cd Year 2
bash: cd: too many arguments

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT
$ cd "Year 2"

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2
$ ls
'Quarter 1'/'Quarter 2'/'Quarter 3'/'Quarter 4'/

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2
$ cd "Quarter 1"

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1
$ ls
'CSE2215 Computer Graphics'/'CSE2520 Big Data Processing'/'CSE2510 Machine Learning'/

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1
$ mkdir my-new-repository

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1
$ cd my-new-repository/

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository
$
```

Figure 3.5: Creating a folder for a new repository

Note that the command output above is based on Windows Git-Bash. The commands will have the same effect on Linux and MacOS, but the visualisation of changing directories may slightly differ.

Now that we have moved to the location we want, we can initialize the repository using the command `git init`.

Note that it is **not** necessary to call the command above for every subdirectory of your project. We only use it once per repository, so you should only call it on the root folder of your project.

Finally, we will also use the command below to move to the `main` line of development, where we will be making changes next. By default, git has started using `main` as initial branch, however, depending on your git version/distribution it is possible to have `master` as default branch. (figure 3.6).

```
git checkout -b main
```

Note: the command above can be regarded as executing both `git branch main` and `git checkout main` at the same time. Later in this tutorial, we will cover what branching means and why it is useful for the development

process.

```
Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository
$ git init
Initialized empty Git repository in D:/TUDELFT/Year 2/Quarter 1/my-new-repositor
y/.git/

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (mast
er)
$ git checkout -b main
Switched to a new branch 'main'

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (main)
$
```

Figure 3.6: Creating a new branch main

You can now begin making new files and working on things inside your repository locally.

3.4 Your first commit

It is good practice to set up a markdown file README.md for your projects, where you explain what your repository contains, how to use it, who created it, how to contribute to it, repository license, etc. This can be accomplished by running the following command and editing the file, which was created via notepad or any other text editor that your operating system has. For instance, nano or vim, which should come installed with Git Bash. (figure 3.7).

```
Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (main)
$ touch README.md
```

Figure 3.7: Creating a README.md file

Let us put the following text inside it:

```
# About the project
This is a test project, where we learn how to use the git version control system.

# Usage
The project has no particular usage or any programming files to run.

# Contributors
Kiril Vasilev -k.v.vasilev-1@student.tudelft.nl
```

You can replace the contributors name and email with yours. Save the file and run the command `ls` to verify that you have successfully made a new file as seen in figure 3.8.

```
Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (main)
$ ls
README.md
```

Figure 3.8: Checking if file was created in our current folder

Use the command `cat "README.md"` to show the contents of the file and verify they are correct (figure 3.9).

```
Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (main)
$ cat README.md
# About the project
This is a test project, where we learn how to use the git version control system.

# Usage
The project has no particular usage or any programming files to run.

# Contributors
Kiril Vasilev - k.v.vasilev-1@student.tudelft.nl
```

Figure 3.9: Check contents of file using cat command

Use the command `git status` to check tracked and untracked files in current repository (figure 3.10).

```
Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (main)
$ git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        README.md

nothing added to commit but untracked files present (use "git add" to track)
```

Figure 3.10: Running `git status`

Since we want to commit our new file, we first need to stage it. That is track it, so that git knows about it and will add it to the repository in the next commit. Let us call the following command: `git add "README.md"`. This will track the file. Alternatively, we can also use `git add *`, which will add all files in the current directory and all subdirectories to the staging area of the repository.

Figure 3.11 shows the steps we explained, namely we first make changes to the files. Next, we stage those changes via `git add` and finally we commit (save) those changes to the repository with `git commit` command.

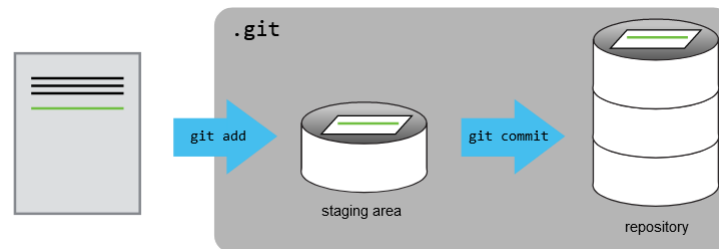


Figure 3.11: Steps to commit a change in a file

We can call `git status` again to check if the file is now being tracked (figure 3.12).


```

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (main)
$ git add README.md

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (main)
$ git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   README.md

```

Figure 3.12: Checking tracked file

Finally, we can make our first commit!

Run the command `git commit -m "My first commit"`. Note that you can have a different commit message simply by changing the text between the quotations (figure 3.13).

It is important to have clear commit messages that show what each change does. A common convention is to have the commit messages in imperative form following this structure: one line giving a summary of the commit, one empty line, followed by a paragraph explaining more in depth the commit. Note that a one-line summary is also sufficient for a commit message. Therefore, a more appropriate message would be "Add README.md file".

```

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (main)
$ git commit -m "My first commit"
[main (root-commit) 7c0f369] My first commit
 1 file changed, 8 insertions(+)
 create mode 100644 README.md

```

Figure 3.13: Committing changes

Tip: When writing a commit message, it is often useful to imagine completing the following sentence: "This commit will ...". For example, using the commit message above, we will end up with the sentence: "This commit will add README.md file".

Congratulations! You have now completed your first commit of changes via git. Calling `git status` will show us that there are no new changes, and everything is up to date (figure 3.14).

```

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (main)
$ git status
On branch main
nothing to commit, working tree clean

```

Figure 3.14: Check for new changes

3.5 Comparing commits – git log and git diff

As we have previously mentioned, one of the key advantages to Git is that it allows you to track changes and see what has been changed, when and by whom via commits.

Run the command `git log` to get that summary (figure 3.15).

```
Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (main)
$ git log
commit 7c0f3694749bc736960619b27ccfb43b64da1191 (HEAD -> main)
Author: Kiril Vasilev <k.v.vasilev-1@student.tudelft.nl>
Date: Sat Apr 30 15:43:36 2022 +0200

    My first commit
```

Figure 3.15: Checking commit history

We will now modify our README.md file by adding a line at the end of the file, specifying the date it was last modified and changing the “About the project” section:

```
# About the project
This is a test project, where we learn how to use the git version control system.

# Usage
The project has no particular usage or any programming files to run.

# Contributors
Kiril Vasilev -k.v.vasilev-1@student.tudelft.nl

Last modification: 30.04.2022
```

We stage the file and commit the changes (figure 3.16).

```
Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (main)
$ git status
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (main)
$ git add README.md

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (main)
$ git commit -m "Modify README.md contents"
[main 448332d] Modify README.md contents
1 file changed, 4 insertions(+), 2 deletions(-)
```

Figure 3.16: Stage and commit new modification

Running `git log` now shows us the two commits we have made (figure 3.17).

```

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELEFT/Year 2/Quarter 1/my-new-repository (main)
$ git log
commit 448332d55ae438b3faa50b119290f536e161fd88 (HEAD -> main)
Author: Kiril Vasilev <k.v.vasilev-1@student.tudelft.nl>
Date: Sat Apr 30 16:34:21 2022 +0200

    Modify README.md contents

commit 7c0f3694749bc736960619b27ccfb43b64da1191
Author: Kiril Vasilev <k.v.vasilev-1@student.tudelft.nl>
Date: Sat Apr 30 15:43:36 2022 +0200

    My first commit

```

Figure 3.17: Two commits we made

It is important to note that every commit is associated with a unique identifier. For example, our first commit has the identifier 7c0f3694749bc736960619b27ccfb43b64da1191 and our second commit has the identifier 448332d55ae438b3faa50b119290f536e161fd88.

We can call the command `git diff 448332d 7c0f369` to show the differences between the two commits in terms of changes of files. Note that, the first 7 characters of the commit id are enough to identify it in commands (figure 3.18):

```

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELEFT/Year 2/Quarter 1/my-new-repository (main)
$ git diff 7c0f369 448332d
diff --git a/README.md b/README.md
index bdaa183..2607fd3 100644
--- a/README.md
+++ b/README.md
@@ -1,8 +1,10 @@
 # About the project
-This is a test project, where we learn how to use the git version control system.
+This is a project, in which we learn how to use the git version control system.

 # Usage
The project has no particular usage or any programming files to run.

 # Contributors
-Kiril Vasilev - k.v.vasilev-1@student.tudelft.nl
\ No newline at end of file
+Kiril Vasilev - k.v.vasilev-1@student.tudelft.nl
+
+Last modification: 30.04.2022
\ No newline at end of file

```

Figure 3.18: Difference between two commits

The new additions are shown by a leading + and deletions by leading — to their respective row.

Note that swapping the commit identifiers also swaps the way changes are displayed (figure 3.19).

```

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (main)
$ git diff 448332d 7c0f369
diff --git a/README.md b/README.md
index 2607fd3..bdaa183 100644
--- a/README.md
+++ b/README.md
@@ -1,10 +1,8 @@
 # About the project
-This is a project, in which we learn how to use the git version control system.
+This is a test project, where we learn how to use the git version control system.

