Gerência de Configuração e Mudança

Projeto Integrador 2024 - Equipe 2º Ano

Versão 3.0

Sumário

Repositório do Projeto:
<https://github.com/ProjetoIntegradorUFV2024/Equipe-2Ano>

Finalidade do Documento

Este documento tem como objetivo organizar as diretrizes para o gerenciamento de configuração e mudança da equipe do 2º ano do Projeto Integrador. Aqui são estabelecidas padronizações a serem seguidas por todos os membros da equipe, incluindo a criação de branches, resolução de bugs e versionamento, abrangendo não apenas o código, mas todos os artefatos do projeto. O objetivo é adotar práticas recomendadas pela Engenharia de Software para gestão de projetos, versionamento e configuração, garantindo consistência e qualidade no desenvolvimento.

Git Flow

Para organizar e gerenciar melhor o fluxo de trabalho dos desenvolvedores, utilizaremos o Git Flow, um modelo de ramificação que define regras para criação e fusão de branches. O Git Flow permitirá gerenciar as fases de desenvolvimento e manutenção de código estável. O modelo garantirá que diferentes etapas do ciclo de vida (como implementação e implantação) estejam bem definidas e controladas, reduzindo a probabilidade de conflitos e garantindo melhor estabilidade do código.

Tutorial de comandos git

https://learngitbranching.js.org/?locale=pt_BR

Sugestão de comandos a serem estudados

```
git clone
```

```
git pull

git push

git add

git commit -m “mensagem do commit”

git checkout

git checkout -b <nome-da-branch>

git status

git merge

git fetch
```

Guia de Instalação do Git Flow - Linux:

*Os comandos são executados no terminal

Caso não tenha o git instalado no seu computador, execute o seguinte comando:

```
sudo apt install git
```

Para instalar a extensão do git flow, execute o seguinte comando:

```
sudo apt install git-flow
```

Branches:

main: A branch principal do projeto. Ela é utilizada **apenas para versões estáveis e deploy final**. Todas as entregas oficiais para produção são realizadas a partir dessa branch. **Observação: Nenhum desenvolvimento direto ocorre nessa branch.**

develop: A branch de integração principal onde ocorre a fusão das features finalizadas. Esta branch representa o estado mais atualizado do desenvolvimento e sempre deve estar funcional. **Observação: Toda feature deve ser analisada antes do merge com a**

develop, ou seja, o desenvolvedor deve solicitar um pull request para que um sênior verifique e aceite/recuse a fusão das branchs.

feature: A branch é criada a partir da develop para **desenvolvimento de novas funcionalidades**. Cada tarefa ou melhoria recebe sua própria branch. Ao finalizar o desenvolvimento de uma feature, ela é revisada e, posteriormente, meclada de volta à develop.

release: As branches de **release** são criadas a partir da **develop** quando o projeto está pronto para uma versão estável. Servem para preparar a versão de lançamento, onde são feitos ajustes finais e correções de bugs menores. Após a finalização, a release é mesclada tanto na **main** quanto na **develop**.

hotfix: Essas branches são criadas a partir da **main** para corrigir problemas críticos que precisam de **resolução imediata** em produção.

****Observação:** Caso uma branch não esteja relacionada diretamente ao fluxo do Git Flow, seu nome deve ser claro e descritivo sobre o conteúdo ou a finalidade. Use termos concisos que identifiquem o objetivo da branch, como "ajuste-doc", "melhoria-login" ou "corrige-erro-login".

Criação/Padronização das branchs - Git Flow:

DEV SENIORS

Branch-Flow/nome-da-branch

Ex.:

- feature/login
- feature/classe-usuario

DEV JUNIORS

disciplinas:

- BD: banco de dados
- POO: Programção Orientada à Objetos

Branch/disciplina/nome-do-dev

Ex.:

- feature/BD/Matheus
- feature/POO/João

Exemplos de criação de branchs usando o git flow (Os comandos são realizados no terminal vscode):

1. Criar uma feature para login:

```
git flow feature start login
```

2. Publicar branch sem finalizar para o GitHub:

```
git flow feature publish login
```

3. Finalizar branch e integrar na devolop local:

```
git flow feature finish login
```

Padrão de Commit:

Uma boa mensagem de commit deve ser clara, concisa e descrever a alteração de maneira informativa.

***Referência a Issues**: Ao realizar commits relacionados a uma issue, inclua o identificador da issue precedido pelo símbolo # na mensagem de commit (ex.: fix #2: corrige bug). Isso facilita o rastreamento e vincula o commit à issue correspondente no repositório, ajudando na organização e histórico de resolução de problemas.

