

Trabalhando em equipe no Git - Fluxo Git

Um Fluxo de trabalho do Git é uma receita ou recomendação sobre como usar o Git para realizar o trabalho de maneira consistente e produtiva. Os fluxos de trabalho do Git incentivam os usuários a aproveitar o Git de modo eficiente e consistente.

1) O que é um fluxo de trabalho bem-sucedido do Git?

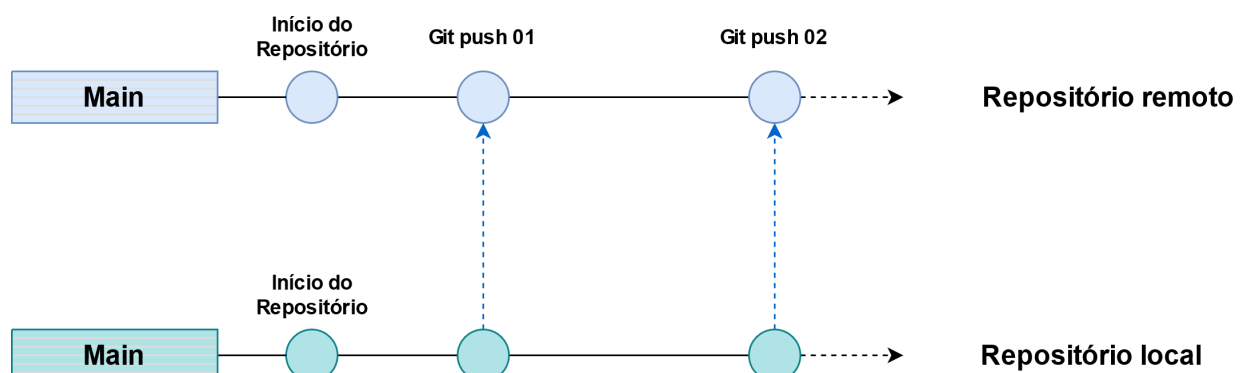
Ao avaliar um fluxo de trabalho para sua equipe, o mais importante é entender a cultura da equipe. O fluxo de trabalho deve melhorar a eficácia da equipe e não ser uma carga que limita a produtividade.

Algumas coisas importantes que devem ser consideradas ao avaliar um fluxo de trabalho do Git são:

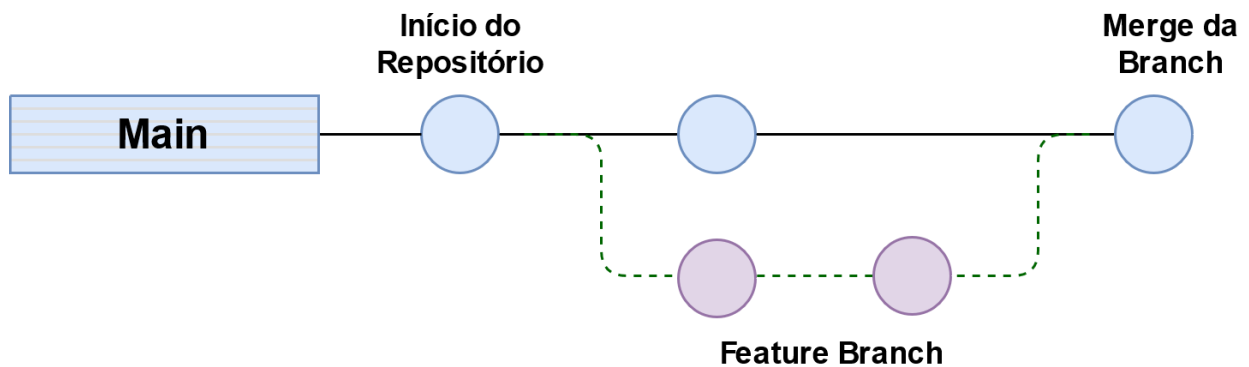
- Este fluxo de trabalho é dimensionado com o tamanho da equipe?
- É fácil desfazer erros com este fluxo de trabalho?
- Este fluxo de trabalho impõe alguma nova sobrecarga cognitiva desnecessária à equipe?

1.1) Tipos de Fluxo

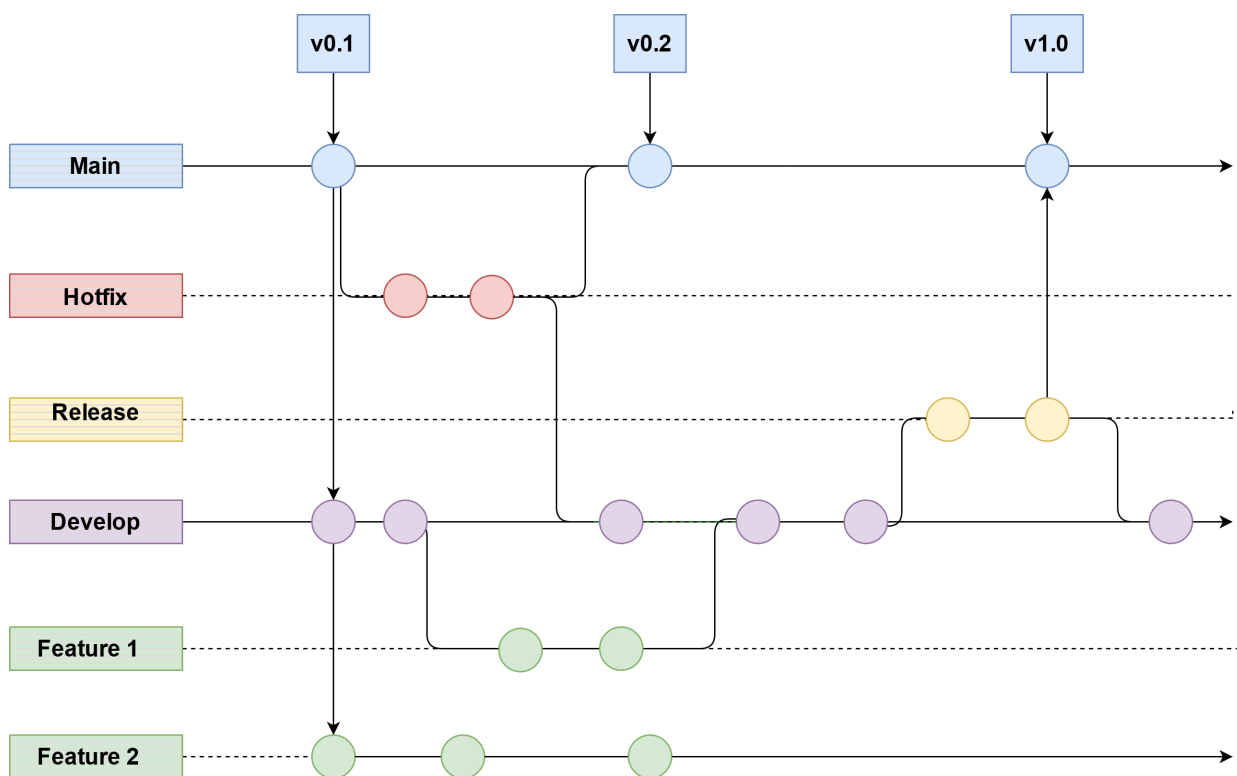
- **Fluxo de trabalho centralizado:** A ideia central por trás do Fluxo de trabalho centralizado é que todo o desenvolvimento de recursos deve ocorrer na branch main.



- **Fluxo de trabalho de ramificação de recurso:** A ideia central por trás do Fluxo de trabalho de ramificação de recursos é que cada feature deve ocorrer em uma Branch dedicada, que só é enviada para a Branch Main quando se torna parte de uma nova versão.

