Expresso 3.0
Utilização e Estrutura do Repositório

1. Objetivo

Este documento tem como objetivo detalhar a estrutura do repositório do Expresso 3.0, bem como este repositório será utilizado pela equipe de desenvolvimento. Dicas de como trabalhar usando o Expresso 3.0 e seu repositório, exemplos de uso e guia de comandos também podem ser encontrados neste documento.

2. Estrutura do Repositório

O repositório do Expresso 3.0 será todo baseado na ferramenta Git. A ideia é trabalhar com dois repositórios na construção do sistema. Um repositório estará no Github (http://github.com/explivre), onde contará com apoio da comunidade para a implementação, abertura de *tickets* e discussões. O outro repositório será hospedado internamente no SERPRO e será usado para versionar partes do sistema que são inerentes aos clientes da empresa. Por exemplo, novos *frontends*, módulos específicos para determinados clientes, relatórios específicos, etc. A estrutura global é apresentada na figura 1.

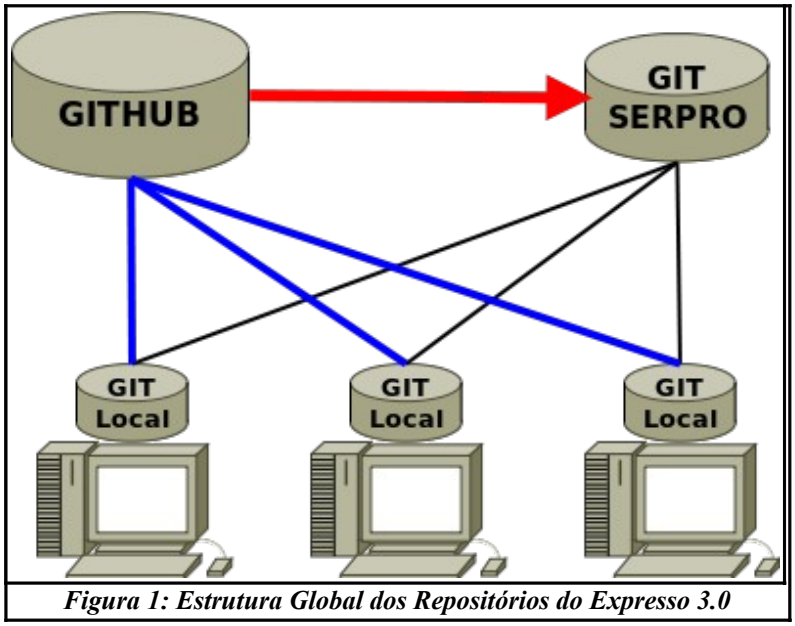

Figura 1: Estrutura Global dos Repositórios do Expresso 3.0

Como o GIT é uma ferramenta de versionamento distribuída, podemos ter GITs locais, onde cada pessoa pode trabalhar com sua versão do repositório. Quando acabar seu trabalho, pode subir seu trabalho para o Github ou para o GIT do SERPRO – vai depender de qual tarefa estava realizando.

Vale salientar, também, que o GIT do SERPRO, de tempo em tempo, terá que pegar as novidades do Github – seja através de comunicação direta com o Github ou através das atualizações dos GITs Locais (que devem estar atualizados com o Github).

2.1. Montagem da Estrutura

Os passos a seguir mostram como montar a estrutura localmente.

a) **Instale o GIT** na sua máquina:

a.1 **Ubuntu**: basta usar o apt-get para instalar o GIT. Use o comando

```
sudo apt-get install git-core
```

com isso, o GIT é instalado na sua máquina. **Opcionalmente**, pode-se instalar também o ***gitg*** (ferramenta GUI para o GIT), o ***git-doc*** (documentação do GIT) e o ***git-completion*** (complementa ou apresenta o(s) comando(s) GIT quando digitamos a tecla **tab**).

a.2 **Windows** e **Mac**: baixe a versão atual do GIT em seu site (http://git-scm.com/download) e instale seguindo as instruções

b) Abra a **Linha de Comando** e vá até a pasta onde deseja criar o repositório do GIT em sua máquina

```
cd /caminho/para/sua/pasta/de/projetos
```

c) Já na pasta, utilize o comando abaixo para **clonar o repositório** do Github

```
git clone https://[USER]@github.com/explivre/ExpressoLivre3.git
```