 # Usage
The project has no particular usage or any programming files to run.

 # Contributors
-Kiril Vasilev - k.v.vasilev-1@student.tudelft.nl
-
-Last modification: 30.04.2022
\ No newline at end of file
+Kiril Vasilev - k.v.vasilev-1@student.tudelft.nl
\ No newline at end of file

```

Figure 3.19: Difference between two commits swapped arguments

4 Branching and merging

4.1 What are branches

So far, we have only been making commits using the `main` branch, however, in practice, you will work on a separate branch, whose progress you will later merge with `main`.

Commits in git have graph structure, where every node is a commit and edges represent the transition (flow) between commits. Branches can be thought of as pointers to commits, whereas HEAD (something you saw when we called `git log`) points to the current commit that we are at. When you switch between branches, you can think of HEAD as the most recent commit on that branch. For example, the most recent commit on branch `main` is “Modify README.md contents”.

The graph in 4.20 below displays the current commit history we have.

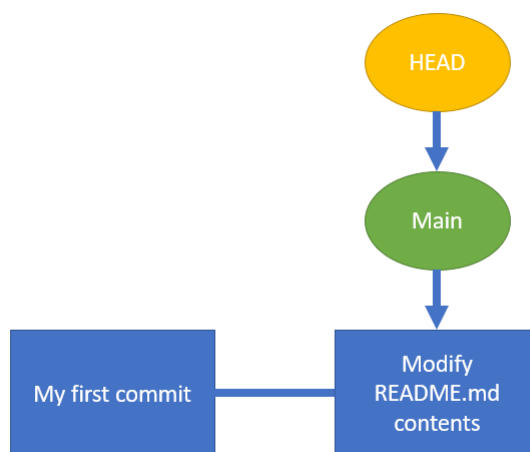


Figure 4.20: Commit graph of our repository

The main advantage of version control is that it allows developers to work together in parallel. During projects you will be working on “feature” branches and separating the work to review and merge it later. A common graphical structure of commits is shown below, where we have developers working on 3 separate branches and merging their work when necessary. This separation offers flexibility, parallelization of work and offers more control over the development process (figure 4.21).

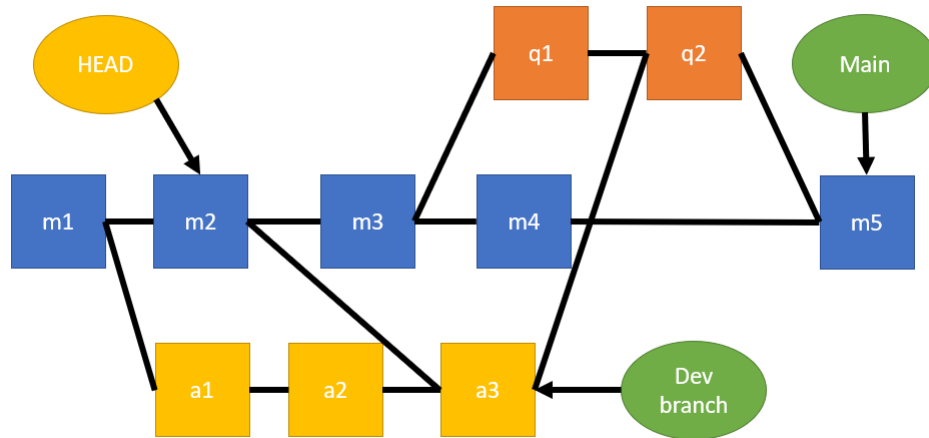


Figure 4.21: Commit graph of a sample repository

4.2 Creating a branch

First let us check on what branch we are currently at. Call the command `git branch` (figure 4.22). The green text shows what branch we are currently at.

```
Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (main)
$ git branch
* main
```

Figure 4.22: Check existing branches

Remember how at the beginning of this tutorial we created branch `main` and checked-out to it? Let us create a new branch from `main`, which we will call `getting-started`. Call the command `git branch "getting-started"`.

We can move to the newly created branch via `git checkout getting-started`. Call `git branch` again to verify that you have moved to a new branch (figure 4.23).

```

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (main)
$ git branch "getting-started"

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (main)
$ git checkout "getting-started"
Switched to branch 'getting-started'

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (getting-started)
$ git branch
* getting-started
  main

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (getting-started)
$

```

Figure 4.23: Moving to branch `getting-started`

Let us modify our README.md file again and add a “Getting started” section to it:

```

# About the project
This is a project, in which we learn how to use the git version control system.

# Getting started
If you want to follow this tutorial, make sure you have git installed on your computer.

# Usage
The project has no particular usage or any programming files to run.

# Contributors
Kiril Vasilev -k.v.vasilev-1@student.tudelft.nl

Last modification: 30.04.2022

```

We will commit the new changes (figure 4.24).

```

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (getting-started)
$ git status
On branch getting-started
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (getting-started)
$ git diff
diff --git a/README.md b/README.md
index 2607fd3..a4a093d 100644
--- a/README.md
+++ b/README.md
@@ -1,6 +1,9 @@
 # About the project
 This is a project, in which we learn how to use the git version control system.

+# Getting started
+If you want to follow this tutorial, make sure you have git installed on your computer.
+
 # Usage
 The project has no particular usage or any programming files to run.

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (getting-started)
$ git add *

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (getting-started)
$ git commit -m "Add 'Getting started' section to README.md"
[getting-started 7c820cd] Add 'Getting started' section to README.md
1 file changed, 3 insertions(+)

```

Figure 4.24: Committing new changes

Note: if you use `git add *`, you can stage all tracked files simultaneously. Be careful that you do not stage more files than you wish to!

4.3 Merging branches

To show how to merge branches, let us return to our main branch and create a new empty file called “new-file.txt”. Next, we will create a new commit for it and return to `getting-started` branch (Figure 4.25).

Note that when we switch to `main`, the `README.md` file does not contain the changes we made on the other branch.

Furthermore, the command `notepad new-file.txt` may not work for you if you are using Linux or MacOS. Instead of executing this command, open any texteditor application and create an empty file with the name `new-file.txt`.

```

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (getting-started)
$ git checkout main
Switched to branch 'main'

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (main)
$ cat README.md
# About the project
This is a project, in which we learn how to use the git version control system.

# Usage
The project has no particular usage or any programming files to run.

# Contributors
Kiril Vasilev - k.v.vasilev-1@student.tudelft.nl

Last modification: 30.04.2022
Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (main)
$ notepad new-file.txt

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (main)
$ git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        new-file.txt

nothing added to commit but untracked files present (use "git add" to track)

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (main)
$ git add new-file.txt

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (main)
$ git commit -m "Add new-file"
[main e6bf1ac] Add new-file
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 new-file.txt

```

Figure 4.25: Committing new changes on `main` branch

We now have a commit, which is ahead of the branch `getting-started`. Call the following command to visualize the current repository graph (figure 4.26):

```
git log --all --graph --decorate --oneline
```



```
Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (main)
$ git log --all --graph --decorate --oneline
* e6bf1ac (HEAD -> main) Add new-file
| * 7c820cd (getting-started) Add 'Getting started' section to README.md
|/
* 448332d Modify README.md contents
* 7c0f369 My first commit
```

Figure 4.26: Show commits graph

Before merging our progress from `getting-started`, it is advisable to merge `main` into it first. This is done because we regard `main` branch as the “face” of our project. When somebody is looking at our repository, they are likely to check the things on `main` first. Moreover, if there are any bugs from merging, it is better to resolve them on our separate branch. Sometimes it happens that after merging your progress to `main` branch, the code on `main` branch is no longer working correctly.

Therefore, we will first checkout to `getting-started`. Note that the new file we made is no longer visible. We can merge the branch `main` into `getting-started` by calling the command `git merge main`. As a result, we can get all the progress from that branch to `getting-started`. Upon merging, this will open a text editor (vim is shown in figure 4.27), through which you can edit your commit message.

Press i to enter “insert mode” and change the message. Press ESC to exit “insert mode”. Finally, press :x to exit the vim environment. You may find those steps a bit difficult, so we recommend that you press : and then x to finish immediately the merge without changing the message. In case you get stuck, follow the guide in the following link: <https://phoenixnap.com/kb/how-to-exit-vim>

```
Merge branch 'getting-started' into main

# Please enter a commit message to explain why this merge is necessary,
# especially if it merges an updated upstream into a topic branch.
#
# Lines starting with '#' will be ignored, and an empty message aborts
# the commit.
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
<r 2/Quarter 1/my-new-repository/.git/MERGE_MSG [unix] (17:41 30/04/2022)1,1 A[1]
</Year 2/Quarter 1/my-new-repository/.git/MERGE_MSG" [unix] 7L, 268C
```

Figure 4.27: Vim text editor for merging branches

We can now inspect the graph again with the new changes. The progress of the other branch (`main`) has been moved to `getting-started` as seen in figure 4.28.

```

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (main)
$ git checkout "getting-started"
Switched to branch 'getting-started'

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (getting-started)
$ git merge main
Merge made by the 'recursive' strategy.
 new-file.txt | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 new-file.txt

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (getting-started)
$ git log --all --graph --decorate --oneline
* 179c4f8 (HEAD -> getting-started) Merge branch 'main' into getting-started
|
| * e6bf1ac (main) Add new-file
| * | 7c820cd Add 'Getting started' section to README.md
|/
* 448332d Modify README.md contents
* 7c0f369 My first commit

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (getting-started)
$ ls
new-file.txt  README.md

```

Figure 4.28: New commits graph

We will now return to `main` and merge `getting-started` into it:

```

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (getting-started)
$ git checkout main
Switched to branch 'main'

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (main)
$ git merge "getting-started"
Updating e6bf1ac..179c4f8
Fast-forward
 README.md | 3 +++
 1 file changed, 3 insertions(+)

```

Figure 4.29: Merging `getting-started` in `main`

Note that this time we are not prompted to enter a commit message for our merge. Since git is a version control system and sees that there are no new changes, it will simply move the pointer of branch `main` to the same commit the branch `getting-started` points:

```
Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (main)
$ git log --all --graph --decorate --oneline
* 179c4f8 (HEAD -> main, getting-started) Merge branch 'main' into getting-started
| \
| * e6bf1ac Add new-file
* | 7c820cd Add 'Getting started' section to README.md
|/
* 448332d Modify README.md contents
* 7c0f369 My first commit
```

captionCommits graph showing a merge

4.4 Deleting branches

Branch `getting-started` has lived its use, so we can delete it as we do not plan on making new changes to it. This will not remove any of the commits made on it. It will only remove the pointer itself. Use `git branch -d getting-started` to delete the branch (figure 4.30).

```
Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (main)
$ git branch -d "getting-started"
Deleted branch getting-started (was 179c4f8).

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (main)
$ git log --all --graph --decorate --oneline
* 179c4f8 (HEAD -> main) Merge branch 'main' into getting-started
| \
| * e6bf1ac Add new-file
* | 7c820cd Add 'Getting started' section to README.md
|/
* 448332d Modify README.md contents
* 7c0f369 My first commit
```

Figure 4.30: Deleting a branch

4.5 Creating a branch from an arbitrary commit (Optional)

It is also possible to branch out from a past commit. This can be done by checking out to existing commit (figure 4.31). This operation will create a so called “detached” HEAD. Here we checkout on commit with id `c2f912f`. Note that the id will be different on your machine. Therefore, replace this id with the id of the commit on your machine, which relates to “Fix date format” commit.

and make a commit as show in figure 5.34.

```
Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (main)
$ git branch change-date-format

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (main)
$ git checkout change-date-format
Switched to branch 'change-date-format'

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (change-date-format)
$ git diff
diff --git a/README.md b/README.md
index a4a093d..d0635ab 100644
--- a/README.md
+++ b/README.md
@@ -10,4 +10,4 @@ The project has no particular usage or any programming files to run.
 # Contributors
 Kiril Vasilev - k.v.vasilev-1@student.tudelft.nl

-Last modification: 30.04.2022
\ No newline at end of file
+Last modification: 30/04/2022
\ No newline at end of file

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (change-date-format)
$ git add *

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (change-date-format)
$ git commit -m "Fix date format"
[change-date-format c2f912f] Fix date format
1 file changed, 1 insertion(+), 1 deletion(-)
```

Figure 5.34: Committing the changes

Next, we will return to `main` branch. We will edit the same line of the `README.md` and then checkout back to `change-date-format` branch (figure 5.35).

In figure 5.36 there is also a graph visualization.

