**Git Basics**

Contents

[***1.*** ***Introduction*** 5](#_Toc89797978)

[***2.*** ***Prerequisites*** 5](#_Toc89797979)

[*2.1. Installation guide (Windows OS)* 5](#_Toc89797980)

[*2.1.1. Unicredit approach* 5](#_Toc89797981)

[*2.1.2. Personal use* 7](#_Toc89797982)

[*2.2 Installation check* 7](#_Toc89797983)

[*2.3 Tools provided and configuration setup* 8](#_Toc89797984)

[*2.4 Learning repository on Bitbucket* 9](#_Toc89797985)

[***3.*** ***Project setup*** 10](#_Toc89797986)

[*3.1. Cloning* 10](#_Toc89797987)

[*3.2. Initialization from local project* 11](#_Toc89797988)

[*3.3. Link to a folder already tracked* 12](#_Toc89797989)

[***4.*** ***Workflow*** 13](#_Toc89797990)

[*4.1. Commit a new file* 13](#_Toc89797991)

[*4.2. Commit a modified file* 15](#_Toc89797992)

[*4.3. Commit a file with two states* 15](#_Toc89797993)

[*4.4. Restore a tracked file* 16](#_Toc89797994)

[*4.5. Renaming a file* 17](#_Toc89797995)

[*4.6. Deleting a tracked file* 18](#_Toc89797996)

[*4.7. History* 19](#_Toc89797997)

[*4.8. Aliases* 21](#_Toc89797998)

[*4.9. Ignoring files* 22](#_Toc89797999)

[*4.10. Comparisons* 23](#_Toc89798000)

[*4.11. Branches* 24](#_Toc89798001)

[*4.12. Rebasing* 28](#_Toc89798002)

[*4.12.1. Simple(basic) rebase* 28](#_Toc89798003)

[*4.12.2. Abort a rebase* 28](#_Toc89798004)

[*4.12.3. Resolution on rebase conflicts* 28](#_Toc89798005)

[*4.12.4. Continue a rebase* 29](#_Toc89798006)

[*4.12.5. Pull with rebase* 29](#_Toc89798007)

[*4.13. Tags* 30](#_Toc89798008)

[*4.13.1. Lightweight tag* 30](#_Toc89798009)

[*4.13.2. Annotated tag* 30](#_Toc89798010)

[*4.13.3. Tag comparison* 31](#_Toc89798011)

[*4.13.4. Tagging a specific commit* 32](#_Toc89798012)

[*4.13.5. Updating a tag* 33](#_Toc89798013)

[*4.13.6. Publishing a tag* 33](#_Toc89798014)

[*4.13.7. Deleting a tag from remote repository* 34](#_Toc89798015)

[*4.14. Stashing* 35](#_Toc89798016)

[*4.14.1. Simple stash* 35](#_Toc89798017)

[*14.4.2. Apply a stash* 36](#_Toc89798018)

[*14.4.3. Delete a stash* 36](#_Toc89798019)

[*14.4.4. Stash untracked files* 36](#_Toc89798020)

[*4.14.5. Multiple stashes* 37](#_Toc89798021)

[*4.14.6. Stash into branch* 38](#_Toc89798022)

[***5.*** ***Bitbucket features*** 39](#_Toc89798023)

[*5.1. Projects’ view* 39](#_Toc89798024)

[*5.2. Repositories’ view* 39](#_Toc89798025)

[*5.3. Repository’s view* 39](#_Toc89798026)

[*5.4. Branches’ view* 40](#_Toc89798027)

[*5.5. Create branch view* 40](#_Toc89798028)

[*5.6. Cloning (from menu)* 41](#_Toc89798029)

# Introduction

A SCM (source control management or version control) is necessary when you need tracking and managing changes to software code, but any text file can be version controlled.

A SCM is mostly used by software developers and helps software teams work faster and smarter.

Accordingly to the official website Git is a [free and open source](https://git-scm.com/about/free-and-open-source) distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

There are many benefits using Git:

* teamwork
* backup
* compare between revisions of the code
* history and undo changes
* isolation (branches)
* to know who is responsible of a certain change
* active community

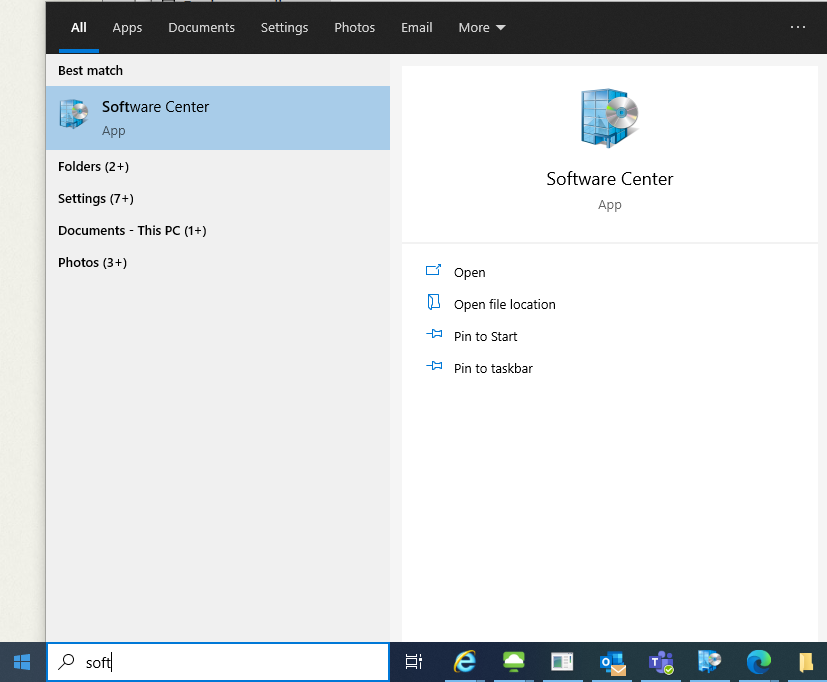
There are multiple Git hosting services. To give some examples I will name Bitbucket[[1]](#footnote-1), GitHub, GitLab etc.

# ***Prerequisites***

## *2.1. Installation guide (Windows OS)*

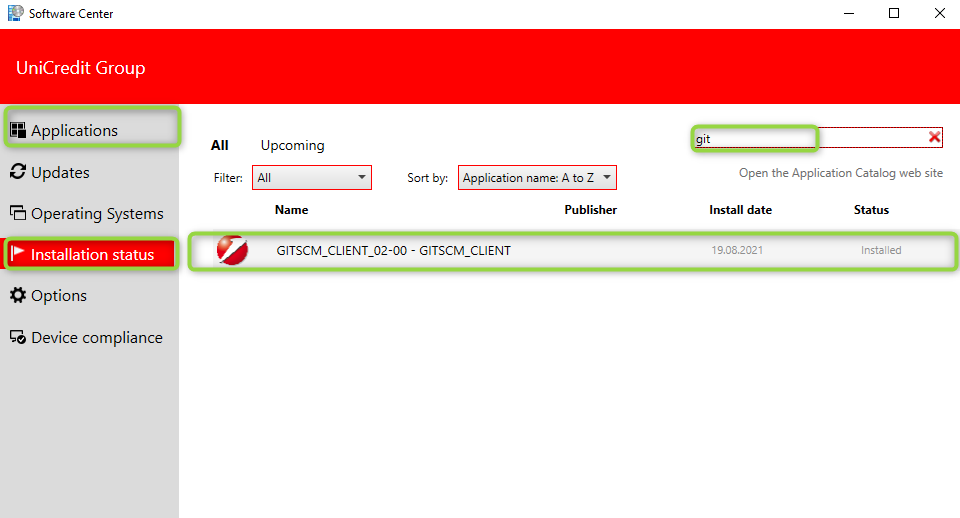
### *2.1.1. Unicredit approach*

On your machine open Software Center.



Once Software Center is opened go to **Applications** tab and search for **‚git’** and install **GITSCM\_CLIENT**

After the software is installed (or it was already installed) it can be found under the **Installation status** tab, having the staus **‚Installed’**.

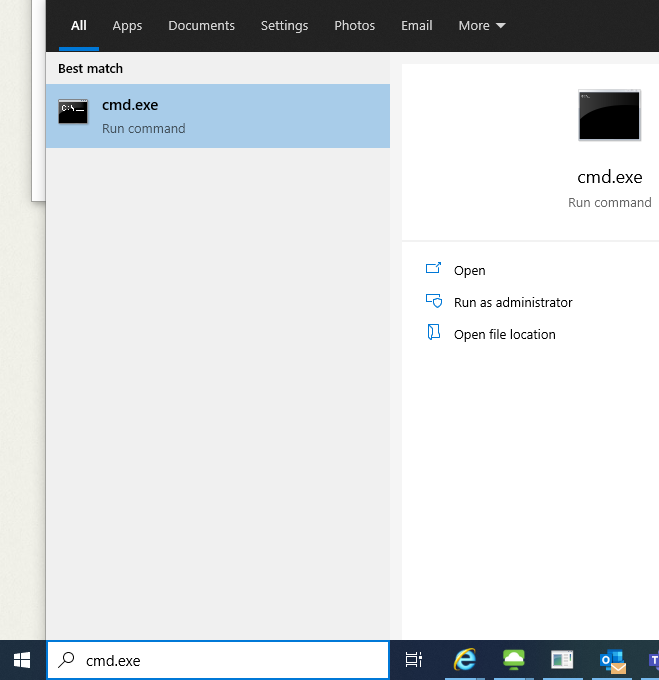


### *2.1.2. Personal use*

On a personal computer, download using a browser the executable file via the following link: [*Git - Downloading Package (git-scm.com)*](http://git-scm.com/download/win)and install the software.

## *2.2 Installation check*

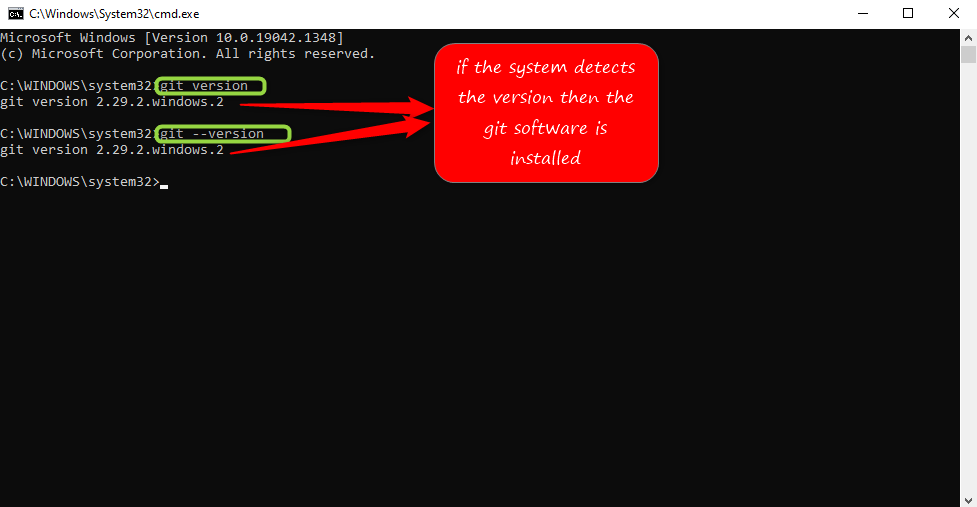
You can check if the installation was successful by opening a command prompt



and running one of the following commands:

**git version**

**git --version**

**

## *2.3 Tools provided and configuration setup*

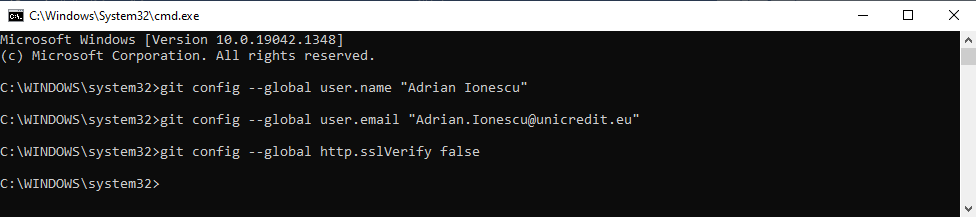
After installation you’ll be able to use the three tools the software provides: **git CMD** (windows command prompt with the git command), **git bash** (command line and stanadard unix commands)and **git GUI** (using git without command line).

You should configure your global username and email address after installing Git.

* **git config --global user.name *"FIRST\_NAME LAST\_NAME"*** (sets your username)
* **git config --global user.email** [***my\_name@example.com***](mailto:my_name@example.com) (sets your email address)
* **git config --global http.sslVerify false** (disable the SSL verification on the global level).

The global configs can be changed at any time.

You can check your global configurations by running git config –-global –-list.

