Desenvolvimento de Sistemas

Controle de versão para projetos *web*

Anteriormente neste curso, você aprendeu a importância de utilizar um sistema de versionamento para seu projeto, centralizando os arquivos em um único local, mantendo um histórico de alterações e permitindo que mais desenvolvedores possam trabalhar e manipular os arquivos de um mesmo projeto. Assim, eliminam-se práticas improvisadas, como o uso de nomenclaturas diferentes em um arquivo (p. ex.: exercício1, exercício1_atualizados, exercício1_v2 etc.) para não alterações realizadas. Também se substitui o uso de ferramentas, como armazenamento comum em nuvem, por recursos adequados, como o repositório *on-line* GitHub.

Introdução

Neste conteúdo, serão relembrados alguns comandos importantes para a utilização do Git e do GitHub e de sua aplicação para projetos *web*. Na prática, você verá que não há diferenças no uso das ferramentas de versionamento quanto à natureza do projeto (*desktop, web, mobile* etc.), mas algumas características do projeto podem ser consideradas nesse processo.

Configuração e utilização

Você começará criando uma pasta em seu computador (um repositório local), então escolha o local mais apropriado conforme sua organização de projeto.

Com a pasta criada, relembre o primeiro comando. Se você tem o Git Bash instalado em sua máquina, basta abrir o terminal na pasta raiz; caso não tenha o Git Bash instalado, faça essa instalação a partir do *site* oficial do Git.

Com o terminal aberto, execute o comando: **git init**. Ele criará um repositório Git vazio na pasta em que você está localizado.

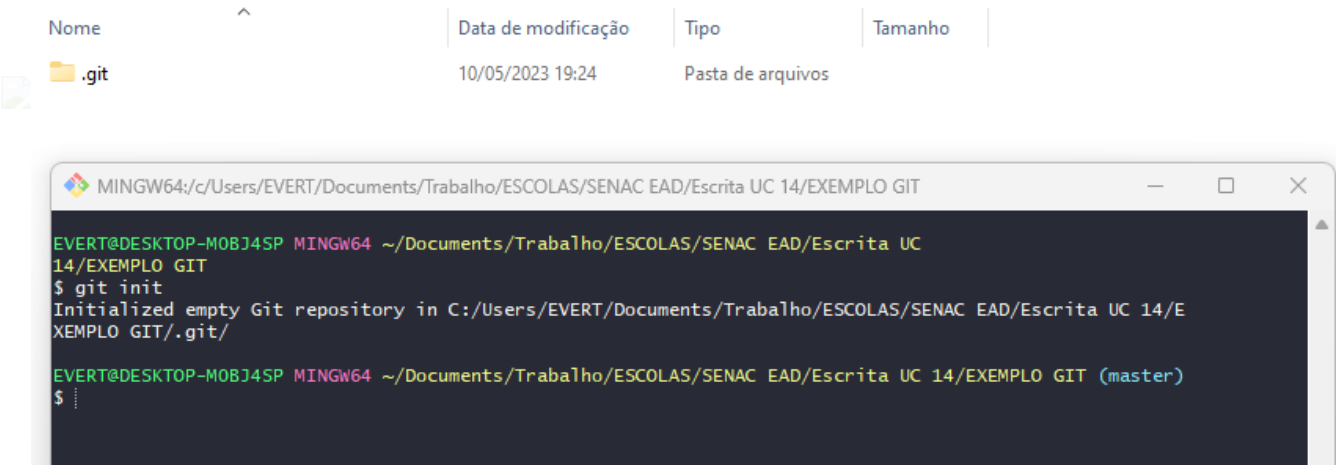

Figura 1 – Comando **git init**
Fonte: Senac EAD (2023)

O próximo comando é o **git add** . Com esse comando, os arquivos e as pastas serão adicionados ao repositório, no entanto, ainda não há nenhum arquivo ou pasta, apenas o diretório raiz.

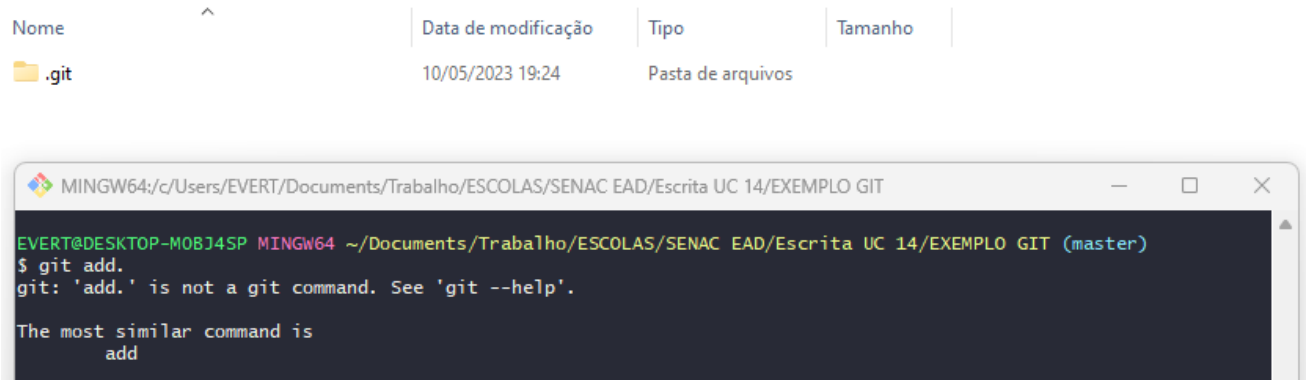

Figura 2 – Comando **git add**
Fonte: Senac EAD (2023)