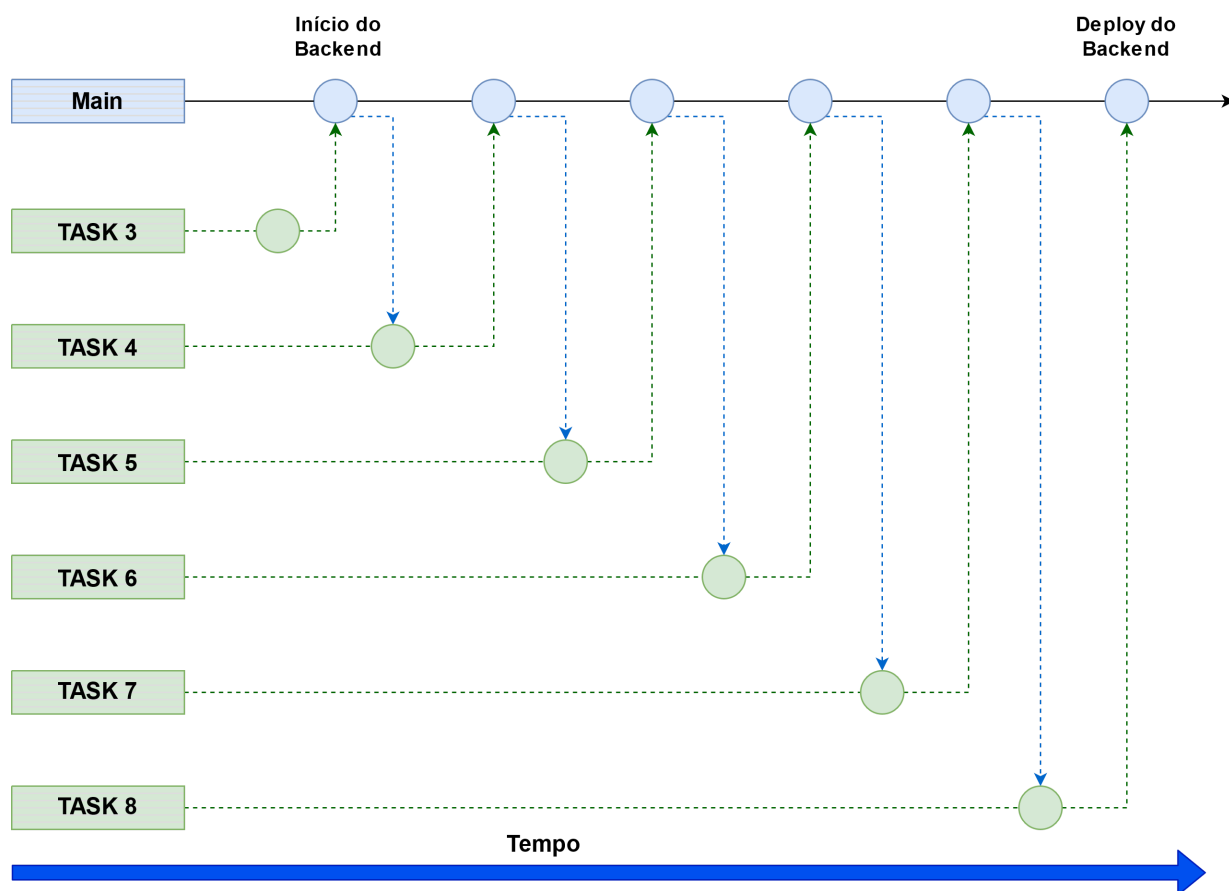


- **Fluxo de trabalho Gitflow:** Define um modelo de ramificação rigoroso projetado com base no lançamento do projeto oferecendo uma estrutura robusta para gerenciar grandes projetos.



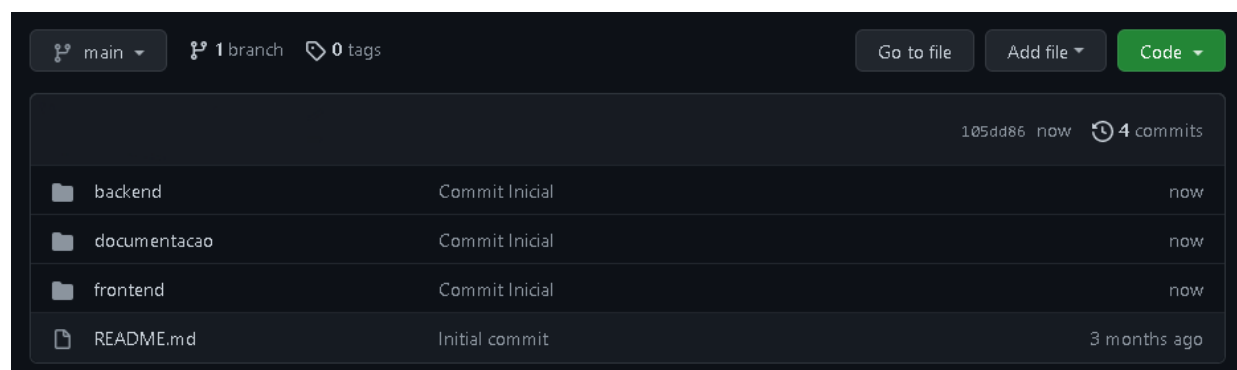
▶ Para saber mais sobre o GitFlow, assista ao vídeo *Trabalhando em equipe com Git Flow* no link: <https://www.youtube.com/watch?v=394mc6PV8t8> (Acessado em: 28/09/2021)

No projeto Integrador, utilizaremos o modelo **Fluxo de trabalho de Ramificação de recurso**, conforme a estrutura detalhada na figura abaixo, onde cada **Task** do Projeto Integrador será uma **Feature Branch** no Repositório Remoto no Github.



Fluxo de trabalho do Bloco 02 - Back-end

Para manter o repositório organizado, utilizaremos a estrutura de pastas apresentada na figura abaixo:



Organização proposta do repositório

Pasta	Conteúdo
Documentação	Arquivos contendo a documentação da API: <ul style="list-style-type: none">- Documentação do Banco de Dados (DER, SQL e Dicionário de dados)- Documentação do Backend (Abertura de projeto, Dicionário de atributos e PDF do Swagger)- Documentação do Frontend
Backend	Projeto Spring completo
Frontend	Projeto Angular/React Completo (Bloco 03)

Nos próximos tópicos faremos uma breve revisão do Git, onde mostraremos como criar um repositório remoto, inserir os colaboradores, iniciar o projeto no Repositório Local, enviar o projeto para o Repositório Remoto e para finalizar faremos algumas simulações com situações comuns no trabalho em equipe no Github.

2) Configurando o Repositório Local

Para começar vamos configurar o Microsoft Visual Studio Code para ser a IDE padrão do Git, desta forma poderemos utilizar os recursos que o VSCode oferece para resolução de conflitos.

1. Verifique se o **VSCode** está instalado na sua máquina
2. Caso não esteja, faça o download no link: <https://code.visualstudio.com/download> e faça a instalação

Caso Você tenha alguma dúvida, utilize o **Guia de Instalação do VSCode** e siga o passo a passo da instalação.

3. Após confirmar ou instalar o VSCode, abra o **Git Bash**
4. Configure o VSCode como o Editor padrão do Git através do comando abaixo:

```
git config --global core.editor 'code --wait'
```

3) Criando repositório Central no Github


Vamos configurar o repositório Central no Github:

1. Acesse a conta do Github onde será criado o Repositório Remoto do projeto (Conta do projeto ou de um participante do grupo)
2. Em **Repositories**, clique no botão **New** para adicionar um novo repositório.
3. Crie um **Repositório Público**, chamado **projeto_integrador** ou utilize o **nome do seu projeto**, e adicione o arquivo **README**, como mostra a figura abaixo. Em seguida clique no botão **Create Repository**.

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)



Owner * Repository name *

 rafaelproinfo ▼ / projeto_integrador ✓

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

Description (optional)

Projeto Integrador

- ☒  **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.](#)

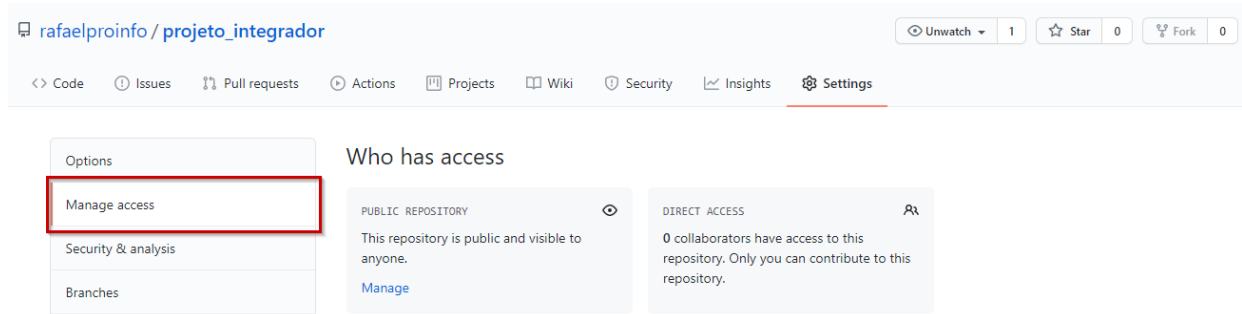
☐ **Choose a license**
A license tells others what they can and can't do with your code. [Learn more.](#)

