GitFlow

Gabriela Bairros Porto[1]

[1]Departamento de Informática
Universidade Federal do Paraná (UFPR) - Curitiba, PR - Brasil

gbp16@inf.ufpr.br

Resumo. *O Git Flow é uma ferramenta altamente recomendada para organizar branches e commits em uma aplicação, oferece um fluxo de trabalho claro e estruturado, que ajuda na colaboração entre desenvolvedores, evita conflitos de código e permite um gerenciamento eficiente do processo de desenvolvimento.*

1. Introdução

No desenvolvimento de projetos de software, uma organização eficiente do fluxo de trabalho é fundamental para garantir a colaboração entre os membros da equipe, evitar conflitos de código e manter um histórico claro das alterações realizadas. Nesse contexto, o Git Flow[1] emerge como uma poderosa ferramenta para auxiliar no gerenciamento de branches e commits em uma aplicação web.

O Git Flow oferece um conjunto de diretrizes e boas práticas que promovem uma estrutura organizada e um fluxo de trabalho claro e estruturado[2].

2. Benefícios do Git Flow

Existem várias vantagens em utilizar o Git Flow para organizar branches e commits em uma aplicação web. O Git Flow define diferentes tipos de branches, como **main**, **develop**, **feature**, **release**, **hotfix** e **fix**, proporcionando uma organização clara do código em diferentes estágios de desenvolvimento. Isso facilita para os desenvolvedores entenderem em qual etapa cada parte do código está.

Uma das principais vantagens do Git Flow é a possibilidade de desenvolvimento paralelo, vários desenvolvedores podem trabalhar em funcionalidades ou correções diferentes ao mesmo tempo, cada um em sua própria branch de *feature*. Isso agiliza o processo de desenvolvimento, permitindo que várias tarefas sejam realizadas simultaneamente, sem a necessidade de esperar pelo término de outras.

Além disso, o Git Flow proporciona maior flexibilidade e autonomia aos membros da equipe. Como não há dependências entre as tarefas em desenvolvimento, cada desenvolvedor pode trabalhar de forma independente, focado em sua própria tarefa. Isso aumenta a eficiência e a produtividade da equipe como um todo, também melhora o processo de revisão de código.

Ao desenvolver uma nova funcionalidade, o desenvolvedor pode criar uma branch específica para essa funcionalidade e enviar um resumo dos commits por e-mail para os revisores de código. Isso facilita a revisão, pois destaca as diferenças em relação ao código anterior, permitindo um feedback mais rápido e uma revisão mais eficiente.

Além desses benefícios, o Git Flow também facilita o desenvolvimento de testes de maneira isolada. É possível criar e executar testes em uma branch separada, sem

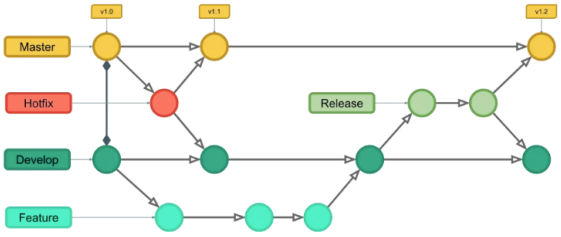

Figura 1. GitFlow

interferir no código principal, proporcionando uma visualização mais clara e efetiva dos testes, ajudando a garantir a qualidade do código.

Para complementar o uso do Git Flow, existem extensões como *Live Share*, *CodeStream* e *ESLint*, que podem auxiliar no desenvolvimento, melhorando a colaboração, a produtividade e a qualidade do código. Essas extensões fornecem recursos adicionais que ampliam as capacidades do Git Flow e aprimoram o fluxo de trabalho da equipe., que podem auxiliar no desenvolvimento, melhorando a colaboração, a produtividade e a qualidade do código.

3. Explicação de branches

A Figura 1 mostra a imagem adaptada de [3].

3.1. Main

A branch *Main* representa o código estável e pronto para produção. Isso implica que os clientes podem utilizá-la com confiabilidade, uma vez que está separada do desenvolvimento contínuo e segue um ciclo de desenvolvimento que visa minimizar erros e regressões no ambiente de produção.

É possível realizar o versionamento e gerenciar os lançamentos por meio das branches de *release*, criadas a partir do desenvolvimento, que consolidam um conjunto específico de funcionalidades para implantação, marcando pontos históricos de lançamento.

Para lidar com bugs e problemas críticos, é criada uma outra branch, a qual discutiremos posteriormente, denominada *hotfix*. Cada branch e commit possuem um propósito e contexto claros, facilitando o rastreamento e auxiliando na auditoria e acompanhamento das modificações das funcionalidades.

A branch *Main* auxilia no desenvolvimento, fornecendo um ponto de referência comum para a colaboração entre os membros.

3.2. Develop

A branch *Develop* atua como a branch de integração principal e facilita a colaboração entre os desenvolvedores. É nessa branch que os desenvolvedores integram suas branches de funcionalidades e correções de bugs, centralizando todo o trabalho convergido e permitindo criar e trabalhar em várias branches de funcionalidades simultaneamente.

Os desenvolvedores podem revisar o código uns dos outros, fornecer feedback e garantir a qualidade geral do software. A branch *Develop* funciona como um ambiente

relativamente estável para testes de integração e garantia de qualidade. Testes mais abrangentes podem ser realizados para garantir que o software funcione corretamente conforme o esperado e não introduza regressões ou conflitos com funcionalidades já existentes.

Essa branch também permite iniciar uma previsão do que será lançado para o ambiente de produção com maior estabilidade.

