

# GIT Project

## Index

1. Problem description.....	2
2. Solution algorithm.....	2
1. Installing SSH and GIT.....	3
2. Setting up SSH connectivity between machines.....	4
3. Initiating our first repository.....	4
4. First commit from Terminal.....	5
5. First commit from GUI.....	5
3. Solution Diagram.....	7
4. Solution configuration.....	7
5. Solution running demonstration.....	8
1. Repository initialization.....	9
1.1. Cloning a repository (Client1 [CLI]).....	9
1.2. Cloning a repository (Client2 [GUI]).....	10
2. Status of the working copy.....	11
2.1. Status of the working copy (CLI).....	11
2.2. Status of the working copy (GUI).....	11
3. Adding files to the working copy.....	12
3.1. Adding files to the working copy (CLI).....	12
3.2. Adding files to the working copy (GUI).....	12
4. Removing files from the working copy.....	13
4.1. Removing files from the working copy (CLI).....	13
4.2. Removing files from the working copy (GUI).....	13
5. Registering changes in the working copy (commit).....	14
5.1. Committing from CLI.....	14
5.2. Committing from GUI.....	14
6. Creation and use of new branches.....	15
6.1. Creation and use of a beta branch from CLI.....	15
6.2. Creation and use of a beta branch from GUI.....	15
7. Merging two branches.....	16
7.1. Merging beta to main from CLI.....	16
7.2. Merging beta to main from GUI.....	17
8. Adding tags to versions.....	18
8.1. Adding a tag for beta0.1 from CLI.....	18
8.2. Adding a tag for beta1.0 from GUI.....	18
9. Eclipse.....	19

## 1. Problem description

We are asked to implement a GIT server for a company with full connectivity between two clients and a server, and full functionality with Eclipse and a GUI of our choice.

## 2. Solution algorithm

The first logical step is to gather all the possible information about GIT, connectivity between machines (SSH), Eclipse, GUIs for GIT... Once that is done, we can start working.

We are working with three virtual machines (VMs), with one of them being the GIT server and the other two clients.

After setting up the three VMs and ensuring connectivity between them all by setting up a NAT Network and establishing static IP directions to each one of them:

VM	Static IP
Server	192.168.125.2
Client 1 (CMD)	192.168.125.3
Client 2 (GUI)	192.168.125.4

With all of that set up, and after checking connectivity between machines by pinging one another, we can start setting up the basic tools we will be working with.

---

## 1. *Installing SSH and GIT*

The very first step is to check whether the tools we are going to work with come pre-installed or are yet to be installed. We will start with SSH.

The ssh client comes pre-installed most distros, but we will check just in case with:

```
ssh -V
```

That only checks the ssh client, which we will need in the client machines, but we will also need to set up a SSH server.

First, we will check whether it is installed or not with the following command:

```
sudo systemctl status ssh
```

That should show the following information, along with some more:

```
● ssh.service – OpenBSD Secure Shell server  
   Loaded: loaded (/lib/systemd/system/ssh.service; enabled; preset: enabled)  
   Active: active (running)
```

If that were not the case, we would proceed to install SSH by means of the following command, and rebooting the machine before checking again:

```
sudo apt install openssh-server
```

After ensuring that SSH is up and running, we will proceed to install GIT, as it is very unlikely it is installed by default. We are going to install GIT by using apt:

```
sudo apt install git
```

And then check with:

```
git -v
```

Which should show something similar to the following (version might be different):

```
git version 2.39.5
```

---

The next step would be to identify ourselves in GIT in each one of the VMs, which requires an e-mail address and a name. For that we are going to execute the following commands:

```
git config --global user.email "24.javier.pedragosa@insdanielblanxart.cat"  
git config --global user.name "JaviPedragosa"
```

Lastly, we will have to set up our defaultBranch to a different one than the default, which is 'master'. We will change ours to 'main':

```
git config --global init.defaultBranch main
```

## ***2. Setting up SSH connectivity between machines***

In both clients, we will generate SSH keys and copy them to the server.

To generate the key, we will run the following command:

```
ssh-keygen -t rsa -b 4096
```

And then check that it has been successfully created:

```
ls ~/.ssh/
```

Lastly, we will copy the key to the server with:

```
ssh-copy-id super@192.168.125.2
```

which should ask us for the password of the 'super' user on the server.

After that, we can verify the connection by using the following command, if we are not asked for a password that means we have correctly set it up:

```
ssh super@192.168.125.2
```

## ***3. Initiating our first repository***

With GIT and SSH installed in the three VMs, we will proceed to set up the server first, which starts by creating a new user we will call 'git':

```
sudo adduser git
```

After that we will switch to that user and create and initiate our first bare repository:

```
cd /home/super/git  
mkdir test  
cd test  
git init --bare
```

---

Lastly, we will clone it, create any kind of file and commit it from the server in order for us to have a non-empty repository to work.

```
git clone ~/git/test ~/test
cd ~/test
echo "First commit from server" > testServer.txt
git add .
git commit -m "First commit from Server"
git push origin main
```

#### 4. *First commit from Terminal*

With the SSH connection set up and GIT repository initiated, we should be able to clone and commit from the clients.

To clone it, we are going to execute the following command:

```
git clone super@192.168.125.2:/home/super/git/test ~/test
```

And then we will create a change to try and commit from the CMD client:

```
cd ~/test
echo "First commit from Client 1" > testClient1.txt
git add .
git commit -m "First commit from Client 1"
git push origin main
```

If no errors are shown, we have successfully set it up and verified that the GIT flow via Terminal is working correctly.

#### 5. *First commit from GUI*

In the second client, the one meant for using a GUI, we are going to install GIT-GUI, which is a simple and lightweight GUI for GIT. For that, we are going to execute the following:

```
sudo apt install git-gui
```

Then, we will open GIT-GUI and clone the repository by following the instructions and using the following URL: 'super@192.168.125.2:/home/super/git/test' with the defined path: '/home/super/test'.

