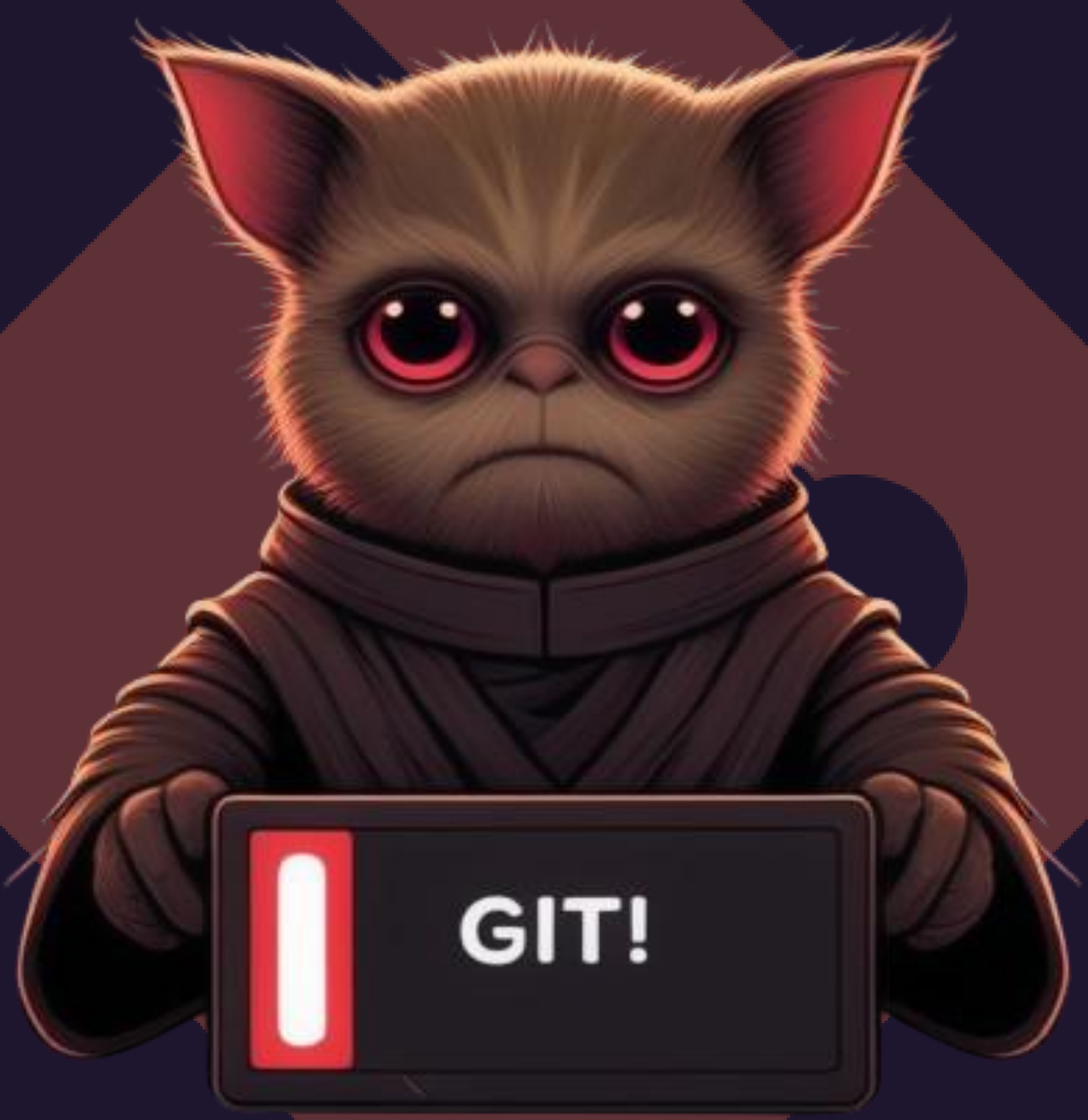
O PODER GITANO DO BRANCH

Dominando os Segredos da Lei do Controle de Versão



Luiz Victorino

E aí, galera nerd! Cansado de perder o controle do seu código? Quer virar um mestre do Git e dar um show de versionamento? Então prepare-se para desvendar os segredos dessa ferramenta épica e dominar seus projetos como um ninja!

01

CONFIGURAÇÃO INICIAL - PREPARANDO O AMBIENTE

Antes de mergulharmos de cabeça no Git, precisamos garantir que tudo está configurado direitinho. Pense nisso como configurar a nave antes de explorar o espaço sideral do código.