```

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (change-date-format)
$ git checkout main
Switched to branch 'main'

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (main)
$ git diff
diff --git a/README.md b/README.md
index a4a093d..33f74e3 100644
--- a/README.md
+++ b/README.md
@@ -10,4 +10,4 @@ The project has no particular usage or any programming files to run.
 # Contributors
 Kiril Vasilev - k.v.vasilev-1@student.tudelft.nl

-Last modification: 30.04.2022
\ No newline at end of file
+Last date modified: 30.04.2022
\ No newline at end of file

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (main)
$ git add *

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (main)
$ git commit -m "Modify phrasing inside README file"
[main d8ec3b3] Modify phrasing inside README file
1 file changed, 1 insertion(+), 1 deletion(-)

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (main)
$ git checkout change-date-format
Switched to branch 'change-date-format'

```

Figure 5.35: Committing the changes

```

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (change-date-format)
$ git log --all --graph --decorate --oneline
* d8ec3b3 (main) Modify phrasing inside README file
| * c2f912f (HEAD -> change-date-format) Fix date format
|/
* 179c4f8 Merge branch 'main' into getting-started
| \
| * e6bf1ac Add new-file
| * 7c820cd Add 'Getting started' section to README.md
|/
* 448332d Modify README.md contents
* 7c0f369 My first commit

```

Figure 5.36: Graph visualisation

Finally, we will merge `main` into `change-date-format` and get a merge conflict as seen in figure 5.37, because we modify the same parts of the file.

```
Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (change-date-format)
$ git merge main
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
Automatic merge failed; fix conflicts and then commit the result.

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (change-date-format|MERGING)
$ cat README.md
# About the project
This is a project, in which we learn how to use the git version control system.

# Getting started
If you want to follow this tutorial, make sure you have git installed on your computer.

# Usage
The project has no particular usage or any programming files to run.

# Contributors
Kiril Vasilev - k.v.vasilev-1@student.tudelft.nl

<<<<<<< HEAD
Last modification: 30/04/2022
=====
Last date modified: 30.04.2022
>>>>>>> main
```

Figure 5.37: Merge conflict

The changes on the current branch are preceded by `<<<<<<< HEAD`, while the changes from `main` branch are preceded by `=====` and followed by `>>>>>>>main`. In order to fix this conflict, we need to open the file `README.md` with some texteditor app and fix the conflict ourselves. That is, edit out the things we do not need: open a text editor and remove the parts of the file we do not need. When done, we just need to make a new commit. Note that the current branch is now in stage “MERGING”. We have decided to take the best out of the 2 branches and merge their changes.


```

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (change-date-format)
rmat|MERGING)
$ cat README.md
# About the project
This is a project, in which we learn how to use the git version control system.

# Getting started
If you want to follow this tutorial, make sure you have git installed on your computer.

# Usage
The project has no particular usage or any programming files to run.

# Contributors
Kiril Vasilev - k.v.vasilev-1@student.tudelft.nl

Last date modified: 30/04/2022

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (change-date-format)
rmat|MERGING)
$ git add README.md
g
Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (change-date-format)
rmat|MERGING)
$ git commit -m "Merge main into change-date-format"
[change-date-format cb28051] Merge main into change-date-format

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (change-date-format)
rmat)
$ git log --all --graph --decorate --oneline
*   cb28051 (HEAD -> change-date-format) Merge main into change-date-format
| \
|  * d8ec3b3 (main) Modify phrasing inside README file
|  * c2f912f Fix date format
| /
| * 179c4f8 Merge branch 'main' into getting-started
| \
|  * e6bf1ac Add new-file
|  * 7c820cd Add 'Getting started' section to README.md
| /
* 448332d Modify README.md contents
* 7c0f369 My first commit

```

Figure 5.38: Completing the merge

Thus, we have succeeded in dealing with a merge conflict! The graph in figure 5.38 displays how the 2 branches have now been merged.

6 Tracking changes

6.1 Git log

We have previously mentioned the command `git log`, which gives a list of all commits (figure 6.39).

```

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (chan
ge-date-format)
$ git log
commit cb280516ed9b3fb31e815e0727db5584a07bf938 (HEAD -> change-date-format)
Merge: c2f912f d8ec3b3
Author: Kiril Vasilev <k.v.vasilev-1@student.tudelft.nl>
Date: Sat Apr 30 19:07:25 2022 +0200

    Merge main into change-date-format

commit d8ec3b32529902e673437aa892dbb1aa56bba495 (main)
Author: Kiril Vasilev <k.v.vasilev-1@student.tudelft.nl>
Date: Sat Apr 30 19:01:45 2022 +0200

    Modify phrasing inside README file

commit c2f912f3936480be493559790e1e4b5c1295da64
Author: Kiril Vasilev <k.v.vasilev-1@student.tudelft.nl>
Date: Sat Apr 30 18:59:13 2022 +0200

    Fix date format

commit 179c4f8b908a21baea45e1d093215fd9222ad401
Merge: 7c820cd e6bf1ac
Author: Kiril Vasilev <k.v.vasilev-1@student.tudelft.nl>
Date: Sat Apr 30 18:14:13 2022 +0200

    Merge branch 'main' into getting-started

commit e6bf1ac2b1d0233bc619d600a22534b42e263217
Author: Kiril Vasilev <k.v.vasilev-1@student.tudelft.nl>
Date: Sat Apr 30 17:31:24 2022 +0200

    Add new-file

commit 7c820cdca1a84bdff425067f4971deefcee03585
Author: Kiril Vasilev <k.v.vasilev-1@student.tudelft.nl>
Date: Sat Apr 30 17:19:51 2022 +0200

    Add 'Getting started' section to README.md

commit 448332d55ae438b3faa50b119290f536e161fd88
Author: Kiril Vasilev <k.v.vasilev-1@student.tudelft.nl>
Date: Sat Apr 30 16:34:21 2022 +0200

    Modify README.md contents

commit 7c0f3694749bc736960619b27ccfb43b64da1191
Author: Kiril Vasilev <k.v.vasilev-1@student.tudelft.nl>
Date: Sat Apr 30 15:43:36 2022 +0200

    My first commit

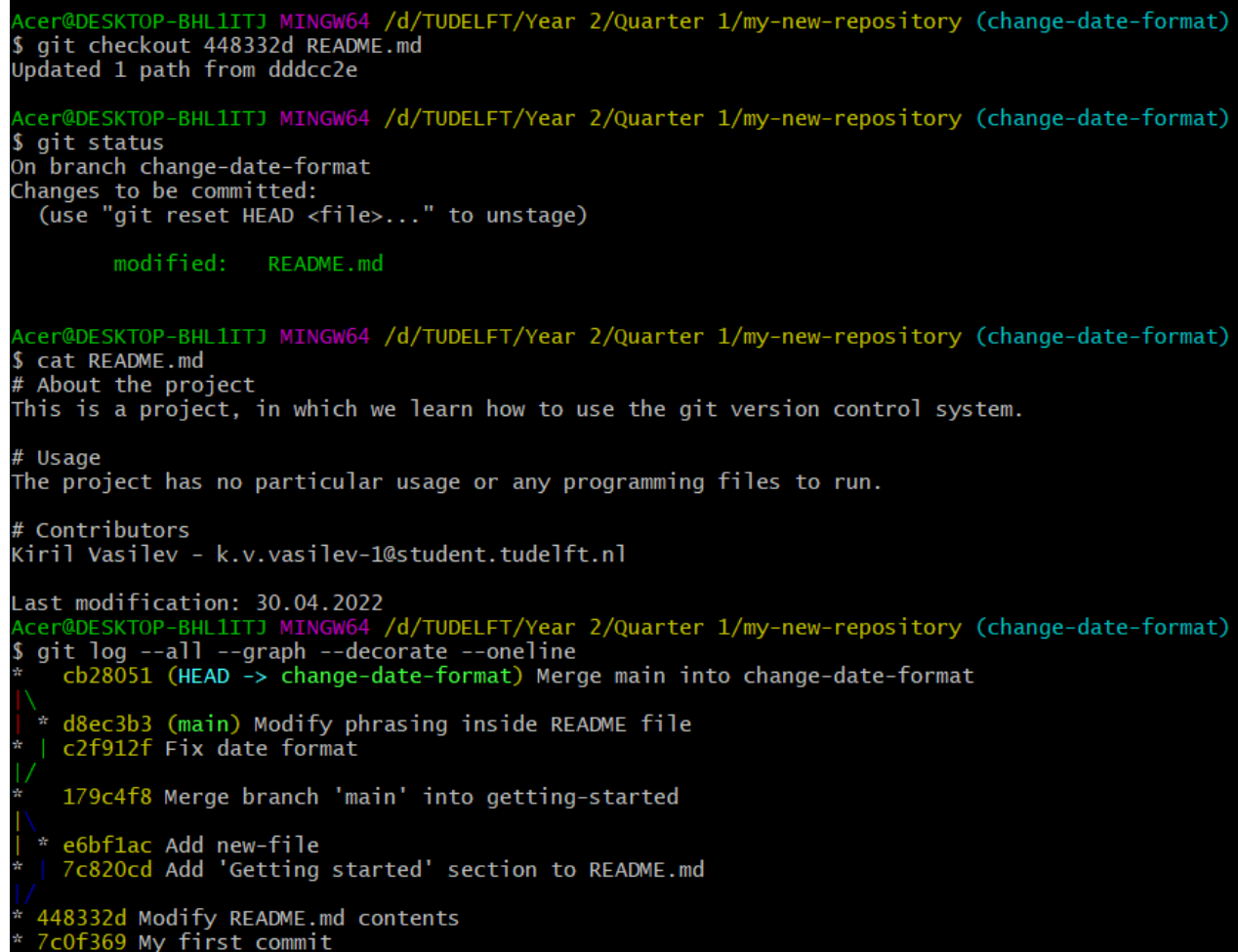
```

Figure 6.39: Results from `git log`

6.2 Git checkout – recover old versions of a file (Optional)

Suppose that we are unhappy with our changes on README.md and wish to return to a previous version, namely the one in commit 448332d. We could use the following format to achieve this:

```
git checkout 448332d README.md
```



```
Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (change-date-format)
$ git checkout 448332d README.md
Updated 1 path from dddcc2e

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (change-date-format)
$ git status
On branch change-date-format
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   README.md

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (change-date-format)
$ cat README.md
# About the project
This is a project, in which we learn how to use the git version control system.

