

Fluxo de Trabalho no 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?

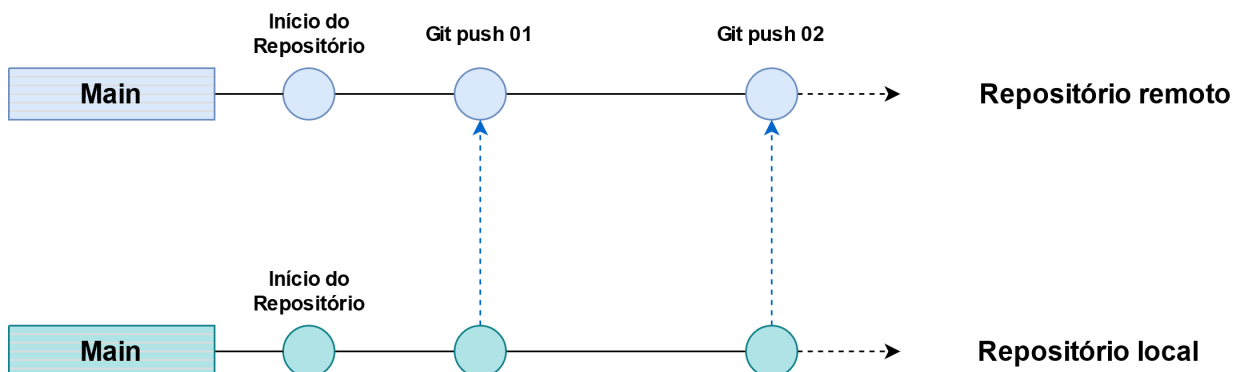
Ao avaliar um fluxo de trabalho para sua equipe, o mais importante é entender a cultura da equipe. O fluxo de trabalho deve melhorar a eficácia da equipe e não ser uma carga que limita a produtividade.

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

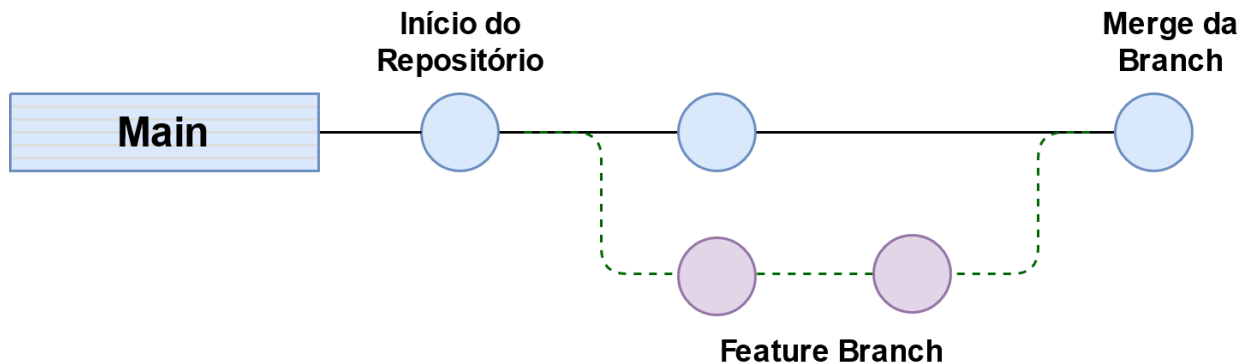
- Este fluxo de trabalho é dimensionado com o tamanho da equipe?
- É fácil desfazer erros com este fluxo de trabalho?
- Este fluxo de trabalho impõe alguma nova sobrecarga cognitiva desnecessária à equipe?

