

**Instituto Superior de Tecnologias Avançadas do Porto**  
**CTESP DS - Curso Técnico Superior Profissional de**  
**Desenvolvimento de Software**

**Ano letivo 2022/2023**  
**DAS - Desenvolvimento Ágil de Software**

# **Trabalho Prático Individual**

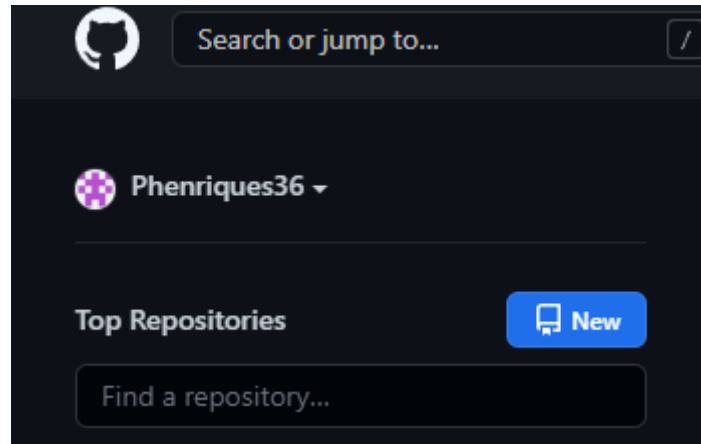
## **Final - DAS**



**Paulo Henriques**

## Ponto 1

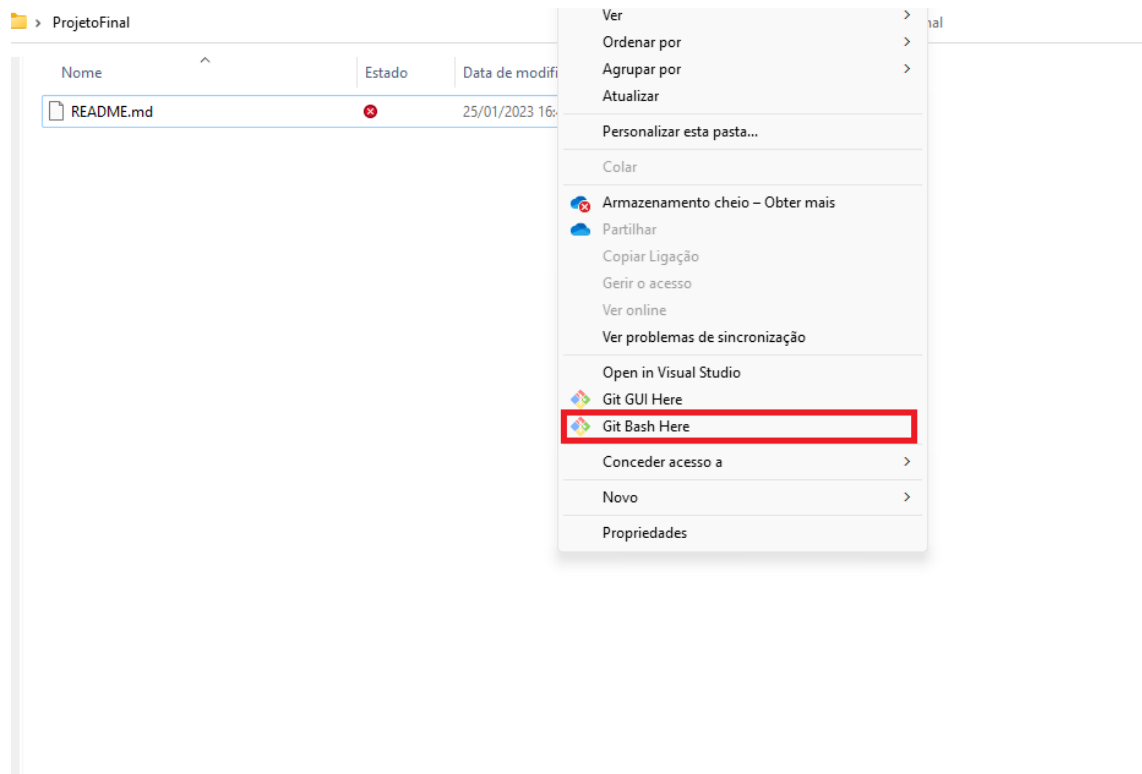
Primeiramente temos de criar um repositório no GitHub para isso depois de entrarmos com a nossa conta no GitHub devemos seleccionar a opção New



Depois vai aparecer este ecrã onde devemos preencher com as informações que queremos

A screenshot of the 'Create a new repository' form on GitHub. The form has a dark background. At the top, it says 'Create a new repository' and provides a brief explanation of what a repository is. Below this, there are two main sections: 'Owner' and 'Repository name'. The 'Owner' dropdown is set to 'Phenriques36'. The 'Repository name' field is empty. Below these fields, there's a note about repository names and a link to 'Import a repository'. The 'Description' field is optional and empty. There are two radio buttons for visibility: 'Public' (selected) and 'Private'. Below this, there's a section for 'Initialize this repository with:' which includes a checkbox for 'Add a README file'. There's also a section for 'Add .gitignore' with a dropdown menu set to 'None'. A 'Choose a license' section has a dropdown menu set to 'None'. At the bottom, there's a blue button labeled 'Create repository' and a note indicating that a public repository is being created in the user's personal account.

