Treinamento GIT no Eclipse

Leslie Harlley Watter

7 de maio de 2019

Sumário

1 Configuração do Git no Eclipse

As configurações do git no eclipse podem ser acessadas através do menu "Window > Preferences > Team > Git".

Por padrão o eclipse utiliza o arquivo `$HOME/.gitconfig` para as configurações do Git. Selecionando a opção "Configuration"na aba "User Settings"é possível verificar a configuração lida.

2 Trabalhando com Repositórios Git no Eclipse

Antes de trabalharmos com os projetos propriamente ditos, são necessárias algumas considerações a respeito do uso de repositórios git no eclipse.

Essas considerações foram extraídas diretamente da documentação do Eclipse/EGit e podem ser acessadas no original em `http://wiki.eclipse.org/EGit/User_Guide`

2.1 Considerações para Repositórios Git a serem usados no Eclipse

Ao configurar repositórios git dentro do eclipse, existem duas recomendações para a criação de repositórios de trabalho:

2.1.1 Recomendações

1. **Não crie o repositório dentro do workspace do eclipse**.
   - Tenha cuidado ao clonar ou criar um repositório.
   - O novo repositório irá contemplar a estrutura completa de diretórios do workspace do eclipse como conteúdo potencial. Isso pode resultar em problemas de performance, por exemplo ao calcular as mudanças antes de comitar (o que incluirá percorrer todo a árvore de diretórios .metadata, por exemplo); mais frequentemente que se espera, o workspace terá diretórios 'mortos' (p.ex. projetos removidos) que semanticamente não são relevantes para o EGit mas não podem ser excluídos facilmente.
   - O diretório de metadados do git (`.git`) estaria dentro do Workspace do Eclipse.
   - Você poderia facilmente destruir seu repositório removendo seu workspace do eclipse.
   - O próprio Eclipse sugere a criação de um diretório `$HOME/git/` onde os repositórios e diretórios de trabalho dos projetos serão armazenados.
2. **Não crie um repositório com o Eclipse estando com o usuário root**. Assim que você alternar de usuário, não conseguirá efetivar modificações no seu repositório.

Workspace do Eclipse e Diretório de Trabalho do Repositório Os repositórios git podem ser criados de diferentes maneiras, por exemplo clonando a partir de um repositório existente, criando um a partir de um diretório vazio, ou até mesmo usando o *EGit Sharing Wizard*.

De qualquer forma (a não ser que você crie um repositório *bare*), o novo repositório é essencialmente um diretório local no disco que contém o "diretório de trabalho"e o diretório de metadados. O diretório de metadados é um subdiretório com o nome `.git`, que contém o repositório atual.

O diretório de metadados é tratado transparentemente pelo cliente Git, enquanto que o diretório de trabalho é usado para expor a versão atual (checkout) dos arquivos do repositório para as ferramentas e editores.

2.2 Orientações

- Utilize o diretório `$HOME/git` para armazenar os repositórios e diretórios de trabalho (*working copy*) dos seus projetos git.
- Importe-os para o Workspace do Eclipse utilizando o wizard que pode ser acessado no menu *File* > *Import* >*Git*.

3 Criando um Projeto / Repositório

Existem diversas maneiras de criar um repositório e/ou importar um projeto git.

3.1 Criando um repositório local

Inicialmente iremos criar um repositório local.

Uma das maneiras de interagir com os repositórios git é mostrando a view *Git Repositories* através do menu *Window* > *Show View* > *Other* > *Git* > *Git Repositories*.

Essa view permite acessar as principais funções dos repositórios Git.

Mostre a view *Git Repositories* e coloque-a abaixo da view *Project Explorer*. Você deve obter a seguinte imagem como resultado:

Clique no link *Create a new local Git repository* para acionar o wizard de criação de repositório.

A seguir clique em *Browse* e crie um novo diretório/pasta chamado `exemplo` para abrigar o repositório.

O novo repositório aparecerá na view *Git Repositories*.

Esse novo repositório ainda não tem nada *comitado* nele. Observe a marcação `[NO-HEAD]` na figura anterior.

3.1.1 Crie um projeto maven exemplo

Crie um novo projeto de exemplo utilizando a sequência *File* > *New* > *Project...* > *Maven* > *Maven Project*.

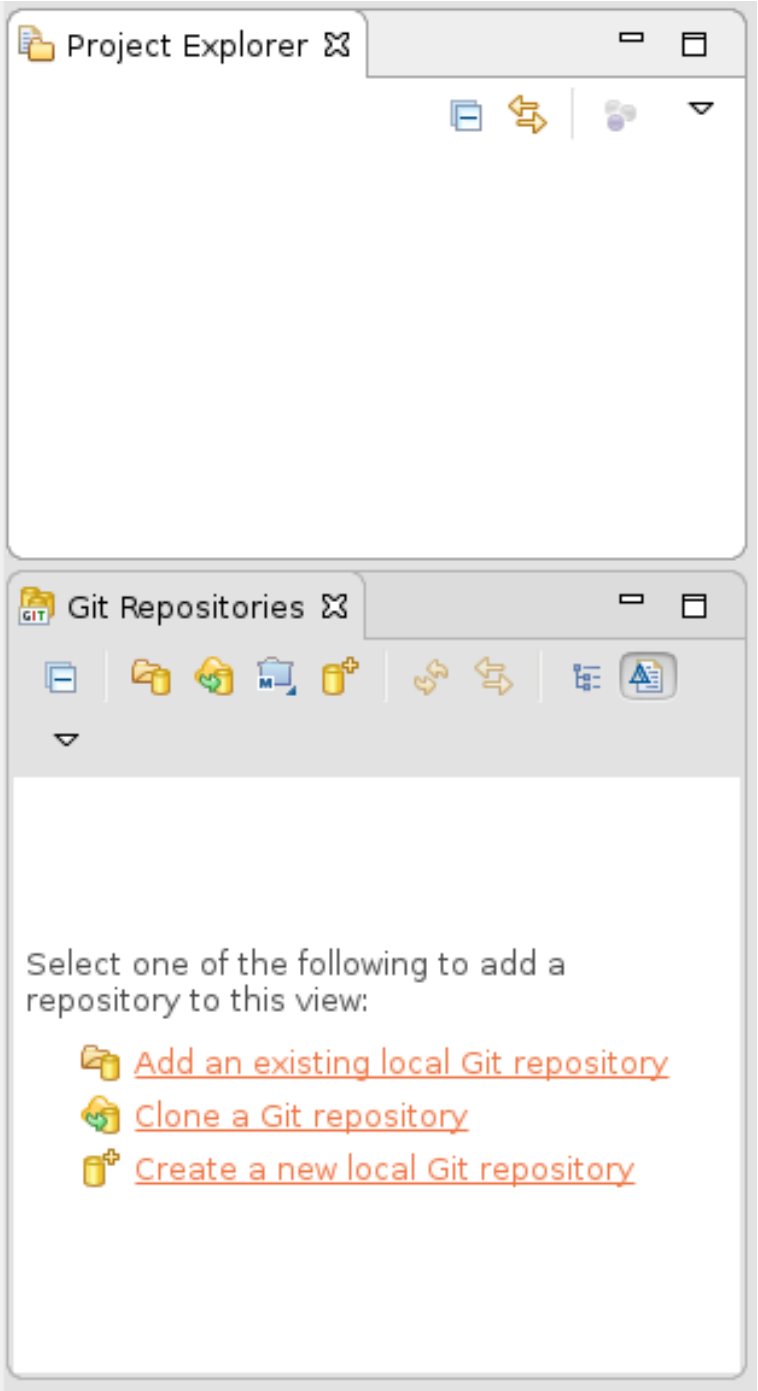
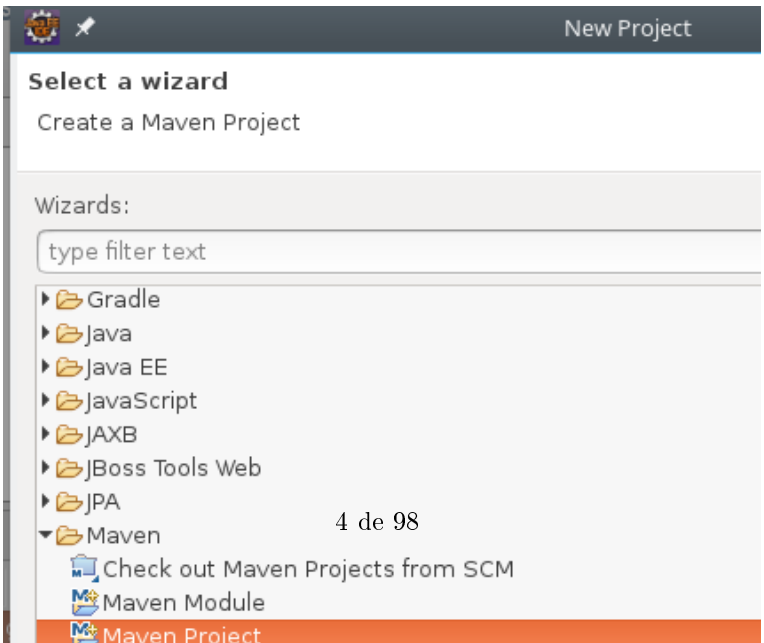


Figura 1: View Git Repositories



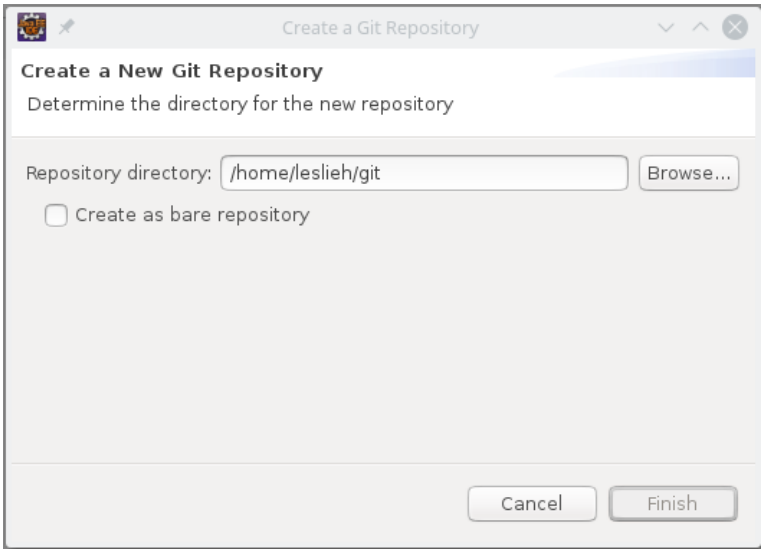


Figura 2:

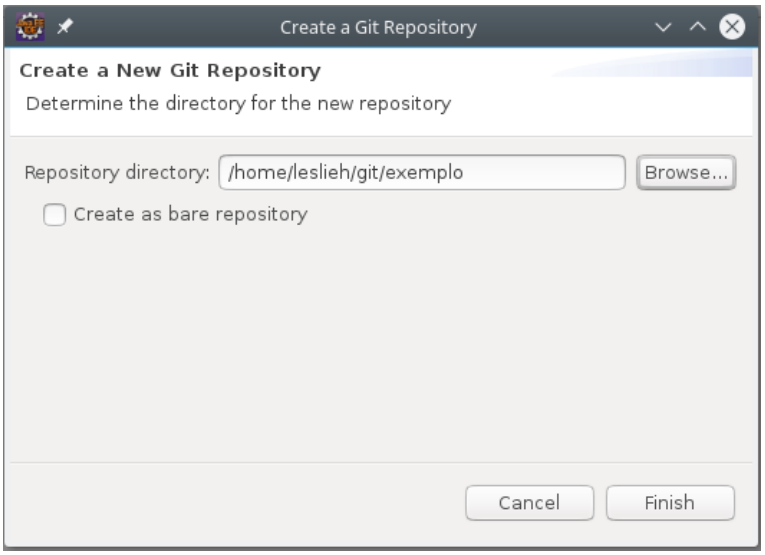


Figura 3:

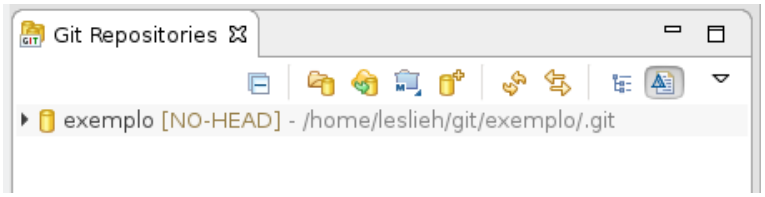


Figura 4:

Clique no botão *Next* para continuar. Marque a opção *Create a simple project (skip archetype selection)* e no botão *Next* novamente.

Crie o projeto exemplo conforme ilustrado na figura projeto exemplo:

3.1.2 Incluindo o projeto no controle de versão

Para incluir o projeto recém criado no controle de versão, selecione o projeto na view *Project Explorer*, clique com o botão direito do mouse sobre o projeto e selecione *Team* > *Share Project*:

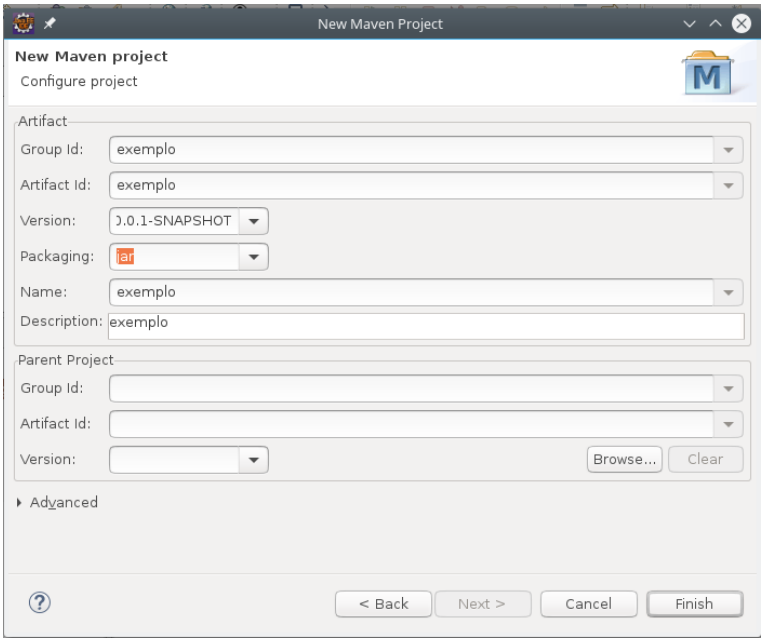
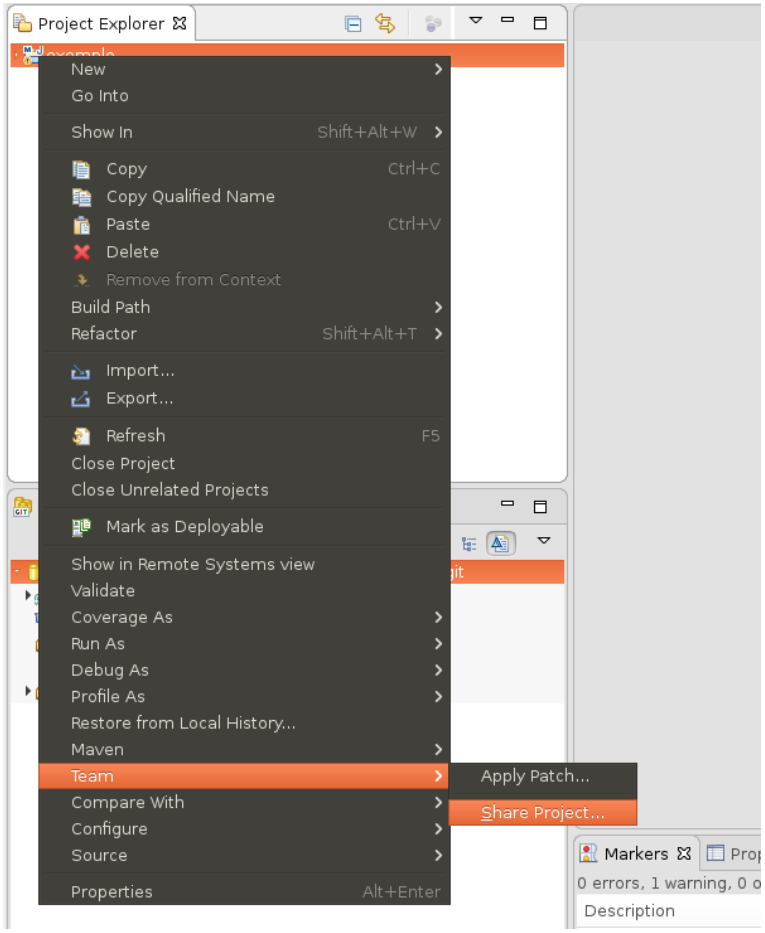
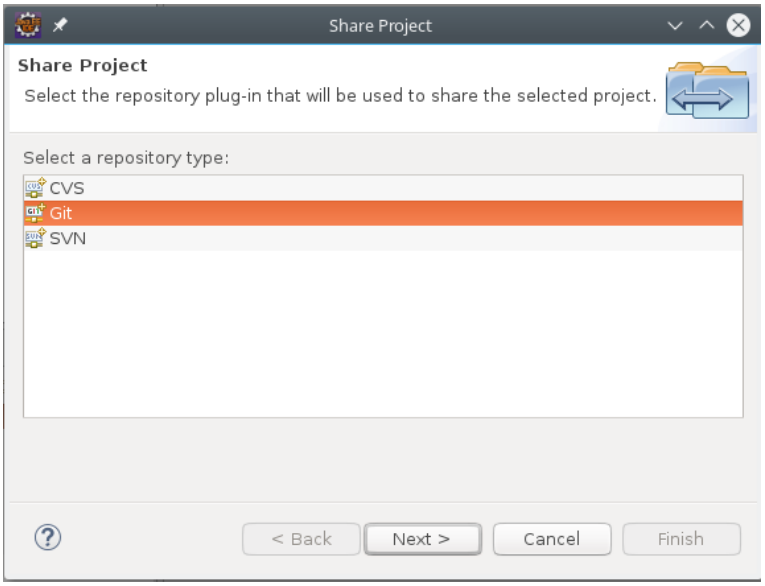


