Manual de GitHub

Introdução ao GitHub

GitHub é uma plataforma de hospedagem de código-fonte e controle de versão usando Git. Ele permite que desenvolvedores ou qualquer usuário cadastrado na plataforma colaborem em projetos de software privados ou Open Source.

GitHub hospeda repositórios do Git e fornece aos desenvolvedores ferramentas para enviar um código melhor por meio das funcionalidades de linha de comando, problemas(discussões encadeadas), pull requests, revisão de código ou o uso de uma coleção de aplicativos grátis e para compra em GitHub Marketplace.

História e Evolução do GitHub

Fundação e Lançamento

- **2008:** GitHub foi lançado em abril de 2008 por Tom Preston-Werner, Chris Wanstrath, PJ Hyett e Scott Chacon. A ideia era criar uma plataforma que facilitasse a colaboração entre desenvolvedores usando Git, um sistema de controle de versão criado por Linus Torvalds.

Primeiros Anos e Crescimento

- **2008-2009:**
    - Adoção rápida pela comunidade de desenvolvedores devido à facilidade de uso e ao modelo de colaboração.
    - Introdução de funcionalidades como issues e pull requests, que ajudaram a melhorar a gestão de projetos.
- **2010:**
    - GitHub atinge 1 milhão de repositórios hospedados.
    - Começa a ganhar atenção de grandes empresas e projetos open-source de renome.

Expansão e Inovações

- **2011-2012:**
    - GitHub apresenta o GitHub Enterprise, uma versão de sua plataforma destinada a grandes organizações que precisam de uma solução privada.
    - Em 2012, GitHub atinge 2 milhões de repositórios e recebe um investimento de $100 milhões da Andreessen Horowitz.
- **2013:**
    - GitHub atinge 3.5 milhões de usuários e continua a expandir suas funcionalidades, incluindo melhor integração com outras ferramentas de desenvolvimento.

Aquisição pela Microsoft

- **2018:**
    - Em junho de 2018, a Microsoft anuncia a aquisição do GitHub por $7.5 bilhões. A aquisição gerou reações mistas na comunidade de desenvolvedores, mas a Microsoft se comprometeu a manter o GitHub como uma plataforma aberta e independente.

Desenvolvimento Recente

- **2019-2020:**
    - Lançamento de GitHub Actions, uma plataforma de automação que permite a integração e entrega contínuas (CI/CD).
    - Introdução do GitHub Codespaces, que permite que desenvolvedores configurem ambientes de desenvolvimento instantâneos na nuvem.
- **2021:**
    - GitHub atinge 56 milhões de desenvolvedores e mais de 200 milhões de repositórios. Continua a adicionar novas funcionalidades e melhorar a experiência do usuário.
- **2023:**
    - GitHub apresenta Copilot, uma ferramenta de inteligência artificial que ajuda a autocompletar código, desenvolvido em parceria com OpenAI.

Principais funcionalidades do Github

1. **Repositórios**

   Um repositório é um local onde todos os arquivos de um projeto são armazenados. Cada repositório tem um histórico de revisões, permitindo que você acompanhe e reverta alterações.

2. **Controle de Versão**

   GitHub utiliza Git para controle de versão, permitindo que múltiplas versões de um projeto sejam gerenciadas de forma eficiente.

3. **Issues (Problemas)**

Issues são usadas para rastrear tarefas, melhorias e bugs em um projeto. Elas permitem discussão e colaboração sobre um tópico específico.

4. **Pull Requests**

   Pull requests permitem que você informe aos outros sobre as mudanças que você fez em um branch (ramo) em um repositório. Elas facilitam a revisão de código e a discussão antes da fusão (merge).

5. **GitHub Actions**

   GitHub Actions permite automatizar fluxos de trabalho diretamente em seu repositório GitHub. Isso inclui integração contínua (CI) e entrega contínua (CD).

6. **GitHub Pages**

   GitHub Pages permite que você hospede sites diretamente de um repositório GitHub. É ideal para documentações e portfólios.

7. **GitHub Projects**

   GitHub Projects oferece uma visão kanban para gerenciar tarefas e projetos. É uma forma visual de organizar o trabalho.

8. **Wikis**

   Wikis em GitHub permitem a criação de documentação colaborativa para projetos. Cada repositório pode ter sua própria wiki.