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

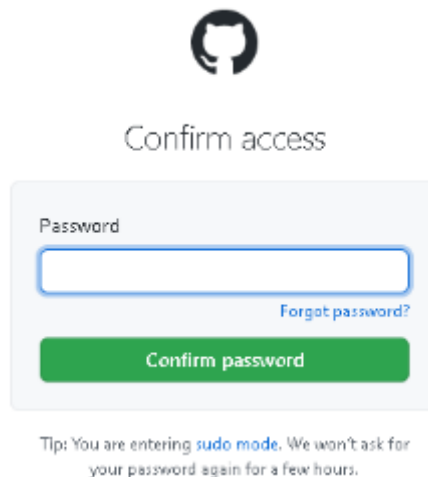
Create repository

3.1) Adicionando os membros da sua Equipe no Repositório como colaboradores

1. Dentro do Repositório do projeto integrador, clique em **Settings** → **Manage access**

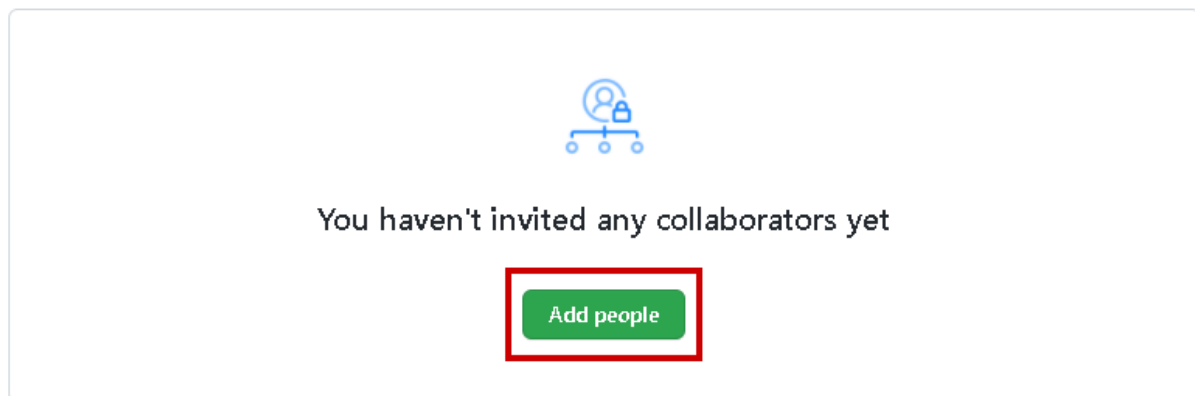


2. Digite a senha do Usuário do Github para continuar, caso seja solicitado

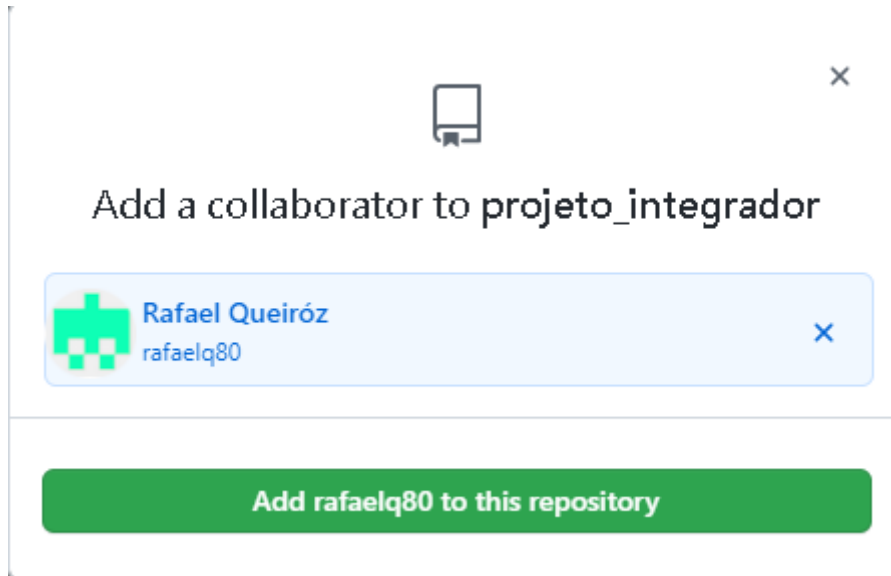


3. Em **Manage access**, clique no botão **Add people**

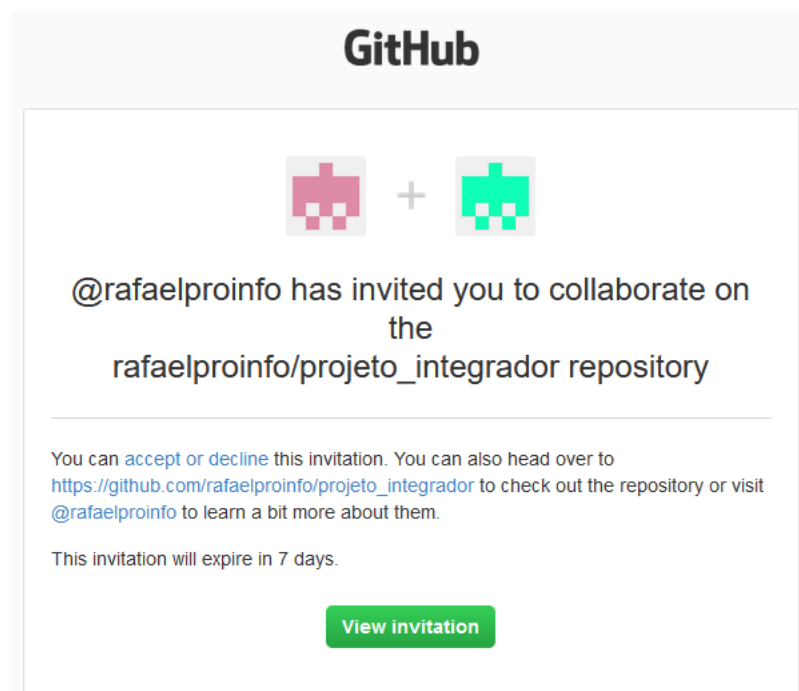
Manage access



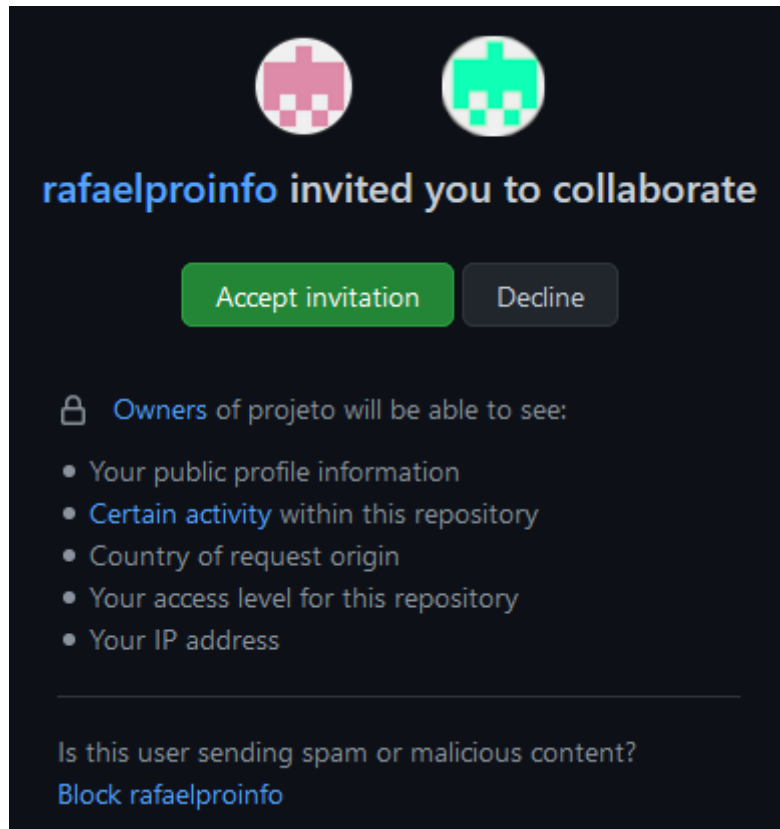
4. Localize o Colaborador que você deseja adicionar e clique no botão **Add** **<nome_usuario_github> to this repository**, como mostra a imagem abaixo:



5. Repita os passos anteriores para adicionar os demais membros do grupo
6. O Colaborador receberá um convite via e-mail. Para aceitar o convite, o Colaborador deverá clicar no link **View invitation**



7. O Colaborador será redirecionado para o site do Github. Para aceitar o convite, o Colaborador deverá clicar no botão **Accept invitation**



8. O acesso ao repositório está liberado

3.2) Insights

Um recurso interessante do Github é o **Insights**. Com ele é possível acompanhar através de gráficos e dados estatísticos a colaboração de cada membro da equipe com o projeto e os dados estatísticos do repositório como um todo.

1. Para acessar, clique no link  **Insights** do repositório remoto no github.
2. Na próxima janela, clique em **Contributors**. Você verá uma janela semelhante a figura abaixo:



Observe que cada usuário adicionado possui o seu gráfico de colaboração no repositório.