Resumo Curto:

- Deve ser uma frase curta que descreva a alteração.
- Deve iniciar com um verbo no imperativo, como "Corrige", "Adiciona", "Remove", "Atualiza", etc.

Identificador: descrição curta (**simples e objetiva**)

git commit -m "<identificador> #<id-issue>: mensagem"

Ex.:

- feat #10: adiciona tela de cadastro de usuario
- task #5: adiciona classe aluno
- fix #2: corrige bug (**exemplo para caso o commit esteja relacionado à uma issue com id 2**)

Temos os seguintes identificadores para as mensagens dos commits:

- **feat:** Novo recurso ou funcionalidade.
- **task:** Indicado para tarefas menores e específicas que não estão diretamente relacionadas a funcionalidade. Pode ser utilizado para adicionar classes simples, criar tabelas de bancos de dados, ou ajustes menores na estrutura do código.
- **fix:** Correção de bugs.
- **refactor:** Refatoração de código.
- **test:** Adição ou atualização de testes

Passos Iniciais:

Passo 1: Clonar o repositório do GitHub para o Git remoto do computador

Exemplo:

- Clique no repositório no github > Clique no botão “<>Code” e copie o link http
- Com a pasta do seu computador (em que você irá trabalhar) aberta no vscode ou cmd, acesse o terminal e execute o comando **git clone**:

```
git clone https://github.com/teste.git
```

Passo 2: Iniciar o Git Flow

Acesse o repositório clonado no **vscode** e execute o comando:

```
git flow init
```

Observação: Quando o comando for executado, irá ser configurado as branches que iremos utilizar no projeto. Basta confirmar todas com a tecla “Enter” com os respectivos nomes sugeridos seguindo o passo a passo do próprio git flow.

Passo a Passo do Fluxo de Trabalho para os Desenvolvedores Juniors (POO e BD):

Antes de iniciar uma feature, **verifique que está na branch develop** com o comando:

```
git branch
```

O comando mostra todas as branchs do projeto localmente e a que você estiver no momento.

Se precisar mudar de branch, use o comando:

```
git checkout <nome-branch>
```

Verifique também se a branch develop está atualizada com o comando:

```
git pull origin develop
```

1. **Criar uma nova feature (disciplina(POO ou BD)):**

   Primeiramente, crie uma branch da sua feature usando o Git Flow:

```
git flow feature start <disciplina-relacionada>/nome-do-dev
```

Isso cria uma nova branch chamada feature/<disciplina-relacionada/nome-do-dev a partir da develop

2. **Fazer as modificações necessárias:**

   Agora, concentre-se em sua tarefa. As tarefas devem ser enviadas na pasta ***Juniors***. Consulte a *issue* relacionada no GitHub e observe sua numeração (ao lado do símbolo '#'). Após realizar as alterações, faça o *commit* localmente.

```
git add.
```

```
git commit -m "task #<id-issue>: <mensagem>"
```

3. **Publicar branch para revisão:**

   Para que a sua branch seja revisada por um desenvolvedor sênior, você precisa **publicar** a branch no repositório remoto.

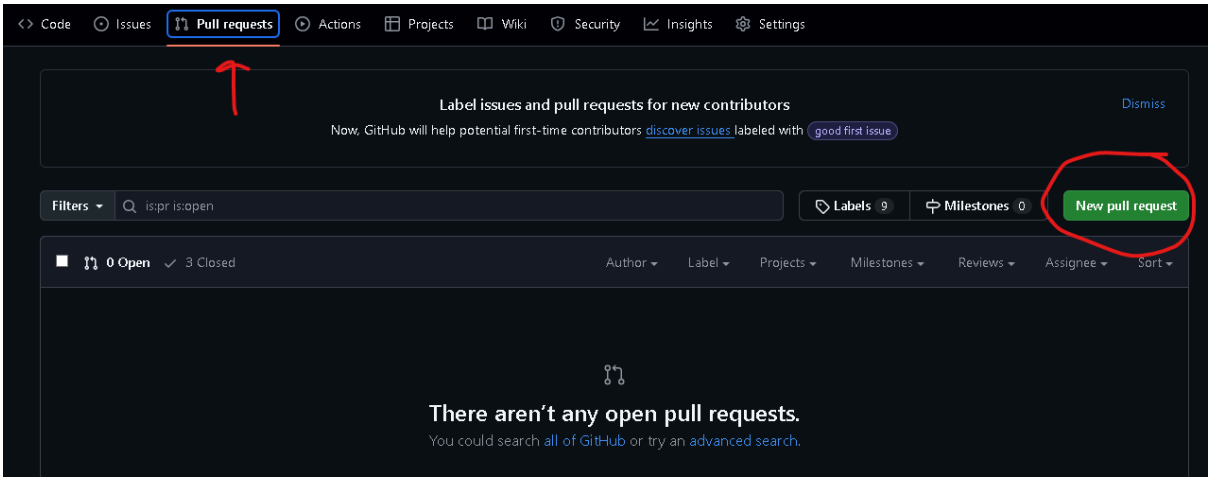
```
git flow feature publish <disciplina-relacionada>/nome-do-dev
```