Benefícios do GitHub

1. **Colaboração Facilitada**

   GitHub facilita a colaboração entre desenvolvedores ao redor do mundo. As ferramentas como pull requests, issues e wikis tornam a comunicação e a revisão de código mais eficientes.

2. **Histórico de Versões**

   O histórico de commits permite que você rastreie todas as mudanças feitas no código, quem as fez e quando. Isso é crucial para gerenciar projetos complexos.

3. **Integração com Ferramentas de Desenvolvimento**

   GitHub se integra com diversas ferramentas de desenvolvimento e serviços de CI/CD, como Jenkins, Travis CI, e muitos outros.

4. **Visibilidade e Controle**

   GitHub oferece uma visão clara de quem está trabalhando em quê, o que foi alterado, e quais são as próximas tarefas. Isso ajuda na gestão de projetos.

5. **Automação de Fluxos de Trabalho**

   Com GitHub Actions, é possível automatizar testes, builds e deploys, aumentando a eficiência e reduzindo erros humanos.

6. **Hospedagem de Sites**

   GitHub Pages permite que você crie e hospede sites de forma gratuita, facilitando a publicação de documentações e portfólios.

Configuração Inicial

Criando uma Conta no GitHub

1. Vá para github.com.
2. Clique em "Sign up" no canto superior direito.
   - Insira um nome de usuário, e-mail e senha. Clique em "Create account".
3. Verifique o e-mail que você usou para registrar a conta e siga o link de verificação enviado pelo GitHub.

Instalando o Git e Configurando no GitHub

Instalação do Git

- **Windows:**
  - **Passos:** Baixe o instalador do Git em git-scm.com e siga as instruções do instalador.
- **macOS:**

  Passos: Use o Homebrew para instalar o Git. No terminal, execute:

```sh
brew install git
```

- **Linux:**

  Passos: Use o gerenciador de pacotes de sua distribuição. Por exemplo, no Ubuntu:

```sh
sudo apt-get install git
```

