INTRODUÇÃO AO GIT

CURSO BÁSICO DE PROGRAMAÇÃO

MAYARA MARQUES

mmrosatab@gmail.com

SUMÁRIO

- O que é git?
- Termos e conceitos relacionados
- Por que usar git?
- Como usar?
- Instalação
- Mão na massa

O QUE É O GIT?

Git é uma ferramenta essencial para o ***controle de versão de código***, muito utilizada em desenvolvimento de software.

TERMOS E CONCEITOS RELACIONADOS

1. Repositório (Repository)

É onde todo o código-fonte e o histórico de versões de um projeto são armazenados. Pode ser local (no seu computador) ou remoto (em servidores como GitHub, GitLab, Bitbucket).

2. Commit

Um commit é uma "foto" do estado do código em um dado momento. Ele registra mudanças feitas no repositório e inclui uma mensagem de descrição. Cada commit tem um ID único.

3. Branch (Ramificação)

Uma branch é uma versão paralela do repositório, permitindo que você trabalhe em novas funcionalidades ou correções sem afetar o código principal. A branch padrão é geralmente chamada de main ou master.

4. Merge

A operação de merge combina as mudanças feitas em diferentes branches. O Git tenta mesclar as alterações automaticamente, mas, às vezes, pode haver conflitos que exigem resolução manual.

TERMOS E CONCEITOS RELACIONADOS

5. Pull Request (PR) / Merge Request (MR)

Um PR (ou MR) é uma solicitação de integração de mudanças de uma branch para outra, geralmente de uma branch de funcionalidade para a branch principal. É uma prática comum em projetos colaborativos.

6. Fork

Um fork é uma cópia de um repositório, geralmente usado para fazer mudanças em projetos públicos sem afetar o repositório original. É frequentemente usado em projetos de código aberto.

7. Clone

O clone é uma cópia exata de um repositório remoto. Você pode clonar um repositório para começar a trabalhar com ele localmente, usando o comando git clone <url>.

8. Push

O comando push envia suas alterações locais para um repositório remoto. Ele é usado para compartilhar suas alterações com outros desenvolvedores.

TERMOS E CONCEITOS RELACIONADOS

9. Pull

O comando pull recupera as alterações de um repositório remoto e as integra no seu repositório local. Ele é uma combinação de git fetch (que baixa as atualizações) e git merge (que as aplica).

10. Fetch

O fetch baixa alterações de um repositório remoto, mas não as integra automaticamente no seu repositório local. Você pode revisar as mudanças antes de decidir mesclá-las.

11. Remote

Remote se refere a repositórios que estão hospedados em servidores ou na nuvem, como GitHub ou GitLab. O termo "remote" também é usado para configurar e gerenciar as conexões com esses repositórios.

12. Staging Area (Área de Preparação)

A staging area é onde você prepara os arquivos antes de fazer o commit. Quando você executa git add <arquivo>, está movendo os arquivos para a área de preparação.

TERMOS E CONCEITOS RELACIONADOS

13. Conflict (Conflito)

Um conflito ocorre quando duas alterações incompatíveis são feitas no mesmo trecho de código em diferentes branches. O Git não pode mesclar automaticamente, e o desenvolvedor precisa resolver o conflito manualmente.

14. HEAD

HEAD é o ponteiro que refere-se ao commit mais recente na branch atual. Em um repositório Git, o HEAD indica a posição do seu repositório no tempo.

15. Checkout

O comando checkout é usado para navegar entre branches ou restaurar arquivos. Por exemplo, git checkout <branch> muda para a branch especificada.

16. Rebase

O rebase é uma alternativa ao merge, usada para aplicar mudanças de uma branch em cima de outra. Em vez de criar um novo commit de merge, ele reescreve o histórico de commits.

TERMOS E CONCEITOS RELACIONADOS

13. Conflict (Conflito)

Um conflito ocorre quando duas alterações incompatíveis são feitas no mesmo trecho de código em diferentes branches. O Git não pode mesclar automaticamente, e o desenvolvedor precisa resolver o conflito manualmente.

