**Centro de Ciências Tecnológicas**

**Disciplina: Gestão de Configuração (T025)**

**Curso: Análise e Desenvolvimento de Sistemas**

**Prof. Everton Guimarães**

**Parte 1 – Revisão Teórica**

**1) Gestão e Itens de Configuração**

A **Gestão de Configuração** de Software consiste na *coordenação* do desenvolvimento de software para minimizar entropia, se tornando um importante elemento para garantia da qualidade de software. No ciclo de desenvolvimento de software existem artefatos que representam as mais diferentes informações do processo de desenvolvimento. Estes artefatos podem ser entendidos como **itens de informação**. Um subconjunto dos **itens de informação** são classificados como **itens de configuração**, uma vez que são escolhidos para terminar a **configuração de software**. Os **itens de configuração** pode ser de dois tipos: (i) produto de software, que está relacionado a programas, procedimentos, documentação e informações designadas *a serem entregues a um cliente ou usuário final*; e (ii) produtor de desenvolvimento de software, que está relacionado a descrição de processos, procedimentos, planos, programas e demais documentos, *que podem ou não ter finalidade de ser entregue ao cliente ou usuário final*. Em suma, diz-se que um conjunto de **itens de configuração** de software que estejam inter-relacionados compõe a **configuração de software**. Para que a **gestão de configuração de software** seja executada, algumas tarefas são definidas:

* Identificação – Determina como uma organização *identifica* e *administra* as diferentes versões existentes dos itens de configuração, possibilitando assim que mudanças sejam realizadas de forma mais eficiente.
* Controle de Mudanças – Determina responsáveis pela aprovação e define quais são as prioridades de mudanças.
* Controle de Versão – texto.
* Auditoria de Configuração – texto.
* Relato de Situação – Determina que mecanismos podem ser utilizados para notificar demais integrantes de um time de desenvolvimento sobre as mudanças realizadas nos itens de configuração.
* Controle de Interface – Determina como ocorre gerenciamento de mudanças nos itens de configuração que são afetados por demais itens que não estejam sendo controlados. Por exemplo, pode-se determina como é realizado o gerenciamento causado por alterações externas ao sistema.
* Controle de Subcontratos e Fornecedores – Preocupa-se em definir meios para garantir que módulos / componentes construídos por terceiros estejam corretos e coerentes com o sistema em fase de desenvolvimento. Ou seja, realiza-se do controle para determinar a forma como os itens de configuração, desenvolvidos por solicitação a outras empresas ou foram adquiridos já prontos, deverão ser incorporados ao repositório do projeto.

É importante salientar que existem algumas **tarefas preliminares** que devem ser executadas anteriormente as atividades inerentes a **gestão de configuração**:

1. Seleção os itens a serem gerenciados – neste tarefa deve-se selecionar quais itens de informação serão incorporados como **itens de configuração** de software. Por exemplo, em processos de desenvolvimento mais complexos pode haver uma ampla gama de documentos. No entanto, somente alguns desses documentos são realmente imprescindíveis para atividades que vão, por exemplo, desde a fase de concepção até o produto final do software. Dessa forma, nesta tarefa devemos definir quais itens devem ser priorizados adotando-se critérios como: (i) itens mais usados no ciclo de vida; (ii) itens mais genéricos que contem informações transversais a todo o processo de desenvolvimento; (iii) itens mais importantes para segurança; (iv) itens projetados para reuso; e (v) itens passíveis de atualização simultânea. Somente os artefatos como **itens de configuração** do software deverão ser controlados (ex. Mudanças, versionamento).
2. Para a seleção dos **itens de configuração** podem ser adotados diferentes critérios, tais como: (i) Identificação dos relacionamentos entre itens – Esta atividade é de suma importância para a manutenção de todos os itens de configuração do sistema, uma vez que permite localizar rapidamente os itens afetados por cada alteração; e (ii) Criação de classes de relacionamento – Como exemplos de classes de relacionamentos entre **itens de configuração**, podemos citar equivalência, dependência, derivação, sucessão, dentre outros. Podemos citar, por exemplo, a derivação de um código fonte a partir de um documento de projeto de software ou ainda um relacionamento de dependências, se considerarmos a descrição do projeto modular que é dependente da especificação da arquitetura do sistema.
3. Planejamento das **linhas de base** – A criação das linhas de base pode considerar como marco do projeto, por exemplo, o término de cada ciclo de interação do ciclo de vida do processo de desenvolvimento de software. Dessa forma, a especificação dos **itens de configuração** devem ser revisados e armazenados/atualizados em casa uma das linhas de base.
4. Descrição do Arquivamento e recuperação dos **itens de configuração** do repositório – Nesta parte define-se a estruturação do repositório e permissões de acesso a operações no repositório do sistema de software.

**2) Linha de Base**

**Conceito:** A linha de base é um ponto de referencia no ciclo de vide do desenvolvimento de software marcado pelo término ou aprovação formal de um conjunto de produtos predefinidos.

**Objetivo:** O objetivo de uma **linha de base** consiste na redução da vulnerabilidade a mudanças não controladas por meio realização do controle de mudança formal dos principais entregáveis do sistema (ou itens de configuração). Em outras palavras, uma **linha de base** ajuda a realizar o controle de mudanças sem impedir que aquelas mudanças que são justificáveis em determinados **itens de configuração** sejam realizadas.

**O propósito das linhas de base pode ser resumido em:**

* Definir pontos de progresso mensuráveis dentro do ciclo de vida de desenvolvimento de software.
* Um ponto base para o controle de mudanças em frases subsequentes do projeto
* Uma referencia estável para o trabalho futuro.
* Pontos intermediários e finais para mensurar a aptidão para o propósito dos produtos provenientes do projeto de software.

A definição de **linhas base** eficazes deve conter características: (i) a **linha de base** deve estar associada com a produção e aprovação formal de um entregável físico, tais como um documento ou componente de software; e (ii) todos os itens associados com a **linha de base** devem ser colocados sob um controle de mudanças formal.

Como exemplo de linhas base tipicamente definidas de acordo com o ciclo de desenvolvimento de software, podemos citar:

* Especificação dos requisitos do sistema (linha de base funcional)
* Projeto de alto nível (linha de base preliminar)
* Projeto detalhado (linha de base de projeto)
* O produto de software gerado após a realização dos testes do sistema (linha de base do produto)
* O produto de software em seu ambiente operacional (linha de base operacional)

**Obs.** Um item de configuração de software que passam por uma linha base, diz-se que ele “*tornou-se”* ou “*faz* *parte*” de uma determinada linha base. Logo, quando um **item de configuração** faz parte de uma linha base, ele certamente possui as seguintes características: (i) O item foi revisto formalmente pelos responsáveis da equipe de desenvolvimento e houve acordo entre as partes para sua aprovação; (ii) O item é armazenado em um repositório de itens de configuração; e (iii) somente por ser modificado por meio de procedimentos de controle de mudanças.

Para especificação de uma **linha de base**, pode-se definir algumas informações:

* Os nomes dos itens físicos que constituem a linha de base (documento de projeto, especificação da arquitetura, código fonte).
* O marco em um projeto onde espera-se o cumprimento / finalização de uma linha de base (e.x. fim de uma das fases definidas no processo de desenvolvimento).
* O método a ser adotado para aprovar os itens de configuração definidos na linha de base.
* Os indivíduos responsáveis pela aprovação dos itens da linha de base (e.x. auditor)

**3) Repositório de Itens de Configuração**

O repositório de itens de configuração consiste em um local de acesso onde são armazenados os **itens de configuração** de software, depois de liberados por uma linha base. De acordo com as linhas de base estabelecidas, os **itens de configuração** devem ser *identificados*, *analisados*, *corrigidos*, *aprovados*, *armazenados* no repositório. Os **itens de configuração** armazenados no repositório somente poderão sofrer alguma alteração mediante uma solicitação de modificação formalmente aprovada pelo gerente de configuração. Dessa forma, permite-se manter o controle sobre cada um dos **itens de configuração**, de forma a evitar inconsistências.

