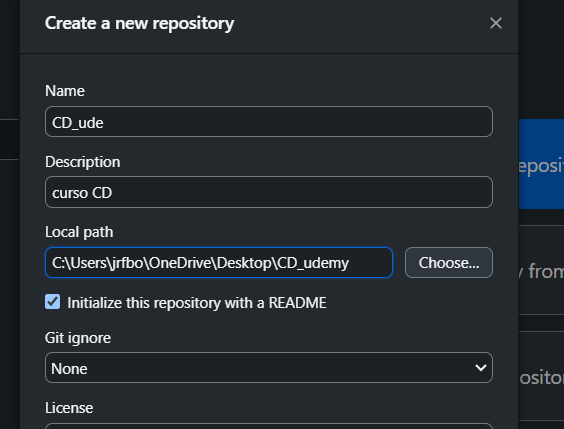
GIT

# AULA 1

O que é GIT (cabeça dura, teimoso, pensa sempre que está certo, gíria inglesa)

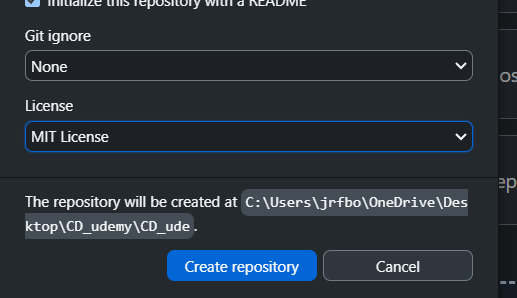
Global information Tracker

CRIAR REPOSITÓRIO COM README



Mit license

$ git config --global user.name "Your Name Comes Here"  
$ git config --global user.email “[you@yourdomain.example.com”](mailto:you@yourdomain.example.com)



Git init = comando para iniciar um git

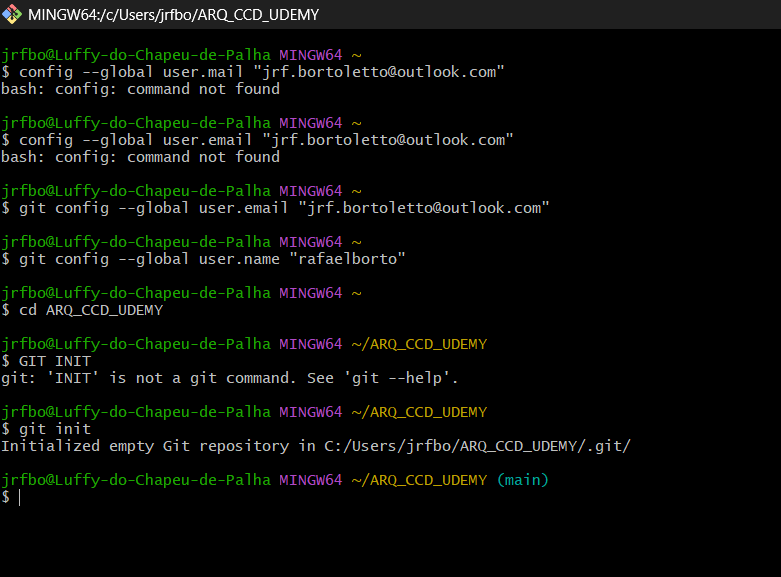
Git clone

MKDIR = cria uma pasta

CD = muda para aquela pasta

Mkdir checks (CRIA O REPOSITORIO CHECKS)