14. HEAD

HEAD é o ponteiro que refere-se ao commit mais recente na branch atual. Em um repositório Git, o HEAD indica a posição do seu repositório no tempo.

15. Checkout

O comando checkout é usado para navegar entre branches ou restaurar arquivos. Por exemplo, git checkout <branch> muda para a branch especificada.

16. Rebase

O rebase é uma alternativa ao merge, usada para aplicar mudanças de uma branch em cima de outra. Em vez de criar um novo commit de merge, ele reescreve o histórico de commits.

TERMOS E CONCEITOS RELACIONADOS

17. Tag

Tags são usadas para marcar pontos específicos no histórico do repositório, como versões de lançamento. Elas geralmente são usadas para marcar versões estáveis do código.

18. .gitignore

O arquivo .gitignore especifica quais arquivos ou diretórios devem ser ignorados pelo Git. Isso é útil para não versionar arquivos temporários ou de configuração que não devem ser compartilhados.

19. Diff

O comando diff mostra as diferenças entre duas versões de arquivos. Ele é usado para ver o que foi alterado entre commits ou entre a área de preparação e o repositório.

20. Blame

O comando blame é usado para ver quem fez qual alteração em uma linha de código específica. Isso ajuda a identificar o autor de uma mudança.

POR QUE USAR GIT?

Por que usar Git?

- Controle de versão: Git mantém um histórico completo de todas as alterações feitas no código, permitindo que você retorne a versões anteriores.
- Colaboração: Vários desenvolvedores podem trabalhar no mesmo projeto simultaneamente, sem sobrescrever o trabalho uns dos outros.
- Rastreamento de alterações: Você pode facilmente ver quais alterações foram feitas, por quem e por quê, graças aos commits.
- Branches: Permite trabalhar em funcionalidades ou correções isoladas, sem afetar o código principal, e depois integrar com segurança.

COMO USAR?

Configurar nome e email do usuário git

```
git config --global user.name "Seu Nome"
git config --global user.email "seuemail@dominio.com"
```

COMO USAR?

Clonar um projeto

```
git clone <url_repositorio>
```

COMO USAR?

Criar branch

```
git branch <nome_da_branch>
```

COMO USAR?

Mudar de branch

```
git checkout <nome_da_branch>
```

COMO USAR?

Criar branch e mudar para esta nova branch

```
git checkout -b <nome_da_branch>
```

COMO USAR?

Deletar uma branch local

```
git branch -d <nome_da_branch>
```

COMO USAR?

Deletar uma branch remota

```
git branch -D <nome_da_branch>
```

COMO USAR?

Adicionar arquivo modificado à lista de área de preparação (Staging Area)

```
git add
<arquivo>
```

COMO USAR?

Adicionar **todos os** arquivos modificados à lista de área de preparação (Staging Area)

```
git add .
```

COMO USAR?

Fazer um commit

```
git commit -m "Mensagem"
```

COMO USAR?

Verificar os status de seus arquivos

```
git status
```

COMO USAR?

Subir alterações para a branch remota

```
git push
```

COMO USAR?

Subir alterações para a branch remota quando a branch ainda não existe no remoto

```
git push --set-upstream origin <branch>
```

COMO USAR?

