PJS – Projeto de Sistemas

Bruno Oliveira

Guia de comandos DOS e Git.

Comandos DOS

Comando	Descrição	Sintaxe
ls	Lista o diretório (pasta) atual	ls
ls -l	Lista o diretório em ordem e em lista	ls -l
ls -a	Lista os arquivos ocultos do diretório atual	ls -a
mkdir	Cria uma pasta no diretório atual	mkdir nome_pasta
cd	Acessa uma pasta	cd nome_pasta
cd ..	Volta uma pasta	cd ..
touch	Cria um arquivo	touch arquivo.txt

Esses comando DOS, são os básicos necessários para se trabalhar na interface terminal do git, com o tempo e utilização esses comandos se tornarão habituais.

Comandos Git

Os comandos listados a seguir são exclusivos da interface git.

Configuração Inicial

Quando você inicia o git pela primeira vez, há algumas configurações que devem ser feitas antes da utilização, como por exemplo, a criação da chave assimétrica, configuração do usuário e e-mail para registro nas versões.

Aproveitando que o git inicia em uma pasta padrão (c:/Users/nome_usuario), vamos criar a chave assimétrica, também conhecida como chave pública e chave privada, para acesso e autenticação de repositórios em nuvem.

BrunoOliveira@Notebook-HP ~
$ ssh-keygen -t rsa -C “bl.oliveira@outlook.com” <enter>

Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/Bruno Oliveira/.ssh/id_rsa): <enter>
Created directory '/c/Users/Bruno Oliveira/.ssh'.
-- Aqui será pedido que você insira uma senha para garantir a segurança da sua instância do git , mas fica a seu critério, caso não queira registrar uma senha apenas aperte o <enter>, se optar por cadastrar uma senha, tome cuidado, ao digitar o git não exibe nada, mas tudo que você estiver pressionando está sendo registrado, então tenha atenção. –
Enter passphrase (empty for no passphrase):

Enter same passphrase again:

Your identification has been saved in /c/Users/Bruno Oliveira/.ssh/id_rsa.
Your public key has been saved in /c/Users/Bruno Oliveira/.ssh/id_rsa.pub.
The key fingerprint is: e4:a5:39:8b:6e:30:d3:43:70:8b:c5:22:6e:56:c0:4c.

Feito isso a sua chave pública e privada está criada, e está salva na pasta padrão:

C:/Users/Nome_Usuario/ .ssh/

Verifique esta pasta no seu computador, liberando a visualização de pastas ocultas no gerenciador de pastas do Windows, pois mais a frente será necessário saber sua localização.

Obs: tenha essa chave sempre com você, pois você poderá usa-la para se conectar ao git de qualquer computador que tenha o git instalado, então a guarde, e quando for usa-la coloque-a sempre na pasta padrão indicada acima.

Chave criada agora vamos a configuração do git, primeiro vamos definir o usuário e e-mail do utilizador, no caso você.

```
BrunoOliveira@Notebook-HP ~
$ git config --global user.name “Bruno Oliveira”          <enter>

BrunoOliveira@Notebook-HP ~
$ git config --global user.email “bl.oliveira@outlook.com”      <enter>
```

Após esse comando o git estará configurado para todos os repositórios que você irá criar, caso em algum repositório você precise mudar o nome ou e-mail, pode-se usar o mesmo comando sem o parêmetro --global, exemplo:

```
BrunoOliveira@Notebook-HP ~\Documents\Teste
$ git config user.name “Linus Torvalds”                   <enter>
```

Assim o nome ficará registrado apenas para o repositório atual. O mesmo vale para o e-mail.

Inicializando um Repositório

Primeiramente, antes de criarmos o repositório vou defini-lo de forma clara, um repositório nada mais é que uma pasta. Usamos a nomenclatura repositório pelo propósito de não confundir com a pasta de projeto e com a pasta temporária (Área de Estágio) criada pelo git.

Quando inicializamos um repositório o git cria uma pasta oculta (.git), dentro da pasta de trabalho onde estará o projeto, essa pasta oculta conterá todas as informações do repositório, além de ser o servidor de referências para as versões criadas.