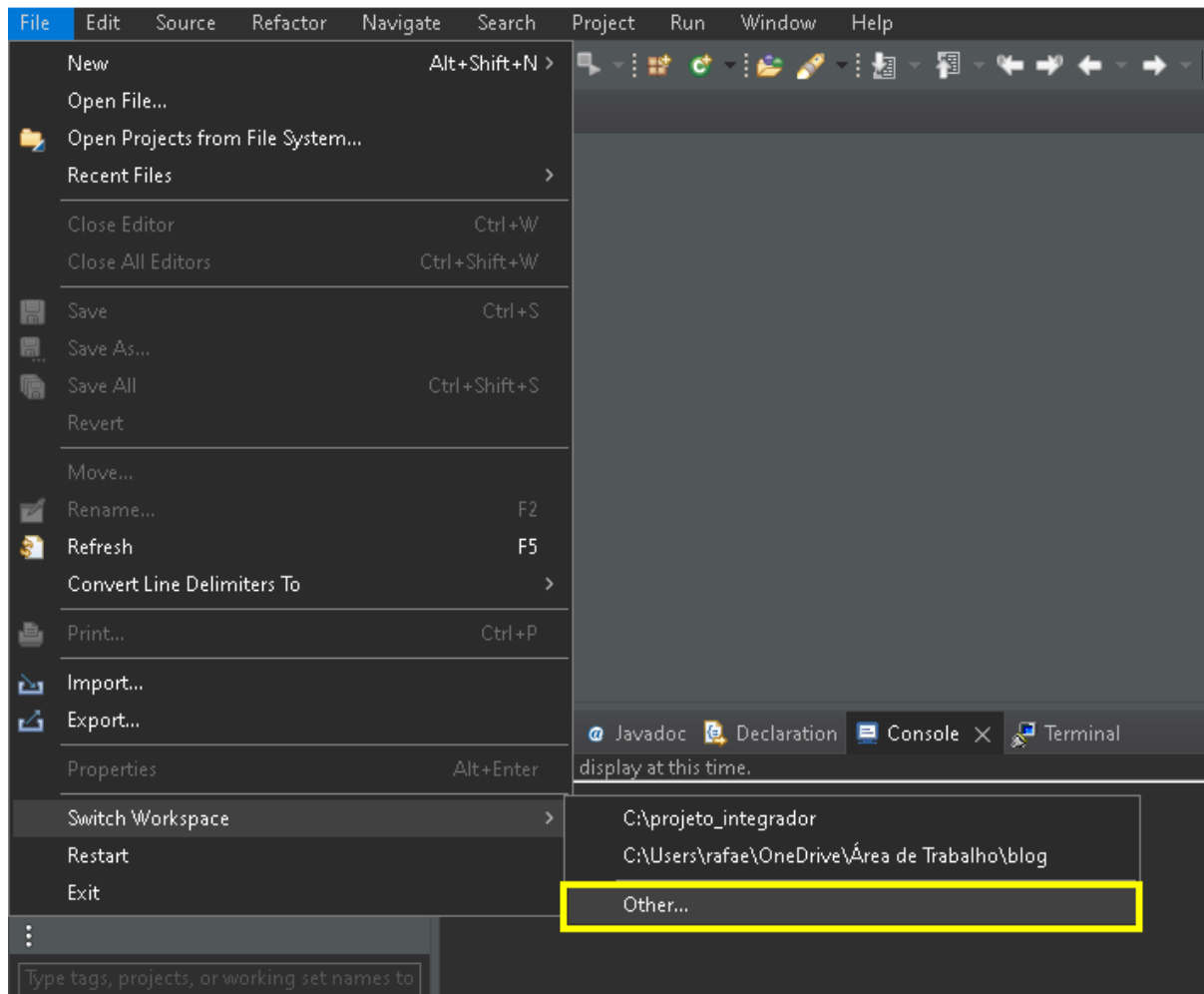
4) Fluxo de Trabalho na máquina local

4.1) Criando o Repositório local

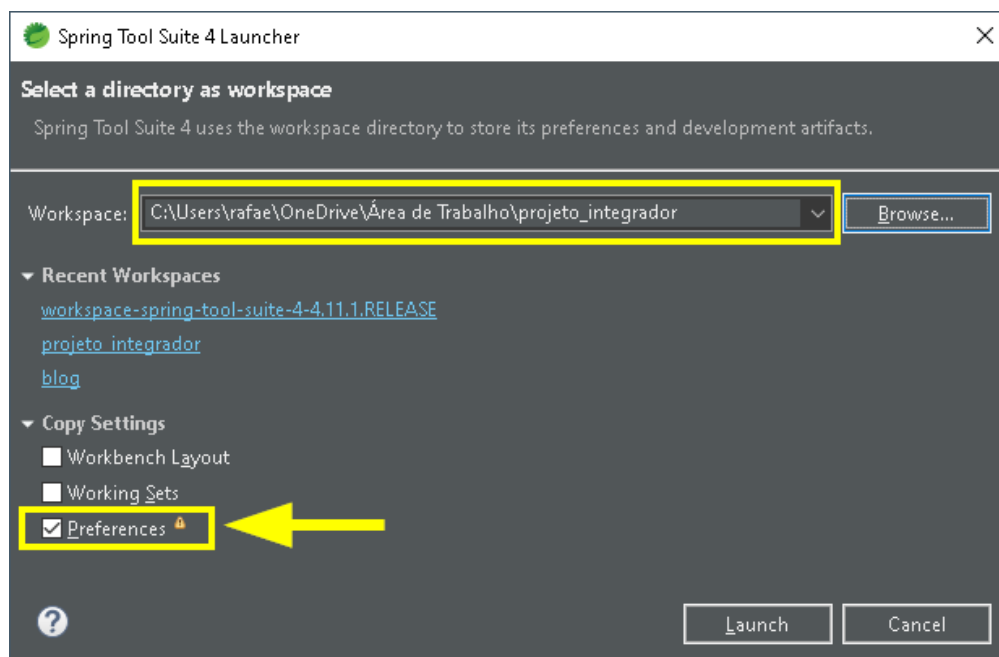
Vamos criar o repositório local, que será conectado ao repositório remoto no Github e configurá-lo como uma Workspace do STS.

1. Crie uma pasta na Área de trabalho chamada **projeto_integrador** ou utilize o **nome do seu projeto**
2. Na pasta **projeto_integrador**, vamos criar uma nova **Workspace** (área de trabalho do STS), do Projeto Integrador

3. Clique no Menu **File** → **Switch Workspace** → **Other...**



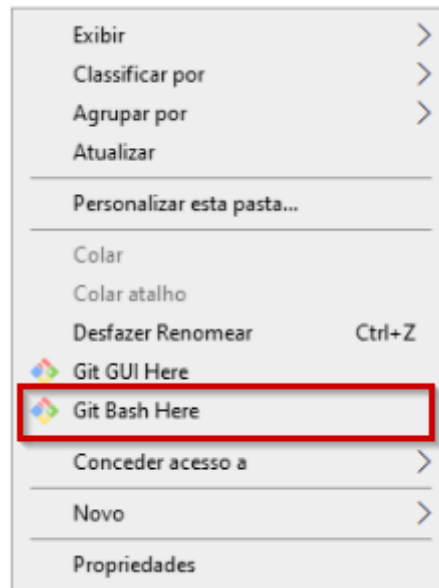
4. No item **Workspace**, adicione a pasta **projeto_integrador**, criada na **Área de Trabalho**. Deixe marcada a opção **Preferences**, para copiar as configurações atuais do STS na nova Workspace. Clique no botão **Launch** para concluir.



5. Dentro da pasta **projeto_integrador**, crie um arquivo chamado **.gitignore** e adicione a linha abaixo:

```
.metadata
```

6. Dentro da pasta **projeto_integrador**, vamos criar a pasta: *documentacao*
7. Dentro da pasta **projeto_integrador**, clique com o botão direito do mouse e clique na opção: **Git Bash Here**



8. No **Git Bash**, execute a sequência de comandos abaixo para conectar a sua pasta com o repositório remoto **projeto_integrador**.

```
git init
```

```
git branch -M master main
```




```
git remote add origin https://github.com/repositorio/projeto_integrador.git
```

```
git pull origin main
```

```
git remote -v
```

Comando	Descrição
<code>git init</code>	Inicializa um repositório git local dentro da pasta projeto_integrador.
<code>git branch -M master main</code>	Renomeia a branch local master para main.
<code>git remote add origin endereço_remoto</code>	Associa o repositório local ao repositório remoto. O endereço_remoto será o endereço do seu repositório.
<code>git pull origin main</code>	Atualiza o seu repositório local com todos os arquivos disponíveis no repositório remoto.
<code>git remote -v</code>	Checa se o seu repositório local está conectado ao repositório remoto

9. A sua pasta local ficará com seguinte estrutura após a execução da sequência de comandos acima:

Nome	Status	Data de modificação	Tipo	Tamanho
.git		05/12/2021 01:26	Pasta de arquivos	
documentacao		05/12/2021 01:20	Pasta de arquivos	
README.md		05/12/2021 01:26	Markdown File	1 KB

10. Na pasta **documentacao**, copie toda a Documentação criada até o momento para o projeto integrador.

11. Volte para o **GitBash** e confirme se os arquivos do projeto estão aguardando para serem adicionados na **Branch Main**, com o comando **git status**

```
git status
```

12. Para atualizar o Repositório Local e enviar o conteúdo para o Repositório Remoto no Github, utilize a sequência de comandos abaixo:

```
git add .
```

```
git commit -m "Commit inicial"
```

```
git push -u origin main
```

13. Verifique se o repositório remoto no Github foi atualizado.

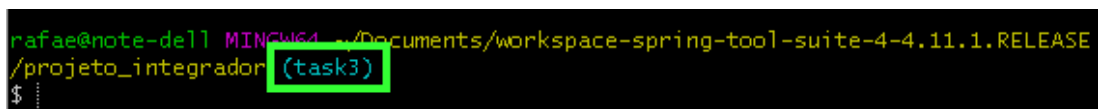
4.2) Criando o projeto Spring

Antes de criar o Projeto Spring, vamos criar a nossa primeira **Branch local**, onde simularemos a **Task3** do Projeto Integrador.

1. No GitBash, crie uma nova Branch com o nome **task3**

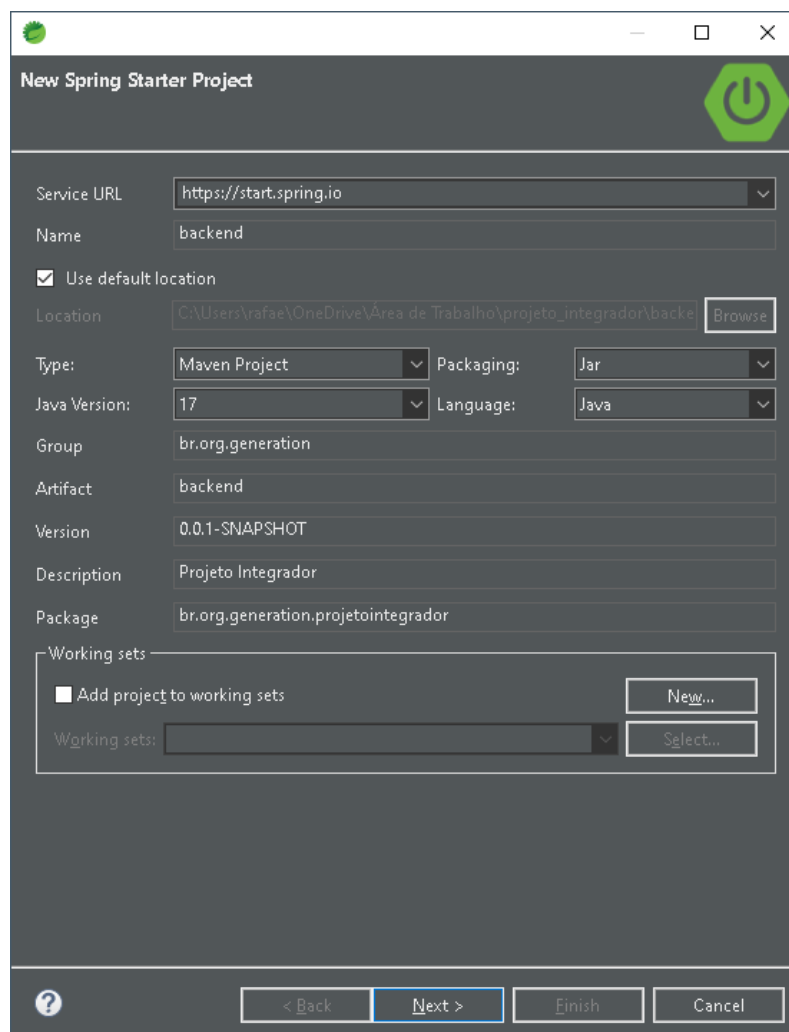
```
git checkout -b task3
```

2. Repare no prompt do Gitbash que ele mudou para a Branch Task3



```
rafae@note-dell MINGW64 ~/Documents/workspace-spring-tool-suite-4-4.11.1.RELEASE
/projeto_integrador (task3)
$
```

