

Git & GitHub

Introdução Engenharia Informática

Mário Antunes

27 de Outubro de 2025

Exercícios

Laboratório Prático: Git & GitHub

De Repositório Local a Colaboração Open-Source

Objetivo: Este laboratório irá guiá-lo através do ciclo de vida completo de um repositório Git. Irá aprender a criar um repositório local, gerir versões, trabalhar com branches, e finalmente, colaborar num projeto remoto usando o GitHub.

Pré-requisitos:

- **Git Instalado:** Tem de ter o Git instalado na sua máquina.
 - **Uma Conta GitHub:** Irá precisar de uma conta GitHub gratuita para os exercícios de colaboração.
 - **Um Editor de Texto:** Qualquer editor de texto (como VS Code, Sublime Text, ou Nano) servirá.
-

Parte 0: Configuração & Autenticação

Antes de podermos trabalhar com repositórios remotos, precisamos de instalar o Git e dizer ao GitHub quem somos.

Passo 1: Instalar o Git

Em sistemas baseados em Debian (como debian trixieOS ou Ubuntu), pode instalar o Git usando o apt.

1. Primeiro, atualize a sua lista de pacotes:

```
$ sudo apt update
```

2. De seguida, instale o Git:

```
$ sudo apt install git
```

Passo 2: Configurar a Sua Identidade

Tem de dizer ao Git o seu nome e email. Esta informação será incorporada em cada commit que fizer.

```
$ git config --global user.name "Your Name"  
$ git config --global user.email "your.email@ua.pt"
```

(Use o seu email da UA.)

Passo 3: Autenticar no GitHub

Para fazer push do seu código para o GitHub, tem de provar quem é. Tem duas opções principais: SSH (recomendado) ou um Personal Access Token (PAT).

Método 1: Usar uma Chave SSH (Recomendado)

Este método é mais seguro e conveniente. Adiciona uma “chave” (key) à sua conta GitHub, e o seu computador usa-a para autenticar automaticamente.

1. Gere uma nova chave SSH ed25519. Este comando cria um par de chaves sem pedir uma password (`-N ""`).

```
$ ssh-keygen -t ed25519 -C "your.email@example.com" -f ~/.ssh/id_ed25519 -N ""
```

2. Mostre a sua nova chave pública (public key) no terminal para a poder copiar.

```
$ cat ~/.ssh/id_ed25519.pub
```

3. Copie o output completo (começando com `ssh-ed25519 ...` e terminando com o seu email).

4. Adicione a chave ao GitHub:

- Vá a **GitHub.com** e clique no ícone do seu perfil no canto superior direito.
- Vá a **Settings** -> **SSH and GPG keys** (na barra lateral “Access”).
- Clique em **New SSH key**.
- Dê-lhe um **Title** (título) (ex: “Portátil TrixieOS”).
- Cole a chave copiada na caixa **Key**.
- Clique em **Add SSH key**.

Método 2: Usar um Fine-Grained Personal Access Token (PAT)

Um PAT é como uma password que pode usar para operações Git.

1. Vá a **GitHub.com** -> **Settings** -> **Developer settings** (no fundo da barra lateral).
2. Vá a **Personal access tokens** -> **Fine-grained tokens**.
3. Clique em **Generate new token**.
4. Defina as seguintes opções:
 - **Token name**: Dê-lhe um nome descriptivo (ex: “Token Aula IEI”).
 - **Expiration**: Selecione **No expiration** (Sem expiração).
 - **Repository access**: Selecione **All repositories** (Todos os repositórios).
 - **Permissions**: Faça scroll até “Repository permissions” e encontre **Contents**. Altere o seu acesso para **Read and write** (Leitura e escrita).
5. Clique em **Generate token**.
6. **IMPORTANTE**: Copie o token (começa por `github_pat_...`) *imediatamente*. Nunca mais o verá depois de sair desta página.
7. Quando precisar de se ligar ao GitHub (nas Partes 2 e 3), irá usar este token no URL: `https://<YOUR_USERNAME>:<YOUR_TOKEN>@github.com/<YOUR_USERNAME>/<REPOSITORY>.git`

Parte 1: O Seu Re却tório Local

Exercício 1: git init (Criar um Re却tório) O nosso primeiro passo é dizer ao Git para começar a seguir (track) um novo projeto.