De seguida vamos abrir o git bash na pasta que queremos usar como repositório local da seguinte forma, neste caso vou utilizar uma pasta com o nome ProjetoFinal e seleccionar a opção marcada a vermelho



Depois do GIT Bash aberto devemos usar o seguinte código para ligar o repositório local ao GitHub

```
echo "# ProjetoFinal" >> README.md
git init
git add README.md
git commit -m "Add README"
git branch -M main
git remote add origin "https://github.com/Phenriques36/TrabalhoFinalGITDAS.git"(o link aqui é diferente para todos, este é apenas o link do meu repositório)
git push -u origin main
git flow init
```

## Ponto 2

Neste ponto é suposto definir níveis de acesso de uma organização, para isso devemos criar uma organização e definir os seguintes passos

The screenshot shows the GitHub organization overview page for 'TrabalhoFinalDAS'. The navigation bar includes links for Overview, Repositories, Projects, Packages, Teams, People, and Settings. The main content area features a welcome message and several task cards: 'Invite your first member', 'Customize members' permissions', 'Create a pull request', and 'Create a branch protection rule'. A sidebar on the right contains sections for 'You can now follow organization', 'You can create a README file', 'You can hide the tasks', 'Discussions', 'Repositories', and 'People'.

The screenshot shows the 'Member privileges' settings page for the organization. It includes sections for 'Base permissions', 'Repository creation', 'Repository forking', and 'Pages creation'. Each section has a 'Save' button and a 'Why is this option disabled?' link. The 'Base permissions' section shows a 'Write' dropdown. The 'Repository creation' section has checkboxes for 'Public' and 'Private'. The 'Repository forking' section has a checkbox for 'Allow forking of private repositories'. The 'Pages creation' section has checkboxes for 'Public' and 'Private'.

## Admin repository permissions

### Repository visibility change

☐ **Allow members to change repository visibilities for this organization**

If enabled, members with admin permissions for the repository will be able to change its visibility. If disabled, only organization owners can change repository visibilities.

Save

### Repository deletion and transfer

☒ **Allow members to delete or transfer repositories for this organization**

If enabled, members with admin permissions for the repository will be able to delete or transfer public and private repositories. If disabled, only organization owners can delete or transfer repositories.

Save

### Issue deletion

☐ **Allow repository administrators to delete issues for this organization**

If enabled, members with admin permissions for the repository will be able to delete issues. If disabled, only organization owners can delete issues. [Learn more.](#)

