GITHUB & VSCODE

Uma introdução

01

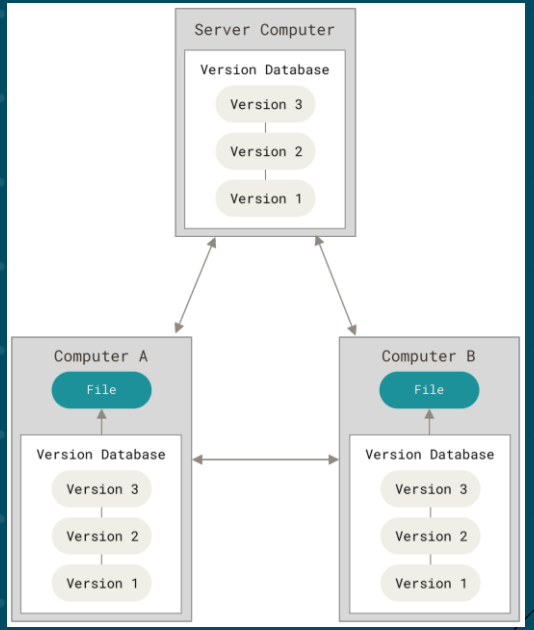
Git, Github e um pouco sobre VCSs

Version control

- Controle das mudanças ao longo do tempo
- Em um trabalho de grupo/time, várias versões de um mesmo produto podem coexistir
- Existem várias implementações em software de VCSs, como por exemplo Git, subversion, CVS, Mercurial

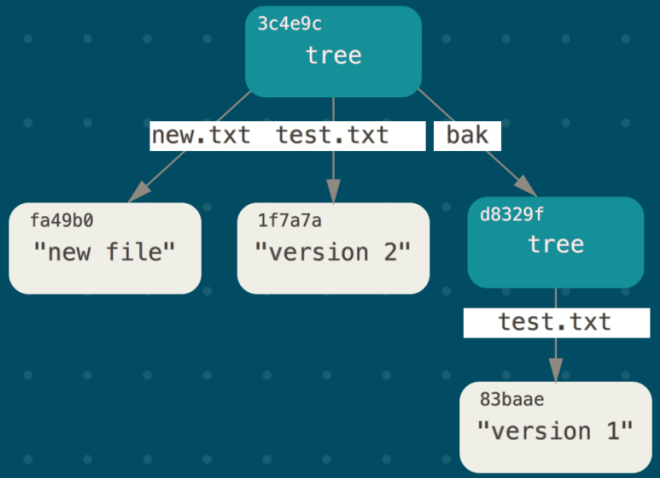
Git(Global Information Tracker)

- Mais utilizado atualmente
- Mais especificamente é um tipo de Distributed Version Control System
- Todas as operações que são feitas em uma base de dados é totalmente independente de um servidor (em seu estado mais simples)
- Git foi concebido para ser utilizado em uma linha de comandos



Git(Global Information Tracker)

- Todas as alterações são validadas com um hash (SHA-1), que é calculada com base no conteúdo dos arquivos
- Tem uma integração grande com as Interfaces de desenvolvimento (IDEs) atualmente como o visual studio, spyder, eclipse, etc.



Trabalhando com Git

GIT GUD

Workflow básico:

I. You modify files in your working tree

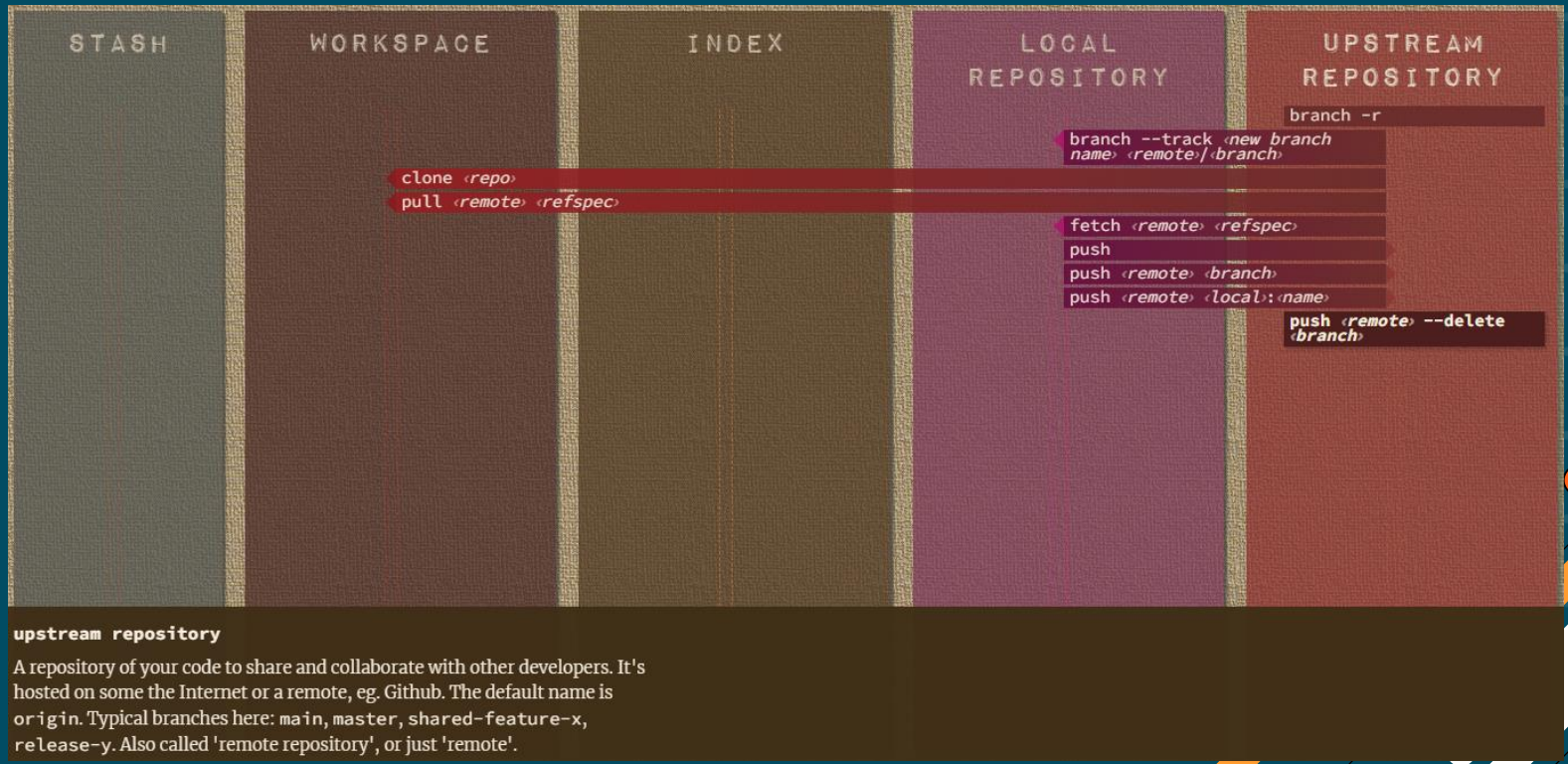
II. You selectively stage just those changes you want to be part of your next commit, which adds only those changes to the staging area

III. You do a commit, which takes the files as they are in the staging area and stores that snapshot permanently to your Git directory

Trabalhando com Git

GIT GUD

"Areas de trabalho"



Trabalhando com Git

GIT GUD

“Areas de trabalho”

Trabalhando com Git

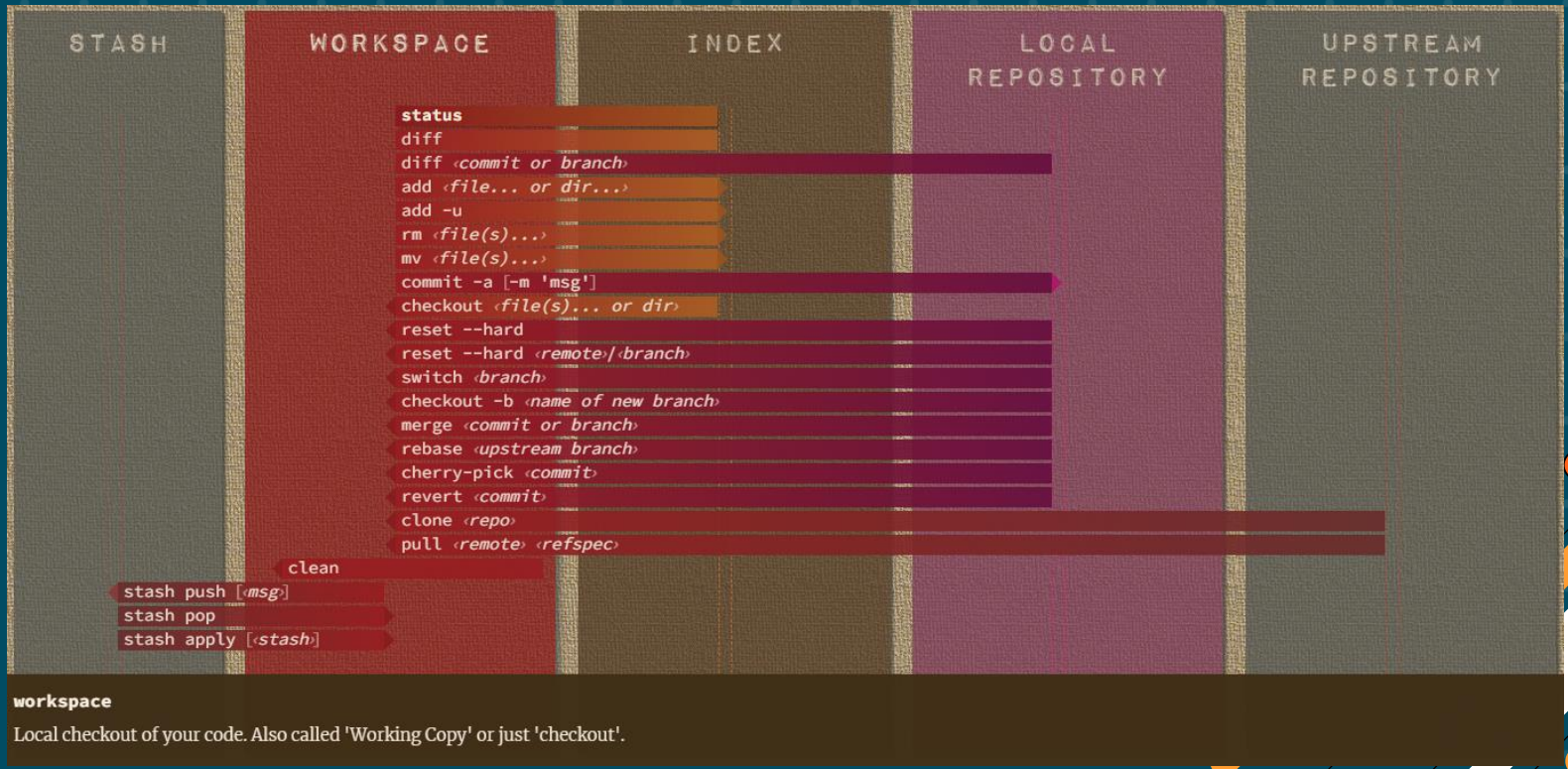
GIT GUD

“Areas de trabalho”

