

UNIVERSIDADE DE SÃO PAULO

Escola de Engenharia de São Carlos

Departamento de Engenharia Elétrica e de Computação

SEL0337/SEL0630

PROJETOS EM SISTEMAS EMBARCADOS

Capítulo 6

**Programação de Interfaces de Visão
Computacional e Versionamento de Código em
Linux Embarcado**

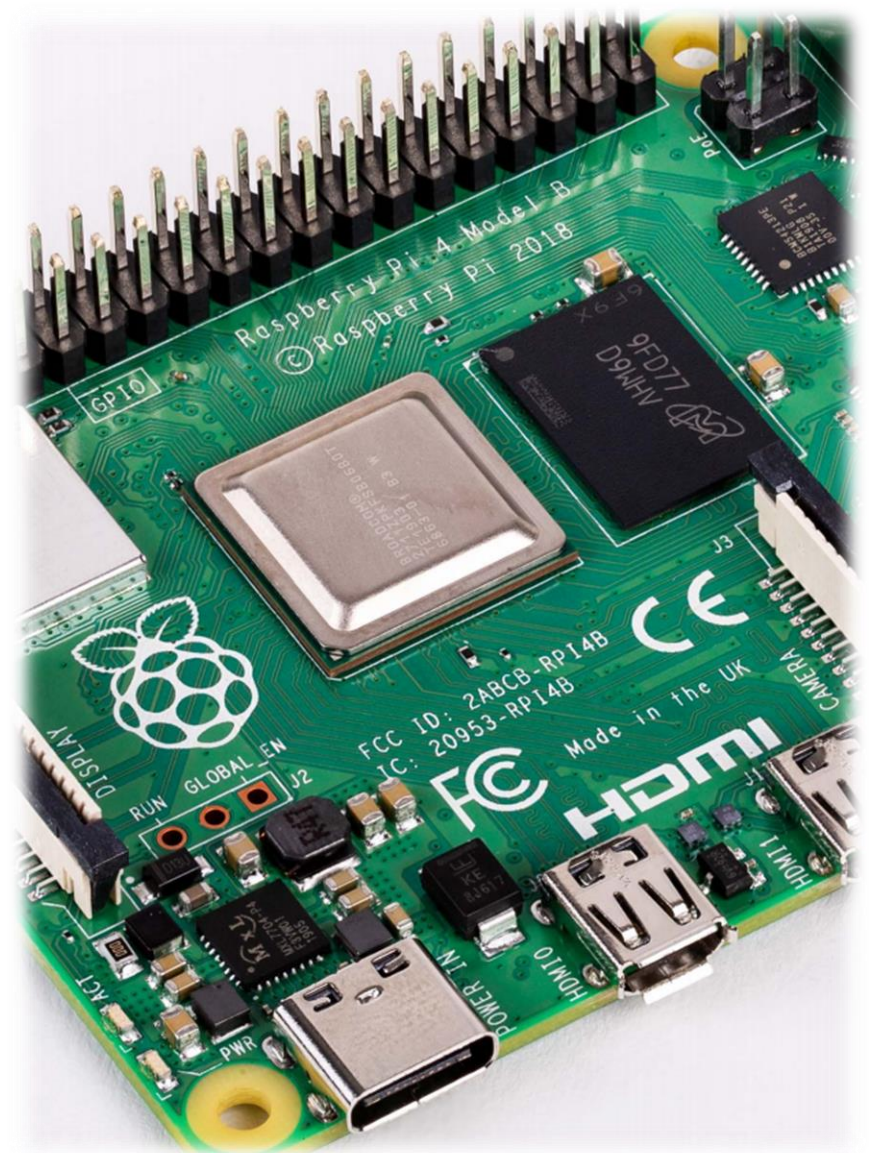
Prof. Pedro de Oliveira C. Junior

pedro.oliveiracjr@usp.br

OBJETIVOS



Desenvolver projetos em sistemas embarcados com controle de versão usando Git e GitHub, interface com a câmera para visão computacional e reconhecimento de padrões



Conceitos de Git e GitHub

Controle de versão V.C.S.

- **Version Control System**
- Técnica/conceito que auxilia no **gerenciamento** do **código-fonte** de um projeto;
- Permite registrar todas as modificações de código, podendo também **revertê-las**;
- Permite criar **versões** de um software em diferentes estágios, podendo alterar facilmente entre elas;
- Cada integrante da **equipe** pode trabalhar em uma versão diferente;
- Existem ferramentas para trabalhar o controle de versão, tais como: **Git e SVN**

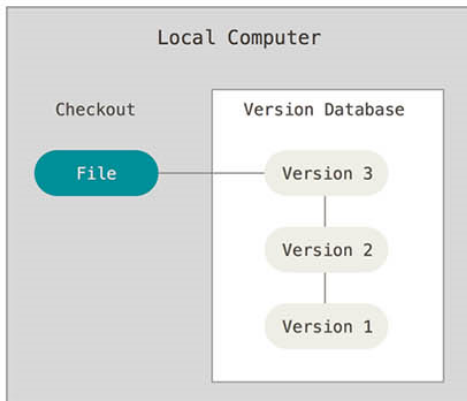
Modelos de V.C.S.

- ✓ **Local Data Model**: modelo mais simples – requer que todos os devs tenham acesso ao mesmo file system – **impossibilidade de colaboração!**
- ✓ **Client-Server Model (C.V.C.S)**: sistema centralizado, i.e., múltiplos devs podem acessar os arquivos em um servidor via internet – SVN (Subversion) – requer que estejam conectados - existe a possibilidade de perda de conexão e arquivos corrompidos!
- ✓ **Distributed Model (D.V.C.S)**: cada dev. trabalha no projeto a partir de seu repositório local. **Modificações são compartilhadas entre os repositórios de forma separada e quando necessário com baixa possibilidade de perda de histórico (usado pelo Git e projetos open-source).**

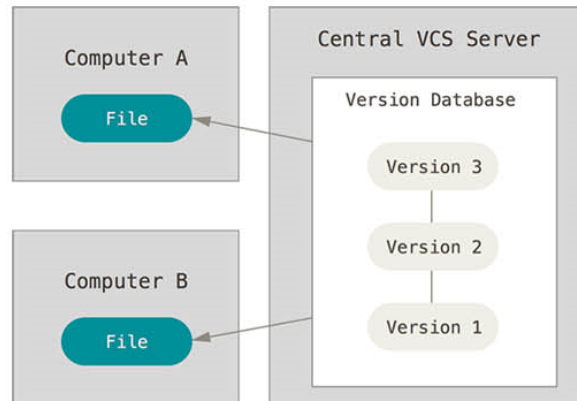
Modelos de V.C.S.

➤ Ilustração dos modelos:

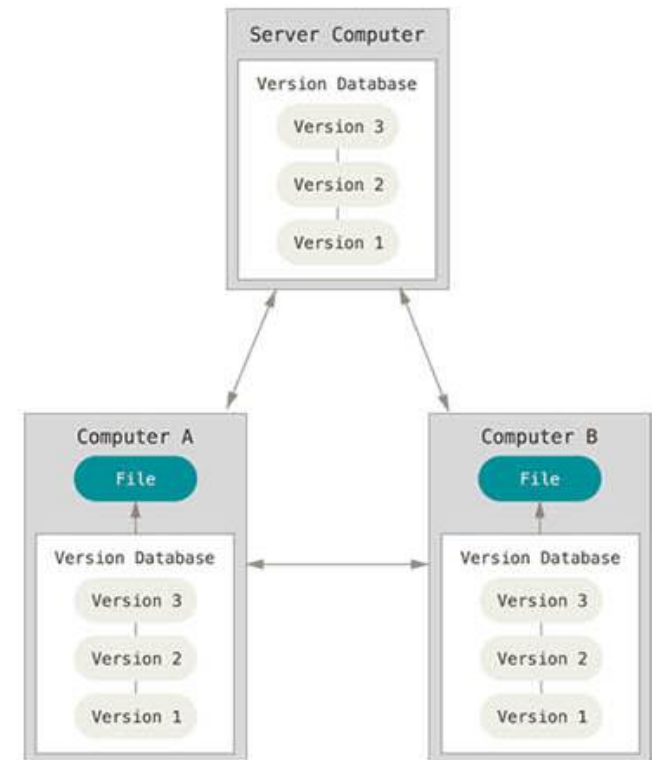
Local Model



Client-Server Model



Distributed Model



Fonte: Chacon S. Straub B. Pro Git. The Expert's Voice. 2nd Apress 2022. <https://git-scm.com/book/en/v2>

Soluções de V.C.S.



- ✓ Open source;
- ✓ Sistema de V.C.S. Distribuído (D.V.C.S)
- ✓ Mais usado no mundo!
- ✓ Multiplataforma
- ✓ Baseado em repositórios;
- ✓ Códigos em diversos locais;
- ✓ Adequado para diferentes qtd de equipes
- ✓ Facilidade

VERSION CONTROL SYSTEMS



- ✓ Open source;
- ✓ Sistema de V.C.S. Centralizado (C.V.C.S)
- ✓ Multiplataforma
- ✓ Interface mais simples e intuitiva (poucos comandos)
- ✓ Não é adequado para trabalhos em grandes equipes
- ✓ Facilidade

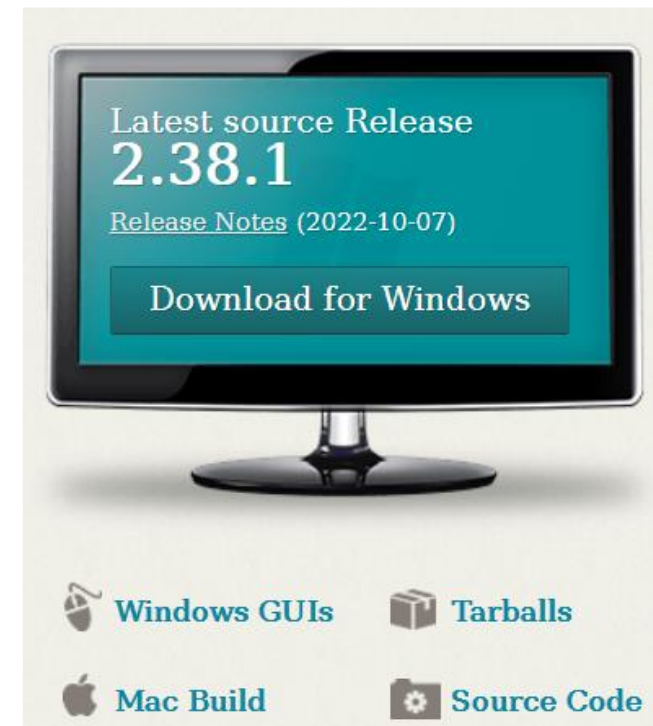
O que é Git?

- ✓ Sistema de controle de versão distribuído desenvolvido em 2005 por Linus Torvalds inicialmente como ferramenta de versionamento para Kernel Linux

➤ Página: <https://git-scm.com>



 About The advantages of Git compared to other source control systems.	 Documentation Command reference pages, Pro Git book content, videos and other material.
 Downloads GUI clients and binary releases for all major platforms.	 Community Get involved! Bug reporting, mailing list, chat, development and more.



O que é Git?