1.1) Tipos de Fluxo

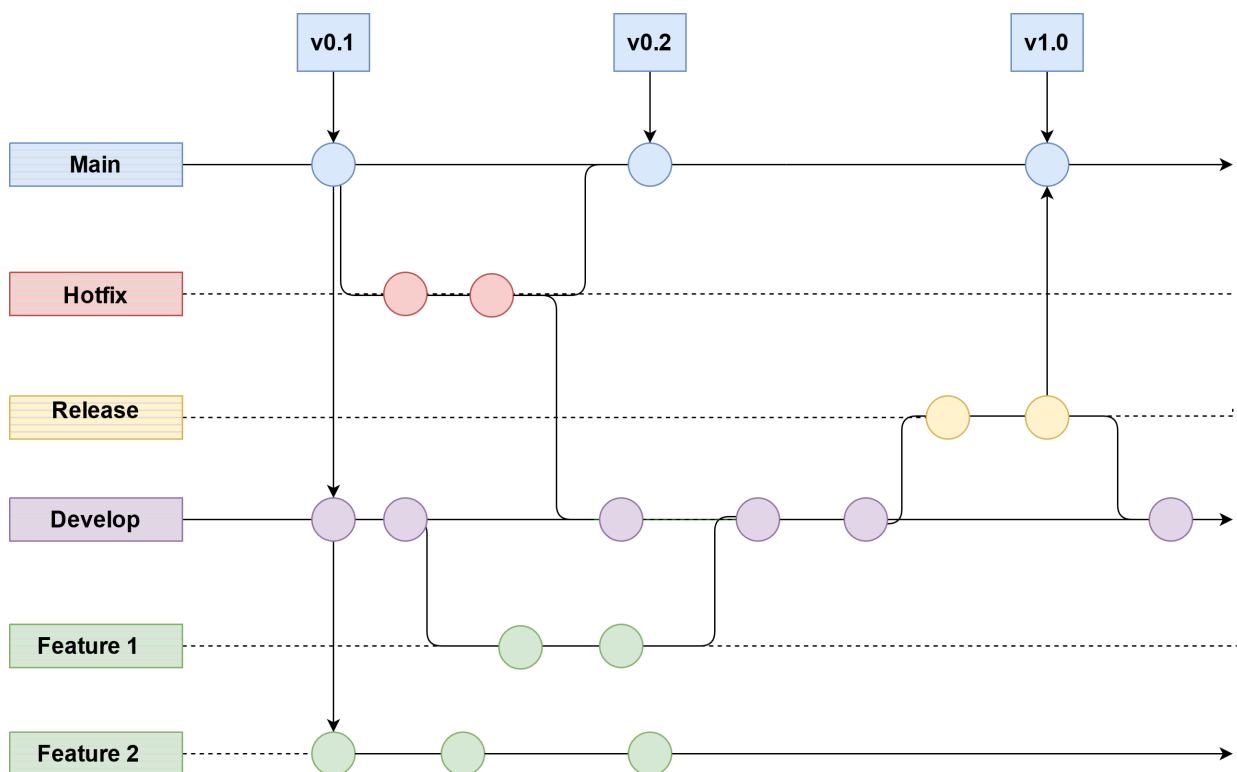
- **Fluxo de trabalho centralizado:** A ideia central por trás do Fluxo de trabalho centralizado é que todo o desenvolvimento de recursos deve ocorrer na branch main.



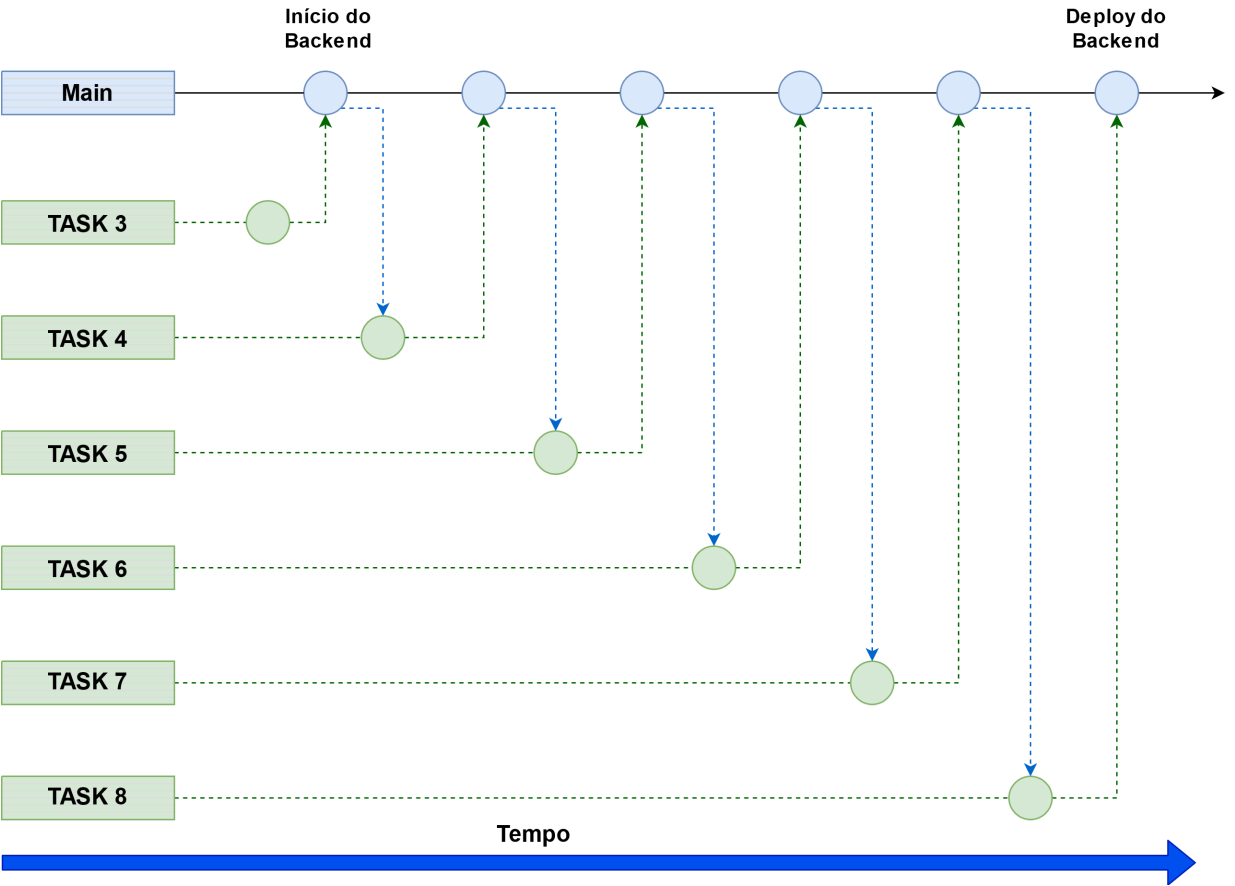
- **Fluxo de trabalho de ramificação de recurso:** A ideia central por trás do Fluxo de trabalho de ramificação de recursos é que cada feature deve ocorrer em uma Branch dedicada, que só é enviada para a Branch Main quando se torna parte de uma nova versão.



