

Gerenciamento de versão

GIT E GITHUB

Maria Elisa Gomes de Matos



Atua como desenvolvedora de software plena na Myrp Soluções Ltda, com ampla experiência em desenvolvimento de aplicações, arquitetura de software, testes automatizados, computação em nuvem, controle de versões e desenvolvimento ágil de software. Pós-graduada em Engenharia de Software com ênfase em DevOps na UNIFOR. Graduada em Ciências da Computação pela UVA e Técnica em Informática no GWA.



@mariaelisagmt



(88) 99794-5263



maria.elisa.gomes.de.matos@gmail.com
mariaelisagmt@edu.unifor.com



<https://www.linkedin.com/in/maria-elisa-gomes-de-matos/>

Introdução

Problemas de Desenvolvimento de Software Resolvidos pelo Controle de Versão



- | | | | |
|----|-----------------------------------|----|---------------------------------------|
| 01 | Perda de Código | 04 | Reverter Alterações |
| 02 | Colaboração | 05 | Testando Novas Funcionalidades |
| 03 | Rastreamento de Alterações | 06 | Gerenciamento de Lançamentos |

Evolução das Ferramentas de Controle de Versão

01

Controle de Versão Local

No início, os desenvolvedores mantinham versões locais de arquivos de código. Isso era propenso a erros, pois era fácil sobrescrever ou perder versões.

02

Controle de Versão Centralizado (CVCS)

Ferramentas como CVS, Subversion e Perforce introduziram um servidor central que mantinha todas as versões do código. Isso facilitou a colaboração, mas a perda do servidor central poderia resultar na perda de todo o histórico de versões.

02

Controle de Versão Distribuído (DVCS)

Ferramentas como Git, Mercurial e Bazaar permitiram que cada desenvolvedor tivesse uma cópia completa do histórico de versões. Isso tornou possível trabalhar offline e deu aos desenvolvedores a capacidade de ter várias “branches” de trabalho.

Introdução ao GIT



É um sistema de controle de versão distribuído (DVCS) criado por Linus Torvalds em 2005. Ele foi projetado para lidar com tudo, desde pequenos a grandes projetos com velocidade e eficiência.

O Git permite que cada desenvolvedor tenha uma cópia completa do histórico do projeto, facilitando o trabalho offline e fornecendo redundância em caso de falha do servidor central.

Ele é otimizado para desempenho, garantindo que as operações sejam rápidas e eficientes. Ele também garante a integridade dos dados através de uma estrutura de dados que protege contra corrupção.

Além disso, o Git é altamente flexível e pode ser personalizado para se adequar a vários fluxos de trabalho.

Vantagens e Desvantagens

01 Distribuído

02 Desempenho

03 Branching e Merging Eficientes

04 Integridade dos Dados

05 Flexibilidade

01 Curva de Aprendizado

02 Complexidade

03 Falta de Controle de Acesso Granular

04 Binários

Estágio do Git

O Git tem três estados principais nos quais seus arquivos podem estar: modificado, preparado e confirmado.

Isso nos leva às três principais seções de um projeto Git: a árvore de trabalho, a área de preparação e o diretório Git.