**Parte 2 – Utilizando Controle de Versões e Mudanças com Git / GitHub**

O seguinte tutorial tem por objetivo exercitar os conceitos relativos a Gestão de Configuração, mais especificamente Controle de Versões e Controle de Mudanças. O cenário criado para execução da prática é similar ao adotado em ambientes de Fabrica de Software.

Alguns passos de como usar o Git no Eclipse.

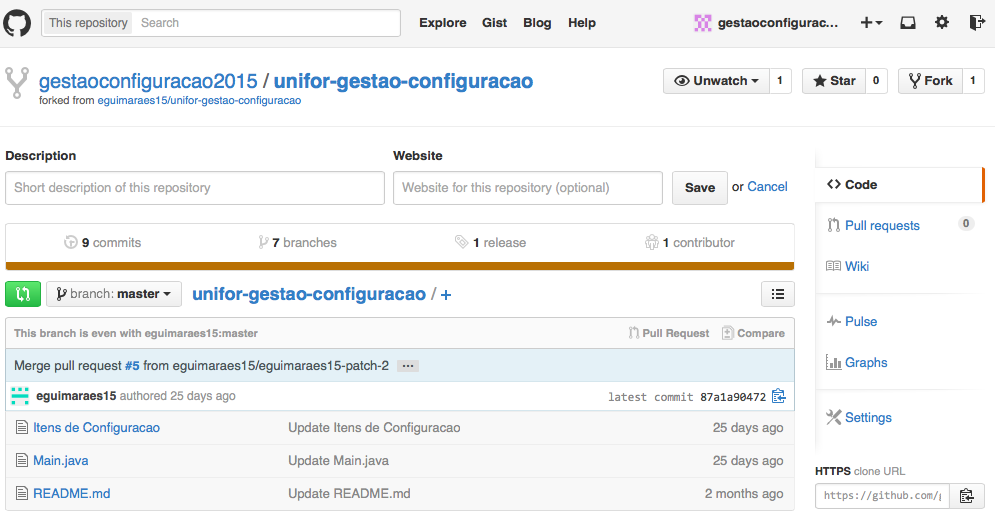
* Crie seu usuário no GitHub
* Acesse o repositório criado pelo professor através do link abaixo.
  + <https://github.com/eguimaraes15/unifor-gestao-configuracao>
* Crie um Fork do repositório para seu usuário no GitHub (veja figura abaixo). Dessa forma, você estará criando uma ramificação do projeto para sua conta do GitHub de forma que agora você pode colaborar com o repositório de outros usuários.

Figura 1 - Criando um fork do repositório

* Veja que foi gerado um Fork do repositório **unifor-gestao-configuracao** para a conta do novo usuário GitHub que foi criado gestaoconfiguracao2015 (ver figura 2).

Figura 2 - O Fork do repositório unifor-gestao-configuracao foi devidamente criado

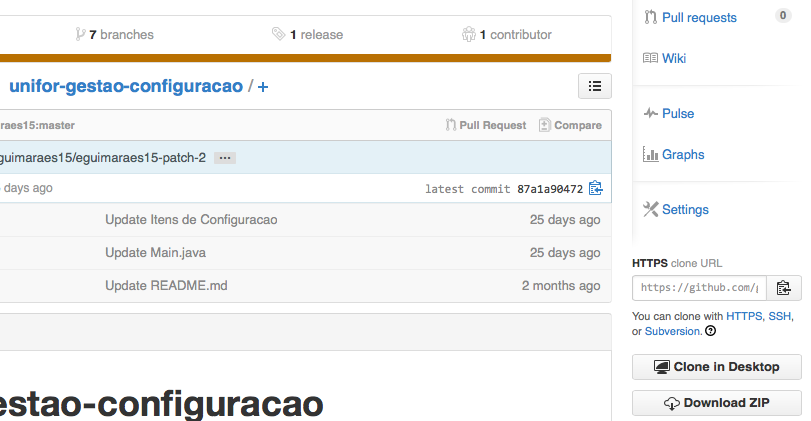
* Uma vez criado um Fork do repositório que se deseja colaborar, você deve agora criar uma cópia local na sua máquina. Lembre-se que o GitHub permite a criação de uma cópia local, onde são feitas as alterações nos **itens de configuração**, após o que cada alteração é salva por meio do commit. Quando se finaliza um conjunto de alterações, pode-se enviar tais mudanças para o servidor remoto onde está hospedado o repositório. Esta operação é realizada por meio do comando de “*pull request*”.
* Para a criação de uma cópia local do repositório que acabamos de realizar um Fork, precisamos clicar na opção “Clone in Desktop” de forma que seja redirecionado para o cliente GitHub.

Figura 3 - Clonando o repositório no desktop.

* Caso o cliente GitHub ainda não esteja instalado e configurado na sua máquina, você deve realizar o download no link: <https://mac.github.com> ou <https://windows.github.com>. Após realizar o download e instalar, você deve configurar seu usuário para poder clonar um cópia local do repositório para sua máquina.

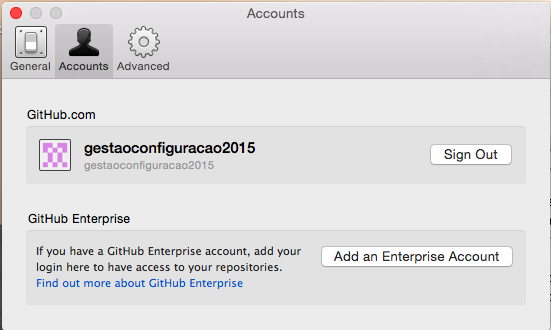
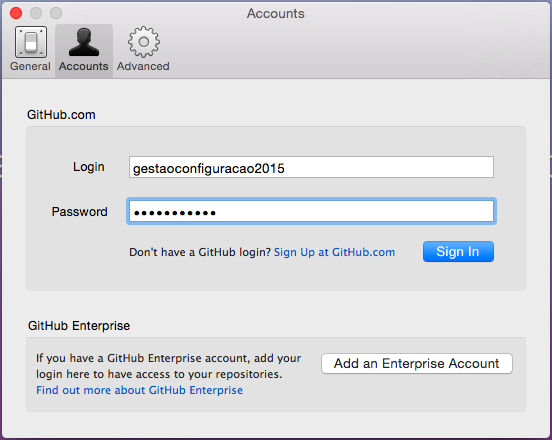


Figura 4 - Configurando a conta do Git no repositório local.

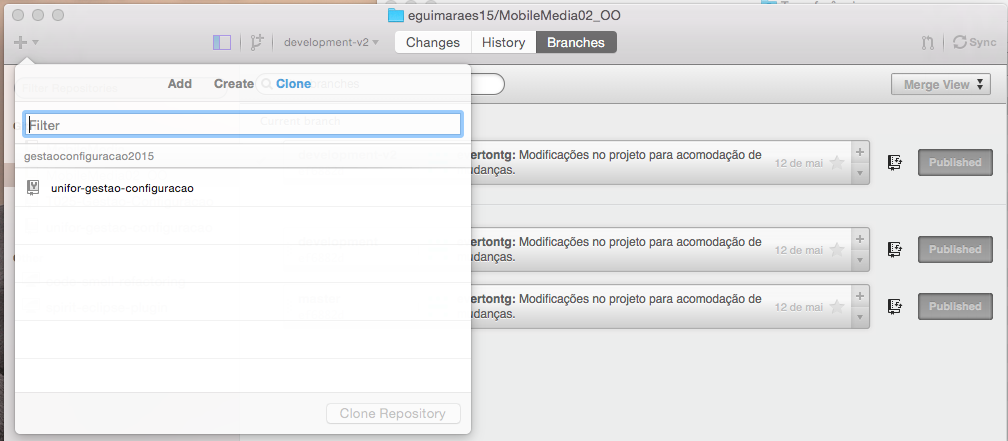
* Uma vez configurado o usuário, podemos finalmente clonar uma cópia do projeto hospedado no repositório remoto para a sua máquina, acessando a opção clone no cliente do GitHub. Você deve ainda selecionar qual o repositório remoto que será copiado para sua área de trabalho. Neste caso usaremos o repositório Unifor-gestao-configuracao que criamos para a disciplina.

Figura 5 - Clonando uma cópia do repositório localmente

* Defina o nome “alias” que você deseja usar para clonar o repositório.