Onde USER é seu usuário do Github. Ele pedirá a senha do Github para poder baixar uam cópia do repositório atual. Agora, tem acesso ao repositório e todos os seus branches, histórico, etc. Também pode fazer a comunicação sem necessidade de senha, utilizando chaves pública e privada. Para tanto, podem ver o tutorial de como montar uma chave e cadastrá-la no Github – http://help.github.com/linux-set-up-git/ (veja a seção **"Next: Set Up SSH Keys")**.

c.1 **ALTERNATIVAMENTE**, hoje, pode-se **clonar do repositório do SERPRO** usando o comando abaixo – hoje, **é a única opção**, pois no Github ainda não temos uma versão definitiva.

```
git clone ssh://[USER]@[HOST]/[CAMINHO]/expresso3.git
```

Onde:

USER = root; HOST = 10.31.80.37; CAMINHO = usr/local/git.repos

No futuro, a melhor opção será pegar diretamente do Github.

Com a cópia, ele cria uma pasta onde todo o Expresso 3.0 vai estar.

3. Utilização do Repositório

A forma de usar o repositório vai depender da intenção de uso. Teremos dois repositórios remotos, um para a comunidade, outro para trabalhos para clientes do SERPRO (este último, poderia ficar na intranet da empresa). Toda vez que construirmos módulos genéricos, consertarmos *bugs* destes módulos, fizermos uma evolução de algo que seja genérico, estas mudanças devem ser versionadas no repositório da comunidade, no Github.

Quando construirmos algo que seja específico para algum cliente SERPRO, ou mesmo para a própria empresa, esta mudança deve ser versionada no repositório interno. Primeiro, iremos detalhar como trabalhar localmente para depois entrar em detalhes de como trabalhar remotamente. Os comandos são os mesmos se usar o Github ou o GIT do SERPRO como repositórios remotos.

3.1. Utilização Local

É interessante cada projeto ter sua própria zona de trabalho. Conseguimos isso usando um ***branch***. Um *branch* é um ramo onde as modificações que fazemos não afetam o ramo principal de desenvolvimento – por padrão, no Github, este ramo principal chama-se ***master***. Para criar um novo ramo localmente, basta digitar o seguinte comando

```
git branch novo_branch
```

Com isto, um novo *branch* é criado. Podemos visualizar todos os *branches* criados com o seguinte comando

```
git branch
# Saída
* master
  novo_branch
```

O asterisco (*) indica o branch no qual estamos trabalhando no momento. Para mudar de branch, basta usar o comando ***checkout***

```
git checkout novo_branch
```

Pronto, estamos trabalhando agora no novo branch. Agora, toda mudança que for feita, refletirá

apenas neste ramo.

Para adicionar um arquivo em um branch qualquer, basta usar o comando ***add***

```
git add arquivo arquivo2 …
```

Como pode ser observado, podemos adicionar ao GIT vários arquivos ao mesmo tempo. Neste momento, o arquivo está pronto para ser versionado localmente. Para isso, basta roda o seguinte comando

```
git commit -m 'Comentário'
```

Agora a versão foi criada localmente. Estas são as etapas locais de versionamento. Pode-se trabalhar tranquilamente desta maneira, enquanto não houver necessidade de sincronização com os repositórios remotos.

DICA: é interessante que os ramos (*branches*) tenham nomes sugestivos que indiquem qual é o projeto desenvolvido nele. Por exemplo, se estivermos trabalhando com o *Active Sync* em uma branch, poderíamos chamar ela de *ActiveSync* (tudo junto mesmo), ou se for o Mensageria Instantânea, chamar de IM ou MensageriaInstantanea.

3.2. Utilização Remota

Para trabalhar remotamente, primeiro registramos os novos *branches* remotamente (no caso de criação de novos *branches*). Para criar estes novos branches remotamente, precisam usar o comando

```
git push origin novo_branch
```

Branch criado no repositório remoto! A palavra ***origin*** é um alias para o endereço do repositório remoto. Se quiser ver para onde o alias ***origin*** aponta, digite isso na linha de comando

```
git remote -v
# Saída
origin  https://[USER]@github.com/project/folder.git (fetch)
origin  https://[USER]@github.com/project/folder.git (push)
```

Esta é a lista de alias. Ele mostra todos os alias com seus endereços para ***push*** – subir as novas versões – e ***fetch*** – baixar novas versões. Podemos criar novos alias para outros repositórios remotos e sempre atualizar mais de um repositório, caso seja necessário. Agora, podemos iniciar o processo para versionar mudanças no Github dentro da *branch* correta. Basta usar o comando abaixo

```
git push
```

Para funcionar corretamente, tem que estar na mesma *branch* onde deseja atualizar. Isto porque, caso não indique para que *branch* está enviando a nova versão, ele envia para a *branch* na qual está trabalhando. Se colocar uma *branch* depois de ***push***, ele envia de *branch* local para *branch* remota

(sempre usando a indicada).

