

Tutorial 3

Git



André Souto e Thibault Langlois

Unidade Curricular de
Laboratório de Programação

2020/2021

Resumo

Este é o tutorial para as aulas de LabP sobre o uso do Git, um sistema de controlo de versões de ficheiros.

Descrição do Git

O Git é um sistema de controlo de versões que permite gerir um projecto ou conjunto de ficheiros à medida que estes vão sendo alterados ao longo do tempo permitindo rever e comparar diferentes versões entre si. Permite também reverter um ficheiro ou projetos inteiros para um estado anterior.

Um sistema de controlo de versões como o Git é fundamental no desenvolvimento de projetos cooperativos em que diversas pessoas podem contribuir simultaneamente, através da criação de novos ficheiros ou da edição de ficheiros já existentes. Ainda assim, é também uma ferramenta extremamente útil mesmo quando estamos a programar sozinhos.

O Git organiza os ficheiros numa estrutura de dados designada por **repositório**. O Git funciona com base em repositórios distribuídos; ao contrário de outros sistemas de gestão de versões, não existe no Git a noção de repositório central, i.e., um único local onde se guardam todas as versões dos ficheiros. Cada repositório distribuído contém todo o historial de todos os ficheiros. A maior parte dos comandos do Git precisam apenas de recursos e arquivos locais, não sendo geralmente necessária informação que esteja guardada noutro computador.

Após efetuar as alterações desejadas num diretório de trabalho, o utilizador regista-as no repositório Git local e pode depois publicá-las num repositório Git remoto, caso este exista. Note que pela natureza distribuída da arquitetura Git, é possível vários utilizadores alterarem a sua cópia local do mesmo ficheiro criando potenciais conflitos que têm de ser resolvidos como veremos mais adiante.

Nos exemplos deste guião vamos usar a linha de comandos e no final veremos a integração com o Eclipse.

Antes de começar

Antes de mais, pressupõe-se que tem o Git instalado na linha de comandos da sua máquina. Caso não tenha, sugere-se que procure online a forma de instalar a versão do sistema Git adequada ao seu sistema operativo.

Na primeira utilização do Git é necessário configurar o email e o nome do utilizador para que os comandos posteriores identifiquem quem fez as alterações ao repositório:

```
> git config --global user.email "xx@yy.com"
> git config --global user.name "myUsername"
```

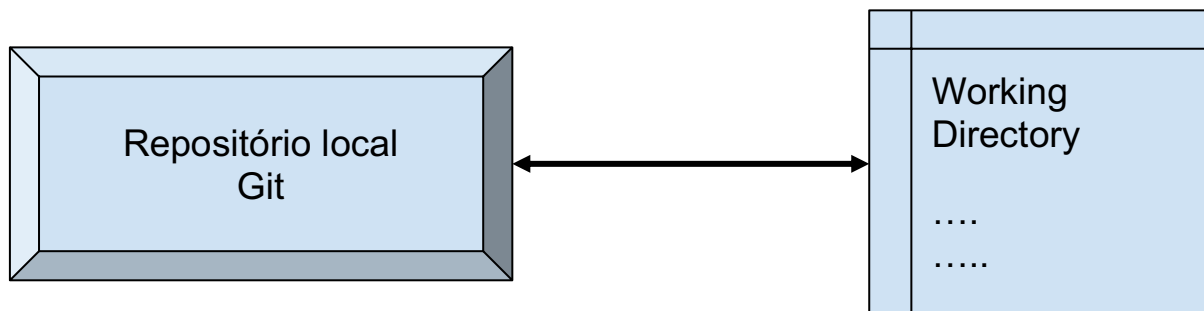
onde xx@yy.com é o seu e-mail e myUsername é o seu nome de utilizador.

Para verificar a sua configuração pode usar o comando:

```
> git config --list
```

Começando a usar o Git (introdução)

Para simplificar vamos supor, que queremos manter apenas um repositório local na nossa máquina e trabalhamos sobre os ficheiros do projecto no Eclipse. Ou seja, na nossa máquina temos armazenados num diretório (o **repositório local**) todas as alterações que produzimos no nosso projecto e que o nosso projecto é elaborado no Eclipse, num *workspace* que contém uma cópia dos ficheiros (e que, para um sistema de controlo de versões, se denomina **working directory**).



Os comandos

A primeira questão que se coloca é: “Como é que criamos um repositório local, isto é, como é que criamos uma estrutura que contém as várias versões dos ficheiros que vamos alterando?” O primeiro passo é a criação de um diretório e de uma instrução git que o sinaliza como contendo a estrutura de repositório:

```
> git init
```

Esta instrução cria dentro do diretório corrente, um subdiretório `.git` que contém toda a informação do estado do repositório. No início este repositório não terá ficheiros¹. Note que este diretório é gerido pelo Git e portanto não é aconselhável alterar o seu conteúdo. Experimente o seguinte conjunto de instruções:

```
> mkdir experienciaGit  
> cd experienciaGit
```

¹ No Git também se usa a palavra *artefacto* para designar ficheiros ou conjuntos de ficheiros.

```
> git init
Initialized empty Git repository in
/Users/ans/Documents/LabP1617/LabP1617/experienciaGit/.git/
> ls -al
total 0
drwxr-xr-x  3 ans  staff  102 Feb 10 08:40 .
drwxr-xr-x  5 ans  staff  170 Feb 10 08:40 ..
drwxr-xr-x 10 ans  staff  340 Feb 10 08:40 .git
```

Esta última nota leva à segunda questão: *Como é que o Git sabe que ficheiros devem estar no repositório?* Teremos de ser nós a dizer-lhe através do comando *add*:

```
> git add nome_do_ficheiro
```

Vejamos um exemplo concreto. Primeiro criamos um ficheiro (neste caso vamos usar o comando *echo* do Linux e redirecionar o output para o ficheiro *hello.txt* com o símbolo ">") e depois indicamos que este ficheiro deve estar contido no repositório:

```
> echo "hello" > hello.txt
> git add hello.txt
```

Note que o comando *add* apenas informa que o ficheiro é suposto ser tratado pelo repositório. Para efetivar a colocação de uma versão (a primeira ou uma versão atualizada) do ficheiro no repositório é necessário *publicar* essas alterações no repositório. Esta acção é concretizada com o comando *commit* (no qual se deve usar uma mensagem descritiva, como veremos adiante):

```
> git commit -m "mensagem elucidativa"
```

Execute a seguinte sequência de comandos e verifique o resultado. Note que o comando *touch* do Linux é aqui usado para criar o ficheiro vazio *ReadMe.txt*.

```
> touch ReadMe.txt
> ls
ReadMe.txt
> git add ReadMe.txt
> git commit -m "o Primeiro commit, adiciona o ReadMe.txt"

[master (root-commit) c233151] o Primeiro commit, adiciona o
ReadMe.txt
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 ReadMe.txt
```

Surge então a terceira questão: “O que acontece aos ficheiros entre a execução do comando *add* e a execução do comando *commit*?”

Na verdade, o Git funciona através do conceito de índice (*index*) ou palco (*stage*). Um *stage* ou *index* é um coletor de *add*'s (ou seja de registos da intenção de alteração) dos ficheiros a serem publicados num próximo *commit*. Após o *commit* o índice é limpo, e no repositório é criado um novo objecto com a informação dos ficheiros alterados e que estavam no índice, com informação sobre quem a executou, e uma mensagem descritiva desta publicação.

De forma resumida, ao usar o Git recorreremos: (i) ao diretório de trabalho (*working directory*), (ii) à área de preparação (*staging area*) e (iii) ao repositório do Git (*repository*). A passagem de (i) para (ii) faz-se com o comando *add* e a passagem de (ii) para (iii) faz-se com o comando *commit* (que requer uma mensagem descritiva das alterações feitas).

No caso de querer remover um ficheiro do repositório usa-se o comando **rm** sobre o ficheiro do diretório e posteriormente reflecte-se essa acção no repositório (ou directamente) através do comando **git rm nome do ficheiro**.

```
> git rm nome_do_ficheiro
```

No exemplo que se segue assuma que o ficheiro com o nome **paraRemover.txt** se encontra no repositório local.

```
> git rm paraRemover.txt
rm 'paraRemover.txt'

> git commit -m "remocao do ficheiro paraRemover"
[master 7448eea] remocao do ficheiro paraRemover
1 file changed, 0 insertions(+), 0 deletions(-)
delete mode 100644 paraRemover.txt
```

A quarta questão, “Como é que podemos saber o estado de um repositório?”, pode ser respondida usando o comando *status*:

```
> git status
```

que informa o utilizador sobre os novos ficheiros, os ficheiros que foram alterados ou inseridos e os que foram removidos. Existe ainda um outro conjunto de informações importantes que o comando *status* indica: quais são os ficheiros que estão no *stage* (isto é indexados para serem publicados) e os que estão no *working directory* e que não são alvo de indexação para publicação. Sugere ainda um conjunto de comandos

alternativos para o que fazer a seguir.