   Esse comando faz o **push** da sua branch `feature/<disciplina-relacionada>/nome-do-dev` para o GitHub, permitindo que seja feito o **pull request**.

4. **Pull Request no GitHub:**

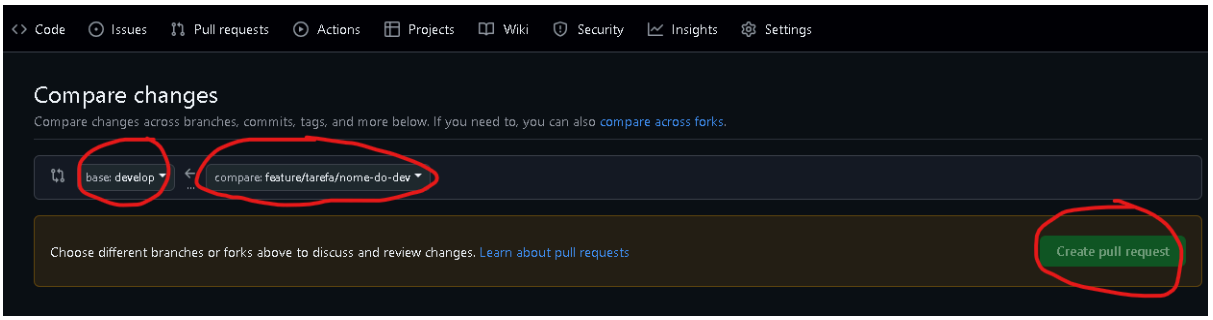
   No GitHub, você vai na sua branch `feature/<disciplina-relacionada>/nome-do-dev` e abre um **Pull Request** para a branch `develop`seguindo o passo a passo a seguir:

   4.1. Clique na aba “Pull Requests” e depois no botão “New pull request”



   4.2. **Preste muita atenção** nessa parte. Logo abaixo de “Compare changes” irá aparecer dois campos para serem preenchidos: “base” e “compare”; No campo **“compare”** deverá ser selecionada a branch nova que foi criada para a tarefa (”feature/<disciplina-relacionada>/nome-do-dev”) ; No campo “base” deverá ser selecionada

a branch “develop”, onde será feito o merge.



4.3. Marque nos comentários os seguintes desenvolvedores sêniors para que eles sejam notificados e avaliem os pull requests:

@CoderAugusto

@Jon-Lemmon

@SabrinaBruni28

5. **Deletar branch local:**

Após a revisão do Pull Request, o desenvolvedor sênior decidirá se o merge será aceito ou recusado. Em seguida, a branch deve ser removida do seu computador (localmente). Para isso, retorne ao VSCode, mude para a branch develop (ou outra branch que não seja a da feature) e exclua a branch com os seguintes comandos:

```
git checkout develop
```

```
git branch -D feature/<disciplina-relacionada>/nome-do-dev
```

6. **Atualizar sua branch develop local:**

Para garantir que sua branch **develop** local esteja atualizada com todas as mudanças recentes (inclusive as feitas por outros desenvolvedores), você deve fazer um **pull** da branch `develop` remota.

```
git checkout develop
git pull origin develop
```

Isso garante que você tenha as últimas alterações na branch `develop` no seu repositório local.

Fluxo de Trabalho para os Desenvolvedores SÊNIORS (ESOF e Arquitetura de Software):

Este fluxo destaca as principais funções e etapas de trabalho dos desenvolvedores que cursam as disciplinas de Engenharia de Software e Arquitetura de Software. Para mais detalhes, consulte a seção de Versionamento.

ATENÇÃO: Antes de iniciar os passos a seguir, verifique se existe uma issue relacionada à sua tarefa, se não é necessário criar no GitHub.

A. Desenvolver uma feature

Antes de iniciar uma feature, **verifique que está na branch develop** com o comando:

```
git branch
```

O comando mostra todas as branchs do projeto localmente e a que você estiver no momento.