Trabalhando com Git

GIT GUD

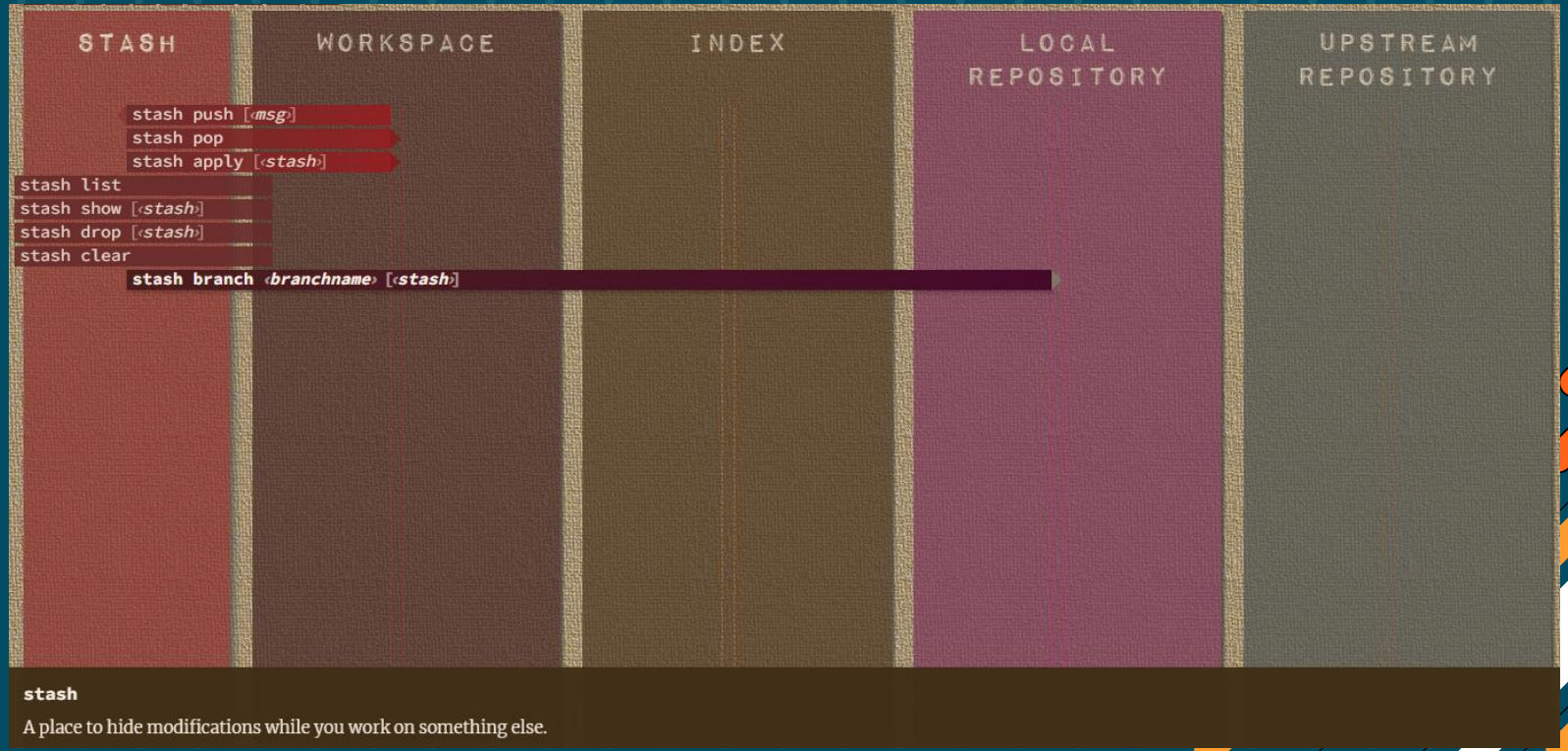
"Areas de trabalho"



Trabalhando com Git

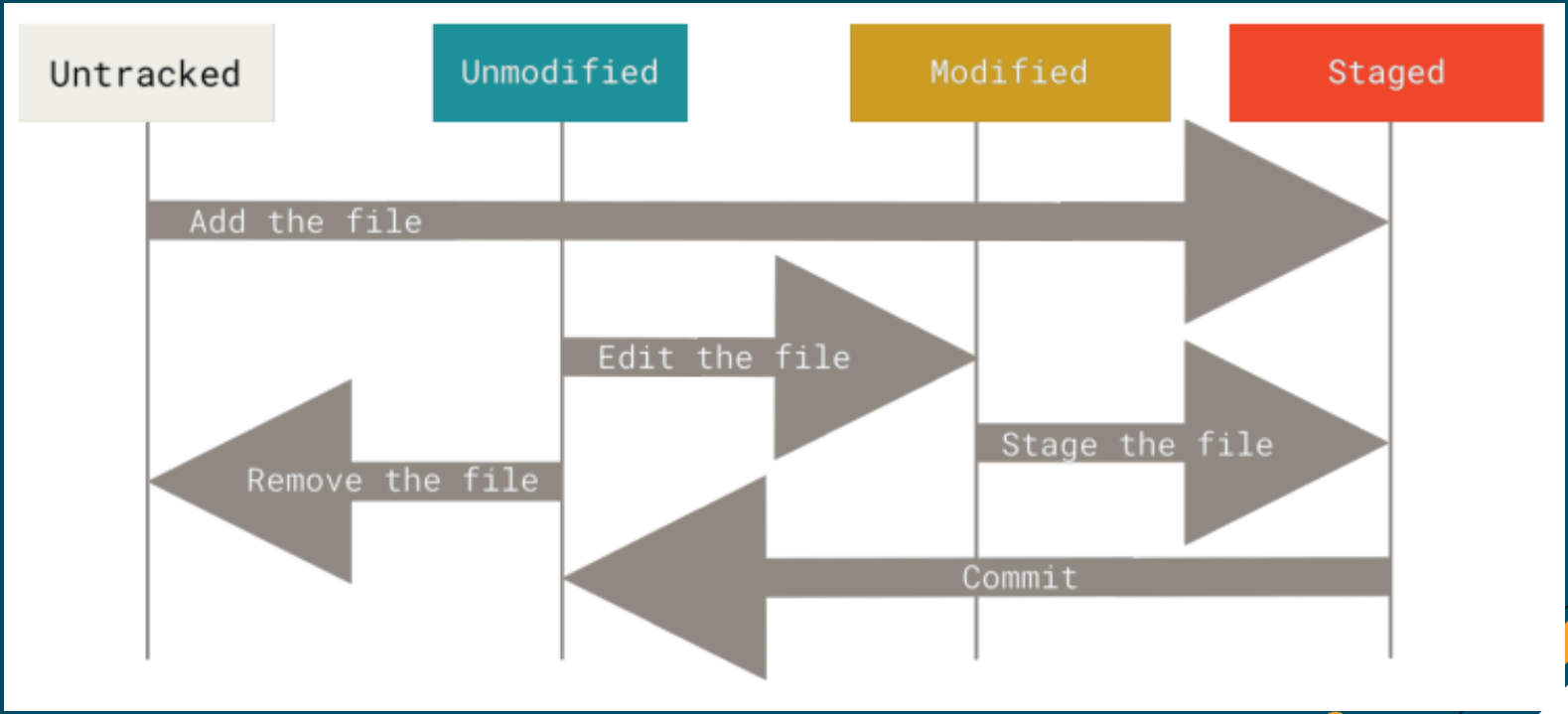
GIT GUD

"Areas de trabalho"



A vida de um arquivo

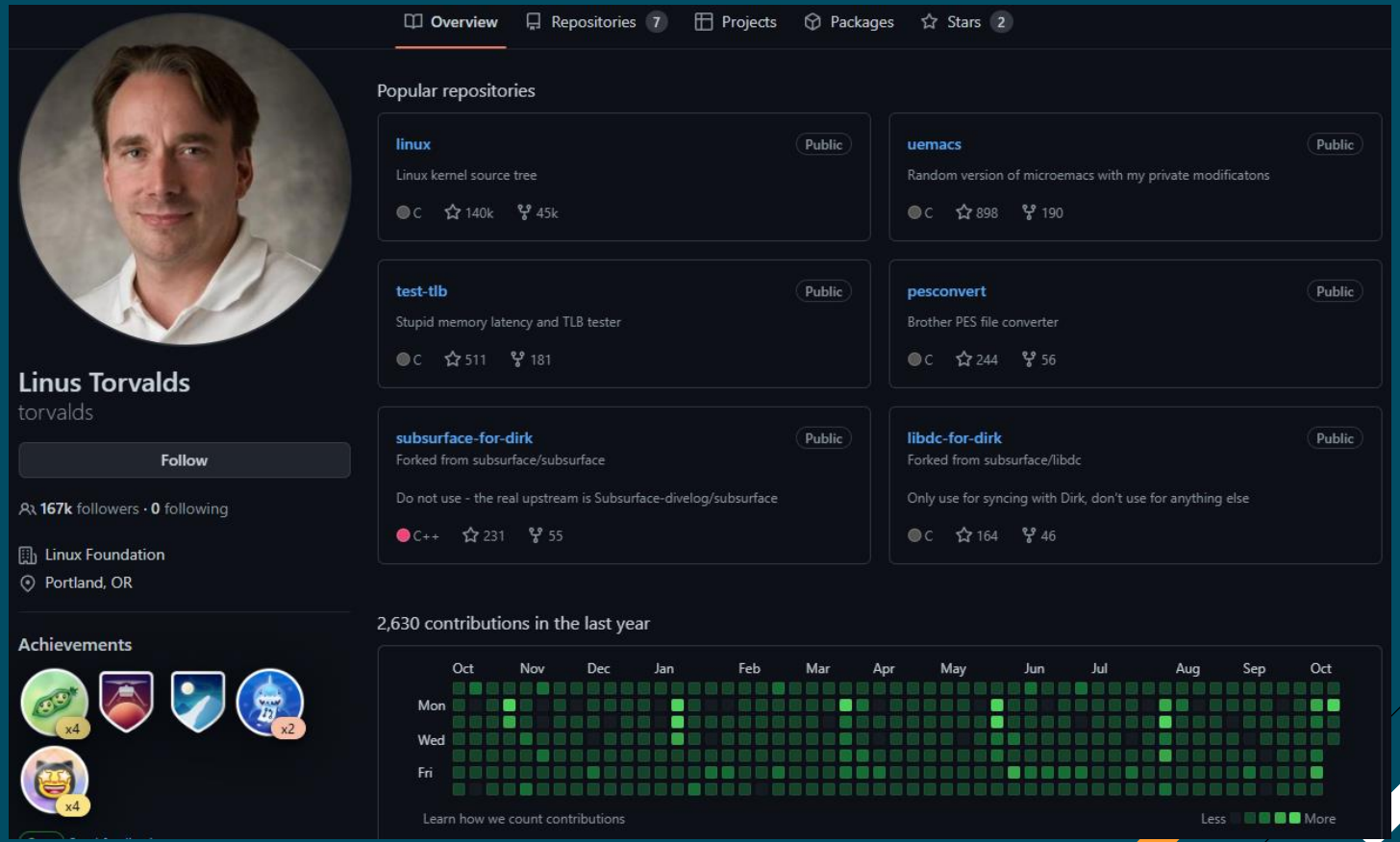
GIT
GUD




GitHub
Where the world builds software

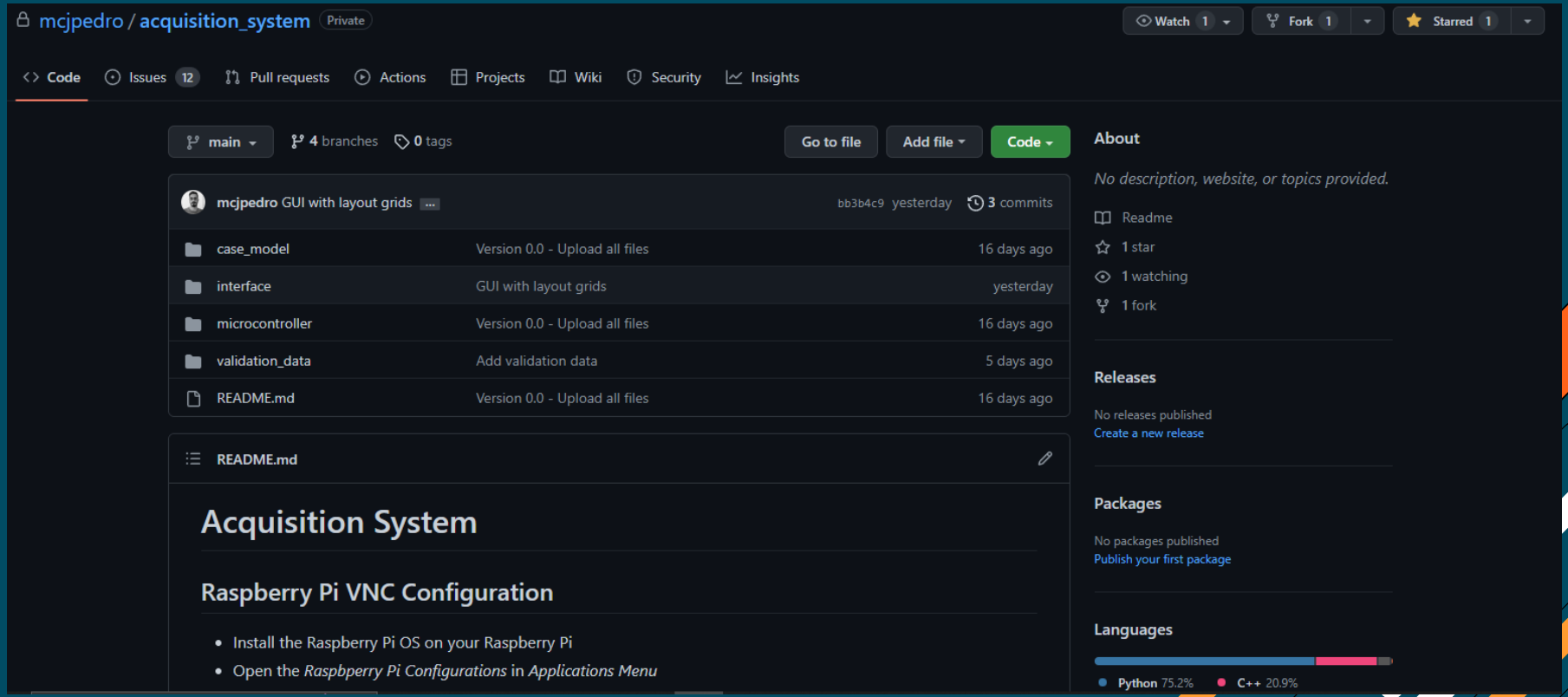
Qual é a cara do GitHub?