Antes de publicar as alterações (i.e., de fazer o *commit*) é possível retirar do *stage* ficheiros que lá se encontrem e que afinal não se pretende guardar no repositório. Tal é realizado com o comando:

```
> git reset HEAD nome_do_ficheiro
```

Neste contexto, o conceito de HEAD é a marca do Git que identifica a versão que se deve considerar a atual do ficheiro que está guardada no repositório. Experimente executar os seguintes comandos:

```
> touch ReadMeNew.txt
> touch ReadMeYetAnother.txt
> git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    ReadMeNew.txt
    ReadMeYetAnother.txt

nothing added to commit but untracked files present (use "git add"
to track)
> git add ReadMeNew.txt ReadMeYetAnother.txt
> git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   ReadMeNew.txt
    new file:   ReadMeYetAnother.txt

> git reset HEAD ReadMeYetAnother.txt
> git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

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

    ReadMeYetAnother.txt
```

Note que com a aplicação do comando `reset HEAD` a versão local fica inalterada, ou seja, as alterações feitas não são perdidas. Contudo na mesma situação, pode

reverter-se as alterações feitas num ficheiro para a última versão guardada no repositório através do comando *checkout*:

```
> git checkout nome_do_ficheiro
```

sendo também possível com o mesmo comando copiar uma versão anterior desde que seja explicitada qual.

```
> echo "Esta eh uma frase do ReadMe" > ReadMe.txt
> git add ReadMe.txt
> git commit -m "ReadMe alterado"
[master d67d191] ReadMe alterado
1 file changed, 1 insertion(+)
> cat ReadMe.txt
Esta eh uma frase no ReadMe
> echo "Esta eh outra" >> ReadMe.txt
> cat ReadMe.txt
Esta eh uma frase no ReadMe
Esta eh outra
> git add ReadMe.txt
> git reset HEAD ReadMe.txt
Unstaged changes after reset:
M   ReadMe.txt
> cat ReadMe.txt
Esta eh uma frase no ReadMe
Esta eh outra
> git checkout ReadMe.txt
> cat ReadMe.txt
Esta eh uma frase no ReadMe
```

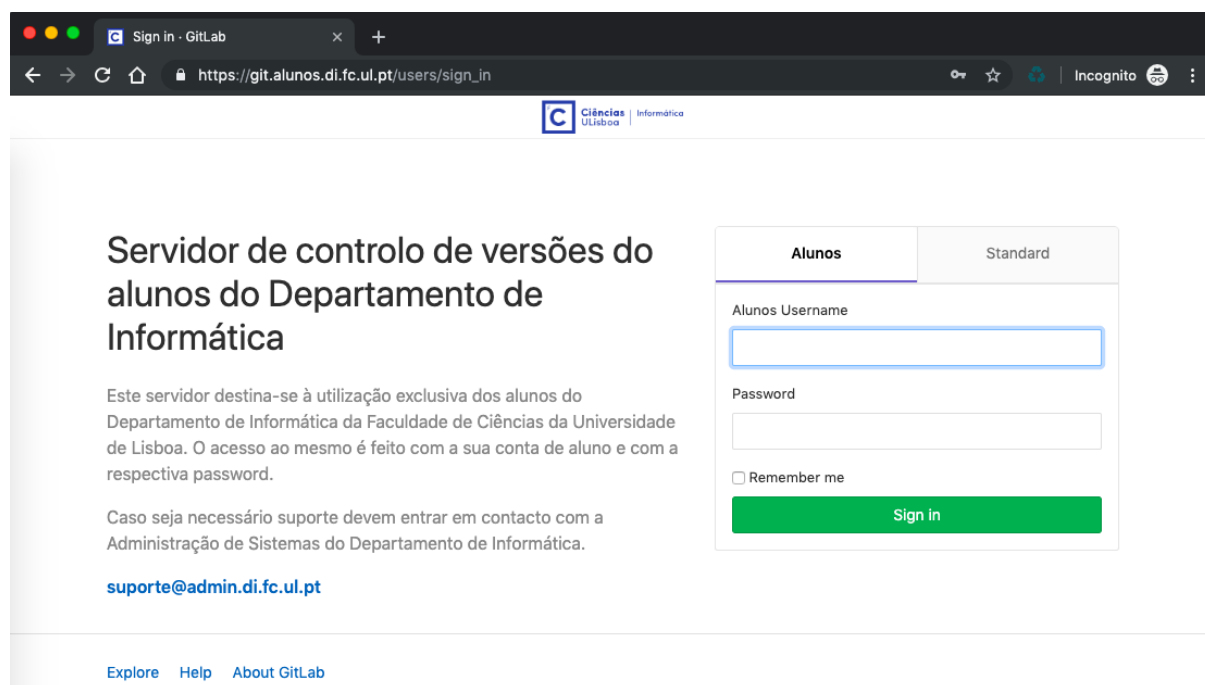
A quinta questão é “O que significa a informação ‘On branch master’?” Indica que estamos no ramo principal ou mestre (*master*) do nosso repositório. O Git permite a funcionalidade de criação de ramos diferenciados de desenvolvimento a partir de um ponto (isto é de um *commit*) de forma a que possam ser aplicadas alterações a ambos os ramos de forma independente e que mais tarde possam ser juntas (ou removidos). É a forma de em equipa, por exemplo, se desenvolver código em simultâneo e de forma independente, sem afetar o trabalho uns dos outros. Este tema será abordado mais à frente no curso noutras unidades curriculares.

A interação com um repositório remoto

Até agora, temos trabalhado apenas num repositório local. O Git não tem um modelo cliente-servidor, mas sim *Peer-to-peer* e cada membro da equipa tem o seu próprio repositório independente. A partilha da informação em Git é feita via protocolos de "ssh" ou "http". De modo a facilitar o modelo de comunicação e gestão recorre-se a um repositório remoto para partilhar o trabalho entre os vários membros.

Desta forma e para que não haja necessidade de todos comunicarem com todos, cada um trabalha no seu próprio repositório local e, quando termina, "*empurra*" o código para o repositório remoto. Por outro lado, os outros elementos podem "*puxar*" o código deste repositório e obter as alterações mais recentes.

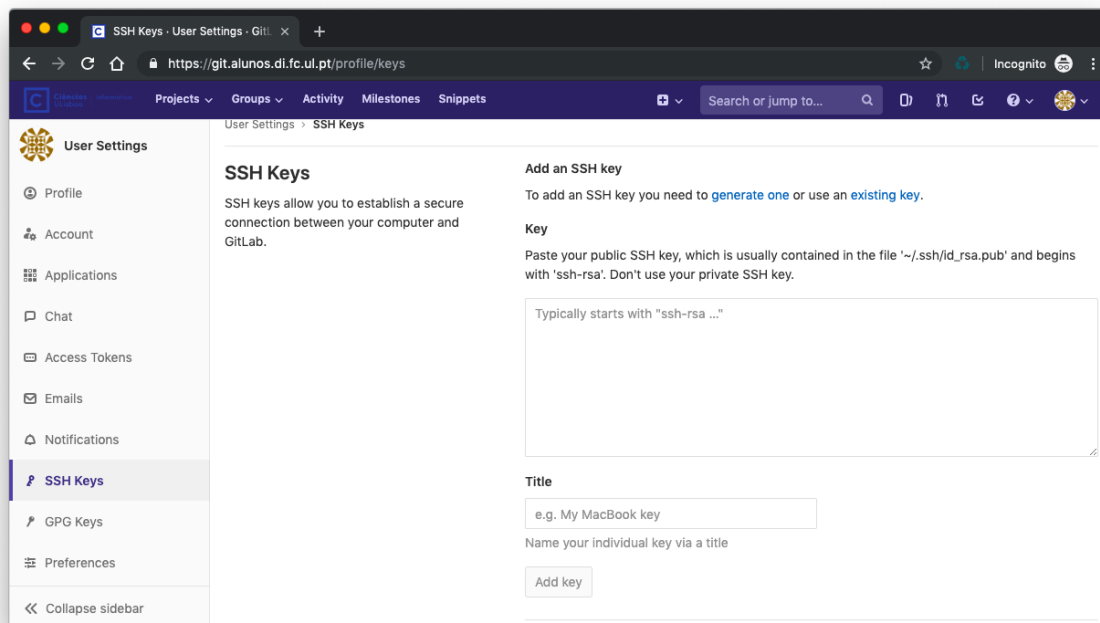
No caso dos alunos do DI, o repositório remoto que iremos usar é o servidor de GitLab instalado nos servidores do departamento (<http://git.alunos.di.fc.ul.pt/>) e que os alunos podem usar com o login e password de aluno desde que dentro da faculdade fisicamente ou com uma ligação VPN.



The screenshot shows a web browser window with the URL https://git.alunos.di.fc.ul.pt/users/sign_in. The page title is "Sign in - GitLab". The main heading is "Servidor de controlo de versões do alunos do Departamento de Informática". Below this, there is a description: "Este servidor destina-se à utilização exclusiva dos alunos do Departamento de Informática da Faculdade de Ciências da Universidade de Lisboa. O acesso ao mesmo é feito com a sua conta de aluno e com a respectiva password." and contact information: "Caso seja necessário suporte devem entrar em contacto com a Administração de Sistemas do Departamento de Informática. suporte@admin.di.fc.ul.pt". On the right, there is a sign-in form with two tabs: "Alunos" (selected) and "Standard". The form fields are "Alunos Username" and "Password". There is a "Remember me" checkbox and a green "Sign in" button. At the bottom, there are links for "Explore", "Help", and "About GitLab".