01

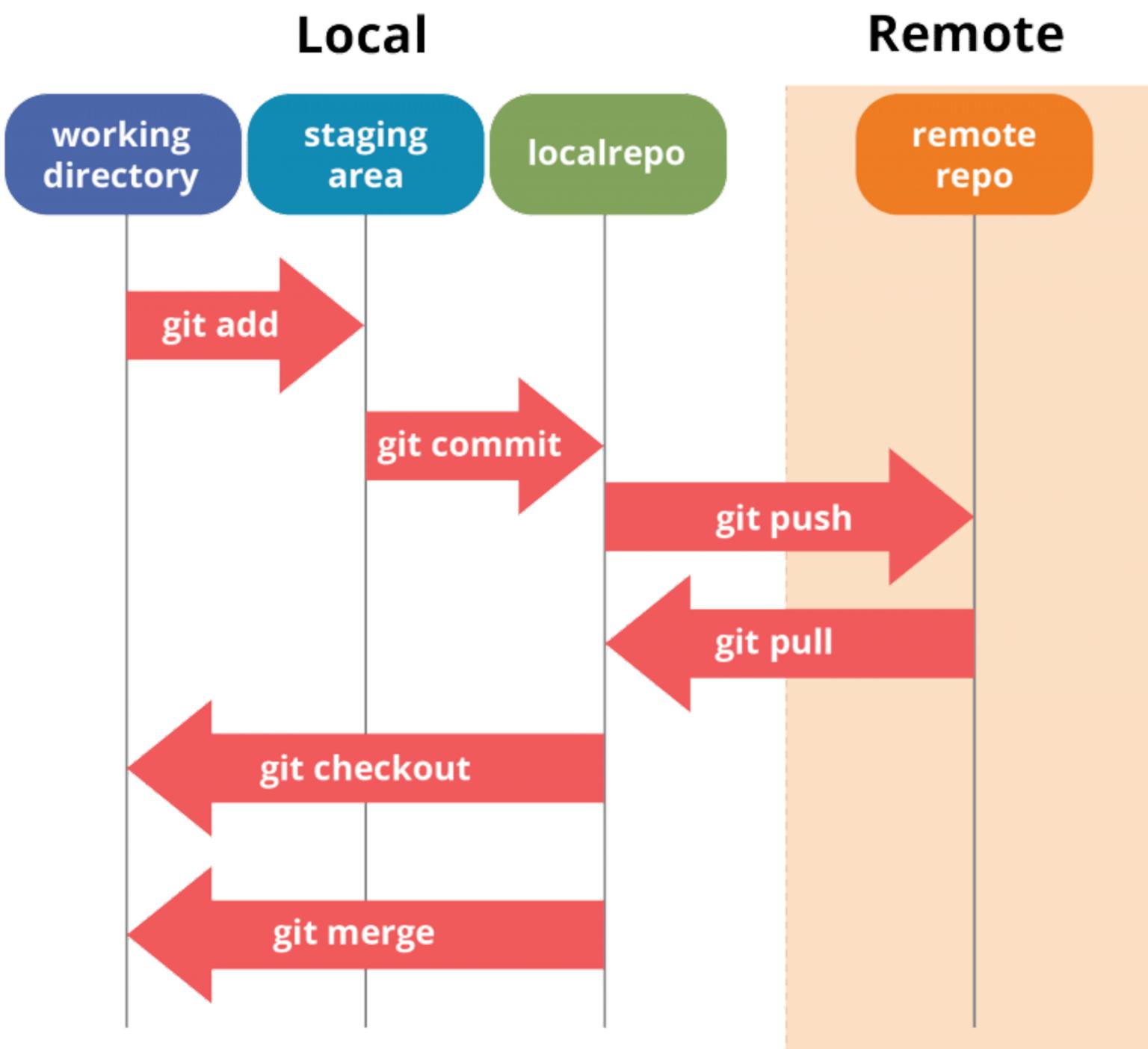
Modificado: significa que você alterou o arquivo, mas ainda não o confirmou no seu banco de dados.

02

Preparado: significa que você marcou um arquivo modificado em sua versão atual para entrar na próxima captura de commit.

03

Confirmado: significa que os dados estão armazenados com segurança no seu banco de dados local.



Comandos Principais

git init: Inicializa um novo repositório Git.

- Exemplo: git init
- Caso de uso: Quando você quer começar a rastrear um novo projeto com o Git.

git clone: Clona um repositório Git existente.

- Exemplo: git clone https://github.com/usuario/projeto.git
- Caso de uso: Quando você quer trabalhar em um projeto que já está sendo rastreado pelo Git.

git add: Adiciona arquivos ao índice do Git para serem rastreados.

