facili



Centro de Formação Profissional da Indústria Eletrónica, Energia, Telecomunicações e Tecnologias da Informação

Sistemas de controlo de versões GitHub

**Versão 1.0**

[Roberta Souza]

[Hélio Eduardo Castel-Branco]

[Nuno Amorim]

Grupo6

CET 90

UFCD 0835 – Sistemas de informação - fundamentos

Formador: Susana Pimentel

**© Portugal 2024**

**Indice…**

1. **Introdução (2 páginas) (Todos)**
   * **Importância do controlo de versões.**
   * **O que são o Git e o GitHub e por que são importantes.**
   * **Breve história e desenvolvimento do Git e GitHub.**
2. **O que é o Git (5 páginas) (Contextualização) (Roberta)**
   * **Conceitos fundamentais do Git: commits, branches, merges, rebase.**
   * **Fluxos de trabalho do Git: Centralizado, Feature Branch, Gitflow.**
   * **Resolução de conflitos no Git.**
3. **Funcionalidades e Utilização do GitHub (5 páginas) (Visão Geral) (Hélio)**
   * **Como o Git e o GitHub funcionam em conjunto.**
   * **Repositório, Pull Requests, problemas, acções.**
   * **GitHub Pages para alojamento de sites.**
   * **GitHub Gist para partilha de snippets de código.**
   * **Integração com outras ferramentas e plataformas. (integração com Jira)**
4. **Colaboração e Contribuição no GitHub (5 páginas) (Especificidades) (Nuno)**
   * **Como o GitHub facilita a colaboração em projetos de código aberto.**
   * **O processo de contribuição para projetos de código aberto no GitHub.**
   * **O papel do mantenedor de projetos.**
   * **A importância da documentação e dos códigos de conduta. (comentários, revisão de código)**
5. **Casos de Uso e Estudos de Caso GitHub (5 páginas) (estudo de caso) (Nuno)**
   * **Como o GitHub é utilizado pelas empresas.**
   * **Estudos de caso de projetos de código aberto bem-sucedidos no GitHub.**
   * **O papel do GitHub no DevOps.**
   * **Integração Contínua/Entrega (CI/CD).**
6. **Melhores Práticas e Dicas para Usar o GitHub (4 páginas) (dicas) (Todos)**
   * **Melhores práticas para usar o GitHub.**
   * **Dicas para resolver problemas comuns no GitHub.**
7. **Conclusão (2 páginas) (Todos)**
   * **Reflexões sobre o futuro do Git e do GitHub.**
   * **Resumo dos pontos-chave.**
   * **Dificuldades encontradas durante a utilização do Git e GitHub.**
8. **Introdução ao GitHub (2 páginas)**
   * **Importância do controlo de versões.**

O controle de versões é uma prática essencial para qualquer projeto que envolva o desenvolvimento de software ou o gerenciamento de arquivos digitais. Essa metodologia permite registar e acompanhar todas as alterações feitas em um arquivo ao longo do tempo, fornecendo um histórico completo e organizado do projeto. Além disso, o controle de versões facilita a colaboração entre equipes de desenvolvimento, garantindo uma gestão eficiente das diferentes versões e contribuições. Neste contexto, exploraremos a importância do controle de versões, destacando seus benefícios e práticas recomendadas para uma gestão eficaz do código-fonte e dos arquivos digitais.

* **O que é o GitHub e por que é importante?**
* **Breve história e desenvolvimento o GitHub.**

1. **Contextualização do Git e GitHub (5 páginas)**

De acordo com Chacon e Straub (2014) o Git é um sistema de controle de versão distribuído amplamente utilizado para o desenvolvimento de software e controle de código-fonte. Ele foi criado por Linus Torvalds em 2005, inicialmente para o desenvolvimento do kernel do Linux, e desde então se tornou uma das ferramentas mais populares entre os desenvolvedores de software.

O Git permite que os desenvolvedores trabalhem em projetos de forma colaborativa, rastreiem as alterações feitas no código-fonte ao longo do tempo e gerenciem diferentes versões de um mesmo projeto de forma eficiente. Algumas características importantes do Git incluem:

* **Sistema de controle de versão distribuído:** Cada desenvolvedor tem uma cópia local completa do repositório de código, o que permite trabalhar offline e de forma independente.
* **Eficiência:** O Git é projetado para ser rápido e eficiente, mesmo em projetos grandes com muitos arquivos e histórico extenso de alterações.
* **Ramificação (branching) e mesclagem (merging):** O Git facilita a criação de ramificações no código-fonte para desenvolver novos recursos ou corrigir bugs sem interferir no código principal. Posteriormente, as alterações de uma ramificação podem ser mescladas de volta ao ramo principal do projeto.
* **Rastreamento de alterações:** O Git regista todas as alterações feitas no código-fonte ao longo do tempo, permitindo que os desenvolvedores visualizem o histórico completo do projeto e revertam para versões anteriores se necessário.
* **Colaboração:** O Git facilita a colaboração entre desenvolvedores, permitindo que eles compartilhem seus trabalhos por meio de repositórios remotos hospedados em plataformas como GitHub, GitLab e Bitbucket.

