

# Laboratório de Desenvolvimento de Software

Célio Carvalho  
[cdf@estg.ipp.pt](mailto:cdf@estg.ipp.pt)

P. PORTO

## CONTEXTUALIZAÇÃO

- GitLab é uma ferramenta *web-based* que permite gerir repositórios Git e suporta todas as práticas de DevOps (*Software Development and Information-Technology Operations*).
- GitLab inclui funcionalidades como: wiki, issue-tracking e CI/CD pipeline.
- Foi criado pelos Ucranianos Dmitriy Zaporozhets e Valery Sizov.
- Foi originalmente desenvolvido em Ruby, com algumas partes desenvolvidas em Go.
- GitLab é considerada como o primeiro Unicórnio Ucraniano.
- Tem características sociais e promoveativamente a comunicação dentro das equipas de desenvolvimento.
- Permite o desenvolvimento de software/projetos com recurso a metodologias de desenvolvimento ágeis.
- Permite automatização de tarefas e controlo de métricas no desenvolvimento do software.

## CONTEXTUALIZAÇÃO

- O GitLab fornece suporte à gestão de projetos:
  - ✓ Etapas de desenvolvimento
  - ✓ Equipas
  - ✓ Builds
  - ✓ Testes
  - ✓ Deployments
- Foi criado em 2011, e é usado por mais de 100 000 empresas.

Nasdaq

Dunelm

CARFAX®

Deutsche Telekom

IRON MOUNTAIN®

IBM

INTERPOL

Alibaba.com®

NASA  
SPACEX



Gartner Magic Quadrant for DevOps Platforms 2023

© Gartner, Inc.  
Gartner

## GIT



- O Git é um sistema de controlo de versões distribuído, utilizado para rastrear alterações em projetos de software, mas pode ser utilizado com outro tipo de recursos (e.g. documentos).
- Quando um programador clona um projeto, recebe na sua máquina uma cópia completa do repositório. Essa cópia permite que possa trabalhar desligado do servidor principal (*offline*). Os utilizadores podem trabalhar em diferentes recursos em simultâneo, tornando o desenvolvimento mais ágil.
- Os *commit* são feitos localmente sem afetar o repositório remoto. Quando pretendido, o programador pode atualizar o servidor principal em lote, enviando todo o trabalho executado (conjunto de *commit*).
- A cópia local completa no computador de cada programador oferece redundância e *backup* dos recursos.
- Cada programador pode escolher o melhor momento para incorporar as alterações dos outros colegas, fazendo assim uma melhor gestão do seu trabalho, e do seu repositório local.
- Diz-se que o Git é um sistema distribuído, porque cada clone do repositório é uma cópia completa e independente, o que oferece flexibilidade, redundância, backup, e facilidade do trabalho em equipa.



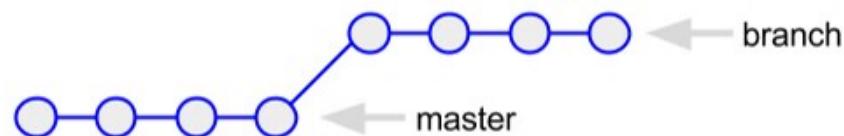
## GIT | FUNCIONAMENTO (1/3)

- O Git permite rastrear as alterações efetuadas em código.
  - ✓ Inicia-se inicializando um diretório como repositório Git, e com a indicação dos ficheiros que devem ser controlados / “versionados”.
  - ✓ Pode-se adicionar e remover ficheiros ao sistema de controlo de versões do Git a qualquer momento.
- Uma das operações mais importantes do Git são os **commit**.
  - ✓ Equivalente a um *snapshot* instantâneo do estado do conteúdo do repositório (ficheiros do diretório).
  - ✓ Trata-se de uma forma de pedir ao Git que registe no seu histórico o conjunto de operações realizadas desde o último **commit** (alterações efetuadas num ou mais ficheiros até aquele momento).
  - ✓ Cada **commit** (i.e. conjunto ou lote de alterações) é identificado através de um **id** atribuído pelo próprio Git, e por um pequeno texto fornecido na criação do **commit**.
  - ✓ Este processo de gravação em lote, facilita a pesquisa e consulta de alterações efetuadas no registo histórico do Git.
  - ✓ Os **commit** podem ser localizados através do **Id** e da mensagem indicada no momento da gravação.
  - ✓ Além disso, é possível saber quando aconteceram os **commit** e, num ambiente em que se usa um repositório remoto, também é possível saber quem os criou (i.e. alterou o conteúdo dos ficheiros).

continua no slide seguinte...

## GIT | FUNCIONAMENTO (2/3)

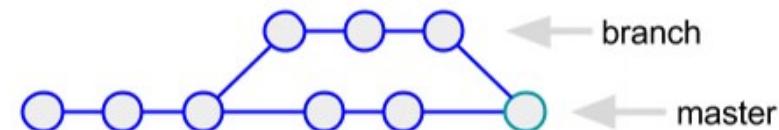
- Uma vantagem na utilização dos sistemas Git, é a possibilidade de *branching*.
  - ✓ Trata-se de uma forma que permite isolar de um branch, trabalhos em curso cujo desenvolvimento poderia impactar a estabilidade do código desse branch (e.g. nova funcionalidade ou módulo).
  - ✓ Criar um novo branch é equivalente a criar um clone do diretório atual para um diretório temporário. No entanto, o diretório mantém-se o mesmo. O sistema de versões é que gera a apresentação do conteúdo do diretório, em função do branch ativo a cada momento.
  - ✓ A persistência de alterações num branch não afeta os restantes branch. Por exemplo, se um novo módulo estiver a ser desenvolvido no branch1, as alterações efetuadas nesse branch não são visíveis nem afetam os demais branch que possam existir no mesmo repositório (diretoria).
  - ✓ A independência e isolamento oferecida pelo sistema de *branching*, permite que se possam fazer e reverter alterações aos ficheiros, sem afetar os mesmos ficheiros noutros branch (mesma diretoria).



continua no slide seguinte...

## GIT | FUNCIONAMENTO (3/3)

- Terminados os trabalhos num branch (e.g. implementação de nova funcionalidade ou módulo) as alterações podem ser integradas em lote noutro branch (e.g. principal).
  - ✓ O processo de fusão de alterações de um branch para outro chama-se merge.
  - ✓ Aquando do merge, o Git tem a capacidade de identificar as diferenças existentes entre o branch origem e destino, e de encaixar / fundir os conteúdos de ambas versões no ficheiro do branch destino.
  - ✓ O histórico de commit (alterações) efetuadas é mantido para consulta futura.
  - ✓ Caso existam conflitos durante o merge, o Git sinaliza-os e pede ao programador que os resolva para continuar o processo de fusão de forma consistente e segura.



- O Git também dá suporte a repositórios remotos (e.g. GitLab, GitHub, Bitbucket)
  - ✓ Desta forma torna-se possível o trabalho em equipa.
  - ✓ Utiliza-se o comando push, para atualizar o repositório remoto com os commit criados localmente.
  - ✓ O comando pull permite atualizar o repositório local com as alterações existentes no repositório remoto, que contém o trabalho realizado pelos demais elementos da equipa.

## GIT | CONCEITOS BÁSICOS



Termo	Descrição
BRANCH	Trata-se de um conceito do Git, que funciona tipo uma ramificação do repositório. A criação de um novo branch permite que se possa trabalhar de forma isolada os recursos do diretório sem afetar os restantes branch do repositório.
COMMIT	Processo executado para pedir ao Git o registo no seu histórico de um lote (conjunto) de alterações efetuadas ao conteúdo do repositório local (indica-se um pequeno texto para facilitar a identificação futura desse conjunto de alterações no histórico do repositório).
MERGE	Processo de fusão de um branch com outro branch. Depois de um ciclo de trabalho num branch distinto, todas as alterações efetuadas nesse branch podem ser persistidas (copiadas) em bloco para o branch principal.
PULL	Processo que permite sincronizar o repositório remoto com o local (recepção do trabalho efetuado pelos outros utilizadores da equipa).
PUSH	Processo que permite sincronizar o repositório local com o remoto (envio para o servidor remoto).
REPOSITÓRIO	<b>Local</b> Diretório local especial onde todo o conteúdo é “versionado” pelo Git. <b>Remoto</b> Diretório remoto para onde o conteúdo do repositório local é sincronizado para ser utilizado por outros.



## GIT | REPOSITÓRIO LOCAL | Configurações iniciais

- Os slides seguintes executam comandos do Git. Para que funcionem, o sistema Git tem que estar instalado no SO. Abaixo apresenta-se o link para o download e o comando para verificar o correto funcionamento.

➤ <https://git-scm.com/>

➤ **git --version** C:\>git --version  
git version 2.37.2.windows.2

celio@cc394:~/demo1\$ git --version  
git version 2.34.1

- Antes de iniciar a utilização do Git, deve configurar-se o nome e o e-mail para que o histórico identifique convenientemente o utilizador com os dados corretos.

➤ **git config --global user.name "Célio Carvalho"**

➤ **git config --global user.email "cdf@estg.ipp.pt"**

- Para validar as configurações atuais execute o comando seguinte.

➤ **git config --list**

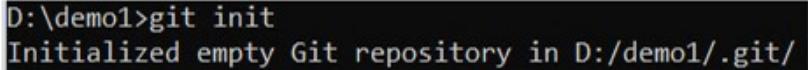
```
core.rscache=true
core.symlinks=false
pull.rebase=false
credential.helper=manager-core
credential.https://dev.azure.com.usehttppath=true
init.defaultbranch=master
core.autocrlf=true
core.excludesfile=C:\Users\Admin\Documents\gitignore_global.txt
user.name=Célio Carvalho
user.email=cdf@estg.ipp.pt
credential.helper=selective-managed-cache
```

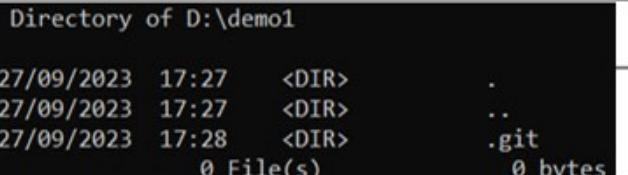
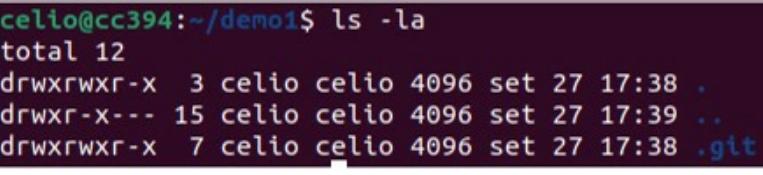


## GIT | REPOSITÓRIO LOCAL | CRIAÇÃO | Demonstração

- A criação de um repositório local, pressupõe a criação de uma diretoria onde o repositório viverá. Notar que pode ser utilizada uma diretoria já existente e até com conteúdo. A primeira instrução abaixo cria um diretório no sistema operativo dentro da pasta atual, e a segunda, entra nesse diretório.

