**Estudos Git e Github**

Criando uma pasta direto pelo terminal GitBash

Comandos:

**1. mkdir git-estudo**

Cria uma nova pasta chamada git-estudo.  
📌 *Usado para organizar seus arquivos em um diretório.*

**2. cd git-estudo**

Entra na pasta git-estudo.  
📌 *Usado para navegar até o diretório onde você quer trabalhar com Git.*

**3. git init**

Inicializa um repositório Git vazio na pasta atual.  
📌 *Cria uma pasta oculta .git que guarda todo o histórico e configurações do Git.*

**4. git status**

Mostra o estado atual do repositório.  
📌 *Informa se há arquivos novos, modificados, ou prontos para commit.*

**5. git add nome-do-arquivo**

Adiciona o arquivo para a "área de preparação" (staging area).  
📌 *Diz ao Git que aquele arquivo deve ser incluído no próximo commit.*

**git add . –** adiciona todo conteúdo que está faltando ou foi alterado

**6. git commit -m "mensagem"**

Registra as alterações salvas no staging area como um ponto no histórico.  
📌 *Cria uma versão com mensagem explicando o que foi feito.*

Exemplo:

git commit -m "Adicionado o arquivo word contendo estudos git/github"

**7. git log**

Exibe o histórico dos commits realizados no repositório.  
📌 *Mostra os autores, mensagens de commit, datas e identificadores únicos.*

$ git log

commit 5c6c15a54ef8c4dc33e81f2648283dd675f8329c (**HEAD** -> **master**)

Author: kadubass2025 <kadubass@hotmail.com>

Date: Mon Apr 7 07:53:43 2025 -0300

Adicionado o arquivo word contendo estudos git/github

## 🧪 ETAPA 1.2 — Trabalhando com alterações

### 🎯 Objetivo:

* Ver como o Git detecta mudanças
* Usar git diff para ver o que foi alterado
* Fazer novos commits para salvar essas mudanças

### 8. git diff

Compara o conteúdo atual do(s) arquivo(s) com a última versão commitada.  
📌 Mostra **o que foi modificado**, **adicionado** ou **removido**, antes de fazer um novo commit.  
Se o arquivo tiver **espaços no nome**, coloque entre aspas:

git diff "Estudos Git e Github.docx"

## ✅ ETAPA 1.3 — **Remoção e recuperação de arquivos**

### 🎯 Objetivo:

Aprender como **remover arquivos com o Git**, e como **restaurar arquivos excluídos ou modificados** antes do commit. Isso vai te ajudar a não entrar em pânico quando “apagar algo sem querer” 😅

### 9. git rm "nome-do-arquivo"

Remove o arquivo do projeto e do controle de versão.  
📌 O arquivo é deletado fisicamente da pasta e o Git marca para remoção no próximo commit.

### 10. git restore "nome-do-arquivo"

Restaura um arquivo modificado ou deletado antes do commit.  
📌 O Git pega a **última versão salva** (do último commit) e recupera o arquivo.

### 11. git restore pode falhar se o arquivo ainda ****não foi commitado****

📌 Se o Git **ainda não salvou nenhuma versão** do arquivo, ele não poderá restaurar.  
📌 Sempre faça um commit ao menos **uma vez** antes de deletar arquivos importantes.

### 12. git checkout HEAD^ -- "nome-do-arquivo"

📌 Restaura a versão do arquivo a partir do **commit anterior ao atual**.  
💡 Útil quando você **já cometeu a remoção** de um arquivo e precisa recuperá-lo.

 HEAD = commit atual (onde o arquivo está **removido**)

 HEAD^ = commit anterior (onde o arquivo ainda **existe**)

git checkout HEAD^ -- "Estudos Git e Github.docx"

## ✅ ETAPA 1.4 — **Branches (ramificações)**

### 🎯 Objetivo:

Aprender a criar **ramificações (branches)** para testar novas ideias, corrigir bugs ou desenvolver recursos **sem mexer na versão principal do projeto**.

