

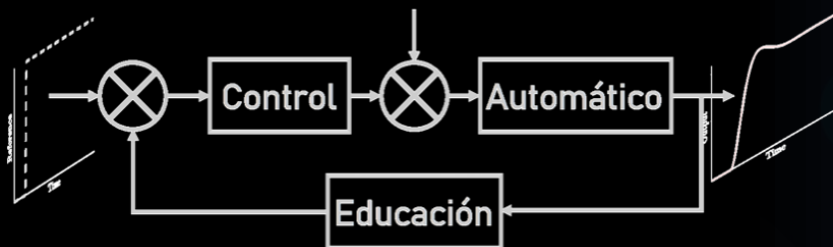


git



GitHub

Sergio Andres Castaño Giraldo



O que é Git?



- O Git é um sistema de controle de versão distribuído gratuito e de código aberto projetado para lidar com tudo, desde projetos pequenos a muito grandes com velocidade e eficiência.
- Em palavras mais simples, o Git é uma ferramenta que os desenvolvedores usam para rastrear e controlar as alterações que fazem em seu código.

Vantagens de usar Git



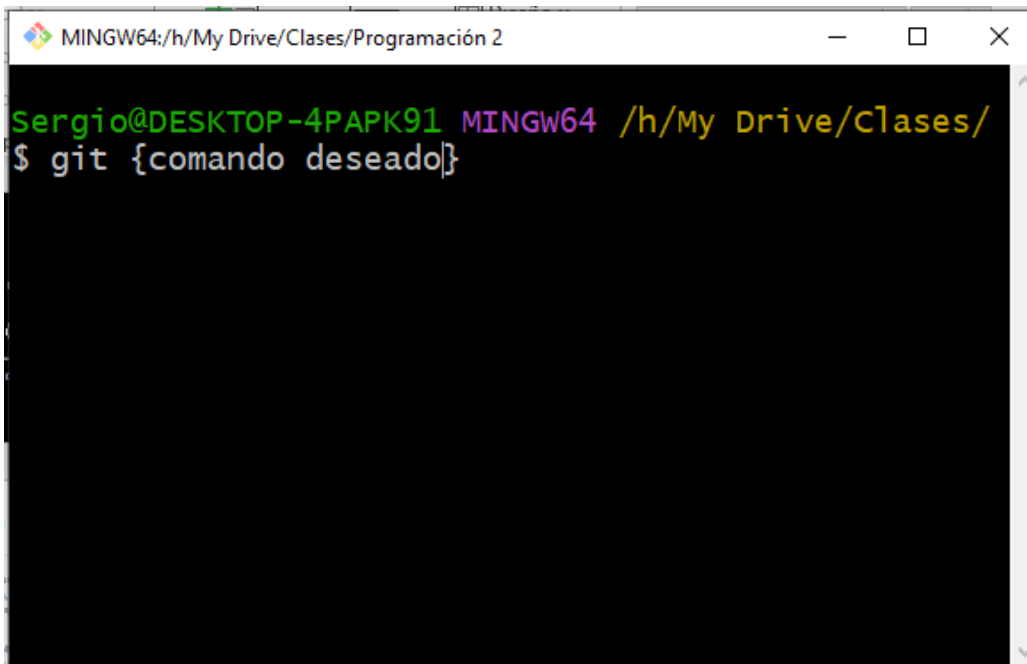
- **Controle de versão:** o Git permite que os desenvolvedores "salvem" diferentes versões de seus projetos, o que significa que eles podem reverter para uma versão anterior se algo der errado.
- **Colaboração:** Git torna mais fácil para as equipes trabalharem juntas no mesmo projeto. Cada desenvolvedor pode trabalhar em seu próprio ramo, fazer alterações e mesclar seu trabalho com o restante da equipe.
- **Distribuído:** Ao contrário de outros sistemas de controle de versão, o Git é distribuído, o que significa que cada desenvolvedor tem uma cópia completa do repositório em sua máquina local. Isso permite que você trabalhe offline e forneça um backup se a cópia central for perdida.
- **Eficiência e velocidade:** Git é incrivelmente rápido e eficiente. Você pode executar a maioria das operações localmente, o que significa que não requer conexão com a Internet para funcionar.
- **Integração com ferramentas de desenvolvimento modernas:** a maioria das ferramentas e serviços de desenvolvimento modernos, como GitHub, GitLab e Bitbucket, são criados em torno do Git, facilitando o trabalho colaborativo e a hospedagem de código na nuvem.

Uso de Git



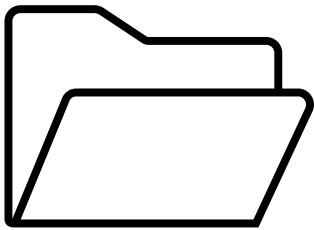
Linha de Comandos

Para fazer uso do GIT vamos começar aprendendo uma série de comandos no terminal para que o Git execute as tarefas que queremos.

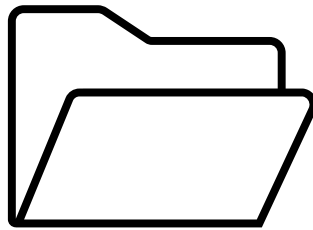
A screenshot of a Windows terminal window. The title bar reads "MINGW64:/h/My Drive/Clases/Programación 2". The prompt is "Sergio@DESKTOP-4PAPK91 MINGW64 /h/My Drive/Clases/". The command entered is "\$ git {comando deseado}" in a light blue monospace font on a black background.

```
MINGW64:/h/My Drive/Clases/Programación 2
Sergio@DESKTOP-4PAPK91 MINGW64 /h/My Drive/Clases/
$ git {comando deseado}
```

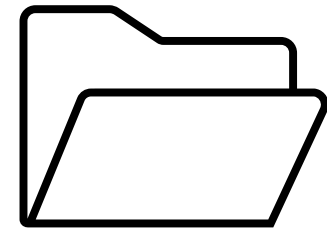
Sem Git



Projeto ver1



Projeto ver2



Projeto ver3

Com Git



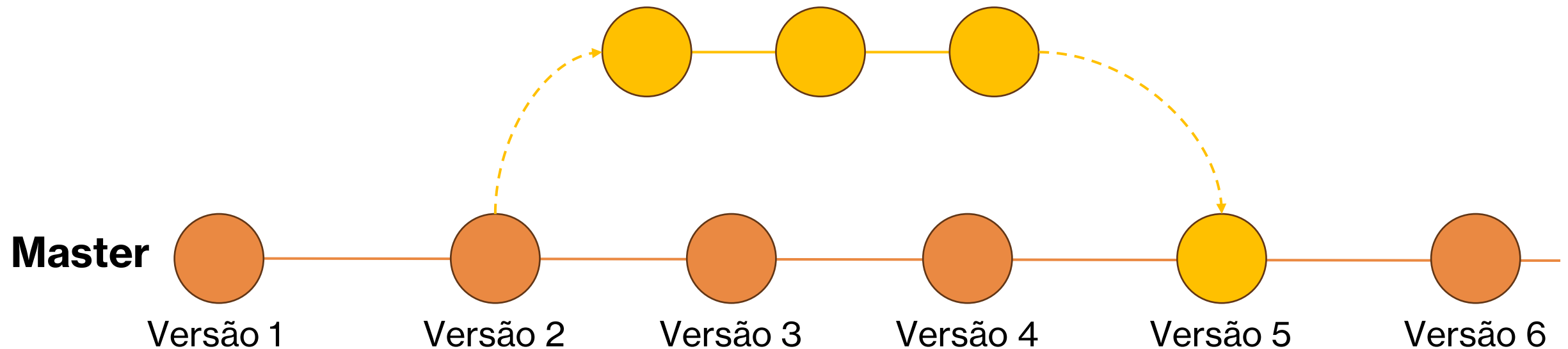
COMMITs (Confirmação)



Com Git



BRANCH (Ramificação)



O que é o GitHub?



- O GitHub é uma plataforma de hospedagem de código que usa o Git para controle de versão.
- Além de ser um local onde os desenvolvedores podem armazenar seu código, o GitHub também oferece diversos recursos adicionais que facilitam o trabalho colaborativo e o gerenciamento de projetos.
- O GitHub permite que os desenvolvedores colaborem em projetos, mantenham um histórico de versão de seus arquivos e revisem e mesclam alterações em seus projetos.

Vantagens de usar GitHub



- **Colaboração:** o GitHub permite que os desenvolvedores trabalhem juntos de forma eficiente. Ferramentas como branches, pull requests e rastreamento de problemas facilitam a colaboração e a revisão do código.
- **Visibilidade:** os repositórios públicos no GitHub podem ser vistos e usados por qualquer pessoa no mundo, facilitando a contribuição aberta e o compartilhamento de código.
- **Integrações:** o GitHub se integra a uma ampla variedade de ferramentas e serviços de desenvolvimento de software, desde sistemas de integração contínua até serviços de implantação e hospedagem.
- **Documentação:** o GitHub fornece uma plataforma para hospedar a documentação do seu projeto junto com o seu código.
- **Contribuições para projetos de código aberto:** o GitHub hospeda um grande número de projetos de código aberto. Contribuir para esses projetos pode ajudá-lo a aprender novas habilidades e fazer conexões na comunidade de desenvolvimento.



Sistema de
Versões



Local para
armazenar
nossos projetos
na nuvem

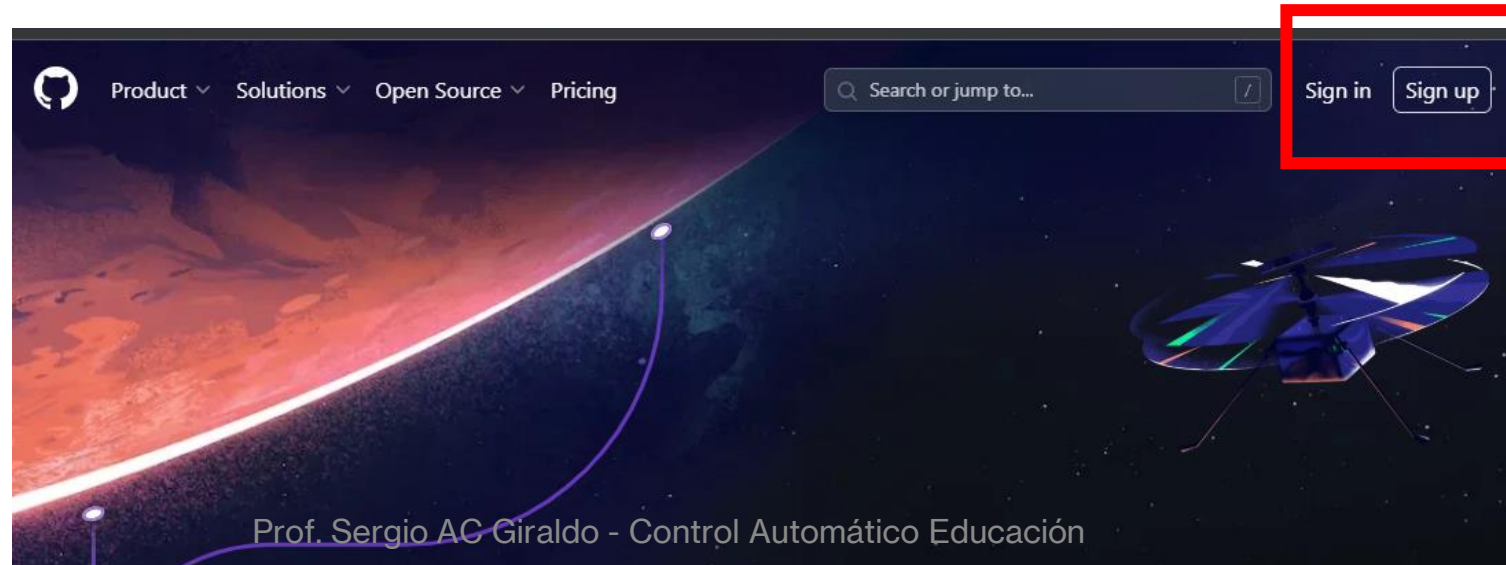
Instalação Git

- Basta acessar o site oficial do GIT: <https://git-scm.com/downloads>

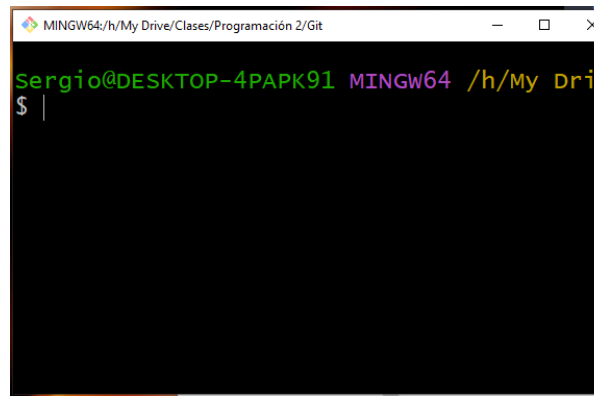


Criação de conta GitHub

- Entre no site oficial e cadastre-se: <https://github.com/>
- Coloque um nome e usuário profissional (Evite colocar apelidos)



Git Bash



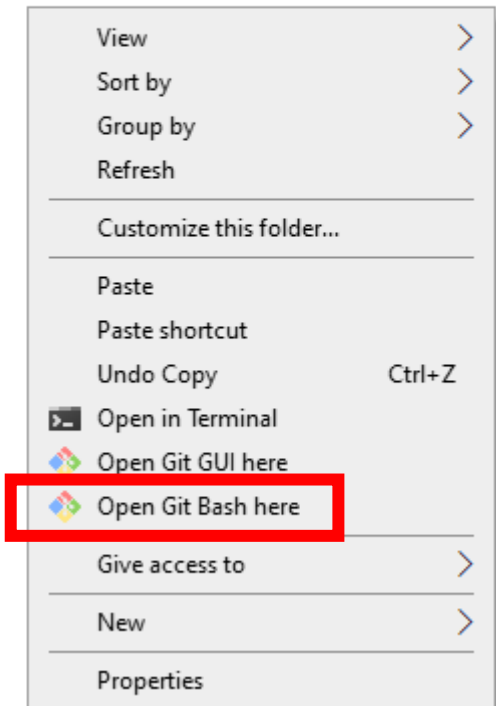
- Git Bash é um aplicativo para Microsoft Windows que fornece uma emulação da interface de linha de comando Bash combinada com uma implementação do Git.
- Em outras palavras, o Git Bash permite que você use o Git a partir da linha de comando, mas também fornece uma série de comandos Unix/Bash que você pode usar no Windows.
- Essa combinação é muito útil para desenvolvedores que trabalham no Windows, mas precisam de um ambiente semelhante ao Unix.

Name

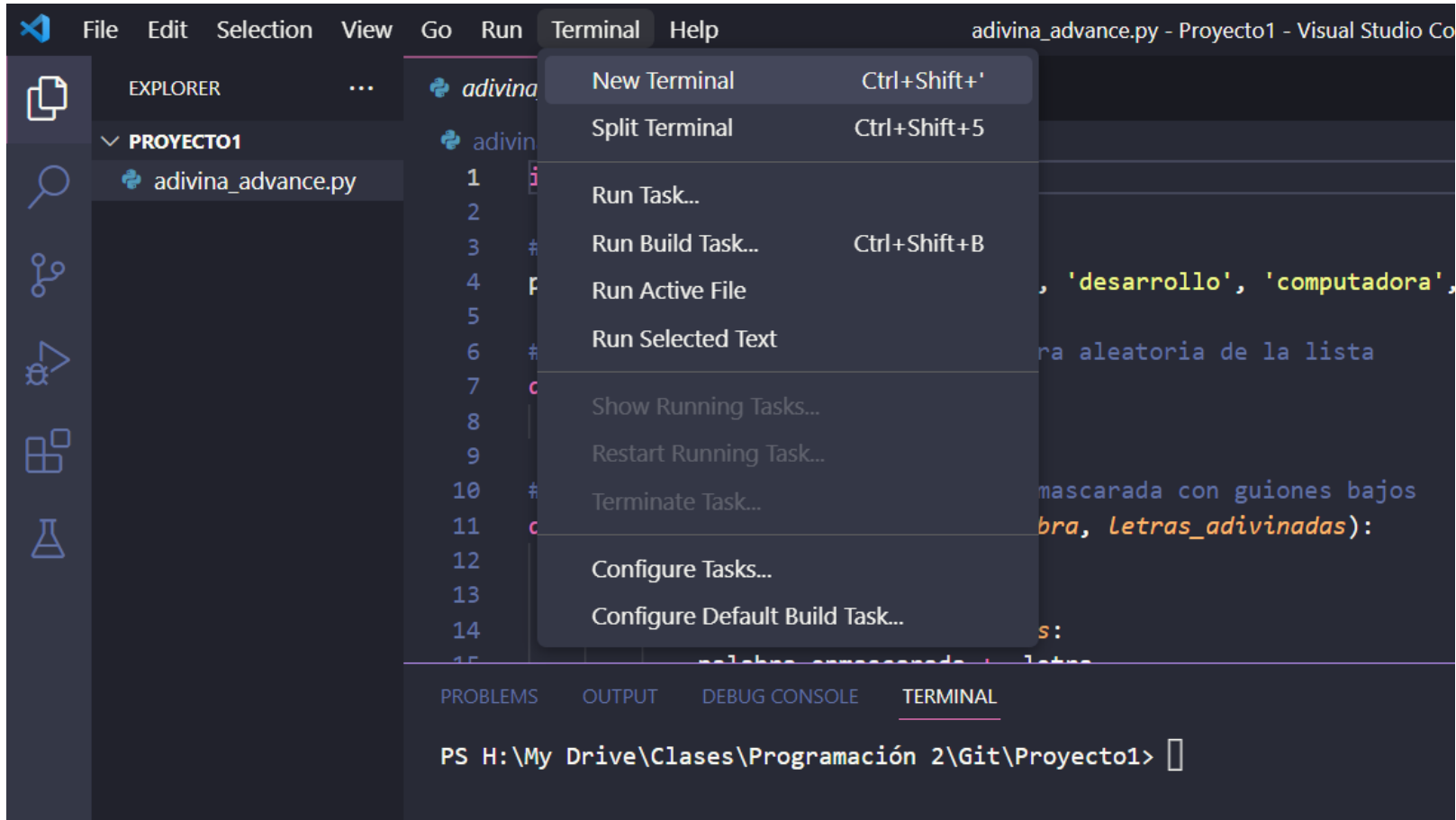
Date modified

Projeto1

7/7/2023 2:03 PM



Git com VS Code



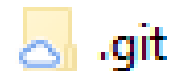
Os comandos do
Git podem ser
usados diretamente
no ambiente VS
Code

Comando git init

- **git init** é a primeira etapa na **criação de um novo repositório Git**. Antes de poder fazer qualquer outra operação no Git (como fazer commit, criar branches, etc.), você precisa ter um repositório para trabalhar.
- Quando você executa o **git init**, o Git cria um novo subdiretório **.git** no diretório atual. Este subdiretório contém todos os arquivos e metadados necessários para o funcionamento do repositório.
- **git init** só deve ser executado uma vez no início de um projeto.

```
PS H:\My Drive\Clases\Programación 2\Git\Proyecto1> git init
Initialized empty Git repository in H:/My Drive/Clases/Programación 2/Git/Proyecto1/.git/
PS H:\My Drive\Clases\Programación 2\Git\Proyecto1> |
```

Name



.git



adivina_advance.py

Arquivos de exemplo



square.py

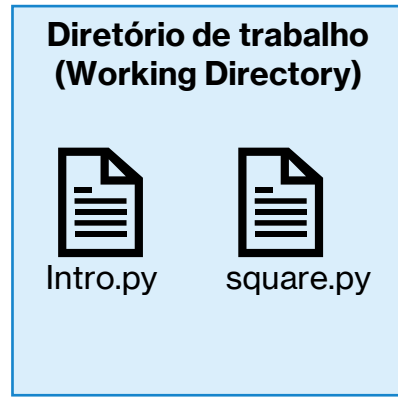
```
for i in range(1, 11):  
    print(f"The square of {i} is {i*i}")
```



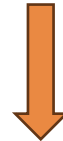
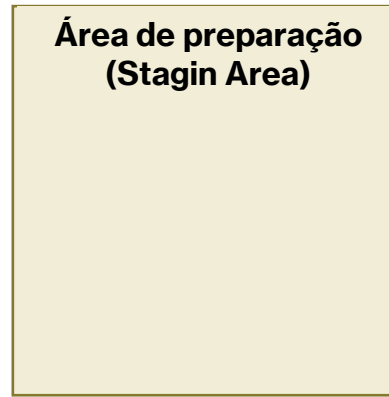
Intro.py

```
print('Hello World!')
```

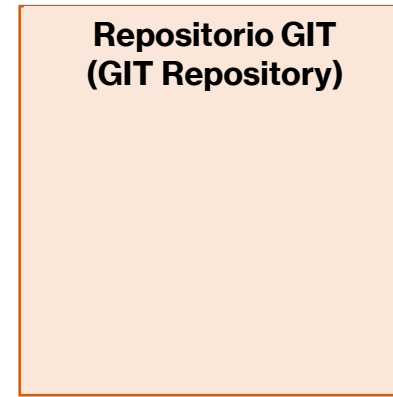

Criação do repositório



Este é o espaço em seu sistema local onde você trabalha com seus arquivos de projeto. O Git não rastreia alterações aqui

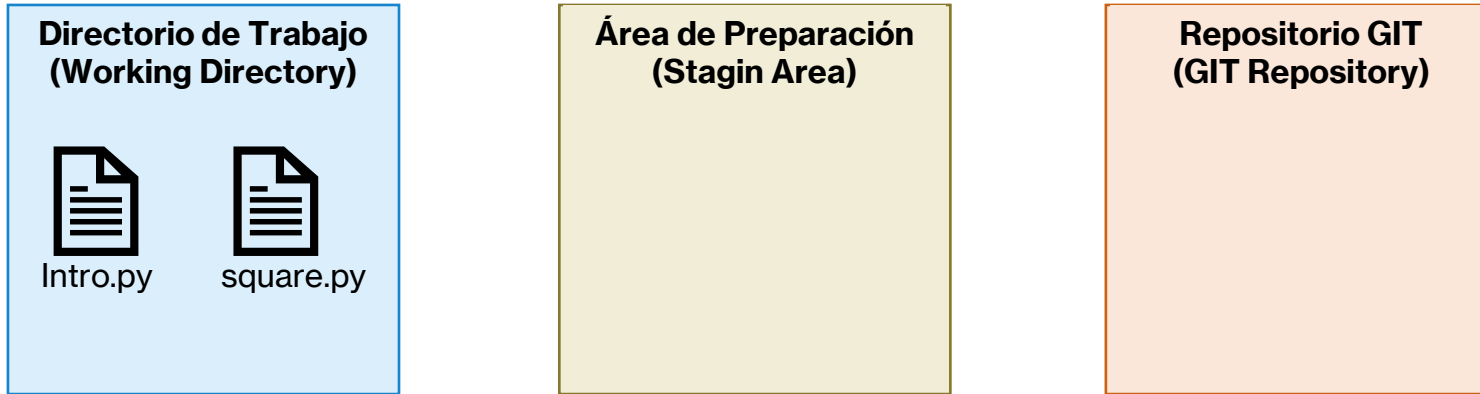


É uma espécie de área de preparação onde o Git rastreia as alterações que você planeja enviar para o seu repositório.



Este é o espaço onde o Git mantém os logs de suas alterações confirmadas.

git status



git status é um comando que exibe o status do diretório de trabalho e da área de preparação.

```
PS H:\My Drive\Clases\Programación 2\Git\Proyecto1> git status
On branch master

No commits yet

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

nothing added to commit but untracked files present (use "git add" to track)
PS H:\My Drive\Clases\Programación 2\Git\Proyecto1>
```

Untracked files: Arquivos que o git não está rastreando ou controlando alterações

Quem é responsável?