Para criarmos um repositório, precisamos primeiro estar na pasta do projeto que iremos trabalhar, então utilize os comandos DOS mostrados anteriormente para acessar a pasta do projeto antes de iniciar o repositório git.

Estando na pasta do projeto use o seguinte comando:

```
BrunoOliveira@Notebook-HP ~/Documents/ProjetoTesteGit
$ git init        <enter>

Initialized empty git repository in c:/User/BrunoOliveira/Documents/
ProjetoTesteGit/.git/
```

Ao aparecer essa mensagem o seu repositório estará criado e dentro da pasta do projeto será criada a pasta .git, em seguida na linha de identificação do usuário e da pasta do projeto aparecerá entre parênteses a palavra máster que identifica uma ramificação, não se preocupe com isso neste momento, serão abordadas as ramificações no momento adequado, mas a principio isso identifica que este é um repositório git.

```
BrunoOliveira@Notebook-HP ~/Documents/ProjetoTesteGit (master)
$
```

Se você ainda não tiver configurado o nome e e-mail, ou mesmo se quiser mudar o nome e e-mail neste repositório em específico, utilize o comando git config na sua versão reduzida, sem o parâmetro --global.

Ignorando arquivos

Outra configuração que devemos fazer sempre que iniciamos um projeto é sobre quais arquivos não devem ser afetados nas criações das versões, para não carregarmos o servidor tanto local como em nuvem com arquivos desnecessários, ou até mesmo grandes demais como banco de dados. Pois bem para ignora-los criamos um arquivo texto onde serão indicadas as extensões dos arquivos que não queremos que façam parte do commit.

Exemplo de arquivos que ignoramos: .exe, .bin, .sql, .base, .bat, enfim arquivos que são criados a cada execução de um projeto, ou até mesmo arquivos temporários.

Este arquivo texto receberá o nome de .gitignore, porém quando vamos salvar um arquivo texto no bloco de notas ou qualquer outro editor de textos ele impede que nomeamos um arquivo iniciando seu nome com ponto (. gitignore).

Para criar esse arquivo, primeiro no bloco de notas listamos as extensões que não queremos que o git trabalhe, em seguida salvamos esse arquivo texto na pasta do projeto e usamos o seguinte comando para renomear com a extensão .gitignore, exemplo:

```
BrunoOliveira@Notebook-HP ~/Documents/ProjetoTesteGit (master)
$ mv arquivo.txt .gitignore
```

Esse arquivo passara a ser um arquivo oculto na pasta com o nome de .gitignore e todos os nomes e extensões que estiverem listados dentro desse arquivo serão ignorados pelo git na criação dos commits.

Trabalhando com o GIT

Para começar de fato a usar o git, devemos conhecer a forma básica de trabalho dele. O Git é baseado em três ações principais que são sempre executadas para gerar uma versão.

- Criar ou modificar um arquivo;
- Adiciona-los na área de estágio;
- Criar o commit;

Essas são as ações básicas, sempre irão acontecer, não importa o tamanho do projeto ou mesmo o número de programadores que estejam trabalhando. Seguindo o nosso tutorial, já temos a pasta de trabalho, temos o repositório criado e configurado agora vamos criar nosso primeiro commit.

Sempre antes de criar um commit devemos salvar o arquivo do projeto, se for um documento, ou uma manipulação de imagem salve as alterações antes, ou mesmo que ainda não tenha feito nada salve o arquivo antes.

Obs: No caso de um projeto no Visual Studio, execute o programa sempre antes de criar o commit.

Depois de executado ou salvo o projeto, volte no git, estando no git iremos verificar antes os arquivos que temos na pasta com o seguinte comando:

```
BrunoOliveira@Notebook-HP ~/Documents/ProjetoTesteGit (master)
$ git status              <enter>
```

```
# On branch master
#
# Initial commit
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       .gitignore
#       ProjetoTesteGit.sln
#       ProjetoTesteGit.v11.suo
#       ProjetoTesteGit/
nothing added to commit but untracked files present (use "git add" to track)
```

Atenção aqui, repare que são dadas diversas informações iniciais, primeira linha mostra a ramificação que você está, no caso a master, ainda não abordamos ramificações então não se preocupe com isso ainda, a próxima informação, indica que o commit a ser realizado, mas também não se preocupe com essa linha, agora a partir da linha que contem a descrição Untracked Files, é a parte que nos interessa, essa mensagem indica que há arquivos que ainda não foram adicionado na área de estágio, ou seja ainda não foram “commitados”.