# Usage
The project has no particular usage or any programming files to run.

# Contributors
Kiril Vasilev - k.v.vasilev-1@student.tudelft.nl

Last modification: 30.04.2022
Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (change-date-format)
$ git log --all --graph --decorate --oneline
*   cb28051 (HEAD -> change-date-format) Merge main into change-date-format
| \
| * d8ec3b3 (main) Modify phrasing inside README file
* | c2f912f Fix date format
|/
*   179c4f8 Merge branch 'main' into getting-started
| \
| * e6bf1ac Add new-file
* | 7c820cd Add 'Getting started' section to README.md
|/
*   448332d Modify README.md contents
*   7c0f369 My first commit
```

Figure 6.40: Recovery of old version

Notice in figure 6.40 that because we included file name in the checkout command we have returned to a previous version of the file, however, we have not moved in the commit timeline, and we remain on our previous position in the graph.

Notice that the file has new changes which are already included in the repository as a staged commit (to see the changes use `git diff --cached`). These changes thus appear as ‘new’ changes but are in fact based on the older version of the same file. If we want to keep this version, we can simply make a commit.

6.3 Git checkout – cancelling staged and unstaged changes (Optional)

Suppose that we no longer deem them necessary and wish to cancel all of them for README.md file. We can run the following command to delete our changes and restore the file contents to its most recent commit – the commit HEAD points to:

```
git checkout HEAD README.md
```

```

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (change-date-format)
$ git checkout HEAD README.md
Updated 1 path from 5fec6e1

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (change-date-format)
$ git status
On branch change-date-format
nothing to commit, working tree clean

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (change-date-format)
$ cat README.md
# About the project
This is a project, in which we learn how to use the git version control system.

# Getting started
If you want to follow this tutorial, make sure you have git installed on your computer.

# Usage
The project has no particular usage or any programming files to run.

# Contributors
Kiril Vasilev - k.v.vasilev-1@student.tudelft.nl

Last date modified: 30/04/2022

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (change-date-format)
$ git log --all --graph --decorate --oneline
*   cb28051 (HEAD -> change-date-format) Merge main into change-date-format
| \
| * d8ec3b3 (main) Modify phrasing inside README file
* | c2f912f Fix date format
|/
*   179c4f8 Merge branch 'main' into getting-started
| \
| * e6bf1ac Add new-file
* | 7c820cd Add 'Getting started' section to README.md
|/
*   448332d Modify README.md contents
*   7c0f369 My first commit

```

Figure 6.41: Cancelling staged and unstaged changes

As previously mentioned, we do not move in the commit timeline since we have neither staged our changes nor moved in branches (figure 6.41).

6.4 Git blame (Optional)

Another way to track who modified a specific file is via the command `git blame <file>`, where in `<file>` you put the file name and extension that you want to check. This way, you will get line-by-line information of who modified the line last, when and in which commit (figure 6.42).

```

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/remote-repository/gitlab-workshop (main)
$ git blame README.md
^7c0f369 (Kiril Vasilev 2022-04-30 15:43:36 +0200 1) # About the project
448332d5 (Kiril Vasilev 2022-04-30 16:34:21 +0200 2) This is a project, in which we learn how to use the git version control system.
^7c0f369 (Kiril Vasilev 2022-04-30 15:43:36 +0200 3)
7c820cdc (Kiril Vasilev 2022-04-30 17:19:51 +0200 4) # Getting started
7c820cdc (Kiril Vasilev 2022-04-30 17:19:51 +0200 5) If you want to follow this tutorial, make sure you have git installed on your computer.
7c820cdc (Kiril Vasilev 2022-04-30 17:19:51 +0200 6)
^7c0f369 (Kiril Vasilev 2022-04-30 15:43:36 +0200 7) # Usage
^7c0f369 (Kiril Vasilev 2022-04-30 15:43:36 +0200 8) The project has no particular usage or any programming files to run.
^7c0f369 (Kiril Vasilev 2022-04-30 15:43:36 +0200 9)
^7c0f369 (Kiril Vasilev 2022-04-30 15:43:36 +0200 10) # Contributors
448332d5 (Kiril Vasilev 2022-04-30 16:34:21 +0200 11) Kiril Vasilev - k.v.vasilev-1@student.tudelft.nl
448332d5 (Kiril Vasilev 2022-04-30 16:34:21 +0200 12)
d8ec3b32 (Kiril Vasilev 2022-04-30 19:01:45 +0200 13) Last date modified: 30.04.2022

```

Figure 6.42: Running `git blame`

7 Reverting changes

While working you may also end up in a situation where you make some changes and commit them, but you are no longer satisfied with them and wish to revert them. Here we will present a different command than the one used in Git log.

This can be done via the following command, where `<commit_id>` should be the id of the commit's changes, you wish to revert:

```
git revert <commit_id>
```

Pay attention to the fact that reverting commits that constitute a merge is more difficult than reverting normal commits, because it is necessary to add more arguments to the command above: you need to specify when reverting to the changes, to which parent commit you wish to revert. This situation is shown in the image 7.43.

```

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (change-date-format)
$ git revert cb28051
error: commit cb28051ed9b3fb31e815e0727db5584a07bf938 is a merge but no -m option was given.
fatal: revert failed

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (change-date-format)
$ git log -1
commit cb28051ed9b3fb31e815e0727db5584a07bf938 (HEAD -> change-date-format)
Merge: c2f912f d8ec3b3
Author: Kiril Vasilev <k.v.vasilev-1@student.tudelft.nl>
Date: Sat Apr 30 19:07:25 2022 +0200

    Merge main into change-date-format

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (change-date-format)
$ git revert cb28051 -m 1
[change-date-format 9126710] Revert "Merge main into change-date-format"
1 file changed, 1 insertion(+), 1 deletion(-)

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (change-date-format)
$ git log --all --graph --decorate --oneline
* 9126710 (HEAD -> change-date-format) Revert "Merge main into change-date-format"
* cb28051 Merge main into change-date-format
| \
| * d8ec3b3 (main) Modify phrasing inside README file
* | c2f912f Fix date format
| /
* 179c4f8 Merge branch 'main' into getting-started
| \
| * e6bf1ac Add new-file
* | 7c820cd Add 'Getting started' section to README.md
| /
* 448332d Modify README.md contents
* 7c0f369 My first commit

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (change-date-format)
$ cat README.md
# About the project
This is a project, in which we learn how to use the git version control system.

# Getting started
If you want to follow this tutorial, make sure you have git installed on your computer.

# Usage
The project has no particular usage or any programming files to run.

# Contributors
Kiril Vasilev - k.v.vasilev-1@student.tudelft.nl

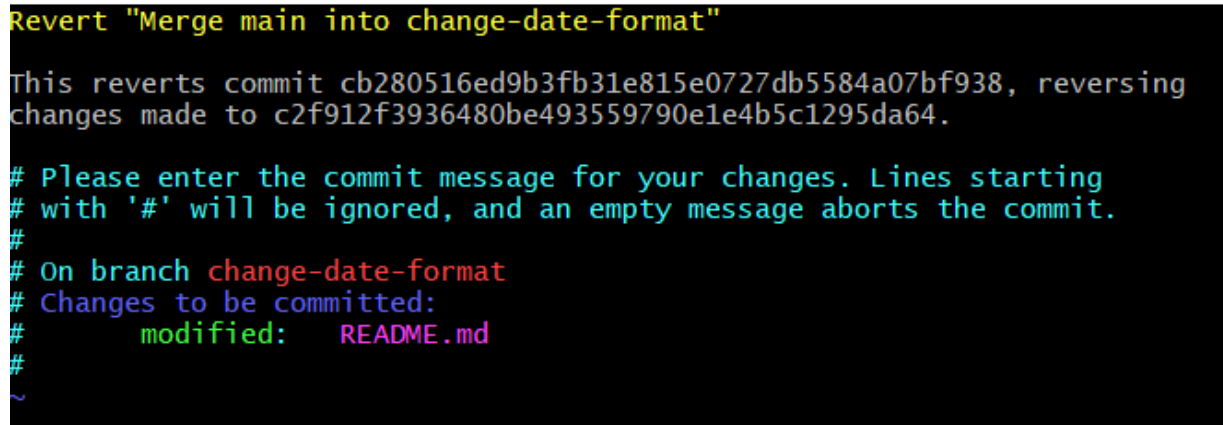
Last modification: 30/04/2022

```

Figure 7.43: Reverting changes

When executing the `git log` command with argument `-1`, we can get the most recent commit. Since that commit is a merge commit, it has 2 parent commits: `c2f912f` and `d8ec3b3`. Therefore, we can choose to which parent commit to revert to by passing an argument `-m 1` to the `git revert` command (we decided to revert to the first commit `c2f912f`).

When typing the command above a text editor window will open to modify the commit message. If you wish to make no changes to the message, just write `:x` to close it (figure 7.44).



```
Revert "Merge main into change-date-format"

This reverts commit cb280516ed9b3fb31e815e0727db5584a07bf938, reversing
changes made to c2f912f3936480be493559790e1e4b5c1295da64.

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch change-date-format
# Changes to be committed:
#   modified:   README.md
#
~
```

Figure 7.44: Revert commit message

Since git is a version control system, when reverting changes, we are making a new commit for that. Run the command for graph visualization to verify this.

8 Resetting changes/branches

Reverting changes does not alter the timeline of previous commits on a particular branch — it simply creates a new commit. Resetting, on the other hand, does alter the timeline. Resetting commits requires extra care as you have the power to reset the entire repository and the power to alter the commit history without leaving a clear record of what the changes were.

`git reset` command comes with 3 types of arguments that you can pass to it:

`--soft` – equivalent to uncommitting the changes. This way the changes to the file(s) will not be deleted and will remain staged. Hence, running `git commit` afterwards can let us commit all the changes immediately (and thereby undoing the soft reset if we made no additional changes).

`--mixed` – equivalent to uncommitting and unstaging the changes. Therefore, contrary to `--soft`, we need to first stage changes and then commit them (if we wish).

`--hard` – equivalent to uncommitting, unstaging and deleting all changes. This is the most dangerous of the 3 options as it can completely alter the commit history and leaves no record of previous changes to the file(s).

In order to show its power, we will reset the revert we did in the previous section and leave no trace of it happening. Note that we also pass as an argument the commit we are resetting to (figure 8.45).