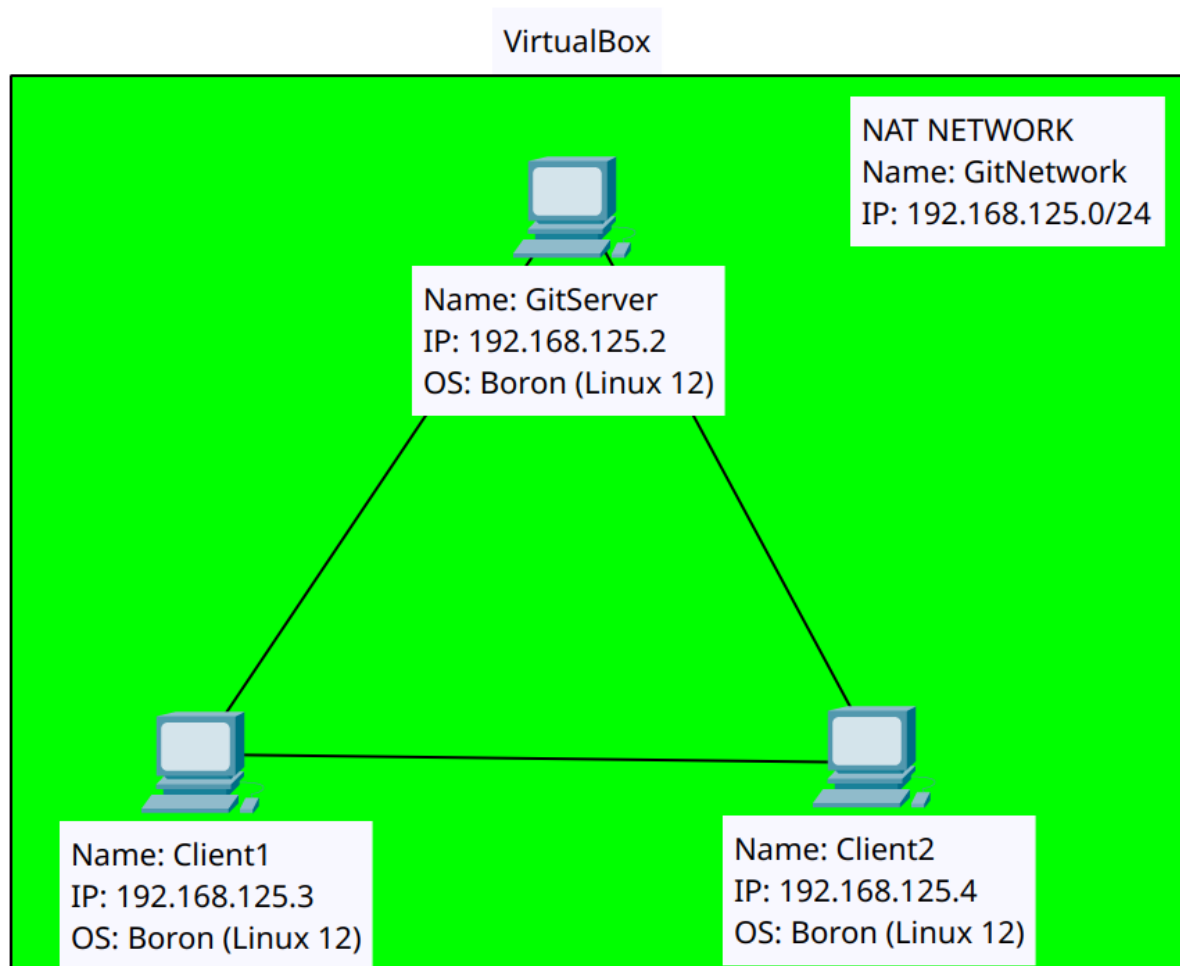
After that, we will generate a similar change as we did from the first client, by doing this:

```
cd ~/test
echo "First commit from Client 2" > testClient2.txt
```

---

Then, the GUI will show the new file 'test2.txt' in the Unstaged Changes tab. We can write a Commit Message and then click on **Stage Changed** > **Commit** > **Push** to successfully commit using GUI.

### 3. Solution Diagram



## 4. Solution configuration

The configuration needed for this project was simply for the networking, as every tool or program we installed and/or used made all the needed configuration for us and the only thing I had to tinker with was for the machines to be able to connect to one another.

