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.

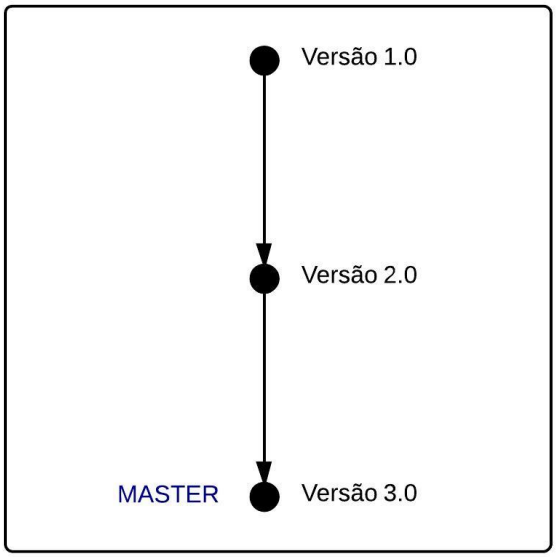

Figura 1 - Exemplo de grafo de versões

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.

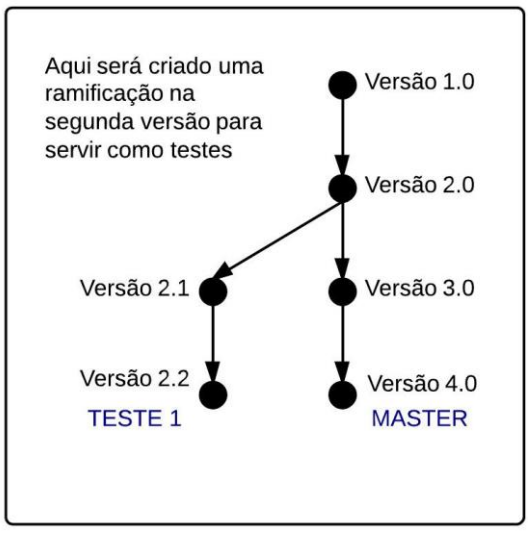

Figura 2 - Exemplo de grafo com ramificação

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:

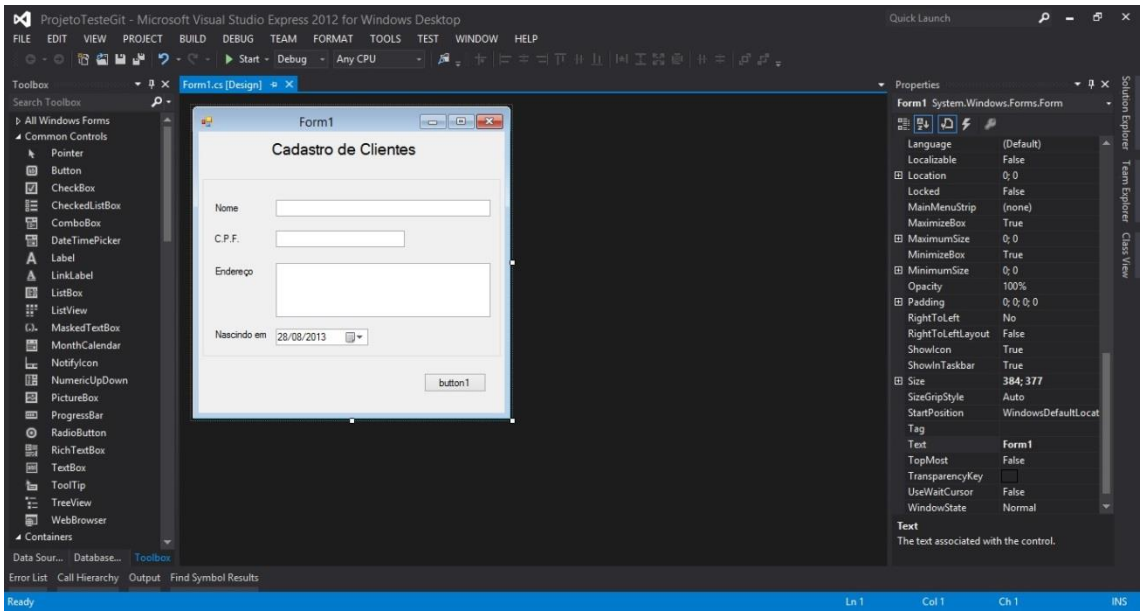

Figura 3 - Tela exemplo de estado atual do projeto

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.

