Estudos DevOps

Desvendando o modelo GitFlow, e mostrando como é utilizado pela T-Systems

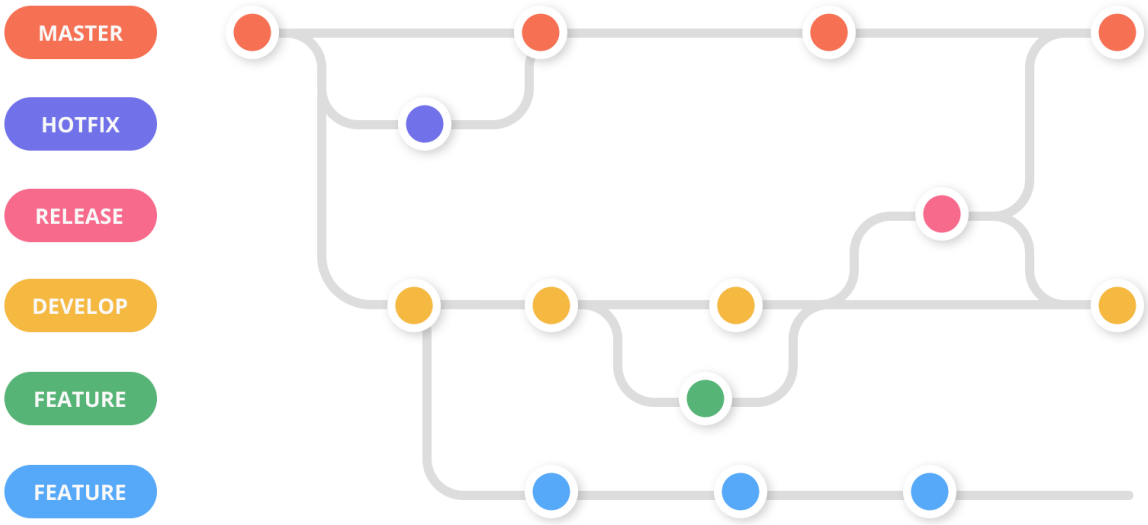
GitFlow é um modelo de fluxo de trabalho para desenvolvimento de software usando o sistema de controle de versão Git.

O GitFlow define um conjunto de regras e procedimentos para o gerenciamento de branches (ramificações) do Git em um projeto. Ele usa duas branches principais: `master` e `develop`, cada uma com um propósito específico.

A branch `master` é usada para armazenar a versão mais estável e confiável do software, que é adequada para ser implantada em produção. A branch `develop` é usada como a branch principal de desenvolvimento, onde os desenvolvedores trabalham para adicionar novos recursos e corrigir bugs.

Além dessas duas branches principais, o GitFlow usa outras branches específicas para diferentes propósitos, como branches de feature (funcionalidades), release (lançamento) e hotfix (correção urgente).