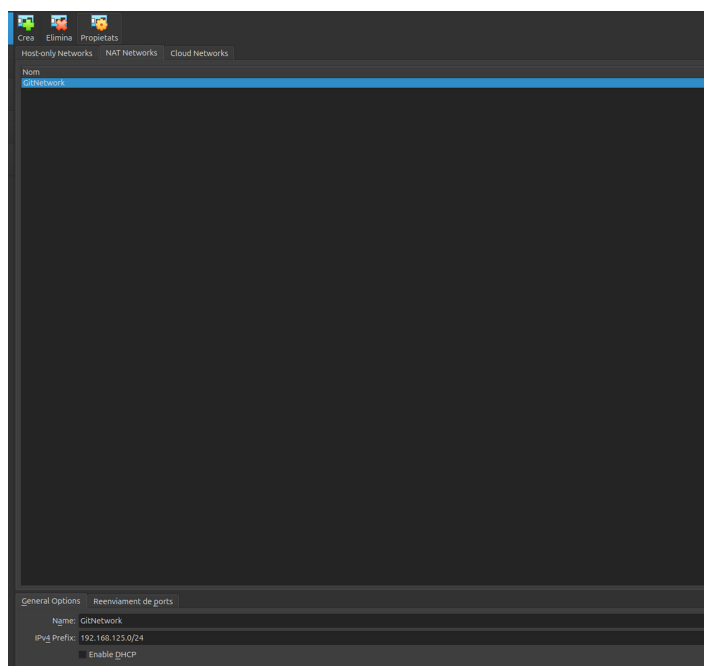


Figure 1: NAT Network in VirtualBox

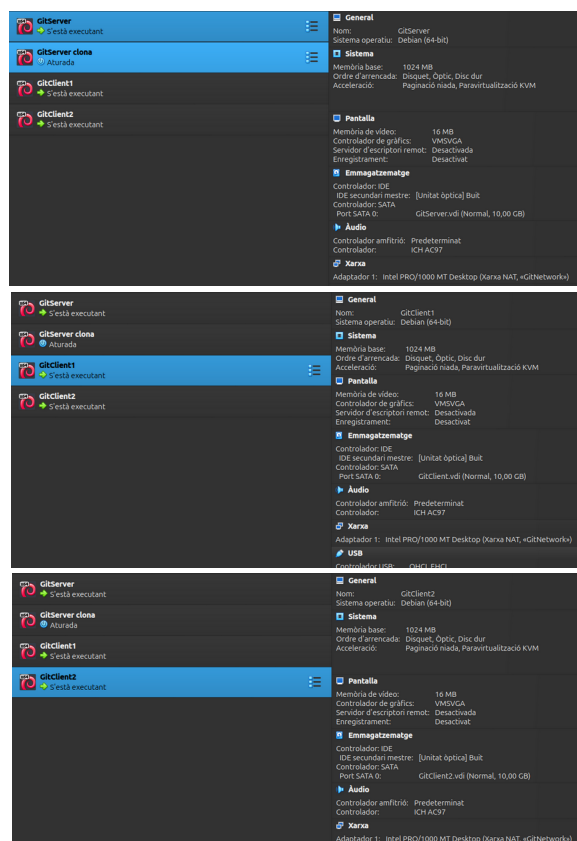


Figure 2: Settings on each one of the 3 VMs

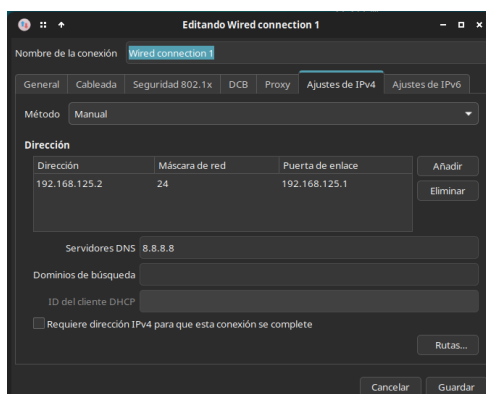


Figure 3: Boron settings where static IP was set (In this case, in the server)

## 5. Solution running demonstration

In this section, we will run through the required operations one by one both from Client1 (CLI) and Client2 (GUI).

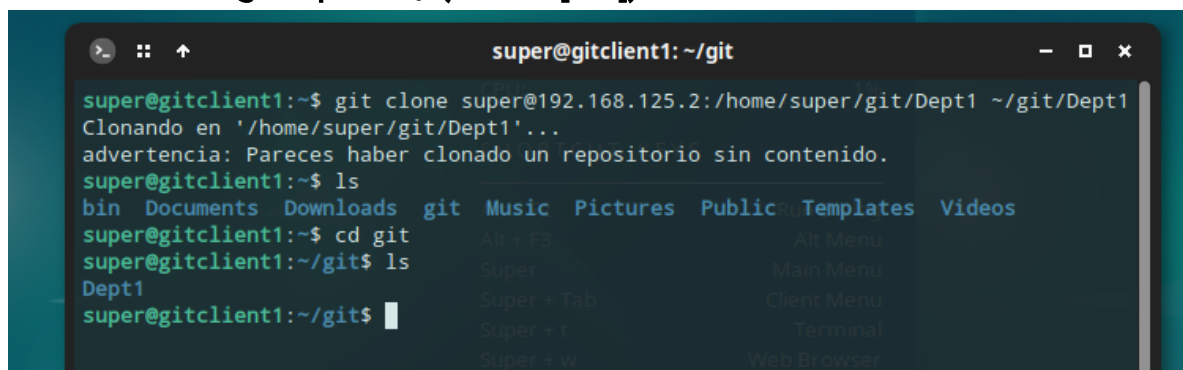
### 1. *Repository initialization*



```
super@gitserver: ~/git/Dept2
super@gitserver:~$ cd git
super@gitserver:~/git$ mkdir Dept1
super@gitserver:~/git$ cd Dept1
super@gitserver:~/git/Dept1$ git init --bare
Iniciado repositorio Git vacío en /home/super/git/Dept1/
super@gitserver:~/git/Dept1$ cd ..
super@gitserver:~/git$ mkdir Dept2
super@gitserver:~/git$ cd Dept2
super@gitserver:~/git/Dept2$ git init --bare
Iniciado repositorio Git vacío en /home/super/git/Dept2/
super@gitserver:~/git/Dept2$
```

Figure 4: Process of initiating two repositories from CLI in the Server VM

### 1.1. Cloning a repository (Client1 [CLI])



```
super@gitclient1: ~/git
super@gitclient1:~$ git clone super@192.168.125.2:/home/super/git/Dept1 ~/git/Dept1
Clonando en '/home/super/git/Dept1'...
advertencia: Parece haber clonado un repositorio sin contenido.
super@gitclient1:~$ ls
bin Documents Downloads git Music Pictures Public Templates Videos
super@gitclient1:~$ cd git
super@gitclient1:~/git$ ls
Dept1
super@gitclient1:~/git$
```

Figure 5: Process and demonstration of a repository being cloned in CLI



## 1.2. Cloning a repository (Client2 [GUI])

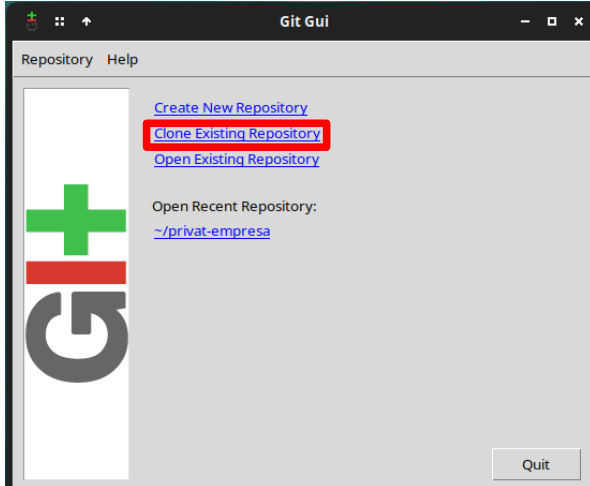


Figure 6: Git Gui's main menu, where the option we will select for cloning a repository is highlighted in a red box

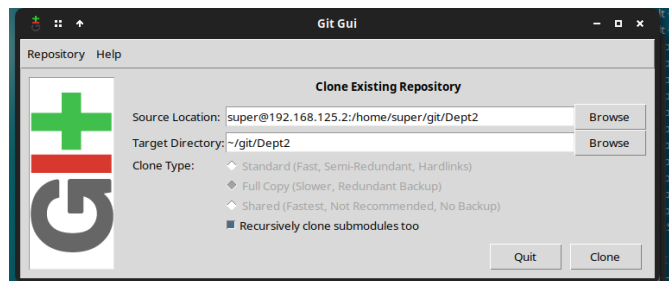


Figure 7: Git Gui's Clone Existing Repo. window where we set the source and the target before clicking on Clone