PREPARANDO O AMBIENTE

git config: Configurando sua Identidade

Esse comando insere seu nome na configuração global do Git. Com a opção --global, você está dizendo ao Git para usar esse nome em todos os repositórios que você clonar ou inicializar na sua máquina. É como se você estivesse gravando seu nome na base de dados do multiverso Git para ser usado em todos os commits.

```
>> git config --global user.name "Seu Nome"
```

Assim como definir seu nome, este comando define seu e-mail. O e-mail serve como um identificador universalmente único (UUID) para o seu alter-ego codificador. Toda vez que você fizer um commit, esse e-mail será associado a ele, facilitando rastrear as contribuições de cada "herói" no projeto.

```
>> git config --global user.email "seu@email.com"
```

Para verificar se seus poderes estão corretamente configurados, você pode usar:

```
>> git config --global user.name
>> git config --global user.email
```

PREPARANDO O AMBIENTE

git init: Iniciando a Missão

‘mkdir’ é a abreviação de "make directory" (criar diretório). Este comando cria um diretório com o nome especificado no caminho atual.

```
>> mkdir novo_projeto
```

‘cd’ significa "change directory" (mudar diretório). Este comando é usado para navegar até o diretório especificado.

```
>> cd novo_projeto
```

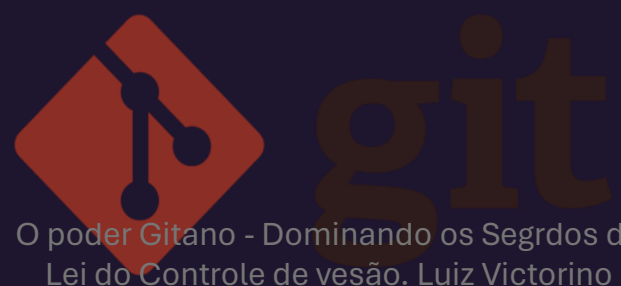
Este comando cria um novo repositório Git vazio ou reinicializa um existente. Ele cria um subdiretório chamado .git que contém todos os arquivos necessários do repositório, incluindo uma cópia completa da história do projeto.

```
>> git init
```

02

COMANDOS BÁSICOS - CONSTRUINDO SUA BASE

Agora que nossa nave está pronta, é hora de começar a construir sua base de operações no Git. Vamos ver como adicionar, confirmar e checar o status das suas mudanças.



CONSTRUINDO SUA BASE

git add: Preparando para o Lançamento

‘git add meu_script’ Este comando é como o "teletransporte" do seu arquivo para a área de staging do Git.

git add prepara (stage) as mudanças feitas em meu_script.py para o próximo commit. É como se você estivesse selecionando os arquivos que quer enviar em uma missão específica.

```
>> git add meu_script.py
```

‘git commit’ pega todos os arquivos que foram adicionados ao estágio (staged) e os "commita" com uma mensagem descritiva. A opção -m permite que você adicione uma mensagem de commit diretamente da linha de comando.

```
>> git commit -m "Adiciona função de autenticação"
```

git status mostra o estado atual do seu repositório, incluindo quais arquivos estão modificados, quais estão no estágio (staged) e quais não estão rastreados.

```
>> git status
```

03

RAMIFICAÇÕES E MERGES - EXPLORANDO NOVOS MUNDOS

Git é fantástico porque você pode criar várias linhas do tempo (branches) para diferentes funcionalidades. Vamos aprender a criar, alternar e mesclar essas branches.

EXPLORANDO NOVOS MUNDOS

git branch: Criando Universos Paralelos

‘git Branch’ é usado para criar uma nova branch. As branches permitem que você trabalhe em diferentes linhas de desenvolvimento simultaneamente, como se estivesse explorando universos paralelos no multiverso.

```
>> git branch nova_feature
```

git checkout é usado para mudar para a branch especificada. Isso é como se você estivesse saltando para uma realidade alternativa onde todas as suas futuras mudanças existirão.

```
>> git checkout nova_feature
```

git checkout main troca de volta para a branch main. É como voltar para a linha temporal principal depois de explorar um universo alternativo.

```
>> git checkout main
```

Unir as mudanças da nova branch com a principal:

```
>> git merge nova_feature
```

04

COLABORAÇÃO - CONECTANDO O MUNDO

Desenvolver em equipe é como uma missão cooperativa. Vamos ver como clonar repositórios, puxar mudanças e enviar suas alterações.

