

Introdução ao git em repositórios locais

- Qualquer dúvida enviar email a: fas@estg.ipp.pt

Documentação complementar Git:

- <https://learngitbranching.js.org/>
- <https://git-scm.com/book/en/v2>

1 Git

1.1 Introdução

Para o desenvolvimento da ficha prática 6, deve ser utilizado um sistema de controlo de versões, o GIT, para registar todas as alterações efectuadas num projeto e promover o trabalho colaborativo em equipa.

O Git é um sistema de controlo de versões distribuído que auxilia o processo de desenvolvimento de software. Estes sistemas, guardam as alterações efetuadas num ou mais ficheiros ao longo do tempo num repositório. Com esta ferramenta é possível registar o histórico de versões/alterações do software a desenvolver. Permite também voltar a versões anteriores de cada ficheiro com base num registo de alterações guardados no repositório.

Um repositório git é representado por uma pasta de trabalho no seu computador local onde existe uma subpasta com o nome “.git” que guarda toda a informação relacionada com o histórico do repositório. Um repositório git pode ser replicado em vários computadores ou servidores com recurso a comandos executados por clientes git que além de partilhar informação do repositório, gerem o trabalho colaborativo de uma equipa utilizando comandos do tipo commit no repositório local para guardar alterações locais e comandos do tipo pull e push para comunicar/obter alterações a versões do mesmo repositório em computadores externos. Um exemplo gráfico da representação de um repositório em vários computadores pode ser encontrado na Figura 1.

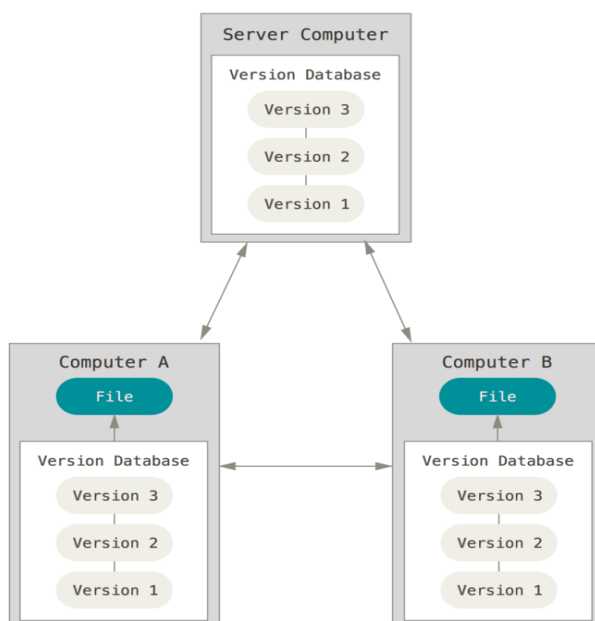


Figura 1 - Repositório git replicado em vários locais

1.2 Instalação

Antes de proceder ao uso da ferramenta git deve instalar um client git no seu computador local. Para este efeito deve obter um cliente de git compatível com o seu sistema operativa em:

- <https://git-scm.com/>



Figura 2 - git-scm.com

Deve proceder à instalação usando as definições por padrão sem alterações.

2 Uso do Git através da linha de comandos

2.1 Criar um repositório git local

Para demonstração iremos usar um simples projeto Hello World em java na plataforma netbeans para exemplificar os conceitos de repositório local. Desta forma deve previamente criar um projeto com estas características no seu editor Netbeans de acordo com a Figura 3.

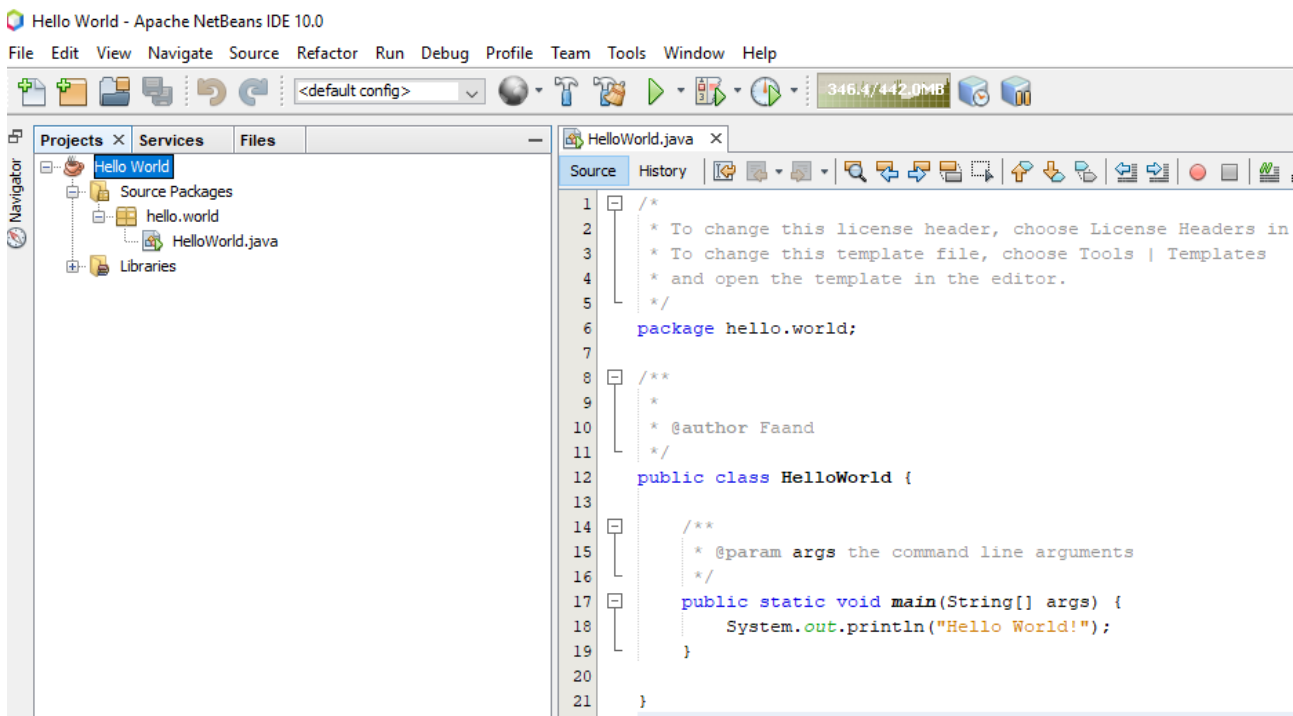


Figura 3 - Projeto Hello World

Para criar um repositório git manualmente é apenas necessário executar o comando:

- "git init"

Estes comandos devem ser executados no terminal/linha de comandos na pasta onde o seu projeto netbeans

se encontra instalado. Um exemplo de como abrir a linha de comandos utilizando windows encontra-se na Figura 4. Este comando irá criar uma nova subpasta “.git” que pode estar oculta por defeito no seu sistema operativo. Utilizando o comando “ls -a” em linux ou “dir” em windows é possível verificar a sua existência.