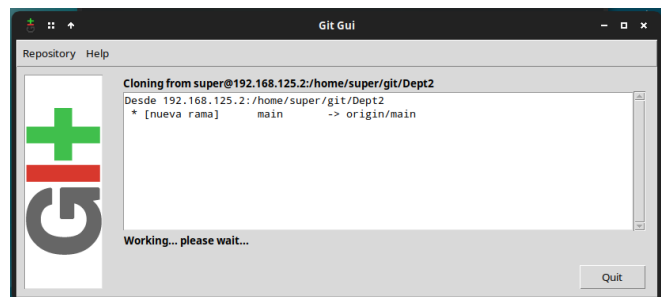


Figure 8: Git Gui working on cloning the repository

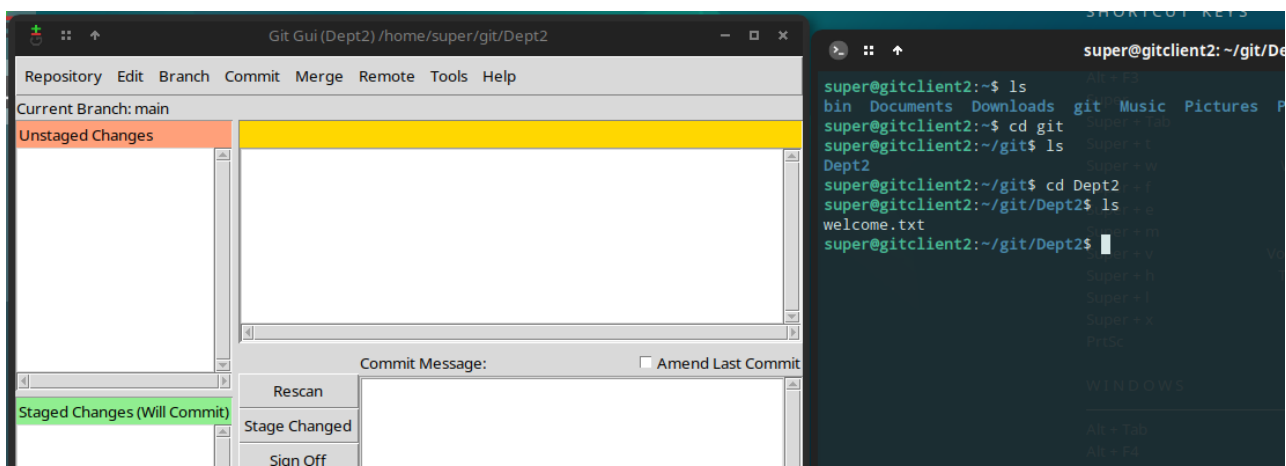


Figure 9: Demonstration of the repository already cloned

## 2. Status of the working copy

### 2.1. Status of the working copy (CLI)

```
super@gitclient1:~/git$ cd Dept1
super@gitclient1:~/git/Dept1$ echo "This is going to be my first commit" > test.txt
super@gitclient1:~/git/Dept1$ git status
En la rama main
No hay commits todavía
Archivos sin seguimiento:
(usa "git add <archivo>..." para incluirlo a lo que será confirmado)
test.txt
no hay nada agregado al commit pero hay archivos sin seguimiento presentes (usa "git add" para hacerles seguimiento)
super@gitclient1:~/git/Dept1$
```

Figure 10: Process of creating changes and showing the current status of the Working Copy in CLI

### 2.2. Status of the working copy (GUI)

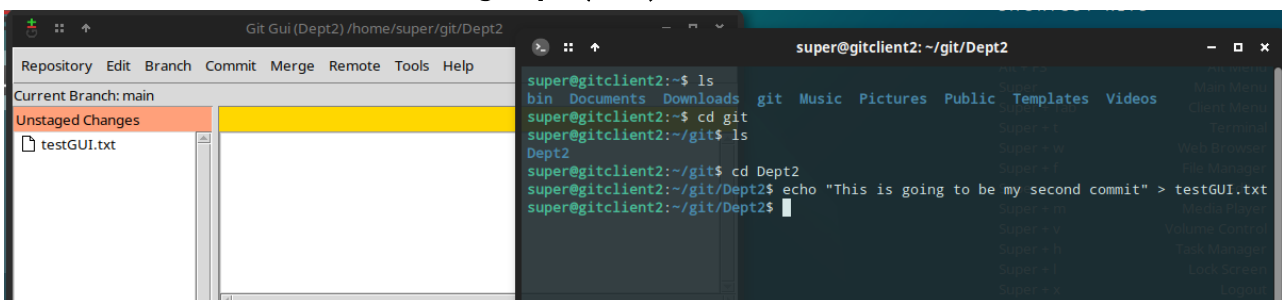


Figure 11: On the right side, CLI showing how I have made a change to the files in the repository. On the left side, Git Gui showing the change as an Unstaged Change after clicking the 'Rescan' button

### 3. Adding files to the working copy

#### 3.1. Adding files to the working copy (CLI)

```
super@gitclient1:~/git/Dept1$ git add test.txt
super@gitclient1:~/git/Dept1$ git status
En la rama main

No hay commits todavía

Cambios a ser confirmados:
(usa "git rm --cached <archivo>..." para sacar del área de stage)
nuevos archivos: test.txt
```

Figure 12: The command 'git add' being used to stage changes and the status to demonstrate

#### 3.2. Adding files to the working copy (GUI)

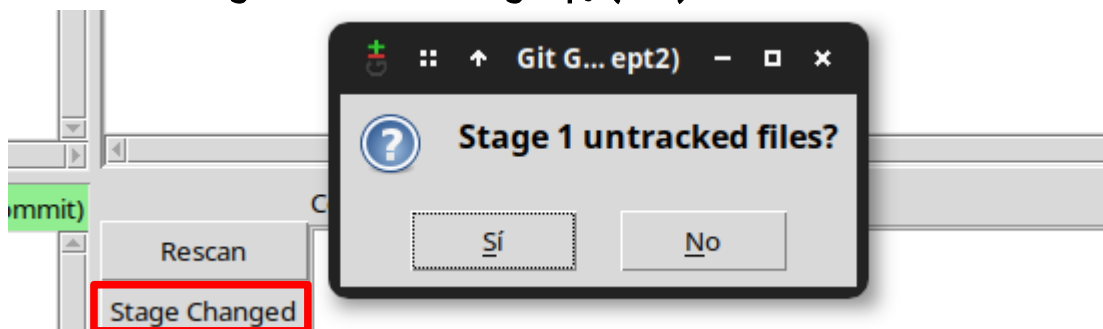


Figure 13: The Stage Changed button highlighted in red is the one that lets us stage changes, after confirmation by clicking 'yes'