GIT GUD



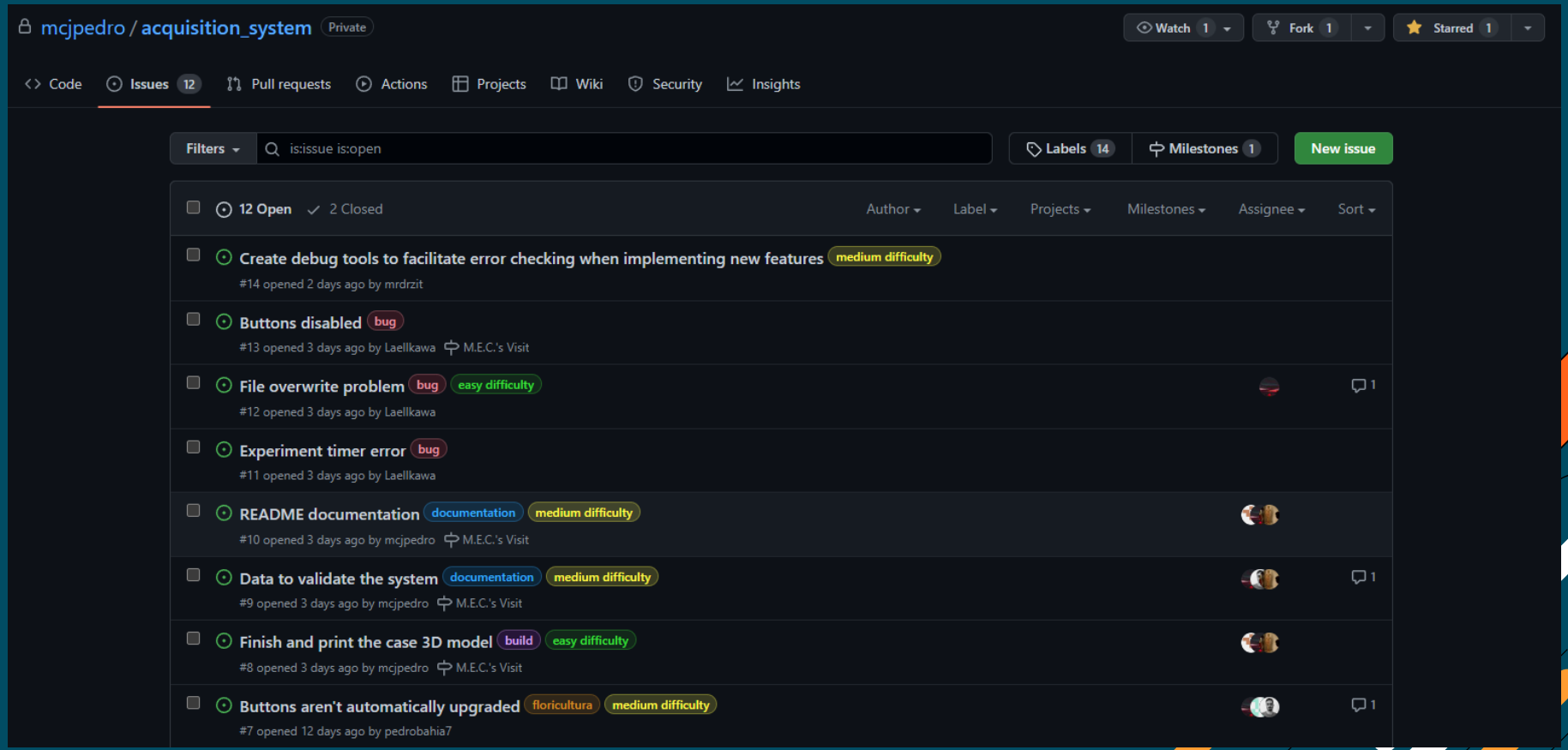
Repositórios

GIT GUD



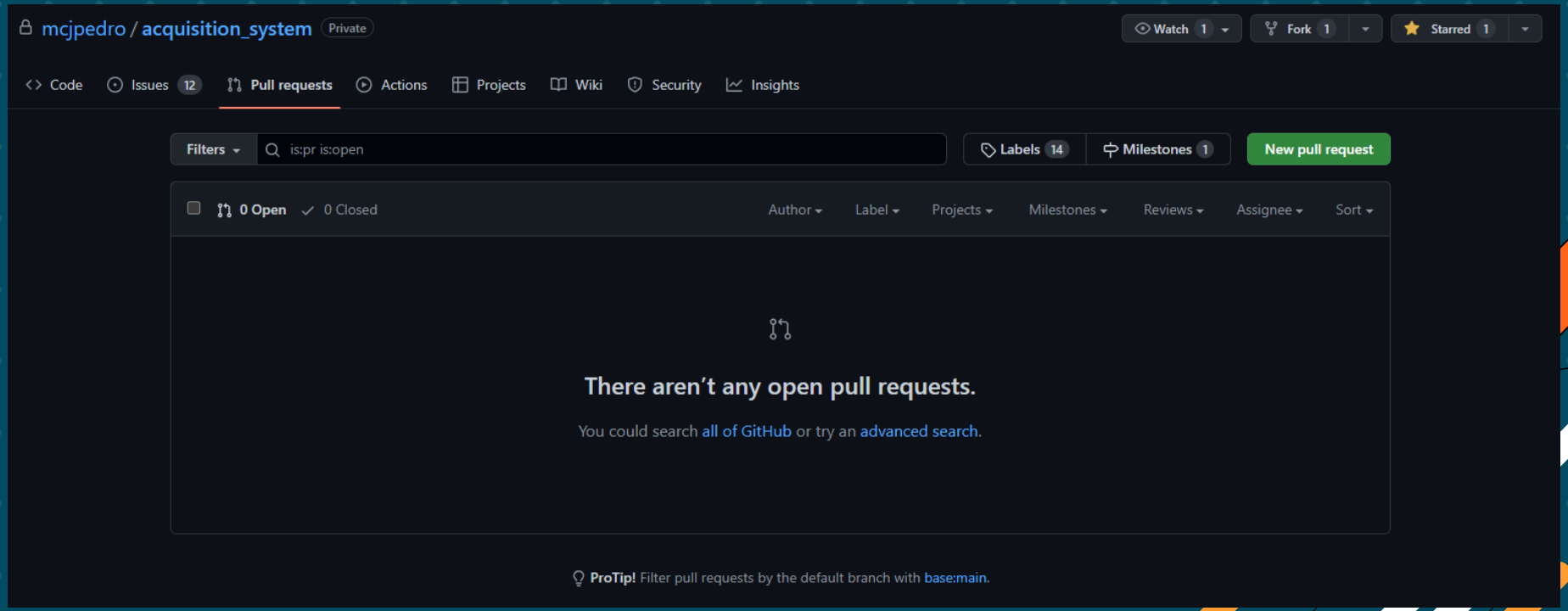
Repositórios

GIT GUD

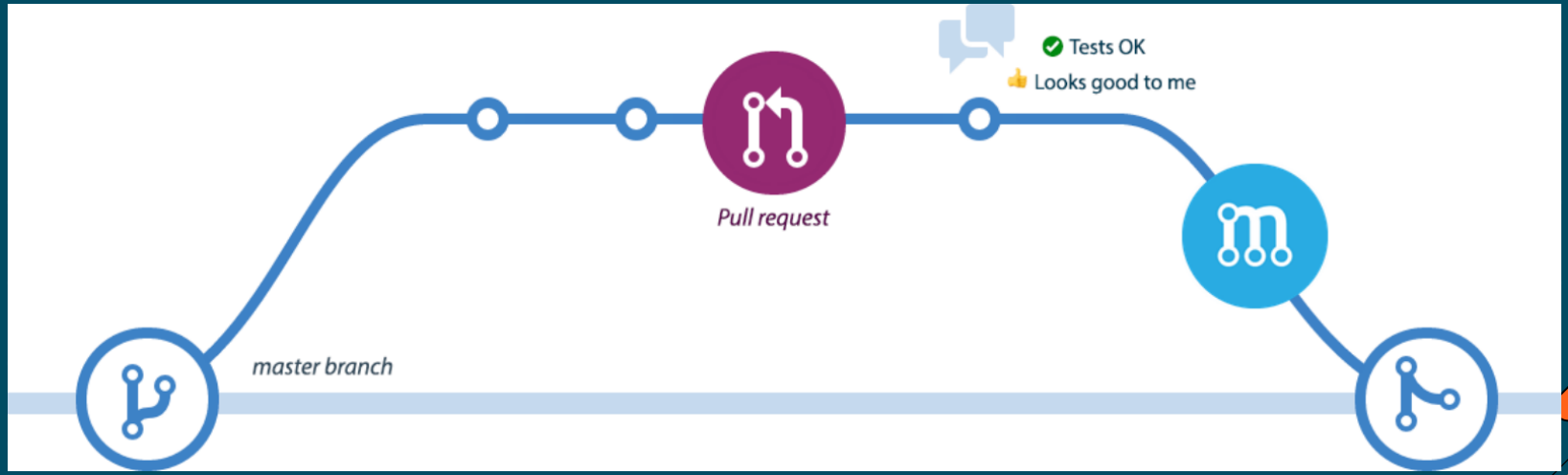


Repositórios

GIT GUD

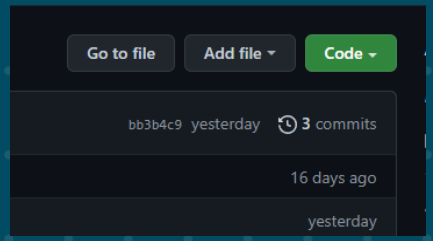


Repositórios



Repositórios

GIT GUD



Code | Issues 12 | Pull requests | Actions | Projects | Wiki | Security | Insights

main

Commits on Oct 11, 2022

GUI with layout grids ...
mcjpedro committed yesterday
bb3b4c9

Commits on Oct 7, 2022

Add validation data ...
mcjpedro committed 5 days ago
6cbee8f

Commits on Sep 26, 2022

Version 0.0 - Upload all files ...
mcjpedro committed 16 days ago
Verified
72055e2

Newer | Older

Commits padronizados

GIT GUD

Commits padronizados

- O sucesso a longo prazo de um projeto depende bastante da sua legibilidade
- Para isso é importante que um código escrito há 1 ano possa ser revisado sem mais problemas, e de forma eficiente
- git log é seu amigo!