Os arquivos aparecem em vermelho quando ainda não estão na área de estágio. Então vamos adiciona-los na área de estágio, para fazer isso usamos o comando add.

```
BrunoOliveira@Notebook-HP ~/Documents/ProjetoTesteGit (master)
$ git add .             <enter>
```

Dessa forma adicionamos todos os arquivos na área de estágio, é a forma mais simples, outro método possível, é adicionar arquivo por arquivo, no caso de querer ignorar algum arquivo que você não quer que seja usado na nova versão, mas como já criamos o arquivo .gitignore então podemos usar o comando git add . para adicionar todos os arquivos de uma vez na área de estágio.

Vamos conferir agora como estão os arquivos com o comando git status.

```
BrunoOliveira@Notebook-HP ~/Documents/ProjetoTesteGit (master)
$ git status            <enter>
```

```
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   .gitignore
#       new file:   ProjetoTesteGit.sln
#       new file:   ProjetoTesteGit.v11.suo
#       new file:   ProjetoTesteGit/App.config
#       new file:   ProjetoTesteGit/Form1.Designer.cs
#       new file:   ProjetoTesteGit/Form1.cs
#       new file:   ProjetoTesteGit/Program.cs
#       new file:   ProjetoTesteGit/ProjetoTesteGit.csproj
#       new file:   ProjetoTesteGit/Properties/AssemblyInfo.cs
#       new file:   ProjetoTesteGit/Properties/Resources.Designer.cs
#       new file:   ProjetoTesteGit/Properties/Resources.resx
#       new file:   ProjetoTesteGit/Properties/Settings.Designer.cs
#       new file:   ProjetoTesteGit/Properties/Settings.settings
#
```

Note que as mensagens mudaram e a cor dos arquivos também, a linha que antes estava Untracked Files agora está como Changes to be Committed, essa mensagem indica que os arquivos estão prontos para serem “Commitados”, então vamos fazer o commit.

Para fazer o commit usamos o comando git commit.

```
BrunoOliveira@Notebook-HP ~/Documents/ProjetoTesteGit (master)
$ git commit -m “Mensagem indicando o que foi feito”          <enter>
```

As informações a seguir serão exibidas:

```
[master (root-commit) d16604c] Inicio do Projeto
23 files changed, 477 insertions(+)
create mode 100644 .gitignore
create mode 100644 ProjetoTesteGit.sln
create mode 100644 ProjetoTesteGit.v11.suo
create mode 100644 ProjetoTesteGit/App.config
create mode 100644 ProjetoTesteGit/Form1.Designer.cs
create mode 100644 ProjetoTesteGit/Form1.cs
create mode 100644 ProjetoTesteGit/Program.cs
create mode 100644 ProjetoTesteGit/ProjetoTesteGit.csproj
create mode 100644 ProjetoTesteGit/Properties/AssemblyInfo.cs
create mode 100644 ProjetoTesteGit/Properties/Resources.Designer.cs
```

create mode 100644 ProjetoTesteGit/Properties/Resources.resx
create mode 100644 ProjetoTesteGit/Properties/Settings.Designer.cs
create mode 100644 ProjetoTesteGit/Properties/Settings.settings

Essas informações indicam quantas alterações foram feitas, em que arquivos ocorreram as alterações, além do código de referência e a mensagem do commit. Agora temos um commit feito. Vamos verificar esse commit com o comando git log

BrunoOliveira@Notebook-HP ~/Documents/ProjetoTesteGit (master)
$ git log <enter>

Aqui são exibidas informações sobre os commits feitos, nestas informações constam o código do commit representado por uma sequencia alfanumérica de 40 bytes, também conhecida como identificação com chave de 160 bits, o autor do commit, o contato de e-mail, a data e hora do commit e por fim a mensagem do commit.

