Git

Melhores práticas

- O nome de um **Branch** deve referenciar seu objetivo
- Faça C**ommit** graduais com descrições claras
- Cada **Pull Request** deve ser o mais granula possível
- Não seja defensivo durante um **Code Review**
- Prefira fazer **Pair Programing** sempre que possível

Site para criar .gitignore

Criar Git Igonre (gitignore.io)

```
## Gerar Chave GITHUB
ssh-keygen -t ed25519 -C "email@gmail.com"

### Gerar Chave de Identificaçao.
cat ~\.ssh\id_ed25519.pub

## Gerar Chave AZURE DEVOPS
ssh-keygen -t rsa-sha2-256 -C "email@outlook.com"

### Gerar Chave de Identificaçao.
cat ~\.ssh\id_rsa.pub
```

Comandos básicos do Git

git init

Esse comando inicia um novo repositório Git em um diretório. Aqui está como usar o **git init** de forma básica:

```
git init
```

Para criar um novo repositório enquanto especifica o nome do projeto, use o seguinte comando:

```
git init [nome do projeto]
```

git add

Esse comando é usado para preparar alterações em arquivos, preparando-os para o próximo commit:

```
git add nome_do_arquivo
```

git commit

Use esse comando para criar uma mensagem de commit para as alterações, tornando-as parte do histórico do seu projeto:

```
git commit -m "Adicionar novo recurso"
```

git status

Esse comando exibe informações importantes sobre as modificações e o status de preparação de seus arquivos.

```
git status
```

git log

Em sua forma básica, o **git log** permite visualizar uma lista cronológica do histórico de commits:

```
git log
```

git diff

Esse comando permite comparar as alterações entre o diretório de trabalho e o commit mais recente. Por exemplo, esse uso do **git diff** identifica as diferenças em um arquivo específico:

```
git diff arquivo1.txt
```

Para comparar as alterações entre dois commits, use o seguinte:

```
git diff commit1 commit2
```

git rm

Esse comando remove arquivos do seu diretório de trabalho e prepara a remoção para o próximo commit.

```
git rm arquivo1.txt
```

git mv

Use esse comando para renomear e mover arquivos em seu diretório de trabalho. Aqui está o comando do Git para renomear um arquivo:

```
git mv arquivo1.txt arquivo2.txt
```

Para mover um arquivo para um diretório diferente, digite:

```
git mv arquivo1.txt nova_pasta/
```

git config

Esse comando configura vários aspectos do Git, incluindo informações e preferências do usuário. Por exemplo, digite esse comando para definir seu endereço de e-mail para os commits:

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

O sinalizador **-global** aplica as configurações universalmente, afetando seu repositório local.

Comandos de branch e merge do Git

git branch

Use esse comando para **gerenciar ramificações em seu repositório Git**. Aqui está o uso básico do **git branch** para listar todas as ramificações existentes:

```
git branch
```

Para criar um branch do Git chamada “recurso”, use:

```
git branch recurso
```

Para **renomear um branch do Git**, digite este comando:

```
git branch -m nome-do-branch novo-nome-do-branch
```

git checkout

Esse comando permite alternar entre ramificações e restaurar arquivos de diferentes commits.

Veja abaixo como usar o **git checkout** para mudar para um branch existente:

```
git checkout nome_do_branch
```

Para descartar alterações em um arquivo específico e revertê-lo para o último commit, use:

```
git checkout -- nome_do_arquivo
```

git merge

Para mesclar um branch de recurso ou tópico no branch principal do Git, use esse comando. Abaixo está um exemplo de uso do **git merge**:

```
git merge nome_do_branch
```

git cherry-pick

Esse comando permite que você aplique commits específicos de um branch para outro sem mesclar um branch inteiro.

```
git cherry-pick commit_hash
```

git rebase

Esse comando é usado para aplicar alterações de um branch do Git em outro, movendo ou combinando commits. Ele ajuda a manter um histórico de commits mais limpo:

```
git rebase main
```

git tag

Esse comando marca pontos específicos em seu histórico do Git, como v1.0 ou v2.0:

```
git tag v1.0
```

Comandos de repositório remoto Git

git clone

Esse comando cria uma cópia de um repositório remoto em seu computador local. Um exemplo de uso básico do **git clone** é clonar um repositório do **GitHub**:

```
git clone https://github.com/username/meu-projeto.git
```

git push

Esse comando envia os commits do branch local do Git para um repositório remoto, atualizando-o com suas alterações mais recentes.

Por exemplo, se você deseja enviar as das alterações do repositório local chamado “principal” para o repositório remoto chamado “origem”:

```
git push origem principal

git push --set-upstream origin principal(criar nova branch no servidor remo
to)
```

git pull

Esse comando obtém e integra as alterações de um repositório remoto em seu branch local atual. Aqui está um exemplo de uso do **git pull** para extrair

alterações do branch principal:

```
git pull origem mestre
```

git fetch

Para recuperar novos commits de um repositório remoto sem mesclá-los automaticamente em seu branch atual, use este comando:

```
git fetch origem
```

git remote

Esse comando gerencia os repositórios remotos associados ao seu repositório local. O uso básico do **git remote** lista o repositório remoto:

```
git remote
```

Para adicionar um novo repositório remoto, especifique seu nome e URL. Por exemplo:

```
git remote add origem https://github.com/username/origem.git
```

git submodule

Esse comando é usado para gerenciar repositórios separados incorporados dentro de um repositório Git.

Para adicionar um submódulo ao seu repositório principal, use:

```
git submodule add https://github.com/username/submodule-repo.git camin
ho/do/submodulo
```

Comandos avançados do Git

git reset

Esse comando serve para desfazer alterações e manipular o histórico de commits. Aqui está um exemplo básico de uso do**git reset** para desfazer alterações:

```
git reset arquivo1.txt
```

git stash

Para armazenar alterações temporárias que ainda não estão prontas para receber o commit, use esse comando:

```
git stash
```

Para ver uma lista dos armazenamentos temporários:

```
git stash list
```

Para aplicar a alteração mais recente e removê-la da lista de alterações temporárias:

```
git stash pop
```

git bisect

Esse comando é usado principalmente para identificar bugs ou problemas no histórico do seu projeto. Para iniciar o processo de bissecção, use esse comando:

```
git bisect start
```

Usando o comando abaixo, o Git navegará automaticamente pelos commits para encontrar os que apresentam problemas:

```
git bisect run <test-script>
```

git blame

Esse comando determina o autor e a alteração mais recente em cada linha do arquivo:

```
git blame arquivo1.txt
```

git reflog

Esse comando faz um registro das alterações de um branch do Git. Ele permite que você acompanhe a linha do tempo do seu repositório, mesmo quando os commits são excluídos ou perdidos:

```
git reflog
```

git clean

Por último, mas não menos importante, esse comando remove arquivos não rastreados de seu diretório de trabalho, o que resulta em um repositório mais limpo e organizado:

```
git clean [options]
```

As **[options]** podem ser personalizadas com base em suas necessidades específicas, como **-n** para uma execução seca (*dry run*), **-f** para forçar ou **-d** para diretórios.