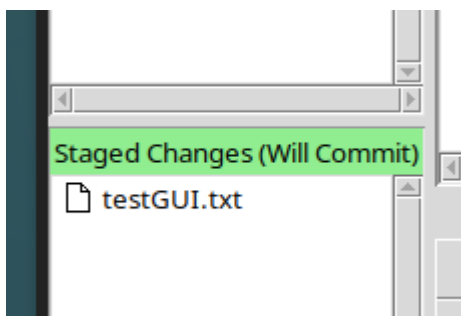


Figure 14: Staged changes ready to be committed will be shown here

## 4. Removing files from the working copy

### 4.1. Removing files from the working copy (CLI)

```
super@gitclient1:~/git/Dept1$ git rm --cached test.txt
rm 'test.txt'
super@gitclient1:~/git/Dept1$ ls
test.txt
super@gitclient1:~/git/Dept1$ git status
En la rama main

No hay commits todavía

Archivos sin seguimiento:
(usa "git add <archivo>..." para incluirlo a lo que será confirmado)
test.txt
```

Figure 15: File test.txt being removed from the staged changes, without deleting the file, and demonstration of the status not showing it as a staged change anymore

### 4.2. Removing files from the working copy (GUI)

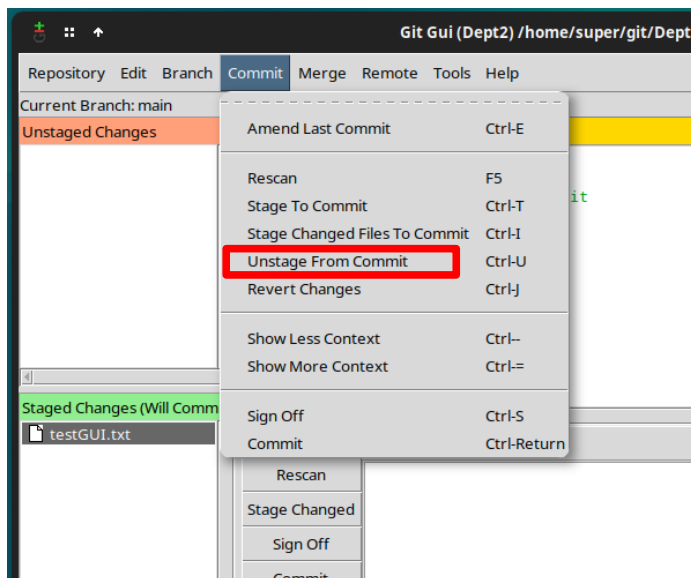


Figure 16: Commit menu where the option that unstages changes is highlighted in red.

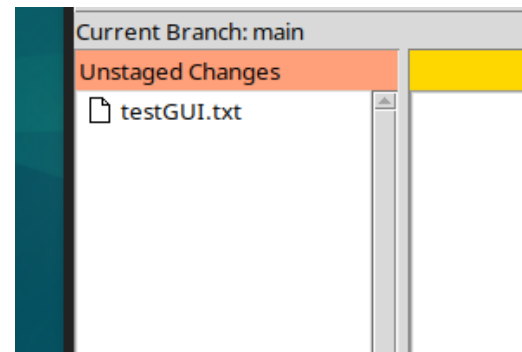


Figure 17: Change showing up in the unstaged changes tab after clicking 'Unstage From Commit'

## 5. Registering changes in the working copy (commit)

### 5.1. Committing from CLI

```
super@gitclient1:~/git/Dept1$ git add .
super@gitclient1:~/git/Dept1$ git commit -m "This is my first commit"
[main (commit-raíz) a569c24] This is my first commit
1 file changed, 1 insertion(+)
create mode 100644 test.txt
```

Figure 18: All changes being staged and then committing with a comment.

### 5.2. Committing from GUI

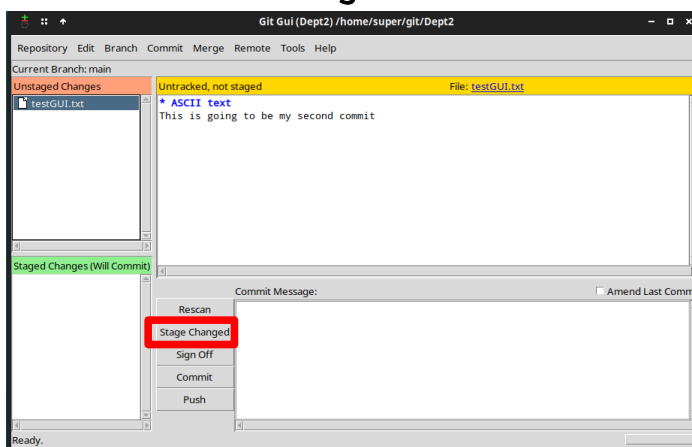


Figure 19: Unstaged Changes will be staged upon clicking the highlighted button

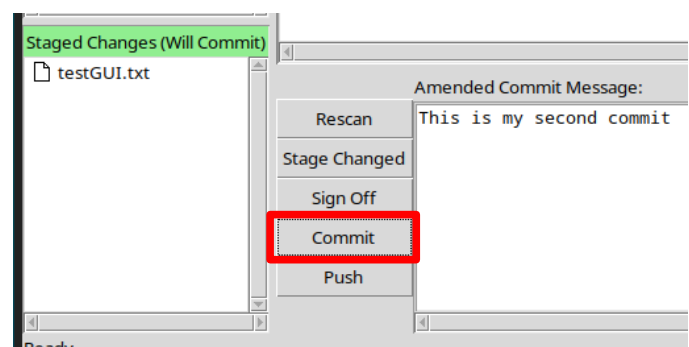


Figure 20: After defining a Commit Message, clicking Commit will register the changes in the working copy

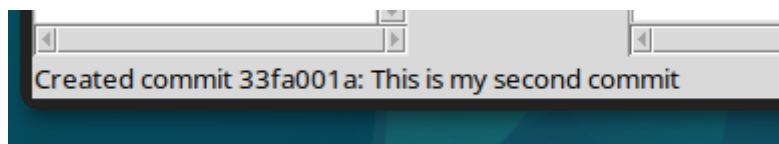


Figure 21: Message telling us our commit was created

## 6. Creation and use of new branches

### 6.1. Creation and use of a beta branch from CLI

```
super@gitclient1:~/git/Dept1$ git checkout -b beta
Cambiado a nueva rama 'beta'
super@gitclient1:~/git/Dept1$ echo "Beta Version" > beta.txt
super@gitclient1:~/git/Dept1$ git add .
super@gitclient1:~/git/Dept1$ git commit -m "Added beta version"
[beta 2c5f33e] Added beta version
1 file changed, 1 insertion(+)
create mode 100644 beta.txt
super@gitclient1:~/git/Dept1$ git push set upstream origin beta
Enumerando objetos: 6, listo.
Contando objetos: 100% (6/6), listo.
Comprimiendo objetos: 100% (3/3), listo.
Escribiendo objetos: 100% (6/6), 502 bytes | 502.00 KiB/s, listo.
Total 6 (delta 1), reusados 0 (delta 0), pack-reusados 0
To 192.168.125.2:/home/super/git/Dept1
* [new branch]      beta -> beta
rama 'beta' configurada para rastrear 'origin/beta'.
super@gitclient1:~/git/Dept1$
```

Figure 22: Creation of beta branch and a new file for that branch. Change is committed and pushed to the beta branch.

