**Tema laboratorului 1:**

"Versionarea codului sursa utilizand GIT". Lucrarea de laborator are ca scop studiul și înțelegerea principiilor de funcționare și utilizare a sistemului distribuit de control al versiunilor numit GIT.

**Terminologie**

* ***repository*** *-* pe server, conține ierarhia de fișiere și informațiile de versiune;
* ***working copy*** *-* varianta locală, obținuta de la server, pe care se fac modificările;
* ***revision*** - o versiune a unui document. (v1, v2, v3...).
* ***checkout*** *- aducerea pe masina locala a versiunii de pe server, sub forma unei working copy*
* ***update/pull****:*actualizarea repozitoriului local în funcție de modificările survenite, intre timp, pe server. Se aduc doar fișierele modificate;
* ***commit*** *-*înregistrează o nouă versiune a fișierului (fișierelor) modificat în repozitoriu.
* ***commit message*** *-* un mesaj asociat unei acțiuni *commit* care descrie schimbările făcute în noua versiune.
* ***changelog*** *-* o listă a versiunilor (commit-urilor) unui fișier/proiect de obicei însoțită de mesajele asociate fiecărui *commit*.
* ***diff****:*Afișează diferențele dintre două versiuni a unui fișier sau dintre fișierul modificat local (pe working copy) și o versiune de pe repository.
* ***revert***- renunțarea la ultimele modificări (locale) făcute într-un fișier din *working copy*, și revenirea la ultima versiune aflată în repozitoriu sau la o versiune la alegere.
* ***branch*** *- c*reează o “copie” a unui fișier/proiect pentru modificări „în paralel” fără a afecta starea actuală a unui proiect.
* ***merge*** *- a*plică ultimele modificări dintr-o versiune a unui fișier peste alt fișier;
* ***conflict*** *- s*ituația în care un *merge* nu se poate executa automat și modificările locale sunt în conflict cu modificările din repozitoriu.
* ***resolve***: rezolvarea (de obicei manuală) a conflictelor apărute într-un fișier după un *merge*.

**Indicatii metodice:**

### Preparing the environment

* install Git on our machine
* create a GitHub account
* create a workspace on our machine

If you’ve already done this, you can go straight to the **GitHub’s workflow and the Terminal** section.

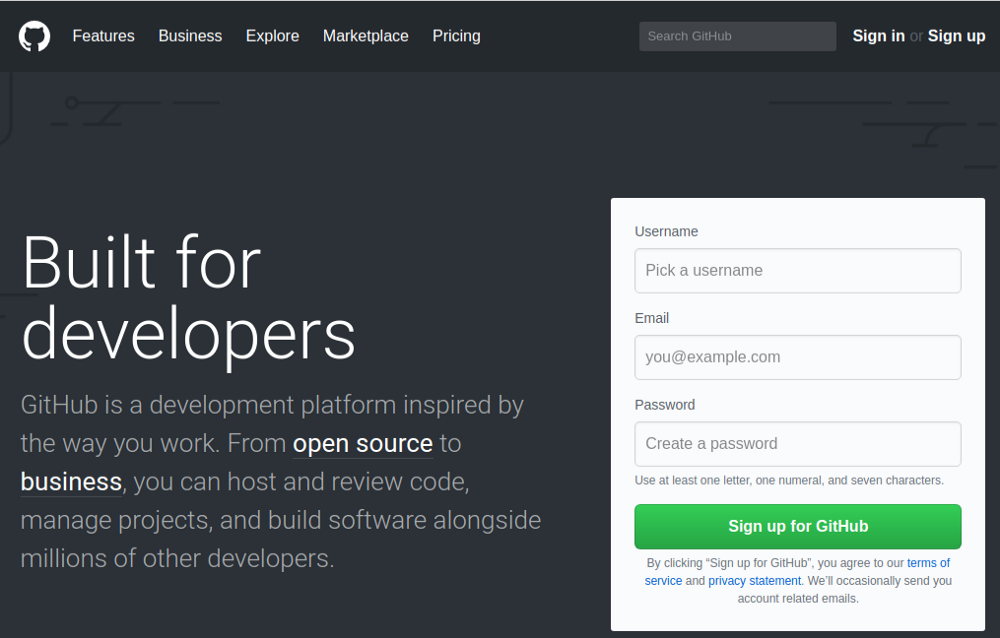
#### **Installing Git on your machine**

Git installation is different on each operation system. Check out [Git’s](https://git-scm.com/downloads) official site to see which way is right for you.