CONECTANDO O MUNDO

git clone: Clonando a Frota Estelar

‘git clone’ é usado para baixar um repositório completo de um servidor remoto (por exemplo, GitHub) para a sua máquina local. É como usar o transportador da Enterprise para trazer uma cópia exata do repositório para o seu sistema.

```
>> git clone https://github.com/usuario/repositorio.git
```

‘git pull’ é uma combinação de git fetch e git merge. Ele baixa as mudanças do repositório remoto (fetch) e as integra na sua branch atual (merge). No caso, você está puxando as mudanças da branch main do repositório remoto origin.

```
>> git pull origin main
```

‘git push’ envia todos os commits feitos na sua branch local para a branch correspondente no repositório remoto. Neste caso, você está enviando suas mudanças na branch main para o repositório remoto origin.

```
>> git push origin main
```

‘git pull’ e ‘git’ ‘push’ mantêm o histórico de commits organizado e asseguram que todas as alterações sejam devidamente registradas e compartilhadas.

05

EXPLORANDO O HISTÓRICO - DESCOBRINDO O PASSADO

Às vezes, é importante ver o que foi feito antes, como um detetive do código. Vamos explorar o histórico de commits e ver diferenças entre versões.

DESCOBRINDO O PASSADO

git log: Investigando o Diário de Bordo

git log mostra uma lista de commits no repositório, fornecendo informações detalhadas sobre cada commit, como o autor, a data, e a mensagem do commit. É como olhar no diário de bordo da Enterprise para ver todos os registros de missão.

```
>> git log
```

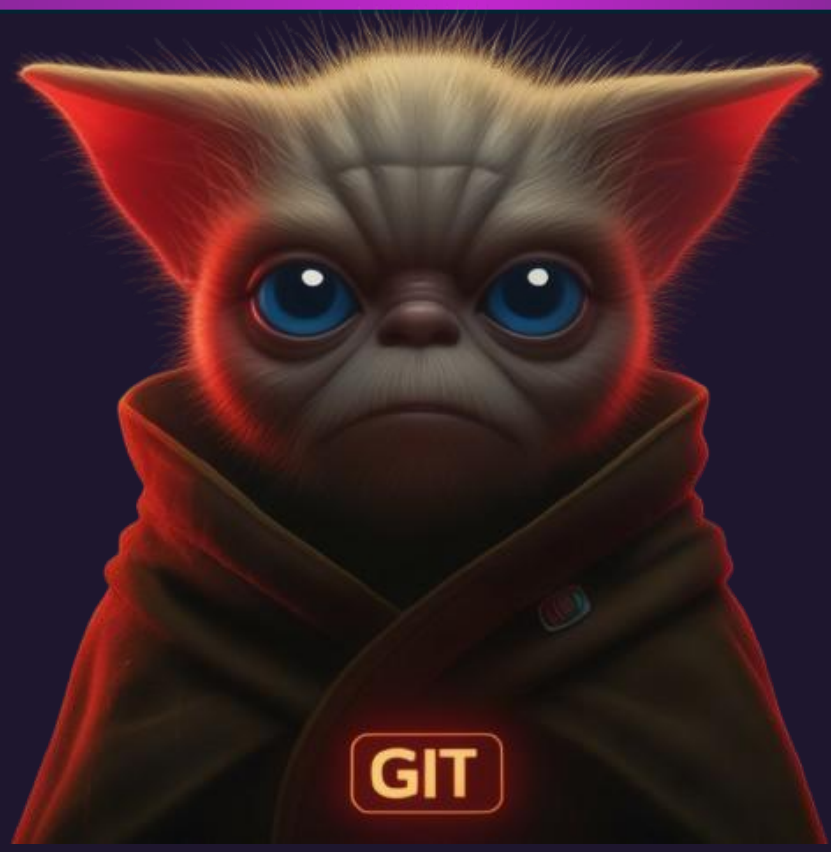
‘git diff’ HEAD mostra as diferenças entre o estado atual do seu diretório de trabalho e o commit mais recente (HEAD). É como comparar os logs de sensores da nave para ver o que mudou desde a última missão.

```
>> git diff HEAD
```

‘git blame’ atribui cada linha de um arquivo a um commit e um autor. É como um scanner de DNA que revela quem escreveu cada linha de código e quando. Isso é extremamente útil para rastrear a origem de mudanças específicas no código.

```
>> git blame meu_script.py
```

CONCLUSÃO



Parabéns, jovem padawan do código! Agora você está equipado com o conhecimento necessário para dominar o Git. Continue explorando, commitando e colaborando para se tornar um verdadeiro ninja do controle de versão. Que seus repositórios estejam sempre limpos e seus merges sem conflitos!

Quer saber mais? Confira a documentação oficial do Git Aqui .

Ebook gerado por IA e diagramado por humano.

Passo a passo se encontra no meu GitHub.

Conteúdo gerado com fins didáticos, não foi realiada validação cuidadosa e pode conter erros gerados por IA.

https://github.com/luizvictorino/criacao-ebook-com-ia-dio

Autor

Luiz Victorino

GitHub | Linkdin |