




Curso: Versionamento de Código com Git e GitHub

🕒	2 hrs
📁 Categoria	Curso
📌 Curso	 <u>Desenvolvimento Java com Cloud AWS</u>
📅 Date	@07/09/2023 19:56
📁 Módulo	Princípios Desenvolvimento de Software
⚙️ Status	In progress

Git

O Git é um sistema de controle de versão distribuído amplamente utilizado na programação. Ele permite que desenvolvedores acompanhem e gerenciem as mudanças em seu código-fonte ao longo do tempo. O Git armazena essas mudanças em um histórico de revisão, o que facilita o rastreamento de quem fez o quê, quando e por quê. Além disso, ele permite que múltiplos colaboradores trabalhem em um projeto simultaneamente sem conflitos significativos.

GitHub

GitHub é uma plataforma de hospedagem de código-fonte que utiliza o Git como sistema de controle de versão subjacente. Ela fornece um ambiente colaborativo para desenvolvedores compartilharem e colaborarem em projetos de software. No GitHub, você pode hospedar seus repositórios Git, colaborar com outros desenvolvedores, acompanhar problemas (issues), criar solicitações de pull (pull requests) e muito mais. É uma plataforma essencial para o desenvolvimento de código aberto e projetos de equipe.

Versionamento de Código

O versionamento de código é o processo de rastrear e gerenciar as mudanças em um código-fonte ao longo do tempo. Ele permite que você mantenha um histórico completo de todas as alterações feitas em um projeto, o que é fundamental para o desenvolvimento de software colaborativo e a manutenção de projetos de longo prazo.

Resumidamente, o processo de versionamento de código funciona da seguinte forma:

1. **Inicialização do Repositório:** Um repositório é criado para o projeto, onde o código-fonte será armazenado e controlado.
2. **Commit:** Quando você faz uma alteração no código, você a "commita" no repositório. Isso cria um ponto no histórico que registra as mudanças feitas.
3. **Histórico de Revisão:** O Git mantém um histórico de todas as revisões anteriores, permitindo que você volte no tempo para qualquer ponto e veja o estado do código naquele momento.
4. **Branches:** Você pode criar branches (ramificações) para desenvolver recursos ou correções separadamente do código principal, evitando conflitos.
5. **Merge:** Quando um desenvolvimento em um branch está pronto, você pode mesclá-lo de volta ao branch principal para incorporar as mudanças.
6. **Solicitações de Pull (Pull Requests):** No GitHub, as solicitações de pull são usadas para propor alterações no código de alguém. Elas permitem a revisão, discussão e integração de mudanças.
7. **Colaboração:** Múltiplos desenvolvedores podem trabalhar em paralelo, cada um com seu próprio branch, facilitando a colaboração em projetos de equipe.

Autenticando usuário com Token

- Crie um token e após isso insira no lugar da senha ao tentar manusear um repositório utilizando o git em sua máquina.
- Assim o git irá salvar sua credencial em sua máquina através do token, no entanto, normalmente os tokens tem uma validade e são usados para um motivo específico. Também é possível substituir o `store` por `cache` e assim salvar temporariamente sua credencial em caso de estar dividindo da máquina com alguém.

```
$ git config --global credential.helper store
```

Autenticando usuário com SSH

A autenticação SSH utiliza um par de chaves (pública e privada) para provar a identidade do usuário. A chave pública é compartilhada com o servidor remoto, e a chave privada é mantida localmente. Durante a conexão, a chave privada é usada para assinar um desafio criptográfico, e o servidor verifica essa assinatura com a chave pública. Se a correspondência for bem-sucedida, o acesso é autorizado de forma segura e criptografada. Isso oferece um método altamente seguro de autenticação em conexões remotas.

- Crie um SSH no GitHub.

```
$ ssh-keygen -t ed25519 -C "email@exemplo.com"
```

- Mantenha o local em que será salvo como padrão, apenas clique em ENTER.
- Insira uma senha, caso não queira uma basta clicar em ENTER.
- Insira a chave privada que acabou de ser criada em um ssh-agent.

```
# start the ssh-agent in the background
$ eval "$(ssh-agent -s)"
> Agent pid 59566
```

- Adicione a sua chave SSH privada ao ssh-agent.

```
$ ssh-add ~/.ssh/id_ed25519
```

- Agora adicione sua chave pública ao GitHub. Ao criar sua chave SSH foi criada a pública e a privada no repositório, então deve ser utilizada a chave pública ao criar uma chave SSH no GitHub.

Clonando repositórios

Clonando diretamente do repositório do GitHub

```
$ git clone URL-do-diretorio-remoto
```

Clonando através de um repositório local

```
$ git init -> Isso irá iniciar o repositório GIT  
$ git remote add origin URL-do-diretorio-remoto
```

- Caso o repositório já tenha arquivos nele, então:

```
$ git pull
```

- Se o repositório não tiver nada, então basta adicionar o que quiser ao commit e subir para o remoto.

```
$ git add .  
$ git commit -m "Novo commit"  
$ git push -u origin main
```

Clonando apenas uma branch do repositório remoto

```
$ git clone URL --branch nome-da-branch --single-branch
```

.gitignore

Serve para excluir arquivos que serão enviados para o repositório remoto.

Para que serve o “-u” ou “—set-upstream”

O argumento -u (ou --set-upstream) no comando git push serve para definir a ramificação remota como uma ramificação de acompanhamento (upstream branch). Uma ramificação de acompanhamento é uma ramificação local que está configurada para rastrear automaticamente uma ramificação remota específica. Isso significa que, após usar -u, você pode simplesmente usar git push ou git pull sem especificar a ramificação remota e o Git saberá para onde enviar ou de onde buscar as alterações.

git status

Mostra a branch atual e o estado dos arquivos.

Desfazendo alterações no repositório local

Como excluir um `git init` de uma pasta

```
$ rm -rf .git
```

Caso tenha modificado algum arquivo e queira que o mesmo volte ao estado do último Commit local

```
$ git restore nome-do-arquivo
```

Como alterar a mensagem do último commit

```
$ git commit --amend -m "Nova mensagem"
```

Como desfazer o último commit localmente

```
$ git log -> Copie o ID do commit que quer desfazer  
$ git reset --soft ID-do-commit
```

Pode utilizar tanto o soft quanto o mixed, a diferença é que o soft já adiciona os arquivos do último commit na área de preparação, enquanto o comportamento normal do reset que o mixed deixa os arquivos do último commit na área de trabalho.

- Caso queira voltar para um commit específico e apagar qualquer alteração posterior em seu arquivo localmente.

```
$ git reset --hard ID-do-commit
```

OBS: O comando acima deleta os arquivos posteriores ao commit selecionado. Muito cuidado.

Para observar todas as alterações feitas com mais detalhes

```
$ git reflog
```

Baixar as alterações do repositório remoto e deixar salvo, mas sem alterar seu repositório local

```
$ git fetch origin main
```

Caso queira adicionar essas alterações no repositório local

```
$ git diff main origin/main  
$ git merge origin/main
```

Criando uma nova branch mas sem as alterações da branch atual

```
$ git stash
```

Caso queira adicionar novamente alterações que foram arquivadas

```
$ git stash apply -> Para aplicar as alterações salvas no último pacote  
$ git stash pop -> Apagar as últimas alterações salvas no último pacote
```