Configuração Inicial do Git

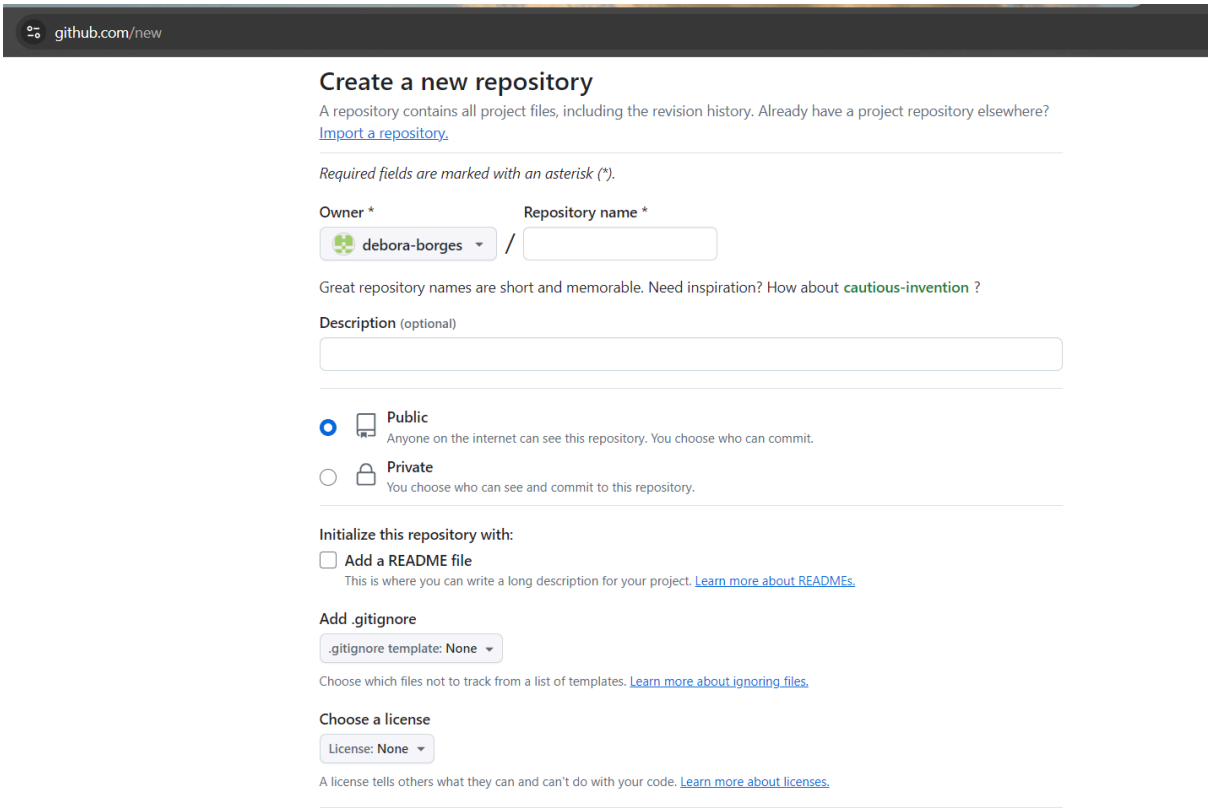
- **Nome de Usuário e E-mail:**

  Defina seu nome de usuário e e-mail para todos os commits.

```sh
git config --global user.name "Seu Nome"
git config --global user.email "seu.email@example.com"
```

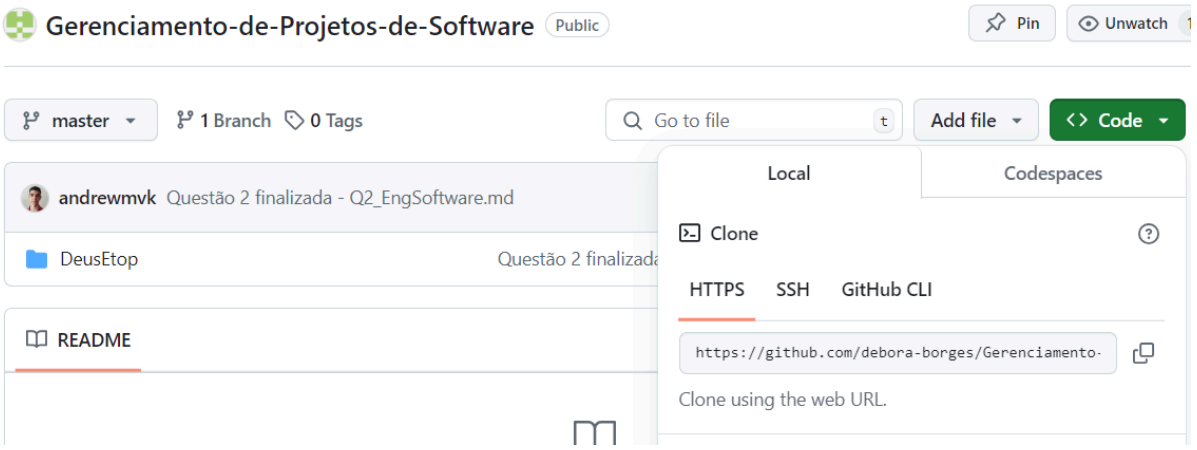
<u>Primeiros Passos: Criando e Clonando Repositórios</u>

1. **Criando um Repositório**
   - **Passos:**
     - No GitHub, clique em "New repository".
     - Preencha o nome do repositório e, opcionalmente, uma descrição.
     - Escolha a visibilidade (público ou privado).
     - Opcionalmente, adicione um README.md, .gitignore ou licença.
     - Clique em "Create repository".



2. Clonando um Repositório

- **Passos:**
  - Copie a URL do repositório que deseja clonar (disponível na página do repositório no GitHub).



  - No terminal, navegue até o diretório onde deseja clonar o repositório.

Comandos Básicos do Git

Estrutura de um repositório Git

1. **.git Directory:** Este subdiretório contém todos os arquivos de controle de versão. Ele é criado quando você inicializa um repositório com `git init`.
2. **Branches:** Branches são ramificações independentes do histórico de commits. A branch padrão é chamada `main` (ou `master` em repositórios mais antigos).
3. **Commits:** Cada alteração feita no repositório é registrada como um commit, que inclui uma mensagem descritiva e um hash único.
4. **Working Directory:** O diretório de trabalho onde você modifica os arquivos.
5. **Staging Area:** Área intermediária onde você pode preparar as alterações antes de fazer um commit.

