MIRANDASTECH · MANUAL PROFISSIONAL

Git e GitHub para pesquisadores

Versionar o projeto de pesquisa do primeiro *commit* ao DOI: instalar o Git, registrar cada versão do texto, dos dados e dos scripts, publicar no GitHub com README, licença e `CITATION.cff`, e obter um DOI pelo Zenodo — **com ferramentas gratuitas, saídas reais de terminal e cada fato datado**.

SÉRIE	VERSÃO	FORMATO	LICENÇA
Manuais MirandasTech · pesquisa científica	1.0 · setembro de 2026	– slides	CC BY 4.0 — cite a fonte

COMO USAR CADA SLIDE É UM PASSO QUE VOCÊ EXECUTA NO TERMINAL OU NO NAVEGADOR

Para quem é — e o que você vai saber fazer ao final

Este manual é para **pós-graduandos e pesquisadores que nunca usaram Git** e têm um projeto com texto, dados e scripts que precisa de histórico, cópia de segurança e, no fim, um DOI. Não exige programação: todo comando está pronto para copiar, e a saída que aparece ao lado é a saída real de uma sessão de terminal feita em 09/09/2026. O manual não faz login no GitHub, no Zenodo, no OSF nem no Overleaf: as páginas públicas aparecem em capturas, e o que exige login é descrito pela documentação oficial, com os nomes exatos dos botões.

Ao final, você vai conseguir

- **Explicar** o que é um repositório, um commit, um branch e uma tag — e por que a evidência recomenda controle de versão na pesquisa (Ram, 2013; Perez-Riverol et al., 2016).
- **Instalar** o Git, configurar nome e e-mail, criar o repositório e fazer o primeiro commit (`init` , `status` , `add` , `commit` , `log`).
- **Decidir** o que entra e o que fica fora do Git: `.gitignore` , dados pessoais, arquivos licenciados, segredos e binários pesados.
- **Publicar** no GitHub: conta gratuita, repositório, `git push` , README, issues, pull requests e GitHub Actions.
- **Preparar** a citação: licença, `CITATION.cff` , o botão «Cite this repository», uma *release* e o DOI pelo Zenodo.
- **Rodar** três scripts: `checar_repo.py` (auditoria), `citacao_cff.py` (arquivo de citação) e `salvar_versao.sh` (commit + tag).

REGRA

Nomes de menu e de botão são os da documentação oficial do GitHub em português (docs.github.com/pt) e das páginas públicas observadas em 09/09/2026; quando a sua tela divergir, vale a tela. As saídas de terminal são reais (Git 2.48.1 nesta máquina), com o caminho trocado por `/home/usuario` e uma autora fictícia, «Aluna Exemplo».

ARMADILHA

Tratar o Git como uma pasta de backup. Tudo o que entra num commit fica no histórico — inclusive uma senha ou um CPF que você apagou no commit seguinte. Antes do primeiro `git add` , o slide «O que entra e o que fica fora» e o `checar_repo.py` (Parte 5) decidem o que pode ser versionado.

MANUAIS IRMÃOS

> Este manual cuida do *versionamento e da publicação*. Os passos vizinhos estão nos manuais de Metodologia (o projeto), Busca e PRISMA (o conteúdo), Zotero (as referências), LaTeX (o texto em `.tex`, que o Git versiona linha a linha), Uso de IA (declarar) e Plugin (o agente que audita o repositório). Hub: mirandastech.com.br/pesquisa.

INTRODUÇÃO O ROTEIRO DO MANUAL SEGUE A ORDEM DE UM PROJETO

Mapa: do primeiro commit ao DOI — e onde cada parte do manual entra

1

Instalar e configurar

Git 2.55.0 em git-scm.com/install; `user.name` e `user.email` uma vez por máquina

2

Repositório local

`git init -b main`, `git add`, `git commit`; o histórico fica na pasta oculta `.git/`

3

O que fica fora

`.gitignore`; dados pessoais, arquivos licenciados, segredos e binários pesados não entram

4

Branch, merge, tag

Uma linha paralela para a revisão do capítulo; merge de volta; uma tag por marco

5

Remoto no GitHub

Conta gratuita, repositório, `git remote add origin`, `git push -u origin main --follow-tags`

6

README, licença, citação

README.md, LICENSE (choosealicense.com), `CITATION.cff` → botão «Cite this repository»

7

Release → DOI

Repositório público com licença, ligado ao Zenodo: cada release ganha um DOI

PARTE	O QUE COBRE	PASSOS
1 O que é e por que usar	Git contra «versão_final_v3» · a evidência · Git × GitHub × GitLab · o que entra e o que fica fora · instalar e configurar	Passos 1 e 3
2 Git no dia a dia	init, status, add, commit · log e mensagens · .gitignore · branch, diff, merge, tag · conflitos · desfazer · onde aprofundar	Passos 2 a 4
3 GitHub passo a passo	Conta e planos · criar repositório · push · a página do repositório · Actions · issues e PRs · Desktop, VS Code, CLI, Codespaces · Pages · LFS · Overleaf e OSF	Passo 5
4 Publicar	README · licença · CITATION.cff · releases · DOI pelo Zenodo · GitLab como alternativa	Passos 6 e 7
5 Scripts e fechamento	checar_repo.py · citacao_cff.py · salvar_versao.sh · plugin · erros comuns · checklist · glossário · referências	Todos

COMO LER

Quem nunca abriu um terminal começa na Parte 1 e faz a Parte 2 com o terminal aberto ao lado. Quem já faz commits e quer o GitHub sem surpresas vai à Parte 3. Quem precisa publicar dados ou código com DOI para um artigo vai direto à Parte 4 — e roda o `checar_repo.py` da Parte 5 antes.

CONVENÇÕES

Nomes de menu e de botão entre «aspas angulares». Comandos em blocos com botão «copiar». Capturas de páginas em 1920 × 938, feitas em 09/09/2026 sem login. Figuras de terminal com «saída real»: sessão desta máquina, caminho trocado por `/home/usuario`, repositório fictício `minha-pesquisa`. Nunca há senha, token ou chave neste manual nem nos scripts.

PARTE 1 O QUE É E POR QUE USAR

Git contra «versão_final_v3_revisada.docx»; o que a evidência diz; Git, GitHub e GitLab; o que entra e o que fica fora; instalar e configurar

01

Antes do primeiro comando: o que o Git guarda e o que ele não deve guardar, por que os guias de computação científica o recomendam, a diferença entre a ferramenta e os serviços que a hospedam — e a instalação em cinco minutos.

Git contra «versão_final_v3_revisada.docx»: um histórico completo, com autor, data e motivo de cada mudança

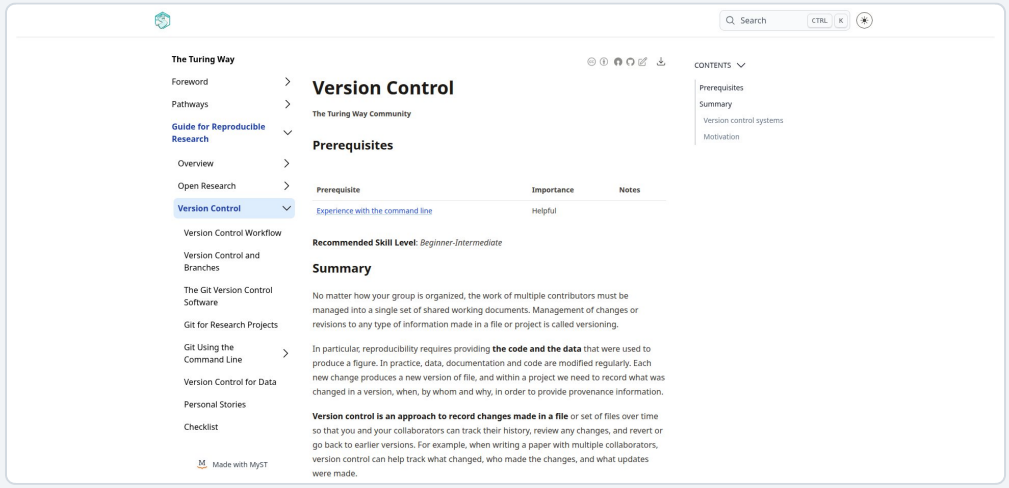

FIGURA 1 O capítulo «Version Control» do The Turing Way (captura de 09/09/2026): o controle de versão é exigido para seguir a proveniência do que se produz; o Git não serve para versionar binários; branches permitem desenvolvimento não linear; plataformas citadas: GitHub, GitLab e Codeberg.

Git é um sistema de controle de versão distribuído, livre e de código aberto (GPLv2), segundo git-scm.com/about — o site cita uma pesquisa de 2022 em que «96% dos desenvolvedores profissionais usam Git», e o histórico do kernel Linux (1,4 milhão de commits) ocupa 5,5 GB. Para a pesquisa, o que importa é outra coisa: cada commit guarda quem mudou, quando, o quê e por quê.

PASTA COM CÓPIAS

- `texto_v1.docx`, `texto_v2.docx`, `texto_final.docx`, `texto_final_REVISADO.docx`…
- Ninguém sabe o que mudou entre duas cópias, nem quem mudou
- Dois autores editando: uma cópia sobrescreve a outra
- O backup é a última cópia que alguém lembrou de fazer

REPOSITÓRIO GIT

- Um único `main.tex`; o histórico fica em `.git/`
- `git log` lista cada versão com autor, data e mensagem; `git diff` mostra linha a linha o que mudou
- Cada autor trabalha na sua cópia; o Git combina as mudanças e aponta os conflitos
- `git push` leva o histórico inteiro para o

	GitHub, o GitLab ou o servidor da instituição

REGRA

O Git versiona *texto*: `.tex`, `.md`, `.csv`, `.py`, `.R`, `.bib`. Um `.docx` ou um PDF entra, mas o Git só sabe dizer que «mudou», não o quê — o *Turing Way* é direto: não é ferramenta para versionar binários. Quem escreve em LaTeX (manual irmão) ganha o *diff* por linha de graça.

O que a evidência e os guias dizem: reprodutibilidade, transparência e «práticas boas o bastante»

FONTE	O QUE DIZ	O QUE MUDA PARA VOCÊ
Ram (2013), Source Code for Biology and Medicine	O Git pode facilitar maior reprodutibilidade e mais transparência na ciência	O histórico do projeto é parte do método, não um detalhe técnico
Perez-Riverol et al. (2016), PLOS Comput. Biol.	«Ten Simple Rules» para tirar proveito do Git e do GitHub	Regras práticas: repositório por projeto, commits pequenos, issues, releases com DOI
Blischak, Davenport e Wilson (2016), PLOS Comput. Biol.	Introdução rápida ao controle de versão com Git e GitHub para cientistas	O tutorial que a Parte 2 segue no espírito: poucos comandos, usados sempre
Wilson et al. (2017), PLOS Comput. Biol.	«Good enough practices»: manuscrito em texto simples e sob controle de versão	Não precisa dominar o Git inteiro; o básico já é «bom o bastante»

O QUE O GIT RESOLVE

- «Qual versão do script gerou a Tabela 3?» — a tag da submissão responde
- «O que mudou desde a qualificação?» — `git diff` entre duas tags
- Orientador e estudante editando o mesmo texto, com histórico de quem fez o quê
- Revisor pedindo o código: um link para o repositório, com DOI

O QUE O GIT NÃO RESOLVE

- Backup dos dados brutos grandes — esses vão para um repositório de dados (OSF, Zenodo, o da instituição)
- Anonimização: um repositório privado não torna um dado pessoal público em «anônimo»
- Organização do projeto: a estrutura de pastas vem do plugin e do manual de Metodologia

REGRA

Comece pequeno, como Wilson et al. (2017) recomendam: um repositório por projeto, `git add` e `git commit` ao fim de cada sessão de trabalho, uma tag a cada marco. Branches, pull requests e

FONTE	O QUE DIZ	O QUE MUDA PARA VOCÊ
The Turing Way (2024), Zenodo	Controle de versão é exigido para seguir a proveniência; nada de binários; branches; GitHub, GitLab, Codeberg	O manual de referência para pesquis reprodutível, ética e colaborativa (CC BY 4.0)
Software Carpentry, Version Control with Git	14 episódios, de «Automated Version Control» a «Citation» e «Hosting» (CC BY 4.0)	Onde praticar depois deste manual (slide «Para aprofundar»)

> CI vêm depois, quando houver mais de uma pessoa ou mais de um caminho a explorar.

Git × GitHub × GitLab: a ferramenta no seu computador e os serviços que hospedam o repositório

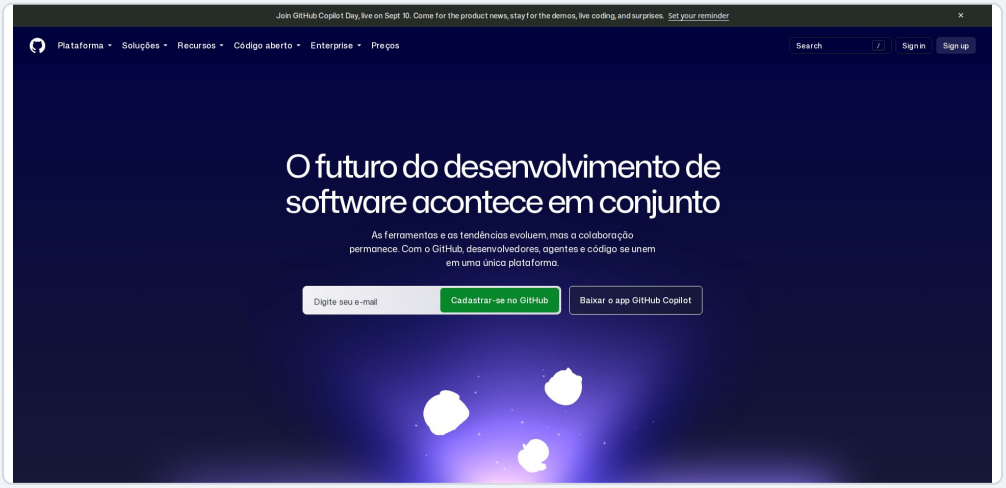

FIGURA 2 A home do GitHub em português (captura de 09/09/2026): «O futuro do desenvolvimento de software acontece em conjunto». O GitHub é um serviço que hospeda repositórios Git e acrescenta a interface web, issues, pull requests, Actions, Pages e Codespaces; a conta pessoal é gratuita (slide «Conta e planos»).

NOME	O QUE É	ONDE RODA · CUSTO
Git	O programa de controle de versão: cria o repositório, registra commits, compara versões, combina branches. Funciona sem internet e sem conta	No seu computador · gratuito (GPLv2); versão 2.5
GitHub	Serviço que hospeda repositórios Git e soma interface web, issues, pull requests, Actions, Pages, Codespaces, «Cite this repository» e a	Na nuvem · Free (US$ 0), Team (US$ 4/usuário/m Enterprise (US$ 21)

NOME	O QUE É	ONDE RODA · CUSTO
	integração com o Zenodo	
GitLab	Alternativa com o mesmo Git por baixo: gerenciamento de código e CI/CD; muitas universidades hospedam uma instância própria	Na nuvem ou na instituição · Gratuito (US$ 0), Premium (US$ 29/usuário/mês, anual), Ultimate consulta)
Codeberg	Plataforma comunitária citada pelo Turing Way como alternativa	Na nuvem

REGRA

Aprenda o Git, não o GitHub: tudo o que este manual faz no terminal funciona igual com GitLab, Codeberg ou um servidor da instituição. O GitHub é a escolha deste manual porque é o mais usado, tem documentação em português e liga-se ao Zenodo para o DOI (Parte 4) — mas o repositório é seu e pode ser levado para outro serviço com um `git push`.

VOCABULÁRIO

> *Local* é o repositório na sua máquina; *remoto* é a cópia no serviço (por convenção chamada `origin`). `git push` envia commits do local para o remoto; `git pull` traz os do remoto; `git clone` cria um local a partir de um remoto.

O que entra e o que fica fora do Git: texto e scripts entram; dados pessoais, arquivos licenciados, segredos e binários pesados ficam fora

ENTRA NO GIT

- Texto: `.tex`, `.md`, `.bib`, `.csv` pequenos, o diário de busca
- Código: `.py`, `.R`, `.sh`, notebooks, `requirements.txt`
- Metadados: README, LICENSE, `CITATION.cff`, `.gitignore`
- Figuras finais leves (PDF, PNG de poucos MB) e tabelas de resultados
- Dados derivados, agregados e anonimizados, quando a licença permite

FICA FORA — SEMPRE

- **Dados pessoais** de participantes (LGPD): nome, CPF, e-mail, gravações, prontuários — mesmo em repositório privado
- **Arquivos licenciados**: PDFs de artigos de bases pagas, exportações completas de bases (Web of Science, Scopus), planilhas de terceiros
- **Segredos**: `.env`, tokens, chaves de API, senhas, chaves privadas (`.pem`, `id_rsa`)
- **Binários pesados**: vídeos, áudio, imagens brutas, modelos treinados, bancos de dados
- Arquivos gerados: `.aux`, `.log`, `__pycache__/`, `.Rhistory`

LIMITE DO GITHUB (DOCS, 09/09/2026)	VALOR
Aviso por arquivo	acima de **50 MiB**
Bloqueio por arquivo	acima de **100 MiB** (o push é recusado)
Upload pelo navegador	até **25 MiB** por arquivo
Tamanho do repositório	idealmente **< 1 GB**; «fortemente recomendado» **< 5 GB**
Acima disso	Git LFS (slide «Arquivos grandes e Git L

REGRA

Uma pasta `dados_FORA_DO_GIT/` (o plugin já a cria e a deixa fora do versionamento) recebe tudo o que é bruto, pessoal ou licenciado; no repositório entram o script que processa e o resultado agregado. Segredos ficam num `.env` listado no `.gitignore` desde o primeiro commit.

ARMADILHA

«**Apaguei no commit seguinte**». Apagar um arquivo não o tira do histórico: quem clonar o repositório ainda encontra o segredo ou o

CPF nos commits anteriores. Um segredo que entrou no Git está vazado — troque a credencial; o `checar_repo.py` (Parte 5) avisa com «BLOQUEIO» antes do push.