```

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (change-date-format)
$ git reset --hard cb28051
HEAD is now at cb28051 Merge main into change-date-format

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (change-date-format)
$ git log --all --graph --decorate --oneline
*   cb28051 (HEAD -> change-date-format) Merge main into change-date-format
|  \
|   * d8ec3b3 (main) Modify phrasing inside README file
|   * c2f912f Fix date format
|  /
| * 179c4f8 Merge branch 'main' into getting-started
|  \
|   * e6bf1ac Add new-file
|   * 7c820cd Add 'Getting started' section to README.md
|  /
| * 448332d Modify README.md contents
| * 7c0f369 My first commit

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (change-date-format)
$ git status
On branch change-date-format
nothing to commit, working tree clean

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (change-date-format)
$ cat README.md
# About the project
This is a project, in which we learn how to use the git version control system.

# Getting started
If you want to follow this tutorial, make sure you have git installed on your computer.

# Usage
The project has no particular usage or any programming files to run.

# Contributors
Kiril Vasilev - k.v.vasilev-1@student.tudelft.nl

Last date modified: 30/04/2022

```

Figure 8.45: Resetting changes

9 Stashing changes (Optional)

Suppose you are working on something which is not ready yet and a friend of yours asks to check his branch and what he has done there. However, if you switch branches, you will either carry over your changes or cause a conflict and git will not allow the checkout. In this situation, it is best to stash your changes, that is save them without staging or committing them, and return to them later.

This is possible using the command `git stash save <name>`, where `<name>` depicts the name, you give to your stashed changes. Let us make some changes to our README.md file and stash them. Note that when you stash your changes, you remove them and store them for later use (figure 9.46).


```
Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (change-date-format)
$ git stash save "temporary changes on readme"
Saved working directory and index state On change-date-format: temporary changes on readme

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (change-date-format)
$ git stash list
stash@{0}: On change-date-format: temporary changes on readme

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (change-date-format)
$ git status
On branch change-date-format
nothing to commit, working tree clean
```

Figure 9.46: Stashing changes

We can use the command `git stash list` to get a list of all stashes. Notice that every stash is associated with an id next to it, which is modified every time we save or pop a stash.

We will make a few more changes to the file and stash them again. The list of stashes will grow as a result as seen in figure 9.47.

```
Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (change-date-format)
$ git stash save "temporary changes on readme 2"
Saved working directory and index state On change-date-format: temporary changes on readme 2

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (change-date-format)
$ git stash list
stash@{0}: On change-date-format: temporary changes on readme 2
stash@{1}: On change-date-format: temporary changes on readme
```

Figure 9.47: Stashing changes second time

We can now safely move to other branches and when ready return to the current one and unstash the changes. This is possible via this command: `git stash pop <index>`, where `<index>` is the index of the stash we want to unstash. We have decided to unstash changes on index 1. Popping the stash will effectively remove it from the list of stashes (figure 9.48).

```
Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (change-date-format)
$ git stash pop 1
On branch change-date-format
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@{1} (bf6471b8db320487e560ccbd248bb368abf14ad2)

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (change-date-format)
$ git stash list
stash@{0}: On change-date-format: temporary changes on readme 2
```

Figure 9.48: Unstashing changes

We may also choose to delete a stash without using it. Using the following command to achieve this: `git stash drop <index>` (figure 9.49).

```
Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (change-date-format)
$ git stash drop 0
Dropped refs/stash@{0} (8ef5e90c8194d45ef3c6ccc14e101cfcb02aa745)

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (change-date-format)
$ git stash list
```

Figure 9.49: Deleting stashed changes by index

10 GitIgnore

Sometimes it happens that you wish to have some files in your repository that you do not want to share with others (do not want to ever commit and stage). For example, database-related files or some cache files left by the applications you are using to develop your projects. The best way to ignore those files is to instruct git to ignore them for you. Hence, if git knows that it should ignore a given file, it will no longer suggest you to stage/commit changes to that file.

Another example of files you should not commit are the .DS_Store files, which are only generated on MacOS platforms.

Suppose we want to ignore all files of type png. Suppose that we also wish to ignore a folder and all its contents called data, which we will create now. In order to do so, we need to create a new file called .gitignore. Notice that the file has no name and has extension gitignore (normally we can use ls to list files (figure 10.50), but since those beginning with a ‘.’ are generally hidden we pass the argument -a to list all files - figure 10.51).

```
Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (change-date-format)
$ ls
new-file.txt  README.md
```

Figure 10.50: Listing visible files

```
Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (change-date-format)
$ ls -a
./ ../ .git/ .gitignore data/ image.png image2.png new-file.txt README.md
```

Figure 10.51: Listing visible and hidden files

To instruct git to ignore that file and everything inside that folder, we must modify the contents of our gitignore file to contain the following (where ‘/’ indicates all files in the directory ‘data’ and the ‘*’ symbol is a wildcard to ignore all files with extension ‘.png’). Note that you can open the gitignore file as a normal text file to modify it (using a text editor app) (figure 10.52).

```
data/
*.png
```

```
Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (change-date-format)
$ cat .gitignore
data/
*.png
```

Figure 10.52: Check contents of .gitignore

We would first need to commit our gitignore file to come into effect and ignore files and directories (figure 10.53).

```
Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (change-date-format)
$ git add .gitignore

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (change-date-format)
$ git commit -m "Add gitignore file"
[change-date-format f731f82] Add gitignore file
1 file changed, 2 insertions(+)
create mode 100644 .gitignore
```

Figure 10.53: Comitting .gitignore

The changes are no longer tracked (the images and files in the data folder cannot be staged). This can be verified by using `git status` (figure 10.54).

```
Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (change-date-format)
$ git status
On branch change-date-format
nothing to commit, working tree clean
```

Figure 10.54: Verify files are ignored

Another advantage you get is that git will not let you stage a file, which you have specifically set to be ignored by git (figure 10.55).

```
Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (change-date-format)
$ git add image.png
The following paths are ignored by one of your .gitignore files:
image.png
Use -f if you really want to add them.
```

Figure 10.55: Verify ignored files cannot be staged

Depending on the projects you work on, there already exist pre-made templates for gitignore files that you can make use of. Just make sure that you pick a gitignore file that matches the language you are working on and/or the IDE (integrated development environment) you are using.

11 Interactive commits (Optional)

So far when we wish to commit changes to a file, we were only able to commit all of them. However, sometimes, we may wish to commit only some of the changes. This can be done by passing an argument `-p` to your `git add` command (figure 11.56).

```
Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (change-date-format)
$ git add -p README.md
diff --git a/README.md b/README.md
index 61f9dd5..7870aa1 100644
--- a/README.md
+++ b/README.md
@@ -1,13 +1,14 @@
 # About the project
-This is a project, in which we learn how to use the git version control system.
+This is a project, in which we learn how to use the git version control system.s

 # Getting started
-If you want to follow this tutorial, make sure you have git installed on your computer.
+If you want to follow this tutorial, make sure you have git installed on your computer.s

 # Usage
-The project has no particular usage or any programming files to run.
+The project has no particular usage or any programming files to run.s

 # Contributors
-Kiril Vasilev - k.v.vasilev-1@student.tudelft.nl
+Kiril Vasilev - k.v.vasilev-1@student.tudelft.nls

Last date modified: 30/04/2022
+S
\ No newline at end of file
Stage this hunk [y,n,q,a,d,s,e,?] ?
```

Figure 11.56: Interactive staging of a file

You will get an overview of all the changes in your file and git will ask you at every step what you wish to do. Use `?` (figure 11.57) to get an explanation of what each option does. A hunk denotes a block of changes (such as the one in the image above). Git allows you to split the hunk into smaller hunks until a hunk becomes as small as 1 change.

```
Stage this hunk [y,n,q,a,d,s,e,?]? ?
y - stage this hunk
n - do not stage this hunk
q - quit; do not stage this hunk or any of the remaining ones
a - stage this hunk and all later hunks in the file
d - do not stage this hunk or any of the later hunks in the file
s - split the current hunk into smaller hunks
e - manually edit the current hunk
? - print help
```

Figure 11.57: Available options for interactive commits

We have decided to split the hunk into smaller hunks and stage only the second, fourth and fifth hunks and leave out the first and third (figure 11.58).

```

Stage this hunk [y,n,q,a,d,s,e,?]? s
Split into 5 hunks.
@@ -1,4 +1,4 @@
# About the project
-This is a project, in which we learn how to use the git version control system.
+This is a project, in which we learn how to use the git version control system.s

# Getting started
Stage this hunk [y,n,q,a,d,j,J,g,/,e,?]? n
@@ -3,5 +3,5 @@

# Getting started
-If you want to follow this tutorial, make sure you have git installed on your computer.
+If you want to follow this tutorial, make sure you have git installed on your computer.s

# Usage
Stage this hunk [y,n,q,a,d,K,j,J,g,/,e,?]? y
@@ -6,5 +6,5 @@

# Usage
-The project has no particular usage or any programming files to run.
+The project has no particular usage or any programming files to run.s

# Contributors
Stage this hunk [y,n,q,a,d,K,j,J,g,/,e,?]? n
@@ -9,5 +9,5 @@

# Contributors
-Kiril Vasilev - k.v.vasilev-1@student.tudelft.nl
+Kiril Vasilev - k.v.vasilev-1@student.tudelft.nls

Last date modified: 30/04/2022
Stage this hunk [y,n,q,a,d,K,j,J,g,/,e,?]? y
@@ -12,2 +12,3 @@

Last date modified: 30/04/2022
+s
\ No newline at end of file
Stage this hunk [y,n,q,a,d,K,g,/,e,?]? y

```

Figure 11.58: Staging second, fourth and fifth hunks

We can verify that we have successfully achieved this by calling `git diff --staged` to compare the HEAD with the staged files – only second, fourth and fifth changes were staged (figure 11.59).

```

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (change-date-format)
$ git status
On branch change-date-format
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   README.md

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   README.md

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (change-date-format)
$ git diff --staged
diff --git a/README.md b/README.md
index 61f9dd5..a86d69a 100644
--- a/README.md
+++ b/README.md
@@ -2,12 +2,13 @@
 This is a project, in which we learn how to use the git version control system.

 # Getting started
-If you want to follow this tutorial, make sure you have git installed on your computer.
+If you want to follow this tutorial, make sure you have git installed on your computer.s

 # Usage
 The project has no particular usage or any programming files to run.

 # Contributors
-Kiril Vasilev - k.v.vasilev-1@student.tudelft.nl
+Kiril Vasilev - k.v.vasilev-1@student.tudelft.nls

 Last date modified: 30/04/2022
+s
\ No newline at end of file

```

Figure 11.59: Show difference between staged and unstaged files

Of course, since those changes contain mistakes (extra s at the end of 3 sentences), we can reset them. We use the `--mixed` argument to unstage the changes, but not delete them (figure 11.60).