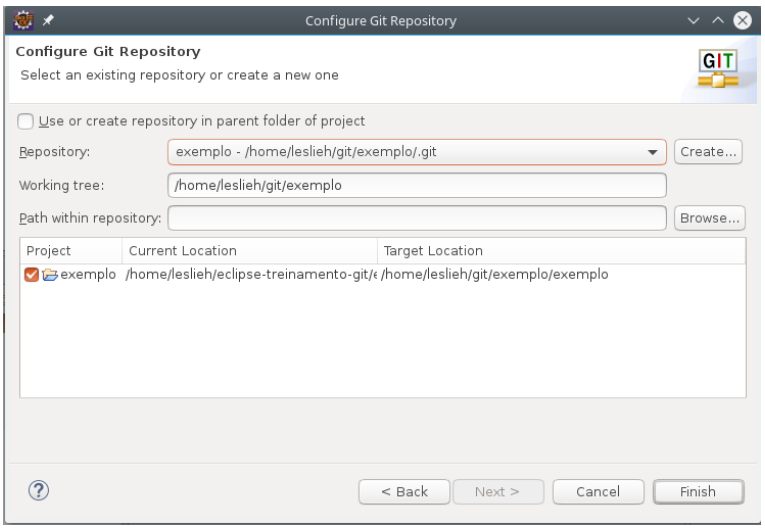
Figura 5: Projeto Exemplo



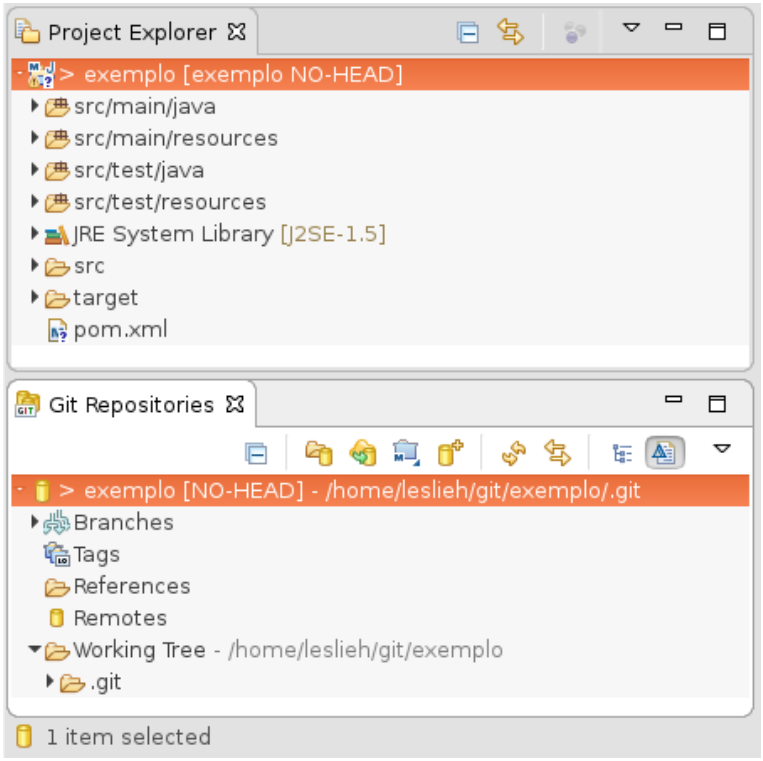
Selecione a opção *Git*



O próximo passo é configurar o repositório. Como já criamos o repositório antes, (no diretório `$HOME/git/exemplo/`) basta selecioná-lo na caixa de seleção e clicar no botão *Finnish*.

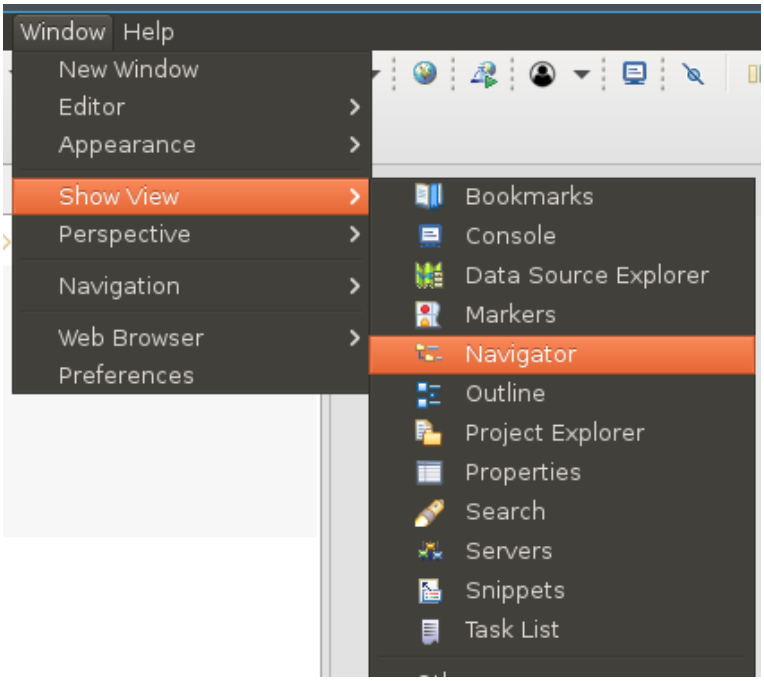


Observe que tanto o projeto novo, quanto o repositório exibem a marcação `[NO-HEAD]`, indicando que não houve nenhum commit ainda.

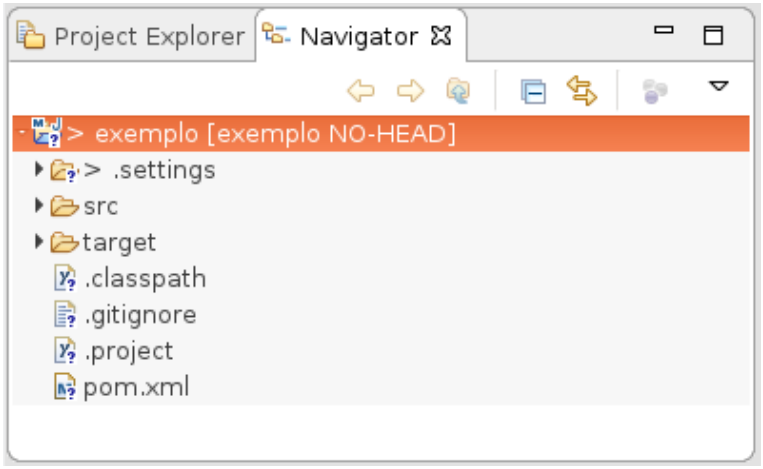


3.1.3 Alternando a Visualização

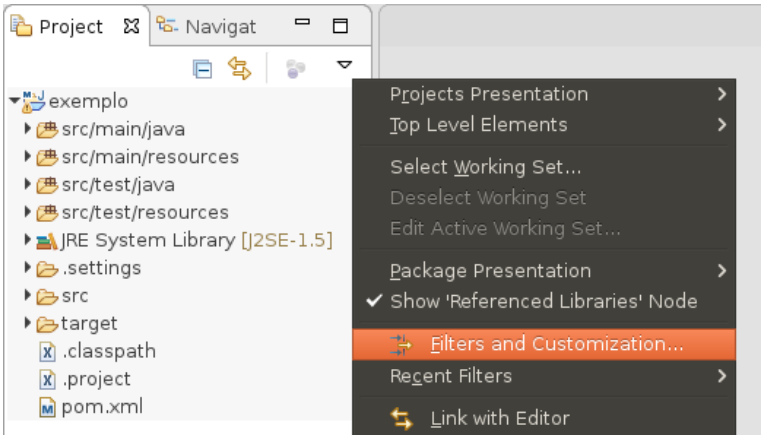
A view *Project Explorer* por padrão esconde algumas informações do projeto que são úteis. Alterne para a view *Navigator* acionando o menu *Window* > *Show View* > *Navigator*:



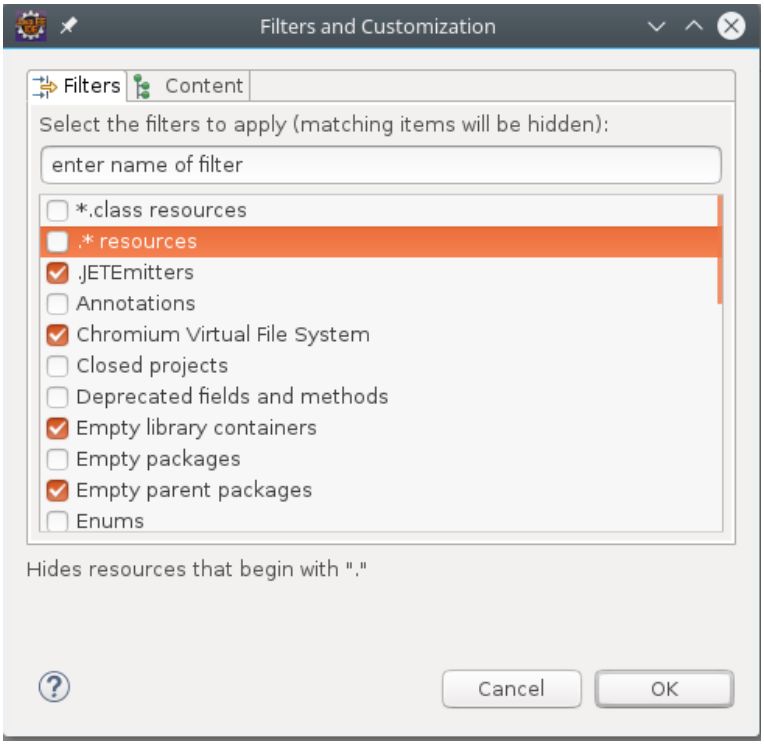
Observe que aparecem alguns arquivos até então escondidos `.classpath` , `.gitignore`, `.project` e também a árvore de diretórios `.settings`.



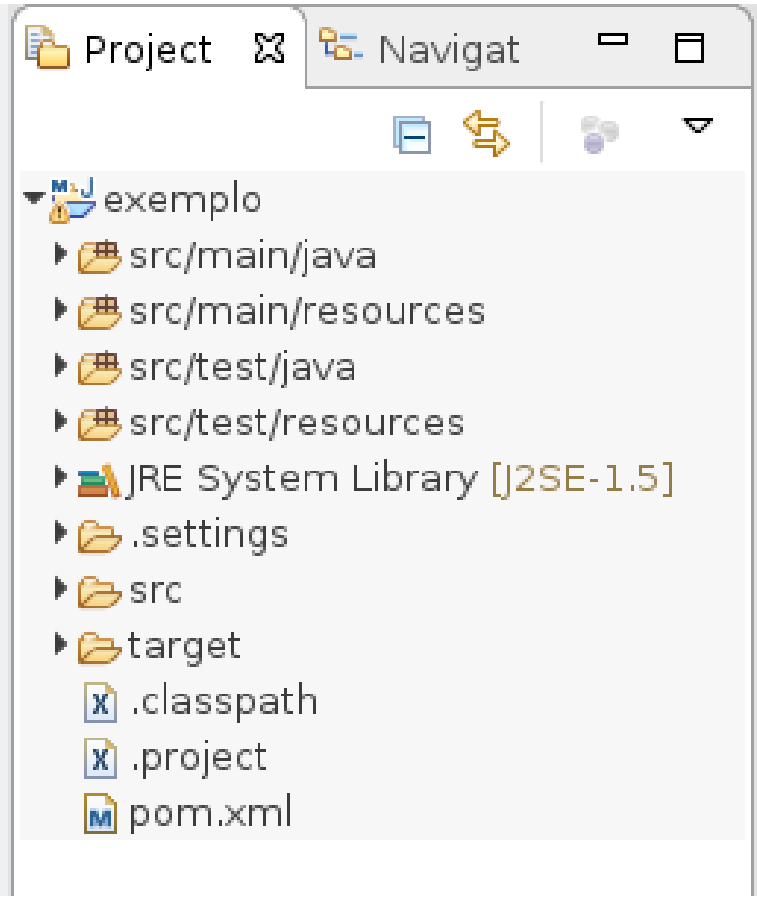
Uma maneira de fazer com que a view *Project Explorer* mostre esses arquivos é editar os filtros padrão da view, clicando com o botão direito na seta para baixo, e depois em *Filters and Customization...* e desmarcar a opção `.* resources`, como ilustrado a seguir:



Desmarque a opção `.* resources`.



O resultado é:



Por hora iremos continuar a utilizar a view *Navigator* por ter menos elementos para distração.

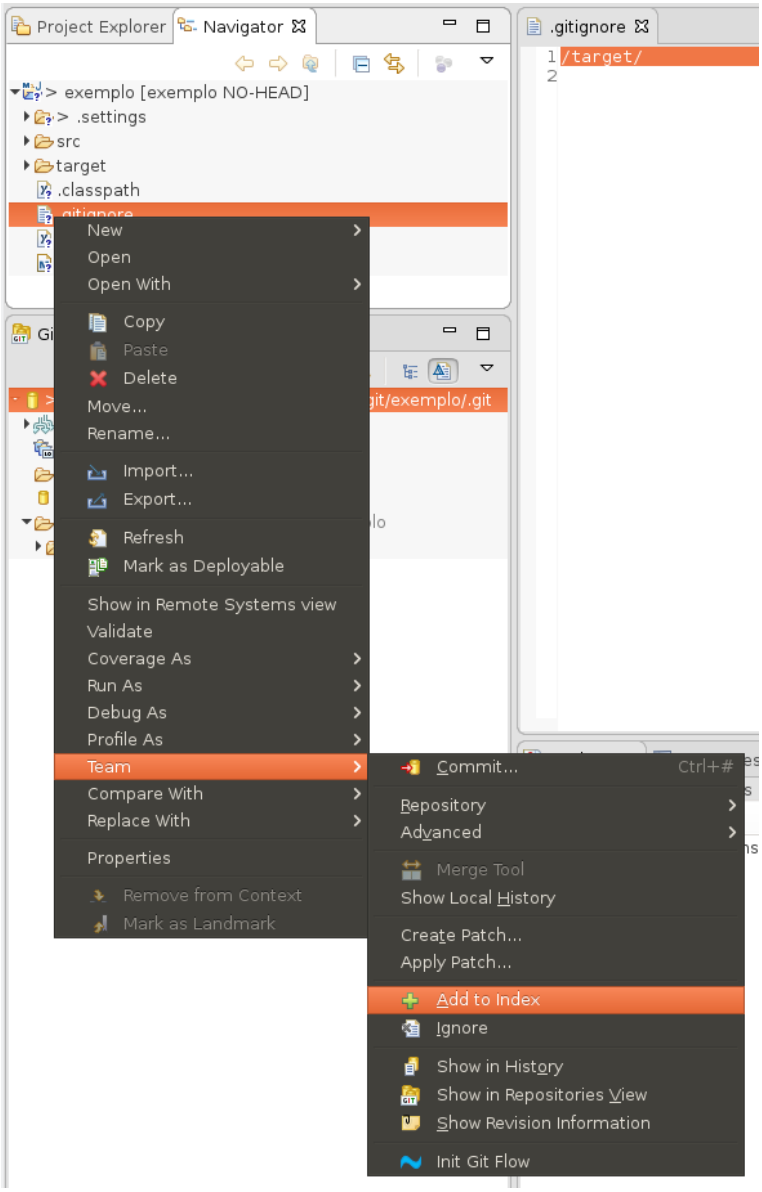
3.1.4 Primeiro commit - Ignorando o diretório `target`

Para o primeiro commit iremos acrescentar o arquivo `.gitignore`, cujo conteúdo foi preparado pelo wizard de criação de projetos maven:

```
/target/
```

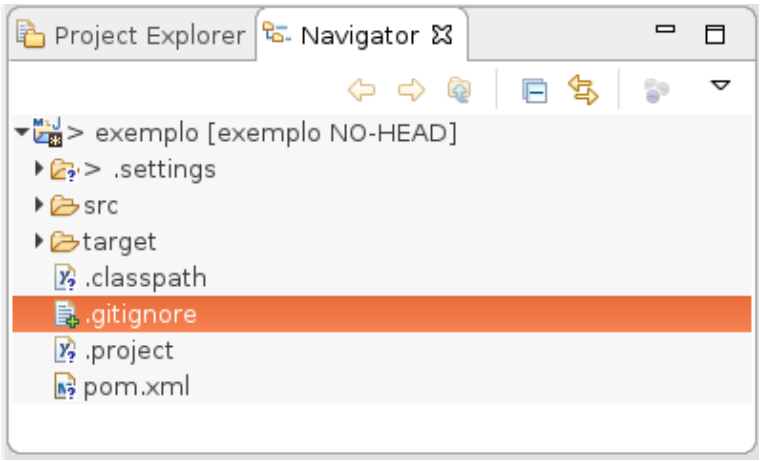
O arquivo `.gitignore` tem como conteúdo a lista de diretórios e/ou arquivos que o git irá ignorar ao percorrer o diretório de trabalho. Como os artefatos gerados pelo maven são criados dentro do diretório `target` esses arquivos não devem ser acrescentados ao repositório.

Para acrescentar o arquivo `.gitignore` ao controle de versão, clique sobre o arquivo e com o botão direito ative o menu *Team* > *Add to Index* conforme ilustrado.