commit d16604cd368be0a467e82a1777fbac632fff04d7
Author: Bruno Oliveira bruni.nho_oliveira@hotmail.com
Date: Tue Aug 27 11:03:29 2013 -0300

Inicio do Projeto

Conforme forem sendo feitos os commits a lista do log vai aumentando, e mostrando tanto os seus commits quanto os commits feitos por outros membros que estejam trabalhando com você. Quando a lista do log ficar muito grande e não couber na visualização ele bloqueara os comandos para voltar a tela de comandos pressione a tecla (q).

Bom feito o commit, podemos alterar o projeto novamente e repetir os passos desse tópico, trabalhando com o git, o processo é sempre o mesmo, modificamos, adicionamos na área de estágio e criamos o commit.

Trabalhando com Ramificações e alternando entre versões

Uma nova etapa de trabalho com o git é a criação de ramificações, para entende uma ramificação vamos imaginar uma estrada, você está nesta estrada e em um determinado ponto você se depara com uma bifurcação, ou seja, a estrada passa a ter dois caminhos diferentes.

A mesma ideia acontece no git, você está fazendo os commits e atrás do outro, seguindo os passos já ensinados você está formando um caminho, para exemplificar melhor isso, estaremos usando um grafo como o mostrado a seguir.

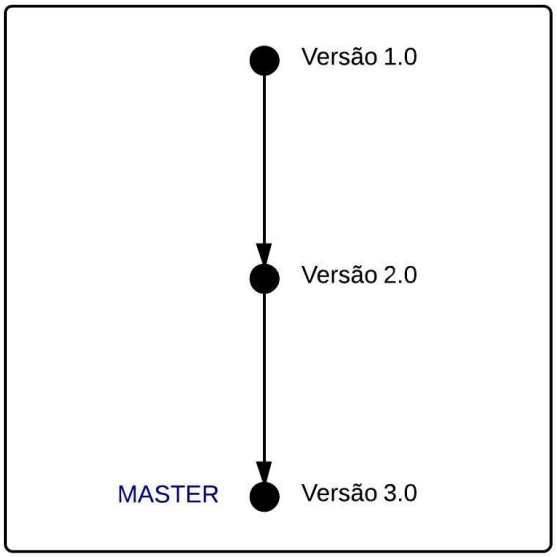


Figura 1 - Exemplo de grafo de versões

Analise bem esse grafo, você consegue perceber que é um fluxo continuo, sempre uma versão após a outra. De fato esse é o processo básico do git, sempre que você terminar qualquer documento você tem um grafo com a versão final sendo o último ponto indicado com o rótulo MASTER.

Isso acontece quando você trabalha em um projeto sozinho e quando você faz modificações em apenas um projeto. Agora vamos ver um grafo que possui uma ramificação.

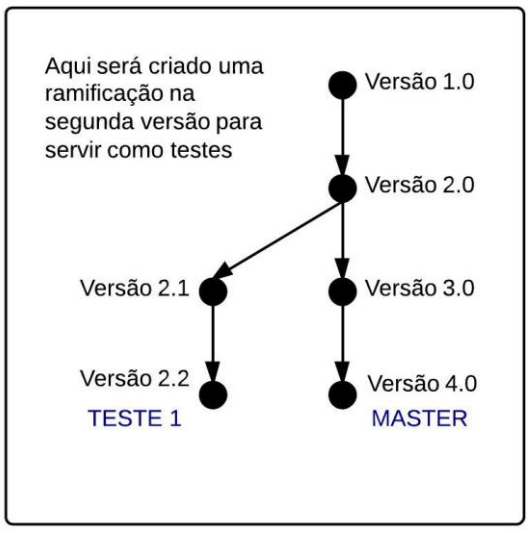


Figura 2 - Exemplo de grafo com ramificação

Note que tivemos uma alteração no grafo, criamos a ramificação com nome TESTE 1, para trabalhar em alguma modificação sobre a Versão 2.0 do nosso sistema.

