Laboratório: Trabalhando com Git

Objetivo: Familiarizar-se com os comandos básicos do Git através de um exercício prático.

Passos:

1. Crie um novo diretório em seu computador e navegue até ele no terminal.
2. Inicialize um novo repositório Git com `git init`.
3. Crie um novo arquivo chamado `index.html` e adicione o seguinte conteúdo:

```
<!DOCTYPE html>
<html>
<head>
    <title>Git Branch</title>
    <style>
        body {
            background-color: #ffcc00;
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
            font-family: Arial, sans-serif;
        }
    </style>
</head>
<body>
    <h1>Hello, Git master branch!</h1>
</body>
</html>
```

4. Use `git add` para adicionar `index.html` ao índice do Git.
5. Faça um commit das alterações com `git commit -m "Adicionado index.html"`.
6. Verifique o status do seu repositório com `git status`. Você deve ver que não há alterações a serem commitadas.
7. Crie uma nova branch com `git branch nova_branch`.
8. Mude para a nova branch com `git checkout nova_branch`.
9. Modifique o arquivo `index.html` para dizer "Hello, new branch!" no elemento `<h1>`.
10. Adicione e faça commit das alterações com `git add` e `git commit`.
11. Mude de volta para a branch principal com `git checkout master`.
12. Faça merge da `nova_branch` na branch principal com `git merge nova_branch`.
13. Verifique o conteúdo de index.html. Deve dizer "Hello, nova branch!" no elemento `<h1>`.

Laboratório Avançado: Gerenciando Ambientes com Git e HTML

Objetivo: Aprender a gerenciar diferentes ambientes de desenvolvimento usando branches no Git e alterando a cor de fundo de uma página HTML de acordo com a branch.

Passos:

1. No mesmo repositório criado anteriormente, crie uma nova branch para cada ambiente com `git branch DEV`, `git branch STAGE` e `git branch PROD`.
2. Mude para a branch DEV com `git checkout DEV`.
3. Modifique o arquivo `index.html` para alterar a cor de fundo para azul (por exemplo, altere a linha com `background-color` para `background-color: #0000ff;`).
4. Adicione e faça commit das alterações com `git add` e `git commit`.
5. Repita os passos 2 a 4 para a branch STAGE, alterando a cor de fundo para verde (`background-color: #00ff00;`).
6. Repita os passos 2 a 4 para a branch PROD, alterando a cor de fundo para vermelho (`background-color: #ff0000;`).
7. Agora, suponha que você tenha uma alteração que precisa ser replicada em todos os ambientes. Volte para a branch master com `git checkout master`.
8. Faça a alteração necessária no arquivo `index.html` (por exemplo, adicione um novo parágrafo `<p>Hello, all branches!</p>`).
9. Adicione e faça commit das alterações com `git add` e `git commit`.
10. Mude para a branch DEV com `git checkout DEV` e faça merge da branch master com `git merge master`.
11. Repita o passo 10 para as branches STAGE e PROD.
12. Verifique o conteúdo de `index.html` em cada branch. Deve conter a alteração comum além da cor de fundo específica de cada ambiente.

Laboratório: Resolvendo conflitos com Git

Para simular um conflito do git com 3 branches, você pode seguir os seguintes passos:

1. Em um repositório git, adicione um arquivo a ele.
2. Crie três branches diferentes (branch1, branch2, branch3)
3. Faça alterações conflitantes no mesmo arquivo em cada branch.
4. Tente mesclar as branches.
5. Aqui está um exemplo de como você pode fazer isso:

```
# Crie um novo arquivo
echo "linha 1" > arquivo.txt

# Adicione o arquivo ao repositório git
```

```
git add arquivo.txt

# Faça o commit do arquivo
git commit -m "Primeiro commit"

# Crie três novas branches
git branch branch1
git branch branch2
git branch branch3

# Vá para a branch1 e faça uma alteração
git checkout branch1
echo "linha 2 na branch1" >> arquivo.txt
git commit -am "Commit na branch1"

# Vá para a branch2 e faça uma alteração conflitante
git checkout branch2
echo "linha 2 na branch2" >> arquivo.txt
git commit -am "Commit na branch2"

# Vá para a branch3 e faça outra alteração conflitante
git checkout branch3
echo "linha 2 na branch3" >> arquivo.txt
git commit -am "Commit na branch3"

# Tente mesclar as branches
git checkout master
git merge branch1
git merge branch2
git merge branch3
```

Resolver um conflito no Git envolve os seguintes passos:

1. Identificar os arquivos com conflitos.
2. Abrir esses arquivos e procurar as linhas com conflitos.
3. Editar as linhas para resolver os conflitos.
4. Adicionar os arquivos resolvidos ao staging area.
5. Fazer commit das alterações resolvidas. Aqui está um exemplo de como você pode fazer isso:

```
# Identifique os arquivos com conflitos
git status

# Abra o arquivo com conflito em um editor de texto
# Procure por linhas que começam com <<<<<<<, =======, >>>>>>>.
# Edite essas linhas para resolver o conflito.
```

```
# Adicione o arquivo resolvido ao staging area
git add nome_do_arquivo

# Faça commit das alterações resolvidas
git commit -m "Resolvido conflito"

# Agora o conflito está resolvido e você pode continuar com o merge
```

Laboratório: Configurando o arquivo .gitignore para arquivos de variáveis de ambiente

1. Crie um arquivo chamado `.env` na raiz do seu projeto. Esse arquivo será usado para armazenar variáveis de ambiente sensíveis, como chaves de API ou senhas.
2. Abra o arquivo `.gitignore` no seu editor de texto.
3. Adicione a seguinte linha ao arquivo `.gitignore` para ignorar o arquivo `.env`:

   ```
   .env
   ```

   Isso garantirá que o arquivo `.env` não seja compartilhado no repositório.
4. Salve o arquivo `.gitignore`.