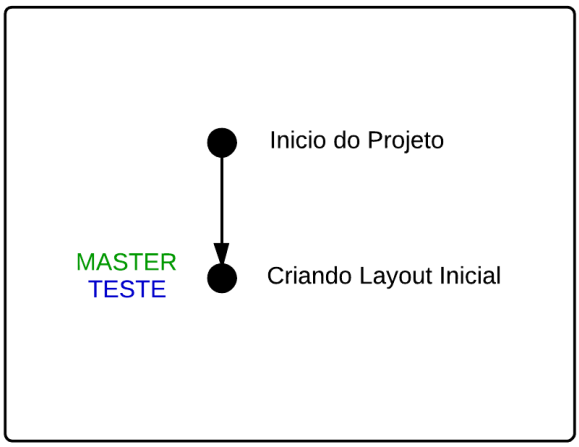

Figura 4 - Situação atual do projeto - Versões já criadas

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
```

Agora de fato estamos na ramificação teste, então para alternarmos de uma ramificação para outra devemos usar o comando git checkout. Agora vamos fazer um commit na ramificação teste e depois um commit na ramificação master e em seguida visualizar o nosso grafo.

Lembre-se de seguir as etapas já aprendidas anteriormente para fazer um novo commit nas duas ramificações.

No nosso projeto, foi adicionado um novo botão que será salvo e gerado uma versão na nossa ramificação teste. O nosso projeto está assim agora:

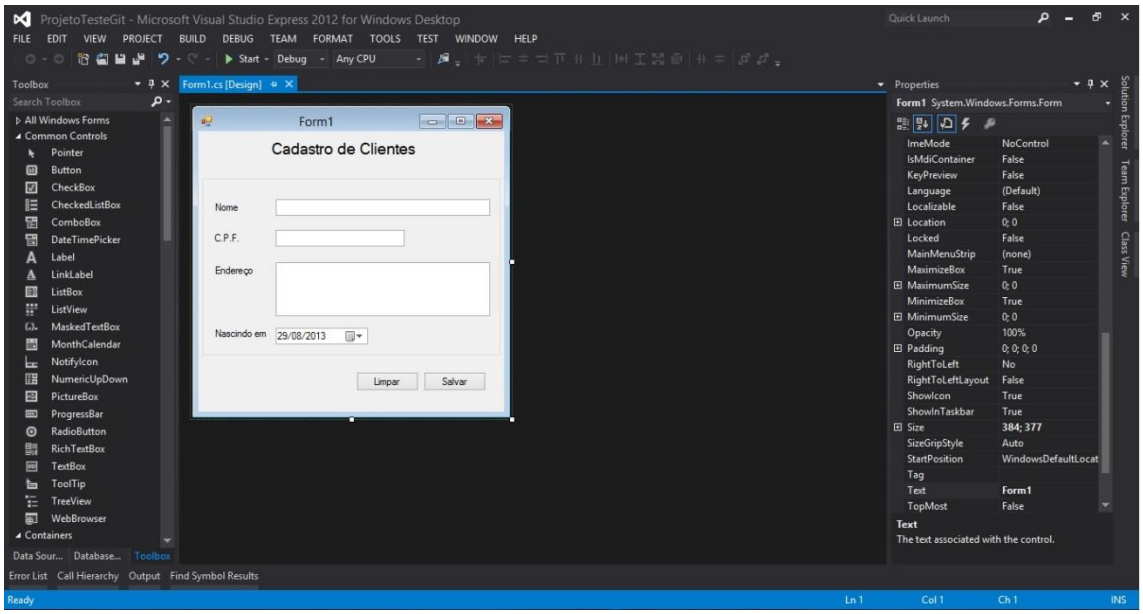

Figura 5 - Alteração no projeto - Adicionado o botão limpar

Agora que alteramos vamos fazer o commit, depois de feito o commit, vamos executar o comando git log para visualizar a lista de commit que temos nesse instante, lembrando que estamos na ramificação teste.

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

commit 2ac0292e23d65dea888f13d56b47fc1f6cc89bd0
Author: Bruno Oliveira bruni.nho_oliveira@hotmail.com
Date:   Thu Aug 29 20:23:30 2013 -0300

    Adicionado o botao limpar

commit 22e85b33686175c6be00ae43a44364b52d516db8
Author: Bruno Oliveira bruni.nho_oliveira@hotmail.com
Date:   Wed Aug 28 09:00:58 2013 -0300

    Criando layout inicial do formulario de cadastro

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

    Inicio do Projeto
```