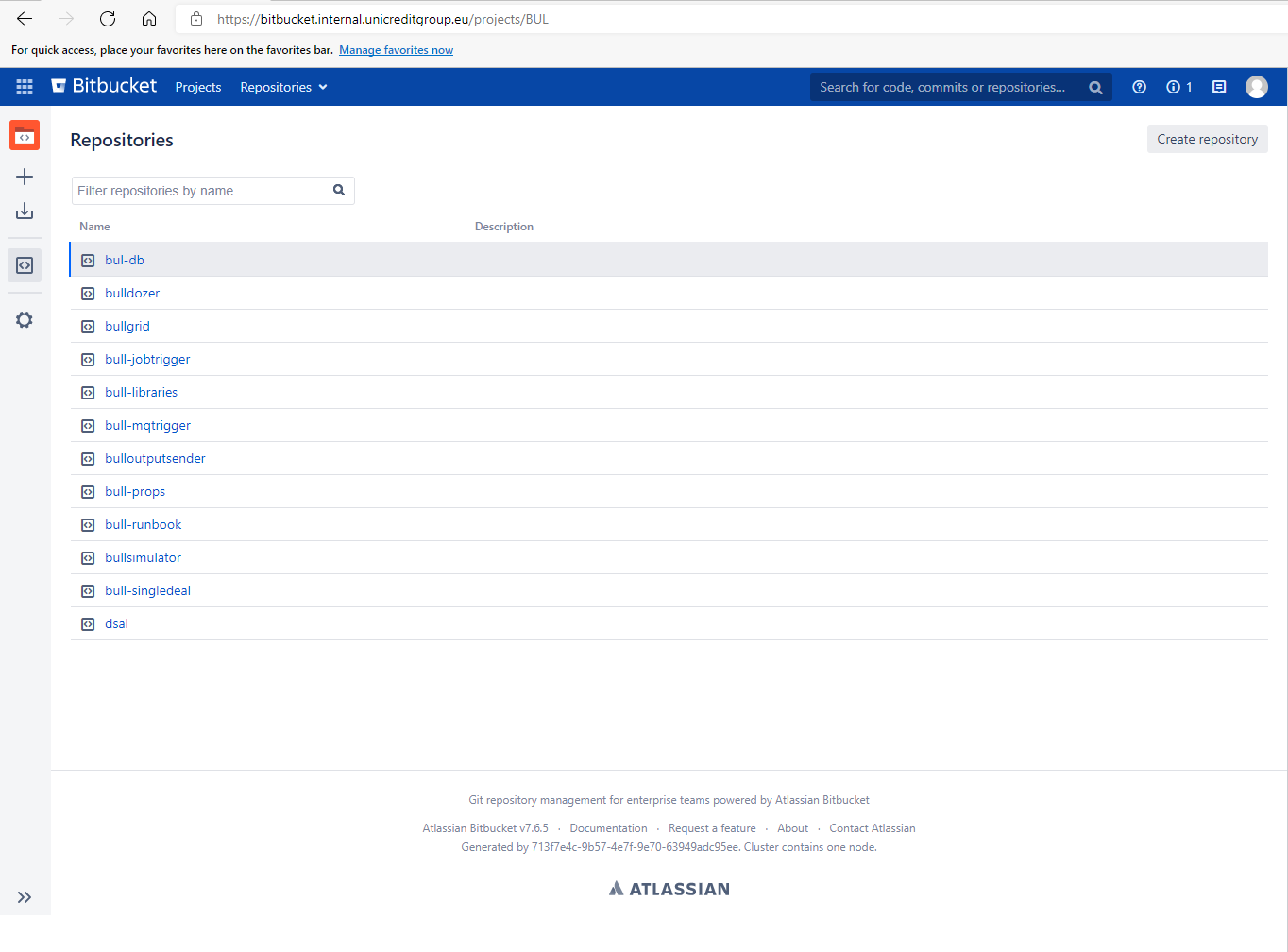


In this course the command line (git bash) will be used because of the numerous benefits:

* powerfull (the GUI implementations does not support all features)
* online support (if you search online the command line is almost always used)
* new features (the new developed features can be immediatelly found on command line)
* consistent (same commands will run on different operating systems)

After we haved been adopted by DevOps programme our git repositories will be host on Bitbucket and you can find several repositories created for BUL application [here](https://bitbucket.internal.unicreditgroup.eu/projects/BUL) (login is required).

Git

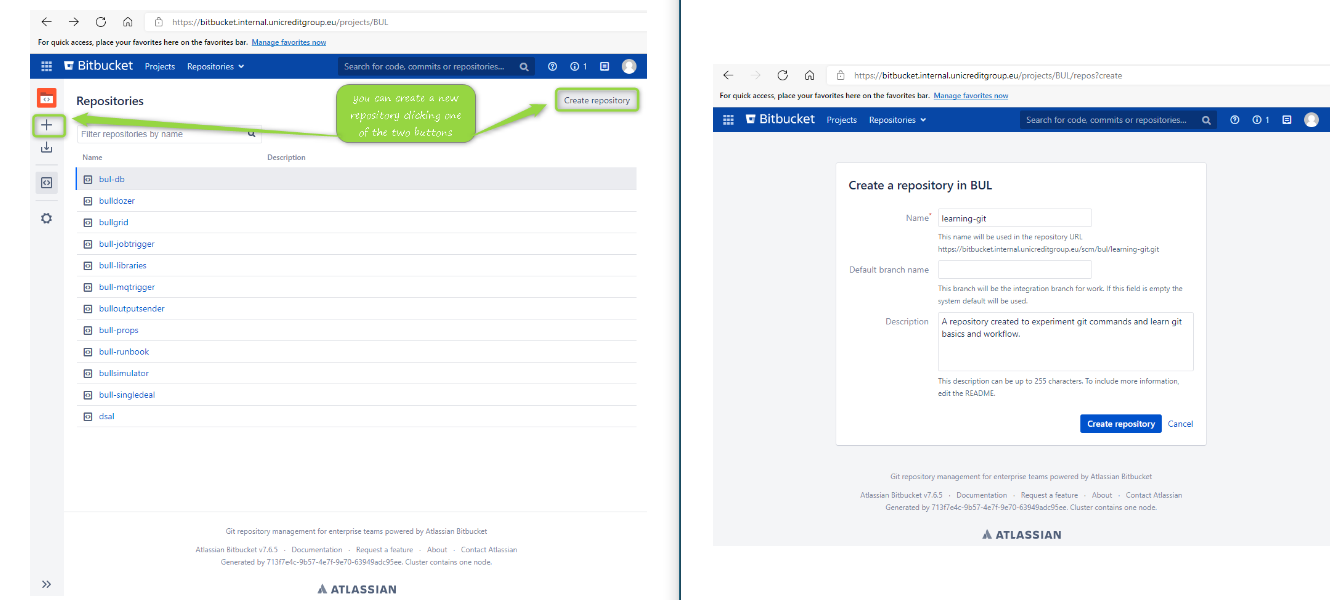


## *2.4 Learning repository on Bitbucket*

In order be able to run git commands throughout this course without affecting the already existing BUL repositories, I will create a new repository[[2]](#footnote-2). This will be possible via web browser.

To create a new repository, navigate to [Repositories view](https://bitbucket.internal.unicreditgroup.eu/projects/BUL) and hit one of the buttons that allows you to create a new repository. You’ll be prompt for a name. The name should be short, self-explanatory, must not contain spaces, characters are in lower space and words are separated by dashes. The description is not mandatory but can help.

This repository will be further referred as the **remote repository**.



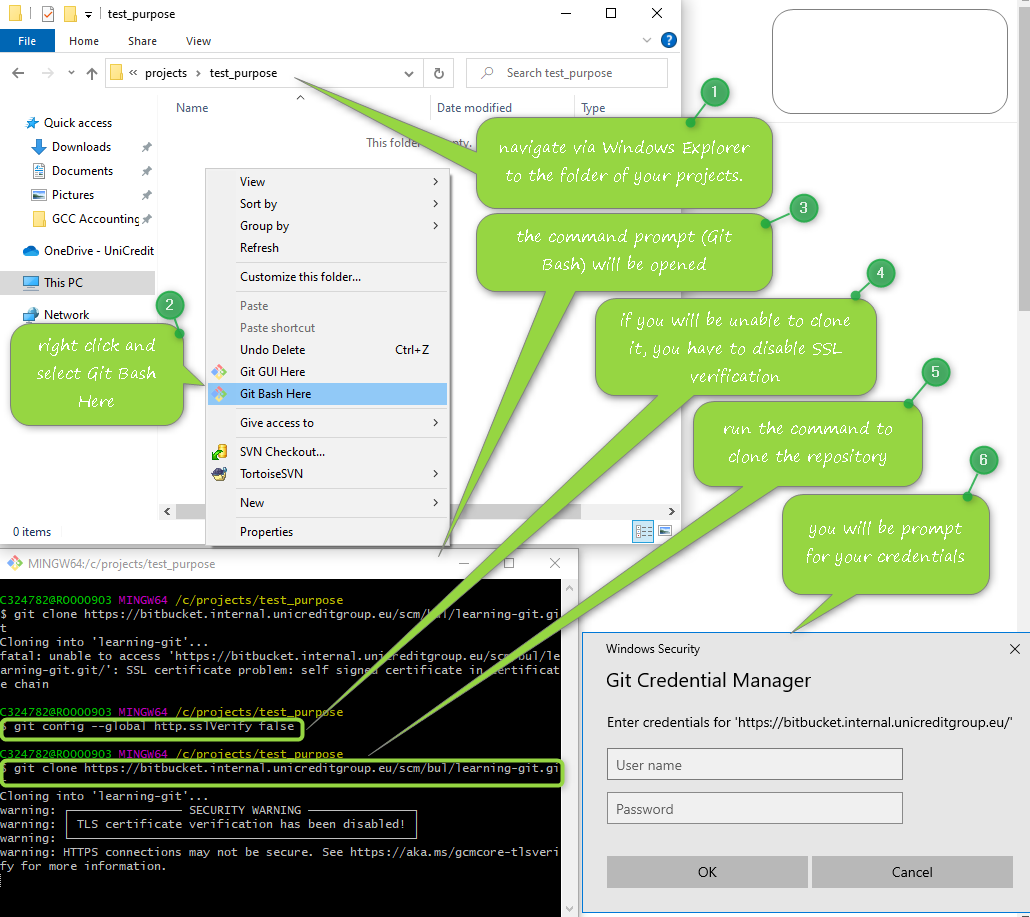
# ***Project setup***

## *3.1. Cloning*

A new repository named ***learning-git*** was created and can be found [here](https://bitbucket.internal.unicreditgroup.eu/projects/BUL/repos/learning-git/browse)[[3]](#footnote-3).

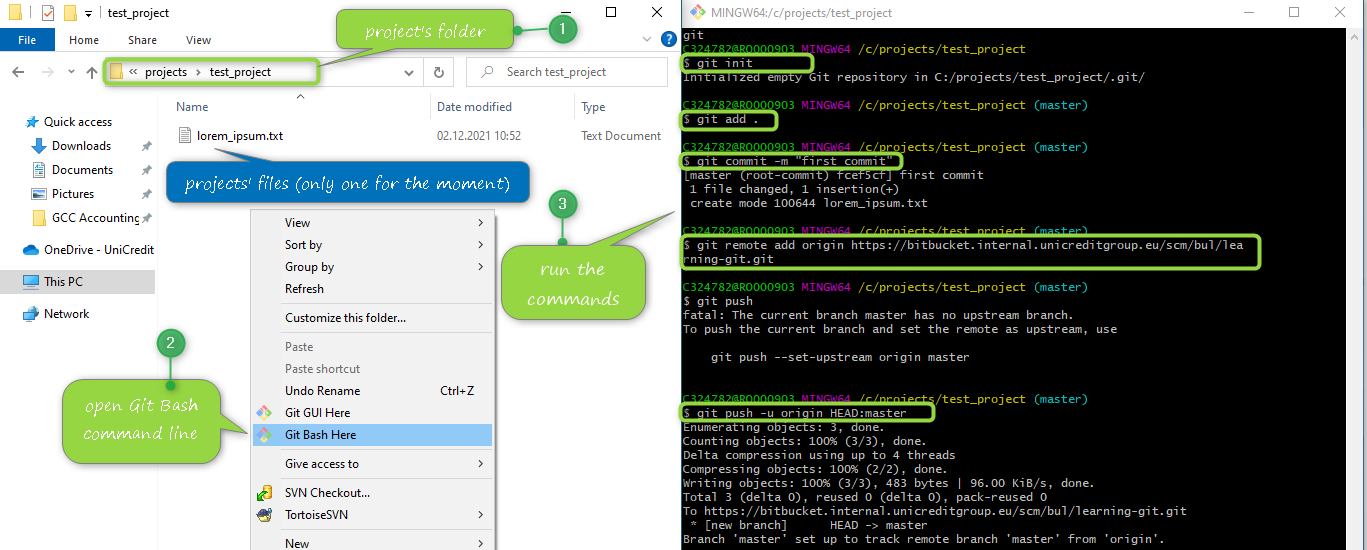
Once the new repository is created you can:

* + clone it locally via SSH or HTTPS (if the project already exists). By cloning you will be able to get a copy of the project on your local machine. This repository will be further called as **local repository.**
* **git clone** [***https://bitbucket.internal.unicreditgroup.eu/scm/bul/learning-git.git***](https://bitbucket.internal.unicreditgroup.eu/scm/bul/learning-git.git)



## *3.2. Initialization from local project*

* link it to a folder on your machine
* **cd** ***folder-of-an-existing-project*** (or navigate to project’s folder via Windows Explorer and open GitBash)
* **git init** (uses the current folder as working directory, **.git** folder will be added at this step)
* **git add –all** (or **git add .**) – to add all the files to the staging area
* **git commit -m** ***"Some defining message for the commit”***, eg: Initial commit – to send the files to the local repository (-m “” adds an inline commit message)
* **git remote add origin** ***<https://bitbucket.internal.unicreditgroup.eu/scm/bul/learning-git.git>*** - will create 'origin' if it doesn't exist
* **git push -u origin HEAD:master** (or **git push --set-upstream origin master**)