Save

## Member team permissions

### Team creation rules

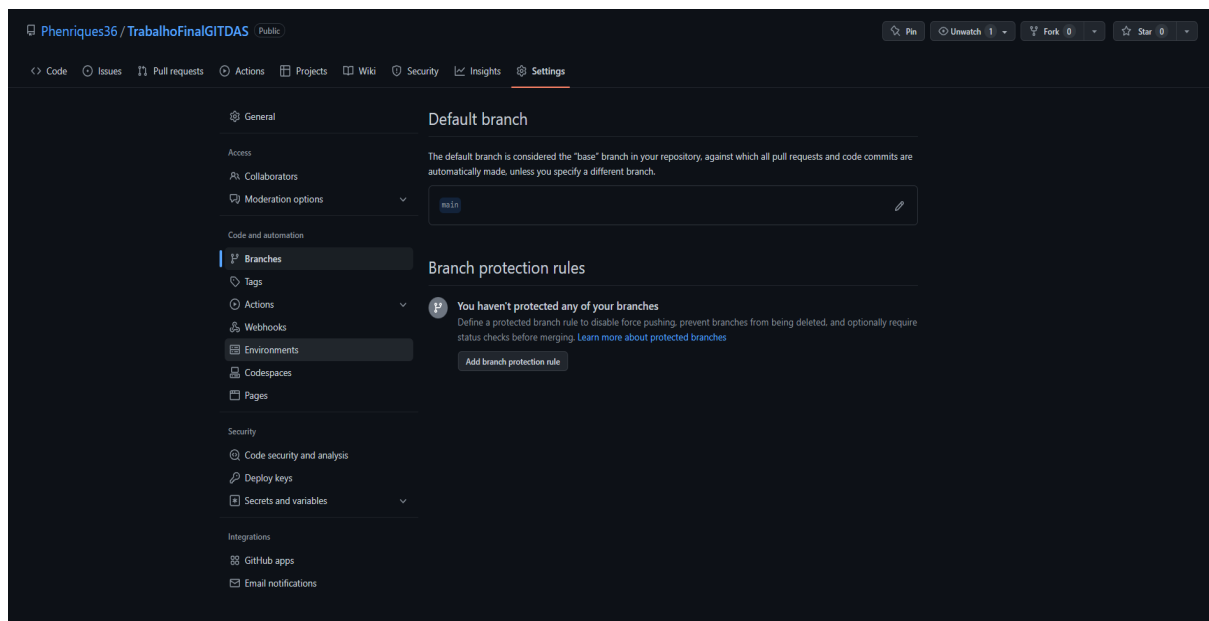
☒ **Allow members to create teams**

If enabled, any member of the organization will be able to create new teams. If disabled, only organization owners can create new teams.

Save

Depois disto feito as permissões estarão bem definidas permitindo a developers a submissão de código, mas nunca a alteração de visibilidade do repositório.

## Ponto 3



## Branch protection rule



### Protect your most important branches

[Branch protection rules](#) define whether collaborators can delete or force push to the branch and set requirements for any pushes to the branch, such as passing status checks or a linear commit history.

Your [GitHub Free plan](#) can only enforce rules on its public repositories, like this one.

#### Branch name pattern \*

#### Protect matching branches

☐ **Require a pull request before merging**

When enabled, all commits must be made to a non-protected branch and submitted via a pull request before they can be merged into a branch that matches this rule.

☐ **Require status checks to pass before merging**

Choose which [status checks](#) must pass before branches can be merged into a branch that matches this rule. When enabled, commits must first be pushed to another branch, then merged or pushed directly to a branch that matches this rule after status checks have passed.

☐ **Require conversation resolution before merging**

When enabled, all conversations on code must be resolved before a pull request can be merged into a branch that matches this rule. [Learn more.](#)

☐ **Require signed commits**

Commits pushed to matching branches must have verified signatures.



### Protect your most important branches

[Branch protection rules](#) define whether collaborators can delete or force push to the branch and set requirements for any pushes to the branch, such as passing status checks or a linear commit history.

Your [GitHub Free plan](#) can only enforce rules on its public repositories, like this one.

#### Branch name pattern \*

#### Protect matching branches

☒ **Require a pull request before merging**

When enabled, all commits must be made to a non-protected branch and submitted via a pull request before they can be merged into a branch that matches this rule.

☒ **Require approvals**

When enabled, pull requests targeting a matching branch require a number of approvals and no changes requested before they can be merged.

Required number of approvals before merging: 1 ▼

☐ **Dismiss stale pull request approvals when new commits are pushed**

New reviewable commits pushed to a matching branch will dismiss pull request review approvals.

☐ **Require review from Code Owners**

Require an approved review in pull requests including files with a designated code owner.

☐ **Require approval of the most recent reviewable push**

Whether the most recent reviewable push must be approved by someone other than the person who pushed it.

☐ **Require status checks to pass before merging**

Choose which [status checks](#) must pass before branches can be merged into a branch that matches this rule. When enabled, commits must first be pushed to another branch, then merged or pushed directly to a branch that matches this rule after status checks have passed.

☐ **Require conversation resolution before merging**

When enabled, all conversations on code must be resolved before a pull request can be merged into a branch that matches this rule. [Learn more.](#)

## Ponto 4

Para adicionar um ficheiro .gitignore basta usar o seguinte código:

```
git checkout develop
nano .gitignore
Inserir isto dentro do documento (*.docx \n *.doc) depois Ctrl O + Enter + Ctrl X
cat .gitignore
git add .
git commit -m "Adicionar .gitignore"
git push -u origin main
git checkout develop
git pull origin main
```

## Ponto 5

O ponto 5 é dividido em 4 partes cada uma diferente:

- A) git flow feature start relatorioinicial  
git add .  
git commit -m "exemplo"  
git flow feature finish relatorioinicial  
git push -u origin
- B) git flow feature start mudancasrelatorio(1,2..)  
git add .  
git commit -m "exemplo"  
git flow feature finish mudancasrelatorio  
git push -u origin  
Repetir 4x
- C) git flow release start REL\_1.0 develop  
git flow release finish REL\_1.0 -m (o -m é importante pois dá merge)  
git push -all
- D) git flow hotfix start REL\_1.0.1 develop  
git add .  
git commit -m "Hotfix"  
git flow hotfix finish REL\_1.0.1  
git pull origin develop  
git push -all