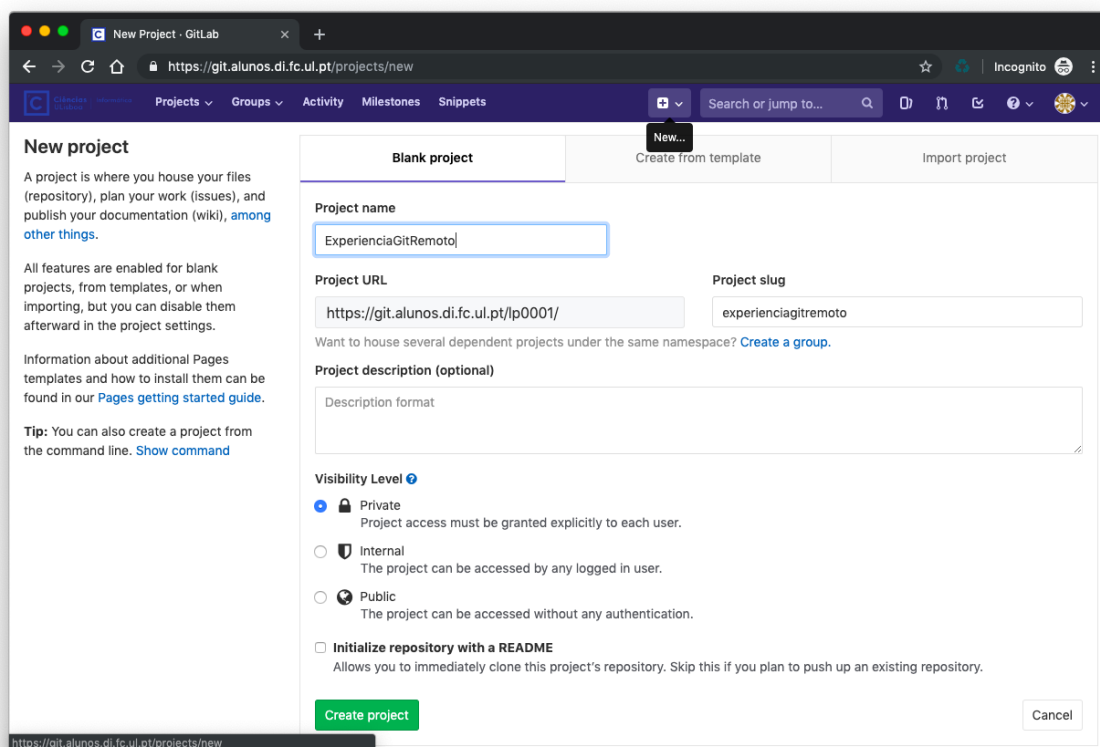
Antes de podermos criar projectos neste repositório é necessário configurar as chaves SSH que identificam os pontos de acesso ao GitLab a partir de uma máquina (esta máquina pode ser própria e/ou dos laboratórios do DI) para que possam aceder ao conteúdo do repositório remotamente.

A configuração é feita em **Profile Settings > SSH keys > Add key** que aparece do lado direito. A página que aparece é a seguinte:

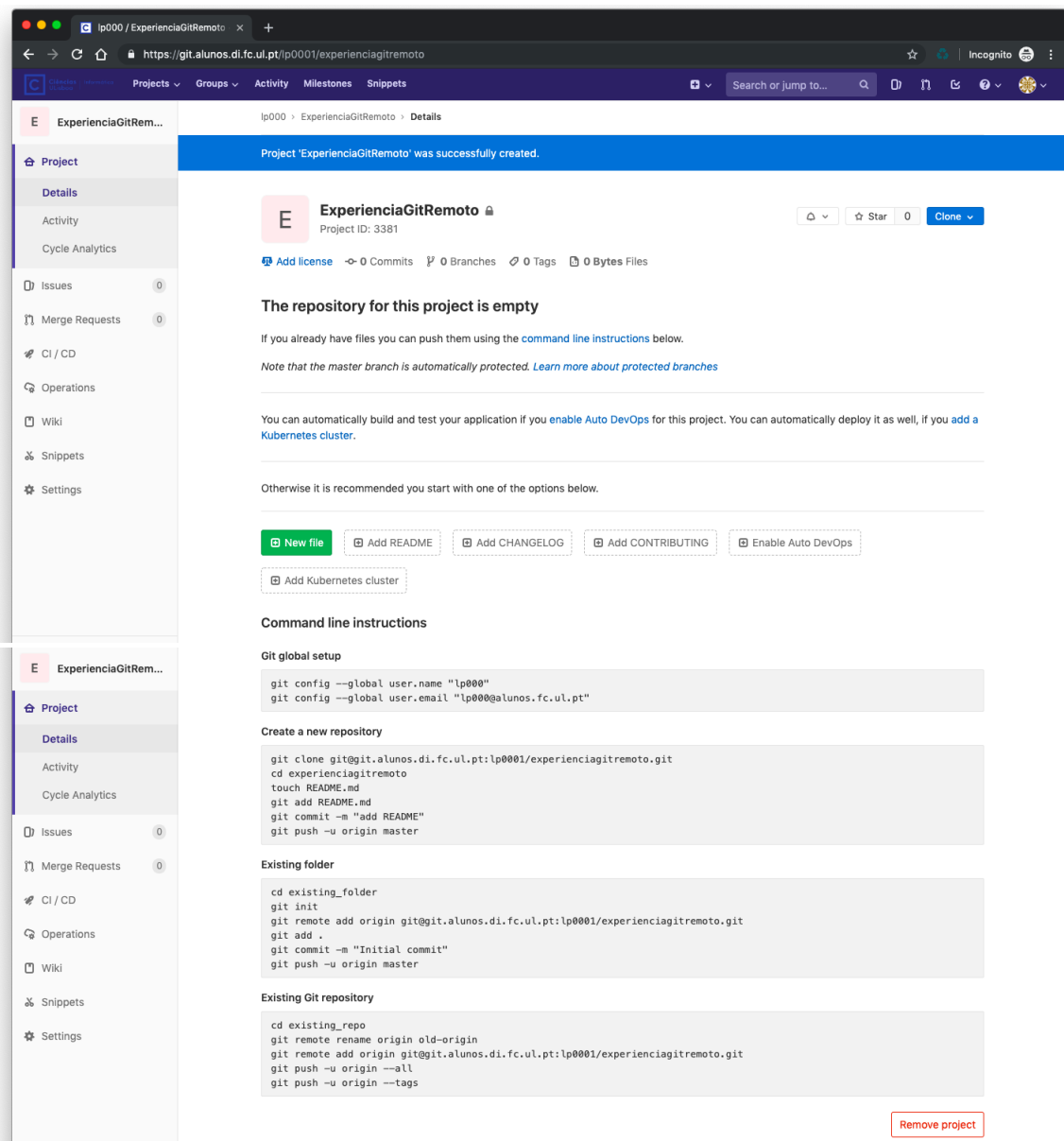


Use a página [SSH Help center](#) que aparece na parte superior da página para gerar e copiar a chave pública RSA que irá usar para aceder localmente aos projectos que a desenvolver neste computador. Depois de gerada a chave copie-a para o espaço reservado à **key** e escreva um título sugestivo.

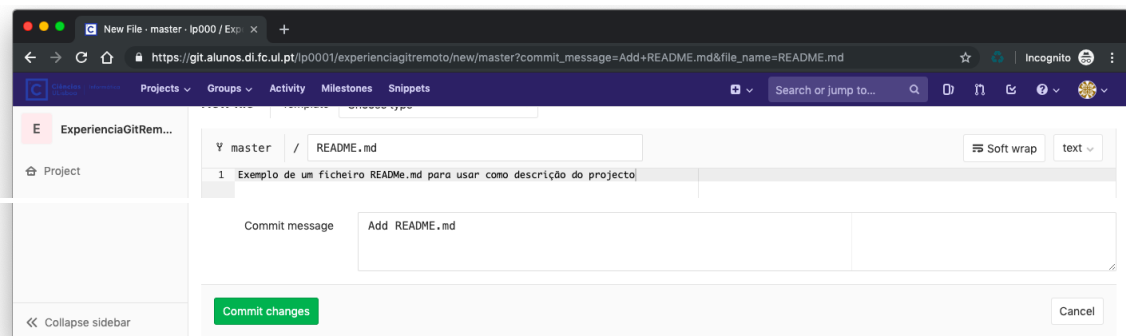
Crie um projeto Git diretamente no servidor



Note a informação que lhe é dada sobre o projecto:



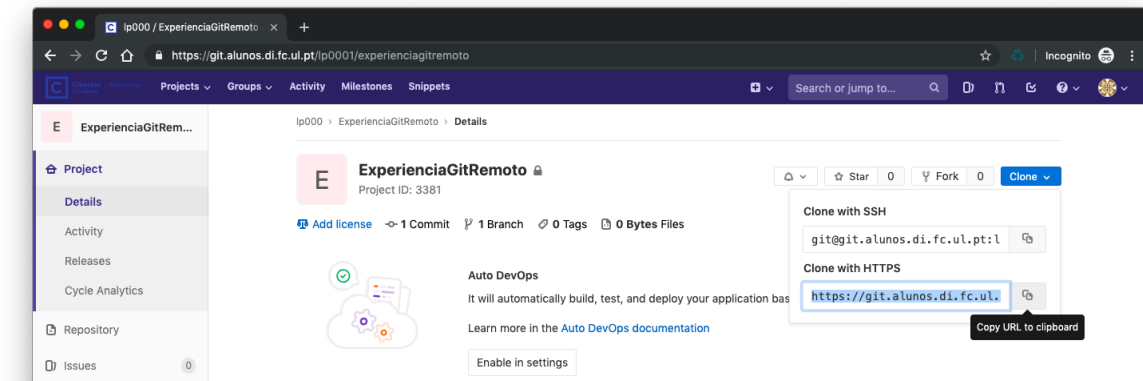
Crie um README diretamente no *browser* para poder clonar imediatamente o repositório na sua máquina.