```

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (change-date-format)
$ git reset --mixed HEAD
Unstaged changes after reset:
M   README.md

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (change-date-format)
$ git status
On branch change-date-format
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")

```

Figure 11.60: Unstaging the changes

You can also commit part of the staged changes (the same way as we did with staging only selected lines) by passing the `-p` parameter to `git commit` command.

12 Common mistakes using Git

While working you may end up making one of the following mistakes, so make sure you double check you are executing the correct commands and that you know what you wish to do:

- Merging from the wrong branch
- Committing on the wrong branch
- Resetting more than you wanted
- Committing the wrong file or changes
- Giving a wrong branch name
- Writing a commit message with a mistake in it
- Deleting the wrong branch
- Et cetera.

13 Interacting with a remote repository

As you might have noticed we have only interacted with a local repository up to now, without any possibility of collaboration with other people. In order to do so, we require a remote repository that is hosted somewhere. For example, on GitLab/GitHub/BitBucket instance. Lucky enough, TU Delft has its own instance of GitLab, where we can make repositories.

Note: You can only make a repository if you are inside our group – MUDE. Without being inside it, you are not able to create repositories on your own!

13.1 Creating a remote repository

Let us first begin by making a new remote repository on GitLab. We are given the opportunity to either create a new blank repository or create a repository from an existing project template or import an existing repository from another project (figure 13.61).

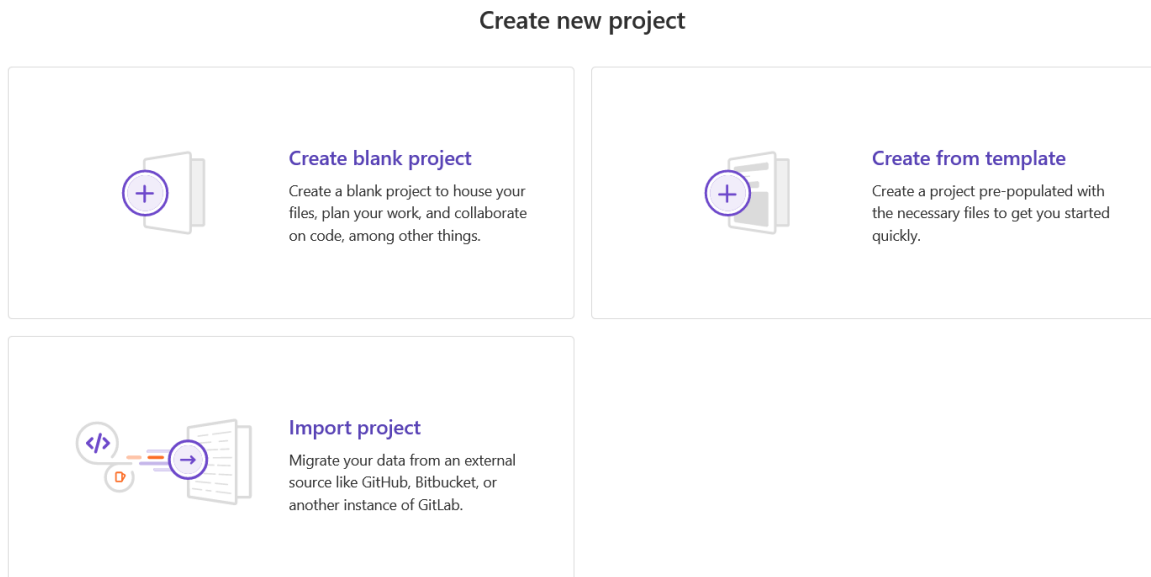


Figure 13.61: Creating a new project

We will start off by making a blank repository. Enter the necessary details such as project name and unselect the checkbox “Initialize repository with a README”, since it is better to create the README file ourselves (figure 13.62).

Project name

Gitlab Workshop

Project URL

https://gitlab.tudelft... mude

Project slug

gitlab-workshop

Project description (optional)

Description format

Visibility Level ?

☒ Private
Project access must be granted explicitly to each user. If this project is part of a group, access will be granted to members of the group.


Project Configuration

☐ Initialize repository with a README
Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing repository.


Create project Cancel



Figure 13.62: Creating a new project - filling details

Upon creating the repository, GitLab gives us a set of instructions to follow depending on what we would like to do (figure 13.63).



Gitlab Workshop

Project ID: 6885 

 Star 0

Gitlab workshop for student army







Invite your team


Add members to this project and start collaborating with your team.

[Invite members](#)

The repository for this project is empty

You can get started by cloning the repository or start adding files to it with one of the following options.

[Clone](#)  Upload File  New file  Add README  Add LICENSE  Add CHANGELOG  Add CONTRIBUTING

 Configure Integrations

Command line instructions

You can also upload existing files from your computer using the instructions below.

Git global setup

```
git config --global user.name "Kiril Vasilev"
git config --global user.email "k.v.vasilev-1@student.tudelft.nl"
```

Create a new repository

```
git clone git@gitlab.tudelft.nl:mude/gitlab-workshop.git
cd gitlab-workshop
git switch -c main
touch README.md
git add README.md
git commit -m "add README"
git push -u origin main
```

Push an existing folder

```
cd existing_folder
git init --initial-branch=main
git remote add origin git@gitlab.tudelft.nl:mude/gitlab-workshop.git
git add .
git commit -m "Initial commit"
git push -u origin main
```

Push an existing Git repository

```
cd existing_repo
git remote rename origin old-origin
git remote add origin git@gitlab.tudelft.nl:mude/gitlab-workshop.git
git push -u origin --all
git push -u origin --tags
```

Figure 13.63: Our new repository

That was it! We have successfully created our own remote repository!

13.2 Setting up SSH agent

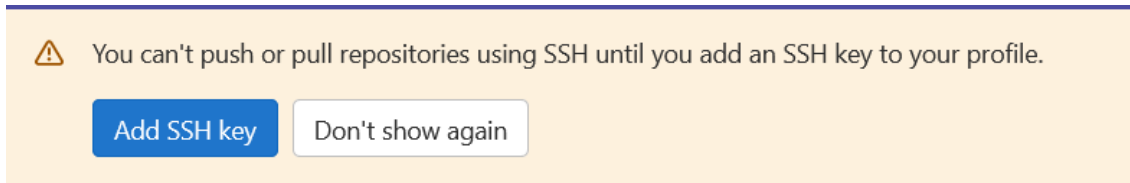


Figure 13.64: SSH prompt on GitLab

Note: you only need to setup SSH key once. You do not need to set it up for every repository!

Note: if you are unable to setup SSH key, do not worry. You can continue pushing to a remote repository, but you will be required to enter your NetID and password for every interaction.

After creating the repository, you may have noticed that this message box appears on top of the webpage (figure 13.64). The reason for this is that we need to follow a few more steps before we can interact with our remote repository. Press the button “Add SSH key” and open again your git bash terminal.

In short SSH keys are an easy way to authenticate yourself when interacting with a remote repository. They will remove the need to use your NetID and password for every commit. SSH keys consist of 2 parts – public and private part. The private key can be viewed as your password. The public key is the one you can share with others without worrying about anything. Make sure you upload your public key later.

Let us begin by generating an SSH key. The official GitLab documentation already contains a nice tutorial for setting up your SSH key, which can be found here: <https://docs.gitlab.com/ee/user/ssh.html> (figure 13.65).

Generate an SSH key pair

If you do not have an existing SSH key pair, generate a new one.

1. Open a terminal.
2. Type `ssh-keygen -t` followed by the key type and an optional comment. This comment is included in the `.pub` file that's created. You may want to use an email address for the comment.

For example, for ED25519:

```
ssh-keygen -t ed25519 -C "<comment>"
```

For 2048-bit RSA:

```
ssh-keygen -t rsa -b 2048 -C "<comment>"
```

3. Press Enter. Output similar to the following is displayed:

```
Generating public/private ed25519 key pair.  
Enter file in which to save the key (/home/user/.ssh/id_ed25519):
```

4. Accept the suggested filename and directory, unless you are generating a [deploy key](#) or want to save in a specific directory where you store other keys.

You can also dedicate the SSH key pair to a [specific host](#).

5. Specify a [passphrase](#) [↗](#):

```
Enter passphrase (empty for no passphrase):  
Enter same passphrase again:
```

6. A confirmation is displayed, including information about where your files are stored.

A public and private key are generated. [Add the public SSH key to your GitLab account](#) and keep the private key secure.

Figure 13.65: Setting up SSH instructions

We will follow the steps and do the same (figure 13.66).

```
Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (change-date-format)
$ ssh-keygen -t ed25519 -C "Gitlab key"
Generating public/private ed25519 key pair.
Enter file in which to save the key (/c/Users/Acer/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in .
Your public key has been saved in .pub.
The key fingerprint is:
SHA256:1ZFz8h2p+u8EYb8L+9WBRHJVv6Yw6NqczDKSyiiNgjk Gitlab key
The key's randomart image is:
+--[ED25519 256]--+
|                .o o.o||
|                +o= o  ||
|                o=o+  ||
|                ...+oo  ||
|                S. +...+ ||
|                . . o.o.o||
|. +      . . . . .o o||
|E +  o o* . . + o ||
|oo o. ..o* . +=  ||
+-----[SHA256]-----+
```

Figure 13.66: Generating SSH key

Locate the folder `/c/Users/<your username>/ssh/` and copy the contents of the file `id_ed25519.pub` and paste them in the textbox named Key (figure 13.67). Setting title and expiration date is optional. Beware that the folder above is for windows only. In case you use GNU/Linux or Mac, the folder, where the keys are stored, will be different. On Mac it would be `/Users/<username>/ssh` and on Linux, it should be `/home/<username>/ssh`, where `<username>` is the username of your OS profile:

Add an SSH key

Add an SSH key for secure access to GitLab. [Learn more.](#)

Key

Title

e.g. My MacBook key

Expiration date

mm / dd / yyyy

Key can still be used after expiration.

Add key

```
Welcome to GitLab, @kvasilev!  
debug1: client_input_channel_req: channel 0 rtype exit-status reply 0  
debug1: client_input_channel_req: channel 0 rtype eow@openssh.com reply 0  
debug1: channel 0: free: client-session, nchannels 1  
Transferred: sent 2764, received 2832 bytes, in 0.1 seconds  
Bytes per second: sent 23254.0, received 23826.1  
debug1: Exit status 0
```

```
git remote add origin git@gitlab.tudelft.nl:mude/gitlab-workshop.git
git push -u origin --all
```


Make sure you copy paste the commands above from your own remote repository on GitLab (figure 13.69)!