- **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 o projeto Integrador, sugerimos aos grupos que utilizem o **fluxo de trabalho de ramificação de recurso**, conforme o modelo abaixo, onde cada **Task** do Projeto Integrador será uma **Feature Branch**.



Fluxo proposto para o Bloco 02 - Back-end

Para manter o repositório organizado, sugerimos a seguinte estrutura de pastas:



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 etc)- Documentação do Backend (PDF do Swagger)- Documentação do Frontend
Backend	Projeto Spring completo
Frontend	Projeto Angular Completo

Nos próximos capítulos faremos uma breve revisão do Git, onde mostraremos como criar um repositório remoto, inserir os colaboradores e para finalizar faremos algumas simulações com situações comuns do trabalho em equipe no Github.

2) Configurando o Git Local

Para começar vamos instalar e configurar o VSCode para ser a IDE padrão do Git:

1. Instale o **VSCode** na sua máquina <https://code.visualstudio.com/download>, caso não esteja instalado
2. Abra o **Git Bash**
3. Configure o Editor padrão com o comando abaixo:

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

2.1) Criando repositório Central no Github

Vamos configurar o repositório Central no Github:

1. Defina a conta do Github onde ficará o Repositório Central do projeto
2. Em **Repositories**, clique no botão **New**.


3. Crie um **Repositório Público**, chamado **projeto_integrador**, e adicione o arquivo [Readme.MD](#). 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 *

 rafaelpinfo ▾


/

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

*O Nome do repositório será o nome do seu projeto.

Próximo passo: adicionar todos os membros do grupo no Repositório Central:

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

rafaelproinfo / projeto_integrador

Unwatch 1 Star 0 Fork 0

[Code](#) [Issues](#) [Pull requests](#) [Actions](#) [Projects](#) [Wiki](#) [Security](#) [Insights](#) [Settings](#)

Options

Manage access

Security & analysis

Branches

Who has access

PUBLIC REPOSITORY

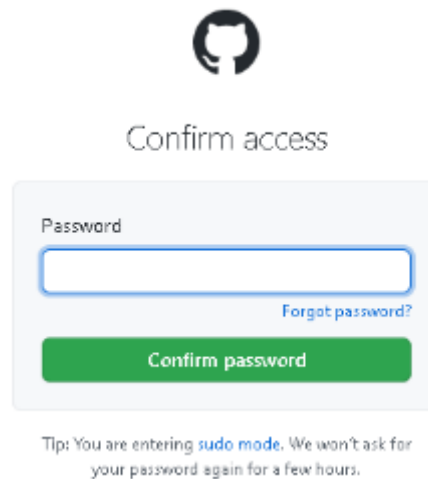
This repository is public and visible to anyone.

Manage

DIRECT ACCESS

0 collaborators have access to this repository. Only you can contribute to this repository.

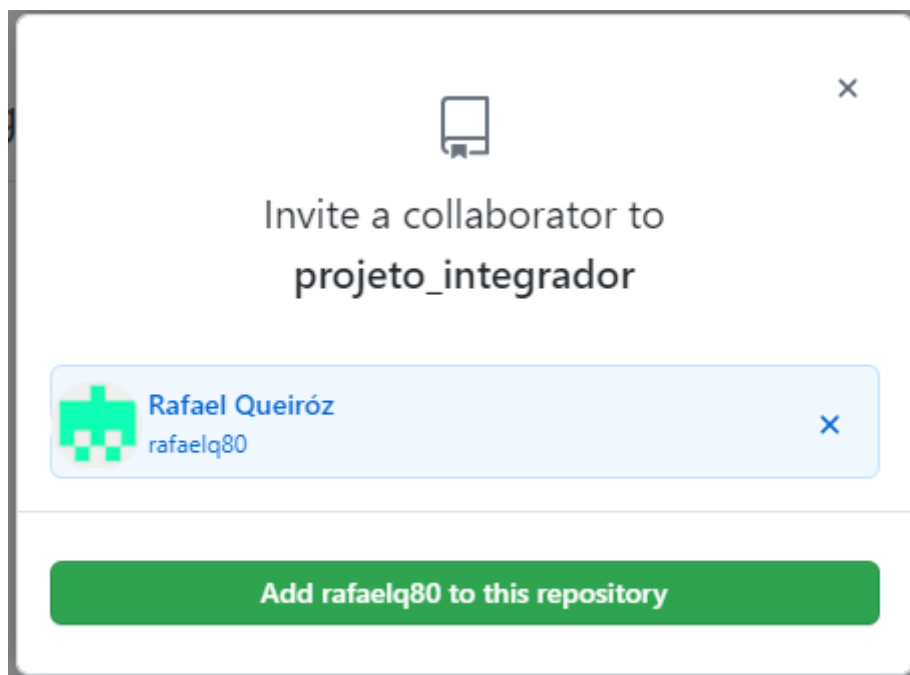
5. Digite a senha do Usuário do GitHub para continuar, caso seja solicitado