Instalar e configurar: Git 2.55.0 para Windows, macOS e Linux; `user.name` e `user.email` uma vez por máquina

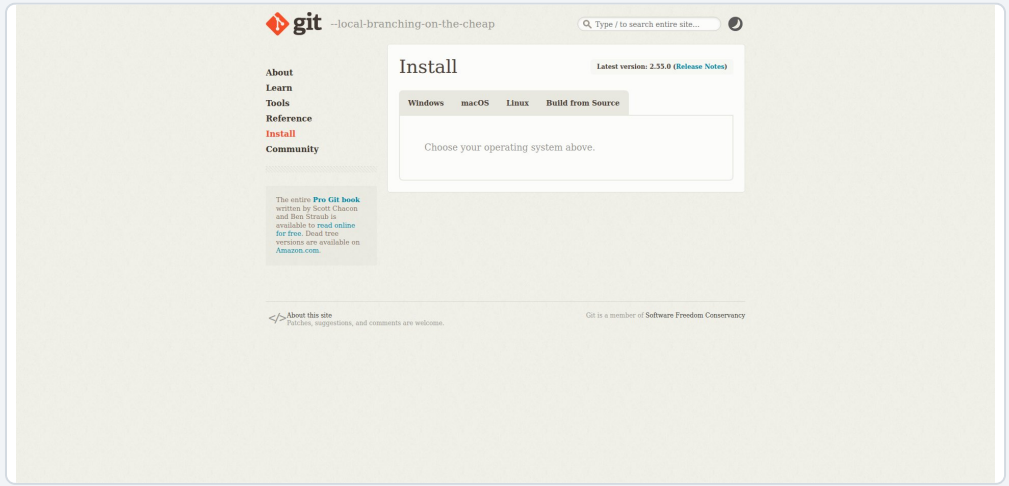

FIGURA 3 git-scm.com/install (captura de 09/09/2026): Windows, macOS e Linux; versão 2.55.0. Windows: instalador «Git for Windows»; macOS: Homebrew (`brew install git`) ou Xcode Command Line Tools; Linux: `apt install git` , `dnf install git` .

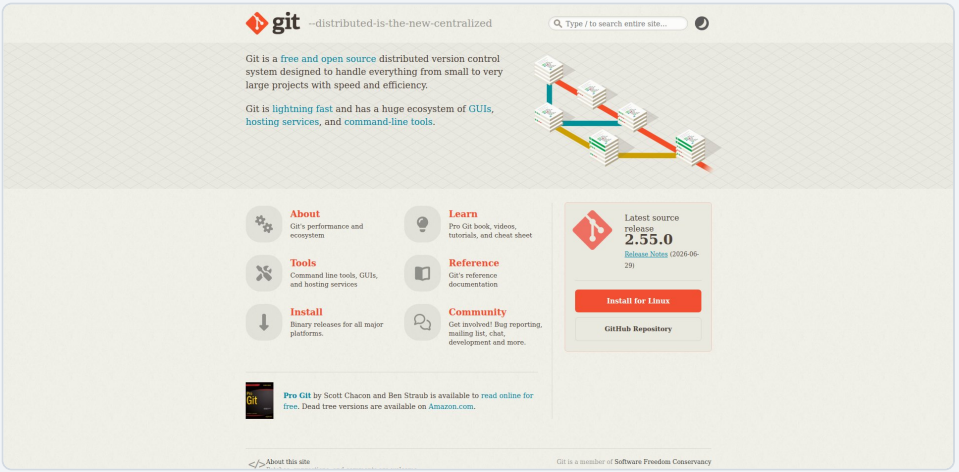

FIGURA 4 A home de git-scm.com (captura de 09/09/2026): «Latest source release 2.55.0»; daqui saem a documentação de referência e o livro Pro Git, traduzido para o português (slide «Para aprofundar»).

TERMINAL — DEPOIS DE INSTALAR (O MESMO EM WINDOWS, MACOS E LINUX)

```
git --version                                    # confere a instalação (nesta máquina: 2.48.1)
git config --global user.name "Aluna Exemplo"   # o nome que vai em cada commit
git config --global user.email "aluna@universidade.br"
```

REGRA

Nome e e-mail vão para o histórico de todos os commits e aparecem no GitHub: use o nome com que você assina os artigos e um e-mail que continue seu depois do curso. O VS Code exige essas duas configurações e Git ≥ 2.0.0 para funcionar (slide «GitHub Desktop e VS Code»).

```
git config --global --list                    # confere o que
ficou gravado
```

> **VERSÃO**
>
> A demonstração deste manual rodou com Git 2.48.1; nenhum comando aqui exige a versão mais nova (2.55.0). O `git init -b main` (Parte 2) cria o repositório já com o branch `main`, sem depender de configuração.

PARTE 2 GIT NO DIA A DIA

init, status, add e commit; log e boas mensagens; .gitignore; branch, diff, merge e tag; conflitos; desfazer com segurança; onde aprofundar

Sete slides com o terminal aberto: cada comando aparece com a saída real da sessão de 09/09/2026 (Git 2.48.1, em português, repositório fictício `minha-pesquisa`). Os conceitos que não foram executados na demonstração — conflitos e desfazer — são explicados sem inventar saída.

02

Começar um repositório: `git init`, `git status`, `git add` e o primeiro `git commit`

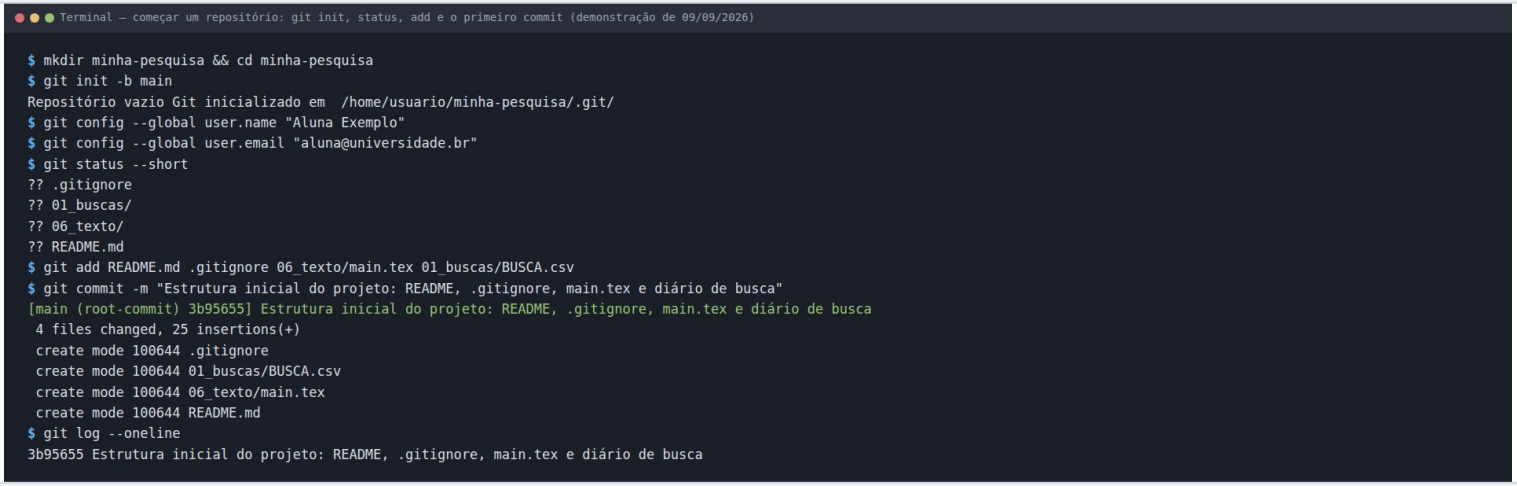

FIGURA 5 A sessão de 09/09/2026 (saída real, caminho trocado por `/home/usuario`): `git init -b main` responde «Repositório vazio Git inicializado em /home/usuario/minha-pesquisa/.git/»; `git status --short` lista quatro itens com `??` (não rastreados); `git add` e `git commit -m` produzem `[main (root-commit) 3b95655] … 4 files changed, 25 insertions(+)`; `git log --oneline` mostra o commit.

1

`git init -b main`

Na pasta do projeto: cria a pasta oculta `.git/` e o branch `main`. Uma vez por projeto

2

`git status --short`

`??` = arquivo que o Git ainda não rastreia; `M` = modificado; `A` = adicionado à área de preparação

3

REGRA

Nomeie os arquivos no `git add` (como na figura: `README.md .gitignore 06_texto/main.tex 01_buscas/BUSCA.csv`) até o `.gitignore` estar pronto (slide seguinte); só então `git add -A` é seguro. Sempre `git status` antes de `git commit`: é a lista do que vai entrar.

O QUE A SAÍDA DIZ

`git add …`

Coloca os arquivos escolhidos na área de preparação (staging): o que o próximo commit vai incluir

4

`git commit -m "…"`

Registra a versão com a mensagem; o código `3b95655` é o identificador do commit

> `root-commit` é o primeiro commit do repositório; `create mode 100644` é o modo de arquivo normal (sem permissão de execução). Os identificadores (`3b95655`) são únicos por repositório: os seus serão diferentes.

`git log` e o que é um bom commit: pequeno, coeso e com uma mensagem que diga o que mudou e por quê

TERMINAL — LER O HISTÓRICO E AS DIFERENÇAS

```
git log --oneline                    # uma linha por commit:
identificador e mensagem
git log --oneline --graph --decorate   # com o desenho dos
branches e as tags (Figura 7)
git diff                             # o que mudou nos
arquivos e ainda não foi adicionado
git diff --staged                    # o que já está na área
de preparação
git show 3b95655                     # tudo o que um commit
mudou
```

MENSAGENS DA DEMONSTRAÇÃO — REAIS, NA FIGURA 5 E NA FIGURA 7

```
Estrutura inicial do projeto: README, .gitignore, main.tex e
diário de busca
Capítulo 2: primeiro parágrafo da revisão
Tira o .env do Git (a chave foi trocada)
Licença e arquivo de citação
```

UM BOM COMMIT	POR QUÊ
Faz uma coisa	«Capítulo 2: primeiro parágrafo da revi — dá para entender, comparar e desfaz sozinho
Mensagem no imperativo ou no presente	«Tira o .env do Git», «Acrescenta a Tab 3»: a primeira linha é o que aparece em `log --oneline` e no GitHub
Diz o porquê quando não é óbvio	«(a chave foi trocada)» — quem ler o histórico daqui a um ano agradece
Frequente	Ao fim de cada sessão de trabalho, ou cada resultado: Wilson et al. (2017) ch isso de prática «boa o bastante»
Não é «tudo»	Um commit gigante «atualizações» esc o que aconteceu; é o erro mais comum iniciante (slide «Erros comuns»)

REGRA

Mensagens em português, como na demonstração; a primeira linha com até uns 70 caracteres. Se precisar explicar mais, `git commit` sem `-m` abre o editor e a partir da segunda linha vai o detalhe. O

> `salvar_versao.sh` (Parte 5) faz o `add -A`, mostra o que entra e faz o commit num comando só.

`.gitignore`: o que o Git deve fingir que não existe — com os modelos oficiais para TeX, Python e R

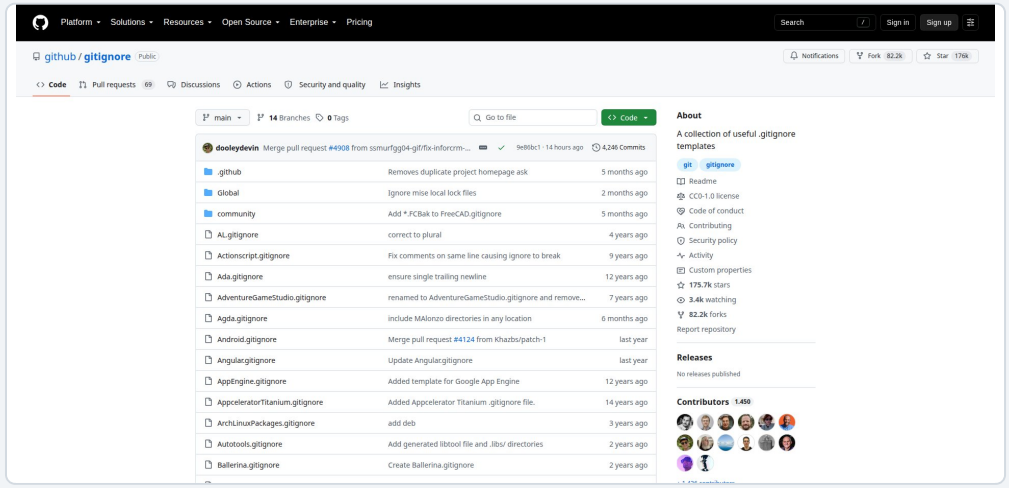

FIGURA 6 github.com/github/gitignore (captura de 09/09/2026): a coleção oficial de modelos, um arquivo por linguagem ou ferramenta — entre eles `TeX.gitignore`, `Python.gitignore` e `R.gitignore`. Copie o conteúdo do que serve para o seu projeto para o `.gitignore` na raiz.

.GITIGNORE — EXEMPLO PARA UM PROJETO COM LATEX, PYTHON, R E DADOS FORA DO GIT (MONTE O SEU A PARTIR DOS MODELOS OFICIAIS)

```
# dados brutos, pessoais ou licenciados: nunca
dados_FORA_DO_GIT/
# segredos
.env
*.pem
# gerados pelo LaTeX
*.aux
*.log
*.bbl
*.blg
*.out
*.toc
*.synctex.gz
# gerados pelo Python e pelo R
__pycache__/
.Rhistory
.ipynb_checkpoints/
```

TERMINAL — ARQUIVO QUE JÁ ENTROU NO GIT SÓ SAI COM --CACHED (NÃO APAGA DO DISCO)

```
git rm --cached .env          # tira do Git; o arquivo
continua na pasta
```

```
echo ".env" >> .gitignore     # e passa a ser ignorado
git commit -m "Tira o .env do Git (a chave foi trocada)"
```

REGRA

O `.gitignore` fica na raiz do repositório e entra no primeiro commit (como na Figura 5). Um padrão vale para toda a árvore; `pasta/` ignora a pasta inteira. Para ignorar algo em todos os repositórios da máquina (arquivos do editor, do sistema), o arquivo é `~/.config/git/ignore`. O gerador gitignore.io monta um arquivo a partir de várias ferramentas.

Branch, diff, merge e tag: uma linha paralela para a revisão do capítulo, de volta ao `main`, e uma tag por marco

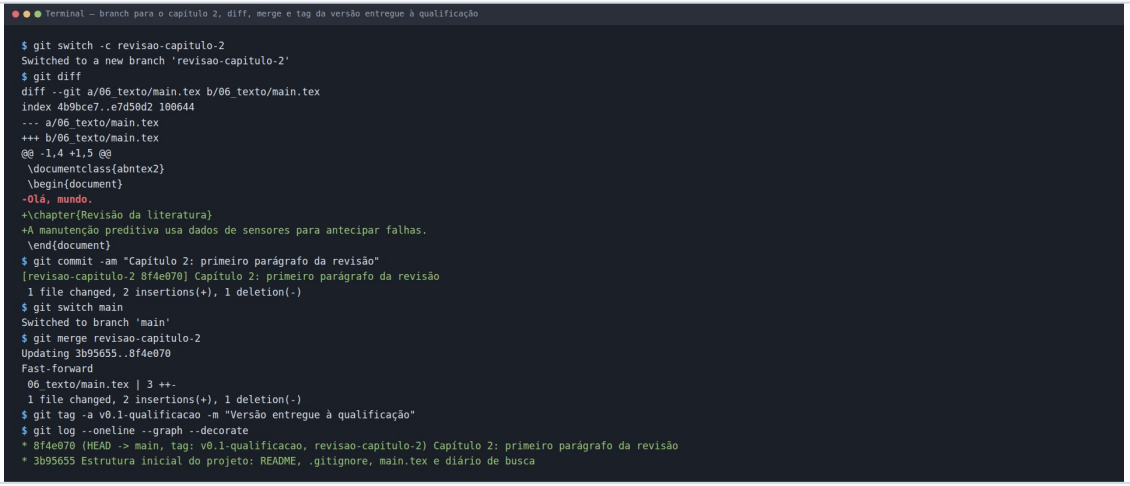

FIGURA 7 A sessão de 09/09/2026 (saída real, caminho trocado por `/home/usuario`): `git switch -c revisao-capitulo-2` cria e entra no branch; `git diff` mostra a linha removida (`-`, em vermelho) e as acrescentadas (`+`, em verde); `git commit -am`; `git switch main`; `git merge revisao-capitulo-2` responde «Fast-forward»; `git tag -a v0.1-qualificacao -m "…"`; e `git log --oneline --graph --decorate` mostra `HEAD -> main, tag: v0.1-qualificacao`.

COMANDO	O QUE FAZ
`git switch -c nome`	Cria um branch e muda para ele; os commits seguintes ficam nessa linha, sem tocar o `main`
`git diff`	Compara o arquivo no disco com o último commit: `-` saiu, `+` entrou
`git commit -am "…"`	

REGRA

Uma tag por marco que alguém pode pedir de volta: qualificação, submissão do artigo, versão que gerou os resultados, defesa. Use sempre `-a` e `-m`: é a tag anotada que carrega a mensagem e que o `git push --follow-tags` envia (Parte 3). Branch para trabalhar sem medo — um capítulo em revisão, uma análise alternativa — e *merge* quando ficar bom.

COMANDO	O QUE FAZ
	Adiciona todos os arquivos já rastreado que mudaram e faz o commit (arquivos novos ainda precisam de `git add`)
`git switch main` → `git merge nome`	Volta ao `main` e traz os commits do branch. «Fast-forward»: o `main` só avançou, porque ninguém mexeu nele enquanto isso
`git tag -a v0.1-qualificacao -m "…"`	Tag anotada (com autor, data e mensagem) no commit atual: o marco «versão entregue à qualificação»

SEM BRANCH TAMBÉM FUNCIONA

Quem trabalha sozinho pode viver no `main` com commits e tags. O branch começa a valer quando há um orientador ou colega editando ao mesmo tempo, ou quando você quer testar algo que talvez não fique.

Conflitos: quando duas pessoas mudam a mesma linha, o Git para e pede que alguém decida

Na demonstração o *merge* foi «Fast-forward» porque só um lado mudou. Um conflito acontece quando os dois lados (dois branches, ou o seu local e o remoto) mudaram *a mesma linha* do mesmo arquivo. O Git não escolhe: ele grava as duas versões no arquivo, entre marcadores, e espera. O exemplo ao lado é **ilustrativo** — não foi executado na sessão deste manual.

06_TEXTO/MAIN.TEX — COMO UM CONFLITO APARECE DENTRO DO ARQUIVO (EXEMPLO ILUSTRATIVO)

```
\chapter{Revisão da literatura}
<<<<<<< HEAD
A manutenção preditiva usa dados de sensores para antecipar
falhas.
=======
A manutenção preditiva usa séries temporais de sensores para
prever falhas.
>>>>>>> revisao-orientador
\end{document}
```

MARCADOR	O QUE É
`<<<<<<< HEAD`	Começa a versão do branch em que você está
`=======`	Separa as duas versões
`>>>>>>> nome`	Termina a versão do outro branch (ou do remoto)

1

`git status`

Lista os arquivos em conflito («both modified»)

2

Abra o arquivo

Escolha uma versão, ou escreva a combinação das duas; apague as três linhas de marcadores

3

`git add arquivo`

Diz ao Git que o conflito desse arquivo foi resolvido

4

`git commit`

Fecha o merge; sem `-m`, o Git propõe a mensagem

REGRA

Conflito não é erro: é o Git protegendo o trabalho dos dois lados. Resolva no editor (o VS Code mostra botões para aceitar cada lado), compile o `.tex` ou rode o script antes do commit final, e nunca deixe um marcador dentro do arquivo. Quanto menores e mais frequentes os commits e os *merges*, menos conflitos.