Principais Comandos Git

1. **git init**
   - **Descrição:** Inicializa um novo repositório Git.

2. **git add**

   - **Descrição:** Adiciona arquivos ao seu repositório

3. **git commit**

   - **Descrição:** Cria um commit com as mudanças na staging area.

```sh
git commit -m "Mensagem descritiva"
```

4. **git push**

- **Descrição:** Envia os commits locais para o repositório remoto.

```sh
git push origin <nome_da_branch>
```

5. git pull

- **Descrição:** Atualiza o repositório local com as mudanças do repositório remoto.

```sh
git pull origin <nome_da_branch>
```

Gerenciamento de Branches

1. **Criar uma Nova Branch**
    - Cria uma nova branch a partir da branch atual.

```sh
git branch <nome_da_branch>
```

2. **Trocar para Outra Branch**

- Muda a branch atual para `<nome_da_branch>`.

```sh
git checkout <nome_da_branch>
```

3. **Criar e Trocar para uma Nova Branch**

   Cria uma nova branch e muda para ela imediatamente.

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

4. **Listar Todas as Branches**

   Lista todas as branches no repositório. A branch atual é marcada com um asterisco (*).

5. **Merge de Branches**

   Comando: `git merge <nome_da_branch>`

   Descrição: Mescla `<nome_da_branch>` com a branch atual.

6. **Excluir uma Branch**

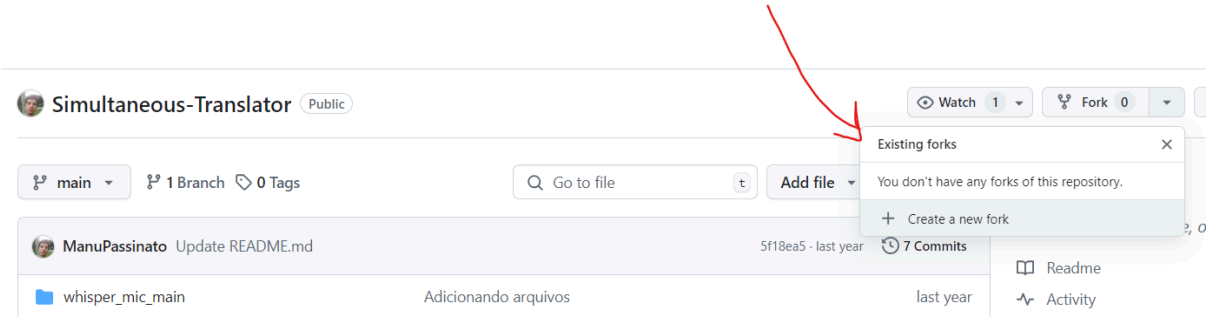
   Comando: `git branch -d <nome_da_branch>`

   Descrição: Exclui a branch especificada. Use `-D` para forçar a exclusão.

Trabalho Colaborativo

- **Forkeando um Repositório**

O fork cria uma cópia do repositório na sua conta GitHub, permitindo que você faça alterações sem afetar o repositório original.



- **Pull Requests: Como Criar e Gerenciar**

Criando um Pull Request

Pull requests permitem que você envie alterações de um branch para o branch principal (ou outro branch) de um repositório.

1. **Passos:**
   1. Faça as alterações desejadas no branch de origem e commit.

      Envie as alterações para o repositório remoto:

```sh
git push origin <nome_do_branch>
```

2.No GitHub, navegue até o repositório e clique em "Pull requests".

3. Clique em "New pull request".

4. Selecione os branches de origem e destino.

5. Adicione um título e descrição para o pull request.

6. Clique em "Create pull request".

- **Revisão de Código**

2. **Passos:**

   A revisão de código é o processo de revisar as alterações propostas em um pull request.

   - **Passos:**
     1. Navegue até a página do pull request.
     2. Revise as alterações usando a visualização de diferenças.
     3. Adicione comentários diretamente nas linhas de código alteradas, se necessário.
     4. Deixe um comentário geral sobre o pull request.
     5. Aprove ou solicite mudanças.