`git log`

Show recent commits, most recent on top. Options:

`--decorate` with branch and tag names on appropriate commits

`--stat` with stats (files changed, insertions, and deletions)

`--author=‹author›` only by a certain author

`--after="MMM DD YYYY"` ex. (`Jun 20 2008`) only commits after a certain date

`--before="MMM DD YYYY"` only commits that occur before a certain date

`--merge` only the commits involved in the current merge conflicts

Commits padronizados

GIT
GUD

Commits padronizados

GIT
GUD

The seven rules of a great Git commit message

> *Keep in mind: This has all been said before.*

1. Separate subject from body with a blank line
2. Limit the subject line to 50 characters
3. Capitalize the subject line
4. Do not end the subject line with a period
5. Use the imperative mood in the subject line
6. Wrap the body at 72 characters
7. Use the body to explain *what* and *why* vs. *how*

Commits padronizados

The seven rules of a great Git commit message

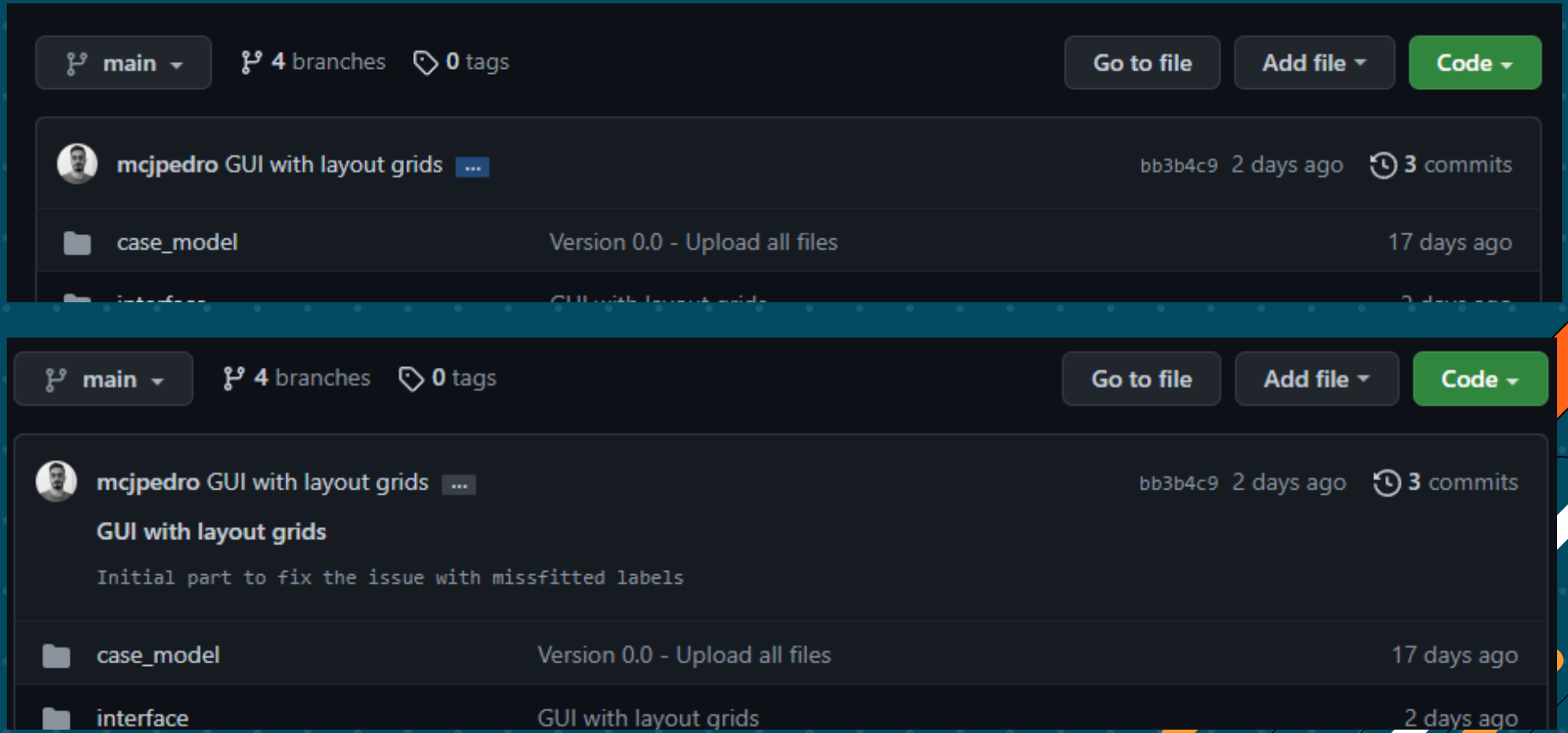
> *Keep in mind: This has all been said before.*

1. Separate subject from body with a blank line
2. Limit the subject line to 50 characters
3. Capitalize the subject line
4. Do not end the subject line with a period
5. Use the imperative mood in the subject line
6. Wrap the body at 72 characters
7. Use the body to explain *what* and *why* vs. *how*

```
# Title: Summary, imperative, start upper case, don't end with a period
# No more than 50 chars. #### 50 chars is here:  #

# Remember blank line between title and body.

# Body: Explain *what* and *why* (not *how*). Include task ID (Jira issue).
# Wrap at 72 chars. ################################ which is here:  #

# At the end: Include Co-authored-by for all contributors.
# Include at least one empty line before it. Format:
# Co-authored-by: name <user@users.noreply.github.com>
#
# How to Write a Git Commit Message:
# https://chris.beams.io/posts/git-commit/
#
# 1. Separate subject from body with a blank line
# 2. Limit the subject line to 50 characters
# 3. Capitalize the subject line
# 4. Do not end the subject line with a period
# 5. Use the imperative mood in the subject line
# 6. Wrap the body at 72 characters
# 7. Use the body to explain what and why vs. how

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch master
# Your branch is up to date with 'origin/main'.
#
# Changes to be committed:
#       new file:   installation.md
#
```

Commits padronizados

GIT GUD



Commits padronizados

GIT GUD

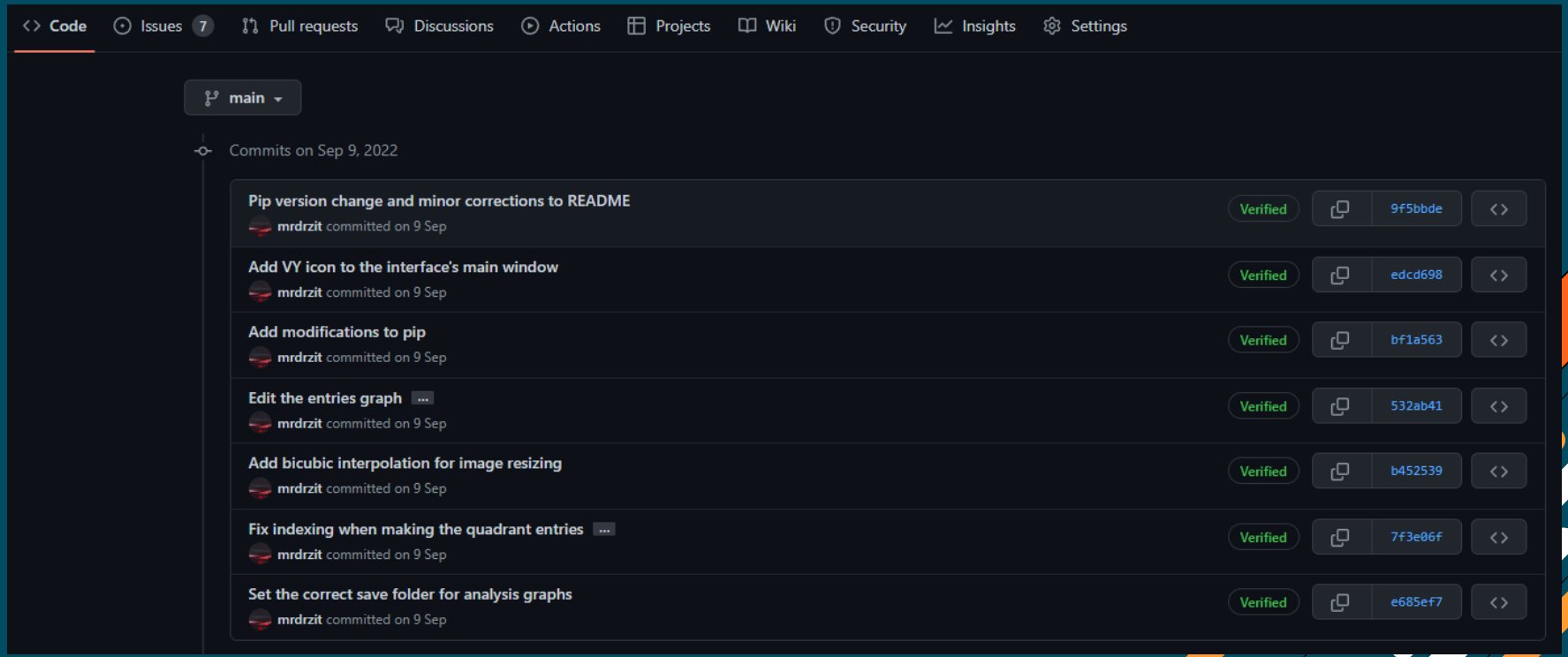
7. Use the body to explain what and why vs. how

This **commit from Bitcoin Core** is a great example of explaining what changed and why:

```
commit eb0b56b19017ab5c16c745e6da39c53126924ed6
Author: Pieter Wuille <pieter.wuille@gmail.com>
Date:   Fri Aug 1 22:57:55 2014 +0200

   Simplify serialize.h's exception handling

   Remove the 'state' and 'exceptmask' from serialize.h's stream
   implementations, as well as related methods.

   As exceptmask always included 'failbit', and setstate was always
   called with bits = failbit, all it did was immediately raise an
   exception. Get rid of those variables, and replace the setstate
   with direct exception throwing (which also removes some dead
   code).

   As a result, good() is never reached after a failure (there are
   only 2 calls, one of which is in tests), and can just be replaced
   by !eof().

   fail(), clear(n) and exceptions() are just never called. Delete
   them.
```

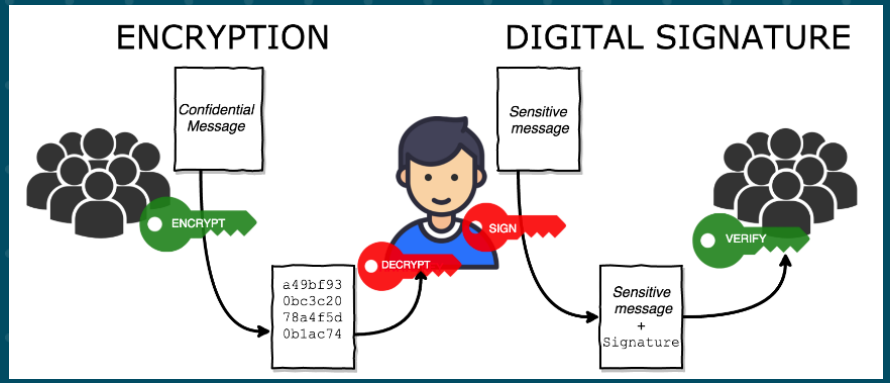
Assinatura de commits

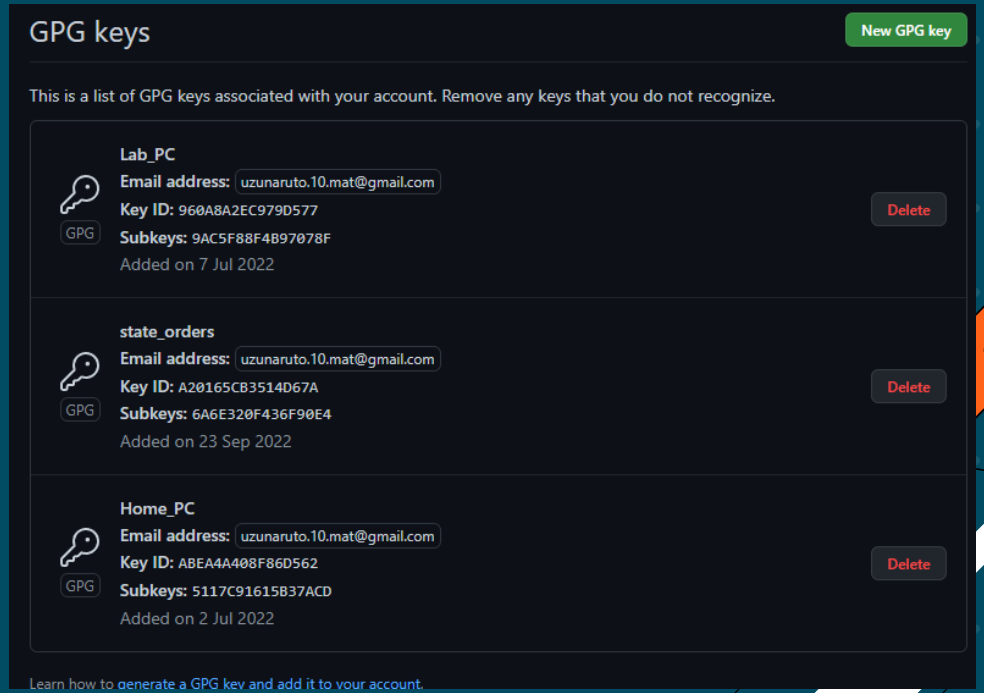
GIT GUD



Assinatura de commits

GIT GUD





Assinatura de commits

GIT GUD

C:\Users\uzuna\.gitconfig - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

.gitconfig .gitmessage

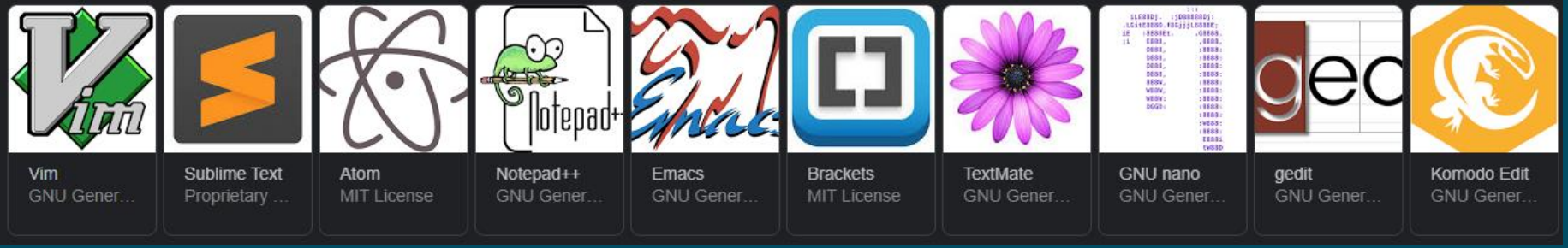
```
1  [core]
2    editor = \"D:\\Installs\\Microsoft VS Code\\Code.exe\" --wait
3  [user]
4    name = mrdrzit
5    email = uzunaruto.10.mat@gmail.com
6    signingkey = ABEA4A408F86D562
7  [gui]
8    recentrepo = C:/Users/uzuna/Documents/Github Projects/Collabs/Behavython
9  [commit]
10   gpgsign = true
11   template = C:\\Users\\uzuna\\.gitmessage
12 [gpg]
13   program = C:\\Program Files (x86)\\GnuPG\\bin\\gpg.exe
```