Windows	Linux
➤ <code>md demo1</code>	➤ <code>mkdir demo1</code>
➤ <code>cd demo1</code>	➤ <code>cd demo1</code>

- A instrução seguinte cria um repositório local dentro da pasta atual.
  - `git init` D:\demo1>git init  
Initialized empty Git repository in D:/demo1/.git/
- A instrução seguinte apresenta o conteúdo da pasta, apresentando o conteúdo escondido. Notar que foi criada uma pasta oculta `.git`, pasta onde ficam gravados os dados associados à gestão do repositório.

Windows	Linux
➤ <code>dir /a</code>  Directory of D:\demo1 27/09/2023 17:27 <DIR> . 27/09/2023 17:27 <DIR> .. 27/09/2023 17:28 <DIR> .git 0 File(s) 0 bytes	➤ <code>ls -la</code>  celio@cc394:~/demo1\$ ls -la total 12 drwxrwxr-x 3 celio celio 4096 set 27 17:38 . drwxr-x--- 15 celio celio 4096 set 27 17:39 .. drwxrwxr-x 7 celio celio 4096 set 27 17:38 .git



## GIT | REPOSITÓRIO LOCAL | COMMIT | Demonstração (1/2)

- Como o objetivo desta demonstração é apenas mostrar o funcionamento do Git, em vez de soluções completas de código, apenas se irá utilizar o ficheiro de texto README.md para demonstrar o funcionamento da gestão de versões pelo Git.
  - ✓ Este ficheiro está presente em quase todos os repositórios.
  - ✓ Contém informações genéricas acerca do repositório e do seu conteúdo.
  - ✓ A sua formatação é definida através da linguagem Markdown (<https://www.markdownguide.org>).
- Crie o ficheiro README.md dentro da pasta com o conteúdo da imagem, utilizando um editor de texto.
- Execute o comando seguinte e analise o seu resultado.
  - **git status**
- O resultado apresentado indica que:
  - Se está a trabalhar no branch master (criado por defeito);
  - Não existem ainda commit efetuados; e
  - Existem ficheiros que não estão a ser rastreados.

```
D:\demo1>git status
On branch master
No commits yet
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    README.md
nothing added to commit but untracked files present (use "git add" to track)
```

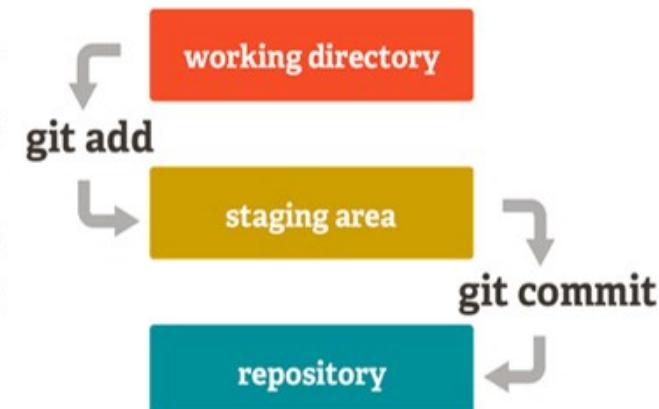
continua no slide seguinte...



## GIT | REPOSITÓRIO LOCAL | COMMIT | Demonstração (2/3)

- O commit é um processo constituído por dois passos:
  - ✓ Em primeiro lugar adiciona-se os ficheiros que se pretende fazer commit à área do *staging* (criação do lote ou lista de ficheiros componentes do futuro commit); e depois
  - ✓ Executa-se o commit propriamente dito. Este comando é que realmente indica ao Git que deve persistir no histórico do repositório, o conjunto de alterações efetuadas nos ficheiros presentes no *staging*.
- ✓ O primeiro comando apresentado abaixo adiciona o ficheiro README.md à área de *staging* (poderiam ser adicionados mais ficheiros). O segundo, desfaz a ação do primeiro, removendo o ficheiro do *staging*. As imagens apresentadas à direita foram obtidas através da execução do git status depois da execução de cada um dos comandos atrás referidos.
  - `git add README.md`
  - ✓ `git rm --cached README.md`

continua no slide seguinte...



```
D:\demo1>git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   README.md
```

```
D:\demo1>git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    README.md

nothing added to commit but untracked files present (use "git add" to track)
```



## GIT | REPOSITÓRIO LOCAL | COMMIT | Demonstração (3/3)

- Depois de definido o lote de ficheiros que farão parte do commit (i.e. depois de colocados na área de *staging* os ficheiros componentes do commit), falta agora realizar efetivamente o commit desses ficheiros.
- Note, mais uma vez, que um commit pode “oficializar” a alteração de vários ficheiros e subpastas no repositório ao mesmo tempo. Nesta demonstração apenas se adicionou o ficheiro README.md à área de *staging* para manter a demonstração simples, mas poderiam ser adicionados mais ficheiros.
- Como anteriormente foi retirado o ficheiro README.md da área de *staging* para demonstrar a operação, agora deverá ser colocado novamente em *staging*, para se realizar o commit (último passo).
  - `git add README.md`
- O comando abaixo termina o processo de commit.
  - `git commit -m "Primeira edição do ReadMe"`

```
D:\demo1>git commit -m "Primeira edição do ReadMe"
[master (root-commit) df64de2] Primeira edição do ReadMe
 1 file changed, 3 insertions(+)
 create mode 100644 README.md
```

- Execute o comando seguinte para consultar o histórico de commit no repositório.
  - `git log`

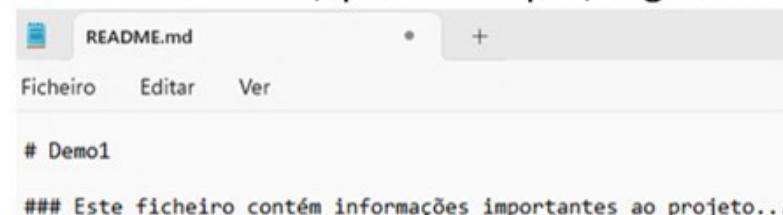
```
D:\demo1>git log
commit df64de2a7f997f17bd9875b676b6c20c04677296 (HEAD -> master)
Author: Célio Carvalho <cdf@estg.ipp.pt>
Date:   Wed Sep 27 23:40:31 2023 +0100

  Primeira edição do ReadMe
```

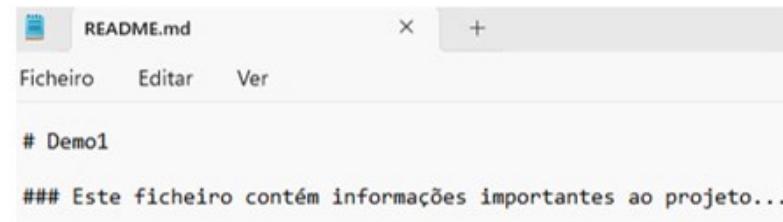


## GIT | REPOSITÓRIO LOCAL | RESTORE| Demonstração

- Uma das vantagens de se utilizar um repositório, é poder-se rever as alterações efetuadas em ficheiros para versões anteriores. Edite novamente o ficheiro README.md e adicione o seu nome, por exemplo, algures no texto.
- Execute o comando a seguir e analise o estado do repositório.  
➤ **git status**
- A informação obtida do Git, indica que foram feitas alterações no ficheiro README.md que ainda não foram persistidas no repositório através de um commit.
- Nesta demonstração pretende-se demonstrar que é possível pedir a reversão das alterações ao repositório. Para isso execute o comando apresentado de seguida. Depois de executar o comando, consulte o conteúdo do ficheiro para validar que o conteúdo foi efetivamente revertido.  
➤ **git restore README.md**



```
D:\demo1>git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
    (use "git restore <file>..." to discard changes in working
     modified: README.md
```



```
# Demo1
### Este ficheiro contém informações importantes ao projeto...
```

## GIT | REPOSITÓRIO LOCAL | Exercício

- Suponha que lhe foi atribuída a tarefa de traduzir o conteúdo do ficheiro README.md para inglês. Edite novamente o ficheiro e substitua a frase traduzindo o seu conteúdo.
- Depois de alterado o conteúdo:
  - Verifique que existem alterações pendentes de commit.
  - Persista essas alterações no repositório utilizando a descrição “Conteúdo traduzido para inglês”.
- No final, consulte o log de repositório, e deverá ter duas entradas similares às abaixo apresentadas.

```
D:\demo1>git log
commit 88e63aaa721a09098eeb06c1fcc8b60e7c781465 (HEAD -> master)
Author: Célio Carvalho <cdf@estg.ipp.pt>
Date:   Thu Sep 28 08:54:04 2023 +0100

    Conteúdo traduzido para inglês

commit df64de2a7f997f17bd9875b676b6c20c04677296
Author: Célio Carvalho <cdf@estg.ipp.pt>
Date:   Wed Sep 27 23:40:31 2023 +0100

    Primeira edição do ReadMe
```



## GIT | REPOSITÓRIO LOCAL | BRANCH

- Por norma, trabalha-se com Git utilizando vários branch (ramos) no mesmo repositório. Os programadores utilizam, pelo menos, dois branch, mas podem existir vários.
- **master** (ou **main**) – é o branch que contém a solução estável e em funcionamento.
  - ✓ O nome do branch não tem que ser necessariamente um destes.
  - ✓ Recomenda-se que não se trabalhe diretamente neste branch para manter o conteúdo estável.
  - ✓ Num ambiente colaborativo (repositório remoto), este é o branch utilizado na sincronização de código entre elementos. Se o código não estiver a compilar, por exemplo, todos os elementos serão afetados quando carregarem alterações do servidor para a sua máquina local.
- «*nova\_funcionalidade\_modulo*» - é o branch criado para suportar temporariamente a execução de uma tarefa, como a criação de uma nova funcionalidade ou módulo, correção de um ou mais bugs, etc.
  - O nome do branch deve ser curto e descriptivo do trabalho que lá será realizado.
  - Criar um novo branch é equivalente a criar um clone da diretoria do projeto.
  - As alterações efetuadas ao conteúdo da diretoria, no contexto do novo branch, não afetam o conteúdo dos ficheiros dos restantes branch incluindo o master (ou main).



## GIT | REPOSITÓRIO LOCAL | BRANCH | Demonstração (1/4)

- Nesta demonstração será simulada a criação de um novo módulo. Esse módulo será composto por, apenas, um ficheiro de código genérico que simbolizará um módulo novo completo. Claro que, num contexto real, o módulo novo poderia ser constituído por vários ficheiros, diretorias, etc.
- Crie um novo branch chamado “modulo1”. O segundo comando abaixo, apresenta os branch existentes. Os commit efetuados afetarão, apenas, o branch ativo (apresentado a verde com o \*).

➤ **git branch modulo1**  
➤ **git branch**

```
D:\demo1>git branch
* master
  modulo1
```

- Também é possível consultar os branch existentes e os seus estados através do comando log.
  - git log**
- O branch ativo continua a ser o master, e ambos os branch apontam para o mesmo commit, o que significa que a versão do seu conteúdo é a mesma.

```
D:\demo1>git log
commit 88e63aaa721a09098eeb06c1fcc8b60e7c781465 (HEAD -> master, modulo1)
Author: Célio Carvalho <cdf@estg.ipp.pt>
Date:   Thu Sep 28 08:54:04 2023 +0100

        Conteúdo traduzido para inglês

commit df64de2a7f997f17bd9875b676b6c20c04677296
Author: Célio Carvalho <cdf@estg.ipp.pt>
Date:   Wed Sep 27 23:40:31 2023 +0100

        Primeira edição do ReadMe
```

continua no slide seguinte...