The image shows the GitHub 'Confirm access' dialog. At the top is the GitHub logo. Below it, the text 'Confirm access' is centered. There is a 'Password' label above a text input field. To the right of the input field is a link that says 'Forgot password?'. Below the input field is a green button labeled 'Confirm password'. At the bottom of the dialog, there is a tip: 'Tip: You are entering `sudo mode`. We won't ask for your password again for a few hours.'

6. Em **Manage access**, clique no botão **Invite a collaborator**

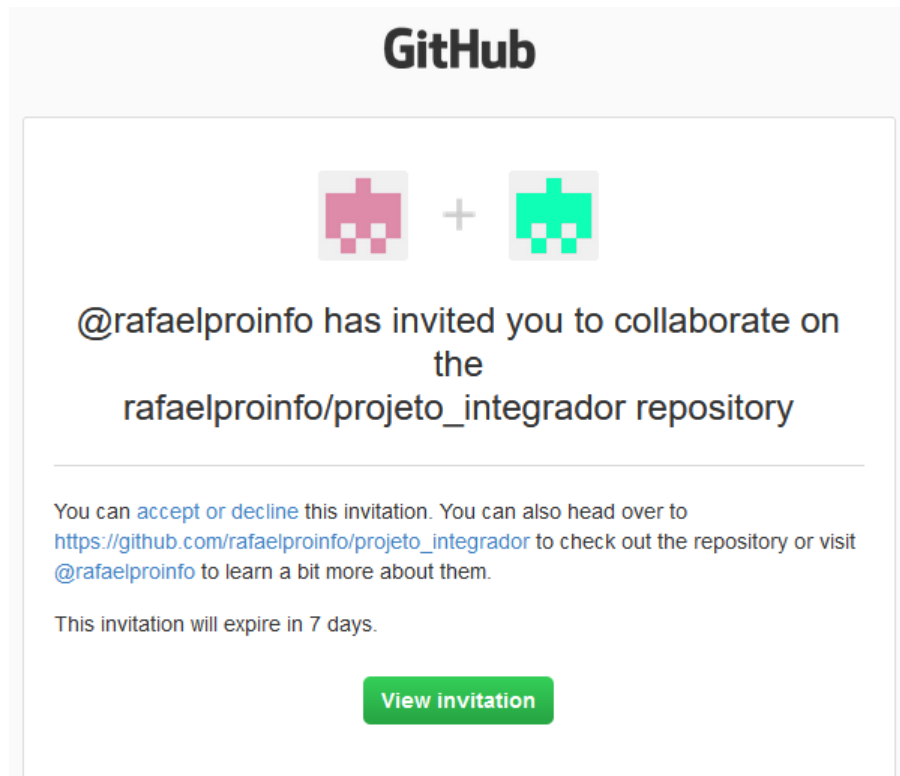
7. Localize o usuário que você deseja adicionar e clique no botão **Add <nome_usuario> to this repository**, como mostra a imagem abaixo:



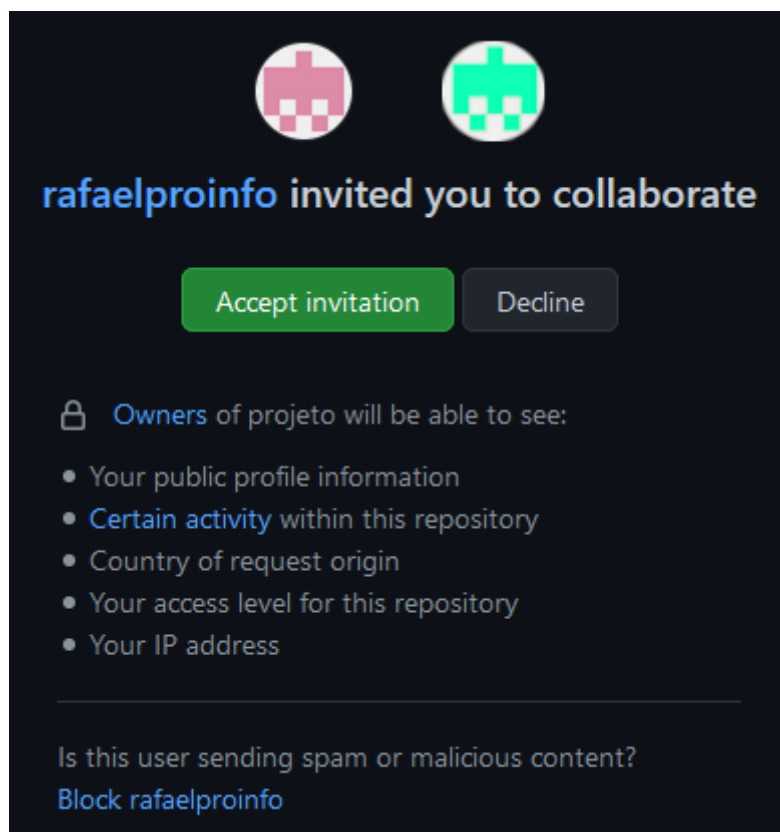
The image shows the GitHub 'Invite a collaborator to projeto_integrador' dialog. At the top is a laptop icon. Below it, the text 'Invite a collaborator to projeto_integrador' is centered. There is a search bar with a green robot icon on the left and a blue 'x' on the right. Below the search bar is a list of users. The first user is 'Rafael Queiróz' with the username 'rafaelq80'. Below the list is a green button labeled 'Add rafaellq80 to this repository'.

8. Repita os passos 6 e 7 para adicionar os demais membros do grupo

9. O Usuário convidado receberá um convite via e-mail. Clique no botão **View invitation**



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



11. O acesso ao repositório está liberado

2.2) 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:

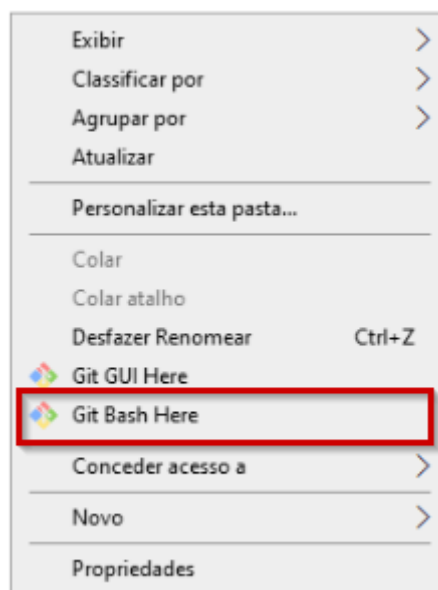


Observe que cada usuário adicionado possui o seu gráfico de colaboração no repositório.

3) Fluxo de Trabalho na máquina local

Vamos clonar o repositório Central do Github para a máquina local e na sequência vamos começar a trabalhar no projeto:

1. Localize no Windows Explorer a sua **Workspace** (Pasta onde o Spring Tools Suite - STS ou o Eclipse grava os projetos Spring)
 - Se você estiver usando o **STS** geralmente a pasta está localizada em:
c:\Usuarios\seuusuario\Documents\workspace-spring-tool-suite-4-4.11.0.RELEASE (a versão pode ser diferente).
 - Se você estiver utilizando o Eclipse, geralmente a pasta está localizada em:
c:\Usuarios\seuusuario\eclipse-workspace
2. Dentro da sua **Workspace**, clique com o botão direito do mouse e clique na opção: **Git Bash here**