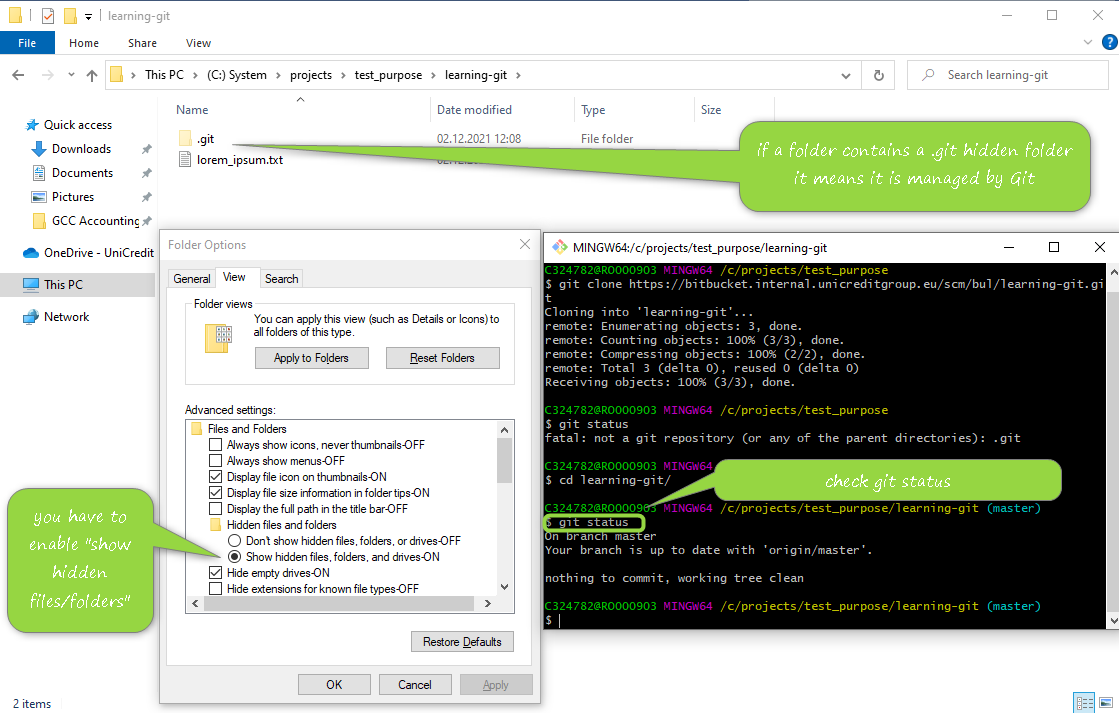


## *3.3. Link to a folder already tracked*

* link it to a folder already tracked by Git
* **cd *folder\_of\_an\_existing\_project*** (or navigate to project’s folder via Windows Explorer and open Git Bash)
* **git remote set-url origin https://bitbucket.internal.unicreditgroup.eu/scm/bul/learning-git.git**
* **git push -u origin --all**
* **git push origin –tags**

After you clone the project, you can ask Git about the status of the project. And it will tell you that:

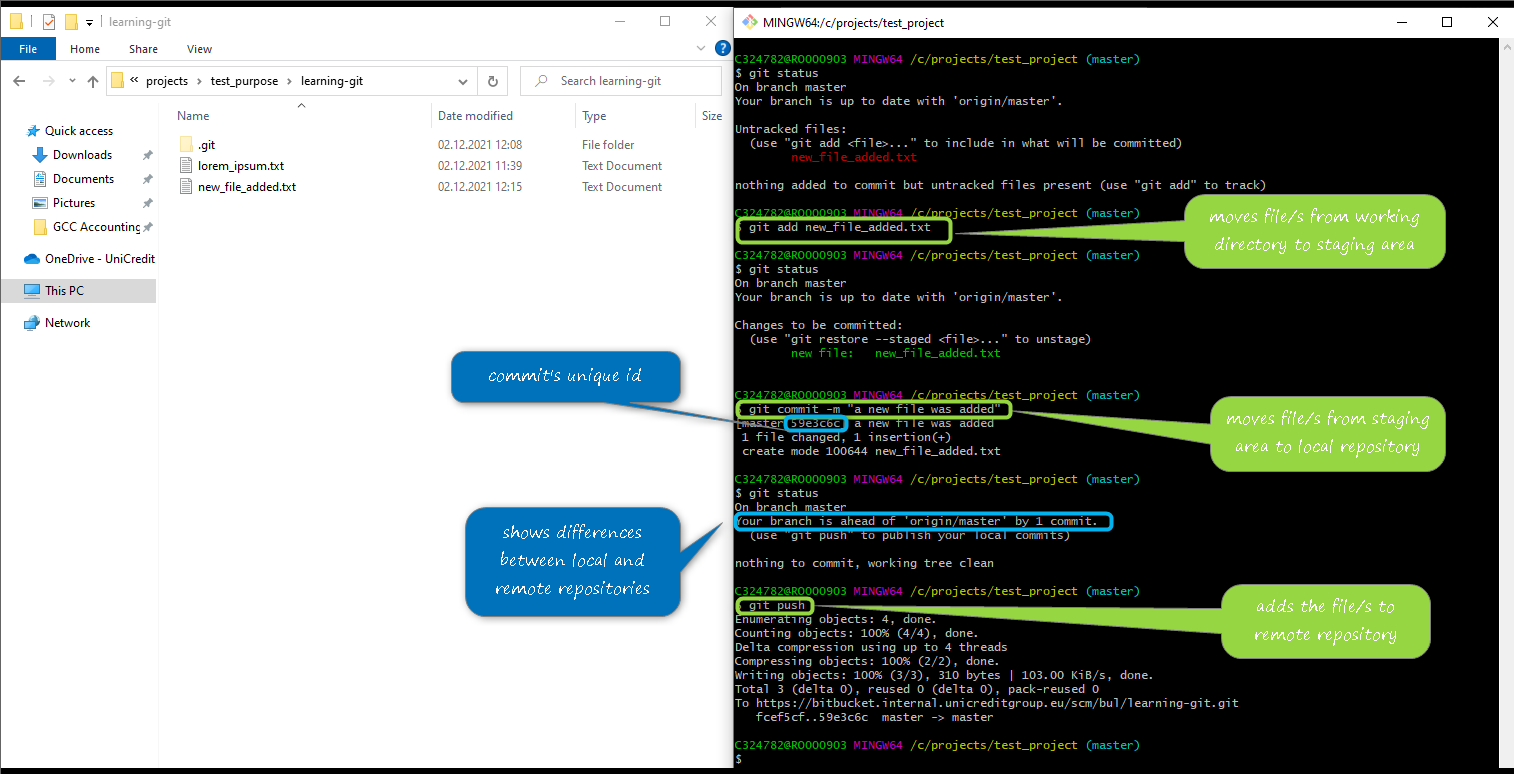
* you are on the **master** branch, which is by convention, the default branch for a Git repository.
* you are up to date with **origin/master**. When the **git clone** command was executed, a relationship was automatically setup with the repository on Bitbucket. This reference’s name is **origin**
* working directory is clean. The working directory is where you do all the work on the local machine, directory which is managed by Git
* a **.git** hidden folder is created. This means that the project’s folder and all its subfolders will be managed by Git.



# ***Workflow***

## *4.1. Commit a new file*

If you add a new file to you project managed by git and the Git status is verified you will see that an untrack file is displayed. This file is found inside working directory and you can tell Git about it by running **git add *name\_of\_the\_file*** or **git add –all** or **git add .** (the last two commands add all files to staging are). You can send the file/s to local repository by running **git commit -m *“a message explaining the commit”***. At this step the file/s cannot be seen on web browser and for that the **git push** must be executed. This will send/publish the modifications on **remote repository**.

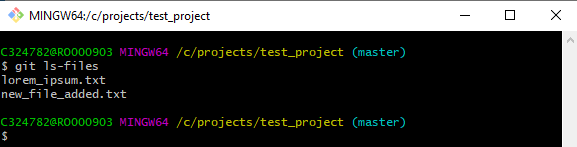


Git workflow

|  |  |  |  |
| --- | --- | --- | --- |
| *Basic Git workflow* | | | |
| Local | | | Remote |
| Working directory | Staging area | Local repository  (.git folder) | Can be accessed via web browser |
| A directory on your local machine that holds all the files of your project. It is not mandatory that all files to be managed by Git, but Git knows about them.  Where the new, deleted or edited file/s can be found. To send them to the next state run the git add file\_name | Also known as git index and stores the changes for the next commit.  Before committing them, these files can be moved back to working directory without affecting the history of the repository.  Run git commit -m “a message” to publish the file/s to the local repository | Manages the commit history.  Until you run git push you will not be able to see the file/s on the remote repository. |

## *4.2. Commit a modified file*

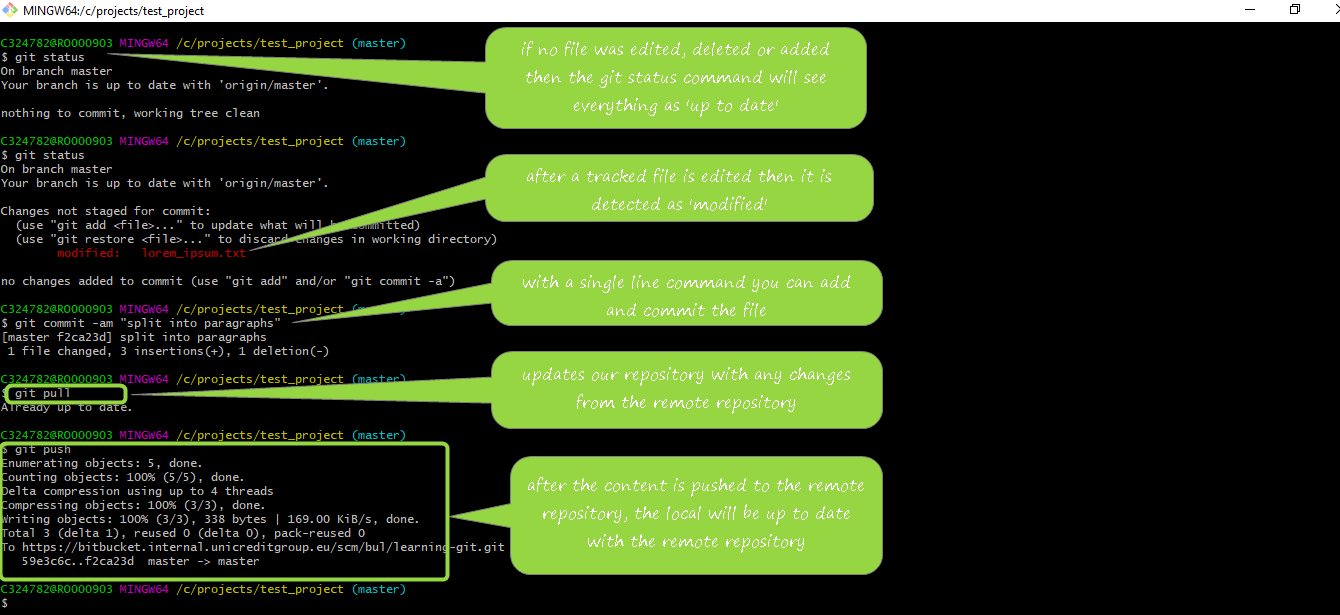
A tracked file is a file any file that was added to the Git index (staging area) or that is already committed to the Git repository. You can check any tracked files by the command **git ls-files**. You can see below all the files being tracked by the current repository.



Just adding a new file inside the working directory will not be seen as a managed file by the Git. You must add it to the staging area.

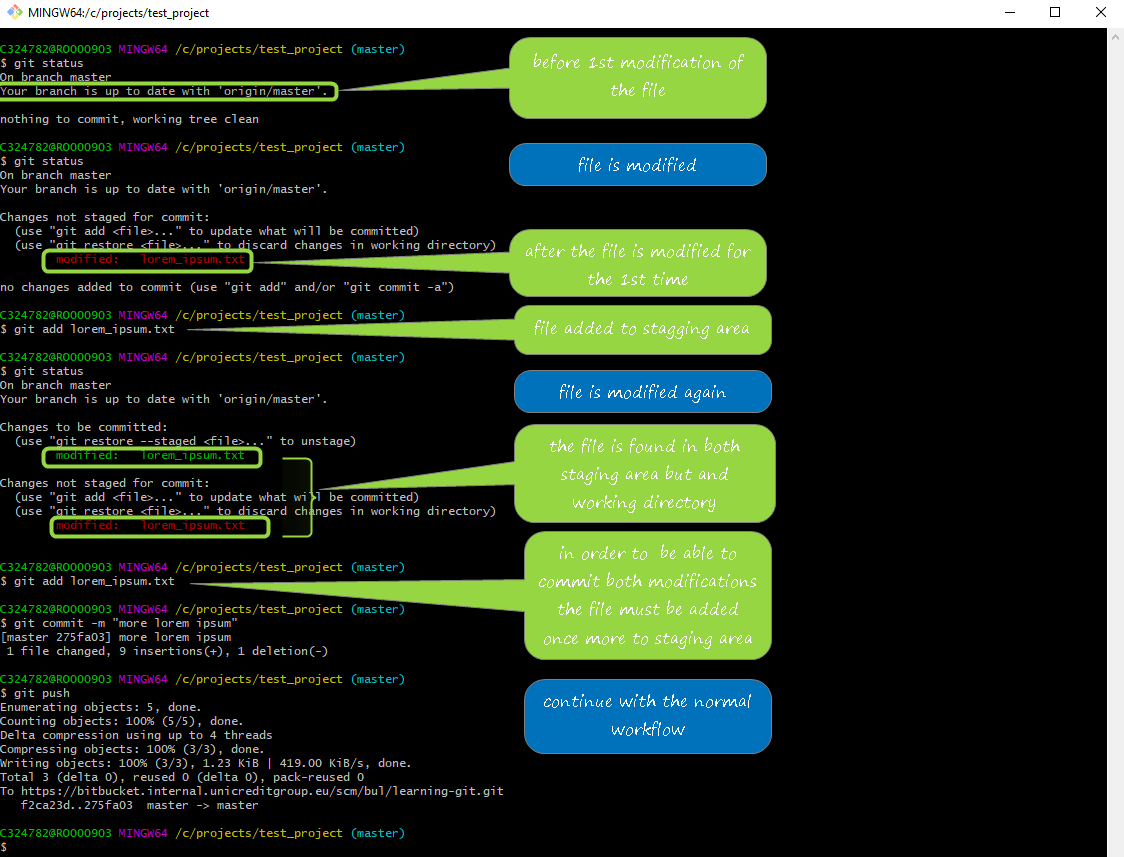
If a tracked file is modified, then you have to follow the git workflow:

* **git add *name\_of\_the\_edited\_file***[[4]](#footnote-4)
* **git commit -m *“an explanatory message for the changes”***
* **git pull** (always perform the pull command before pushing to the remote repository)
* **git push**



## *4.3. Commit a file with two states*

After a tracked file is modified it can be added to the staging area (**git add *name\_of\_file****)*. In the situation that the file was added to the staging and modified once more then the **git add *name\_of\_file*** must be run again.