PARA DESISTIR

`git merge --abort` devolve o repositório ao estado anterior ao *merge*, sem perder nada.

Desfazer com segurança: `git restore` para o arquivo, `git revert` para o commit — e por que `reset --hard` fica para o fim

QUERO…	COMANDO	O QUE ACONTECE	RISCO
Voltar um arquivo ao último commit	`git restore arquivo`	Descarta as mudanças não commitadas desse arquivo	PERDE O QUE NÃO FOI
Tirar da área de preparação	`git restore --staged arquivo`	Desfaz o `git add`; o arquivo continua modificado no disco	SEGURO
Desfazer um commit já feito	`git revert 8f4e070`	Cria um novo commit que anula o anterior; o histórico continua íntegro	SEGURO; PODE DAR PUS
	`git commit --amend`		NÃO USE DEPOIS DO PU

REGRA

Três verbos resolvem quase tudo: `restore` (arquivo), `revert` (commit) e `switch` (branch). Prefira sempre o que *acrescenta* um commit ao que *apaga* — o `revert` deixa registrado que a Tabela 3 foi refeita e por quê. O `reset --hard` e o *force push* reescrevem história: em repositório compartilhado, apagam o trabalho de outra pessoa.

ARMADILHA

«Desfazer» um segredo com `revert`. O `revert` tira o arquivo da versão atual, mas o commit original continua no histórico — e no GitHub. Segredo que subiu está vazado: troque a credencial. O `git rm --cached` resolve para o que ainda não foi enviado (Figura 8).

ANTES DE QUALQUER COMANDO DESTRUTIVO

`git status` e `git log --oneline`. Se estiver em dúvida, faça uma cópia da pasta inteira — o Git não impede, e o *Pro Git* dedica capítulos («Ferramentas do Git», «Git Internals») ao que pode ser recuperado.

QUERO…	COMANDO	O QUE ACONTECE	RISCO
Corrigir a mensagem do último commit		Reescreve o último commit (só se ainda não foi enviado ao remoto)	
Ver um arquivo como estava numa tag	`git show v0.1-qualificacao:06_texto/main.tex`	Imprime a versão antiga sem mudar nada	SEGURO
Jogar fora tudo desde um commit	`git reset --hard 3b95655`	Apaga commits posteriores e as mudanças no disco	DESTRUTIVO

Para aprofundar: o Pro Git em português (gratuito) e a oficina Version Control with Git do Software Carpentry

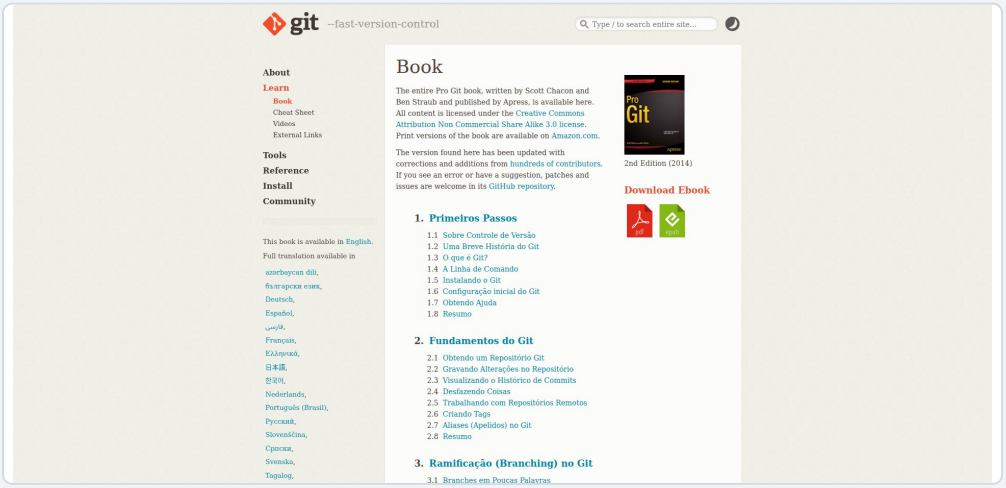

FIGURA 8 Pro Git, 2ª edição (Chacon e Straub, Apress, 2014), em git-scm.com/book/pt-br/v2 (captura de 09/09/2026): tradução completa para o português, licença CC BY-NC-SA 3.0, PDF e EPUB gratuitos. Capítulos: Primeiros Passos, Fundamentos do Git, Ramificação, Git no Servidor, Git Distribuído, GitHub, Ferramentas do Git, Customizando o Git, Git e Outros Sistemas, Git Internals e apêndices.

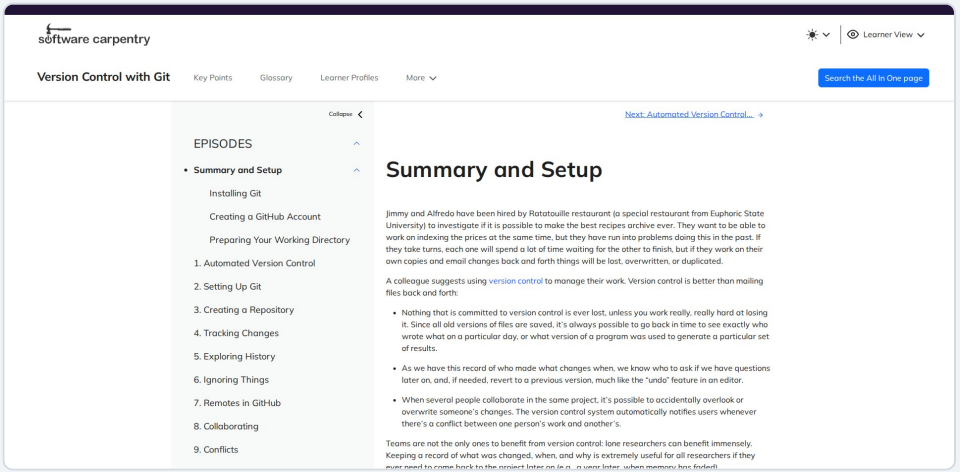

FIGURA 9 Version Control with Git, do Software Carpentry (captura de 09/09/2026; CC BY 4.0): 14 episódios — Automated Version Control, Setting Up Git, Creating a Repository, Tracking Changes, Exploring History, Ignoring Things, Remotes in GitHub, Collaborating, Conflicts, Open Science, Licensing, Citation, Hosting e Using Git from RStudio.

DEPOIS DESTE MANUAL, LEIA	PARA
Pro Git, «Fundamentos do Git»	Tudo sobre `status` `add`, `commit`, `log`, `diff`, desfazer e tags
Pro Git, «Ramificação»	

REGRA

Quando um comando deste manual não bastar, a resposta está no *Pro Git* — em português, gratuito e mantido pelo próprio projeto Git. Prefira-o a respostas soltas em fóruns: elas costumam trazer o `reset --hard` e o *force push* como se fossem inofensivos.

DEPOIS DESTE MANUAL, LEIA	PARA
	Branches e merges com desenhos; conflitos com calma
Pro Git, «GitHub»	Conta, fork, pull request, organização
Software Carpentry, episódios 1 a 9	A mesma sequência da Parte 2, com exercícios; «Remotes in GitHub» e «Collaborating» cobrem a Parte 3
Software Carpentry, «Open Science», «Licensing», «Citation»	O que a Parte 4 deste manual faz com licença, `CITATION.cff` e DOI

> RSTUDIO
>
> O episódio suplementar «Using Git from RStudio» mostra a aba Git do RStudio para quem trabalha em R; os comandos são os mesmos.

PARTE 3 GITHUB PASSO A PASSO

Conta e planos; criar repositório e «Olá, Mundo»; remoto e push; a página de um repositório real; commits, Actions e CI; issues e pull requests; Desktop, VS Code, CLI e Codespaces; Pages; arquivos grandes e LFS; Overleaf e OSF

As páginas públicas (home, preços, documentação em português, um repositório público desta série) aparecem em capturas de 09/09/2026. O que exige login — criar o repositório, o painel de configurações — é descrito pela documentação oficial em português, com os nomes exatos dos botões. O manual não faz login e não pede senha.

03

Conta e planos: o GitHub Free basta para a pesquisa — e o Student Developer Pack dá o Pro a estudantes verificados

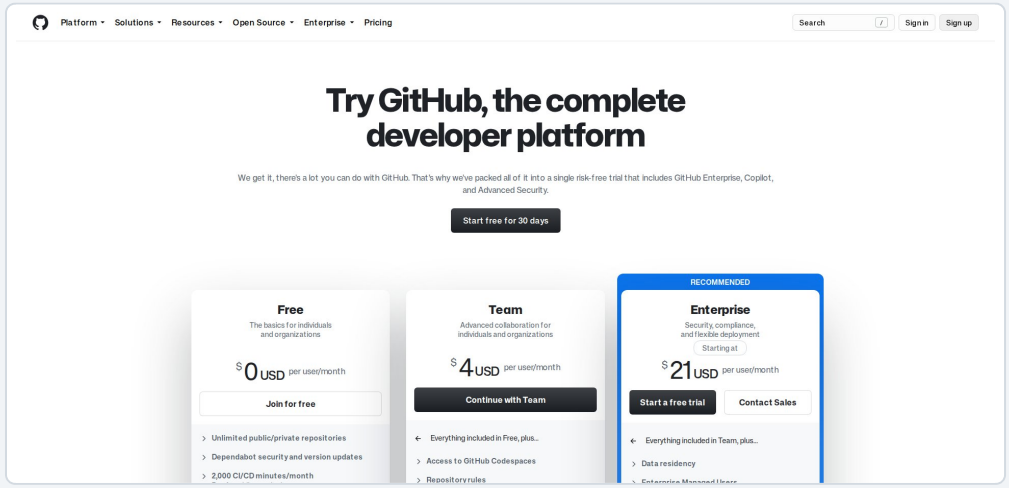

FIGURA 10 github.com/pricing (captura de 09/09/2026): «Free» US$ 0; «Team» US$ 4 por usuário/mês («first 12 months»); «Enterprise» US$ 21 por usuário/mês. O plano gratuito já inclui repositórios públicos e privados ilimitados.

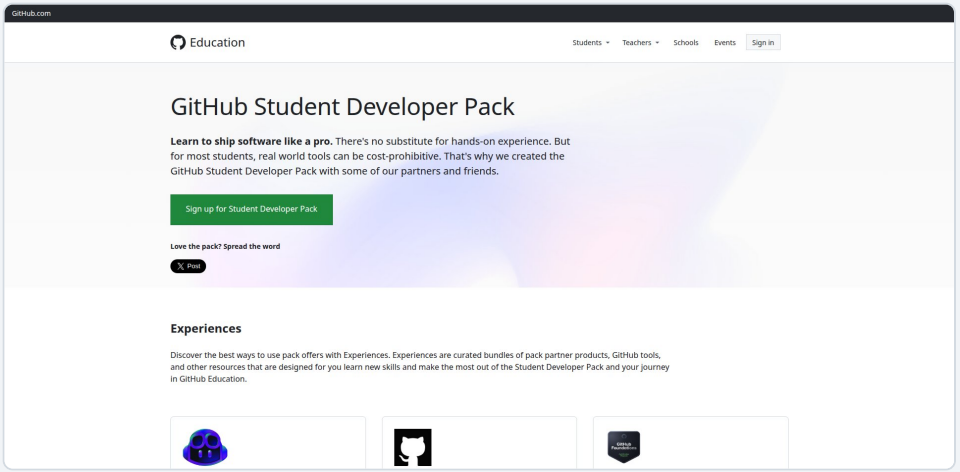

FIGURA 11 education.github.com/pack (captura de 09/09/2026): «GitHub Pro gratuito enquanto você for estudante», Copilot, «acesso Pro ao Codespaces» e mais de 50 ferramentas de parceiros; o pedido é feito em github.com/settings/education/benefits, para estudantes verificados pelo GitHub.

PLANO	PREÇO	O QUE INCLUI (GITHUB.COM/PRICING, 09/09/2026)
Free	US$ 0	Repositórios públicos e privados ilimitados; colaboradores ilimitados; 2.000 minutos de GitHu Actions por mês; 500 MB no Packages; 120 horas núcleo e 15 GB de Codespaces por mês; GitHub Pages em repositórios públicos; alertas do Dependabot; suporte da comunidade

REGRA

Crie a conta gratuita em github.com com o e-mail que você configurou no Git (slide «Instalar e configurar») e um nome de usuário que continue fazendo sentido depois do curso: ele entra na URL do repositório, no `CITATION.cff` e no DOI. Este manual não cria conta nem faz login: a senha é digitada só na página do GitHub.

PLANO	PREÇO	O QUE INCLUI (GITHUB.COM/PRICING, 09/09/2026)
Team	US$ 4 por usuário/mês	3.000 minutos de Actions; 2 GB de Packages; revisores obrigatórios, branches protegidas, code owners; suporte por e-mail/web
Enterprise	US$ 21 por usuário/mês	50.000 minutos; 50 GB; SAML SSO; SLA de 99,9%
Pro (individual)	gratuito no Student Developer Pack	Não aparece em destaque na página de preços; vem com o pacote de estudante

> **SEM PREÇO EM REAIS**
>
> A página de preços mostra dólares; o manual não converte nem afirma promoções. O Student Developer Pack exige verificação de estudante feita pelo GitHub — a página não detalha os documentos.

Criar o repositório e o «Olá, Mundo»: «+» → «Novo repositório» → nome → «Criar repositório»; depois branch, edição, pull request e merge

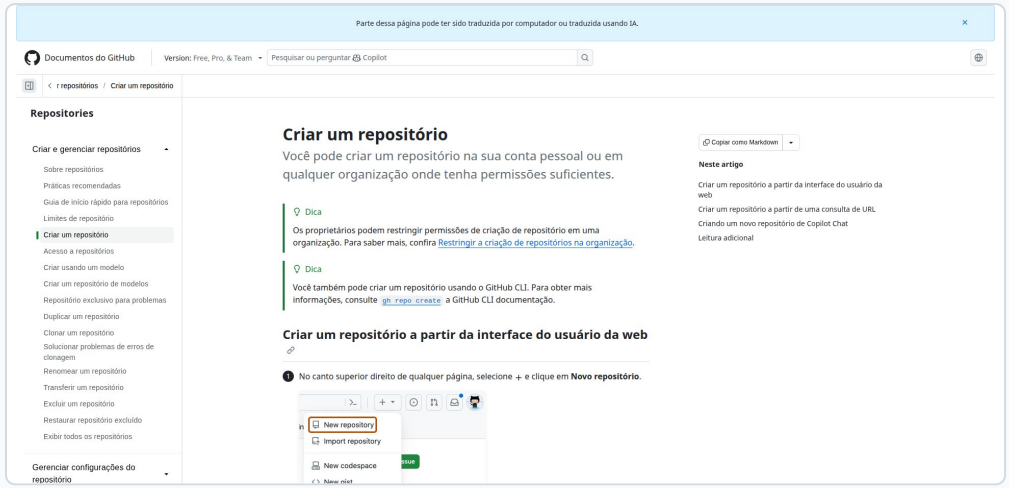

FIGURA 12 «Criar um repositório», em docs.github.com/pt (captura de 09/09/2026): «+» no canto superior → «Novo repositório» → nome, descrição, Público ou Privado, «Adicionar um arquivo LEIAME» → «Criar repositório». A página github.com/new exige login e não foi capturada.

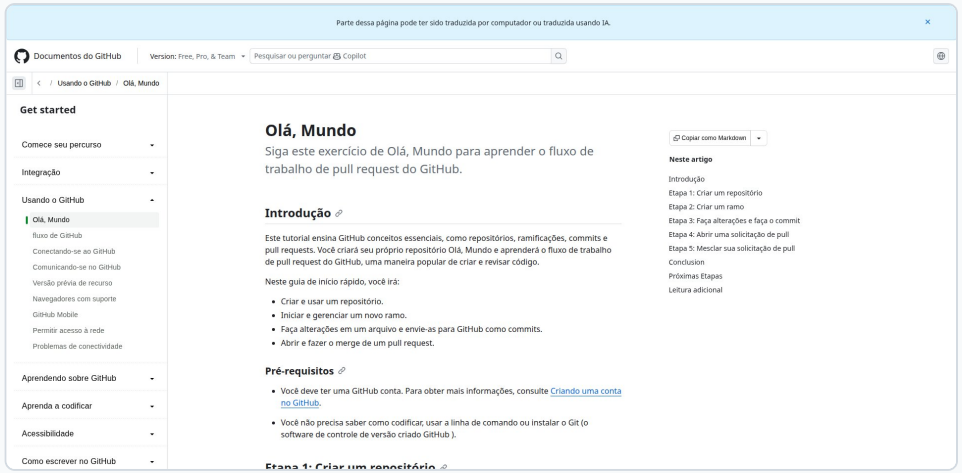

FIGURA 13 «Olá, Mundo», em docs.github.com/pt (captura de 09/09/2026): o tutorial oficial que cria o repositório `hello-world`, o branch `readme-edits`, edita o README, abre e mescla uma solicitação de pull — tudo pelo navegador.

1

«+» → «Novo repositório»

Nome (`hello-world` no tutorial; `minha-pesquisa` aqui), descrição, Público ou Privado, «Adicionar um arquivo LEIAME» → «Criar repositório»

2

Branch

Menu «main» → digitar `readme-edits` → «Criar branch: readme-edits a partir do main»

REGRA

Para o projeto de pesquisa que já existe no seu computador (Parte 2), crie o repositório no GitHub *sem* «Adicionar um arquivo LEIAME» e sem `.gitignore`: o histórico vem do `git push` (slide seguinte). O «Olá, Mundo» é para entender a interface — faça-o uma vez, num repositório de teste.

3

Editar e confirmar

Editar o README → «Confirmar alterações» (mensagem de commit) → «Confirmar alterações»

4

Pull request

«Solicitações de pull» → «New pull request» → «Criar solicitação de pull»

5

Mesclar

«Mesclar solicitação de pull» → «Confirmar mesclagem» → «Excluir ramo»

PÚBLICO OU PRIVADO

Privado enquanto o texto está em construção, se preferir; público quando for citar ou pedir o DOI — o Zenodo só arquiva repositórios públicos (Parte 4). Dado pessoal não entra em nenhum dos dois.

