Manual prático de utilização do Git

# Conceitos

* **Versionar**: ato de armazenar diferentes estados
* **Repositório**: local onde todas as versões dos arquivos serão armazenadas;
* **Versão**: valor que representa um estado/tempo específico de um arquivo;
* **Revisão**: é um tipo de versão que é atribuída a um arquivo quando ele sofre algum tipo de alteração;
* **Workspace/Work directory**: espaço temporário para manter uma cópia local do trabalho que está sendo modificada. Neste momento o arquivo pode estar versionado ou não;
* **Local repo**: repositório utilizado para versionar os arquivos localmente;
* **Remote repo**: repositório utilizado para versionar os arquivos remotamente;
* **Branch**: versão paralela do arquivo principal. Pode ser paralela ou remota;
* **Master**: branch principal de desenvolvimento.
* **Tag**: marcação em determinado momento para identificação de um instante específico no ciclo de vida dos arquivos que estão sendo versionados. Normalmente é utilizado para identificar que uma versão foi finalizada;
* **HEAD**: identifica qual o branch que está sendo utilizado no momento;
* **Clone**: cópia local (repositório local) idêntica ao repositório remoto;
* **Commit**: envia as alterações feitas em um arquivo no espaço de trabalho para o repositório local;
* **Pull**: Atualiza o repositório local com eventuais alterações que foram feitas no repositório remoto depois de ser clonado
* **Push**: envia para o repositório remoto as alterações que foram commitadas para o repositório local e ainda não foram submetidas;
* **Fetch**: similar a o Pull;
* **Merge**: mescla as alterações feitas em branches diferentes;
* **Rebase**: similar ao Merge, porém reorganiza os commits feitos de forma a melhorar o entendimento da sequencia de alterações;
* **Diff**: compara as alterações feitas por duas pessoas diferentes. Normalmente utilizado quando o mesmo arquivo é alterado ao mesmo tempo por pessoas distintas;

# Fluxo básico

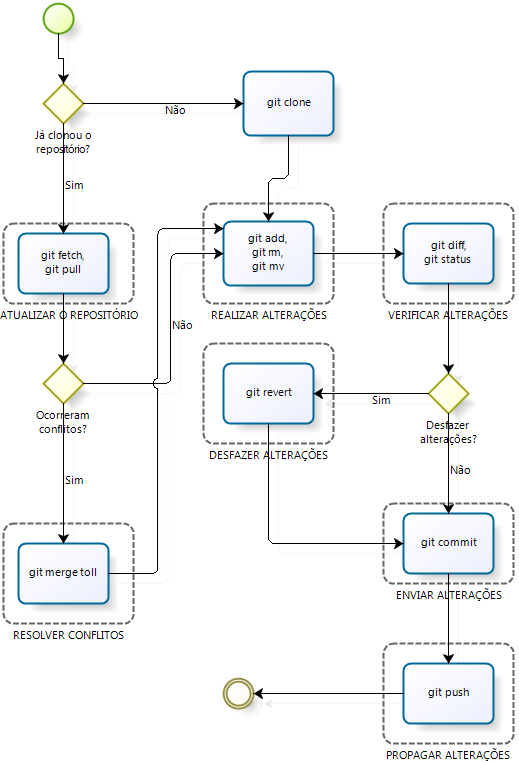
A característica mais marcante do Git, e, portanto, a que deve ser evidenciada logo de início é sua arquitetura que foi desenhada de forma que ele trabalhe de forma distribuída e não centralizada como em muitos outros controladores de versão. Apenas dessa não ser uma característica única do Git e nem mesmo pioneira, hoje certamente é a ferramenta distribuída mais utilizada no mercado.