## 🧠 O que é uma branch?

Pense na **branch** como uma **cópia da sua linha de trabalho**.

Você pode:

✅ Criar uma branch nova  
✅ Testar ou desenvolver algo nela  
✅ E depois juntar de volta na principal (master ou main) quando estiver tudo ok

### 13. git branch

📌 Mostra todas as branches do projeto e qual está ativa no momento.

### 14. git branch nome-da-branch

📌 Cria uma nova branch com o nome indicado, baseada na atual.

### 14.1 git switch -c nome-da-branch

📌 Cria uma nova branch **e já muda para ela imediatamente**.  
💡 É um atalho para os comandos git branch seguido de git switch.  
Ideal para iniciar rapidamente uma nova linha de desenvolvimento.

### 15. git checkout nome-da-branch ou git switch nome-da-branch

📌 Muda para a branch especificada.

### 16. git commit -m "mensagem"

📌 Registra alterações na branch atual.

🗑️ Como excluir uma branch local no Git

### 17. git branch -d nome-da-branch

📌 Exclui uma branch local, **somente se todas as alterações já foram salvas com commit e/ou mescladas**.

### 18. git branch -D nome-da-branch

📌 Exclui uma branch local **forçadamente**, mesmo se não foi mesclada.  
⚠️ Use com cuidado para não perder trabalho não salvo.

### 19. git reflog

📌 Mostra um histórico completo de tudo que você fez no Git (commits, checkouts, merges...), mesmo o que foi apagado.

eef008c HEAD@{0}: checkout: moving from nova-ideia to master

**8d8547e** HEAD@{1}: commit: Removido arquivo para teste git rm (usa e código em negrito para recuperar a branch apagada)

### 20. git checkout -b nome-da-branch hash

📌 Cria uma nova branch a partir de um commit específico.  
💡 Útil para **recuperar branches apagadas** ou voltar no tempo.

git checkout -b nova-ideia **8d8547e**

## ✅ ETAPA 1.5 — git merge: Juntando as branches

### 🎯 Objetivo:

Aprender como unir o que foi feito em uma branch (ex: nova-ideia) com outra (ex: master), sem perder nada.

## 🧠 Conceito rápido:

O git merge é usado para **trazer as mudanças de uma branch para outra**.  
Geralmente, você faz isso da seguinte forma:

“Tô na branch master, quero juntar nela o que foi feito na nova-ideia.”

✅ **Certifique-se de estar na branch principal** (ex: master)

### 21. git merge nome-da-branch

📌 Junta o conteúdo da branch indicada na branch atual.  
💡 É assim que se unem diferentes linhas de desenvolvimento no Git.

### 22. git log --oneline --graph --all

📌 Mostra um histórico visual simplificado de todos os commits em todas as branches.  
🧠 Ótimo pra visualizar merges e divergências de branches.

## ✅ O que acontece com a branch nova-ideia depois do merge?

👉 **Nada é apagado automaticamente.**  
A branch nova-ideia ainda existe, intacta, com o histórico dela, **mas agora tudo o que foi feito nela também está presente na master**.

### Ou seja:

* A master agora tem **todo o conteúdo da nova-ideia**
* Mas a nova-ideia ainda pode ser usada separadamente se você quiser
* Só que... **ela se torna meio “inútil” se não for mais continuar trabalhando nela**

## 📌 Então o que normalmente se faz?

### ✅ ****Depois do merge, a gente costuma excluir a branch temporária****, pra manter o projeto limpo:

bash

CopiarEditar

git branch -d nova-ideia

📌 Esse comando exclui a branch local nova-ideia, **sem afetar em nada o que foi mesclado na master**.

### 23. O que acontece após o git merge?

📌 Após o merge, a branch que foi mesclada **continua existindo**, mas suas alterações agora fazem parte da branch atual (ex: master).

## 🚀 ETAPA 2 — Trabalhando com GitHub (Repositórios Remotos)

### 🎯 Objetivo:

Agora que você já domina o Git local, vamos aprender a **conectar seus projetos ao GitHub**, subir códigos, baixar, colaborar e gerenciar versões remotamente.