Exemplo



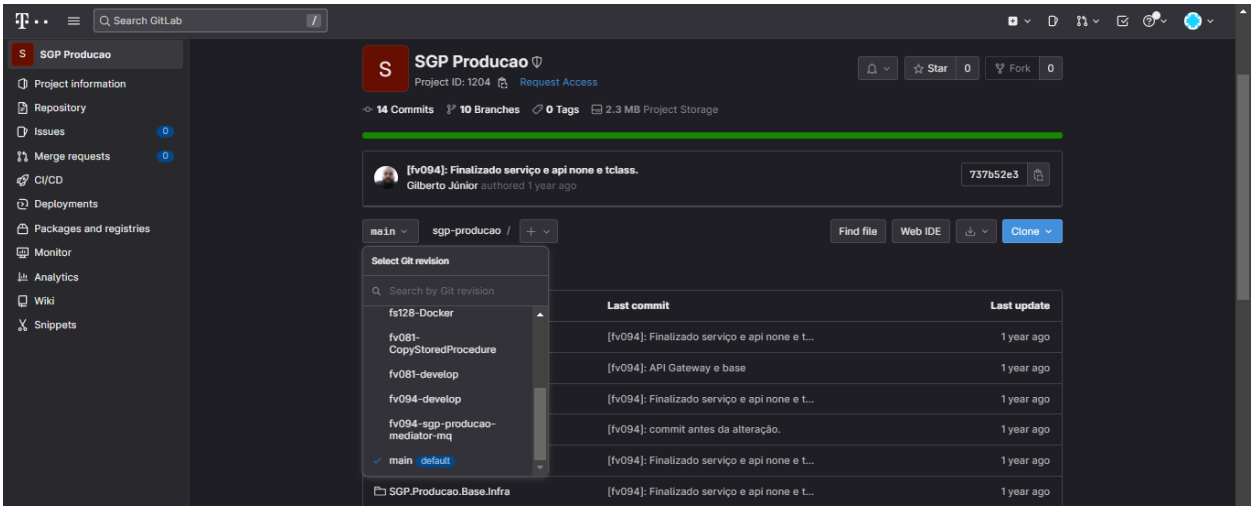
O fluxo de trabalho do GitFlow segue um processo rigoroso para gerenciamento de código, onde as novas funcionalidades são desenvolvidas em branches separadas a partir da branch `develop`. Quando a funcionalidade está pronta, ela é mesclada de volta para a branch `develop` através de um processo de revisão de código.

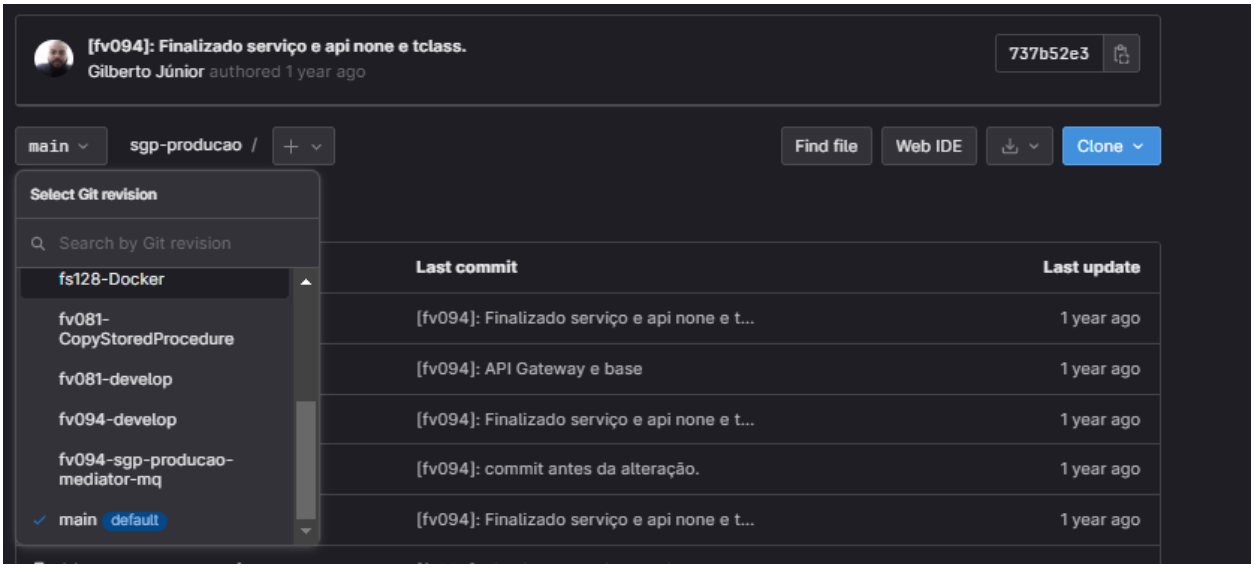
Quando a versão atual está pronta para ser lançada, uma nova branch de release é criada a partir da branch `develop`. Nessa branch, ocorrem testes finais, ajustes de última hora e preparação para o lançamento. Quando tudo está pronto, a branch de release é mesclada de volta para a branch `develop` e `master`, e uma nova versão é lançada.

