GIT & GITHUB

O que é o Git?

O Git é um sistema de controle de versão distribuído, que permite que indivíduos e equipes controlem as mudanças em seu código ao longo do tempo. Ele é usado em conjunto com plataformas como o GitHub para permitir o compartilhamento e colaboração em projetos de software.

Como usar o Git

Para usar o Git, você precisa instalar o software em seu computador e criar um repositório local. Em seguida, você pode adicionar arquivos ao repositório e fazer o commit dessas alterações. Para colaborar em projetos com outras pessoas, você pode enviar seu código para um repositório remoto no GitHub.

Benefícios do uso do Git

O Git fornece controle de versão, o que significa que você pode manter o controle das mudanças em seu código ao longo do tempo. Além disso, ele permite que você trabalhe em projetos com outras pessoas sem conflitos de alterações, já que cada pessoa tem sua própria cópia do código. Isso torna a colaboração mais fácil e eficiente.

Palavras chave GIT

1. **Repositório**: é o local onde os arquivos e o histórico de um projeto são armazenados. O Git é um sistema de controle de versão distribuído, o que significa que cada usuário possui uma cópia completa do repositório.
2. **Commit**: é uma alteração no projeto que é registrada no histórico de commits do Git. Cada commit tem uma mensagem que descreve as mudanças feitas e é identificado por um código hash exclusivo.
3. **Branch**: é uma ramificação independente do histórico de commits do Git. Os branchs permitem que diferentes fluxos de trabalho ocorram simultaneamente, sem afetar o branch principal.
4. **Merge**: é o processo de combinar duas ou mais branchs do Git em um único histórico de commits. Isso permite que as alterações feitas em uma branch sejam incorporadas ao branch principal.

5. **Pull Request**: é uma solicitação para que as alterações feitas em um branch de um repositório sejam mescladas ao branch principal. Isso é comumente usado em projetos de código aberto, onde os colaboradores podem enviar sugestões de mudanças.
6. **Clone**: é a ação de criar uma cópia local de um repositório Git. Isso permite que você trabalhe no projeto em sua própria máquina.
7. **Push**: é a ação de enviar as alterações locais feitas em um repositório Git para um repositório remoto, permitindo que outras pessoas possam acessá-las.
8. **Pull**: é a ação de baixar as alterações mais recentes de um repositório remoto e mesclá-las ao branch atual.
9. **Fork**: é a ação de criar uma cópia completa de um repositório Git hospedado em um servidor remoto, permitindo que você trabalhe na cópia sem afetar o repositório original.
10. F**eature:** nome de branch comumente utilizado para definir novas implementações.
11. F**ix/Hotfix:** diz respeito ao nível do erro encontrado e a urgência dele para liberação.
12. R**elease:** produto pronto para liberação, em fase de testes ainda.

Git Flow

Git flow é um modelo de fluxo de trabalho (workflow) para o gerenciamento de branchs em projetos de desenvolvimento de software com o Git. Ele foi desenvolvido por Vincent Driessen e tem como objetivo fornecer uma estrutura organizada e consistente para a criação e manutenção de branchs em um projeto.

O modelo Git flow é baseado em duas branchs principais: a branch `master`, que contém o código estável e pronto para produção, e a branch `develop`, que contém o código em desenvolvimento.

Além dessas duas branchs principais, o Git flow define quatro tipos de branchs auxiliares que podem ser criadas a partir da branch `develop`:

- Feature: criada a partir da branch `develop` para desenvolver novas funcionalidades;
- Release: criada a partir da branch `develop` para preparar a próxima versão de produção;

- Hotfix: criada a partir da branch `master` para corrigir problemas críticos na produção;
- Support: criada a partir da branch `master` para fornecer suporte a versões legadas do software.

Ao utilizar o Git flow, os desenvolvedores têm uma estrutura clara e bem definida para trabalhar em diferentes funcionalidades e correções de forma organizada e colaborativa, facilitando a revisão de código, a integração contínua e a entrega do software.