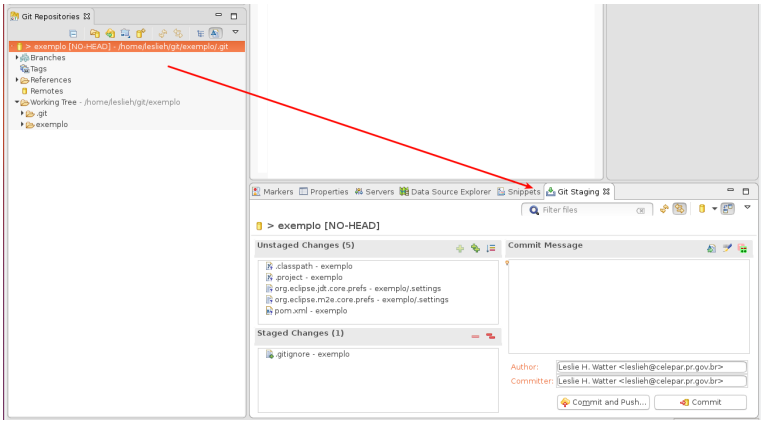


Observe que a marcação ao lado do arquivo `.gitignore` foi alterada (compare com a do arquivo `pom.xml`) para um sinal de mais (+) indicando que essa mudança está para ser commitada.

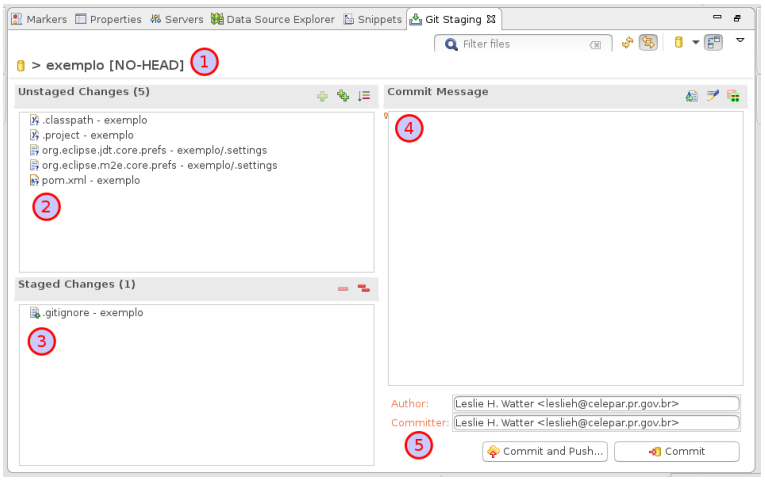
Outra modificação nas marcações surge ao lado do nome do projeto, com um asterisco preto (*), indicando que há uma modificação adicionada para commit no projeto.



O próximo passo é efetivar a mudança, acionando o menu *Team* > *Commit* ou pressionando as teclas *Ctrl+#*. Ao acessar esse menu, será aberta a view *Git Staging*.



Essa view costuma ser aberta ao lado da view *Snippets*. Vejamos em mais detalhes os elementos dessa view:



As marcações:

- **1** indica o projeto e o branch onde está sendo feita a alteração.
- **2** lista de arquivos do projeto que não foram adicionados ao index.
- **3** arquivos/modificações adicionadas ao index, que serão comitadas
- **4** local para mensagem de Commit
- **5** Identificação do *Autor* (autor da mudança) e *Commiter* (quem efetivamente comita a mudança). E botões para:

  Commit commit no repositório local.

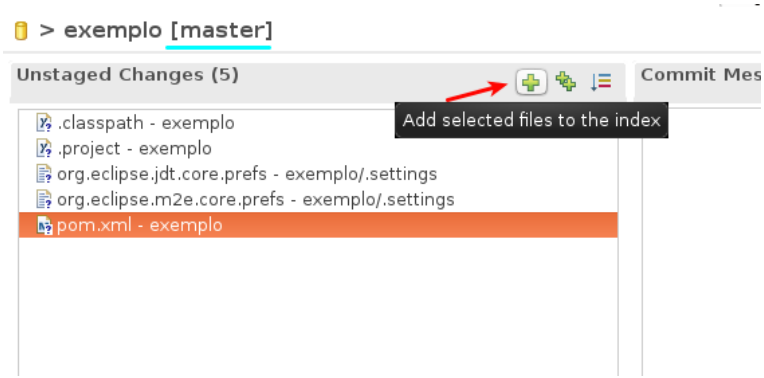
  Commit and Push commit no repositório local seguido de um *Push* ... (envio do commit para o repositório remoto).

Preencha a mensagem de commit com o seguinte texto:

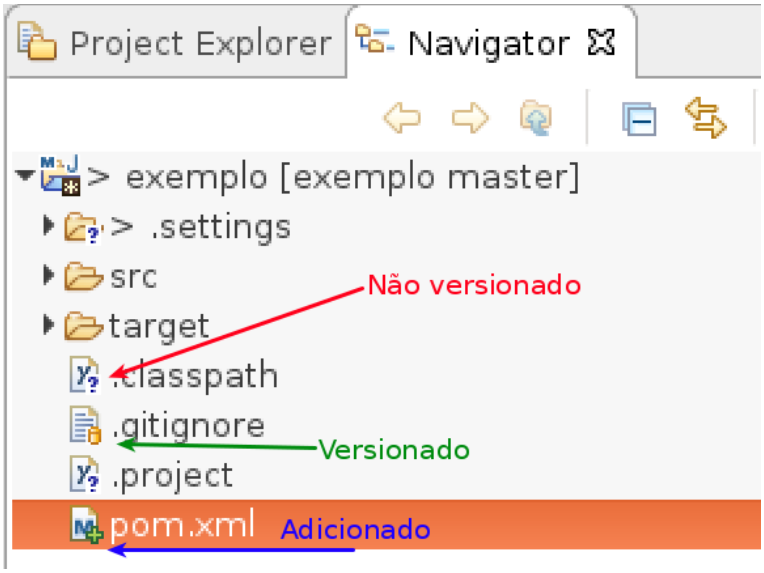
```
Configurações de ignore para projetos maven
```

E clique no botão *Commit*. Observe que o commit é feito e a view mantém a mesma aparência de quando foi acionada. Isso ocorre porque você pode querer preparar vários commits na sequência, acrescentando as mudanças nessa view.

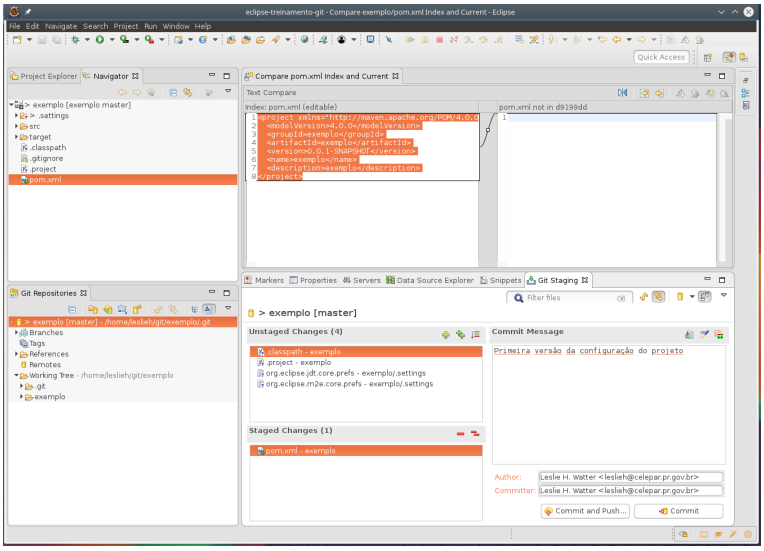
Observe também que o *branch* (sublinhado em azul claro) mudou para `master`. A seta vermelha indica como adicionar mais arquivos ao index e consequentemente preparar commits subsequentes.



Para exercitar, acrescente o arquivo `pom.xml` ao *Index* (observe a marcação com sinal de + ao lado do arquivo). Observe as marcações:



Após adicionar o arquivo `pom.xml` ao *Index*, área de preparação de commits, ou ainda *Staged Changes*, um duplo click sobre o arquivo abre a view de comparação das mudanças entre versões. Nesse caso, o arquvo inteiro é novo.



Preencha a mensagem de commit com o texto:

```
Primeira versão do arquivo de configuração do projeto
```

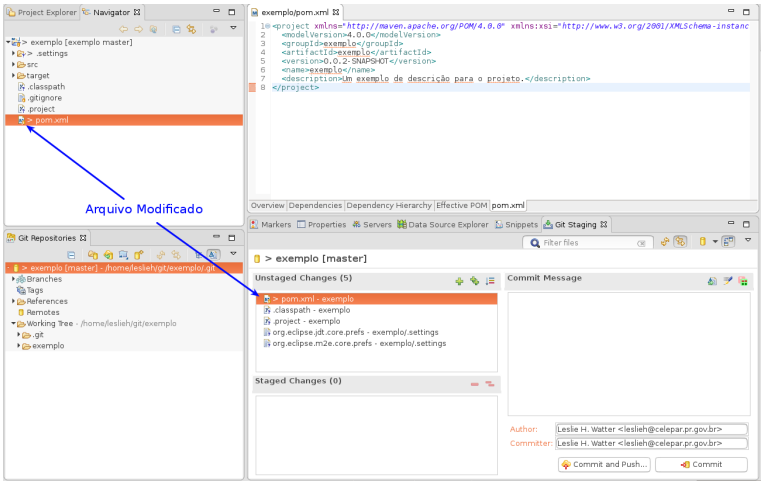
Commite o arquivo clicando no botão *Commit*.

3.1.5 Modifique a versão no arquivo pom.xml

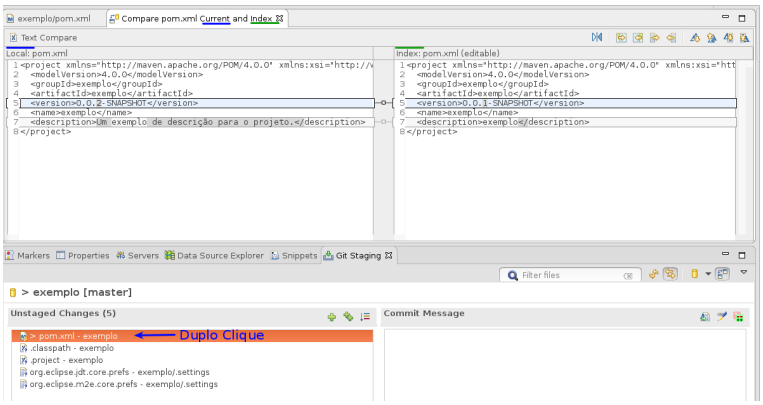
Edite o arquivo `pom.xml` e efetue as modificações nas tags *<version>* e *<description>* de forma a refletir o exemplo abaixo:

```
<version>0.0.2-SNAPSHOT</version>
<description>Um exemplo de descrição para o projeto.</description>
```

Salve o arquivo. E observe a modificação nas views *Navigator* e *Git Staging*:



Novamente um duplo clique sobre o arquivo `pom.xml` na view *Git Staging* abre a comparação, agora com as modificações que foram efetuadas:



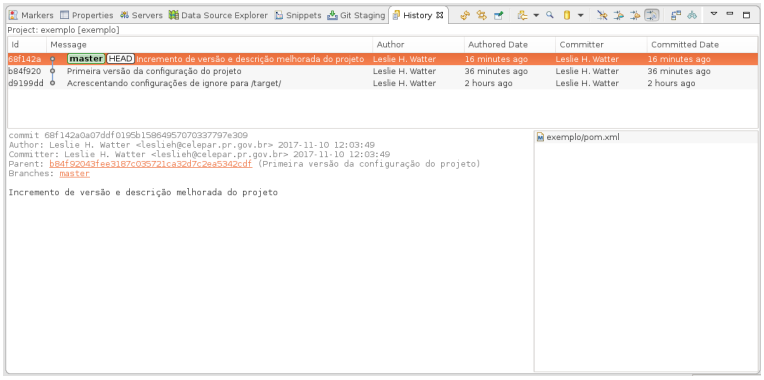
Adicione as mudanças feitas no arquivo `pom.xml` ao *Index* e comite-as usando a mensagem de commit:

```
Incremento de versão e descrição melhorada do projeto
```

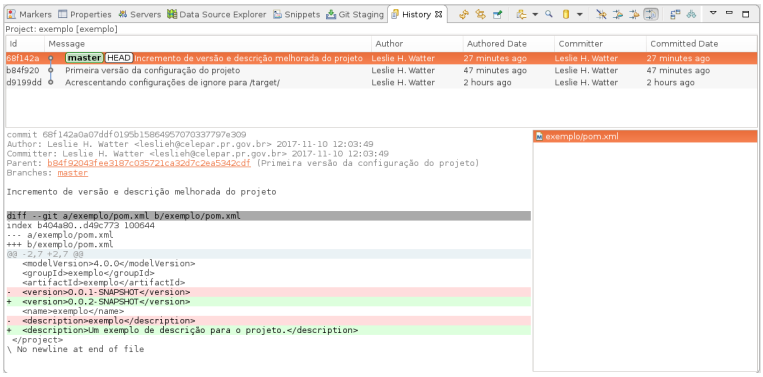
Agora que já temos algumas modificações vejamos o histórico.

3.1.6 Observando o Histórico

Clique sobre o **nome do projeto** e a seguir em *Team* > *Show in History*. Nessa view é possível observar o pequeno histórico do projeto, suas mensagens de commit e a evolução até o momento.



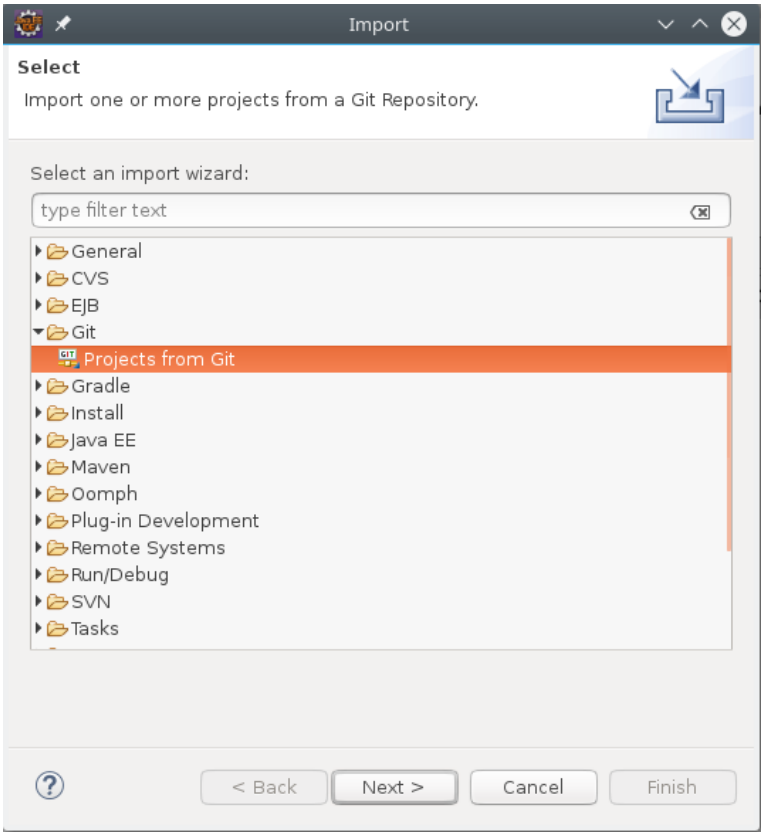
Ao clicar sobre um arquivo pertencente ao commit, nesse caso `pom.xml` é possível ver as diferenças no formato do diff, como ilustrado a seguir:



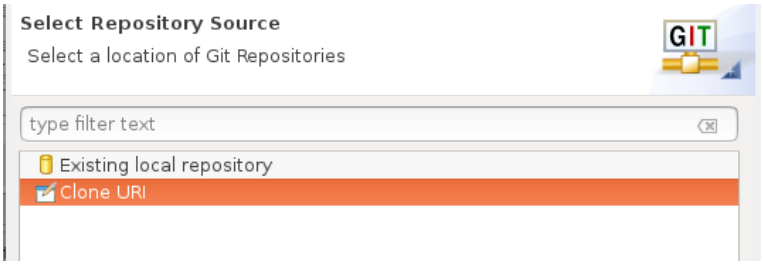
3.2 Importando um projeto de um repositório existente

Na grande maioria das vezes iremos importar um projeto pré-existente do repositório. Para isso utilizaremos o assistente de importação.

Utilize a sequência de menus *File* > *Import* > *Git* > *Projects from Git* como segue:



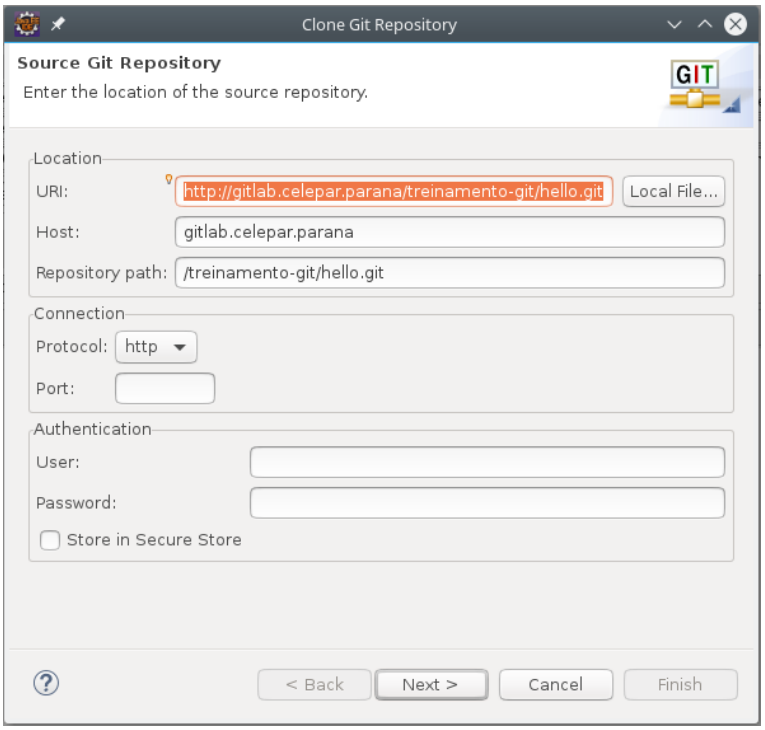
Selecione a opção **Clone Uri**.