02

Vscode e editores de texto

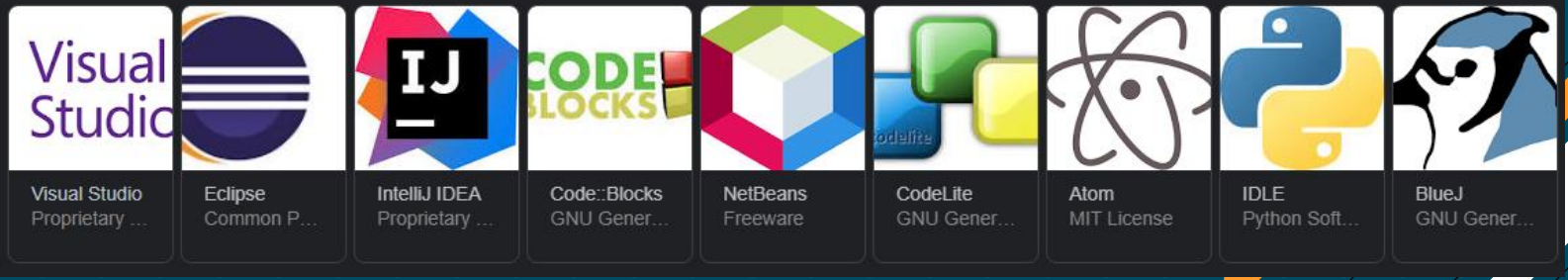
O que é Vscode?

Text editor Software

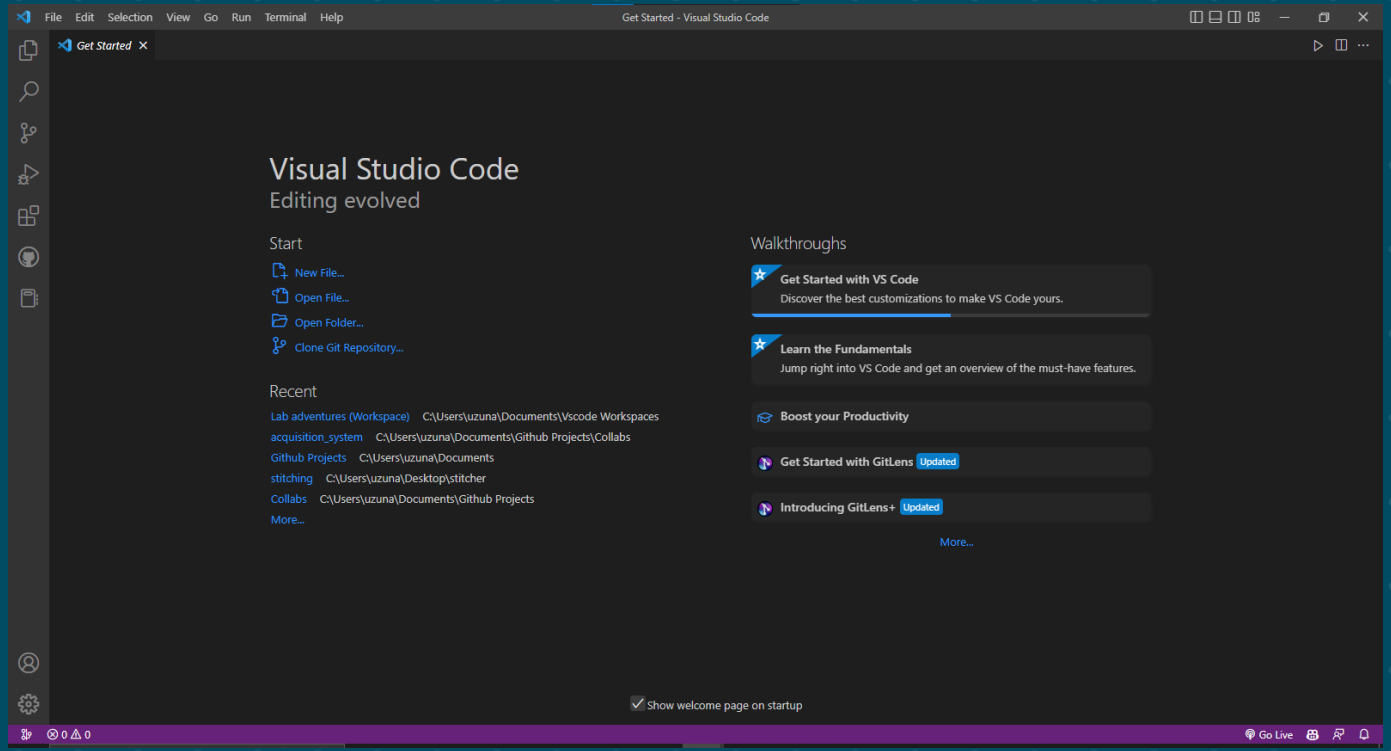


≠

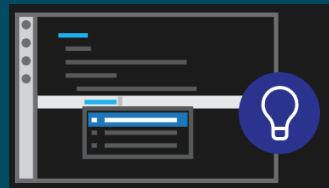
Integrated development environment Software



O que é Vscode?



O que é Vscode?

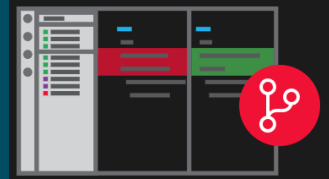


Meet IntelliSense.

Go beyond syntax highlighting and autocomplete with IntelliSense, which provides smart completions based on variable types, function definitions, and imported modules.

Print statement debugging is a thing of the past.

Debug code right from the editor. Launch or attach to your running apps and debug with break points, call stacks, and an interactive console.

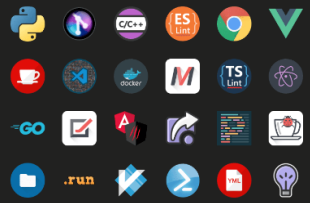


Git commands built-in.

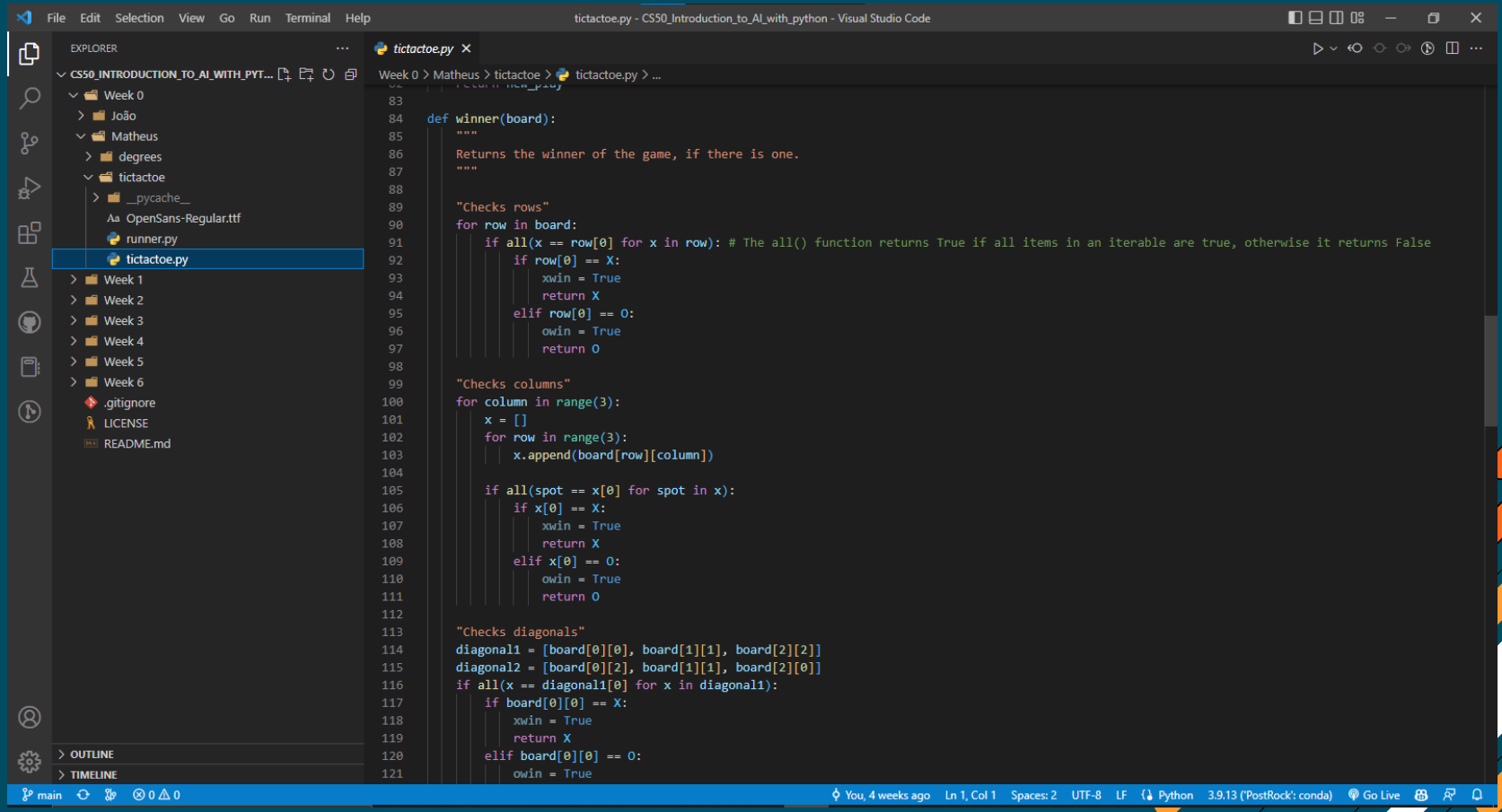
Working with Git and other SCM providers has never been easier. Review diffs, stage files, and make commits right from the editor. Push and pull from any hosted SCM service.