Ao fazer um **commit**, o Git registra a identidade do autor na forma de nome e endereço de e-mail. Esses valores podem ser definidos globalmente (para todos os repositórios da sua máquina) ou especificamente para um repositório individual.

```
git config --global user.name "your name"
```

```
git config --global user.email "teu.email@example.com"
```

Esses comandos configuram seu nome e endereço de e-mail globalmente. Se você deseja apenas configurá-los para um repositório específico, pule a opção **--global**

```
*** Please tell me who you are.
```

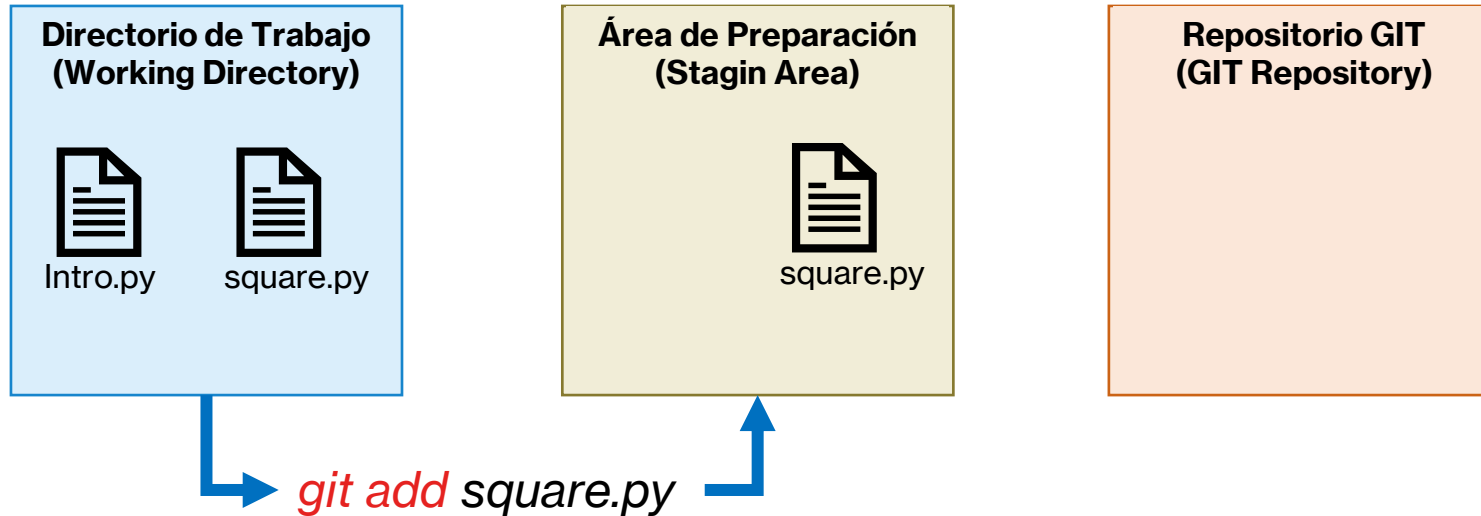
```
Run
```

```
git config --global user.email "you@example.com"  
git config --global user.name "Your Name"
```

```
to set your account's default identity.
```

```
Omit --global to set the identity only in this repository.
```

git add



git add é um comando usado para adicionar alterações de diretório de trabalho à área de preparação do Git.

```
PS H:\My Drive\Clases\Programación 2\Git\Proyecto1> git add square.py
```

```
PS H:\My Drive\Clases\Programación 2\Git\Proyecto1> git status
On branch master
```

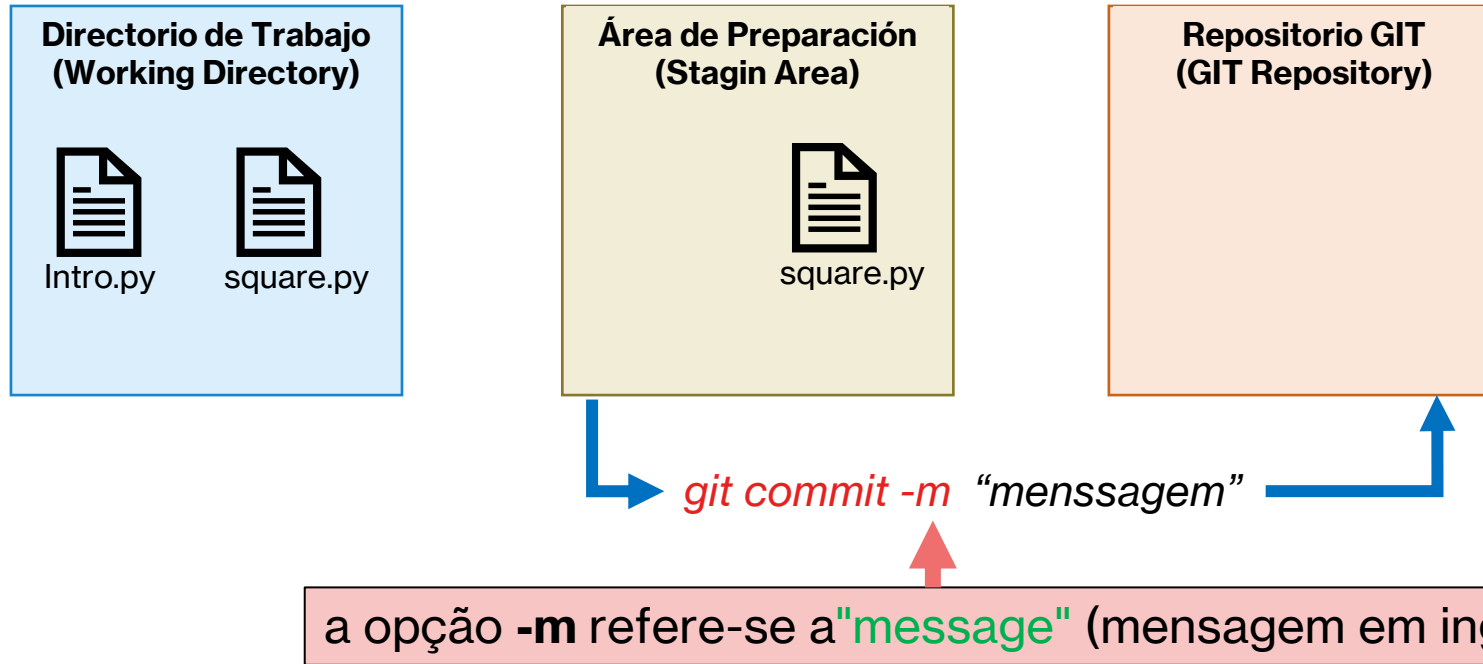
```
No commits yet
```

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

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

Novo arquivo: dentro de stagin área.

git commit



git commit é o comando usado para salvar as alterações que você adicionou à área de preparação em seu repositório Git.

```
PS H:\My Drive\Clases\Programación 2\Git\Proyecto1> git commit -m "inicializamos el proyecto"
[master (root-commit) 91f46f4] inicializamos el proyecto
1 file changed, 2 insertions(+)
 create mode 100644 square.py
PS H:\My Drive\Clases\Programación 2\Git\Proyecto1> █
```

Ao fazer um commit, você está criando uma foto das alterações, juntamente com uma mensagem descritiva que permite lembrar quais alterações foram feitas e por quê.

git log

Directorio de Trabajo
(Working Directory)



Intro.py



square.py

Área de Preparación
(Stagin Area)

Repositorio GIT
(GIT Repository)



square.py

git log é um comando que
exibe um log de histórico de
todos os commits feitos em
um repositório.

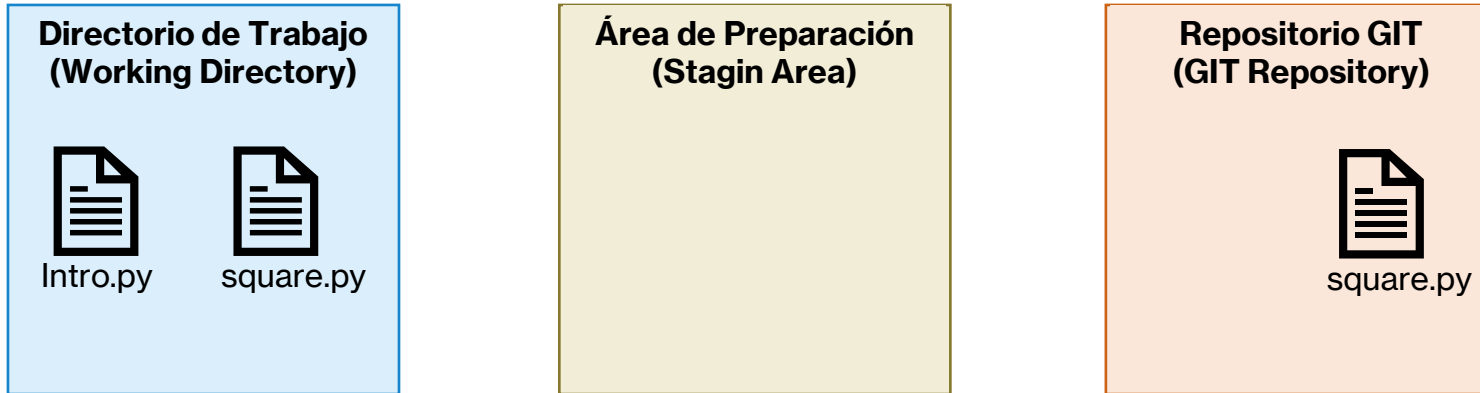
Fornece uma visão detalhada de **quem** fez o commit, **quando** e **quais** mudanças foram incluídas.

HASH: ID da
versão

```
PS H:\My Drive\Clases\Programación 2\Git\Proyecto1> git log
commit 91f46f461c337ff6049e96c3c6ecffeeef305d2a (HEAD -> master)
Author: Sergio Andres Castaño Giraldo <s@outlook.com>
Date: Tue Aug 8 13:11:55 2023 -0300

    inicializamos el proyecto
PS H:\My Drive\Clases\Programación 2\Git\Proyecto1>
```

git log



Variantes e opções úteis

git log --oneline : mostra cada commit em uma única linha, ideal para obter uma visão rápida do histórico.

git log -n 5 : Mostra os últimos 5 commits.

git log --graph --decorate --oneline : Exibe um gráfico ASCII do histórico de commits, indicando branches e tags.

git log --since="2 days ago": Mostra os commits desde 2 dias atrás.

Modificar archivo



square.py

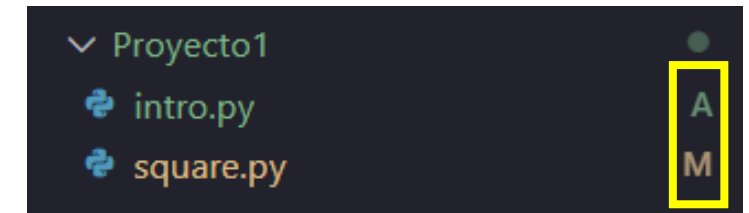
```
for i in range(1, 15):  
    print(f"The square of {i} is {i*i}")
```

Modificamos o range até 15!

git add – Vários Arquivos



`git add .` é um comando usado para adicionar TODOS os arquivos à área de preparação do Git.



```
PS C:\Users\Sergio\My Drive\Clases\Programación 2\Git\Proyecto1> git add .
PS C:\Users\Sergio\My Drive\Clases\Programación 2\Git\Proyecto1> git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   intro.py
    modified:   square.py
```

Novo arquivo: dentro de stagin área.

Arquivo Modificado: dentro de stagin área.

git commit



`git commit` é o comando usado para salvar as alterações que você adicionou à área de preparação em seu repositório Git.

`git commit -m "adicionei o arquivo intro.py e modifiquei square.py"`

```
PS C:\Users\Sergio\My Drive\Clases\Programación 2\Git\Proyecto1> git commit -m "adicionamos
archivo intro.py y modificamos square.py"
[master dd40531] adicionamos archivo intro.py y modificamos square.py
2 files changed, 3 insertions(+), 1 deletion(-)
create mode 100644 intro.py
```

git log – Verificando nuestras versiones

Directorio de Trabajo
(Working Directory)



Intro.py



square.py

Área de Preparación
(Stagin Area)

Repositorio GIT
(GIT Repository)



Intro.py



square.py

git log é um comando que
exibe um log de histórico de
todos os commits feitos em
um repositório.

Com um novo git log podemos verificar as versões atuais do projeto!

```
PS C:\Users\Sergio\My Drive\Clases\Programación 2\Git\Proyecto1> git log
commit dd405318de779cbe8fb83188ee091576fa492292 (HEAD -> master)
Author: Sergio Andres Castaño Giraldo <s[redacted]@outlook.com>
Date: Mon Aug 14 15:17:28 2023 -0300

    adicionamos archivo intro.py y modificamos square.py

commit 91f46f461c337ff6049e96c3c6ecffeeef305d2a
Author: Sergio Andres Castaño Giraldo <s[redacted]@outlook.com>
Date: Tue Aug 8 13:11:55 2023 -0300

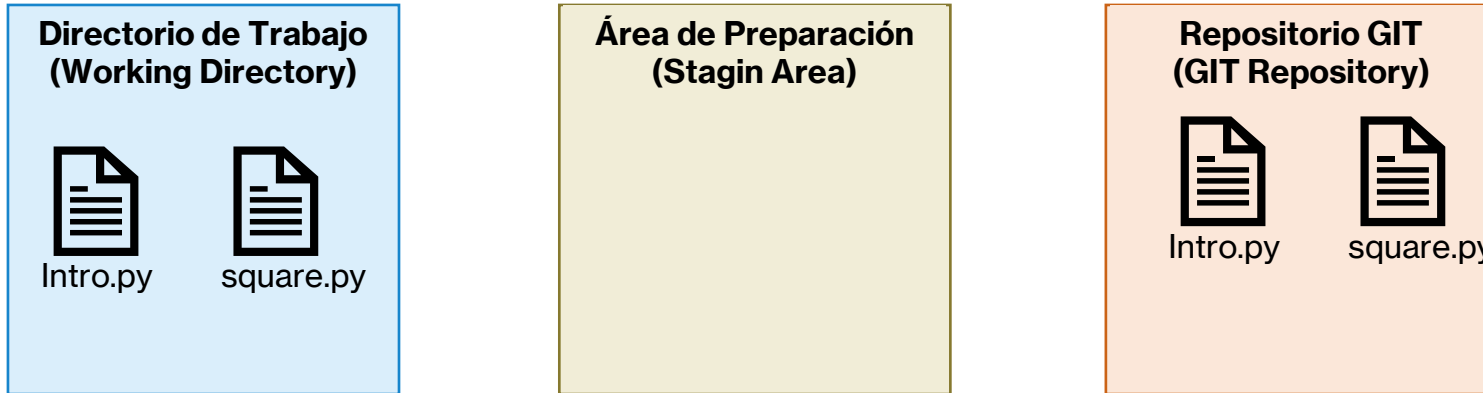
    inicializamos el proyecto

PS C:\Users\Sergio\My Drive\Clases\Programación 2\Git\Proyecto1>
```

Versión 2

Versión 1

head -> master o head -> main



HEAD: é uma notação que informa a posição atual no repositório e aponta para o último commit no branch atual.

```
PS C:\Users\Sergio\My Drive\Clases\Programación 2\Git\Proyecto1> git log
commit dd405318de779cbe8fb83188ee091576fa492292 (HEAD -> master)
Author: Sergio Andres Castaño Giraldo <[redacted]@outlook.com>
Date: Mon Aug 14 15:17:28 2023 -0300

    adicionamos archivo intro.py y modificamos square.py

commit 91f46f461c337ff6049e96c3c6ecffeeef305d2a
Author: Sergio Andres Castaño Giraldo <[redacted]@outlook.com>
Date: Tue Aug 8 13:11:55 2023 -0300

    inicializamos el proyecto
PS C:\Users\Sergio\My Drive\Clases\Programación 2\Git\Proyecto1>
```

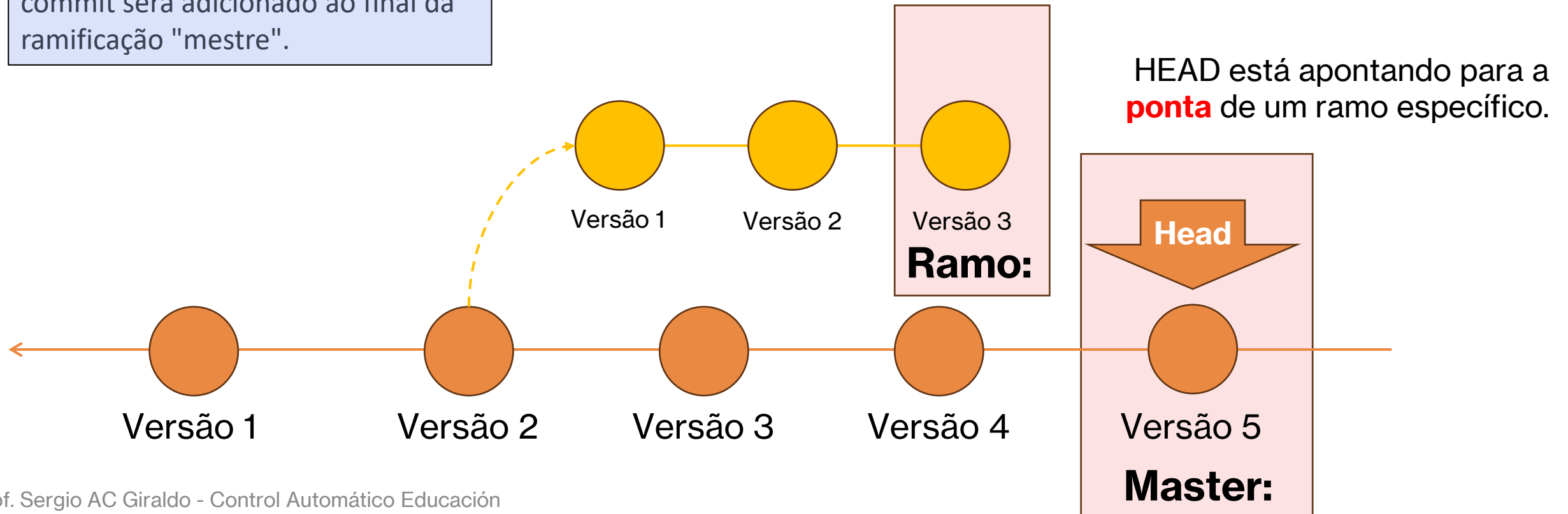
MASTER: Atualmente conhecido como main, indica o nome do branch principal onde está localizado o código de produção ou a versão mais estável do projeto.

Ramos



Se você fizer um novo commit enquanto estiver nesta posição, esse commit será adicionado ao final da ramificação "mestre".

BRANCH (Ramificação)

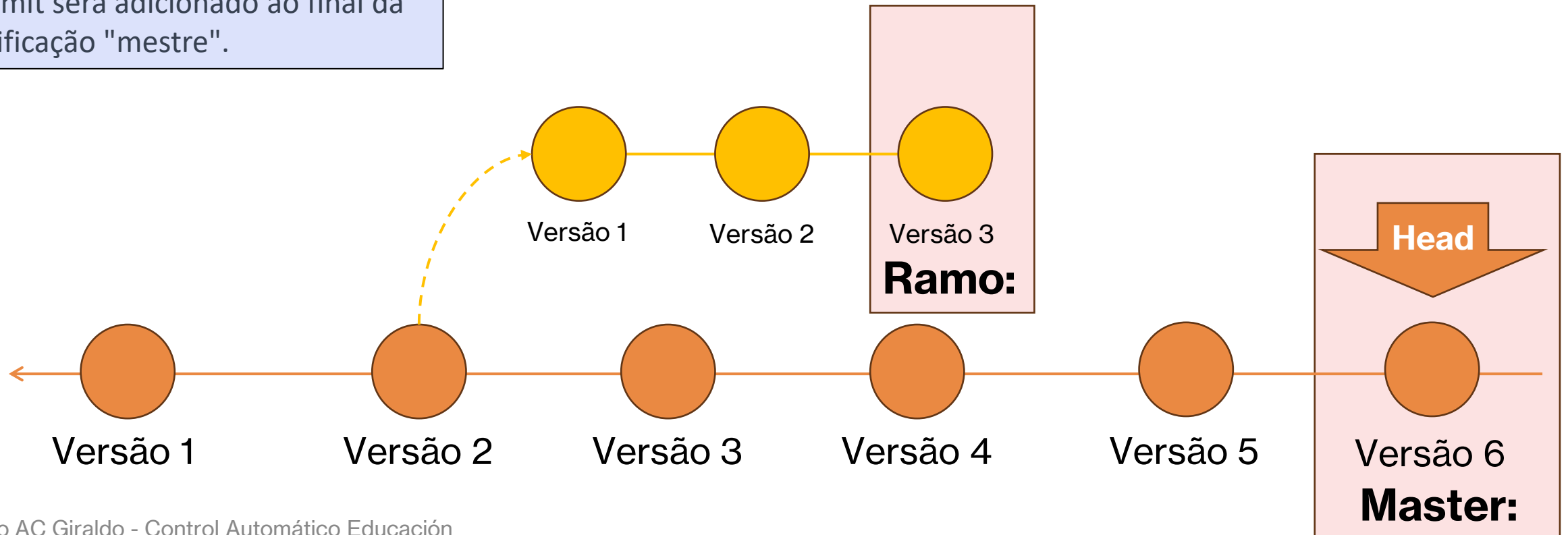


Ramos

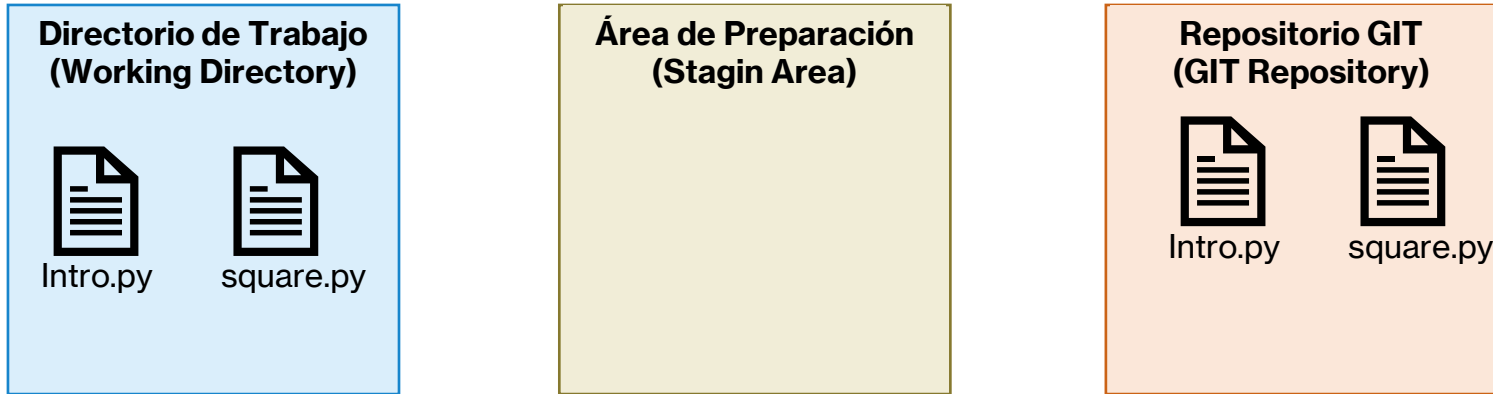


Se você fizer um novo commit enquanto estiver nesta posição, esse commit será adicionado ao final da ramificação "mestre".