Provavelmente pode surgir a pergunta, mas por que usar uma ramificação ainda mais trabalhando no mesmo computador? Não parece óbvio no começo mesmo, mas é bem simples, vamos explicar a partir de um exemplo.

Digamos que você esteja trabalhando em um projeto, esteja elaborando um documento ou mesmo um desenvolvimento de uma aplicação, e você está fazendo as modificações e consequentemente criando as versões com o git, e em um determinado momento do projeto surge uma funcionalidade para você inserir neste software que você nunca fez, ou que tenha um pouco de dificuldades para executar, ou no caso de um documento que você já formatou e te mandaram um trecho todo fora da formatação e pior que deve ser adicionado em várias partes do documento. Enfim alguma modificação que de certa forma vá atrapalhar o andamento do projeto, que precise ser testada antes para saber se realmente vai funcionar.

Então, agora imagine que você adicionou essa funcionalidade diretamente no projeto principal e conforme vai fazendo outras coisas, vai desenvolvendo esta nova função, e fazendo os commits, criando as versões no fluxo continuo, mas para que essa funcionalidade passe a funcionar você precisa atrelar ela a vários outros arquivos que já estão certos. Até ai nada de assustador não é? Mas digamos que o cliente mudou de ideia (o que acontece e muito), agora pare e pense, você estava trabalhando em um mesmo fluxo, ou seja, conforme foi fazendo os commits alterou essa função e mais dezenas de outras funções e precisa tirar essa nova função do código.

O que fazer? Voltar a versão? mas isso fará com que você perca as outras modificações que já havia feito junto com a nova funcionalidade.

Porém você aprendeu a criar ramificações, e criou uma ramificação de teste no seu projeto e nessa ramificação você implementou essa funcionalidade que o cliente pediu e ao mesmo tempo que você voltava a ramificação principal (Master), e continuava o desenvolvimento original do seu software ou documento. O que acontece agora?

Percebe que facilitou? Pois aquela ramificação teste serviu como uma cópia do projeto, você pode fazer qualquer modificação nela sem afetar a versão principal do projeto, e se a funcionalidade não for utilizada você pode simplesmente descartar aquela ramificação sem ter perdas na continuidade do projeto. E no caso da funcionalidade funcionar e tiver que ser implementada no projeto original, basta você mesclar as ramificações, bem mais fácil não é? Bom vamos a parte prática então, neste tópico abordarei a criação das ramificações, e como alternar entre as mesmas e entre versões, no caso de

mesclar, fique tranquilo(a), o próximo tópico tratará apenas sobre a mescla de ramificações.

Vamos aproveitar o projeto que já estávamos fazendo e vamos criar as ramificações por ele. O projeto está assim:

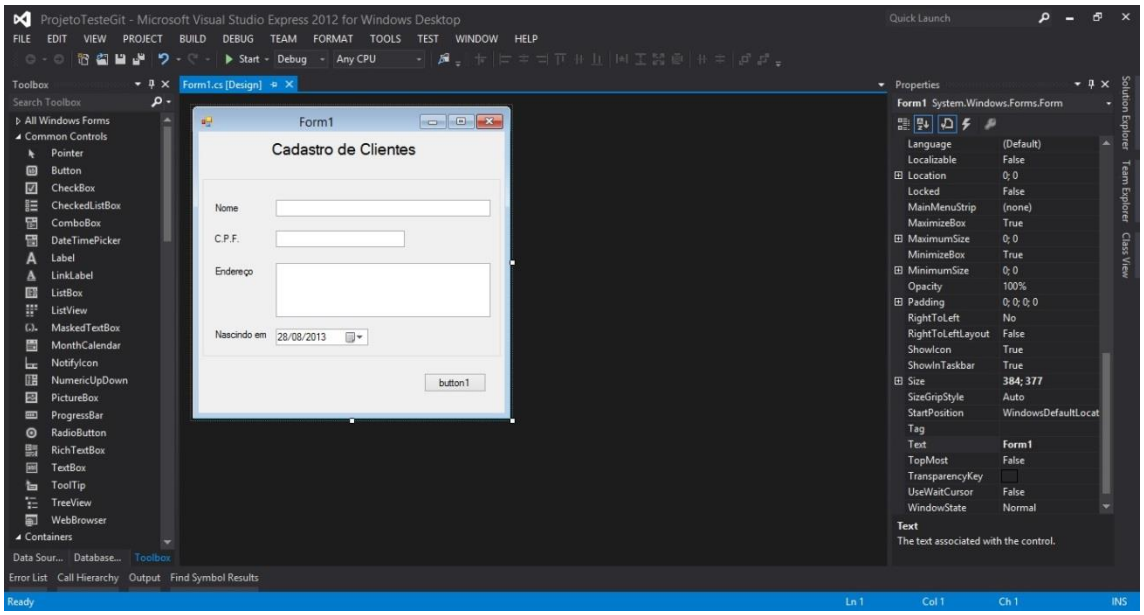


Figura 3 - Tela exemplo de estado atual do projeto

Iremos criar uma ramificação para adicionar a inserção de um imagem para registrar com o cadastro. Vamos testar a partir da ramificação como ficará o layout do projeto, e ao mesmo tempo na ramificação principal trabalharemos com a função do botão de cadastro.

Antes de criar uma ramificação vamos ver quais ramificações temos com o seguinte comando:

```
BrunoOliveira@Notebook-HP ~/Documents/ProjetoTesteGit (master)
$ git branch                    <enter>

* master
```

Após o comando visualizamos que existe apenas a ramificação master e que estamos utilizando esta atualmente, pela indicação do asterisco (*).

Agora vamos criar uma ramificação, para criar uma ramificação a partir do git usamos o seguinte comando:

```
BrunoOliveira@Notebook-HP ~/Documents/ProjetoTesteGit (master)
$ git branch Teste              <enter>
```

Com isso criamos a ramificação teste do nosso projeto, vamos verificar usando o comando anterior git branch.

```
BrunoOliveira@Notebook-HP ~/Documents/ProjetoTesteGit (master)
$ git branch                    <enter>

* master
  teste
```

Agora as informações são, duas ramificações, a master e a teste, sendo que estamos usando a master.

Detalhe ainda não estamos com projeto na ramificação teste, para exemplificar isso mostraremos o grafo a seguir.

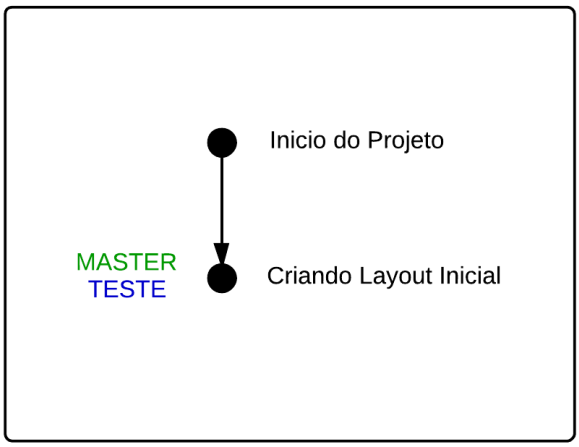


Figura 4 - Situação atual do projeto - Versões já criadas

Na figura 4, temos dois commits, e a indicação das duas ramificações, porém ainda não dividimos os caminhos, para que tenhamos de fato uma ramificação temos que ter um commit diferente em cada uma das ramificações, para fazer os commits em cada ramificação temos antes que estar na ramificação para fazer o commit, no nosso caso estamos na ramificação master para verificar use o comando git branch.

```
BrunoOliveira@Notebook-HP ~/Documents/ProjetoTesteGit (master)
$ git branch                    <enter>

* master
  teste
```

Agora vamos mudar de ramificação, com o seguinte comando:

```
BrunoOliveira@Notebook-HP ~/Documents/ProjetoTesteGit (master)
$ git checkout teste            <enter>
```

Switched to branch 'teste'

Usando o git checkout, conseguimos alternar entre ramificações e até mesmo entre versões o que será mostrado a logo mais. Agora para ver que alteramos de ramificação preste atenção nos seguintes pontos.

```
BrunoOliveira@Notebook-HP ~/Documents/ProjetoTesteGit (teste)
$
```