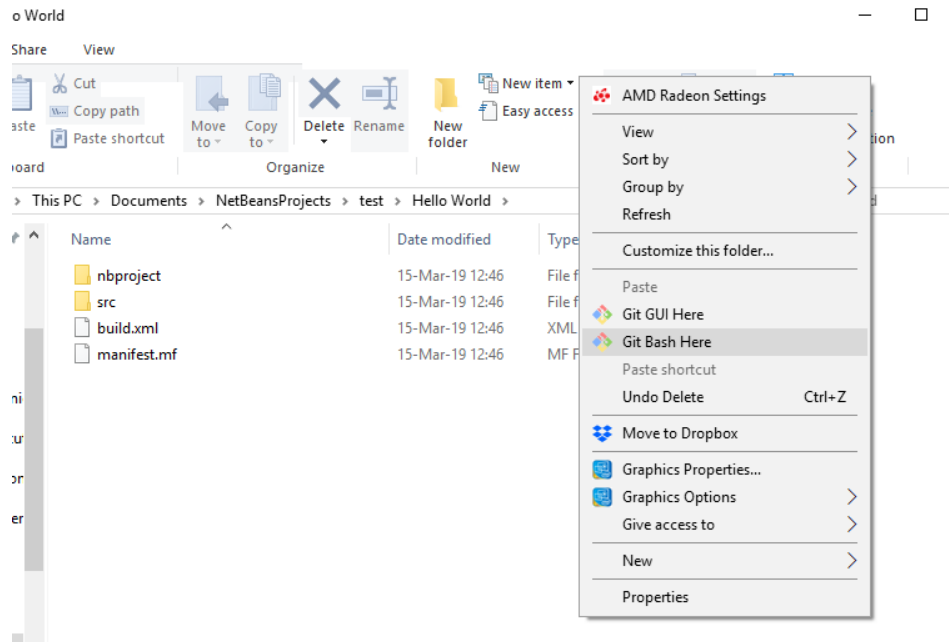


Figura 4 - Abrir linha de comandos git em windows

```
Faand@FAB MINGW64 ~/Documents/NetBeansProjects/test/Hello World
$ git init
Initialized empty Git repository in C:/Users/Faand/Documents/NetBeansProjects/test/Hello World/.git/
```

Figura 5 - Consola git após inicialização do comando git init

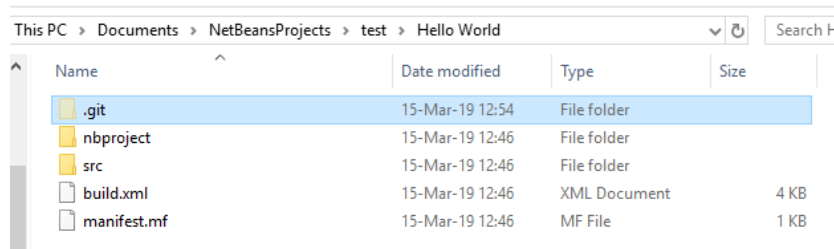


Figura 6 - Alterações na pasta do projeto

Após criado um repositório é preciso associar ficheiros com presentes na pasta com o repositório git. Isto pode ser feito em duas fases. Em primeiro lugar iremos indicar os ficheiros que queremos incluir no repositório. Podemos fazer-lo com os comandos:

- “git add nome_ficheiro” – adiciona o ficheiro “nome do ficheiro.java” ao repositório git
- “git add src/*” – adiciona todos os ficheiros dentro da pasta “src” ao repositório git
- “git add *” – adiciona todos os ficheiros da pasta ao repositório git

Esta etapa é conhecida como staging, ou seja ou ficheiros estão marcados mas ainda não estão no repositório. Para adicionar os ficheiros em staging ao repositório é necessário utilizar o comando:

- “git commit -m ‘primeiro commit’ ” – comando que adiciona todos os ficheiros na versão atual ao repositório. Caso um ficheiro já exista no repositório ele será substituído pela nova versão, contudo o histórico desse ficheiro será guardado

```

Faand@FAB MINGW64 ~/Documents/NetBeansProjects/test/Hello World (master)
$ git add *

Faand@FAB MINGW64 ~/Documents/NetBeansProjects/test/Hello World (master)
$ git commit -m 'primeiro commit'
[master (root-commit) 6f567f9] primeiro commit
8 files changed, 1985 insertions(+)
create mode 100644 build.xml
create mode 100644 manifest.mf
create mode 100644 nbproject/build-impl.xml
create mode 100644 nbproject/genfiles.properties
create mode 100644 nbproject/private/private.properties
create mode 100644 nbproject/project.properties
create mode 100644 nbproject/project.xml
create mode 100644 src/hello/world/HelloWorld.java

```

Figura 7 - Execução dos comandos "git add" e "git commit"

Podemos agora verificar o estado do nosso repositório com o comando:

- "git log" – indica o estado do repositório, o seu último commit indentificado pela expressão HEAD e o histórico de alterações. Para cada commit está também disponível o hash do commit que funciona como identificador único do commit. (ex. na Figura 8 o hash do primeiro commit é 6f567f969a0b8a0168709e735bca44cf76cdf4d9)

```

Faand@FAB MINGW64 ~/Documents/NetBeansProjects/test/Hello World (master)
$ git log
commit 6f567f969a0b8a0168709e735bca44cf76cdf4d9 (HEAD -> master)
Author: fabio <fabiosilva@di.uminho.pt>
Date: Fri Mar 15 12:57:11 2019 +0000

    primeiro commit

```

Figura 8 - Execução do comando "git log"

2.2 Registrar Alterações num repositório local

Podemos registar alterações num repositório local com uma combinação de comandos "git add" e "git commit". Estes comando irão respectivamente adicionar ficheiros ao staging do repositório e fazer commit das alterações para o repositório. Após cada commit, o HEAD do repositório é modificado para apontar para o hash do commit mais recente.

Podemos agora no nosso projeto efetuar uma simples alteração como por exemplo traduzir a mensagem "Hello World" para português e executar os comandos add e commit no nosso repositório.

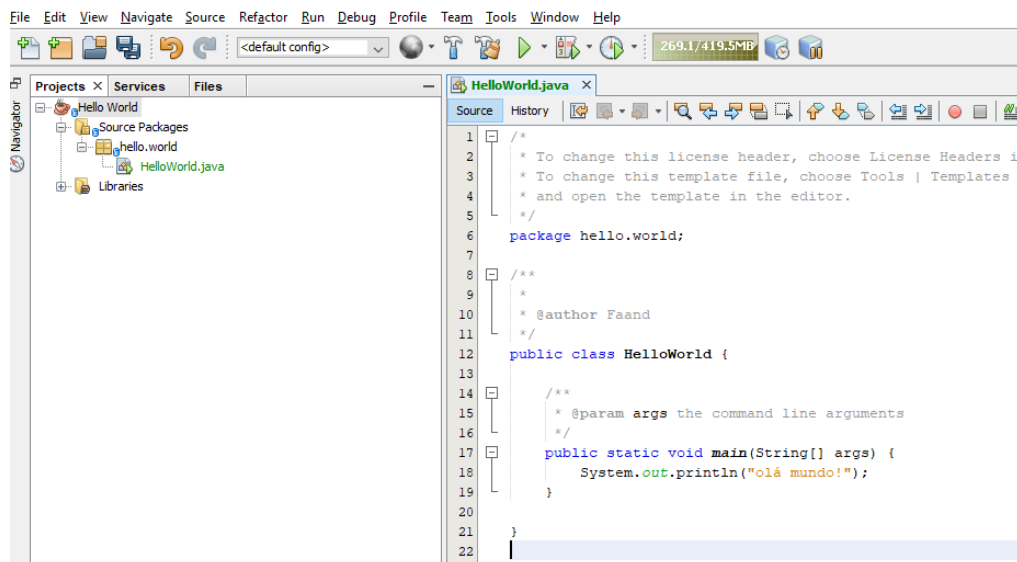


Figura 9 - Alterações ao projeto Hello World

```

Faand@FAB MINGW64 ~/Documents/NetBeansProjects/test/Hello World (master)
$ git add *

Faand@FAB MINGW64 ~/Documents/NetBeansProjects/test/Hello World (master)
$ git commit -m 'segundo commit'
[master 183b02c] segundo commit
1 file changed, 1 insertion(+), 1 deletion(-)

Faand@FAB MINGW64 ~/Documents/NetBeansProjects/test/Hello World (master)
$ git log
commit 183b02c2868b0232046e85439c6c00c4687c3bfb (HEAD -> master)
Author: fabio <fabiosilva@di.uminho.pt>
Date:   Fri Mar 15 13:13:53 2019 +0000

    segundo commit

commit 6f567f969a0b8a0168709e735bca44cf76cdf4d9
Author: fabio <fabiosilva@di.uminho.pt>
Date:   Fri Mar 15 12:57:11 2019 +0000

    primeiro commit

```

Figura 10 - Execução dos comandos add, commit e log

2.3 Reverter alterações no repositório local

Por vezes, é necessário voltar a uma versão anterior do nosso projeto. Com a ferramenta git, este processo é simplificado pois o repositório guarda todo o histórico dos nossos ficheiros.