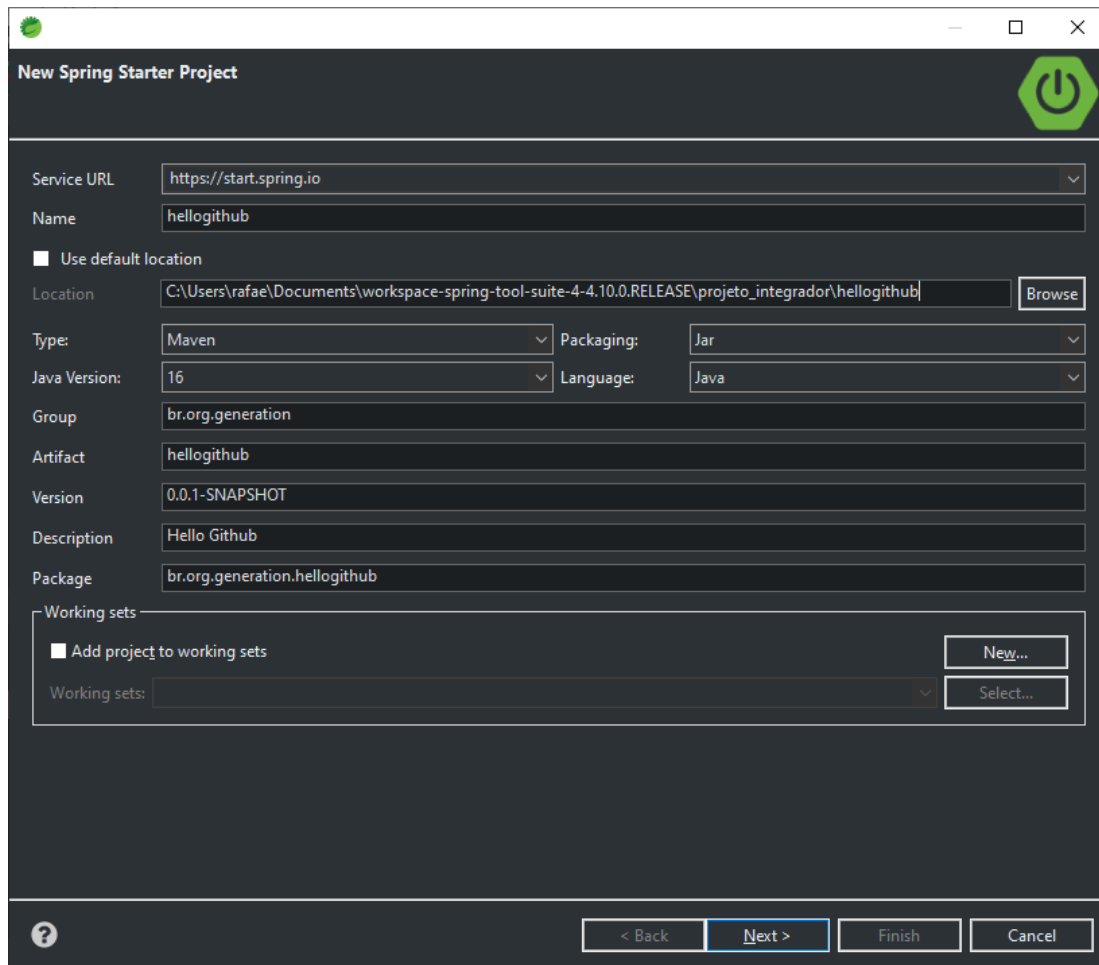


3. No Git Bash, clone o repositório do projeto para a sua **Workspace**

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

Agora vamos criar o projeto Spring no STS

4. No STS, crie o projeto Spring, ajustando o caminho da pasta, como mostra a figura abaixo:



The screenshot shows the 'New Spring Starter Project' dialog box. The 'Service URL' is set to 'https://start.spring.io'. The 'Name' is 'hellogithub'. The 'Location' is 'C:\Users\rafael\Documents\workspace-spring-tool-suite-4-4.10.0.RELEASE\projeto_integrador\hellogithub'. The 'Type' is 'Maven', 'Packaging' is 'Jar', 'Java Version' is '16', and 'Language' is 'Java'. The 'Group' is 'br.org.generation', 'Artifact' is 'hellogithub', 'Version' is '0.0.1-SNAPSHOT', 'Description' is 'Hello Github', and 'Package' is 'br.org.generation.hellogithub'. There are checkboxes for 'Use default location' and 'Add project to working sets'. At the bottom, there are buttons for '< Back', 'Next >', 'Finish', and 'Cancel'.

5. Os demais itens do projeto permanecem semelhantes aos projetos anteriores

6. Crie a Classe **HelloController**, igual fizemos no Projeto **Hello World**, teste o Projeto e Salve.

7. No GitBash, abra a pasta do **projeto_integrador** na sua **Workspace**

```
cd projeto_integrador
```

8. Confirme se os arquivos do projeto estão aguardando para serem adicionados na branch, com o comando **git status**

```
git status
```

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

nothing added to commit but untracked files present (use "git add" to track)
```

9. Adicione todas as alterações na Branch **main** com o comando **git add**

```
git add .
```

10. Confirme se os arquivos foram adicionados

```
git status
```

```
rafae@note-dell MINGW64 ~/Documents/workspace-spring-tool-suite-4-4.10.0.RELEASE/projeto_integrador (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   hellogithub/.gitignore
    new file:   hellogithub/.mvn/wrapper/MavenWrapperDownloader.java
    new file:   hellogithub/.mvn/wrapper/maven-wrapper.jar
    new file:   hellogithub/.mvn/wrapper/maven-wrapper.properties
    new file:   hellogithub/mvnw
    new file:   hellogithub/mvnw.cmd
    new file:   hellogithub/pom.xml
    new file:   hellogithub/src/main/java/br/org/generation/hellogithub/HellogithubApplication.java
    new file:   hellogithub/src/main/java/br/org/generation/hellogithub/controller/HelloController.java
    new file:   hellogithub/src/main/resources/application.properties
    new file:   hellogithub/src/test/java/br/org/generation/hellogithub/HellogithubApplicationTests.java
```

11. Execute o comando **Commit** para efetivar as alterações na Branch Main

```
git commit -m "Upload do Projeto"
```

12. Confirme se os arquivos foram "**Commitados**" com o comando **git status**

```
git status
```

```
rafae@note-dell MINGW64 ~/Documents/workspace-spring-tool-suite-4-4.10.0.RELEASE/projeto_integrador (main)
$ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

