Curso de Versionamento de Código com Git/GitHub da DIO

Em um projeto de desenvolvimento, o código passa por diversas alterações. A cada modificação, uma nova versão é criada. Automatizar esse processo permite que os envolvidos acompanhem as mudanças, colaborem de forma eficiente e mantenham um histórico do que foi modificado, adicionado ou removido. Além disso, traz mais segurança ao projeto, caso ocorram problemas, como falhas no disco local.

Para isso, existem os sistemas de controle de versão. Um dos mais populares é o Git, que é um VCS (Sistema de Controle de Versão) distribuído. Ele funciona como um servidor que contém o banco de versões. Além de ser gratuito, é leve e rápido.

Fluxo básico no Git:

1. Clonar o repositório remoto para o repositório local (que está no seu dispositivo de armazenamento local).
2. Fazer as alterações necessárias.
3. Comitar as alterações localmente (registrá-las no seu repositório local).
4. Puxar a versão mais atualizada do código, caso haja.
5. Subir/enviar as suas alterações para o repositório remoto.

Para acessar a primeira versão, os desenvolvedores devem clonar o repositório remoto existente para um novo diretório local:

```
git clone URL-do-repositorio
```

A partir daí, os desenvolvedores podem fazer as alterações necessárias no código, gerando novas versões, prontas para serem enviadas ao repositório remoto. Para registrar essas alterações localmente, utiliza-se o comando:

```
git commit
```

Como mencionado anteriormente, o Git fornece um ambiente compartilhado, onde cada desenvolvedor tem liberdade para modificar o que desejar em seu repositório local, sem impactar imediatamente a versão que está no repositório remoto.

Para exemplificar, suponha que temos dois desenvolvedores trabalhando no back-end de um projeto, implementando funcionalidades distintas, mas que dependem de uma versão "main" atualizada. O desenvolvedor 1 termina de implementar sua funcionalidade, mas antes de subi-la para o repositório remoto, ele precisa verificar se o desenvolvedor 2 fez alguma alteração na "main". O Git permite que as alterações feitas pelo desenvolvedor 2 sejam "puxadas" para o repositório local do desenvolvedor 1 de forma simples:

```
git pull
```

Após esse comando, o desenvolvedor 1 terá a versão mais atualizada do código. Em seguida, ele pode "subir" suas alterações do repositório local para o remoto:

```
git push
```

Pronto! As alterações feitas pelo desenvolvedor 1 estarão no repositório remoto e poderão ser "puxadas" pelo desenvolvedor 2.

Uma plataforma muito conhecida pelos programadores é o GitHub, que hospeda repositórios para controle de versões com o GIT. Ele fornece uma interface para armazenar, visualizar e colaborar em projetos versionados em servidores remotos.

Aprofundando:

▼ O que é o README?

O README é um arquivo de texto, geralmente nomeado como README.md (para arquivos Markdown), que fornece informações importantes sobre um projeto e é colocado na raiz do repositório. Ele serve como uma introdução ao projeto e é frequentemente a primeira coisa que os usuários veem ao acessá-lo. Para ser eficaz, o README deve ser claro e conciso, evitando jargões desnecessários. A estrutura deve incluir seções bem definidas com títulos e subtítulos para facilitar a navegação. O uso de Markdown para formatar o texto, com listas, links e imagens, torna-o mais apresentável. Além disso, é importante mantê-lo atualizado para refletir o estado atual do projeto.

▼ Como criar um commit?

1. Adicione o conteúdo que deseja inserir no commit:

```
git add . (adiciona todos os arquivos e alterações)
git add nome-do-arquivo
```

2. Crie um commit e adicione uma mensagem descritiva:

(IMPORTANTE: é uma boa prática de versionamento manter um padrão nas mensagens, para facilitar a compreensão do que foi alterado em cada commit e ajudar a criar um histórico de mudanças mais detalhado)

```
git commit -m "feat: funcionalidade X"
git commit -m "feat: funcionalidade Y"
```

▼ Como desfazer commits?

Se você acidentalmente removeu um arquivo ou deseja desfazer alterações, existem várias abordagens.