```
Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (main)
$ git remote add origin git@gitlab.tudelft.nl:mude/gitlab-workshop.git

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (main)
$ git push -u origin --all
Enumerating objects: 26, done.
Counting objects: 100% (26/26), done.
Delta compression using up to 12 threads
Compressing objects: 100% (21/21), done.
Writing objects: 100% (26/26), 2.62 KiB | 670.00 KiB/s, done.
Total 26 (delta 7), reused 0 (delta 0)
remote:
remote: To create a merge request for change-date-format, visit:
remote:   https://gitlab.tudelft.nl/mude/gitlab-workshop/-/merge_requests/new?merge_request%
5Bsource_branch%5D=change-date-format
remote:
To gitlab.tudelft.nl:mude/gitlab-workshop.git
 * [new branch]      change-date-format -> change-date-format
 * [new branch]      main -> main
Branch 'change-date-format' set up to track remote branch 'change-date-format' from 'origin'
Branch 'main' set up to track remote branch 'main' from 'origin'.
```

Figure 13.69: Pushing local repository to GitLab

Open the GitLab page of your repository and refresh the page. You should now see it there (figure 13.70).


Gitlab Workshop
Project ID: 6885


🔔
☆ Star 0
🍴 Fork 0

🔗 6 Commits
🌿 2 Branches
🏷️ 0 Tags
📄 215 KB Files
💾 215 KB Storage

Gitlab workshop for student army

main
gitlab-workshop /
+

History
Find file
Web IDE
📄
Clone


Modify phrasing inside README file
d8ec3b32

Kiril Vasilev authored 6 days ago

📁 Upload File
📄 README
⚙️ Auto DevOps enabled
📄 Add LICENSE
📄 Add CHANGELOG
📄 Add CONTRIBUTING

📄 Add Kubernetes cluster
⚙️ Configure Integrations

Name	Last commit	Last update
📄 README.md	Modify phrasing inside README file	6 days ago
📄 new-file.txt	Add new-file	6 days ago

📄 README.md

About the project

This is a project, in which we learn how to use the git version control system.

Getting started

If you want to follow this tutorial, make sure you have git installed on your computer.

Usage

The project has no particular usage or any programming files to run.

Contributors

Kiril Vasilev - k.v.vasilev-1@student.tudelft.nl

Last date modified: 30.04.2022

Figure 13.70: Uploaded repository on GitLab

Now we have pushed (uploaded) all our commits to the online repository.

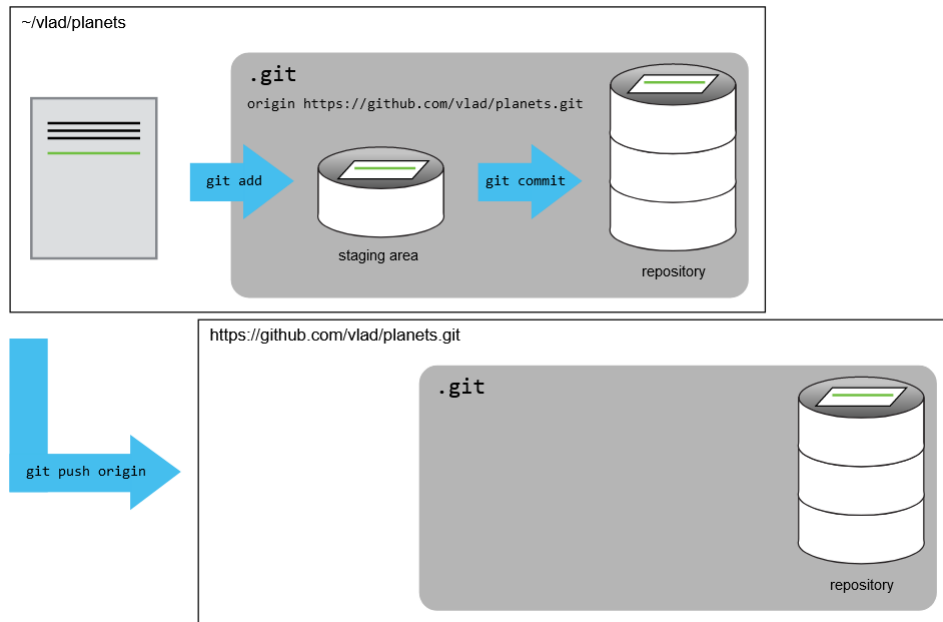


Figure 13.71: Pushing to a repository

Figure 13.71 shows an extra step, which we have not discussed before, namely `git push`. This command allows us to push local changes to a remote repository. To show this, let us edit our "new-file.txt" file and add a single line, stage the changes, commit them and finally push them (figure 13.72).

```

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELEFT/Year 2/Quarter 1/my-new-repository (main)
$ cat new-file.txt
line
Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELEFT/Year 2/Quarter 1/my-new-repository (main)
$ git add new-file.txt

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELEFT/Year 2/Quarter 1/my-new-repository (main)
$ git commit -m "Modify new-file"
[main 1b3c21a] Modify new-file
1 file changed, 1 insertion(+)

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELEFT/Year 2/Quarter 1/my-new-repository (main)
$ git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 12 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 298 bytes | 298.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To gitlab.tudelft.nl:mude/gitlab-workshop.git
d8ec3b3..1b3c21a  main -> main

```

Figure 13.72: Pushing the new changes

On the GitLab webpage, we can verify the change has been pushed (figure 13.73).

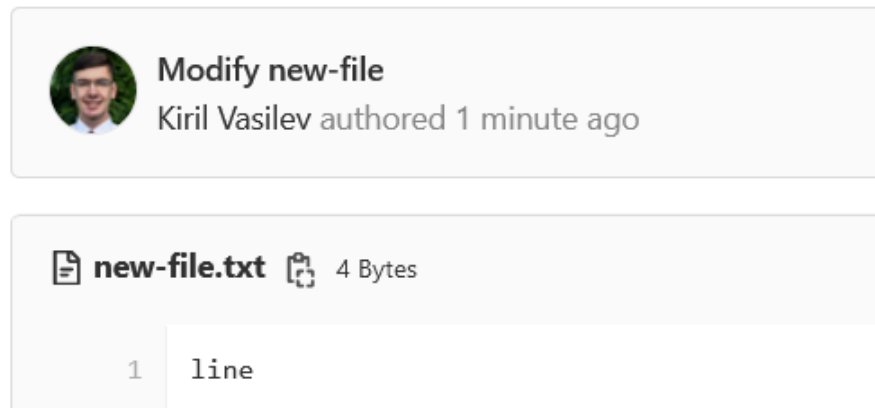


Figure 13.73: Verify pushed commit

13.4 Force pushing changes (Dangerous)

Remember we saw that it is possible to reset commits. If your commit history does not match that of the remote repository, you will be unable to push changes. Sometimes it is necessary to overwrite your remote repository and delete things that used to exist. We strongly recommend that you take extra care when doing this, because it is possible for you to lose your progress. Moreover, force pushing can mess up other people's work and cause a lot of trouble. Therefore, a better alternative as we discussed previously is reverting commits.

The command for force pushing commits on a specific branch is (replace `<branch_name>` with the name of your branch):

```
git push origin <branch_name> --force
```

13.5 Fetching/Pulling from a repository

When working with other people, it may happen that someone else has pushed something to the remote repository without your knowledge. In that case, you will need to pull their changes. For the sake of this exercise, to show how pulling/fetching works, we will first hard reset our branch 1 commit back to simulate this (figure 13.74).

```
Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (main)
$ git log --all --graph --decorate --oneline
* 1b3c21a (HEAD -> main, origin/main) Modify new-file
| * f731f82 (origin/change-date-format, change-date-format) Add gitignore file
| * cb28051 Merge main into change-date-format
|
| \
| /
|
| * d8ec3b3 Modify phrasing inside README file
| * c2f912f Fix date format
|
| /
|
| * 179c4f8 Merge branch 'main' into getting-started
| \
|  * e6bf1ac Add new-file
| * | 7c820cd Add 'Getting started' section to README.md
| /
|
| * 448332d Modify README.md contents
| * 7c0f369 My first commit

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (main)
$ git reset --hard HEAD~1
HEAD is now at d8ec3b3 Modify phrasing inside README file

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (main)
$ git log --all --graph --decorate --oneline
* 1b3c21a (origin/main) Modify new-file
| * f731f82 (origin/change-date-format, change-date-format) Add gitignore file
| * cb28051 Merge main into change-date-format
|
| \
| /
|
| * d8ec3b3 (HEAD -> main) Modify phrasing inside README file
| * c2f912f Fix date format
|
| /
|
| * 179c4f8 Merge branch 'main' into getting-started
| \
|  * e6bf1ac Add new-file
| * | 7c820cd Add 'Getting started' section to README.md
| /
|
| * 448332d Modify README.md contents
| * 7c0f369 My first commit
```

Figure 13.74: Resetting local commit

Notice that we were at main branch and that passed a parameter HEAD~1 when resetting the branch 1 commit back. This argument tells git to reset to 1 commit behind HEAD. If we wanted to reset 2 commits behind HEAD, we could have used HEAD~2.

Furthermore, notice that remote branches are denoted with red text and are preceded by origin text, while local branches are colored green.

There is a major distinction between fetching and pulling changes. Fetching will download all the changes locally but will not merge them yet. Therefore, fetching is useful to check the progress from the remote repository. Pulling on the other hand, will fetch and attempt to merge the changes to the local repository. If there is a conflict, git will alert the user and let them deal with the conflict.

Moreover, before committing and pushing changes, it is highly recommended to first check if anyone has committed on the remote repository before doing so. Otherwise, you will be unable to push to the remote, because it has a different commit history compared to the local.

Use `git fetch` to fetch changes and `git pull` to pull changes from remote repository (figure 13.75).