### 🧠 Git vs GitHub — Entenda a diferença

| **Git** | **GitHub** |
| --- | --- |
| Funciona no seu computador (local) | Plataforma online para guardar e compartilhar seus projetos |
| Salva, controla versões | Permite colaboração, portfólio, histórico de commits |
| Ex: git commit, git branch | Ex: site github.com, comandos como git push e git pull |

## ✅ Vamos praticar: subir um repositório local para o GitHub

Você provavelmente já fez isso antes, mas agora vamos **reforçar com explicações completas**!

**🛠️ PASSO A PASSO**

**1. 🖥️ Crie um novo repositório no GitHub:**

* Acesse: <https://github.com>
* Clique em ➕ "New repository"
* Nome: estudos-git-github
* **Não marque nada** (nem README, .gitignore, etc.)
* Clique em **Create repository**

**cd** /c/Git-Estudo

🔗 Conecte seu repositório local ao GitHub:

git remote add origin <https://github.com/seu-usuario/estudos-git-github.git>

**Se já tiver um origin configurado**, use:

git remote set-url origin <https://github.com/seu-usuario/estudos-git-github.git>

📤 Suba o projeto pro GitHub:

git push -u origin master

### ✅ Pronto! Agora seu projeto estará online no GitHub

Você pode acessar no navegador, compartilhar o link e usar como **portfólio**!

### 25. git remote add origin URL

📌 Conecta seu repositório local a um repositório remoto (no GitHub).  
💡 A URL pode ser HTTPS ou SSH.

### 26. git push -u origin nome-da-branch

📌 Envia os commits da sua branch local para o GitHub.  
💡 O -u salva essa referência para os próximos pushs.

### 27. git remote set-url origin nova-url

📌 Atualiza a URL do repositório remoto se precisar corrigir ou trocar.

## ✅ 2. Precisa mudar a **branch** de master para **main**?

### 🧠 Resposta curta: ****não é obrigatório****, mas é ****recomendado****.

### 📝 Explicação:

* Antigamente, o Git usava master como nome padrão da primeira branch.
* Hoje, o padrão mudou para main (mais neutro e moderno).
* O GitHub já cria repositórios novos com main.

### ✅ Vantagens de usar main:

* Evita confusões quando você clona repositórios modernos
* Fica alinhado com o padrão atual
* Mais profissional para portfólio e trabalho em equipe

🔄 Como mudar de master para main (se quiser):

git branch -m master main

git push origin -u main

Depois, no GitHub, vá em **Settings > Branches** e defina main como padrão.

### 28. git branch -m master main

📌 Renomeia a branch atual de master para main.  
💡 Recomendado para seguir o padrão moderno do GitHub.

### 29. git push -u origin nome-da-branch

📌 Envia a branch e **vincula ela à branch remota** no GitHub.  
💡 Assim, nos próximos push/pull você pode usar só git push.

✅ COMANDOS COM **CD** (change directory)

### 30. cd ..

📌 Volta uma pasta na estrutura.

### 31. cd nome-da-pasta

📌 Entra na pasta especificada.

### 32. cd nome1/nome2

📌 Entra em subpastas encadeadas.

### 33. cd (sozinho)

📌 Vai para a pasta pessoal do usuário.

### 34. cd ~

📌 Vai para a pasta pessoal (atalho igual ao anterior).

### 35. cd /

📌 Vai para a raiz do sistema.

### 36. cd -

📌 Volta para o diretório anterior.

### 37. ls

📌 Lista os arquivos e pastas da pasta atual.

### 38. git clone URL

📌 Copia um repositório inteiro do GitHub para o seu computador.  
💡 Ideal para baixar projetos de terceiros ou começar a trabalhar em um projeto já existente.

### 🎯 Objetivo:

Copiar um repositório inteiro que está no GitHub **para o seu computador**, com todo o histórico de commits, branches, arquivos, etc.