## GIT | REPOSITÓRIO LOCAL | BRANCH | Demonstração (2/4)

- Para indicar ao Git que se quer trabalhar no novo branch deve executar o checkout para esse branch. Todas as alterações realizadas a partir desta alteração, serão apenas persistidas no branch modulo1.

```
➤ git checkout modulo1          D:\demo1>git checkout modulo1
➤ git branch                    Switched to branch 'modulo1'
                                master
* modulo1
```

- A alteração do branch ativo é também visível através do log.

```
➤ git log
```

```
D:\demo1>git log
commit 88e63aaa721a09098eeb06c1fcc8b60e7c781465 (HEAD -> modulo1, master)
Author: Célio Carvalho <cdf@estg.ipp.pt>
Date:   Thu Sep 28 08:54:04 2023 +0100

        Conteúdo traduzido para inglês

commit df64de2a7f997f17bd9875b676b6c20c04677296
Author: Célio Carvalho <cdf@estg.ipp.pt>
Date:   Wed Sep 27 23:40:31 2023 +0100

        Primeira edição do ReadMe
```

- Isto significa que todas as alterações efetuadas a partir de agora, serão refletidas no novo branch.

- Também é possível criar um novo branch e mudar automaticamente para esse novo branch, num único comando. Para isso utiliza-se a opção -b do checkout como apresentado abaixo.

```
➤ git checkout -b modulo1
```

continua no slide seguinte...

## GIT | REPOSITÓRIO LOCAL | BRANCH | Demonstração (3/4)

- Simule-se agora a criação do módulo, através da criação do ficheiro `file1.js` e com a atualização do `README.md` para indicar a existência do novo módulo na aplicação. Utilize a sugestão de conteúdos das imagens ao lado.
  - `git status`
- Depois das alterações, consulte o estado do repositório. Serão perceptíveis as modificações efetuadas no estado pendente de commit (um ficheiro novo, e outro modificado).
  - `git add .` (o ponto simboliza todos os ficheiros)
  - `git commit -m "Desenvolvimento do módulo 1"`
  - `git log`
- Verifique que, agora, os dois branch apontam para versões diferentes do repositório.

continua no slide seguinte...

```
D: > demo1 > js file1.js > ...
1  function getNumber(min, max) {
2  |   return Math.floor(Math.random() * (max - min + 1)) + min;
3  }

# Demo1

### This file contains important information for the project...

## Modules
* Module 1
```

```
D:\demo1>git status
On branch modulo1
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
    (use "git restore <file>..." to discard changes in working directory)
      modified: README.md

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    file1.js

no changes added to commit (use "git add" and/or "git commit -a")
```

```
D:\demo1>git log
commit d3cc8d50943910b7fde20b312f29ca0456b12d89 (HEAD -> modulo1)
Author: Célio Carvalho <cdf@estg.ipp.pt>
Date:   Sat Dec 16 15:26:18 2023 +0000

  Desenvolvimento do módulo 1
```



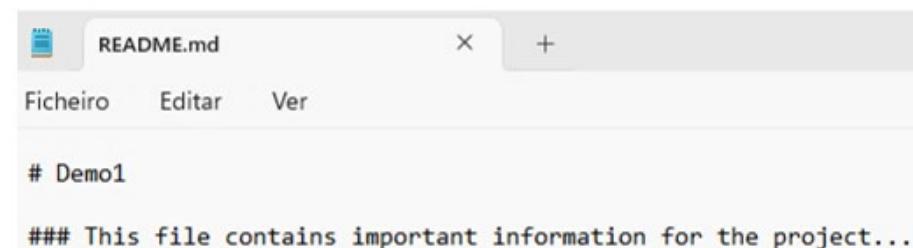
## GIT | REPOSITÓRIO LOCAL | BRANCH | Demonstração (4/4)

- Se os branch existentes apontam para versões de código diferentes, significa que o conteúdo do diretório e do ficheiro README.md será diferente em cada branch.
- Mude para o branch master para validar que o conteúdo é, efetivamente, diferente em cada um dos branch (versões diferentes porque cada um aponta para um commit diferente).
  - `git checkout master`
- Consulte o conteúdo da pasta para confirmar que o file1.js não é visível no branch master. Consulte também o conteúdo do ficheiro README.md para confirmar que não apresenta as alterações efetuadas no branch modulo1.

```
D:\demo1>dir
Volume in drive D is Dados
Volume Serial Number is 98AF-69FB

Directory of D:\demo1

28/09/2023  10:34    <DIR>
28/09/2023  10:34                74 README.md
               1 File(s)           74 bytes
```



A screenshot of a Windows file explorer window titled "README.md". The window shows a single file named "README.md" with a size of 74 bytes. The file icon is a blue document. Below the title bar are buttons for "Ficheiro", "Editar", and "Ver". The main content area displays the file's contents:  

```
# Demo1
### This file contains important information for the project...
```



## GIT | REPOSITÓRIO LOCAL | MERGE | Demonstração (1/2)

- Assuma-se agora que o novo módulo foi concluído. Todas as suas funcionalidades foram implementadas e testadas. Significa que as suas alterações podem ser persistidas no branch master – o tal que deve conter sempre código estável.
- O processo de fusão entre branch chama-se merge. O merge deve ser feito a partir do branch destino. Portanto, antes de avançar esta demonstração, garanta estar no branch master.
  - `git branch`
  - `git merge modulo1`
- No final verifique novamente o conteúdo da diretoria e do ficheiro, para confirmar que o conteúdo desenvolvido no branch `modulo1` já está consolidado no `master`.

```
# Demo1

### This file contains important information for the project...

## Modules
* Module 1
```

continua no slide seguinte...

```
D:\demo1>git merge modulo1
Updating 88e63aa..b7c890d
Fast-forward
 README.md | 7 ++++++-
 file1.js   | 4 +++
 2 files changed, 10 insertions(+), 1 deletion(-)
 create mode 100644 file1.js
```

```
D:\demo1>git branch
* master
 modulo1

D:\demo1>dir
 Volume in drive D is Dados
 Volume Serial Number is 98AF-69FB

 Directory of D:\demo1

28/09/2023  10:59    <DIR>
28/09/2023  10:59                  98 file1.js
28/09/2023  10:59                 106 README.md
                           2 File(s)      204 bytes
```



## GIT | REPOSITÓRIO LOCAL | MERGE | Demonstração (2/2)

- Consulte o log e verifique que ambos os branch apontam agora novamente para o último commit.

➤ **git log**

- Pode também pedir o log do repositório de forma mais resumida utilizando o argumento `--oneline`. O argumento `--all` apresenta todos os commit.

➤ **git log --all --oneline**

```
D:\demo1>git log --all --oneline
'b7c890d (HEAD -> master, modulo1) Desenvolvimento do módulo 1
88e63aa Conteúdo traduzido para inglês
df64de2 Primeira edição do ReadMe'
```

- O branch `modulo1`, pode agora, ser apagado.

➤ **git branch -d modulo1**

➤ **git branch**

```
D:\demo1>git log
commit b7c890dfdb853a84a73b8b9fc205c7fdd175f18a (HEAD -> master)
Author: Célio Carvalho <cdf@estg.ipp.pt>
Date: Thu Sep 28 10:27:03 2023 +0100

    Desenvolvimento do módulo 1

commit 88e63aaa721a09098eeb06c1fcc8b60e7c781465
Author: Célio Carvalho <cdf@estg.ipp.pt>
Date: Thu Sep 28 08:54:04 2023 +0100

    Conteúdo traduzido para inglês

commit df64de2a7f997f17bd9875b676b6c20c04677296
Author: Célio Carvalho <cdf@estg.ipp.pt>
Date: Wed Sep 27 23:40:31 2023 +0100

    Primeira edição do ReadMe
```



## GIT | REPOSITÓRIO LOCAL | STASH | Demonstração (1/3)

- Quando se trabalha com Git, é também possível “esconder” as alterações feitas no repositório, utilizando o comando `stash`. Quando se faz `stash` num repositório, as alterações feitas são “postas de lado” ou “escondidas” podendo posteriormente ser reaplicadas.
- Seguindo as boas práticas, esta demonstração será feita num novo branch chamado `modulo2`. Crie o novo branch e mude-se para ele.
  - `git checkout -b modulo2` (comando cria e muda para o novo branch)
  - `git log --oneline`

```
D:\demo1>git log --oneline
b7c890d (HEAD -> modulo2, master) Desenvolvimento do módulo 1
88e63aa Conteúdo traduzido para inglês
df64de2 Primeira edição do ReadMe
```
- Faça algumas alterações ao conteúdo dos ficheiros, e execute o código abaixo para as “esconder”.
  - `git stash push`
- Consulte o conteúdo para verificar que as alterações efetuadas já não estão visíveis, porque foram colocadas de lado ou “escondidas”. Se solicitar o `status`, verá que as alterações estão escondidas (como se nunca tivessem existido naquele branch).

continua no slide seguinte...



## GIT | REPOSITÓRIO LOCAL | STASH | Demonstração (2/3)

- As alterações podem ainda ser recuperadas porque apenas foram “escondidas”. Antes de abordar os comandos para recuperar as alterações, o comando abaixo apresenta todas as alterações “escondidas”.

➤ **git stash list**

```
D:\demo1>git stash list
stash@{0}: WIP on modulo2: b7c890d Desenvolvimento do módulo 1
```

- Para reaplicar as alterações escondidas, execute o comando seguinte.

➤ **git stash pop**

```
D:\demo1>git stash pop
On branch modulo2
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working
    modified: README.md
    modified: file1.js
```

- A lista de alterações escondidas deverá, agora aparecer vazia.

➤ **git stash list**

```
D:\demo1>git stash list
```

- Caso pretenda reaplicar as alterações sem as retirar da lista de alterações escondidas, poderá executar o command **apply**. Se quiser, posteriormente, apagar essa lista de forma manual, poderá ser utilizado o comando **clear**.

➤ **Git stash apply**

➤ **Git stash clear**

continua no slide seguinte...