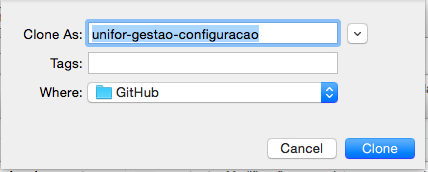


Figura 6 - Definição o nome do arquivo local do repositório

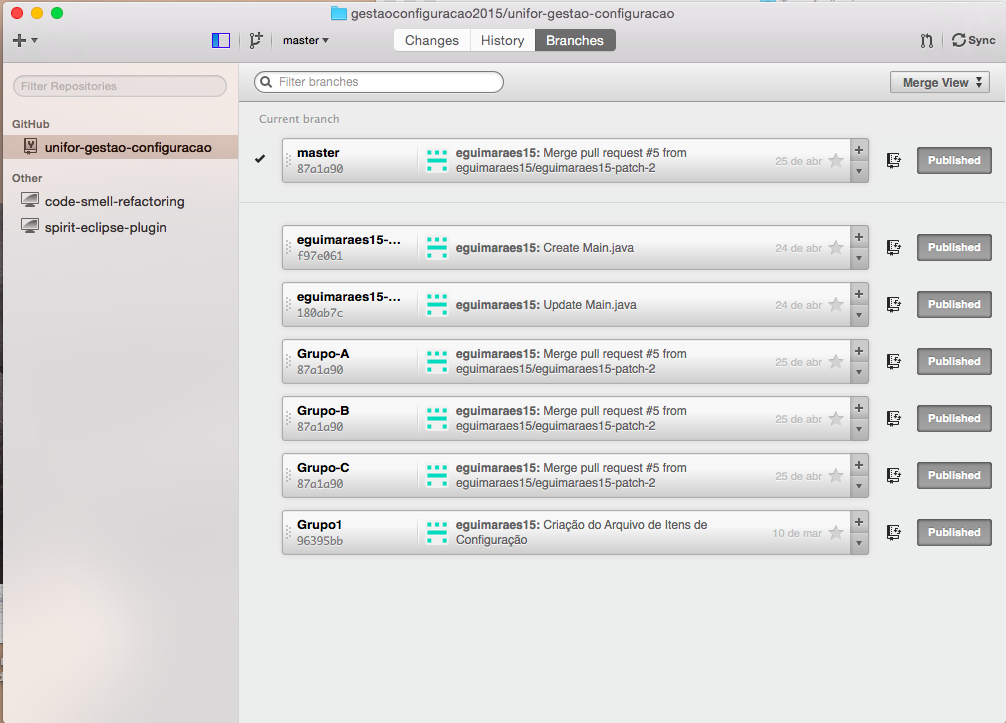
* Ao confirmar a operação, toda a estrutura do repositório remoto (arquivos, diretórios, histórico, commits, requisições, etc. foi “clonada” como uma cópia local na sua máquina. Agora você pode iniciar as mudanças e colaborar com o repositório.

Figura 7 - Estrutura do repositório criada localmente

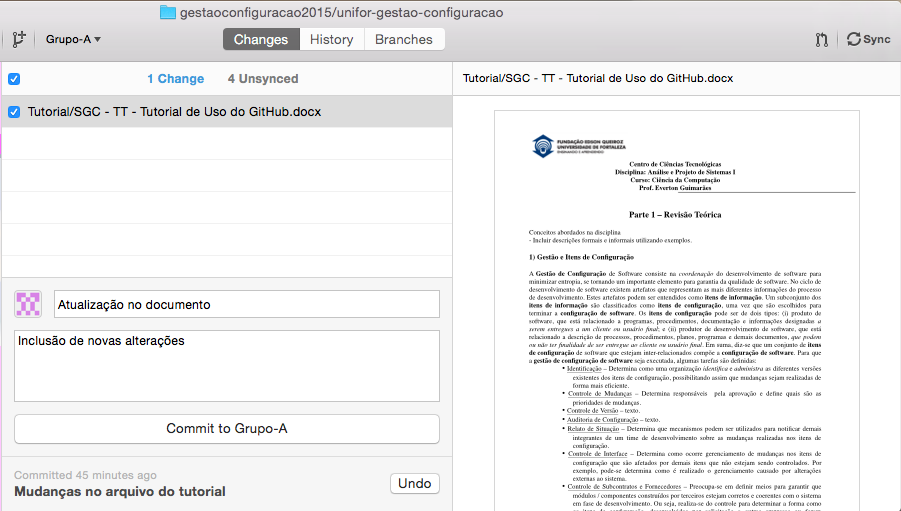
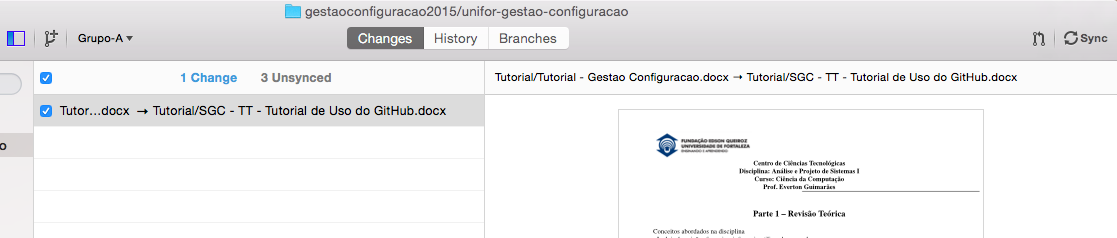
* Realize todas as mudanças necessárias, tais como criação, edição, e remoção de arquivos, diretórios, dentre outros. Ao finalizar todas as alterações desejadas para o item de configuração, você pode realizar o commit para que essas mudanças sejam salvas no repositório local.
* Quando todos os commits forem realizados, e quisermos enviar todas as mudanças para o repositório remoto devemos criar uma “pull request” para enviar as modificações do arquivo para o servidor. Mas antes disso, vamos utilizar um identificação adequada para o documento do tutorial que estamos enviando/compartilhando em nosso repositório do GitHub. Altere o nome do documento, de acordo com o especificado abaixo e perceba que a mudança também é rapidamente sincronizada com o cliente do GitHub mostrando a alteração de nome que foi realizada.

Figure 8 - Realizando Commit de Alterações

Tutorial/Tutorial – Gestão Configuração 🡺 Tutorial/ SGC – TT – Tutorial de Uso do GitHub



* Repare que após realizadas as modificações e realizado o commit com as novas alterações, agora podemos visualizar que o commit contém duas mudanças a serem salvas.

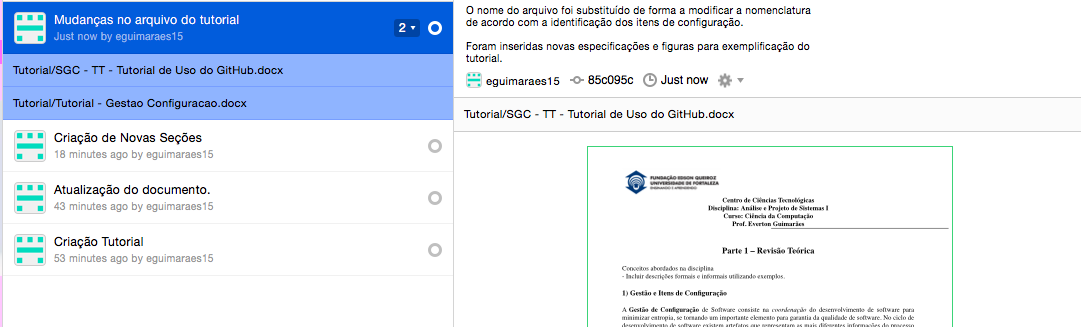


Figura 9 - Visualizando alterações no commit

* Realizando um novo commit. Perceba que a cada commit realizado, devemos especificar o título, descrição, e qual a ramificação onde essa modificação será salva.
  + Titulo: Atualização no documento.
  + Descrição: Inclusão de novos passos no tutorial, Inclusão de novas figuras para melhor ilustrar o passo-a-passo das operações.
  + Além disso, vejam que a ramificação em que as modificações serão salvas corresponde ao “Grupo A”, onde o desenvolvedor está trabalhando especificamente neste documento.