## *4.4. Restore a tracked file*

After a file is modified and is added to staging area you can:

* back out it to the working directory and/or
* remove all its modifications



## *4.5. Renaming a file*

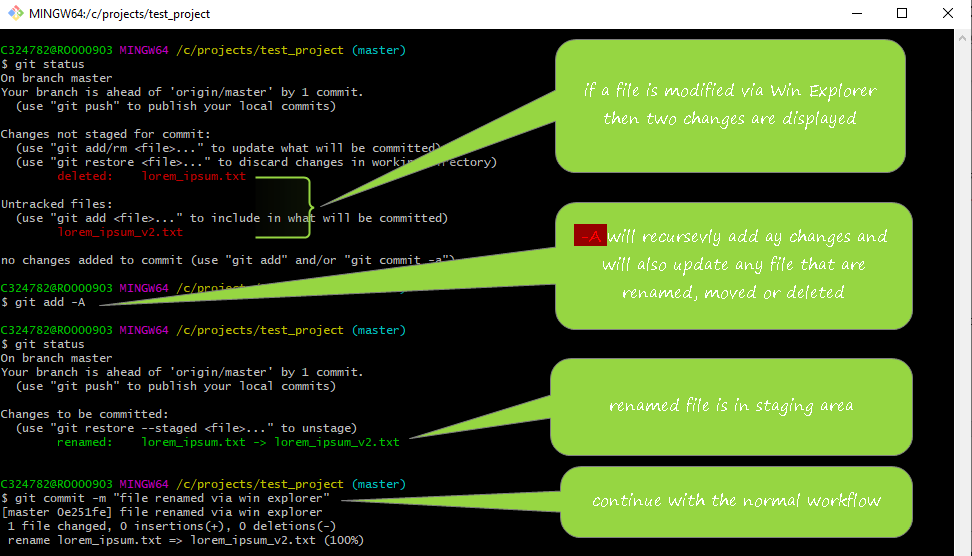
You can rename[[5]](#footnote-5) a file via Windows Explorer or by a Git command: **git mv** ***name\_of\_the\_file new\_name\_of\_the\_file***. This command can be used

* to rename file,
* to move the files to a different folder,
* to back out a file renaming.



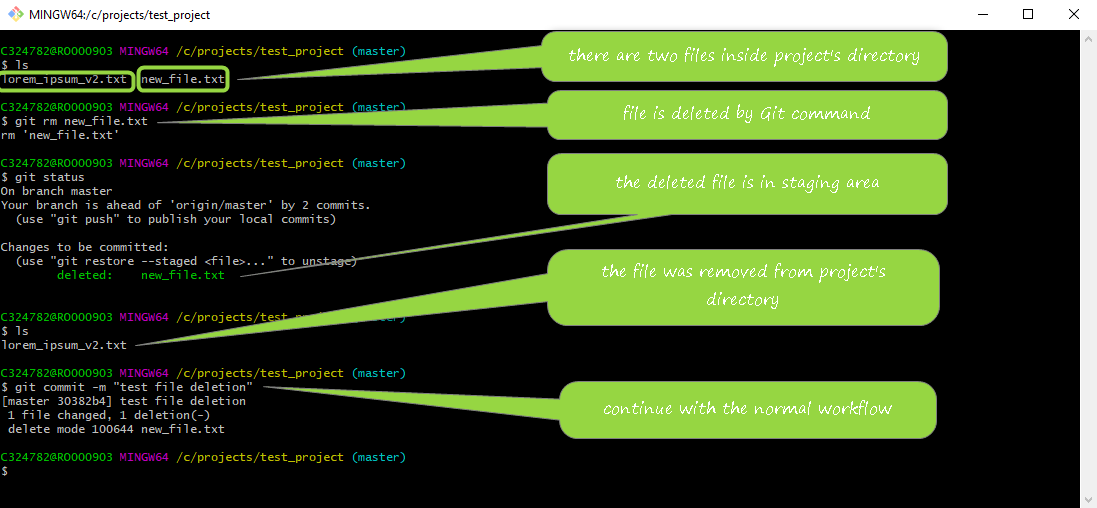
If you rename the file via Windows Explorer, then the git will detect two separate operations:

* a deleted file (the old name)
* an untracked file (the new name)



## *4.6. Deleting a tracked file*

You can delete a tracked file by the following command: **git rm *name\_of\_the\_file***

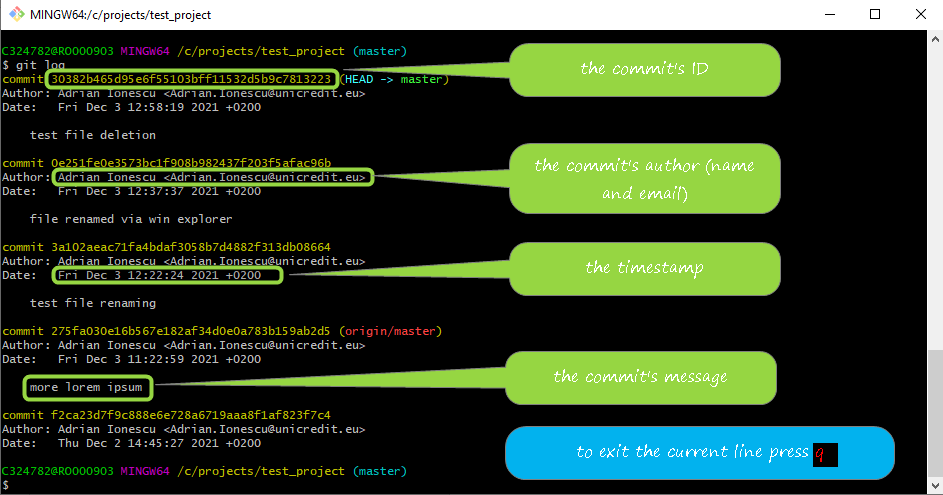
**

If you want to restore a deleted file after you used the **git rm** command

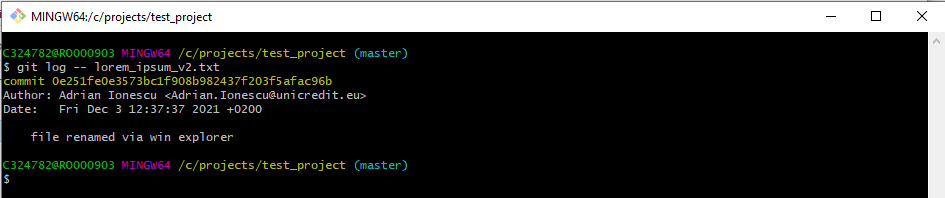


## *4.7. History*

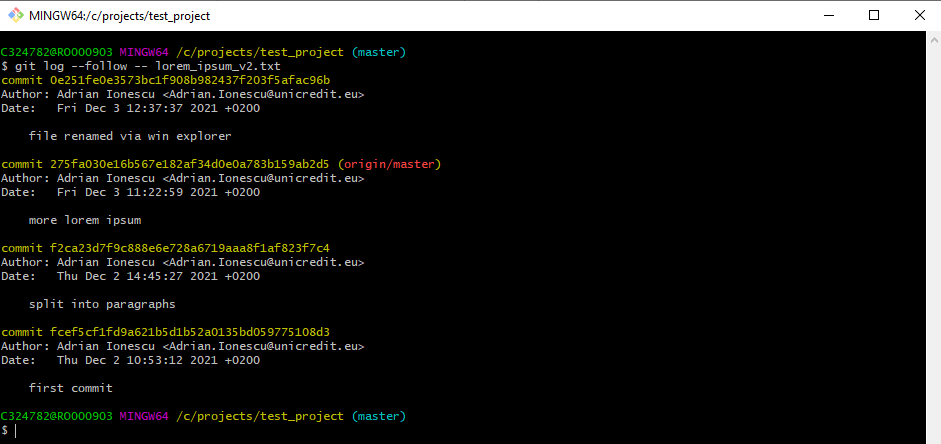
* **git log** will show all the commits made for the branch in reverse chronological order



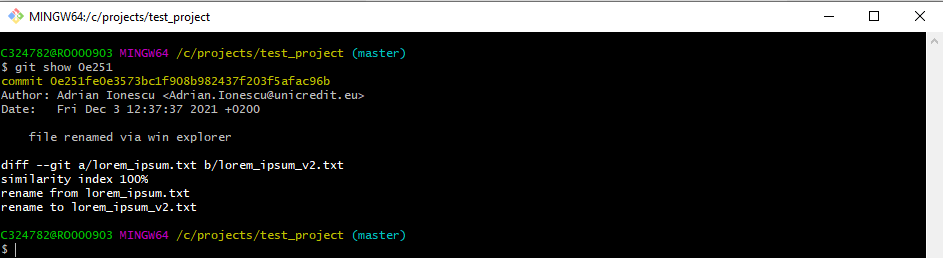
* **git log –-abbrev-commit** will show the commit ID abbreviated
* **git log -–oneline –-graph –-decorate** display the history compressing the infos into one line(**--oneline**), depicts the branching graph(**--graph**) and adding the labels, tags, annotations, etc. **(--decorate)**
* **git log *commit\_ID*…*other\_commit\_ID*** will display only the commits between the specified ones
* **git log –-since=*”2 days ago”*** – displays commits made in a period of time(eg: past 2 days)
* **git log –- *file\_name*** – display the history of a certain file



* **git log --follow –-** **file\_name** – displays the history of a specific file going through the renames



* **git show *commit\_ID*** (can be abbreviated) show the detailed infos of a specific commit

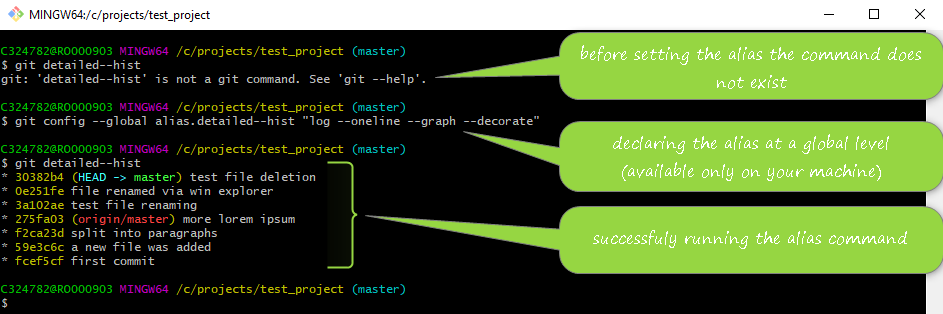


* **git log help** will show all the **git log** commands

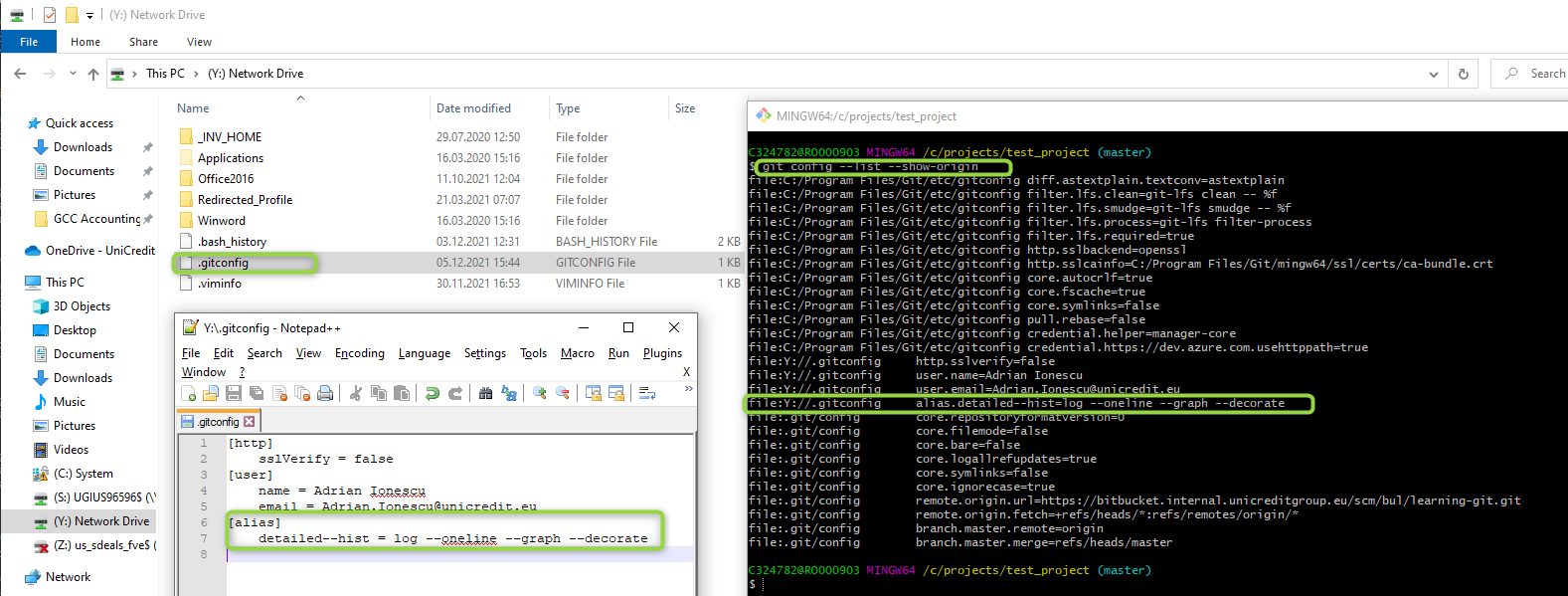
## *4.8. Aliases*

Supposing you are often using a very complex command, like the one used to display the history of the commits in one line (**git** **log -–oneline –-graph –-decorate**). Git offers a feature the give an alias to a specific command:

**git config –-global alias.*detailed--hist “log -–oneline –-graph –-decorate”***

****

The aliases are stored in the **.gitconfig** file. To see where the **.gitconfig** is stored you can use the following command: **git config --list --show-origin**. You can edit the configuration file to configure an alias.