### 🧠 Quando usar git clone?

* Quando você quer **baixar um projeto pronto**
* Quando você quer estudar um repositório que viu no GitHub
* Quando vai contribuir com um projeto open source ou da empresa

## 🛠️ PASSO A PASSO

### 1. 🔗 Pegue a URL do repositório no GitHub

Exemplo (pode usar esse pra testar):

👉 <https://github.com/kadubass2025/CursoJava.git>

### 2. 🌍 No terminal, escolha uma pasta onde você quer salvar o projeto

cd /c/Users/Ricardo/Documentos/Projetos

**3. 🧬 Use o comando git clone:**

git clone https://github.com/kadubass2025/CursoJava.git

📌 Isso vai:

* Criar uma nova pasta chamada CursoJava
* Baixar todos os arquivos do repositório
* Trazer o histórico de commits
* Já deixar o repositório conectado ao GitHub (com origin configurado)

### 4. 🔍 Entre na pasta:

bash

cd CursoJava

E rode:

bash

git status

git log

Pra ver que já é um repositório Git completo.

## ✅ 39. git pull — Atualizando seu repositório local com o remoto

### 39. git pull

📌 Atualiza o repositório local com as mudanças do repositório remoto.  
💡 Une o que está no GitHub com o que está no seu computador.

### 🎯 Objetivo:

Baixar as alterações mais recentes do repositório remoto (GitHub) para o seu repositório local.

### 🧠 Quando usar git pull?

* Quando **outra pessoa** fez mudanças no GitHub
* Quando **você mesmo alterou algo no GitHub** (README.md, por exemplo)
* Quando quer garantir que está com a **versão mais atualizada**

## 🛠️ EXEMPLO PRÁTICO

### 1. 🌐 Suponha que alguém alterou algo no GitHub (ou você mesmo editou o README lá direto)

### 2. Vá no terminal na pasta clonada:

bash

cd /c/Projetos-github/SpringBoot2CascadeDropDownEx

### 3. Rode:

bash

git pull

📌 O Git vai:

* Conectar com o GitHub
* Comparar os commits
* Baixar os novos arquivos ou atualizações

**Observação importante:**

* 🧠 O git pull na verdade é um atalho para dois comandos:

git fetch

* git merge
* Ou seja, ele **busca** do GitHub e depois **mescla** com sua branch atual.

## ✅ 41. .gitignore — O que é e pra que serve?

### 41. .gitignore

📌 Arquivo onde você define o que o Git deve ignorar.  
💡 Impede que arquivos como .idea/, out/, \*.class, etc., sejam enviados ao GitHub.  
🔐 Também usado para ocultar arquivos com dados sensíveis.

### 🎯 Objetivo:

Evitar que arquivos **desnecessários ou sensíveis** sejam enviados ao GitHub (ex: pastas .idea/, arquivos temporários, backups, etc.)

## 🧠 Por que usar .gitignore?

* Evita subir arquivos **do seu editor** (ex: .idea/, out/, .vscode/)
* Não polui o GitHub com arquivos que **ninguém precisa ver**
* Protege arquivos locais, como chaves de API, senhas, configs pessoais

## 🛠️ COMO CRIAR E USAR .gitignore

### 1. Crie um arquivo chamado:

bash

.gitignore

No terminal, ou direto pelo VSCode:

* Clique com o direito > Novo arquivo > .gitignore

### 2. Adicione os arquivos ou pastas que você quer ****ignorar****

Aqui vai um exemplo clássico pra projetos Java com IntelliJ:

csharp

# Pasta do IntelliJ IDEA

.idea/

\*.iml

# Saída de compilação

out/

# Arquivos temporários

\*.class

\*.log

\*.tmp

~\*

# Sistema operacional

.DS\_Store

Thumbs.db

### 3. Salve o arquivo

Depois de criado, o Git **ignora tudo que estiver listado ali**, desde que **você ainda não tenha feito commit desses arquivos antes**.