BRANCH (Ramificação)



git checkout



- **git checkout** é um comando tradicionalmente usado no Git para alternar entre diferentes ramificações ou para restaurar arquivos da área de preparação ou de um commit específico.
- Ele permite que os desenvolvedores se movam por diferentes pontos (ou "instantâneos") do código no repositório.
- Visualmente podemos usar esta ferramenta para entender melhor: <https://git-school.github.io/visualizing-git/#free-remote>

git checkout - Restaurar um arquivo anterior

Directorio de Trabajo
(Working Directory)



Intro.py



square.py

Área de Preparación
(Stagin Area)

Repositorio GIT
(GIT Repository)



Intro.py

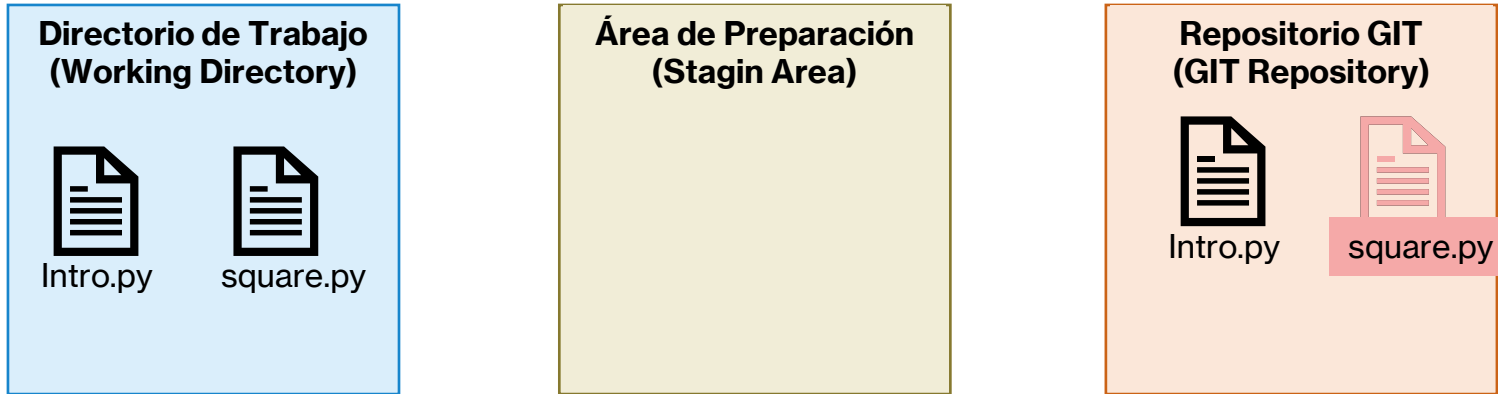


square.py

git checkout HASH <nombre_archivo>

```
PS C:\Users\Sergio\Documents\GitHubCourse\curso-git> git checkout a660bae7deb6ee18eefa051cd395ed7b393db7ae square.py
Updated 1 path from 2fca09a
```


git checkout - Restaurar um arquivo anterior



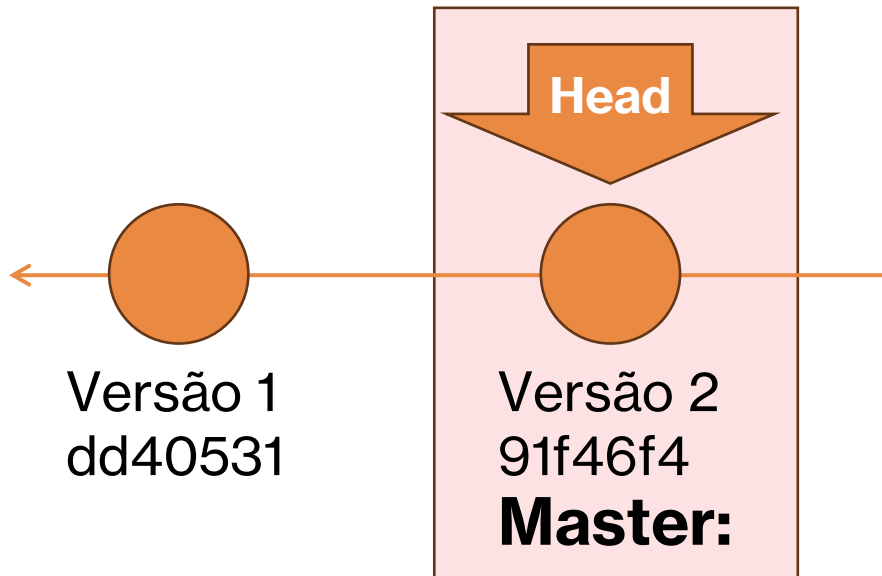
git checkout HASH <nombre_archivo>

```
PS C:\Users\Sergio\Documents\GitHubCourse\curso-git> git checkout a660bae7deb6ee18eefa051cd395ed7b393db7ae square.py
Updated 1 path from 2fca09a
```

git add .

git commit -m "earlier version"

Ramos do Projeto



git checkout - Restaurar un archivo ao seu estado no último commit

Directorio de Trabajo
(Working Directory)



Intro.py



square.py

Área de Preparación
(Stagin Area)

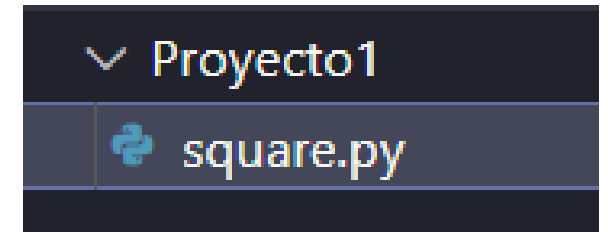
Repositorio GIT
(GIT Repository)



Intro.py



square.py



git checkout HASH commit anterior

```
PS C:\Users\Sergio\My Drive\Clases\Programación 2\Git\Proyecto1> git log
commit dd405318de779cbe8fb83188ee091576fa492292 (HEAD -> master)
Author: Sergio Andres Castaño Giraldo <s[redacted]@outlook.com>
Date: Mon Aug 14 15:17:28 2023 -0300
```

adicionamos archivo intro.py y modificamos square.py

```
commit 91f46f461c337ff6049e96c3c6ecffeeef305d2a
Author: Sergio Andres Castaño Giraldo <s[redacted]@outlook.com>
Date: Tue Aug 8 13:11:55 2023 -0300
```

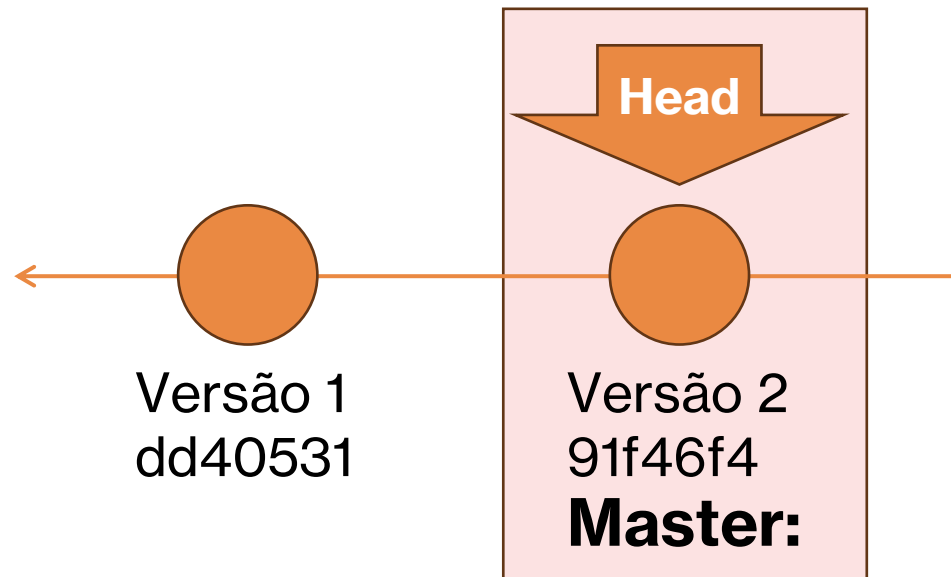
inicializamos el proyecto

```
PS C:\Users\Sergio\My Drive\Clases\Programación 2\Git\Proyecto1>
```

```
PS C:\Users\Sergio\My Drive\Clases\Programación 2\Git\Proyecto1> git checkout 91f46f461c337ff6049e96c3c6ecffeeef305d2a
```

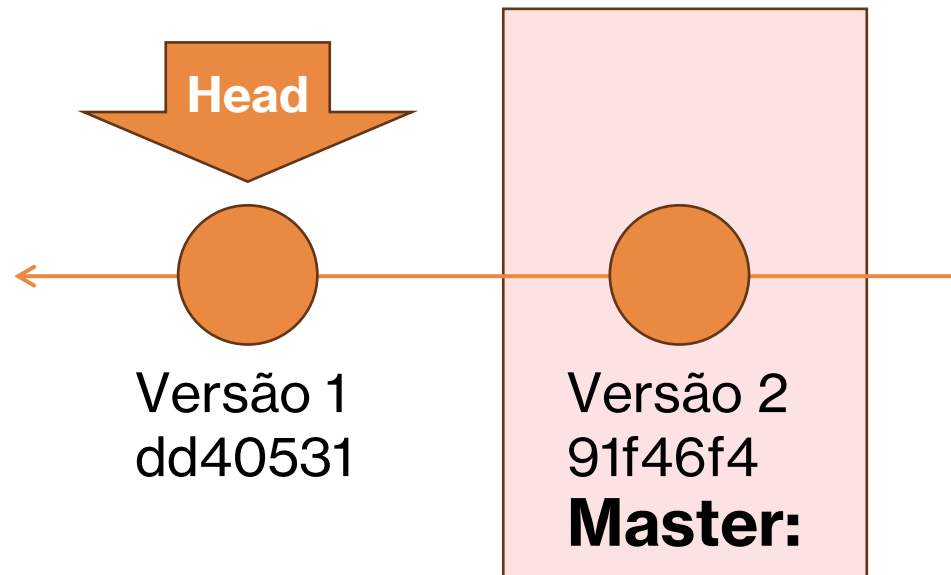
```
Note: switching to '91f46f461c337ff6049e96c3c6ecffeeef305d2a'.
```

Ramos do Projeto



HEAD em Modo "Detached" (Desprendido):
Nesse estado, HEAD está apontando diretamente para um commit e não para a ponta de um branch.

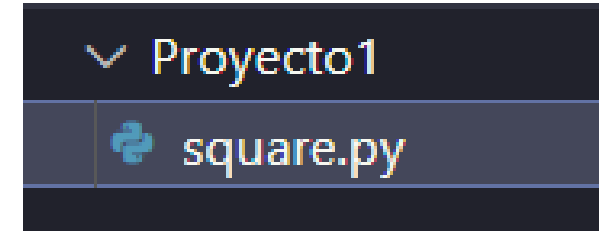
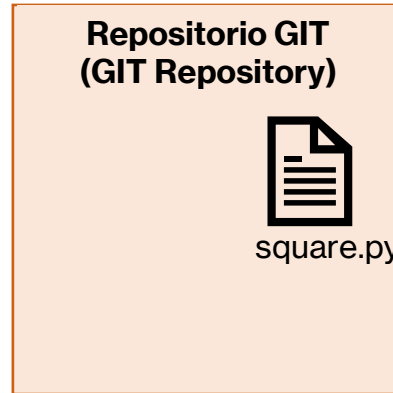
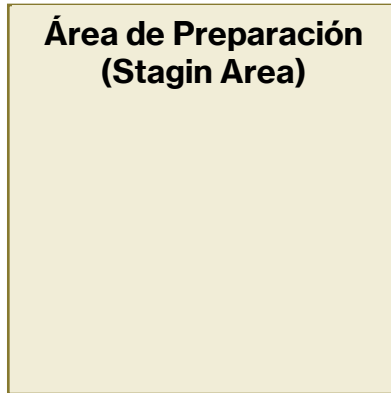
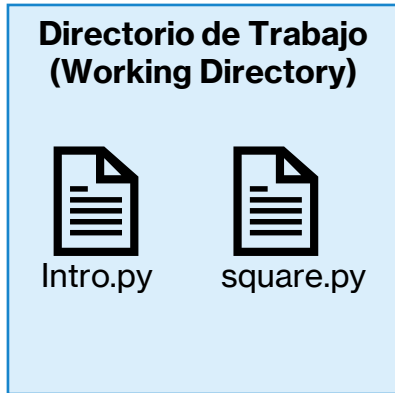
Ramos do Projeto



HEAD en Modo "Detached" (Desprendido):
Nesse estado, HEAD está apontando diretamente para um commit e não para a ponta de um branch.

Nesse modo, se você fizer alterações e criar um novo commit, **não haverá um ramo com esse commit**. Se você mudar para outra ramificação ou confirmar sem criar uma nova ramificação para essas alterações, poderá **perder esse trabalho**, pois não haverá referência apontando para ela.

git checkout - Restaurar um arquivo ao seu estado no último commit



git log --all



A opção **--all** modifica o comportamento do *git log* para mostrar o histórico de commits de todas as ramificações no repositório.

```
inicializamos el proyecto
PS C:\Users\Sergio\My Drive\Clases\Programación 2\Git\Proyecto1> git log --all
commit dd405318de779cbe8fb83188ee091576fa492292 (master)
Author: Sergio Andres Castaño Giraldo <sac_ing@outlook.com>
Date: Mon Aug 14 15:17:28 2023 -0300

    adicionamos archivo intro.py y modificamos square.py

commit 91f46f461c337ff6049e96c3c6ecffeeef305d2a (HEAD)
Author: Sergio Andres Castaño Giraldo <sac_ing@outlook.com>
Date: Tue Aug 8 13:11:55 2023 -0300

    inicializamos el proyecto
```

Modificar archivo



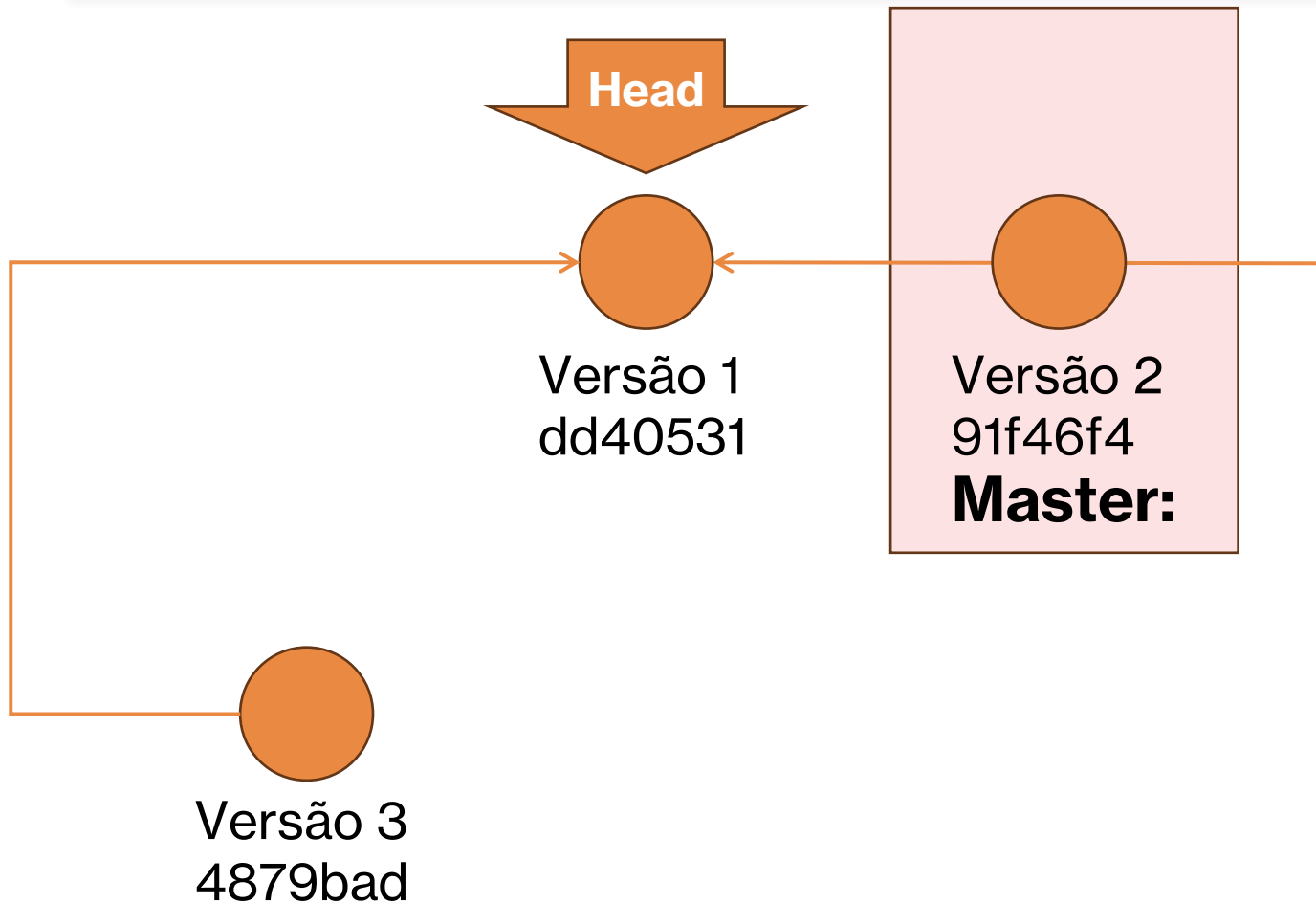
square.py

```
for i in range(1, 11):  
    print(f"The square of {i} is {i*i}")  
print('end')
```

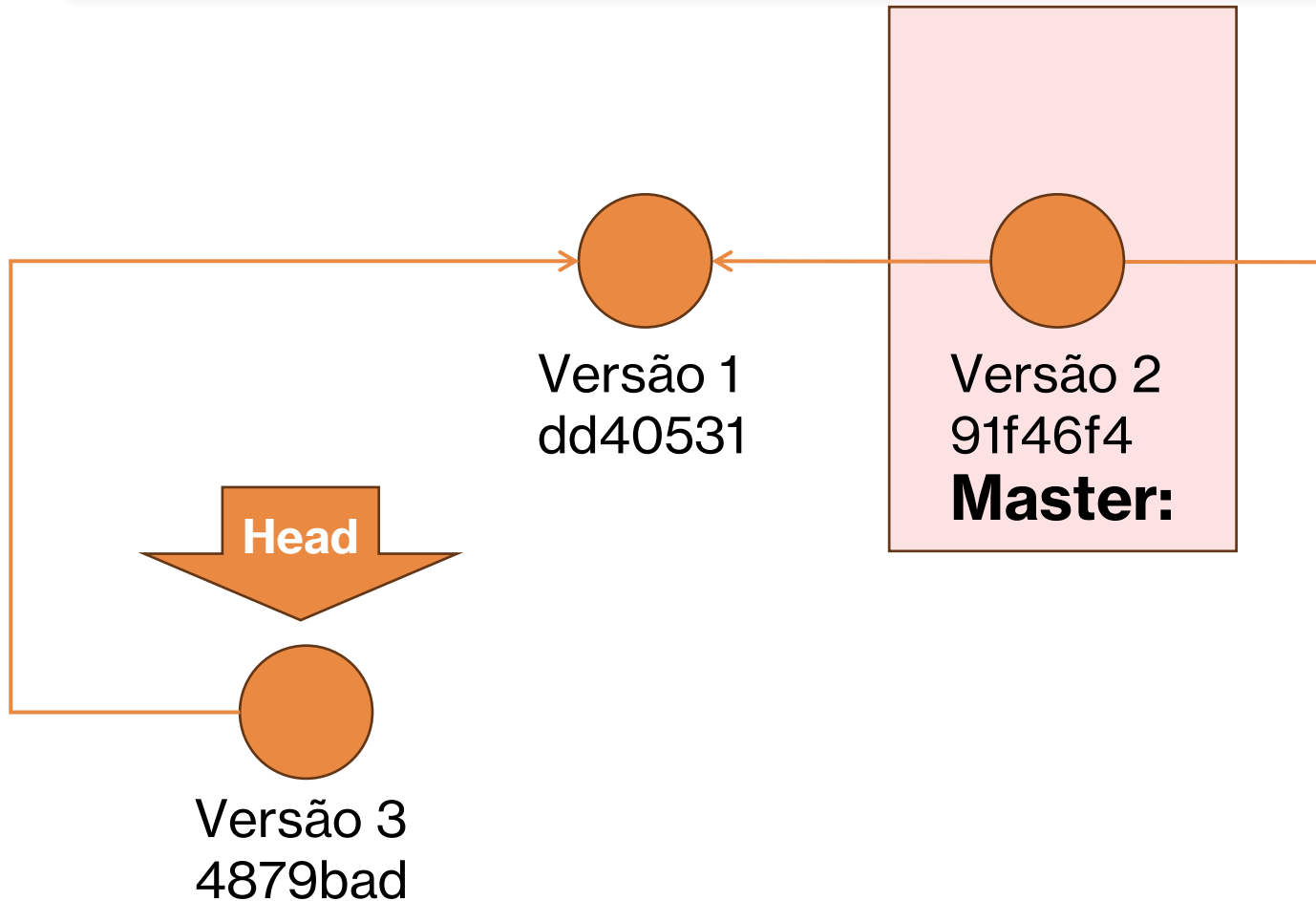
Agregar print

1. git add .
2. git commit "testando commit em head desanexado"

Ramos do Projeto



Ramos do Projeto



git checkout - Restaurar um arquivo de versão de produção

Directorio de Trabajo
(Working Directory)



Intro.py



square.py

Área de Preparación
(Stagin Area)

Repositorio GIT
(GIT Repository)



Intro.py



square.py

git checkout master

```
PS C:\Users\Sergio\My Drive\Clases\Programación 2\Git\Proyecto1> git checkout master
Warning: you are leaving 1 commit behind, not connected to
any of your branches:
```

```
4879bad probando commit en head detached
```

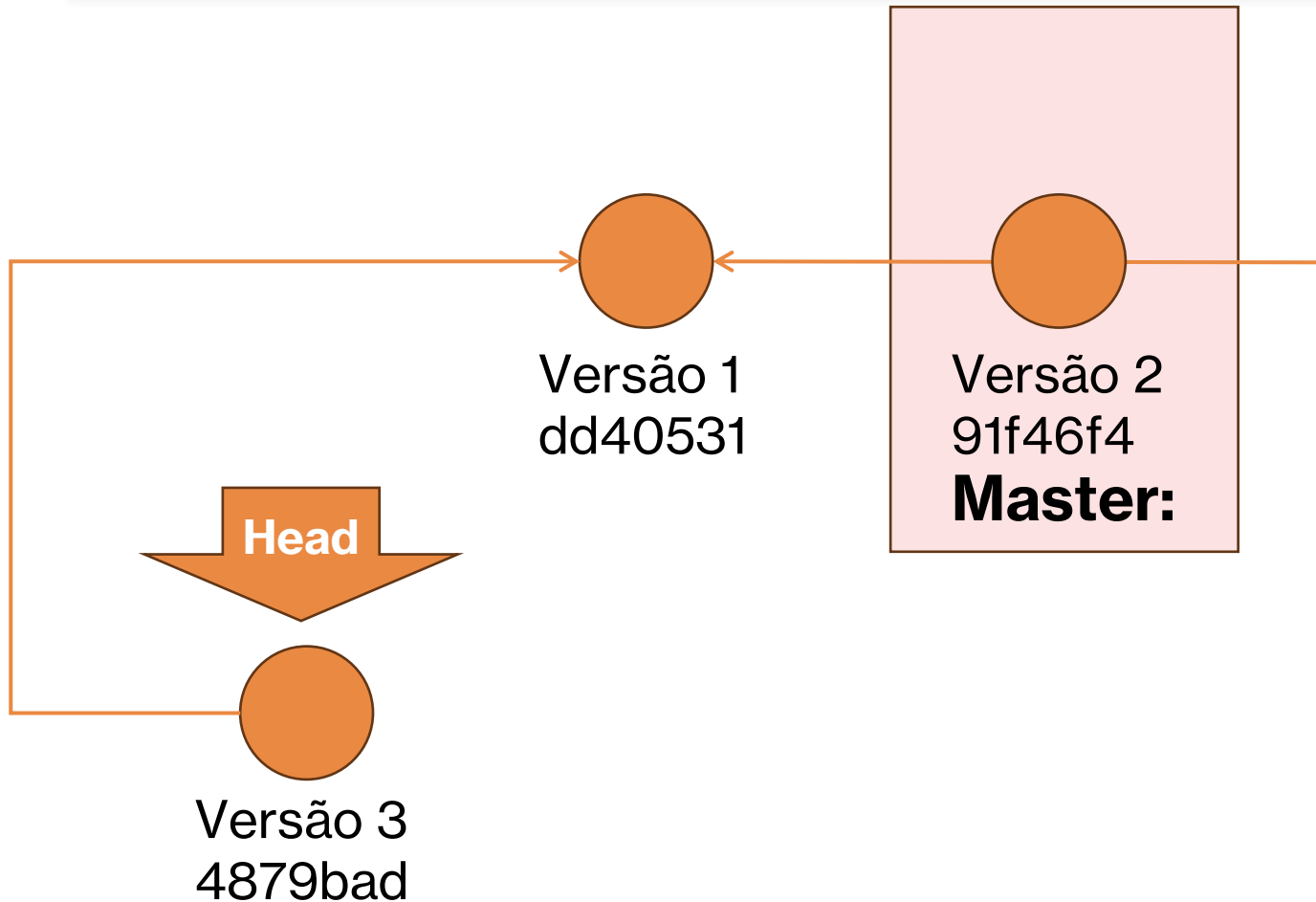
```
If you want to keep it by creating a new branch, this may be a good time
to do so with:
```

```
git branch <new-branch-name> 4879bad
```