Informe a URI do repositório `hello.git` no servidor gitlab:

`http://gitlab.celepar.parana/treinamento-git/hello.git`

E observe que o próprio eclipse preenche os campos *Host* e *Repository Path* para você.

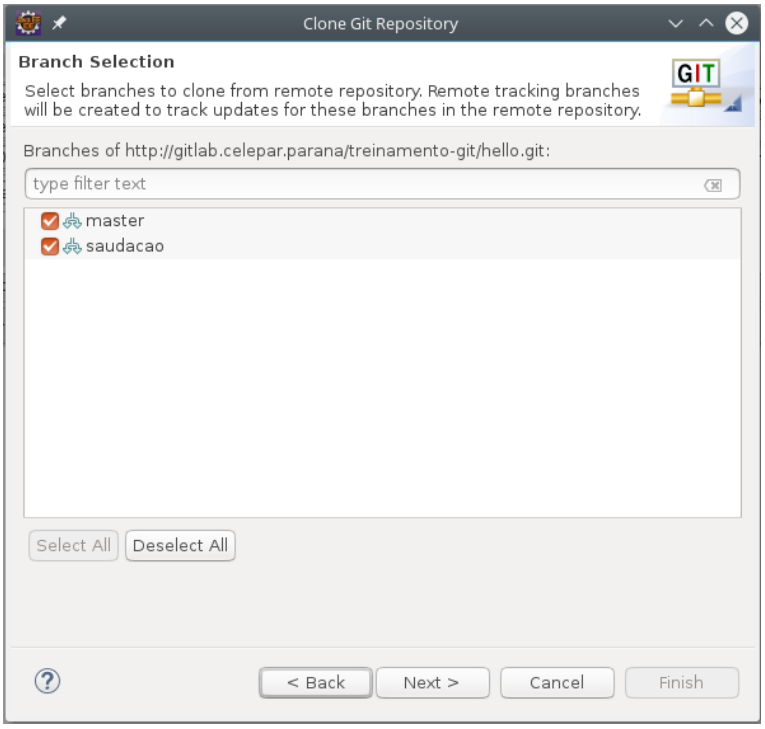


O gitlab foi configurado para utilizar autenticação a partir do LDAP, sendo assim, utilize seu usuário e senha do LDAP para poder baixar o projeto.

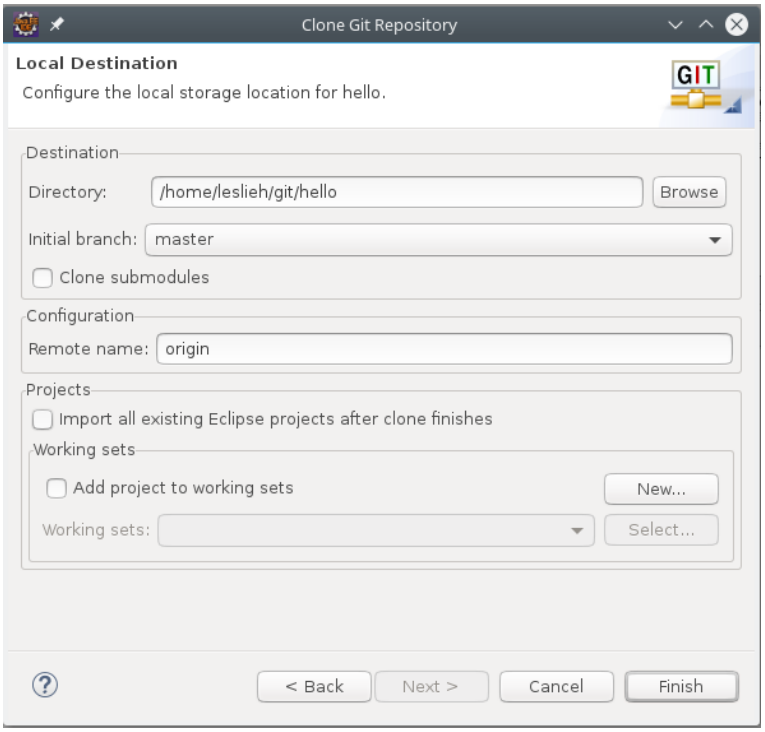
Para evitar ter de digitar a URI do projeto quando for baixar, acesse a página do projeto no gitlab e copie a URL para a área de transferência.

Já no eclipse, selecione a view *Git Repositories* e pressione as teclas `Ctrl + V` para preencher o wizard automaticamente.

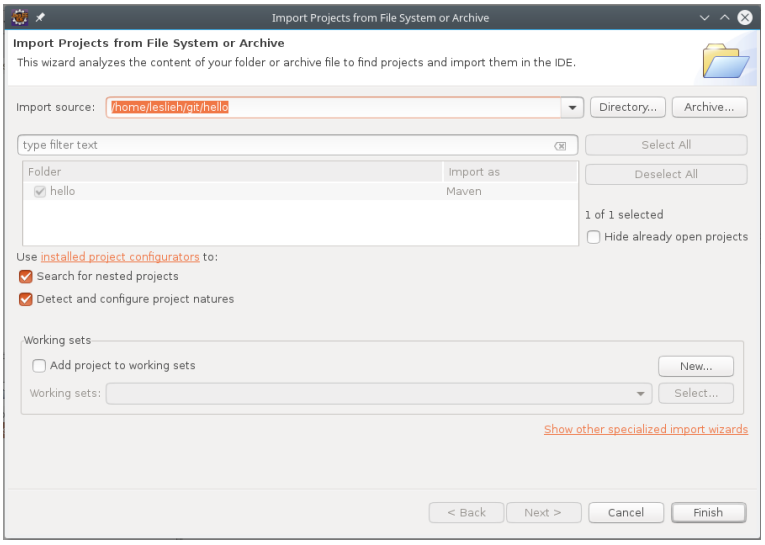
A janela sequinte pergunta quais *branches* você deseja baixar do repositório. Mantenha os branches *master* e *saudacao* selecionados, clique no botão *Next*.



A última janela permite informar a localização do repositório: `$HOME/git/hello`, o branch inicial: `master`, o nome do remoto: `origin`. Mantenha todas essas configurações inalteradas. Clique no botão *Finish* para criar um novo repositório local do seu projeto.



O repositório *hello* será criado na view *Git Repositories*. Selecione o repositório recém criado e clique com o botão direito do mouse para acionar o menu *Import Projects...*

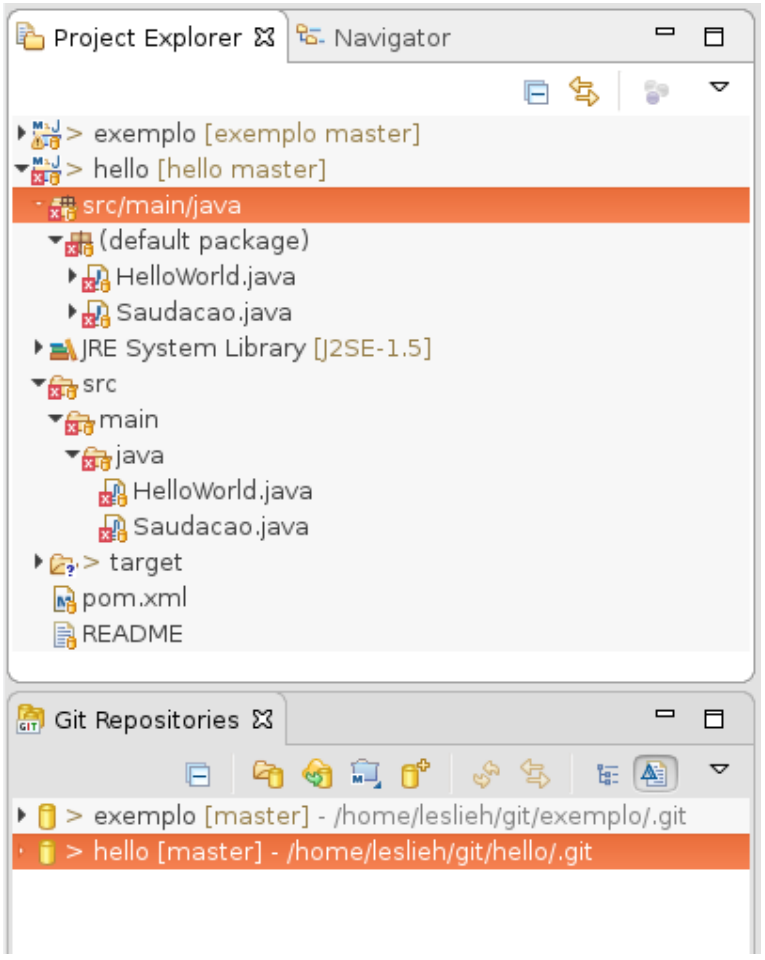


O projeto será adicionado ao workspace do eclipse e já com suas 'naturezas' (maven) configuradas.

É possível observar que, para fim de simplicidade na apresentação no modo texto, a estrutura de pacotes correta do maven não foi criada. Iremos corrigir a estrutura na sequência.

4 Verificando o Estado Atual

Após importado, o projeto hello mostra as deficiências mencionadas na estrutura padrão.



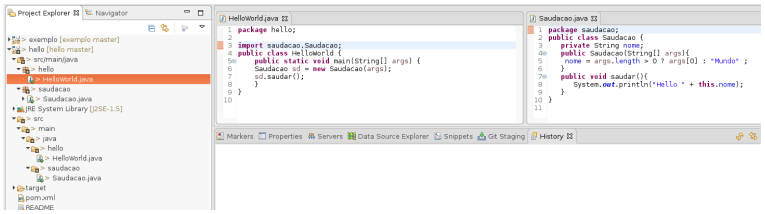
Para corrigir o projeto execute os seguintes passos:

1. crie os pacotes `hello` e `saudacao`

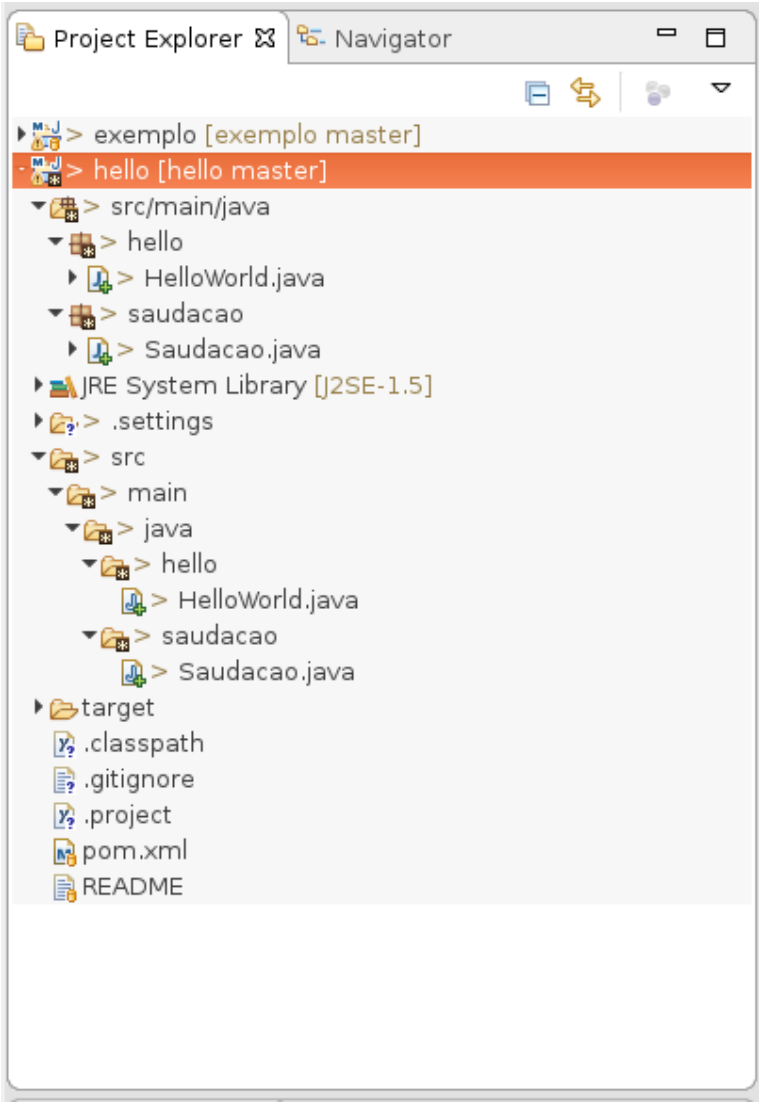
2. mova o arquivo `HelloWorld.java` para o diretório `/src/main/java/hello`. Selecione o arquivo listado logo antes do diretório `target`.

3. mova o arquivo `Saudacao.java` para o diretório `/src/main/java/saudacao`, da mesma forma que o anterior.

4. corrija a caixa do pacote saudacao de `Saudacao` para `saudacao`.



Observe que, mesmo movendo os arquivos de lugar, os marcadores indicam que os arquivos `HelloWorld.java` e `Saudacao.java` já estão rastreados pelo controle de versão. Veja o marcador (+) ao lado dos arquivos.



Por padrão o eclipse automaticamente adiciona ao *Index* os arquivos que estão sendo movidos. Essa configuração se encontra no menu *Window* > *Preferences* > *Git* > *Commiting* no item *Automatically stage files being moved*. Mantenha essa configuração como está.

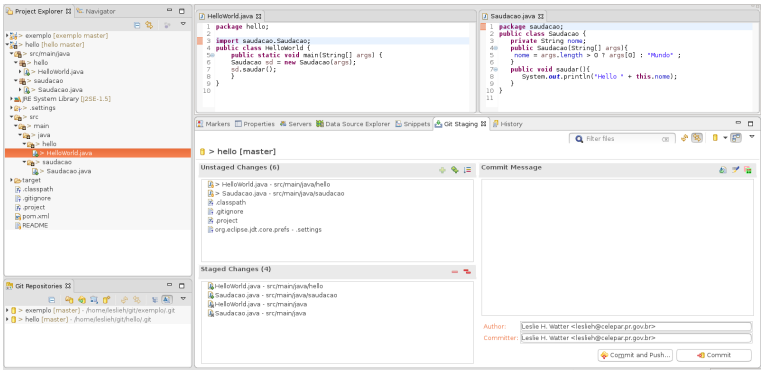
5 Adicionando as Mudanças

Observe que foram feitos dois tipos de mudanças nos arquivos:

1. mudança de local – já rastreada no repositório e adicionada ao index automaticamente pelo eclipse

2. mudança do nome do pacote – **ainda não acrescentada ao futuro commit**

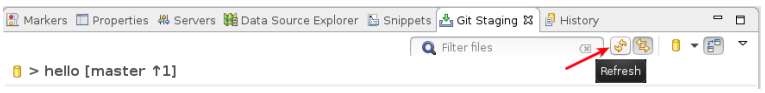
Essas mudanças refletem na visualização do commit. Acione o menu *Team* > *Commit* e observe que os arquivos java encontram-se tanto na área *Unstaged Changes* quando na área *Staged Changes*.



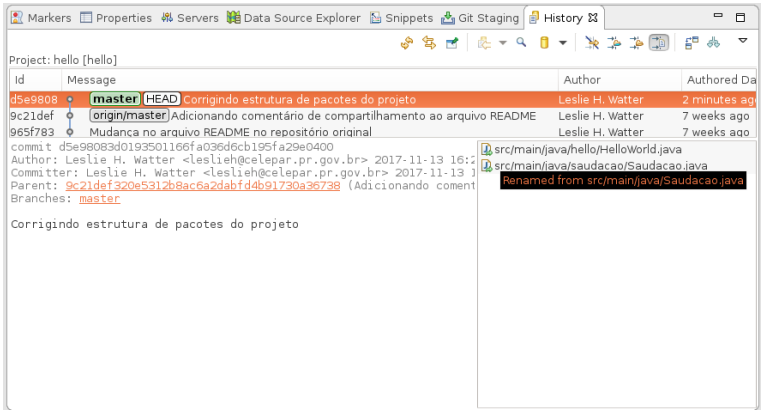
Commite e efetive ele usando a seguinte mensagem:

```
Corrigindo estrutura de pacotes do projeto
```

Caso após o commit a view *Git Staging* não seja atualizada, clique no botão *Refresh*. Se esse comportamento persistir, reinicie o eclipse.



Vejamos o resultado do commit no histórico, para isso selecione o projeto e acione o menu *Team* > *Show in History*, para observar o histórico. Ao passar o mouse sobre os arquivos que foram movimentados é possível obter a mensagem de movimentação:



6 Comitando as Mudanças usando Staging View

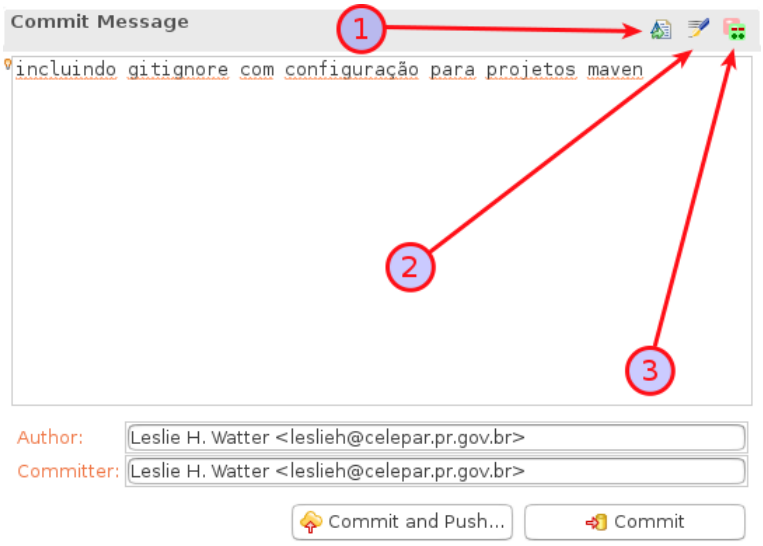
Volte para a view *Git Staging*. Observe que alguns arquivos não foram adicionados ao projeto, entre eles o arquivo `.gitignore`.