13. Envie o projeto para o Repositório do Github com o comando **git push**

```
git push
```

3.1) Criando uma Branch

Agora vamos criar a nossa primeira Branch

1. No GitBash, crie uma nova Branch com o nome **task1**

```
git checkout -b task1
```

2. Edite a classe **HelloController** e altere a mensagem que será exibida na tela.

3. Confirme se os arquivos foram salvos

```
git status
```

4. Antes de atualizar a Branch task1, verifique se você está na Branch correta

```
git branch
```

5. Adicione as alterações na Branch task1

```
git add .
```

6. Confirme se os arquivos foram adicionados

```
git status
```

7. Faça o Commit das alterações

```
git commit -m "Criação da Task 1"
```

8. Confirme se os arquivos foram "Commitados"

```
git status
```

9. Envie a Branch task1 para o Github

```
git push -u origin task1
```

Próximo passo: vamos simular a **task1** finalizada e pronta para ser transferida para a Branch Main

10. Volte para a Branch Main

```
git checkout main
```

11. Atualize a Branch Main com as implementações realizadas na Branch task1 com o comando **git merge**

```
git merge task1
```

24. Envie as atualizações para o Github

```
git push
```

3.2) Atualizando o repositório local

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

1. Acesse Repositório Central, 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 remota.

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

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

4) Desfazendo mudanças no repositório local

Agora vamos simular como desfazer alterações no repositório local.

1. Crie uma nova Branch com o nome **task2**

```
git checkout -b task2
```

2. Crie uma pasta chamada **db** em **src/main/resources**

3. Dentro da pasta, crie um arquivo chamado **projeto.sql**

4. Insira uma nova consulta SQL e salve o arquivo

```
select * from tb_hellogithub;
```

5. Volte para o Git Bash

6. Adicione as alterações na Branch task2

```
git add .
```

```
git commit -m "Criação da Task 2"
```

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 task2

```
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 "Criação da Task 2"
```

11. Agora, vamos desfazer este último commit e apagar o arquivo projeto.sql

```
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 do passo 2 até o passo 7

14. Volte para a Branch Main

```
git checkout main
```

15. Atualize a Branch Main com as implementações realizadas na Branch task2

```
git merge task2
```

16. Envie as atualizações para o Github

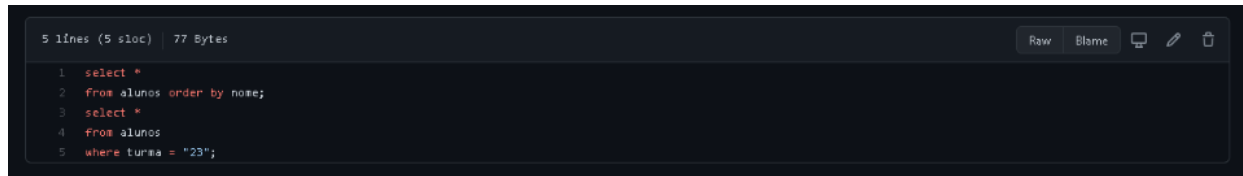
```
git push
```

Observe que a Branch Task2 não foi enviada para o Github, porque diferente da Task1 ela não recebeu o comando: `git push -u origin task2` que envia para o Github.

5) Resolução de Conflitos

5.1) Criando o conflito no Github

1. Altere o arquivo projeto.sql na branch main, no repositório remoto (github)

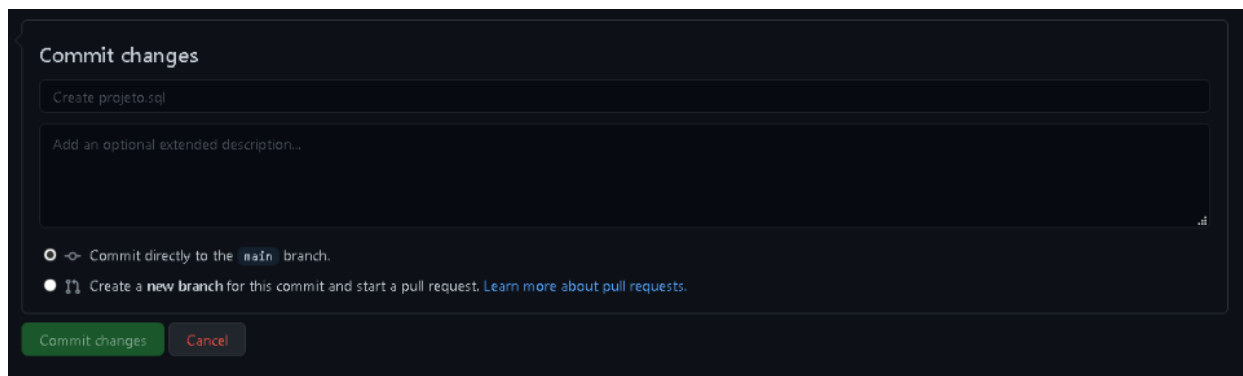


A screenshot of a code editor interface. At the top, it says "5 lines (5 sloc) | 77 Bytes". On the right, there are buttons for "Raw", "Blame", and icons for a monitor, edit, and delete. The code is as follows:

```
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



A screenshot of the "Commit changes" dialog box. It has a title bar "Commit changes". Below it is a text input field containing "Create projeto.sql". Underneath is a larger text area with the placeholder "Add an optional extended description...". At the bottom, there are two radio button options: "Commit directly to the main branch." (which is selected) and "Create a new branch for this commit and start a pull request. Learn more about pull requests." Below these options are two buttons: "Commit changes" (highlighted in green) and "Cancel".