Crie um arquivo README.md. Para isso, abra o seu utilitário de texto, como o Bloco de Notas, no Windows, ou similar, como o Vim ou Nano, no Linux (verifique a disponibilidade de sua distribuição), e adicione as seguintes informações: seu nome, o curso que está realizando, a unidade curricular, o polo em que está matriculado e o nome de seu tutor. Segue um exemplo de estrutura:

Informações pessoais

- **Nome:** João da Silva

- **Curso:** Técnico em desenvolvimento de sistemas

- **Unidade curricular:** Programar aplicativos computacionais com integração de banco de dados para web. (back-end)

- **Polo:** Senac Gravatai

- **Tutor:** Antônio Siqueira

Salve o arquivo com a extensão .md.

Agora repita o comando **git add .** e veja o arquivo sendo adicionado ao repositório.

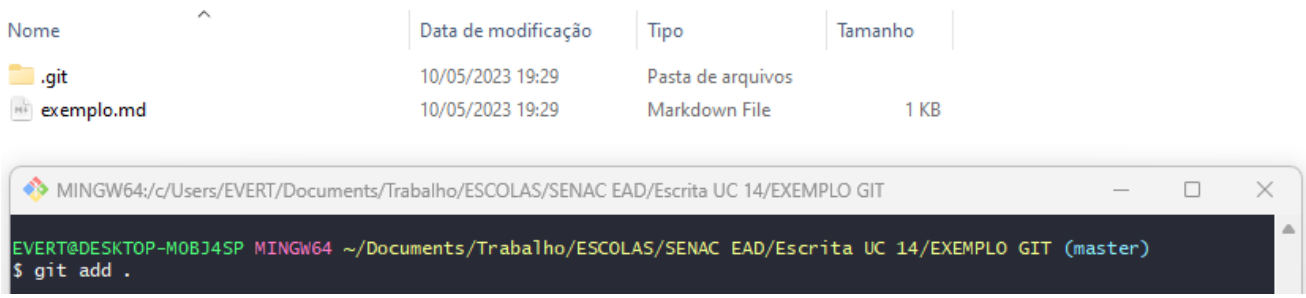

Figura 3 – Comando **git add .** após criação do arquivo **.md** Fonte: Senac EAD (2023)

O próximo passo é executar o comando git commit -m “Frase de identificação do commit”. Esse comando salvará as alterações no repositório, deixando-o pronto para uso.

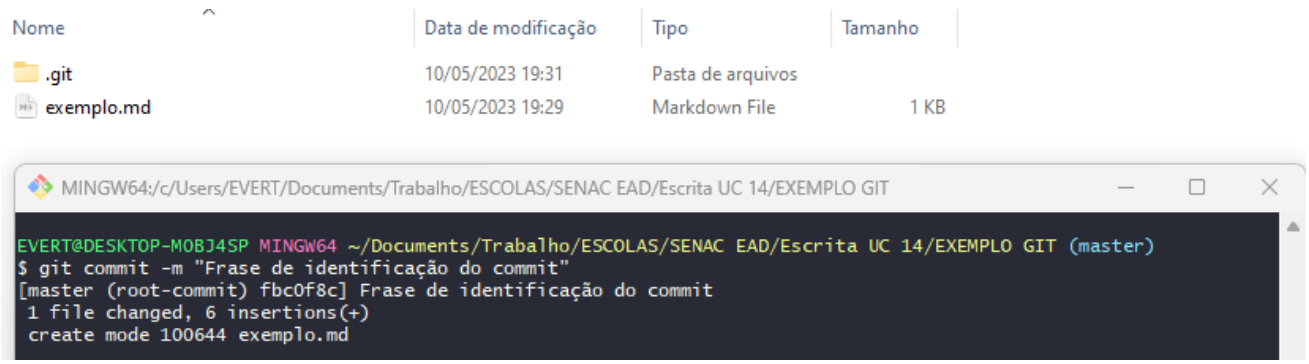

Figura 4 – Comando **git commit -m** Fonte: Senac EAD (2023)

Você acabou de criar um repositório local, porém, a intenção é trabalhar em equipe, desta forma, é preciso criar um repositório remoto e vinculá-lo ao seu. Portanto, acesse o Git Hub, efetue *login* em sua conta e escolha a opção New na página inicial. Defina um nome para o repositório e certifique-se de que a opção Initialize this repositor with: None esteja assinalada. Essa opção criará o repositório totalmente vazio, evitando problemas de conflitos, pois o repositório local já contém um arquivo README.md.