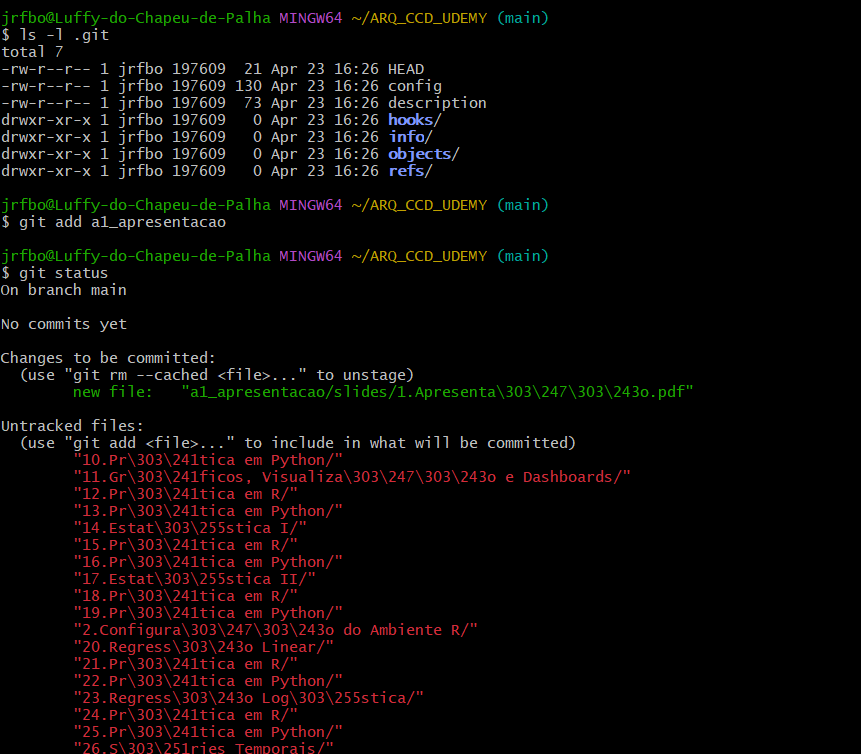
Cd checks (MUDA PARA A PASTA REPOSITORIO CHECKS)