Por exemplo, se usar o comando

```
git push origin novo_branch
```

Ele enviar da branch local novo_branch para a branch remota novo_branch, independente em que branch local esteja trabalhando.

Para atualizar seu versionamento local a partir do repositório remoto, temos dois comandos: ***fetch*** e ***pull***. O primeiro comando (***fetch***) verifica as modificações dentro do ramo onde está e baixa as modificações. Assim, consegue fazer comparações, ver o que está diferente, etc. O segundo (***pull***) faz tudo o que ***fetch*** faz e ainda por cima tenta fazer uma junção (***merge***) automática das informações. Ou seja, se tiver arquivos novos, ele cria; se tiver arquivos tiverem novos conteúdos, ele atualiza, etc. Resumindo, ***pull*** = ***fetch*** + ***merge***. Um exemplo de uso destes comandos, pode ser visto a seguir:

```
# Baixa as modificações para o branch local no qual
# está trabalhando a partir do master remoto
git fetch origin master

# Atualiza o branch local no qual está trabalhando
# com o branch master remoto
git merge origin master

# Baixa e atualiza o branch local no qual
# está trabalhando com o branch master remoto
git pull origin master
```

No caso do ***merge***, a tentativa de mesclar as informações pode gerar conflitos, onde trechos de um arquivo podem ter mudanças diferentes nas versões. O próprio resultado do ***merge*** indica estes tipos de problema. Quando isto acontecer, podemos ver as mudanças usando o comando ***diff***. Basta usar da seguinte maneira

```
git diff master novo_branch
```

Assim poderemos ver as mudanças entre as duas branches, ***master*** e ***novo_branch***. Várias IDEs, como Eclipse e Netbeans, também têm este recurso, com uma melhor apresentação, ajudando nosso trabalho. O uso do ***diff*** também pode ser feito com duas branches locais.

3.3. Exemplo de Uso

Para mostrar como podemos trabalhar com o Github, por exemplo, fica um passo-a passo inicial.

Primeiro, criamos as *branches* nas quais iremos trabalhar localmente (figura 2)

```
git branch IM
```

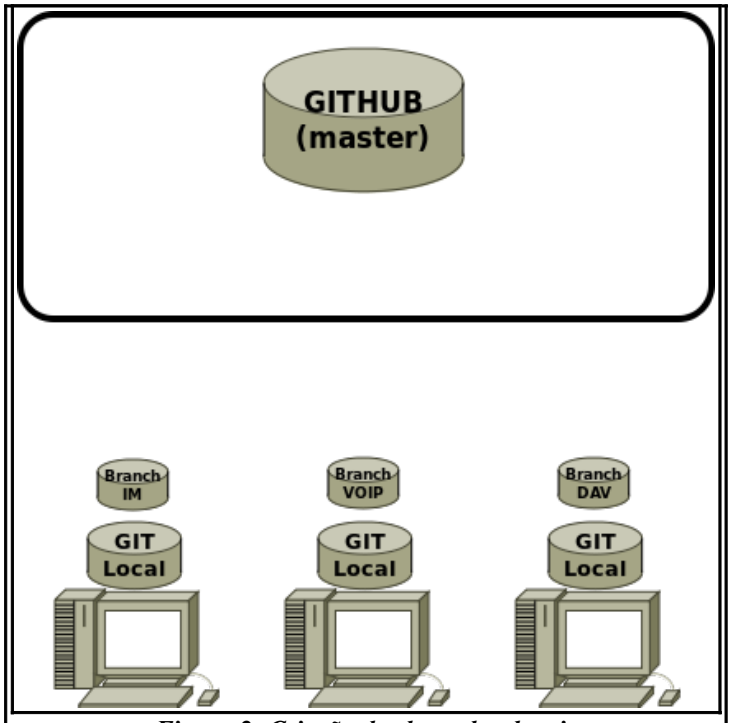

Figura 2: Criação das branches locais

Depois, subimos (criamos) as *branches* para o repositório remoto – Github, por exemplo (figura 3). Com isto, podemos compartilhar nossa *branch* com outras pessoas que queiram trabalhar em algum projeto especificamente.

```
git push origin IM
```