5.2) Criando o Conflito no Git Local

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

```
git add .
```

3. Confirme se os arquivos foram adicionados

```
git status
```

4. Faça o Commit das alterações

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


5. Confirme se os arquivos foram “Commitados”

```
git status
```

6. Execute um Git Pull para atualizar o repositório local com as atualizações do repositório remoto

```
git pull
```

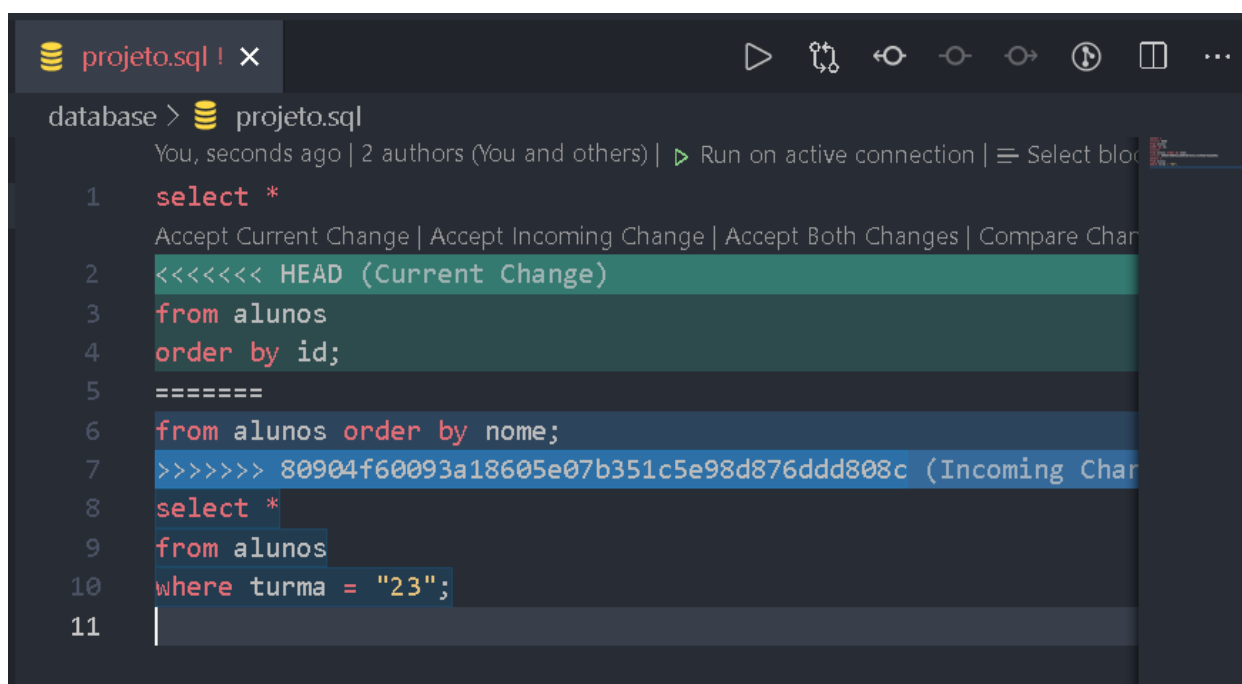
7. Observe que no final da Mensagem aparece a palavra **CONFLICT**

```
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.
```

8. Vamos abrir o arquivo no VSCode e verificar os conflitos

```
code .
```

9. Serão exibidas as diferenças encontradas nos dois arquivos: Local e Remoto



10. 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.

11. Clique na opção que melhor se encaixa com o código e salve o arquivo

12. Adicione as alterações na Branch Main Local

```
git add .
```

13. Confirme se os arquivos foram adicionados

```
git status
```

14. Observe que o conflito foi resolvido

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

15. Faça o Commit das alterações

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

16. Confirme se os arquivos foram "Commitados"

```
git status
```

17. Envie as atualizações para o Github

```
git push
```

18. Apague a branch Task2

```
git branch -d task2
```

*** DICA IMPORTANTE ***

Para evitar conflitos, uma boa prática é tentar fazer com que todos na equipe sempre estejam atualizados com o repositório remoto através da execução do comando **git pull** regularmente no Repositório Local.

6) 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. Renomear a Branch Master para Main

```
git branch -m master main
```

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