O próximo passo é conectar o repositório local com o repositório remoto, então faça o seguinte: volte à pasta do repositório em seu computador, acesse o terminal e digite o seguinte comando: git remote add origin . Esse comando vinculará o seu repositório remoto ao seu repositório local. Agora, envie o repositório local com os arquivos criados, neste

caso, o README.md, portanto execute o comando *git push* -u origin main, enviando os *commits* e arquivos de seu repositório local para o remoto. Aproveite para relembrar a funcionalidade do comando *push*, que é enviar as alterações do repositório local para o repositório remoto no Git Hub.

Comando	Descrição
git clone	Clona um repositório Git remoto ou local, existente para o seu computador.
git pull	Baixa as alterações do repositório remoto para o repositório local.
git branch	Cria e gerencia *branches* (ramificações) do projeto.
git merge	Une as alterações de diferentes *branches* do projeto.
git diff	Mostra as diferenças entre arquivos e versões do projeto.
git status	Mostra o *status* atual do repositório, incluindo os arquivos que foram modificados, adicionados ou excluídos.
git log	Mostra um histórico de *commits* do repositório, incluindo informações como data, autor e mensagem de *commit.*
git checkout	Alterna para a *branch* especificada.
git fetch	Busca as atualizações do repositório remoto, mas não as mescla com o repositório local.

Git ignore

O *git ignore* é um recurso importante para evitar que arquivos desnecessários sejam enviados para o repositório remoto, pois estes podem aumentar o tempo de espera de sincronismo de um *push* ou de um *pull*. Entende-se por arquivos desnecessários aqueles que são gerados automaticamente ao executar ou criar o projeto, como arquivos de *logs*, arquivos temporários, *caches* e arquivos de configuração (que contêm informações de configuração da aplicação).

Agora, você criará um arquivo com as regras para o *git ignore* específicas para um projeto Spring Web no NetBeans. Para isso, é necessário acessar a pasta raiz do repositório e criar o arquivo **gitignore**. Abra o arquivo e adicione:

```
# Arquivos gerados pelo NetBeans
nbproject/private/
build/
nbbuild/
dist/
nbdist/
# Arquivos gerados pelo Spring Boot
target/
mvnw
mvnw.cmd
*.jar
```

No primeiro bloco, foram ignorados os arquivos e as pastas gerados pelo NetBeans: diretórios **build, nbbuild, dist e nbist**, que armazenam os arquivos gerados pela compilação do projeto (os diretórios são indicados pelo caractere “/”).

Já no segundo bloco, ficaram de fora os arquivos de configuração criados pelo Spring Boot: o diretório **target**, que é gerado pelo **maven** contém arquivos **mvnw** e **mvnw.cmd**, que são scripts usados para construir e executar o projeto. Todo arquivo com extensão JAR (*.jar) é ignorado também.

Por fim, salve o arquivo e realize o commit para o repositório remoto.

Branches

No Git Hub, o uso de *branches* permite que sejam criadas ramificações de desenvolvimento, de modo que possibilitem aos desenvolvedores criar diferentes versões, mantendo uma versão estável e realizando alterações, correções e inclusões em outros *branches*, quando necessário. Ao final, é possível realizar a “união” dessas ramificações ao projeto principal por meio do comando **merge**.

No Git, ao criar um novo repositório, por padrão, é criado o *branch* principal, que se chama “master”, ao utilizar o comando **git *branch* testeBranch**, no qual **testeBranch** é o nome dado para a ramificação que acaba de ser criada. Com esse comando, é possível criar as *branches* necessárias, sem ter um limite de quantidade. Para alternar entre as *branches*, basta a aplicação do comando **checkout**, informando o nome do branche que deseja utilizar na sequência, ficando desta forma: **git checkout testeBranch.**

Continuous integration

A CI (integração contínua) garantirá a integração da produção realizada por diferentes desenvolvedores, ao realizar *commits* para um repositório, com o intuito de encontrar e detectar o mais cedo possível erros e falhas no projeto. Dessa forma, é possível concluir que a CI garante a integridade dos códigos desenvolvidos.

Ao utilizar o Git Hub como serviço de versionamento de repositório remoto, é possível integrar ao repositório em uso a ferramenta **GitHub Actions**, que possibilita aos usuários a criação de fluxos que validarão a execução de cada *pull*, permitindo, por exemplo, a criação de alertas automatizados sempre que um novo problema surgir no repositório.

Também é possível integrar a ferramenta **SonarQube** ao Git Actions. Essa ferramenta tem como finalidade analisar o código e auxiliar na detecção de problemas de qualidade no desenvolvimento dos projetos, permitindo a verificação de códigos com vulnerabilidades ou brechas de segurança, mal estruturação do código, entre outros problemas que podem causar transtornos na manutenção e na qualidade do projeto.

Encerramento

Neste tópico, você revisitou conceitos básicos de versionamento, como: inicializar um repositório local e remoto, realizar a conectividade entre ambos, realizar *commits* e atualizar as pastas de versionamento, assim como identificar quais arquivos ignorar. Além disso, você compreendeu a importância de manter um histórico de versões trabalhando em grupo. Você foi apresentado a ferramentas que podem auxiliá-lo a controlar a produtividade e minimizar os problemas que podem surgir em uma produção em grupo.