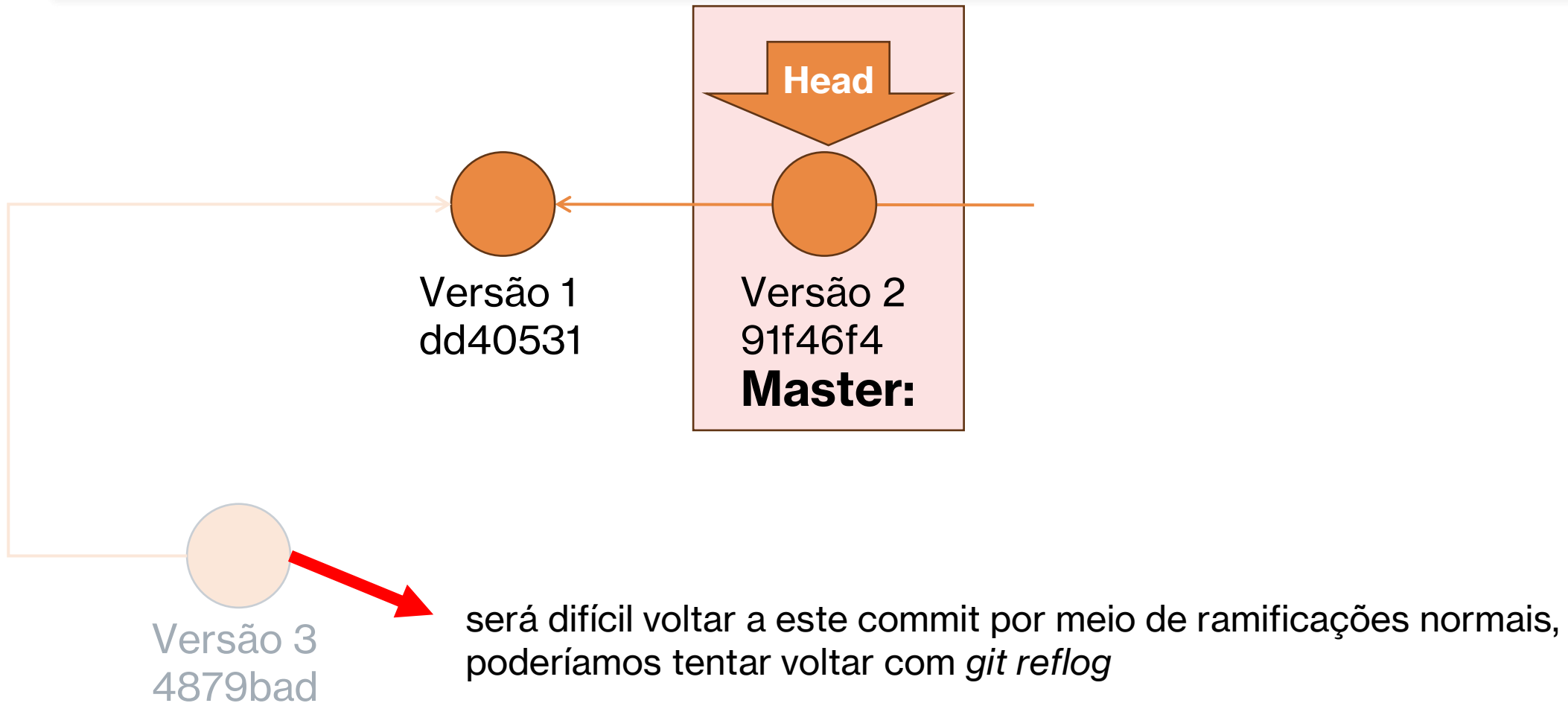
```
Switched to branch 'master'
```

```
PS C:\Users\Sergio\My Drive\Clases\Programación 2\Git\Proyecto1> █
```

Ramos do Projeto



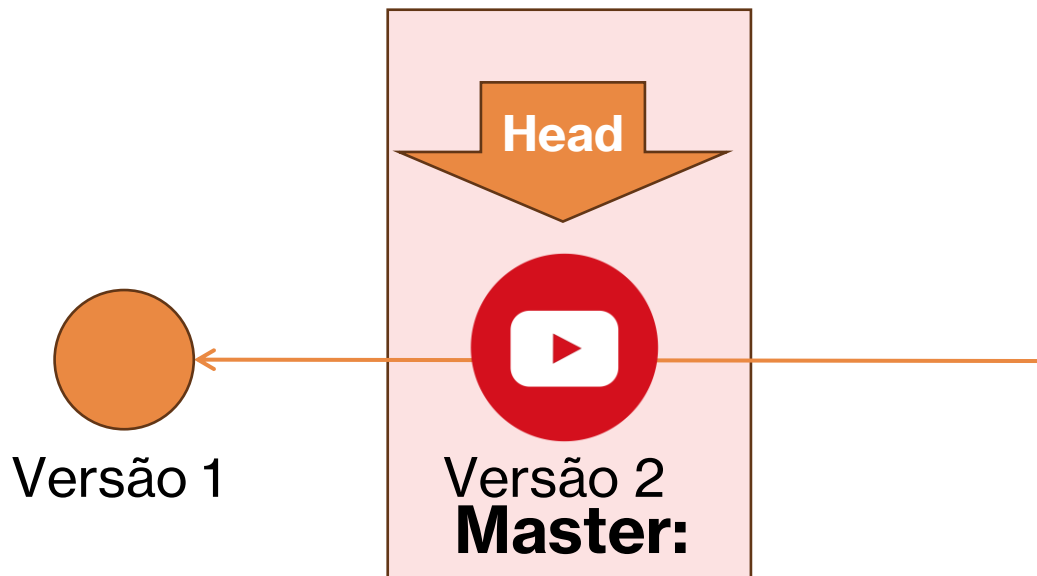
Ramos do Projeto



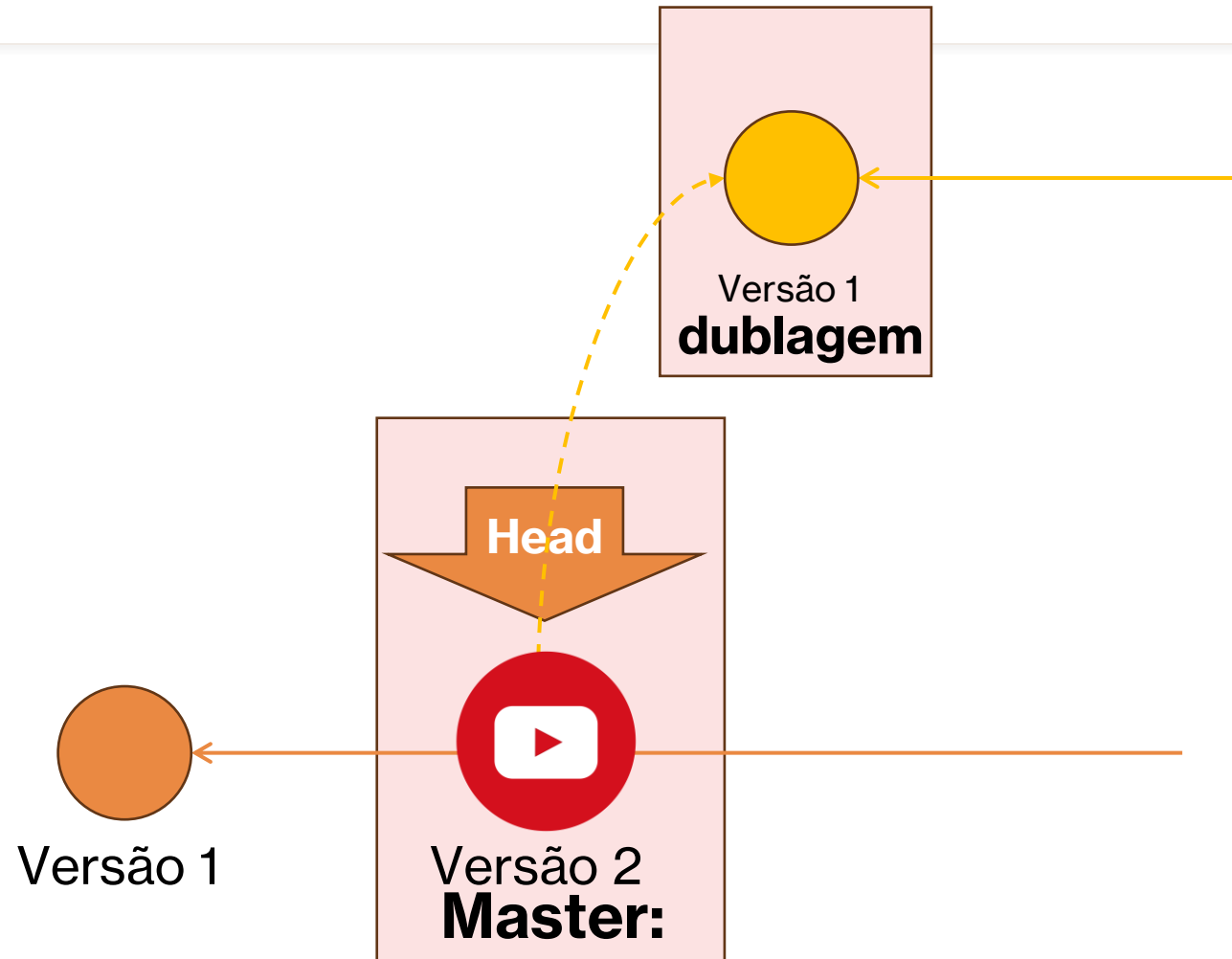
Branch - Ramos

- As ramificações no Git permitem que você se desvie do fluxo principal de desenvolvimento e continue trabalhando sem perturbar esse fluxo principal.
- **Por que usar ramificações?**
- Para desenvolver recursos, corrija bugs ou experimente sem afetar a base de código principal.
- Comando Básico
- *git branch* para ver todos os ramos.
- *git checkout nome_novo_ramo* para mudar para um ramo.
- *git checkout -b nome_novo_ramo* para criar e mudar para uma nova ramificação.