LS –LA

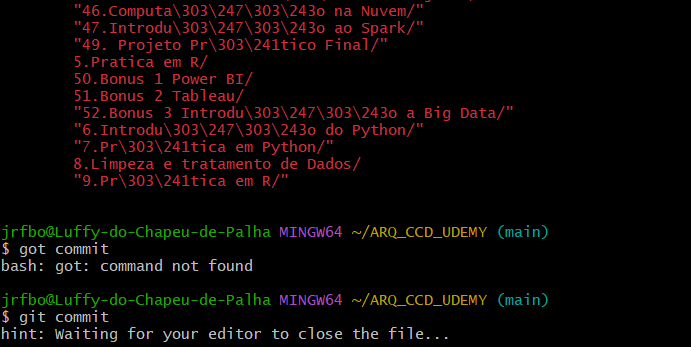


LS –L .GIT

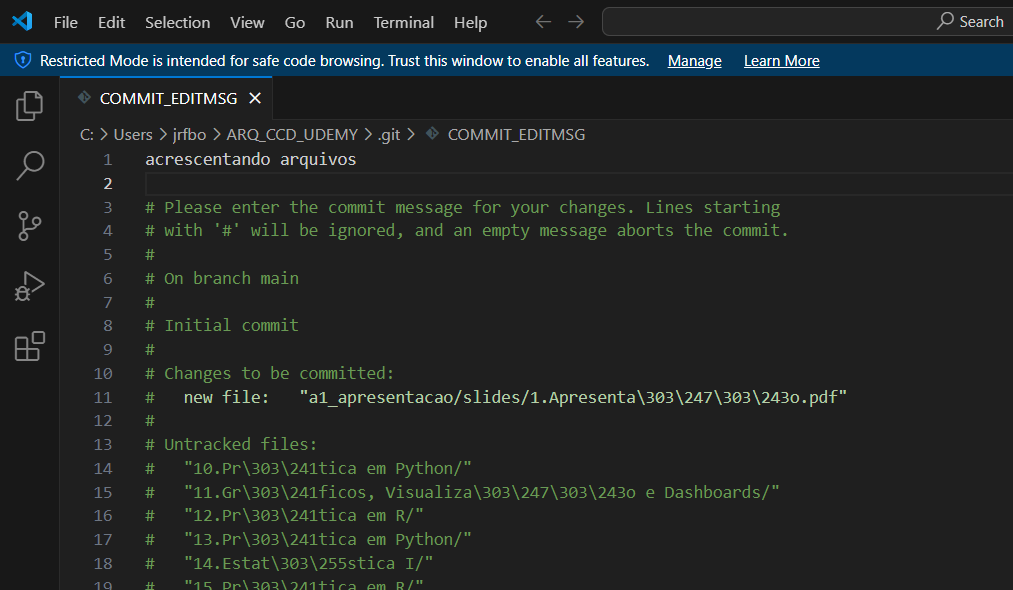
git status



Adicionei ao repositório a primeira pasta a1\_apresetancao

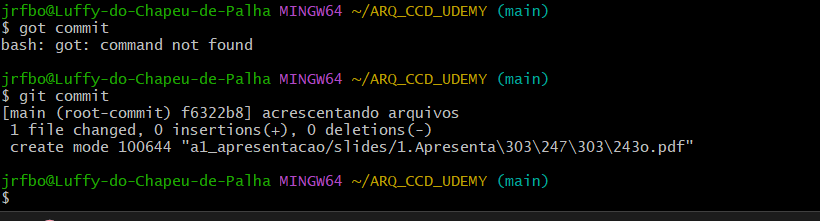


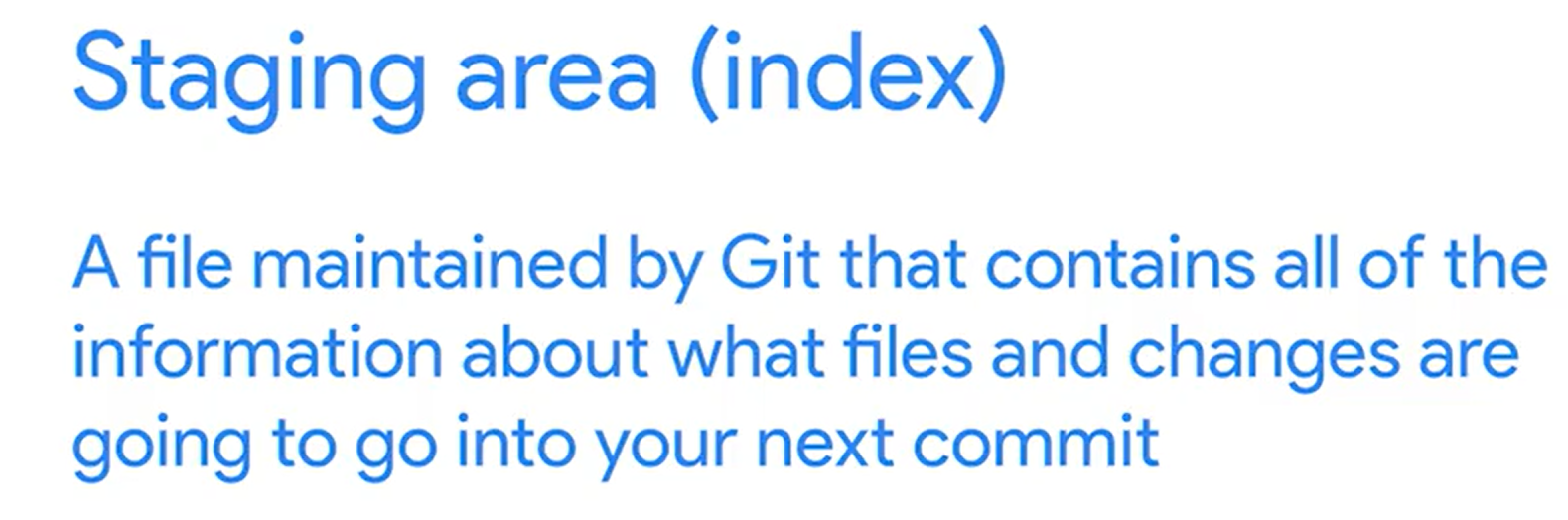
Abriu o Visual Studio



Mas há os UNTRACKED FILES, que não foram adicionados ao repositório

Só sair do editor,





# AULA 2

In our last video, we mentioned that any Git project will consist of three sections. The Git directory, the working tree, and the staging area. The Git directory contains the history of all the files and changes. The working tree contains the current state of the project, including any changes that we've made. And the staging area contains the changes that have been marked to be included in the next commit.