3. No STS, crie o projeto Spring, como mostra a figura abaixo:



New Spring Starter Project

Service URL:

Name:

☒ Use default location

Location:

Type: Packaging:

Java Version: Language:

Group:

Artifact:

Version:

Description:

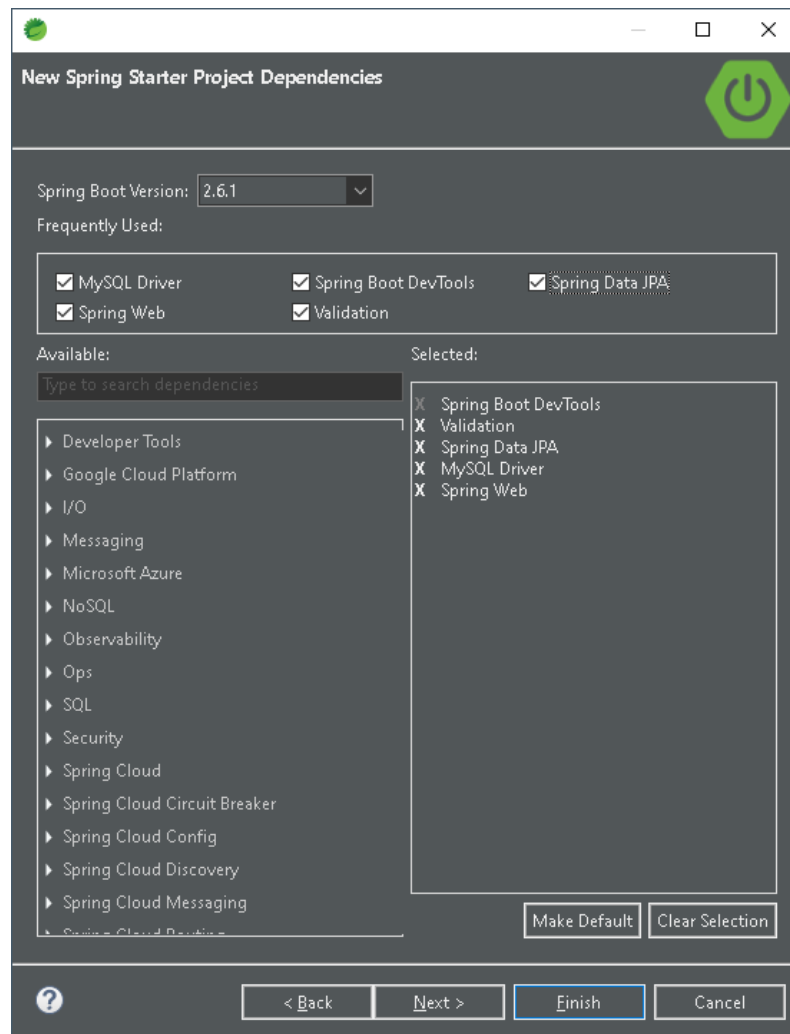
Package:

Working sets

☐ Add project to working sets

Working sets:

4. Adicione as Dependências do Projeto, como mostra a figura abaixo:



5. Configure o Banco de Dados no arquivo **application.properties**

```
spring.jpa.hibernate.ddl-auto=update
spring.jpa.database=mysql
spring.datasource.url=jdbc:mysql://localhost/db_projetointegrador?
createDatabaseIfNotExist=true&serverTimezone=America/Sao_Paulo&useSSL=false
spring.datasource.username=root
spring.datasource.password=root
```

```
spring.jpa.show-sql=true
```

```
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL8Dialect
```

```
spring.jackson.date-format=yyyy-MM-dd HH:mm:ss
```

```
spring.jackson.time-zone=Brazil/East
```

Observação: A terceira linha do arquivo **application.properties** está dividida em 2 linhas no pdf. No STS mantenha tudo em uma única linha.

6. Confirme se os arquivos foram salvos e estão prontos para serem “Commitados”

```
git status
```

7. Antes de atualizar a Branch task3, verifique se você está na Branch correta com o comando **git branch**

```
git branch
```

8. Na sequência vamos atualizar a **Branch task3** com os arquivos do projeto que foi criado

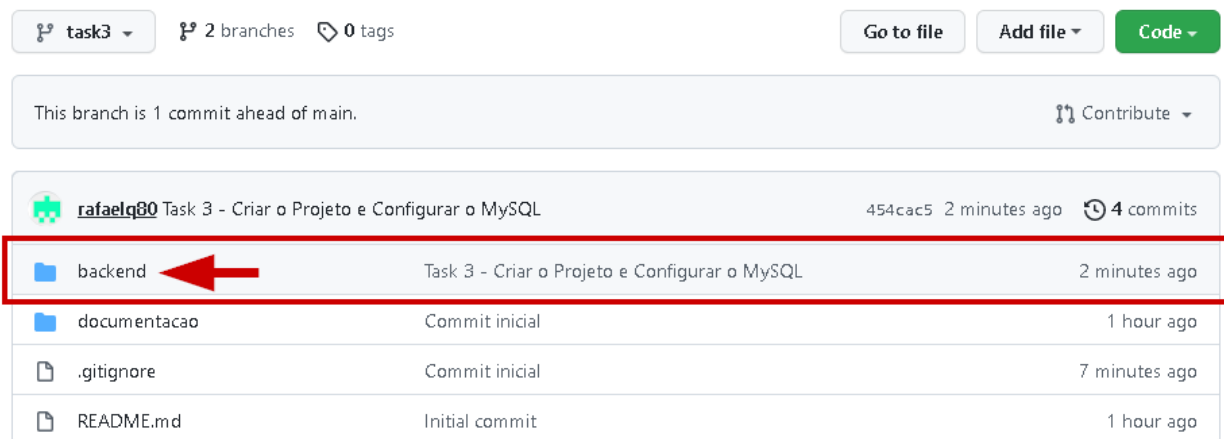
```
git add .
```

```
git commit -m "Criar o Projeto Spring"
```

```
git push -u origin task3
```

9. Verifique se o repositório remoto no Github foi atualizado e possui **2 Branches**.

Na Task3 temos o Projeto Spring na pasta Backend, como mostra a figura abaixo:



4.3) Finalizando a branch task3

Agora vamos finalizar a Branch task3, ou seja, após concluir a criação e a configuração do projeto, vamos unir o conteúdo da Branch Task3 com a Branch Main através do comando **git merge**.

1. Volte para a Branch Main

```
git checkout main
```

2. Atualize a Branch Main com o conteúdo da Branch task3

```
git merge task3
```

3. Envie as atualizações para o Repositório Remoto no Github

```
git push
```

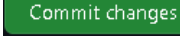
4. Observe que a **branch main** agora possui todo o conteúdo da **branch task3**

4.4) Atualizando o repositório local

Agora vamos simular como atualizar o repositório local com todas as alterações do Repositório remoto.

1. Acesse Repositório Remoto no Github

2. Faça alguma alteração no arquivo [Readme.md](#) na **branch main**

3. Faça o commit das alterações clicando no botão 

4. Volte para o Git Bash

5. Atualize o Repositório Local com o comando **git pull**

```
git pull
```

6. Se a alteração acima for realizada em uma **nova branch**, por outro usuário, o comando **git pull** falhará porque não encontrará a nova branch no repositório local. Utilize o comando abaixo para criar a nova branch no repositório local e vincular com a branch do repositório remoto.

```
git checkout --track -b nova_branch origin/nova_branch
```

** Observe que ambas a Branches devem possuir o mesmo nome.*

5) Desfazendo mudanças no repositório local

Agora vamos simular como desfazer alterações no repositório local.

1. Crie uma nova Branch com o nome **task4**

```
git checkout -b task4
```

2. Crie uma pasta chamada **db** em **src/main/resources**
3. Dentro da pasta, crie um arquivo chamado **projeto.sql**
4. Insira a instrução SQL abaixo no arquivo projeto.sql e salve o arquivo

```
select * from alunos;
```

5. Volte para o Git Bash
6. Adicione as alterações na **branch task4**

```
git add .
```

```
git commit -m "Criação da Task 4"
```

7. Confirme se os arquivos foram "Commitados"

```
git status
```

8. Agora, vamos desfazer este **último commit**

```
git reset HEAD~1
```

9. Observe que o commit foi desfeito, mas o arquivo **projeto.sql** continua existindo e está pronto para ser adicionado na branch task4

```
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   projeto.sql

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

10. Vamos refazer este último Commit

```
git add .
```

```
git commit -m "Criar a Task 4"
```

11. Agora, vamos desfazer este último commit e apagar a pasta db e todo o seu conteúdo

```
git reset --hard HEAD~1
```

12. Observe que além de desfazer o commit, o arquivo que você criou foi apagado (Observe no STS)

13. Refaça todas tarefas a partir do passo 2 até o passo 7

14. Após reafazer as tarefas, volte para a Branch Main

```
git checkout main
```

15. Atualize a Branch Main com as implementações realizadas na Branch **task4**