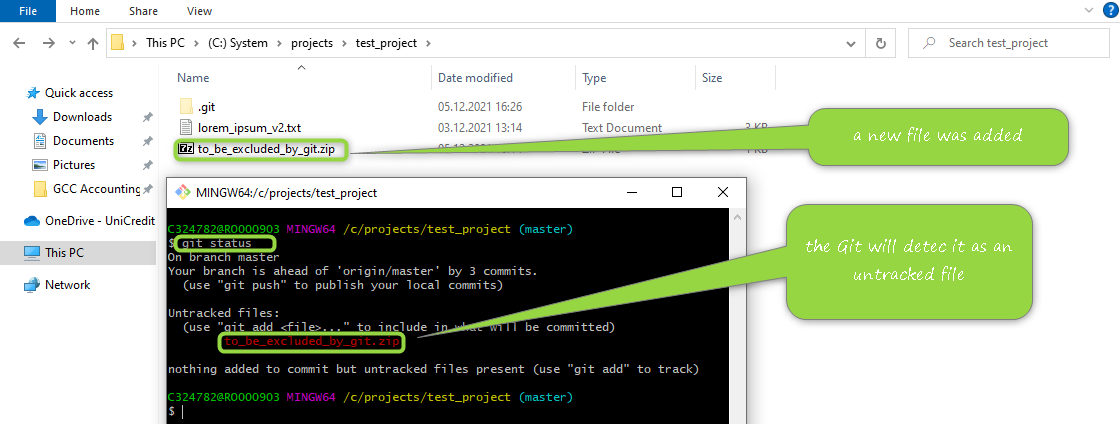
****

## *4.9. Ignoring files*

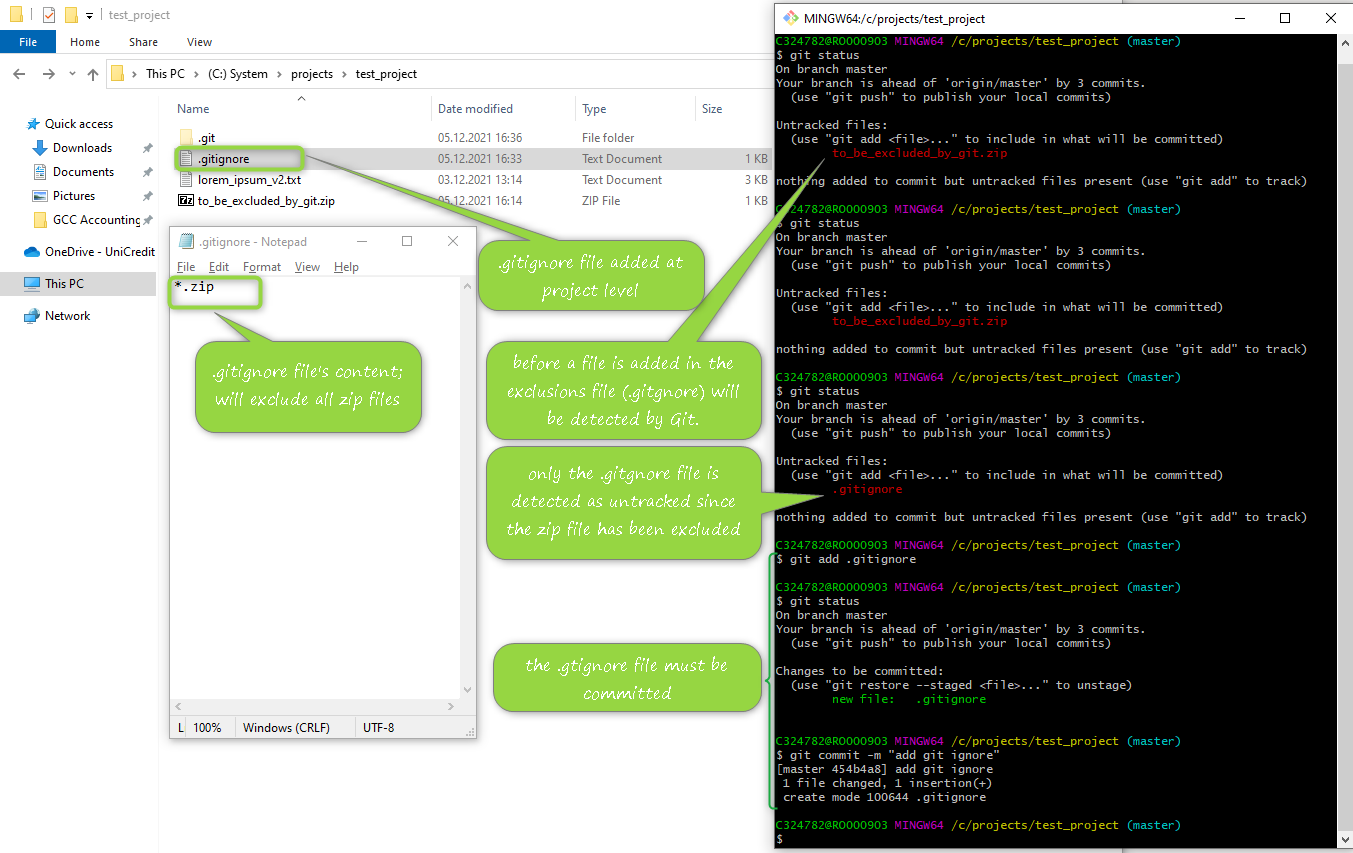
If, for any reason, you want to exclude some files or folders from being managed by git it is necessary to create a text file (without extension) named **.gitignore**. This file will be stored inside project’s folder and may contain:

* **name\_of\_a\_file**, e.g.: to\_be\_excluded\_by\_git.zip
* **a file pattern**, e.g.: \*.zip
* **a folder**, eg: my\_excluded\_folder/

Each instruction will be placed in a separate line.



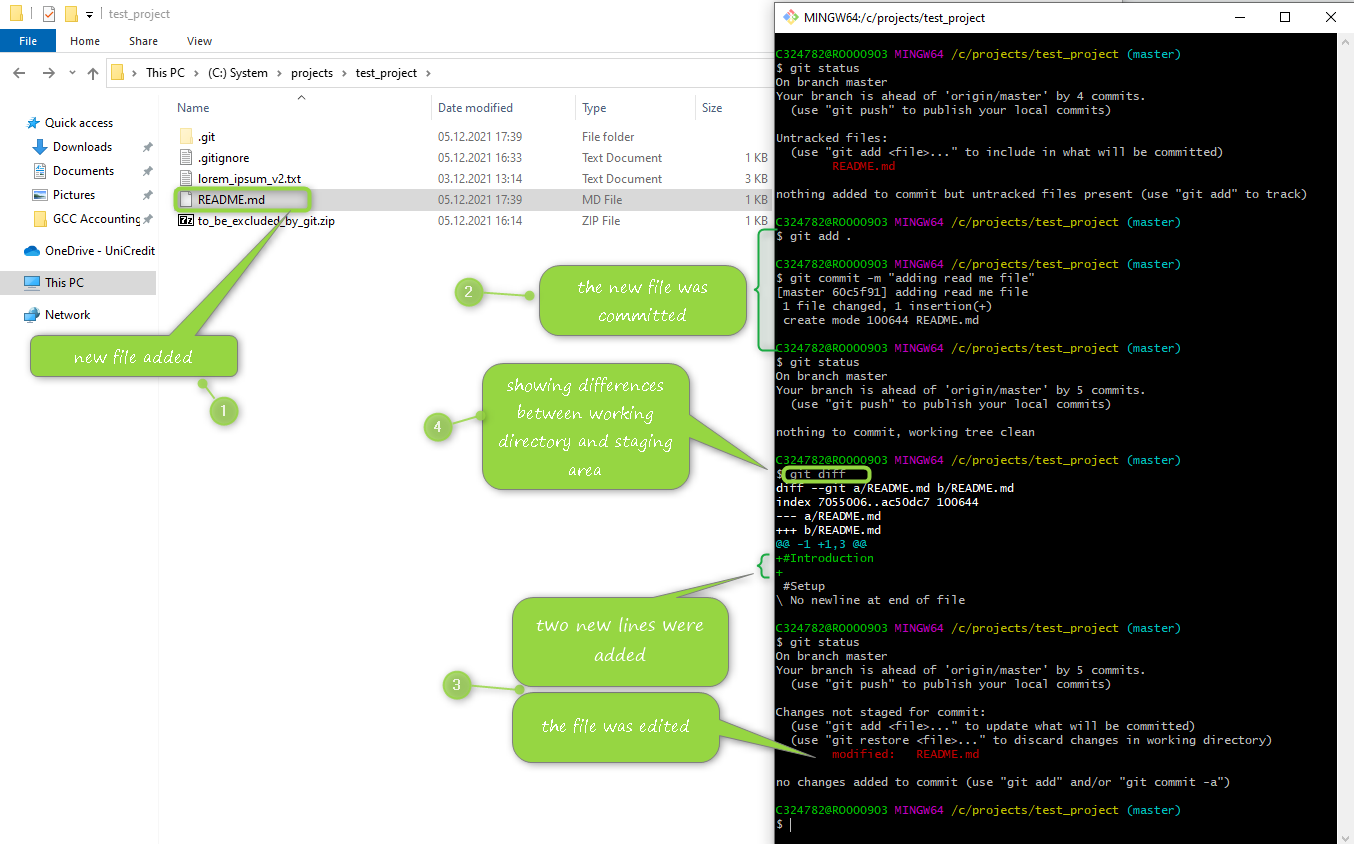
After the **.gitignore** file is created it must be committed and will be available at project level.



## *4.10. Comparisons*

Working directory vs Staging Area

* **git diff** command will display differences between working directory and staging area
* **git diff -– name\_of\_the\_file** - will display differencesfor the specified file

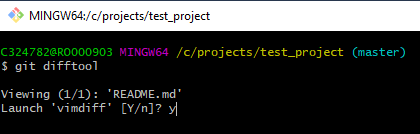


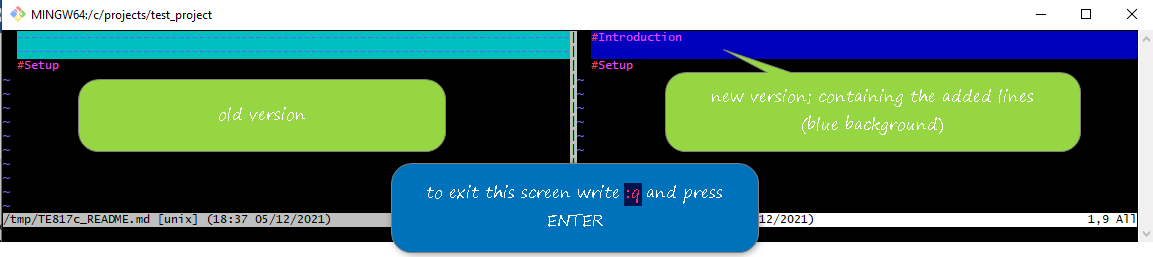
As you can see it is very hard to distinguish between what was modified especially if there are many modifications.

To proper compare and check differences between two versions of the same file two new configurations must be set:

* **git config –-global diff.tool vimdiff** - allows to better compare versions of the same file
* **git config –-global merge.tool vimdiff** – allows to solve conflicts and merge after the conflicts were solved

Now you can run **git difftool** command and a new screen will be shown.



****

Working directory vs Local Repository

To compare differences between the working directory and the branch’s last commit you have to run the **git diff HEAD** command.

Staging Area vs. Local Repository

**git diff --staged HEAD**

Commits comparison

* **git diff commit\_ID other\_commit\_ID** (both commit IDs can be abbreviated) - Will display all the differences between all the changes that happened between specified commits.
* **Git dif HEAD HEAD^** - will compare last commit (HEAD) and the second last commit (HEAD - 1)

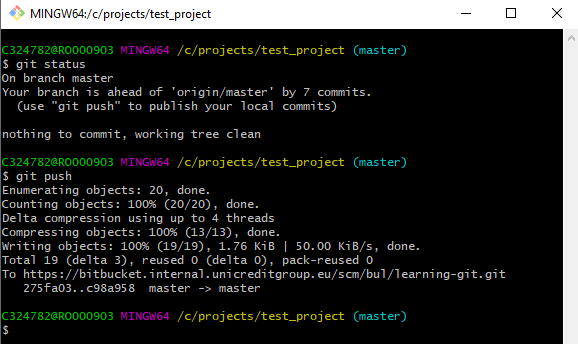
**Branches comparison**

Since there are no branches created the following command will compare the Local Repository vs Remote Repository:

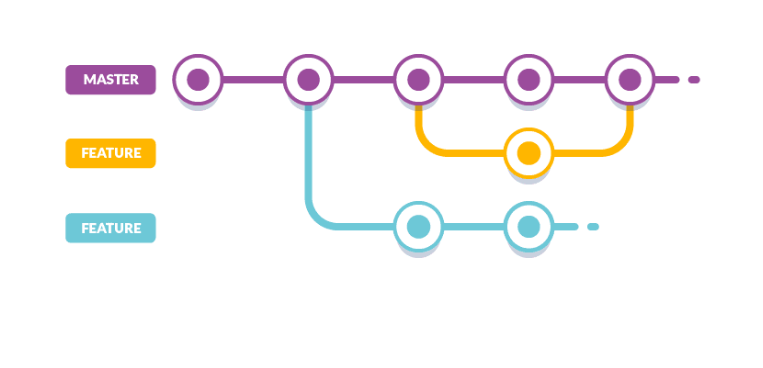
**git diff master origin/master** – master is the local repository and origin/master is the repository on Bitbucket

## *4.11. Branches*

You do not have to publish every commit immediately after is made. As you can see in the picture below 7(seven) commits were published at once.



Until now, the **master** branch was used to describe several Git features, but this is not recommended. Instead of using the master branch you have to use so called feature branches.



To list all the branches, you can run the following command: **git branch -a**.