```
Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (main)
$ git fetch

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (main)
$ git pull
Updating d8ec3b3..1b3c21a
Fast-forward
 new-file.txt | 1 +
 1 file changed, 1 insertion(+)
```

Figure 13.75: Fetching from remote

13.6 Cloning existing repository

Suppose that we had no local repository and we wanted to work on the remote repository (which we created ourselves). In order to do this, we need to clone the remote repository before we can start working on it.

Open the GitLab webpage of your repository and press "Clone" and copy the "Clone with SSH" textbox. You can also clone using HTTPS, but that may require you to write your NetID and password every time you push or pull changes from the remote repository (figure 13.76).

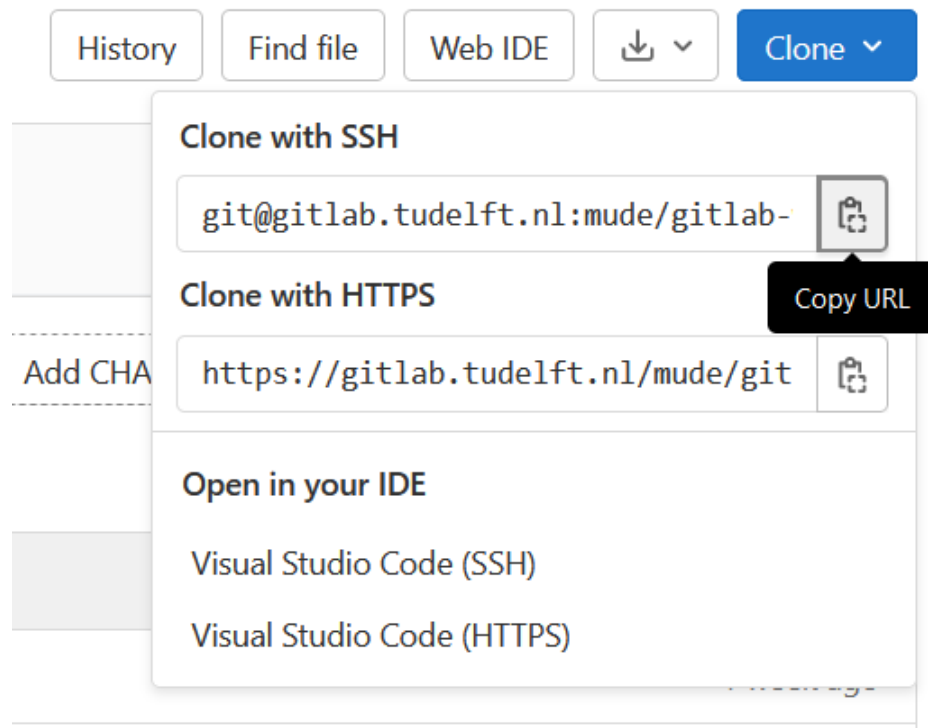


Figure 13.76: Copy repository url

Next make a new folder, where you wish to place the repository (figure 13.77).

Note that it is not mandatory to create a new folder. You can clone a repository in an existing folder as well.

```
Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (main)
$ cd ..

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1 (change-date-format)
$ cd my-new-repository/

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/my-new-repository (main)
$ cd ..

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1
$ mkdir remote-repository

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1
$ cd remote-repository/

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/remote-repository
$ |
```

Figure 13.77: Creating a new folder

Next, run the `git clone` command and provide it with the SSH link you copied (figure 13.78).

```
Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/remote-repository
$ git clone git@gitlab.tudelft.nl:mude/gitlab-workshop.git
Cloning into 'gitlab-workshop'...
remote: Enumerating objects: 29, done.
remote: Counting objects: 100% (29/29), done.
remote: Compressing objects: 100% (23/23), done.
remote: Total 29 (delta 8), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (29/29), done.
Resolving deltas: 100% (8/8), done.
```

Figure 13.78: Cloning a repository

```
Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/remote-repository
$ ls
gitlab-workshop/

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/remote-repository
$ cd gitlab-workshop/

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/remote-repository/gitlab-workshop (
main)
$ git log --all --graph --decorate --oneline
* 1b3c21a (HEAD -> main, origin/main, origin/HEAD) Modify new-file
| * f731f82 (origin/change-date-format) Add gitignore file
| * cb28051 Merge main into change-date-format
| \
| |
| | \
| | /
| /
| * d8ec3b3 Modify phrasing inside README file
| * c2f912f Fix date format
| /
| * 179c4f8 Merge branch 'main' into getting-started
| \
| * e6bf1ac Add new-file
| * 7c820cd Add 'Getting started' section to README.md
| /
| * 448332d Modify README.md contents
| * 7c0f369 My first commit

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/remote-repository/gitlab-workshop (
main)
$
```

Figure 13.79: Inspecting cloned repository graph

We can observe that we have successfully cloned the repository (figure 13.79).

Notice that we have no local branch for `change-date-format`. To make one, we need to run the following command (figure 13.80):

```
git checkout -b change-date-format origin/change-date-format
```

```

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/remote-repository/gitlab-workshop (
main)
$ git checkout -b change-date-format origin/change-date-format
Switched to a new branch 'change-date-format'
Branch 'change-date-format' set up to track remote branch 'change-date-format' from 'origin'
.

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/remote-repository/gitlab-workshop (
change-date-format)
$ git log --all --graph --decorate --oneline
* 1b3c21a (origin/main, origin/HEAD, main) Modify new-file
| * f731f82 (HEAD -> change-date-format, origin/change-date-format) Add gitignore file
| * cb28051 Merge main into change-date-format
|/
|/
| * d8ec3b3 Modify phrasing inside README file
| * c2f912f Fix date format
|/
| * 179c4f8 Merge branch 'main' into getting-started
|/
| * e6bf1ac Add new-file
| * 7c820cd Add 'Getting started' section to README.md
|/
| * 448332d Modify README.md contents
| * 7c0f369 My first commit

```

Figure 13.80: Checkout a remote branch

As the output by git suggests, now the local and remote branches are tied together. Therefore, if we make commits on our local version of the branch and push them to the remote repository, git will know to which branch to append those changes.

14 GitLab videos

On top of this tutorial, there are 2 videos, which show the Basics and Advanced parts (optional) of using GitLab.

- GitLab basics: <https://youtu.be/H23G1582d1o>
- GitLab Advanced (optional): <https://youtu.be/B7syxptEnZ0>. Note that in the advanced part of the GitLab tutorial, we present how Merge Requests are made and handled. The steps to achieve this are very similar to the steps you took during the PEP8 peer-review assignment.

15 Aliasing commands (Optional)

This section of the workshop is optional.

You may be feeling fed up with writing this long command to visualize the commit graph. This can easily be fixed by introducing an alias for it. Namely, you can set up an alias called graph for it. Hence, you can replace the long command with a shorter one called git graph:

```
git config --global alias.graph "log --all --graph --decorate --oneline"
```

Try using `git graph` afterwards to verify it worked (figure 15.81).


```
Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/remote-repository/gitlab-workshop (change-date-format)
$ git config --global alias.graph "log --all --graph --decorate --oneline"

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/remote-repository/gitlab-workshop (change-date-format)
$ git graph
* 1b3c21a (origin/main, origin/HEAD, main) Modify new-file
| * f731f82 (HEAD -> change-date-format, origin/change-date-format) Add gitignore file
| * cb28051 Merge main into change-date-format
| | \
| | /
| / \
| / \
* | d8ec3b3 Modify phrasing inside README file
* | c2f912f Fix date format
| / \
* | 179c4f8 Merge branch 'main' into getting-started
| | \
| | * e6bf1ac Add new-file
| | * 7c820cd Add 'Getting started' section to README.md
| / \
* 448332d Modify README.md contents
* 7c0f369 My first commit
```

Figure 15.81: Aliasing a command

If you wish to remove an alias, simply execute the following:

```
git config --global --unset alias.graph
```

```
Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/remote-repository/gitlab-workshop (
change-date-format)
$ git config --global --unset alias.graph

Acer@DESKTOP-BHL1ITJ MINGW64 /d/TUDELFT/Year 2/Quarter 1/remote-repository/gitlab-workshop (
change-date-format)
$ git graph
git: 'graph' is not a git command. See 'git --help'.

The most similar commands are
    branch
    grep
```

Figure 15.82: Alias is removed

16 Conclusion

At first glance, Git has a steep learning curve and requires a lot of knowledge and time to spend learning and practicing. However, knowing how to use Git will pay off in the future! We hope that you found this tutorial useful. Make sure you return to it occasionally when you are in doubt about something. Remember that you can always Google things to learn more. What is more, this tutorial is not exhaustive, but covers the required basics of Git, which should be sufficient for you to work in group projects.

Finally, in case something goes wrong, remember to follow the instructions in figure 16.83 ;)



Figure 16.83: Source: <https://xkcd.com/1597/>

17 References and used resources

While creating this tutorial, we made use of the following external sources:

- <http://swcarpentry.github.io/git-novice/>
- <https://coderefinery.github.io/git-intro/>
- <https://coderefinery.github.io/github-without-command-line/>
- <https://coderefinery.github.io/git-collaborative/>

18 Appendices

In the appendices you can find commonly used Git and Bash commands. You may notice that we surround some of them with quotations (“) . These are only necessary when the argument you are passing on contains whitespaces. For example, this is usually the case in commit messages. However, if you have no spaces in your arguments, you can omit the quotations.

18.1 Appendix A: Commonly used Git commands

Table with commonly used Git commands:

Explanation of git command	Git command	Example
Initialise repository	git init	git init
Add files to the staging area	git add	git add "README.md"
Commit changes	git commit	git commit -m "Add README.md file"
Merge the changes from one branch to another	git merge	git merge "development"
Get a list of all commits	git log	git log
Get an overview of who was the last one to modify lines in a file	git blame	git blame README.md
Show difference between commits (and/or files)	git diff	git diff 1234567 7654321
Create a branch	git branch	git branch feature1
Move to another branch	git checkout	git checkout feature1
Revert a commit	git revert	git revert 1234567
Store changes (without committing or staging)	git stash	git stash save "my-stash"
Uncommit and unstage changes on a branch	git reset --mixed	git reset --mixed development
Push changes to a remote repository	git push	git push
Fetch and merge changes from a remote repository	git pull	git pull
Fetch changes from a remote repository	git fetch	git fetch
Clone a remote repository	git clone	git clone git@gitlab.ewi.tudelft/myrepository.git
Associate current repository with remote one	git remote add origin	git remote add origin git@gitlab.ewi.tudelft/myrepository.git
List staged/unstaged/untracked files	git status	git status
Set email used for commits	git config --global user.email <email>	git config --global user.email myemail@tudelft.nl
Recover old versions of a file	git checkout	git checkout 1234567 README.md

18.2 Appendix B: Commonly used bash commands

Table with commonly used bash commands:

Explanation of bash command	Bash command	Example
List files/folders in current directory including hidden ones	ls -a	ls -a
Create an empty file	touch	touch README.md
Create an empty directory	mkdir	mkdir "util"
Enter a folder inside a directory	cd	cd util
Leave a directory	cd	cd ../
Remove a folder	rmdir	rmdir util
View the contents of a file	cat	cat README.md
Clear the terminal contents	clear	clear
Move a file or directory to a different location. Note that the first argument is location of folder/file to move and the second argument is the new location of it	mv	mv /c/util /d/