É importante que esse arquivo esteja presente no repositório do projeto para evitar o commit de arquivos compilados e/ou indesejados.

Iremos adicioná-lo ao projeto. Para tanto selecione o arquivo `/.gitignore` na lista e clique no botão +, conforme ilustrado a seguir:

Observe que existem alguns botões extras ao lado do título *Commit Message*. Marcados na figura a seguir estão:

- **1** - Ammend - Complementa e/ou edita o commit anterior. Usualmente utilizado quando é preciso acrescentar um arquivo que foi esquecido em um commit local.
- **2** - Acrescenta a tag *Signed-off-by:*, indicando que o usuário validou esse commit e assina-o. Muito usado em situações que um segundo desenvolvedor validou o commit.
- **3** - Acrescenta a tag *Change-Id:*, utilizada pelo gerrit para rastreamento de tíquetes e código.



Por hora iremos utilizar apenas o botão **2** acrescentando a tag *Signed-off-by:*.

> Sign-off is a requirement for getting patches into the Linux kernel and a few other projects, but most projects don't actually use it.
>
> It was introduced in the wake of the SCO lawsuit, (and other accusations of copyright infringement from SCO, most of which they never actually took to court), as a Developers Certificate of Origin. It is used to say that you certify that you have created the patch in question, or that you certify that to the best of your knowledge, it was created under an appropriate open-source license, or that it has been provided to you by someone else under those terms. This can help establish a chain of people who take responsibility

for the copyright status of the code in question, to help ensure that copyrighted code not released under an appropriate free software (open source) license is not included in the kernel.

`https://stackoverflow.com/questions/1962094/what-is-the-sign-off-feature-in-git-for`

O significado da linha *Signed-off-by* ao final da mensagem de log varia de projeto para projeto, mas via de regra certifica que o *committer* tem os direitos de submeter seu trabalho sob a mesma licensa do projeto e concorda com o *Developer Certificate of Origin* (veja `https://developercertificate.org` para mais informações)

O conteúdo do site `https://developercertificate.org/`

Developer Certificate of Origin Version 1.1

Copyright (C) 2004, 2006 The Linux Foundation and its contributors. 1 Letterman Drive Suite D4700 San Francisco, CA, 94129

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Developer's Certificate of Origin 1.1

By making a contribution to this project, I certify that:

(a) The contribution was created in whole or in part by me and I have the right to submit it under the open source license indicated in the file; or

(b) The contribution is based upon previous work that, to the best of my knowledge, is covered under an appropriate open source license and I have the right under that license to submit that work with modifications, whether created in whole or in part by me, under the same open source license (unless I am permitted to submit under a different license), as indicated in the file; or

(c) The contribution was provided directly to me by some other person who certified (a), (b) or (c) and I have not modified it.

(d) I understand and agree that this project and the contribution are public and that a record of the contribution (including all personal information I submit with it, including my sign-off) is maintained indefinitely and may be redistributed consistent with this project or the open source license(s) involved.

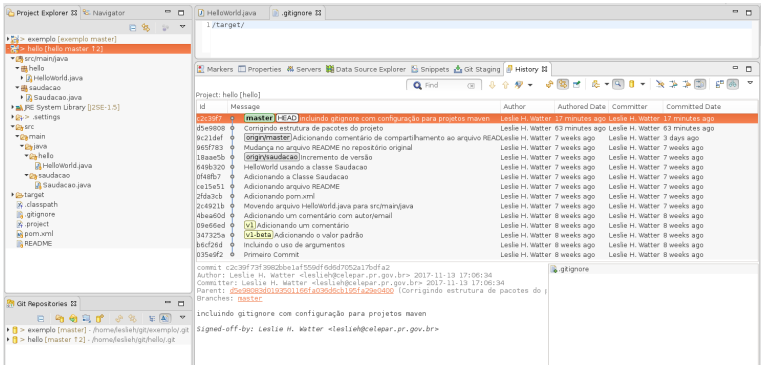
Para assinar os commits usando o GPG, veja o endereço: Ferramentas Git – Assinando seu Trabalho

Commite o arquivo.

7 Visualizando o Histórico

Vejamos agora como o histórico do git pode nos ajudar a encontrar as modificações e também a importância de boas mensagens de commit.

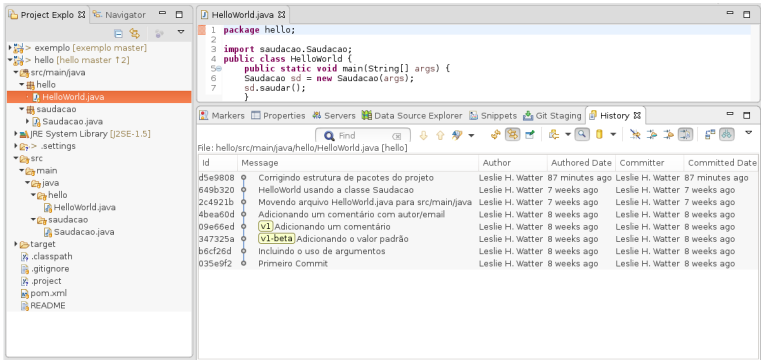
Selecione o projeto *hello* e com o botão direito do mouse acione o menu *Team* > *Show in History*.



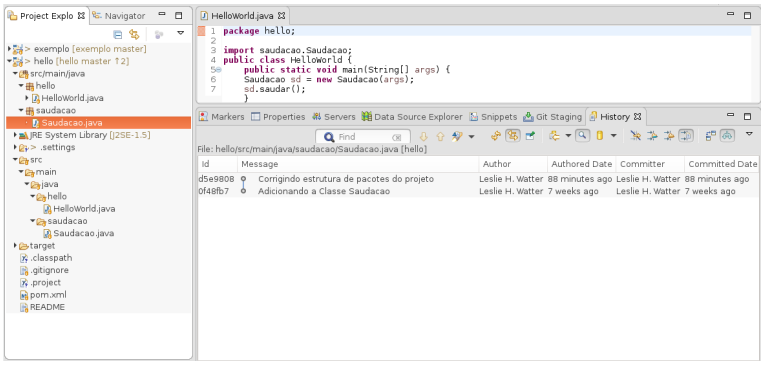
Alguns elementos dessa figura estão marcados abaixo:

- **1** – barra de pesquisa.
- **2** – botão *Link with Editor and Selection* – permite alternar as visões do histórico do repositório dependendo da seleção.
- **3** – botões que restringem o que será mostrado no histórico conforme a seleção. O botão selecionado é *Show all changes of selected resource and its children* que mostra todas as mudanças do recurso selecionado e filhos. Mantenha esse botão ativado.
- **4** – *Compare Mode* – permite comparar a versão de recursos atuais com a revisão selecionada com um duplo clique sobre o recurso/arquivo.
- **5** – *Show All Branches and Tags* – Mostra todos os branches e tags do repositório selecionado. Útil para acompanhar o desenvolvimento de diferentes branches.

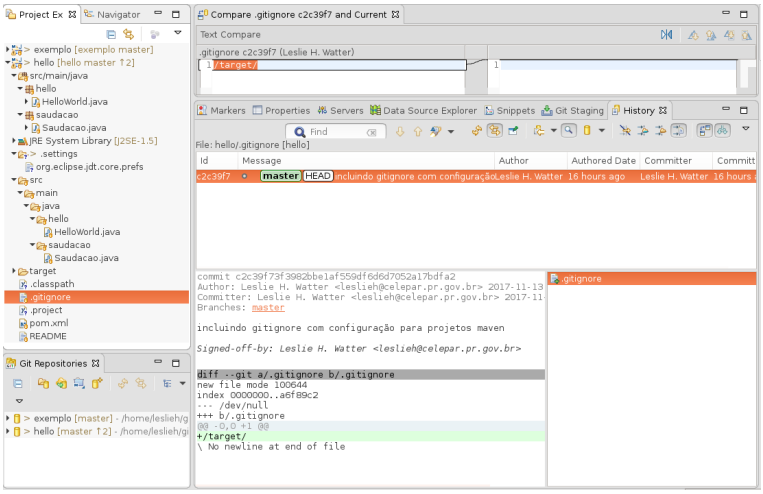
Ative o botão **2** *Link with Editor and Selection* para os próximos passos:
Verifique o histórico do arquivo *HelloWorld.java* clicando sobre ele. Apenas os commits onde foram feitas modificações nesse arquivo são listados.



Verifique o histórico do arquivo *Saudacao.java* clicando sobre ele. Novamente, apenas os commits onde houve modificações no arquivo são listados.



Observe que a navegação nos commits tem como base várias informações, porém a principal delas é a mensagem de commit.
É perceptível a importância de uma boa mensagem de commit conforme já mencionado anteriormente!
Por último, selecione o arquivo `.gitignore`, ative o botão *Compare Mode* e clique duas vezes sobre o nome do arquivo ao lado da mensagem de commit. Você deverá obter o seguinte resultado:



Pode-se observar algumas informações nessa imagem:

- Na view *Project Explorer* é possível ver que ainda existem arquivos no projeto que não estão versionados nem ignorados.
  - Por exemplo os arquivos: `.classpath`, `.project` e o diretório `.settings`
- Na view *Compare .gitignore c2c39f7 and Current* aparece somente o diretório `/target/` adicionado ao arquivo `.gitignore`, enquanto que alguns dos arquivos anteriores também deveriam ser ignorados (`.classpath`, `.project` e o diretório `.settings`).
- Na view *History* pode-se ver o único commit no arquivo `.gitignore`, que esse commit está no branch *master* e é marcado pelo *HEAD*. Logo abaixo encontra-se:
  - o commit: `c2c39f73f3982bbe1af559df6d6d7052a17bdfa2`,
  - o autor e commiter além da data e hora que foram feitos: `Author: Leslie H. Watter <leslieh@celepar.pr.gov.b` `2017-11-13 17:06:34`,
  - a mensagem de commit:

```
incluindo gitignore com configuração para projetos maven

Signed-off-by: Leslie H. Watter <leslieh@celepar.pr.gov.br>
```

- as diferenças incluídas para o arquivo `.gitignore` (que foi selecionado)

```
diff --git a/.gitignore b/.gitignore
new file mode 100644
index 0000000..a6f89c2
--- /dev/null
+++ b/.gitignore
@@ -0,0 +1 @@
+/target/
\ No newline at end of file
```

- Ao lado, os arquivos modificados. Nesse caso apenas `.gitignore`.

8 Corrigindo o commit anterior (com `--amend`)

Como identificado há pouco, se faz necessário acrescentar os arquivos `.classpath`, `.project` e o diretório `.settings` à lista de arquivos e diretórios ignorados.

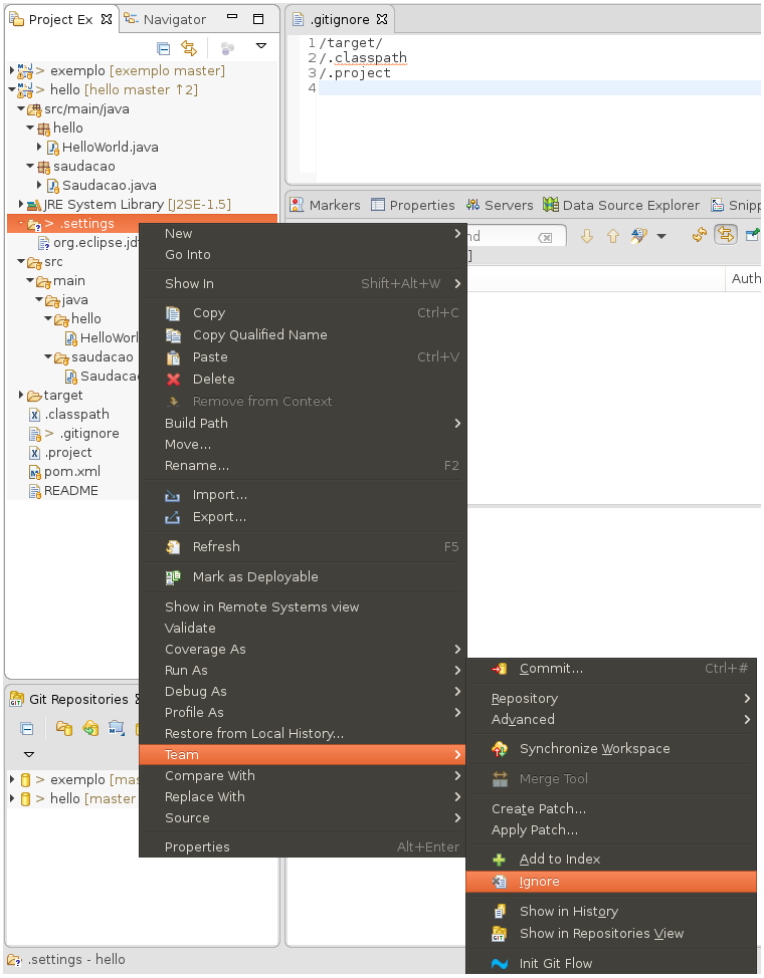
Têm-se duas maneiras simples de fazer isso. A primeira é editar o arquivo `.gitignore` e acrescentar uma linha ao final desse arquivo com o nome do arquivo a ser ignorado.

Acrescente os arquivos `.project` e `.classpath`, um por linha, lembrando de acrescentar uma `/` no começo, para indicar que é o arquivo no raiz do repositório.

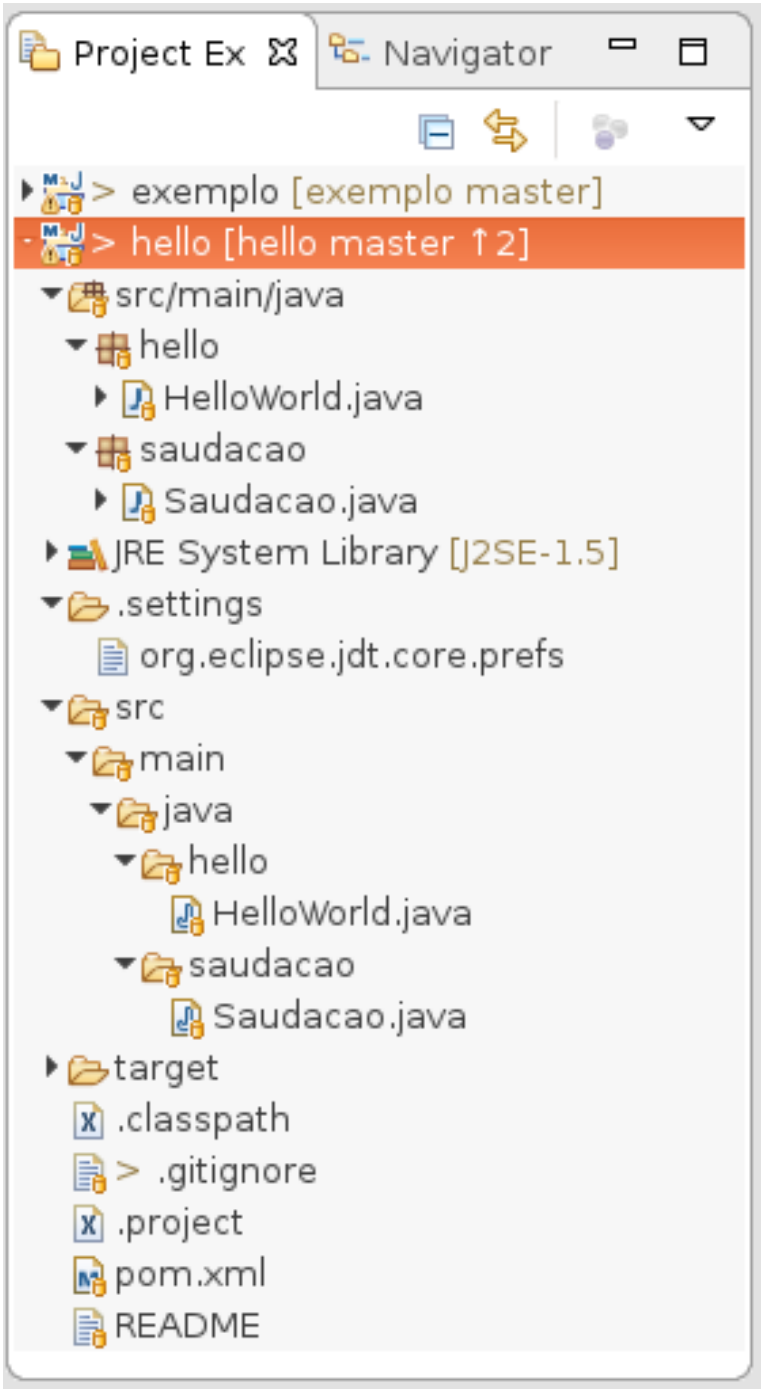
Dessa maneira o arquivo `.gitignore` ficará assim:

```
/target/
/.classpath
/.project
/.settings/
```

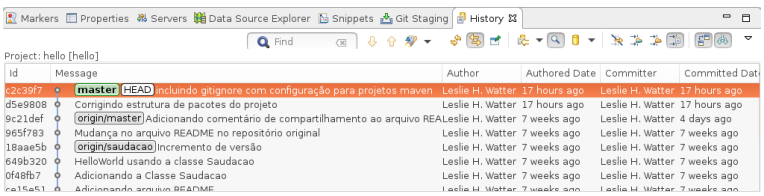
A outra maneira é selecionar o arquivo/diretório na view *Project Explorer* e acionar com o botão direito do mouse o menu *Team* > *Ignore*, conforme ilustrado a seguir:



O próprio eclipse se encarrega de salvar o arquivo para você. OBserve agora a view *Project Explorer* e veja que apenas o arquivo `.gitignore` encontra-se modificado no projeto *hello*.