Remoto e push: `git remote add origin` e `git push -u origin main --follow-tags` levam o histórico e as tags para o GitHub

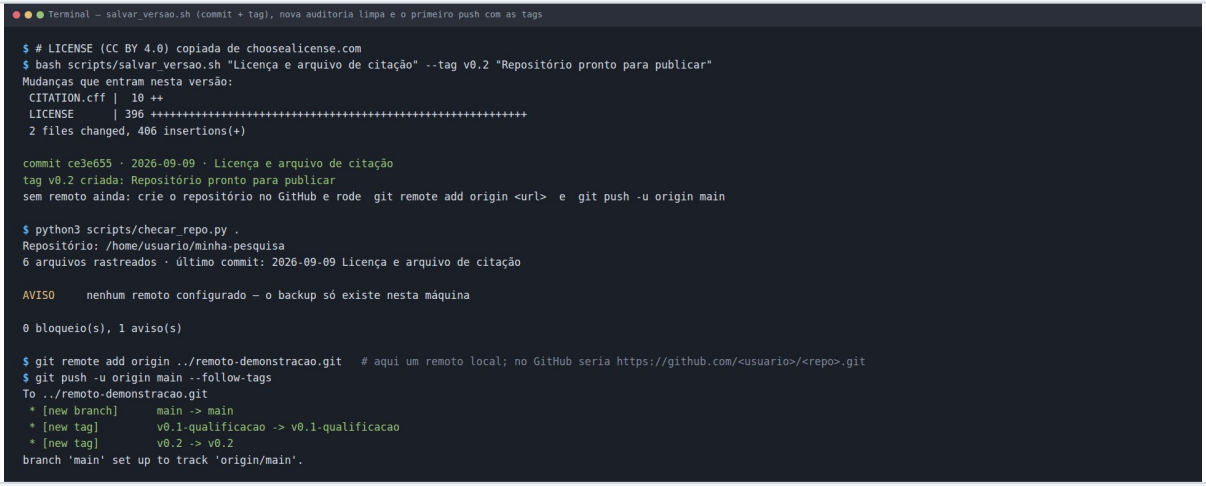

FIGURA 14 A sessão de 09/09/2026 (saída real, caminho trocado por `/home/usuario`): `salvar_versao.sh` faz o commit e a tag `v0.2` (Parte 5); `checar_repo.py` devolve «0 bloqueio(s), 1 aviso(s)» — o aviso é «nenhum remoto configurado»; `git remote add origin` e `git push -u origin main --follow-tags` respondem `* [new branch] main -> main`, `* [new tag] v0.1-qualificacao`, `* [new tag] v0.2` e «branch 'main' set up to track 'origin/main'». O remoto aqui é um repositório bare local (`../remoto-demonstracao.git`), porque nenhum repositório foi criado no GitHub para a demonstração; no GitHub, a URL é `https://github.com/<usuário>/<repo>.git`.

TERMINAL — LIGAR O PROJETO LOCAL AO REPOSITÓRIO RECÉM-CRIADO NO GITHUB

```
git remote add origin https://github.com/<usuário>/minha-pesquisa.git
git push -u origin main --follow-tags     # primeira vez: -u grava o vínculo; --follow-tags envia as tags anotadas
# das próximas vezes, na pasta do projeto:
git push                                  # envia os commits novos
```

REGRA

Ao fim de cada sessão: `git status` → commit → `git push`. Só depois do push o repositório existe em dois lugares — até lá, o `checar_repo.py` avisa: «o backup só existe nesta máquina». A URL do remoto se lê com `git remote -v`.

AUTENTICAÇÃO

```
git push --follow-tags                    # commits e tags novas
git pull                                  # traz o que mudou no
GitHub (edições pelo navegador, colegas)
```

> No primeiro push o Git pede que você se autentique no GitHub: siga o que a tela mostrar (a documentação de autenticação está em docs.github.com/pt). Nunca cole uma credencial em script, em arquivo versionado nem neste manual — o `checar_repo.py` procura exatamente esse tipo de conteúdo antes do push.

A página de um repositório real: árvore de arquivos, README renderizado, «About», releases, linguagens — e o arquivo aberto

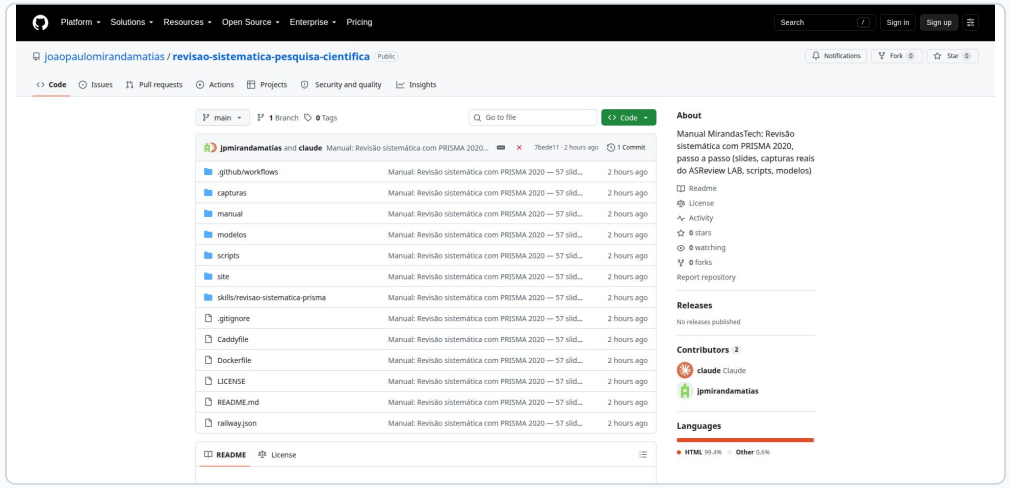

FIGURA 15 A página inicial de um repositório público desta série — o do manual de revisão sistemática (captura de 09/09/2026): árvore de arquivos com o último commit por item, README renderizado abaixo, e na barra lateral «About», «Releases» (ainda vazio) e «Languages».

FIGURA 16 O `README.md` do mesmo repositório aberto no navegador (captura de 09/09/2026): qualquer arquivo de texto pode ser lido, editado e ter o histórico consultado pela interface.

ELEMENTO	O QUE É
Árvore de arquivos	As pastas e arquivos do último commit do branch escolhid ao lado, a mensagem do commit que mexeu em cada um
README	Renderizado logo abaixo da árvore: é a «capa» do projeto (slide «README»)
«About»	Descrição, site e tópicos; edita-se pela engrenagem ao lad
«Releases»	

REGRA

A página do repositório é o que um revisor, um colega ou um avaliador vê primeiro. Três coisas a fazem legível: um README que diga o que o projeto é e como reproduzir, um LICENSE e um `CITATION.cff` — os três estão na Parte 4 e os três são o que o `checar_repo.py` cobra.

ELEMENTO	O QUE É
	Versões publicadas com notas e anexos (Parte 4); vazio até a primeira
«Languages»	Proporção de cada linguagem, calculada pelos arquivos
Abas	Commits, issues, pull requests, Actions e configurações — aparecem nos slides seguintes

NÃO CAPTURADO

O grafo de rede do repositório, que desenha branches e forks, devolveu uma página de erro nas tentativas sem login em 09/09/2026 e não aparece neste manual.

Commits e GitHub Actions: o histórico na web e um workflow que confere o repositório a cada push (integração contínua)

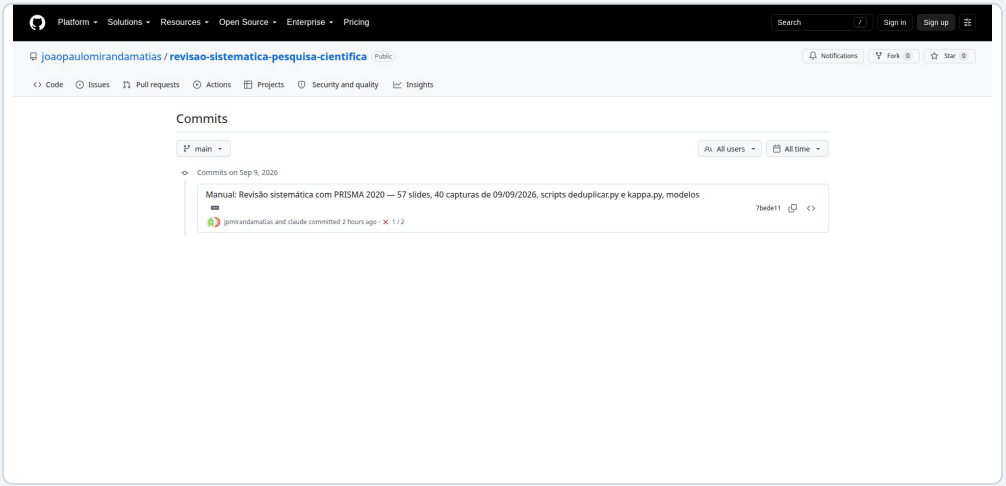

FIGURA 17 A lista de commits do mesmo repositório (captura de 09/09/2026): é o `git log` na web — mensagem, autor, data e identificador; cada linha abre o diff daquele commit.

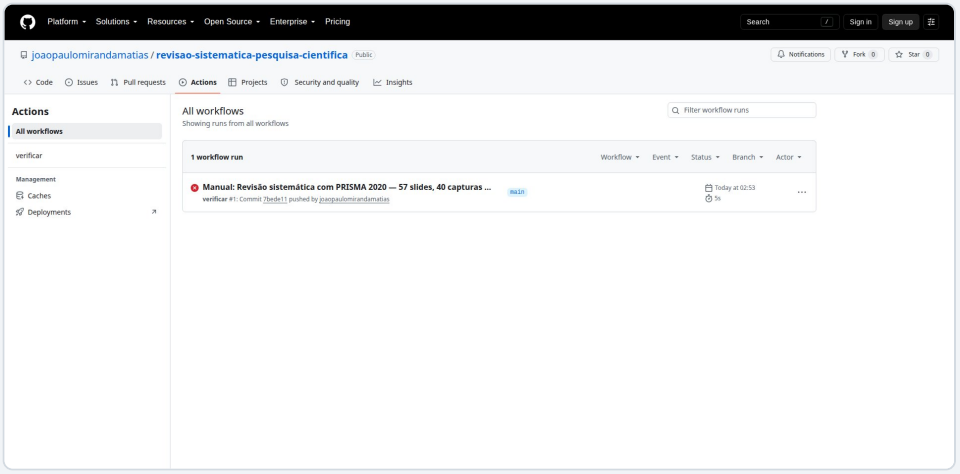

FIGURA 18 A aba «Actions» do mesmo repositório (captura de 09/09/2026): o workflow «verificar» (arquivo `verificar.yml`) rodou a cada push — o GitHub executou um script de conferência numa máquina virtual e registrou o resultado.

.GITHUB/WORKFLOWS/VERIFICAR.YML — UM WORKFLOW MÍNIMO QUE RODA O CHECAR_REPO.PY A CADA PUSH (EXEMPLO ILUSTRATIVO; SINTAXE EM DOCS.GITHUB.COM/PT/ACTIONS)

```
name: verificar
on: [push, pull_request]
jobs:
  checar:
    runs-on: ubuntu-latest
    steps:
```

TERMO	O QUE É
CI (integração contínua)	Rodar conferências e testes automaticamente a cada commit enviado
GitHub Actions	O serviço de CI do GitHub: arquivos YAML em `.githu` `workflows/`; 2.000 minutos por mês no plano Free
Workflow	Um arquivo YAML: quando rodar (`on`), em que máqu (`runs-on`) e os passos (`steps`)

```
      - uses: actions/checkout@v4
      - run: python3 scripts/checar_repo.py .
```

TERMO	O QUE É
Status	Verde ou vermelho ao lado de cada commit e em cada request

REGRA

Para a pesquisa, um *workflow* que rode a auditoria (`checar_repo.py`) e, se houver, os testes do código já vale: um push com segredo ou com arquivo grande fica vermelho antes de alguém clonar. Compilar o LaTeX ou reexecutar a análise no Actions é o passo seguinte, quando o projeto estabilizar.

Issues e pull requests: a lista de tarefas do projeto e a proposta de mudança com revisão

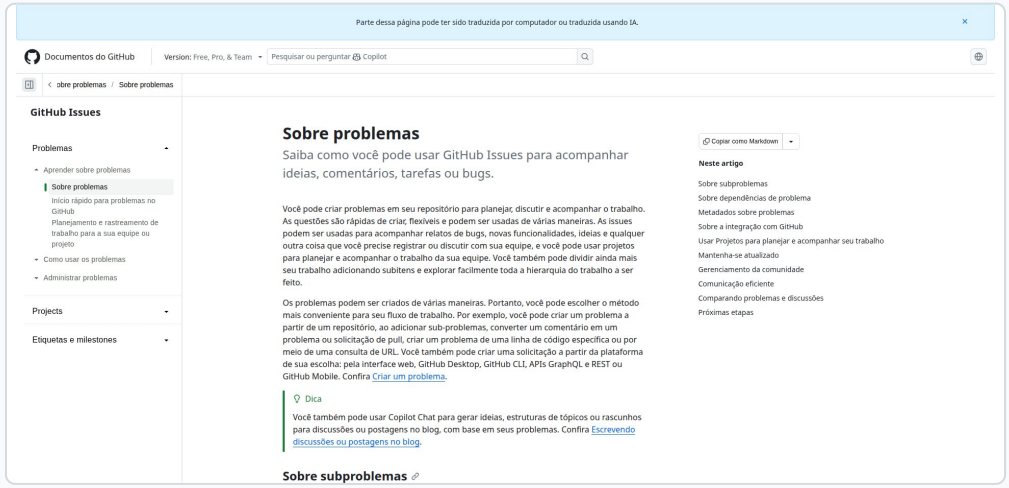

FIGURA 19 «Sobre problemas» (issues), em docs.github.com/pt (captura de 09/09/2026): as issues rastreiam tarefas, bugs e ideias no repositório — cada uma com título, texto, comentários, etiquetas e responsável.

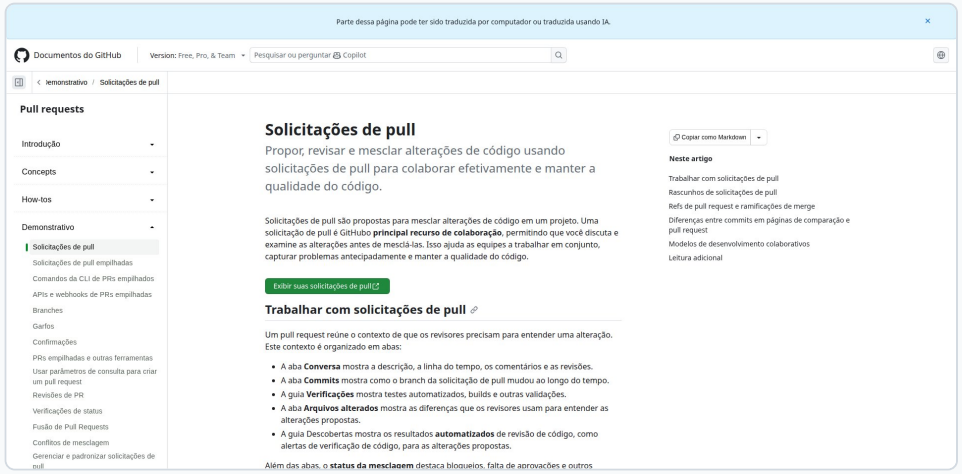

FIGURA 20 «Solicitações de pull», em docs.github.com/pt (captura de 09/09/2026): uma pull request propõe as mudanças de um branch para outro, com discussão e revisão antes do merge — o que o «Olá, Mundo» fez pelo navegador (Figura 13).

NA PESQUISA	USE ASSIM
Issue como tarefa	«Refazer a Tabela 3 com os dados de 2025», «Conferir a NBR 6023 nas referências» — numerada atribuída, fechada quando resolvida; o commit pod citar «#12»
Issue como dúvida do orientador	Um comentário por trecho, fora do e-mail, com histórico
Pull request	

> **REGRA**
>
> Issues e pull requests são públicas num repositório público: nada de dado de participante, nem de nome de avaliador, nem de trecho de artigo de base paga nos comentários. O que precisa ficar restrito fica no repositório privado ou fora do GitHub.

FORK

NA PESQUISA	USE ASSIM
	O branch `revisao-capitulo-2` vira uma PR: o orientador lê o diff, comenta linha a linha, e o merg acontece na web
Sozinho no projeto	Issues ainda valem como lista de pendências; a PR opcional — o merge local (Figura 7) faz o mesmo

> Para colaborar num repositório de outra pessoa sem permissão de escrita, faz-se um fork (cópia na sua conta), trabalha-se num branch e abre-se a pull request de volta — é como se contribui para pacotes de código aberto.

Sem terminal: o GitHub Desktop e a visão «Source Control» do VS Code fazem os mesmos commits e pushes

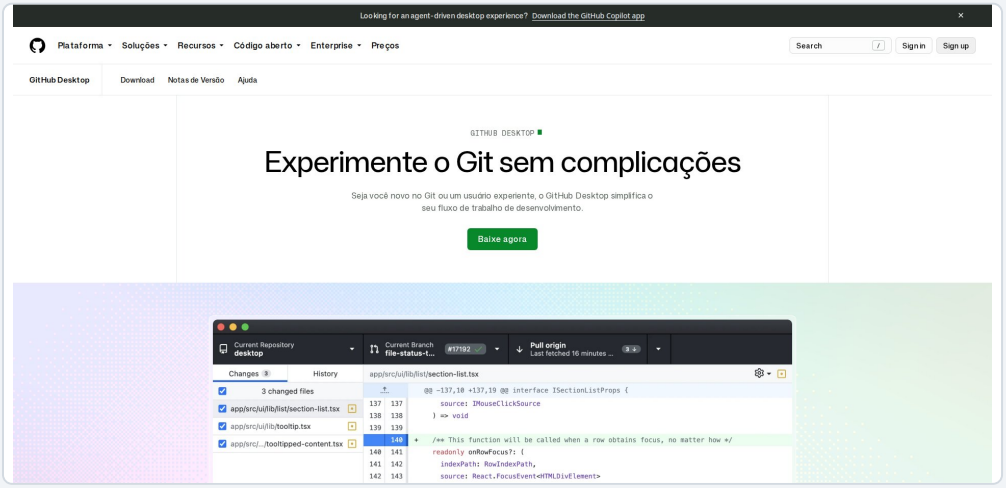

FIGURA 21 github.com/apps/desktop (captura de 09/09/2026): «Experimente o Git sem complicações». Cliente gráfico de código aberto (MIT), release 3.6.5 (04/09/2026), para Windows e macOS: clonar, ver mudanças, fazer commit, push e pull com botões.