```
git merge task4
```

16. Envie as atualizações para o Github

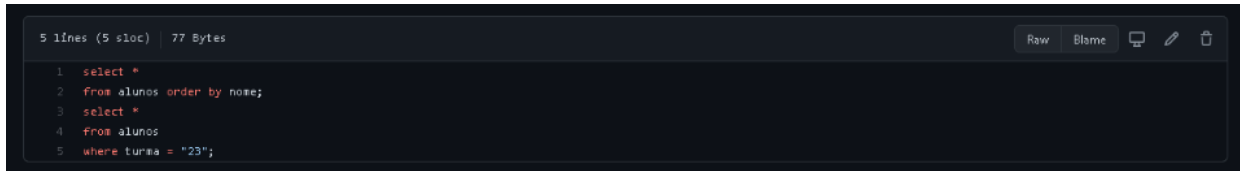
```
git push
```

Observe que a **Branch Task4** não foi enviada para o Github, porque diferente da Branch Task3 ela não recebeu o comando: `git push -u origin task4` que envia uma branch específica para o repositório remoto no Github.

6) Resolução de Conflitos

6.1) Criando o conflito no Github

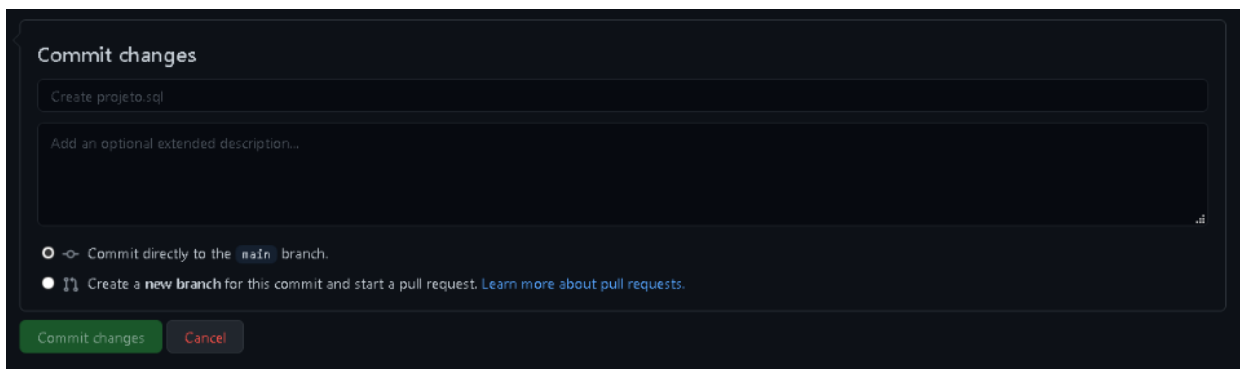
1. Altere o arquivo **projeto.sql** na branch main, no repositório remoto no Github conforme a imagem abaixo:



```
5 lines (5 sloc) | 77 Bytes
1 select *
2 from alunos order by nome;
3 select *
4 from alunos
5 where turma = "23";
```

2. Faça o commit das alterações clicando no botão

Commit changes



Commit changes

Create projeto.sql

Add an optional extended description...

☒ Commit directly to the `main` branch.

☐ Create a new branch for this commit and start a pull request. [Learn more about pull requests.](#)

Commit changes Cancel

6.2) Criando o Conflito no Git Local

1. Altere o arquivo **projeto.sql**, na **Branch Main**, no seu repositório local

```
select * from alunos order by id;
```

2. Adicione as alterações na Branch Main

```
git add .
```

```
git commit -m "Update do arquivo SQL"
```

3. Confirme se os arquivos foram "Commitados"

```
git status
```

4. Execute o comando **git pull** para atualizar o repositório local com as atualizações do repositório remoto

```
git pull
```

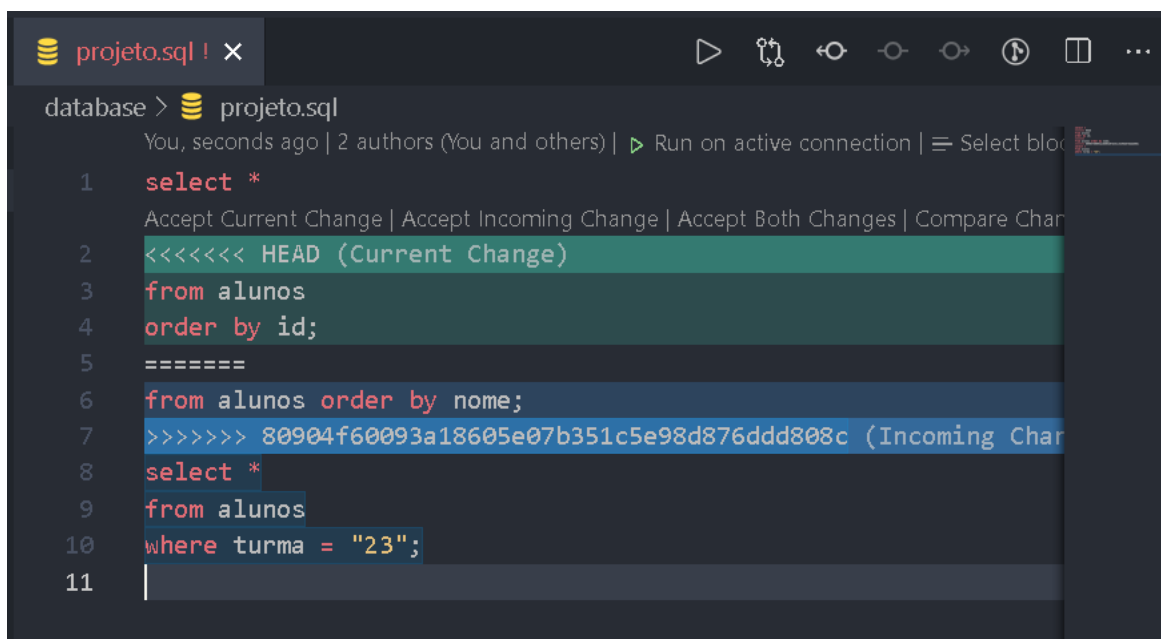
5. Observe que no final da Mensagem aparece a palavra **CONFLICT**

```
rafael@RFL_DELL MINGW64 ~/Desktop/github/projeto (main)
$ git pull
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (4/4), 780 bytes | 4.00 KiB/s, done.
From https://github.com/rafaelproinfo/projeto
   c549575..80904f6  main       -> origin/main
Auto-merging database/projeto.sql
CONFLICT (content): Merge conflict in database/projeto.sql
Automatic merge failed; fix conflicts and then commit the result.
```

6. Vamos abrir o arquivo no **VSCode** e verificar os conflitos

```
code .
```

7. Serão exibidas as diferenças encontradas nos dois arquivos: Local e Remoto, como mostra a figura abaixo:



8. O **VSCode** oferece **3 opções** para resolver o conflito e mais uma para ajudar na decisão:

- **Accept Current Change:** Mantém a mudança local
- **Accept Incoming Change:** Mantém a mudança remota
- **Accept Both Changes:** Mantém as 2 mudanças
- **Compare Changes:** Exibe os 2 arquivos lado a lado, para que você possa comparar

9. Clique em uma das opções e salve o arquivo para concluir

10. Adicione as alterações na Branch Main Local

```
git add .
```

11. Confirme se os arquivos foram adicionados

```
git status
```

12. Observe que o conflito foi resolvido

```
All conflicts fixed but you are still merging.  
(use "git commit" to conclude merge)
```

13. Faça o Commit das alterações

```
git commit -m "Resolução do Conflito"
```

14. Envie as atualizações para o Github