- O Git é baseado em **repositórios**
- Contêm todas as **versões** de um **código-fonte** e as cópias de cada desenvolvedor!
- Cada diretório do Git é um **repositório** com um **histórico** completo.
- Pode registrar o histórico de edições de qualquer tipo de arquivo (**livros digitais, documentos científicos etc.**)
- Não dependente de acesso a uma rede ou a um servidor central.
- É um **software livre** (licença GNU G.P.L)



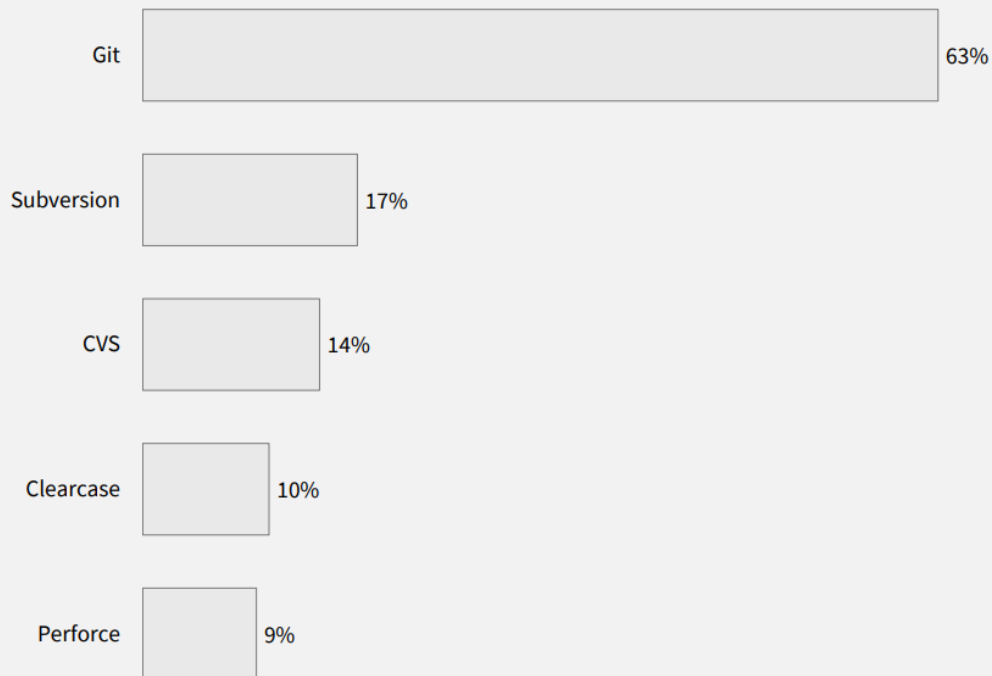
Marcas/empresas que utilizam Git



Uso de Git em sistemas embarcados

- Segundo o Relatório “The Current State of Embedded Development” - 2023

Git is the most widely used version control software



Fonte: <https://www.embedded.com/wp-content/uploads/2023/05/Embedded-Market-Study-For-Webinar-Recording-April-2023.pdf>

Git features

- ✓ Sempre que um projeto é modificado **localmente** é possível enviar, i.e., fazer o **push** dessas mudanças para um **repositório (remoto)**;
 - ✓ Da mesma forma, outros devs podem fazer o **pull** das modificações enviadas ao repo. para continuar o trabalho a partir dos **upgrades** realizados
-
- No repositório local, você possui cópia do “repositório completo” do projeto a partir do **git clone**.
 - Você realiza **commit** nas modificações dos arquivos para o repo. remoto.
 - O trabalho pode ser feito **offline** e a sincronização pode ser feita quando tiver acesso **online** ou quando necessário.

Ref.: <http://ditech.com.br/git-beneficios-ferramenta-projetos/>

Git features

- ✓ Possui alta performance: operações com milhares de arquivos são rápidas, **executadas em segundos!**
- ✓ É possível verificar o **status** das alterações e **ignorar** certos arquivos, saber quando um **bug** foi introduzido.

➤ Branches: como o Git separa as versões do projeto

- Quando um projeto é criado ele inicia na **branch main** (anteriormente **master***).
- Você pode trabalhar em uma nova **feature** de teste **sem interferir no código principal a partir de novos branches**
- Permitem diferentes **devs** trabalhar em diferentes **features** sem interferir no trabalho de cada um
- Após a conclusão, é possível fazer um **merge** das mudanças para o branch principal (**main**).
- É possível **salvar o status de um branch do seu projeto sem commitar as modificações e retornar a ele futuramente.**

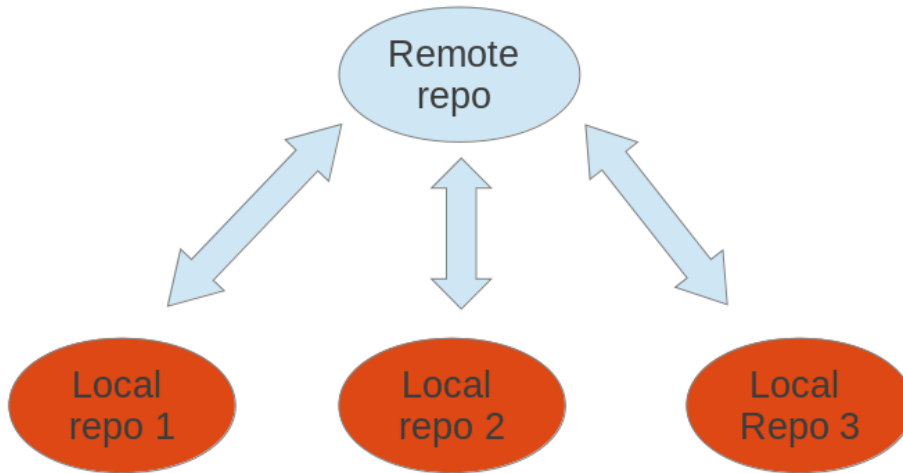
Ref.: <http://ditech.com.br/git-beneficios-ferramenta-projetos/>

Repositórios

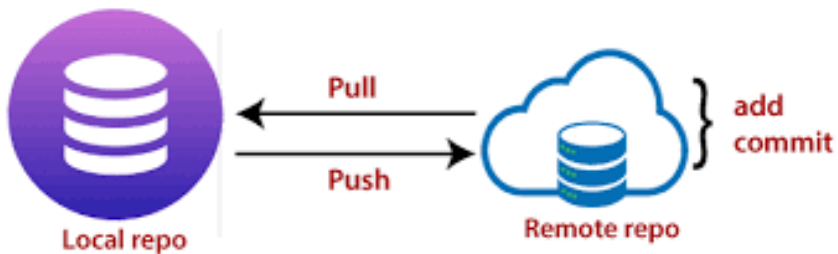
- ✓ Onde o **código-fonte** ou arquivos de um projeto é armazenado;
- ✓ Na maioria das vezes, cada projeto possui um repositório;
- ✓ Quando criamos um repositório estamos iniciando um projeto;
- ✓ O repositório pode ir para servidores que são especializados em gerência-los, como: **GitHub e Bitbucket**.
 - ✓ Portanto: arquivos de um projeto são armazenados em um repo (servidor) e histórico de modificações (V.C.S.) fica salvo nele!



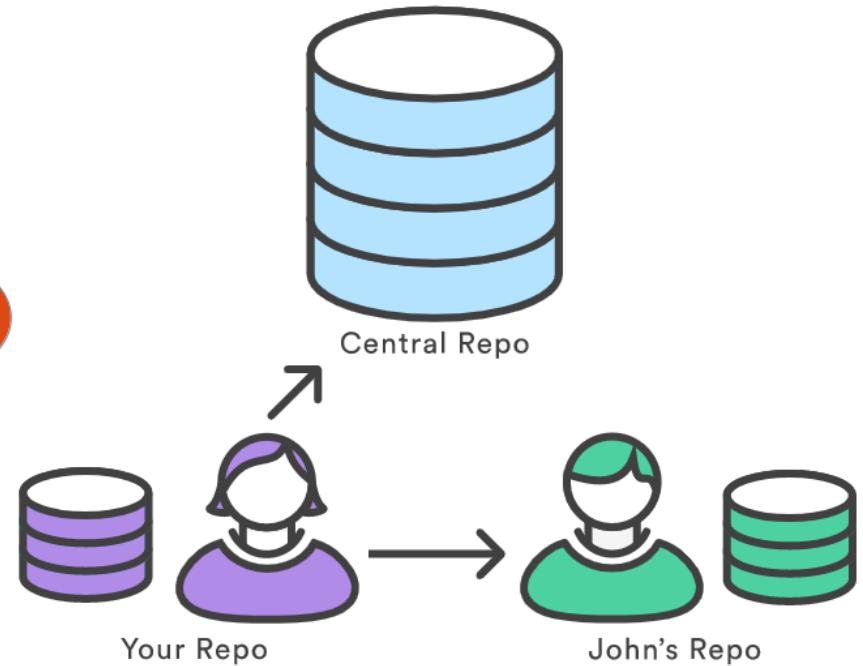
Repositórios



https://miro.medium.com/max/768/0*UrGOzV44WJtvJXNT.png



<https://static.javatpoint.com/tutorial/git/images/git-push.png>



<https://www.atlassian.com/git/tutorials/syncing>

O que é GitHub?

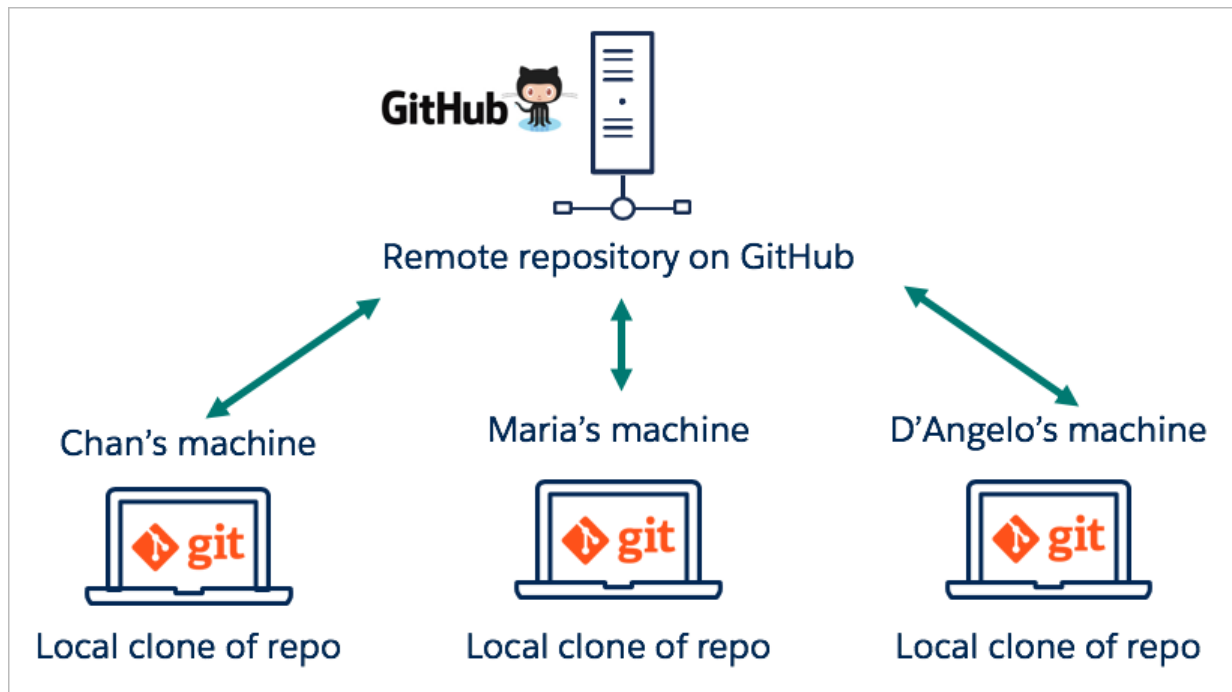
- ✓ É uma plataforma de hospedagem de códigos-fonte com V.C.S. e de gerenciamento de repositórios usando Git.
- ✓ Foi desenvolvido em 2008 pela empresa GitHub, Inc. e adquirido pela empresa Microsoft em 2018. Concorrentes: GitLab; Bitbucket

- GitHub permite que devs contribuam para projetos open-source ou privados de qualquer lugar do mundo!
- Amplamente utilizado para divulgação e comunicação de códigos em repositórios remotos!
- Página: <https://github.com> – Blog: <https://github.blog>



O que é GitHub?