As modificações que acabaram de ser feitas, referem-se ao arquivo `.gitignore`, que foi o último commit executado.



Como ainda não compartilhamos (`push`) esse histórico com nenhum outro repositório, podemos usar a opção `--amend` sem problemas. Caso já tivéssemos compartilhado o repositório, seria necessário criar um novo commit para não confundir o histórico dos outros desenvolvedores.

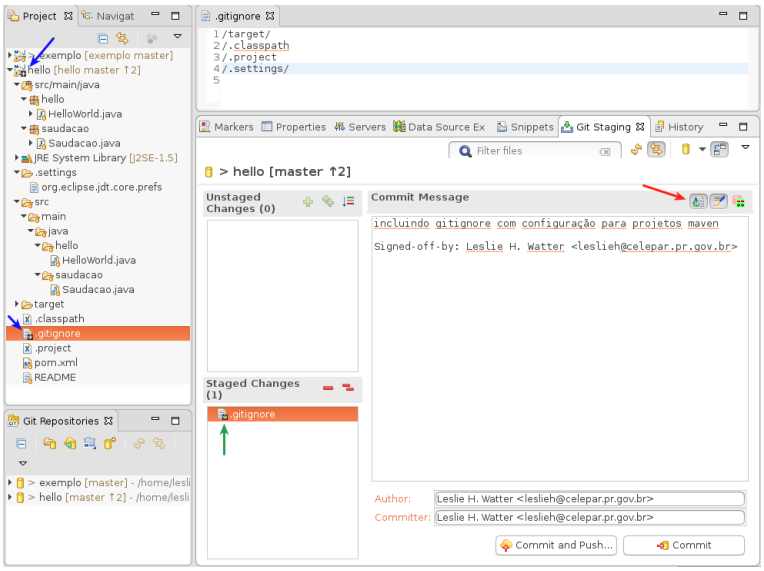
Iremos agora "corrigir"o último commit, acrescentando também as mudanças que acabaram de ser feitas.

As modificações poderiam ser em quaisquer arquivos do repositório, não necessariamente no mesmo arquivo que foi incluído anteriormente.

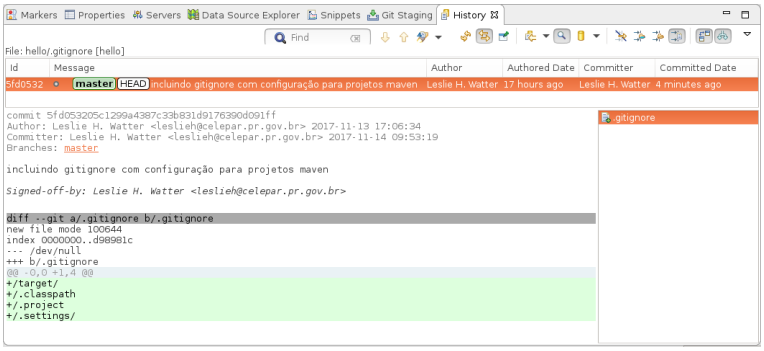
Lembre-se o git registra **mudanças** e não arquivos!

Para registrar as mudanças que acabaram de ser feitas, o processo é o mesmo. Acrescente-as ao *Index*, selecionando o arquivo `.gitignore` e acionando o menu *Team* > *Add to Index*. Observe que a marcação ao lado do arquivo e repositório passam para um asterisco preto, indicando que há modificações preparadas para o commit.

Alterne para a view *Git Staging* e observe que o arquivo `.gitignore` já está na seção *Staged Changes*. Clique no botão *Amend (Edit Previous Commit)* e o eclipse irá preencher a mensagem de commit com a mensagem utilizada no último commit.



1. Clique no botão *Commit*.
2. Alterne para a view *History*.
3. Clique no último commit e observe o diff na parte de baixo da view. Os arquivos `.classpath`, `.project` e o diretório `.settings` estão listados no último commit.



Observe que o hash desse commit mudou de `c2c39f73f3982bbe1af559df6d6d7052a17bdfa2` para `5fd053205c1299a4387c33b` indicando que houve mudanças no conteúdo.

9 Colocando mudanças "de lado temporariamente"(`stash`)

Uma das funcionalidades interessantes do git é poder armazenar as mudanças ainda não comitadas em uma área separada, chamada de *stash*, e continuar a trabalhar com o projeto como se nada tivesse sido modificado.

Edite o arquivo `Saudacao.java` para incluir um cumprimento de "bom dia", "boa tarde"e/ou "boa noite"na saudação, de maneira a refletir o código a seguir:

```
package saudacao;

import java.util.Calendar;

public class Saudacao {
   private String nome;
   private String cumprimento;
```

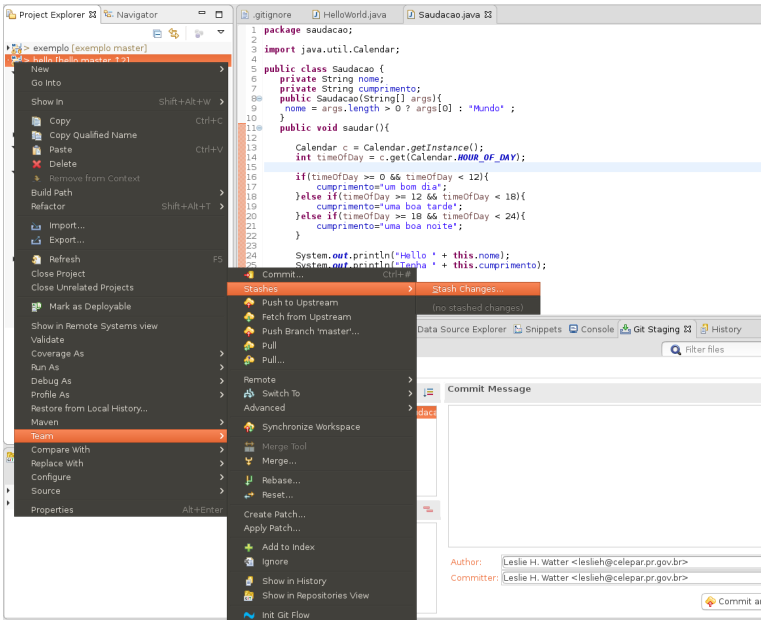
```
   public Saudacao(String[] args){
nome = args.length > 0 ? args[0] : "Mundo" ;
   }
   public void saudar(){

      Calendar c = Calendar.getInstance();
      int timeOfDay = c.get(Calendar.HOUR_OF_DAY);

      if(timeOfDay >= 0 && timeOfDay < 12){
  cumprimento="um bom dia";
      }else if(timeOfDay >= 12 && timeOfDay < 18){
  cumprimento="uma boa tarde";
      }else if(timeOfDay >= 18 && timeOfDay < 24){
  cumprimento="uma boa noite";
      }

      System.out.println("Hello " + this.nome);
      System.out.println("Tenha " + this.cumprimento);

   }
}
```

Você também pode aplicar o patch presente no arquivo `saudacao.patch`, listado a seguir. O arquivo de patch traz o benefício de listar as adições diretamente.

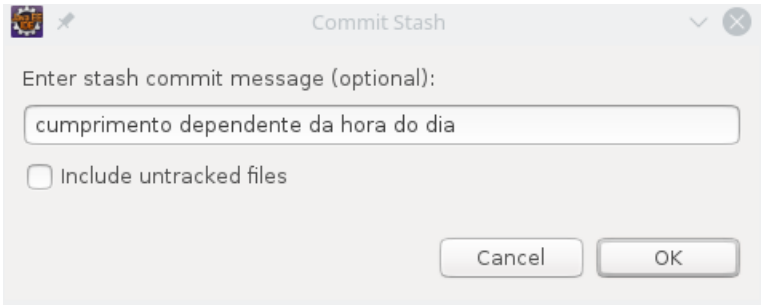
```
diff --git a/src/main/java/saudacao/Saudacao.java b/src/main/java/saudacao/Saudacao.java
index af38879..c8af2ae 100644
--- a/src/main/java/saudacao/Saudacao.java
+++ b/src/main/java/saudacao/Saudacao.java
@@ -1,10 +1,28 @@
 package saudacao;
+
+import java.util.Calendar;
+
 public class Saudacao {
    private String nome;
+   private String cumprimento;
    public Saudacao(String[] args){
nome = args.length > 0 ? args[0] : "Mundo" ;
    }
    public void saudar(){
+
+      Calendar c = Calendar.getInstance();
+      int timeOfDay = c.get(Calendar.HOUR_OF_DAY);
+
+      if(timeOfDay >= 0 && timeOfDay < 12){
+          cumprimento="um bom dia";
+      }else if(timeOfDay >= 12 && timeOfDay < 18){
+          cumprimento="uma boa tarde";
+      }else if(timeOfDay >= 18 && timeOfDay < 24){
+          cumprimento="uma boa noite";
+      }
+
       System.out.println("Hello " + this.nome);
+      System.out.println("Tenha " + this.cumprimento);
+
    }
 }
```

Entretanto, ao invés de comitar as modificações, percebemos que essas modificações ficariam melhor alocadas em um branch separado, de nome `cumprimento`.

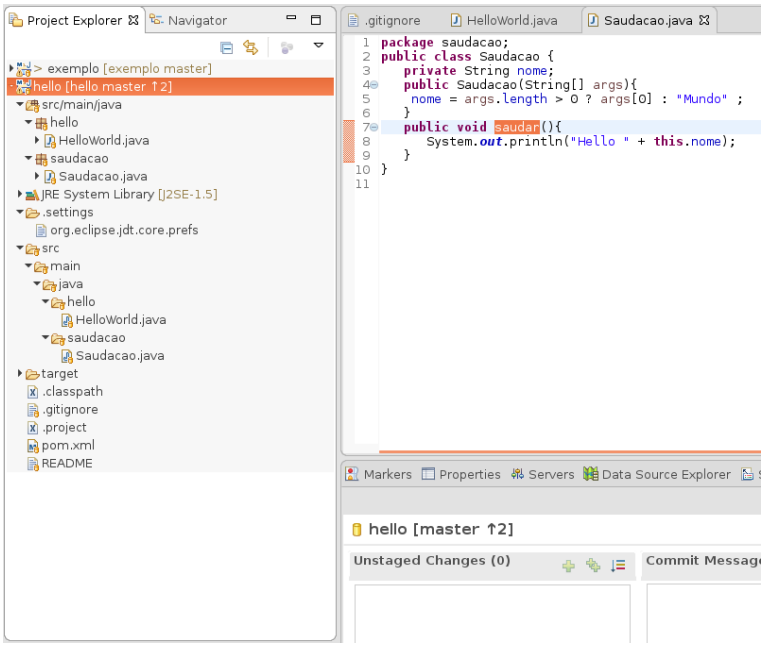
Para não perder suas modificações e poder voltar à situação anterior do commit, faça o `stash` dessas modificações selecionando o projeto e acionando o menu *Team* > *Stashes* > *Stash Changes...*.



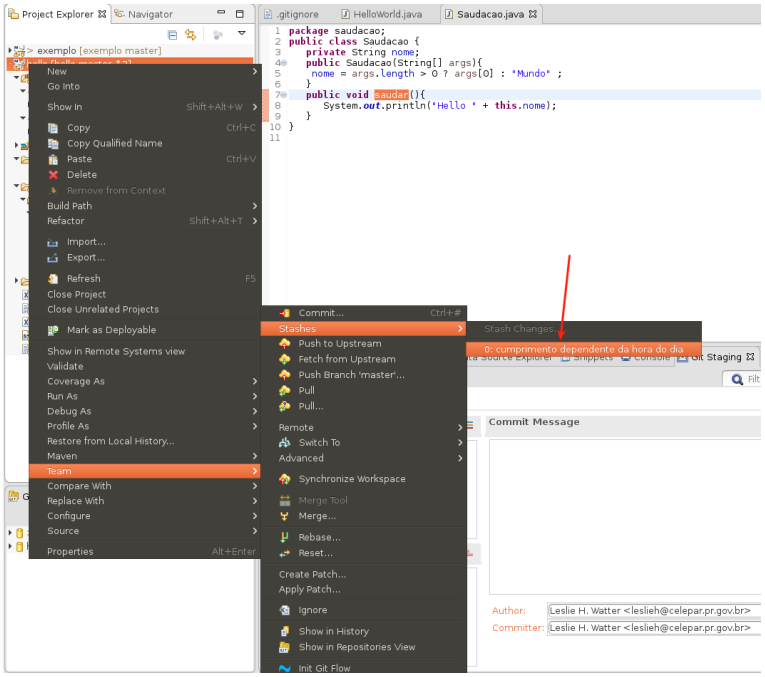
Informe uma mensagem de commit para o stash, por exemplo, `cumprimento dependente da hora do dia`. Essa mensagem irá identificar o conjunto de modificações.



Observe que as modificações feitas no projeto simplesmente "sumiram":



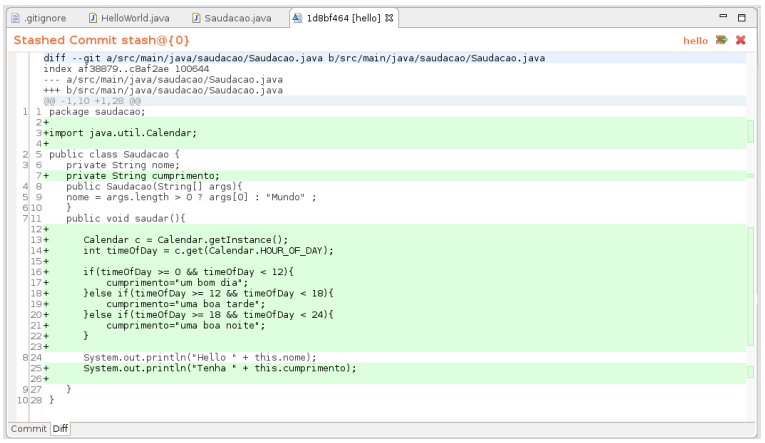
Porém, observando o menu *Team* > *Stashes*, é possível encontrar uma entrada: `0: cumprimento dependente da hora do dia`



Selecione a opção `0: cumprimento dependente da hora do dia` uma view com o resultado do stash será aberta. Nessa view é possível:

- (1) visualizar a mensagem de commit e onde foram feitas as modificações,
- (2) aplicar as mudanças,
- (3) descartar o stash.

e também as diferenças que essa mudança trará (selecionando a aba *Diff*):



Por hora feche essa view pois iremos fazer alguns ajustes na estrutura do projeto antes de aplicar essas modificações.

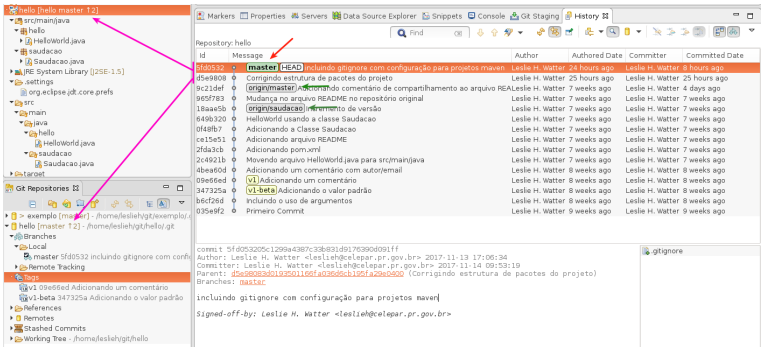
10 Versionamento usando Tags

Uma vez feita a correção na estrutura do projeto, é importante acrescentar um ponto de recuperação a esse momento. Para isso criaremos uma tag.

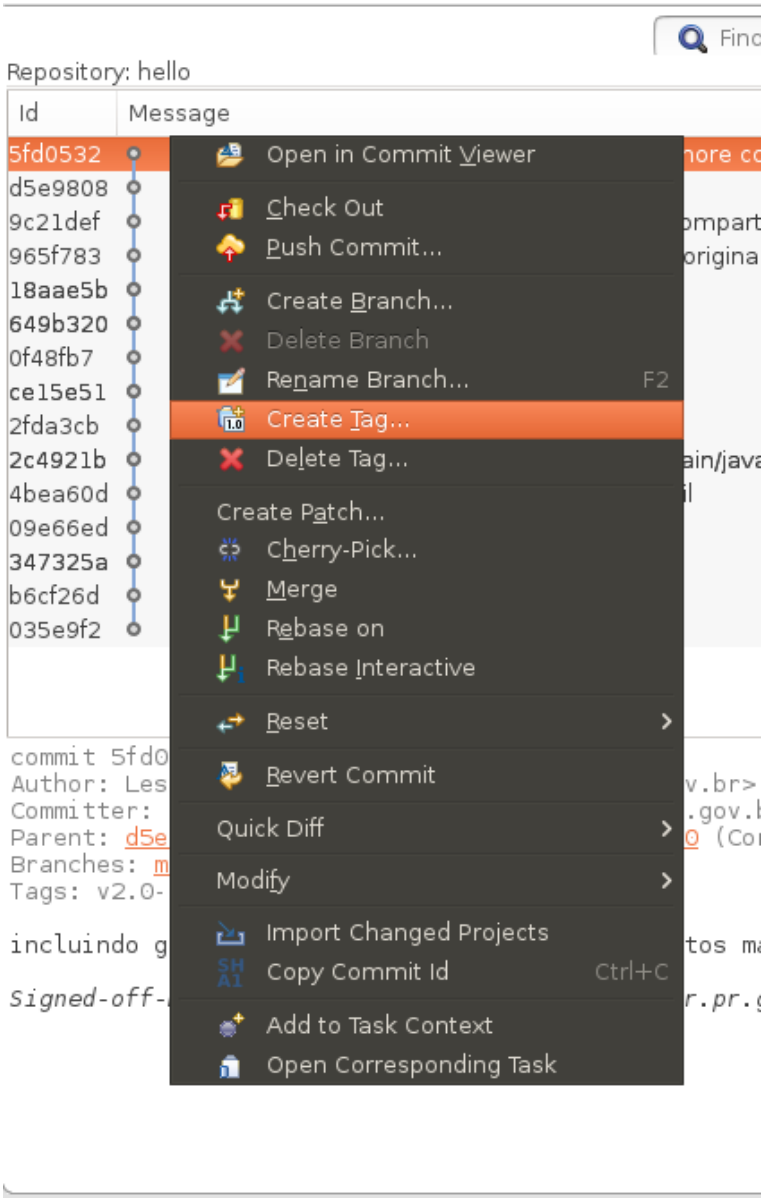
10.1 Criando uma tag

Abra para a view *History*. Observer que o último commit (incluindo gitignore com configuração para projetos maven) tem duas marcações: `master` e `HEAD`, indicando que estamos no branch `master` local. Observando dois commits abaixo, é possível ver a marcação `origin/master` e na sequência `origin/saudação`.