Nesta situação o caso mais simples será reverter o projeto para um estado/commit anterior. Para executar esta tarefa teremos de usar um conjunto de passos:

- `git checkout hash_do_commit` – comando que indica que queremos voltar ao commit com o hash indicado no comando. Atenção que este comando não será suficiente para eliminar as alterações que já estão efectuadas no projeto, apenas coloca o HEAD do repositório neste commit.
- `git reset --hard` – elimina todas as alterações que estão além do commit referenciado pelo HEAD. Assim, se o HEAD estiver a apontar para um commit anterior ao último commit, todas as alterações posteriores a este commit serão revertidas.

Considere a alteração ao projeto documentada na Figura 11, após um terceiro commit.

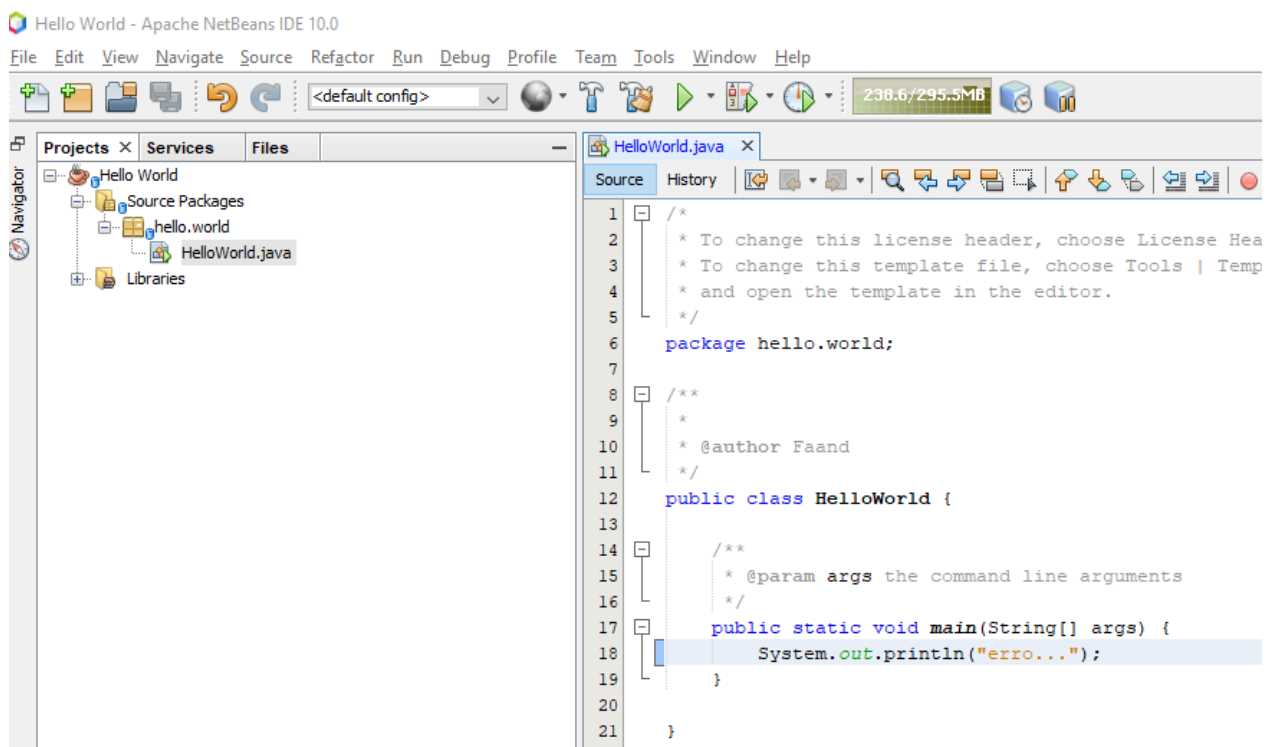


Figura 11 - Estado do projeto após um terceiro commit

Uma vez feito o commit da alteração verificou-se que a mensagem de erro era inadequada e por isso é

necessário reverter o projeto a um momento anterior. Com esse efeito, na Figura 12, está documentada a sequência de comandos a seguir na consola git, para retroceder o projeto ao estado do terceiro commit.

A Figura 13, documenta o estado do projeto final, com o seu estado recuperado .

```
Faand@FAB MINGW64 ~/Documents/NetBeansProjects/test/Hello World (master)
$ git log
commit 25a49d7eebae623ee50878562927c4b3e053ddce (HEAD -> master)
Author: fabio <fabiosilva@di.uminho.pt>
Date:   Fri Mar 15 13:49:03 2019 +0000

    terceiro commit

commit 183b02c2868b0232046e85439c6c00c4687c3bfb
Author: fabio <fabiosilva@di.uminho.pt>
Date:   Fri Mar 15 13:13:53 2019 +0000

    segundo commit

commit 6f567f969a0b8a0168709e735bca44cf76cdf4d9
Author: fabio <fabiosilva@di.uminho.pt>
Date:   Fri Mar 15 12:57:11 2019 +0000

    primeiro commit

Faand@FAB MINGW64 ~/Documents/NetBeansProjects/test/Hello World (master)
$ git checkout 183b02c2868b0232046e85439c6c00c4687c3bfb
Note: checking out '183b02c2868b0232046e85439c6c00c4687c3bfb'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

    git checkout -b <new-branch-name>

HEAD is now at 183b02c segundo commit

Faand@FAB MINGW64 ~/Documents/NetBeansProjects/test/Hello World ((183b02c...))
$ git reset --hard
HEAD is now at 183b02c segundo commit
```

Figura 12 - Comandos a executar para recuperar o projeto até ao segundo commit

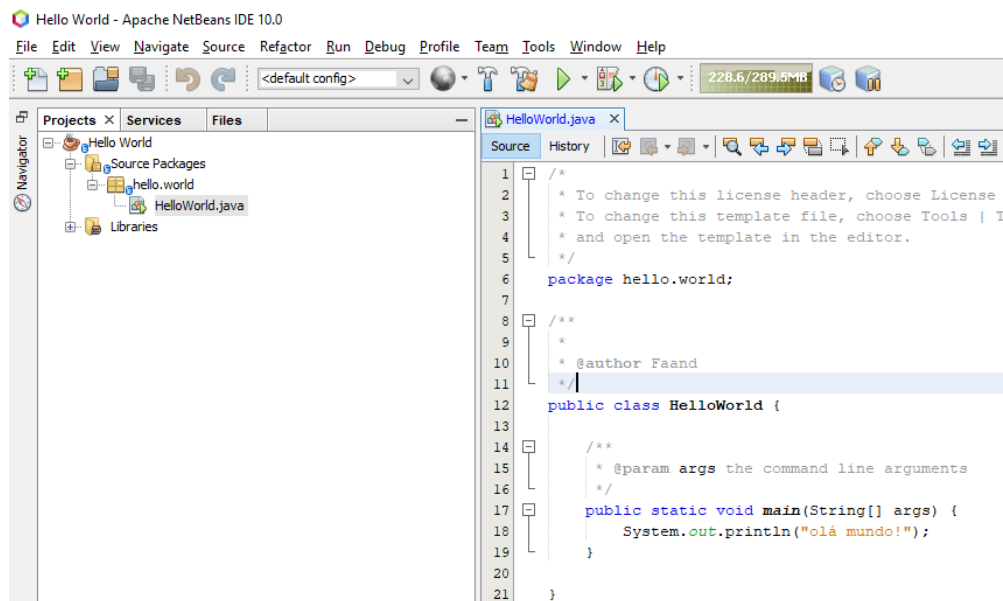


Figura 13 - Estado do projeto após recuperar o projeto para o segundo commit

Devemos após este procedimento efectuar um novo staging e commit com as correções ao projeto, para que o git assuma a recuperação do projeto.