This can still be confusing. So it might be helpful to think about Git as representing your project. Which is the code and associated files and a series of snapshots. Each time you make a commit, Git records a new snapshot of the state of your project at that moment. It's a picture of exactly how all these files looked at a certain moment in time. Combined, these snapshots make up the history of your project, and it's information that gets stored in the Git directory.

Now, let's dive into the details of how we track changes to our files. When we operate with Git, our files can be either tracked or untracked. Tracked files are part of the snapshots, while untracked files aren't a part of snapshots yet. This is the usual case for new files. Each track file can be in one of three main states, modified, staged or committed. Let's look at what each of these mean.

Play video starting at :1:19 and follow transcript1:19

If a file is in the modified state, it means that we've made changes to it that we haven't committed yet. The changes could be adding, modifying or deleting the contents of the file. Git notices anytime we modify our files. But won't store any changes until we add them to the staging area.

Play video starting at :1:37 and follow transcript1:37

So, the next step is to stage those changes.

Play video starting at :1:40 and follow transcript1:40

When we do this, our modified files become stage files. In other words, the changes to those files are ready to be committed to the project. All files that are staged will be part of the next snapshot we take. And finally, when a file gets committed, the changes made to it are safely stored in a snapshot in the Git directory.

Play video starting at :2:1 and follow transcript2:01

This means that typically a file tracked by Git, will first be modified when we change it in any way. Then it becomes staged when we mark those changes for tracking. And finally it will get committed when we store those changes in the VCS. Let's see this in action in our example Git repo. First, let's check the contents of the current working tree using ls-l. And then the current status of our files using t the Git status command. When we run Git status, Git tells us a bunch of things, including that we're on the master branch. We'll learn about branches later in the course. For now, notice how it says that there's nothing to commit and that the working tree is clean. Let's modify a file to change that.

Play video starting at :2:50 and follow transcript2:50

For example, we'll just add periods at the end of the message that our script presents to the user.

Play video starting at :3:25 and follow transcript3:25

So, now that we've made the change, let's call Git status again and see the new output. Again, Git tells us a lot of things, including giving us some tips for commands that we might want to use. These tips can come in real handy, especially when we're familiarizing ourselves with Git. See how the file we changed is now marked as modified? And that it's currently not staged for commit?

Play video starting at :3:51 and follow transcript3:51

Let's change that by running the Git add command, passing the disk usage py file as a parameter.

Play video starting at :4:4 and follow transcript4:04

When we call Git add, we're telling Git that we want to add the current changes in that file to the list of changes to be committed. This means that our file is currently part of the staging area, and it will be committed once we run the next Git command, Git commit. In this case, instead of opening up an editor, let's pass the commit message using the dash m flag, stating that we added periods at the end of the sentences.

Play video starting at :4:38 and follow transcript4:38

So, we've now committed our stage changes.

This creates a new snapshot in the Git directory.

Play video starting at :4:45 and follow transcript4:45

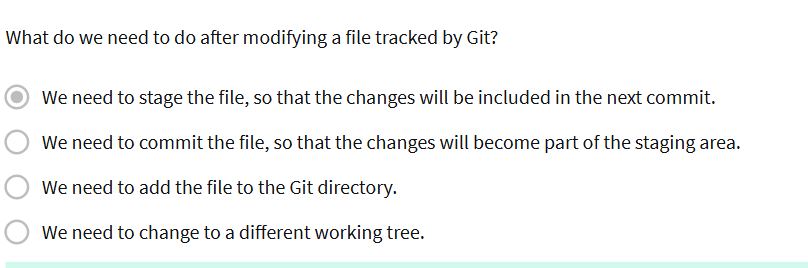
The command shows us some stats for the change made. Let's do one last status check.