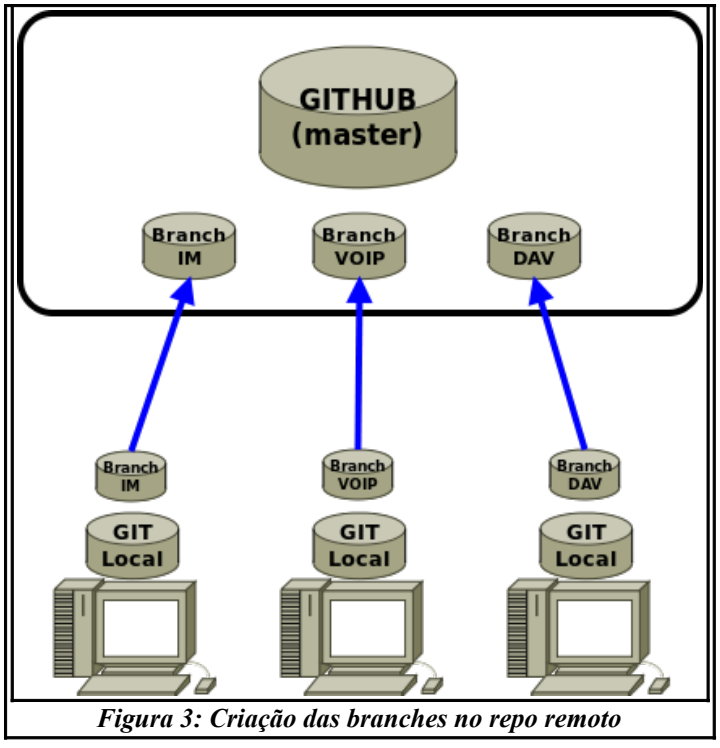

Figura 3: Criação das branches no repo remoto

Em seguida, atualizamos o *master* local através do *master* remoto (figura 4).

```
git checkout master
git fetch origin master
git merge origin master
# OU esta abaixo, que substitui as duas últimas linhas
git pull origin master
```

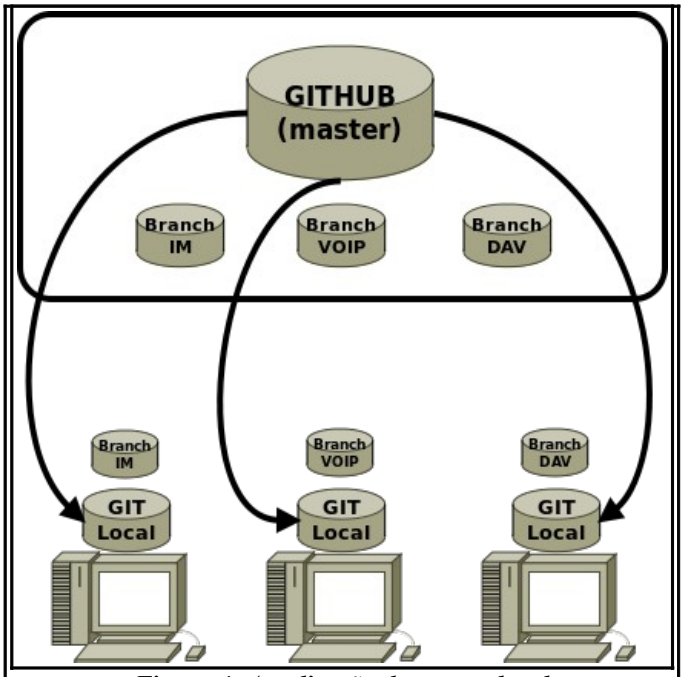

Figura 4: Atualização do master local

Depois, atualizamos nossa *branch* local, a partir do master local (figura 5).

```
git checkout IM
git merge master
```