Merge de Pull Requests

1. **Passos:**
   - Após a aprovação das alterações, você pode fazer o merge do pull request.
   - **Passos:**
     1. Navegue até a página do pull request.
     2. Clique em "Merge pull request".
     3. Confirme o merge clicando em "Confirm merge".
     4. Opcionalmente, exclua o branch após o merge.

Resolvendo Conflitos

Detecção de Conflitos

Conflitos ocorrem quando duas alterações diferentes afetam a mesma linha de código.

Resolução de Conflitos

1. **Passos:**

Para resolver conflitos, você precisa editar os arquivos conflitantes manualmente.

   - **Passos:**

No terminal, faça o pull das últimas mudanças do branch de destino:

```sh
git pull origin <nome_do_branch_destino>
```

1. Abra os arquivos conflitantes em um editor de texto.
2. Localize as seções conflitantes, marcadas por `<<<<<<<`, `=======`, e `>>>>>>>`.
3. Edite o arquivo para resolver o conflito, mantendo as alterações desejadas.
4. Adicione as mudanças à staging area:

   ```
   git add <arquivo_conflitante>
   ```
5. Faça o commit das alterações:
   ```
   git commit -m "Resolvendo conflitos"
   ```
6. Envie as alterações para o repositório remoto:

   ```
   git push origin <nome_do_branch>
   ```

Funcionalidades Avançadas

GitHub Actions: Automatizando Fluxos de Trabalho

O Que é GitHub Actions?

- GitHub Actions permite automatizar fluxos de trabalho de desenvolvimento, como integração contínua (CI) e entrega contínua (CD).

Criando um Workflow

Um workflow é um conjunto de etapas automatizadas definidas em um arquivo YAML.

- **Passos:**
  1. No repositório, vá até a aba "Actions".
  2. Clique em "New workflow" e selecione um template ou crie um do zero.
  3. Edite o arquivo YAML para definir os passos do workflow.
  4. Commit e push do arquivo para o repositório.

Exemplo de Arquivo YAML:

```yaml
name: CI

on: [push, pull_request]

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v2
    - name: Set up Node.js
      uses: actions/setup-node@v2
      with:
        node-version: '14'
    - run: npm install
    - run: npm test
```

Executando e Monitorando Workflows

Depois de configurar um workflow, ele será executado automaticamente nos gatilhos definidos.

- **Passos:**
  1. Faça uma alteração no repositório que atenda aos gatilhos (como push ou pull request).
  2. Vá para a aba "Actions" para ver o progresso e resultados do workflow.

Issues e Projects: Gerenciamento de Tarefas e Projetos

Issues

Issues são usadas para rastrear tarefas, melhorias e bugs em um projeto.

- **Passos:**
    1. Vá para a aba "Issues" no repositório.
    2. Clique em "New issue".
    3. Preencha o título e a descrição da issue.
    4. Adicione labels, milestones, e assignees conforme necessário.
    5. Clique em "Submit new issue".

Projects

Projects permitem organizar e priorizar o trabalho com quadros Kanban.

- **Passos:**
    1. Vá para a aba "Projects" no repositório.
    2. Clique em "New project" e selecione um template ou crie um do zero.
    3. Adicione colunas como "To do", "In progress", e "Done".
    4. Adicione issues ao quadro arrastando e soltando.

GitHub Pages: Criando Sites Estáticos com GitHub

O Que é GitHub Pages?

- GitHub Pages permite hospedar sites estáticos diretamente de um repositório GitHub.

Configurando GitHub Pages

É fácil configurar um site estático com GitHub Pages.

- **Passos:**
    1. Vá para as configurações do repositório.
    2. Role até "GitHub Pages".
    3. Selecione a branch que deseja usar e o diretório (root ou /docs).
    4. Clique em "Save".

2. **Personalizando o Site**
    - **Passos:**
        1. Crie um arquivo `index.html` no repositório ou use Jekyll para temas personalizados.
        2. Commit e push dos arquivos.

Integrações e APIs

Integrações

GitHub oferece integrações com várias ferramentas de desenvolvimento, como Travis CI, Slack, e outros.

- **Passos:**
    1. Vá para as configurações do repositório.
    2. Navegue até "Integrations & services".

3. Adicione a integração desejada seguindo as instruções específicas.

APIs

GitHub oferece APIs REST e GraphQL para automatizar e interagir programaticamente com o GitHub.