Play video starting at :4:54 and follow transcript4:54

We see that once again, we have no changes to commit. Because the change we made has gone through the full cycle of modified, staged and committed. So to sum up, we work on modified files in our working tree. When they're ready, we staged these files by adding them to the staging area. Finally, we commit the changes sitting in our staging area, which takes a snapshot of those files and stores them in the database that lives in the Git directory. If the way Git works is not totally clear yet, don't worry. It will all sink in with a bit more practice.

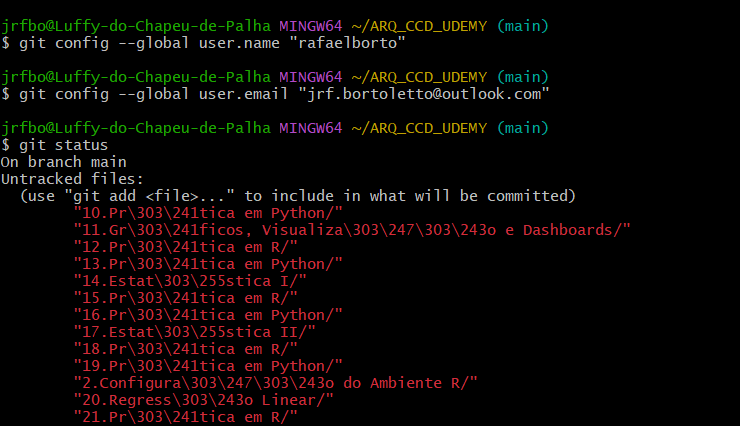
Play video starting at :5:27 and follow transcript5:27

In our next video, we'll put this all together and go over the typical workflow when working with Git.