- ✓ Possui mais de **36 milhões de usuários** ativos mundialmente e mais de **100 milhões de projetos** (Dentre eles: **GNU/Linux**).
- ✓ Usado por grandes empresas: Google, Nubank, WordPress etc.
- ✓ **O GitHub é gratuito tanto para projetos públicos como privados**

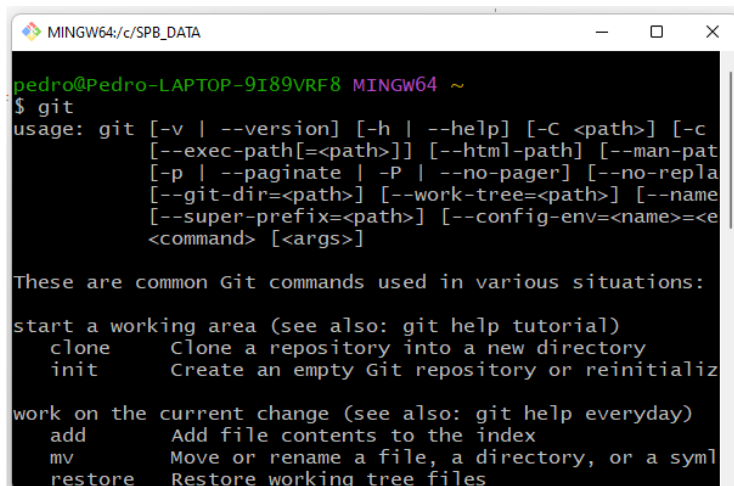


<https://sfdctechie.files.wordpress.com/2019/12/github.png>

Getting started with Git/GitHub

✓ Instalação:

- **Git** - <https://git-scm.com/downloads> (para Windows, MacOS, Linux)
- **GitHub** - <https://desktop.github.com> (GitHub Desktop para Windows ou MacOS)



```

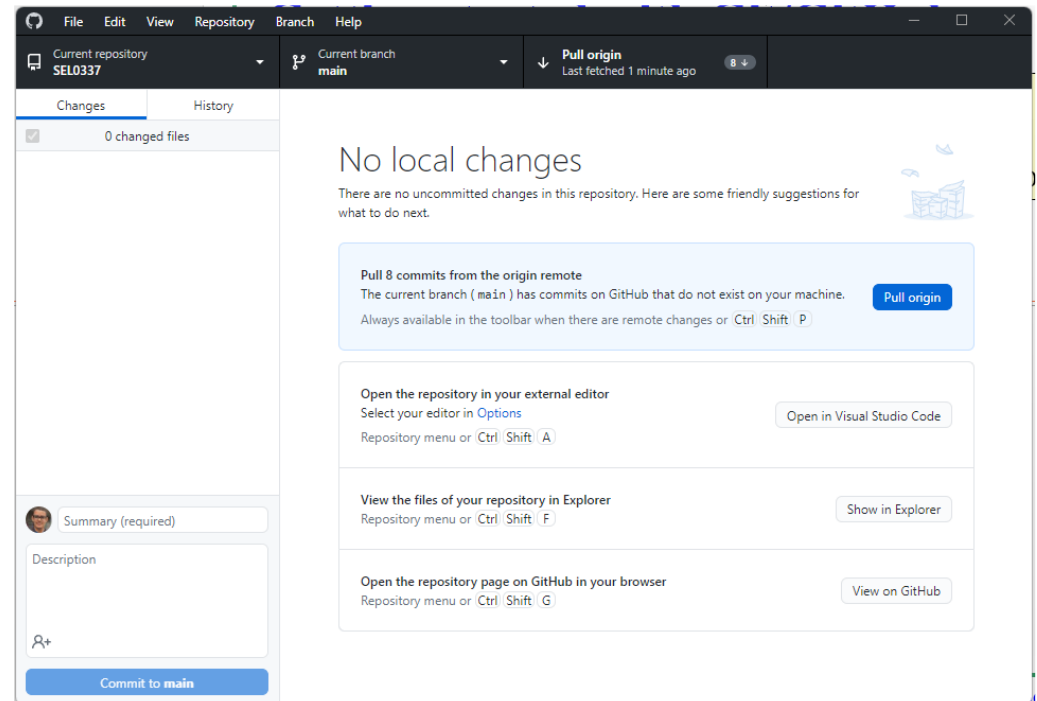
MINGW64; c:/SPB_DATA
pedro@Pedro-LAPTOP-9I89VRF8 MINGW64 ~
$ git
usage: git [-v | --version] [-h | --help] [-C <path>] [-c
[--exec-path=<path>]] [--html-path] [--man-path]
[-p | --paginate | -P | --no-pager] [--no-repla
[--git-dir=<path>] [--work-tree=<path>] [--name
[--super-prefix=<path>] [--config-env=<name>=<e
<command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
clone      Clone a repository into a new directory
init       Create an empty Git repository or reinitializ

work on the current change (see also: git help everyday)
add        Add file contents to the index
mv         Move or rename a file, a directory, or a syml
restore    Restore working tree files
  
```

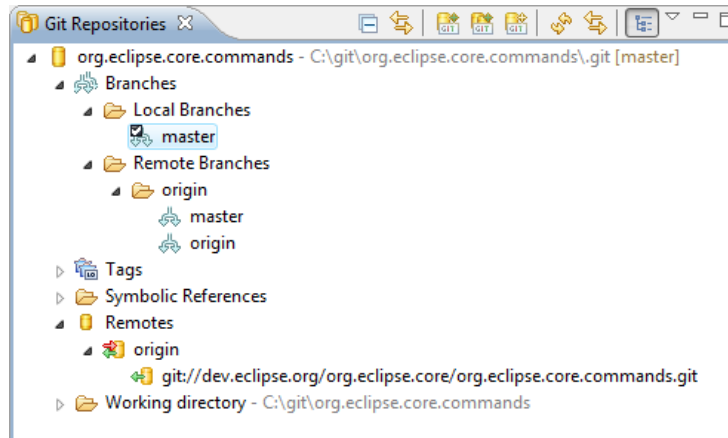
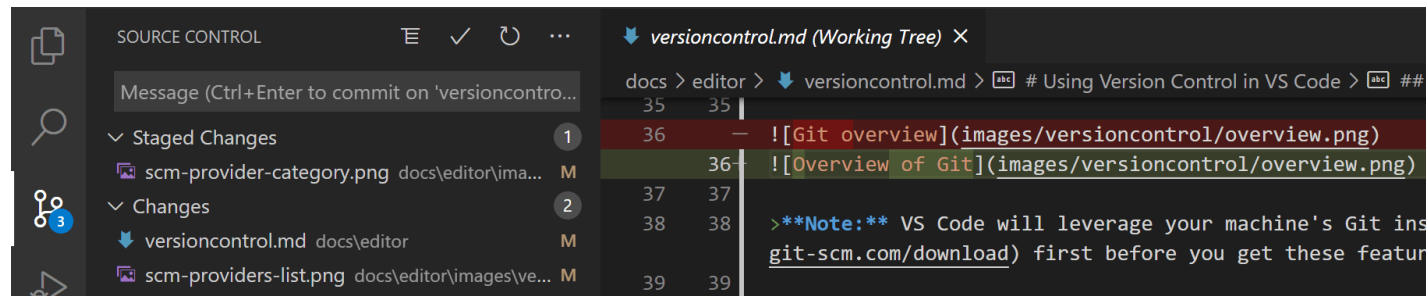
Terminal Git Bash e GitHub Desktop no Windows



Getting started with Git/GitHub

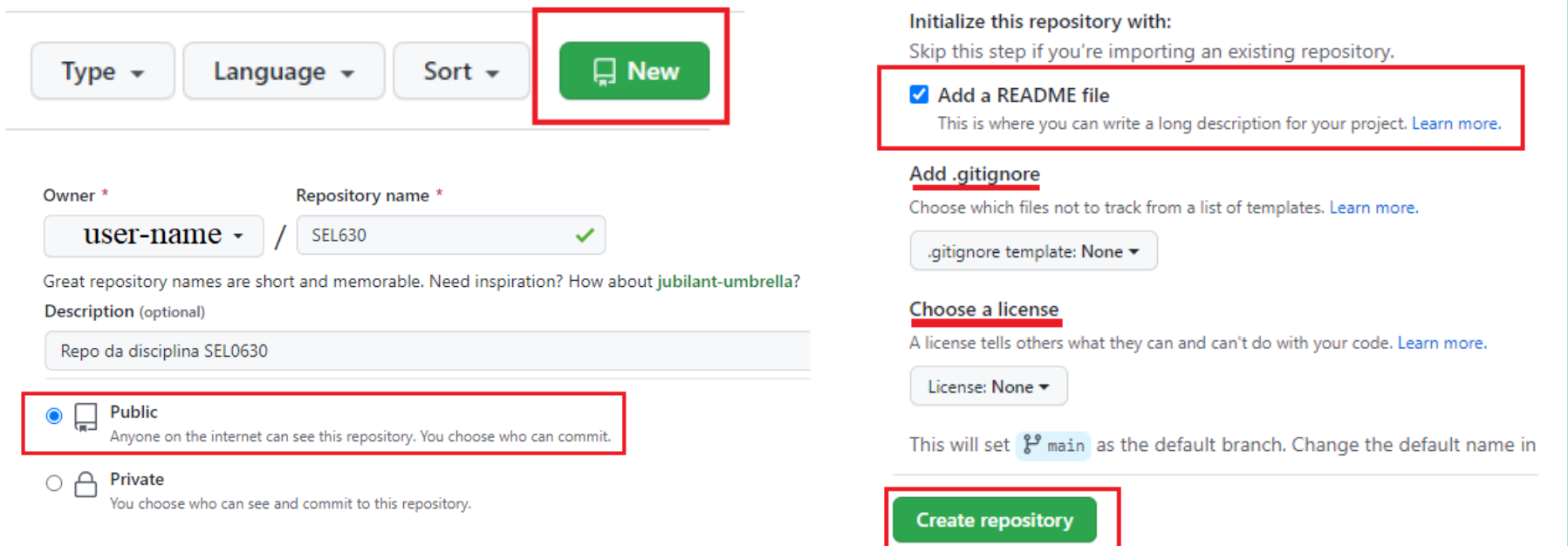
✓ Integração com ambientes de desenvolvimento:

- VSCode - <https://code.visualstudio.com/docs/sourcecontrol/overview>
- Eclipse - <https://projects.eclipse.org/projects/technology.egit>



Explorando o GitHub

- ✓ Acessar sua conta no GitHub e criar um repo público para projetos na disciplina e explorar suas funcionalidades
 - ✓ **README file** (descrição do projeto). Adicione ao projeto!
 - ✓ **“.gitignore”** – ignora arquivos sensíveis ou desnecessários;
 - ✓ **License:** em caso de um projeto open-source que será compartilhado sob uma licença.



Type ▾ Language ▾ Sort ▾ **New**

Owner * / Repository name * ✓

Great repository names are short and memorable. Need inspiration? How about [jubilant-umbrella?](#)

Description (optional)

☒ **Public**
Anyone on the internet can see this repository. You choose who can commit.

☐ **Private**
You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.

☒ **Add a README file**
This is where you can write a long description for your project. [Learn more.](#)

Add .gitignore
Choose which files not to track from a list of templates. [Learn more.](#)
.gitignore template: None ▾

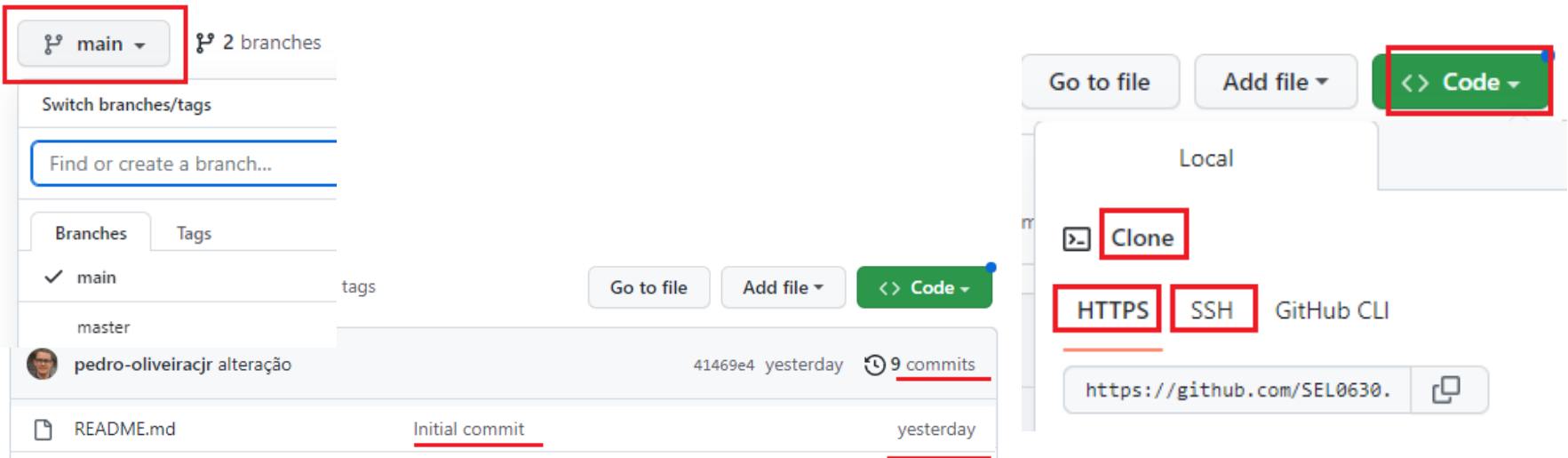
Choose a license
A license tells others what they can and can't do with your code. [Learn more.](#)
License: None ▾