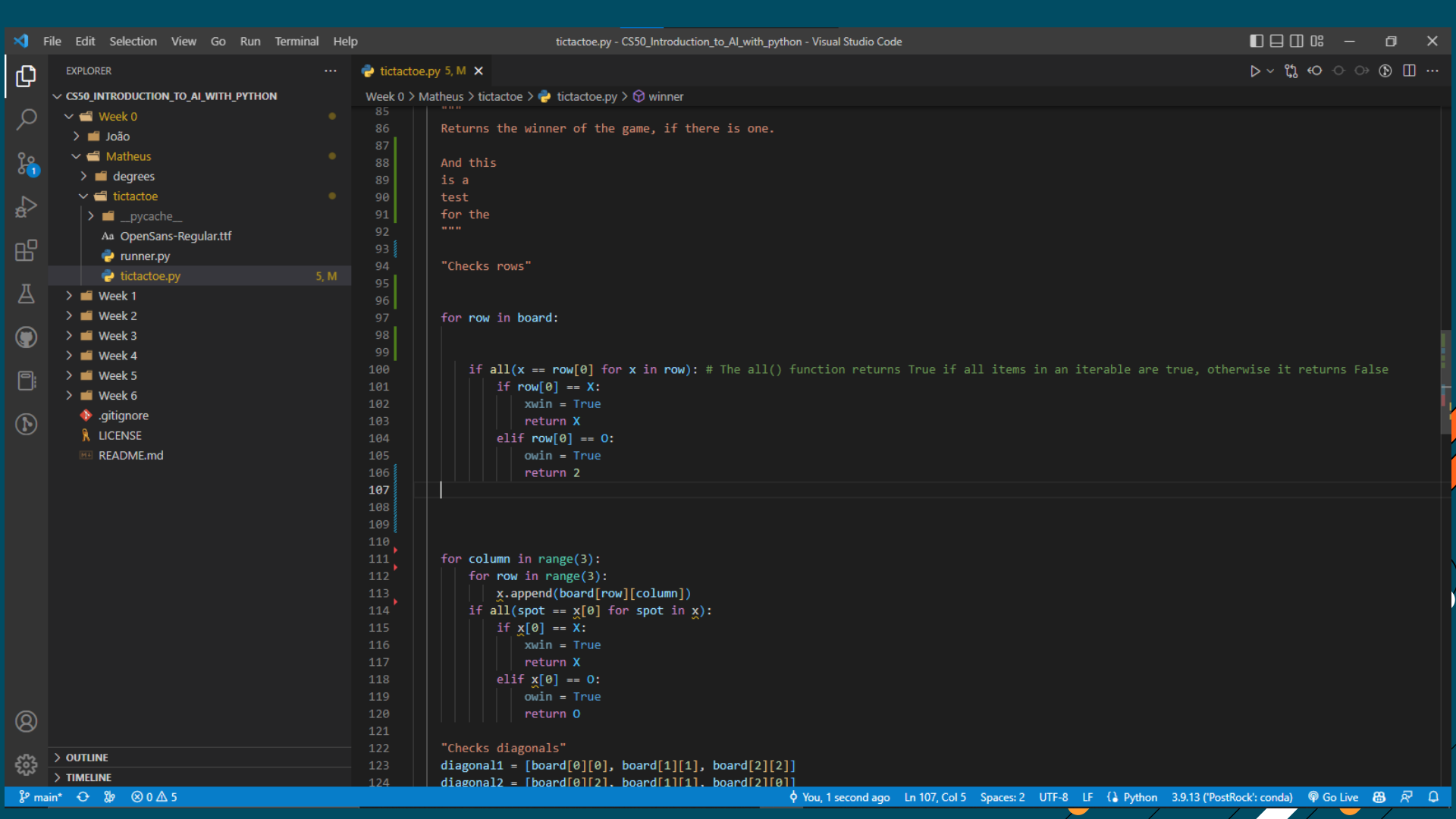
Extensible and customizable.

Want even more features? Install extensions to add new languages, themes, debuggers, and to connect to additional services. Extensions run in separate processes, ensuring they won't slow down your editor. Learn more about extensions.

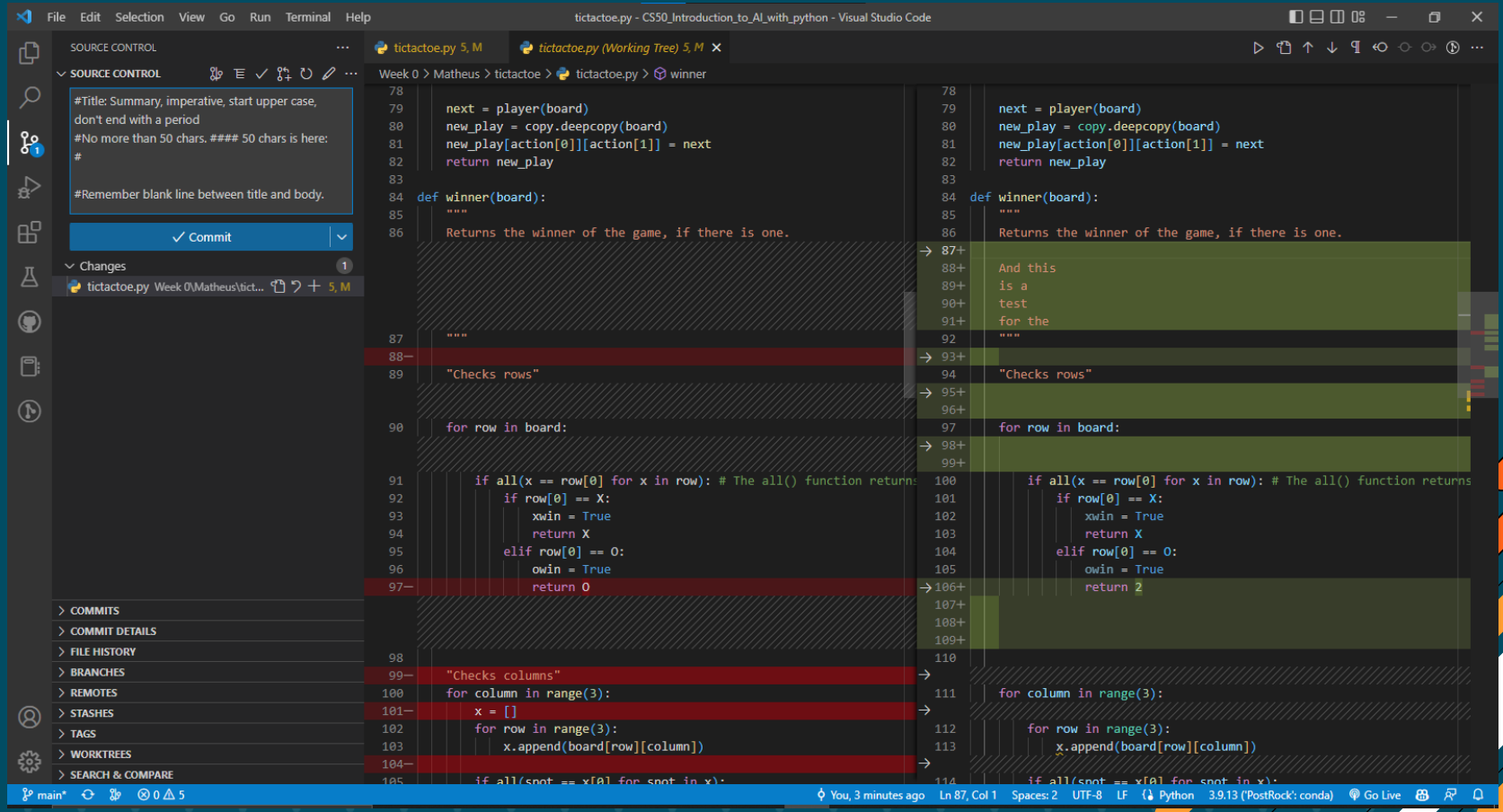


O que é Vscode?

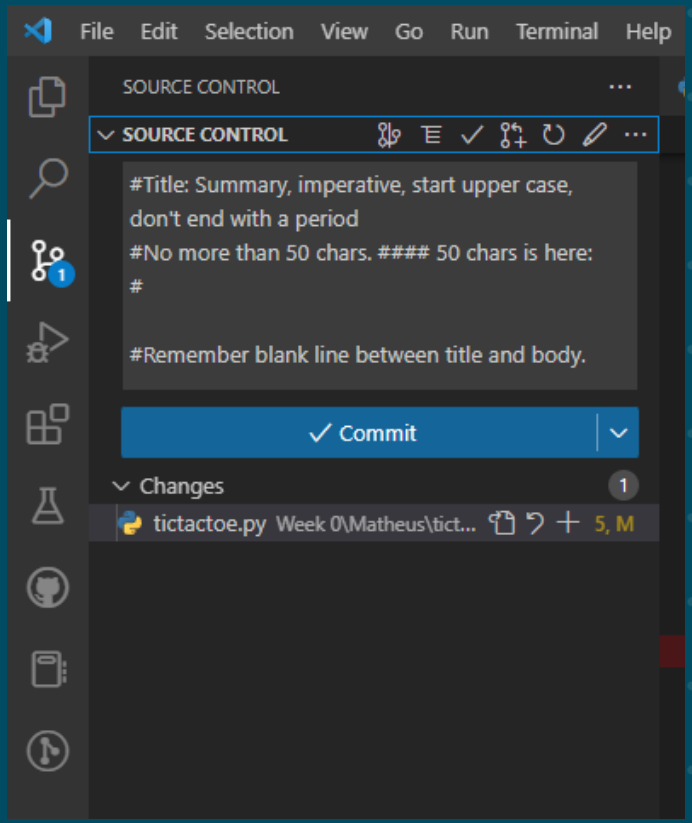


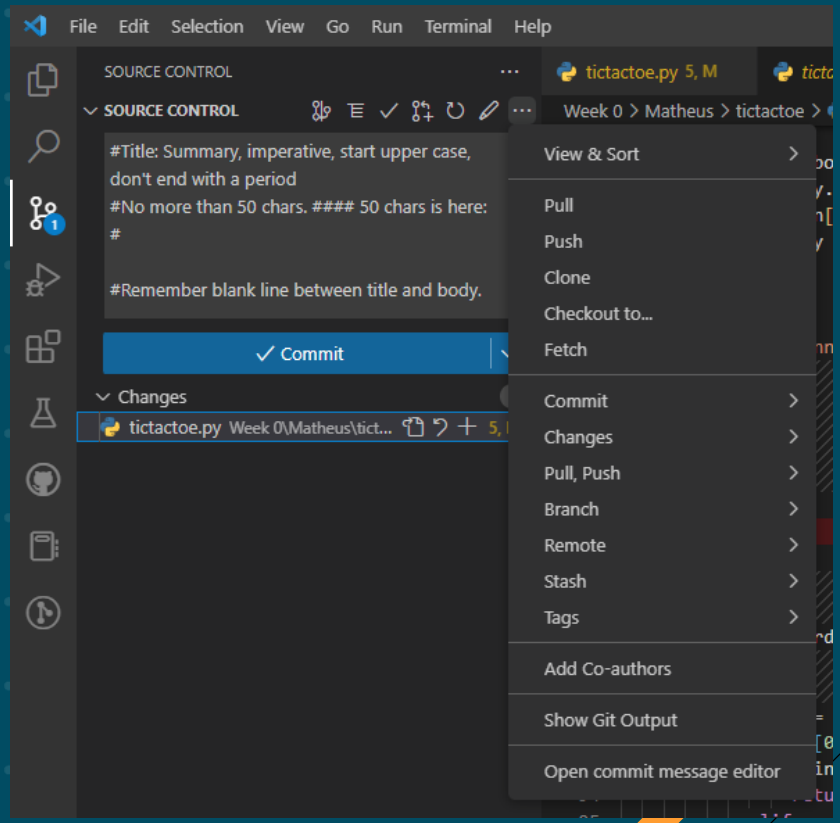

File Edit Selection View Go Run Terminal Help
tictactoe.py - CS50_Introduction_to_AI_with_python - Visual Studio Code
EXPLORER
CS50_INTRODUCTION_TO_AI_WITH_PYTHON
Week 0
João
Matheus
degrees
tictactoe
__pycache__
OpenSans-Regular.ttf
runner.py
tictactoe.py 5, M
Week 1
Week 2
Week 3
Week 4
Week 5
Week 6
.gitignore
LICENSE
README.md
OUTLINE
TIMELINE
tictactoe.py 5, M
Week 0 > Matheus > tictactoe > tictactoe.py > winner
"""
Returns the winner of the game, if there is one.
And this
is a
test
for the
"""
"Checks rows"
for row in board:
if all(x == row[0] for x in row): # The all() function returns True if all items in an iterable are true, otherwise it returns False
if row[0] == X:
xwin = True
return X
elif row[0] == O:
owin = True
return 2
for column in range(3):
for row in range(3):
x.append(board[row][column])
if all(spot == x[0] for spot in x):
if x[0] == X:
xwin = True
return X
elif x[0] == O:
owin = True
return O
"Checks diagonals"
diagonal1 = [board[0][0], board[1][1], board[2][2]]
diagonal2 = [board[0][2], board[1][1], board[2][0]]
main*
0 5
You, 1 second ago
Ln 107, Col 5
Spaces: 2
UTF-8
LF
Python
3.9.13 ('PostRock': conda)
Go Live

O que é Vscode?