A principal característica de um SCV distribuído começa na máquina do desenvolvedor. Diferente de ferramentas centralizadas (como o SVN), onde tempos um servidor e todos os membros da equipe enviam suas alterações e também atualizam seus códigos baseados no repositório central. Em controladores de versão distribuídos, membro da equipe possui seu próprio repositório local onde faz seu próprio controle de versão, criando seus próprios branches e organizando seu trabalho da melhor maneira possível. Como ter o código num único lugar, mesmo estando versionado, sempre existe o erro de perde de informação, portanto é utilizado um paradigma semelhante aos controladores de versão centralizados. Nesse caso, é criado (em um servidor na rede), um repositório onde os códigos de todos os repositórios locais são sincronizados. Ou seja, é criando um repositório remoto para que as informações sejam armazenadas em um segundo ambiente. Dessa forma o desenvolvedor tem uma autonomia maior no controle de versão, visto que ele pode criar branches (locais) como achar melhor e assim se organizar antes de submeter suas alterações para o repositório remoto.

O fluxo de trabalho loca é dividido em três estados muito bem definidos. A primeira fase, conhecida como Working Directory, representa a fase em que o arquivo está em processo de alteração pelo desenvolvedor. A segunda fase é conhecida como Staging Area e representa o momento onde o responsável pelo repositório local escolhe quais arquivos alterados ele quer versionar em seu repositório. Lembrando que os arquivos (novos, editados ou excluídos), por convenção não são adicionados automaticamente na Staging Area, portanto o próprio usuário da ferramenta deve indicar quais arquivos serão versionados. Por fim, a terceira fase que é o Repositório local propriamente dito. A partir do momento que os arquivos são commitados, eles jã estão sendo versionados e, portanto, aquela ação específica já faz parte de uma revisão que no Git é uma informação alfanumérica (ex.: ca82a6d).

C:\Users\mollivier\Desktop\Untitled Diagram.png

A partir do momento que temos alterações feitas em nosso repositório local, é possível que enviemos nossas alterações para o repositório remoto através de um push. Vale lembrar que nesse momento estamos enviando informação para um repositório onde, provavelmente outras pessoas também enviaram informações, portanto, é recomendado que antes de enviar as suas alterações, seja feito um pull/fetch para verificar se não houve nenhuma informação alterada nesse meio tempo de desenvolvimento. Caso haja algum conflito, basta resolver localmente e depois disso enviar as alterações para o repositório remoto.



O que é muito comum de acontecer em equipes de desenvolvimento é trabalhar diretamente na linha principal de trabalho (conhecida como master ou trunk, dependendo do controlador que está sendo utilizado). Apesar de não ser considerado errado, este é um hábito que não é estimulado, pois é dado como uma boa prática deixar o código do master sempre estável de forma que possa ser compilado, testado e até mesmo tornar uma versão para ser disponibilizadas para o cliente. Principalmente em ambientes que já existe um sistema de integração ou até mesmo entrega contínua onde em alguns casos o sistema controlador de versão é monitorado para ver se houve alguma alteração no código, é preciso que seja garantido o mínimo de qualidade no código que está sendo mesclado com esse branch.



# Utilização

Normalmente a forma mais comum de se utilizar o Git é através de linhas de comando no terminal. Entretanto essa não é a única forma de utilização da ferramenta. Existem diversos clientes (gratuitos e pagos) que ajudam, de forma gráfica, a execução de todas as possíveis ações que a ferramenta oferece.

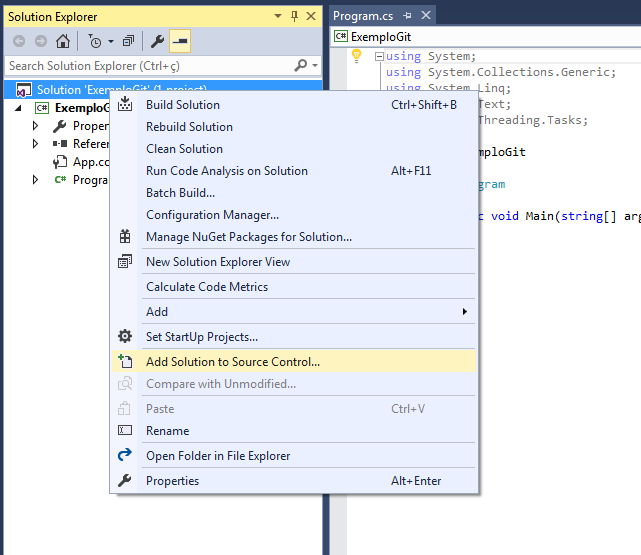
Para esse guia será levado em conta que o ambiente utilizado baseado em alguma versão do MS Windows, porém a maioria das atividades e ferramentas que serão mostradas independe de Sistema Operacional.