- **Passos:**
  1. Crie um token de acesso pessoal nas configurações da conta.
  2. Use o token para autenticação em chamadas API.

```
curl -H "Authorization: token <seu_token>" https://api.github.com/user/repos
```

2. **Exemplo de Uso da API REST**

Listando repositórios do usuário autenticado.

```
curl -H "Authorization: token <seu_token>" https://api.github.com/user/repos
```

Boas Práticas e Dicas

Escrevendo Bons Commits e Mensagens

1. **Mensagens de Commit Claras e Concisas**
   - **Descrição:** As mensagens de commit devem ser claras, concisas e descritivas.
   - **Estrutura Recomendada:**
     - **Linha de Assunto:** Resumo breve das mudanças (máximo 50 caracteres).
     - **Corpo (Opcional):** Descrição detalhada das mudanças (máximo 72 caracteres por linha).

Exemplo:

```sql
Add user authentication

Implement user login and registration functionality. Add password hashing and session management.
```

Commits Atômicos

- **Descrição:** Cada commit deve representar uma única mudança lógica, facilitando o rastreamento de alterações e a resolução de problemas.

Uso de Verbos no Imperativo

- **Descrição:** Use verbos no imperativo na linha de assunto, como "Add", "Fix", "Update", etc.
- **Exemplo:**

```
Fix broken link in README
```

Estrutura Organizacional de Repositórios

1. **Organização de Pastas e Arquivos**
    - **Descrição:** Mantenha uma estrutura de pastas clara e organizada.
    - **Exemplo:**

1. **Documentação**
    - **Descrição:** Mantenha a documentação atualizada no repositório, incluindo um arquivo `README.md` detalhado e outros documentos relevantes.
    - **Recursos Visuais:** Exemplo de um arquivo `README.md` bem estruturado.

Segurança e Permissões

1. **Gerenciamento de Colaboradores**
    - **Descrição:** Adicione colaboradores com permissões adequadas ao repositório.
    - **Passos:**
        1. Vá para as configurações do repositório.
        2. Navegue até "Manage access".
        3. Adicione colaboradores e defina suas permissões (leitura, escrita, admin).
    - **Recursos Visuais:** Capturas de tela das configurações de acesso.
2. **Proteção de Branches**
    - **Descrição:** Proteja branches críticos, como `main` ou `master`, para evitar commits diretos e forçar revisões de código.
    - **Passos:**
        1. Vá para as configurações do repositório.
        2. Navegue até "Branches" e clique em "Add rule".
        3. Defina as regras de proteção (revisão obrigatória, testes obrigatórios, etc.).

    - **Recursos Visuais:** Capturas de tela das configurações de proteção de branches.
3. **Uso de Tokens de Acesso Pessoal**
    - **Descrição:** Use tokens de acesso pessoal para autenticação em vez de senhas.
    - **Passos:**
        1. Vá para as configurações da conta GitHub.
        2. Navegue até "Developer settings" > "Personal access tokens".
        3. Gere um novo token com as permissões necessárias.
    - **Recursos Visuais:** Capturas de tela do processo de geração de tokens.

Uso de Templates e Arquivos de Configuração

1. **Arquivo .gitignore**
    - **Descrição:** O arquivo `.gitignore` especifica quais arquivos e pastas devem ser ignorados pelo Git.

Arquivo README.md

- **Descrição:** O `README.md` é a documentação principal do repositório, explicando o propósito, instalação, uso e outros detalhes importantes.
- **Exemplo:**

````
# Nome do Projeto

Descrição breve do projeto.

## Instalação

Instruções para instalação.

```bash
comando de instalação
````

Uso
Instruções para uso.

Contribuição
Guia para contribuir com o projeto.
Licença
Detalhes da licença.
````

Templates de Issues e Pull Requests

- **Descrição:** Templates ajudam a padronizar a criação de issues e pull requests.
- **Passos:**
    1. Crie uma pasta `.github` no repositório.
    2. Adicione templates para issues (`ISSUE_TEMPLATE.md`) e pull requests (`PULL_REQUEST_TEMPLATE.md`).

Discentes:
- Debora da Silva Borges
- Andrew Medeiros Alves Cavalcante