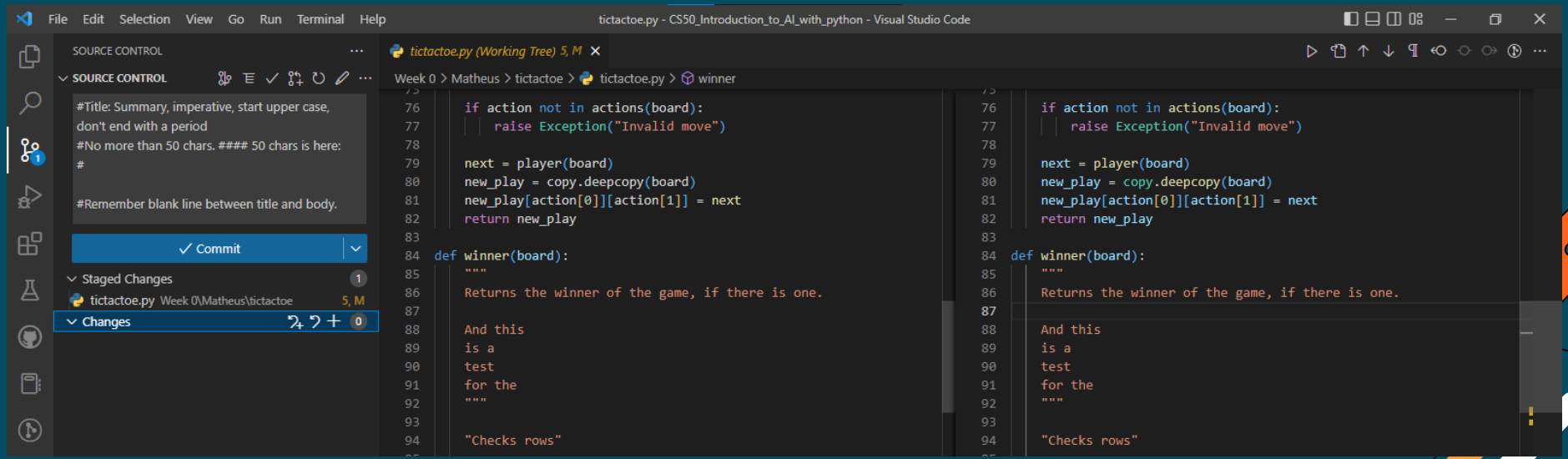


O que é Vscode?

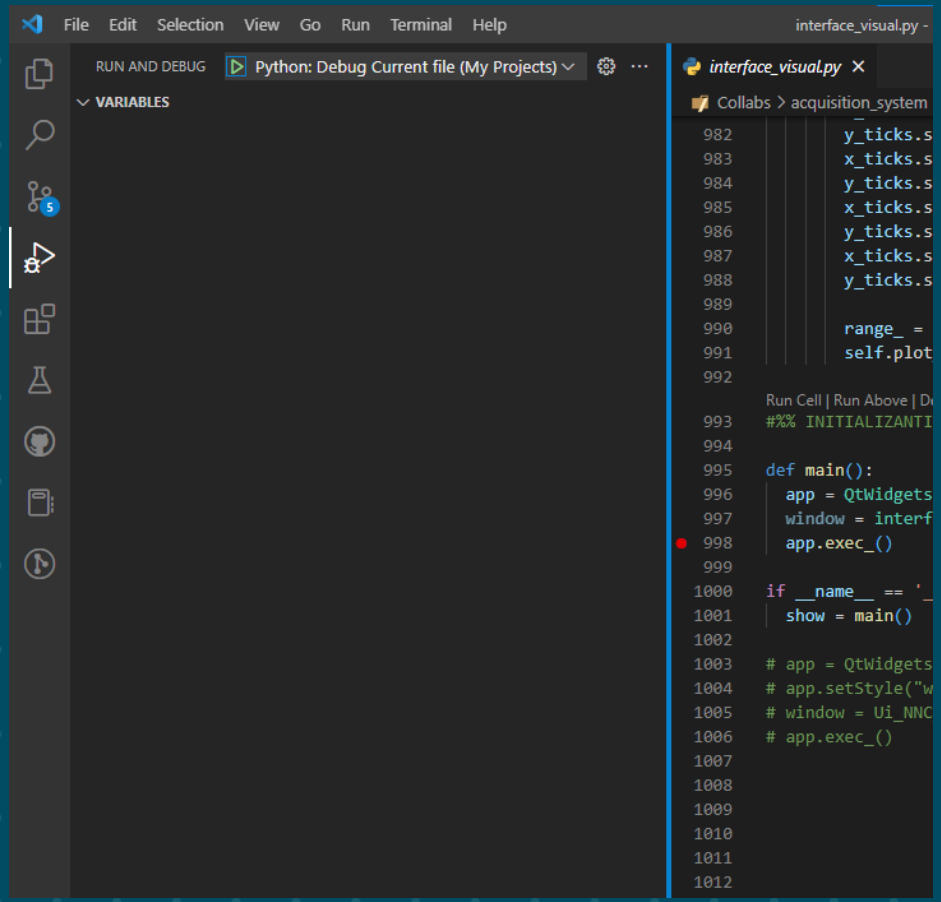




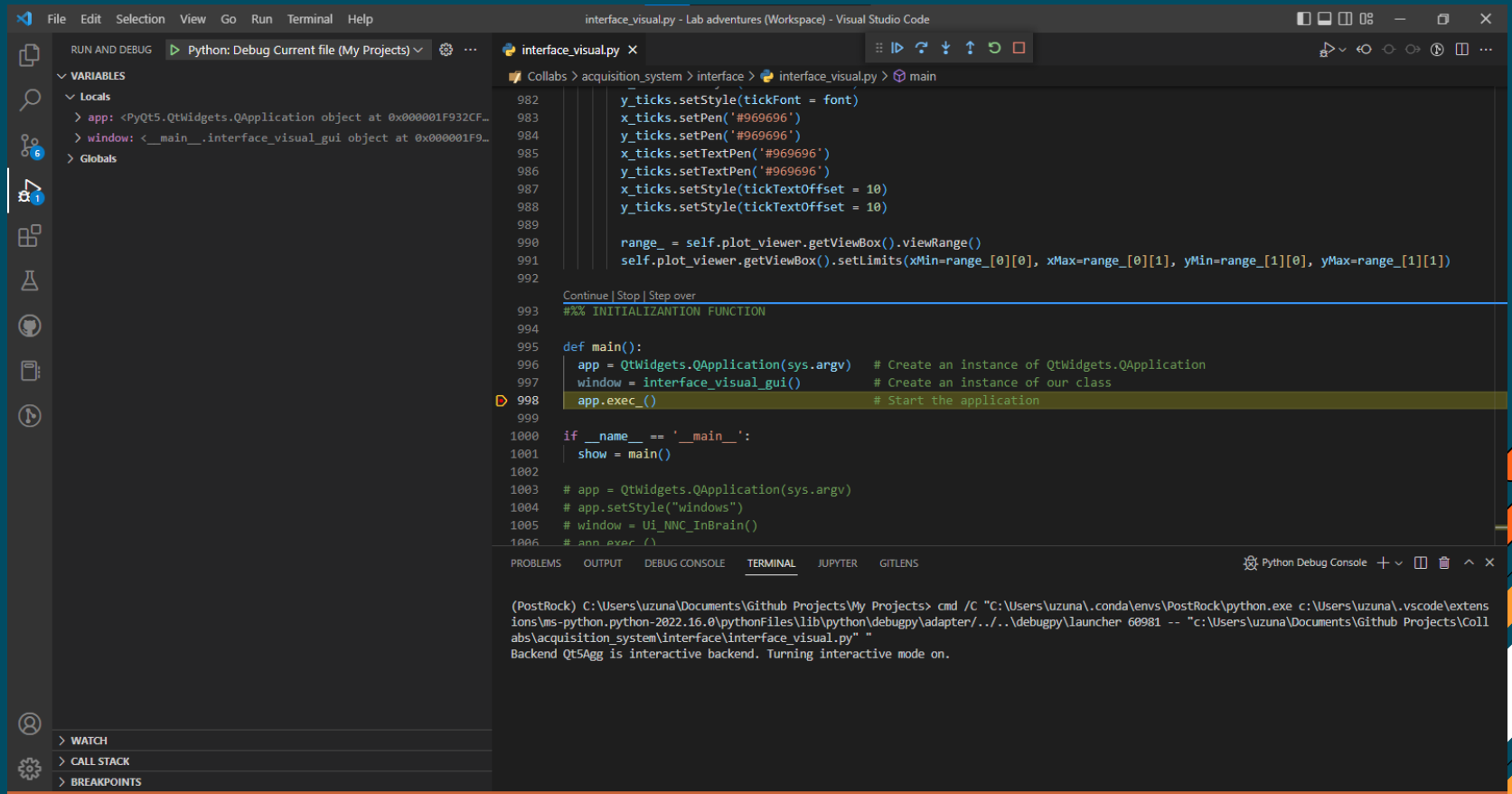
O que é Vscode?



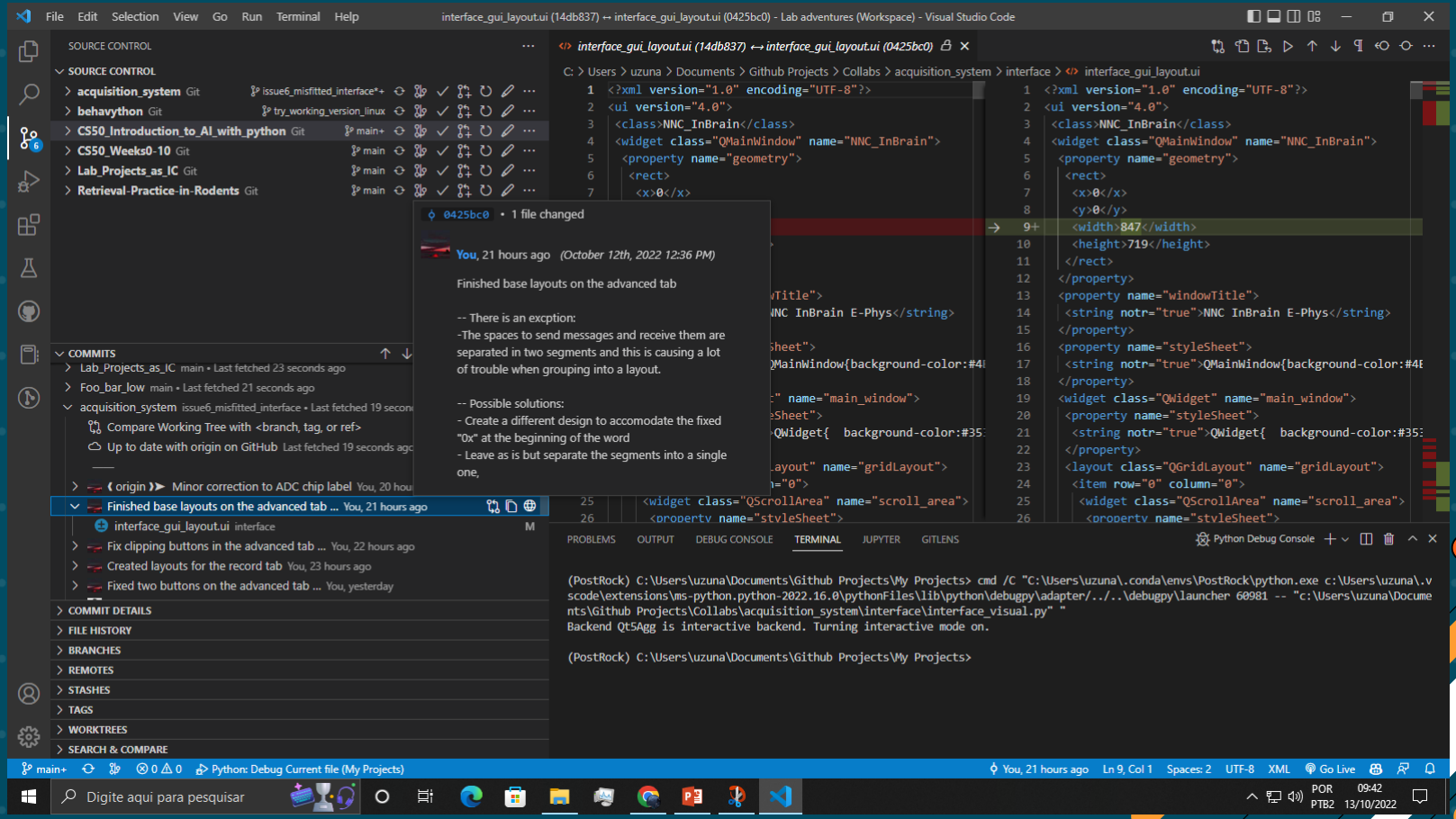
Deu pau, e agora?