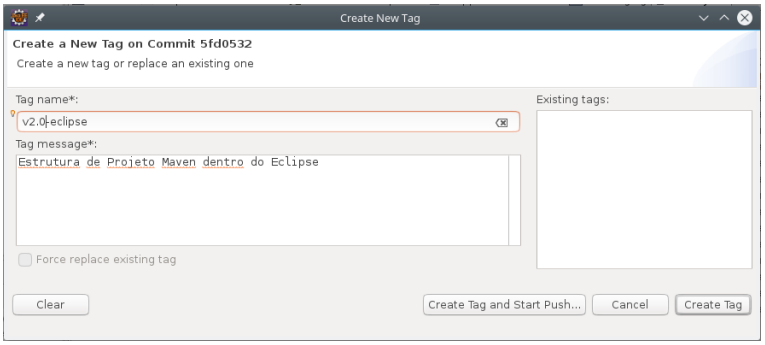
A marcação ao lado do branch tanto na view *Project Explorer* quanto *Git Repositories* indica que o repositório local está 2 commits à frente do repo



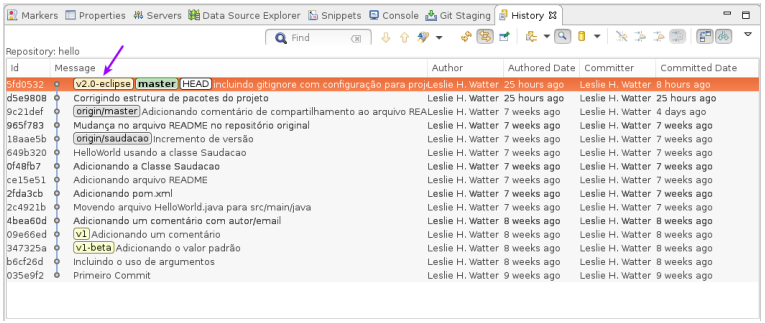
Para criar uma tag, clique sobre o commit ao qual a tag estará associada e acione o menu do botão direito do mouse *Create Tag...*



Especifique o nome da tag e a mensagem associada:



Pronto! A tag foi criada.



10.2 Substituindo uma Tag existente

O que fazer se você marcou com uma tag um commit errado ou acabou cometendo um erro de digitação?

- Se você ainda não publicou essa tag em um servidor, você pode substituir a tag e pronto.
- **Se você já publicou, não deve substituir a tag** e sim usar um novo nome uma vez que você teria que avisar a todos que receberam a tag antiga para substituirem **manualmente** com sua tag atualizada. Isso acontece porque **o git não muda as tags sem o usuário saber** (e não deve mudar). Então se alguém já recebeu a tag antiga, fazer um `pull` da sua árvore não irá sobrescrever a tag antiga.

O git **NÃO SOBRESCREVE TAGS** já publicadas em outros repositórios automaticamente.

Para substituir uma tag, estando na view *History* clique com o botão direito no commit em que deseja aplicar a tag e a seguir acione o menu *Create Tag...*. Selecione a tag existente na lista de tags e marque o item *Force replace existing tag*, a seguir clique no botão *Create Tag*.

10.3 Removendo uma tag

Para remover uma tag, basta clicar com o mouse sobre o botão direito sobre o commit e acionar a opção *Delete Tag...* que irá pedir confirmação para apagar a tag.

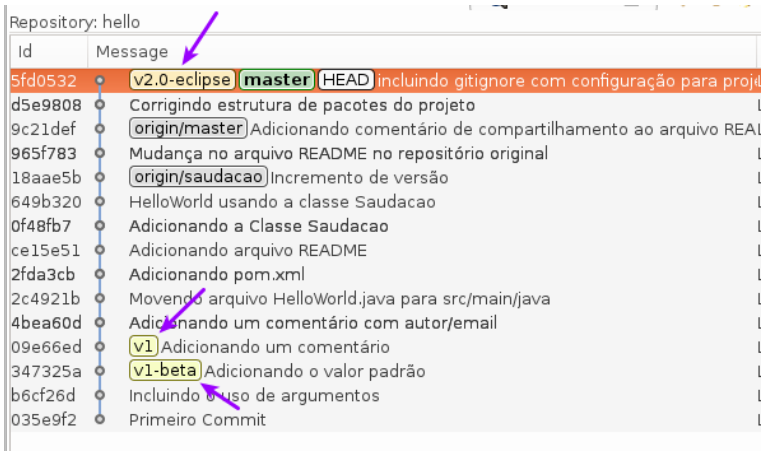
É uma prática extremamente RUIM apagar uma tag que já foi publicada em um servidor público. Alguns servidores inclusivem impedem a remoção de tags publicadas para garantir a rastreabilidade do que já foi marcado com tags.

10.4 Fazendo o checkout de uma tag

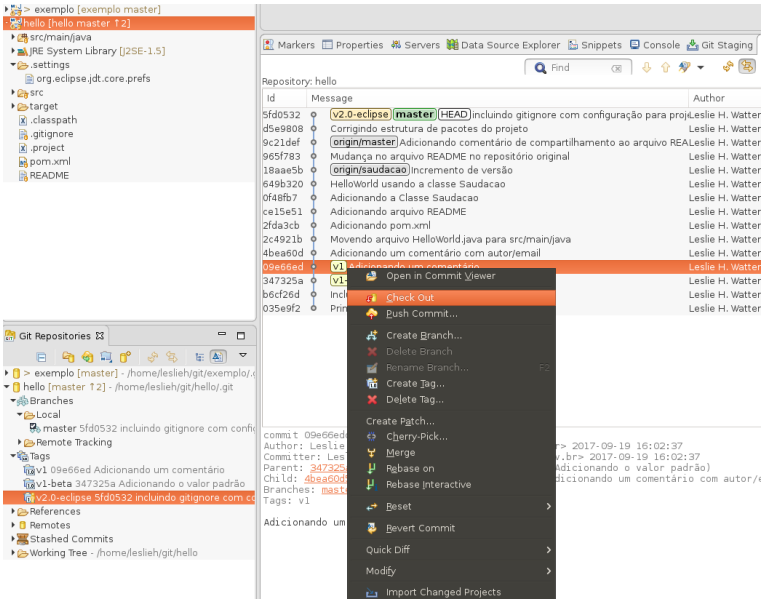
Na view *Project Explorer* acione o menu *Team* > *Show in History* para mostrar a view *History*.

Na view *History* são listadas todas as tags que já foram criadas. Na figura a seguir é possível observar as tags (em ordem da última para a primeira):

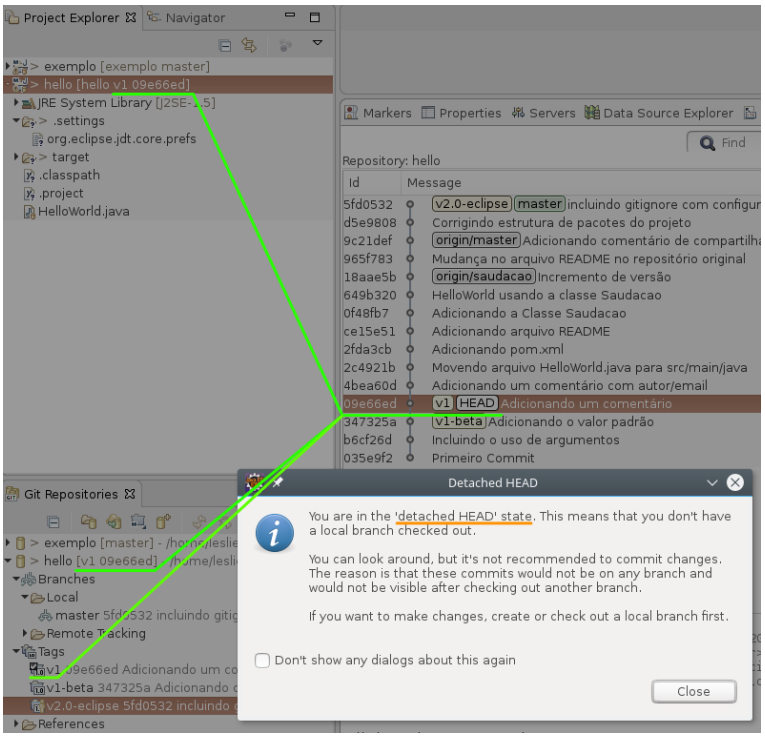
- v2.0-eclipse
- v1
- v1-beta



Para fazer o checkout de um commit marcado com uma tag, basta clicar com o botão direito do mouse sobre o commit com a tag e acionar a opção *Checkout* conforme ilustrado a seguir.



Observe que o simples checkout da tag/commit muda as marcações e também indica que o estado atual é *Detached HEAD*, onde o ponteiro *HEAD* não está associado a nenhum branch específico.



Nesse momento você pode explorar o repositório, porém não faça nenhum commit, uma vez que não está em nenhum branch nomeado. Veremos mais para frente o processo de criação de branches.

Observe que todas as modificações feitas para adequar o projeto à estrutura do eclipse não estão presentes, bem como o arquivo `.gitignore` também não aparece.

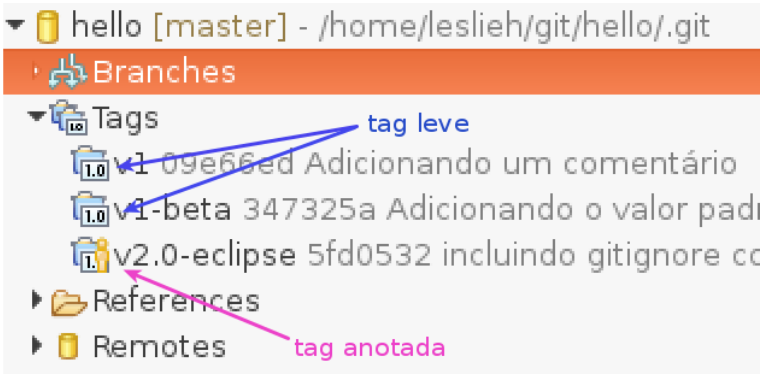
Agora volte o projeto para o branch *master* para que possamos prosseguir.

10.5 Tags Leves versus Tags Anotadas

Como vimos anteriormente, tags leves são simples ponteiros para commits, sem nenhuma informação adicional.

O eclipse por sua vez, utiliza por padrão tags anotadas, que contém mais informações, como autor, data, hora e mensagem de commit para a tag. Tags anotadas podem ser assinadas e verificadas (com GPG), sendo ideais para tags de liberação de versão.

O eclipse diferencia as tags leves das anotadas colocando um 'bonequinho dourado' ao lado de uma tag anotada.



Ao pressionar a sequência de teclas `Ctrl+Shift+L` o eclipse mostrará uma janela com a lista de teclas de atalho configuradas.

Pressionando novamente `Ctrl+Shift+L` o eclipse abre a configuração para permitir a edição das teclas de atalho associadas a comandos.

11 Publicando (`push`) commits e tags no repositório remoto

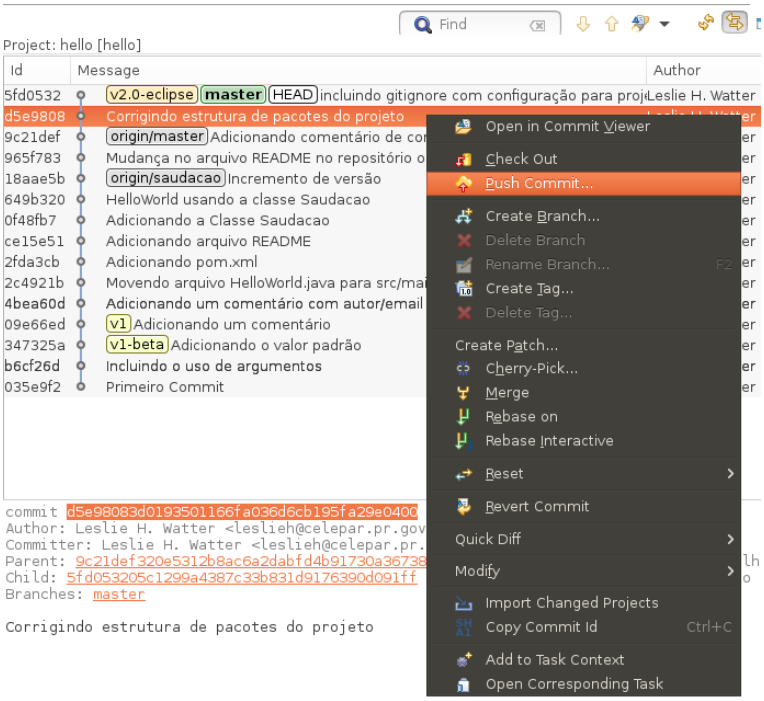
Objetivo: nosso objetivo é atualizar o quanto antes o branch `origin/master` no servidor com as modificações que acabamos de fazer.

Podemos publicar commits, tags e branches no repositório remoto, nesse momento iremos publicar os commits e a tag que criamos de maneira separada apenas a título de exemplo. As operações a seguir poderiam resumir-se na publicação da tag e/ou do último commit feito.

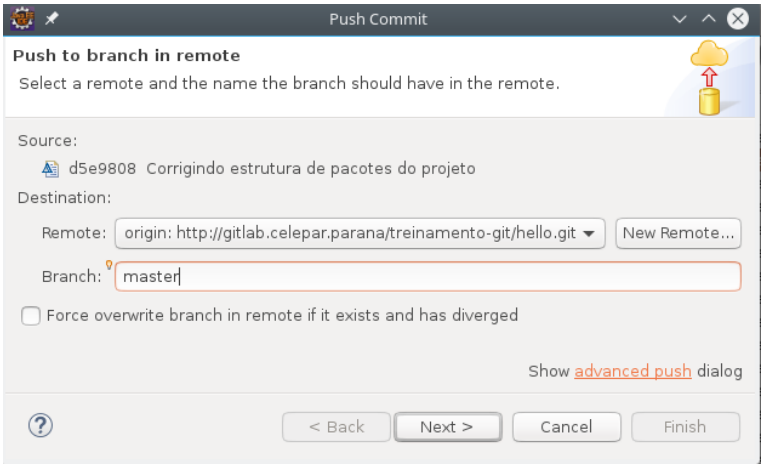
11.1 Publicando commits

Iremos atualizar o branch `origin/master` com a correção feita no commit *Corrigindo Estrutura de pacotes do projeto* ou `d5e9808`.

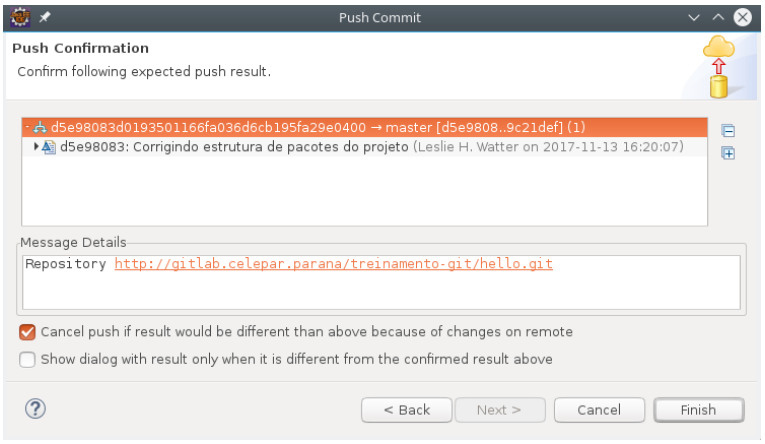
Para isso clique sobre o commit e acione o menu *Push Commit*.



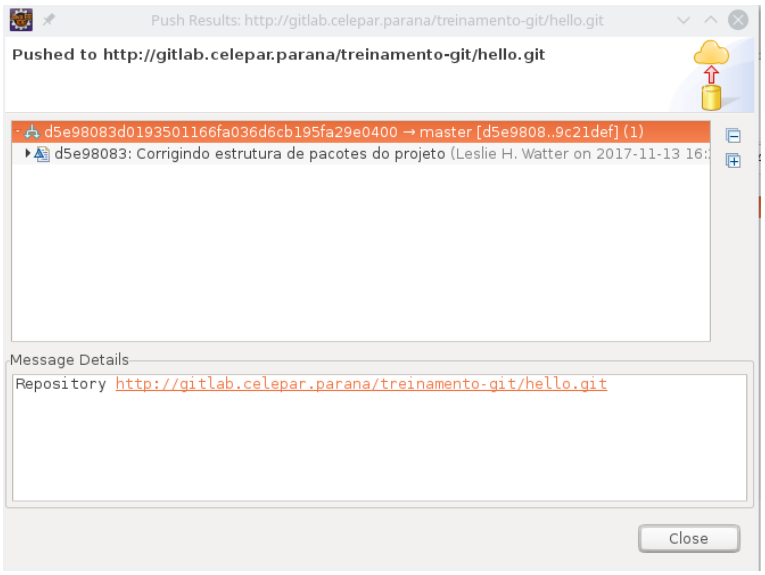
A seguir, informe o branch `master` (observe que `origin` está implícito pois estamos fazendo o push para o servidor) na janela que segue, informando seu usuário e senha quando solicitado.