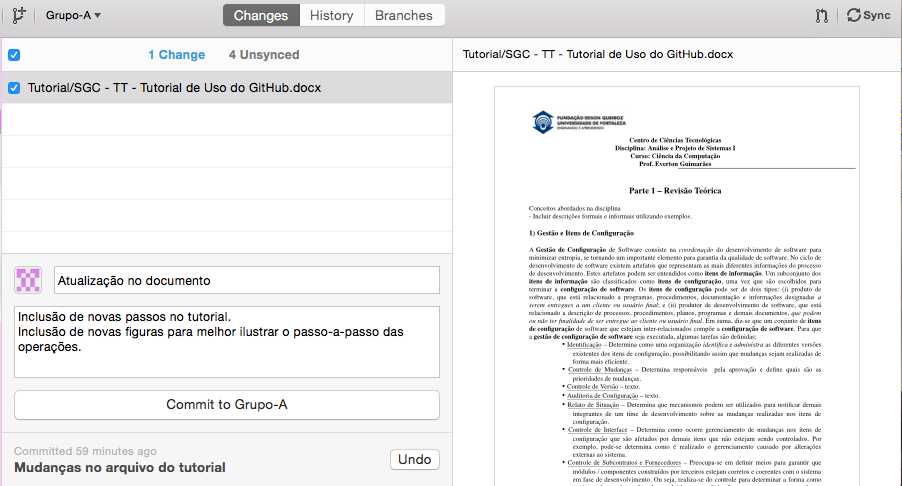


Figura 10 - Parâmetros informados no Commit

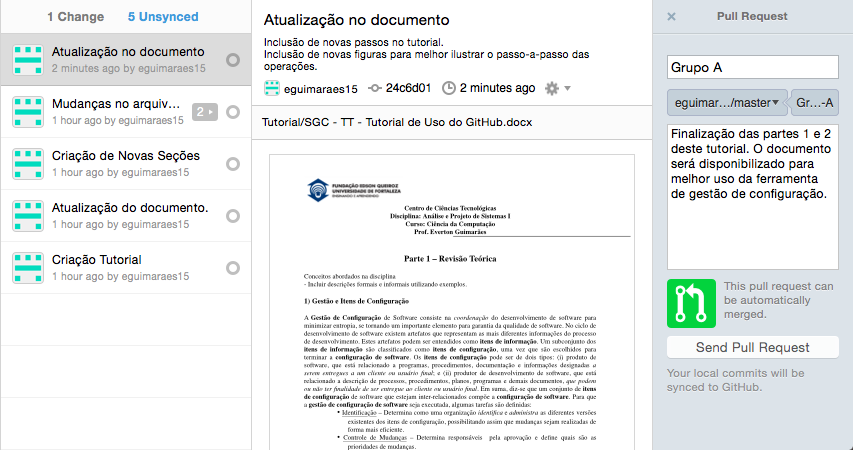


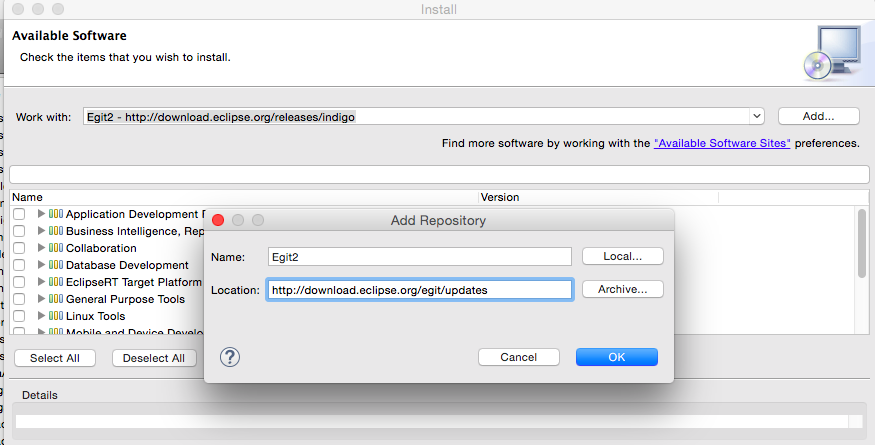
Figure 11 - Realizando uma "pull request"

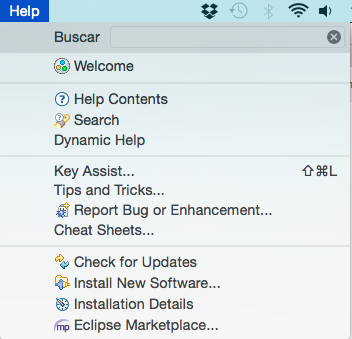
* Uma vez salvas todas as alterações, iremos realizar uma solicitação de envio (*pull request*)para o servidor onde o repositório está hospedado remotamente. Dessa forma, devemos escolher para qual ramificação (branch) queremos que nossas modificações sejam replicadas. Neste caso, temos 5 mudanças que não estão sincronizadas com o repositório remoto. Portanto, podemos enviar uma “pull request” para somente uma das mudanças, ou podemos “empacotar” todas as mudanças dentro uma única solicitação. Para realizar a “pull request” definir o título da requisição. Logo, trocaremos “Grupo A” por “Especificação do Tutorial” e também descreveremos sucintamente o que o arquivo representa. Uma vez definidos parâmetros, devemos indicar para qual ramificação iremos replicar as mudanças, sendo que neste caso iremos replicar da ramificação atual “Grupo A” para a ramificação “master” do repositório que estamos colaborando.

**Parte 3 – Preparação de Ambiente – Usando Git no Eclipse**

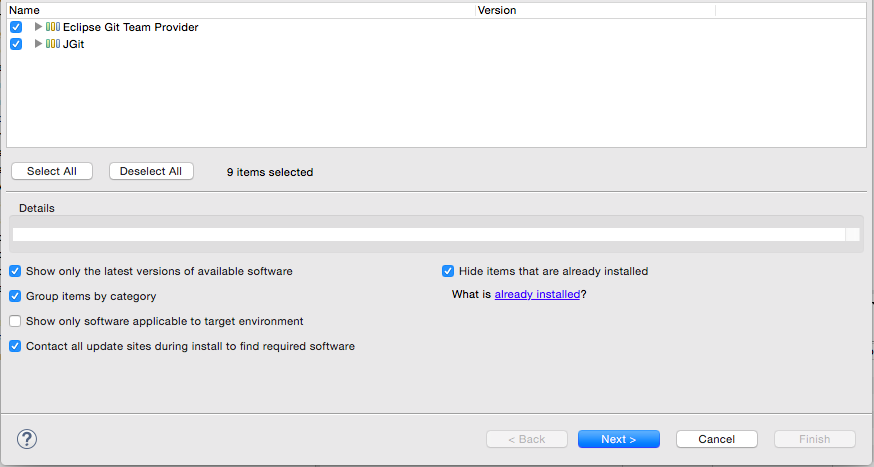
* **Instalação**

Antes de iniciar nosso trabalho de desenvolvimento, primeiramente iremos configurar o acesso ao GitHub no ambiente de desenvolvimento. Para isso devemos instalar o plug-in para Eclipse chamado Egit, que permite gerenciar os repositórios existentes na sua conta do GitHub. Para iniciarmos a instalação, abre o Eclipse e acesso o menu Help -> Install New Software”.