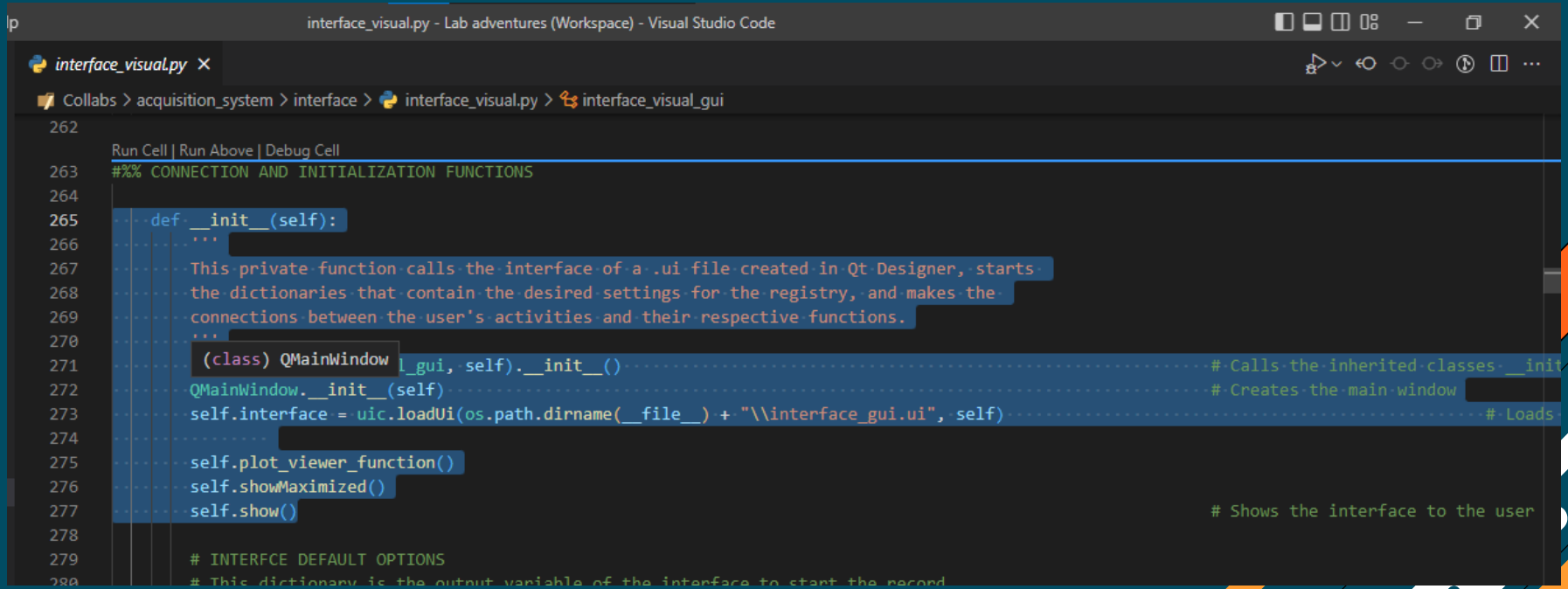
Deu pau, e agora?



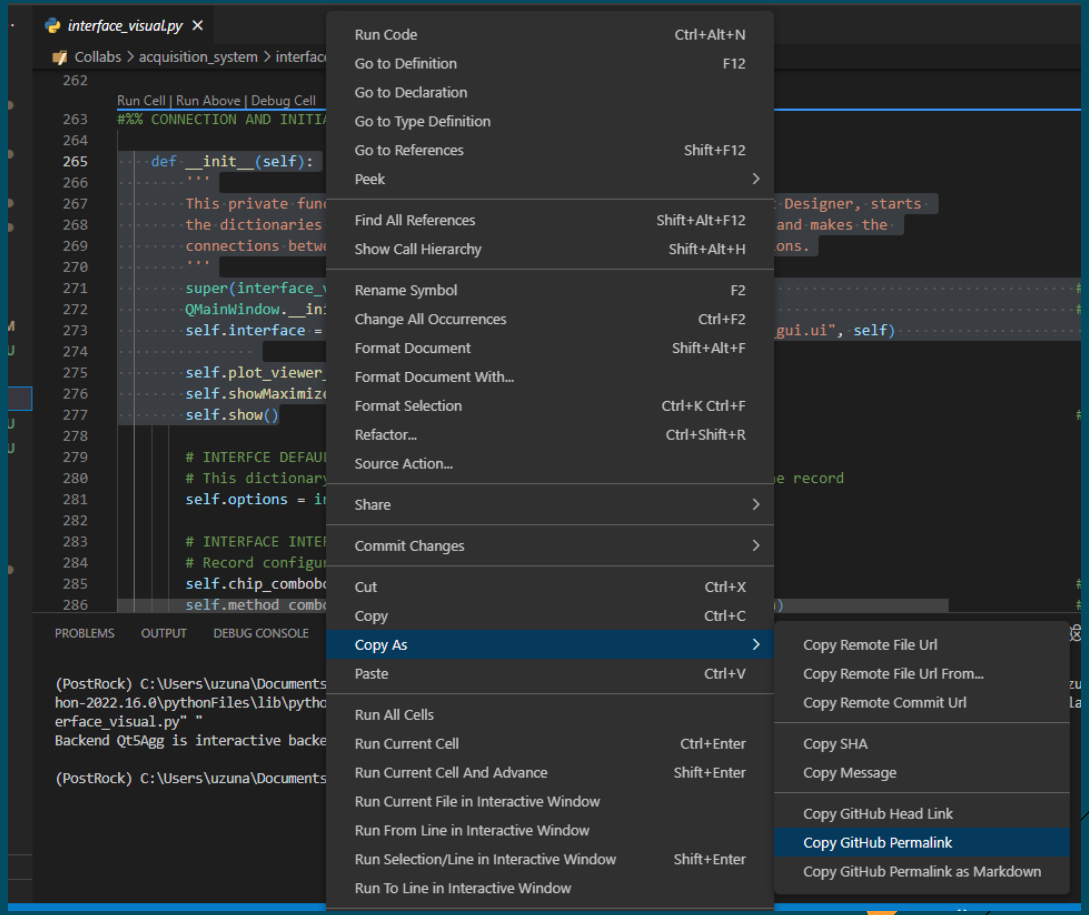
Deu pau, e agora?



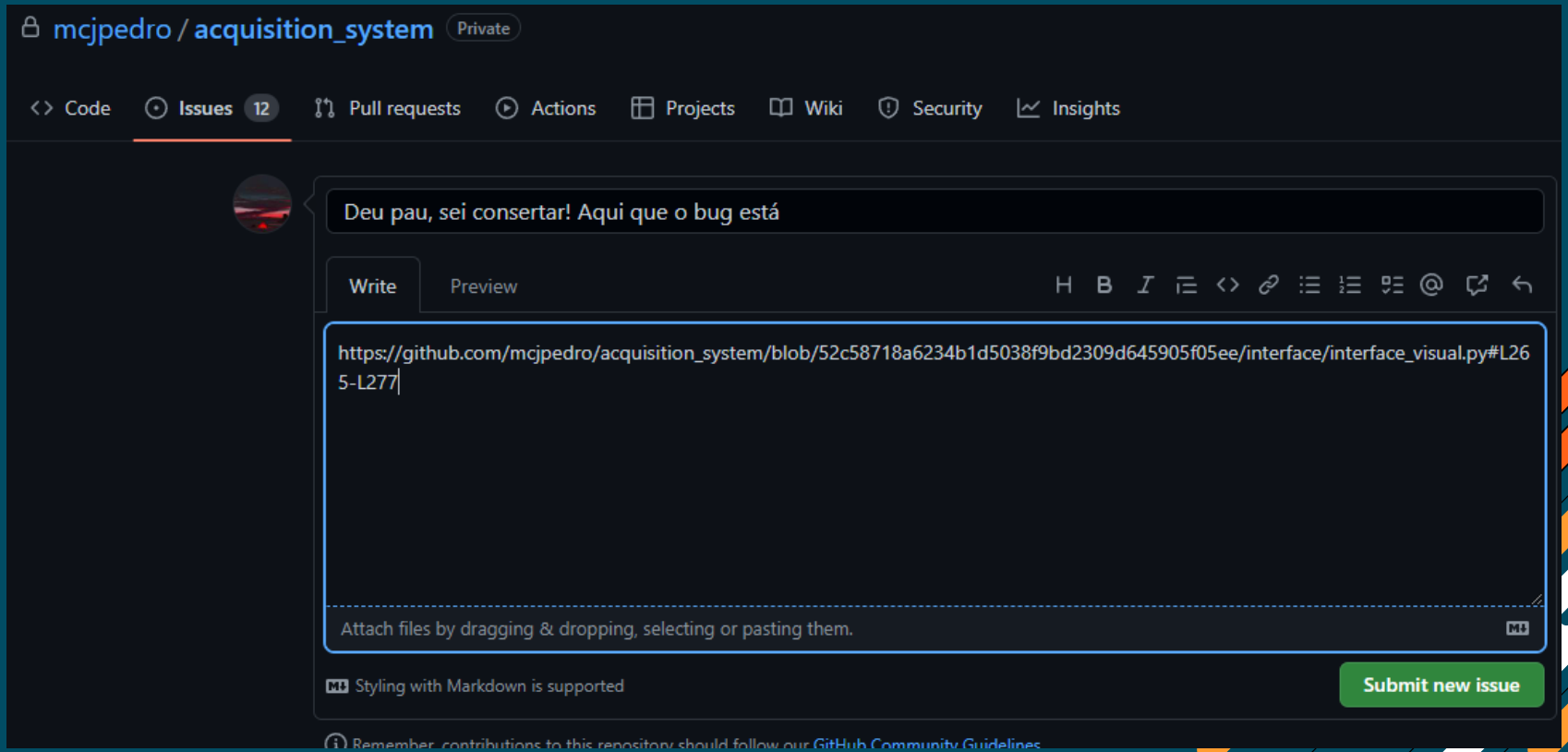
Deu pau, sei consertar. Como mostrar onde está?



Deu pau, sei consertar. Como mostrar onde está?

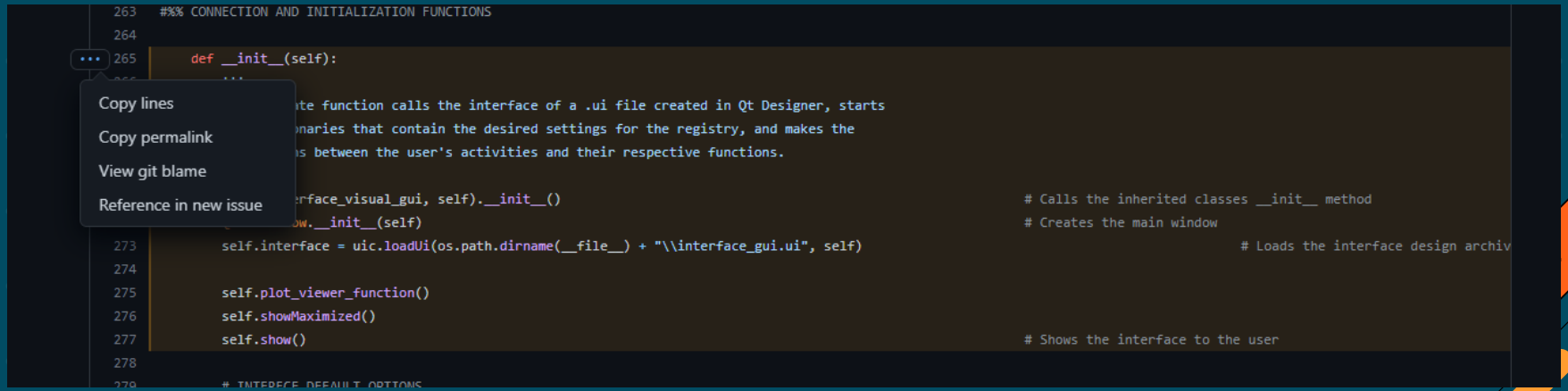


Deu pau, sei consertar. Como mostrar onde está?



Deu pau, sei consertar. Como mostrar onde está?

Deu pau, sei consertar. Como mostrar onde está?



Extensões são o poder