### 4. Se já cometeu os arquivos antes, use:

bash

git rm -r --cached nome-da-pasta

Exemplo:

bash

git rm -r --cached .idea

git rm -r --cached out

Isso **remove os arquivos do Git**, mas **não apaga do seu computador.**

### 5. Agora commit e push normalmente:

bash

git add .gitignore

git commit -m "Adicionado .gitignore para limpar arquivos desnecessários"

git push

## ✅ **Fork & Pull Request** – colaboração em projetos reais

### 🎯 Pra que serve isso?

Esses dois conceitos são usados **quando você quer contribuir com um projeto de outra pessoa** ou com um projeto **open source**.

É o que você faria, por exemplo, se quisesse ajudar num repositório famoso no GitHub ou colaborar com outro dev.

## ✅ 42. FORK — Copiar um projeto de outra pessoa pra sua conta

### 🧠 O que é?

O **Fork** cria uma **cópia completa** de um repositório que está em outra conta **pra dentro da sua conta do GitHub**, com todos os arquivos e histórico.

### 👣 Como fazer:

1. Acesse um repositório público de outra pessoa no GitHub  
   (exemplo: <https://github.com/spring-projects/spring-boot>)
2. Clique no botão **Fork** no topo direito
3. Escolha sua conta para criar a cópia

Pronto! Agora esse projeto está na sua conta e você pode fazer alterações como quiser.

## ✅ 43. PULL REQUEST — Pedir pra sua alteração ser adicionada ao projeto original

### 🧠 O que é?

Depois de fazer alterações no projeto que você "forkou", você pode pedir ao **autor original** para revisar e aceitar suas alterações. Esse pedido é o **Pull Request (PR)**.

### 👣 Como funciona:

1. Faça um **Fork** do projeto
2. Clone o repositório na sua máquina
3. Crie uma nova branch, faça alterações
4. Dê git push
5. Vá no GitHub → Vai aparecer um botão verde:  
   **“Compare & pull request”**
6. Clique, escreva uma mensagem explicando a alteração e envie

Agora o dono do repositório vai revisar e, se aprovar, ele **aceita seu código no projeto original** 🎉

### 42. Fork

📌 Cópia de um repositório de outro usuário do GitHub para a sua conta.  
💡 Serve para você estudar ou contribuir com o projeto sem alterar o original.

### 43. Pull Request

📌 Requisição para que suas alterações em um repositório forkado sejam adicionadas ao projeto original.  
💡 Muito usado para colaborar com projetos open source.

# ✅ 44. git tag – marcando versões no seu projeto

## 🧠 O que é git tag?

Serve pra marcar **momentos importantes** do seu projeto, como:

* Primeira versão estável (v1.0)
* Grande atualização (v2.0)
* Versão pronta pra produção (release-1.0)

📌 Ele não altera nada no código, é só um **marcador de referência** no histórico do Git.

## ✅ Exemplo prático

Imagine que você finalizou seu projeto, testou tudo e agora quer marcar a **versão 1.0**:

bash

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

🔍 Explicando:

* -a v1.0 → nome da tag
* -m → mensagem descritiva

## ✅ Ver todas as tags criadas:

bash

git tag

## ✅ Subir a tag pro GitHub:

bash

git push origin v1.0

🔁 Isso manda a tag pro repositório remoto (GitHub), e ela aparecerá como uma versão no projeto.

## ✅ Excluir uma tag (se quiser remover):

bash

git tag -d v1.0 # Apaga local

git push origin :refs/tags/v1.0 # Apaga do GitHub

### 44. git tag

📌 Usado para marcar versões importantes no histórico do projeto.  
💡 Exemplo: v1.0, v2.1-beta, release-2025.

bash

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

git push origin v1.0

### 💼 Por que as tags (git tag) são tão importantes nas empresas?

1. **Organização de versões**
   * Toda entrega ou versão importante do sistema recebe uma **tag**, tipo:

arduino

v1.0.0

v2.3.1

release-2025-04

1. **Deploys e entregas**
   * Muitas empresas fazem **deploy (publicar o sistema)** baseado em uma tag específica.
   * Ex: o servidor de produção puxa sempre a v3.2.
2. **Rastreamento de bugs**
   * Se der erro numa versão específica, a equipe consegue ver **exatamente como estava o código naquela tag**.
3. **Controle de versões entre ambientes**
   * Ex: a tag v2.1-rc (release candidate) está em homologação.
   * Depois de testada, criam a v2.1 e ela vai pra produção.