**Conceitos Fundamentais do Git:**

* **Commits**: é uma alteração ou conjunto de alterações em um repositório Git. Cada commit tem uma mensagem associada que descreve as alterações feitas. Os commits são usados para registar o histórico de alterações no projeto.

* **Branches:** é uma linha de desenvolvimento separada no repositório Git. Ele permite que os desenvolvedores trabalhem em novos recursos ou correções de bugs sem interferir no código principal (ramo principal ou "master"). Os branches podem ser criados, mesclados e excluídos conforme necessário.

* **Merges**: é o processo de combinar as alterações de um branch em outro. Ela é usada para incorporar o trabalho feito em um branch de recurso de volta ao branch principal do projeto. O Git mescla automaticamente as alterações quando possível, mas pode ser necessário resolver conflitos manualmente.

* **Rebase**: é uma alternativa à merge que reescreve o histórico de commits de um branch. Ele move as alterações de um branch para o topo de outro, criando uma linha de desenvolvimento linear. O rebase é útil para manter um histórico de commits limpo e organizado.

**Fluxos de Trabalho do Git:**

* **Centralizado:** No fluxo de trabalho centralizado, todos os desenvolvedores trabalham diretamente no branch principal do projeto (geralmente chamado de "master" ou "main"). As alterações são feitas localmente e, em seguida, enviadas diretamente para o repositório central.

* **Feature Branch:** No fluxo de trabalho de branch de recurso, cada nova funcionalidade ou correção de bug é desenvolvida em seu próprio branch separado. Após a conclusão, o branch de recurso é mesclado de volta ao branch principal do projeto.

* **Gitflow:** É um modelo de fluxo de trabalho mais avançado que define uma estrutura específica para o gerenciamento de branches. Ele utiliza branches dedicados para diferentes tipos de desenvolvimento, como desenvolvimento de novas funcionalidades, correções de bugs e releases.

**Resolução de Conflitos no Git**

Os conflitos ocorrem quando o Git não consegue mesclar automaticamente as alterações devido a alterações conflitantes em áreas semelhantes do código. A resolução de conflitos geralmente envolve a análise das alterações conflitantes, a edição manual do código para resolver os conflitos e, em seguida, a criação de um novo commit para finalizar a merge.

Para resolver conflitos, os desenvolvedores podem usar ferramentas como o comando git mergetool para ajudar na resolução manual ou editarem os arquivos diretamente usando um editor de texto. Após resolver os conflitos, os desenvolvedores devem adicionar os arquivos alterados, criar um novo commit e, em seguida, concluir a merge.

* **Como é que o Git e o GitHub funcionam juntos.**
* **Repositório, Pull Requests, problemas, ações**
* **GitHub Pages para hospedagem de sites.**
* **GitHub Gist para partilhar de snippets de código.**
* **Integração com outras ferramentas e plataforma.**

1. **Colaboração de contribuição no GitHub (5 páginas)**

* **(Visão geral de como funciona; Colaboração em projetos de código aberto; pedidos de pull e revisão de código no GitHub;)**
* **Como o GitHub facilita a colaboração em projetos de código aberto.**
* **Que problemas é que o Github resolve x**
* **Que aplicações práticas tem (Que utilização tem sem ser Desenvolvimento)**
* **O processo de contribuição para projetos de códigos aberto no GitHub.**
* **O mantenedor de projetos.**
* **A importância da documentação e dos códigos de conduta. (comentários, code review) x**

1. **Casos de Uso e Estudos de caso GitHub (5 páginas)**

* **Como o GitHub é utilizado pelas empresas.**
* **Estudos de caso de projetos de código aberto bem-sucedido no GitHub.**
* **O papel do GitHub no DevOps**
* **Integração Contínua/Entrega (CI/CD)**

1. **Melhores Práticas e Dicas para Usar o GitHub (4 páginas)**

* **Melhores práticas para usar o GitHub.**
* **Dicas para resolver problemas comuns no GitHub.**

1. **Conclusão (2 páginas)**

* **Pontos chaves**
* **Dificuldades**
* **Reflexões sobre o futuro do GitHub.**

**Bibliografia**

CHACON, Scott; STRAUB, Ben. Pro Git. Disponível em: <https://git-scm.com/book/pt-br/v2>. Acesso em: 15/03/2024.

GIT - Documentação oficial. Disponível em: <https://git-scm.com/doc>. Acesso em: 14/03/2024.