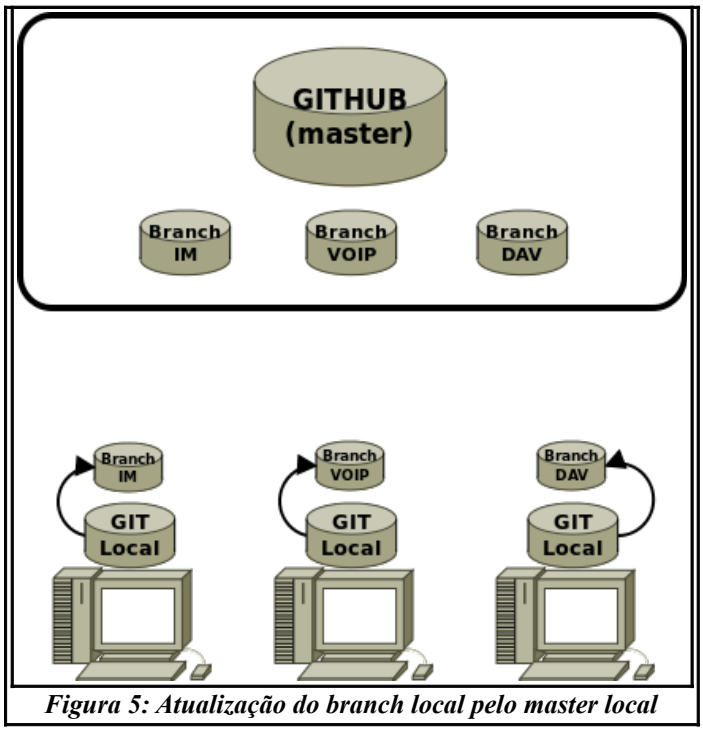

Figura 5: Atualização do branch local pelo master local

Desta forma já podemos trabalhar em nosso projeto dentro de nosso ambiente controlado (*branch*).

Na hora em que quisermos versionar as mudanças que fizemos, podemos versionar localmente, usando os comandos ***`add`*** e ***`commit`***, da maneira como já foi mostrado. É possível também versionarmos remotamente, usando o comando ***`push`*** (se estiver no branch correto, ele envia de branch local para branch remoto)

```
git checkout IM
git push
```

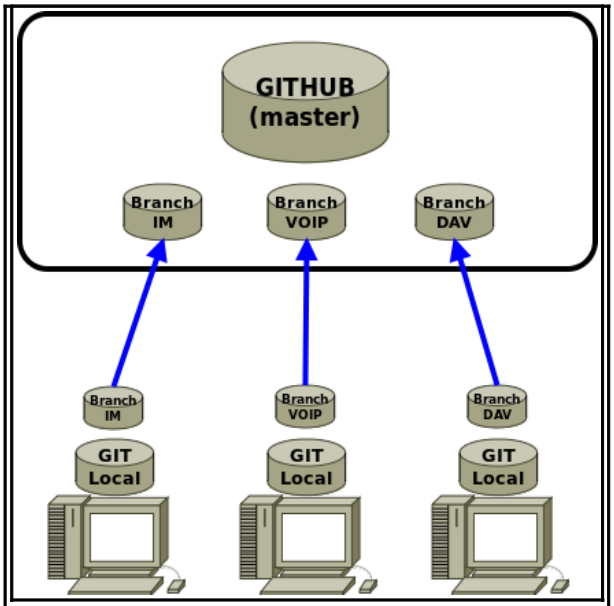

Figura 6: Atualização de branch remota

O mais interessante é testarmos a versão da nossa *branch* local. Se todos os testes passarem, fazemos um *merge* com o GIT Local (figura 7).

```
git checkout master
git merge IM
```