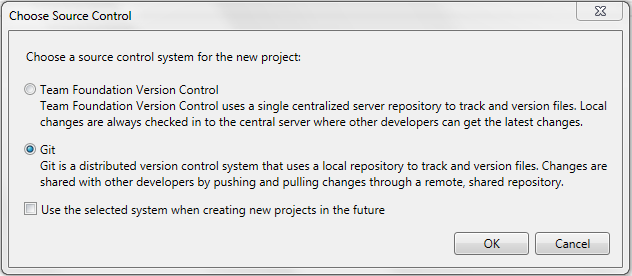
Devido à demanda de utilização tanto em projetos em Java quanto em projetos desenvolvidos para plataforma Microsoft, os exemplos serão feitos utilizando: Visual Studio 2015, Eclipse e a ferramenta gráfica padrão do Git (Git GUI). Ocasionalmente também será apresentado o cliente da Atlassian SourceTree por ser um cliente Git muito fácil de usar e bastante completo, além de mostrar graficamente a evolução do código que está sendo versionado.

## **Visual Studio 2015**

### Criando um repositório local

Para dar início aos trabalhos, vamos levar em conta que estamos começando um projeto novo e que a única coisa que já temos é um **repositório** **remoto vazio**. Sendo assim a primeira atividade que deve ser feita é a criação de um **repositório local**.





### Ligando a um repositório remoto

Como já existe um repositório remoto vazio e agora também existe um repositório local. O próximo passo que deve ser feito a criação de uma ligação entre os dois repositórios para que o código que está sendo versionado localmente possa ser compartilhado na rede.

## 

### Clonando um repositório remoto

A outra forma de iniciar um trabalho de versionamento de código usando Git seria se já existisse um projeto no repositório remoto. Esse cenário é, talvez, o mais comum dos casos, pois isso acontece quando vamos dar continuidade a um trabalho que tinha sido iniciado anteriormente. Nesse caso o que será feito é um **clone do repositório remoto**. Quando um repositório remoto é clonado, o usuário que fez o clone passa a ter um **repositório local idêntico ao repositório que foi clonado**. Vale lembrar que quando se clona um repositório remoto, ele automaticamente já possui uma ligação entre o local e o remoto.

## 

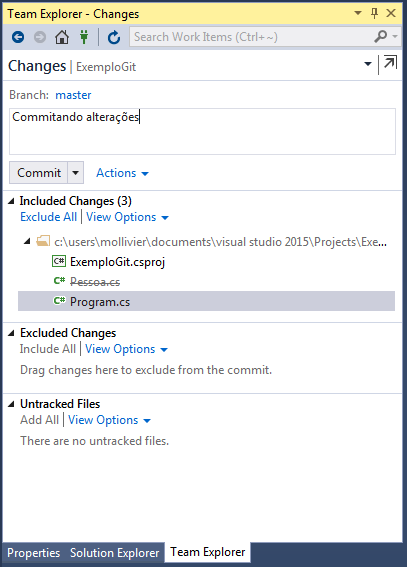
### Verificando o status dos arquivos do repositório

Como já foi visto inicialmente, quando foi falado sobre o ciclo de vida dos arquivos versionados, um arquivo não é adicionado automaticamente a ***staging area*** de modo que é necessário indicar quais arquivos devem ser adicionados. Para saber quais arquivos foram adicionados, alterados ou excluídos é necessário ver o ***status*** dos arquivos que estão no *working directory* do repositório.