1. Crie uma nova pasta para o seu projeto e navegue para dentro dela.

```
$ mkdir my-git-project  
$ cd my-git-project
```

2. Agora, inicialize-o como um re却tório Git.

```
$ git init
```

3. Isto cria uma pasta oculta `.git`. Criou oficialmente um re却tório!

Exercício 2: O Ciclo Principal (add, commit, status, log) Vamos criar um ficheiro, adicioná-lo à “staging area” e fazer “commit” para o nosso histórico.

1. Crie um ficheiro chamado `index.html` dentro da sua pasta `my-git-project` e adicione o seguinte conteúdo:

```
<h1>Welcome to My Project</h1>
```

2. Verifique o “status” do seu re却atorio.

```
$ git status
```

O Git irá mostrar-lhe `index.html` como um “untracked file” (ficheiro não seguido).

3. Diga ao Git que quer seguir este ficheiro, adicionando-o à **Staging Area**.

```
$ git add index.html
```

4. Verifique o status novamente. O ficheiro está agora “staged” (em staging) e pronto para o `commit`.

```
$ git status
```

5. Agora, guarde esta “snapshot” (fotografia) no seu histórico com um **commit**.

```
$ git commit -m "Initial commit: Add homepage"
```

6. Finalmente, veja o histórico (log).

```
$ git log
```

Exercício 3: Corrigir um Commit Mau (--amend) Boas mensagens de `commit` são vitais. Vamos corrigir uma má.

1. Faça uma pequena alteração ao `index.html`. Por exemplo, adicione um parágrafo:

```
<h1>Welcome to My Project</h1>
<p>This is a project for my IEI class.</p>
```

2. Faça `commit` desta alteração com uma mensagem **má**. A flag `-a` é um atalho para `git add` (para ficheiros seguidos) e `git commit`.

```
$ git commit -a -m "fix stuff"
```

3. Verifique o seu log: `git log --oneline`. Verá a sua mensagem “fix stuff”. Vamos corrigi-la.

4. Execute o comando **amend**. Isto irá substituir o seu *previous commit* anterior por um novo.

```
$ git commit --amend -m "Doc: Update homepage text"
```

5. Verifique o seu log novamente: `git log --oneline`. O commit “fix stuff” desapareceu, substituído pela sua mensagem melhor.

Exercício 4: Ignorar Ficheiros (.gitignore) Nunca queremos fazer `commit` de chaves secretas ou ficheiros temporários.

1. Crie um ficheiro chamado `.env` e adicione um “segredo” a ele.

```
$ echo "DATABASE_PASSWORD=12345" > .env
```

2. Execute `git status`. Verá que o Git quer adicionar o `.env`. Nós não queremos isto.

3. Crie um ficheiro chamado `.gitignore` (sim, começa com um ponto).

4. Adicione a seguinte linha inside `.gitignore`:

```
.env
```

5. Execute `git status` novamente. O ficheiro `.env` desapareceu da lista, mas o Git agora quer seguir o ficheiro `.gitignore`, que é exatamente o que queremos.

6. Adicione e faça `commit` do ficheiro `.gitignore`.

```
$ git add .gitignore
$ git commit -m "Feat: Add .gitignore to ignore environment files"
```

Exercício 5: Ramificações (branch, checkout) Vamos trabalhar numa nova funcionalidade (feature) isoladamente, sem estragar o nosso código principal.

1. Crie um novo `branch` para uma nova página “about”.

```
$ git branch feature/about-page
```

2. Mude para o seu novo `branch`.

```
$ git checkout feature/about-page
```

(**Atalho:** git checkout -b <nome-do-branch> cria e muda num só comando.)

- Crie um ficheiro about.html com este conteúdo:

```
<h1>About Us</h1>  
<p>This is the about page.</p>
```

- Adicione e faça commit deste novo ficheiro no seu feature branch.

```
$ git add about.html  
$ git commit -m "Feat: Add new about page"
```

- Agora, volte para o seu branch main e veja os seus ficheiros.

```
$ git checkout main  
$ ls
```

O ficheiro about.html desapareceu! Isto acontece porque ele apenas existe no feature branch.