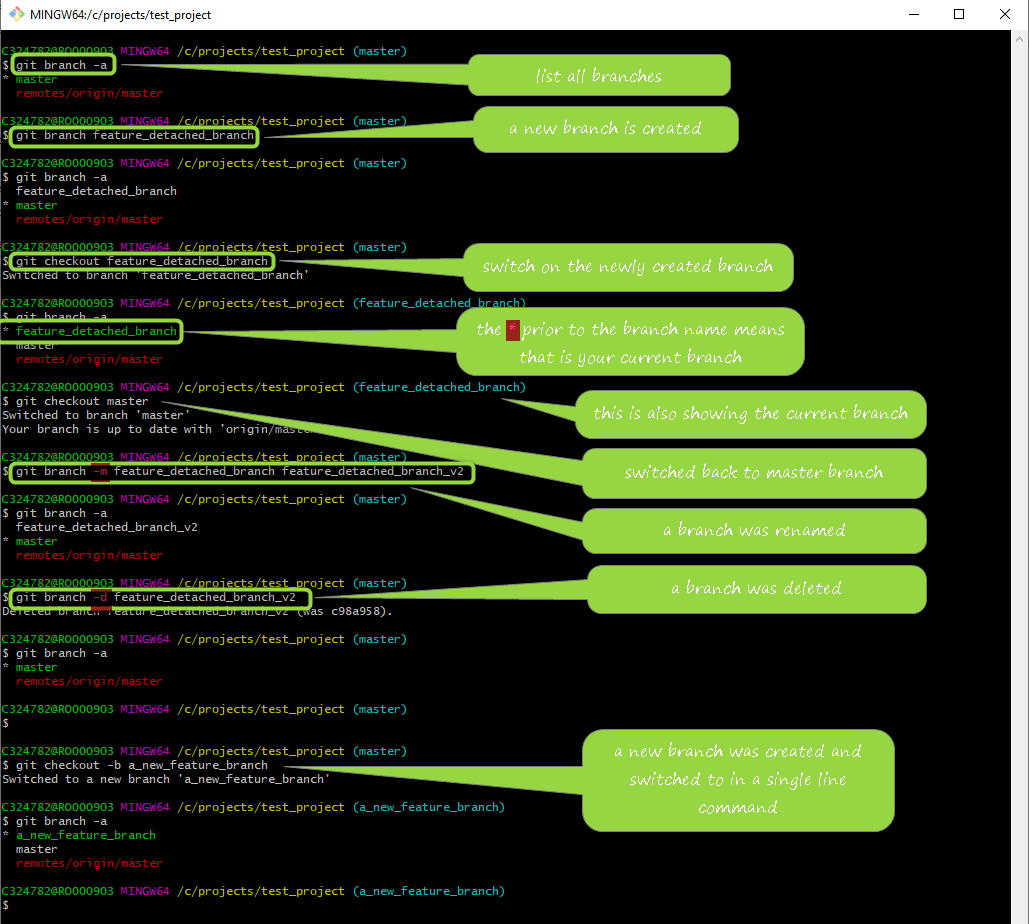
To create a new branch, **git branch *the\_name\_of\_the\_new\_branch*** should be used.

To switch to the new branch: **git checkout *the\_name\_of\_the\_branch***.

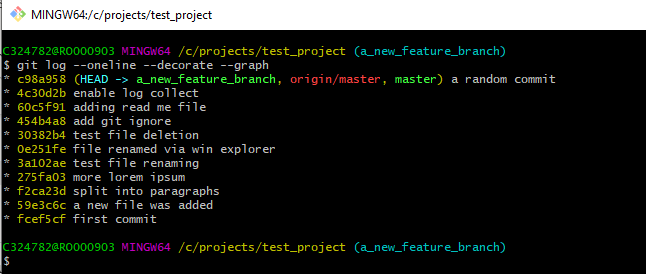
But if you switch on a new branch once is created you can use **git checkout -b *the\_name\_of\_the\_new\_branch***.

To rename a branch: **git branch -m *branch\_old\_name branch\_new\_name***, where **-m** means move.

To delete a branch: **git branch -d *branch\_name***, where **-d** means delete. But be aware that you cannot delete a branch you are current on. You can use -D to delete a branch that is not fully merge: **git branch -D *the\_name\_of\_the\_branch***.



As you can see below, the last commit has several labels. Since there is no commit on a different branch all the labels point to the same commit.



TODO add explanations about HEAD, master, origin/master etc

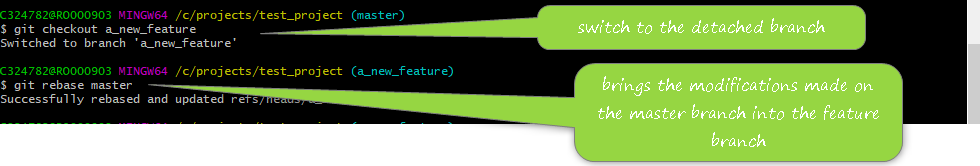
## *4.12. Rebasing*

### *4.12.1. Simple(basic) rebase*

In the scenario where you create a new branch, modify and commit something there and you switch back on master and modify and commit something also on master you can use rebase whenever you did not finish the work on the detached branch.

**

The rebase is used whenever you want to incorporate any changes done on master branch into the detached branch and to simplify the merge procedure.

**

The command is**git rebase *the\_name\_of\_the\_branch***.

### *4.12.2. Abort a rebase*

**git rebase --abort**

### *4.12.3. Resolution on rebase conflicts*

*Check merge conflicts resolution section*

### *4.12.4. Continue a rebase*

After you abort a rebase because of some conflicts, you can continue with the rebase after you solve the merge conflicts:

**git rebase -–continue**

### *4.12.5. Pull with rebase*

If a colleague has published some modifications and you want to benefit of those modifications, you can use the following command:

**git pull –-rebase *the\_name\_of\_the\_branch*** (e.g.: **git pull –-rebase origin master**).

## *4.13. Tags*

### *4.13.1. Lightweight tag*

Tags are labels that can be applied on any commit. The tagging system is mainly used to mark milestones in the repository.

To check which are the tags run the following command: **git tag --list**.

To create a tag (lightweight tag) use **git tag** ***the\_name\_of\_the\_tag***.

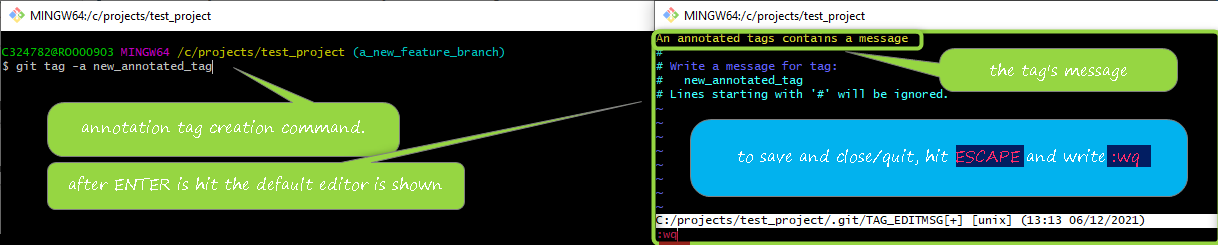
The command to display a commit link to a tag is **git show** ***the\_name\_of\_the\_tag***.

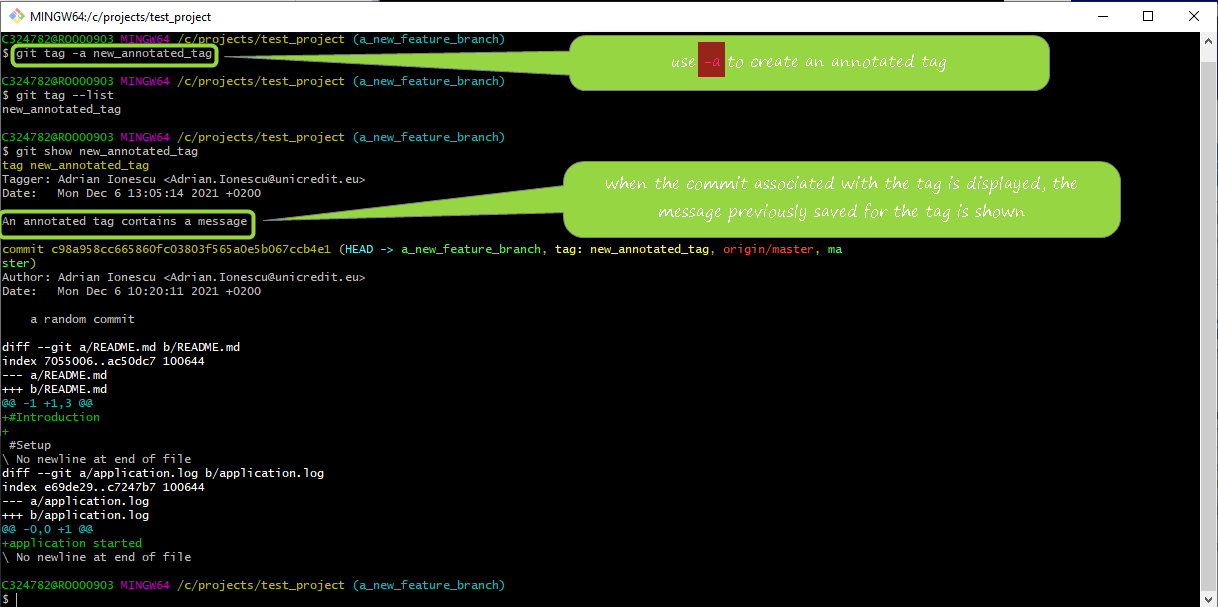
If you want to get rid of a tag the command is **git tag --delete** ***the\_name\_of\_the\_tag***.



### *4.13.2. Annotated tag*

To create an annotated tag there is another command: **git tag -a** ***the\_name\_of\_the\_tag***, where -a means annotated. When an annotated tag is created the default editor is displayed and you have to write down the tag message.





### *4.13.3. Tag comparison*

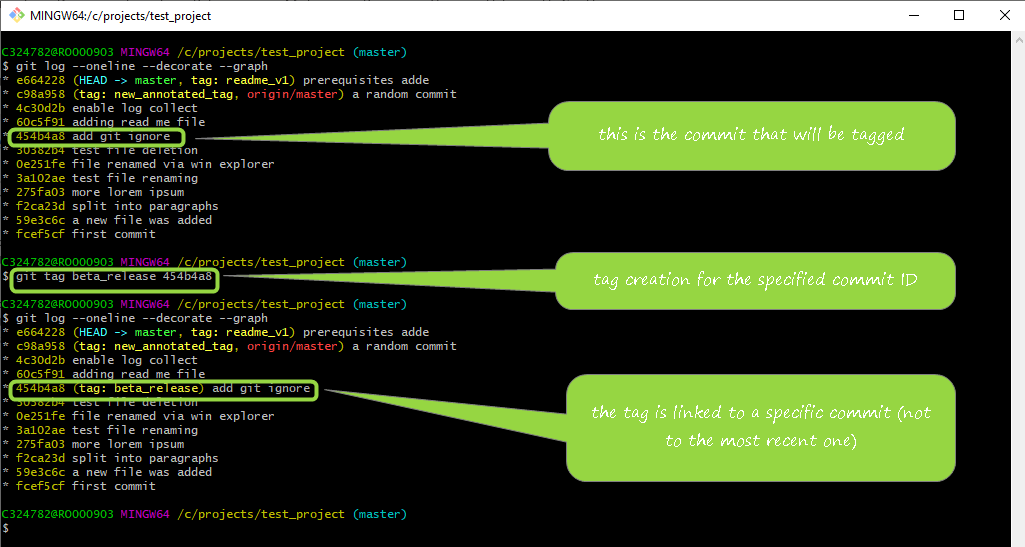
The command used is: **git diff *tag\_name other\_tag\_name***



### *4.13.4. Tagging a specific commit*

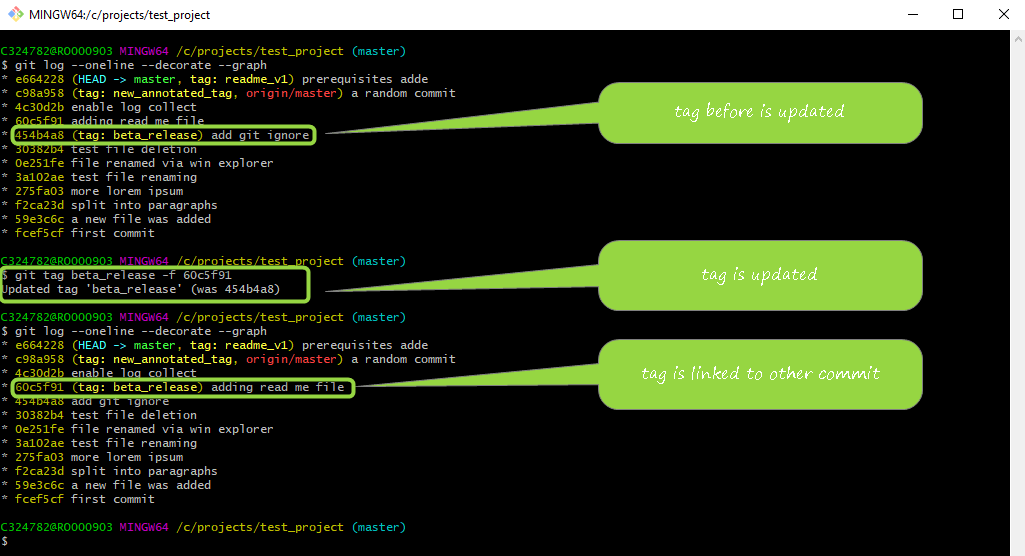
If you want to tag a commit that was made in the past use the following command:

**git tag *name\_of\_the\_tag the\_commit\_ID*** (can be abbreviated).



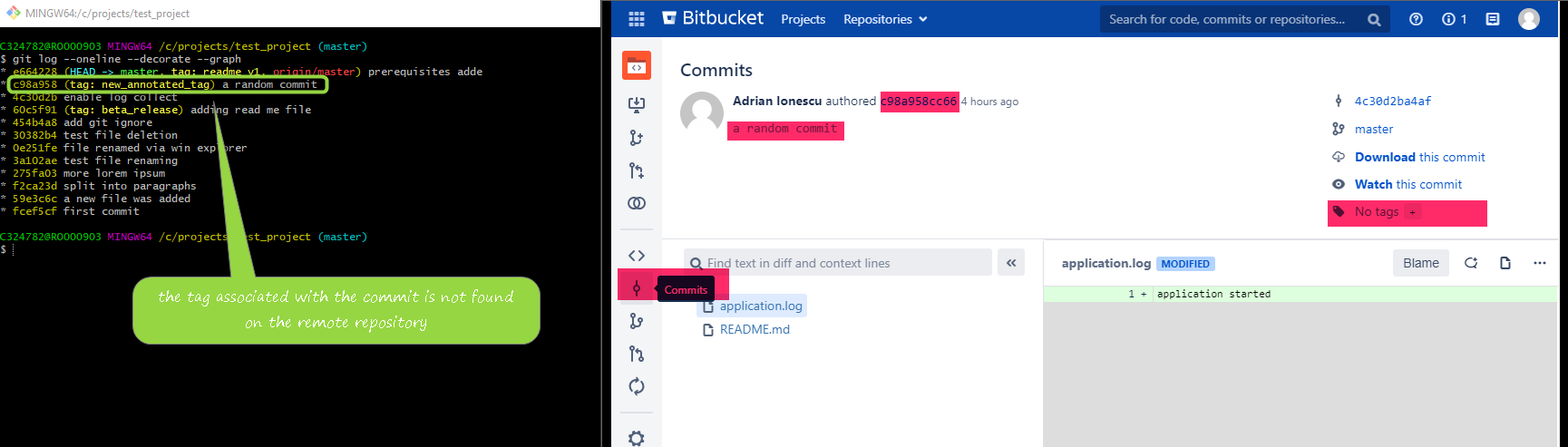
### *4.13.5. Updating a tag*

In the case you want to move a tag from a commit to a different commit you can, of course, to delete the tag and re-create it for the other commit. If you do not want to use this approach there is a command to do it: **git tag *name\_of\_the\_tag* -f *the\_commit\_ID***, where -f means force.