## C:\Users\mollivier\Desktop\tutorial_git\imagens\print_staging_area.png

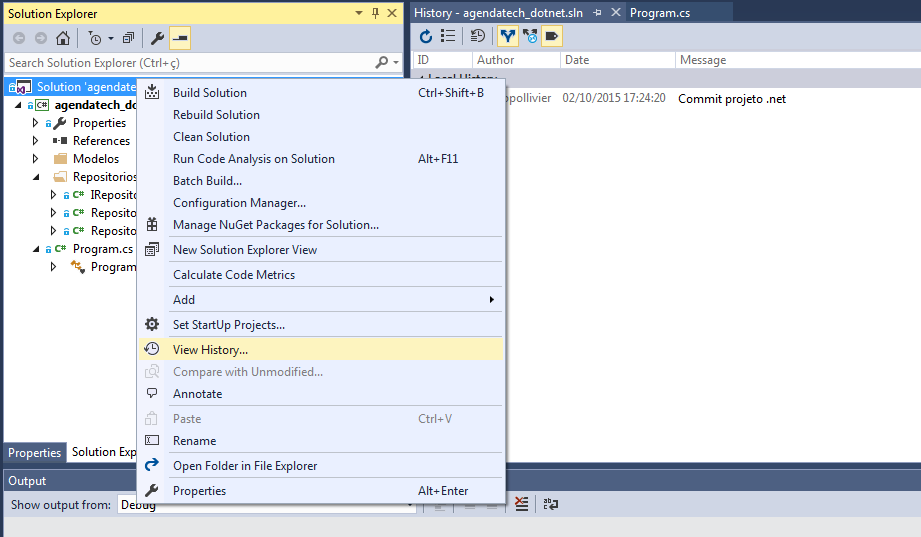
### Gravando o arquivo no repositório

A partir do momento em que um ou mais arquivos foram adicionados a *staging area*, os mesmos já estão sendo rastreados pelo Git e, portanto é possível gravar efetivamente no repositório local. O nome dessa ação é ***commit***. Vale lembrar que, no Git, todos os *commits* precisam ter uma mensagem descrevendo o que foi feito.



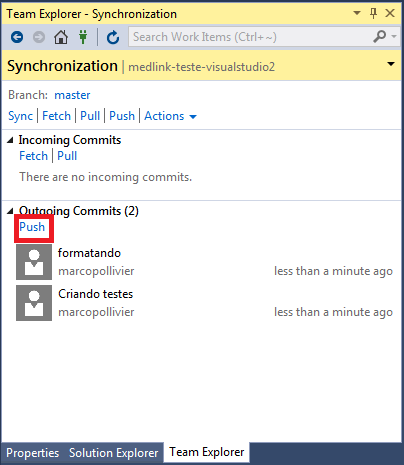
### Verificando o histórico de alterações

Para verificar todo o histórico de alterações que foram feitas no repositório local, é possível ver através do log do Git.

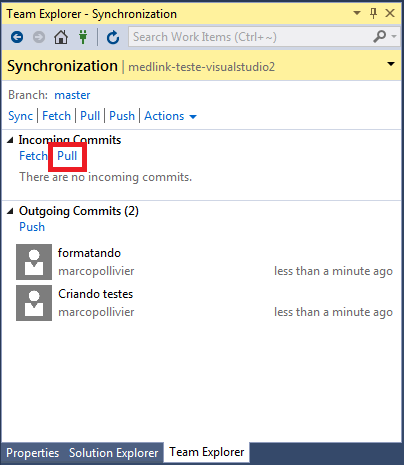


### Enviando alterações para um repositório remoto

Como já existe um repositório ligado ao repositório local que foi criado. Agora é o momento de enviar para a rede todas as alterações que foram feitas localmente. Para isso, será utilizado o ***push***.