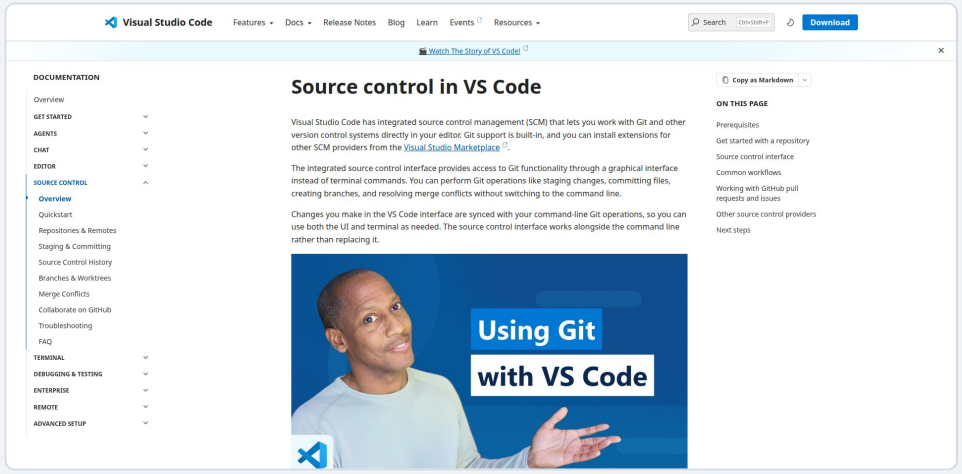

FIGURA 22 «Source control in VS Code» (captura de 09/09/2026): a visão «Source Control» (ícone na barra lateral; Ctrl+Shift+G) lista as mudanças; o «+» ao lado do arquivo faz o staging (também por trecho, na visão de diff); o commit leva a mensagem; a barra de status mostra a sincronização e os commits de entrada e saída.

FERRAMENTA	VANTAGENS	O QUE SABER
GitHub Desktop	Tudo por botões; diff visual; integra a conta do GitHub	Windows e macOS; código aberto sob MI release 3.6.5
VS Code	O editor onde você já escreve o `.tex` ou o `.py`; staging por arquivo ou por trecho; troca de	Exige Git ≥ 2.0.0 instalado e `user.name` / `user.email`

REGRA

Interface gráfica e terminal fazem a mesma coisa no mesmo repositório: use o VS Code no dia a dia e o terminal quando precisar de um comando que o botão não tem (tags anotadas, `rm --cached`, os scripts da Parte 5). Aprender os comandos primeiro (Parte 2) é o que faz os botões fazerem sentido.

FERRAMENTA	VANTAGENS	O QUE SABER
	branches; gráfico de commits e Timeline por arquivo; extensão «GitHub Pull Requests and Issues» para revisar PRs no editor	configurados; há geração de mensagem de commit por IA

> **LINUX**
>
> A página do GitHub Desktop não lista plataformas; o repositório distribui para Windows e macOS. No Linux, o VS Code, o terminal e a GitHub CLI cobrem tudo.

GitHub CLI (`gh`) e Codespaces: o GitHub no terminal e um computador de desenvolvimento no navegador

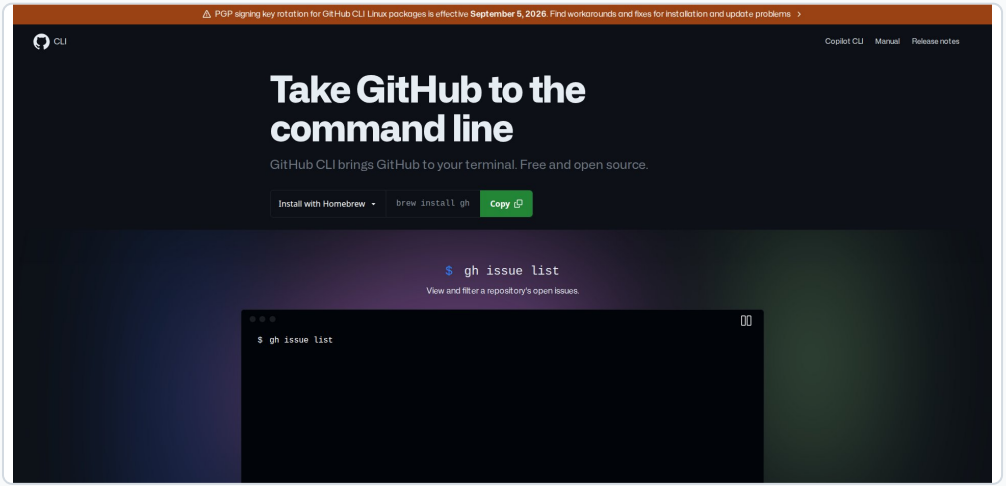

FIGURA 23 cli.github.com (captura de 09/09/2026): «GitHub CLI brings GitHub to your terminal. Free and open source» — repositórios, issues, pull requests, releases, checks e aliases pelo terminal; instalação por `brew install gh`, WinGet, apt/dnf/zypper; versão v2.100.0 (03/09/2026). Nesta máquina: gh 2.46.0.

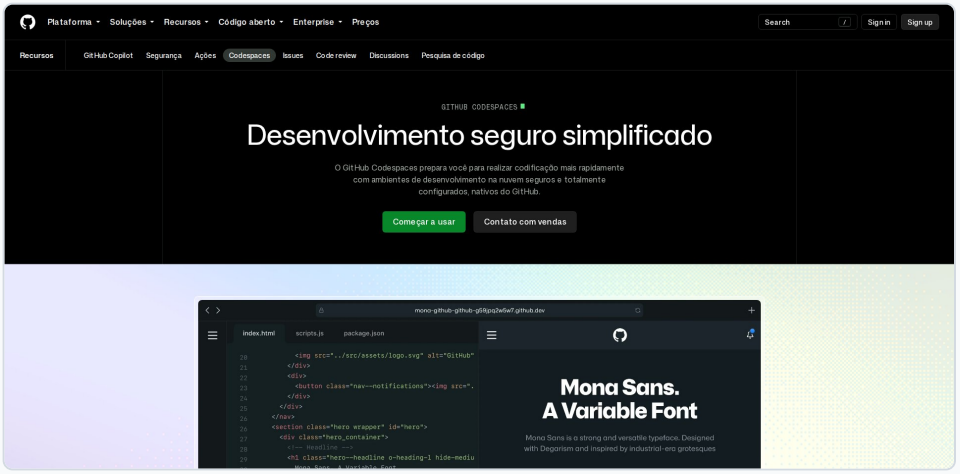

FIGURA 24 github.com/features/codespaces (captura de 09/09/2026): «ambiente de desenvolvimento hospedado na nuvem», usado pelo navegador, pelo VS Code ou pela CLI; em contas pessoais, 120 horas-núcleo e 15 GB por mês no plano Free.

TERMINAL — O QUE A GITHUB CLI FAZ SEM ABRIR O NAVEGADOR (A SINTAXE COMPLETA ESTÁ EM CLI.GITHUB.COM/MANUAL)

```
gh repo create minha-pesquisa --private --source=. --push   # cria o repositório no GitHub a partir da pasta e faz o push
gh issue create --title "Refazer a Tabela 3"                 # abre uma issue
gh pr create                                                 # abre a pull request do branch atual
```

REGRA

A CLI é o caminho mais curto de «pasta no computador» para «repositório no GitHub» sem clicar em formulário — e o que o plugin da Parte 5 usa quando o aluno pede para criar o repositório ou a release. Confira cada comando na página do manual antes de rodar:

```
gh release create v0.2 --notes "Repositório pronto para
publicar"   # publica uma release (Parte 4)
```

a sintaxe acima é a da documentação, não foi executada na demonstração.

CODESPACES NA PESQUISA

Serve para rodar um script ou compilar o LaTeX num computador emprestado, sem instalar nada: o repositório abre num VS Code no navegador. A cota gratuita mensal (120 horas-núcleo, 15 GB) acaba; não é lugar para dados sensíveis, que não devem estar no repositório de qualquer forma.

GitHub Pages: um site estático a partir do repositório — a página do projeto, a documentação, o manual

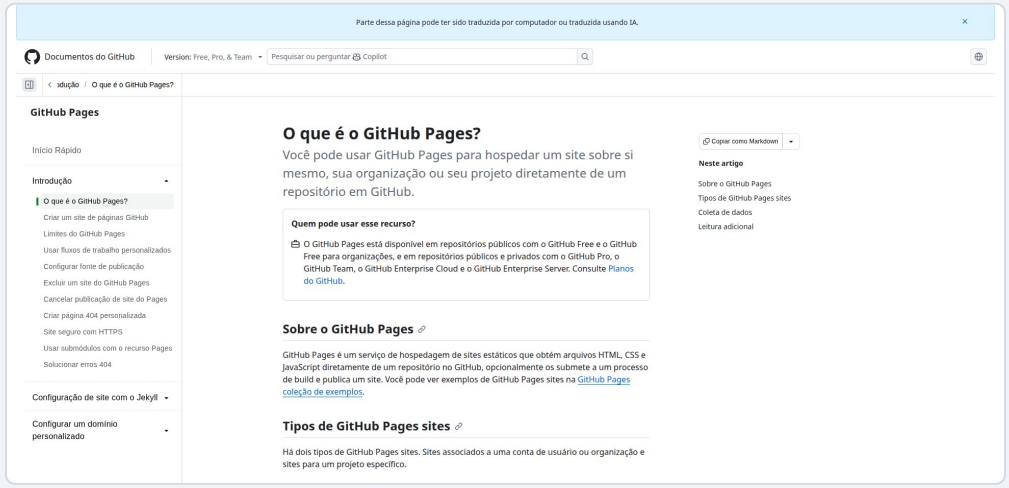

FIGURA 25 «O que é o GitHub Pages?», em docs.github.com/pt (captura de 09/09/2026): hospedagem de site estático (HTML, CSS, JavaScript) publicado diretamente de um repositório; gratuito em repositórios públicos no plano Free.

ITEM (DOCS, 09/09/2026)	VALOR
Site de usuário	Repositório `<owner>.github.io` → `https://<owner>.github.io`; um por conta
Site de projeto	`https://<owner>.github.io/<repo>`; um po repositório
Domínio próprio	Opcional
Limites	Repositório recomendado ≤ 1 GB; site ≤ 1 GB; larg banda flexível de 100 GB/mês; 10 builds por hora com Actions próprio)
Proibido	Uso comercial e e-commerce; envio de senhas ou cartões; os IPs dos visitantes são registrados

REGRA

Para a pesquisa, o Pages publica a página do projeto (o README com um índice), a documentação de um pacote ou um relatório em HTML. Dados de participantes e resultados sob embargo não entram: o site é público e o repositório também. Um formulário de coleta de dados no Pages viola a regra de não receber senhas ou cartões e, mais importante, a LGPD.

TELA LOGADA

A ativação fica nas configurações do repositório, que exigem login e não foram capturadas: consulte «O que é o GitHub Pages?» e os guias vizinhos em docs.github.com/pt.

Arquivos grandes e Git LFS: 50 MiB avisa, 100 MiB bloqueia — e o LFS guarda o conteúdo fora do histórico

FIGURA 26 «Sobre arquivos grandes no GitHub», em docs.github.com/pt (captura de 09/09/2026): aviso acima de 50 MiB, bloqueio acima de 100 MiB, 25 MiB pelo navegador; repositório idealmente abaixo de 1 GB e «fortemente recomendado» abaixo de 5 GB.

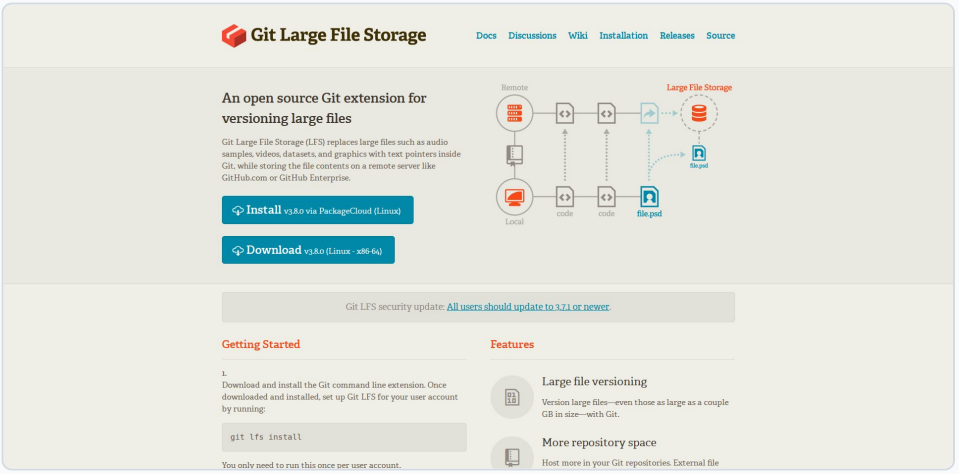

FIGURA 27 git-lfs.com (captura de 09/09/2026): o Git LFS «substitui arquivos grandes como amostras de áudio, vídeos, conjuntos de dados e gráficos por ponteiros de texto dentro do Git, guardando o conteúdo num servidor remoto»; versão v3.8.0 (28/08/2026).

TERMINAL — GIT LFS (GIT-LFS.COM; NÃO EXECUTADO NA DEMONSTRAÇÃO: NESTA MÁQUINA O LFS NÃO ESTÁ INSTALADO)

```
git lfs install                  # uma vez por máquina (Windows: instalador; macOS: brew install git-lfs; Linux: PackageCloud ou tarball)
git lfs track "*.psd"            # ou "*.mp4", "dados/*.parquet": grava a regra em .gitattributes
git add .gitattributes
git add dados/amostras.parquet   # commit e push normais: o Git
```

GIT LFS NO GITHUB (DOCS DE COBRANÇA)	VALOR
Free e Pro	10 GiB de armazenamento e 10 GiB de largura de
Team e Enterprise Cloud	250 GiB
Excedente	Cobrado por GiB baixado e por hora de armazenan (calculadora de preços do GitHub)

```
guarda o ponteiro, o LFS o conteúdo
git commit -m "Amostras via LFS"
```

REGRA

Antes do LFS, pergunte se o arquivo precisa estar no repositório: dado bruto grande pertence a um repositório de dados (OSF, Zenodo, o da instituição) com DOI, e o Git guarda o script que o lê. O LFS é para o que precisa acompanhar o código e cabe na cota. Um arquivo acima de 100 MiB já commitado bloqueia o push: tire-o do histórico antes de enviar — o `checar_repo.py` e o `salvar_versao.sh` recusam esses arquivos.

Overleaf: a integração Git e o GitHub Synchronization são recursos premium — no plano gratuito, «Download → Source» e commit local

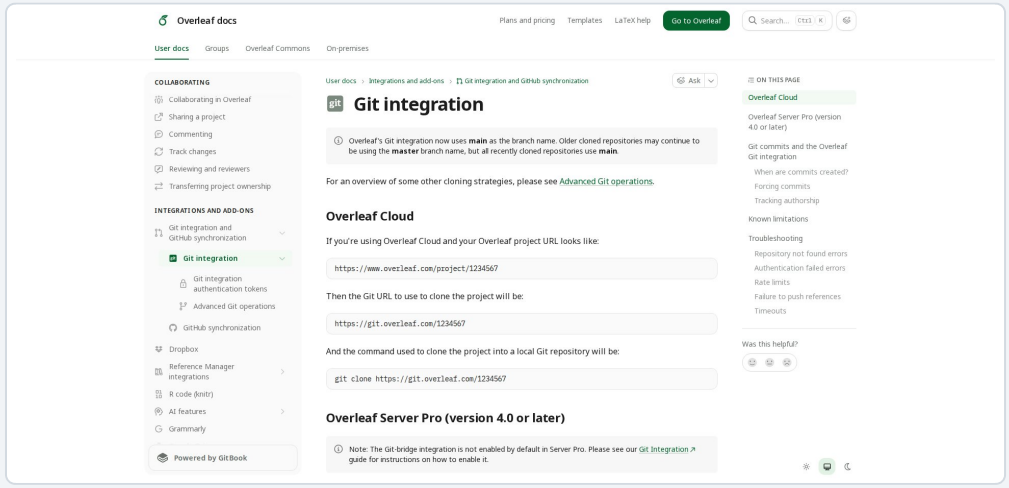

FIGURA 28 «Git integration and GitHub synchronization», em docs.overleaf.com (captura de 09/09/2026): ambos são recursos premium (Overleaf Cloud pago; Server Pro 4.0+ para Git). Clone: `https://git.overleaf.com/<ID_DO_PROJETO>`, com autenticação por token criado nas configurações da conta.

RECURSO	COMO (CONFORME A DOCUMENTAÇÃO)	PLANO
Git integration	Menu → Sync → Git; clonar `https://git.overleaf.com/<ID_DO_PROJETO>` com o token; o branch chama-se `main`; **não suporta** branches, Git LFS, tags nem submodules; symlinks viram arquivos	PREMIUM
GitHub Synchronization	Liga um projeto Overleaf a um repositório GitHub e os mantém sincronizados	PREMIUM (OVERLEAF CL
Plano gratuito	«Download → Source» (zip) → descompactar na pasta do repositório local → `git add`,	GRATUITO

RECURSO	COMO (CONFORME A DOCUMENTAÇÃO)	PLANO
	`git commit`, `git push`	

REGRA

No gratuito, o Overleaf é o editor e o Git é o histórico: baixe o «Source» ao fim de cada sessão de escrita, faça o commit e o push. As tags por marco (qualificação, defesa) só existem no Git — a integração premium do Overleaf não as suporta. O manual irmão de LaTeX detalha o editor e os planos.

TOKEN

O token do Overleaf é uma credencial: fica no gerenciador de credenciais do Git ou é digitado a cada clone — nunca num arquivo do repositório.

OSF: o add-on GitHub liga um repositório a um projeto do Open Science Framework — sincroniza os arquivos, não faz backup

1

Projeto → «Add-ons»

No projeto do OSF, a seção «Add-ons» → «Additional Storage»

2

«Connect» em GitHub

Nome da conta → «Authorize» → login no GitHub → «Authorize CenterForOpenScience»

3

Escolher o repositório

Um repositório por projeto do OSF

4

«Files»

Os arquivos do repositório aparecem na seção «Files» do projeto, com sincronização bidirecional

LIMITE (HELP.OSF.IO)	VALOR
Repositórios por projeto	Um
Backup	O OSF **não faz backup** do conteúdo do add-o o que está no GitHub continua só no GitHub
Arquivo enviado do OSF para o GitHub	Até 100 MB

O **Open Science Framework** é onde muitos programas pedem o pré-registro e o depósito de materiais. Com o add-on, o código e os scripts continuam versionados no GitHub e ficam visíveis no projeto do OSF, ao lado dos dados e do pré-registro — sem duplicar arquivos à mão.

> **REGRA**
>
> Divida por natureza: dados (brutos anonimizados, derivados, dicionário de variáveis) no armazenamento do OSF ou num repositório de dados com DOI; código, scripts e texto no GitHub, ligado ao projeto pelo add-on. O add-on não substitui o DOI do código — esse vem do Zenodo (Parte 4).

