G Guideline de Branch e Commit

Project overview
Details
Activity
Releases
Repository
Issues 0
Merge Requests 0
CI / CD
Operations
Packages & Registries
Analytics

The Auto DevOps pipeline has been enabled and will be used if no alternative CI configuration file is found.
Container registry is not enabled on this GitLab instance. Ask an administrator to enable it in order for Auto DevOps to work.

Settings More information

blockchain › Guideline de Branch e Commit

Guideline de Branch e Commit

Project ID: 62

Star 0 Fork 0

22 Commits **1** Branch **0** Tags **1.5 MB** Files **1.5 MB** Storage

Guideline com os modelos de branch e commit que o time de Pesquisa Blockchain deve utilizar nas PoCs e projetos internos.

master guideline-de-branch-e-commit / History Find file Web IDE Clone

Adicionar instruções de como manter o histórico de commits linear
Roberto Karpinski authored 1 month ago

14d07798

README Auto DevOps enabled Add LICENSE Add CHANGELOG Add CONTRIBUTING Add Kubernetes cluster

Name	Last commit	Last update
Imagens	imagens para o slide 6 da apresentação do guideline	2 months ago
Dicas_de_como_gerenciar_modificações_d...	Update Dicas_de_como_gerenciar_modificações_dos_arq...	1 month ago
Dicas_de_como_usar_o_git_stash.md	Incluir as dicas de como usar o git stash	4 months ago
Dicas_de_resolucao_de_conflitos.md	Incluir as dicas para resolução de conflitos	4 months ago
Exemplo_de_aplicacao_do_Guideline.zip	Adicionar referência e projeto de exemplo	4 months ago
Guia-config.md	Update Guia-config.md	2 months ago
README.md	Adicionar instruções de como manter o histórico de com...	1 month ago
Simulacao_de_um_projeto_exemplo.zip	Adicionar referência e projeto de simulação	4 months ago

README.md

Guideline de branching e commit

Este Guideline explica o modelo utilizado pela equipe de Pesquisa e Desenvolvimento em seus projetos.

A estrutura de branchs dos projetos é composta de branchs principais como a **master** e a **develop**, e branchs de apoio como a **feature**, **release** e a **hotfix**.

Já o modelo de commits utiliza as **boas práticas** recomendadas pela comunidade de desenvolvedores de plataformas como GitHub.