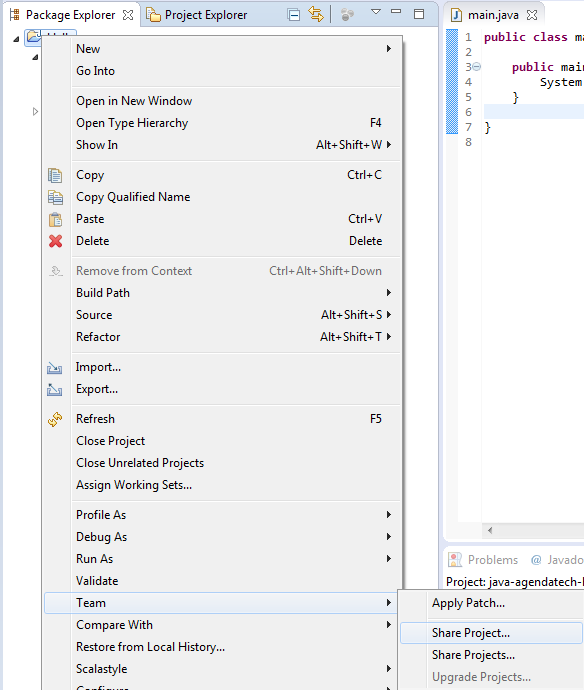
Lembrando que como a partir do momento em que um repositório remoto está sendo utilizado, acredita-se que não existe apenas uma única pessoa fazendo alterações nesse repositório remoto. Sendo assim, para que não haja conflito com eventuais trabalhos feitos por outras pessoas, é recomendado que antes do *push*, seja feito um ***pull*** para pegar eventuais alterações feitas por terceiros.

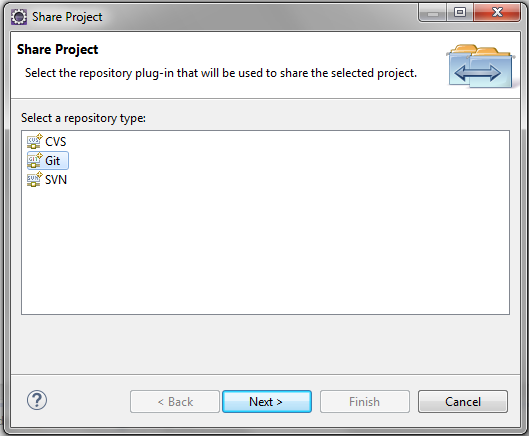


## Utilizando no Eclipse

### Criando um repositório local

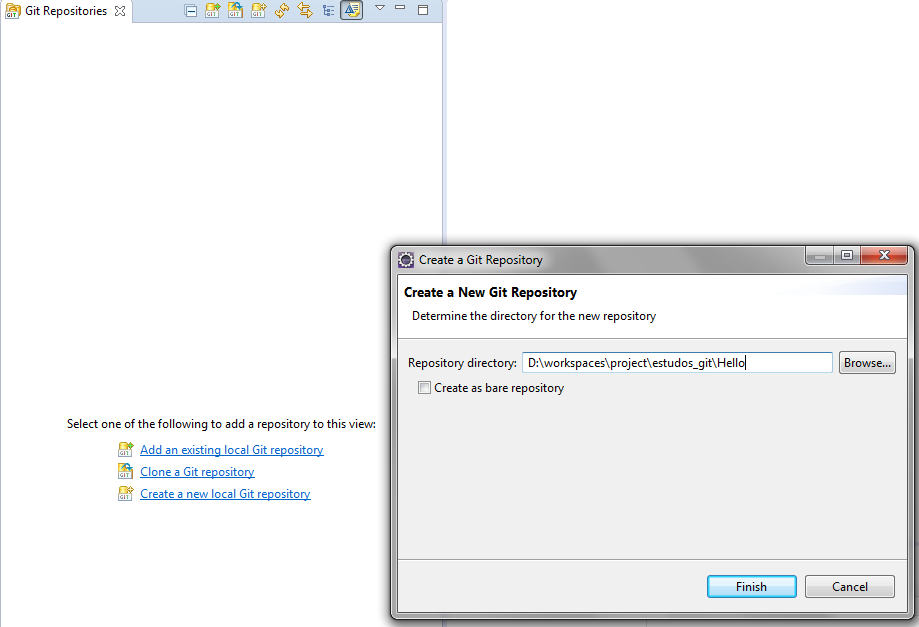
Para dar início aos trabalhos, vamos levar em conta que estamos começando um projeto novo e que a única coisa que já temos é um **repositório** **remoto vazio**. Sendo assim a primeira atividade que deve ser feita é a criação de um **repositório local**.