Clonar (comando *clone*) é a operação de Git que permite criar uma cópia do

repositório Git, neste caso na sua máquina local. Se configurou a chave SSH deve utilizar o URL para clone with SSH

```
> git clone endereço_url
```



Vamos experimentar clonar o repositório que criámos no início desta secção para um repositório local, copiando o URL utilizando o botão *Clone->Copy URL to clipboard*. Para isso execute no seu terminal os comandos que se seguem:

```
> git clone
https://git.alunos.di.fc.ul.pt/lp0001/experienciaGitRemoto.git
Cloning into 'experienciaGitRemoto'...
Username for 'https://git.alunos.di.fc.ul.pt': lp000 (para os alunos
é fcXXXXX)
Password for 'https://lp000@git.alunos.di.fc.ul.pt':
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)
Unpacking objects: 100% (3/3), done.
> ls
experienciaGitRemoto
> cd experienciaGitRemoto/
> ls -al
total 8
drwxr-xr-x  4 ans  staff  136 Feb 10 13:39 .
drwxr-xr-x  6 ans  staff  204 Feb 10 13:38 ..
drwxr-xr-x 13 ans  staff  442 Feb 10 13:39 .git
-rw-r--r--  1 ans  staff   48 Feb 10 13:39 README.md
> git status
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working directory clean
```

Com um servidor remoto e um clone local torna-se necessário distinguir os dois. O Git faz isso usando um ficheiro (designado .git/config) identificando a origem com o

nome *origin*, no caso o repositório no servidor, e a cópia local como no ramo *master*. Esta informação é fundamental para que a sincronização entre o repositório local e o repositório guardado no servidor.

Podemos então criar ficheiros novos no nosso repositório local. Observe os comandos e a informação que é devolvida:

```
> echo "Ola servidor" > testeSinconizacao.txt
> git add testeSinconizacao.txt
> git commit -m "Ficheiro criado localmente para colocar no
servidor"
[master 9050317] Ficheiro criado localmente para colocar no
servidor
 1 file changed, 1 insertion(+)
 create mode 100644 testeSinconizacao.txt
> git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
 (use "git push" to publish your local commits)
nothing to commit, working directory clean
```

Note que o próprio Git informa que existe uma publicação adiantada em relação ao repositório no servidor (o *origin*) e de facto se for ver no browser os ficheiros que lá constam o testeSinconizacao.txt não está lá porque apenas se fez a publicação no repositório local. Para efetivar a publicação na cópia do repositório no servidor é necessário comunicar essa alteração ao servidor através do comando *push*:

```
> git push origin master
Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 327 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To http://git.alunos.di.fc.ul.pt/lp0001/experienciaGitRemoto.git
 867ac89..9050317  master -> master
> git status
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working directory clean
```

É importante frisar que apenas os ficheiros que estão publicados localmente serão “empurrados” para o repositório no servidor. Aqueles que se encontram no *stage* (isto é preparados para publicação) mas não estão publicados, não serão empurrados. Por isso é importante que confirme todas as alterações antes de colocar as alterações no

repositório remoto.

Para “puxar” as últimas atualizações do repositório guardado no servidor deve usar o comando *pull* da origem:

```
> git pull origin
Already up-to-date.
```

Se voltar ao browser e acrescentar um ficheiro direto no repositório, a cópia que tem localmente já não estará actualizada e se repetir novamente o comando tem:

```
> git pull origin
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From http://git.alunos.di.fc.ul.pt/lp0001/experienciaGitRemoto.git
   e3ef63f..e751835  master      -> origin/master
Updating e3ef63f..e751835
Fast-forward
 CHANGELOG | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 CHANGELOG
```

Síntese

A imagem seguinte (crédito à JRebel) sintetiza os comandos *Git* e a interação entres repositórios locais e remotos. O original pode ser obtido em

http://files.zereturnaround.com/pdf/zt_git_cheat_sheet.pdf



Git Cheat Sheet

Create a Repository

From scratch -- Create a new local repository
\$ git init [project name]
Download from an existing repository
\$ git clone my_url

Observe your Repository

List new or modified files not yet committed
\$ git status
Show the changes to files not yet staged
\$ git diff
Show the changes to staged files
\$ git diff --cached
Show all staged and unstaged file changes
\$ git diff HEAD
Show the changes between two commit ids
\$ git diff commit1 commit2
List the change dates and authors for a file
\$ git blame [file]
Show the file changes for a commit id and/or file
\$ git show [commit] : [file]
Show full change history
\$ git log
Show change history for file/directory including diffs
\$ git log -p [file/directory]

Working with Branches

List all local branches
\$ git branch
List all branches, local and remote
\$ git branch -av
Switch to a branch, my_branch, and update working directory
\$ git checkout my_branch
Create a new branch called new_branch
\$ git branch new_branch
Delete the branch called my_branch
\$ git branch -d my_branch
Merge branch_a into branch_b
\$ git checkout branch_b
\$ git merge branch_a
Tag the current commit
\$ git tag my_tag

Make a change

Stages the file, ready for commit
\$ git add [file]
Stage all changed files, ready for commit
\$ git add .
Commit all staged files to versioned history
\$ git commit -m "commit message"
Commit all your tracked files to versioned history
\$ git commit -am "commit message"
Unstages file, keeping the file changes
\$ git reset [file]
Revert everything to the last commit
\$ git reset --hard

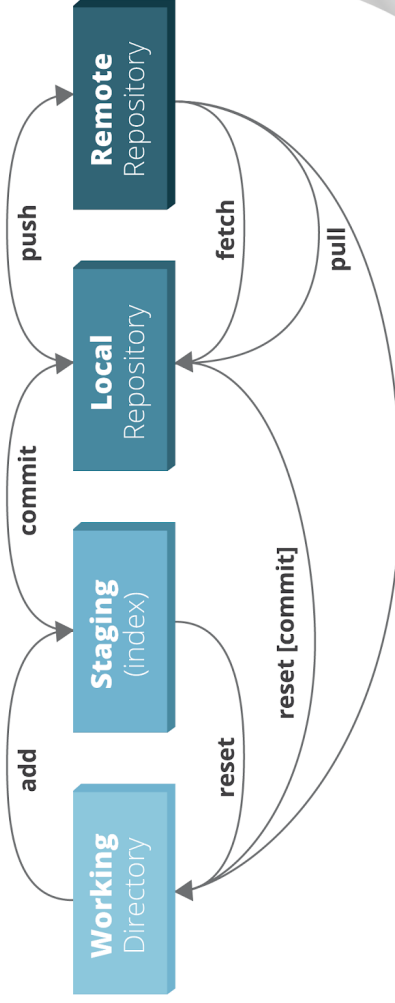
Synchronize

Get the latest changes from origin (no merge)
\$ git fetch
Fetch the latest changes from origin and merge
\$ git pull
Fetch the latest changes from origin and rebase
\$ git pull --rebase
Push local changes to the origin
\$ git push

Finally!

When in doubt, use git help
\$ git command --help

Or visit <https://training.github.com/> for official GitHub training.