2.4 Clonar um repositório git disponível online

É possível também clonar um repositório em plataformas da internet como por exemplo o github.com directamente para uma pasta local do nosso computador. O processo é relativamente simples bastando apenas efectuar o comando:

- “git clone url_exemplo” – em que url_exemplo será o endereço do repositório git de onde queremos clonar o nosso repositório.

Este comando deve ser efectuado na pasta onde queremos guardar o projeto.

Atenção, clonar um repositório significa também clonar todo o histórico do repositório, ou seja, a sua versão actual e informação sobre commit anteriores. Isto significa que se ao copiar um repositório através do comando “git clone” temos acesso as alterações efectuadas nesse repositório por outros utilizadores com acesso.

Após o comando efectuado ficaremos com um repositório local, para trabalhar localmente, tal e qual como um repositório criado com o comando “git init”

Do lado do IDE netbeans, apenas necessitamos de criar um projeto com “existing sources” para importar o projeto e trabalhar nele usando o Netbeans.

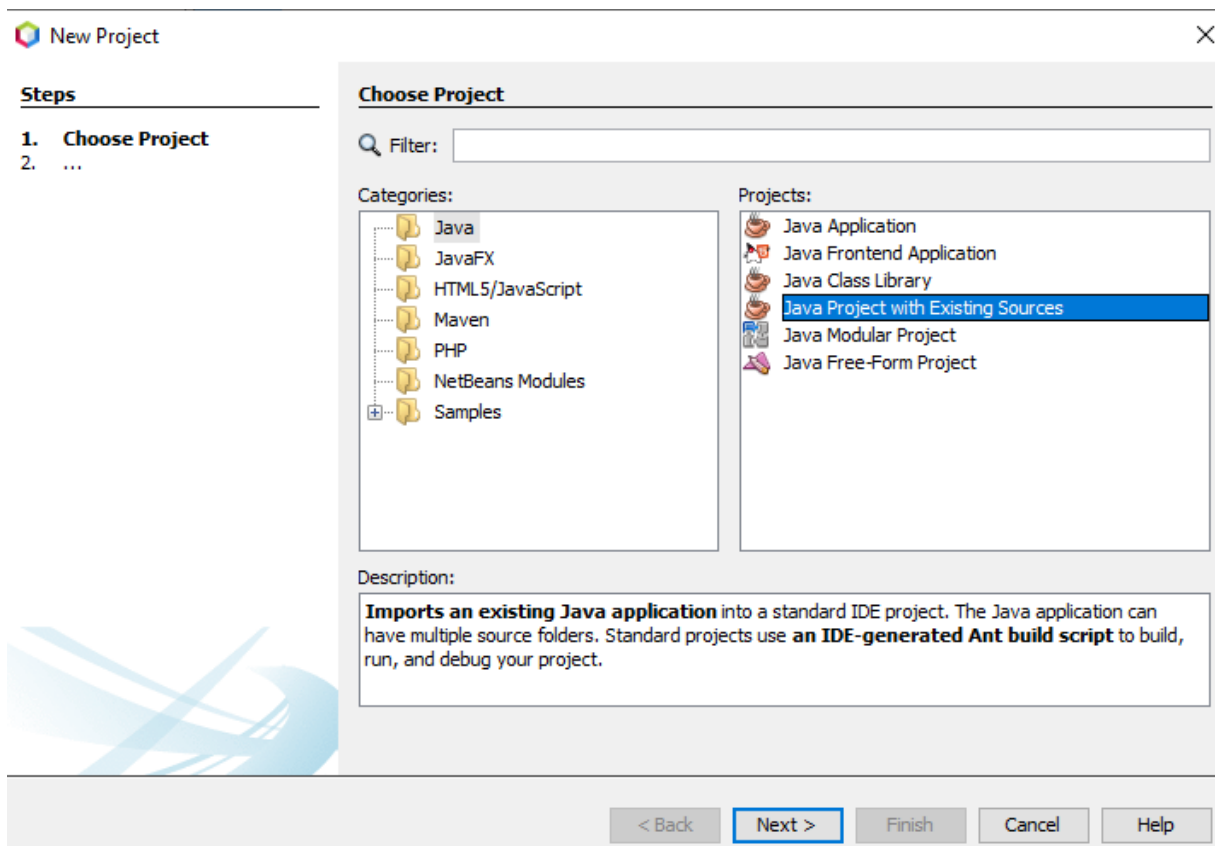


Figura 14 - Importar projeto para o Netbeans

2.5 Upload de alterações ao repositório local para um repositório online

Quando estamos a trabalhar com repositórios git, temos de ter a noção que o repositório do projeto pode estar replicado por vários computadores. Esta é, na verdade, um dos pontos fortes da ferramenta git, um sistema de controlo de versões distribuído.

Até agora, com excepção do comando git clone, estamos apenas a gerir o nosso repositório git local, contudo o trabalho em equipa exige a partilha de conteúdos com os restantes elementos da equipa. Embora possamos sincronizar directamente com computadores de outros utilizadores, tal está fora do âmbito deste tutorial. Assim sendo, iremos adoptar uma abordagem em que existirá uma cópia do nosso repositório online numa plataforma a sua escolha (ex: github.com ou gitlab.com).

Tomando como exemplo a plataforma github podemos após o registo do utilizador na plataforma, usar como referência o conjunto de figuras: Figura 15, Figura 16, Figura 17.

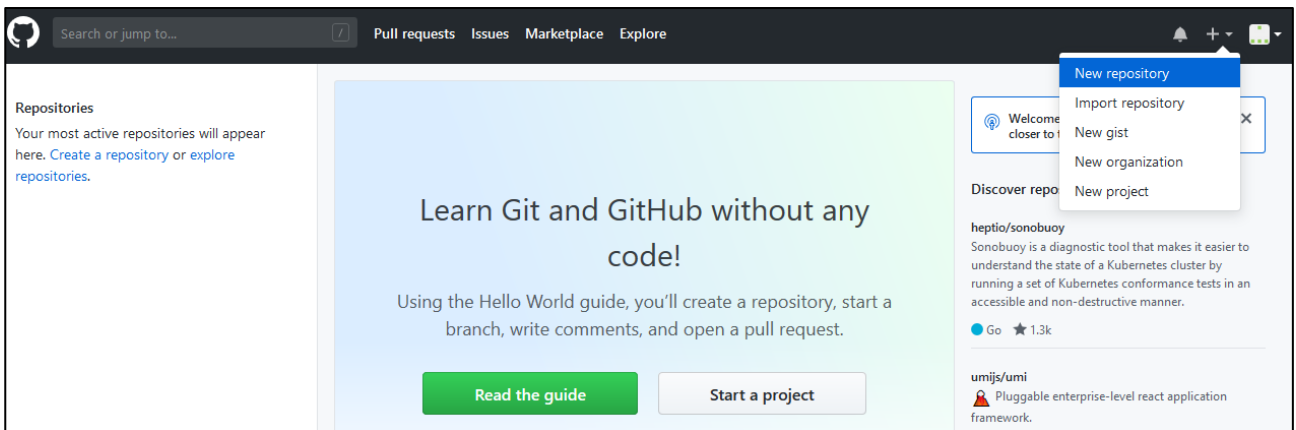




Figura 15 - Criar um repositório no público, parte 1

Create a new repository

A repository contains all project files, including the revision history.

Owner **Repository name ***

 faslPP / 

Great repository names are short and memorable. Need inspiration? How about **cautious-carnival**?

Description (optional)

☒ **Public**
Anyone can see this repository. You choose who can commit.