Outras referências

Dicas de resolução de conflitos

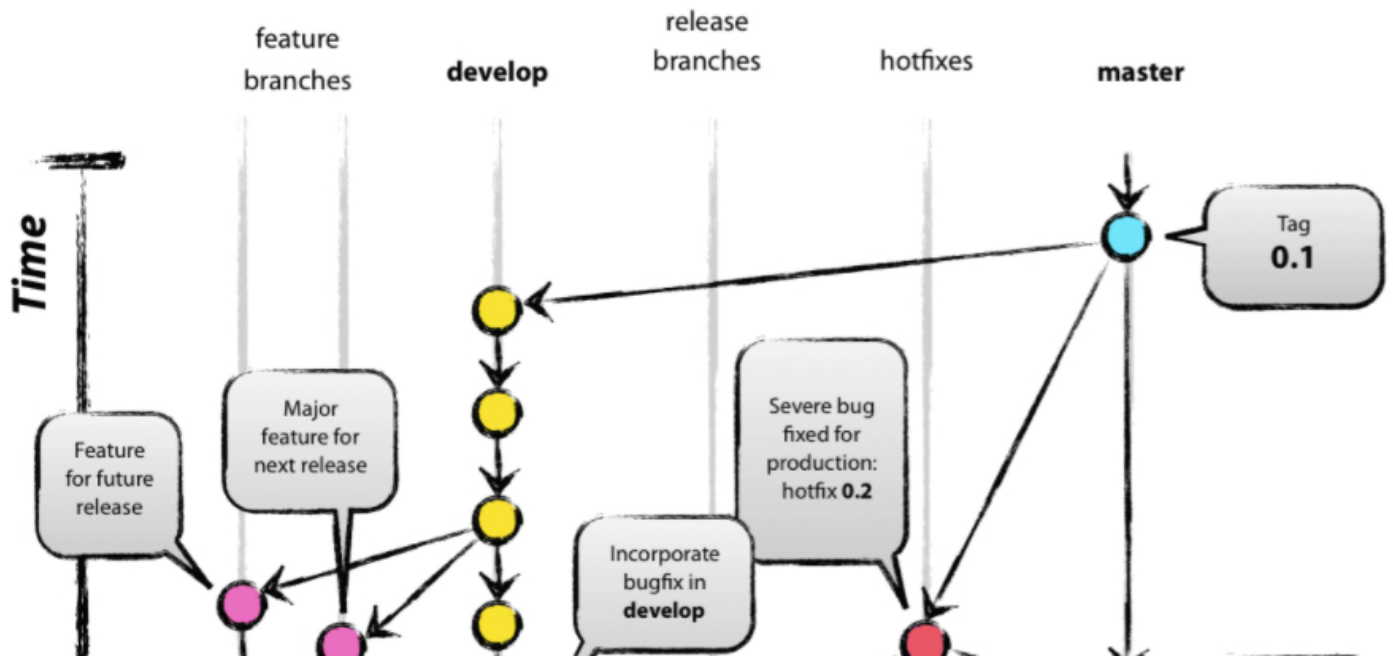
Dicas de como usar o git stash

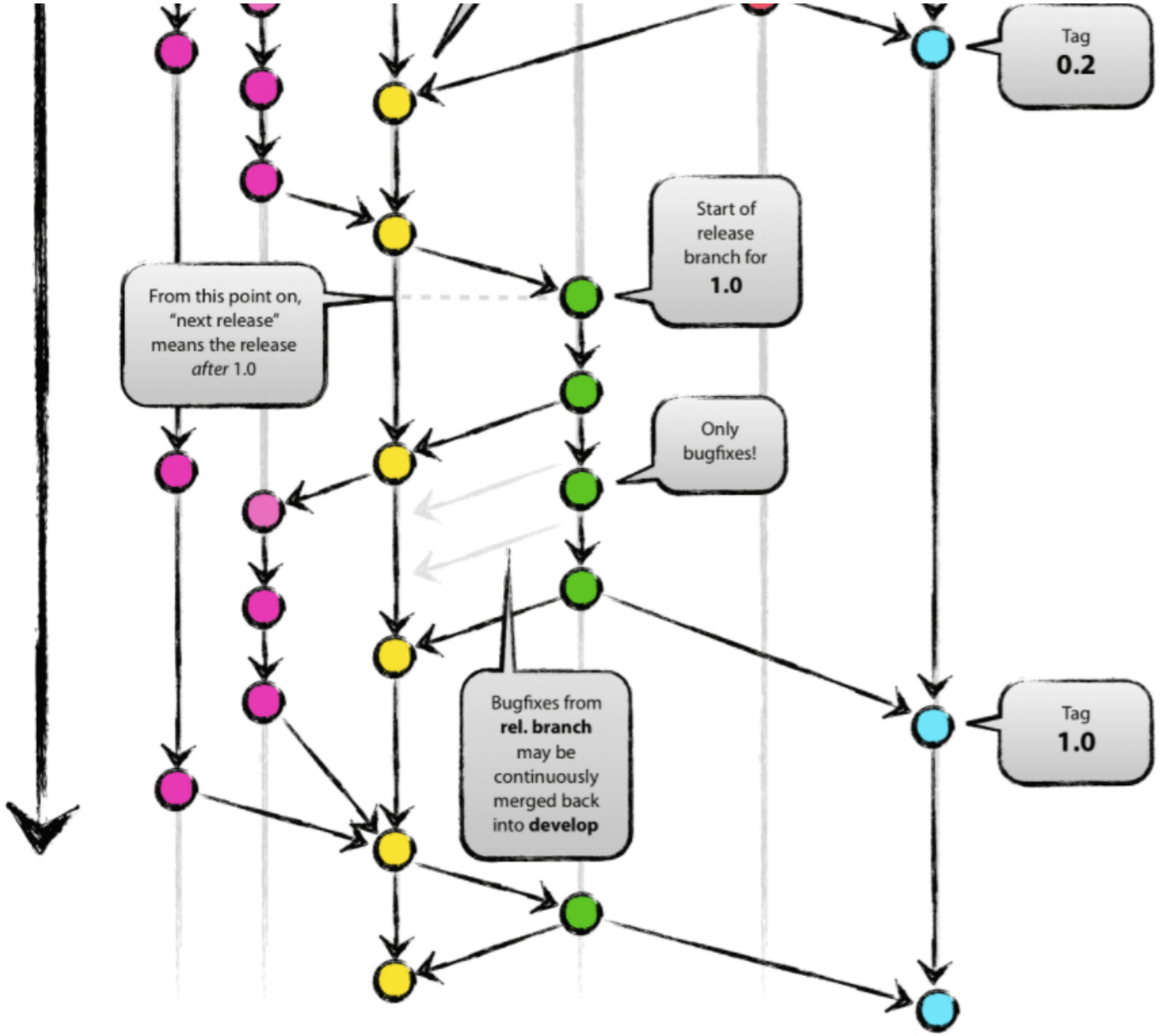
Simulação de um projeto exemplo

Exemplo de aplicação do Guideline

Modelo de Branching

Utilizamos como referência o modelo de branching proposto por Vincent Driessen: A successful Git branching model