This will set `main` as the default branch. Change the default name in

Create repository

Explorando o GitHub

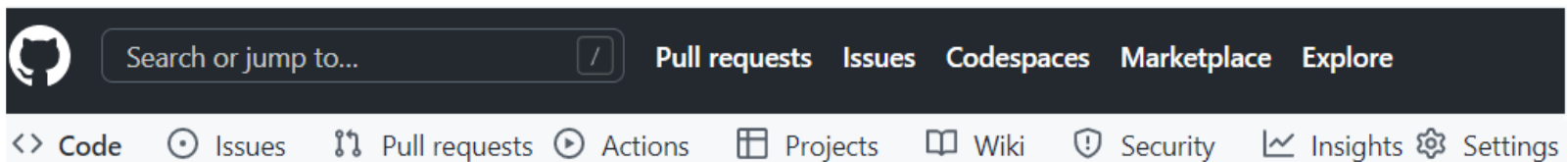
- ✓ Acessar sua conta no GitHub e criar um repo público para projetos na disciplina e explorar suas funcionalidades
 - ✓ **Branches do repo:** main (padrão); master
 - ✓ **Informações:** arquivo commitado, qtde de *commits*
 - ✓ **Code:** informações para clonar o repo localmente via https ou SSH
 - ✓ **Go to file:** acessar um arquivo dentro do repo
 - ✓ **Add file:** criar um novo arquivo pelo GitHub



The screenshot displays the GitHub repository page for a user named 'pedro-oliveiracrj'. The repository is named 'pedro-oliveiracrj alteração' and has a commit hash of '41469e4' from 'yesterday' with '9 commits'. The 'main' branch is selected, and there are '2 branches' in total. The interface includes a 'Switch branches/tags' section with a search bar 'Find or create a branch...'. Below this, there are tabs for 'Branches' and 'Tags', with 'main' selected. A 'Go to file' button is visible. On the right side, there are buttons for 'Go to file', 'Add file', and a green 'Code' button. Below the 'Code' button, there are options to 'Clone' the repository using 'HTTPS' or 'SSH', and a 'GitHub CLI' option. A text box shows the repository URL 'https://github.com/SEL0630.' with a copy icon.

Explorando outros recursos do GitHub

- ✓ **Issue**: criar tarefas ou possíveis *bugs* do projeto;
- ✓ **Pull Request**: envio do código para resolver as *issues* ou add novas funcionalidades
- ✓ **Actions**: cria as automatizações de *deploy* (ciclo de vida de software);
- ✓ **Projects**: criar um projeto e utilizar um quadro de tarefas ou **Kanban** e pode ajudar a organizar sua equipe,
- ✓ **Wiki**: para criar documentação mais extensa para o projeto
- ✓ **Insights**: informações detalhadas de equipe, *commits*, *forks*
- ✓ **Settings**: alterar nome do repo, remover/add features ou colaboradores(as), remover repo
- ✓ **Markdown**: recurso para adicionar estilo personalizado a textos na web
- ✓ **GitHub Pages**: forma **gratuita** de criar um portfólio do projeto, sem necessidade de domínio ou servidor.
- ✓ **Criando releases**: disponibilizar no GitHub diferentes versões de um software.



Cadastrando Token no GitHub p/ uso na Raspberry Pi

- ✓ Gerar uma token na sua conta do GitHub para usar na assinatura de *commits*
 - ✓ **Gerando Token:** acessar a aba “**Settings**” na sua conta e a guia “**Developer Settings**” – acesse a aba **Personal access tokens** – **generate a new token** – **Classic**. Defina um nome. Em **Expiration** escolha o prazo mais curto; em **Select scope** habilite todas as opções.
 - ✓ **OBS.:** Ao gerar o token, copie a chave e salve em um local seguro e de fácil acesso, pois será usada nos *commits* no Git (é importante copiar para outro local em razão do GitHub só permitir a visualização da chave uma única vez, i.e., após você deixar a página do token e, ao retornar, não conseguirá mais ter acesso a chave. Portanto, para evitar a necessidade de criar outra chave ou várias, mantenha a cópia em doc separado assim que gerar o primeiro token)

Explorando tutoriais do GitHub

➤ Acessar: <https://docs.github.com/en>



GitHub Docs

Get started

Account and profile

Authentication

Repositories

Enterprise administrators

Billing and payments

Site policy

Organizations

Code security

Pull requests



Get started

Get started

Account and profile

Authentication

Billing and payments

Site policy

🛡️ Security

Code security

Supply chain security

Security advisories

Dependabot

Code scanning

Secret scanning



Collaborative coding

GitHub Codespaces

Repositories

Pull requests

GitHub Discussions

GitHub Copilot

📱 Client apps

GitHub CLI

GitHub Desktop



CI/CD and DevOps

GitHub Actions

GitHub Packages

GitHub Pages



Project management

GitHub Issues

Search on GitHub

Explorando tutoriais do GitHub

➤ **Acessar:** <https://docs.github.com/en/get-started/quickstart>

Get started

Learn how to start building, shipping, a account, and connect with the world's

Quickstart

Guides

GitHub's products

An overview of GitHub's products and pricing plans.

Getting started with your GitHub account

With a personal account on GitHub, you can import or create rep others, and connect with the GitHub community.

Getting started with GitHub Team

With GitHub Team groups of people can collaborate across many in an organization account.

Getting started with GitHub Enterprise Cloud

Get started with setting up and managing your GitHub Enterprise enterprise account.

Getting started with GitHub Enterprise Server

Get started with setting up and managing GitHub.com.

Quickstart

Get started using GitHub to manage Git repositories and collaborate with others.

[Hello World](#)

[Set up Git](#)

[Create a repo](#)

[Fork a repo](#)

[GitHub flow](#)

[Contributing to projects](#)

[Be social](#)

[Communicating on GitHub](#)

[GitHub glossary](#)

[Git cheatsheet](#)

[Git and GitHub learning resources](#)

Explorando recursos do Git

✓ Instalar o Git na Rasp. e explorar os recursos a seguir

```
# Formato para Debian/Ubuntu:
apt-get install git && git

#verificar lista de comandos
git help --all

# configurar dados de usuário para armazenamento de modificações
git config --global user.name "username" # usuário do GitHub entre aspas
git config --global user.email "seu_email" # e-mail do GitHub entre aspas

#verificar configurações armazenadas
git config --list --show-origin
git config -l

# Outras configurações:
git config --global color.ui auto # configuração de ações no modo colorido
git config --global core.editor gvim # define editor de texto padrão
git status # retorna o status do diretório de trabalho
```


Explorando recursos do Git

✓ **LAB**—Verifique o resultado executando os seguintes comandos:

```
git clone https://github.com/git/git && cd git  
  
git log --reverse  
  
cd  
  
git clone https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git  
  
cd linux  
  
git log -reverse
```

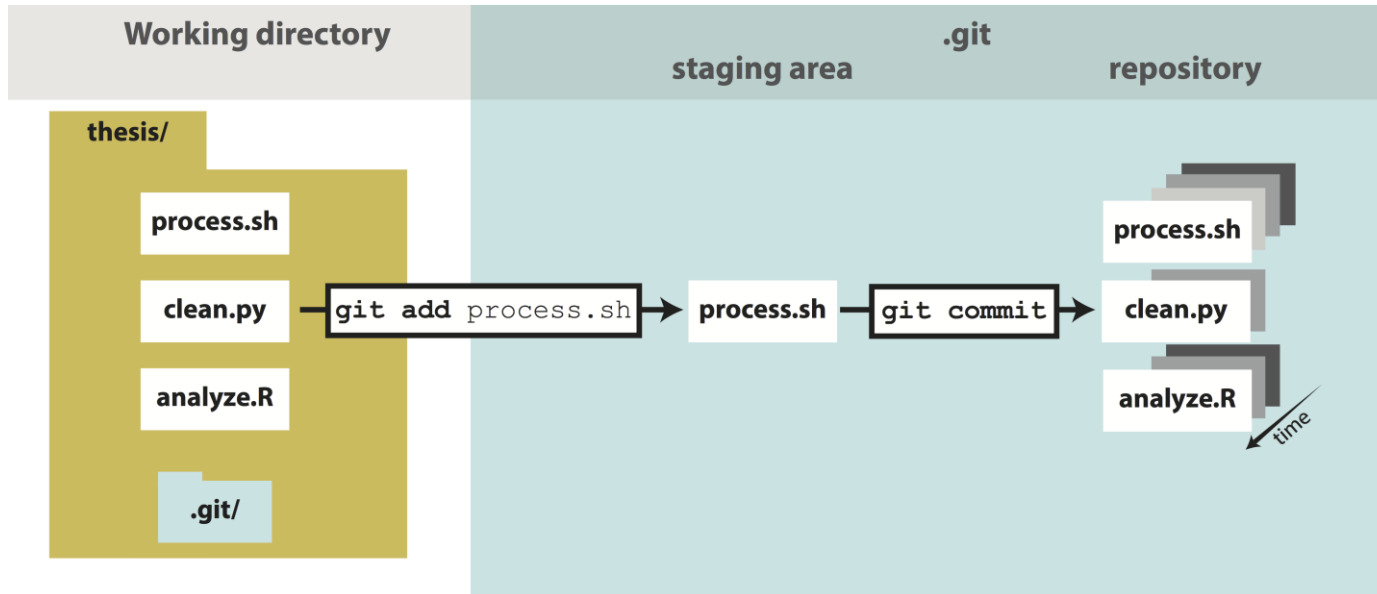
Explorando recursos do Git

- ✓ Iniciando um novo projeto, criando repo novo e preparando ambiente de trabalho na Raspberry Pi
- ✓ A criação de um novo **repo local** (**git init**) ou clonagem de um **remoto** (**git clone**) é feita somente no início de um novo projeto.
- ✓ Esses dois comandos (**git init** e **git clone**) são usados poucas vezes em razão de projetos serem mantidos em um único repo e em raras exceções é separado em mais de um repo. **Geralmente isso não é feito!**

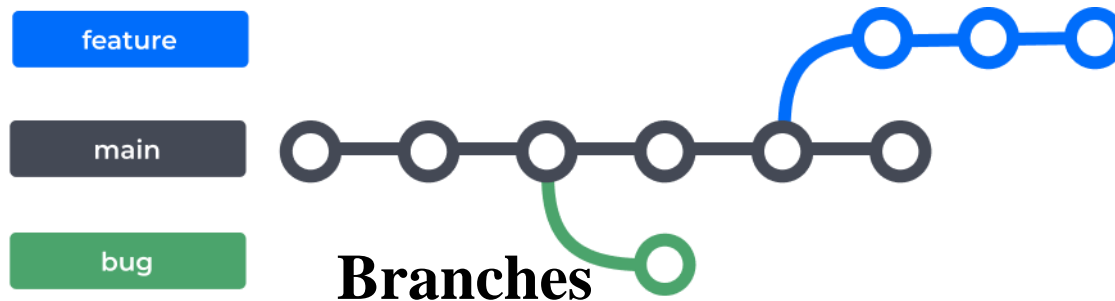
```
# git init = cria novo repo local
mkdir projetos_git && cd projetos_git # criar um diretório local para os trabalhos
git status # verifica se existe algum repo
git init
git status # agora sinaliza o repositório

# git clone = clona um repo remoto para a workspace local
git clone https://github.com/caminhoxxx/SEL0337 # aqui utilizar a origem daquele repo #público
criado anteriormente em sua conta no GitHub
ls
cd SEL0337 # acessando o repo remoto clonado e que agora está disponível localmente
git status
```