```
git push
```

***** DICA IMPORTANTE *****

Para evitar conflitos, crie o hábito de sempre atualizar o repositório local, com o conteúdo do repositório remoto, através da execução do comando **git pull** antes de começar a trabalhar no projeto.

7) Trabalhando com o Fork

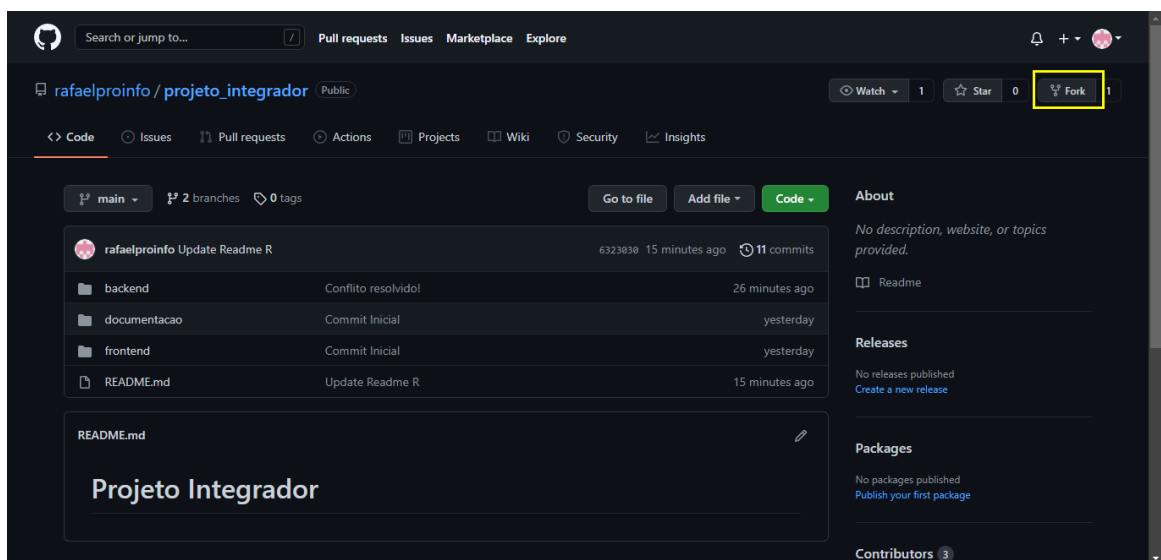
O **Fork** é um recurso do Github, que permite fazer a cópia integral de um repositório de uma pessoa desenvolvedora para o seu repositório no Github. Após você efetuar o Fork de uma repositório, você poderá editar os arquivos e posteriormente enviar para o repositório original as suas implementações e melhorias do projeto. Caso a pessoa desenvolvedora aceite as suas implementações, seus códigos passaram a fazer parte do repositório original. O processo de enviar contribuições para o repositório de origem do Fork é chamado de **Pull Request**.

Para manter o repositório atualizado com os últimos commits realizados no repositório origem, utilizamos o recurso chamado **Fetch upstream**.

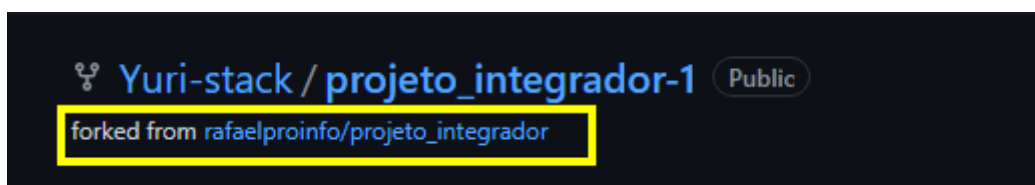
7.1) Adicionando um Repositório via Fork

1. Acesse o repositório que você deseja adicionar no seu repositório remoto

2. Clique no botão , como mostra a figura abaixo:



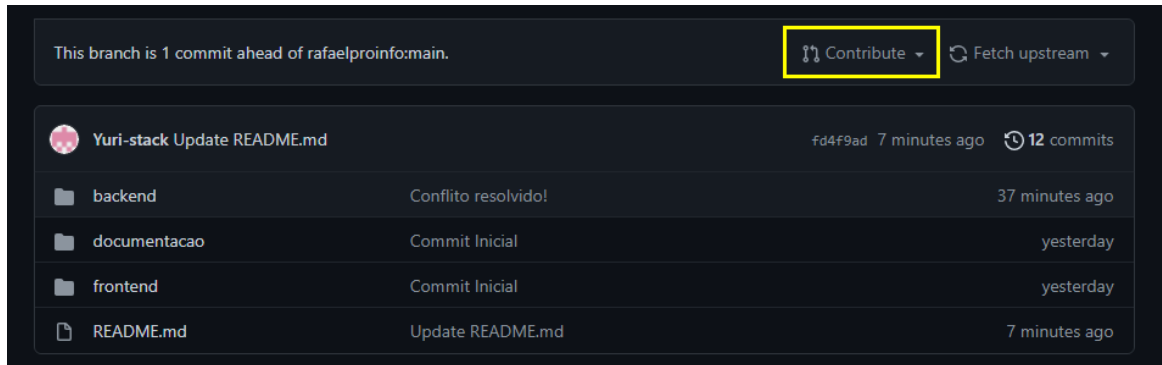
3. Após a conclusão do Fork, o Github redirecionará para o repositório copiado. Observe que abaixo do nome do repositório está indicado a origem do repositório.



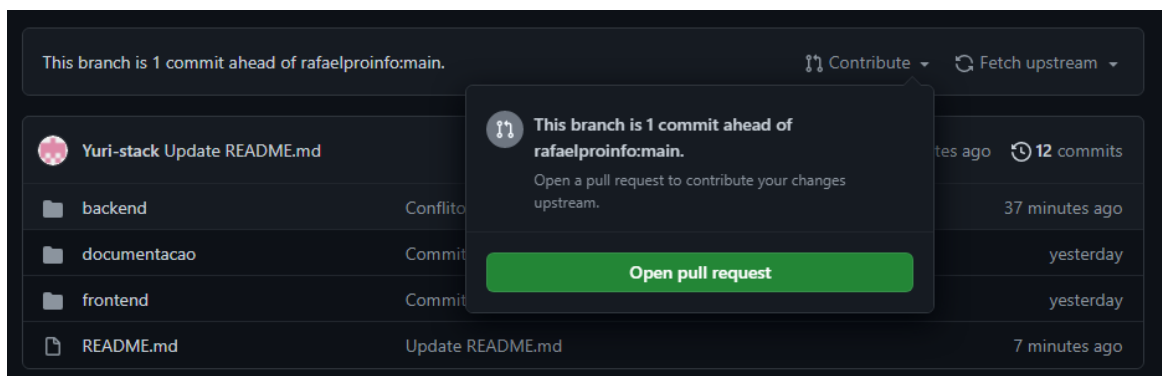
7.2) Pull Request

Agora vamos enviar uma contribuição para o repositório origem.

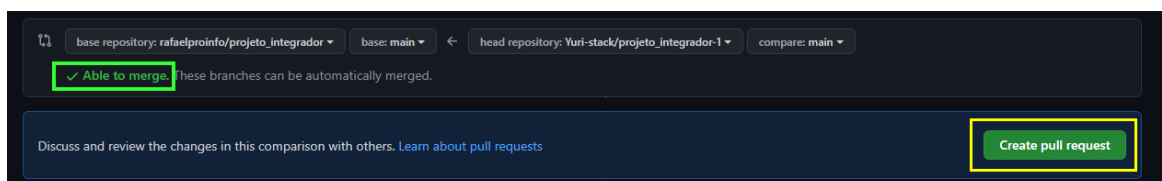
1. Vamos fazer uma alteração no arquivo [README.md](#)
2. Em seguida, clique no botão  para enviar uma contribuição



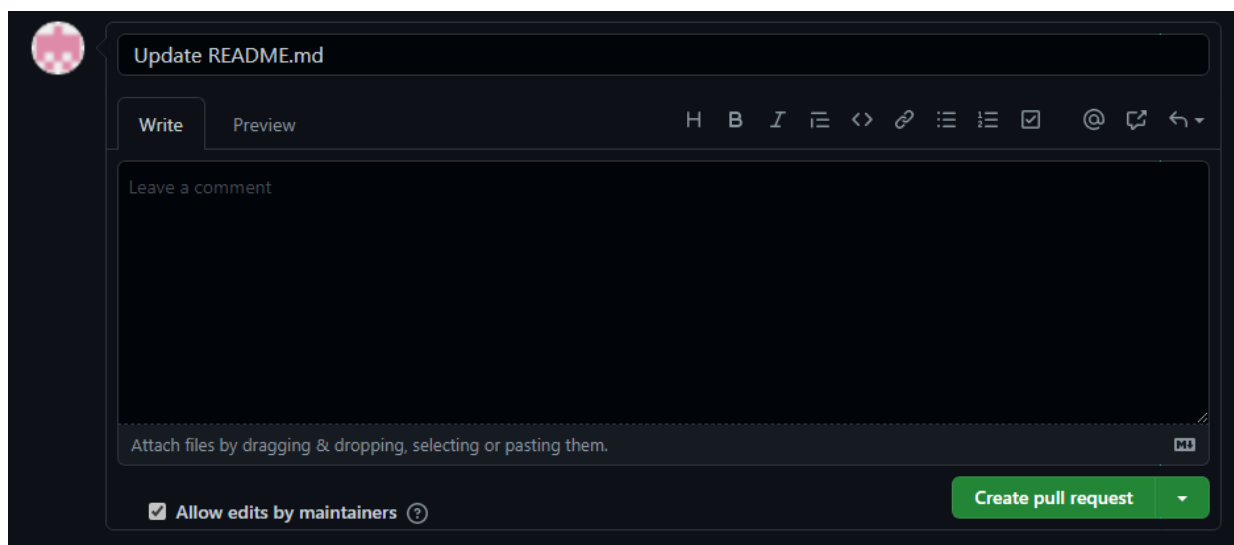
3. Será aberto um menu informando que existe um commit pronto para ser enviado. Clique no botão **Open pull request** para iniciar o processo de envio da contribuição.



4. O Github verificará se o seu commit está apto para ser enviado para o repositório origem do Fork. Se estiver tudo OK, será exibida a mensagem **Able to merge** (marcado em verde na figura abaixo). Para continuar, clique no botão **Create pull request**.

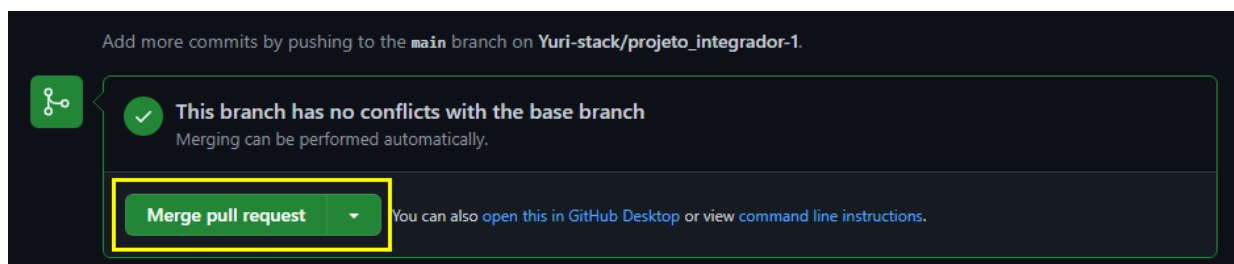


5. Será exibida uma janela para você documentar a alteração proposta. Preencha os dados e clique no botão **Create pull request** para concluir.



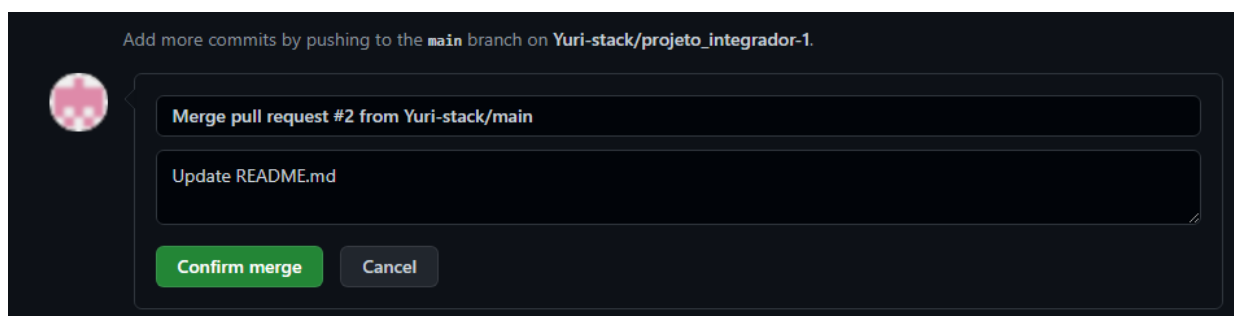