☐ **Private**
You choose who can see and commit to this repository.

☐ **Initialize this repository with a README**
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.


| 

Figura 16 - Criar um repositório público, parte 2

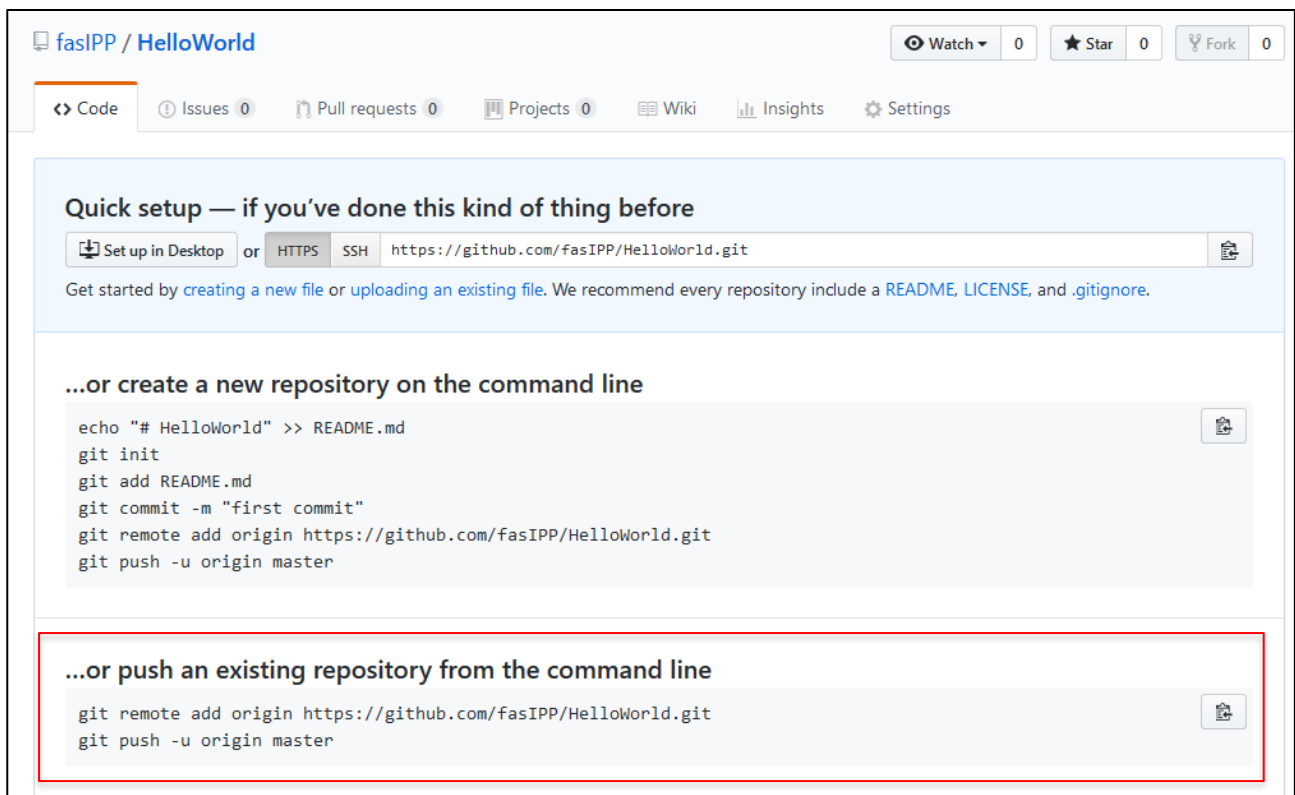


Figura 17 - Repositório criado, com instruções de como fazer upload do nosso projeto local para a plataforma, assinalado a vermelho

Devemos executar agora os comandos assinalados na posição a vermelho da conta do github pessoal. Os comando executam as seguintes tarefas:

- “git remote add origin url_repositório” – adiciona uma repositório externo ao nosso cliente git com a referência origin
- “git push -u master” – faz upload do nosso repositório, para o branch master do repositório master da plataforma github. O conceito de branch ainda não foi explicado, mas considere-se, para simplificação, a linha temporal principal do nosso repositório.

Utilizando o projeto Hello World criado acima, após a execução dos comandos acima, Figura 18, verificamos na plataforma github o upload do nosso projeto.

```
Faand@FAB MINGW64 ~/Documents/NetBeansProjects/test/Hello World ((183b02c...))
$ git remote add origin https://github.com/fasIPP/HelloWorld.git
Faand@FAB MINGW64 ~/Documents/NetBeansProjects/test/Hello World ((183b02c...))
$ git push -u origin master
Enumerating objects: 27, done.
Counting objects: 100% (27/27), done.
Delta compression using up to 4 threads
Compressing objects: 100% (17/17), done.
Writing objects: 100% (27/27), 16.40 KiB | 730.00 KiB/s, done.
Total 27 (delta 4), reused 0 (delta 0)
remote: Resolving deltas: 100% (4/4), done.
To https://github.com/fasIPP/HelloWorld.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
```

Figura 18 - Upload do projeto HelloWorld para a plataforma github

A partir deste momento o nosso repositório encontra-se replicado na plataforma online github, Figura 19. Eventuais membros da equipa de desenvolvimento devem efectuar o comando “git clone” para obterem a cópia do repositório que será usado para desenvolvimento em equipa.

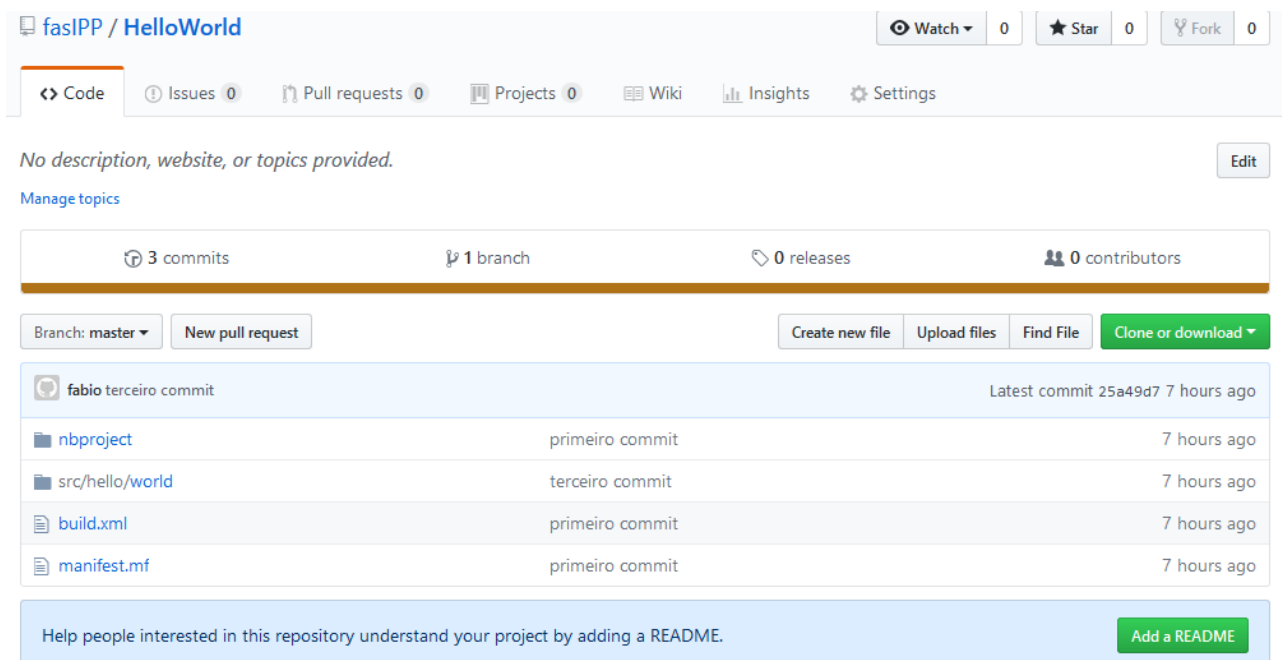


Figura 19 - Repositório github com upload do projeto

2.6 Download de alterações no repositório online para o repositório local

Durante o desenvolvimento dos projetos de software, é frequente efetuar alterações ao projeto. Estas alterações locais necessitam de ser partilhadas com os restantes membros da equipa. Para o efeito deve-se criar o hábito de executar o comando:

- “git pull --rebase origin” – obtêm as últimas alterações do repositório online e transfere-as para o repositório local. Além disso irá adicionar as alterações locais feitas pelo utilizador já sincronizadas no repositório.

```
Faand@FAB MINGW64 ~/Documents/NetBeansProjects/test/Hello World (master)
$ git pull --rebase origin
Already up to date.
Current branch master is up to date.
```

Figura 20 - Update do repositório local com últimas modificações do repositório online

Após efetuar o comando “git pull” surgem, por vezes, ter conflitos com ficheiros que foram alterados do lado do servidor e nos quais o utilizador também efetuou alterações locais. Neste caso é preciso resolver estes problemas antes de efetuar o upload de qualquer commit local novo.

Para simular esta situação, voltemos à versão do segundo commit do nosso projeto Figura 21.

```

$ git log
commit d03f144dd80dc39ec622e49c928301e775bb7afd (HEAD -> master, origin/master)
Author: fas@estg.ipp.pt <fas@estg.ipp.pt>
Date:   Fri Mar 15 21:18:54 2019 +0000

    quarto

commit 25a49d7eebae623ee50878562927c4b3e053ddce
Author: fabio <fabiosilva@di.uminho.pt>
Date:   Fri Mar 15 13:49:03 2019 +0000

    terceiro commit

commit 183b02c2868b0232046e85439c6c00c4687c3bfb
Author: fabio <fabiosilva@di.uminho.pt>
Date:   Fri Mar 15 13:13:53 2019 +0000

    segundo commit

commit 6f567f969a0b8a0168709e735bca44cf76cdf4d9
Author: fabio <fabiosilva@di.uminho.pt>
Date:   Fri Mar 15 12:57:11 2019 +0000

    primeiro commit

Faand@FAB MINGW64 ~/Documents/NetBeansProjects/test/Hello World (master)
$ git checkout 183b02c2868b0232046e85439c6c00c4687c3bfb
Note: checking out '183b02c2868b0232046e85439c6c00c4687c3bfb'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

    git checkout -b <new-branch-name>

HEAD is now at 183b02c segundo commit

Faand@FAB MINGW64 ~/Documents/NetBeansProjects/test/Hello World ((183b02c...))
$ git reset --hard
HEAD is now at 183b02c segundo commit

```

Figura 21 - Reverter o projeto à versão do segundo commit

Após voltar à versão do segundo commit devemos efectuar uma alteração ao ficheiro “HelloWorld.java” e fazer commit no nosso repositório local.

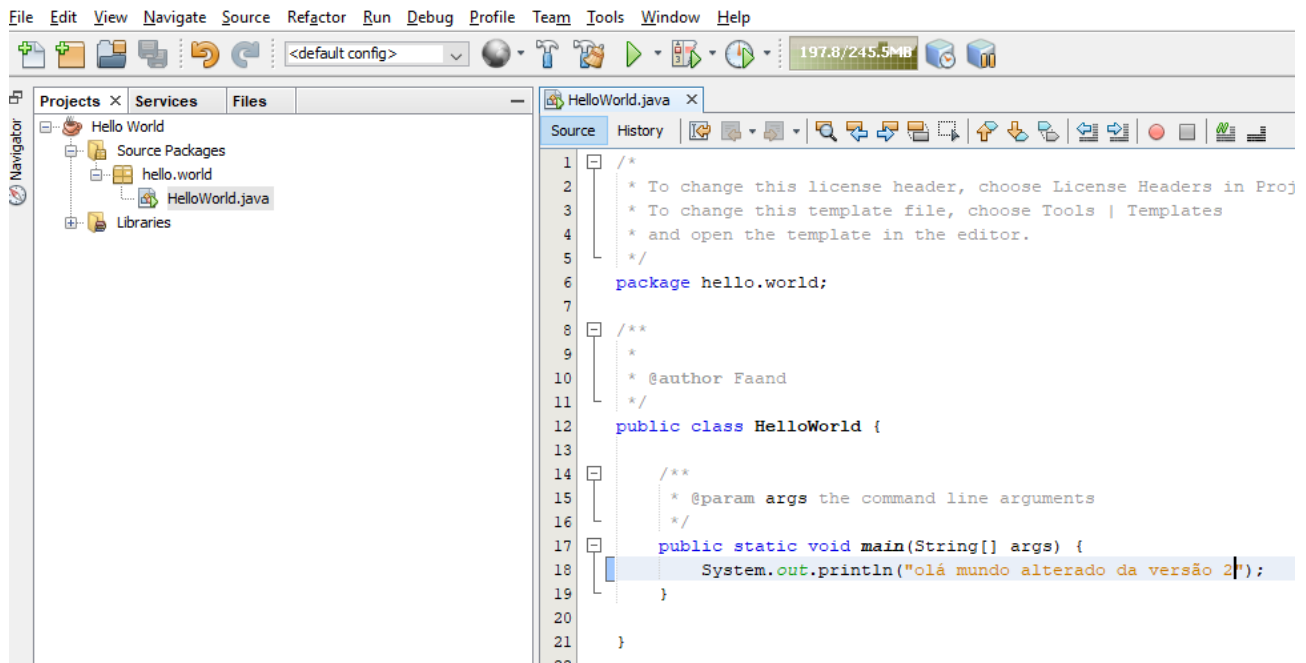


Figura 22 - Projecto alterado a partir da versão do segundo commit

Após alterarmos o projeto que recuperamos a partir do segundo commit, deveremos fazer novo commit das alterações.

```
Faand@FAB MINGW64 ~/Documents/NetBeansProjects/test/Hello World ((183b02c...))
$ git add *

Faand@FAB MINGW64 ~/Documents/NetBeansProjects/test/Hello World ((183b02c...))
$ git commit -m 'terceiro alternativo'
[detached HEAD db58e14] terceiro alternativo
1 file changed, 1 insertion(+), 1 deletion(-)

Faand@FAB MINGW64 ~/Documents/NetBeansProjects/test/Hello World ((db58e14...))
$ git log
commit db58e145528aafdb3c1bc33563d31459b9dabae0 (HEAD)
Author: fas@estg.ipp.pt <fas@estg.ipp.pt>
Date: Fri Mar 15 21:45:03 2019 +0000

    terceiro alternativo

commit 183b02c2868b0232046e85439c6c00c4687c3bfb
Author: fabio <fabiosilva@di.uminho.pt>
Date: Fri Mar 15 13:13:53 2019 +0000

    segundo commit

commit 6f567f969a0b8a0168709e735bca44cf76cdf4d9
Author: fabio <fabiosilva@di.uminho.pt>
Date: Fri Mar 15 12:57:11 2019 +0000

    primeiro commit
```

Figura 23 - Commit das alterações feitas para um terceiro commit alternativo

A partir daqui temos um problema. No repositório online vemos 4 commits, mas no repositório local vemos apenas 3 commits. O problema está em que o nosso terceiro commit é diferente do terceiro commit online, pois têm hash diferentes. Quando recuperamos uma versão anterior do projeto e efectuamos novo commit sobre ela, estamos a destruir os commits já feitos à posteriori. Desta forma existe uma edição concorrente do ficheiro a partir do segundo commit. Uma que está online e outra que está local.