Estágios de um projeto com Git e GitHub

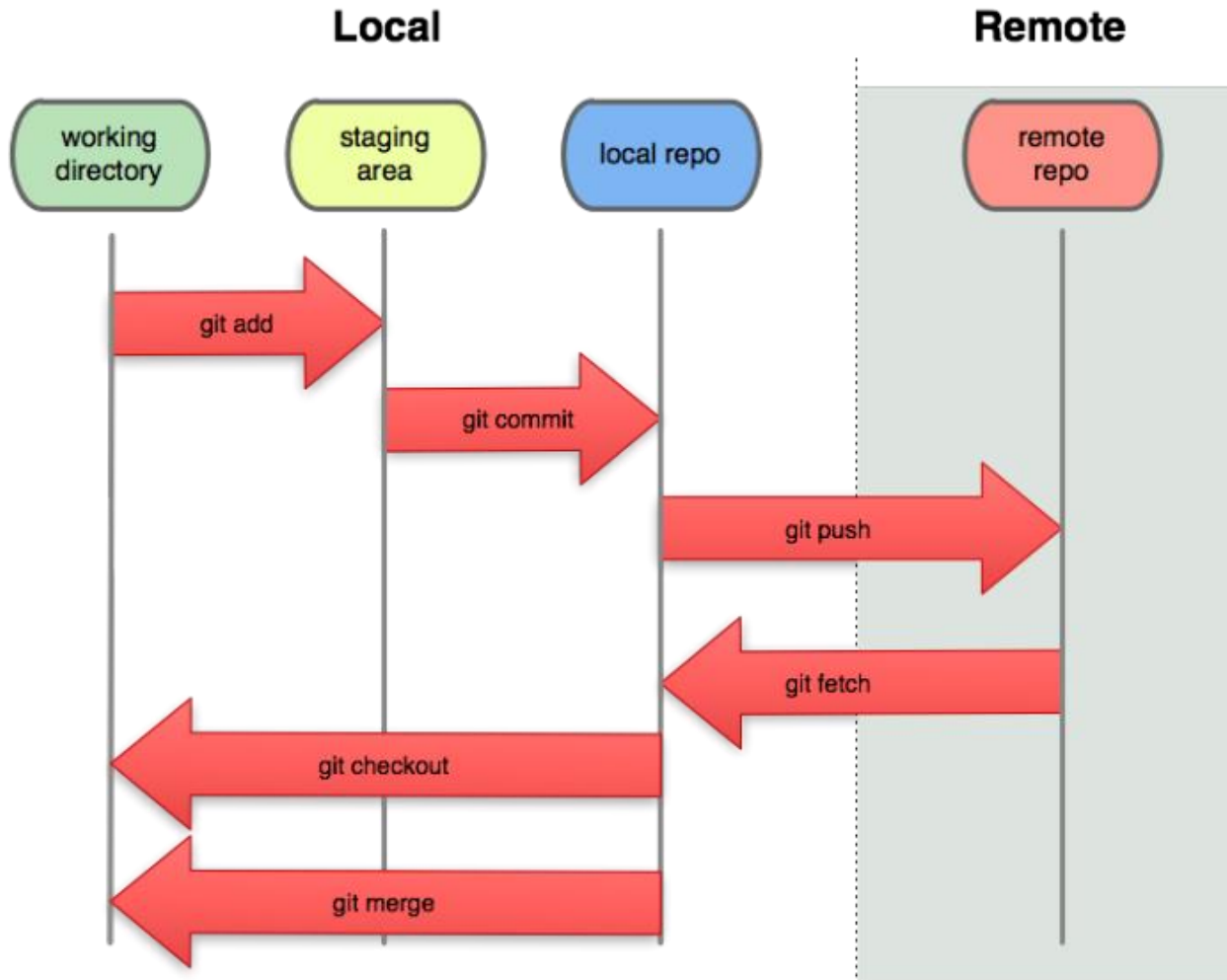


Fonte: <https://doi.org/10.1371/journal.pcbi.1004668>



https://miro.medium.com/max/800/1*RTgnIs0GY8r0rSPsAzf8NQ.png

Estágios de um projeto com Git e GitHub



Fonte: https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcTtwTqu1h5G7cu7S0-bhTeCivfhJQGn17S-7BU_XLMDUKLFPGxh7p88u3-kgKEnp6-SE3g&usqp=CAU

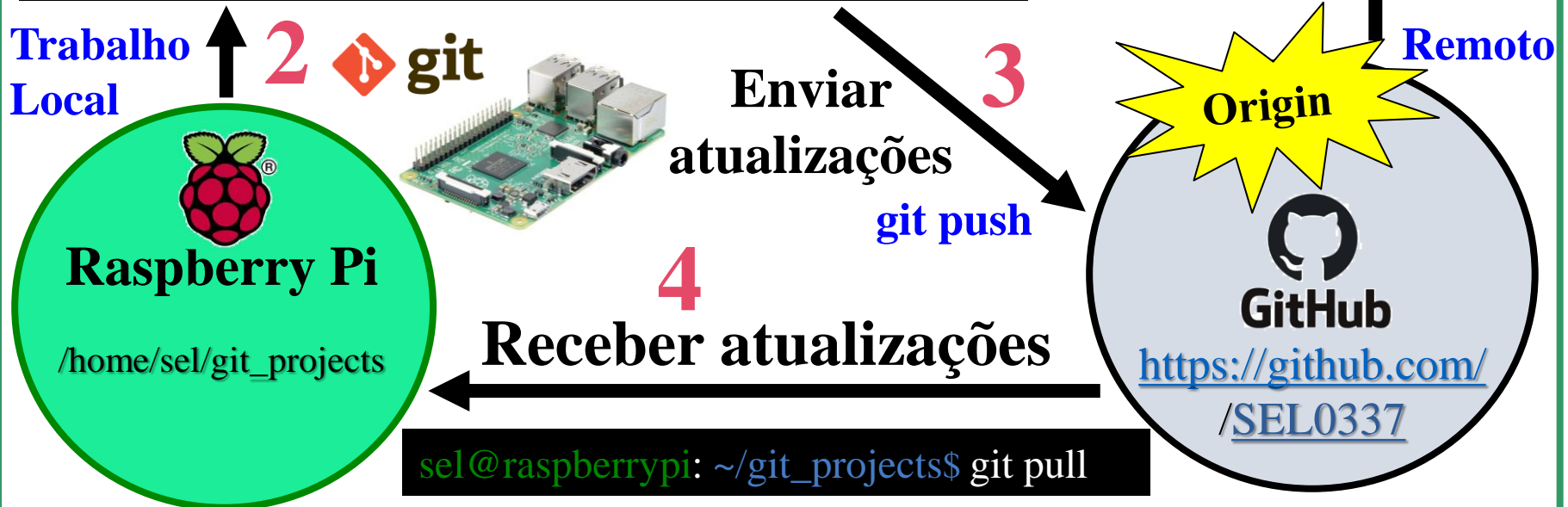
Estágios de um projeto com Git e GitHub



Práticas com Git e GitHub usando a Rasp.

Ambiente de trabalho local: criação edição, remoção...

```
sel@raspberrypi: ~/git_projects$ git clone https://github.com/user/SEL0630
sel@raspberrypi: ~/git_projects$ git add .
sel@raspberrypi: ~/git_projects$ git commit -a -m "Mensagem"
sel@raspberrypi: ~/git_projects$ git rm <file> && git status
sel@raspberrypi: ~/git_projects$ git checkout && git reset
sel@raspberrypi: ~/git_projects$ git push && git pull
sel@raspberrypi: ~/git_projects$ git log
sel@raspberrypi: ~/git_projects$ nano .gitignore
```



Trabalhando em um projeto com Git/GitHub

- ✓ Desenvolver um projeto com controle de versão usando Git, trabalhando em um repositório local (working area: Raspberry Pi) e repo remoto com GitHub. Reproduzir os passos 1-8 a seguir:

- ✓ **Passo 1** - Criando arquivos no repo local e os enviando para repo remoto.
 - **Git commit:** `-m` = “mensagem entre aspas”. `Git commit -a -m “mensagem”` . Parâmetro “-a” envia commit de vários arquivos que estão na *staging area*.
 - É importante sempre comunicar por meio de uma mensagem do que se trata o commit para registrar no histórico ou comunicar equipe quando enviado ao repo remoto.

```
touch teste.txt
git status
git add teste.txt
git status
git commit -m "Enviando arquivo teste"
git status
git push
#user-name: seu nome de usuário no GitHub
#passwd: colar o token gerado na sua conta GitHub
git status
# acessar o repo na sua conta no GitHub e verifique os commits
```

Trabalhando em um projeto com Git/GitHub

- ✓ **Passo 2** – Realizando novas alterações/edição de arquivos e enviando as atualizações para o repo remoto.
 - **Git add** . = Colocando “.” após “git add”, todos os novos arquivos ou alterações são adicionadas à **staging área**.
 - **Git reset** = Desfaz a ação do comando git add, retirando os arquivos da **staging área**.

```
# exemplo de edição
nano config.txt # use uma IDE de sua preferência - modifique o arquivo
git status # lista a modificação
git add config.txt # adiciona a atualização
git status

git reset config.txt # caso deseje desfazer a adição de arquivos

git reset . # se vários arquivos haviam sido adicionados, desfaz todos

# altere novamente o arquivo se for o caso

#adicionando novamente:
git add .
git commit -a -m "Adicionando funcionalidade Y"
git push # envia as mudanças para o GitHub - checar!
```

Trabalhando em um projeto com Git/GitHub

- ✓ **Passo 3**— Recebendo atualizações do repo remoto ou de outros repos locais
 - Acesse o repo remoto no GitHub e adicione ou modifique algum arquivo;
 - **Testes:** faça um novo clone do repo remoto para um outro diretório local e faça modificações localmente entre os dois repositórios locais e mantenha os conteúdos iguais com git push e git pull.

```
# o comando git pull irá buscar as atualizações no repo remoto e disponibilizar localmente:  
git pull  
ls # verifique os novos arquivos ou mudanças nos existentes
```

```
#atualização entre repos locais diferentes: crie outro diretório em outro local na /home/sel ou dentro de  
~/git_projects/  
mkdir repo2 && cd repo2  
git clone https://github.com/SEL0337 . # o ponto ao final com espaço, clona diretamente os arquivos para  
dentro do diretório criado localmente "repo2"  
  
# faça atualizações, visando simular que colegas estão trabalhando com você no mesmo projeto, mas em máquinas  
locais diferentes.  
touch PWM.py  
touch comandos_cmd.txt  
git add . && git commit -a -m "atualização da lib X"  
git push  
  
cd <> #retorne ao repo 1, e receba as atualizações  
git status && ls  
git pull && ls #veja os arquivos inseridos no repo2 agora também neste repo. Também poderia ser feito a  
operação contrária
```

Trabalhando em um projeto com Git/GitHub

✓ Passo 4— Removendo arquivos entre os repositórios

- ✓ A remoção de arquivos também deve ser reportada por **git push** e **git pull** entre os repos locais e remotos!

```
git rm PWM.py #comando "rm" remove localmente o arquivo
git status # sinaliza a alteração
git add . # adiciona a alteração
git commit -a -m "Arquivo não é necessário"
git push
```

```
# a partir de agora, no repo remoto o arquivo foi
removido: verifique!
# outros repos locais devem atualizar com "git pull"
```

Trabalhando em um projeto com Git/GitHub

- ✓ **Passo 5**– Movendo e renomeando arquivos entre repositórios
 - ✓ O comando “mv” é usando nos dois sentidos: mover e renomear

```
mkdir python #criar uma pasta dentro do repo
git mv comandos_cmd.txt python/comandos_cmd.txt # move arquivo para a
pasta python

git status
git add .
git commit -a -m "Alteração de local"
git push

git mv python/comandos_cmd.txt python/comandos_cmd_2.txt # "renomeia"
arquivo

# comunicar alterações novamente: git add, commit e push...
```

Trabalhando em um projeto com Git/GitHub

- ✓ **Passo 6**– Desfazendo alterações e ignorando arquivos
 - ✓ Comando “git checkout”: remove arquivo que foi inserido na árvore de alterações com “git add”
 - ✓ Arquivo com extensão “.gitignore”: ignora arquivo indicado dentro dele no commit

```
nano proj.html # modificar arquivo
git status
git checkout # remove da árvore de alterações
git status
```

```
#exemplo: arquivo que tenha dados confidenciais
touch sense.txt
nano .gitignore # digitar dentro do arquivo “.gitignore” o
nome do arquivo a ser ignorado: "sense.txt"
git status
git add .
git commit -a -m "Ignorar"
```