The screenshot shows the GitHub interface for creating a pull request. At the top, the title 'Update README.md' is entered. Below the title are tabs for 'Write' and 'Preview'. A rich text editor is present with a 'Leave a comment' placeholder. At the bottom, there is a checkbox labeled 'Allow edits by maintainers' which is checked, and a green button labeled 'Create pull request'.

6. Em nosso exemplo, a pessoa desenvolvedora que fez o Fork é também uma pessoa colaboradora do projeto. Nesta situação específica, após o envio do pull request, o Github exibirá a área de aprovação do pull request. Para aceitar o pull request, clique no botão **Merge pull request** para adicionar o novo código no projeto.



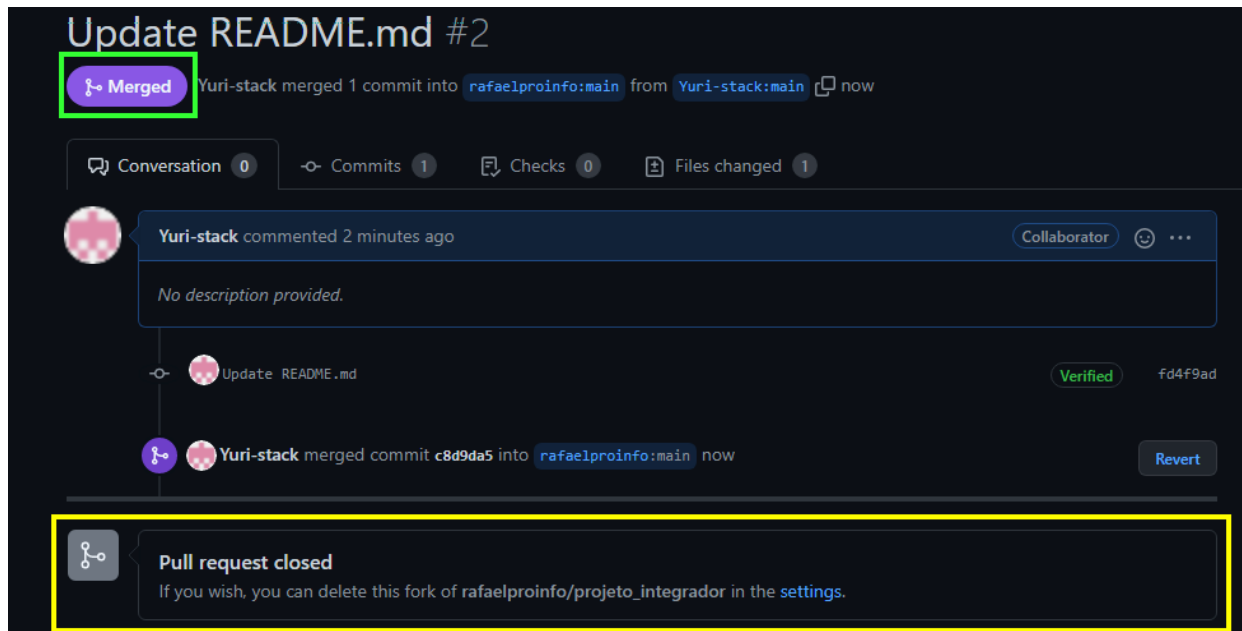
The screenshot shows the GitHub interface for merging a pull request. It features a green checkmark icon and the text 'This branch has no conflicts with the base branch'. Below this, a green button labeled 'Merge pull request' is highlighted with a yellow box. To the right of the button, there is a link to 'GitHub Desktop' and a link to 'command line instructions'.

7. Será exibida uma janela para você documentar a alteração proposta. Preencha os dados e clique no botão **Confirm merge** para concluir.



The screenshot shows the GitHub interface for confirming a merge. It features a green checkmark icon and the text 'Merge pull request #2 from Yuri-stack/main'. Below this, there is a text input field containing 'Update README.md'. At the bottom, there are two buttons: 'Confirm merge' (green) and 'Cancel' (gray).

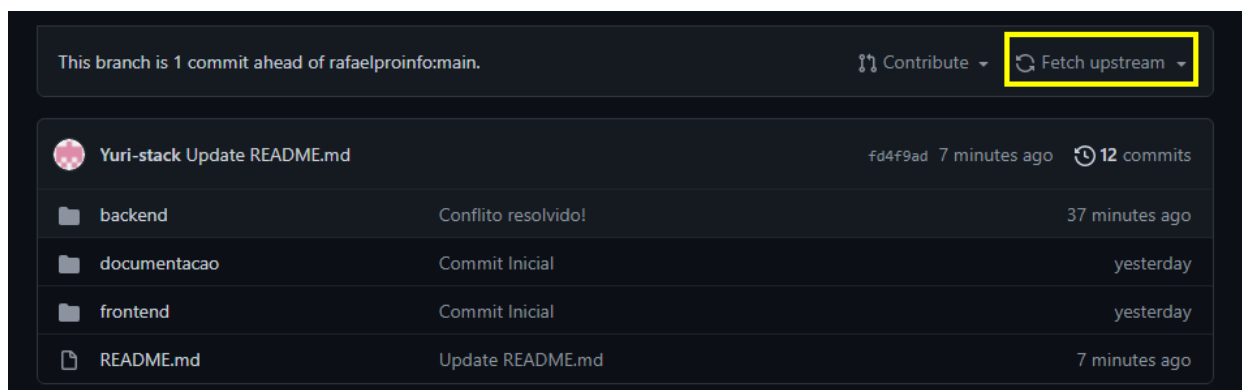
8. Observe que na parte superior da janela será exibida a mensagem **Merged**, indicando que as alterações foram adicionadas e na parte inferior será exibida a mensagem **Pull request closed**, indicando que a Pull request foi finalizada.



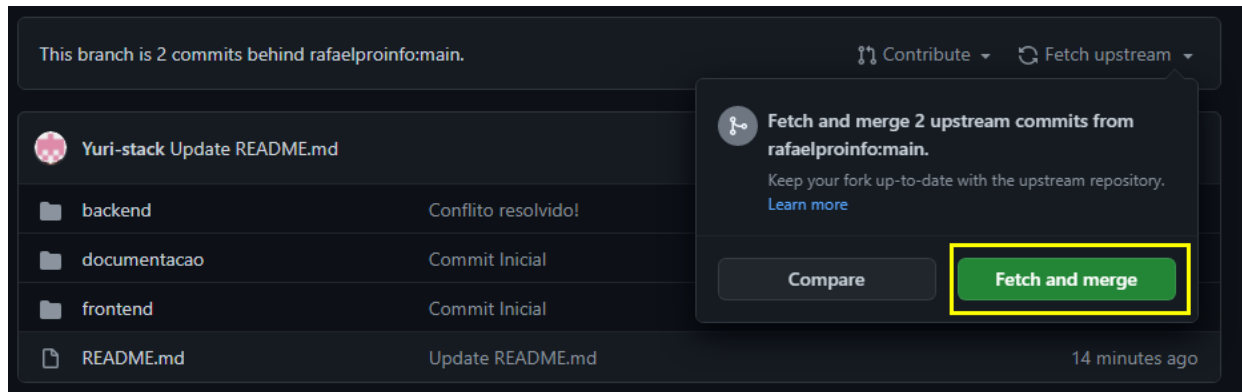
7.3) Fetch upstream

Agora vamos atualizar o repositório Fork com todos os commits recentes do repositório origem

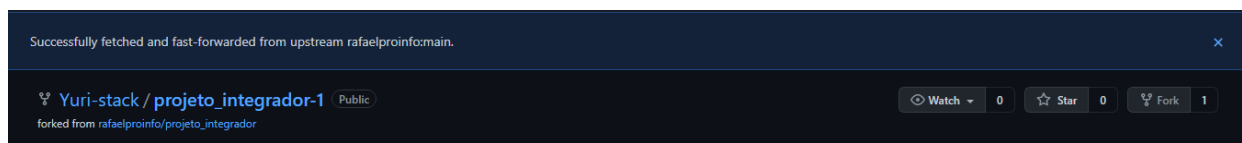
1. No repositório Fork, clique no botão  para iniciar a atualização.



2. Clique no botão **Fetch and merge** para concluir.



3. Ao concluir será exibida a mensagem abaixo:



8) Comandos úteis

1. Criar uma Branch no Github

```
git push origin task3:new-branch
```

2. Apagar uma Branch no github

```
git push origin:task3
```

3. Renomear uma Branch

```
git branch -m novonome
```

4. Apagar uma branch no repositório local

```
git branch -d nome_da_branch
```

5. Clonar repositório

```
git clone https://github.com/rafaelpinfo/projeto_integrador.git
```