Se você apagou um arquivo sem querer, use os seguintes comandos para restaurá-lo:

```
git status #para checar o arquivo que foi removido
git restore nome-do-arquivo #para restaurá-lo
```

Se você deseja alterar a mensagem do seu último commit, utilize:

```
git log #para ver histórico de commits
git commit --amend -m "nova mensagem" #Para alterar a mensage
```

Se você deseja desfazer um ou mais commits, pode usar o ID do commit para retornar ao estado desejado. Primeiro, obtenha o ID do commit que você deseja restaurar usando `git log`. Depois, você pode usar:

1. Para desfazer o commit, mas manter as mudanças prontas para novo commit:

```
git reset --soft id-do-commit #Desfaz o commit, mas mantém as
```

2. Para desfazer o commit, mas manter as mudanças nos arquivos, só não preparadas para commit:

```
git reset --mixed id-do-commit
```

3. Para desfazer o commit e **apagar todas as alterações** desde esse commit (cuidado, pois isso é destrutivo):

```
git reset --hard id-do-commit
```

Se você quer apenas remover os arquivos já preparados para commit (usando o git add), sem alterar o conteúdo deles:

```
git restore --staged nome-do-arquivo
```

▼ Como enviar commits do repositório local para o remoto?

Para enviar seus commits para o repositório remoto, primeiro adicione o repositório remoto (se ainda não o fez):

```
git remote add origin url-do-repositorio
```

Depois disso, basta executar o seguinte comando para enviar suas alterações:

```
git push -u origin main
```

Pronto! Suas alterações já estão disponíveis no seu repositório remoto.

▼ Como baixar os commits do repositório remoto para o repositório local?

Para baixar todas as alterações do repositório remoto para o seu repositório local, use:

```
git pull
```

▼ O que são Branches? Como trabalhar com elas?

Uma **branch** é uma ramificação do seu projeto. Por padrão, o repositório é criado com uma branch principal chamada "main" (ou, em alguns casos, "master"). Se você quiser trabalhar em funcionalidades específicas sem modificar o código da branch principal, é recomendável criar uma nova branch.

Por exemplo, se você vai trabalhar em testes no back-end, crie uma branch separada para suas mudanças.

Para criar uma nova branch, use:

```
git checkout -b nome-da-nova-branch
```

Para alternar entre branches, use:

```
git checkout nome-da-branch
```

Para listar as branches existentes e ver o último commit de cada uma:

```
git branch -v #lista o último commit de cada branch
git branch #lista todas as branches existentes
```

Para mesclar alterações de uma branch específica na branch principal, utilize:

```
git merge nome-da-branch
```

Se você quiser excluir uma branch após mesclar suas alterações com a branch principal, use:

```
git branch -d nome-da-branch-que-sera-excluida
```

▼ Como tratar conflitos de merge nas Branches?

Os conflitos de merge ocorrem quando duas ou mais branches fazem alterações conflitantes no mesmo trecho de código. Quando você tenta mesclar essas branches, o Git não consegue decidir qual versão manter e sinaliza um conflito.

Então, você deve ver quais arquivos estão em conflito:

```
git status
```

Após identificar os conflitos, abra o arquivo em um editor de texto. Você verá as seções do código marcadas pelo Git, indicando as partes conflitantes. Edite o arquivo para combinar ou escolher as alterações que deseja manter, removendo as marcações de conflito. Uma vez que você tiver resolvido todos os conflitos, salve o arquivo. Realize um novo commit, envie as alterações e finalize o merge.

▼ Trabalhando com Branches no dia a dia

Se você quiser baixar o conteúdo da branch remota sem mesclar com a branch local:

```
git fetch nome-da-branch-remota
```

Para mesclar o conteúdo da branch remota com a branch local após baixá-lo:

```
git merge nome-da-branch
```

Para clonar um repositório que possui várias branches, selecionando apenas uma delas, utilize:

```
git clone url-do-repositório --branch nome-da-branch single--
```

▼ Comandos extras

Para mostrar o status da árvore de trabalho e a área de preparação:

```
git status
```

Para visualizar o histórico de commits:

```
git log
```

Para visualizar o histórico de referências:

```
git reflog
```

Para comparar as diferenças entre duas branches:

```
git diff nome-da-branch1 nome-da-branch2
```