### 6.2. Creation and use of a beta branch from GUI

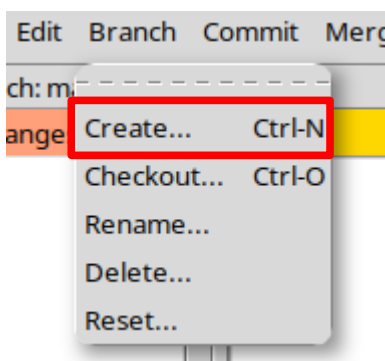


Figure 24: Branch menu on Git Gui, where the Create button will open a window that will allow us to define and create a new branch

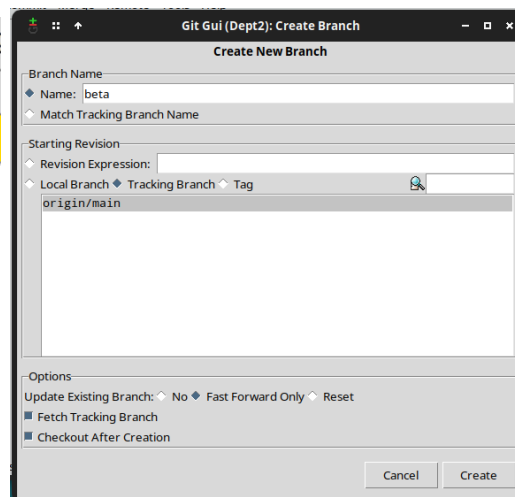


Figure 23: The create branch window where we have defined our branch

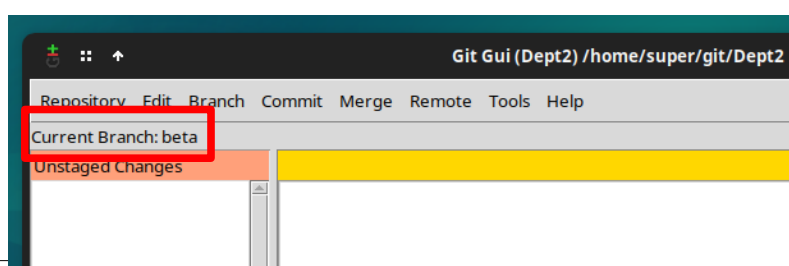


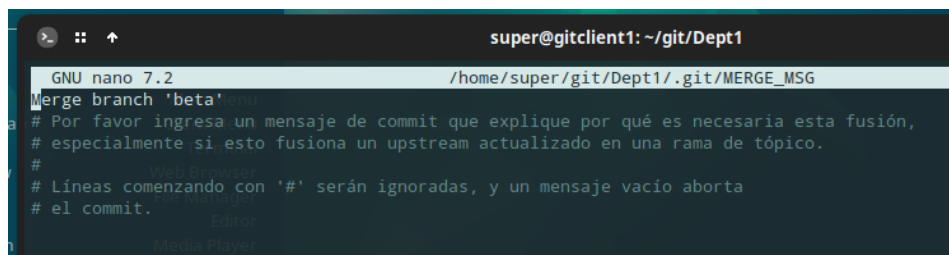
Figure 25: Git Gui telling us we are currently working on the beta branch

## 7. Merging two branches

### 7.1. Merging beta to main from CLI

```
6741763..cb78cd8 beta -> beta
super@gitclient1:~/git/Dept1$ git checkout main
Cambiado a rama 'main'
super@gitclient1:~/git/Dept1$ git merge beta
Merge made by the 'ort' strategy.
 betaTest.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 betaTest.txt
```

Figure 26: Swapping to the main branch and then calling the merge function



```
super@gitclient1: ~/git/Dept1
GNU nano 7.2 /home/super/git/Dept1/.git/MERGE_MSG
Merge branch 'beta' into main
# Por favor ingresa un mensaje de commit que explique por qué es necesaria esta fusión,
# especialmente si esto fusiona un upstream actualizado en una rama de tópico.
#
# Las líneas comenzando con '#' serán ignoradas, y un mensaje vacío aborta
# el commit.
Editor
Media Player
```

Figure 27: Automatically nano opens up a file to add a comment about why we are merging the two branches

```
super@gitclient1:~/git/Dept1$ git push origin main
Enumerando objetos: 1, listo.
Contando objetos: 100% (1/1), listo.
Escribiendo objetos: 100% (1/1), 238 bytes | 238.00 KiB/s, listo.
Total 1 (delta 0), reusados 0 (delta 0), pack-reusados 0
To 192.168.125.2:/home/super/git/Dept1
 df47fe2..f5f2275 main -> main
super@gitclient1:~/git/Dept1$ ls
betaTest.txt  test.txt
super@gitclient1:~/git/Dept1$
```

Figure 28: Merge being pushed and files from both branches showing up together

## 7.2. Merging beta to main from GUI

```
super@gitclient2:~/git/Dept2$ git checkout beta
Ya en 'beta'
super@gitclient2:~/git/Dept2$ ls
betaTest.txt
super@gitclient2:~/git/Dept2$ git checkout main
Cambiado a rama 'main'
super@gitclient2:~/git/Dept2$ ls
testGUI.txt  welcome.txt
```

Figure 29: Two branches with different files

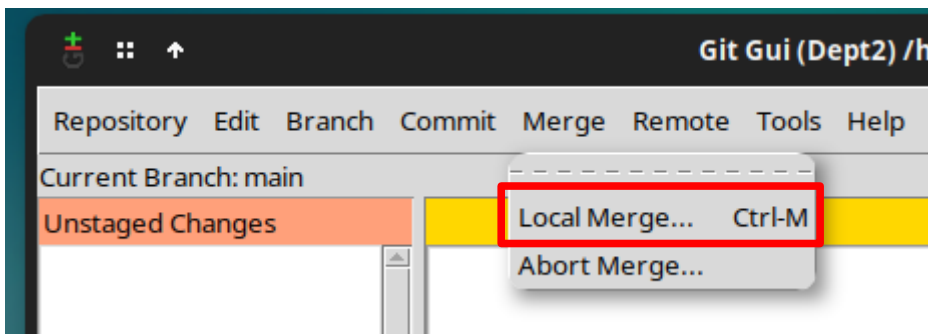


Figure 30: While in main, we open up the Merge menu and click on 'Local Merge...'