3.3. Feature

A branch *Feature* oferece um ambiente isolado para o desenvolvimento de novas funcionalidades, permitindo um foco maior por parte do desenvolvedor. Ela permite o desenvolvimento contínuo e em paralelo de diferentes funcionalidades, tornando o processo em equipe mais eficiente, uma vez que os desenvolvedores não possuem dependências entre si para o desenvolvimento.

A branch *Feature* também auxilia na prevenção de conflitos e na redução do risco de conflitos de código, permitindo uma visão clara das branches do que foi implementado e corrigido. Ela facilita a colaboração entre os membros, pois é possível compartilhar as branches e realizar revisões de código e feedback pelos pares.

Essas branches incentivam uma abordagem de desenvolvimento incremental. Os desenvolvedores podem dividir funcionalidades complexas em tarefas menores e gerenciáveis, criando branches separadas para cada uma. Esse desenvolvimento incremental facilita o progresso iterativo, com commits frequentes e a capacidade de receber feedback no início do ciclo de desenvolvimento.

3.4. Fix

A branch *Fix* fornece uma abordagem mais estruturada para lidar e resolver bugs ou problemas no código. Ela permite que os desenvolvedores se concentrem na solução e isolem as alterações relacionadas ao bug específico. Dessa forma, é possível garantir uma correção que não interfere na estabilidade e no progresso de outras funcionalidades em desenvolvimento, permitindo testes isolados.

As correções realizadas na branch *Fix* possuem ciclos de lançamento separados e podem ser mescladas na branch principal aplicável, além de permitir um histórico claro e rastreável das correções de bugs.

3.5. Hotfix

Considerando a abordagem já apresentada com a branch *Fix*, a branch *Hotfix* permite um processo mais simples e acelerado para lidar com problemas urgentes ou críticos no ambiente, geralmente na produção. Quando um bug grave é descoberto no ambiente de produção, uma branch *Hotfix* é criada para resolver o problema sem interromper o desenvolvimento em andamento ou aguardar o próximo ciclo de lançamento regular.

Basicamente, a branch *Hotfix* introduz a mesma estrutura do *Fix*, adicionando a urgência necessária para lidar com os problemas críticos.

4. Boas práticas para fazer commits

Para manter um histórico de commits claro e bem estruturado, é importante seguir algumas boas práticas ao realizar commits em diferentes branches do Git. Aqui estão algumas diretrizes gerais para fazer commits de forma eficaz:

4.1. Main Branch

Os commits na branch `main` devem representar versões ou alterações significativas. Cada commit deve ser claro e fornecer informações sobre as mudanças realizadas. É importante fornecer uma descrição concisa do que foi alterado ou adicionado no commit.

4.2. Develop Branch

Cada commit na branch `develop` deve se concentrar em uma nova funcionalidade, melhoria ou correção de bug. Os commits devem ser atômicos, ou seja, encapsular uma única mudança lógica. É fundamental que os commits sejam descritivos e forneçam contexto sobre a alteração realizada.

4.3. Feature Branches

As branches de `feature` devem ser focadas e específicas em relação ao desenvolvimento em andamento. Cada commit deve encapsular um pequeno passo para completar a funcionalidade em desenvolvimento. Os commits devem ser frequentes e pequenos, facilitando revisões, reversões e o acompanhamento do progresso da feature.

4.4. Hotfix Branches

As branches de `hotfix` devem ter poucos commits e serem direcionadas a bugs críticos que ocorrem no ambiente de produção. É essencial que os hotfixes sejam testados e validados de forma rigorosa, pois eles serão mesclados diretamente na branch `main`.

Independentemente da branch em que esteja trabalhando, os commits devem seguir um caminho lógico de desenvolvimento. Eles devem ser descritos de maneira clara e concisa, fornecendo informações sobre as alterações realizadas. Além disso, é recomendado revisar e refatorar os commits para garantir um histórico claro e organizado.

5. Conclusão

Em conclusão, o Git Flow é uma ferramenta essencial para o desenvolvimento de projetos de software, especialmente em equipes que buscam organização, colaboração e eficiência. Ao adotar o Git Flow, as equipes podem se beneficiar de um fluxo de trabalho claro e estruturado, que promove o desenvolvimento paralelo, a revisão de código eficiente, os testes isolados e a autonomia dos membros.

Ao seguir as diretrizes do Git Flow, como utilizar as branches principais (*main* e *develop*) de forma adequada, criar branches de *feature*, *release*, *hotfix* e *fix* para cada tipo de modificação, e realizar commits com mensagens claras e descritivas, é possível manter um histórico de alterações organizado e compreensível.

As boas práticas de commits, como manter os commits atômicos, focados em uma única alteração lógica, e revisar regularmente o histórico para garantir sua clareza e coesão, contribuem para uma melhor rastreabilidade e facilidade na identificação e solução de problemas.

Em suma, o Git Flow é uma metodologia poderosa que auxilia no gerenciamento de branches e commits em projetos de desenvolvimento de software. Sua estrutura organizada e suas diretrizes claras proporcionam benefícios tangíveis, como uma melhor colaboração entre os membros da equipe, a prevenção de conflitos de código e a facilitação de testes e revisões. Portanto, o uso adequado do Git Flow pode impulsionar a eficiência, a qualidade e o sucesso dos projetos de desenvolvimento de aplicativos web.

Referências

[1] DataSift. Introducing gitflow. https://datasift.github.io/gitflow/IntroducingGitFlow.html.

[2] Daniel Kummer. Git flow cheat sheet. https://danielkummer.github.io/git-flow-cheatsheet/.

[3] Programador Viking. Git flow: O guia completo para iniciantes. https://programadorviking.com.br/git-flow-o-guia-completo-para-iniciantes/.