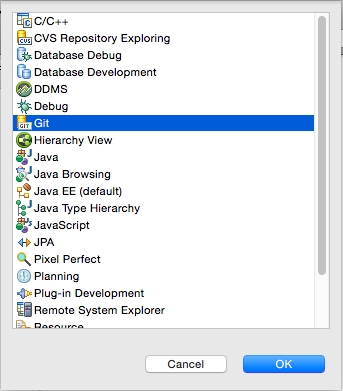
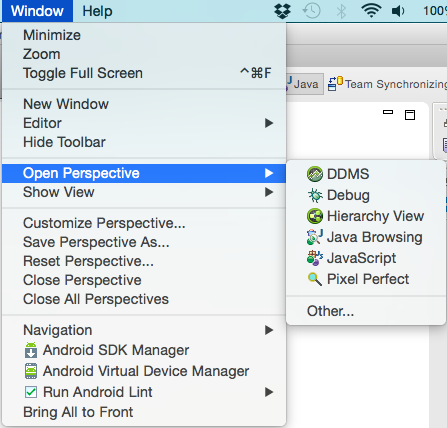




Após abrir a tela para instalação do novo software, clique no botão “Add” e deverá aparecer uma nova tela “Add Repository”. Neste momento, você deve especificar o nome do plug-in que você deseja salvar (Egit) e qual o repositório de onde será feito o download (<http://download.eclipse.org/egit/updates>). Uma vez inseridas as informações clique em OK e aguarde que o Eclipse carregue o conteúdo remoto disponível para download. Um vez carregado o conteúdo remoto, o Eclipse deverá disponibilizar as duas bibliotecas que compõe o Egit e demais recursos que podem ser utilizados pelo plug-in conforme representado na figura abaixo.



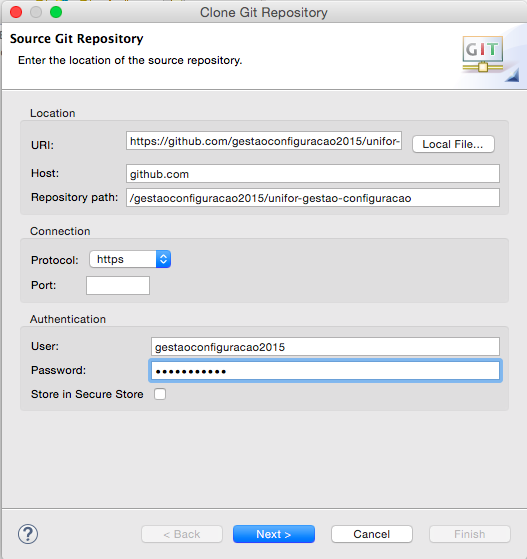
Selecione a instalação de ambos os pacotes e selecione “Next”. O Eclipse irá então calcular possíveis dependências entre os projetos, após o que você deve selecionar “Finish” para iniciar o download dos plug-ins e realizar a instalação. Após finalizada a instalação, reinicie o Eclipse para ter acesso a perspectiva do Egit. Caso o Eclipse não abra diretamente a perspectiva do Egit, acesse pelo caminho: Window -> Open Perspective -> Other. O Eclipse abrirá uma janela para seleciona a perspectiva desejada, e nesse momento clique na opção “Git”.



A princípio existem dois caminhos para trabalharmos com o repositório GitHub dentro da IDE Eclipse usando o plug-in do Egit.. O primeiro deles seria por meio da “clonagem” do repositório remoto para sua máquina local. Outra forma seria importando um repositório já existente, ou seja, que porventura já tenha sido clonado utilizando um cliente GitHub.

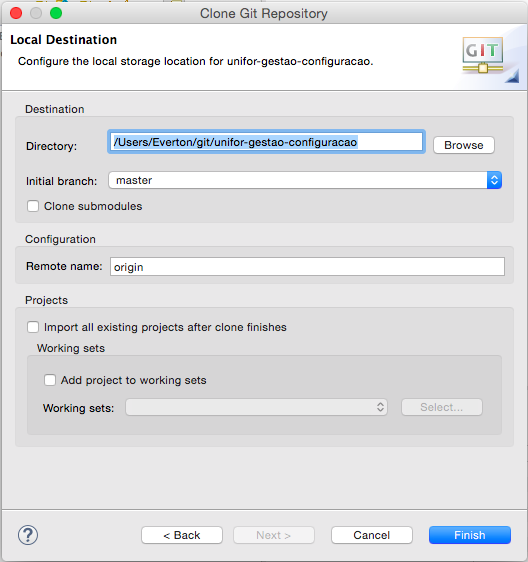
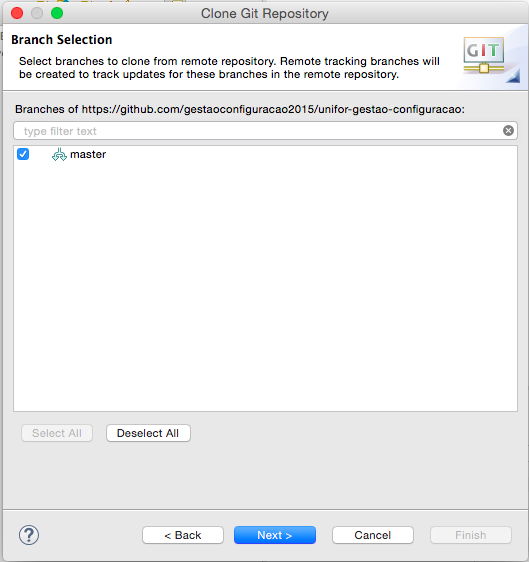
1. Explorando a perspectiva do Git -> “Git Repository Exploring”

A primeira forma de trabalhar com o GitHub diretamente na IDE é clonando o repositório remoto para sua máquina local. Para isso, selecione a opção “Clone a Git Repository”. No painel de controle, selecione a opção “Clone Git Repository” e o Eclipse irá abrir a tela de configuração do repositório. Para clonar o repositório você deve informar os seguintes parâmetros:

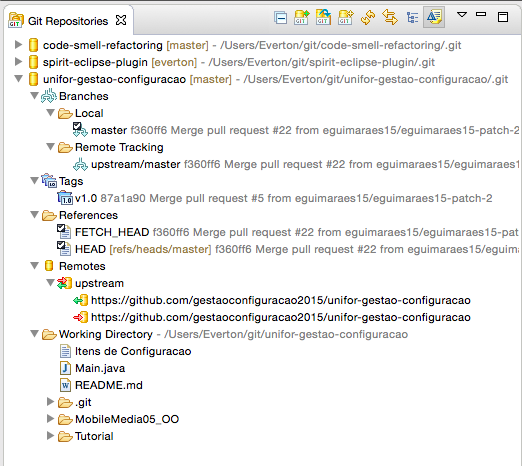
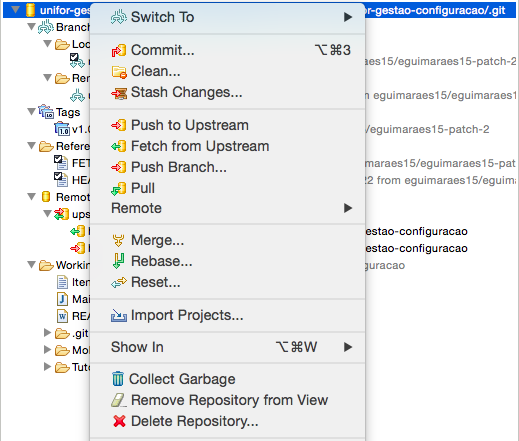


* **Configurando Conexão com Repositório.**
  + Location
    - URI: <http://github.com/gestaoconfiguracao2015/unifor-gestao-configuracao> (Corresponde ao endereço completo do repositório GitHub)
    - Host: github.com (corresponde ao servidor onde está hospedado o repositório)
    - Repository Path: [/gestaoconfiguracao2015/unifor-gestao-configuracao](http://github.com/gestaoconfiguracao2015/unifor-gestao-configuracao) (corresponde ao caminho de acesso do repositório por meio do usuário da sua conta GitHub).
  + Connection
    - Protocol: https (Corresponde ao protocolo de conexão usado para acesso seguro ao repositório)
  + Authentication (Devem ser especificadas as informações de usuário e senha para acesso a sua conta do GitHub).
    - User: gestaoconfiguracao2015
    - Password: \*\*\*\*\*\*\*\*\*\*
* **Configurar o diretório de importação do repositório.**

Uma vez realizada a configuração de conexão, precisamos definir qual *branch* (ou ramificação do repositório) queremos clonar do repositório remoto para a maquina local. Portanto, devemos configurar ainda onde o repositório será clonado localmente. Normalmente, clonamos o *branch master* do repositório remoto pois é onde as mudanças realizadas nos diversos itens de configuração deverão ser atualizadas, por exemplo, ao final do desenvolvimento de um componente.