- Exemplo: git add . (adiciona todos os arquivos modificados) ou git add arquivo.txt (adiciona um arquivo específico)
- Caso de uso: Quando você fez alterações em um ou mais arquivos e quer que essas alterações sejam rastreadas pelo Git.

git commit: Cria um novo commit com as alterações rastreadas.

- Exemplo: git commit-m "Mensagem do commit"
- Caso de uso: Quando você quer salvar um ponto específico no histórico do projeto.

Comandos Principais

git branch: Lista, cria ou deleta branches.

- Exemplo: git branch (lista branches), git branch nova_branch (cria nova branch), git branch-d branch_antiga (deleta branch)
- Caso de uso: Quando você quer trabalhar em uma nova funcionalidade ou corrigir um bug sem afetar o código principal.

git merge: Une as alterações de uma branch em outra.

- Exemplo: git merge branch_fonte
- Caso de uso: Quando você terminou de trabalhar em uma branch e quer unir as alterações na branch principal.

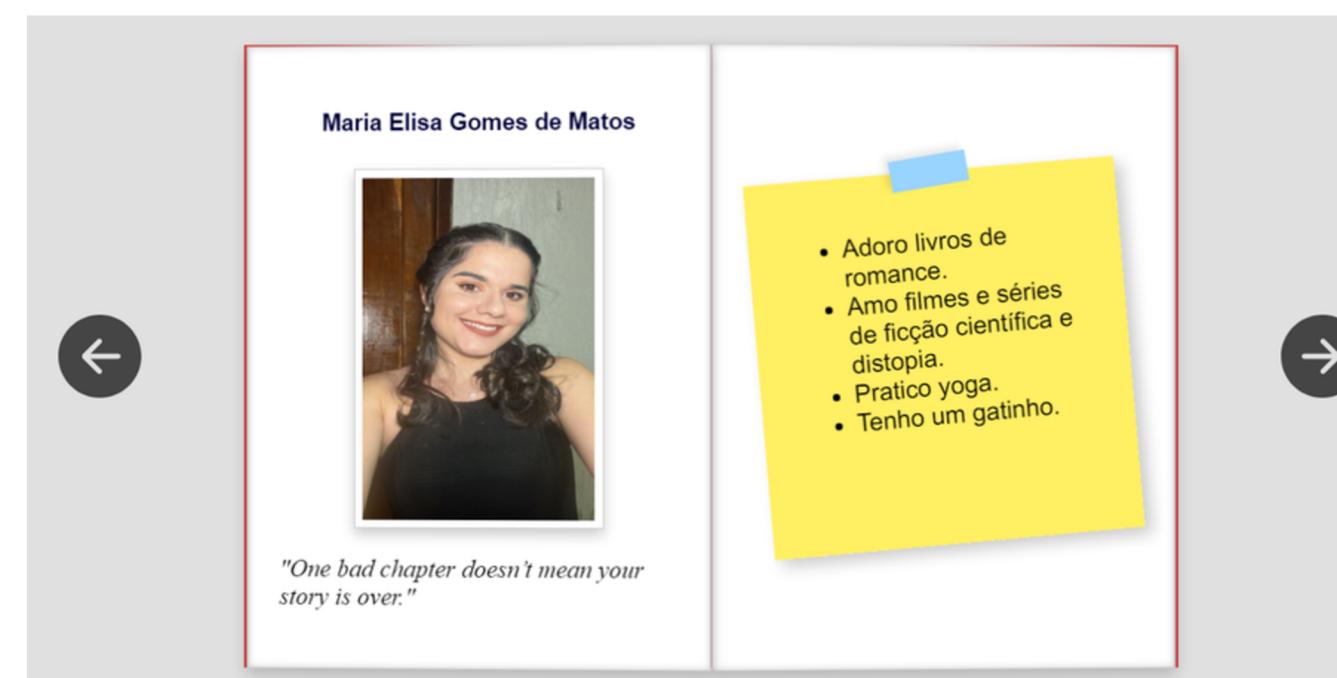
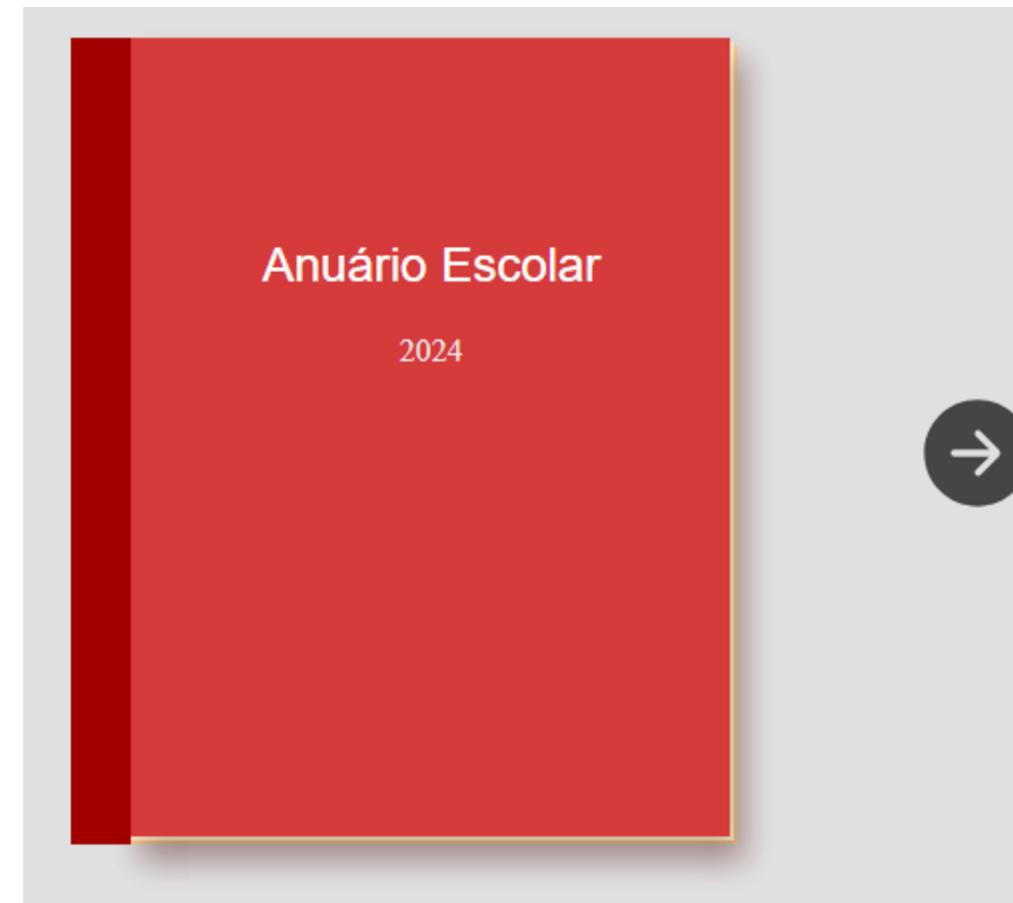
git status: Mostra o estado atual do repositório.

- Exemplo: git status
- Caso de uso: Quando você quer ver quais arquivos foram modificados e quais alterações estão prontas para serem commitadas.

Extras: git log, git checkout e commit hash.

Laboratório

Projeto Final



1. Fork do projeto no seu github:
2. Clone do projeto no seu ambiente local
3. Criei uma branch com o modelo feature/add-page-[nome-aluno]
4. Adicione um arquivo pagina[numero].html
5. Faça um commit
6. Adicione seus dados e fotos
7. Faça um commit
8. Faça o push das alterações
9. Crie um pull request
10. Faça o merge na sua master/main
11. Crie um pull request para o projeto principal

Muito Obrigada!

Caso tenham dúvida podem entrar em contato.