Executando o comando “git pull –rebase origin” temos um merge conflict Figura 24. Este merge conflict gera no projeto informação sobre as linhas em conflito demarcadas nos ficheiros por “<<<<”, “=====”, “>>>>”, Figura 25. Devemos corrigir os conflitos identificados no netbeans, Figura 26, e depois correr os comandos, Figura 27:

- “git add *” – adiciona as alterações
- “git rebase --continue” – resolve o merge conflict

```
Faand@FAB MINGW64 ~/Documents/NetBeansProjects/test/Hello World ((db58e14...))
$ git pull --rebase origin master
From https://github.com/fasIPP/HelloWorld
* branch          master      -> FETCH_HEAD
First, rewinding head to replay your work on top of it...
Applying: terceiro alternativo
Using index info to reconstruct a base tree...
M       src/hello/world/HelloWorld.java
Falling back to patching base and 3-way merge...
Auto-merging src/hello/world/HelloWorld.java
CONFLICT (content): Merge conflict in src/hello/world/HelloWorld.java
error: Failed to merge in the changes.
hint: Use 'git am --show-current-patch' to see the failed patch
Patch failed at 0001 terceiro alternativo

Resolve all conflicts manually, mark them as resolved with
"git add/rm <conflicted_files>", then run "git rebase --continue".
You can instead skip this commit: run "git rebase --skip".
To abort and get back to the state before "git rebase", run "git rebase --abort"
.
```

Figura 24 - Exemplo git pull com merge conflicts

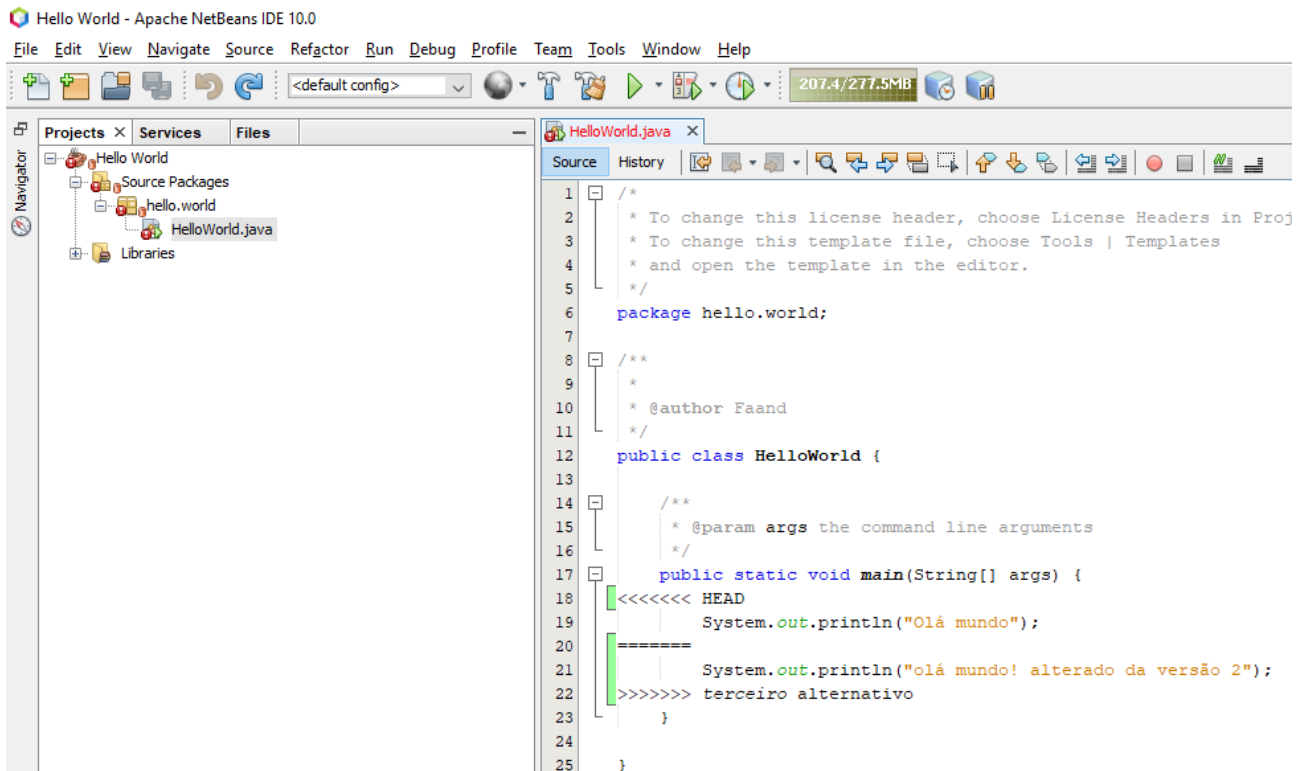


Figura 25 - Evidência dos problemas nos ficheiros do projeto

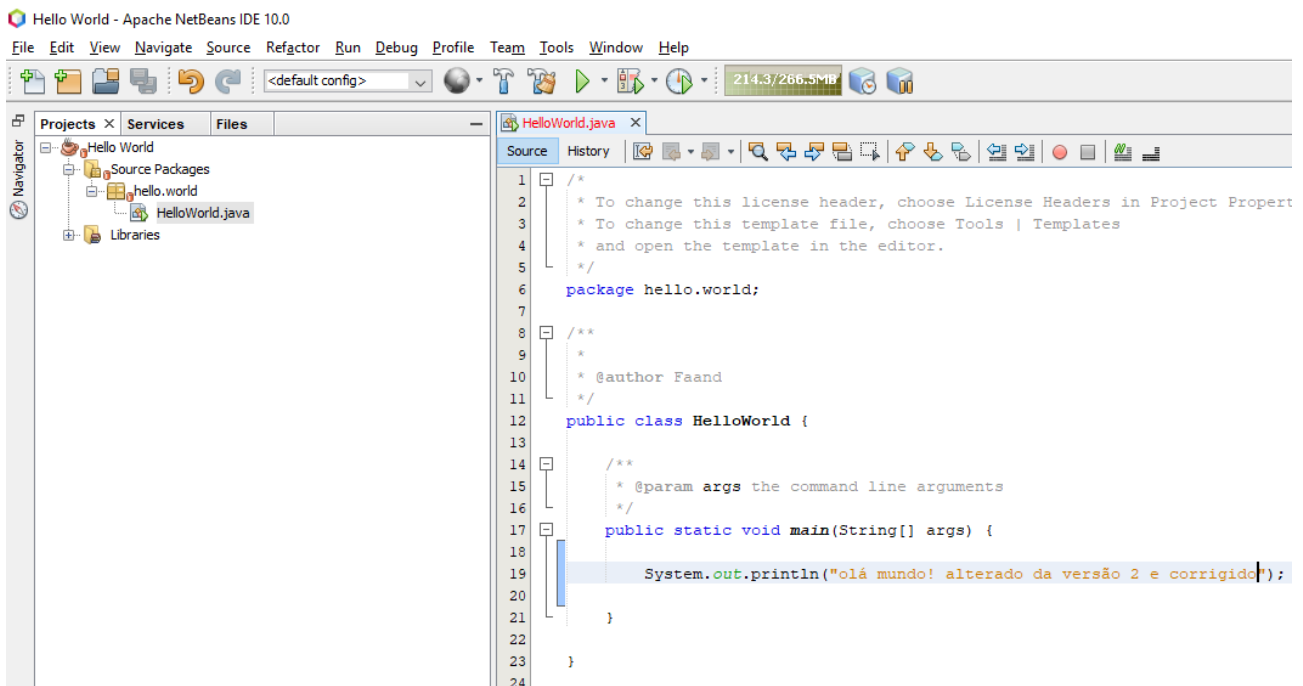


Figura 26 - Remoção das tags resultantes do merge conflict

```
Faand@FAB MINGW64 ~/Documents/NetBeansProjects/test/Hello World (detached HEAD|REBASE 1/1)
$ git add *

Faand@FAB MINGW64 ~/Documents/NetBeansProjects/test/Hello World (detached HEAD|REBASE 1/1)
$ git rebase --continue
Applying: terceiro alternativo
```

Figura 27 - Eliminar o merge conflict

2.7 Upload das alterações do repositório local para o repositório online

O upload das alterações locais para o repositório online deve ser sempre feito após uma sincronização/download das alterações que existam no repositório online e depois de solucionar todos os merge conflicts que podem resultar destas acções.

