

VS Code + Git + GitHub (do zero): guia rapido para subir seu primeiro repositorio

Passo a passo explicado - configuracao, autenticacao, primeiro commit e primeiro push

Visao geral

Este guia resume o que configuramos: instalar e ajustar Git, preparar VS Code, autenticar com GitHub usando o GitHub CLI (gh) e subir seu primeiro repositorio para o GitHub.

Ambiente assumido: Windows. Os comandos funcionam no PowerShell e no terminal do VS Code.

1. Conceitos essenciais (bem rapido)

- Git: sistema de controle de versao (historico de mudancas local).
- GitHub: plataforma na nuvem para hospedar repositorios Git e colaborar.
- Repositorio (repo): pasta versionada pelo Git.
- Commit: um 'snapshot' das mudancas (um ponto no historico).
- Remote (origin): endereco do repo no GitHub.
- Push: envia commits locais para o GitHub. Pull: traz mudancas do GitHub para sua maquina.

2. Criar/ajustar conta no GitHub

- 1 Crie uma conta no GitHub (se ja tiver, pule).
- 2 Em Settings -> Emails, confirme seu e-mail.
- 3 (Opcional, recomendado) Ative 2FA em Settings -> Password and authentication.

3. Instalar o Git e conferir

- 1 Instale o Git for Windows (site oficial).
- 2 Depois, abra o PowerShell e rode:

```
git --version
```

Se aparecer a versao do Git, esta OK.

4. Configurar identidade e branch padrao

Configure nome e email para que seus commits fiquem corretos no GitHub. Troque pelos seus dados:

```
git config --global user.name "Seu Nome"
git config --global user.email "seuemail@exemplo.com"
git config --global init.defaultBranch main
```

Verificar configuracoes:

```
git config --global --list
```

5. VS Code: instalacao e extensoes essenciais

- 1 Instale o VS Code (site oficial).
- 2 Instale extensoes recomendadas:
- 3 - GitLens (historico e blame bem melhores)
- 4 - Python (Microsoft) e Pylance
- 5 - Jupyter (se usar notebooks)
- 6 Confirme que o VS Code enxerga o Git: Ctrl+Shift+P -> 'Git: Show Git Output' (nao deve dar erro).

6. Autenticacao no GitHub (recomendado: GitHub CLI)

O jeito mais simples e profissional para iniciar e usar HTTPS com login no navegador e credenciais salvas.

- 1 Instale o GitHub CLI (gh).
- 2 No terminal, confira e faca login:

```
gh --version  
gh auth login
```

Respostas recomendadas no assistente:

- Where do you use GitHub? -> GitHub.com
- Preferred protocol for Git operations -> HTTPS
- Authenticate -> Login with a web browser (autorize no navegador)

Verificar status:

```
gh auth status
```

7. Criar seu primeiro projeto local (pasta + arquivos)

Crie uma pasta para seu repo e abra no VS Code:

```
cd C:\dev  
mkdir meu-primeiro-repo  
cd meu-primeiro-repo  
code .
```

Crie estes arquivos basicos:

- README.md (explica objetivo, como rodar e proximos passos)
- .gitignore (arquivos que NAO devem ir para o GitHub)
- (Python) requirements.txt ou pyproject.toml (dependencias)

8. O que e .gitignore e por que importa

O .gitignore informa ao Git o que NAO deve ser versionado. Ele evita: repo poluido, conflitos entre maquinas e principalmente vazamento de segredos (tokens/senhas).

Exemplo comum para Python/DS:

```
__pycache__/  
*.pyc  
.venv/  
.env  
.ipynb_checkpoints/  
.DS_Store
```

- __pycache__/ e *.pyc: arquivos gerados automaticamente pelo Python.
- .venv/: ambiente virtual (nao commitar; cada maquina cria o seu).
- .env: variaveis de ambiente (pode conter chaves da AWS, tokens, senhas).
- ipynb_checkpoints/: lixo tecnico do Jupyter.
- .DS_Store: arquivo do macOS que nao tem valor no repo.

Regra de ouro: se um arquivo pode ser gerado automaticamente, ele nao deve ir para o GitHub.

9. Inicializar Git, primeiro commit e primeiro push

Dentro da pasta do projeto, inicialize o Git e faca o primeiro commit:

```
git init  
git status  
git add .  
git commit -m "Initial commit"
```

Agora crie o repo no GitHub e suba tudo automaticamente (mais facil):

```
gh repo create meu-primeiro-repo --public --source=. --remote=origin --push
```

Alternativa manual (criando no site e conectando depois):

```
git remote add origin https://github.com/SEU_USUARIO/meu-primeiro-repo.git  
git push -u origin main
```

10. Fluxo do dia a dia (o ciclo)

- Ver o que mudou: git status
- Adicionar mudancas: git add . (ou selecione arquivos especificos)
- Criar commit: git commit -m "mensagem clara"
- Enviar para o GitHub: git push
- Trazer mudancas do GitHub: git pull

Dica de mensagem de commit: use verbo no presente e seja especifico (ex.: 'Adiciona README', 'Corrige caminho do dataset').

11. Proximo nivel: branches e Pull Request (padrao de time)

Em times, o comum é criar uma branch por tarefa, enviar e abrir Pull Request:

```
git checkout -b feature/minha-feature
# faça mudanças
git add .
git commit -m "Implementa X"
git push -u origin feature/minha-feature
```

Depois, no GitHub: abra um Pull Request e faça merge para main.

12. Checklist profissional (DS/ML)

- README com: objetivo, dados, como rodar, resultados e próximos passos.
- Não commitar: .venv/, .env, caches e checkpoints.
- Dependências sempre em requirements.txt/pyproject.toml.
- Commits pequenos e frequentes, com mensagens claras.
- Evite subir datasets grandes no GitHub; use links, amostras pequenas ou DVC/LFS quando fizer sentido.