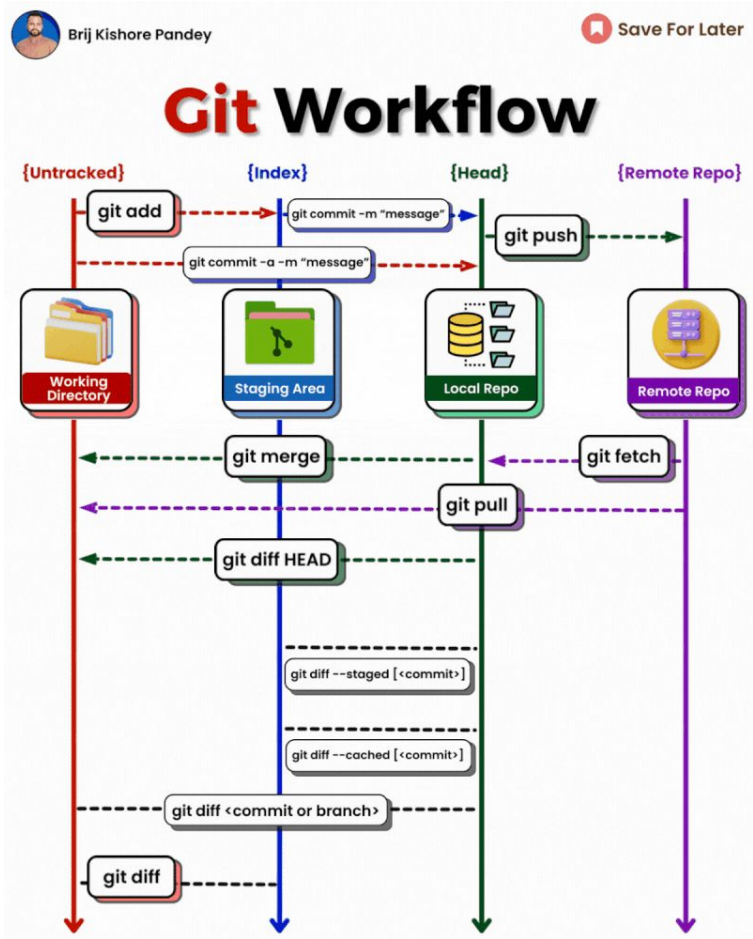
Fazer um merge

```
git merge <nome_da_branch>
```

COMO USAR?

Dicas para usar Git de forma eficaz

- Commits pequenos e frequentes: Realize commits com frequência e com mudanças pequenas para manter um histórico limpo e claro.
- Mensagens de commit claras: Escreva mensagens de commit descritivas para entender facilmente o que foi alterado.
- Branches para novas funcionalidades: Crie branches separadas para novas funcionalidades ou correções de bugs, mantendo a branch principal (geralmente main ou master) estável.
- Sincronize frequentemente: Execute git pull regularmente para manter seu repositório local atualizado com o repositório remoto.
- Evite mudanças de grande porte em uma única vez: Faça commits mais modestos e evite alterações muito grandes em um único commit.

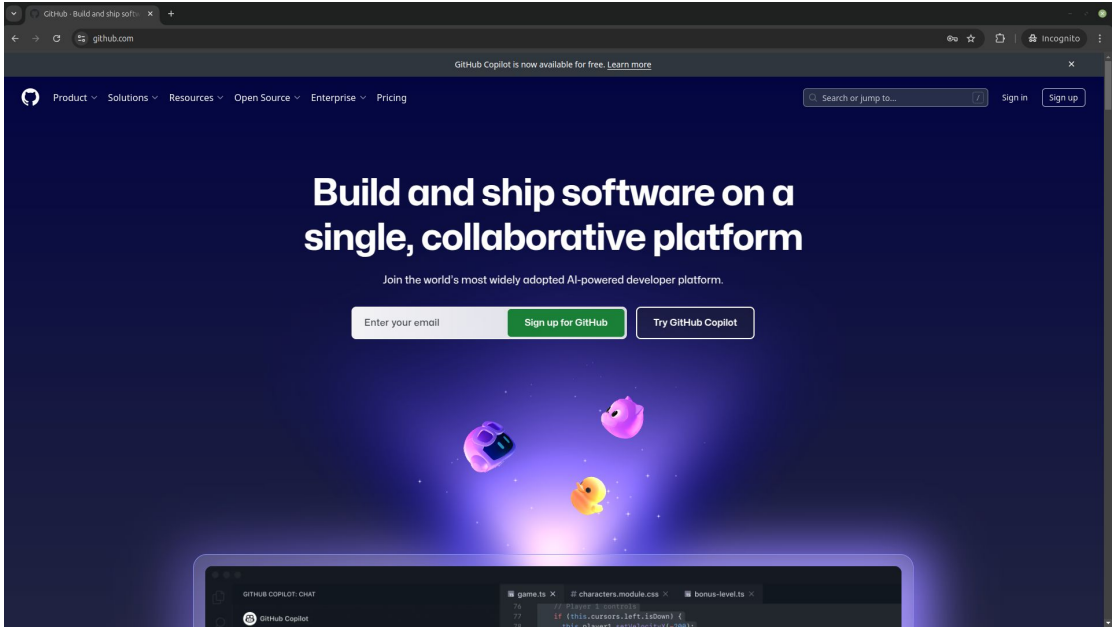


https://www.linkedin.com/feed/update/urn:li:activity:7164818604482379776?utm_source=share&utm_medium=member_desktop

O QUE É GITHUB?