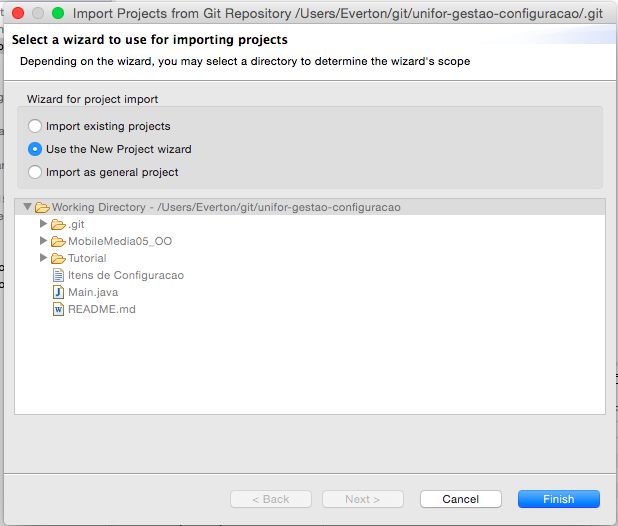
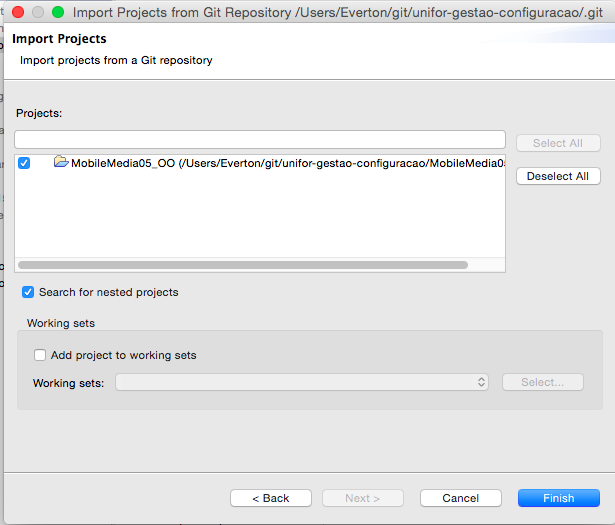
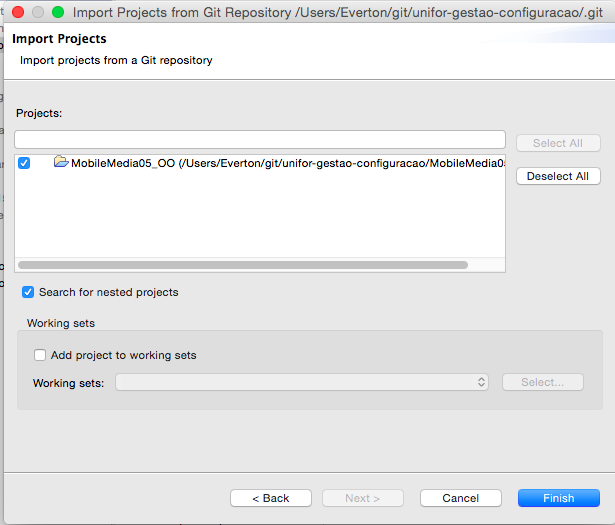


Havendo finalizado a configuração do repositório, o Eclipse irá gerenciar a configuração de conexão e criação da cópia local do repositório. Ao final, a cópia local do repositório (bem como seu histórico de operações, dentre outras informações) deverão ser apresentadas na perspectiva do Git, conforme ilustrado na figura a seguir. A partir de então, você pode começar a “brincar” com as diversas funcionalidades providas pelo cliente do Egit. Para ver as operações disponíveis para gerenciamento do repositório por meio do plug-in, clique com o botão direito sobre o repositório, e será aberta uma janela com diversas funções (ex. Commit, Push, Fetch, Pull, Merge, etc).



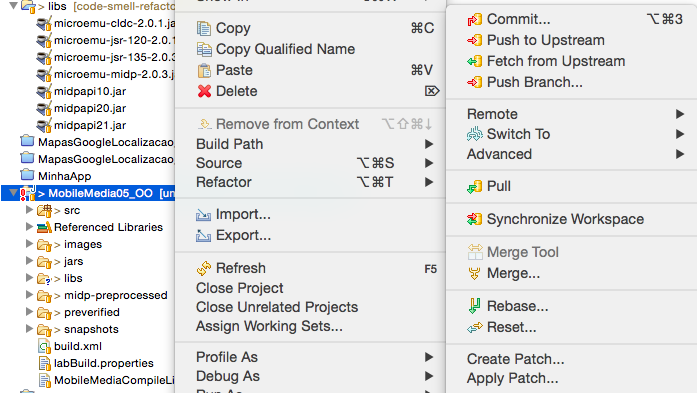
Como podemos ver, além dos arquivos gerados automaticamente no momento de criação do repositório (ex. README.txt), existem dois diretórios utilizados para construção de artefatos. O primeiro deles contem o presente tutorial que encontra-se em fase de construção. Por outro lado, o diretório MobileMedia05\_OO corresponde quinta versão da linha de produto Mobile Media utilizada para o gerenciamento de mídias em dispositivos móveis (utilizando tecnologia J2ME). Suponhamos que clonamos o repositório com o intuito de desenvolver uma nova funcionalidade do Mobile Media. Para isso temos que importar o projeto do repositório local para a workspace do Eclipse, onde iremos aplicar as mudanças diretamente no código fonte.

Para importar o projeto do Mobile Media, deve-se clicar com o botão direito no repositório e selecionar a opção “Import Projects”. A partir de então o Eclipse irá abrir a tela para criação de projeto, podendo ser importado de 3 formas: (i) importação de projetos existentes; (ii) criação de novo projeto; ou (iii) importação como um projeto geral. No nosso caso, iremos importar um projeto (Java) existente no repositório recentemente clonado. Portanto, devemos selecionar somente o projeto do MobileMedia05\_OO e prosseguir até a finalização de importação do projeto.

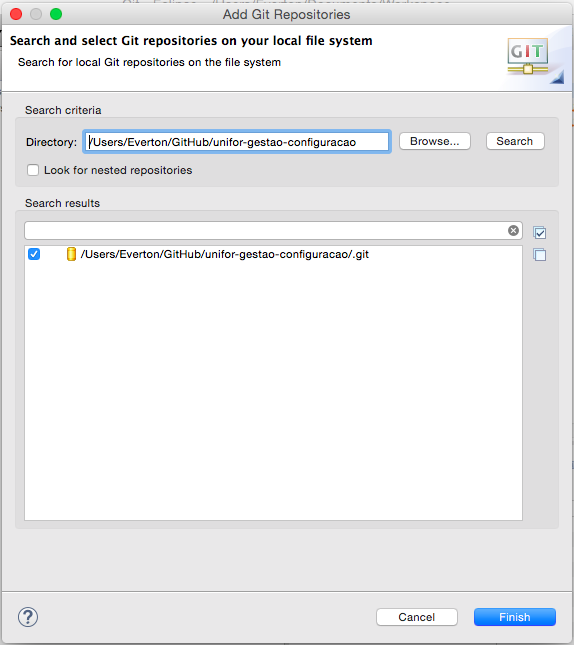


Assim que finalizada a importação do projeto Java do MobileMedia05\_OO, a perspectiva será alterada para “Java Project”, onde deverá mostrar o projeto recém importado para a workspace do Eclipse.

Finalmente, após ter importado o projeto Java para a workspace podemos trabalhar nas alterações necessárias para incorporar novas funcionalidades / características ao projeto. Uma vez realizadas as devidas mudanças podemos fazer commit localmente, e posteriormente enviar requisições de envio (*pull request*) para o repositório remoto de forma similar como fazemos com o cliente do GitHub. Para acessar essas funções de gerenciamento de mudanças, podemos clicar no projeto Java com o botão direito, navegar até a opção “Team” e a partir de então acessar as diversas funcionalidades do repositório.