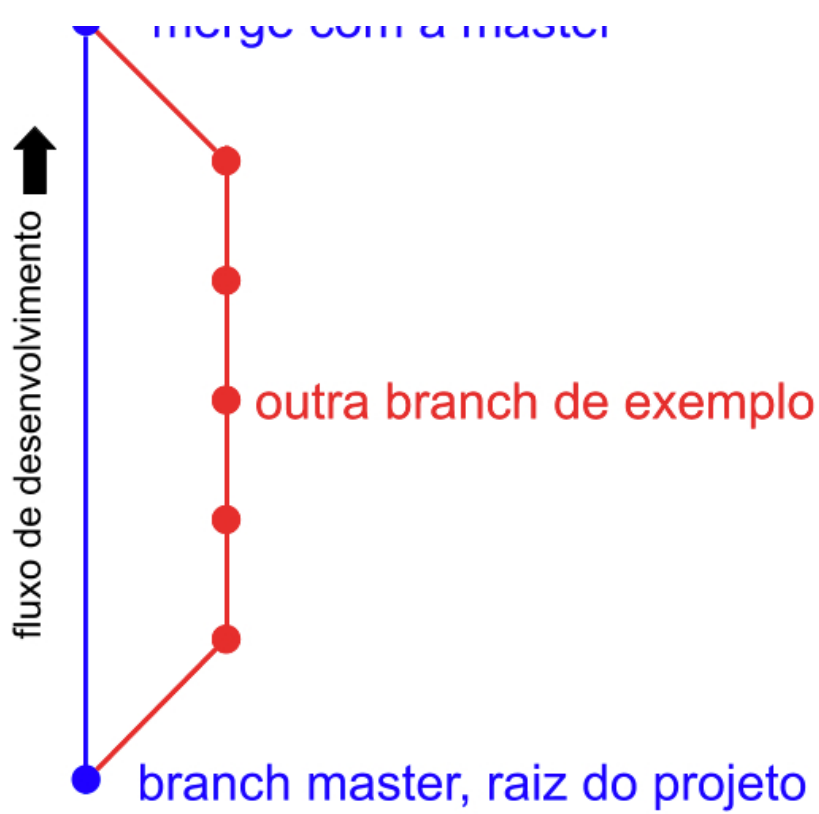




Branch master

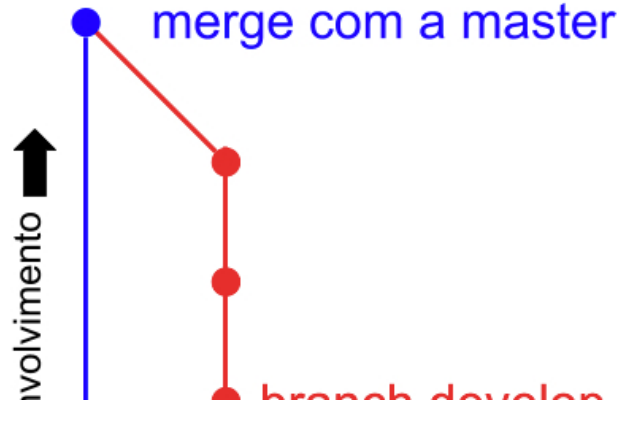
- É a **branch principal** do projeto. Sempre é criada como **raiz do projeto** e onde serão publicadas as **versões de produção** do projeto. Na figura abaixo ela é representada pela linha **azul**.

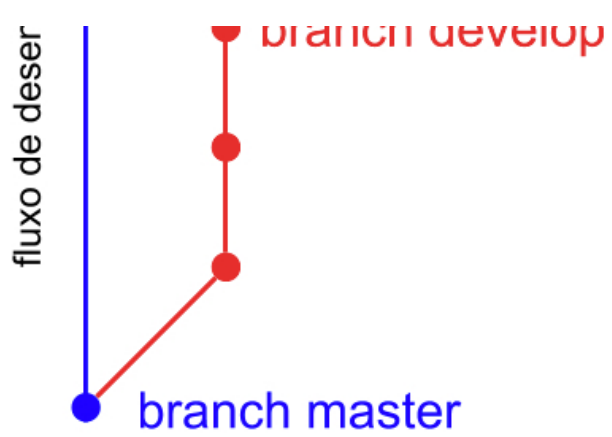
merge com a master



Branch develop

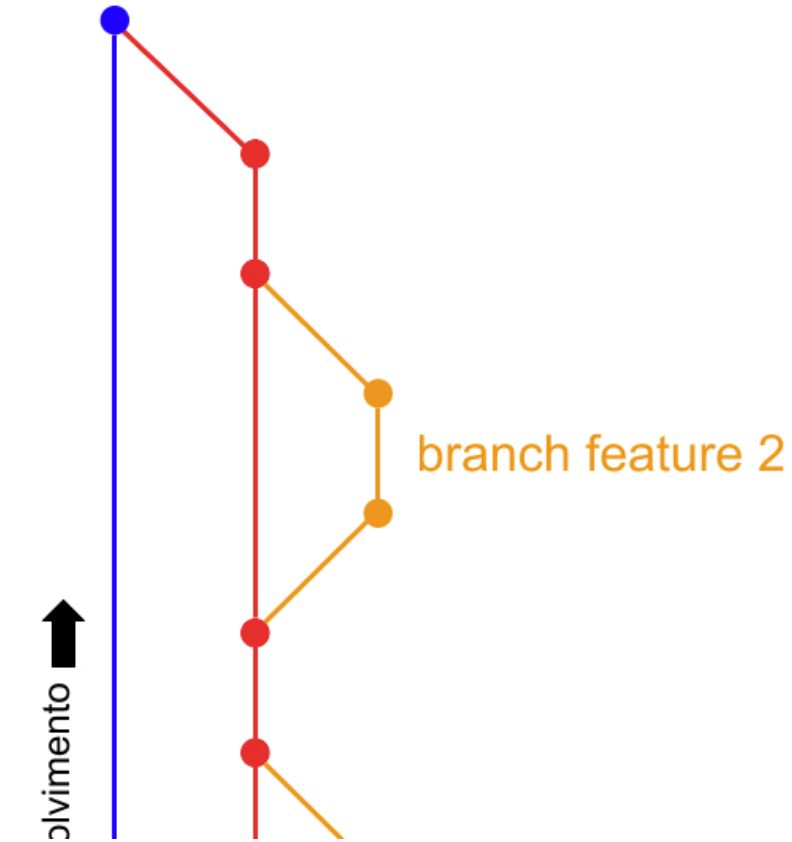
- É a principal branch de **atualizações do projeto** e de onde se originam as versões mais estáveis para implantação/lançamento. Na figura abaixo ela é representada pela linha **vermelha**.

