UNIT 2: Control Version Systems. GIT

LAB 1: Getting started

1. Installing and setting up GIT

To install the basic Git tools run the following command:

sudo apt install git-all

The first thing when you create a Git directory is to set your user name and email address. This is important because every Git commit uses this information, and it's immutably baked into the commits you start creating:

git config --global user.name "yourname"
git config --global user.email yourname@correo.com

You can also configure the default text editor that will be used when Git needs you to type in a message. If not configured, Git uses your system's default editor. If you want to use a different text editor, such as "pluma", you can do the following:

git config --global core.editor pluma

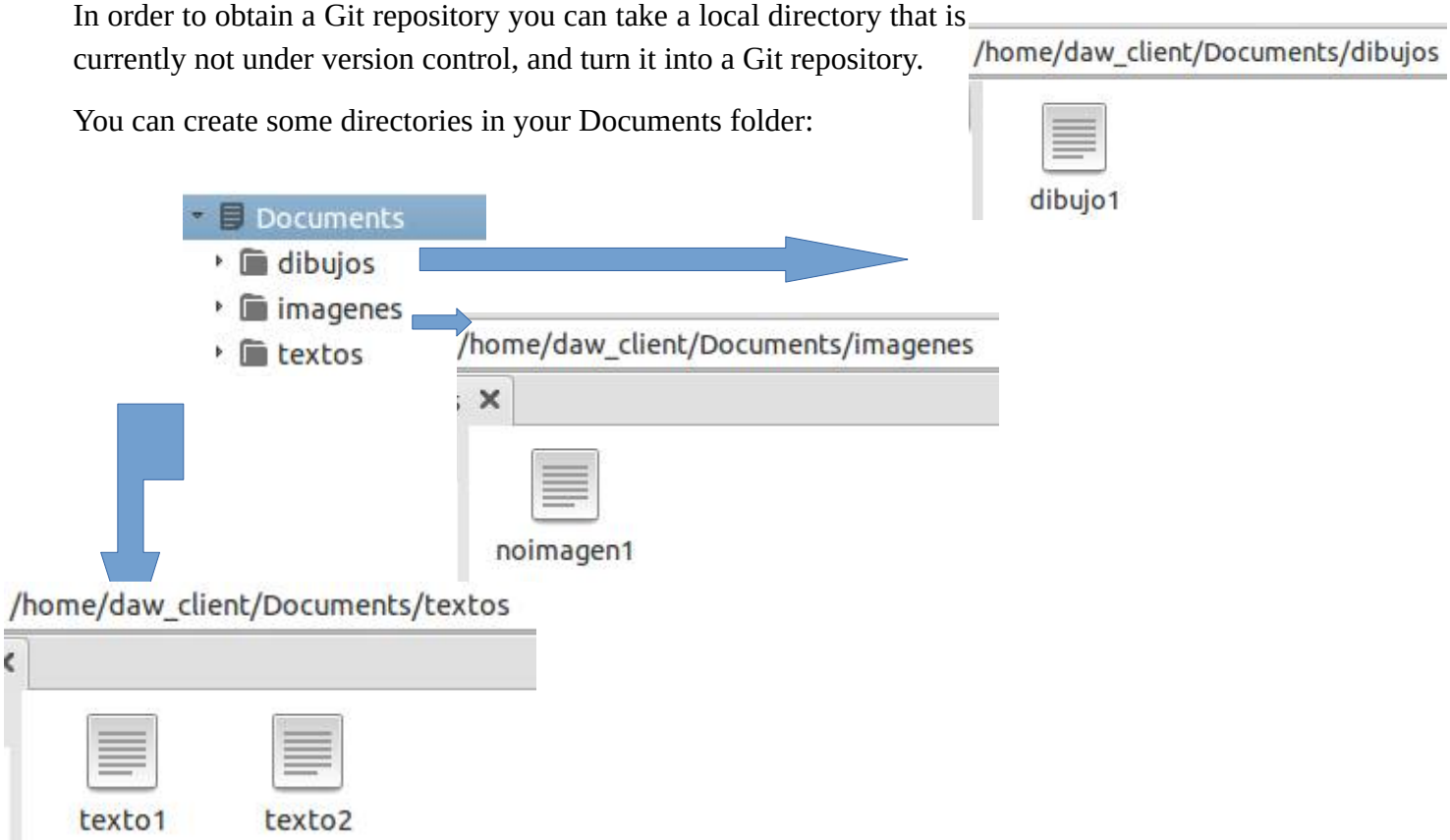
If you want to check your configuration settings, you can use the following command:

git config --list

2. Getting a GIT repository

In order to obtain a Git repository you can take a local directory that is currently not under version control, and turn it into a Git repository.

You can create some directories in your Documents folder:



You have to write the following commands to get this directory tree:

cd /home/daw_client/Documents
mkdir textos
mkdir imagenes
mkdir dibujos
cd textos
cat >texto1
Este es texto1 v.0
Crl+C
cat >texto2
Este es texto2 v.0
Ctrl+C
cd ../imagenes
cat >noimagen1
Este no es una imagen v.0
Crl+C
cd ../dibujos
cat >dibujo1
No voy a durar mucho v.0
Crl+C

Now, you have to go to the */home/daw_client/Documents directory and run:*

git init

Now configure your directory as above.

This creates a new subdirectory named "`.git`" that contains all of your necessary repository files, a Git repository skeleton. At this point, nothing in your project is tracked yet.
To start version-controlling, you should probably begin tracking the files you want to be tracked, and do an initial commit. You can accomplish that with a few ***`git add`*** commands that specify the files you want to track, followed by a ***`git commit.`*** As we want to track everything in Documents folder, we will run:

cd /home/daw-client/Documents
git add .