**

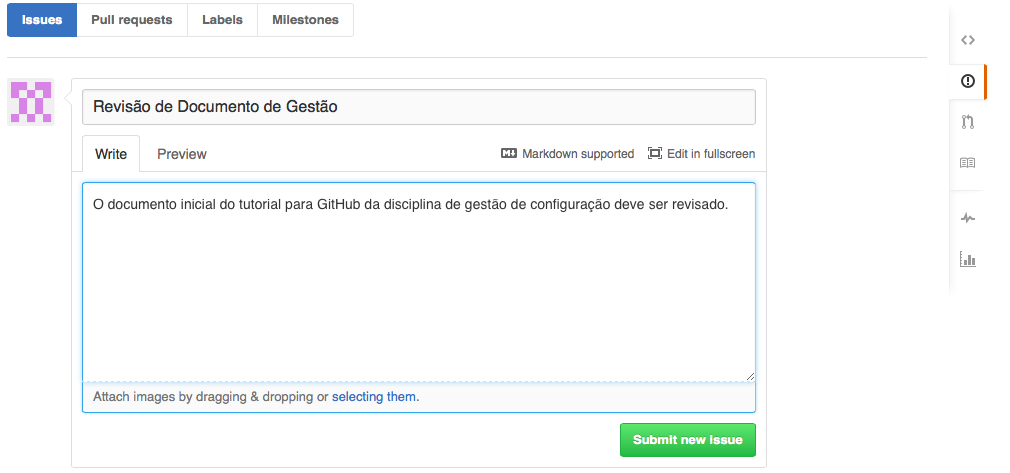
* **Configurar o diretório de importação do repositório.**

A outra forma de trabalharmos com o repositório GitHub na IDE Eclipse é importando um repositório já existente localmente, ou seja, que já foi clonado do repositório remoto que se deseja colaborar. Para isso selecionamos a opção “Add an Existing local Git Repository”. Como próximo passo você deve selecionar o repositório local, apontando para o diretório onde se encontra o arquivo .git para que todas as configurações / informações sejam devidamente carregadas no Eclipse. Após importado o repositório, você somente precisa realizar a importação do projeto Java como feito no passo anterior, e começar a trabalhar nas alterações do seu projeto.

**Parte 4 – Outros Recursos GitHub (Issues, Labels, Releases)**

1. Issues

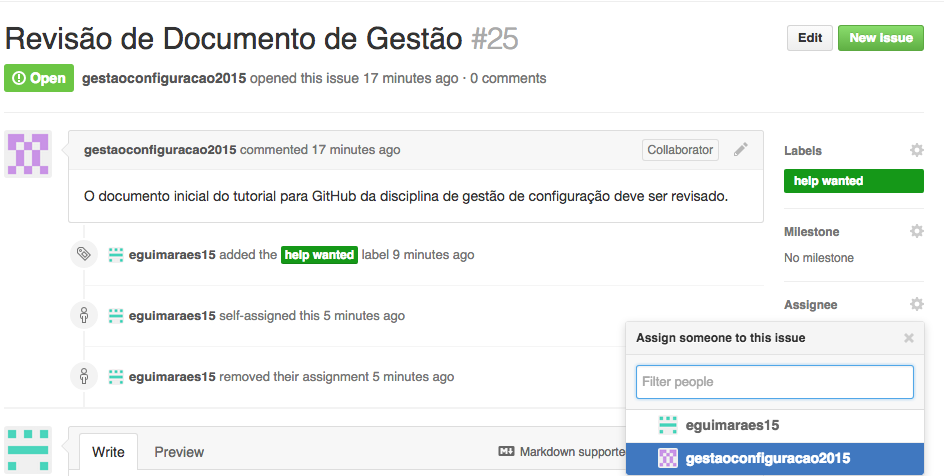
Descrição de Issues.

Para criação de uma *Issue* devemos colocar no título a descrição sucinta da atividade que merece atenção durante o processo de desenvolvimento, e posteriormente descrever que as atividades relacionadas a criação da *Issue*. Neste caso estamos criando uma *Issue* para “Revisão do Documento de Gestão” para que possam colaborar na revisão do documento relacionado ao Plano de Gestão de Configuração gerenciado para um determinado projeto.

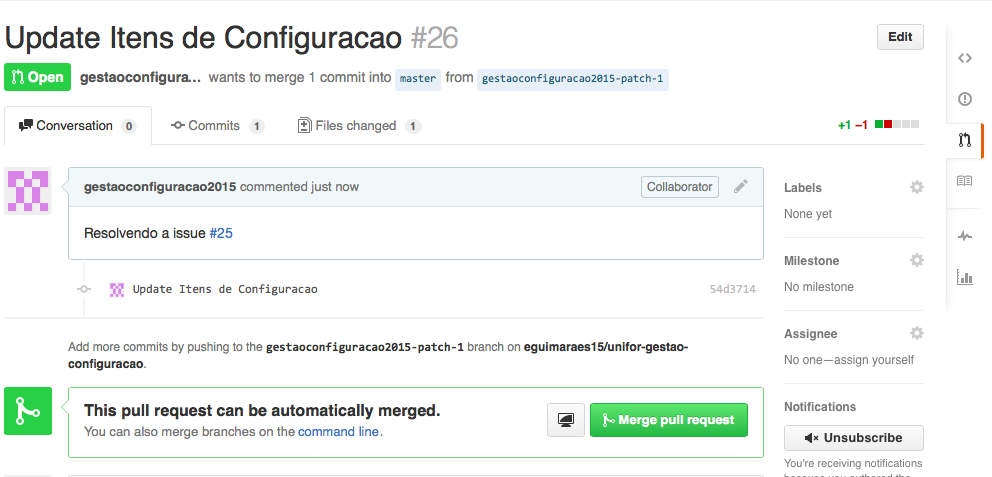
Uma vez especificados os dados para criação da *Issue*, clique no botão “Submit new issue” para que ela seja finalmente criada. Quando a *Issue* é criada gera-se um número de identificação para que posteriormente ela possa ser referenciada quando na realização do commit que contém mudanças realizadas no documento do Plano de Gestão de Configuração. Por exemplo, a *Issue* #25 foi criada para a revisão do documento de gestão. Logo, os colaboradores do repositório podem trabalhar nessa *Issue* que deve ser devidamente atribuídas a eles.



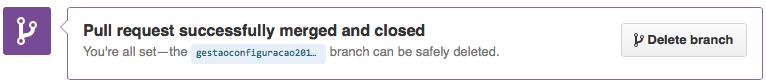
A atribuição de uma *Issue* criada no repositório normalmente pode ser feita pelo administrador do repositório ou pelo próprio colaborador que deseje trabalhar numa determinada *Issue*. Neste caso iremos trabalhar com a atribuição de *Issues* por parte do administrador do repositório, eguimaraes15 que está atribuindo a *Issue* #25 para o colaborador gestaoconfiguracao15. Perceba ainda que foi utilizado um *label* “help wanted” para identificar o tipo de *Issue* aberta pelo desenvolvedor, que neste caso necessita de ajuda para a revisão do documento. Existem alguns *labels* que são padronizados no GitHub, mas a ferramenta também permite a criação de novos *labels* (rótulos).



Quando deseja-se associar as modificações em um documento a uma determinada *Issue*, basta apensar passar como parâmetro a descrição da *Issue* na mensagem de *commit*. Caso o desenvolvedor não saiba o número especifico da *Issue* que ele deseja referenciar, poderá ser feita, por exemplo, uma busca por rótulo utilizado na *Issue* ou mesmo verificar quais *Issue* foram atribuídas a este desenvolvedor. Em um primeiro momento o colaborador *gestaoconfiguracao15* realizou algumas mudanças no arquivo **Itens de Configuracao** e gerou o *commit* dessa mudanças. Posteriormente, ele gerou uma solicitação de envio dessas mudanças por meio de uma *pull request*. Perceba que na mensagem de *commit* foi feita referencia a *Issue* #25 criada para revisão de documentos de gestão.