## GIT | REPOSITÓRIO LOCAL | STASH | Demonstração (3/3)

- Por fim, como o branch modulo2 foi criado apenas para testar o stash, pode ser removido sem merge. Como não é possível remover um branch ativo, o primeiro comando abaixo muda para o branch master. Só após esta alteração é que o modulo2 pode ser removido
  - `git checkout master`
  - `git branch -d modulo2`
- Por fim, volte a criar o branch modulo2 e faça algumas alterações no conteúdo. Posteriormente, faça um commit dessas alterações e tente apagar novamente o modulo2 seguindo o procedimento anterior.

```
D:\demo1>git branch -d modulo2
error: The branch 'modulo2' is not fully merged.
If you are sure you want to delete it, run 'git branch -D modulo2'.
```

- Como o modulo2 já tem commit registados, terá que ser utilizado o argumento -D em vez do -d.

```
➤ git branch -D modulo2 D:\demo1>git branch -D modulo2
Deleted branch modulo2 (was 788b1a3).
```

## GIT | COMANDOS (1/2)



Comando	Descrição
git config --global user.name “...”	Indica o nome de utilizadora ser utilizado no registo do histórico Git.
git config --global user.email “...”	Indica o e-mail do nome de utilizador.
Git config -global init.defaultBranch main	Indica o default branch do Git como sendo o main.
git init	Inicializa o diretório atual como um repositório.
git status	Apresenta informações acerca do estado atual do repositório.
git add t1.txt	Adiciona o ficheiro t1.txt à área de <i>staging</i> para commit.
git commit -m “...”	Regista um commit identificando-o com a mensagem indicada entre “”.
git rm --cached t1.txt	Remove o ficheiro t1.txt da área de <i>staging</i> para commit.
git branch -m xpto	Muda o nome do branch atual para xpto.
git log	Permite consultar o histórico do repositório.
Git log --all --online	Apresenta o histórico de todos os commit resumindo numa linha.
git branch modulo1	Cria um branch novo com a designação modulo1.
git branch	Apresenta uma lista dos branch existentes, e assinala o que está ativo. continua no slide seguinte...

## GIT | COMANDOS (2/2)



Comando	Descrição
git checkout modulo1	Muda para o branch modulo1
git checkout -b modulo1	Cria o branch modulo1 e muda para esse novo branch.
git stash push	Esconde as alterações efetuadas no branch.
git stash pop	Reaplica as alterações escondidas.
git stash apply	Aplica as alterações escondidas, mas não as retira da lista das escondidas.
git stash clear	Limpa as alterações escondidas (destarta essas alterações).
git branch -d modulo2	Remove o módulo 2
git branch -D modulo2	Remove o módulo 2 mesmo que já tenha um ou mais commit.



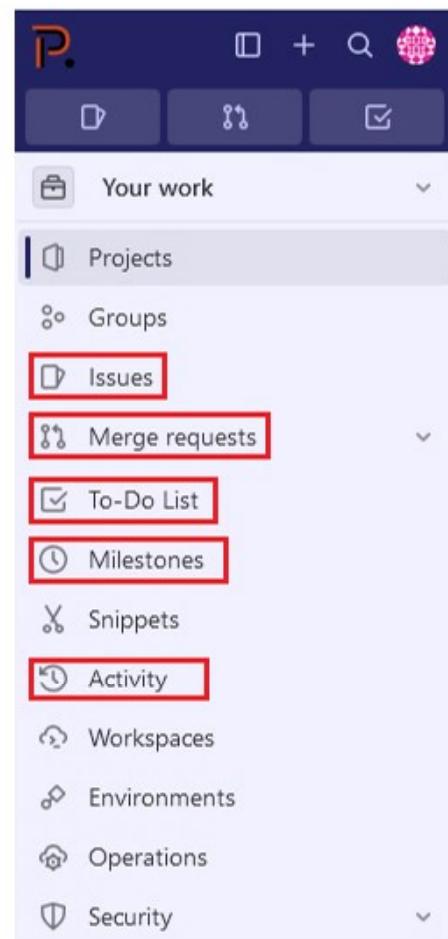
# git Basics

## Branches & Commits

## FUNCIONALIDADES PRINCIPAIS

O GitLab é uma plataforma de gestão do ciclo de vida de desenvolvimento de software que oferece várias funcionalidades relacionadas com a gestão de projetos:

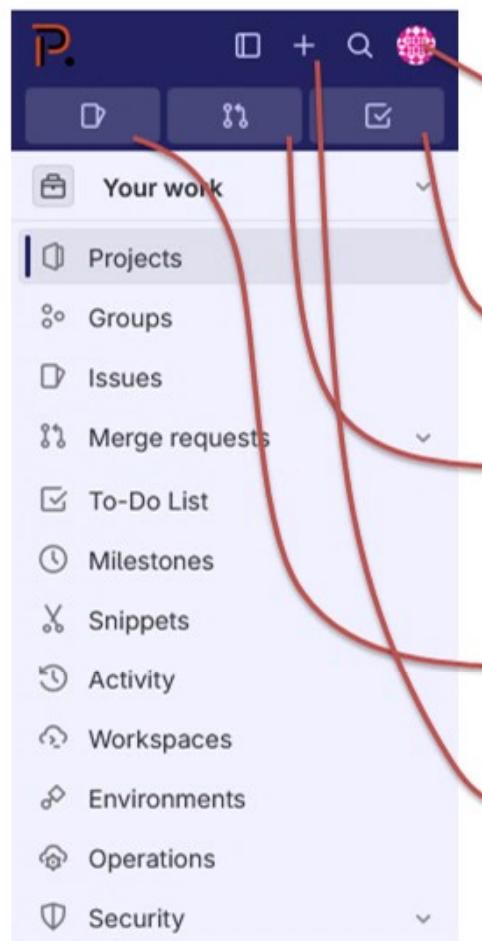
- ✓ **Controlo de versões** (ou versionamento) de código (Git)
- ✓ **Issue tracking** – permite o rastreamento de problemas e tarefas associadas que ocorrem no âmbito da gestão do projeto.
- ✓ **Kanban boards** – que facilita o acompanhamento e análise visual do andamento (progresso) dos fluxos de trabalho em curso.
- ✓ **Gráficos burndown** – para rastrear o andamento do projeto e estimar a conclusão das tarefas.
- ✓ **CI/CD** – permite configurar pipelines para *Continuous Integration & Continuous Delivery*.
- ✓ **Análises** – oferece vários relatórios para análise (e.g. desempenho das equipas, projeto).
- ✓ **Outros** – *Forks, clones, branching, merge requests, Wiki (GitLab pages, Markdown), snippets, gestão de permissões*, etc.



## VOCABULÁRIO GITLAB

Termo	Descrição
GROUP	Permite gerir configurações entre vários projetos em simultâneo. Os grupos podem ser vistos como formas de organizar utilizadores ou projetos. Permite aceder a conteúdo numa perspetiva global aos vários projetos do mesmo grupo.
ISSUE	É uma forma de rastrear trabalho relacionado com um projeto. Os <i>issues</i> são utilizados para reportar <i>bugs</i> , rastrear tarefas, pedir novas funcionalidades, colocar questões, etc.
MEMBER	Membros são utilizadores ou grupos de utilizadores que tem acesso a um projeto. As permissões são geridas através <i>roles</i> associadas a cada membro.
MERGE REQUEST	Permite fazer <i>merge</i> de um <i>branch</i> para outro. Em GitLab é um espaço que interação escrita entre utilizadores acerca das alterações efetuadas no <i>branch</i> . Um <i>merge request</i> é o local na aplicação onde as alterações efetuadas ao código são revistas e validadas.
PROJECT	Pode ser visto como um <i>container</i> para um repositório Git. Um projeto GitLab tem um repositório. Um projeto tem de base CI/CD. Pode fazer-se <i>issue tracking</i> dentro de um projeto. Contém também ferramentas de colaboração (e.g. <i>merge requests</i> ).

## GITLAB INTERFACE



Componente	Descrição
PROFILE	<b>Edit Profile</b> Editar os dados pessoais do utilizador autenticado. <b>Preferences</b> Alterar as preferências de utilização do utilizador atual.
TO-DO LIST	Trabalho para ser executado pelo utilizador atual (lista de itens.)
MERGE REQUEST	<b>Assigned to you</b> Consultar os <i>merge requests</i> que foram atribuídos ao utilizador atual. <b>Review requests for you</b> Consultar os <i>merge requests</i> onde o utilizador atual foi definido como reviewer desse <i>merge request</i> .
ISSUES	Procurar <i>issues</i> entre projetos. Está disponível a possibilidade de aplicar vários filtros para encontrar <i>issues</i> .
NEW	<b>New project/repository</b> Criar um novo projeto / repositório. <b>New group</b> Criar um novo grupo (agregação de vários projetos). <b>New snippet</b> Criar um excerto de código.

## GITLAB | GROUPS | CRIAÇÃO | Demonstração (1/2)

- Para criar um novo grupo, preencher o formulário para o efeito.

➤ [Groups | New Group](#)

Smart Cities Dev Team

Group ID: 75401421

Subgroups and projects Shared projects Archived projects

New subgroup New project

- Também é possível criar subgrupos dentro de grupos. A título de exemplo, a equipa de desenvolvimento das *Smart Cities* poderá ser desdobrada na subequipa das *Smart Lights* e do *Smart Traffic*. Abaixo apresenta-se as imagens de criação de apenas um dos subgrupos.

Subgroup name

Smart Lights

Must start with letter, digit, emoji, or underscore. Can also contain periods, dashes, spaces, and parentheses.

Subgroup URL

https://gitlab.com/ smart-cities-dev-team / smart-lights

Subgroup slug

smart-lights

Group path is available.

Smart Cities Dev Team > Smart Lights

Smart Lights

Group ID: 75402171

New subgroup New project

continua no slide seguinte...

Create group

Groups allow you to manage and collaborate across multiple projects. Members of a group have access to all of its projects.

Groups can also be nested by creating subgroups.

Group name

Smart Cities Dev Team

Must start with letter, digit, emoji, or underscore. Can also contain periods, dashes, spaces, and parentheses.

Group URL

https://gitlab.com/ smart-cities-dev-team

Group path is available.

Visibility level

Who will be able to see this group? [View the documentation](#)

Private The group and its projects can only be viewed by members.

Public The group and any public projects can be viewed without any authentication.

Now, personalize your GitLab experience

We'll use this to help surface the right features and information to you.

Role

Software Developer

Who will be using this group?

My company or team

Just me

What will you use this group for?

I want to store my code

Invite Members (optional)

Invited users will be added with developer level permissions. [View the documentation](#) to see how to change this later.

Email 1

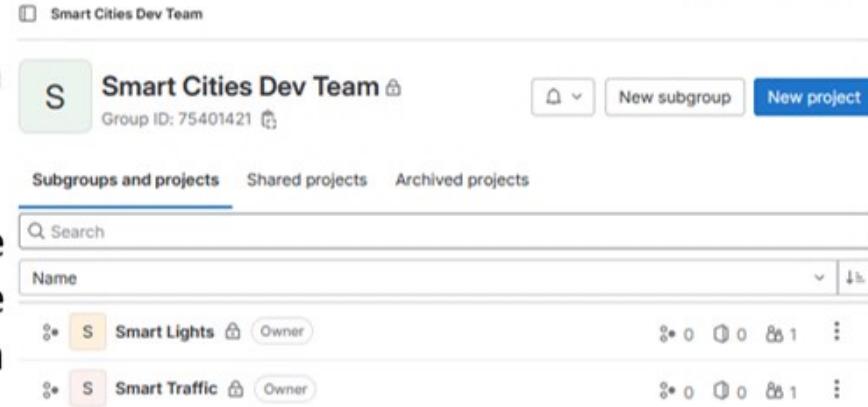
cdf@estg.ipp.pt

+ Invite another member

Create group Cancel

## GITLAB | GROUPS | CRIAÇÃO | Demonstração (2/2)

- Após a criação dos subgrupos, a estrutura final do grupo seria como apresentado na imagem.
- Depois de criada a estrutura de grupos e subgrupos, pode iniciar-se a gestão dos projetos, nomeadamente com o convite de membros, criação de *labels* (etiquetas que servem para categorizar issues, por exemplo), etc.
- A opção Issues quando acedida dentro do grupo, permite consultar os issues de forma transversal aos vários projetos dentro do grupo.
- O mesmo acontece com os Merge requests. Quando consultados dentro de um grupo, estarão visíveis todos aqueles que estejam relacionados com projetos deste grupo
- Para que as demonstrações decorram de forma simples, vai-se fazer a gestão, apenas, do grupo das *Smart Cities*, i.e., os subgrupos vão ser removidos.



