



Info
Oeste
2023

Git e Github em Ação: Criação de Repositórios e Colaboração Eficiente



1.

Introdução ao Controle de Versões e ao Git

Vamos começar com a base!



O que é o controle de versões?

É um sistema que registra as mudanças feitas em arquivos ao longo do tempo, permitindo o acompanhamento e gerenciamento de diferentes versões.

Por que usar controle de versões?

- 
- A dark blue background features white stars at the top. In the center, there are stylized white clouds. Five small grey rockets are launching vertically from behind the clouds. One rocket on the right side has a large, expressive face with wide eyes and a sad mouth looking upwards.
- 1. Rastreamento de Alterações
 - 2. Colaboração Eficiente
 - 3. Documentação de Código
 - 4. Backup de Código
 - 5. Desenvolvimento Paralelo
 - 6. Gestão de Versões
 - 7. Análise de Erros e Depuração
 - 8. Facilidade de Distribuição

Entendendo alguns termos...

Repositório (Repo): é o diretório no qual vai se trabalhar com o versionamento utilizando uma ferramenta como o git.

Staging Area: Uma área temporária onde você prepara as alterações antes de fazer um commit.

Commit: Uma operação que salva as alterações no repositório Git, criando um registro identificável e comentado do conjunto de alterações.

Branches (Ramificações): são referências que apontam para um commit específico.

Issues (Problemas): As issues são uma maneira de rastrear aprimoramentos, tarefas ou bugs para seu trabalho.

Introdução ao Git

- O Git é uma das ferramentas mais importantes e amplamente usadas no mundo do desenvolvimento de software.
- É um sistema de controle de versões distribuído que revolucionou a forma como os desenvolvedores colaboram, acompanham e gerenciam o código-fonte de seus projetos.
- O Git oferece uma abordagem eficiente e flexível para rastrear alterações em arquivos, coordenar o trabalho em equipe e facilitar o desenvolvimento de software.

Configurando o Git

- Após feita a instalação do git devemos realizar algumas configurações iniciais para termos uma assinatura em nossas alterações e algumas padronizações.
- E são elas o user.name, user.email e core.editor

```
git-bash  
git config --global user.name "Seu Nome ou username do github"  
git config --global user.email "email@example.com"  
git config --global core.editor code | nano | micro | vim
```



2.

Trabalhando com repositórios locais

Vamos botar em prática o que vimos até agora!



git init

```
git init # inicia um repositório no diretório atual  
  
git init -b | --initial-branch nome-da-branch  
# inicia o repositório em uma branch padrão específica  
  
git init nome-do-repo  
# cria um diretório e inicializa um repositório git  
dentro
```

git add

```
git add .
git add file1.txt file2.txt
# Ele é a parte essencial do fluxo de trabalho do Git e
# é usado para preparar (ou "stage") mudanças específicas
# em seus arquivos antes de criar um commit. Ele
# desempenha um papel fundamental no controle de versões,
# permitindo que você selecione quais alterações deseja
# incluir no próximo commit.
```

git status



git-bash

git status

nos permite verificar o status do seu repositório local em relação às alterações que foram feitas no seu diretório de trabalho e na área de preparação (staging area). Ele fornece informações sobre quais arquivos foram modificados, quais estão na área de preparação e o estado geral do seu repositório.

git commit

```
git commit -m | --message "mensagem aqui" # cria o commit com  
uma mensagem descritiva  
git commit --amend # atualiza o último commit com as atuais  
alterações em stage
```

git log

```
git log --reverse | -n 5 | --author="John Doe" |
--since="2023-01-01" | --oneline | --graph
# ele nos permite visualizar o histórico de commits em
um repositório Git. Ele exibe uma lista dos commits no
repositório, incluindo informações como o hash do
commit, autor, data, mensagem de commit e os pais do
commit (as alterações das quais o commit se originou).
```

git branch

```
git branch # lista todas as branches  
git branch minha-nova-feature # cria uma branch sem mudar para  
ela  
git checkout -b minha-nova-feature # cria uma branch e muda  
para ela  
git branch -d minha-nova-feature # deleta uma branch (ps.: com  
-D realiza uma exclusão forçada)  
git branch -m minha-nova-feature meu-novo-nome # renomeia uma  
branch
```

git merge

git-bash

```
## Mergin Simples ("Fast-Forward")
git checkout # Muda para a branch main
git merge minha-nova-feature # Mescla as alterações da branch minha-nova-
feature na main

## Merging com Flags Específicas
git merge --no-ff minha-nova-feature # usa-se o no-ff poi força a criação de
um novo commit de mesclagem
```

git reset

```
git reset <commit-hash>
# permite voltar para um commit anterior no histórico
do projeto. Isso desfaz os commits posteriores e
redefine a branch para o commit especificado.

git reset --hard <commit-hash>
# permite que você descarte não apenas os commits
posteriores, mas também as alterações não registradas
no diretório de trabalho.

git reset --soft <commit-hash>
# permite que você desfaça commits sem descartar as
alterações no diretório de trabalho.
```

git diff

```
git-bash

git diff
# Para ver as diferenças entre o seu diretório de
trabalho e a área de teste (staging area)

git diff --staged
# Se você deseja ver as diferenças entre a staging area
e o último commit

git diff abc123 def456
git diff minha-branch outra-branch
# Diferença entre dois commits ou duas branches

# Ele nos permite visualizar as diferenças entre
alterações no seu projeto versionado pelo Git. Ele é
uma ferramenta poderosa para comparar mudanças entre
commits, branches, arquivos ou até mesmo alterações não
registradas.
```

git cherry-pick

```
git checkout minha-outra-branch # Muda para a branch de destino  
git cherry-pick -e abc12345 # Aplica o commit com o hash "abc12345" podendo editar a mensagem do commit  
  
# Ele nos permite aplicar (ou "pegar") um commit específico de uma branch e aplicá-lo em outra branch. Isso é útil quando você deseja incorporar mudanças específicas de um commit em uma branch diferente sem mesclar toda a branch
```

git reflog



git-bash

git reflog

Por último, mas não menos importante, temos o reflog
que mantém todas as alterações que fizemos em nosso
repositório local.

.gitignore

Especifica arquivos intencionalmente não rastreados
a serem ignorados

Mas o que colocar dentro deste arquivo?

1. Arquivos de compilação e saída: `*.exe, *.so, *.o, /build/`
2. Arquivos temporários: `*~, .* .swp, .DS_Store`
3. Diretórios de dependências: `node_modules/, vendor/`
4. Arquivos de configuração: `.env, config.json`
5. Logs e arquivos de registro: `*.log, /log/`
6. Arquivos de cache: `*.cache, /cache/`
7. Diretórios de uploads ou arquivos grandes: `/uploads`
8. Arquivos específicos de IDE: `.idea/ .vscode/`



That's all Folks!

por hoje...