Verifique também se a branch develop está atualizada com o comando:

```
git pull origin develop
```

1. **Criar uma nova feature (”tarefa”):**

   Primeiramente, crie uma branch da sua feature usando o Git Flow:

```
git flow feature start tarefa
```

   Isso cria uma nova branch chamada feature/tarefa a partir da develop.

2. **Fazer as modificações necessárias:**

   Agora você trabalha na sua tarefa. Após realizar as alterações , você faz o commit localmente.

```
git add.
git commit -m "feat #<id-issue>: realiza tarefa"
```

3. **Publicar branch para revisão:**

   Para que a sua branch seja revisada por um desenvolvedor sênior, você precisa **publicar** a branch no repositório remoto.

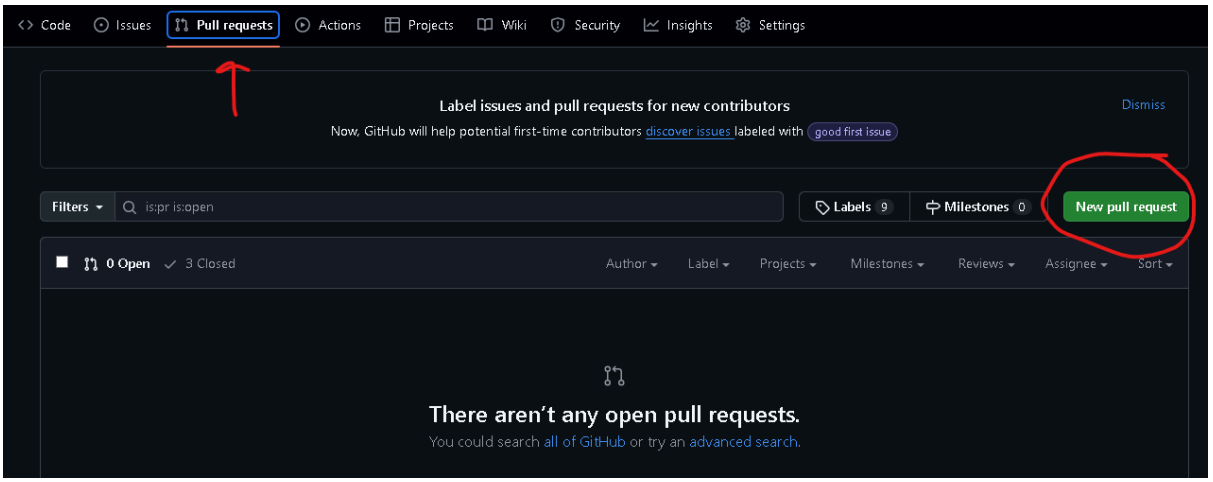
```
git flow feature publish tarefa
```

   Esse comando faz o **push** da sua branch `feature/tarefa` para o GitHub, permitindo que seja feito o **pull request**.

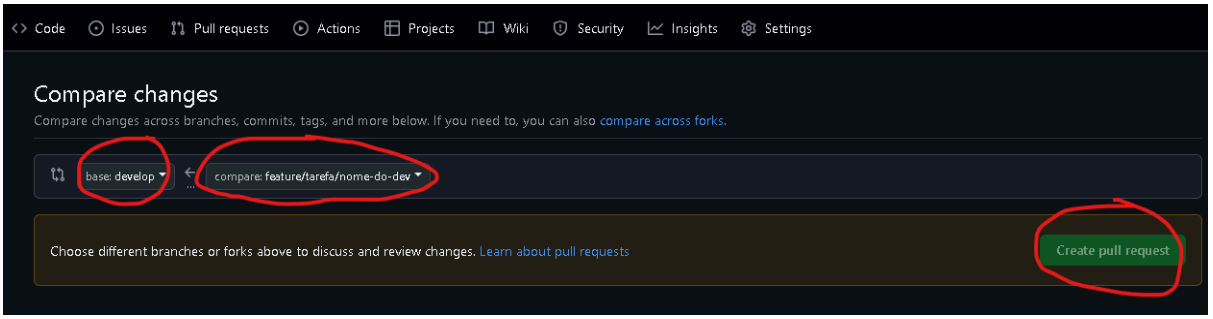
4. **Pull Request no GitHub:**

   No GitHub, você vai na sua branch `feature/tarefa` e abre um **Pull Request** para a branch `develop` seguindo o passo a passo a seguir:

   4.1. Clique na aba “Pull Requests” e depois no botão “New pull request”



   4.2. **Preste muita atenção** nessa parte. Logo abaixo de “Compare changes” irá aparecer dois campos para serem preenchidos: “base” e “compare”; No campo **“compare”** deverá ser selecionada a branch nova que foi criada para a tarefa (”feature/tarefa/nome-do-dev”) ; No campo “base” deverá ser selecionada a branch “develop”, onde será feito o merge.