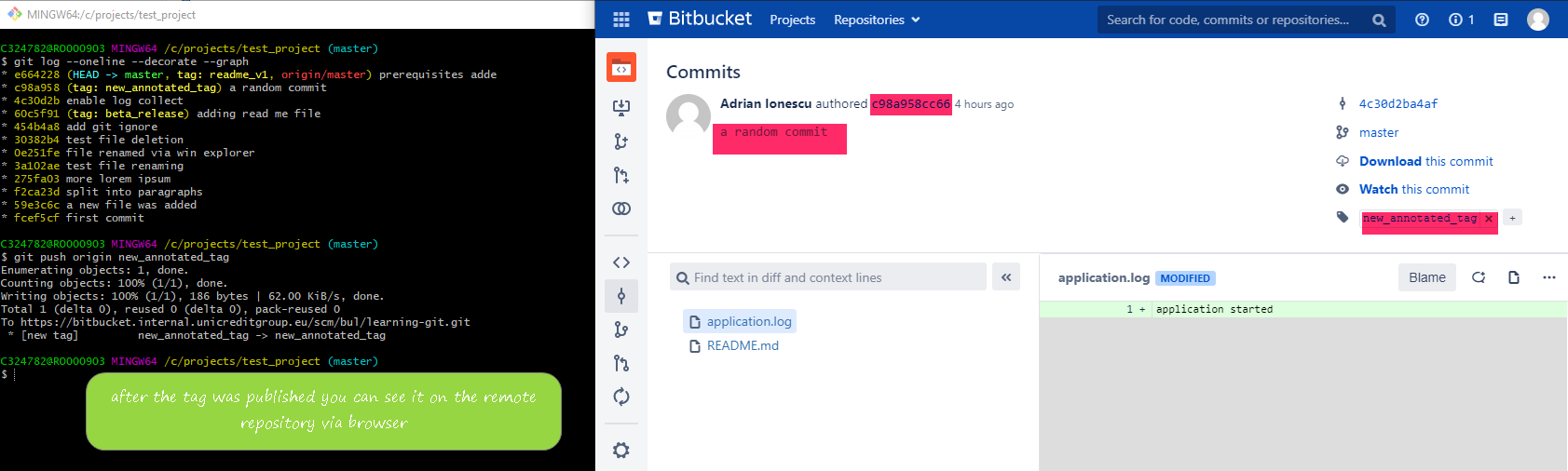


### *4.13.6. Publishing a tag*

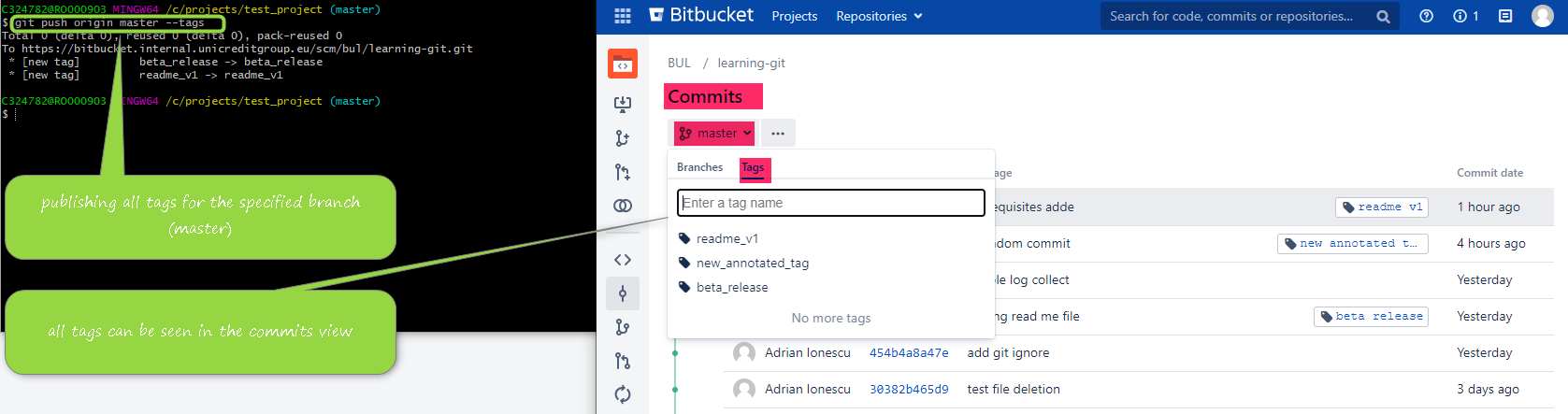
If you navigate via a browser to Bitbucket an select a commit that you know it is tagged, you will find out that no tags are associated with that commit.



So, you must publish it to the remote repository: **git push origin *the\_name\_of\_the\_tag***.

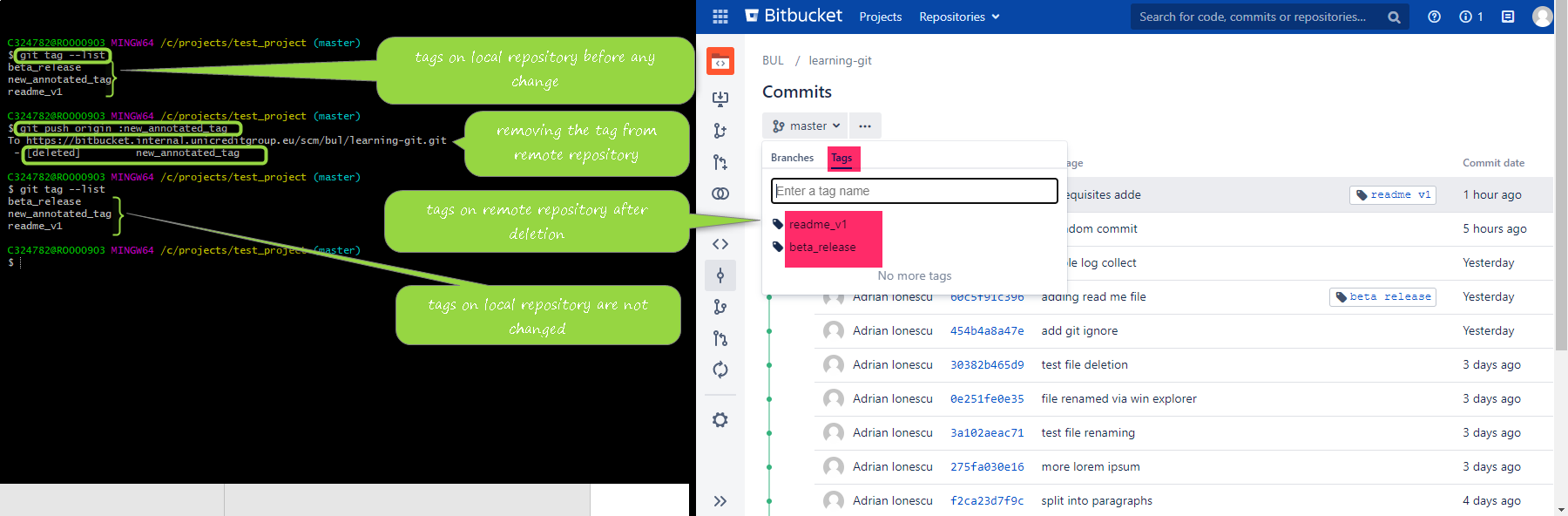


The command to push all the tags at once is: **git push origin *the\_branch\_name* –-tags**, e.g.: **git push origin *master* –-tags**.



### *4.13.7. Deleting a tag from remote repository*

By running the following command, you can remove a tag already published on remote repository: **git push origin *:name\_of\_the\_tag***. It basically means that you are replacing the respective tag with nothing.



## *4.14. Stashing*

If you have changes you've made to your working copy and you do not want to commit the modifications but you want to save them so you can work on something else, you can stash the changes and then come back and re-apply them later on.

To display all the stashes, use **git stash list**.

### *4.14.1. Simple stash*

To stash any changes that are work in progress run the command **git stash** or **git stash save**. This command will only stash tracked files.

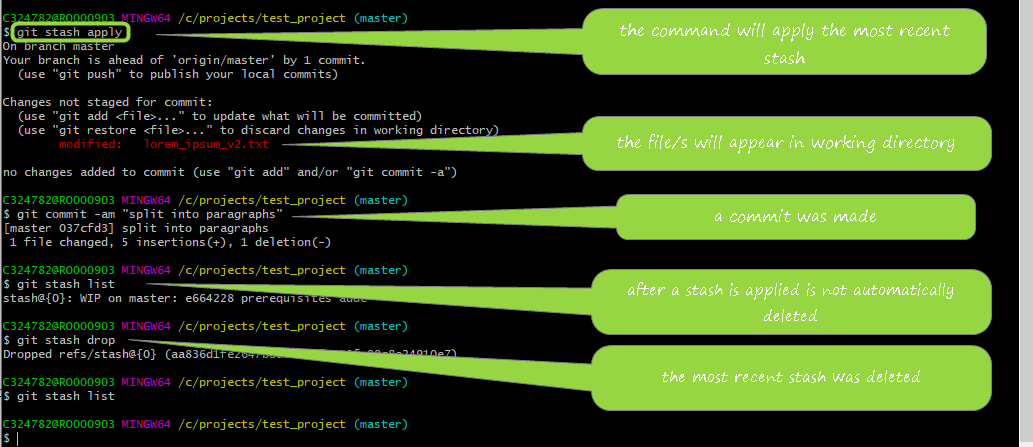


### *14.4.2. Apply a stash*

To apply the most recent created stash: **git stash apply**.

### *14.4.3. Delete a stash*

To delete the last stash: **git stash drop**.

****

### *14.4.4. Stash untracked files*

**git stash -u** will stash both tracked and untracked files.

**git stash pop** will apply the most recent stash and automatically delete it.



### *4.14.5. Multiple stashes*

To differentiate between different stashes, you can create a stash with a message: **git stash save *"a message"***.

Several commands can be ran using the stash’s index - the most recent stash has index 0 (zero):

* **git stash show stash@{*index\_of\_the\_stash*}**– shows all the modifications included in the specified stash;
* **git stash apply stash@{*index\_of\_the\_stash*}**;
* **git stash drop stash@{*index\_of\_the\_stash*}**;
* **git stash pop stash@{*index\_of\_the\_stash*}**.



To delete all stashes, use the command **git stash clear**.

### *4.14.6. Stash into branch*

If a stash contains modifications that should form a new branch, you can use the following command to create a new branch from that stash:

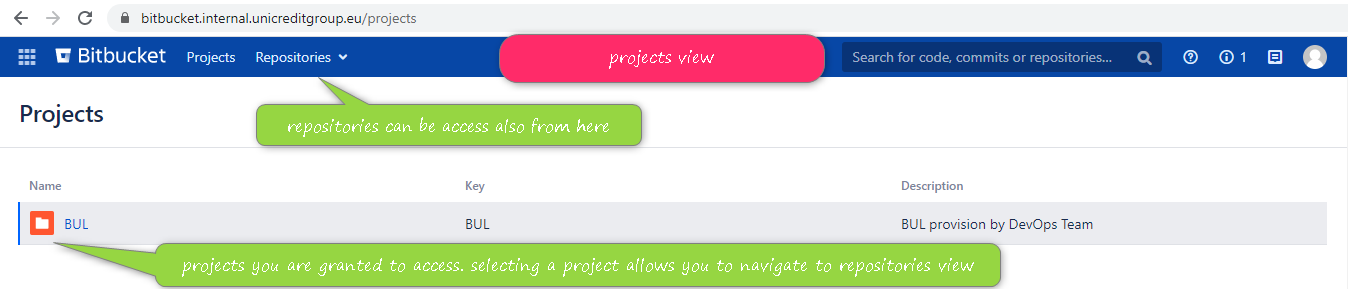
**git stash branch *name\_of\_the\_branch***.