> **CONFORME A DOCUMENTAÇÃO**
>
> Os nomes «Add-ons», «Additional Storage», «Connect», «Authorize» e «Authorize CenterForOpenScience» são os de help.osf.io em 09/09/2026; o manual não fez login no OSF nem no GitHub para conferir a tela.

PARTE 4 PUBLICAR: README, LICENÇA, CITAÇÃO E DOI

README; licença; CITATION.cff e «Cite this repository»; cffinit; releases; DOI pelo Zenodo em dois slides; GitLab como alternativa

O repositório vira algo citável em quatro arquivos e dois cliques: README, LICENSE, `CITATION.cff` e uma *release* que o Zenodo arquiva com DOI. Tudo conferido em docs.github.com/pt, choosealicense.com, citation-file-format.github.io e help.zenodo.org em 09/09/2026, sem login.

04

README: o que o projeto faz, por que é útil, como começar, onde obter ajuda e quem mantém — em Markdown, na raiz

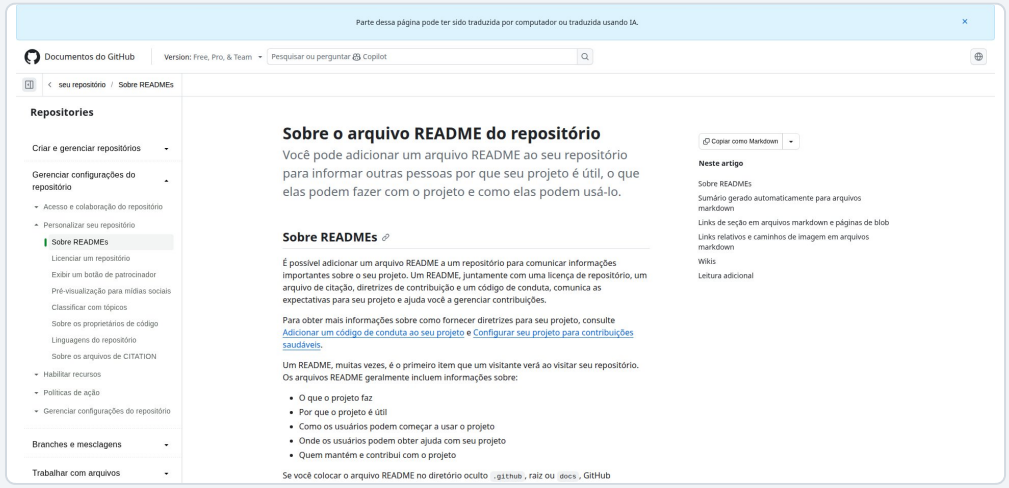

FIGURA 29 «Sobre o arquivo README do repositório», em docs.github.com/pt (captura de 09/09/2026): o GitHub procura o README em `.github/`, na raiz ou em `docs/` (nessa ordem); gera um sumário automático; acima de 500 KiB o arquivo é truncado; um README num repositório com o nome do usuário aparece no perfil.

README.MD — ESQUELETO PARA UM PROJETO DE PESQUISA (MARKDOWN; O GITHUB RENDERIZA)

```
# Manutenção preditiva em guindastes portuários: dados e
scripts

Scripts e dados derivados da dissertação «…» (versão 0.2,
set. 2026).

## O que há aqui
- `01_buscas/` — diário de busca e estratégias por base
- `06_texto/` — o texto em LaTeX
- `scripts/` — `checar_repo.py`, `citacao_cff.py`,
`salvar_versao.sh`

## Como reproduzir
1. `git clone https://github.com/<usuário>/minha-
pesquisa.git`
2. `python3 scripts/checar_repo.py .`

## Dados
Os dados brutos não estão neste repositório (LGPD); os
derivados
estão em `dados/` sob CC BY 4.0. Contato: e-mail
institucional.

## Como citar
```

```
Veja `CITATION.cff` ou o botão «Cite this repository».

## Licença
CC BY 4.0 (texto e dados) — ver `LICENSE`.
```

REGRA

Cinco perguntas, na ordem da documentação: o que faz, por que é útil, como começar, onde obter ajuda, quem mantém. Um README que diz onde os dados estão (e por que não estão aqui) evita a issue mais comum de um revisor. O `checar_repo.py` avisa quando o README falta.

Licença: sem um arquivo LICENSE ninguém pode reutilizar legalmente — MIT, GPLv3 ou Apache 2.0 para código; CC BY 4.0 para dados e texto

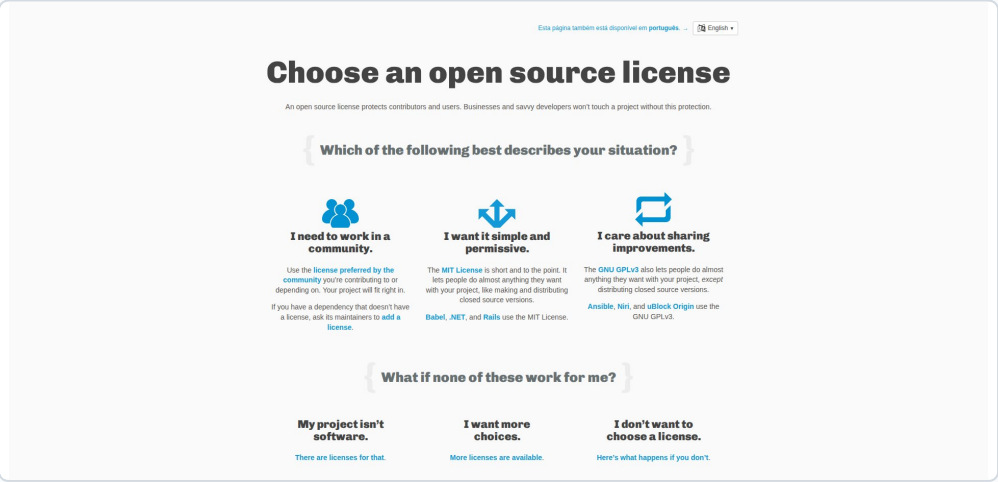

FIGURA 30 choosealicense.com (captura de 09/09/2026): em destaque, MIT («short and to the point»; permite quase tudo, inclusive versões fechadas) e GNU GPLv3 (igual, «except distributing closed source versions»); Apache 2.0 na página de licenças; as seções «I don't want to choose a license» e «My project isn't software». O site é CC BY.

LICENÇA	PARA	O QUE PERMITE (CHOOSEALICENSE.COM
MIT	Código	Curta e direta; quase tudo, inclusive versõe fechadas; exige manter o aviso de copyrigh
GNU GPLv3	Código	O mesmo, exceto distribuir versões fechada quem redistribui precisa abrir o código
Apache 2.0	Código	Permissiva; listada na página de licenças d — leia o texto antes de escolher
CC BY 4.0	Dados, texto, figuras	«My project isn't software»: uso, cópia e adaptação com atribuição — a licença desta
Nenhuma	—	«I don't want to choose a license»: sem lice ninguém pode legalmente reutilizar, mesm o repositório público

REGRA

Um arquivo `LICENSE` na raiz com o texto integral da licença (copie de choosealicense.com — na demonstração, a CC BY 4.0 veio do repositório github/choosealicense.com). Código e dados no mesmo repositório podem ter licenças diferentes: diga no README qual

vale para o quê. Licença é condição para o DOI pelo Zenodo (slide «DOI pelo Zenodo»).

NÃO É ACONSELHAMENTO JURÍDICO

A escolha depende do programa, do financiador e de coautores; dados de terceiros e artigos de bases pagas têm licença própria e não entram no repositório. Em dúvida, pergunte à instituição.

`CITATION.cff`: um arquivo na raiz e o GitHub mostra «Cite this repository», com APA e BibTeX prontos

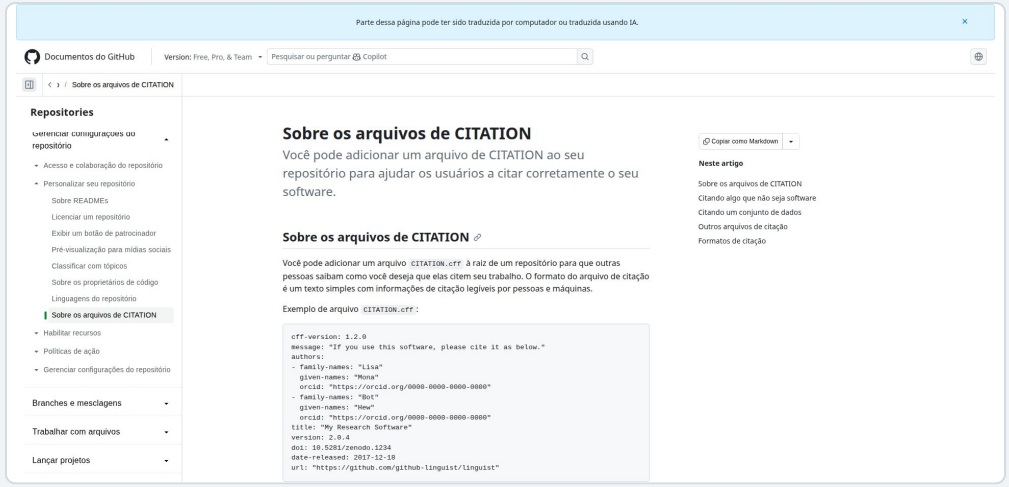

FIGURA 31 «Sobre os arquivos de CITATION», em docs.github.com/pt (captura de 09/09/2026): com um `CITATION.cff` na raiz, a barra lateral do repositório ganha o botão «Cite this repository», com exportação em APA e BibTeX; `preferred-citation` aponta para o artigo quando se quer que citem o artigo, não o software.

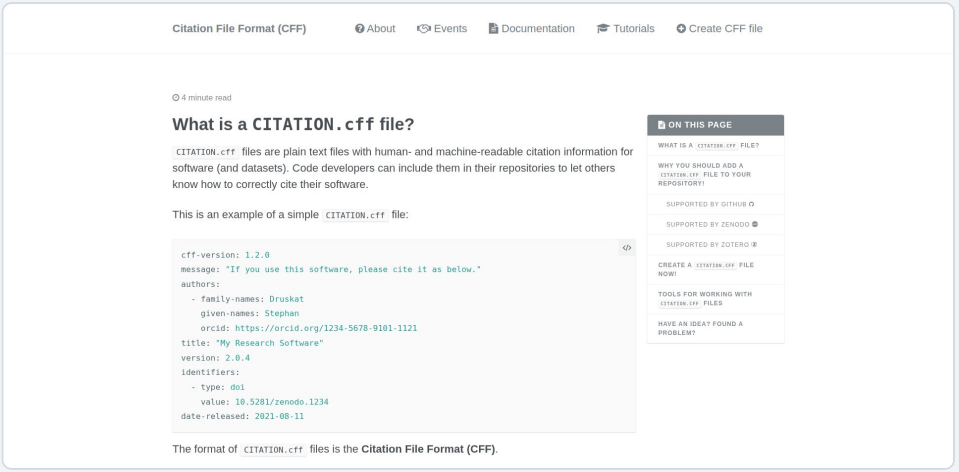

FIGURA 32 citation-file-format.github.io (captura de 09/09/2026): Citation File Format, versão 1.2.0; ferramentas **cffinit** (formulário web) e **cff-converter-python** (`cffconvert`, para BibTeX, RIS e CodeMeta); o arquivo é lido pelo GitHub, pelo Zenodo e pelo Zotero.

CITATION.CFF — O GERADO NA DEMONSTRAÇÃO PELO CITACAO_CFF.PY (FIGURA 40); CAMPOS BÁSICOS DA DOCUMENTAÇÃO

```
cff-version: 1.2.0
message: "Se você usar este software ou material, cite-o como abaixo."
title: "Manutenção preditiva em guindastes portuários: dados e scripts"
```

CAMPO	O QUE É
`cff-version`	Versão do formato: 1.2.0
`message`	O pedido de citação que o leitor vê
`authors`	Lista com `family-names` e `given-names` (e, se h `orcid`)

```
version: "0.1"
date-released: 2026-09-09
authors:
  - family-names: "Exemplo"
    given-names: "Aluna"
url: "https://github.com/aluna/minha-pesquisa"
license: CC-BY-4.0
```

CAMPO	O QUE É
`title` · `version`	Título e versão citadas; `date-released`, `url`, `license` e `doi` completam
`preferred-citation`	Quando há artigo: os metadados do artigo, para que o cite o artigo

REGRA

Um `CITATION.cff` por repositório, na raiz, atualizado a cada *release* (versão, data e, depois do Zenodo, o DOI). O Zenodo lê o arquivo para preencher os metadados do registro; o Zotero o importa. O `citacao_cff.py` (Parte 5) gera o arquivo e imprime o BibTeX e a referência ABNT para conferir com a biblioteca.

cffinit: o formulário web que monta ou atualiza o `CITATION.cff` sem escrever YAML à mão

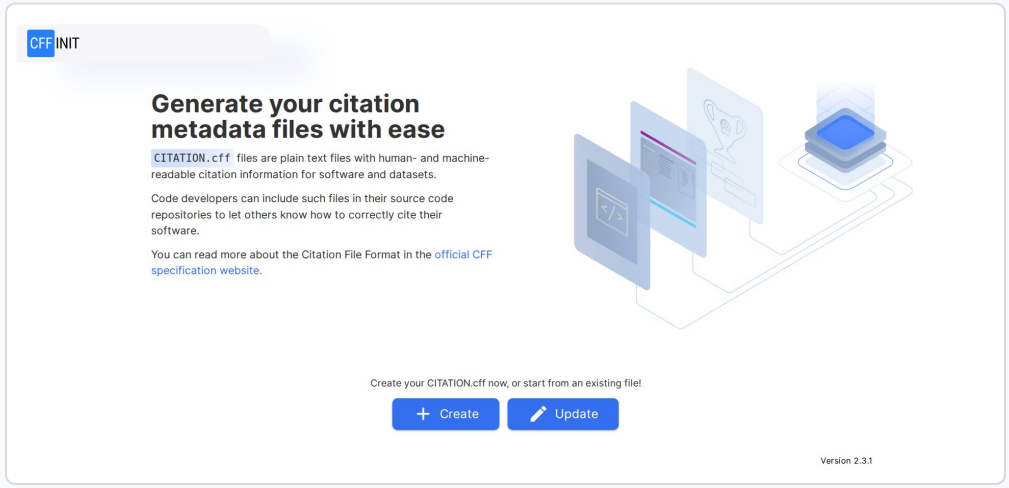

FIGURA 33 cffinit (captura de 09/09/2026): «Create» começa um arquivo novo; «Update» carrega um `CITATION.cff` existente para editar. O formulário guia campo a campo e entrega o arquivo pronto para salvar na raiz do repositório.

1

«Create» ou «Update»

Novo, ou carregando o arquivo que já existe no repositório

2

Preencher

Os campos do formato: título, autores (`family-names` , `given-names`), versão, data, licença, URL, DOI se já houver

3

Baixar

O `CITATION.cff` gerado; salve na raiz do repositório

4

Commit e push

`git add CITATION.cff` → commit → push; o botão «Cite this repository» aparece

FERRAMENTA	QUANDO
cffinit (web)	Primeira vez, ou para quem prefere formulá valida o arquivo
`citacao_cff.py` (Parte 5)	Pelo terminal, com BibTeX e ABNT impress recusa sobrescrever sem `--forcar`
`cffconvert` (cff-converter-python)	

FERRAMENTA	QUANDO
	Converter o `.cff` para BibTeX, RIS ou CodeMeta

REGRA

Os autores do `CITATION.cff` são os autores do *repositório* (quem escreveu o código e organizou os dados), não necessariamente os do artigo — para o artigo há o `preferred-citation`. Use o ORCID de cada autor: é o que liga o DOI do Zenodo ao perfil.

Releases: «Versões» → «Rascunho de uma nova versão» → tag → título → notas → «Publicar versão» — cada release traz ZIP e tar.gz do código

FIGURA 34 «Gerenciar versões em repositórios», em docs.github.com/pt (captura de 09/09/2026): «Versões» → «Rascunho de uma nova versão» → «Escolher uma marca» (tag existente ou nova) → «Título da versão» → «Descrever esta versão» (ou «Gerar notas de lançamento») → anexos opcionais → «Este é um pré-lançamento» → «Definir como versão mais recente» → «Publicar versão» ou «Salvar rascunho».

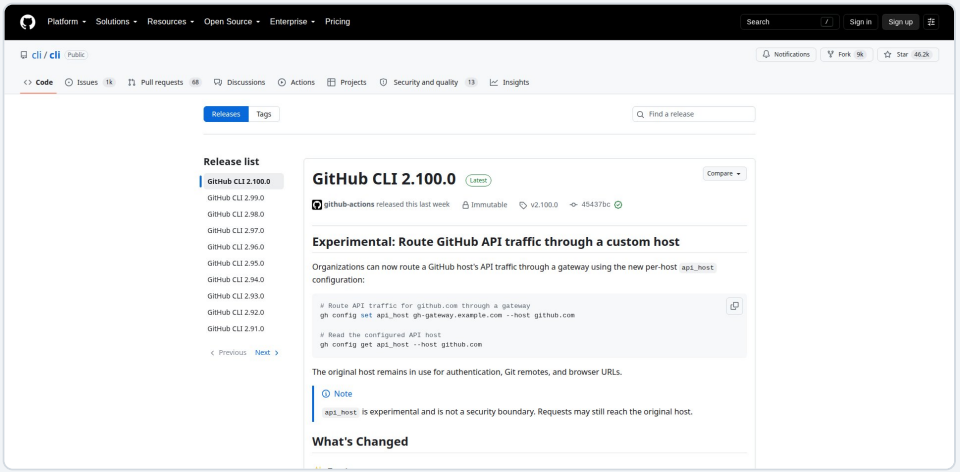

FIGURA 35 A página de releases de um projeto de terceiros — a GitHub CLI (captura de 09/09/2026): cada versão com notas, anexos e, automaticamente, «Source code (zip)» e «Source code (tar.gz)». É a aparência que o seu repositório terá depois da primeira release.

PASSO (DOCS, 09/09/2026)	O QUE FAZER
«Versões» → «Rascunho de uma nova versão»	Na página do repositório
«Escolher uma marca»	A tag anotada que você enviou com `--follow-tags` (`v0.2`), ou uma nova criada ali
«Título da versão» · «Descrever esta versão»	

REGRA

Uma release por marco citável: a versão do código que gerou os resultados do artigo, a versão depositada com a dissertação. A tag no Git (Parte 2) é o ponto fixo; a release é a tag com nome, notas e anexos — e é ela que o Zenodo arquiva. «Este é um pré-lançamento» serve para versões em revisão.