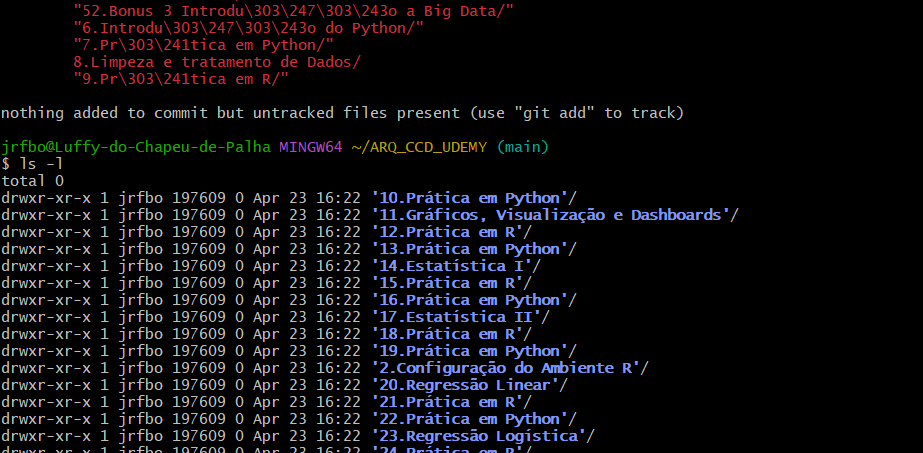


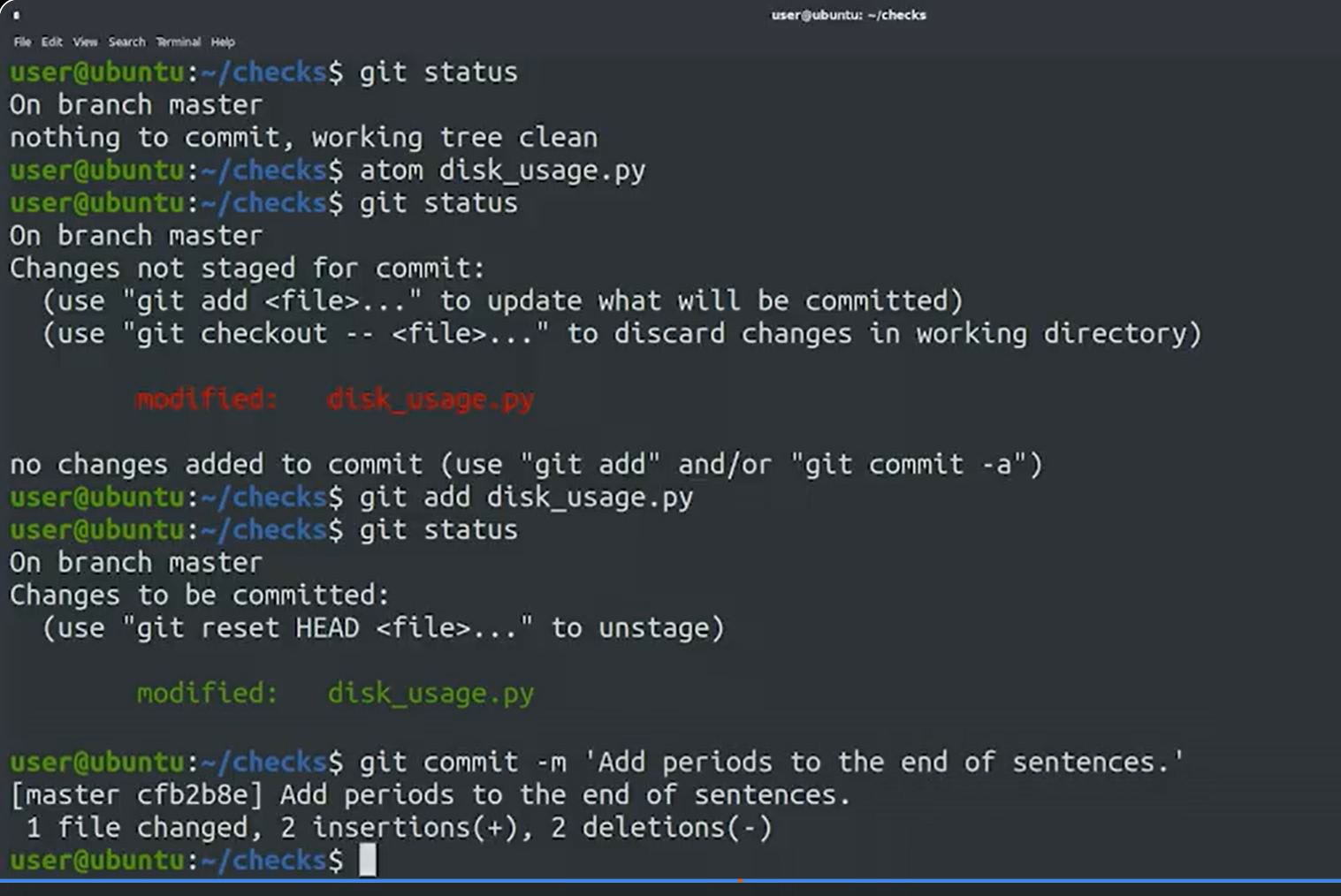
ENTREI DIRETAMENTE NO GIT BASH PELA PASTA

Usei o GIT STATUS



E O GIT LS –L





# AULA 3

mkdir scripts

cd scripts

git init

**Code output:**

Initialized empty Git repository in /home/user/scripts/.git/

git config -l

**Code output:**

[user.email=me@example.com](mailto:user.email=me@example.com)

user.name=My name

core.repositoryformatversion=0

core.filemode=true

core.bare=false

core.logallrefupdates=true

git status

git add all\_checks.py

git commit

git commit -m 'Add a check\_reboot function'