Once Git is installed, we need to create a GitHub account and configure it on our machine.

#### **Creating an account on GitHub**

To create an account, go to the [GitHub](https://github.com/) web site and fill out the main form.

You need to inform GitHub which plan you want to use. Choose the **free** option. The only difference is that you can setup private repositories with the paid plan.

#### **Setting up our system with our GitHub data**

Open up your Terminal. In Windows, you have to open the **start menu** and type **cmd.**Then click enter.

*git config --global user.name "our\_GitHub\_user\_name"*

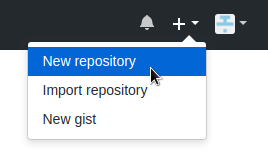
*git config --global user.email "our\_GitHub\_user\_email"*

#### **Setting up your GitHub access key**

To create this key, follow the process outlined in the GitHub [documentation](https://help.github.com/articles/adding-a-new-ssh-key-to-your-github-account/).

#### **Create a new project**

Let’s get back to your GitHub page and click on the plus icon (+) at the top of the page.



1. **Create your workspace**

Run the command *mkdir folder\_name*to create the workspace. For example:

*mkdir workspace*

#### **Clone your repositories**

Cloning a repository means that you’ll copy all of the files and directories on the GitHub server onto your machine so you can work with them.

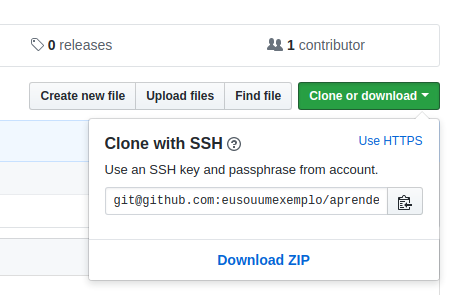
Now you need to clone the project that you created on GitHub to your **workspace.**To do this, go to the folder that you just created. On cmder, type:

cd workspace\

**Tip**: if you created the folder or want to access one which already exists, you can start typing its name and hit TAB, and cmder will autocomplete the name for you.

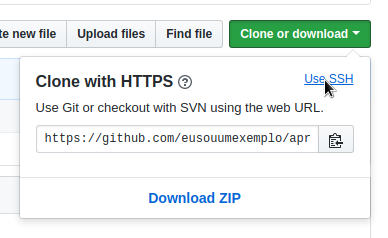
With that, go to your project page on GitHub and get the link that you need to clone the repository.

The link is in that green button named **Clone or Download**:



Getting the link to clone your repo.

Change from HTTPS to SSH, because you already configured your access key in your account.

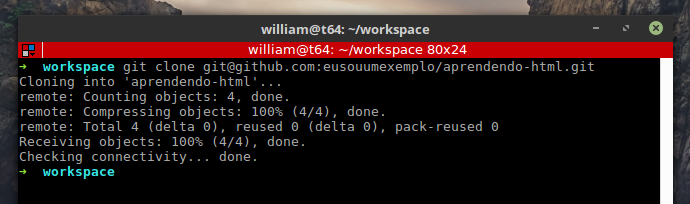


Changing https to ssh link.

Now you can run the git clone command and pass the link that you get. Just like that:

*git clone git@github.com:our-username/learning-html.git*

And your repository will be cloned, like in the following picture:



Clone confirmation message.

You can access the repository folder which was created in your workspace when you cloned it.

Type the command: *cd learning-html/*

**Attention**: I’m assuming that you are inside the *workspace*diretory now. If you aren’t, the above command will not work. Use *cd %home%\workspace\*and then the above command.