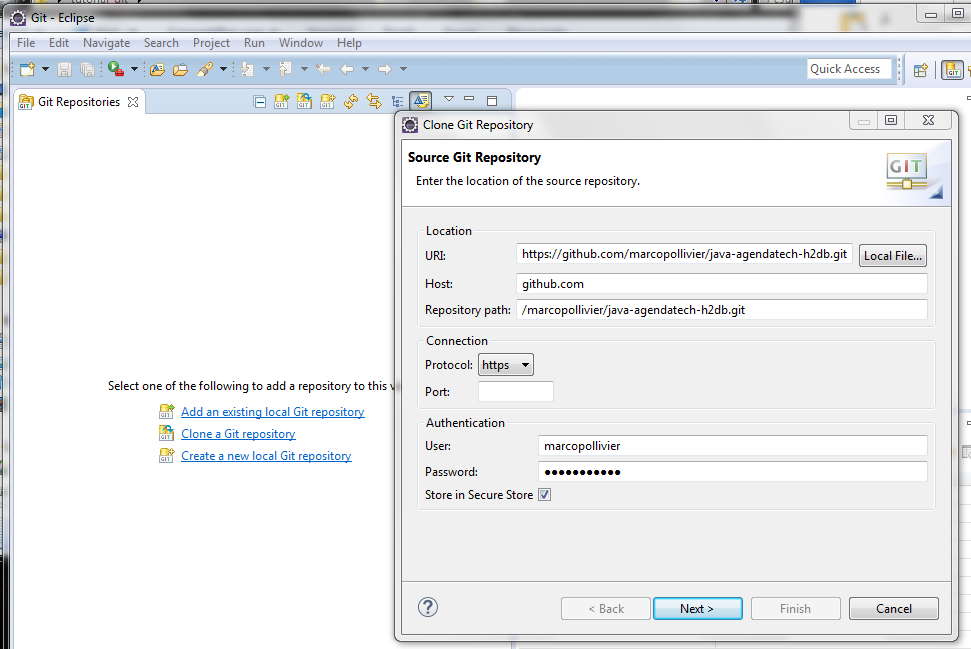
### Ligando a um repositório remoto

Como já existe um repositório remoto vazio e agora também existe um repositório local. O próximo passo que deve ser feito a criação de uma ligação entre os dois repositórios para que o código que está sendo versionado localmente possa ser compartilhado na rede.



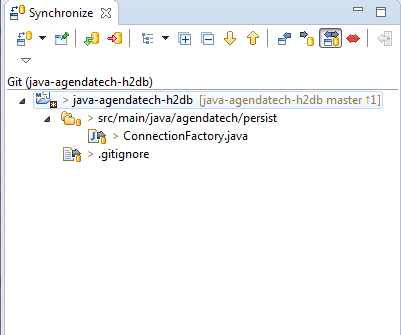
### Clonando um repositório remoto

A outra forma de iniciar um trabalho de versionamento de código usando Git seria se já existisse um projeto no repositório remoto. Esse cenário é, talvez, o mais comum dos casos, pois isso acontece quando vamos dar continuidade a um trabalho que tinha sido iniciado anteriormente. Nesse caso o que será feito é um **clone do repositório remoto**. Quando um repositório remoto é clonado, o usuário que fez o clone passa a ter um **repositório local idêntico ao repositório que foi clonado**. Vale lembrar que quando se clona um repositório remoto, ele automaticamente já possui uma ligação entre o local e o remoto.