Trabalhando em um projeto com Git/GitHub

- ✓ **Passo 7**– Histórico de versões, branches, merging, e revertendo alterações
 - ✓ **HEAD:** ponto para a referência do branch atual (**versão local que se está trabalhando**) que, por sua vez, é um ponto para o ultimo commit realizado nesse branch;

```
git log # lista histórico dos commits
git log -follow <arquivo> # histórico de um dado arquivo
git log - - reverse # ordem reversa
git revert <commit ID> # reverte um commit informando seu ID a partir de git log
Atenção: o comando acima reverte as alterações do commit e do arquivo de forma
permanente no diretório local

git branch <nome> #cria novo branch
git branch -a #visualiza branches criados
git branch -d <nome> #remove branch
git checkout <nome> # altera branch

git merge <branch> # faz o merge de dois ou mais históricos, unindo em único branch

git reset HEAD~1 # Desfaz o último commit e retorna para working area. Ou: ~2; ~3.
git reset -hard # desfaz um commit e exclui o arquivo, mantendo o estágio anterior
git reset -hard origin/main #dando reset para o estado no branch no último push
```

Trabalhando em um projeto com Git/GitHub

✓ Passo 8— Explorando outros comandos do Git

```
git fetch # similar ao git pull -> busca atualizações, porém,  
não incorpora diretamente ao repo local, sendo uma forma mais "segura"
```

```
git diff # indica diferenças entre versões de arquivos alterados, com  
base nas últimas versões commitadas.
```

```
git rm -r <diretório> # remove diretório
```

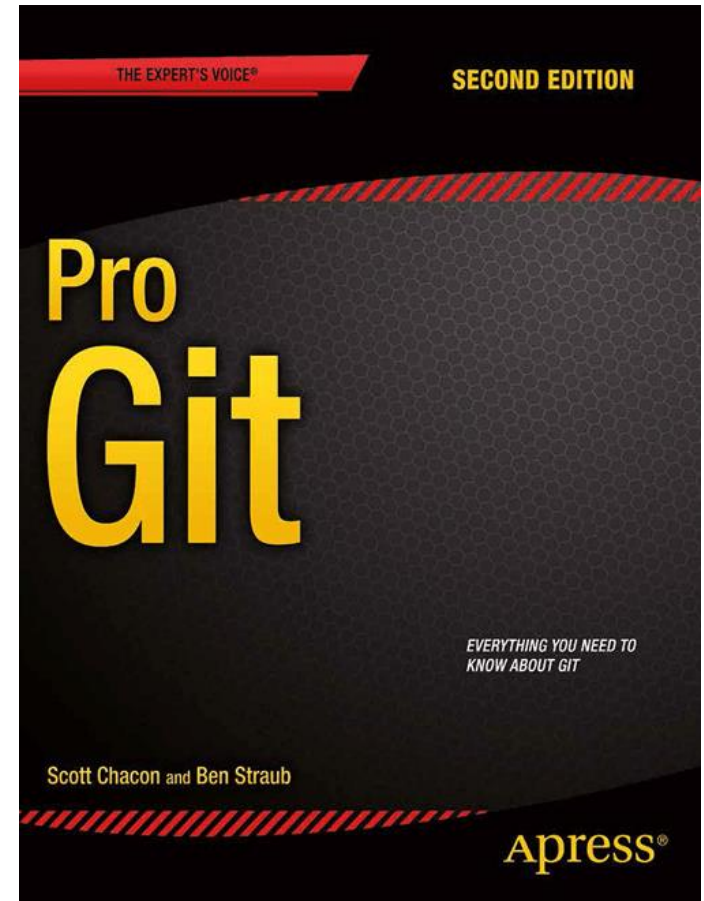
```
git stash # os arquivos serão armazenados em uma área reservada para não  
sofrer modificações. O parâmetro: git stash pop desfaz a operação
```

```
git remote add origin https://github.com/... # adiciona origem remota  
quando um repo local é iniciado com git init, e não clonado.
```

```
git push -u origin master # transfere commits locais para GitHub no  
branch "master" - atualmente sendo substituído por "main"
```


Material de apoio/complementar sobre Git

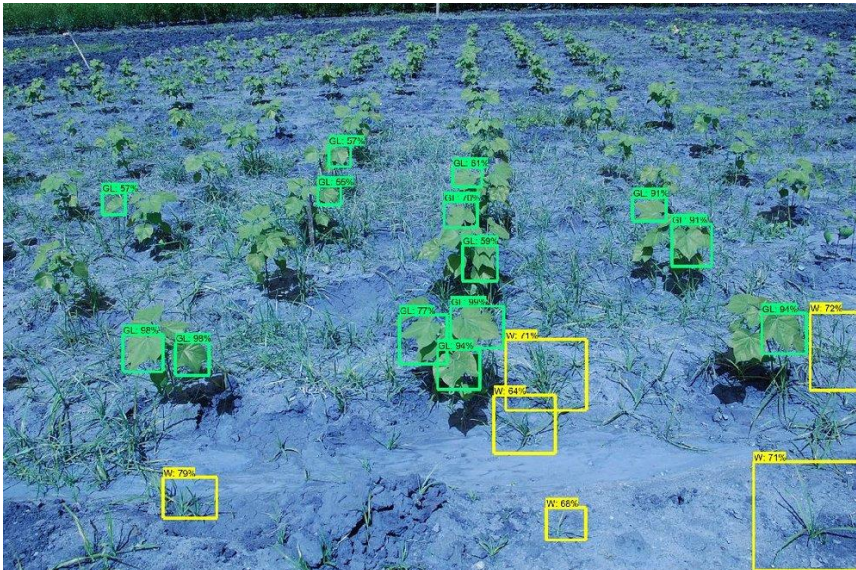
- **Pro Git book** -Download gratuito em:
- <https://git-scm.com/book/pt-br/v2>
- **Documentação oficial**
<https://git-scm.com/doc>
- **Licenças open source:**
<https://choosealicense.com>
- **Praticando Branchs:**
<https://learngitbranching.js.org>



Interface com a Câmera e Visão Computacional

Visão computacional (*computer/machine vision*)

- ✓ Área relativa à ciência da computação e inteligência artificial que busca a obtenção de informação a partir de imagens e vídeos.
- ✓ Análise, interpretação e extração de informações relevantes para tomada de decisão, com diversas aplicações importantes: agricultura, veículos autônomos, biometria facial, imagens médicas, manufatura... **Recurso que pode ser contemplado em sistemas embarcados!**



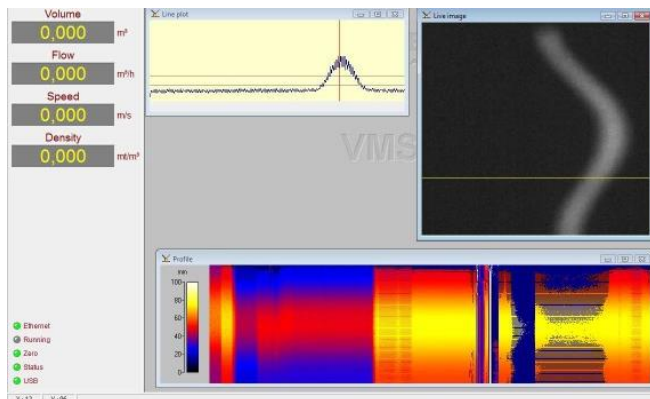
<https://images.squarespace-cdn.com/content/v1/55fdcdce6e4b0e8002e80318/1523955264411-KQL0ACFK0665LY0X2NWU/IMG-20180313-WA0019.jpg?format=1000w>



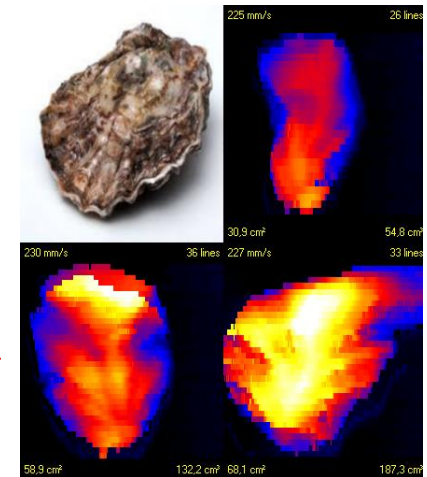
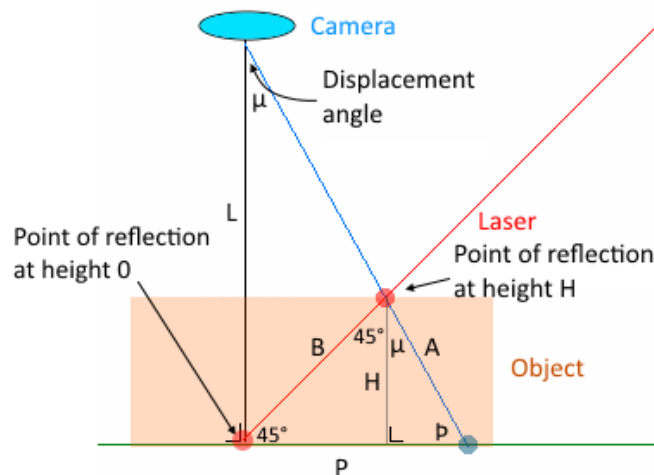
https://editor.analyticsvidhya.com/uploads/23757Computer_vision.jpeg

Visão computacional embarcada

- **Embedded vision - exemplos:** <https://qengineering.eu/embedded-vision.html>

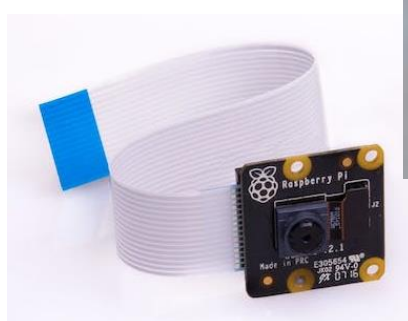
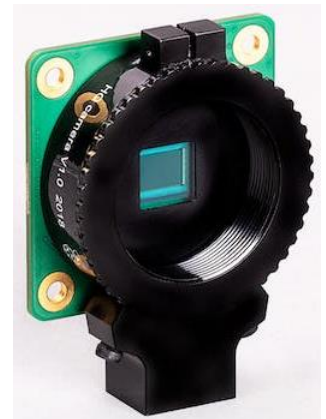


<https://qengineering.eu/embedded-vision.html>



Visão computacional na Raspberry Pi

- ✓ Considerações importantes: **Linguagem de programação, sistema operacional, drivers, periféricos, GPU**
- ✓ Cameras da Rasp. Foundation: <https://www.raspberrypi.com/products/>
- ✓ <https://www.raspberrypi.com/documentation/accessories/camera.html>



<https://www.secedstudio.com/blog/2020/01/20/20-best-raspberry-pi-camera-projects-to-try-in-2020/>

<https://www.raspberrypi.com/products/>

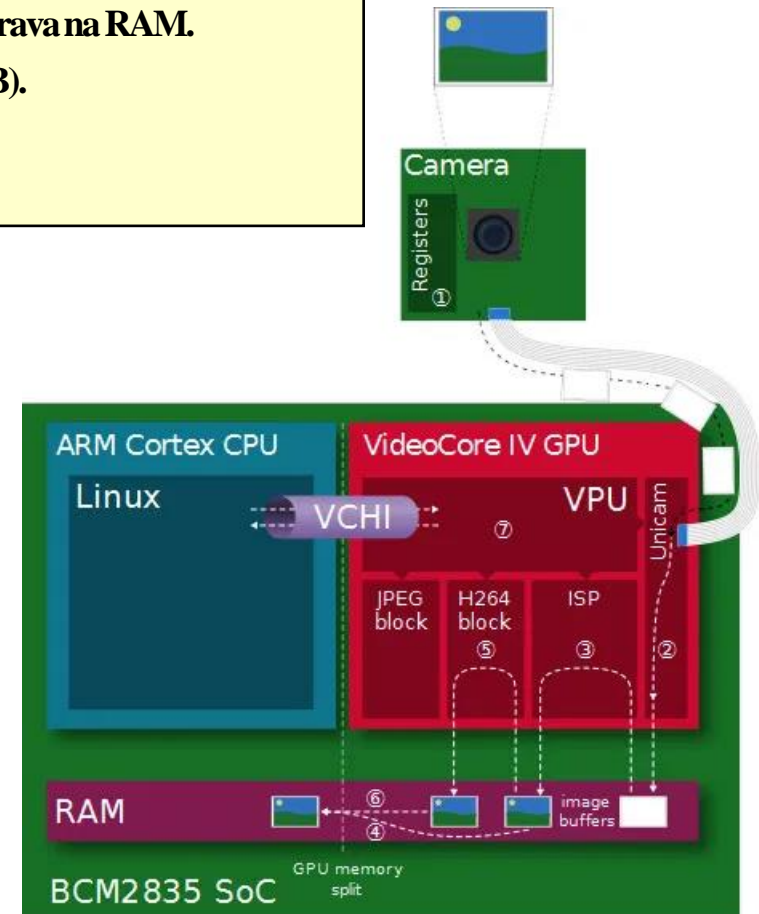


Papel da GPU da Raspberry Pi

- ✓ Recebe os pixels capturados por meio da interface CSI-2 (Unicam) e grava na RAM.
- ✓ A imagem será exibida como uma saída não codificada (YUV ou RGB).
- ✓ Os dados são copiados para a CPU via DMA (*direct memory access*).
- ✓ A codificação do vídeo ocorre por H264 ou MPEG