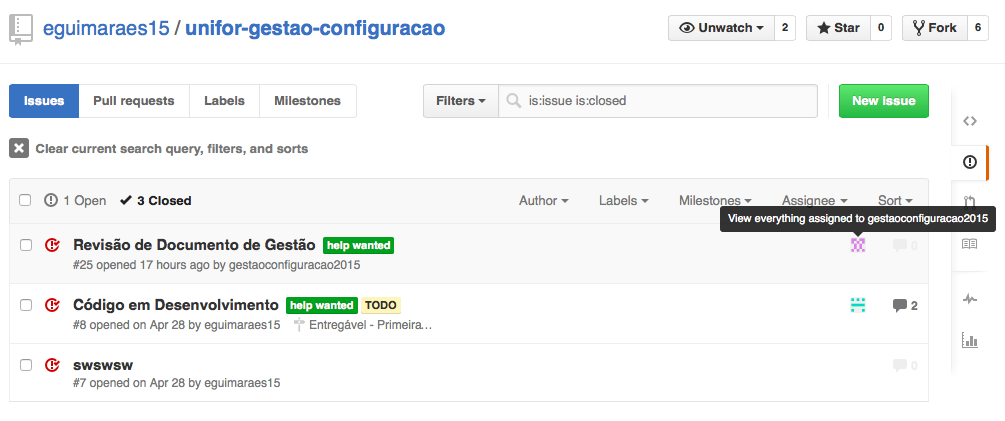


No momento de geração da *pull request* foi criado um *branch* para que a mudança não foi realizada diretamente na origem *master* do repositório, de forma que as mudanças somente serão atualizadas no servidor remoto após aprovação do administrador do repositório.

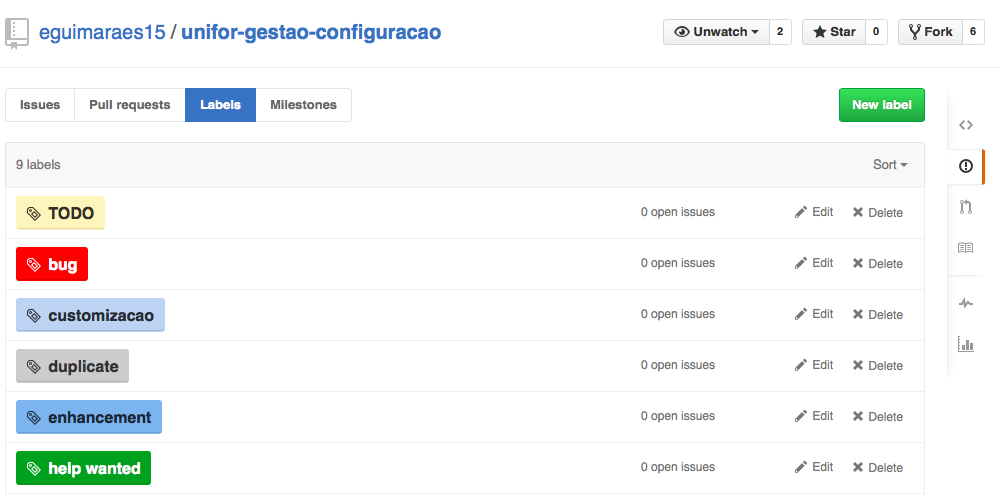


Ao finalizar o desenvolvimento de um componente ou nova funcionalidade, o desenvolvedor pode fechar a *Issue* utilizando a palavra-chave “close”“, closes” ou “closed” no corpo do texto onde descreve-se as modificações realizadas por um commit. Se por outro lado esse desenvolvedor estava trabalhando em uma *Issue* para consertar um *bug* identificado por outro desenvolvedor você pode utilizar a palavra-chave “fix”, “fixes” ou “fixed” para finalizar a *Issue* indicando que o *bug* foi corretamente removido.

A seguir podemos ver que a *Issue* #25 – “Revisão de Documento de Gestão” foi criado e posteriormente fechado pelo colaborador *gestaoconfiguracao15*. Ou seja, caso o colaborador/desenvolvedor tenha utilizado as palavras-chave para fechamento de *Issues* na mensagem de *commit*, e posteriormente enviado uma *pull request* para enviar tais modificações para o servidor removo, a *Issue* será automaticamente fechada uma vez que a *pull request* é avaliada e aceita pelo administrador do repositório.



1. Labels

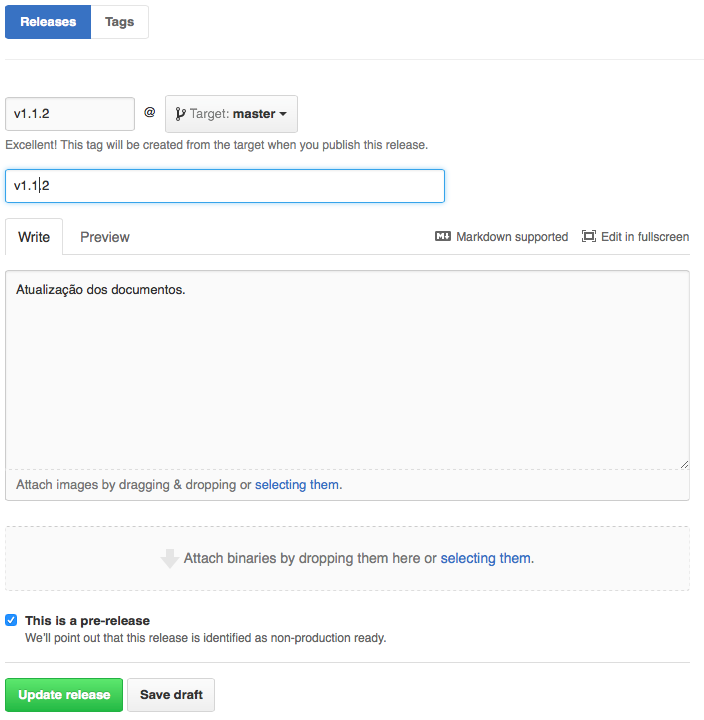
Como pontuado na sessão anterior, podemos criar rótulos (*labels*) como forma de melhor gerenciar as *Issues* criadas no repositório e que eventualmente terá, por exemplo, diversos desenvolvedores colaborando. Neste caso, podemos verificar que inicialmente não existem rótulos associadas a uma determinada *Issue*. Essa associação normalmente é feita após a criação da *Issue*, onde o administrador do repositório também irá atribuir / delegar uma atividade para um (ou mais) desenvolvedor(es).

1. Releases / Tags

As Releases é uma parte do fluxo de desenvolvimento de software que engloba a entrega do produto de software ao usuário final. Dessa forma, conceitualmente as releases são objetos de primeira classe que possuem log de mudanças (changelogs) e ativos binários que apresentam o historico completo dos artefatos (itens de configuração) gerenciados no repositório GitHub.

Como colaborador de um repositório, você pode “rascunhar” o log de mudanças por meio de “notas de release”. Portanto, toda release que não representam o produto final (alfa, beta, candidatos a release) podem ser identificados por meio do *checkbox* “*pre-release*”. As releases são acompanhadas por notas e links para realizar o download, por exemplo, do código fonte correspondente ao produto entregue ao cliente numa dada versão do software.

De acordo com as converções do repositório GitHub, as releases podem ser comparadas com Tags. Você pode utilizar uma Tag existente ou deixar a release criar uma Tag automaticamente quando a mesma é publicada. Além disso, permite-se a adição de ativos binários (ex. Scripts, documentação, executaveis compilados, libs) a uma release. Uma vez publicada a release, todos os detalhes e ativos empacotados estarão disponíveis para qualquer colaborador que possa visualizar o repositório.



Após a finalização da release, podemos perceber que uma Tag com a mesma identificação foi criada. As Tags podem ser usadas para empacotar mudanças durante o desenvolvimento dentro de uma linha de base. Ou seja, a cada entregáel (deliverable) definido no marco do projeto dentro da uma linda de base, pode-se gerar uma Tag que designa uma versão estável de um produto mas ainda ainda encontra-se em desenvolvimento, uma vez que não foram finalizadas todas as atividades para aquela determinada linha de base. Ao final da linha de base onde tem-se o conjunto completo de alterações que correspondem a entrega do produto de software ao cliente, pode-se gerar uma release para o produto. Adicionalmente, as Tags também podem ser geradas incrementalmente de forma que a cada iteração do processo de desenvolvimento seja gerada uma versão estável, ainda que incompleta do produto de software.