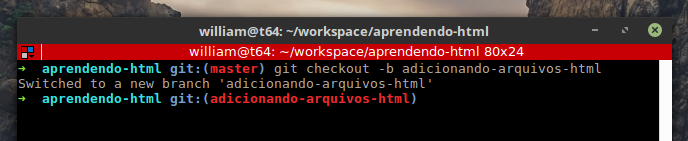
#### **Create a branch**

Every time you change something in a project versioned with Git, you should create a **branch**with the name of the task which you’re working on. This prevents you from messing up the “main” code located on the **master** branch. For this, you can use the following command:

*git checkout -b task\_name*

A **branch** is like a tree branch. It’s part of the trunk of the tree. So you can make changes in parallel with the main part of the project without affecting it.

For example:



Changing branch.

Once you’ve done this, you can change automatically to the newly created branch and can code like crazy now.

#### **Commit the changes**

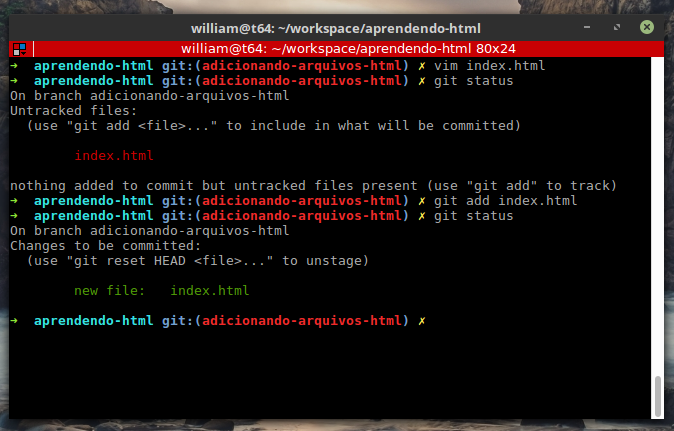
Once you finish a change to your project, you should **commit**the change to your remote repository (the one on GitHub’s servers).

To **commit** something is to tell Git that you are putting your changes in the queue to be pushed (sent) to your remote repository.

Imagine that you just created an HTML page and added some titles and text to it. You have the first version of this document now, so you should commit it.

To do this, run some commands so that Git understands that we want to send our changes do the remote repo. Run *git add file\_name*to tell Git to stage the file.

Alternatively, you can run *git add --all*to send all the files that you made some changes to. With the *git status*command, you can see which changed files you will commit to the server.



Example of the first version of a file.

In the above example, the **index.html**file was createdand the **git status** command was run to see what was changed. Then the file was added with **git add** and **git status** was run again to see which file was added to the Git workspace.

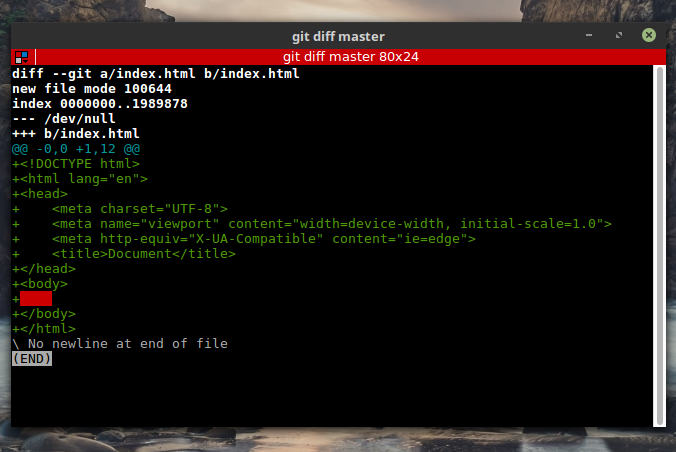
With that you can now **commit** the changes. Just run the **git commit** command, just like *git commit -m "commit\_message".* Remember to include a descriptive message of what was added to the commit.

#### **Merging the changes**

After you’ve committed the changes, you now have a branch with modifications ahead of the ones in the **master** branch. That means that you have a different version of the project, and you need to merge those changes with the main version of the project. Before doing that, verify what the differences are between the branches. On your branch, perform the command:

*git diff master*

The output will be something like:

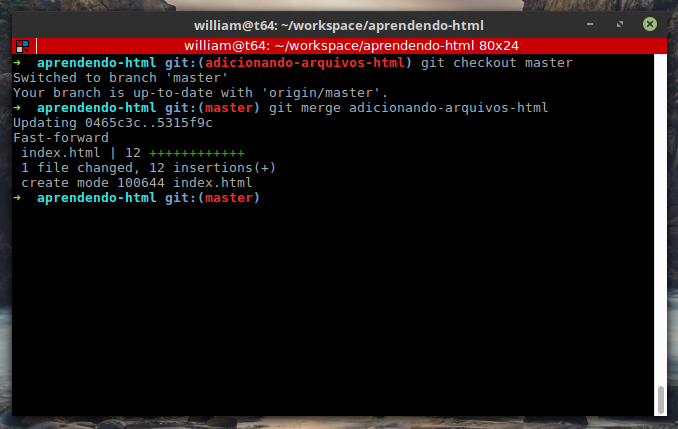


Git diff output.

Git shows you the newest commit made, which files were added or changed, and what was changed as well.

Since you know that you have differences between your branch and the master, you need to **merge**them to **join**the new commits, which you made in your branch, with the code in the master. To do this, you need to go to the master branch, on cmd, and run the command *git merge*.

To get back to the master, run *git checkout master*. To merge the commits, run *git merge our\_branch\_name* .



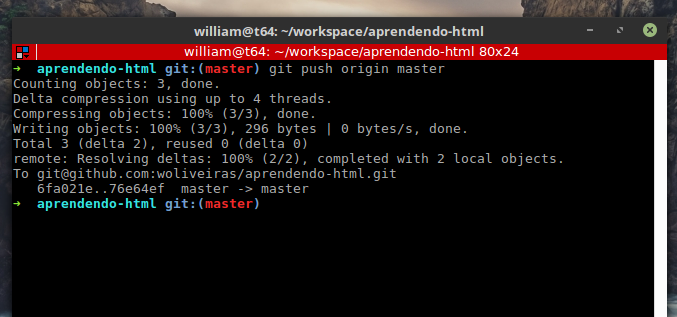
Merging example.

Git will show you an output confirming what was added.

#### **Sending it to GitHub**

After you’ve made and merged all the changes, you can now send them to your remote repository on GitHub.

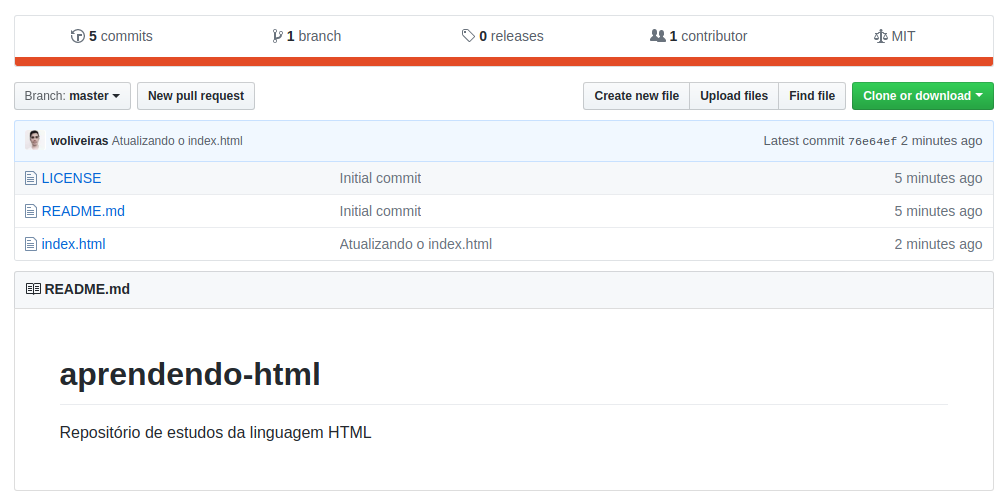
You will use *git push origin master*to do this.



Push our changes to the remote repo.

You can also just use *git push*. It’ll have the same result. But when you push changes for the first time on your workspace, you need to do*git push origin master* so that Git will know that your workspace is the origin of the push.

Now your commit will appear on your GitHub repository’s page:



The last commit which we just did is showed on the project’s page on github.

**Documentatie:**

* <https://help.github.com/articles/adding-a-new-ssh-key-to-your-github-account/#platform-windows>
* <https://services.github.com/on-demand/downloads/github-git-cheat-sheet.pdf>