Vantagens

1. Estrutura organizada: o Git flow oferece uma estrutura bem definida e organizada para gerenciar as branchs em um projeto, o que facilita a colaboração e o trabalho em equipe.
2. Melhor controle de qualidade: ao utilizar branchs separadas para novas funcionalidades e correções de bugs, é possível controlar melhor a qualidade do código que é mesclado na branch principal.
3. Fácil manutenção de versões: com o Git flow, é possível criar branchs para versões específicas do software, o que facilita a manutenção de diferentes versões em produção.
4. Integração contínua: o Git flow é compatível com a prática de integração contínua, o que ajuda a garantir a qualidade do código e a prevenir problemas de integração.

Desvantagens

1. Complexidade: o Git flow pode ser mais complexo do que outros modelos de fluxo de trabalho, o que pode tornar a sua implementação mais difícil e exigir um maior esforço de treinamento da equipe.
2. Mais branchs: o Git flow envolve a criação de várias branchs adicionais, o que pode tornar o processo de gerenciamento de branchs mais confuso e complexo.
3. Dificuldade na manutenção: se a equipe não seguir adequadamente as convenções do Git flow, pode haver dificuldades na manutenção das branchs e na garantia da qualidade do código.
4. Demora na entrega: o Git flow pode levar a um processo de entrega mais lento, já que é necessário passar por várias etapas antes que o código seja mesclado

na branch principal.

5. Não é adequado para entrega contínua, pois não necessariamente o fluxo é o melhor para liberação.
6. Devido à sobrecarga necessária para liberar, isso pode levar a um acúmulo de dívida técnica.

Em resumo, o Git flow pode ser uma boa opção para projetos maiores ou mais complexos que exigem um maior controle de qualidade e uma estrutura mais organizada para gerenciamento de branchs. No entanto, ele pode ser mais difícil de implementar e exigir mais esforço de treinamento da equipe em comparação com outros modelos de fluxo de trabalho mais simples.

Dicas para commit message

- Separar o título do corpo do commit, sempre utilizando uma linha em branco. Caso use um git client que traz as informações em campos separados não há problema;
- Limite de 50 caracteres no título do commit, tente ao máximo não ultrapassar essa informação, caso contrario, o próprio git quebra a linha;
- Utilizar as letras maiúsculas que forem necessárias, pensando em destacar informações que precisam ser destacadas, e realmente tratar isso como um texto;
- Não utilizar pontos, é algo muito mais visual, mas evitar de usar pontos finais, exclamações, interrogações e etc;
- Utilizar o modo imperativo no título, esse é importante, e caso ficar na dúvida segue uma forma de pensar que ajuda. Basta escrever o seguinte “se aprovado, esse commit irá…” e a partir daqui começa a escrever o título do commit;
- Quebrar linhas no corpo do commit para que a leitura fique mais fácil. Utilizar poucos caracteres para descrever aquilo que se deseja e separar em paragrafos;
- Utilizar a mensagem para explicar o que foi feito e como foi feito, separando cara “coisa” feita em novas linhas;

Principais comandos GIT

1. `git init`: cria um novo repositório Git vazio no diretório atual.

2. `git clone`: cria uma cópia local de um repositório Git existente.
3. `git fork`: cria uma cópia completa de um repositório Git hospedado em um servidor remoto.
4. `git add`: adiciona alterações ao index do Git, preparando-as para serem commitadas.
5. `git commit`: registra uma nova versão do projeto no histórico de commits do Git.
6. `git push`: envia as alterações locais para um repositório remoto.
7. `git pull`: busca as alterações mais recentes de um repositório remoto e mescla com o branch atual.
8. `git branch`: lista, cria ou deleta branchs do projeto.
9. `git checkout`: altera o branch atual e permite navegar entre branchs e commits.
10. `git merge`: combina alterações de dois ou mais branchs do projeto.
11. `git status`: exibe informações sobre o estado atual do repositório, como arquivos modificados ou não adicionados ao index.
12. `git log`: exibe o histórico de commits do repositório, mostrando as alterações feitas em cada versão.
13. `git diff`: mostra as diferenças entre o estado atual do projeto e um commit ou branch específico.

Materiais de apoio

Git cheat sheet - principais comandos git

https://education.github.com/git-cheat-sheet-education.pdf

Dicas para commit message

https://cbea.ms/git-commit/

Mais sobre git flow

https://danielkummer.github.io/git-flow-cheatsheet/index.pt_BR.html