Exercício 6: Juntar (merge) A sua funcionalidade “about page” está completa. Vamos fazer merge dela para o branch main.

- Certifique-se de que está no branch que quer receber as alterações (ou seja, main).

```
$ git checkout main
```

- Execute o comando merge para puxar as alterações do seu feature branch.

```
$ git merge feature/about-page
```

- Verifique os seus ficheiros com ls. O ficheiro about.html está agora presente no main.

- Veja o seu histórico para ver o merge commit.

```
$ git log --oneline --graph
```

Exercício 7: Resolver Conflitos de Merge O que acontece quando dois branches editam a mesma linha?

- A partir do seu branch main, crie um novo branch.

```
$ git checkout -b change-title-A
```

- Neste branch change-title-A, edite o index.html para dizer:

```
<h1>Welcome to the IEI Project</h1>
```

- Faça commit desta alteração.

```
$ git commit -a -m "Update title on branch A"
```

- Agora, volte ao main e crie uma alteração conflituante.

```
$ git checkout main  
$ git checkout -b change-title-B
```

- Neste branch change-title-B, edite a mesma linha no index.html para dizer:

```
<h1>Welcome to the TIA Project</h1>
```

- Faça commit desta alteração: git commit -a -m "Update title on branch B"

- Agora, vamos tentar fazer merge do change-title-B para o change-title-A.

```
$ git checkout change-title-A  
$ git merge change-title-B
```

CONFLITO (CONFLICT)! O Git irá parar e dizer-lhe que há um conflito no index.html.

- Corrija-o:** Abra o index.html. Verá os marcadores de conflito (<<<<, ==, >>>>). Edite o ficheiro para ficar correto (ex: apague os marcadores e escolha um título, ou escreva um novo).

- Finalize:** Assim que estiver corrigido, faça add do ficheiro e commit.

```
$ git add index.html  
$ git commit -m "Merge: Resolve title conflict"
```

Parte 2: GitHub - Colaboração

Exercício 8: clone, remote, & origin Vamos ligar o nosso repositório local a um repositório remoto no GitHub.

1. Vá a [GitHub.com](#). Crie um **repositório (repository) novo, vazio e público**. Dê-lhe o nome `git-practice-repo`.
2. **NÃO** o initialize com um README. Queremos que esteja vazio.
3. O GitHub irá mostrar-lhe URLs. Encontre o botão **Code**.
 - **Se configurou uma Chave SSH:** Selecione o separador (tab) **SSH** e copie o URL (ex: `git@github.com:<0_SEU_USERNAME>/git-practice-repo.git`).
 - **Se criou um PAT:** Selecione o separador **HTTPS** e copie o URL (ex: `https://github.com/<0_SEU_USERNAME>/git-practice-repo.git`).
4. No seu terminal local, volte para a sua pasta `my-git-project`.
5. Adicione este novo repositório GitHub como o seu “remote” chamado “origin”, usando o URL que corresponde ao seu método de autenticação.
 - **Se estiver a usar SSH:**
`$ git remote add origin <PASTE_0_SEU_SSH_URL_AQUI>`
 - **Se estiver a usar PAT:** Use o formato de URL especial da Parte 0, substituindo os placeholders.
`$ git remote add origin https://<0_SEU_USERNAME>:<0_SEU_TOKEN>@github.com/<0_SEU_USERNAME>/git-practice-repo.git`
6. Verifique se o `remote` foi adicionado.
`$ git remote -v`

Exercício 9: push (Enviar o Seu Trabalho) O seu repositório local tem histórico, mas o remoto está vazio. Vamos fazer push do seu trabalho.

1. Primeiro, vamos renomear o nosso branch local `master` para `main` para corresponder ao padrão do GitHub.
`$ git branch -M main`
2. Agora, faça **push** do seu branch local `main` para o `remote origin`. A flag `-u` define-o como o padrão, para que possa usar apenas `git push` no futuro.
`$ git push -u origin main`
3. Atualize a sua página do repositório GitHub. Todos os seus ficheiros (`index.html`, `about.html`, `.gitignore`) e o seu histórico de commits estão agora online!