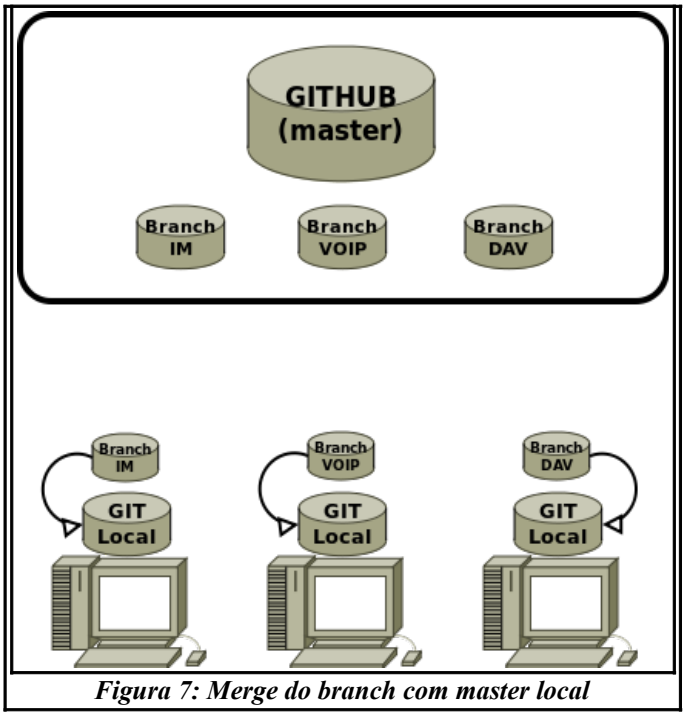

Figura 7: Merge do branch com master local

E logo depois, quando os testes de integração passarem localmente, por exemplo, subimos as mudanças do *master* local para o *master* remoto (figura 8).

```
git push origin master
```

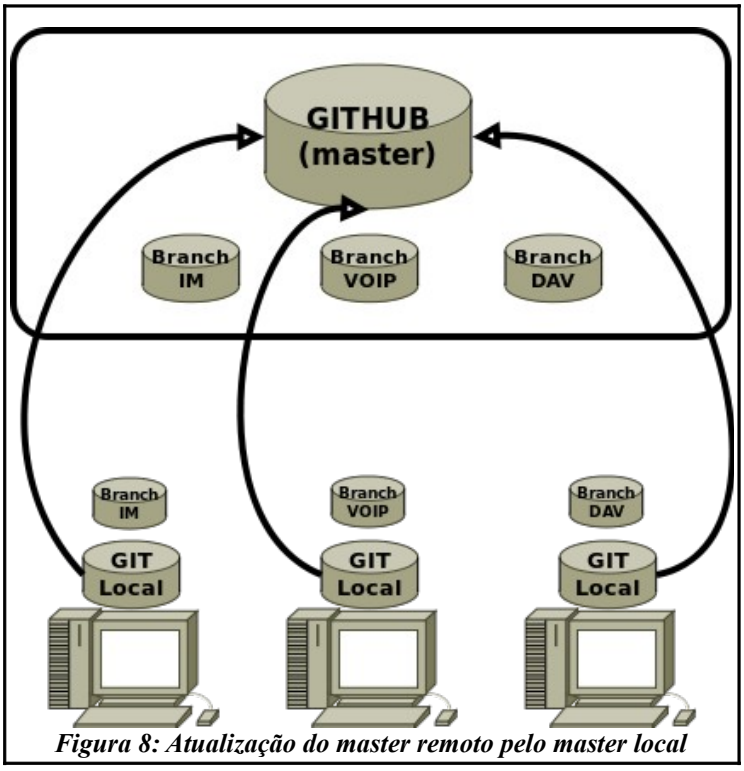

Figura 8: Atualização do master remoto pelo master local

3.4. Estrutura padrão inicial

Inicialmente, teremos os seguintes *branches* (representando cada um dos projetos):

- Projeto Frontend (Porto Alegre) => *branch* Frontend
- Projeto Mensageria Instantânea (Salvador) => *branch* IM
- Projeto VOIP (Fortaleza) => *branch* VOIP
- Projeto CalDAV, WebDAV, etc (Brasília) => *branch* DAV
- Projeto Active Sync (Curitiba) => *branch* ActiveSync

Basta criar a respectiva *branch* local e iniciar os trabalhos. Para disponibilizar para outras pessoas trabalharem, basta dar um *push* da respectiva branch para o repositório remoto.

4. Disposições Finais

Qualquer dúvida ou sugestão relativa a estrutura ou utilização, podem enviar um e-mail para marcio.albuquerque@serpro.gov.br. Se quiserem saber sobre os comandos do GIT, acessem o site http://gitref.org. Existem vários, mas este é bem sucinto e explica bem. Se quiserem um mais completo e em português, podem acessar o post de Fábio Akita sobre o assunto – http://akitaonrails.com/2008/04/02/micro-tutorial-de-git.