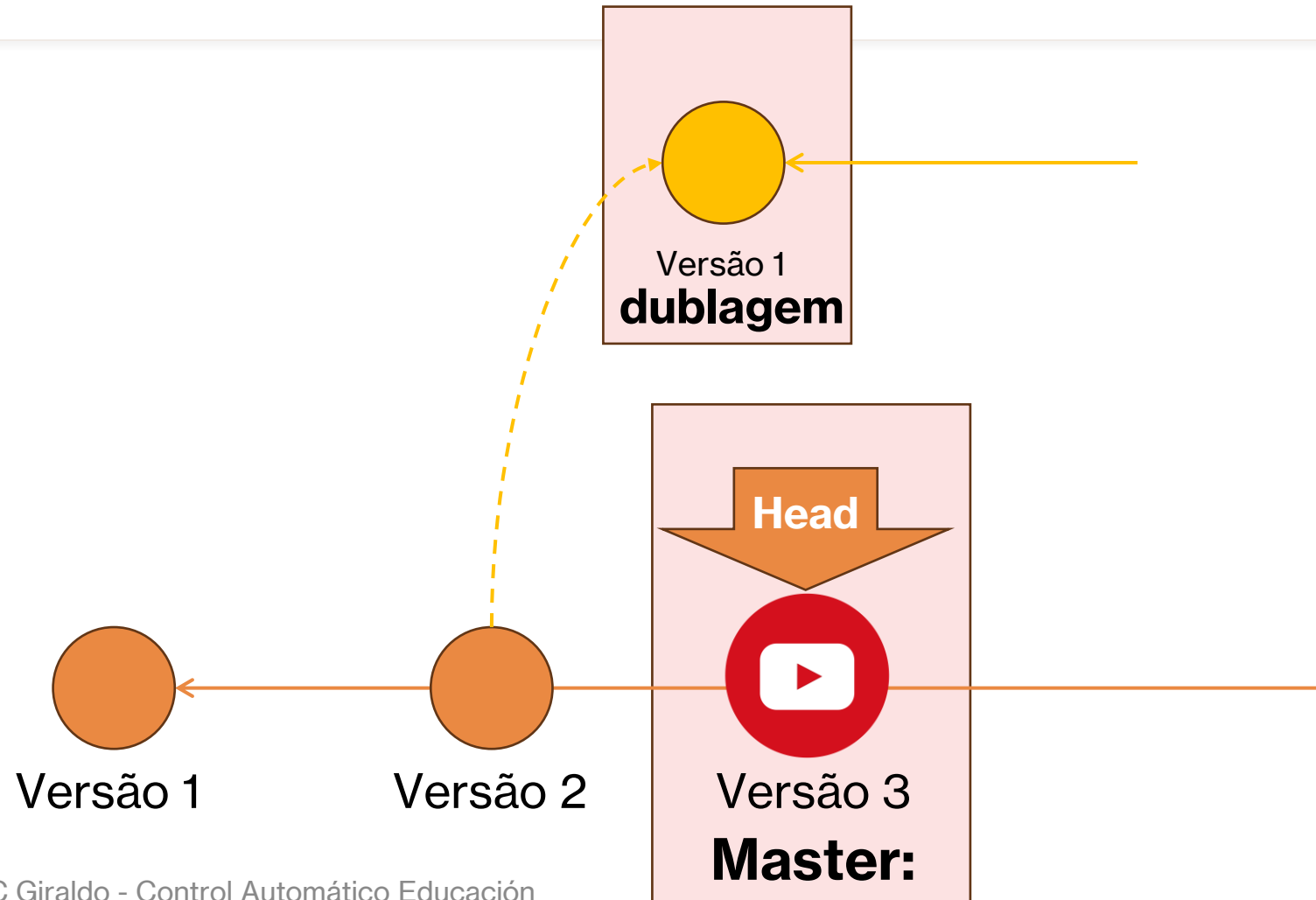
Ramos - Exemplo



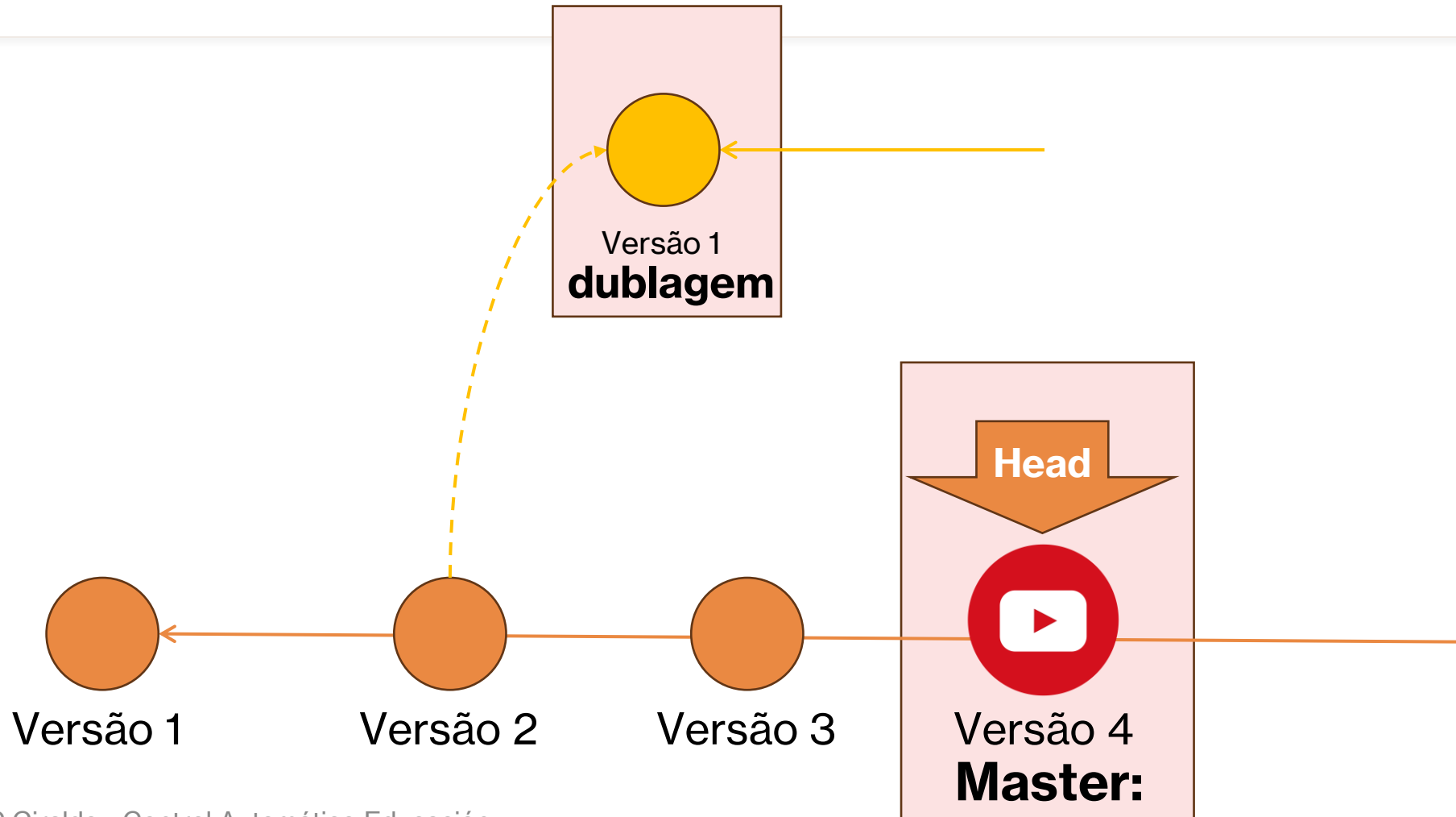
Ramos - Exemplo



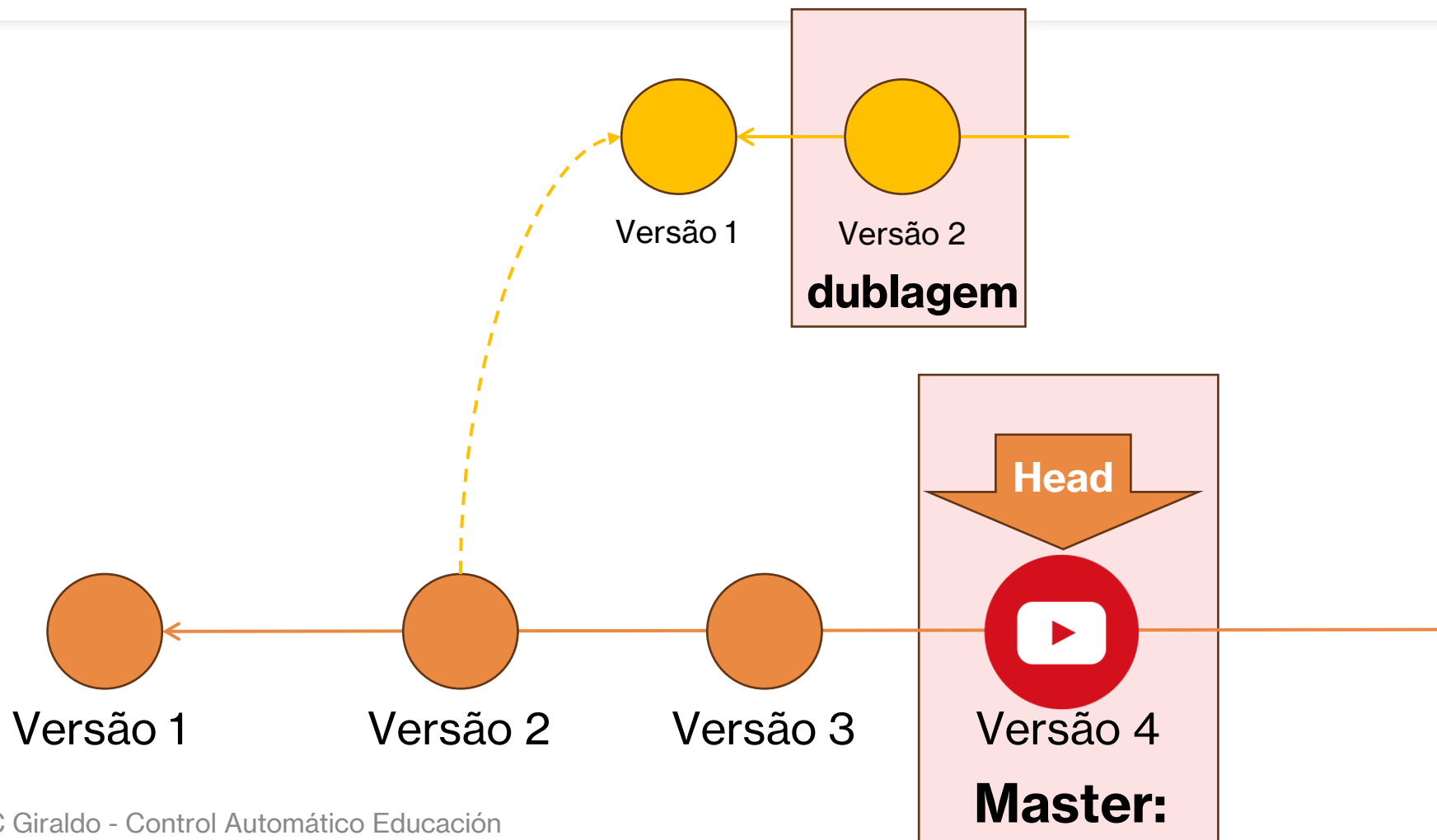
Ramos - Exemplo



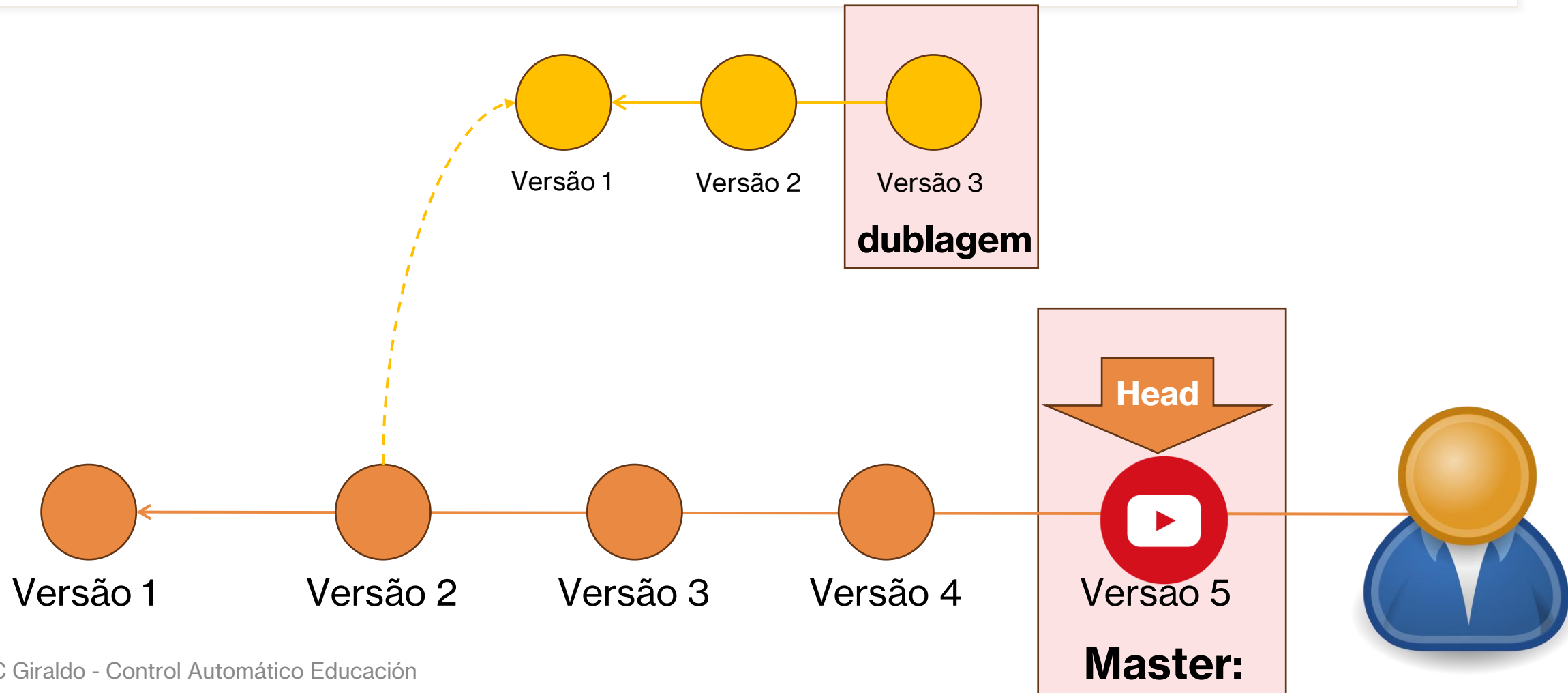
Ramos - Exemplo



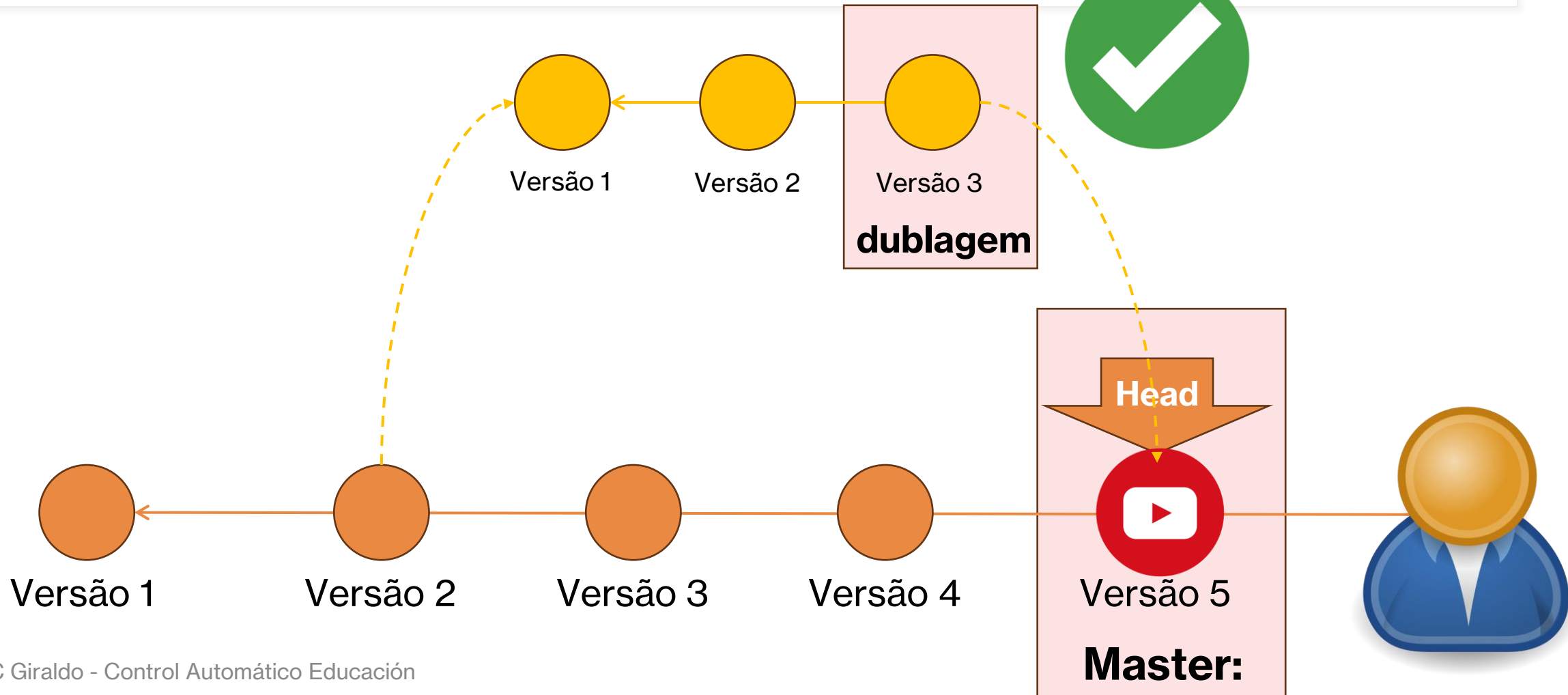
Ramos - Exemplo



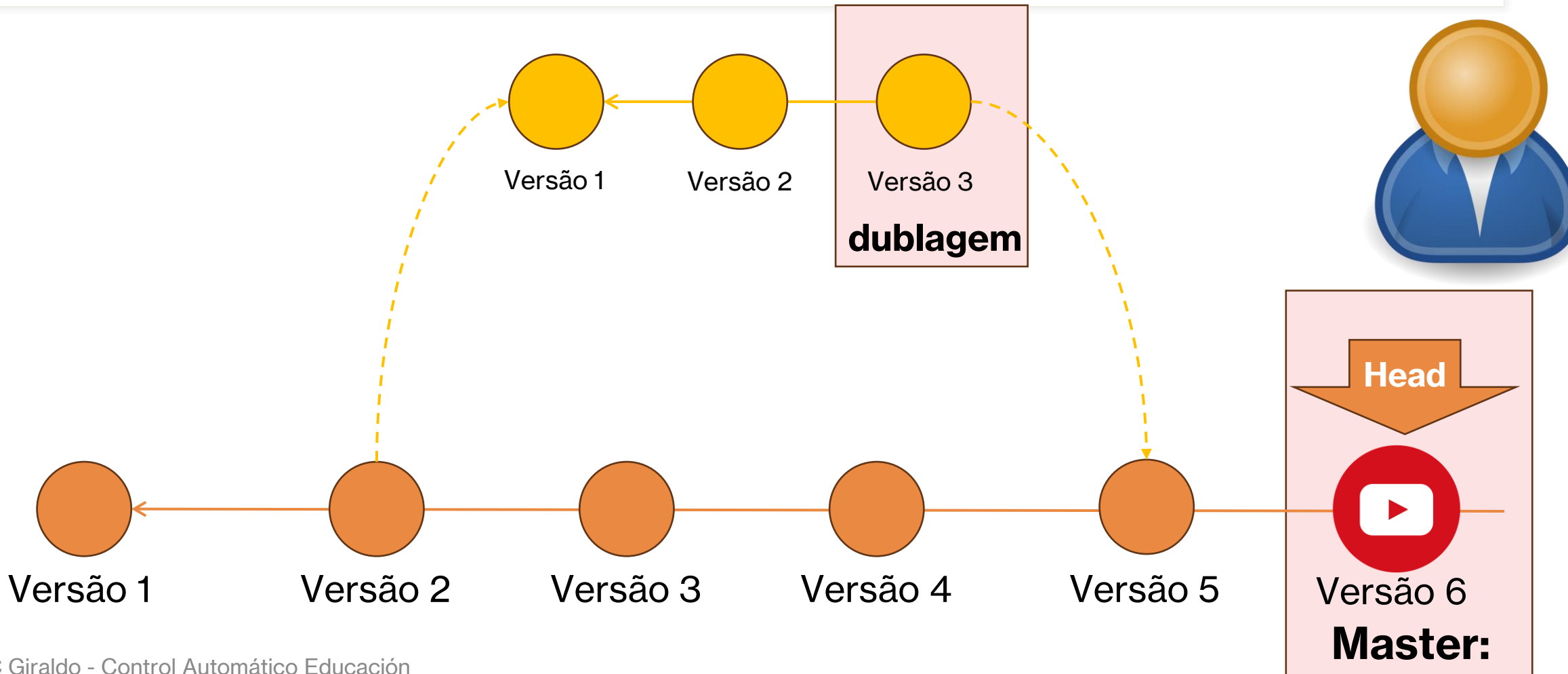
Ramos - Exemplo



Ramos - Exemplo



Ramos - Exemplo

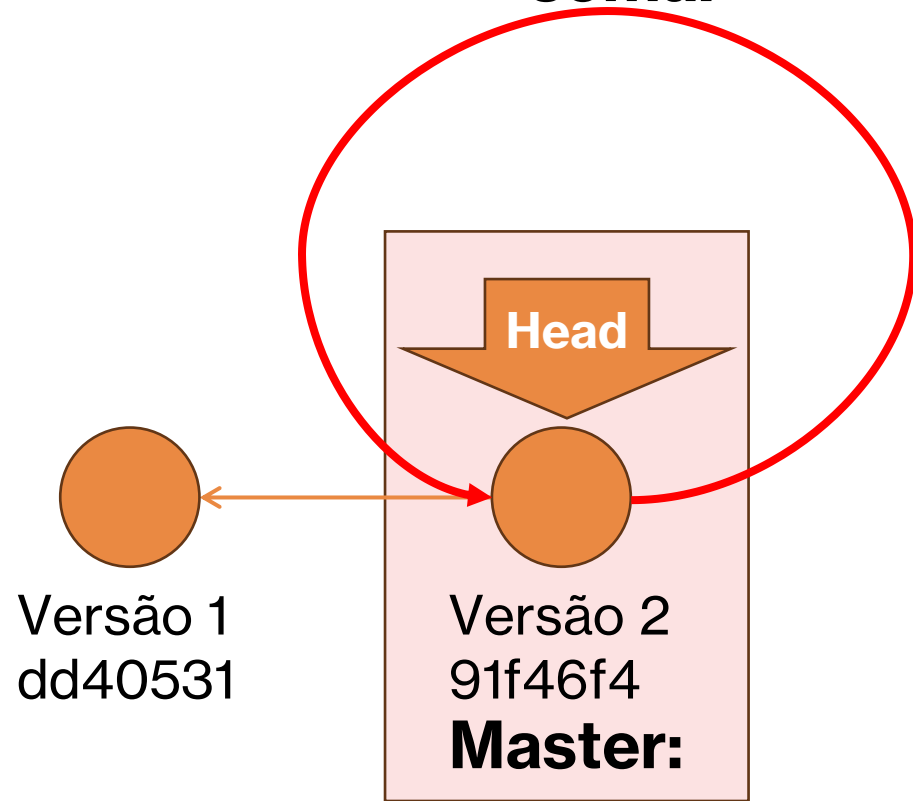


Criar NUEVA Branch

```
git checkout -b soma
```

branch nome

soma:



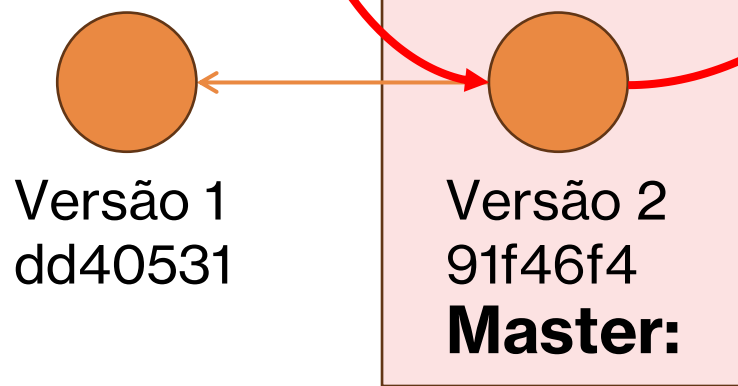
Crear NUEVA Branch

```
git checkout -b soma
```

branch nome

Head

soma:



```
(HEAD -> soma, master)
```

lista de todos os branches locais em seu repositório. A ramificação em que você está atualmente será destacada e precedida por um asterisco (*)

```
git branch
```

master
* soma

Modificar archivo

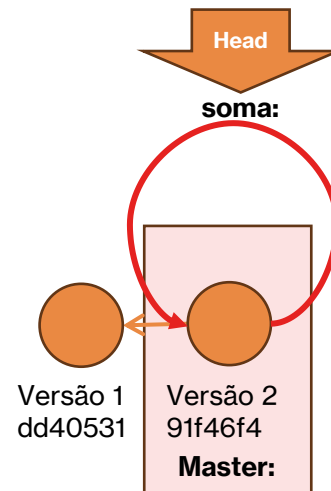


square.py

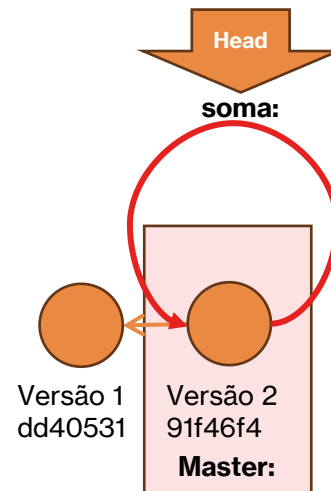
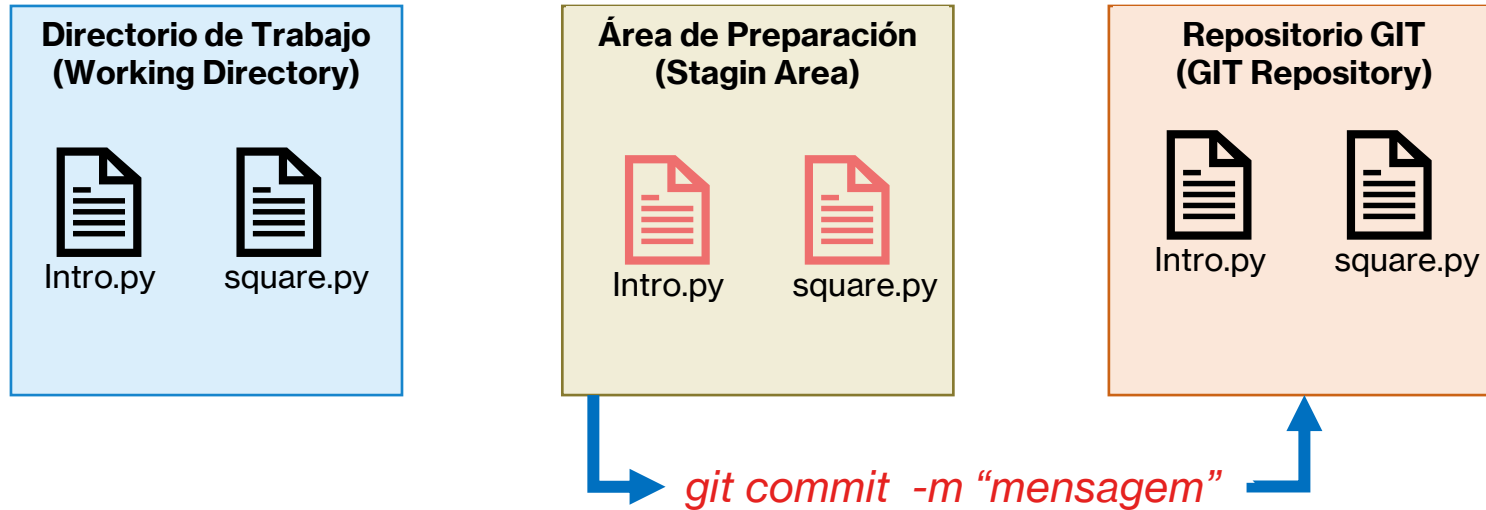
```
suma = 0
for i in range(1, 15):
    suma += i
    print(f"The square of {i} is {i*i}")

print(f'The sum of the indices is {suma}')
```

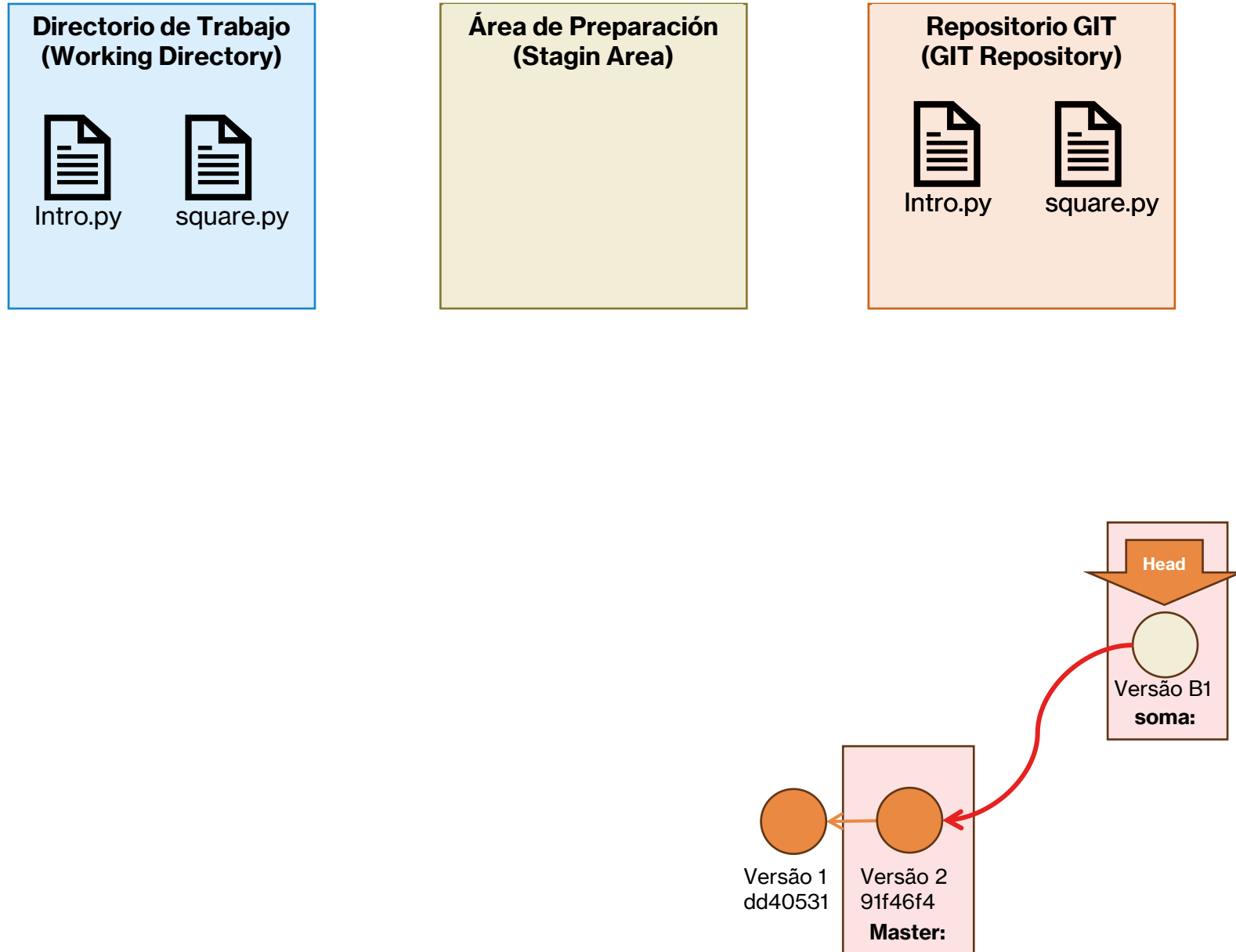

Operando no ramo



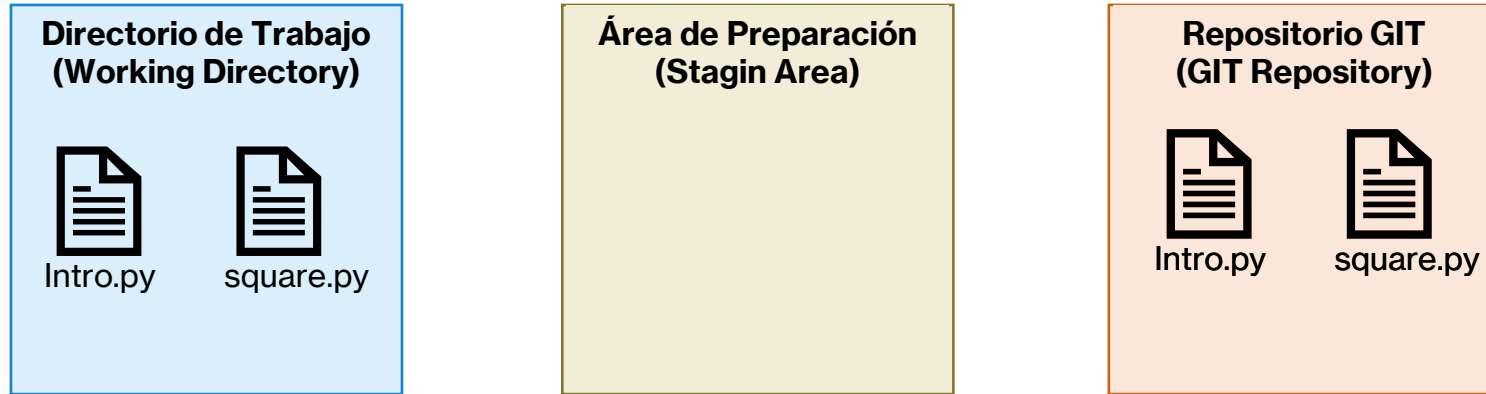
Operando no ramo



Cambiando de Ramas

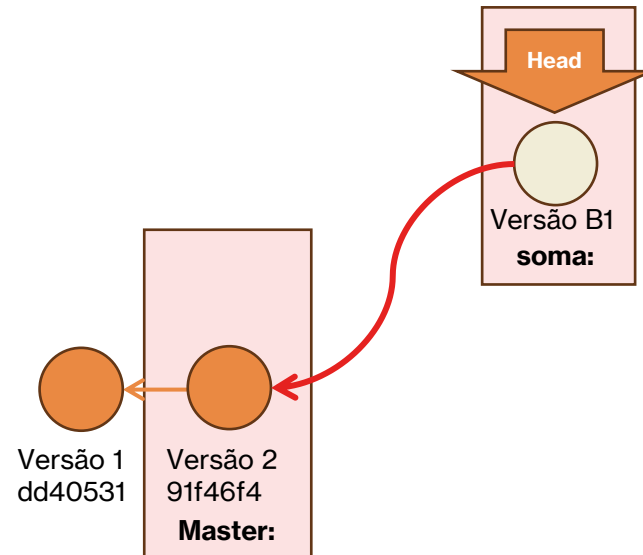


Cambiando de Ramas



git checkout master

git checkout soma



git merge - Mesclando Ramos

O comando **git merge** mescla alterações de um ramo para outro.

```
git checkout master # Você muda para o ramo principal.  
git merge soma      # Você combina as mudanças de soma no mestre.
```

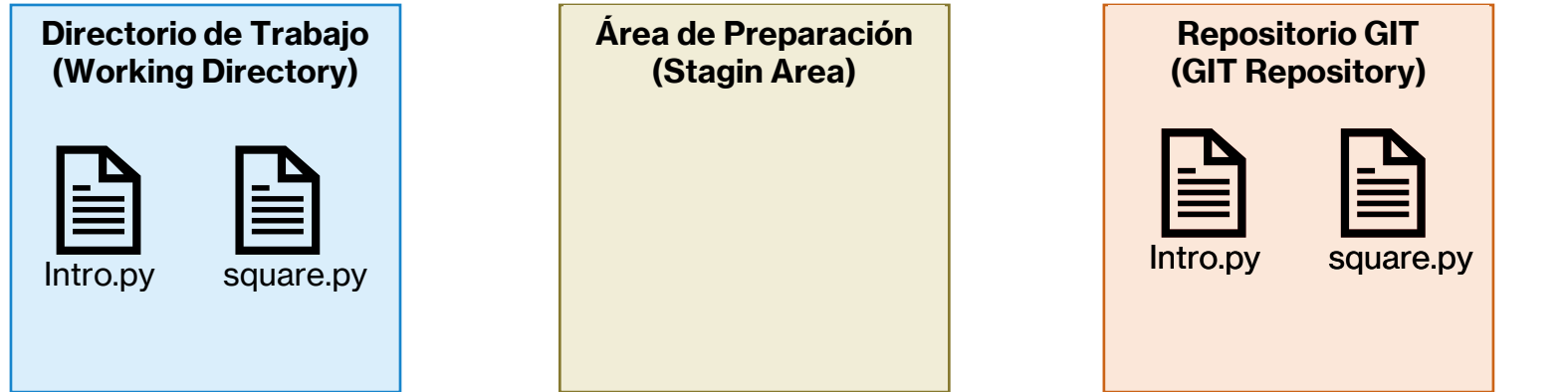
Tipos de Fusões :

- **Fast-forward Merge:** Se não houver mudanças divergentes, o Git simplesmente move o ponteiro para frente.
- **Fusão com Commit:** Quando ambas as ramificações tiverem alterações, o Git criará um merge commit.