GitHub é uma **plataforma** baseada na nuvem para **hospedagem** e **gerenciamento de repositórios Git**. Ele é amplamente utilizado por desenvolvedores e equipes para **controle de versão, colaboração e publicação de código**.

O QUE É GITHUB?



https://github.com/

FERRAMENTAS SIMILARES AO GITHUB

GitLab
Recursos avançados de CI/CD integrados.

Gratuito para repositórios privados e públicos.

Permite planejamento DevOps, desde a criação de issues até o deployment.

Bitbucket
Integração nativa com o Jira e outras ferramentas da Atlassian.

Focado em equipes corporativas e pequenas startups.

Possui um limite generoso para repositórios privados gratuitos.

Sourceforge
Plataforma veterana usada especialmente para projetos open-source.

Oferece recursos como análises de download e integração com sistemas de ticket.

Imagem disponível em: https://blog.geekhunter.com.br/o-que-e-commit-e-como-usar-commits-semanticos/

Bónus

Leitura recomendada O que é Commit e como usar Commits Semânticos

MÃO NA MASSA

MÃO NA MASSA

Exercício 1: Iniciar um repositório Git

1. Crie um diretório chamado `meu-repositorio`.
2. Navegue até esse diretório no terminal e inicialize um repositório Git com o comando adequado.
3. Verifique se o repositório foi inicializado corretamente com o comando para listar arquivos ocultos.

Exercício 2: Adicionar arquivos e fazer commit

1. Crie um arquivo chamado `index.html` no diretório do repositório.
2. Adicione um conteúdo simples no arquivo, como:

```
<!DOCTYPE html>
<html>
    <body>
        <h1>Hello, Git!</h1>
    </body>
</html>
```

MÃO NA MASSA

3. Use o comando `git status` para verificar o estado do repositório.

4. Adicione o arquivo ao stage (índice) e faça um commit com uma mensagem descritiva.

MÃO NA MASSA

Exercício 3: Criar e mudar para uma nova branch

1. Crie uma nova branch chamada `feature/adiciona-estilo`.
2. Faça checkout para a nova branch.
3. Confirme que você está na branch correta usando o comando `git branch`.

Exercício 4: Fazer alterações e commit em uma branch

1. Na branch `feature/adiciona-estilo`, crie um arquivo chamado `styles.css` e adicione o seguinte conteúdo:
2. Adicione e faça commit do novo arquivo.
3. Verifique o log de commits para confirmar.

```
body {
    background-color: lightblue;
}
```

MÃO NA MASSA

Exercício 5: Mesclar branches

1. Retorne à branch principal (`main` ou `master`).
2. Mescle a branch `feature/adiciona-estilo` à branch principal.
3. Verifique se o arquivo `styles.css` está presente na branch principal após o merge.

Exercício 6: Resolver um conflito de merge

1. Crie duas branches: `feature/header` e `feature/footer`.
2. Na branch `feature/header`, altere o arquivo `index.html` adicionando um cabeçalho:

```
<h2>Cabeçalho da Página</h2>
```

3. Na branch `feature/footer`, altere o mesmo arquivo adicionando um rodapé

```
<footer>Rodapé da Página</footer>
```

MÃO NA MASSA

4. Mescle as duas branches e resolva o conflito manualmente no arquivo `index.html`.

5. Faça commit da resolução do conflito.

INSTALAÇÃO

Link para download e instruções de instalação: https://git-scm.com/downloads

REFERÊNCIAS

- https://git-scm.com/doc
- https://git-scm.com/book/pt-br/v2/Come%C3%A7ando-Configura%C3%A7%C3%A3o-Inicial-do-Git