PASSO (DOCS, 09/09/2026)	O QUE FAZER
	O que mudou desde a anterior; o botã «Gerar notas de lançamento» escreve as notas automaticamente
Anexos	Opcionais: o PDF da dissertação, uma planilha de resultados
«Publicar versão»	A release fica pública; se o Zenodo estiver ligado, o DOI é emitido (slides seguintes)

> PELO TERMINAL
>
> `gh release create v0.2 --notes "…"` (GitHub CLI) faz o mesmo sem abrir o navegador.

DOI pelo Zenodo: repositório público, com licença, ligado à conta — e cada release ganha um DOI

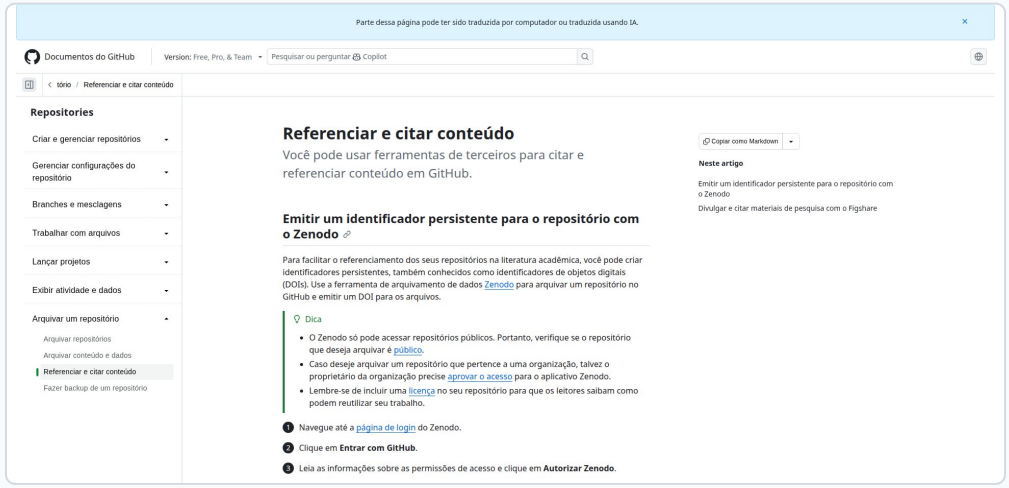

FIGURA 36 «Referenciar e citar conteúdo», em docs.github.com/pt (captura de 09/09/2026): condições — repositório público e com licença; passos — zenodo.org → «Entrar com GitHub» → «Autorizar Zenodo» → página GitHub do Zenodo → alternar o repositório para «Ativado»; «o Zenodo arquiva seu repositório e emite um novo DOI sempre que você cria uma versão (release) do GitHub».

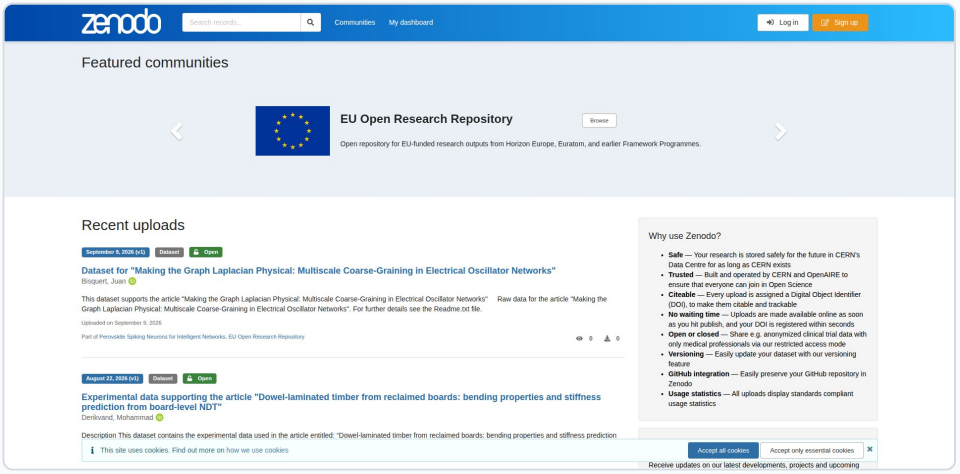

FIGURA 37 zenodo.org (captura de 09/09/2026): o repositório aberto que arquiva o código, os dados e o texto e emite o DOI — o mesmo que registra o The Turing Way (DOI 10.5281/zenodo.3332807).

CONDIÇÃO	POR QUÊ
Repositório público	O Zenodo só arquiva o que pode ler; um repositóri privado não recebe DOI por esta via
Arquivo LICENSE	É a segunda condição da documentação do GitHul o registro arquivado precisa dizer sob que termos pode ser reutilizado

REGRA

Ordem: LICENSE e `CITATION.cff` no repositório → repositório público → ligar no Zenodo → release. O `CITATION.cff` (ou um `.zenodo.json`) preenche os metadados do registro — título, autores com ORCID, licença — e evita corrigir à mão no Zenodo

CONDIÇÃO	POR QUÊ
Conta no Zenodo ligada ao GitHub	«Entrar com GitHub» → «Autorizar Zenodo»: é o que permite ao Zenodo ver as releases
Uma release	O DOI é emitido por release, não por commit; a primeira release depois de «Ativado» gera o primeiro DOI

> depois. Depois do DOI, coloque-o no `CITATION.cff` e no README.

> O QUE NÃO ESTÁ AQUI
>
> Limites de tamanho por registro no Zenodo não foram conferidos nesta data e não aparecem no manual. Telas logadas do Zenodo não foram capturadas: os nomes de botão são os da documentação do GitHub em português e de help.zenodo.org.

DOI pelo Zenodo, passo a passo: perfil → GitHub → «Sync now» → ligar o controle deslizante → publicar a release

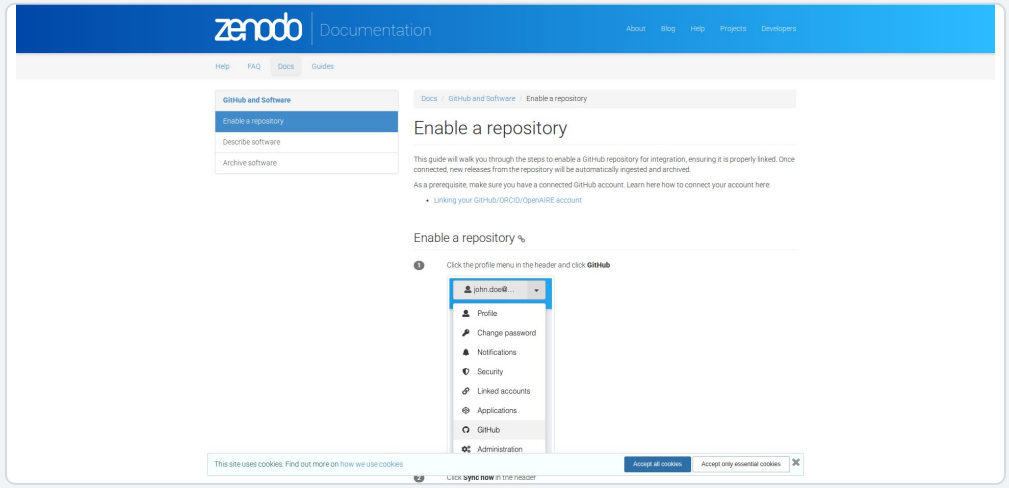

FIGURA 38 «Enable a repository», em help.zenodo.org (captura de 09/09/2026): menu do perfil → GitHub → «Sync now» → ligar o controle deslizante do repositório; as novas releases «são automaticamente ingeridas e arquivadas»; metadados por `CITATION.cff` ou `.zenodo.json`; integração com o Software Heritage.

1

zenodo.org → «Entrar com GitHub»

→ «Autorizar Zenodo» (docs do GitHub em português)

2

Perfil → GitHub

A página GitHub do Zenodo (zenodo.org/account/settings/github/) lista os seus repositórios; «Sync now» atualiza a lista

3

Ligar o controle deslizante

Alternar o repositório para «Ativado»

4

Publicar a release no GitHub

O Zenodo arquiva e emite o DOI; cada release seguinte ganha um DOI novo

DEPOIS DO DOI	ONDE COLOCAR
`CITATION.cff`	Campo `doi`; commit → a próxima release já nasc com ele nos metadados
README	Na seção «Como citar»
Artigo ou dissertação	

DEPOIS DO DOI	ONDE COLOCAR
	Na seção de disponibilidade de dados e código, e r referências (a referência ABNT sai do `citacao_cff.py`)

REGRA

O DOI aponta para o registro no Zenodo, não para o GitHub: mesmo que o repositório mude de nome ou saia do ar, o arquivo permanece. Cite a versão que gerou os resultados (o DOI daquela release), e não «o repositório».

GitLab como alternativa: o mesmo Git por baixo, plano gratuito sem cartão — e muitas universidades hospedam o próprio

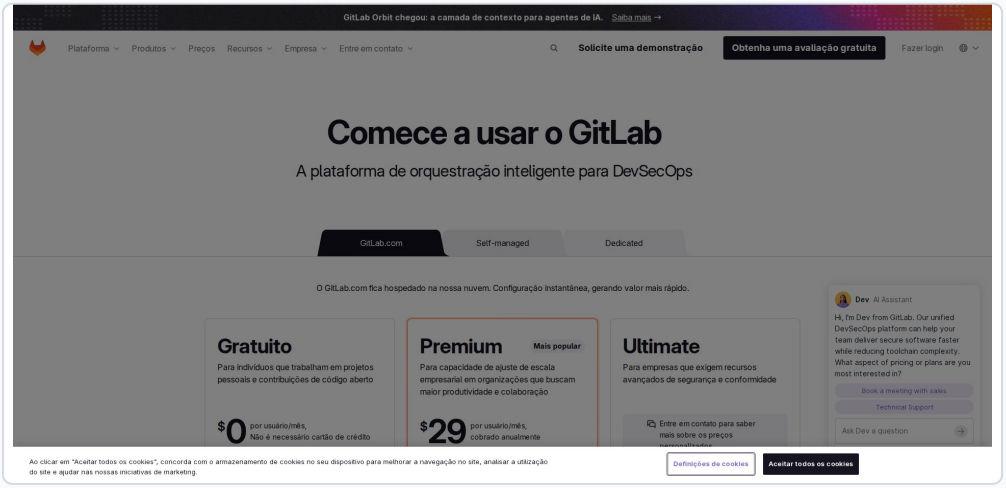

FIGURA 39 about.gitlab.com/pricing em português (captura de 09/09/2026): «Gratuito» US$ 0 por usuário/mês, «Não é necessário cartão de crédito», «5 usuários por grupo de nível superior», «400 minutos de computação por mês», gerenciamento de código-fonte e CI/CD, «Armazenamento ajustável de 10 GiB»; «Premium» US$ 29 por usuário/mês cobrado anualmente; «Ultimate» sob consulta.

PLANO GITLAB	O QUE INCLUI (PRICING, 09/09/202
Gratuito (US$ 0)	5 usuários por grupo de nível superior; 40 minutos de computação por mês; código-f e CI/CD; armazenamento ajustável de 10 sem cartão de crédito
Premium (US$ 29/usuário/mês, anual)	Recursos de equipe e de planejamento
Ultimate	Preço sob consulta na página

TERMINAL — LEVAR O MESMO REPOSITÓRIO PARA O GITLAB (OU PARA A INSTÂNCIA DA INSTITUIÇÃO)

```
git remote add gitlab https://gitlab.com/<usuário>/minha-pesquisa.git
git push -u gitlab main --follow-tags     # o histórico e as tags vão inteiros; o GitHub continua como origin
```

REGRA

Se a instituição hospeda um GitLab, ele pode ser o lugar do trabalho em andamento (privado, dentro da rede) e o GitHub o da publicação (público, com «Cite this repository» e Zenodo). Um repositório pode ter dois remotos; os comandos são os mesmos. O *Turing Way* cita ainda o Codeberg como alternativa comunitária.

PARTE 5 OS SCRIPTS E O FECHAMENTO

`checar_repo.py`, `citacao_cff.py` e `salvar_versao.sh`; o plugin; erros comuns; checklist; glossário; referências; como citar

Três scripts curtos, rodados de verdade em 09/09/2026, fazem o que este manual pede antes de cada push: auditar o repositório, gerar o arquivo de citação e registrar a versão com tag. Depois, o que costuma dar errado, a lista para conferir, os termos e as fontes.

05

`checar_repo.py`: a auditoria antes do push — segredos, dados pessoais, arquivos grandes, README, LICENSE, CITATION.cff, remoto

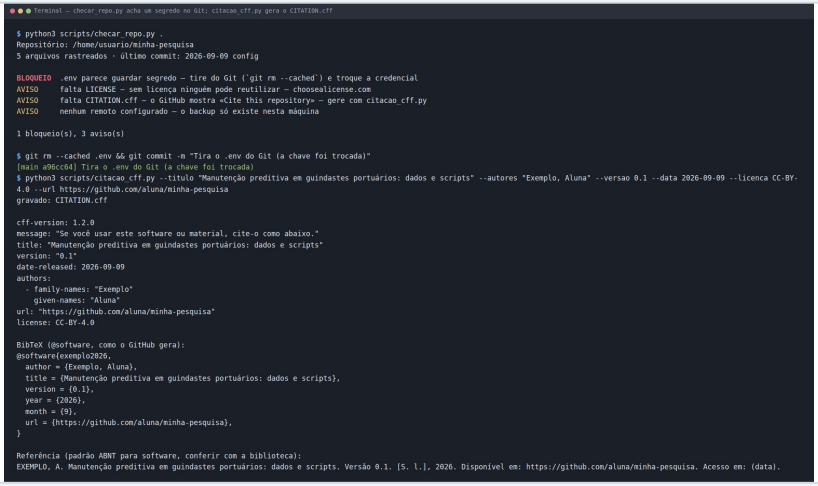

FIGURA 40 A sessão de 09/09/2026 (saída real, caminho trocado por `/home/usuario`): `python3 scripts/checar_repo.py .` lista «5 arquivos rastreados», um **BLOQUEIO** («.env parece guardar segredo — tire do Git (`git rm --cached`) e troque a credencial») e três **AVISO**s (falta LICENSE, falta CITATION.cff, nenhum remoto configurado); depois do `git rm --cached .env` e do commit, o `citacao_cff.py` grava o `CITATION.cff` e imprime o BibTeX `@software` e a referência ABNT (slide seguinte).

O QUE CONFERE	COMO APARECE
Arquivos grandes	Rastreados acima de 50 MiB → AVISO; acima de 100 MiB → BLOQUEIO (os limites do GitHub); binários acima de 5 MiB → aviso
Segredos	Nomes (`.env`, `.pem`, `id_rsa`) e conteúdos (tokens `ghp_`, `AKIA`, `sk-`, chave privada PEM) → BLOQUEIO
Dados pessoais (LGPD)	

TERMINAL — USO

```
python3 scripts/checar_repo.py .          # a pasta atual (ou o caminho de outro repositório)
python3 scripts/checar_repo.py . && git push --follow-tags
# só faz o push se não houver bloqueio
```

REGRA

O QUE CONFERE	COMO APARECE
	Padrões de CPF e listas de e-mails em arquivos de texto; pastas de dados brutos (`dados_FORA_DO_GIT/` etc.) rastreadas
Publicação	Falta de README, LICENSE, `CITATION.cff` ou `.gitignore` ; ausência de remoto; mudanças sem commit
Saída	«N bloqueio(s), M aviso(s)»; código de saída 1 se houver bloqueio — por isso serve no GitHub Actions (Figura 18)

> O script só lê: não apaga, não faz commit, não envia nada. Um BLOQUEIO é decisão sua de resolver — na demonstração, `git rm --cached .env` tirou o arquivo do Git e a mensagem do commit registrou que a chave foi trocada. Os AVISOs de LICENSE e `CITATION.cff` sumiram na segunda auditoria (Figura 14), depois dos slides da Parte 4.

`citacao_cff.py`: grava o `CITATION.cff` 1.2.0 e imprime o BibTeX `@software` e a referência ABNT para conferir com a biblioteca

TERMINAL — O COMANDO DA DEMONSTRAÇÃO (FIGURA 40)

```
python3 scripts/citacao_cff.py \
    --titulo "Manutenção preditiva em guindastes portuários:
dados e scripts" \
    --autores "Exemplo, Aluna" \
    --versao 0.1 --data 2026-09-09 \
    --licenca CC-BY-4.0 \
    --url https://github.com/aluna/minha-pesquisa
# gravado: CITATION.cff
# depois do Zenodo: --doi 10.5281/zenodo.NNNNNNN --forcar
(regrava com o DOI)
```

SAÍDA — O QUE O SCRIPT IMPRIME DEPOIS DO ARQUIVO (FIGURA 40)

```
BibTeX (@software, como o GitHub gera):
@software{exemplo2026,
  author = {Exemplo, Aluna},
  title = {Manutenção preditiva em guindastes portuários: dados
e scripts},
  version = {0.1},
  year = {2026},
  month = {9},
  url = {https://github.com/aluna/minha-pesquisa},
}

Referência (padrão ABNT para software, conferir com a
```

ARGUMENTO	USO
`--titulo` · `--autores`	Obrigatórios; autores como «Sobrenome, Nome; Sobrenome, Nome» — cada um vira `family-names` / `given-names`
`--versao` · `--data`	A versão citada (a mesma da tag e release) e a data de lançamento (A MM-DD)
`--doi` · `--url` · `--licenca` · `--orcid`	Opcionais; o DOI entra depois da primeira release no Zenodo
`--forcar`	O script recusa sobrescrever um `CITATION.cff` existente sem e opção

REGRA

A referência ABNT impressa é um ponto de partida «para conferir com a biblioteca»: a NBR 6023 para software tem detalhes (local, «[S. l.]», data de acesso) que a sua biblioteca confirma. O BibTeX vai para o `.bib` de quem citar o seu repositório — é o mesmo que o botão «Cite this repository» exporta.

```
biblioteca):
EXEMPLO, A. Manutenção preditiva em guindastes portuários: dados
e scripts.
Versão 0.1. [S. l.], 2026. Disponível em: https://github.com/
aluna/minha-pesquisa.
Acesso em: (data).
```

MANUAL IRMÃO

Como citar software e dados no texto e nas referências é assunto do manual de Zotero (o Zotero lê o `CITATION.cff`) e do manual de LaTeX (o `.bib`).