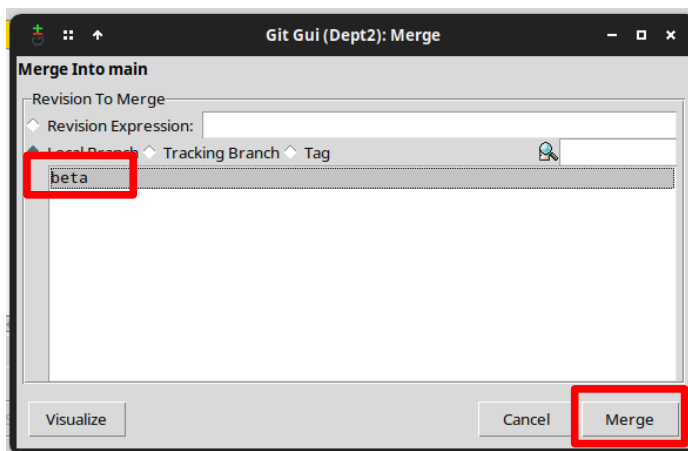


Figure 31: We select the branch we want to merge into main and click on 'Merge'

```
merge beta
Actualizando 33fa001..5638e57
Fast-forward
 betaTest.txt | 1 +
 testGUI.txt  | 1 -
 welcome.txt  | 1 -
 3 files changed, 1 insertion(+), 2 deletions(-)
 create mode 100644 betaTest.txt
 delete mode 100644 testGUI.txt
 delete mode 100644 welcome.txt

Success
```

Figure 32: Message telling us we have successfully merged beta into main



## 8. Adding tags to versions

### 8.1. Adding a tag for beta0.1 from CLI

```
super@gitclient1:~/git/Dept1$ git tag beta0.1
super@gitclient1:~/git/Dept1$ git push origin beta0.1
Total 0 (delta 0), reusados 0 (delta 0), pack-reusados 0
To 192.168.125.2:/home/super/git/Dept1
* [new tag]         beta0.1 -> beta0.1
super@gitclient1:~/git/Dept1$
```

Figure 33: Tag being created and pushed

### 8.2. Adding a tag for beta1.0 from GUI

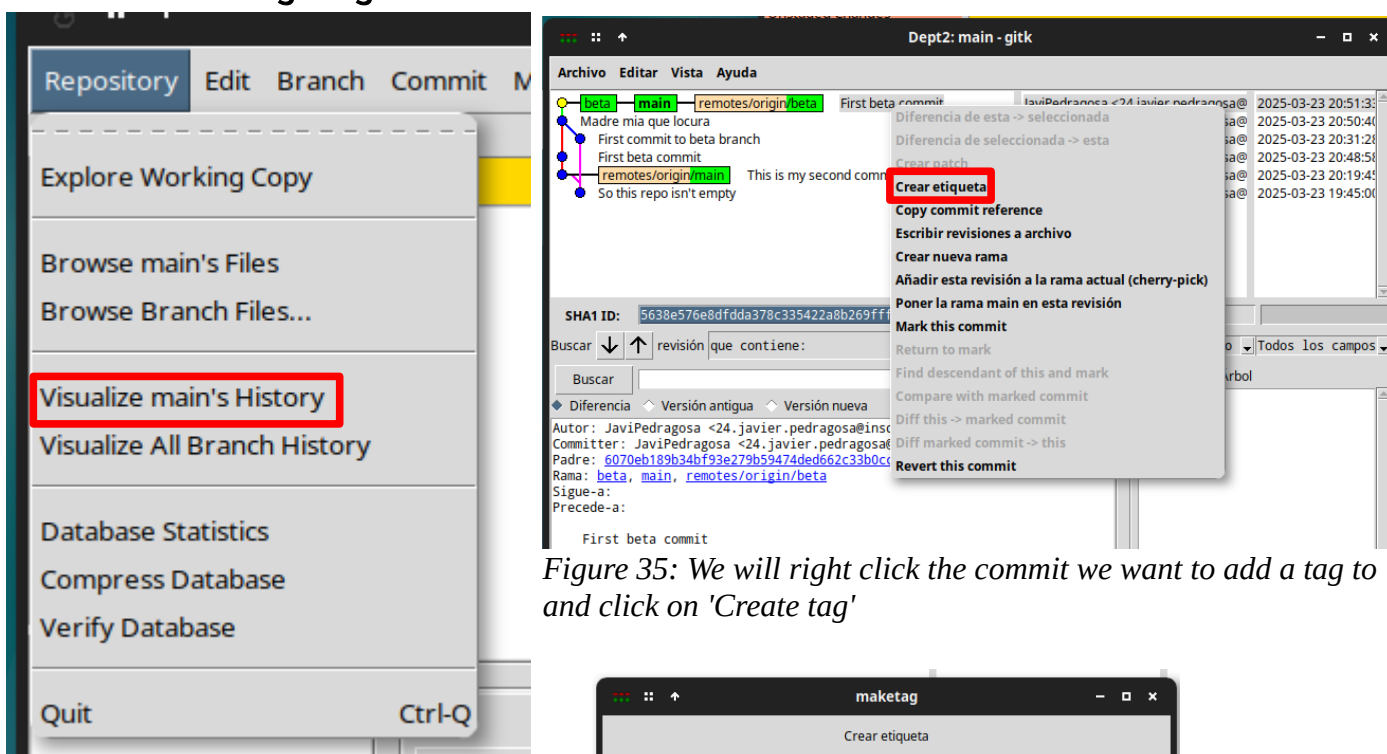


Figure 34: In the repository menu, we will click on 'Visualize main's History'

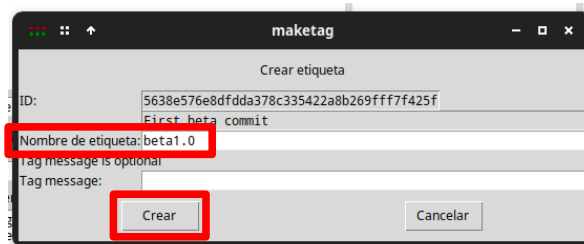


Figure 36: We set the name we want to our tag, and click on 'Create'