This command will:

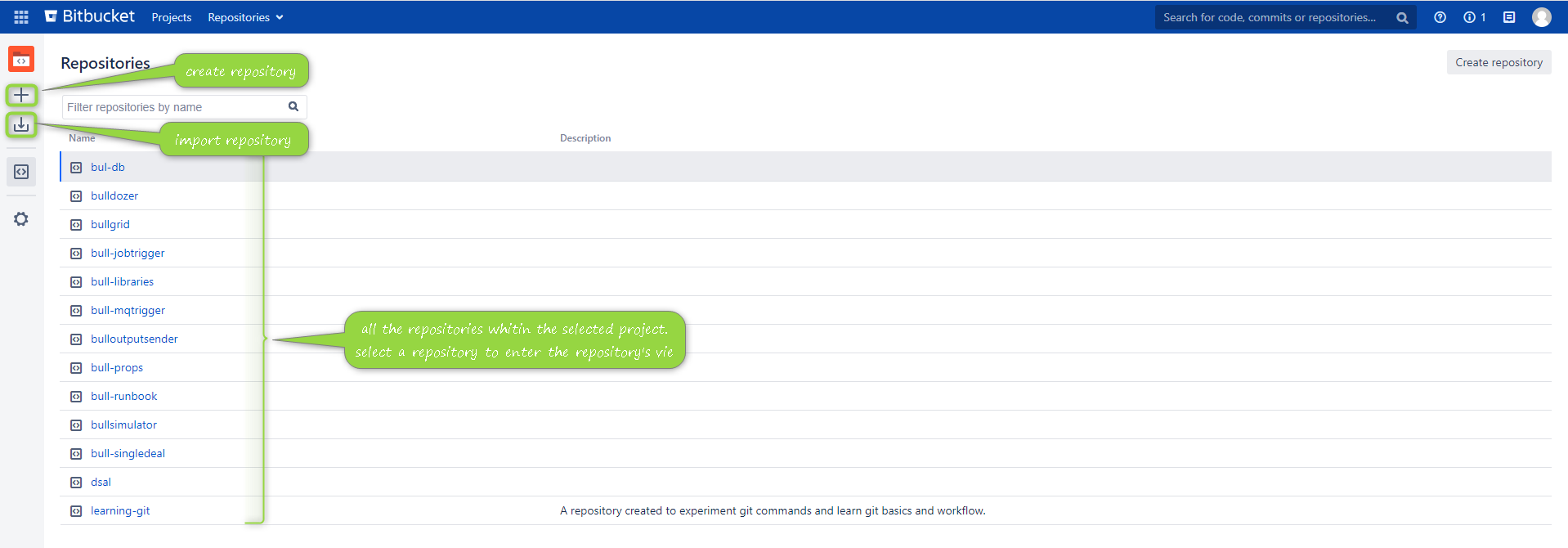
* create a new branch,
* you are switched to the newly created branch,
* apply the stash,
* drop the stash.

# ***Bitbucket features***

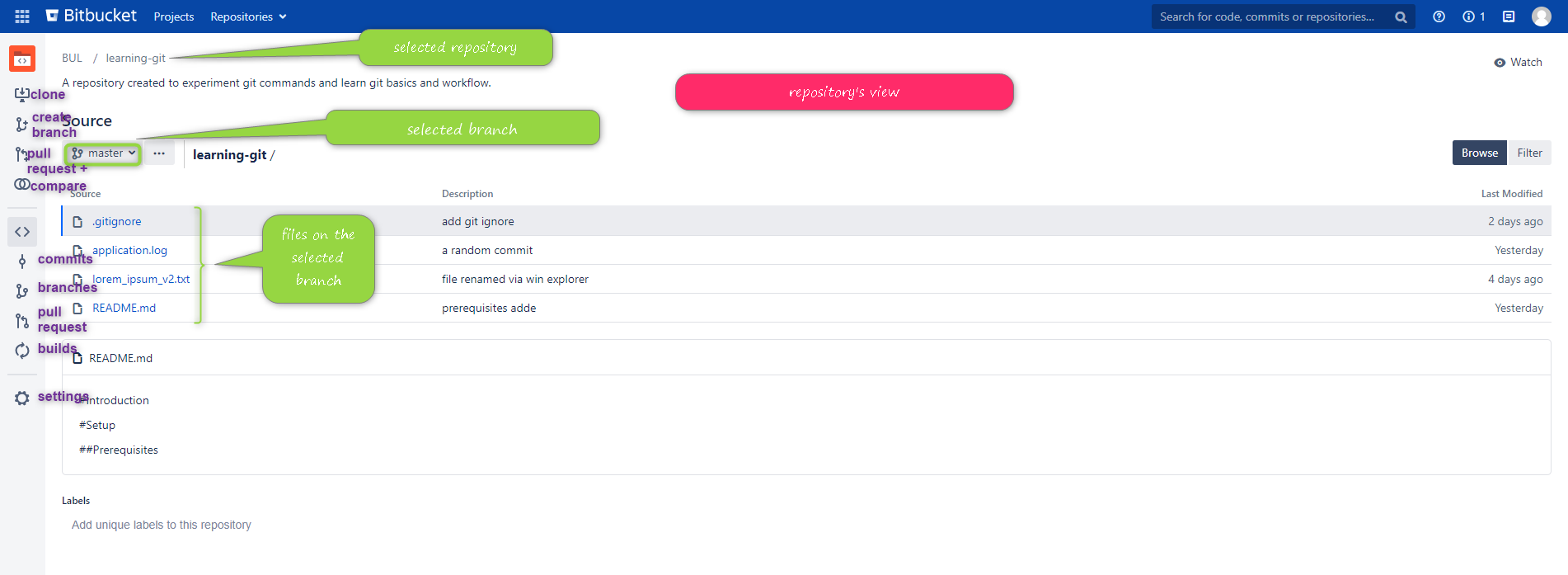
## *5.1. Projects’ view*



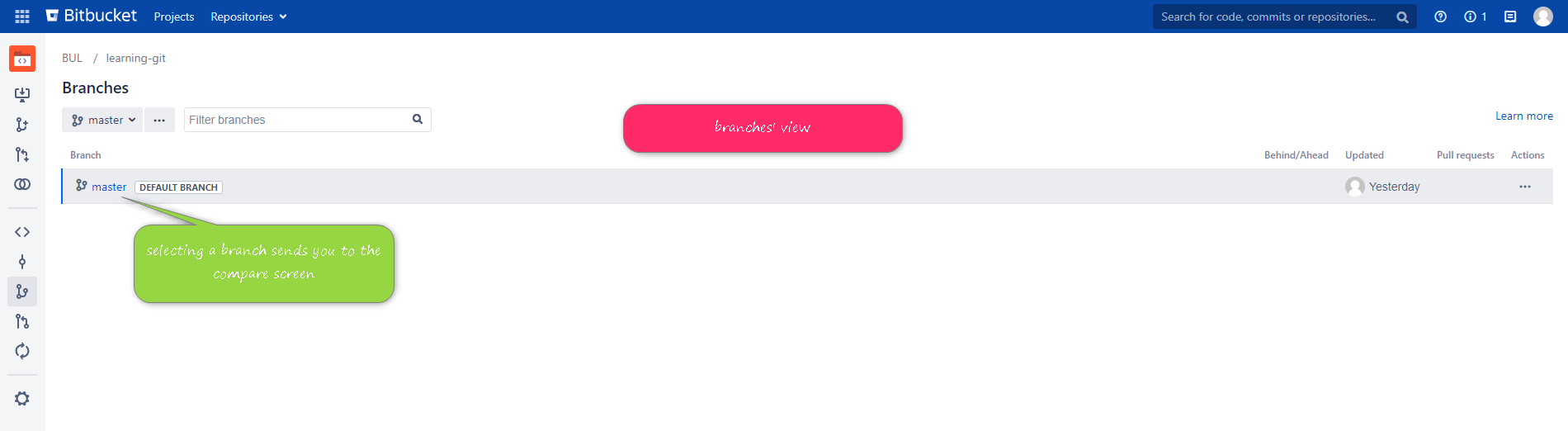
## *5.2. Repositories’ view*



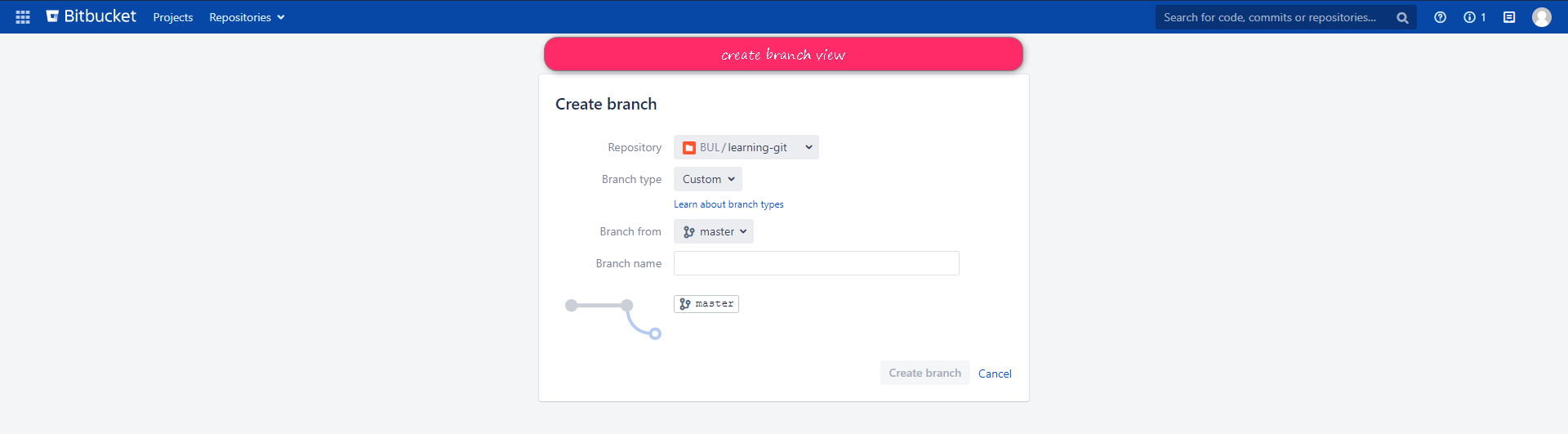
## *5.3. Repository’s view*



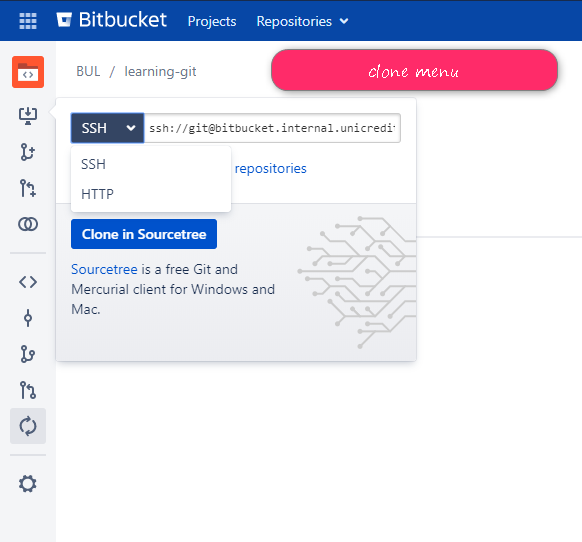
## *5.4. Branches’ view*



## *5.5. Create branch view*

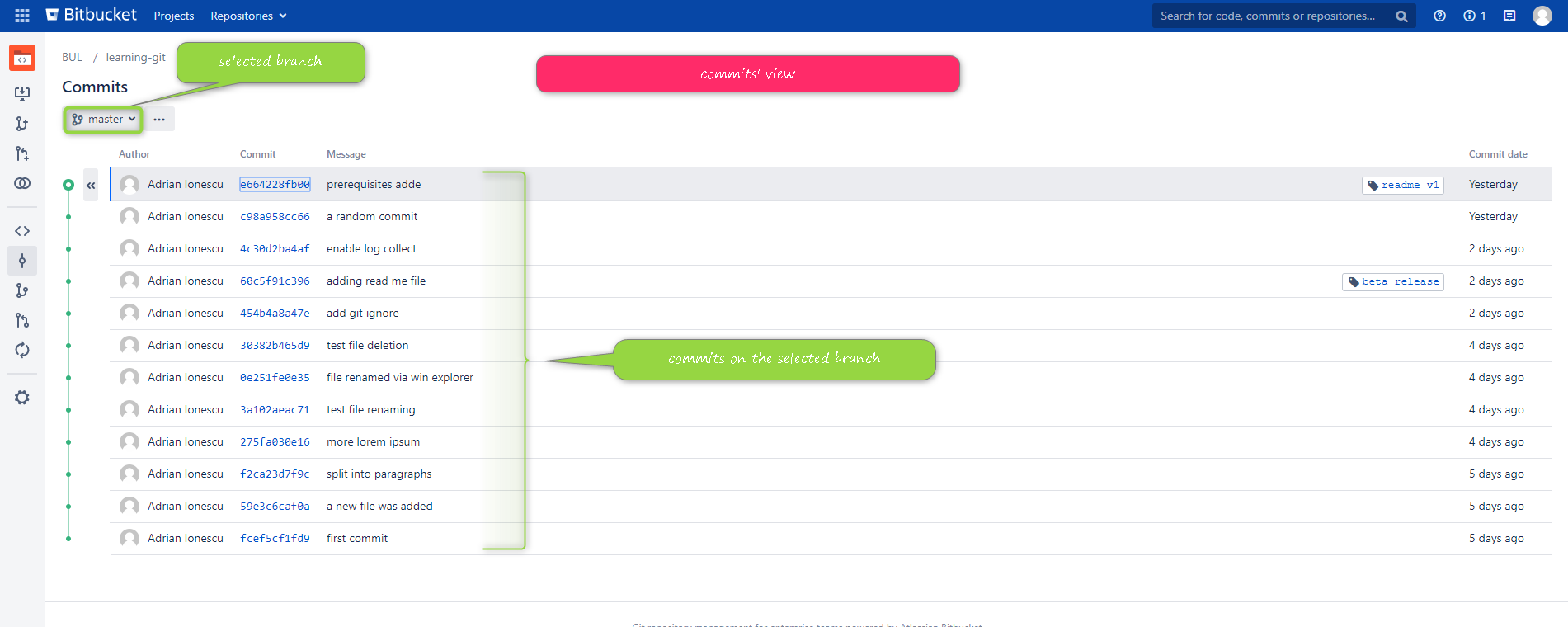


## *5.6. Cloning (from menu)*



## *5.7. Comparison’s view*

## *5.8. Commits’ view*



1. For personal use you can work with the service that suites you most. However, this course will use Bitbucket, but the approach is similar for any of the Git hosting services. [↑](#footnote-ref-1)
2. You will get explanations about Git terminology and Git workflow further in this course. [↑](#footnote-ref-2)
3. For more features offered by Bitbucket check the dedicated section [↑](#footnote-ref-3)
4. You can run a single command to add all files and commit them: git commit -am “the message” [↑](#footnote-ref-4)
5. As a recommendation, it is useful to rename a file before changing it. [↑](#footnote-ref-5)