The screenshot shows the 'Subgroups and projects' tab for the 'Smart Cities Dev Team' group. The group ID is 75401421. There are two subgroups listed: 'Smart Lights' and 'Smart Traffic', both of which are owned by the group. The interface includes a search bar, a 'Name' dropdown, and buttons for 'New subgroup' and 'New project'.

## GITLAB | PROJECTS | CRIAÇÃO | Demonstração (1/2)

- A criação de projetos pode ser feita, por exemplo, a partir de vários locais (e.g. homepage, dentro do group).
- Após a criação, será apresentado o ecrã do projeto como apresentado abaixo. Como foi pedida a inclusão do README.md, o conteúdo é apresentado em destaque.

**T Traffic Operations Frontend**

Project ID: 50823734

1 Commit 1 Branch 0 Tags 3 KIB Project Storage

Initial commit Célio Carvalho authored just now

main traffic-operations-frontend / + v

History Find file Edit v Clone v

README Add LICENSE Add CHANGELOG Add CONTRIBUTING Add Kubernetes cluster Set up CI/CD Add Wiki Configure Integrations

Name Last commit Last update

README.md Initial commit just now

README.md

**Traffic Operations Frontend**

**Getting started**

To make it easy for you to get started with GitLab, here's a list of recommended next steps.

Already a pro? Just edit this README.md and make it your own. Want to make it easy? Use the template at the bottom!

Add your files

**Create blank project**

Create a blank project to store your files, plan your work, and collaborate on code, among other things.

**Project name**  Must start with a lowercase or uppercase letter, digit, emoji, or underscore. Can also contain dots, pluses, dashes, or spaces.

**Project URL**  **Project slug**  1

Want to organize several dependent projects under the same namespace? Create a group.

**Project deployment target (optional)** Select the deployment target

**Visibility Level**  Private Project access must be granted explicitly to each user. If this project is part of a group, access is granted to members of the group.  
 Public This project cannot be public because the visibility of Smart Cities Dev Team is private. To make this project public, you must first change the visibility of the parent group.

**Project Configuration**

Initialize repository with a README Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing repository.

Enable Static Application Security Testing (SAST) Analyze your source code for known security vulnerabilities. Learn more.

**Create project** **Cancel**

continua no slide seguinte...

## GITLAB | PROJECTS | CRIAÇÃO | Demonstração (2/2)



- Na parte superior do projeto é possível consultar os commit feitos nesse projeto, é possível também criar ou mudar de branch e criar ou fazer upload de um ficheiro. É também possível fazer download do repositório ou fazer *clone*. Existe também um Web IDE na opção *Edit*, que permite editar o projeto online.

The screenshot shows the GitLab project page for 'Traffic Operations Frontend'. At the top, there's a header with a 'T' icon, the project name, a lock icon, a 'Project ID: 50823734' link, and social sharing icons for GitHub, LinkedIn, and others. Below the header, a summary bar shows '1 Commit', '1 Branch', '0 Tags', and '3 KiB Project Storage'. The main area displays a single commit from 'Célio Carvalho' titled 'Initial commit' made 11 minutes ago. To the right of the commit, there's a commit hash '48d084e9' with a copy icon and two red arrows pointing down towards the 'Clone' button. Below the commit, there are buttons for 'History', 'Find file', 'Edit', 'Download', and 'Clone'. A modal window is open over the file list, showing options to 'New file', 'Upload file', or 'New directory' under 'This directory'. Another modal window is open at the bottom, showing options to 'New branch' or 'New tag' under 'This repository'. On the left, there's a sidebar with file navigation ('main', 'traffic-operations-frontend /'), file creation buttons ('README', 'Add LICENSE', 'Add'), and a list of files ('README.md'). The footer shows the last update was 11 minutes ago.

## GITLAB | MILESTONES | CRIAÇÃO | Demonstração



- Os *milestones* são marcos temporais que fazem referência a um momento com significado num projeto (e.g. *sprint*). Esta é a forma como em GitLab se definem os *sprints*.
- As *milestones* são úteis para acompanhar o andamento de tarefas. São metas temporais definidas dentro do projeto em que determinados objetivos devem ser atingidos.

The screenshot shows the 'Milestones' section of a GitLab project. At the top, there is a search bar labeled 'Filter by milestone name', a dropdown for 'Due soon', and a blue 'New milestone' button. Below this, there are filters for 'Open 1', 'Closed 0', and 'All 1'. A specific milestone is highlighted with a red circle containing the number '2':  
**Spring #3**  
Nov 7, 2023–Nov 21, 2023  
Upcoming  
0 Issue · 0 Merge requests  
0% complete  
Close Milestone

### New Milestone

Title: Sprint #3

Start Date: 2024-10-01

Due Date: 2024-10-15

Description: Development of the sales module.

Buttons: Create milestone (blue), Cancel (white)

## GITLAB | LABELS | CRIAÇÃO | Demonstração

- As labels podem ser aplicadas aos issue e aos merge request, para facilitar o processo de pesquisa (exemplos abaixo).
- Por exemplo, se um determinado issue for etiquetado com a label UX, num processo de pesquisa futuro em que seja aplicado um filtro de label com esse valor, a issue etiquetada será apresentada como um dos resultados.

**BD-DDL**  
Smart Cities Dev Team / Traffic Operations Frontend  
Data Definition Language

**C#**  
Smart Cities Dev Team / Traffic Operations Frontend  
C#

**React**  
Smart Cities Dev Team / Traffic Operations Frontend  
React

**BD-DML**  
Smart Cities Dev Team / Traffic Operations Frontend  
Data Manipulation Language

**JS**  
Smart Cities Dev Team / Traffic Operations Frontend  
Javascript

**HTML**  
Smart Cities Dev Team / Traffic Operations Frontend  
HTML

**EF**  
Smart Cities Dev Team / Traffic Operations Frontend  
Entity Framework

**UX**  
Smart Cities Dev Team / Traffic Operations Frontend  
User eXperience

New Label

Title

Description (optional)

Background color  #6699cc

Select a color from the color picker or from the presets below.

**Create label** **Cancel**



GITLAB | ISSUES | CRIAÇÃO | Demonstração (1/3)

- Os issue é uma opção muito importante do GitLab na gestão de projetos de software, porque é onde se indica e configura o trabalho a realizar dentro do projeto.
  - Um issue pode ser uma *user story*, *bug* a corrigir, proposta de melhoria, etc.
  - Na criação de um issue, é indicado um título que identifica a tarefa, uma descrição, e outros dados como:
    - O responsável pela execução da tarefa;
    - A *milestone* que indica quando a tarefa deve ser executada (e.g. id do *sprint*);
    - Data em que a tarefa deve estar concluída;
    - Etiquetas (*labels*) associadas ao issue que podem ser utilizadas, posteriormente, em filtros de pesquisa.

continua no slide seguinte...

## GITLAB | ISSUES | CRIAÇÃO | Demonstração (2/3)

- Alguns dos dados anteriores, podem ser preenchidos posteriormente, (e.g. no início de cada *sprint*) (e.g. criação da *milestone*).
- A imagem à direita demonstra a atribuição do issue criado anteriormente a um utilizador para o *sprint* #3.
- Após a gravação dessas alterações, a apresentação visual do issue é atualizada, e poder-se-ão aplicar filtros tendo por base os novos valores atribuídos (imagem abaixo).

The screenshot shows the 'Issues' page for the 'Traffic Operations Frontend' project. At the top, there are buttons for 'Bulk edit' and 'New issue'. Below that, a filter bar shows 'Open 1', 'Closed 0', and 'All 1'. The main area displays two issues:

- #2 - Prepare the ReadMe file: created just now by Célio Carvalho. It has a status dropdown set to 'In Progress'.
- #1 - Left menu development: created just now by Célio Carvalho. It has a status dropdown set to 'In Progress'.

A search bar at the bottom includes filters for 'Assignee' (set to 'Célio Carvalho') and 'Created date'.

The screenshot shows the 'New issue' dialog box. Several fields have red arrows pointing to them:

- 'Status': 'Select status' dropdown.
- 'Assignee': 'Célio Carvalho' dropdown.
- 'Milestone': 'Spring #3' dropdown.

At the bottom right, there are buttons for 'Update all' and 'Cancel'.

continua no slide seguinte...

## GITLAB | ISSUES | CRIAÇÃO | Demonstração (3/3)



- As descrições dos issue podem conter caixas de seleção para melhorar controlo das tarefas terminadas de cada issue. Nas imagens abaixo edita-se o issue acabado de criar para conter checkbox's correspondentes às duas tarefas a realizar no issue.

The screenshot shows the 'Issue' creation form for a project titled 'Left menu development'. The 'Description' field contains the following text:

```
- It must be responsive.  
- Each menu option should be presented with an icon on the left and a hyperlink pointing to an address.  
- ...  
Tasks to perform:  
- [ ] CSS style definition  
- [ ] HTML development
```

At the bottom of the description field, there is a link 'Switch to rich text editing' and two buttons: 'Save changes' (blue) and 'Cancel' (white).