Conflitos das Fusões :

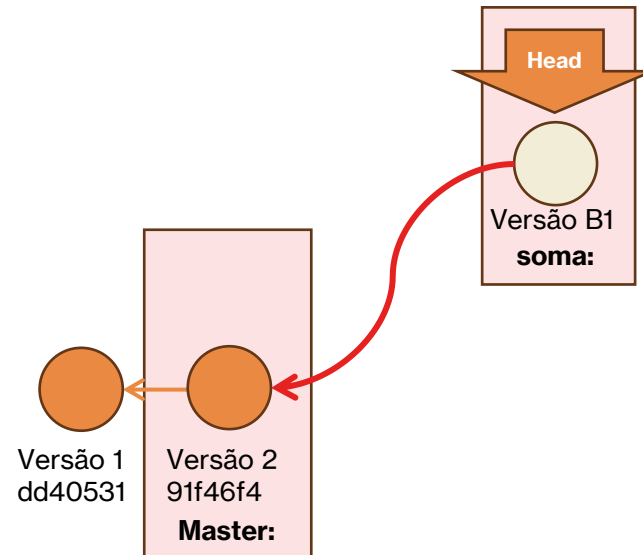
- Ocasionalmente, o git merge pode encontrar conflitos.
- Eles devem ser resolvidos manualmente.
- Marcadores como <<<<<<, ===== e >>>>>> indicam conflitos.

git merge - Mesclando Ramos



git checkout master

git merge soma



git merge - Mesclando Ramos

Directorio de Trabajo
(Working Directory)



Intro.py



square.py

Área de Preparación
(Stagin Area)

Repositorio GIT
(GIT Repository)



Intro.py



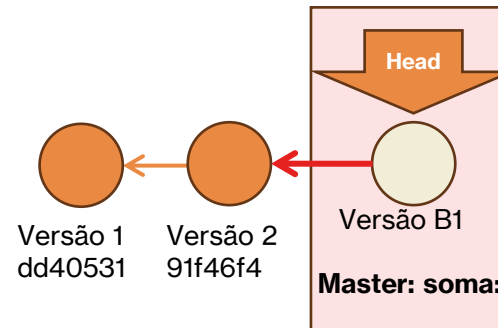
square.py

```
git checkout master
```

```
git merge soma
```

```
PS H:\My Drive\Clases\Programación 2\Git\Proyecto1> git merge soma
Updating dd40531..890d9ff
Fast-forward
 square.py | 7 ++++++-
 1 file changed, 6 insertions(+), 1 deletion(-)
```

Fusão Rápida (Fast-forward)



git merge - Mesclando Ramos

```
git checkout -b greet
```

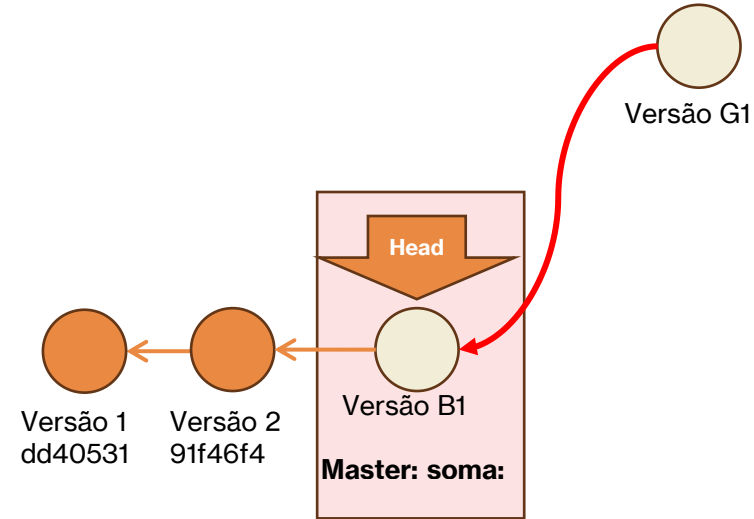


Intro.py

```
print('Hello World!')  
print('Ramo Greet')
```

```
git add .
```

```
git commit -m "ramo add"
```



git merge - Mesclando Ramos

```
git checkout -b greet
```

Intro.py

```
print('Hello World!')  
print('Rama Greet')
```

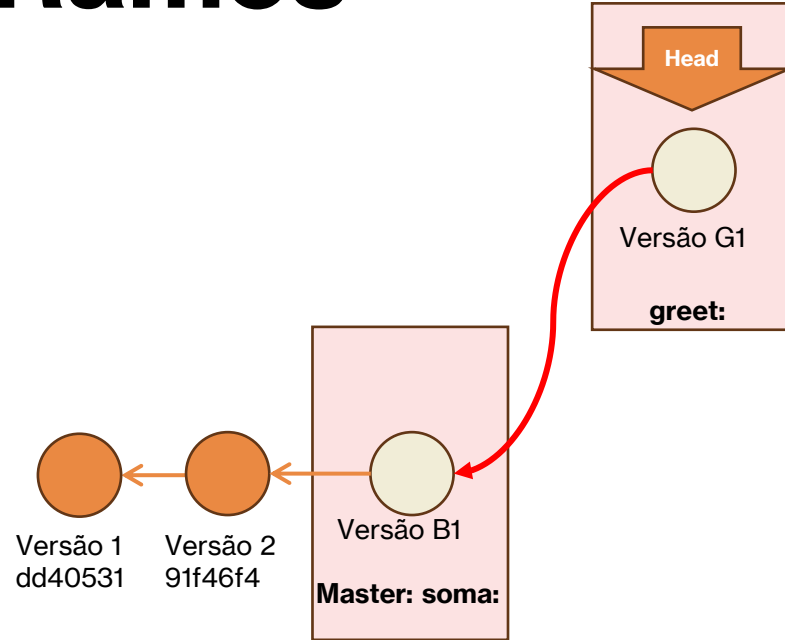
```
git add .  
git commit -m "rama add"
```

```
git checkout master
```

Intro.py

```
print('Hello World!')  
print('Rama Master')
```

```
git add .  
git commit -m "rama master"
```



git merge - Mesclando Ramos

git checkout -b greet



Intro.py

```
print('Hello World!')  
print('Ramo Greet')
```

git add .
git commit -m "ramo add"

git checkout master



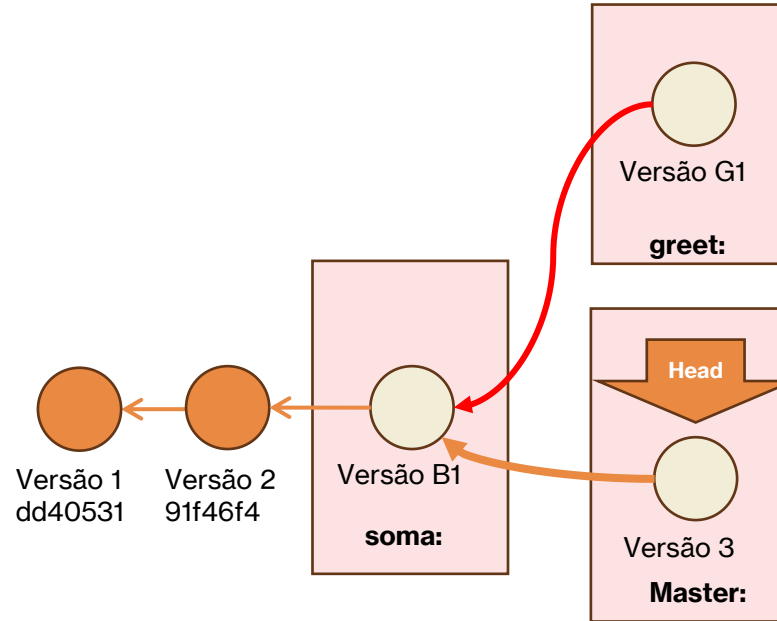
Intro.py

```
print('Hello World!')  
print('Ramo Master')
```

git add .
git commit -m "ramo master"

git merge greet

git add .
git commit

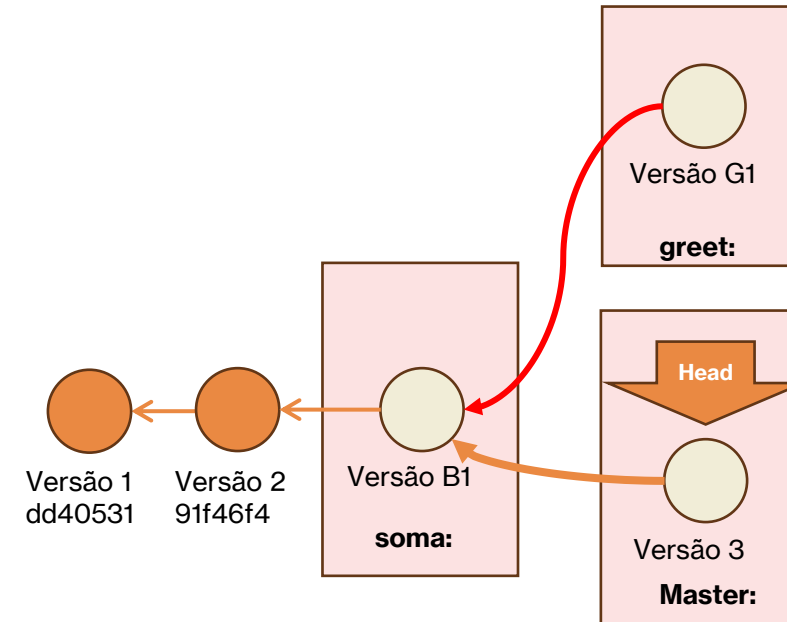


```
1  print('Hello World!')
   Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
2  <<<<<<< HEAD (Current Change)
   print('Ramo Master')
3  =====
4  print('Ramo Greet')
5  >>>>>> greet (Incoming Change)
6
7
```

Resolve in Merge Editor

git merge - Mesclando Ramos

Automatic merge failed; fix conflicts and then commit the result.



```
1 print('Hello World!')
2 <<<<<<< HEAD (Current Change)
3 print('Rama Master')
4 =====
5 print('Rama Greet')
6 >>>>>> greet (Incoming Change)
7
```

Resolve in Merge Editor

git merge - Mesclando Ramos

```
git checkout -b greet
```



Intro.py

```
print('Hello World!')  
print('Ramo Greet')
```

```
git add .  
git commit -m "ramo add"
```

```
git checkout master
```



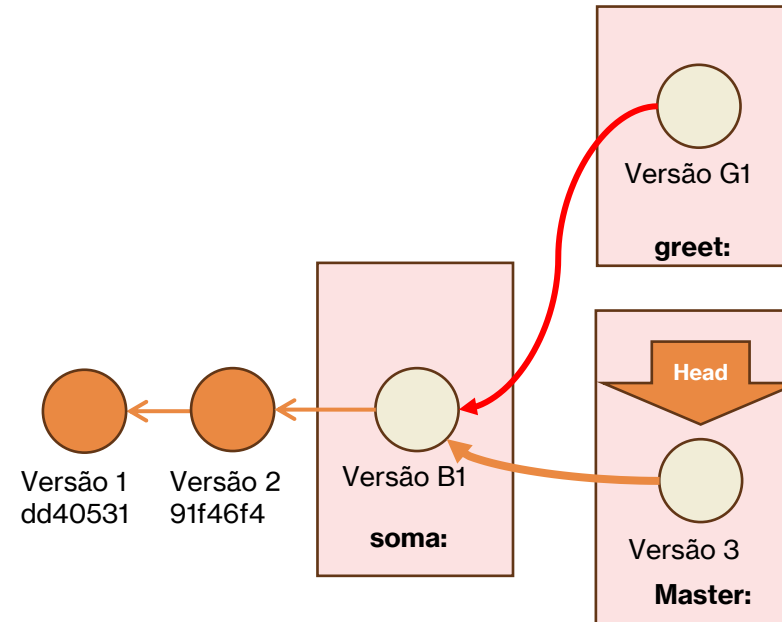
Intro.py

```
print('Hello World!')  
print('Ramo Master')
```

```
git add .  
git commit -m "ramo master"
```

```
git merge greet
```

```
git add .  
git commit
```



("w" es para escrever/salvar e "q" es para sair)

```
Merge branch 'greet'
```

```
# Conflicts:  
#   intro.py  
#  
# It looks like you may be committing a merge.  
# If this is not correct, please run  
#   git update-ref -d MERGE_HEAD  
# and try again.
```

```
# Please enter the commit message for your changes. Lines starting  
# with '#' will be ignored, and an empty message aborts the commit.  
#  
# On branch master  
.git/COMMIT_EDITMSG[+] [unix] (15:25 15/08/2023)  
:
```

Vim

Shift + : Digite
wq e aperte Enter

Nano

Ctrl + X
aperte **Y** e aperte
Enter

git merge - Mesclando Ramos

```
git checkout -b greet
```



Intro.py

```
print('Hello World!')  
print('Ramo Greet')
```

```
git add .  
git commit -m "ramo add"
```

```
git checkout master
```

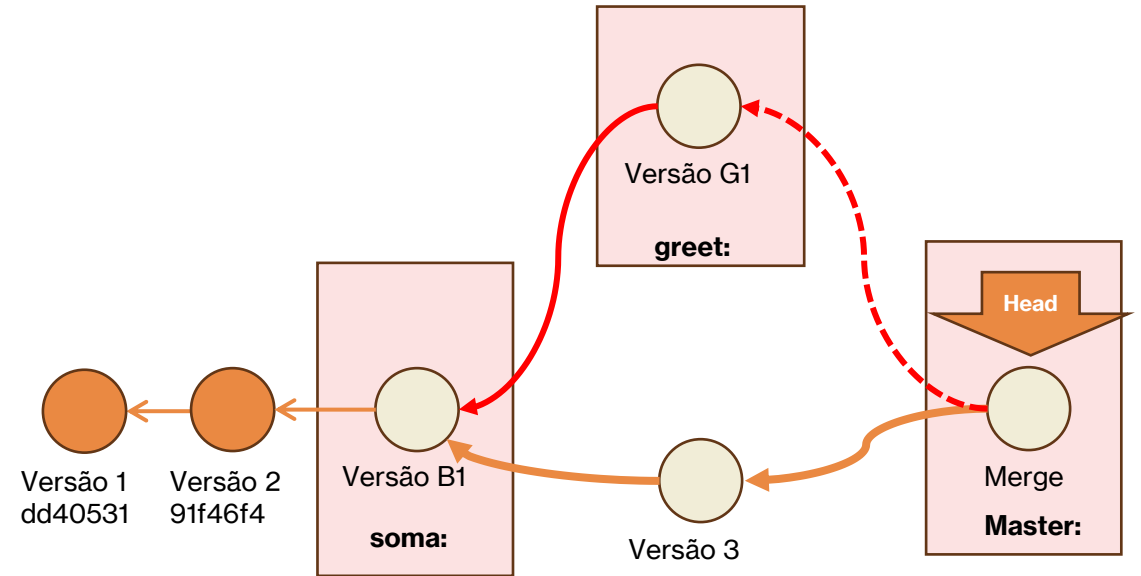


Intro.py

```
print('Hello World!')  
print('Ramo Master')
```

```
git add .  
git commit -m "ramo master"
```

```
git merge greet
```



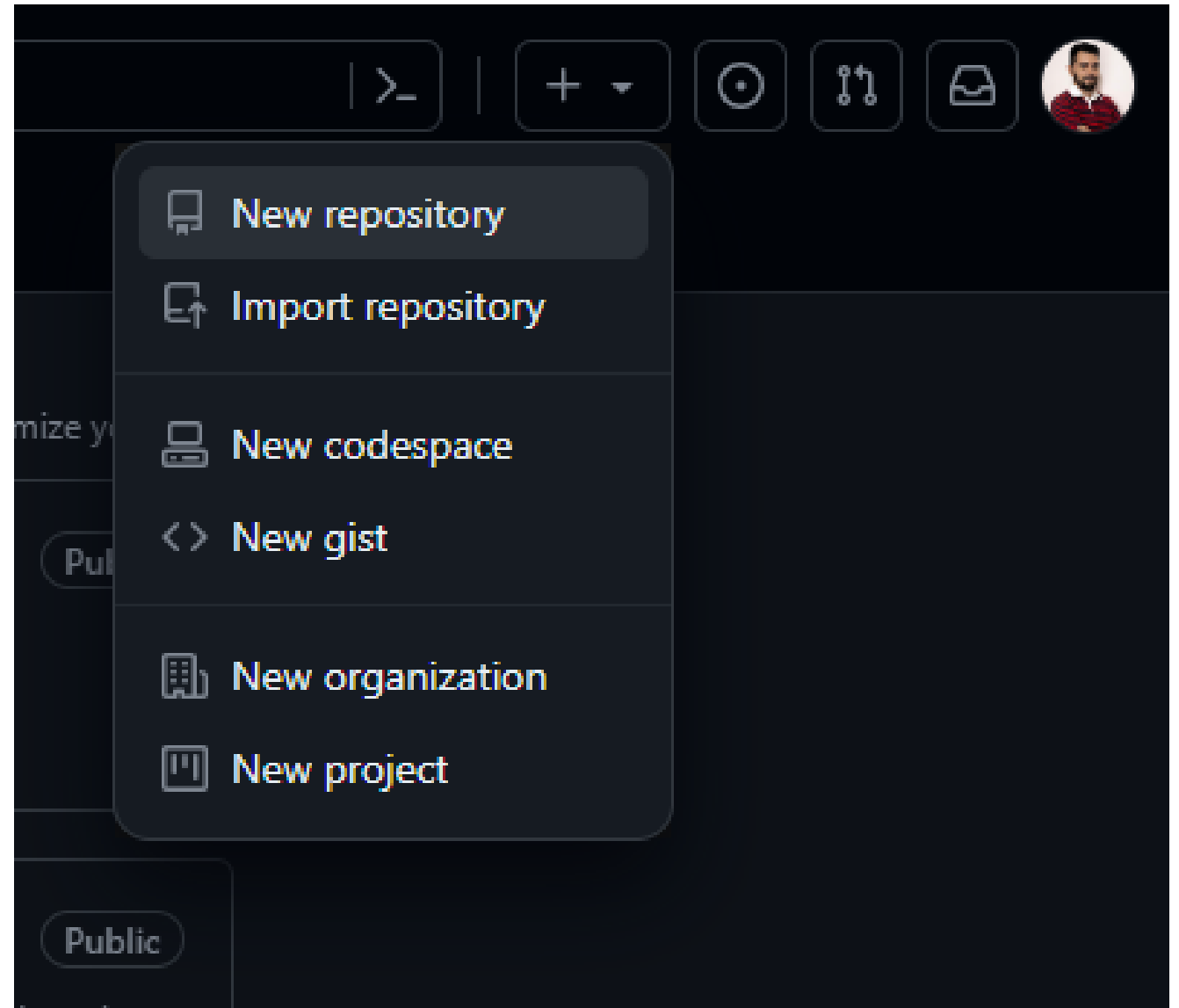
Fusão com Commit: quando ambas as ramificações têm alterações, o Git cria uma confirmação de mesclagem.



Colaboração do projeto com o GitHub

GitHub

Como primeiro passo vamos entrar no GitHub e vamos criar um novo repositório.



Novo Repositorio


Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?

[Import a repository.](#)

Required fields are marked with an asterisk ().*

Owner *

 sergioacg ▾

Repository name *



/ curso-git

✔ curso-git is available.

Great repository names are short and memorable. Need inspiration? How about **turbo-giggle** ?

Description (optional)

Git and GitHub Course

- ☐  **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:

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

Add .gitignore


.gitignore template: **None** ▾


Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

License: **None** ▾

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

This will set  **master** as the default branch. Change the default name in your [settings](#).

 You are creating a private repository in your personal account.

Create repository

Conectando nosso repositório local com o GitHub

Vincule nosso repositório local a um repositório no GitHub para que possamos compartilhar nosso projeto com outras pessoas e colaborar online.

```
git remote add origin URL_DO_REPOSITORIO
```

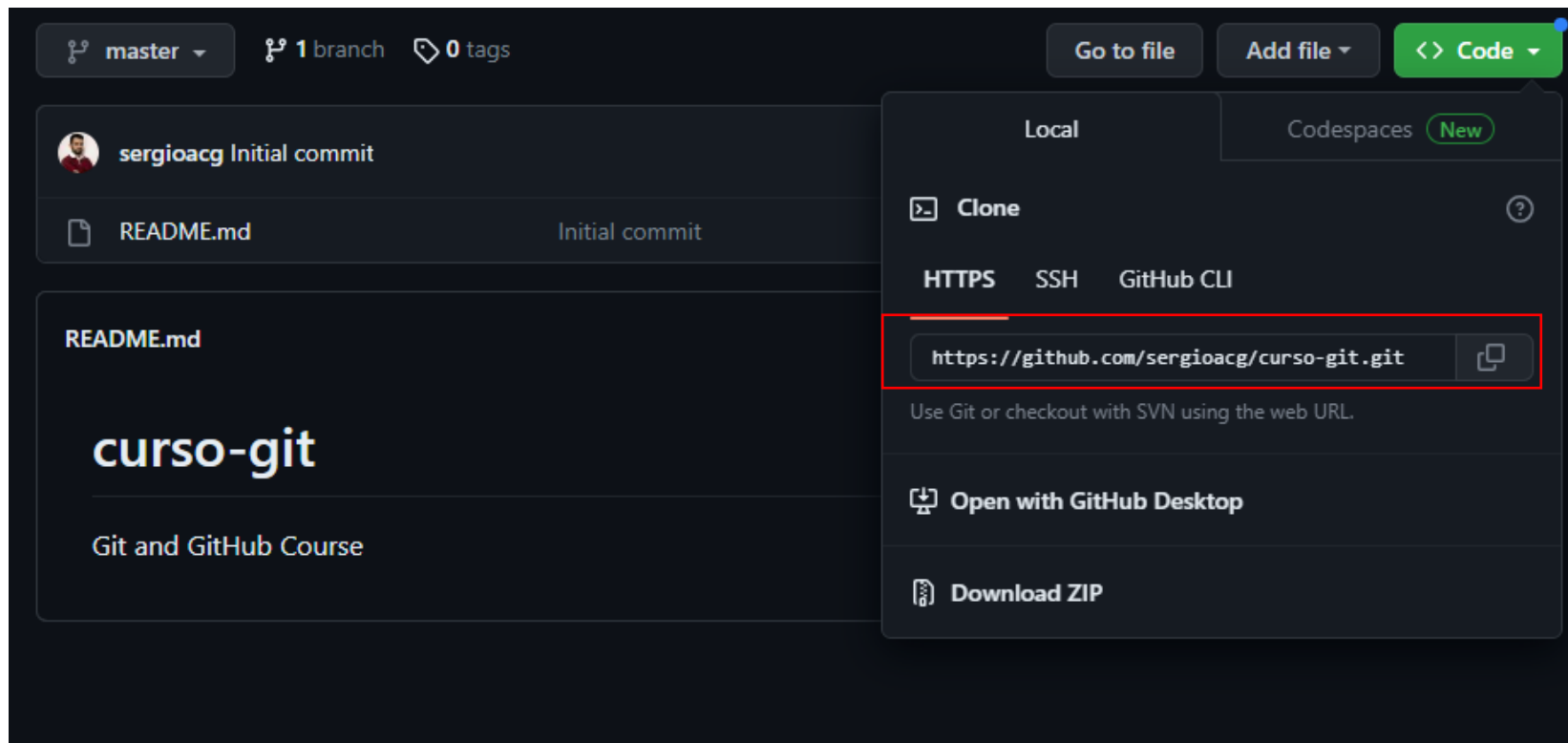
Esta é a URL do repositório que você criou no GitHub.