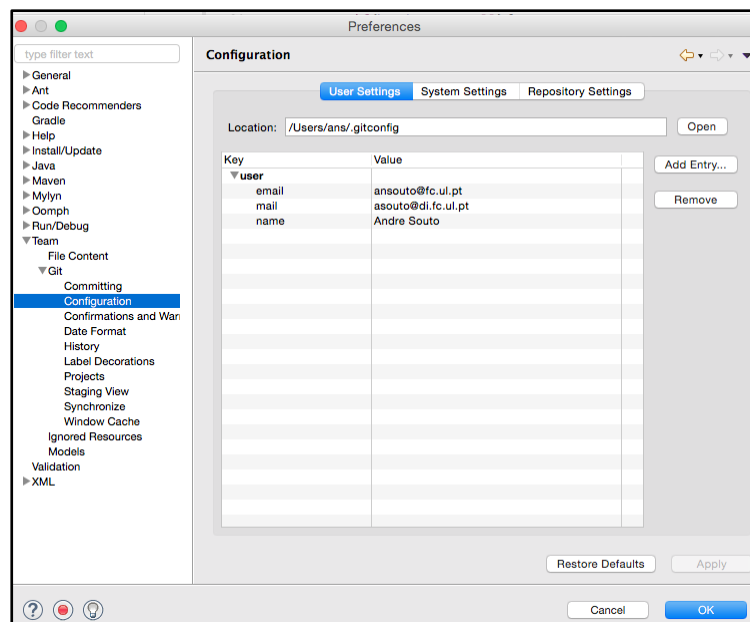
Integração com o Eclipse

Esta secção foca-se no desenvolvimento de código em ambiente Eclipse, descrevendo a forma como podemos integrar o uso de git neste IDE.

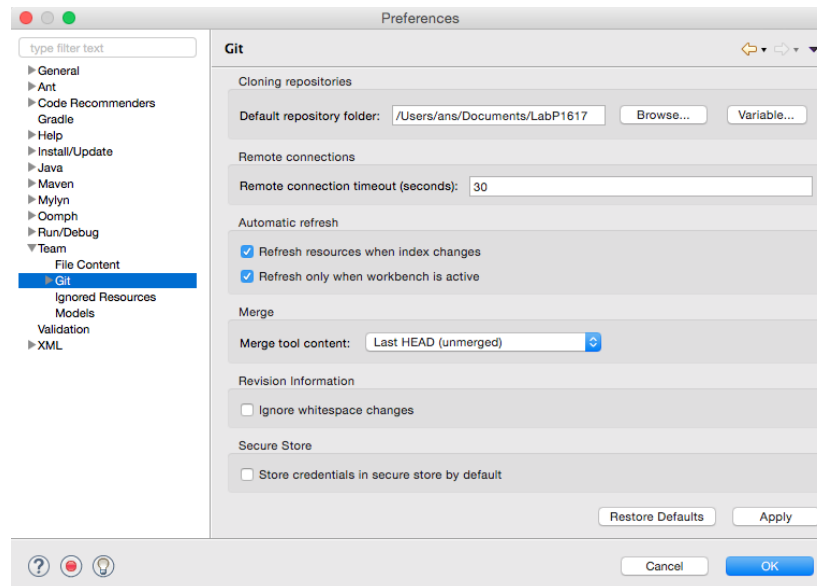
Por norma, a instalação standard de Eclipse traz incorporado o EGit, um *software* de integração de Git com o Eclipse. Por essa razão irá permitir a interação direta entre o *workspace* do Eclipse e o repositório remoto onde irá guardar os seus projectos para trabalhar em locais diferentes.

Antes de começar a trabalhar conjuntamente com o git e o Eclipse é necessário fazer as configurações adequadas para que tudo funcione corretamente.

Primeiro, é necessário configurar o nome e o email a usar no Git, informação que será utilizada para informar sobre os *commits* feitos. Se não fez esta configuração via linha de comandos ou se precisa confirmar alterações, pode no Eclipse ir ao menu **Window** (dependendo do sistema operativo) > **Preferences** > **Team** > **Git** > **Configuration** para ver a informação mencionada acima e corrigi-la caso seja necessário.



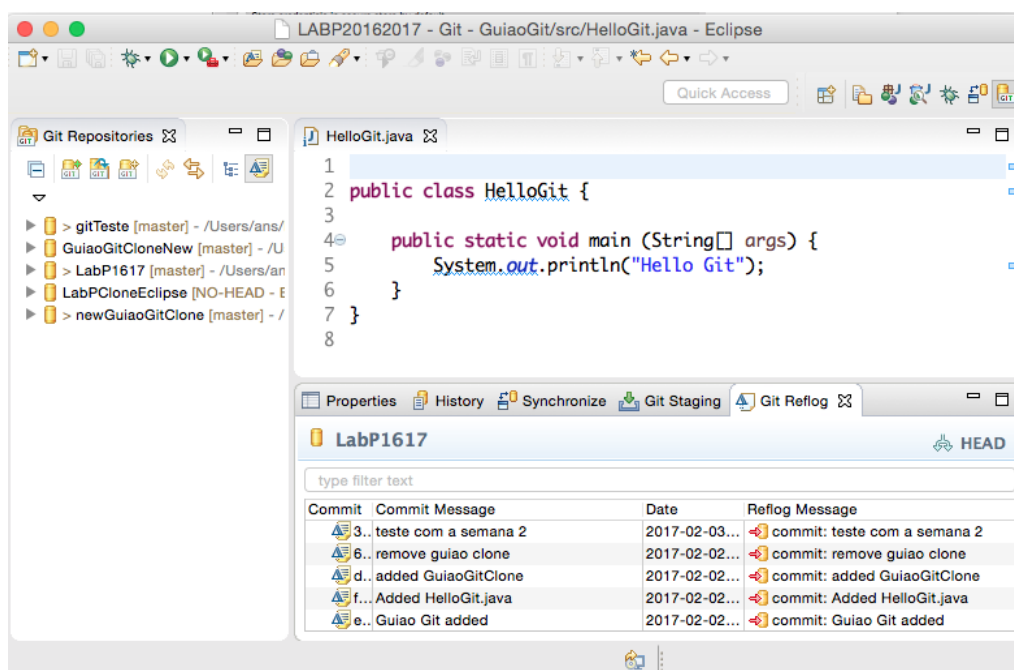
Outro ponto importante é o local padrão do repositório local Git. No exemplo da figura é o diretório **/Users/ans/Documents/LabP1617**



Assim, se clonar um novo repositório via Eclipse Git, será criado um subdiretório para o novo repositório Git no diretório dado como *default*.

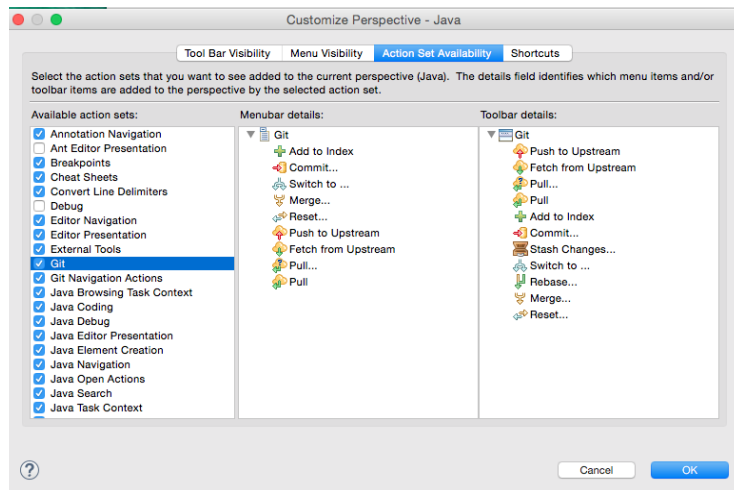
Perspectiva Git

Para melhor visualizar as alterações referentes aos repositórios pode abrir a perspectiva Git através de **Window > Open Perspective > Other > Git**.

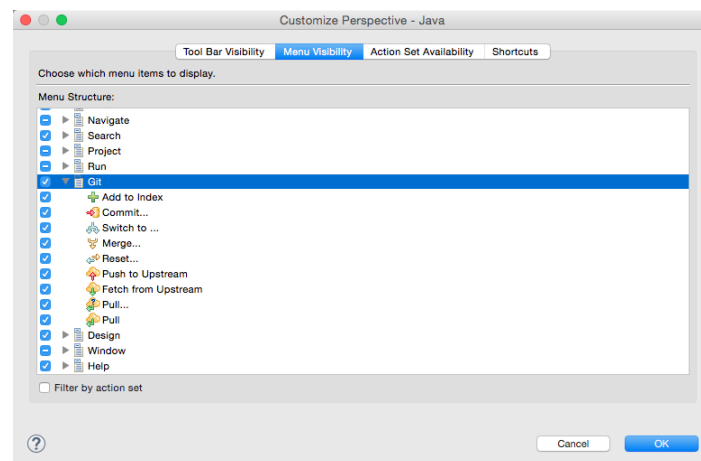


No nosso caso, optamos por continuar com a perspectiva de java e acrescentar parte da perspectiva Git. De modo a simplificar o acesso às operações simples de Git deverá ativar a barra de Git através de **Window > Perspective > Customize perspective e**:

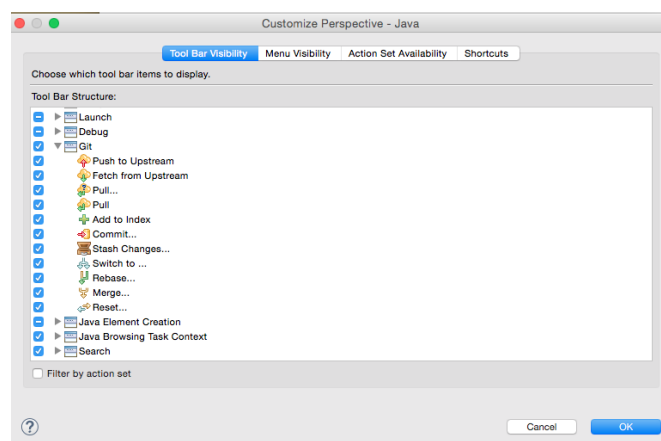
1. No separador **Action Set Available** activar o Git