O GitFlow é um modelo de fluxo de trabalho robusto e estruturado, que ajuda a manter o controle sobre o desenvolvimento de software, permitindo que as equipes gerenciem efetivamente o ciclo de vida do software.

Veja como a T-Systems usa o GitFlow em seus projetos

Para esse exemplo, vamos observar o repositório de SGP Produção. Observe na imagem abaixo, o projeto possui 10 branches (ramificações), incluindo a main (branch principal) .





Nesse repositório, temos as principais branches:

`develop` (ambientes de desenvolvimento, onde os desenvolvedores atuam)

A Branch relacionada ao `Docker`

E a Branch `Main` (projeto principal, versão mais estável do projeto).

Cada Branch tem o seu nível de hierarquia, sendo a main o projeto principal.

Hotfix, o que é ?

ℹ️ *Hotfix é uma correção de bug ou problema crítico que é aplicada imediatamente em um software em produção, sem passar pelo processo normal de desenvolvimento e teste. Um hotfix é geralmente aplicado para resolver um problema que pode afetar negativamente os usuários ou a funcionalidade do sistema.*

Release

ℹ️ *Um lançamento de sucesso pode ajudar a manter a confiança do usuário no software, enquanto um lançamento malsucedido pode levar a problemas de reputação e impactos financeiros negativos para a empresa. Por isso, é importante que as equipes de desenvolvimento de software tenham processos claros e rigorosos para garantir que cada release seja de alta qualidade e atenda às expectativas do usuário. Por isso a importância do modelo GitFlow em projetos.*

Padronização de Commits utilizado na T-Systems

ℹ️ *A padronização de commits é uma prática que consiste em adotar um padrão consistente para as mensagens de commit em um projeto de software. Essa prática ajuda a manter um histórico de commits mais legível e fácil de entender, além de facilitar a colaboração em equipe e a revisão de código.*

Na T-Systems temos um padrão para realizar os *commits* no Git. Esse padrão é com base em uma convenção de mercado, com algumas adequações para as necessidades.

Tipos de Commit

Os **tipos** representam a intenção do commit. Abaixo estão relacionados os **tipos** que utilizamos no momento:

- `FIX` - Indicam que o trecho de código commitado está solucionando um problema (bug fix).
- `FEAT`Indicam que o trecho de código está incluindo um novo recurso.
- `DOCS` - Indicam que houveram mudanças na documentação, como por exemplo no Readme do repositório. **(Não inclui alterações em código).**
- `STYLE` - Indicam que houveram alterações referentes a formatações de código, de telas, HTML, layouts de aplicações desktop...
- `REFACTOR` - Referem-se a mudanças devido a refatorações que não alterem o comportamento da funcionalidade, ou alterações devido ao **Code Review**.
- `BUILD` - São utilizados quando são realizadas modificações em arquivos de build e dependências. Por exemplo, arquivos de esteira.
- `PERFORMANCE`Tem como finalidade informar que as alterações foram para melhorar o desempenho da aplicação.
- `TEST` - São utilizados quando são realizadas alterações em testes, seja criando, alterando ou excluindo testes (Qualquer tipo de teste automatizado).
- `CHORE` - Indicam atualizações de tarefas de build, configurações de administrador, pacotes... como por exemplo adicionar um pacote no gitignore. **(Não inclui alterações em código)**

Exemplos

Importante deixar claro que os **tipos** citados anteriormente, são obrigatórios em todos os commits. A descrição e o número da issue são opcionais. Nos modelos a seguir vamos ver alguns exemplos:

Mensagem de commit com descrição e número da issue

> [FEAT]: Criação da funcionalidade de relatórios
>
> A nova funcionalidade visa substituir a exportação para Excel.
>
> Issues: #30

Mensagem de commit com descrição fechando a issue

> [FIX]: Correção do bug no relatório
>
> A coluna contendo o nome não estava com as informações corretas.
>
> Closes issue: #31

Fonte: https://sbs.t-systems.com.br/gitlab/mercedes/collab/guias/git/-/wikis/Padronização de Commits