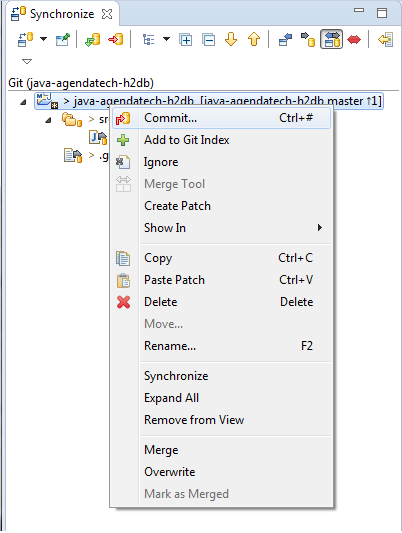
### Verificando o status dos arquivos do repositório

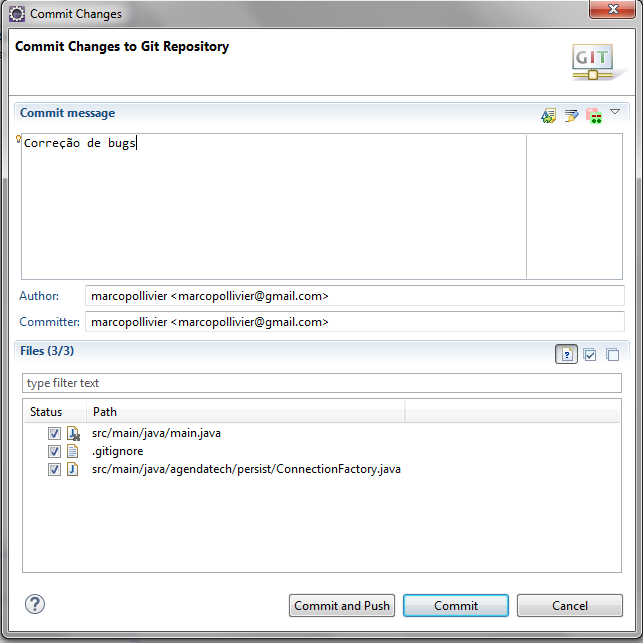
Como já foi visto inicialmente, quando foi falado sobre o ciclo de vida dos arquivos versionados, um arquivo não é adicionado automaticamente a ***staging area*** de modo que é necessário indicar quais arquivos devem ser adicionados. Para saber quais arquivos foram adicionados, alterados ou excluídos é necessário ver o ***status*** dos arquivos que estão no *working directory* do repositório



### Gravando o arquivo no repositório

A partir do momento em que um ou mais arquivos foram adicionados a *staging area*, os mesmos já estão sendo rastreados pelo Git e, portanto é possível gravar efetivamente no repositório local. O nome dessa ação é ***commit***. Vale lembrar que, no Git, todos os *commits* precisam ter uma mensagem descrevendo o que foi feito.





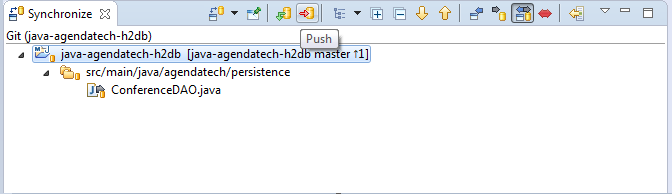
### Verificando o histórico de alterações

Para verificar todo o histórico de alterações que foram feitas no repositório local, é possível ver através do log do Git.