### Left menu development

The screenshot shows the 'Issue actions' section for the same issue. The description has been updated to include a completed task:

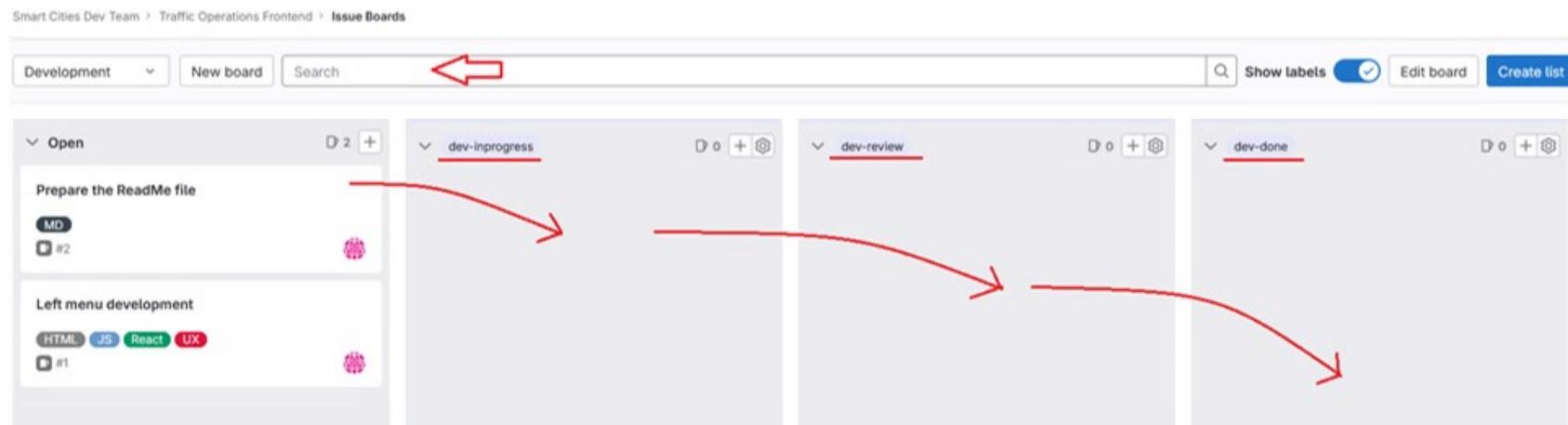
- It must be responsive.
- Each menu option should be presented with an icon on the left and a hyperlink pointing to an address.
- ... Tasks to perform:
  - CSS style definition
  - HTML development

Below the description, it says '1 of 2 checklist items completed · Edited just now by Célio Carvalho'. At the bottom are three interaction buttons: a thumbs up (0), a thumbs down (0), and a smiley face.

## GITLAB | ISSUE BOARD | Setup



- Os issue board (quadro de tarefas) permite visualizar a organização dos issue em pistas. O GitLab suporta a existência de várias em simultâneo. O objetivo principal das issue board é organizar visualmente as tarefas a executar pelos utilizadores.
- Os issue board são utilizados em várias realidades, e são muito utilizados em desenvolvimento de software. O conceito dado a cada List (pista) varia em função do objetivo pretendido. Em software, é habitual configurar pistas em função da estado em que se encontra cada tarefa a executar pelos vários utilizadores. Nesta estratégia, as tarefas devem ser movidas por cada utilizador, à medida que iniciem o desenvolvimento, concluem e passem para revisão, e se concluam os trabalhos.



## GITLAB | GIT | REPOSITÓRIO REMOTO (SSH) | Setup (1/3)

- Para se poder utilizar Git com o repositório criado (projeto) GitLab, tem que se configurar a ligação do Git local ao Git remoto. Nesta demonstração será configurada uma ligação SSH. Para isso, será necessário gerar uma chave public/private (assuma os valores *default* nas questões colocadas pelo ssh-keygen).
  - `ssh-keygen -t ed25519 -C "GitLab (autenticação)"`

- A chave pública gerada, terá que ser inserida no GitLab para que seja possível estabelecer a comunicação entre os dois sistemas. Copie o conteúdo do ficheiro .pub gerado para o *clipboard*.
  - Profile | Edit profile | SSH Keys | Add new key

```
D:\demo2>ssh-keygen -t ed25519 -C "GitLab (autenticação)"
Generating public/private ed25519 key pair.
Enter file in which to save the key (C:\Users\Admin/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in C:\Users\Admin/.ssh/id_ed25519
Your public key has been saved in C:\Users\Admin/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:[REDACTED]yQ GitLab (autenticação)
The key's randomart image is:
+--[ED25519 256]--+
|          .o|
|          .o|
|          .o|
```

**Add an SSH key**

Add an SSH key for secure access to GitLab. [Learn more.](#)

**Key**

```
ssh-ed25519 [REDACTED]
```

Begins with 'ssh-rsa', 'ssh-dss', 'ecdsa-sha2-nistp256', 'ecdsa-sha2-nistp384', 'ecdsa-sha2-nistp521', 'ssh-ed25519', 'sk-ecdsa-sha2-nistp256@openssh.com', or 'sk-ssh-ed25519@openssh.com'.

**Title**

Key titles are publicly visible.

**Usage type**

**Expiration date**

Optional but recommended. If set, key becomes invalid on the specified date. 

**Add key** **Cancel**

continua no slide seguinte...

## GITLAB | GIT | REPOSITÓRIO REMOTO (SSH) | Setup (2/3)

- Depois desta configuração do chave SSH, já será possível conseguir-se fazer uma autenticação a partir da linha de comandos. O comando abaixo tenta estabelecer uma ligação com o servidor GitLab e, se bem sucedida, apresenta uma mensagem de boas vindas.

➤ `ssh -T git@gitlab.com`

```
D:\demo2>ssh -T git@gitlab.com
Welcome to GitLab, @celio.carvalho!
```

- Note que, para conseguir estabelecer uma comunicação SSH, o serviço remoto tem que estar devidamente configurado. Se durante a execução do comando anterior, um erro de *time out* ou outro similar aconteça, poderá significar que não está disponível a comunicação SSH com esse servidor.
- À data de criação destes slides, não estava disponível a comunicação SSH com o GitLab da ESTG|IPP, a partir da rede pública.

➤ `ssh -T git@gitlab.estg.ipp.com`

```
D:\demo2>ssh -T git@gitlab.estg.ipp.pt
ssh: connect to host gitlab.estg.ipp.pt port 22: Connection timed out
```

continua no slide seguinte...

## GITLAB | GIT | REPOSITÓRIO REMOTO (SSH) | Setup (3/3)



- Um projeto GitLab é um repositório Git que pode ser trabalhado localmente como repositório remoto. Para que tal seja possível, o projeto tem que ser clonado na máquina local. A indicação do endereço a utilizar no processo de clonagem, encontra-se na opção Clone do GitLab (ver imagem abaixo).

The screenshot shows a GitLab repository named "Traffic Operations Frontend". The repository has 1 commit, 1 branch, 0 tags, and 3 Kib Project Storage. The main commit is by Célio Carvalho, authored 2 hours ago. Below the commit, there are several project management buttons: README, Add LICENSE, Add CHANGELOG, Add CONTRIBUTING, Enable Auto DevOps, Add Kubernetes, and Configure Integrations. To the right, there are navigation buttons for History, Find file, Edit, and a Clone button. The Clone section displays two options: "Clone with SSH" (traffic-operations-frontend.git) and "Clone with HTTPS" (https://gitlab.com/smart-cities/traffic-operations-frontend). The "Clone with SSH" option is highlighted with a red box.

- Execute o comando seguinte no sistema local para que a clonagem por SSH aconteça. Note que a informação que coloca no comando depende do projeto que pretende clonar.

➤ `git clone git@gitlab.com:smart-cities-dev-team/traffic-operations-frontend.git`

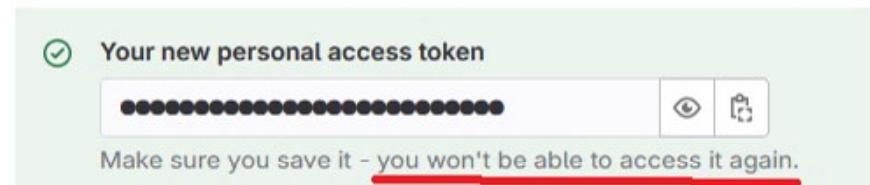
```
D:\demo2>git clone git@gitlab.com:smart-cities-dev-team/traffic-operations-frontend.git
Cloning into 'traffic-operations-frontend'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.
```

## GITLAB | GIT | REPOSITÓRIO REMOTO (HTTPS) | Setup (1/3)



- Existe outra forma de se conseguir ligação com servidor remoto GitLab – a ligação HTTPS. Esta demonstração faz um novo clone do repositório existente no servidor online, agora utilizando o protocolo HTTPS. Normalmente a comunicação HTTPS está sempre disponível, por tratar-se do protocolo utilizado para navegar na aplicação.
- Gere um novo Access Token para poder autenticar o Git via HTTPS, e indique convenientemente as permissões que pretende atribuir a esse novo *token*. Atenda à data de expiração escolhida. Logo após a criação, será disponibilizado o *token* que será utilizado na autenticação. **Guarde-o** porque não será apresentado novamente.

➤ [Profile](#) | [Edit profile](#) | [Access Tokens](#) | [Add new token](#)



Token name	Scopes	Created	Last Used	Expires	Action
Acesso Maquina Local Célio Carvalho	api, read_api, read_user, create_runner, k8s_proxy, read_repository, write_repository, read_registry, write_registry, ai_features	Sep 29, 2023	15 hours ago	in 11 months	

continua no slide seguinte...

## GITLAB | GIT | REPOSITÓRIO REMOTO (HTTPS) | Setup (2/3)



- O token gerado anteriormente será utilizado durante a autenticação HTTPS. Para clonar um projeto através deste método recorra também à opção Clone do GitLab. Copie o endereço do campo assinalado a vermelho da imagem abaixo, e utilize-o na instrução abaixo.

➤ `git clone https://«username»:«token»@gitlab.com/smart-cities-dev-team/traffic-operations-frontend.git`

```
D:\demo3>git clone https://celio:glpat-10071271-100je@gitlab.com/smart-cities-dev-team/traffic-operations-frontend.git
Cloning into 'traffic-operations-frontend'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.
```

continua no slide seguinte...

## GITLAB | GIT | REPOSITÓRIO REMOTO (HTTPS) | Setup (3/3)



- Note que, antes de executar o comando `git clone`, além de utilizar o endereço indicado pelo GitLab, deve substituir o «`username`» e «`token`» pelos valores reais.
- Caso execute o `clone` indicando apenas o endereço (i.e. sem dados autenticação), será apresentada uma janela de autenticação do GitLab.

```
➤ git clone https://gitlab.com/smart-cities-dev-team/traffic-operations-frontend.git
```

The figure consists of three side-by-side screenshots of a 'Connect to GitLab' sign-in window. Each window features the GitLab logo and the text 'Sign in'. Below the logo are three tabs: 'Browser', 'Token', and 'Password'. The first screenshot shows the 'Browser' tab is selected, with a blue button labeled 'Sign in with your browser'. The second screenshot shows the 'Token' tab is selected, with two input fields: 'Username or email (optional)' and 'Personal access token', and a 'Sign in' button below them. The third screenshot shows the 'Password' tab is selected, with two input fields: 'Username or email' and 'Password', and a 'Sign in' button below them. At the bottom of each window, there is a link 'Don't have an account? Sign up'.

## GITLAB | GIT | REPOSITÓRIO REMOTO | Demonstração (1/10)



- Após integrar o repositório local com o remoto, pode utilizar-se o Git para alterar / atualizar os conteúdos do projeto GitLab. No entanto, antes de avançar, pode verificar-se o estado e *log* do repositório. Execute os comandos seguintes, dentro da pasta criada pelo `git clone`.