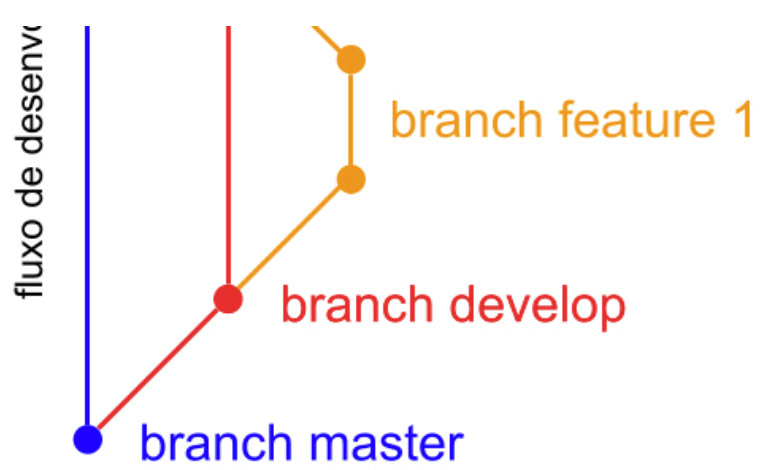




Branch features

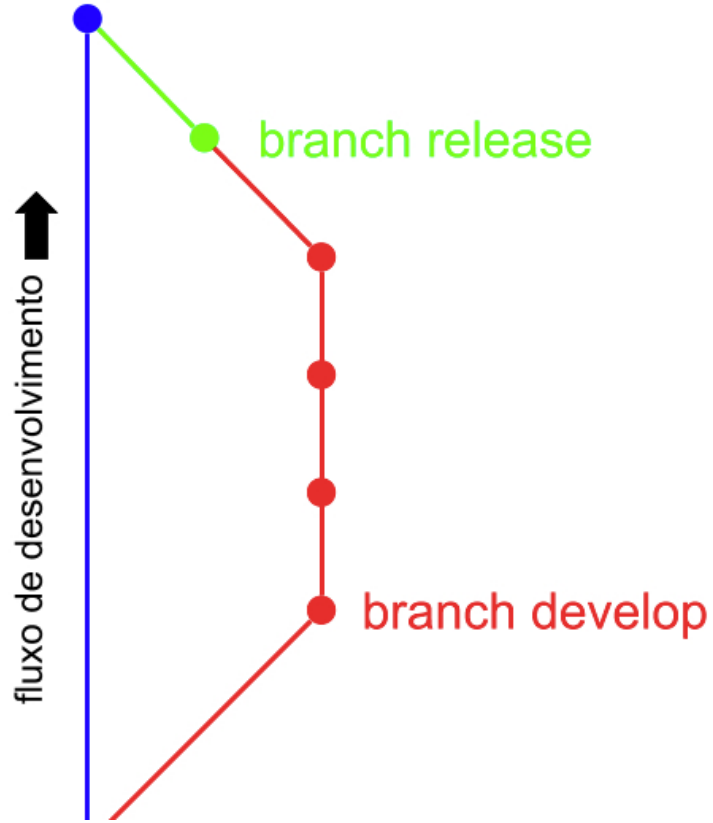
- É um conjunto de branchs criadas **a partir da develop** e utilizadas para criação de **novas funcionalidades para o projeto**. Cada funcionalidade deve ser separada em sua própria branch. Na figura abaixo ela é representada pelas linhas **laranja**.