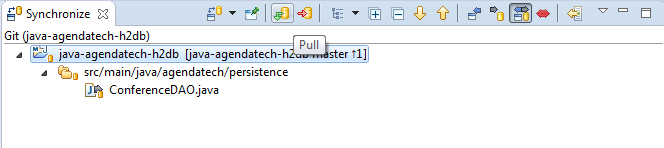
## 

### Enviando alterações para um repositório remoto

Como já existe um repositório ligado ao repositório local que foi criado. Agora é o momento de enviar para a rede todas as alterações que foram feitas localmente. Para isso, será utilizado o ***push***.



Lembrando que como a partir do momento em que um repositório remoto está sendo utilizado, acredita-se que não existe apenas uma única pessoa fazendo alterações nesse repositório remoto. Sendo assim, para que não haja conflito com eventuais trabalhos feitos por outras pessoas, é recomendado que antes do *push*, seja feito um ***pull*** para pegar eventuais alterações feitas por terceiros.



## Utilizando via linha de comando

### Criando um repositório local

Para dar início aos trabalhos, vamos levar em conta que estamos começando um projeto novo e que a única coisa que já temos é um **repositório** **remoto vazio**. Sendo assim a primeira atividade que deve ser feita é a criação de um **repositório local**.

$ git init

### Ligando a um repositório remoto

Como já existe um repositório remoto vazio e agora também existe um repositório local. O próximo passo que deve ser feito a criação de uma ligação entre os dois repositórios para que o código que está sendo versionado localmente possa ser compartilhado na rede.

$ git remote add origin https://github.com/fulanodasilva/projeto.git

### Clonando um repositório remoto

A outra forma de iniciar um trabalho de versionamento de código usando Git seria se já existisse um projeto no repositório remoto. Esse cenário é, talvez, o mais comum dos casos, pois isso acontece quando vamos dar continuidade a um trabalho que tinha sido iniciado anteriormente. Nesse caso o que será feito é um **clone do repositório remoto**. Quando um repositório remoto é clonado, o usuário que fez o clone passa a ter um **repositório local idêntico ao repositório que foi clonado**. Vale lembrar que quando se clona um repositório remoto, ele automaticamente já possui uma ligação entre o local e o remoto.

$ git clone https://github.com/fulanodasilva/projeto.git

### Verificando o status dos arquivos do repositório

Como já foi visto inicialmente, quando foi falado sobre o ciclo de vida dos arquivos versionados, um arquivo não é adicionado automaticamente a ***staging area*** de modo que é necessário indicar quais arquivos devem ser adicionados. Para saber quais arquivos foram adicionados, alterados ou excluídos é necessário ver o ***status*** dos arquivos que estão no *working directory* do repositório

$ git status

### Gravando o arquivo no repositório

A partir do momento em que um ou mais arquivos foram adicionados a *staging area*, os mesmos já estão sendo rastreados pelo Git e, portanto é possível gravar efetivamente no repositório local. O nome dessa ação é ***commit***. Vale lembrar que, no Git, todos os *commits* precisam ter uma mensagem descrevendo o que foi feito.

$ git commit –m “mensagem”

### Verificando o histórico de alterações

Para verificar todo o histórico de alterações que foram feitas no repositório local, é possível ver através do log do Git.

$ git log

### Enviando alterações para um repositório remoto

Como já existe um repositório ligado ao repositório local que foi criado. Agora é o momento de enviar para a rede todas as alterações que foram feitas localmente. Para isso, será utilizado o ***push***.

$ git push origin

Lembrando que como a partir do momento em que um repositório remoto está sendo utilizado, acredita-se que não existe apenas uma única pessoa fazendo alterações nesse repositório remoto. Sendo assim, para que não haja conflito com eventuais trabalhos feitos por outras pessoas, é recomendado que antes do *push*, seja feito um ***pull*** para pegar eventuais alterações feitas por terceiros.

$ git pull