If we run ***git status*** we will get

```
daw_client@daw-client:~/Documents$ git status
On branch master

Initial commit

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

        new file:   dibujos/dibujo1
        new file:   imagenes/noimagen1
        new file:   textos/texto1
        new file:   textos/texto2
```

To create the first commit we will run:

git commit -m "Documents version v.0"

3. Recording changes to the repository

From now on, we are going to start making changes and committing snapshots of those changes into the repository each time the project reaches a state we want to record.

a) Add "Este fichero se modifica para la v.1" at the end of "*texto2" file*
b) Create "dibujo2" file in "dibujos" directory.
c) To see all the changes not recorded yet, run "git status"

```
daw_client@daw-client:~/Documents$ git status
On branch master
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:   textos/texto2

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

        dibujos/dibujo2
        textos/texto2~

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

We can see that we have a modified file: texto2, a new file: dibujo2, untracked yet, and a temporary file texto2. Neither texto2 nor dibujo2 are staged, to staged them we will run:

cd /home/daw_client/Documents
git add textos/texto2
git add dibujos/dibujo2

Now, they will be included in the next commit.

d) Delete "dibujo1" file.

To delete a tracked file, the best way is to remove it not only from the disk but also from the tracked file list. The following command get these purpose.

git rm /home/daw_client/Documents/dibujos/dibujo1

Running again "git status" command

```
daw_client@daw-client:~/Documents$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        deleted:    dibujos/dibujo1
        new file:   dibujos/dibujo2
        modified:   textos/texto2
```

Now, we will commit all the changes:

git commit -m "Documents v1.0

To checkout the commit logs:

git log

and the list of commits is shown

```
daw_client@daw-client:~/Documents$ git log
commit fc484986b8b1ff418ae101e9f87ba4416e7c76d2
Author: luz <luz@iesabastos.org>
Date:   Wed Jan 6 03:07:31 2016 +0100

    Documents v1.0

commit 0bada92e4da9ad7bce36c43640a7df8378cd483a
Author: luz <luz@iesabastos.org>
Date:   Wed Jan 6 02:10:50 2016 +0100

    Documents version v.0
```

For further information you can run: git log -p.

Keep on going with some more changes, change something in "texto1" and "texto2" files. After that, we are going to stage and commit at the same time (-a option).

git commit -a -m "Documents v2.0"

4. Working with branches.

We are going to start working with branches. Our first branch will be "mayusculas":

git branch mayusculas
git checkout mayusculas

Now we will rewrite all the content of texto1 in capital letters and commit the changes.

gedit /home/daw_client/Documents/textos/texto1
git commit -a -m "cambio texto1 a mayusculas"

To check the branches you can use the following command:

git branch

```
daw_client@daw-client:~/Documents$ git branch
  master
* mayusculas
```

Add -v option to get more information:

```
daw_client@daw-client:~/Documents$ git branch -v
  master     78ee517 Documents v2.0
* mayusculas ddbe69d cambio texto1 a mayúsculas
```

Notice that the current branch is preceded by *. Now, come back to master branch.

git checkout master

Change the first line and add a new line to “text1”. Modify the second line of “text2”. Finally commit the changes.

git commit -a -m “commit en master después de crear la rama mayúsculas”

Now, it is time to merge the two branches. We will be on master branch and run the next command:

git merge mayusculas

```
daw_client@daw-client:~/Documents$ git merge mayusculas
Auto-merging textos/texto1
CONFLICT (content): Merge conflict in textos/texto1
Automatic merge failed; fix conflicts and then commit the result.
```

As we can see, there are some problems: “Merge conflict in textos/texto1”. Git suggest us to fix conflicts ant then commit the result. To fix the conflict we have to edit the file that will show something similar to:

```
<<<<<<< HEAD
Este es texto1 v.3
Esta modificación se incluye en la v2.0
Esta nueva línea se añade en el commit de master después de hacer la
rama mayúsculas y  cambiar de minúsculas a mayúsculas las 2 primeras
líneas.
=======
ESTE ES TEXTO1 V.0
ESTA MODIFICACIÓN SE INCLUYE EN LA V2.0
>>>>>>> mayusculas
```

This means the version in HEAD (master branch), is the top part of that block (everything above the =======), while the version in “mayusculas” branch looks like everything in the bottom part. In order to resolve the conflict, you have to either choose one side or the other or merge the contents yourself, Then, commit the changes:

git commit -a

If we are not going to use the “mayusculas” branch any more, we can delete it. Before deleting a branch it is convenient to see which branches are already merged into the branch you’re on, you can run the next command:

git branch - - merge

```
daw_client@daw-client:~/Documents$ git branch --merge
* master
  mayusculas
```

*Branches on this list without the * in front of them are generally fine to delete; you've already incorporated their work into another branch, so you're not going to lose anything.*

git branch -d mayusculas

5. Coming back to a previous commit

Resets the index and working tree. Any changes to tracked files in the working tree since 78ee517e commit are discarded.

git reset - - hard 78ee517e

6. .gitignore files

From time to time, there are files untracked that Git should ignore. A gitignore file specifies intentionally untracked files that Git should ignore Files already tracked by Git are not affected. To stop tracking a file that is currently tracked, use *git rm --cached.*
Add a .gitignore file to avoid temporary files will be tracked, file names ending with "~" and file names ending with ".exe".

Checkout that your .gitignore file works.