Agora, o arquivo `.env` será ignorado pelo Git e não será compartilhado no repositório. Certifique-se de nunca adicionar o arquivo `.env` ao Git, pois ele pode conter informações sensíveis que não devem ser expostas publicamente.

Laboratório: Executando um script Python com arquivo .env no terminal do Git no Windows

1. Crie um arquivo chamado `.env` na raiz do seu projeto. Esse arquivo será usado para armazenar variáveis de ambiente sensíveis, como chaves de API ou senhas.

   Abra o terminal do Git e navegue até o diretório raiz do seu projeto.

   Execute o seguinte comando para criar o arquivo `.env`:

   ```
   touch .env
   ```

   Isso criará um arquivo vazio chamado `.env` no diretório raiz do seu projeto.
2. Abra o arquivo `.env` no seu editor de texto e adicione as variáveis de ambiente necessárias, cada uma em uma nova linha. Por exemplo:

   ```
   API_KEY=your_api_key
   DATABASE_URL=your_database_url
   ```

Substitua `your_api_key` e `your_database_url` pelos valores reais das suas variáveis de ambiente.

Certifique-se de não incluir aspas ou espaços em branco desnecessários ao definir os valores das variáveis.

3. Salve o arquivo `.env`.
4. Certifique-se de ter o Python instalado no seu sistema. Você pode verificar digitando `python --version` no terminal do Git. Se o Python estiver instalado corretamente, você verá a versão do Python sendo exibida.
5. Crie um arquivo Python chamado `script.py` no diretório do seu projeto. Você pode usar o comando `touch script.py` para criar o arquivo.
6. Abra o arquivo `script.py` no seu editor de texto e adicione o código Python que você deseja executar. Por exemplo:

```
import os
from dotenv import load_dotenv

# Load environment variables from .env file
load_dotenv()

# Access environment variables
api_key = os.getenv("API_KEY")
database_url = os.getenv("DATABASE_URL")

# Use the environment variables in your code
print(f"API Key: {api_key}")
print(f"Database URL: {database_url}")
```

Agora você criou o arquivo `.env` e adicionou as variáveis de ambiente necessárias. O script Python poderá carregar essas variáveis usando a biblioteca `python-dotenv` conforme mostrado no código fornecido.

Laboratório: Utilizando Git Tags

1. Crie uma nova tag no seu repositório Git. As tags são usadas para marcar pontos específicos na história do seu projeto. Elas podem ser usadas para marcar versões estáveis, releases ou qualquer outro marco importante.
2. Para criar uma nova tag, use o comando `git tag` seguido pelo nome da tag. Por exemplo, para criar uma tag chamada "v1.0", você pode executar o seguinte comando:

   ```
   git tag v1.0
   ```

   Isso criará uma nova tag chamada "v1.0" no commit atual.

3. Você também pode adicionar uma mensagem descritiva à tag usando a opção `-a` e o comando `git tag -a <tagname> -m "<message>"`. Por exemplo:

   ```
   git tag -a v1.0 -m "Versão 1.0"
   ```

   Isso criará uma nova tag chamada "v1.0" com a mensagem "Versão 1.0" associada a ela.

4. Para visualizar todas as tags existentes no seu repositório, você pode usar o comando `git tag` sem argumentos:

   ```
   git tag
   ```

   Isso listará todas as tags existentes.

5. Para exibir informações detalhadas sobre uma tag específica, use o comando `git show` seguido pelo nome da tag. Por exemplo:

   ```
   git show v1.0
   ```

   Isso exibirá informações sobre a tag "v1.0", incluindo o commit associado a ela e a mensagem descritiva, se houver.

6. Para compartilhar as tags criadas no seu repositório remoto, você precisa fazer o push delas. Use o comando `git push` seguido da opção `--tags`:

   ```
   git push --tags
   ```

   Isso enviará todas as tags locais para o repositório remoto.

7. Agora que você criou uma tag, experimente alternar para ela usando o comando `git checkout` seguido pelo nome da tag. Por exemplo:

   ```
   git checkout v1.0
   ```

   Isso mudará o estado do seu repositório para o commit associado à tag "v1.0".

8. Lembre-se de que as tags são imutáveis por padrão. Isso significa que você não pode fazer alterações diretas em uma tag existente. Se você precisar fazer alterações em um commit marcado por uma tag, será necessário criar uma nova tag.

9. As tags são úteis para marcar versões estáveis do seu projeto, releases importantes ou qualquer outro marco significativo na história do seu código. Elas ajudam a identificar pontos específicos na linha do tempo do seu projeto e facilitam a referência a versões específicas.

10. Além das tags locais, você também pode trabalhar com tags remotas. As tags remotas são tags que foram criadas em um repositório remoto e podem ser compartilhadas com outros colaboradores do projeto.

11. Para visualizar as tags remotas disponíveis em um repositório remoto, você pode usar o comando `git ls-remote --tags <remote>` substituindo `<remote>` pelo nome do repositório remoto. Por exemplo:

    ```
    git ls-remote --tags origin
    ```

    Isso listará todas as tags remotas disponíveis no repositório remoto chamado "origin".

12. Para baixar todas as tags remotas para o seu repositório local, você pode usar o comando `git fetch --tags`. Isso atualizará o seu repositório local com as tags remotas mais recentes.

Agora que você aprendeu sobre o uso de tags no Git, experimente criar tags para marcar versões importantes do seu projeto e explore as funcionalidades adicionais que as tags podem oferecer.