É um nome curto e amigável que damos à URL do repositório remoto. É um padrão da indústria usar "origem" para o repositório principal no GitHub.

Indica que vamos adicionar uma nova conexão remota.

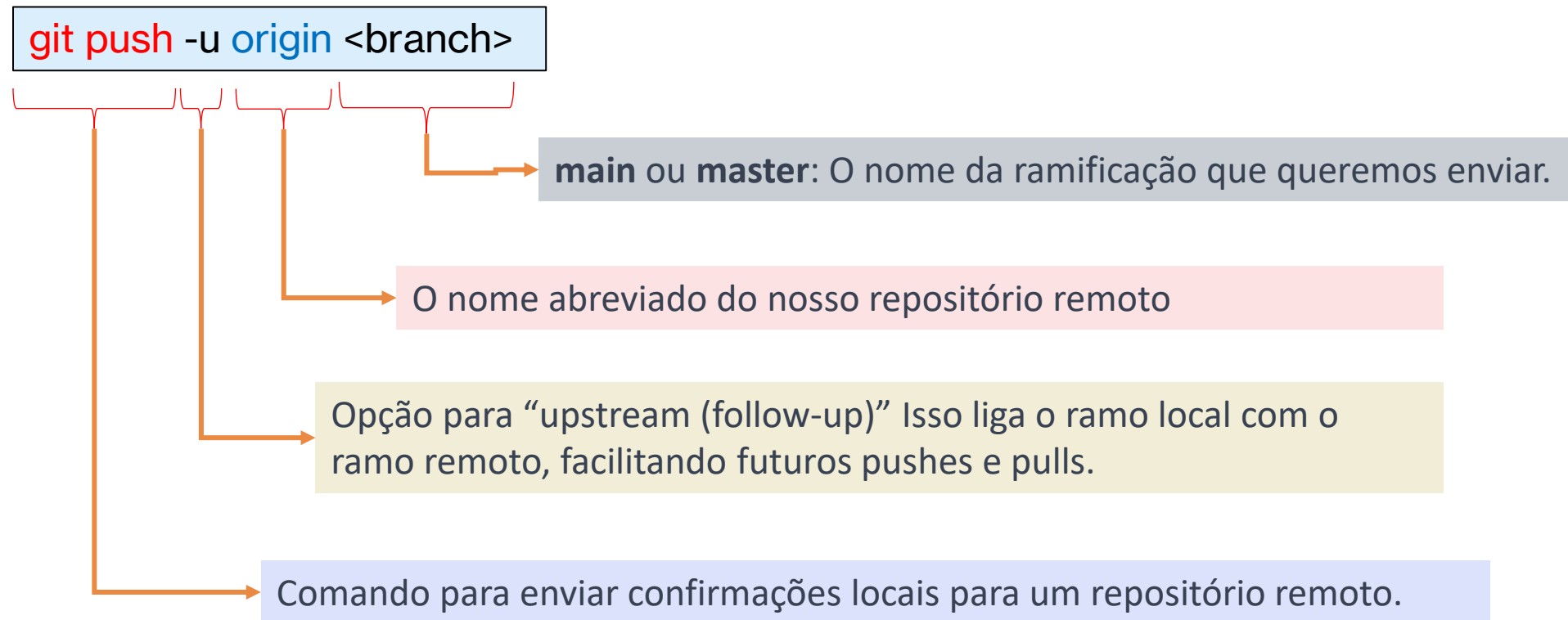
É o comando usado para gerenciar conexões com repositórios remotos.

Conectando nosso repositório local com o GitHub



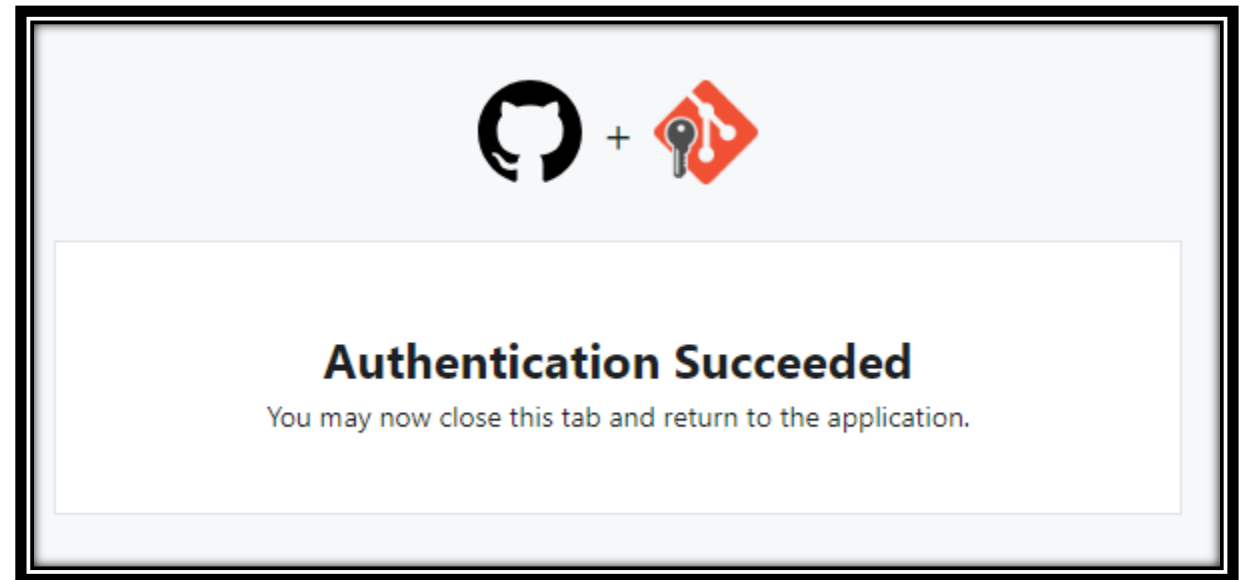
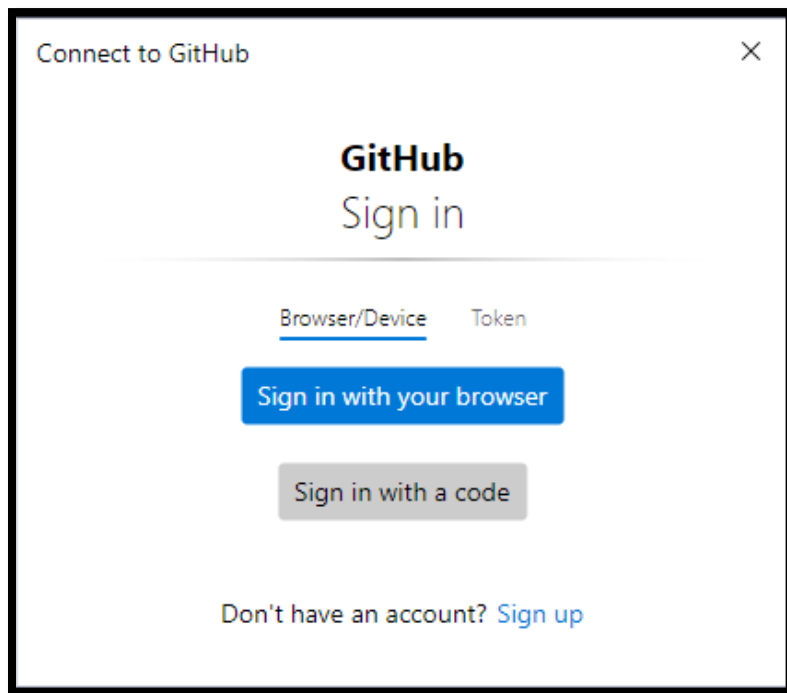
Empurrando nossas alterações para o GitHub

Carregue nossas alterações locais no repositório remoto no GitHub, permitindo-nos compartilhar nosso projeto e colaborar com outras pessoas.



Empurrando nossas alterações para o GitHub

```
git remote add origin https://github.com/sergioacg/curso-git.git  
git push -u origin master
```



Resolvendo conflitos de push

Um erro bastante comum ao trabalhar com repositórios Git remotos é quando você tenta enviar para uma ramificação remota que possui alterações que você não possui em sua ramificação local. Em outras palavras, alguém fez alterações no repositório remoto que você não incorporou à sua cópia local.

```
info: please complete authentication in your browser...
To https://github.com/sergioacg/curso-git.git
! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'https://github.com/sergioacg/curso-git.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
PS H:\My Drive\Clases\Programación 2\Git\Proyecto1>
```

Resolvendo conflitos de push

```
! [rejected]      master -> master (fetch first)
```

Opción 1: *git pull*

1. Executa *git pull origin master* para mesclar as alterações remotas em sua ramificação local.
2. Resolva quaisquer conflitos, se existirem.
3. Faz um *git push origin master* de novo.

Opción 2: *git pull con rebase*

1. Executa:
git pull --rebase origin master para aplicar suas alterações locais sobre as alterações remotas.
2. Resolva quaisquer conflitos, se existirem.
3. Faz um *git push origin master* de novo.

Resolvendo conflitos de push

Esse erro é uma variação do erro anterior e está relacionado ao que é conhecido como "**non-fast-forward**". É outro exemplo de tentativa de **push** para um branch remoto que foi atualizado por outra pessoa desde a última vez que verificamos.

```
! [rejected]        master -> master (non-fast-forward)
error: failed to push some refs to 'https://github.com/sergioacg/curso-git.git'
hint: Updates were rejected because the tip of your current branch is behind
hint: its remote counterpart. Integrate the remote changes (e.g.
hint: 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

Visualizando todos os ramos

Como visualizar ramos locais:

Comando: *git branch*

- Mostrar todas os ramos locais.

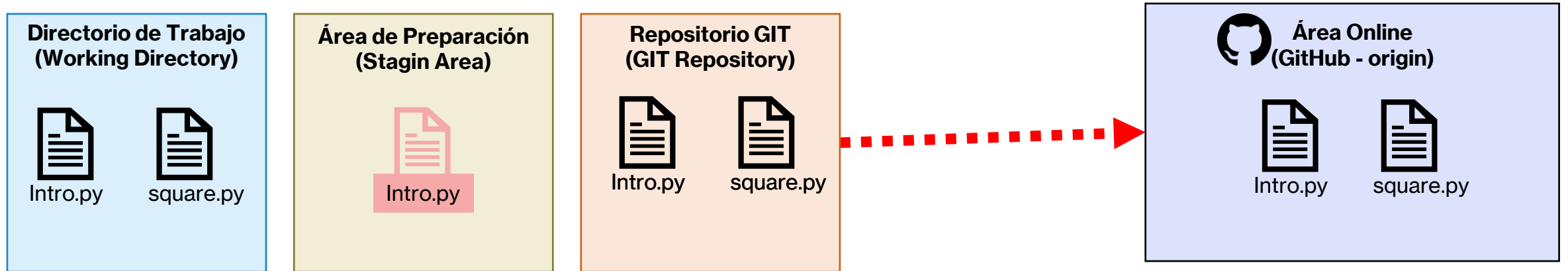
Como visualizar ramos Remotos:

- Comando: *git branch -r*
- Mostrar todas os ramos remotos.

Como visualizar Ambos Ramos Locais e Remotos:

- Comando: *git branch -a*
- Mostrar todas os ramos.

Empurrando para o repositório remoto



```
print('Hello World!')
print('Hello GitHub')
```

git add .

git commit -m "Hi GitHub"

git push origin master

Nome do ramo

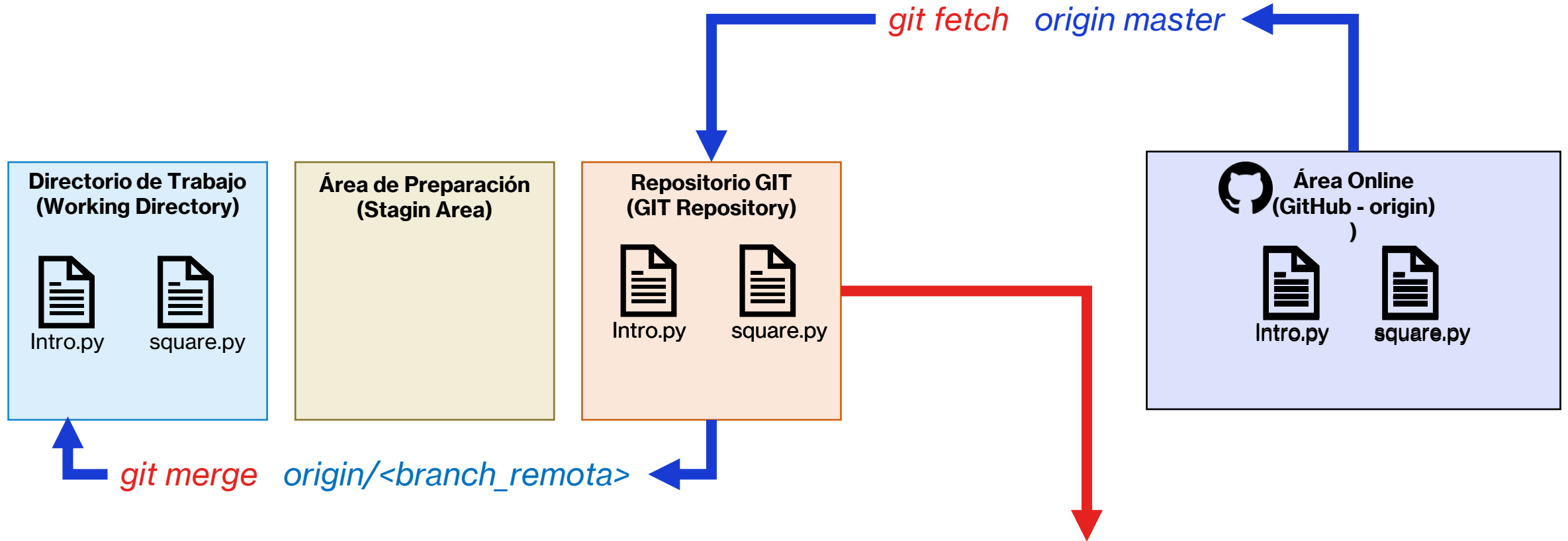
```
PS H:\My Drive\Clases\Programación 2\Git\Proyecto1> git push origin master
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 16 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 391 bytes | 195.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/sergioacg/curso-git.git
   cfb310..14407d6  master -> master
```

Atualizando o repositório local com o remoto

Se alguém da sua equipe fez alterações no repositório remoto (GitHub) e você deseja atualizar os arquivos do repositório local com essas alterações, há duas maneiras:

1. *git fetch*: busca alterações do repositório remoto para seu repositório local, mas NÃO modifica seu espaço de trabalho local. Isso permite que você verifique as alterações e, se concordar, pode mesclar seu repositório local com seu espaço de trabalho usando um *git merge*.
2. *git pull*: Puxa as alterações do repositório remoto e as mescla automaticamente em seu espaço de trabalho e em seu repositório local.

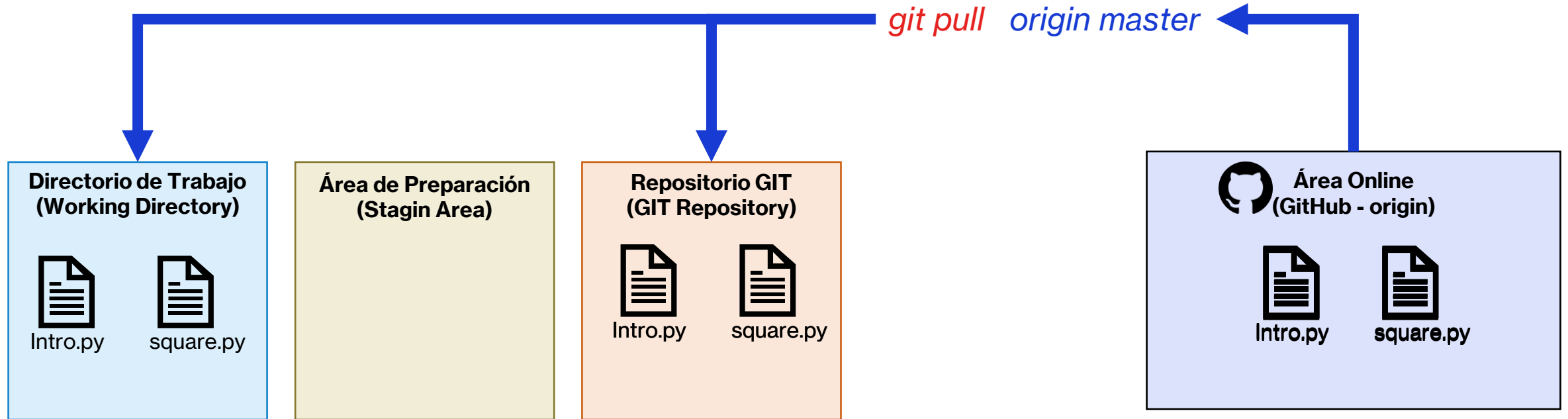
Atualizando o repositório local com o remoto



Verificar alterações:

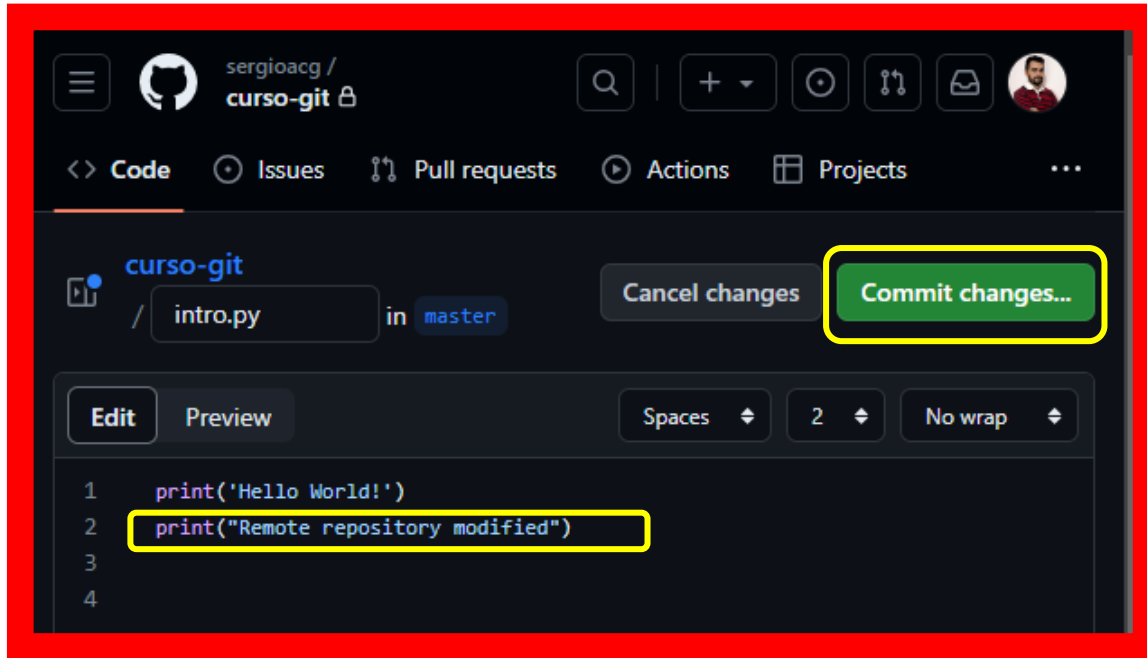
- `git diff <branch_local> origin/<branch_remota>`
- `git log --oneline --graph --all`

Atualizando o repositório local com o remoto

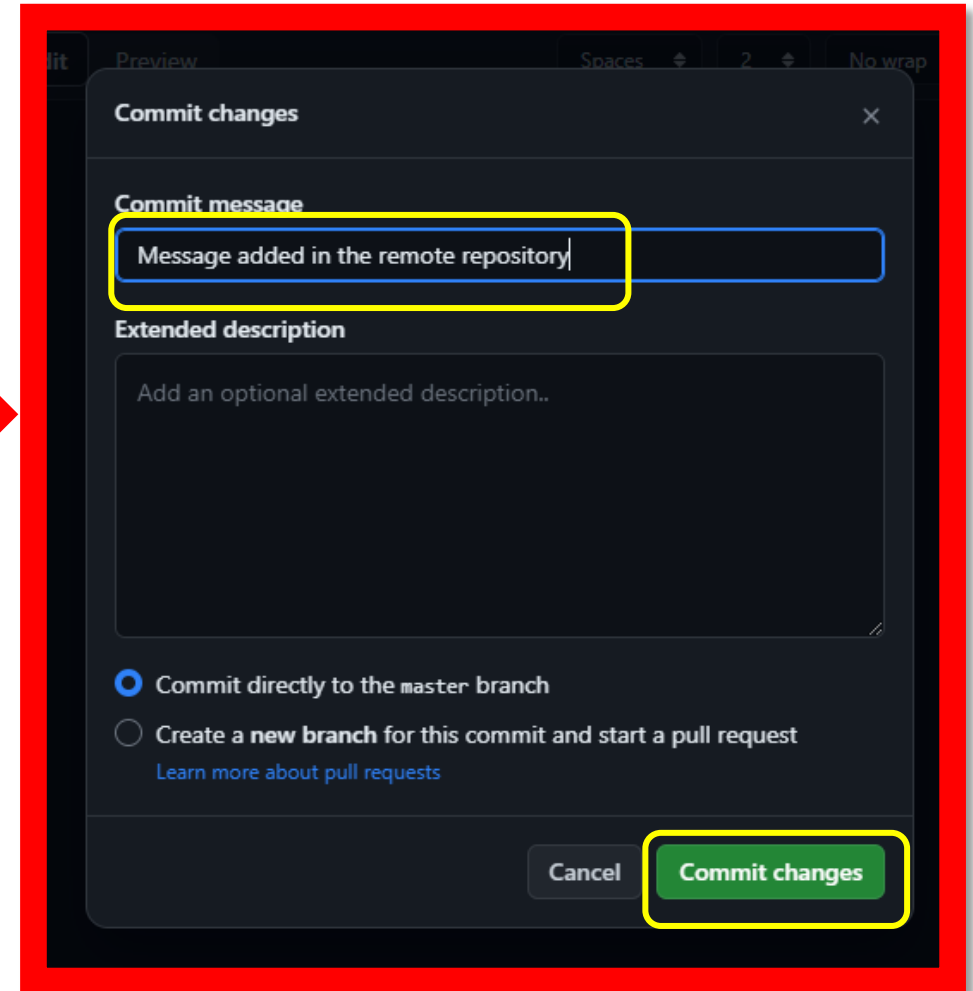


Modificando o repositório remoto via GitHub

1.



2.



Clonar um repositório do GitHub



O que significa clonar?

Clonar um repositório significa obter uma cópia completa do repositório remoto em sua máquina local. Inclui todo o histórico de commit e branch.



Escenario: quero trabalhar em grupo com outros colegas

Sem problemas, podemos clonar o repositório do GitHub e fazer um trabalho colaborativo

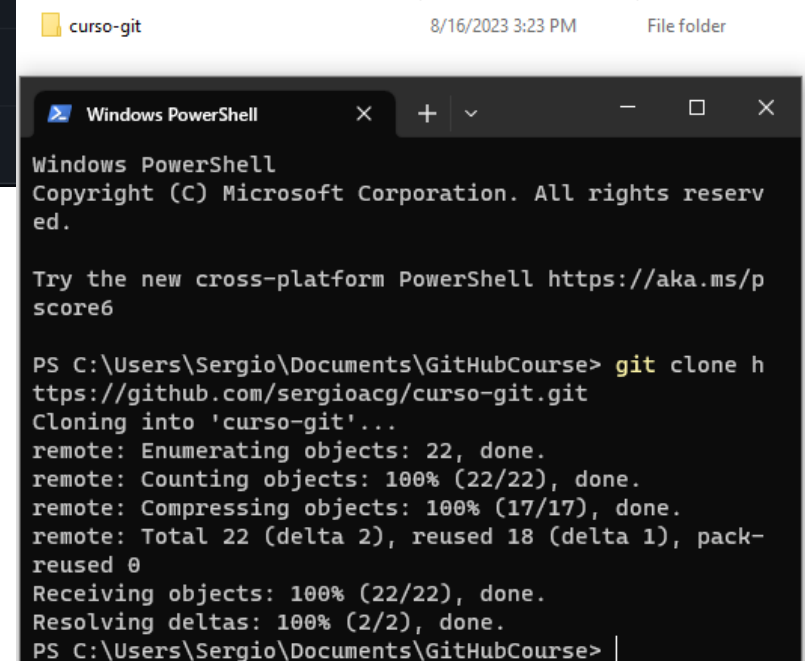
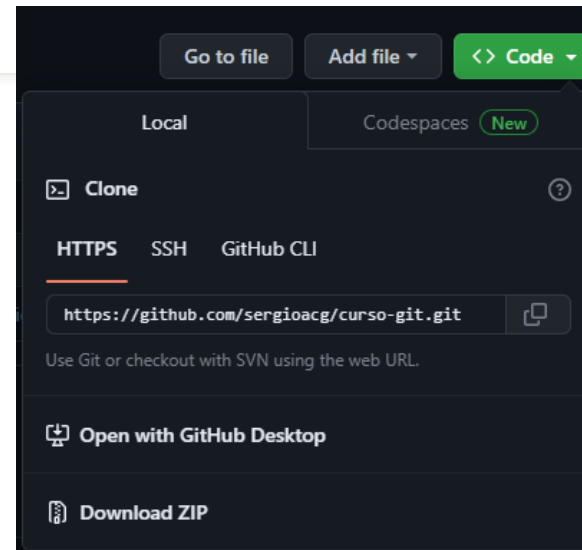
Clonar um repositório do GitHub

Passo 1: Encontre a URL do repositório no GitHub.

- Use HTTPS ou SSH, dependendo da sua configuração.

Passo 2: Usa o comando *git clone*.

- Exemplo:
git clone URL_REPO
- Isso criará uma nova pasta com o nome do projeto e copiará todo o conteúdo para lá.



Trabalhando de forma colaborativa no GitHub

Clonamos o repositório de outra conta no GitHub