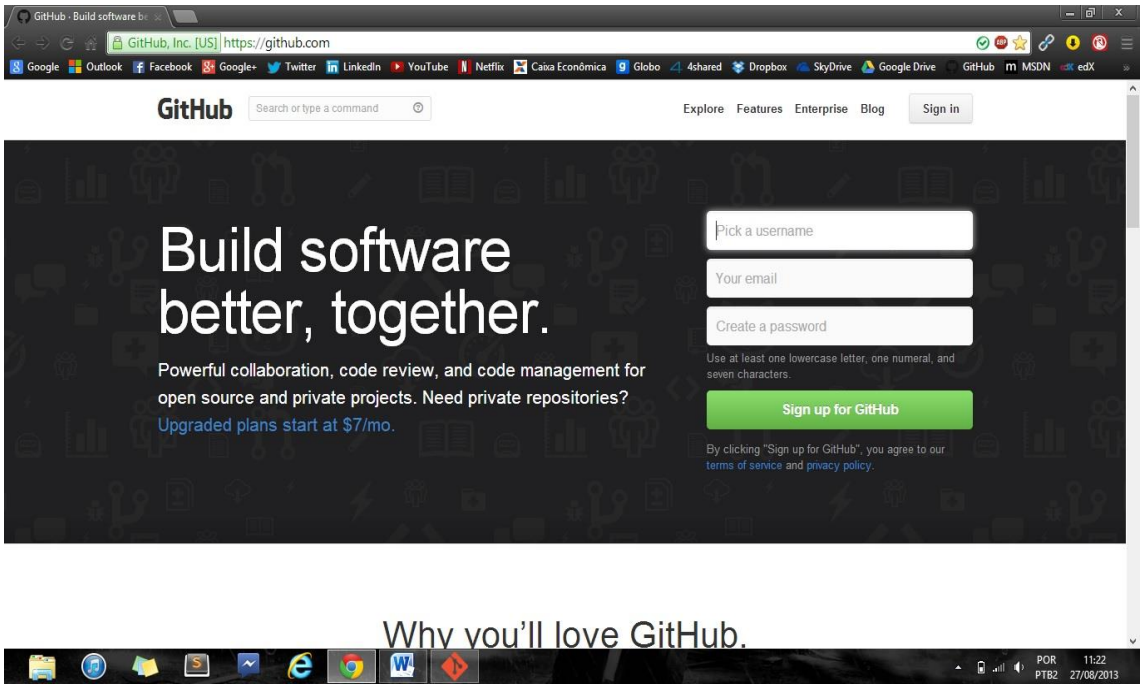
Note que a indicação ao lado do endereço da pasta do projeto mudou de master para teste. E se executar o comando git branch veremos outra mudança.

```
BrunoOliveira@Notebook-HP ~/Documents/ProjetoTesteGit (teste)
$ git branch                    <enter>

  master
* teste
```


Já vimos como trabalhar na forma local com o git, agora usaremos um novo recurso, o repositório em nuvem, para isso temos que ter um servidor online para armazenar o projeto e fazer algumas novas configurações.

O git trabalha com um repositório em nuvem chamado de GITHUB acessado pelo endereço www.github.com , no primeiro acesso é necessário fazer um cadastro no site, informando um nome de usuário, um e-mail válido e a senha de acesso.


GitHub - Build software be
GitHub, Inc. [US] https://github.com
Google Outlook Facebook Google+ Twitter LinkedIn YouTube Netflix Caixa Econômica Globo 4shared Dropbox SkyDrive Google Drive GitHub MSDN edX
GitHub
Search or type a command
Explore Features Enterprise Blog
Sign in
Build software better, together.
Powerful collaboration, code review, and code management for open source and private projects. Need private repositories?
Upgraded plans start at $7/mo.
Pick a username
Your email
Create a password
Use at least one lowercase letter, one numeral, and seven characters.
Sign up for GitHub
By clicking "Sign up for GitHub", you agree to our terms of service and privacy policy.
Why you'll love GitHub.
POR
PTB2
11:22
27/08/2013