Sensor elements								->	Frame 1	
1	1	1	1	1	1	1	1			
0	0	0	0	0	0	0	0	Rst		
2	2	2	2	2	2	2	2			
2	2	2	2	2	2	2	2			
2	2	2	2	2	2	2	2			
2	2	2	2	2	2	2	2			

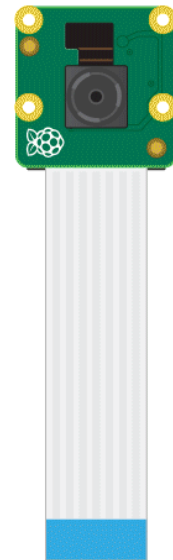
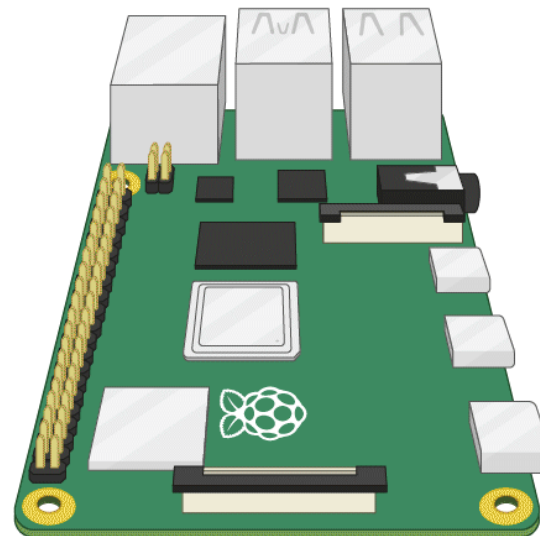
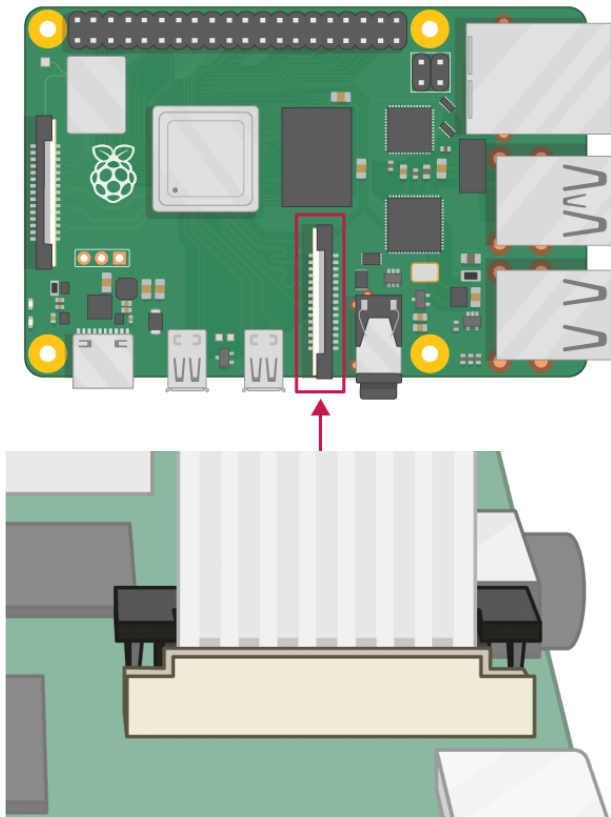
#	Resolution	Aspect Ratio	Framerates	Video	Image	FoV	Binning
1	1920x1080	16:9	1 < fps <= 30	x		Partial	None
2	2592x1944	4:3	1 < fps <= 15	x	x	Full	None
3	2592x1944	4:3	1/6 <= fps <= 1	x	x	Full	None
4	1296x972	4:3	1 < fps <= 42	x		Full	2x2



<https://picamera.readthedocs.io/en/release-1.13/fov.html>

Getting Started with the Camera Module

- ✓ **LAB:** Conecte o módulo da câmera na Raspberry Pi. Atente-se à conexão conforme imagens, com a placa desligada, verificando a posição correta do cabo flat. Vide tutorial aqui: <https://projects.raspberrypi.org/en/projects/getting-started-with-picamera/2>



Documentação e biblioteca da câmera na Raspberry Pi

✓ Acessar: https://www.raspberrypi.com/documentation/computers/camera_software.html

Computers

Getting started

Raspberry Pi OS

Configuration

The config.txt file

Legacy config.txt options

The Linux kernel

Remote access

Camera software

Introducing the Raspberry Pi Cameras

libcamera and libcamera-apps

Introduction

Getting Started

Troubleshooting

libcamera-hello

libcamera-jpeg

libcamera-still

libcamera-vid

libav integration with libcamera-vid

libcamera-raw

libcamera-detect

Common Camera Linux Options

Camera software

Introducing the Raspberry Pi Cameras

[Edit this on GitHub](#)

There are now several official Raspberry Pi camera modules. The original 5-megapixel model was **released** in 2013, it was followed by an 8-megapixel **Camera Module 2** which was **released** in 2016. The latest camera model is the 12-megapixel **Camera Module 3** which was **released** in 2023. The original 5MP device is no longer available from Raspberry Pi.

Additionally a 12-megapixel **High Quality Camera** with CS- or M12-mount variants for use with external lenses was **released in 2020 and 2023** respectively. There is no infrared version of the HQ Camera.

All of these cameras come in visible light and infrared versions, while the Camera Module 3 also comes as a standard or wide FoV model for a total of four different variants.

Further details on the camera modules can be found in the **camera hardware** page.

All Raspberry Pi cameras are capable of taking high-resolution photographs, along with full HD 1080p video, and can be fully controlled programmatically. This documentation describes how to use the camera in various scenarios, and how to use the various software tools.

Once you've **installed your camera module**, there are various ways the cameras can be used. The simplest option is to use one of the provided camera applications, such as **libcamera-still** or **libcamera-vid**.

libcamera and libcamera-apps

[Edit this on GitHub](#)

Testando a câmera na Raspberry Pi

- ✓ Acessar: https://www.raspberrypi.com/documentation/computers/camera_software.html
- ✓ Testar os comandos nativos para capturar fotos e vídeos da biblioteca do link acima

```
libcamera-hello
```

```
libcamera-still -o test.jpg
```

```
libcamera-hello -t 0
```

```
libcamera-jpeg -o test.jpg
```

```
libcamera-still -e png -o test.png  
libcamera-still -e bmp -o test.bmp  
libcamera-still -e rgb -o test.data  
libcamera-still -e yuv420 -o test.data
```

```
libcamera-jpeg -o test.jpg -t 2000 --width 640 --height 480
```

```
libcamera-vid -t 10000 -o test.h264
```

```
vlc test.h264
```

Python PiCamera2 Library

- ✓ **Documentação e exemplos:** <https://pypi.org/project/picamera2/>
- ✓ <https://raspberrypi.com/picamera2-raspberry-pi/>
- ✓ **Manual (Raspberry Pi):** <https://datasheets.raspberrypi.com/camera/picamera2-manual.pdf>
- ✓ Biblioteca específica para interagir com a câmera, permitindo controlar e capturar imagens ou vídeos com controle de foco, exposição, resolução etc.

picamera2 0.3.12

`pip install picamera2==0.3.12`

The Picamera2 Library

A libcamera-based Python library for Raspberry Pi cameras

```
import time
from picamera2 import Picamera2, Preview
picam = Picamera2()
config = picam.create_preview_configuration()

picam.configure(config) picam.start_preview(Preview.QTGL)

picam.start()
time.sleep(2)
picam.capture_file("test-python.jpg")

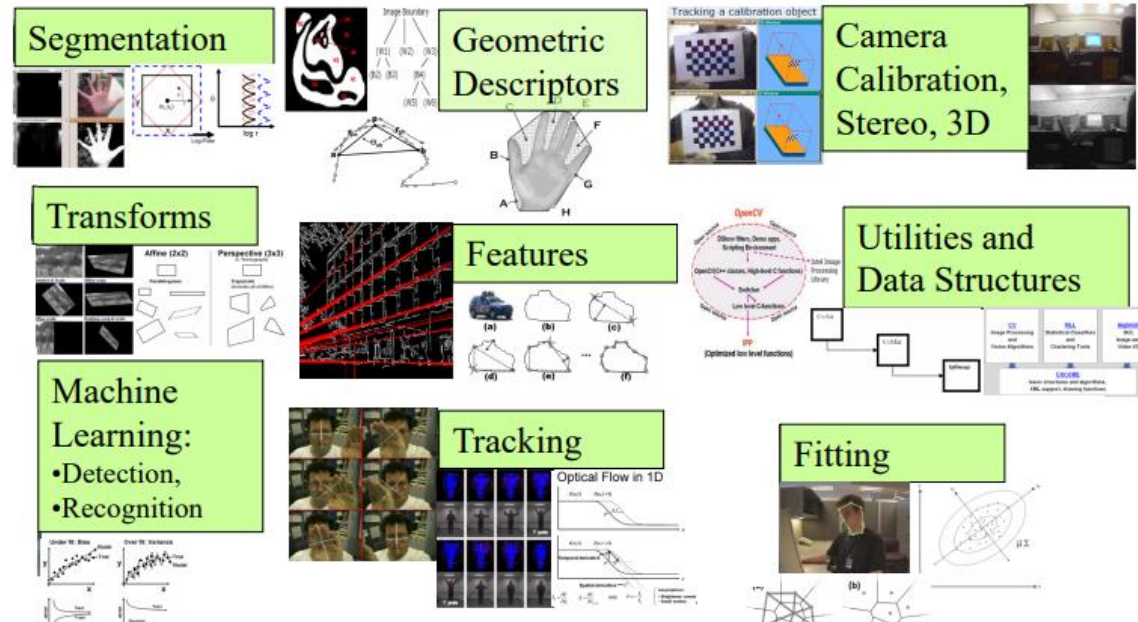
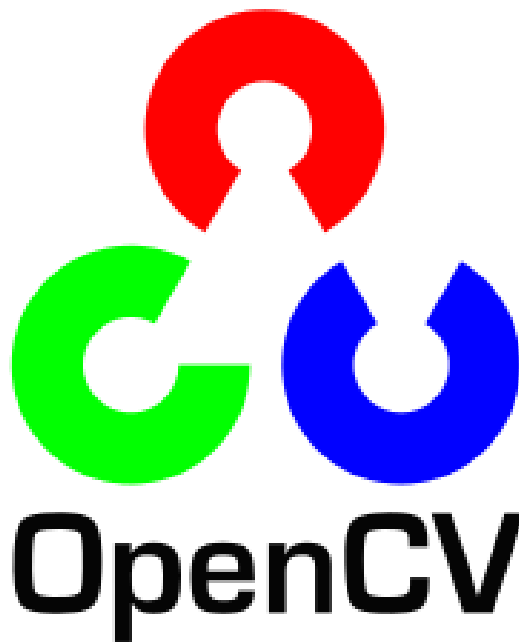
picam.close()
```