## 9. Eclipse

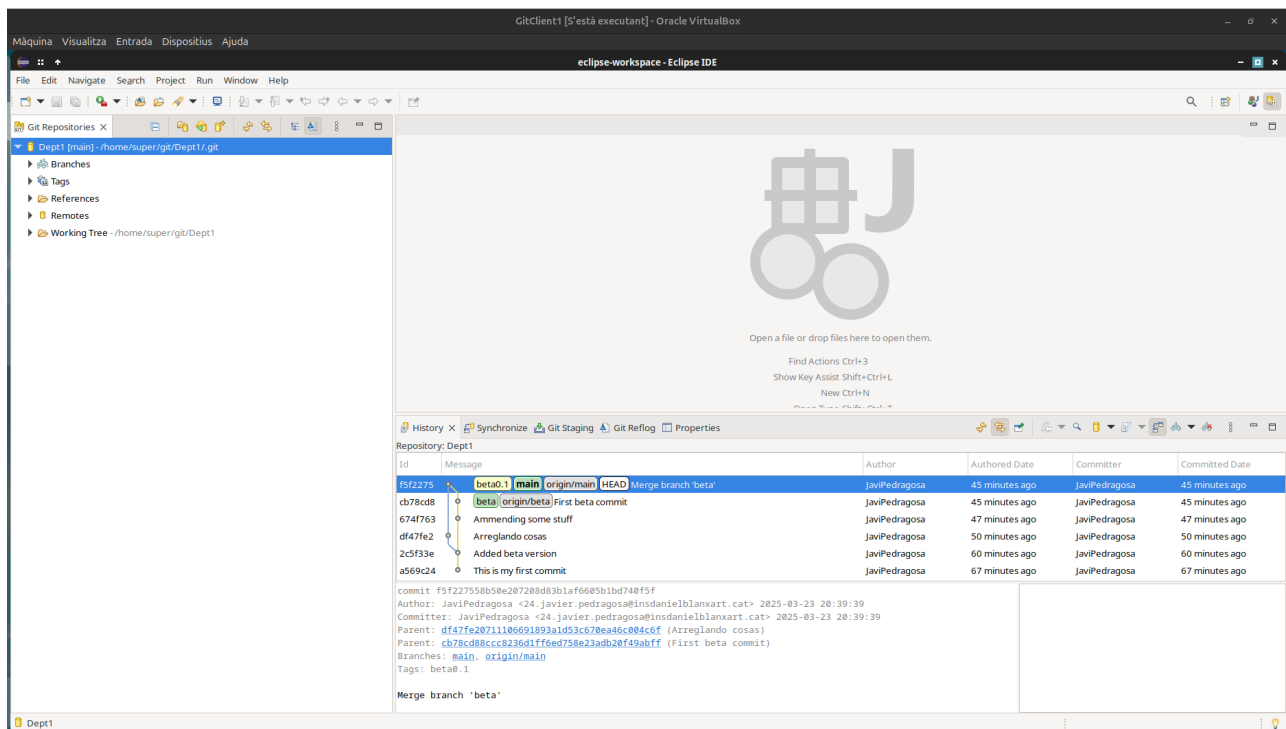


Figure 37: Eclipse showing the Dept1 Repository with its versions history

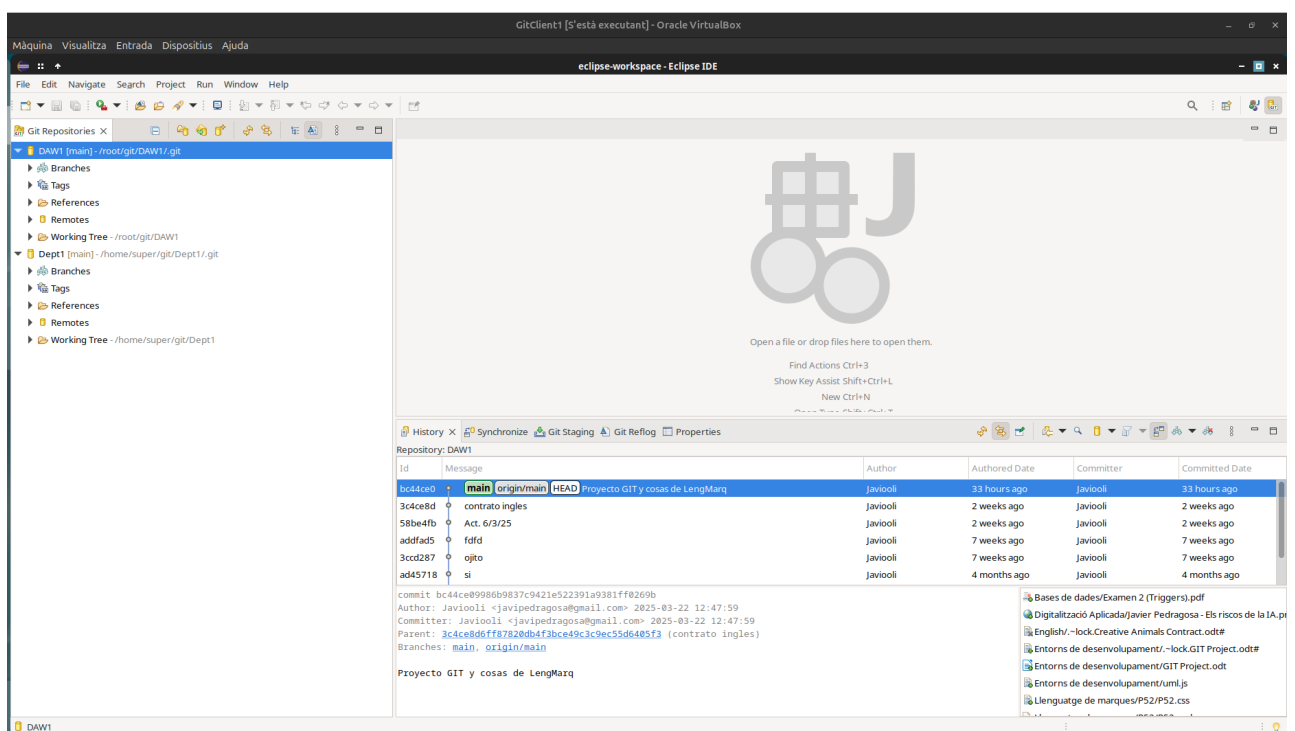


Figure 38: Eclipse showing my GitHub repository which can be found on <https://www.github.com/Javiooli/DAW1>