

# Git & Github

## Tópicos de Informática para Automação

Mário Antunes

27 de Outubro de 2025

## Exercícios

### Laboratório Prático: Git & GitHub

#### Do Repositório Local à 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 1: O Seu Repositório Local

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

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

```
$ mkdir o-meu-projeto-git  
$ cd o-meu-projeto-git
```

2. Agora, inicialize-a como um repositório Git.

```
$ git init
```

3. Isto cria uma pasta oculta `.git`. Criou oficialmente um repositório!

**Exercício 2: O Ciclo Central (add, commit, status, log)** Vamos criar um ficheiro, prepará-lo ("stage") e guardá-lo ("commit") no nosso histórico.

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

```
<h1>Bem-vindo ao Meu Projeto</h1>
```

2. Verifique o "estado" (status) do seu repositório.

```
$ 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 estado novamente. O ficheiro está agora "staged" (em preparação) e pronto para o commit.

```
$ git status
```

5. Agora, guarde este "snapshot" no seu histórico com um **commit**.

```
$ git commit -m "Commit inicial: Adiciona homepage"
```

6. Finalmente, veja o log (registo) do histórico.

```
$ 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>Bem-vindo ao Meu Projeto</h1>
<p>Este é um projeto para a minha cadeira de IEI.</p>
```

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

```
$ git commit -a -m "corrigir coisas"
```

3. Veja o seu log: `git log --oneline`. Verá a sua mensagem "corrigir coisas". Vamos corrigi-la.

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

```
$ git commit --amend -m "Doc: Atualiza texto da homepage"
```

5. Veja o seu log novamente: `git log --oneline`. O commit "corrigir coisas" 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-lhe um "segredo".

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

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

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

4. Adicione a seguinte linha dentro do `.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: Adiciona .gitignore para ignorar ficheiros de ambiente"
```

**Exercício 5: Criar Ramificações (Branching) (branch, checkout)** Vamos trabalhar numa nova funcionalidade isolados, sem estragar o nosso código principal.

1. Crie um novo branch para uma nova página "sobre".

```
$ git branch feature/pagina-sobre
```

2. Mude para o seu novo branch.

```
$ git checkout feature/pagina-sobre
```

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

3. Crie um ficheiro `sobre.html` com este conteúdo:

```
<h1>Sobre Nós</h1>
<p>Esta é a página sobre.</p>
```

4. Adicione e faça commit deste novo ficheiro *no seu branch de funcionalidade*.

```
$ git add sobre.html
$ git commit -m "Feat: Adiciona nova página sobre"
```

5. Agora, volte para o seu branch principal e veja os seus ficheiros.

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

O ficheiro sobre.html desapareceu! Isto acontece porque ele apenas existe no branch feature/pagina-sobre.

**Exercício 6: Fazer Merge (merge)** A sua funcionalidade “página sobre” está completa. Vamos fazer merge dela de volta para o branch main.

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

```
$ git checkout main
```
2. Execute o comando merge para “puxar” as alterações do seu branch de funcionalidade.  

```
$ git merge feature/pagina-sobre
```
3. Verifique os seus ficheiros com ls. O ficheiro sobre.html está agora presente no main.
4. 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?

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

```
$ git checkout -b alterar-titulo-A
```
  2. Neste branch alterar-titulo-A, edite o index.html para dizer:  

```
<h1>Bem-vindo ao Projeto IEI</h1>
```
  3. Faça commit desta alteração.  

```
$ git commit -a -m "Atualiza título no branch A"
```
  4. Agora, volte ao main e crie uma alteração *conflituante*.  

```
$ git checkout main
$ git checkout -b alterar-titulo-B
```
  5. Neste branch alterar-titulo-B, edite a *mesma linha* no index.html para dizer:  

```
<h1>Bem-vindo ao Projeto TIA</h1>
```
  6. Faça commit desta alteração: 

```
git commit -a -m "Atualiza título no branch B"
```
  7. Agora, vamos tentar fazer merge do alterar-titulo-B para o alterar-titulo-A.  

```
$ git checkout alterar-titulo-A
$ git merge alterar-titulo-B
```

**CONFLITO!** O Git irá parar e dizer-lhe que há um conflito no index.html.
  8. **Corrija:** Abra o index.html. Verá os marcadores de conflito (<<<<<, =====, >>>>>). Edite o ficheiro para ficar correto (p. ex., apague os marcadores e escolha um título, ou escreva um novo).
  9. **Finalize:** Assim que estiver corrigido, faça add do ficheiro e commit.  

```
$ git add index.html
$ git commit -m "Merge: Resolve conflito de título"
```
-

## Parte 2: GitHub - Colaboração

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

1. Vá ao **GitHub.com**. Crie um **repositório público, novo e vazio**. Dê-lhe o nome `git-practice-repo`.
2. **NÃO** o inicialize com um README. Queremos que esteja vazio.
3. O GitHub irá mostrar-lhe um URL. Copie o URL HTTPS.
4. No seu terminal local, volte para a pasta `o-meu-projeto-git`.
5. Adicione este novo repositório GitHub como o seu "remote" chamado "origin".  

```
$ git remote add origin <COLE_O_SEU_URL_DO_GITHUB_AQUI>
```
6. Verifique que 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 enviar o 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 `main` local para o `origin` remoto. A flag `-u` define-o como o padrão, para que no futuro possa usar apenas `git push`.  

```
$ git push -u origin main
```
3. Atualize a página do seu repositório GitHub. Todos os seus ficheiros (`index.html`, `sobre.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 como versão 1.0.

1. Crie uma "tag" que aponte para o seu último commit.  

```
$ git tag -a v1.0.0 -m "Primeiro lançamento estável"
```
  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 "Versão 1.0.0", e escreva uma breve descrição. Clique em "Publish release". Agora tem um lançamento (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 aula passada): <https://github.com/marjolpantunes/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 GitHub do **seu** fork, clique no botão verde "<> Code" e copie o URL HTTPS.
4. No seu terminal (fora da pasta do seu projeto antigo), faça **clone** do *seu* fork.

```
$ git clone <COLE_O_URL_DO_SEU_FORK_AQUI>
$ cd 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 adicionar-o-meu-nome
```

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 "Adiciona [O Seu Nome] à lista de contribuidores"
```

4. Faça push deste novo branch *para o seu fork* (origin).

```
$ git push origin adicionar-o-meu-nome
```

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 Extra: 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 ~/o-meu-projeto-git # Volte para o seu primeiro projeto
$ git checkout main
$ git reset --hard HEAD~1 # Isto rebobina o 'main' em um commit (apaga o merge)
```

2. Você ainda tem o seu branch feature/pagina-sobre. Vamos fazer outro commit no main para criar uma divergência.

```
$ echo "" >> index.html
$ git commit -a -m "Adiciona um comentário à homepage"
```

3. Agora, o main tem um commit que o feature/pagina-sobre não tem.

4. Mude para o seu branch de funcionalidade e use rebase para reaplicar os commits do seu branch *em cima* do novo main.

```
$ git checkout feature/pagina-sobre
$ git rebase main
```

5. Agora, volte ao main e faça merge.

```
$ git checkout main
$ git merge feature/pagina-sobre
```

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