Table of contents

Colophon	1
Legal disclaimer notice	1
1. Introduction	4
2. Getting started	5
2.1. Requirements	5
2.2. Installation and updating	5
2.3. A first example	6
2.4. Picamera2's high-level API	7
2.5. Multiple Cameras	8
2.6. Additional software	8
2.6.1. OpenCV	8
2.6.2. TensorFlow Lite	8
2.6.3. FFmpeg	8
2.7. Further examples	8
3. Preview windows	9
3.1. Preview window parameters	9
3.2. Preview window implementations	9
3.2.1. QtGL preview	9
3.2.2. DRM/KMS preview	10
3.2.3. Qt preview	10
3.2.4. NULL preview	11
3.3. Starting and stopping previews	11
3.4. Remote preview windows	12
3.5. Other Preview Features	13
3.5.1. Setting the Preview Title Bar	13
3.5.2. Further Preview Topics	13
3.6. Further examples	13

OpenCV Library

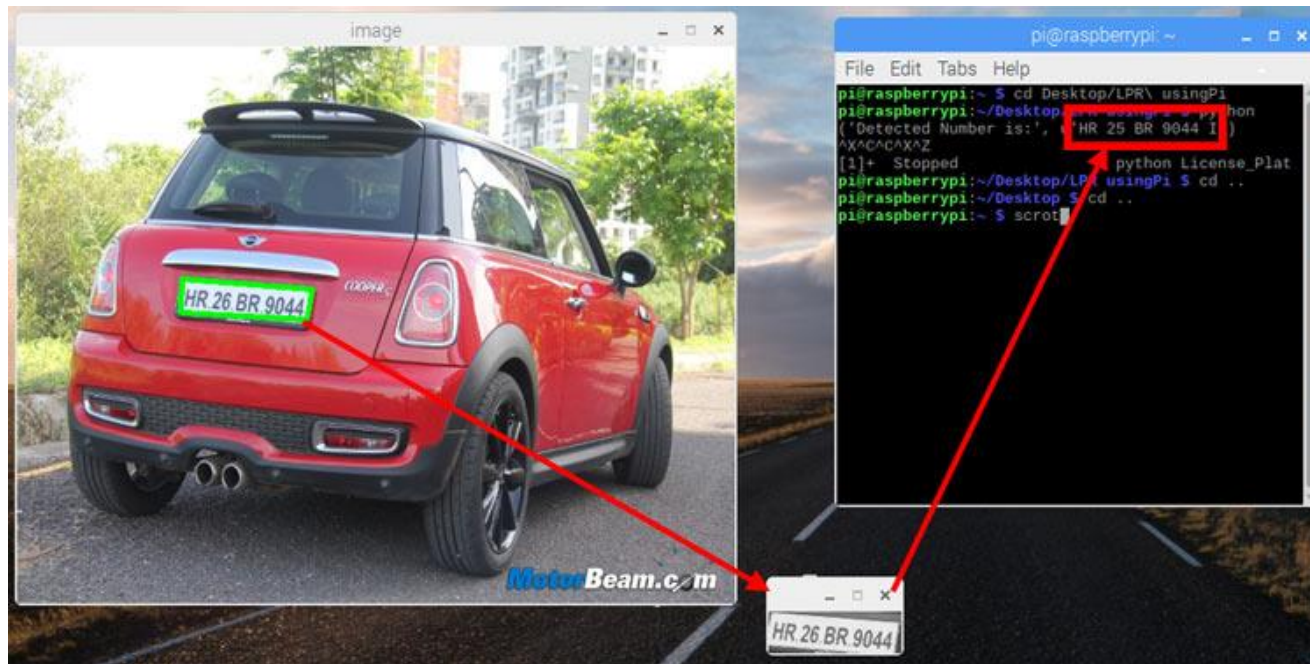
- ✓ *Open Source Computer Vision Library*
- ✓ Biblioteca open source para aplicações de visão computacional e processamento de imagem
- ✓ Integra algoritmos de filtro de imagem, calibração de câmera, reconhecimento de objetos
- ✓ Processamento em tempo real
- ✓ Página e documentação: <https://opencv.org>
- ✓ Compatível com Python - https://docs.opencv.org/3.2.0/d6/d00/tutorial_py_root.html



http://int.eff.cs/-mv/MV1718_v1.pdf

OpenCV Library na Raspberry Pi

- ✓ *sudo apt install python3-opencv*
- ✓ Guia de uso e exemplos: <https://raspberrypi.com/install-opencv-on-raspberry-pi/>



OpenCV Library na Raspberry Pi

- ✓ *sudo apt install python3-opencv*
- ✓ Guia de uso e exemplos: <https://raspberrytips.com/install-opencv-on-raspberry-pi/>

```
1 import cv2
2
3 image = cv2.imread("/home/pat/Downloads/test.jpg")
4 dimensions = image.shape
5 print("Picture dimensions: ", dimensions)
6 |
7 (h, w, d) = dimensions
```

Shell

```
>>> %Run basic.py
Picture dimensions: (450, 800, 3)
```

basic.py ✕

```
1 import cv2
2
3 image = cv2.imread("/home/pat/Downloads/test.jpg")
4 image = cv2.resize(image, (200, 200))
5
6 cv2.imshow('Preview', image)
7 cv2.waitKey(0)
```

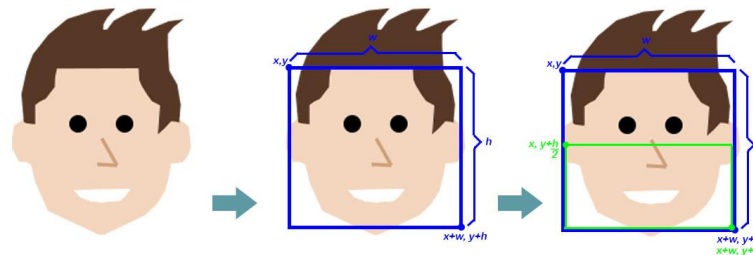

Reconhecimento facial com aprendizado de máquina

Machine learning é a capacidade dos computadores de reconhecer padrões, não apenas a sistemas com câmeras, mas também a filtragem de emails, sistemas de verificação e chatbots. A tecnologia aprende com dados para tomar decisões e oferecer funcionalidades, sendo muito utilizada em sistemas embarcados.



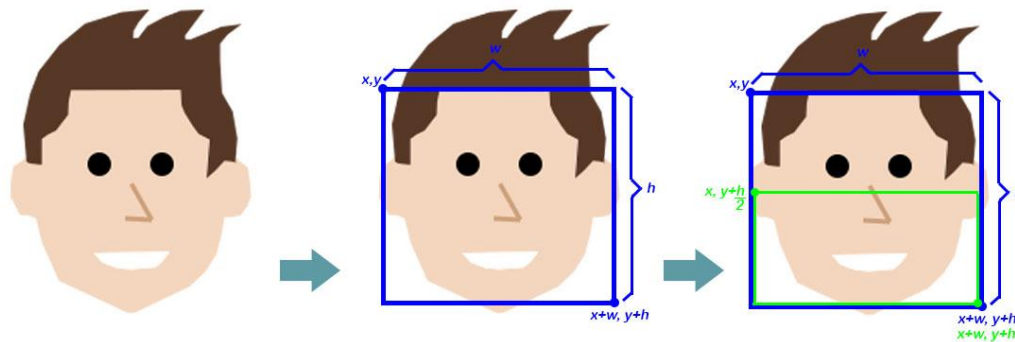
Algoritmo Haar Cascade

- ✓ O método de classificação de imagens **Haar Cascade**, proposto por Viola e Jones em 2001, envolve treinar um modelo para reconhecer padrões em imagens.
- ✓ Esse modelo é treinado com imagens positivas (contendo o objeto a ser reconhecido) e negativas (sem o objeto).
- ✓ O algoritmo analisa a intensidade dos pixels e busca padrões em cascata, otimizando o processo. Neste caso, a biblioteca **OpenCV** fornece modelos pré-treinados para detecção de faces, olhos, sorrisos, etc., facilitando seu uso.
- ✓ Os modelos são **arquivos XML** (linguagem de marcação com tags para organizar e estruturar informações) com coordenadas para verificar características nas imagens de entrada.



Projeto de reconhecimento facial

- ✓ Fazer o download o do algoritmo Haar Cascade por meio do arquivo **haarcascade_frontalface_default.xml** [disponível aqui](#)
- ✓ Instale as bibliotecas Python OpenCV e PiCamera2:
- ✓ ***`pip install opencv-python && pip install picamera2`***
- ✓ Importe os módulos “*PiCamera2*”, e “*cv2*” (OpenCV) e utilize o código base fornecido a seguir.
- ✓ O código a seguir implementa a detecção de rostos pelo método Haar Cascade usando a OpenCV. O script tira fotos do rosto detectado e salva em um diretório que é criado diretamente no programa (testar)



Tutorial para uso na Raspberry Pi

```
#!/usr/bin/python3
import cv2 # Biblioteca OpenCV
import os  # Biblioteca para operações do sistema
import time # Biblioteca de tempo
from picamera2 import Picamera2 # Biblioteca da câmera da Raspberry Pi

# Carrega o classificador para detecção facial (informar o caminho do arquivo)
face_detector = cv2.CascadeClassifier("/home/sel/haarcascade_frontalface_default.xml")

# Inicia uma thread para gerenciar janelas de visualização
cv2.startWindowThread()

# Inicializa a câmera da Raspberry Pi
picam2 = Picamera2()

# Configura a câmera para criar uma visualização com formato de representação de cores 32 bits
"XRGB8888" e resolução de 640x480 pixels
picam2.configure(picam2.create_preview_configuration(main={"format": 'XRGB8888', "size": (640,
480)}))

# Inicia a câmera
picam2.start()

# Define o diretório onde as imagens com rostos detectados serão armazenadas
output_directory = "detected_faces"

# Cria o diretório, se ele não existir
os.makedirs(output_directory, exist_ok=True)
```

Tutorial para uso na Raspberry Pi

```
# continuação do código do slide anterior...

# Loop para captura e detecção de rostos
while True:
    # Captura um quadro da câmera e armazena na variável
    im = picam2.capture_array()

    # Converte a imagem colorida para escala de cinza
    grey = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)

    # Usa o classificador em cascata para detectar rostos na imagem em escala de cinza
    faces = face_detector.detectMultiScale(grey, 1.1, 5)

    # Loop para processar cada rosto detectado
    for (x, y, w, h) in faces:
        # Desenha um retângulo verde ao redor do rosto na imagem original
        cv2.rectangle(im, (x, y), (x + w, y + h), (0, 255, 0))

        # Gera um nome de arquivo único com base no carimbo de data/hora
        timestamp = int(time.time())
        filename = os.path.join(output_directory, f"face_{timestamp}.jpg")

        # Salva apenas a porção da imagem que contém o rosto detectado como um arquivo JPEG
        cv2.imwrite(filename, im[y:y+h, x:x+w])

    # Exibe a imagem com os retângulos desenhados em uma janela com o título "Camera"
    cv2.imshow("Camera", im)

    # Aguarda 1 milissegundo antes de continuar o loop e capturar a próxima imagem
    cv2.waitKey(1)
```

Mais detalhes e aprofundamento c/ módulo da câmera

- **Documentação dos Módulos de Câmera da Raspberry Pi**
<https://www.raspberrypi.com/documentation/accessories/camera.html>
- **Documentação e manual de uso da biblioteca da câmera da Raspberry Pi:**
https://www.raspberrypi.com/documentation/computers/camera_software.html
<https://datasheets.raspberrypi.com/camera/picamera2-manual.pdf>
- **Exemplos de uso da biblioteca PiCamera2:**
<https://raspberrytips.com/picamera2-raspberry-pi/>
- **Exemplos de uso da biblioteca OpenCV:**
<https://raspberrytips.com/install-opencv-on-raspberry-pi/>
- **Disciplina “SEL0339 Introdução à Visão Computacional”**

Referências e créditos

- Chacon S. Straub B. Pro Git – Everthing you need to know about git. 2nd. Apress. 2014.
- GitHub, Inc. Disponível em: <https://github.com>
- GIT SCM. Disponível em: <https://git-scm.com>
- Portal Embarcados. Disponível em: <https://embarcados.com.br>
- Python –Disponível em: <https://www.python.org>.
- Raspberry Pi Foundation. Disponível em <https://www.raspberrypi.org> - <https://www.raspberrypi.com>