2. no separador **Menu Visibility** activar o Git



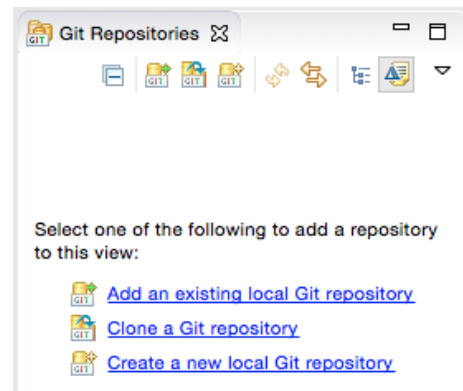
3. no separador **Tool Bar Visibility** activar o Git



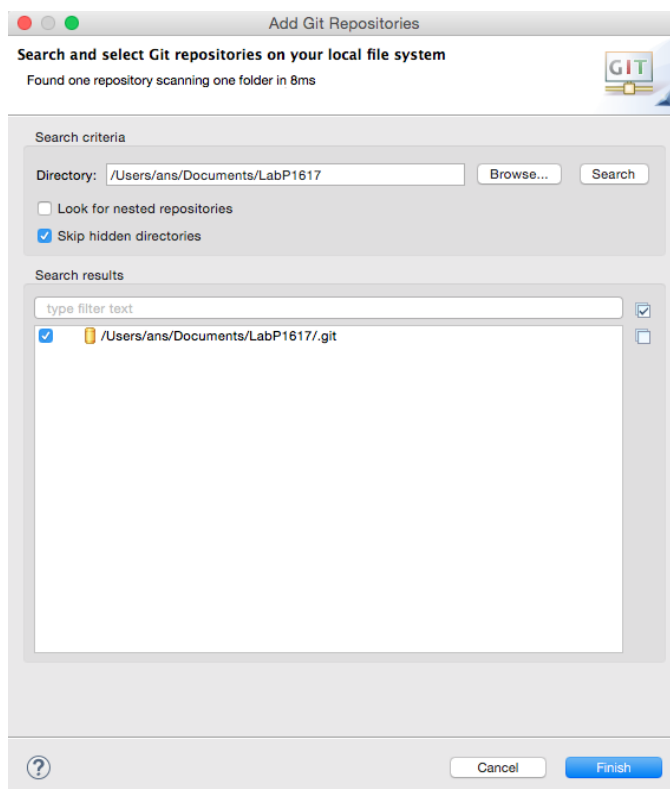
No menu **Windows > Show View > other** pode adicionar as visualizações das abas supramencionadas de Git.

Criar um projecto Java para colocar no Repositório Remoto

Dado que já temos o repositório local ligado ao repositório remoto, vamos começar por colocá-lo acessível no Eclipse através de **Add an existing local repository** acessível na perspectiva Git como mostra a figura.

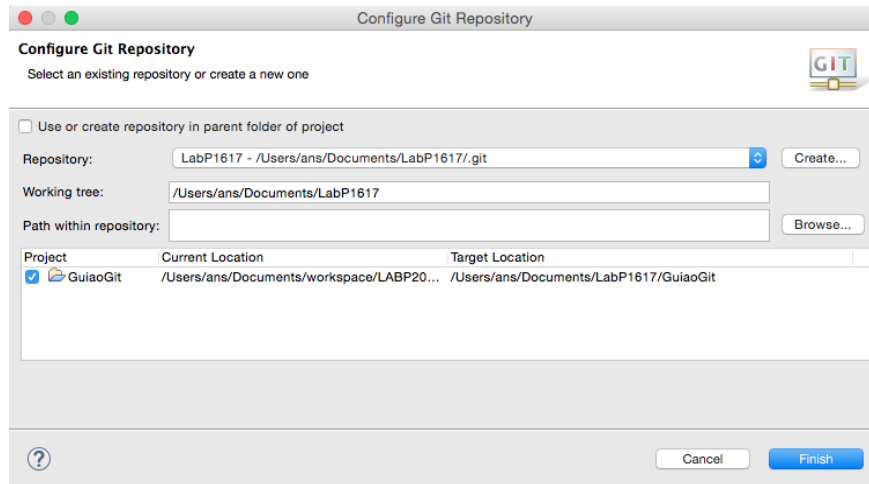


Agora é só localizar o nosso repositório:



No separador de **Git Repositories**, vemos então o repositório associado. Falta associar um projecto Java no nosso *workspace* a este repositório para que mais tarde possa ser acedido a partir de outra máquina.

Crie um projecto Java com o nome GuiaoGit e com o botão direito vá a **Team > Share Project** e preencha os campos como indicado.




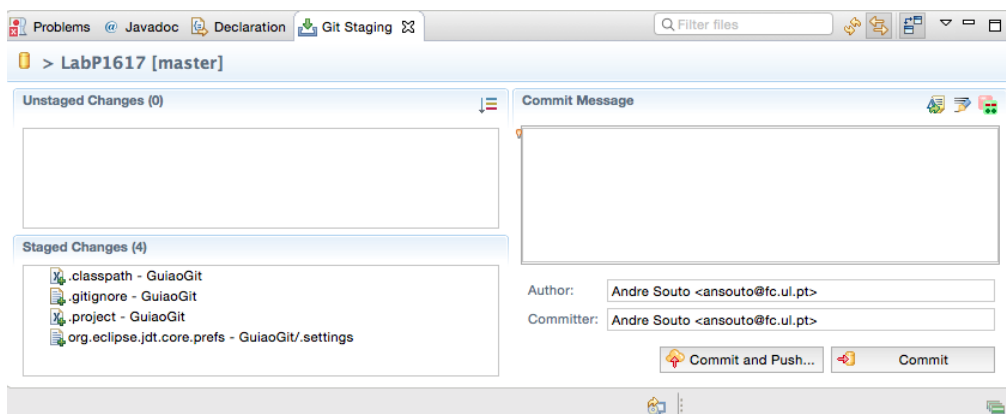
Deste modo o projecto Java fica associado ao repositório de projetos git que criamos no repositório remoto GitLab dos alunos.

Podemos então começar por usar os botões próprios de EGit, push, fetch, specified pull, pull, add commit.



Os outros botões para já não serão discutidos. Deixa-se ao cuidado do aluno explorar e perceber para que servem.

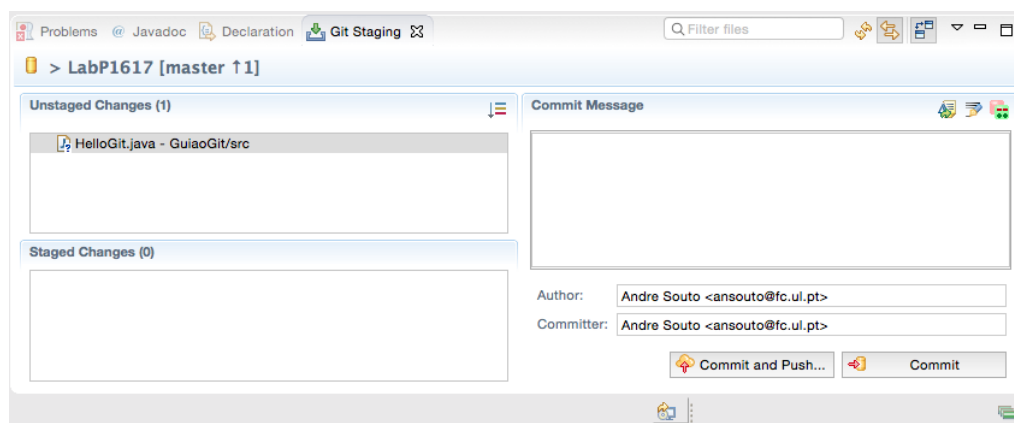
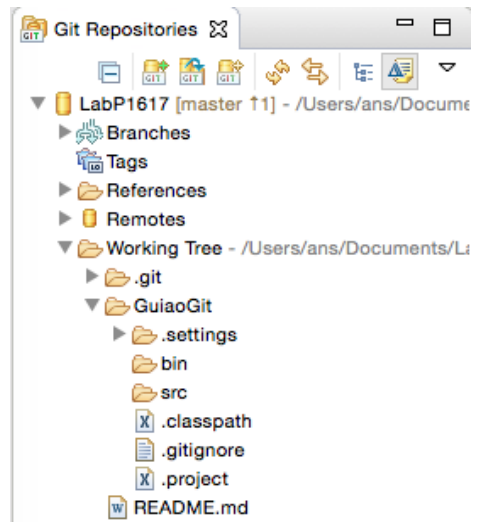
Selecione o projecto e clique em  para fazer *add* e depois *commit*. Note que no separador de Git Staging (que possivelmente se abriu ou que já estava integrada na visualização) aparece um conteúdo preparado para ser enviado para :



Escrever uma mensagem de *commit* “Project GuiaoGit added”. Na vista de repositório aparece então o *commit* feito. E no servidor? Não, porque o *commit* é local e só fica no repositório local.