`salvar_versao.sh`: `git add -A`, o resumo do que entra, o commit e a tag anotada num comando — e o push impresso, não executado

TERMINAL — O COMANDO DA DEMONSTRAÇÃO (FIGURA 14) E A SAÍDA QUE ELE IMPRIMIU

```
bash scripts/salvar_versao.sh "Licença e arquivo de citação" --
tag v0.2 "Repositório pronto para publicar"
Mudanças que entram nesta versão:
 CITATION.cff |  10 ++
 LICENSE      | 396 ++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++
 2 files changed, 406 insertions(+)

commit ce3e655 · 2026-09-09 · Licença e arquivo de citação
tag v0.2 criada: Repositório pronto para publicar
sem remoto ainda: crie o repositório no GitHub e rode  git
remote add origin <url>  e  git push -u origin main
```

TERMINAL — USO NO DIA A DIA

```
bash scripts/salvar_versao.sh "Capítulo 3: resultados da
busca"          # só o commit
bash scripts/salvar_versao.sh "Versão da defesa" --tag v1.0
"Versão entregue à banca"   # commit + tag anotada
git push --follow-
tags                                                    # o
script imprime este comando; você decide rodar
```

1

`git add -A`

Adiciona tudo o que o `.gitignore` não exclui

2

Mostra o `--stat`

A lista de arquivos e linhas que entram: leia antes de continuar

3

Recusa > 100 MiB

Um arquivo acima do bloqueio do GitHub interrompe o script antes do commit

4

Commit e tag

O commit com a mensagem; com `--tag`, a tag anotada (`-a -m`) com a descrição

5

Imprime o push

`git push --follow-tags` — ou orienta a criar o remoto, como na demonstração

REGRA

O script não faz push: enviar é decisão sua, depois do `checar_repo.py`. A tag só quando houver um marco — a

mensagem da tag («Versão entregue à banca») é o que aparece no GitHub ao criar a release com «Escolher uma marca» (Parte 4).

SEM BASH

O script é bash; onde não houver um terminal bash, troque pelo equivalente manual: `git add -A`, `git diff --staged --stat`, `git commit -m`, `git tag -a … -m`.

Tudo num só plugin: a skill `git-github-pesquisadores` audita e explica; o aluno cria o repositório, a release e o DOI

O plugin Pesquisa MirandasTech (para Claude Code e Codex) coloca os papéis dos manuais dentro do assistente, com um estado único (`PROGRESSO.md`) e um gerente — o Pesquisador — que só fecha uma fase com evidência. O `iniciar_projeto.py` do plugin já roda o `git init` e deixa a pasta `dados_FORA_DO_GIT/` fora do versionamento. A skill `git-github-pesquisadores` segue este manual: roda o `checar_repo.py`, explica cada BLOQUEIO e AVISO, gera o `CITATION.cff` e prepara a versão — mas quem cria o repositório no GitHub, publica a release e liga o Zenodo é o aluno, na própria conta.

PAPEL	O QUE FAZ COM ESTE MANUAL	SLIDES
Pesquisador	Confere que o repositório existe, que o `checar_repo.py` dá zero bloqueios, que há README, LICENSE e `CITATION.cff`, e que a versão entregue tem tag	Partes 2, 4 e 5
Skill git-github-pesquisadores	Explica os comandos, lê a saída do Git, monta o `.gitignore`, roda os três scripts, orienta a release e o DOI	Partes 2 a 5
Bibliotecário		Parte 5

CLAUDE CODE — PEDIDOS NA PASTA DO PROJETO (O PLUGIN LÊ PROGRESSO.MD)

```
Pesquisador, onde paramos?
Audite o repositório com o checar_repo.py e me explique cada
aviso.
Gere o CITATION.cff com os autores do projeto e mostre a
referência ABNT.
Salve a versão «Capítulo 3: resultados» e me diga se devo
fazer push.
O que falta para pedir o DOI no Zenodo?
/pesquisa:status
/pesquisa:proxima
```

REGRA

O agente nunca faz push, nunca cria repositório na conta do aluno e nunca toca em credencial: ele audita, explica e prepara. Criar o repositório, publicar a release e autorizar o Zenodo são cliques do aluno, logado na própria conta — como neste manual, que não fez login em nenhum serviço.

ONDE CADA MANUAL ENTRA

PAPEL	O QUE FAZ COM ESTE MANUAL	SLIDES
	Confere a referência ABNT do software e importa o `CITATION.cff` no Zotero	
Orientador (uso de IA)	Declara o uso do assistente no repositório e no texto	Último slide

> Metodologia decide o projeto; Busca e PRISMA produzem o conteúdo; Zotero guarda as referências; LaTeX escreve o texto; este manual versiona e publica; Uso de IA declara; o plugin mantém a ordem. Hub: mirandastech.com.br/pesquisa.

Erros comuns — como aparecem e como corrigir

ERRO	COMO APARECE	CAUSA	CORREÇÃO
Segredo no histórico	`.env`, token ou chave num commit — mesmo que apagado depois	`git add -A` antes do `.gitignore`	ESTÁ VAZADO: TROCAR A CREDENCIAL; GIT RM --CACHED; .GITIGNORE
Dado pessoal em repositório privado	CPF, nome, e-mail de participante num `.csv` «só para mim»	Confundir privado com anonimizado (LGPD)	TIRAR DO GIT; DADOS_FORA_DO_GIT/; SÓ O DERIVADO ANONIMIZADO
Arquivo acima de 100 MiB	Push recusado pelo GitHub	Vídeo, banco de dados ou modelo commitado	TIRAR DO HISTÓRICO; REPOSITÓRIO DE DADOS OU GIT LFS
Commit gigante «tudo»	Um commit «atualizações» com 40 arquivos	Semanas sem commit	COMMIT POR SESSÃO E POR ASSUNTO; SALVAR_VERSAO.SH
Sem licença	Repositório público que ninguém pode reutilizar; Zenodo sem DOI	Achou que público bastava	LICENSE DE CHOOSEALICENSE.COM
Tag sem mensagem	Tag leve, sem autor, data nem descrição	`git tag v1` sem `-a -m`	GIT TAG -A VX -M "…"; PUSH --FOLLOW-TAGS
Force push	`git push --force` reescreve o remoto; o commit do colega some	Tentativa de «consertar» o histórico	NUNCA EM BRANCH COMPARTILHADO; REVERT EM VEZ DE RESET
`.bib` ou PDF de bases pagas	Exportação completa da Web of Science, PDFs de artigos, no repositório público	Dados licenciados tratados como «meus»	FORA DO GIT; SÓ A ESTRATÉGIA DE BUSCA E OS RESULTADOS AGREGADOS
Nome e e-mail errados nos commits	«usuario@notebook» como autor no GitHub	`user.name` / `user.email` não configurados	GIT CONFIG --GLOBAL (SLIDE «INSTALAR E CONFIGURAR»)

ERRO	COMO APARECE	CAUSA	CORREÇÃO
Marcador de conflito no arquivo	O LaTeX não compila; `<<<<<<<` no meio do texto	Commit feito antes de resolver o conflito	RESOLVER, APAGAR OS MARCADORES, ADD, COMMIT
Release sem CITATION.cff	Zenodo com metadados incompletos; sem «Cite this repository»	Arquivo criado depois da release	CITATION.CFF ANTES; NOVA RELEASE COM O DOI NO ARQUIVO
Backup «só nesta máquina»	AVISO do checar_repo.py; notebook roubado = projeto perdido	Nunca fez push	REMOTO E GIT PUSH AO FIM DE CADA SESSÃO

Checklist final: antes de publicar o repositório e pedir o DOI

Marque conforme concluir. O estado fica salvo neste navegador — não é compartilhado nem enviado a lugar nenhum.

- [ ] **FORA DO GIT** — Dados pessoais, arquivos licenciados, segredos e binários pesados em `dados_FORA_DO_GIT/` ou no `.gitignore`; nenhum arquivo acima de 100 MiB
- [ ] **COMMITS** — Pequenos, por sessão e por assunto, com mensagem que diga o que mudou; `git status` antes de cada um
- [ ] **GIT** — Instalado (`git --version`); `user.name` e `user.email` configurados com o nome que assina os artigos
- [ ] **REPOSITÓRIO** — Um por projeto, criado com `git init -b main`; primeiro commit com README e `.gitignore`
- [ ] **AUDITORIA** — `checar_repo.py` com 0 bloqueios antes de cada push; se um segredo entrou, a credencial foi trocada
- [ ] **README** — O que faz, por que é útil, como reproduzir, onde estão os dados (e por que não estão aqui), quem mantém
- [ ] **TAGS** — Uma tag anotada (`-a -m`) por marco: qualificação, submissão, defesa
- [ ] **REMOTO** — Repositório no GitHub (ou GitLab); `git push --follow-tags` ao fim de cada sessão; `checar_repo.py` sem o aviso de remoto
- [ ] **RELEASE E DOI** — Repositório público com licença, ligado ao Zenodo («Ativado»); release publicada; DOI copiado para o `CITATION.cff`, o README e o texto
- [ ] **IA DECLARADA** — Uso do assistente no repositório (README ou arquivo próprio) e no texto, conforme o manual de Uso de IA
- [ ] **LICENÇA** — Arquivo `LICENSE` na raiz (MIT, GPLv3 ou Apache 2.0 para código; CC BY 4.0
- [ ] **CITAÇÃO** — `CITATION.cff` 1.2.0 na raiz (cffinit ou `citacao_cff.py`); botão «Cite this

		para dados e texto), escolhida com o programa	repository» visível; referência ABNT conferida com a biblioteca

0 de 12

limpar

Glossário

Tudo que aparece no manual sem ser explicado no próprio slide.

Git

Repositório
A pasta do projeto com a pasta oculta `.git/`, onde o Git guarda todo o histórico; `git init` o cria.

Commit
Uma versão registrada do projeto: autor, data, mensagem e o conteúdo dos arquivos; identificado por um código como `3b95655`.

Staging area (área de preparação)
O que o próximo commit vai incluir; `git add` põe, `git restore --staged` tira.

HEAD
O commit em que você está agora — normalmente o último do branch atual; aparece em `git log --decorate`.

Branch
Linha de desenvolvimento paralela; `main` é a principal; `git switch -c` cria outra.

Merge
Combinar os commits de um branch em outro; «fast-forward» quando o destino só precisa avançar.

Fast-forward
Merge sem commit novo: o branch de destino não mudou, então só avança até o commit do outro.

Conflito
Duas versões mudaram a mesma linha; o Git marca o trecho com `<<<<<<<`, `=======` e `>>>>>>>` e espera uma pessoa escolher.

Tag
Nome fixo dado a um commit (`v0.1-qualificacao`); a tag anotada (`-a -m`) guarda autor, data e mensagem.

Remoto
Cópia do repositório num servidor (GitHub, GitLab); o primeiro chama-se `origin` por convenção.

Push · pull · clone
`push` envia commits ao remoto; `pull` traz os do remoto; `clone` cria um repositório local a partir de um remoto.

Bare repository
Repositório só com o histórico, sem arquivos de trabalho — é o que um servidor guarda; o «remoto» da demonstração (Figura 14) é um bare local.

.gitignore
Arquivo na raiz com os padrões do que o Git não deve rastrear; modelos em github.com/github/gitignore.

Git LFS

GitHub, publicação e citação

Fork
Cópia de um repositório de outra pessoa na sua conta, para propor mudanças por pull request.

Pull request
Pedido para incorporar os commits de um branch a outro, com revisão e discussão; «solicitação de pull» na interface em português.

Issue
Item de acompanhamento no repositório: tarefa, dúvida, erro; numerada, com comentários e responsável.

CI / GitHub Actions
Integração contínua: rodar conferências e testes a cada push; no GitHub, por arquivos YAML em `.github/workflows/`.

GitHub Pages
Hospedagem gratuita de site estático a partir de um repositório público (`https://<owner>.github.io/<repo>`).

Codespaces
Ambiente de desenvolvimento na nuvem, pelo navegador ou VS Code; 120 horas-núcleo e 15 GB por mês no Free.

GitHub CLI (gh)
O GitHub no terminal: repositórios, issues, pull requests, releases; v2.100.0.

Release
Uma tag publicada no GitHub com título, notas e anexos; traz ZIP e tar.gz do código; é o que o Zenodo arquiva.

README
Arquivo Markdown na raiz que o GitHub renderiza na página do repositório: o que é, como usar, como citar.

Licença
Os termos de reutilização (MIT, GPLv3, Apache 2.0, CC BY 4.0) num arquivo `LICENSE`; sem ela ninguém pode reutilizar legalmente.

CITATION.cff
Arquivo no Citation File Format (1.2.0) que diz como citar o repositório; ativa o botão «Cite this repository».

DOI
Digital Object Identifier: identificador permanente; para código, emitido pelo Zenodo a cada release.

Zenodo
Repositório aberto que arquiva releases do GitHub e emite DOIs; lê o `CITATION.cff` para os metadados.

OSF

Large File Storage: guarda arquivos grandes fora do histórico e deixa um ponteiro de texto no Git; 10 GiB no GitHub Free.

Open Science Framework: plataforma de projetos e pré-registro; o add-on GitHub mostra o repositório na seção «Files».

Overleaf

Editor LaTeX on-line; a integração Git e o GitHub Synchronization são recursos premium.

Referências (1 de 2): a evidência, os guias e os livros

1. RAM, K. Git can facilitate greater reproducibility and increased transparency in science. Source Code for Biology and Medicine, v. 8, n. 7, 2013. doi.org/10.1186/1751-0473-8-7

2. PEREZ-RIVEROL, Y.; GATTO, L.; WANG, R.; SACHSENBERG, T. et al. Ten Simple Rules for Taking Advantage of Git and GitHub. PLOS Computational Biology, v. 12, n. 7, e1004947, 2016. doi.org/10.1371/journal.pcbi.1004947

3. BLISCHAK, J. D.; DAVENPORT, E. R.; WILSON, G. A Quick Introduction to Version Control with Git and GitHub. PLOS Computational Biology, v. 12, n. 1, e1004668, 2016. doi.org/10.1371/journal.pcbi.1004668

4. WILSON, G.; BRYAN, J.; CRANSTON, K.; KITZES, J. et al. Good enough practices in scientific computing. PLOS Computational Biology, v. 13, n. 6, e1005510, 2017. doi.org/10.1371/journal.pcbi.1005510

5. THE TURING WAY COMMUNITY; SCRIBERIA. The Turing Way: A handbook for reproducible, ethical and collaborative research. Zenodo, 2024. CC BY 4.0. Capítulo «Version Control». doi.org/10.5281/zenodo.3332807 book.the-turing-way.org

6. SOFTWARE CARPENTRY. Version Control with Git. 14 episódios. CC BY 4.0. Acesso em 9 set. 2026. swcarpentry.github.io/git-novice

7. CHACON, S.; STRAUB, B. Pro Git. 2. ed. Apress, 2014. Tradução completa para o português do Brasil; CC BY-NC-SA 3.0. git-scm.com/book/pt-br/v2

8. GIT. About; Downloads (versão 2.55.0). git-scm.com. Acesso em 9 set. 2026. git-scm.com/about git-scm.com/install

9. GIT LFS. Git Large File Storage (v3.8.0, 28 ago. 2026). Acesso em 9 set. 2026. git-lfs.com

Referências (2 de 2): documentação do GitHub, licenças, Citation File Format, Zenodo, Overleaf, OSF, GitLab e ferramentas

1. GITHUB. GitHub Docs (pt): «Olá, Mundo»; «Criar um repositório»; «Ignorar arquivos»; «Sobre arquivos grandes no GitHub»; «Sobre o arquivo README do repositório»; «Gerenciar versões em repositórios»; «Sobre os arquivos de CITATION»; «Referenciar e citar conteúdo»; «O que é o GitHub Pages?»; «Solicitações de pull»; «Sobre problemas». Acesso em 9 set. 2026. docs.github.com/pt

2. GITHUB. Pricing; Student Developer Pack; GitHub Desktop (3.6.5); GitHub CLI (v2.100.0); Codespaces; gitignore templates. Acesso em 9 set. 2026. github.com/pricing education.github.com/pack github.com/apps/desktop cli.github.com github.com/github/gitignore

3. GITHUB. Choose an open source license. CC BY. Acesso em 9 set. 2026. choosealicense.com

4. CITATION FILE FORMAT. Citation File Format (CFF), versão 1.2.0; cffinit; cff-converter-python. Acesso em 9 set. 2026. citation-file-format.github.io

5. ZENODO. Enable a repository (GitHub integration). help.zenodo.org. Acesso em 9 set. 2026. help.zenodo.org zenodo.org

6. OVERLEAF. Git integration and GitHub synchronization. docs.overleaf.com. Acesso em 9 set. 2026. docs.overleaf.com

7. CENTER FOR OPEN SCIENCE. Connect GitHub to a project. help.osf.io. Acesso em 9 set. 2026. help.osf.io

8. GITLAB. Preços (pt). Acesso em 9 set. 2026. about.gitlab.com/pricing

9. MICROSOFT. Source control in VS Code. code.visualstudio.com. Acesso em 9 set. 2026. code.visualstudio.com/docs/sourcecontrol/overview

> SOBRE AS DATAS
>
> Versões (Git 2.55.0, Git LFS v3.8.0, GitHub Desktop 3.6.5, GitHub CLI v2.100.0, CFF 1.2.0), preços em dólar, limites e nomes de botão foram lidos nas páginas acima em 09/09/2026 e mudam sem aviso. Não há preço em reais, número de usuários do GitHub nem limite de tamanho do Zenodo neste manual porque não foram conferidos nessa data.

Como citar este manual, declaração de uso de IA e licença

COMO CITAR

MATIAS, J. P. M. *Git e GitHub para pesquisadores: passo a passo.* MirandasTech, v1.0, set. 2026. CC BY 4.0.

Este manual: git.mirandastech.com.br (PDF em /manual.pdf). Manuais irmãos: prisma.mirandastech.com.br · busca.mirandastech.com.br · latex.mirandastech.com.br · zotero.mirandastech.com.br · metodologia.mirandastech.com.br · notebook.mirandastech.com.br · manual.mirandastech.com.br · plugin.mirandastech.com.br · hub: mirandastech.com.br/pesquisa

LICENÇA

DECLARAÇÃO DE USO DE IA

Este manual foi escrito com assistência de IA (Claude, Anthropic) na estruturação, na redação, na automação das capturas de tela e na execução dos scripts, sob as regras do manual de Uso de IA na pesquisa científica. Todos os fatos, números, nomes de botões, preços, limites, versões e referências foram conferidos nas fontes indicadas em 09/09/2026. Nenhuma captura foi feita com login; nenhum repositório foi criado na conta do autor para a demonstração — o «remoto» das figuras é um repositório *bare* local. Decisões de conteúdo, seleção das fontes e responsabilidade pelo texto são do autor.

Sem garantia: interfaces, planos, preços, limites e versões mudam; cada fato traz a data em que foi verificado. Quando a sua tela divergir, vale a tela. A escolha da licença, o que pode ou não ser publicado (dados de participantes, materiais licenciados, resultados sob embargo), a citação do software no seu texto e a declaração de uso de IA da sua pesquisa são seus e seguem o seu programa, o seu orientador, a biblioteca da sua instituição e, quando houver, o comitê de ética.