➤ `git status`  
➤ `git log`

```
D:\demo3\traffic-operations-frontend>git status
On branch main
Your branch is up to date with 'origin/main'.
```

```
D:\demo3\traffic-operations-frontend>git log
commit 48d084e93ba1fa654a7afe9a4ea9215fd4c94136 (HEAD -> main, origin/main, origin/HEAD)
Author: Célio Carvalho <cdf@estg.ipp.pt>
Date:   Fri Sep 29 14:38:12 2023 +0000
```

- O projeto contém, apenas, duas *user stories*. Considerando o contexto demonstrativo deste projeto, assumir-se-á que o trabalho a realizar resume-se à execução dessas *user stories*. Note que, num projeto real, existiriam várias *user stories*, que seriam executadas ao longo de vários *sprints*.
- Apenas lembrar, a primeira *user story* diz respeito à preparação (personalização) do README.md. A segunda, refere-se ao desenvolvimento do menu lateral esquerdo da aplicação. Considerando o contexto demonstrativo, considera-se como concluída a primeira tarefa com a estruturação do ficheiro ReadMe, e a segunda com a criação do ficheiro leftMenu.html e leftMenu.js contendo, por exemplo um comentário simples dentro de cada um.

The screenshot shows a checklist interface from GitLab. It contains two items:

- Prepare the ReadMe file  
#2 · created 22 minutes ago by Célio Carvalho · Spring #3  
MD
- Left menu development 1 of 2 checklist items completed  
#1 · created 1 day ago by Célio Carvalho · Spring #3  
HTML JS React UX

continua no slide seguinte...

## GITLAB | GIT | REPOSITÓRIO REMOTO | Demonstração (2/10)



- Como já referido, a implementação de um novo módulo / funcionalidade / recurso, deve ser sempre executada num branch criado para o efeito. Comece por indicar no Issue Board, que iniciou o desenvolvimento desta *user story*.
- A primeira *user story* refere-se à preparação do ficheiro ReadMe. O comando abaixo cria um novo branch que suportará a execução desta tarefa.
  - `git checkout -b prepare-readme-file`
- Estruture o conteúdo do README.md e, posteriormente, faça o commit no repositório executando os comandos abaixo (sugere-se também a verificação de estado e consulta de log). Nesta primeira edição do ficheiro, apenas se pretende que insira as secções principais do documento como apresentado na imagem.
  - `git status`
  - `git add README.md`
  - `git status`
  - `git commit -m "readme file structure implementation"`
  - `git status`
  - `git log`

```
D:\demo3\traffic-operations-frontend>git checkout -b prepare-readme-file
Switched to a new branch 'prepare-readme-file'
```

continua no slide seguinte...

```
# Traffic Operations Frontend
## Introduction
...
## Architecture Details
...
## Module Description
...
## Implementation Notes
...
## Smart Cities Dev Members
* Célio Carvalho
* ...
```

## GITLAB | GIT | REPOSITÓRIO REMOTO | Demonstração (3/10)

- Edite novamente o ficheiro ReadMe e preencha as secções que estruturou na edição anterior com conteúdo a seu gosto. Depois, faça novo commit para guardar também estas alterações no repositório.
  - `git add README.md`
  - `git commit -m "filling in the readme file content"`
- Note que, num ambiente real (não demonstrativo), provavelmente não existiriam duas iterações de edição do ficheiro ReadMe. Possivelmente, considerando tratar-se de duas tarefas simples e encadeadas, talvez a estruturação e preenchimento das secções ocorressem em simultâneo e originassem apenas um commit. No entanto, considerando o cariz formativo desta demonstração, optou-se por executar esta tarefa em duas fases como forma de simular um comportamento próximo do real. Pretende-se reforçar a ideia de que, a seguir à conclusão de cada parte da tarefa principal (subtarefa), deve existir um commit confirmando a conclusão bem sucedida de parte do problema a resolver (tarefa principal).
- Resumindo, deve criar-se um novo branch para a execução de uma tarefa (e.g. implementação de uma nova funcionalidade). À medida que cada subtarefa é concluída, deve haver um commit no repositório a oficializar esse acréscimo. Desta forma, a grão da rastreabilidade é maior. Entre outras vantagens, esta abordagem permite regredir para numa versão anterior de código de forma mais fina.

continua no slide seguinte...

## GITLAB | GIT | REPOSITÓRIO REMOTO | Demonstração (4/10)

- Todas as alterações efetuadas até ao momento, impactaram apenas o repositório local. Consulte o GitLab e verifique que o branch criado localmente ainda não existe, assim como as alterações efetuadas ao ficheiro README.md (o commit existente foi gerado pelo sistema na criação do projeto).
- Para que as alterações sejam enviadas para o servidor, tem que se executar um push da informação. No entanto, antes de se enviar a informação, é necessário integrar previamente as possíveis alterações existentes no servidor (alterações submetidas por outros utilizadores). Execute o comando abaixo para receber e integrar as alterações existentes no repositório remoto no repositório local.

➤ **git pull**

- Depois de integradas as atualizações do servidor, pode-se agora enviar as alterações locais para o repositório remoto. Execute o comando abaixo para sincronizar o repositório local com o remoto. O argumento -u indica ao Git que deve criar o branch no destino.

➤ **git push -u origin prepare-readme-file**



```
D:\demo3\traffic-operations-frontend>git push -u origin prepare-readme-file
Enumerating objects: 8, done.
Counting objects: 100% (8/8), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (6/6), 685 bytes | 685.00 KiB/s, done.
Total 6 (delta 1), reused 0 (delta 0), pack-reused 0
remote:
remote: To create a merge request for prepare-readme-file, visit:
remote:   https://gitlab.com/smart-cities-dev-team/traffic-operations-fronte
remote:   st%5Bsource_branch%5D=prepare-readme-file
remote:
To https://gitlab.com/smart-cities-dev-team/traffic-operations-frontend.git
 * [new branch]      prepare-readme-file -> prepare-readme-file
branch 'prepare-readme-file' set up to track 'origin/prepare-readme-file'.
```

continua no slide seguinte...

## GITLAB | GIT | REPOSITÓRIO REMOTO | Demonstração (5/10)

- Na aplicação GitLab poderá agora consultar as alterações enviadas para o servidor. O branch acabado de criar já estará disponível, assim como o histórico de commit efetuados.
- Alterne entre os dois branch disponíveis, para verificar que a versão do conteúdo do README.md é diferente entre ambos.

Name	Last commit
README.md	<u>Initial commit</u>

**Traffic Operations Frontend**

**Getting started**

To make it easy for you to get started with GitLab, here's a list of recommended next steps

Already a pro? Just edit this README.md and make it your own. Want to make it easy? [UI](#)

[Add your files](#)

Name	Last commit
README.md	<u>filling in the readme file content</u>

**Traffic Operations Frontend**

**Introduction**

...here content

**Architecture Details**

continua no slide seguinte...

Smart Cities Dev Team > Traffic Operations Frontend > Commits

prepare-readme-file traffic-operations-frontend Author Search by message

Sep 30, 2023

filling in the readme file content Célio Carvalho authored 55 minutes ago 92cd549b

readme file structure implementation Célio Carvalho authored 2 hours ago bd11686b

Sep 29, 2023

Initial commit Célio Carvalho authored 1 day ago 48d084e9

Smart Cities Dev Team > Traffic Operations Frontend > Activity

You pushed to **prepare-readme-file** 21 minutes ago

[Create merge request](#)

All Push events Merge events Issue events Comments Wiki Designs

Célio Carvalho @celio.carvalho 22 minutes ago

-> Pushed new branch **prepare-readme-file**

Célio Carvalho @celio.carvalho 1 day ago

-> Pushed new branch **main**

## GITLAB | GIT | REPOSITÓRIO REMOTO | Demonstração (6/10)

- Assumindo a conclusão da tarefa “Prepare the ReadMe file”, falta agora persistir todo o conteúdo do branch `prepare-readme-file` no branch `main`, e informar o GitLab que a *user story* foi concluída.
- Para criar um Merge Request utilize a opção indicada abaixo.
  - [Projects | Traffic Operations Frontend | Code | Merge Requests | New merge request](#)
- Indique o branch origem e destino, e preencha o formulário de Merge Request (sugestão na imagem ao lado). Num ambiente real, seriam indicados outros utilizadores para revisão e concretização do Merge.
- Por fim, seguindo a estratégia definida no Issue Board, arraste a tarefa para a pista do dev-review, como apresentado na imagem.



New merge request

From `prepare-readme-file` into `main` Change branches

Title (required)

`Prepare readme file (completed)`

Mark as draft  
Drafts cannot be merged until marked ready.

Description

Preview

~~I have just finished preparing the ReadMe file (structure and content).~~

Switch to rich text editing

Add description templates to help your contributors to communicate effectively!

Assignee

Célio Carvalho

Reviewer

Célio Carvalho

Approvals are optional.  
➤ Approval rules

Milestone

Spring #3

Labels

Select label

Merge options

Delete source branch when merge request is accepted.

Squash commits when merge request is accepted. ⓘ

Create merge request Cancel

continua no slide seguinte...

## GITLAB | GIT | REPOSITÓRIO REMOTO | Demonstração (7/10)

- Os Merge Request pendentes podem ser consultados no GitLab.
  - [Projects](#) | [Traffic Operations Frontend](#) | [Merge requests](#)
- Dentro de um Merge Request, além de ser possível rever a atividade dos utilizadores e commit efetuados, é possível:
  - ✓ Analisar visualmente as alterações efetuadas no repositório (podem selecionar-se os 2 branch a comparar, e também a forma de comparação: *inline* ou *side-by-side*).
  - ✓ Aprovar o Merge Request estará disponível se o utilizador atual for o *reviewer* (a opção de *review* pode ser configurada como obrigatória).
  - ✓ Efetuar o Merge Request.

The screenshot shows the GitLab interface for a merge request. On the left, there's a sidebar with file browser options (List view, Tree view) and compare changes settings (Inline, Side-by-side, Show whitespace changes, Show one file at a time). The main area displays a merge request from 'Célio Carvalho' to merge 'prepare-readme-file' into 'main'. It shows a 'Changes 1' tab with a diff view for 'README.md'. The diff highlights additions in green and deletions in red. The changes are: 1: # Traffic Operations Frontend, 2: , 3: + ## Introduction, 4: + ...here content, 5: , 6: + ## Architecture Details, 7: + ...here content, 8: , 9: + ## Getting started, 10: + ## Module Description.

The screenshot shows the GitLab interface for a merge request. At the top, it says 'Smart Cities Dev Team > Traffic Operations Frontend > Merge requests'. Below is a table with columns: Open (1), Merged (0), Closed (0), All (1). Under 'Recent searches' is a search bar. Below that is a 'Created date' dropdown. A message 'Prepare readme file (completed)' is shown, followed by a timestamp '11 - created 12 hours ago by Célio Carvalho'. A red arrow points to this message. At the bottom right, it says 'updated 12 hours ago'.

The screenshot shows a detailed file comparison between 'main' and 'latest version' for 'README.md'. It highlights changes in green and red. The changes are: 1: # Traffic Operations Frontend, 2: , 3: + ## Introduction, 4: + ...here content, 5: , 6: + ## Architecture Details, 7: + ...here content, 8: , 9: + ## Getting started, 10: + ## Module Description.

continua no slide seguinte...