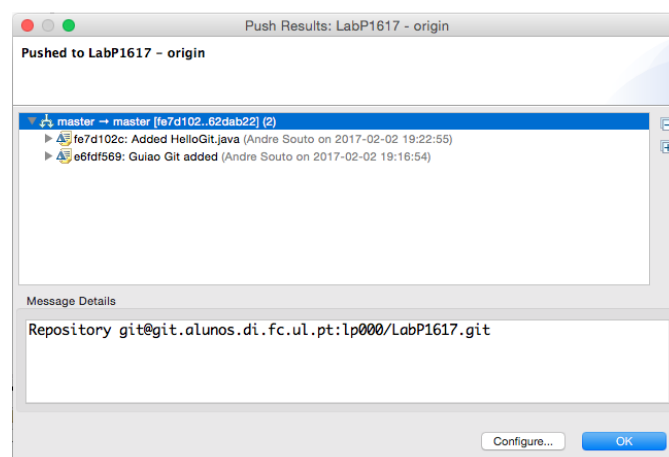
Crie agora uma classe `HelloGit` que imprime uma frase do tipo “Hello Git!”.

Note que agora surgem pontos de interrogação assinalando a existência de alterações que ainda não foram feitas no repositório local. Note que na vista de *Git Staging* a alteração que fizemos (criámos o ficheiro `HelloGit.java`) aparece *unstaged*, como mostra a imagem.



Podemos arrastar este ficheiro para *staged changes* ou adicionar com o .

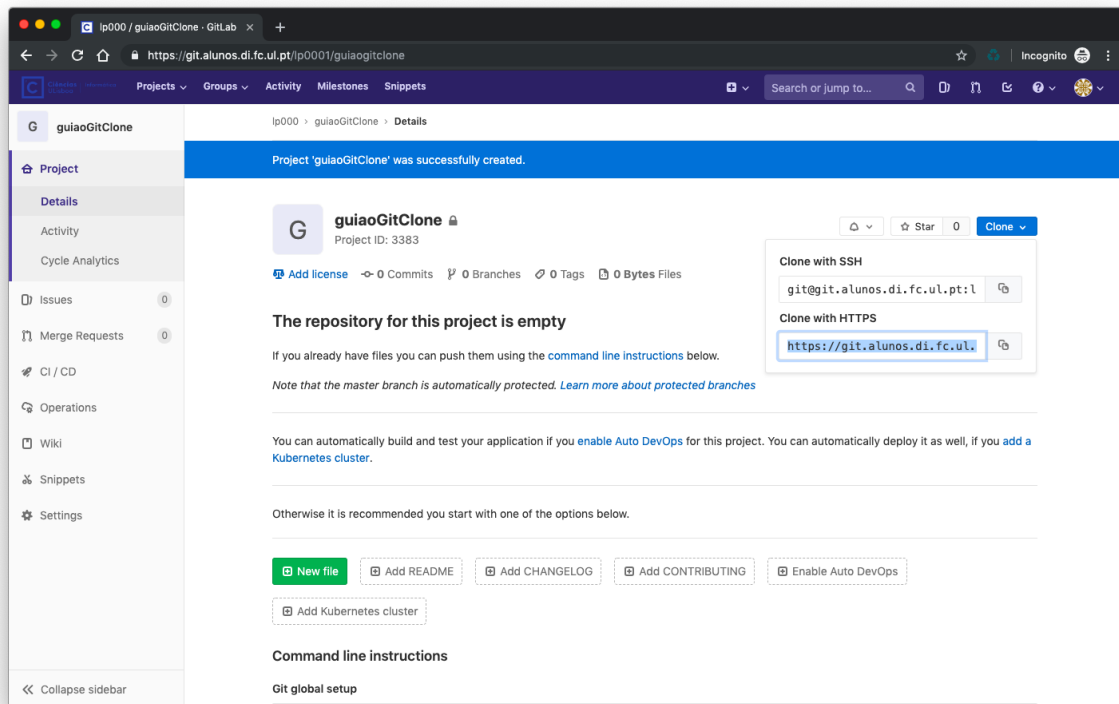
Se carregar em *commit and push* obtém, depois de introduzir a palavra passe, a seguinte informação:



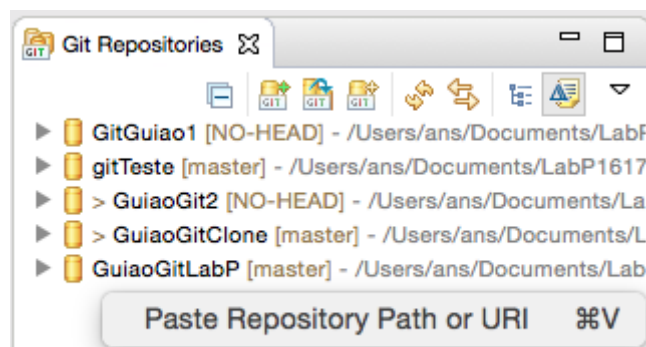
que significa que acrescentou ao repositório remoto todos os ficheiros com os quais fez *commit*.

Importar ou clonar um projecto já existente

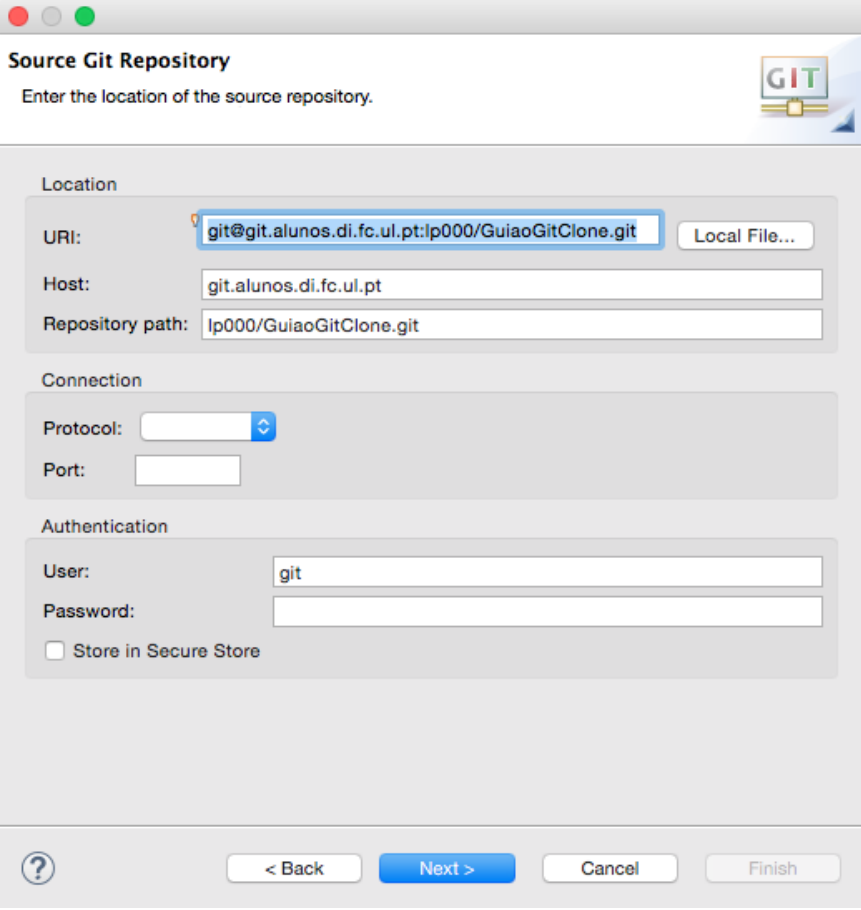
Para importar ou clonar um projeto já existente podemos proceder de duas maneiras distintas. A mais simples é copiando o endereço do projecto diretamente do *browser*:



e com o botão direito do rato sobre o separador de Git Repositories colar o caminho ou URI diretamente como sugere a figura abaixo:



Também se pode recorrer à vista do Git Repositories *Clone a Git Repository* ou então ao menu **file > import > Git > Projects from git** e escolhendo a opção *clone URI*. Aparecem os campos já preenchidos.



Source Git Repository
Enter the location of the source repository.