```
Sergio@DESKTOP-4PAPK91 MINGW64 ~/Documentos
$ git clone https://github.com/lades-
hub/CursoGit.git
Cloning into 'CursoGit'...
remote: Enumerating objects: 3, done
remote: Counting objects: 100% (3/3),
done.
remote: Total 3 (delta 0), reused 0 (
delta 0), pack-reused 0
```


Trabalhando de forma colaborativa no GitHub

Modificamos o arquivo

```
git add .  
git commit -m "modified"
```

```
git push origin master
```



ERRO



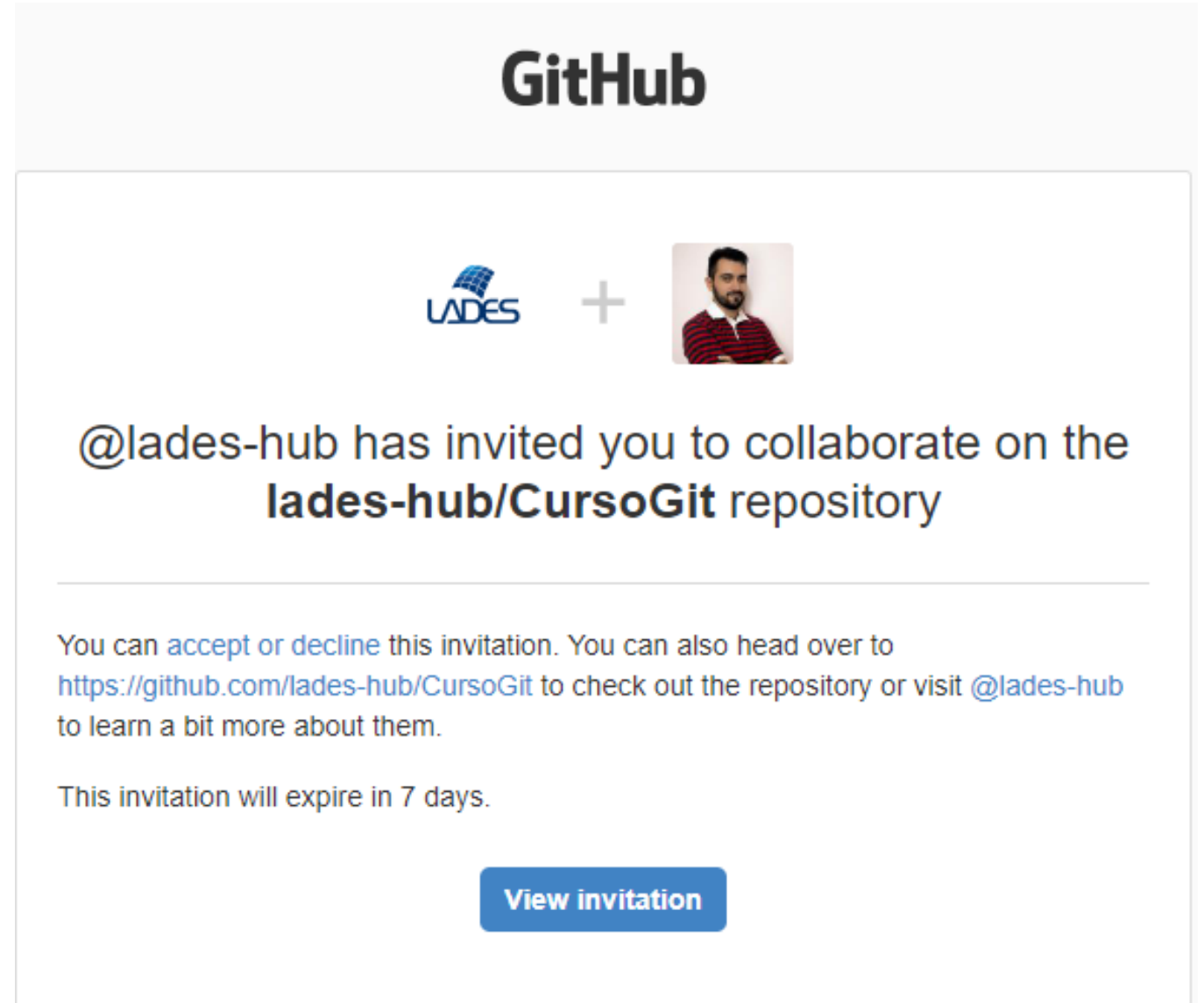
Não podemos atualizar nosso repositório de **grupo de trabalho** até que o proprietário do repositório nos coloque como colaboradores

Trabalhando de forma colaborativa no GitHub

The screenshot shows the GitHub repository settings page for 'lades-hub / CursoGit'. The 'Settings' tab is selected in the top navigation bar, highlighted with a red box and labeled '1.'. In the left sidebar, the 'Collaborators' option is selected, highlighted with a red box and labeled '2.'. The main content area shows 'Who has access' with 'PUBLIC REPOSITORY' and 'DIRECT ACCESS' sections. Below this, the 'Manage access' section is expanded, showing a message 'You haven't invited any collaborators yet' and a green 'Add people' button, which is highlighted with a red box and labeled '3.'.

The profile card for Sergio Andres Castaño Giraldo is shown. It features a circular profile picture of a man with a beard wearing a red and black striped shirt. Below the picture, the name 'Sergio Andres Castaño Giraldo' is displayed, and the username 'sergioacg' is highlighted with a yellow box and labeled '4.'.

Verificar e-mail de colaboração



Trabalhando de forma colaborativa no GitHub

IMPORTANTE

A forma de trabalhar em grupos depois de serem colaboradores em um projeto é fazê-lo em um **BRANCH** separado do Branch **master** e cada colaborador ou dono do repositório deve trabalhar lá e só mesclar as alterações para main ou master quando aceitarem as alterações produzidas.

```
git checkout -b sergio-code
```

Trabalhando de forma colaborativa no GitHub

Criamos algumas modificações e enviamos para o repositório de origem.



```
EXPLORER  ...  README.md  product.py X
└─ CURSOGIT
   └─ product.py
      └─ README.md

product.py > ...
1
2
3  def product(a, b):
4      return a * b
5
6  if __name__ == '__main__':
7      print(product(2, 3))
```

```
git add .
```

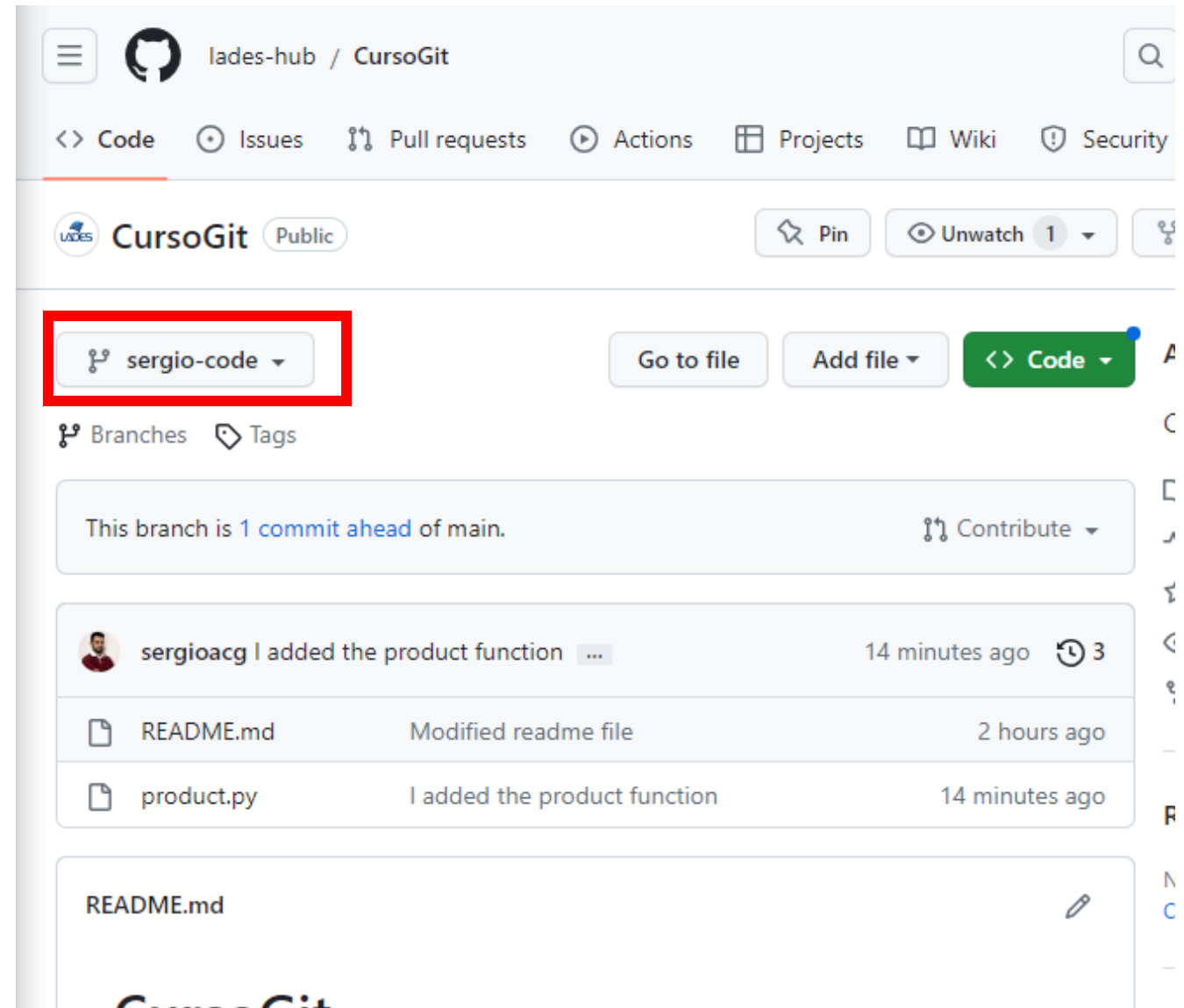
```
git commit -m "function"
```

```
git pull origin sergio-code
```

```
git push origin sergio-code
```

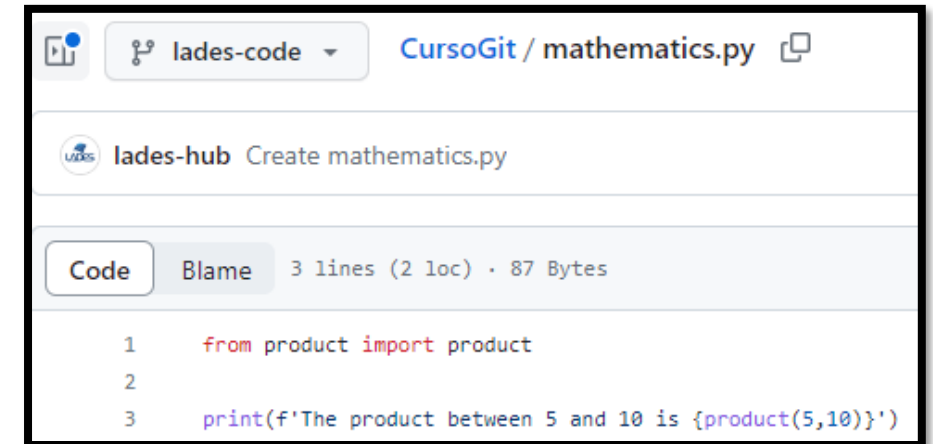
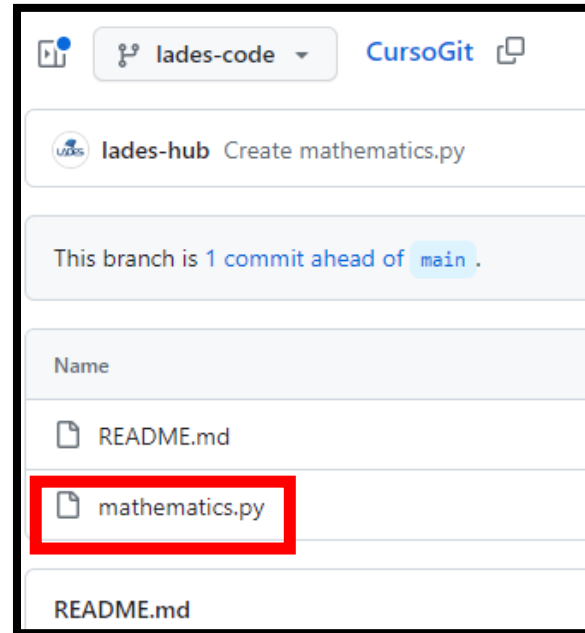
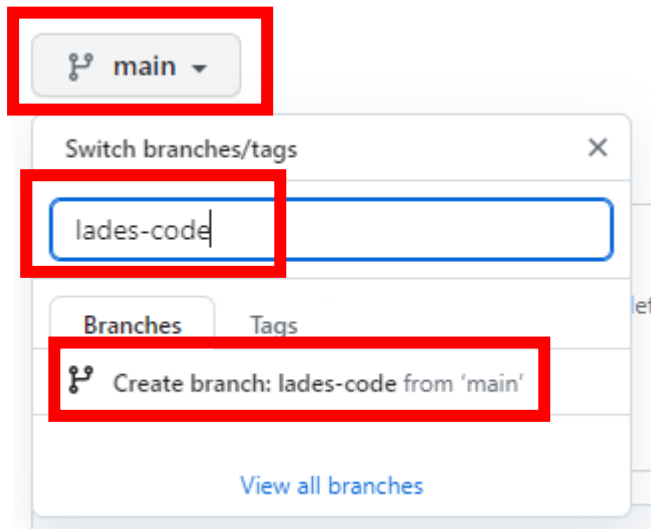
Trabalhando de forma colaborativa no GitHub

Podemos ver o Branch online



Trabalhando de forma colaborativa no GitHub

Criamos alguma modificação com o dono do repositório em **outra** Branch criada a partir do **main**.



Código FINAL (Produção)

- Um dos membros da equipe deve ser responsável por mesclar o código para o ramo **main** ou **master**. (pode ser o dono do repositório ou um colaborador)

```
git add checkout main
```

```
HEAD -> main
```

```
git merge sergio-code
```

➔ Eu mesclo o ramo com o principal

```
git pull origin main
```

➔ Verifico que não houve alterações no repo da internet

```
git push origin main
```

➔ Eu mando os dados para o repo da internet

```
git checkout lades-code origin/lades-code
```

Somente se o ramo não existir localmente

```
git merge lades-code
```

➔ Eu mesclo o ramo com o principal

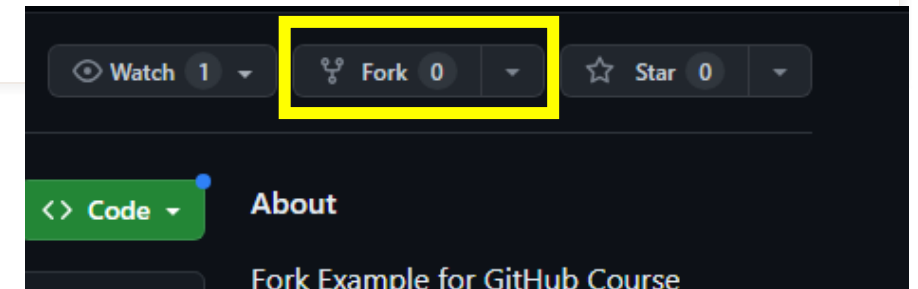
```
git pull origin main
```

➔ Verifico que não houve alterações no repo da internet

```
git push origin main
```

➔ Eu mando os dados para o repo da internet

FORK: Contribuir para um Repositório no qual Você Não é Colaborador



- **Fazer Fork do Repositório:** Isso cria uma cópia do repositório na sua conta GitHub. Você pode fazer isso clicando no botão "Fork" no canto superior direito da página do repositório de interesse.

<https://github.com/lades-hub/ExampleFork>

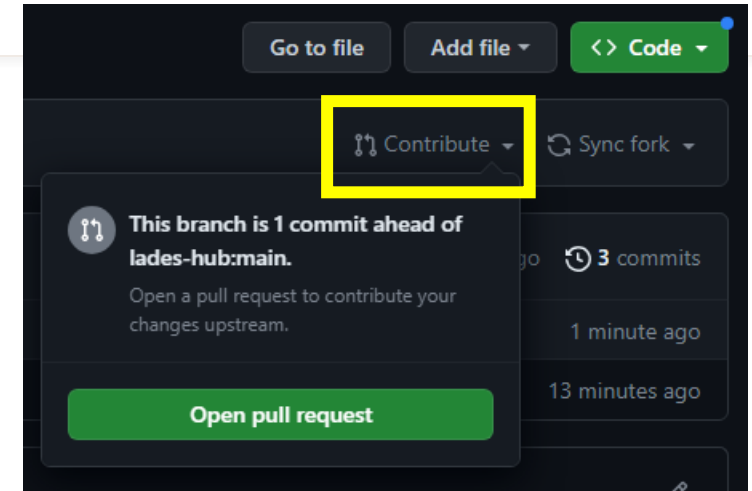
- **Clonar o seu Fork:** Após fazer o fork do repositório, você pode cloná-lo para sua máquina local usando git clone URL_DO_SEU_FORK.

FORK: Contribuir para um Repositório no qual Você Não é Colaborador

- **Criar uma Ramificação para suas Alterações:** É uma boa prática criar uma ramificação para suas alterações. Você pode fazer isso com `git checkout -b nome_da_sua_ramo`.
- **Realizar Alterações:** Faça as alterações que você deseja na sua ramificação.
- **Commit e Push:** Faça o commit de suas alterações com `git commit -m "Descrição das alterações"` e depois faça o push para o seu fork com `git push origin nome_da_sua_ramo`.

FORK: Contribuir para um Repositório no qual Você Não é Colaborador

Criar uma Solicitação de Pull: Vá para a página do seu fork no GitHub e clique em "New Pull Request". Isso permitirá que você compare as alterações no seu fork com o repositório original. Preencha as informações necessárias e clique em "Create Pull Request".



- **Aguardar Revisão:** O proprietário do repositório original revisará suas alterações. Eles podem aceitá-las, solicitar mudanças ou fechar a solicitação de pull.
- **Sincronizar Alterações Futuras:** Se você planeja continuar contribuindo, vai querer manter o seu fork sincronizado com o repositório original. Você pode fazer isso adicionando o repositório original como um remoto e depois usando git fetch e git merge.

Organização no GitHub: LADES-PEQ

O que é uma Organização no GitHub?

- Uma organização é um espaço onde os projetos podem ser gerenciados de maneira colaborativa.
- Ideal para grupos, equipes ou projetos de pesquisa.
- Facilita o gerenciamento de permissões e o controle de acesso.

Organização no GitHub: LADES-PEQ

- 1. Visite o link da organização:** [LADES-PEQ no GitHub](#)
- 2. Solicite um Convite:** Fale com o administrador ou o responsável pela organização para que lhe envie um convite.
- 3. Aceite o Convite:** Uma vez que você receber o convite por e-mail ou em sua conta GitHub, clique no link para aceitar.
- 4. Comece a Colaborar:** Após fazer parte da organização, você pode começar a contribuir com os repositórios disponíveis, criar novos projetos ou participar das discussões.

Enviando um Repositório para a Organização LADES-PEQ

Se você tem um repositório em sua conta pessoal e deseja transferi-lo para a organização LADES-PEQ, siga estes passos:

- 1. Acesse o Repositório:** Entre no repositório que deseja transferir.
- 2. Clique em 'Settings':** Na parte inferior da página, vá para a seção "Settings" (Configurações).
- 3. Vá até 'Transfer Ownership':** Desça até a seção "Danger Zone" e clique em "Transfer ownership" (Transferir propriedade).
- 4. Digite o Nome da Organização:** No campo fornecido, digite "LADES-PEQ" como o novo proprietário.
- 5. Confirme a Senha:** Você será solicitado a confirmar sua senha para garantir que tem permissão para realizar esta ação.
- 6. Aceite o Convite:** Um administrador da LADES-PEQ deve aceitar a transferência. Se você é o administrador, apenas aceite o convite.
- 7. Pronto!:** Seu repositório agora faz parte da organização LADES-PEQ, e você ainda permanece como colaborador.