Após se assegurar que o histórico do repositório online está sincronizado com o repositório local, para efetuar upload das alterações locais deve ser executado o comando:

- “git push origin master” – Faz upload dos commits locais para o repositório online

Editamos novamente o projeto Hello World e criaremos um novo commit local.

Agora iremos fazer push para o repositório online no github.

Se tudo correu bem temos a informação na plataforma github actualizada com as últimas alterações locais.

2.8 Considerações finais

Esta demonstração da ferramenta git, cobre apenas os aspectos essenciais para o uso mínimo da ferramenta. Utilizadores mais avançados fazem um uso diferente da ferramenta com conceitos não abordados neste tutorial como branching, pull e merge requests que são essenciais para dominar a ferramenta git. Estes conceitos podem ser adquiridos completando o tutorial disponível em <https://learngitbranching.js.org/>. Estes conceitos poderão ser também abordados no futuras em fichas práticas.

3 Utilizar o cliente através do Netbeans

Após a criação ou clone de um repositório git, devemos importar o projeto para o IDE Netbeans como explicado na secção 2.4. A partir deste ponto podemos usar exclusivamente o cliente git disponível no próprio IDE netbeans para interagir com o repositório git local e remoto (online).

As ferramentas para interação com estão no menu “Team” da toolbar do IDE Netbeans, Figura 28.

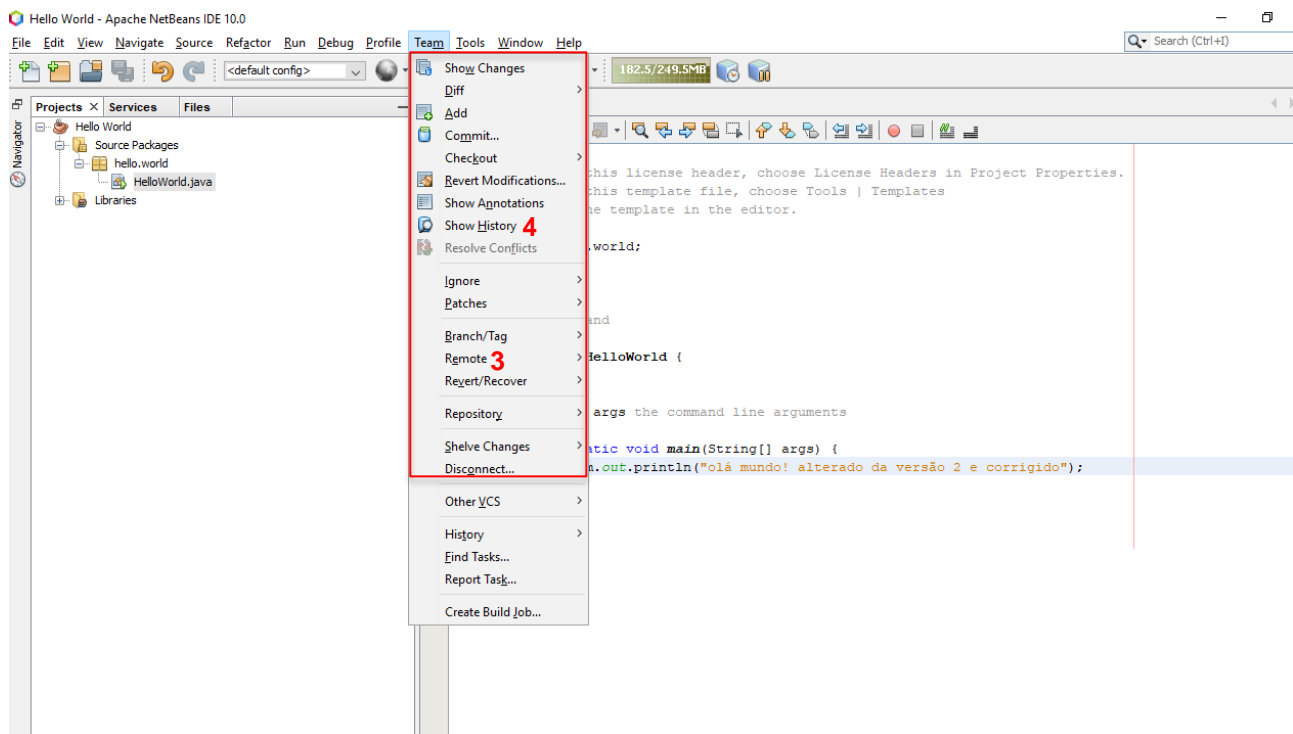


Figura 28 - Menus do git no IDE Netbeans, parte 1

Alternativamente estão também disponíveis clicando com o botão do lado direito do rato sobre o explorador do projecto do IDE Netbeans Figura 29.

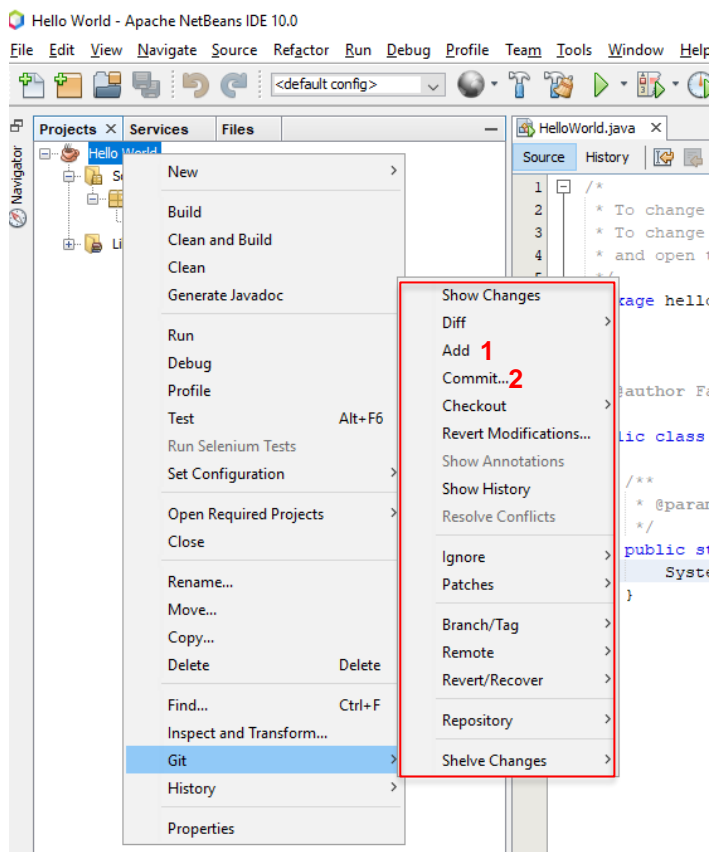


Figura 29 - Menus do git no IDE Netbeans, parte 2

Com estes menus é possível:

- 1 – Adicionar ficheiros ao staging
- 2 – Fazer commit local dos ficheiros
- 3 – Fazer pull e push de e para repositório online
- 4 – Verificar o estado do respositório local git.

Os menus presentes no IDE Netbeans simplificam a interação com o repositório git e na verdade dispensam o uso da consola do git. No entanto, para tarefas avançadas e para solução de problemas com repositórios git é aconselhavam o uso do cliente git em linha de comando que tem todas as opções sobre o repositório git. No IDE Netbeans apenas as opções parametrizadas estarão disponíveis, embora estas sejam suficientes para o uso de repositórios git.