Location

URI:

Host:

Repository path:

Connection

Protocol:

Port:

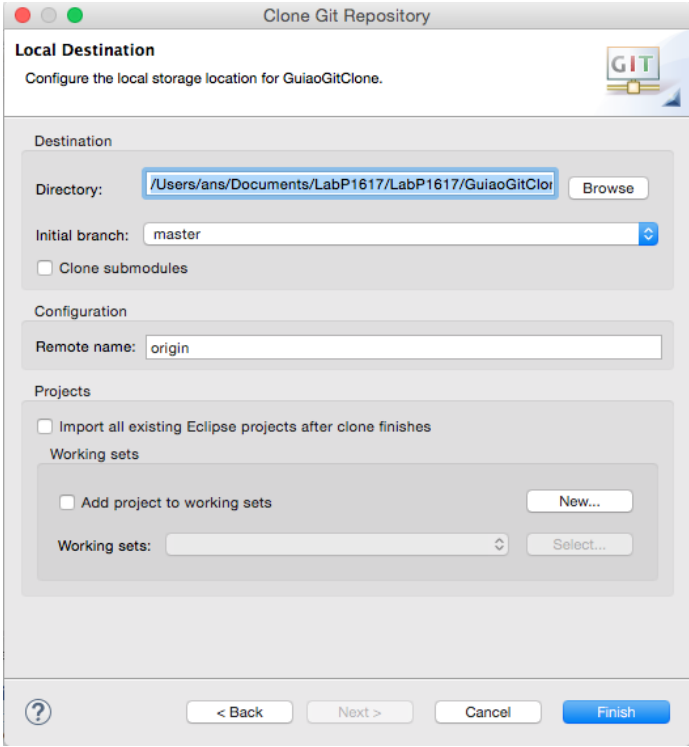
Authentication

User:

Password:

☐ Store in Secure Store

Clicando em Next (e possivelmente depois de preenchidas as credenciais de acesso ao repositório remoto) escolhe-se o local onde se quer clonar o repositório.



Clone Git Repository
Configure the local storage location for GuiaoGitClone.

Destination

Directory:

Initial branch:

☐ Clone submodules

Configuration

Remote name:

Projects

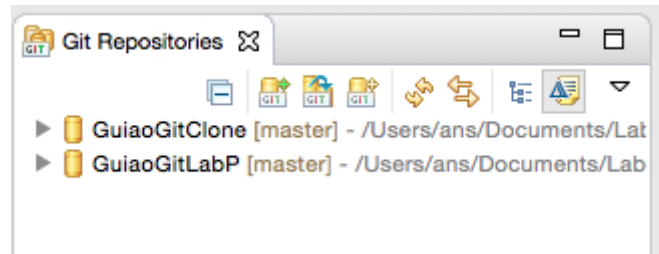
☐ Import all existing Eclipse projects after clone finishes

Working sets

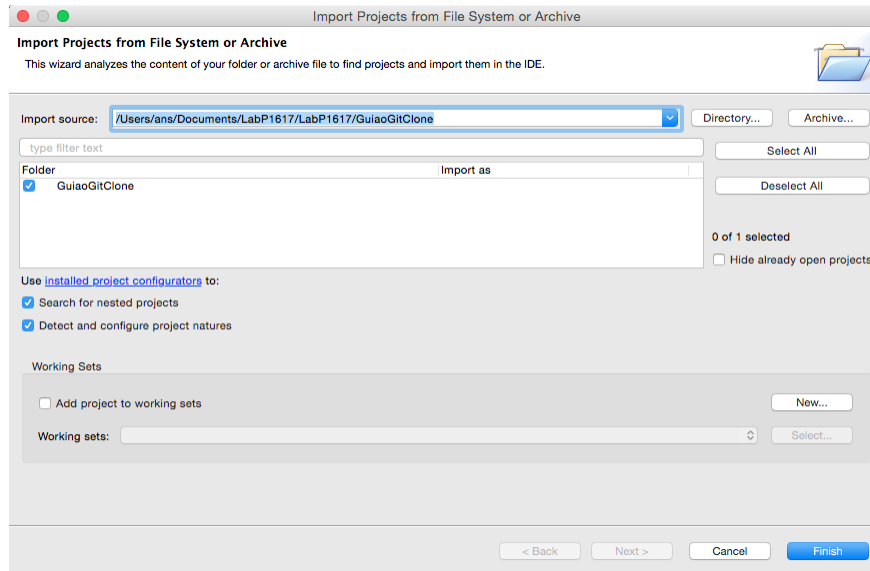
☐ Add project to working sets

Working sets:

Agora no separador *Git Repositories* aparece o novo repositório:



Para importarmos o seu conteúdo basta clicar com o botão direito sobre o repositório e escolher a opção **Import Project**.



Uso do Histórico

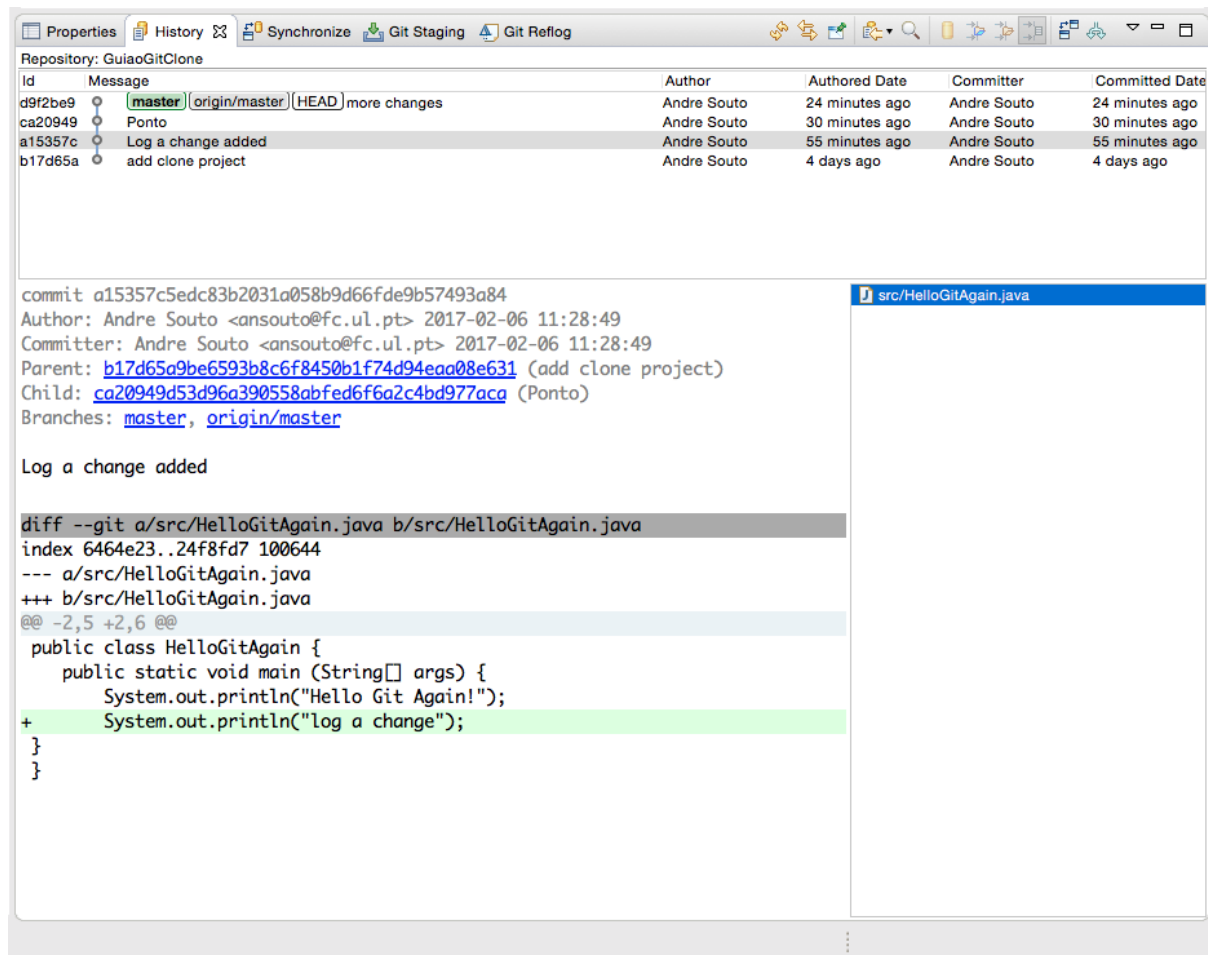
A grande vantagem do uso do Git é podermos ver o histórico das alterações feitas.

Comece por adicionar a linha de código

```
System.out.println("log a change");
```

ao *HelloGitAgain.java* e faça o *add*, *commit* e *push* com a mensagem **log a change added**.

Para consultar o histórico, mude a perspectiva para a perspectiva Git e consulte o separador *Git RefLog* ou com o botão direito sobre o repositório veja a opção **Show in > History**.



Observe que aparecem as informações sobre os *commits* feitos (id, mensagem, autor, data), sobre os ficheiros que foram alterados e clicando no próprio ficheiros as diferenças que foram registadas. Estas diferenças podem ser realçadas através do botão direito e opção **compare to previous versions**.

Pode voltar a um ponto anterior de compromisso e desfazer um *commit* basta usar o histórico e seleccionar o ponto para o qual se quer reverter o repositório e com o botão direito seleccionar a opção **reset > hard (Head, index and working tree)**.

Este guião apresenta algumas possibilidades relativas ao uso do Git. Podem encontrar informação adicional em várias páginas Web, por exemplo, em:

[Learn Git Branching](#) - jogo interativo que ensina os conceitos fundamentais do git

ou

[Git - Book](#)

ou

[Git Cheat Sheet: Commands and Best Practices | JRebel](#)