**Observação: Marque nos comentários da pull request algum desenvolvedor que ache necessário para que o mesmo seja notificado

5. **Finalizar a feature após o merge no GitHub:**

Após a revisão do Pull Request, se for revisado e aceito no merge, o desenvolver deve finalizar a branch local no VsCode com o comando:

```
git flow feature finish tarefa
```

6. **Atualizar sua branch develop local:**

Para garantir que sua branch **develop** local esteja atualizada com todas as mudanças recentes (inclusive as feitas por outros desenvolvedores), você deve fazer um **pull** da branch `develop` remota.

```
git checkout develop
 git pull origin develop
```

Isso garante que você tenha as últimas alterações na branch `develop` no seu repositório local.

B. Contribuir com uma Feature

Quando uma nova feature já foi criada por um desenvolvedor e está disponível no repositório do GitHub, outro desenvolvedor que desejar contribuir com essa feature deve seguir os passos abaixo:

Atualizar o repositório local:

```
git fetch
```

Acessar a branch desejada:

```
git checkout <nome-branch>
```

Atualizar a branch local com as últimas mudanças enviadas no GitHub:

```
git pull origin <nome-branch>
```

A partir disso, o desenvolvedor poderá criar outra branch a partir da feature para fazer suas tarefas e enviar como pull request.

C. Pull Request

- **Revisar Pull Requests dos Desenvolvedores Juniors**

Será necessário revisar todos os pull requests enviados pelos desenvolvedores juniores. Apenas um pull request, referente à melhor tarefa, deverá ser aceito. Caso um pull request seja recusado, você pode adicionar um comentário explicando o motivo, se achar necessário. Após aceitar ou recusar o pull request, é importante excluir a branch correspondente no GitHub para manter a organização. Ao tomar a decisão, o GitHub exibirá uma opção para deletar a branch automaticamente.

Versionamento de Documentações, Códigos e Protótipos

Para centralizar e facilitar o gerenciamento do projeto, foi criada a branch "arquivos" no repositório, onde serão armazenados todos os documentos, protótipos e sprints relacionados.

a) Adições/Atualizações de Novos Documentos/Arquivos

Para adicionar novos documentos/arquivos ou adicionar uma nova versão, o desenvolvedor responsável pela documentação **deve criar uma issue relacionada** e **enviar como Pull Request para a branch de "arquivos"** solicitando alguma revisão, caso necessária, dos analistas de qualidade ou outro integrante que julgue necessário.

Se o Pull Request for recusado, deve ser mencionado o problema nos comentários do pull request e criar uma issue relacionada.

Se for algum arquivo que não necessite de uma análise como dependência, como cronogramas criados pelo líder, pode ser feito diretamente sem necessidade de uma Pull Request.

b) Protótipos (Figma)

O designer de jogo, responsável pela criação das telas no Figma, deve criar uma branch com o novo protótipo da sprint e enviar um pull request para a branch "arquivos", atribuindo-o aos analistas de qualidade. Após a análise, se o protótipo atender aos

requisitos, os analistas devem aprovar o pull request. Caso contrário, devem justificar a recusa nos comentários do pull request e abrir uma issue detalhando o que precisa ser ajustado.

c) Issue Tracking

Principais situações para abertura de issues:

- **Gerar uma tarefa:** Deve ser criado uma issue para cada tarefa que for realizada por um integrante da equipe sênior, tudo de acordo com o clickup.

- **Recusa de Pull Requests:** Se um pull request for recusado, a pessoa responsável pela recusa deve explicar o motivo e abrir uma issue detalhando as alterações necessárias, atribuindo-a ao desenvolvedor ou gerente responsável.

- **Relatórios de Bugs ou Pendências:** Desenvolvedores sêniors devem abrir issues para reportar bugs ou pendências de sprint e fechá-las quando resolvidas.

- **Solicitações de Revisão de Código:** Desenvolvedores que precisarem de revisão para seus pull requests devem abrir issues, especialmente para revisões de código feitas por desenvolvedores juniores.

- **Modificações em Documentação:** Issues podem ser abertas por gerentes, analistas ou desenvolvedores para solicitar alterações em casos de uso, modelos de classe ou qualquer outro artefato computacional do projeto.