Exercício 10: tag & release (Marcar uma Versão) O seu projeto está num ponto estável. Vamos marcá-lo (tag) como versão 1.0.

1. Crie uma “tag” que aponte para o seu último commit.
`$ git tag -a v1.0.0 -m "First stable release"`
2. Faça push da sua nova tag para o GitHub (as tags não são enviadas automaticamente).
`$ git push origin v1.0.0`
3. **No GitHub:** Vá à página principal do seu repositório. Encontre “Releases” no lado direito. Clique em “Create a new release” (ou “Draft a new release”).
4. Selecione a sua tag `v1.0.0`, dê-lhe um título como “Version 1.0.0”, e escreva uma breve descrição. Clique em “Publish release”. Agora tem uma release oficial!

Parte 3: O Fluxo de Trabalho Open-Source Completo

Exercício 11: fork (Contribuir para um Projeto) Irá agora contribuir para um projeto que não lhe pertence.

1. Vá a este repositório no GitHub (o repositório tictactoe da última aula): <https://github.com/mariolpantunes/tictactoe>
2. No canto superior direito, clique no botão “**Fork**”. Isto irá criar uma cópia do repositório na sua própria conta GitHub.
3. Agora, na página do seu fork no GitHub, clique no botão verde “<> Code”.
 - **Se configurou uma Chave SSH:** Selecione o separador **SSH** e copie o URL (ex: git@github.com:<0_SEU_USERTOKEN>/tictactoe.git)
 - **Se criou um PAT:** Selecione o separador **HTTPS** e copie o URL (ex: https://github.com/<0_SEU_USERNAME>/tictactoe)
4. No seu terminal (fora da pasta do seu projeto anterior), faça **clone** do seu fork usando o comando correto para o seu método de autenticação.
 - **Se estiver a usar SSH:**
\$ git clone <PASTE_0_SEU_SSH_URL_AQUI>
\$ cd tictactoe
 - **Se estiver a usar PAT:**
\$ git clone https://<0_SEU_USERNAME>:<0_SEU_TOKEN>@github.com/<0_SEU_USERNAME>/tictactoe

Exercício 12: O Pull Request (pull request) Vamos fazer uma alteração e propô-la ao projeto original.

1. Crie um novo branch para a sua alteração.
\$ git checkout -b add-my-name
2. Edite o ficheiro CONTRIBUTORS.md e adicione o seu nome à lista.
3. Adicione e faça commit da sua alteração.
\$ git add CONTRIBUTORS.md
\$ git commit -m "Add [Your Name] to contributors list"
4. Faça push deste novo branch *para o seu fork (origin)*.
\$ git push origin add-my-name
5. **Vá ao GitHub:** Vá à página do seu fork. Deverá ver uma faixa verde a dizer “This branch is 1 commit ahead...” Clique no botão “**Contribute**” e depois em “**Open a pull request**”.
6. Reveja as alterações, adicione uma mensagem simpática, e clique em “**Create pull request**”.
7. **Parabéns!** Acabou de fazer um pull request, o coração da colaboração open-source.

Desafio Bónus: rebase (Limpar o Histórico)

Vamos refazer o Exercício 6, mas com rebase para um histórico mais limpo.

1. Volte a um estado anterior ao merge. Uma boa forma é fazer **reset** ao **main**.
\$ cd ~/my-git-project # Volte para o seu primeiro projeto
\$ git checkout main
\$ git reset --hard HEAD~1 # Isto rebobina o 'main' um commit (apaga o merge)
2. Você ainda tem o seu branch **feature/about-page**. Vamos fazer outro commit no **main** para criar uma divergência.
\$ echo "" >> index.html
\$ git commit -a -m "Add a comment to homepage"
3. Agora, o **main** tem um commit que o **feature/about-page** não tem.
4. Mude para o seu **feature** branch e use **rebase** para “reaplicar” os commits do seu branch *por cima* do novo **main**.

```
$ git checkout feature/about-page  
$ git rebase main
```

5. Agora, volte ao main e faça merge.

```
$ git checkout main  
$ git merge feature/about-page
```

6. Irá dizer “Fast-forward”. Veja o seu log (`git log --oneline --graph`). O histórico está perfeitamente linear e limpo!