## 🔁 Tags são como "checkpoints" no projeto

Elas são muito úteis pra:

✅ Equipe de QA (testes)  
✅ Equipe de DevOps (implantação)  
✅ Devs que precisam saber “o que mudou da versão 2.0 pra 2.1”

## ✅ Próximo Tópico:

# 46. **Conectando seu repositório ao VSCode com SSH (e não mais com HTTPS)**

### 🧠 Por que isso importa?

Atualmente você está usando comandos como:

bash

git clone https://github.com/kadubass2025/seu-projeto.git

Esse método usa **HTTPS**, que exige login/senha ou token toda vez.

➡️ Isso cansa, trava automações e exige mais esforço.

### ✅ Solução moderna: ****usar SSH****

Com SSH, você não precisa digitar nada — o Git e GitHub se conectam **de forma segura e automática**.

## 🛠️ Bora fazer agora? Você vai:

1. Criar sua chave SSH (caso ainda não tenha)
2. Adicionar a chave no GitHub
3. Testar conexão
4. Clonar usando SSH

### ⚠️ Posso verificar primeiro se você já tem uma chave SSH criada?

No seu terminal (Git Bash), digita:

bash

ls ~/.ssh

E me diz se aparece algo como:

rust

id\_rsa

id\_rsa.pub

## ✅ Agora o próximo passo:

### 1️⃣ Copiar a ****chave pública**** pro GitHub

Roda esse comando pra copiar o conteúdo da chave pública:

bash

cat ~/.ssh/id\_ed25519.pub

Ele vai exibir algo assim:

css

ssh-ed25519 AAAAC3...kadubass@Ricardo

### 2️⃣ Vá no GitHub e adicione sua chave SSH:

* Acesse: <https://github.com/settings/keys>
* Clique em **"New SSH key"**
* No campo **Title**, coloca: Chave SSH Ricardo
* No campo **Key**, cola o que você copiou do terminal (id\_ed25519.pub)
* Clica em **"Add SSH key"**

### 3️⃣ Agora teste a conexão com o GitHub via SSH:

bash

ssh -T git@github.com

Se tudo estiver certo, o terminal vai dizer algo como:

vbnet

Hi kadubass2025! You've successfully authenticated, but GitHub does not provide shell access.

✅ Isso significa: **autenticado com sucesso!**

Depois disso, você pode clonar e fazer push assim:

bash

git clone git@github.com:kadubass2025/seu-repo.git

E **nunca mais precisa digitar token, login ou senha**! 😎

Vamos agora integrar o **Git com o IntelliJ IDEA**, que é essencial pra quem trabalha com **Java e projetos em equipe**.

## ✅ Objetivo:

Ter o Git funcionando dentro do IntelliJ, podendo fazer **commit, push, pull, criar branch, resolver conflitos**, tudo visualmente — sem precisar usar o terminal o tempo todo.

## 🔧 Passo a passo da Integração Git + IntelliJ

### ✅ 1. Verifique se o Git está configurado no IntelliJ:

1. Abra o IntelliJ
2. Vá em **File > Settings** (ou Ctrl + Alt + S)
3. Navegue até:  
   Version Control > Git
4. Verifique se o caminho do Git está detectado. Normalmente é:

makefile

CopiarEditar

C:\Program Files\Git\bin\git.exe

1. Clique em **Test**  
   → Se aparecer **"Git executed successfully"**, tá pronto!

### ✅ 2. Abra um projeto com Git (ou clone um):

Se você já tem seu projeto local com Git:

* Abra o projeto normalmente
* O IntelliJ detecta e ativa o controle de versão automaticamente

Se quiser **clonar direto via SSH** no IntelliJ:

1. Vá em **File > New > Project from Version Control**
2. Cole a URL SSH do seu GitHub:

scss

CopiarEditar

git@github.com:kadubass2025/Estudos-Git-Github.git

1. Clique em **Clone**

### ✅ 3. Confirme que o Git está ativo no projeto:

* No menu superior, vá em **VCS (Version Control System)**
* Veja se aparece opções como:
  + Commit
  + Git > Branches
  + Push, Pull, Log, etc.

Se isso tudo estiver disponível, o Git está 100% integrado ao seu IntelliJ.

### ✅ 4. Agora use Git pelo IntelliJ!

* 🟢 **Commit** → Ctrl + K
* 🔼 **Push** → Ctrl + Shift + K
* 🔽 **Pull** → VCS > Git > Pull
* 🌿 **Branch** → canto inferior direito (ícone com o nome da branch)
* 📋 **Log** → Alt + 9 (abas inferiores → Version Control)

1. git init Guia Completo de Comandos Git e GitHub - Ricardo Santos Inicializa um novo repositório Git na pasta atual.

2. git status Mostra o estado atual do repositório, arquivos modificados, não rastreados ou prontos para commit.

3. git add Adiciona arquivos ao staging area para serem commitados. 4. git add . Adiciona todos os arquivos modificados ao staging.

5. git commit -m 'mensagem' Salva as alterações do staging com uma mensagem descritiva.

6. git log Mostra o histórico de commits. 7. git diff Mostra as diferenças entre arquivos modificados e o último commit. 8. git restore Restaura o arquivo modificado para o estado anterior. 9. git rm Remove um arquivo do repositório e do sistema de arquivos. 10. git rm --cached Remove um arquivo do repositório mas mantém no sistema local. 11. git branch Lista todas as branches. 12. git branch Cria uma nova branch. 13. git switch Troca para outra branch. 14. git checkout -b Cria e já muda para a nova branch. 15. git merge Mescla a branch mencionada com a atual. 16. git branch -d Deleta uma branch local. 17. git checkout Navega para um commit específico. 18. git checkout -b Cria uma branch a partir de um commit passado. 19. git remote add origin Conecta seu repositório local ao remoto no GitHub. 20. git remote -v Exibe os repositórios remotos configurados. 21. git push -u origin main Envia sua branch local main para o repositório remoto. 22. git push Envia os commits para o repositório remoto (após -u). 23. git pull Puxa as últimas alterações do repositório remoto. 24. git pull origin main Atualiza o repositório local com a branch main do remoto. 25. git clone Clona um repositório remoto na sua máquina. 26. git log --oneline Mostra o log de commits de forma resumida. 27. git reflog Mostra todas as referências recentes, inclusive mudanças de branch. .gitignore Arquivo que define quais arquivos e pastas o Git deve ignorar. 28. git tag -a v1.0 -m 'mensagem' Cria uma tag anotada com nome e descrição. 29. git tag Lista todas as tags. 30. git push origin v1.0 Envia a tag para o GitHub. 31. git tag -d v1.0 Apaga a tag local. 32. git push origin :refs/tags/v1.0 Apaga a tag do GitHub. 33. git checkout v1.0 Acessa o estado do projeto naquela tag. 34. Fork Ação feita no GitHub para copiar um repositório de outro usuário para sua conta. 35. Pull Request Solicitação para o autor do projeto aceitar suas alterações. 36. ssh-keygen -t ed25519 -C 'seu\_email' Gera uma chave SSH para autenticação com o GitHub. 37. ssh -T git@github.com Testa a conexão via SSH com o GitHub. 38. git@github.com:usuario/repositorio.git URL para clonar via SSH. 39. Integração Git no IntelliJ Permite usar Git visualmente: commit, push, pull, branch, tudo pelo IDE