Clique no botão *Next* e confirme o destino. Marque a opção *Cancel push if result would be different than above because of changes on remote*, para garantir que não irá sobrescrever nenhum código no servidor.

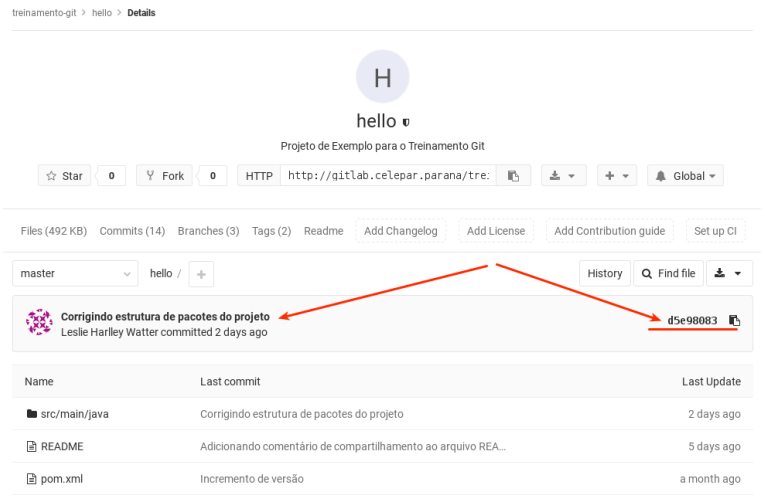


A mensagem de resposta, confirmando que o commit foi publicado:



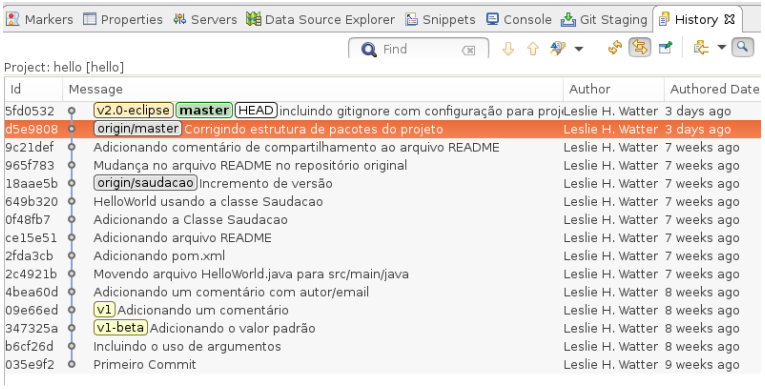
Apenas a título de curiosidade, visite o link do repositório no gitlab para confirmar a operação. Para facilitar: `http://gitlab.celepar.parana/treinamento-git/hello`

Observe que ao abrir o link acima, a última modificação aparece logo no início.



Volte para o eclipse.

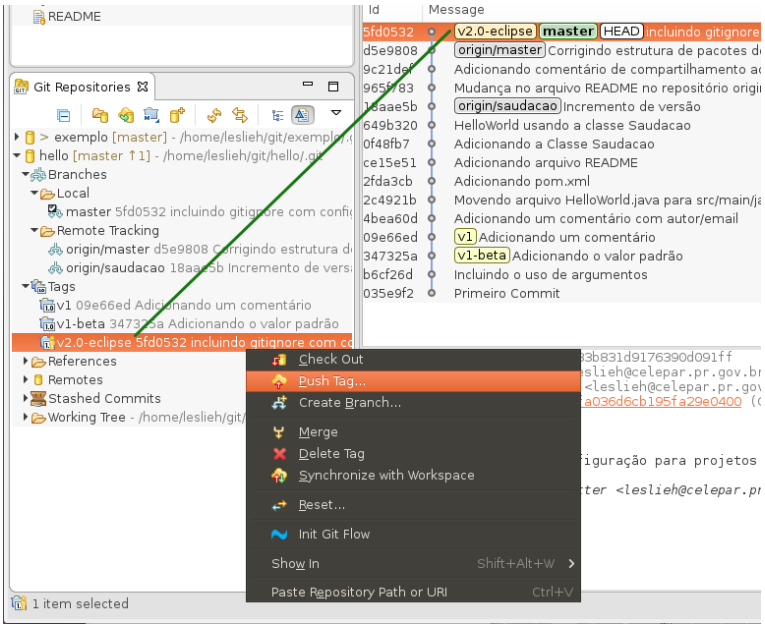
Observe que a marcação `origin/master` mudou para o commit que foi feito o push, indicando que o branch remoto foi atualizado.

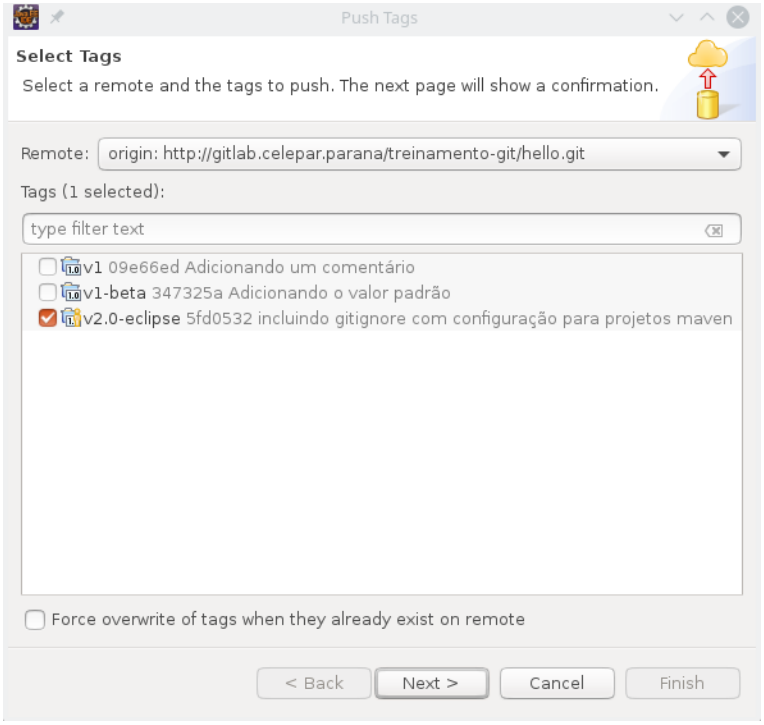


11.2 Publicando uma tag no servidor

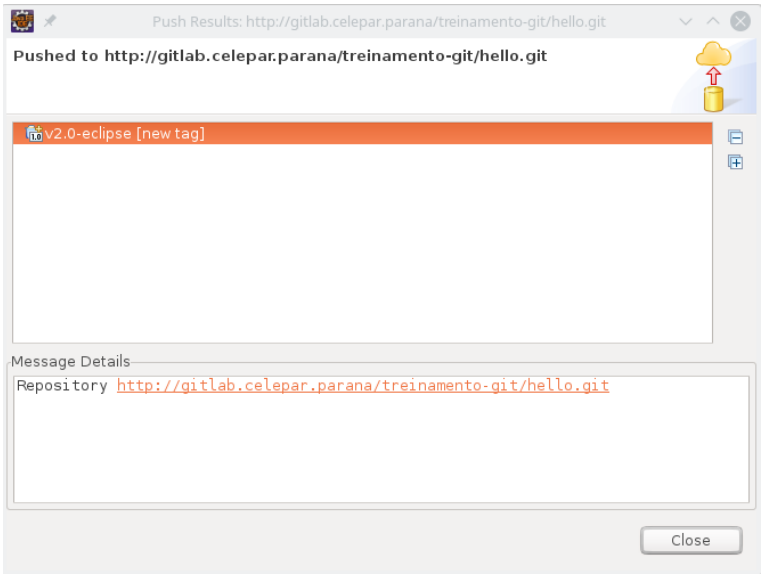
Para publicar a tag *v2.0-eclipse* que foi criada, iremos utilizar a view *Git Repositories*. Observe que a tag aparece na árvore do repositório, associada ao commit `5fd0532`.

Para publicar essa tag, basta clicar com o botão direito sobre ela e acionar o menu *Push Tag...* conforme ilustrado a seguir.

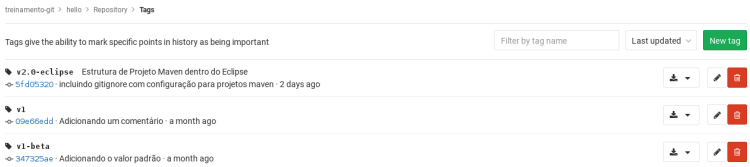




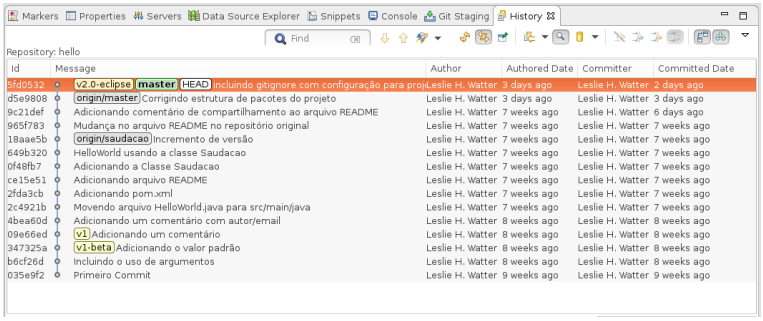
O eclipse confirmará a criação da nova tag:



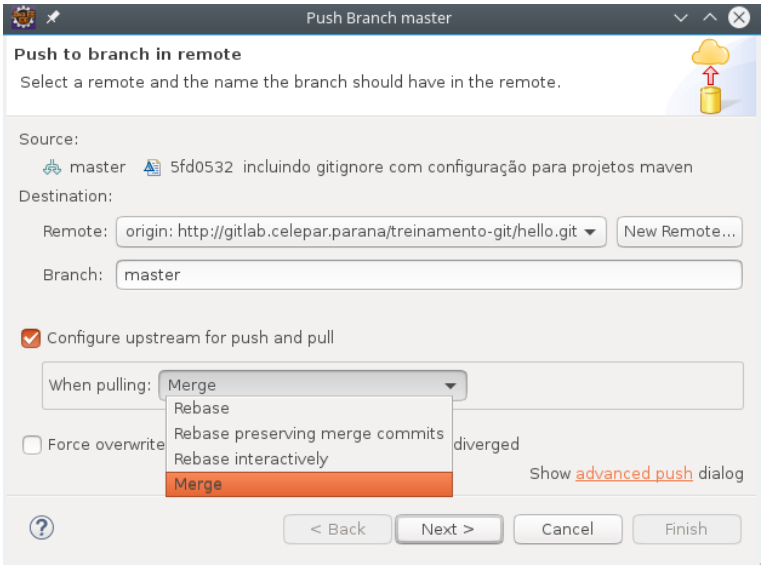
A título de exemplo, para confirmar a criação da tag, acesse o repositório do gitlab no endereço: `http://gitlab.celepar.parana/treinamento-git/hello/tags`



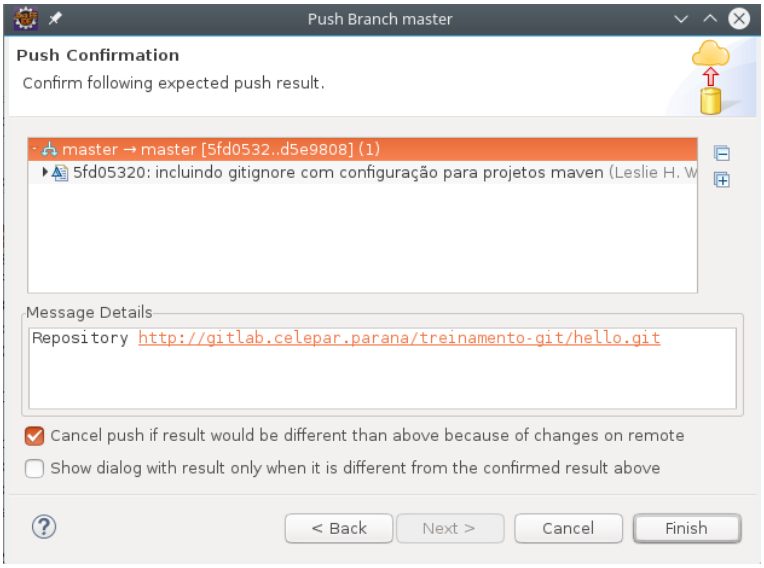
Voltando ao eclipse, observe que não houve modificação na view *History* após o push da tag. A marcação do branch `origin/master` continua no commit anterior.



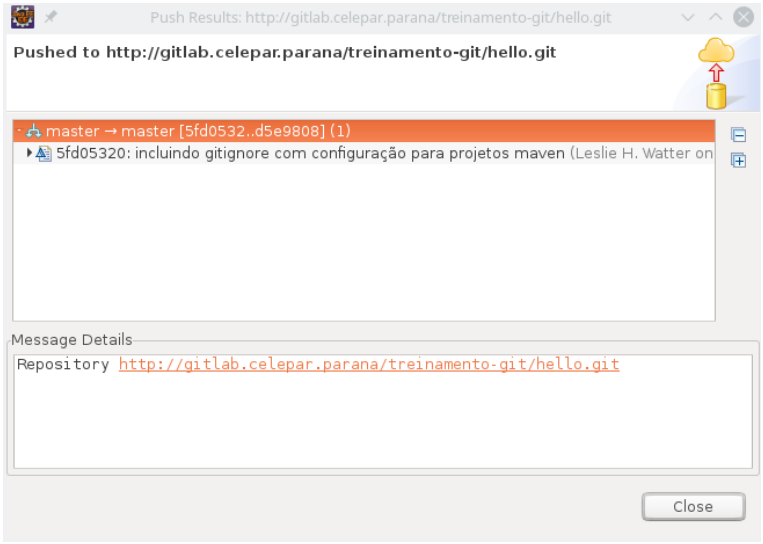
Ao publicar o commit com a tag no servidor do gitlab, o eclipse permitirá configurar a opção de upstream para push e pull. Mantenha a opção *Merge* e clique no botão *Next*.



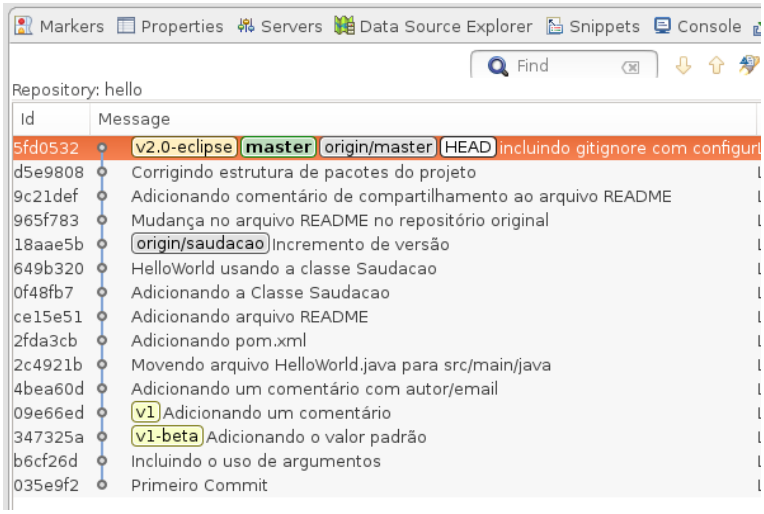
O eclipse pedirá a confirmação da operação antes de executá-la. Marque a opção *Cancel push if result would be different than above because changes on remote* e clique no botão *Finish*.



O retorno indica que a operação foi bem sucedida.

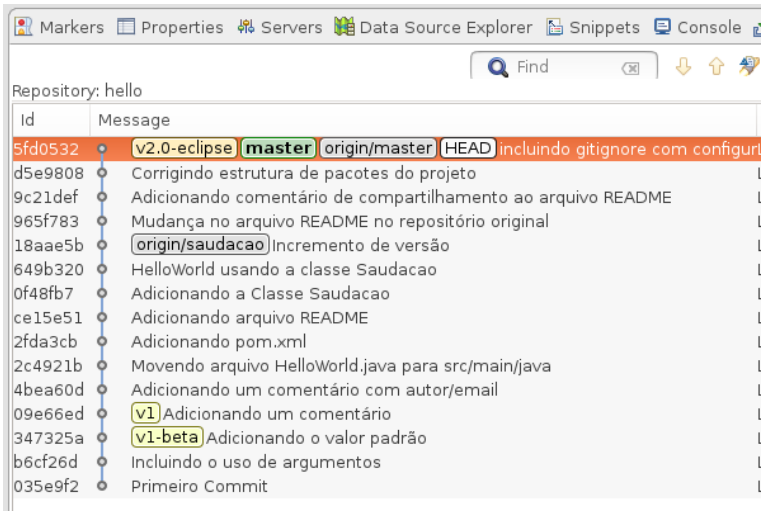


E a view *History* apresenta tanto a tag *v2.0-eclipse*, quanto os branches `master` e `origin/master` quanto o apontador `HEAD` no commit `5fd0532`, indicando que a operação foi completada.



12 Trabalhando com Branches

12.1 Criando um novo Branch a partir de uma Referência Remota



Observando o histórico, vemos que o branch `origin/saudacao` ficou para trás alguns commits. Nosso objetivo é atualizá-lo de modo a refletir as mudanças que fizemos para corrigir a estrutura do projeto.