GITLAB | GIT | REPOSITÓRIO REMOTO | Demonstração (8/10)



- Ainda durante a revisão do código, é possível comentar linhas de conteúdo. Neste caso, a título demonstrativo, vai ser pedida a revisão do conteúdo de uma secção do README.md
  - A partir do momento em a Review é solicitada, na secção Activity do separador overview, passa a existir uma opção para indicar que o problema foi resolvido – Resolve Thread.

Célio Carvalho @celio.carvalho started a thread on the diff ^ Hide thread just now

README.md 

```
1 1 # Traffic Operations Frontend
2 2
3 + ## Introduction
4 + ...here content
```

Comment on lines [+3](#) to [+4](#)

 Célio Carvalho @celio.carvalho Author Owner      
just now

This should be a little more descriptive.



Open Célio Carvalho requested to merge `prepare-readme-file` into `main`  
12 hours ago

[Mark as done](#)

Overview 0 Commits 2 Pipelines 0 Changes 1

Compare main ▾ and latest version ▾

README.md

+11 -86 Viewed

Add a comment to this line or drag for multiple lines Operations Frontend

3 + ## Introduction  
4 + ...here content

Commenting on lines +3 to +4

Preview

This should be a little more descriptive.

Switch to rich text editing

Start a review Add comment now Cancel

continua no slide seguinte...

## GITLAB | GIT | REPOSITÓRIO REMOTO | Demonstração (9/10)

- Altere a secção apontada pelo *reviewer*, e submeta novamente o código ao servidor remoto. Quando fizer o push, notará no *output* o *match* da alteração com o Merge Request pendente.
- ```
Writing objects: 100% (3/3), 327 bytes | 327.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote:
remote: View merge request for prepare-readme-file:
remote: https://gitlab.com/smart-cities-dev-team/traffic-operations-frontend/-/merge_requests/1
```
- Após efetuar o push das alterações, use a opção Resolve Thread para indicar ao *reviewer* que o problema foi resolvido.
  - O *reviewer* pode agora aprovar o conteúdo do repositório e fazer o Merge Request. Antes de aprovar a revisão, pode consultar as alterações efetuadas em reação ao que solicitou.
  - Dentro do Merge Request, utilize os botões respetivos para aprovar e concluir o Merge Request (apague o branch que suportou o desenvolvimento desta tarefa).
  - Depois de concluído, o Merge Request pode ainda ser revertido (ver imagem acima). Pode também ser útil reutilizar alguns commit para um outro branch.
  - Confirme que existe, apenas, o branch main, e que todas as alterações foram fundidas.
- continua no slide seguinte...

Célio Carvalho @celio.carvalho started a thread on the diff 14 minutes ago

**README.md**

```

1 1  # Traffic Operations Frontend
2 2
3 + ## Introduction
4 + ...here content

```

Comment on lines +3 to +4

Célio Carvalho @celio.carvalho · 14 minutes ago Author Owner

This should be a little more descriptive.

Reply... Resolve thread

Célio Carvalho added 1 commit 4 minutes ago

- d619803c - introduction rewritten to be more descriptive

Compare with previous version

```

3 3  ## Introduction
4 4  ...here content
5 + ...extra content here.....

```

Merged by Célio Carvalho 20 minutes ago

Revert Cherry-pick

Merge details

- Changes merged into main with a7602d87.
- Deleted the source branch.

## GITLAB | GIT | REPOSITÓRIO REMOTO | Demonstração (10/10)



- Faltam apenas dois passos: mover a tarefa para a pista *done* do Issue Bord; e atualizar o repositório local com as últimas atualizações do repositório (no repositório local ainda existe o branch apagado).
- Mova a *user story* para a pista *done*. O tratamento dado posteriormente dependerá da estratégia da equipa de desenvolvimento. Por exemplo, poderá seguir para testes, ou ser arquivada num fim de *sprint*.
- Para atualizar o repositório local execute os comandos abaixo. Como se vai apagar o branch antigo, primeiro muda-se o contexto para outro branch (neste caso o main). Depois apaga-se o branch antigo. Por fim, pede-se a atualização do repositório local trazendo as alterações do repositório remoto. Este segundo comando, também remove referências locais a branch remotas que já não existem.

```
➤ git branch
➤ git checkout main
➤ git branch -d prepare-readme-file
➤ git pull --prune
```

```
D:\demo3\traffic-operations-frontend>git pull --prune
From https://gitlab.com/smart-cities-dev-team/traffic-operations-frontend
 * [deleted]          (none)    -> origin/prepare-readme-file
remote: Enumerating objects: 12, done.
remote: Counting objects: 100% (12/12), done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 10 (delta 2), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (10/10), 1.18 KiB | 172.00 KiB/s, done.
  48d084e..a7602d8  main      -> origin/main
Updating 48d084e..a7602d8
Fast-forward
 README.md | 98 ++++++-
 1 file changed, 12 insertions(+), 86 deletions(-)
```

## GITLAB | GIT | REPOSITÓRIO REMOTO | Exercício



- Continue e termine o exercício anterior. A segunda *user story* representa uma tarefa para desenvolvimento do menu lateral esquerdo da aplicação.
  - ✓ Comece por mover a *user story* correspondente para a pista `dev-inprogress`. Não esqueça de ir movendo para as pistas seguintes, quando o estado de conclusão assim o justificar.
  - ✓ Crie um novo branch chamado `left-menu-implementation`.
  - ✓ Implemente os ficheiros `leftmenu.html` e `leftMenu.js`. Considerando que apenas se pretende praticar a componente do GitLab, coloque apenas um comentário dentro de cada um dos ficheiros.
  - ✓ Preferencialmente, divida o seu trabalho em vários commit. Não esqueça que não é suposto fazer commit, apenas quando terminar todo o trabalho. Deve fazer um commit a cada subparte concluída.
  - ✓ No final, submeta as alterações ao repositório remoto. Não esqueça que tem que indicar a criação do branch do lado do servidor.
  - ✓ Depois de confirmar a sincronização no GitLab, crie um Merge Request. Como no seu ambiente não existirão mais utilizadores, atribua a responsabilidade de rever o código e de fazer o *merge* a si próprio.
  - ✓ Efetue um pedido revisão de código com uma alteração à sua escolha.
  - ✓ Volte ao seu repositório local, implemente as alterações solicitadas, e envie-as novamente ao servidor.
  - ✓ Conclua o Merge Request apagando o branch `left-menu-implementation`.
  - ✓ Sincronize o seu repositório local com o remoto.

## GITLAB | GIT | TAG & RELEASES | Demonstração

- Depois de concluir as *user story* previstas para esta aplicação, crie uma Tag que simbolizará a versão v1.0 acabada de terminar.
  - [Projects](#) | [Traffic Operations Frontend](#) | [Code](#) | [Tags](#) | [New tag](#)
- Depois de criada a Tag, pode utilizar-se a opção Create release para criar uma *release* da v1.0.
- A nova *release* pode ser encontrada no menu [Projects](#) | [Traffic Operations Frontend](#) | [Deploy](#) | [Releases](#).
- Faça um novo pull para sincronizar novamente o repositório local, incluindo a nova release.

```
D:\demo3\traffic-operations-frontend>git pull
remote: Enumerating objects: 1, done.
remote: Total 1 (delta 0), reused 0 (delta 0), pack-reused 1
Unpacking objects: 100% (1/1), 166 bytes | 83.00 KiB/s, done.
From https://gitlab.com/smart-cities-dev-team/traffic-operations-frontend
 * [new tag]      v1.0      -> v1.0
Already up to date.
```

**New Tag**

Do you want to create a release with the new tag? You can do that in the [New release page](#).

Tag name

Create from

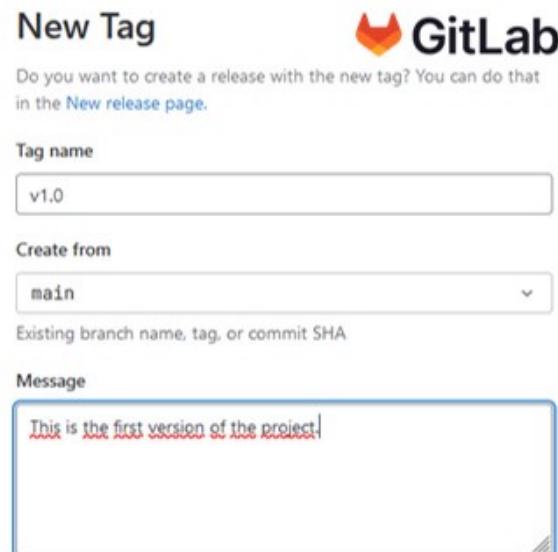
Existing branch name, tag, or commit SHA

Message

This is the first version of the project.
 

This is the first version of the project.

[Create tag](#) [Cancel](#)



Optionally, add a message to the tag. Leaving this blank creates a lightweight tag.

[Create tag](#) [Cancel](#)

**Version 1.0**

|             |                     |                                               |
|-------------|---------------------|-----------------------------------------------|
| 0% complete | Milestone           | Issues <span style="color: #007bff;">2</span> |
| Spring #3   | Open: 2 • Closed: 0 |                                               |

Assets 4

- [Source code \(zip\)](#)
- [Source code \(tar.gz\)](#)
- [Source code \(tar.bz2\)](#)
- [Source code \(tar\)](#)

Evidence collection

- [v1.0-evidences-6736019.json](#) 43ed7839

Collected just now

b6a0fa53 v1.0 Created just now by 

## GITLAB | EXPLORAÇÃO DE DADOS (ANÁLISES)

- Consulte e analise os seguintes recursos oferecidos pelo GitLab dentro de cada projeto:
  - ✓ Manage | Activity
  - ✓ Plan | Milestones | All (*apenas as issue fechadas – closed – são consideradas como concluídas*)
  - ✓ Code | Commits
  - ✓ Code | Repository graph
  - ✓ Analyze | Value stream analytics
  - ✓ Analyze | Contributor statistics
  - ✓ Analyze | Repository Analytics

## GITLAB | ANEXOS | Instruções GitLab

- Quando cria um repositório vazio no GitLab, a própria aplicação sugere os comandos necessários para se sincronizar o repositório local com o servidor. Dada a utilidade desta informação, é deixada aqui como anexo.

### Git global setup

```
git config --global user.name "Célio Carvalho"  
git config --global user.email "cdf@estg.ipp.pt"
```

### Create a new repository

```
git clone git@gitlab.com:demos152740/tests.git  
cd tests  
git switch --create main  
touch README.md  
git add README.md  
git commit -m "add README"  
git push --set-upstream origin main
```

### Push an existing folder

```
cd existing_folder  
git init --initial-branch=main  
git remote add origin git@gitlab.com:demos152740/tests.git  
git add .  
git commit -m "Initial commit"  
git push --set-upstream origin main
```

### Push an existing Git repository

```
cd existing_repo  
git remote rename origin old-origin  
git remote add origin git@gitlab.com:demos152740/tests.git  
git push --set-upstream origin --all  
git push --set-upstream origin --tags
```

## GITLAB | DOCUMENTAÇÃO

- Explore o funcionamento do Wiki do GitLab.
  - [Plan](#) | [Wiki](#)

P. PORTO