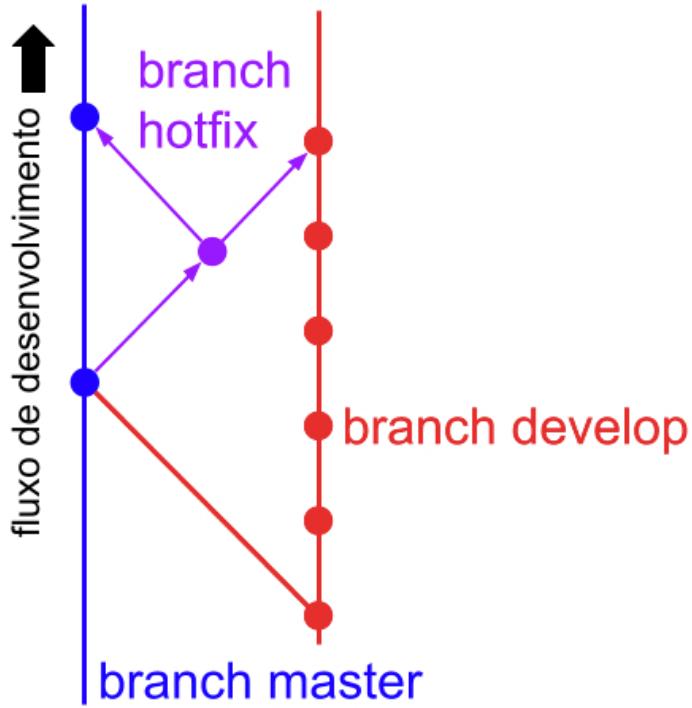
Branch release

- É a branch responsável por **publicar na master a versão mais estável vinda da develop** e serve como **refência de controle** das versões do projeto.



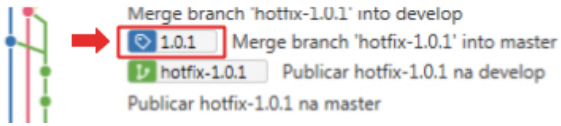
Branch hotfix

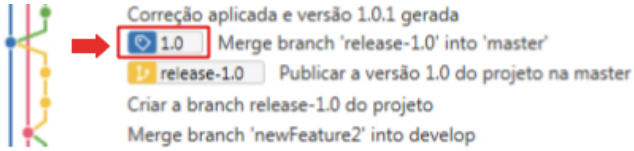
- É a branch responsável por **corrigir problemas das versões em produção inadiáveis**, isto é, não podem ser corrigidos a partir da develop. Se origina da master e publica suas correções de volta na develop e na master. **Obs:** Recomenda-se publicar a atualização primeiro na develop e por último na master, para que o gráfico do projeto fique mais compreensível em ferramentas de visualização como o SourceTree.



Tag

- São utilizadas como indicadores da versão do projeto, sempre devem ser aplicados após o lançamento (**release**) ou correção (**hotfix**) do projeto.





Modelo de Commits

Utilizamos como referência uma combinação dos exemplos e modelos sugeridos por Chris Beams (How to Write a Git Commit Message) e pela Udacity (Udacity Git Commit Message Style Guide).

7 Boas práticas de um Commit

1. Separar o assunto do corpo com uma linha vazia
2. Limitar o assunto a 50 caracteres (72 é o limite para ser truncado pelo GitHub)
3. Começar o assunto com letra maiúscula
4. Não terminar o assunto com ponto
5. Usar humor imperativo no assunto (como se desse uma ordem ou instrução)
6. Encapsular o corpo em 72 caracteres
7. Use o corpo para explicar **o que** e **por que** VS **como**

Tipos de indicadores de um commit

feat: um novo recurso.
fix: uma correção de bug.
docs: alterações na documentação.
style: formatação, falta de dois pontos, etc; nenhuma alteração de código.
refactor: refatoração do código de produção.
test: adição de testes, teste de refatoração; nenhuma alteração no código de produção.
chore: atualizar tarefas de compilação, configurações do gerenciador de pacotes, etc; nenhuma alteração no código de produção.

Como manter o histórico de commits linear

1. Faz o commit: $ git commit
2. Git fetch (from upstream): $ git fetch [upstream]
3. Se estiver X pra frente e Y pra trás, faz o Rebase: $ git rebase
4. Se o repositório estiver X pra frente (e nenhum pra trás), faz o push to upstream: $git push upstream

Como utilizar o Modelo de Branching

Os passos descritos abaixo são os comandos necessários para utilizar da maneira correta o modelo proposto.

