Git , GitHub e Gitlab

Professora Patrícia Lima Felismino
Bacharel em Ciência da Computação - UNICSUL

Git

- Controle de versão: Conceito e importância
- Git como ferramenta distribuída
- Github e Gitlab como plataformas de colaboração

Controle de Versão: Conceito e Importância

- Controle de versão também conhecido como gerenciamento de código-fonte, é um sistema que rastreia e gerencia as alterações feitas em arquivos de código ao longo do tempo. Ele permite que equipes de desenvolvimento colaborem de forma eficiente, mantenham um histórico de alterações e *revertam para versões anteriores*, se necessário.

Benefícios do Controle de Versão

- **Colaboração:** Permite que várias pessoas trabalhem no mesmo projeto simultaneamente, sem conflitos.
- **Histórico:** Mantém um registro de completo de todas as alterações, facilitando a localização de erros e a recuperação de versões anteriores.
- **Recuperação:** Permite reverter para uma versão anterior do projeto em caso de erros ou problemas.
- **Segurança:** Protege o código-fonte contra perda de dados e alterações acidentais.
- **Documentação:** Facilita a documentação das alterações. Ajudando a enteder o histórico do projeto.

Existem Diferentes Tipos de Controle de Versão

- **Controle de versão centralizado:** Um servidor central armazena o histórico de versões, e os desenvolvedores acessam e modificam os arquivos através do servidor.
- **Controle de versão distribuído:** Cada desenvolvedor tem uma cópia completa do histórico em sua máquina local, permitindo que trabalhem offline e colaborem de forma mais flexível.

Exemplos de Controle de Versão

- GIT – um sistema de controle de versão distribuído amplamente utilizado para o desenvolvimento de software.
- Subversion (SVN) - Um sistema de controle de versão popular.
- Mercurial – Um sistema de controle de versão distribuído semelhante ao git.

- OBS! O controle de versão é uma FERRAMENTA essencial para equipes de desenvolvimento de software, ajudando a garantir a qualidade, a colaboração e a eficiência do processo de desenvolvimento.

GitHub e GitLab

- São Plataformas baseadas em nuvem que usam o sistema de controle de versão GIT para hospedar e gerenciar código, mas diferem principalmente em suas ofertas de DevOps e foco.

Diferenças entre GitHub e GitLab

- Gitlab – fornece um conjunto integrado de ferramentas de DevOps e CI/CD em um único aplicativo. Se destaca por, oferecer recursos de segurança mais robustos e é frequentemente preferido para equipes corporativas com necessidades de um pipeline DevOps completo e pronto para uso.
- GitHub – Depende mais de Integrações de terceiros para recursos semelhantes, ele se destaca por colaboração em código aberto e sua vasta comunidade.

Pipeline de DevOps

- Uma pipeline DevOps é uma sequência de processos e ferramentas que automatizam as etapas necessárias para levar o código desde o desenvolvimento até a produção. O objetivo principal é acelerar e otimizar o ciclo de vida do desenvolvimento de software, facilitando a colaboração entre as equipes de desenvolvimento, facilitando a colaboração entre as equipes e operações.

- Pipelines de CI/CD, que otimizam e aceleram o desenvolvimento de softwares, são um reflexo da metodologia DevOps, um conjunto de ideias e práticas que incentiva a colaboração entre as equipes de desenvolvimento e de operações de TI.

GitHub e GitLab

- Atividade Para os Treinandos

- Pesquisem e listem as principais diferenças e semelhanças entre o Github e o Gitlab e listem em um pdf e "subam" para o repositório do Github (Patricia17991), coloquem o arquivo na pasta Git-Github-Gitlab.

GitHub e Gitlab – Hospedando seu código na nuvem

- Criando repositório remoto
- Clonando com `git clone`
- Enviando com `git push`
- Recebendo com `git pull`

Comandos Básicos do Git

- `git init` # Inicializa um repositório
- `git status` # Verifica o estado atual
- `git add .` # Adiciona arquivos para commit
- `git commit -m "Mensagem"` # Salva alterações
- `git log` # Histórico de commits

Branches e Colaboração - Trabalhando em equipe com o Git

- Bash
- git branch nome-da-branch
- git checkout nome-da-branch
- git merge nome-da-branch

- Pull Request (GitHub) / Merge Request (GitLab)
- Revisão de código e comentários

Boas Práticas - Git com qualidade

- Commits claros e objetivos
- Uso de `.gitignore`
- Documentação com `README.md`
- Segurança: SSH e tokens

Projeto Prático - Colaboração real com o Git

- Criar um projeto em grupo
- Dividir tarefas com branches
- Usar Issues e Boards
- Entregar versão final com tag

Vamos a prática:

- # 1. Criar repositório local
- git init

- # 2. Criar README.md e fazer commit
- echo "# Meu Projeto" > README.md
- git add README.md
- git commit -m "Adiciona README"

- # 3. Criar repositório no GitHub/GitLab e conectar
- git remote add origin https://github.com/seuusuario/seurepo.git (aqui vai o link do seu perfil no github)
- git push -u origin master