Início do projeto

1. Criar um repositório do Git no diretório atual:

```
$ git init
```

2. Criar o arquivo README com conteúdo inicial:

Opção 1: via linha de comando:

```
$ echo "# Documentação do projeto" > README.md
```

Opção 2: usando um editor de texto:

```
$ <comando_editor_texto> README.md
```

- Exemplo com VScode:

```
$ code README.md
```

- Exemplo com Vim:

```
$ vim README.md
```

- Exemplo com Nano:

```
$ nano README.md
```

3. Commitar a criação do README na master:

```
$ git add README.md
$ git commit -m "Commit inicial do projeto"
```

4. O Resultado esperado é algo semelhante a imagem abaixo:

(IMAGEM AQUI)

Criando a branch develop e modificando o projeto

1. Criar a branch develop a partir da master:

```
$ git checkout -b develop master
```

2. Adicionar arquivo(s) e/ou diretório(s) básicos ao projeto.

3. Commitar a adição dos arquivo(s) e/ou diretório(s):

```
$ git add diretorio1 arquivo1.txt arquivo2.txt
$ git commit -m "Adicionar arquivos básicos do projeto"
```

4. O Resultado esperado é algo semelhante a imagem abaixo:

(IMAGEM AQUI)

Atualizando a evolução do projeto da branch develop na branch master

1. Mudar para a branch master:

```
$ git checkout master
```

2. Fundir o conteúdo da develop na master e commitar este evento:

```
$ git merge --no-ff develop master
```

- Uma janela vai abrir para editar a mensagem do commit, apenas feche-a.

3. O Resultado esperado é algo semelhante a imagem abaixo:

(IMAGEM AQUI)

Criar uma branch a partir da develop

1. Para criar uma nova branch a partir da develop, subsititua o argumento pelo desejado.

```
$ git checkout -b <nome da branch> develop
```

Exemplo:

```
$ git checkout -b ConsultarSaldo develop
```

Atualizando a evolução de uma branch na branch develop

1. Mudar para a branch develop:

```
$ git checkout develop
```

2. Fundir o conteúdo da branch desejada na develop:

```
$ git checkout -b <nome da branch> develop
```

Exemplo:

```
$ git merge --no-ff ConsultarSaldo develop
```

3. Commitar este evento.

Criando a branch Release

1. Criar a branch Release a partir da develop:

```
$ git checkout -b <nome da release> develop
```

Exemplo:

```
$ git checkout -b release-1.0 develop
```

2. Commitar a entrega versão:

```
$ git commit -m "Publicar versão 1.0"
```

Publicando a Release na branch master

1. Mudar para a branch master:

```
$ git checkout master
```

2. Fundir o conteúdo da na master e commitar este evento:

```
$ git merge --no-ff <nome da release> master
```

Exemplo:

```
$ git merge --no-ff release-1.0 master
```

3. Aplicar a tag da versão X.Y:

```
$ git tag -a X.Y
```

Exemplo:

```
$ git tag -a 1.0
```

Criando a branch Hotfix e arrumando um problema em produção

1. Criar a branch Hotfix a partir da master:

```
$ git checkout -b hotfix-<número da versão> master
```

Exemplo:

```
$ git checkout -b hotfix-1.0.1 master
```

2. Fazer as alterações necessárias.
3. Commitar a mudança do projeto:

```
$ git commit -m "Corrigir <problema encontrado>"
```

Publicando o hotfix na branch develop

1. Mudar para a branch develop:

```
$ git checkout develop
```

2. Fundir o conteúdo da hotfix na develop e commitar este evento:

```
$ git merge --no-ff hotfix-<número da versão> develop
```

Exemplo:

```
$ git merge --no-ff hotfix-1.0.1 develop
```

Publicando o hotfix na branch master

1. Mudar para a branch master:

```
$ git checkout master
```

2. Fundir o conteúdo da Hotfix na master e commitar este evento:

```
$ git merge --no-ff hotfix-<número da versão> master
```

Exemplo:

```
$ git merge --no-ff hotfix-1.0.1 master
```

3. Aplicar a tag da versão:

```
$ git tag -a <número da versão>
```

Exemplo:

```
$ git tag -a 1.0.1
```