Repare que estão listados os commits feitos, até então nenhuma novidade. Agora vamos voltar para a ramificação master e rever o log de commits.

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

Feito isso voltamos à ramificação master e se verificarmos o log de commits, veremos o seguinte:

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

commit 22e85b33686175c6be00ae43a44364b52d516db8
Author: Bruno Oliveira bruni.nho_oliveira@hotmail.com
Date:   Wed Aug 28 09:00:58 2013 -0300

    Criando layout inicial do formulario de cadastro

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

    Inicio do Projeto
```

Pode-se perceber que o ultimo commit que fizemos sumiu, mágica? Mistério? Não, isso é o git e o que faz dele ser uma ferramenta tão disseminada no mundo. Mas vamos explicar o que aconteceu, quando fizemos o commit anterior estávamos na ramificação teste, lembra? Então agora estamos na ramificação master e o git voltou todo projeto para o ultimo commit feito na ramificação master. Vamos ver como está o projeto.

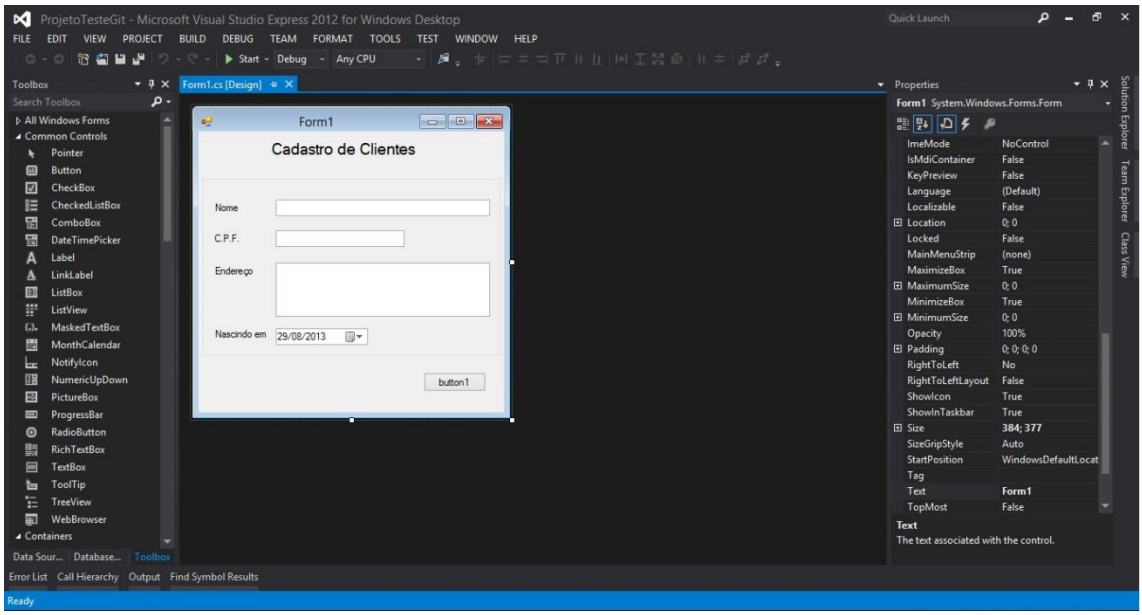

Figura 6 - Visualização da versão na ramificação master

Veja que o botão limpar não está no formulário. Ou seja não estamos mais naquela versão.

Agora podemos fazer uma alteração no projeto e fazer um commit na ramificação master e em seguida vamos visualizar o nosso grafo de commits. Então seguindo os comandos básicos será feito um novo commit com os comandos git add . – git commit –m “Mensagem”. Feito o commit visualizamos com o git log e temos o seguinte:

commit 2b10eb2a96fd61e9882b2109f8aa726397a8312c
Author: Bruno Oliveira bruni.nho_oliveira@hotmail.com
Date: Thu Aug 29 21:05:04 2013 -0300

Invertendo o lado dos botoes

commit 22e85b33686175c6be00ae43a44364b52d516db8
Author: Bruno Oliveira bruni.nho_oliveira@hotmail.com
Date: Wed Aug 28 09:00:58 2013 -0300

Criando layout inicial do formulario de cadastro

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

    Inicio do Projeto

Temos mais um commit, agora veja nosso grafo:

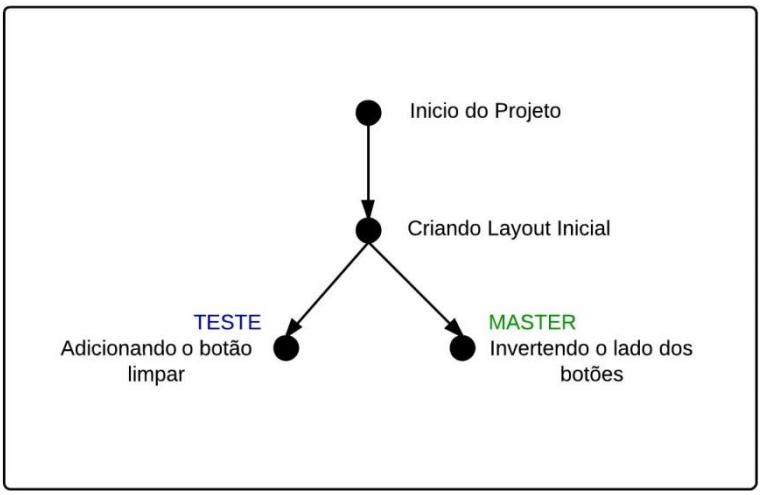

Figura 7 - Grafo com as ramificações master e teste

Agora o nosso grafo passou a ter dois caminhos (ramificações) e nosso projeto agora pode correr de duas formas diferentes e simultâneas e ao mesmo tempo em que altera uma ramificação, você pode mudar de ramificação e testar o outro projeto e modificar e voltar pra ramificação principal e modifica-la também, como se estivesse trabalhando com duas ou mais cópias do seu projeto, porém com economia de espaço e sem se preocupar onde cada versão está salva, pois com o git todas as versões (cópias), alterações estão na mesma pasta, basta usar o comando git checkout para alternar entre ramificações e entre versões.

Até agora alternamos entre ramificações, mas como dito no paragrafo anterior também podemos alternar entre as versões, ou seja, entre os commits que já fizemos, para alternar entre as versões podemos fazer de duas formas.

1º forma: Código de identificação do commit

Para alternar entre versões com o código do commit vamos primeiro visualizar os commits que temos com o git log.

commit **2b10eb2a96fd61e9882b2109f8aa726397a8312c**
Author: Bruno Oliveira bruni.nho_oliveira@hotmail.com
Date: Thu Aug 29 21:05:04 2013 -0300

Invertendo o lado dos botoes

commit **22e85b33686175c6be00ae43a44364b52d516db8**
Author: Bruno Oliveira bruni.nho_oliveira@hotmail.com
Date: Wed Aug 28 09:00:58 2013 -0300

Criando layout inicial do formulario de cadastro

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

Inicio do Projeto

Estamos vendo os commits, dessa vez estão grifados os códigos de identificação para que seja mais fácil o entendimento. Esses códigos gigantes são a identificação do commit feito. São os códigos de 160 bits. Para usa-los e mudar de uma versão para outra pegamos os seis primeiros dígitos da esquerda para a direita e colocamos no comando git checkout, exemplo.

Vou mudar da ultima versão que é a invertendo botões para a do início do projeto. Lembrando que estou na ramificação master.

BrunoOliveira@Notebook-HP ~/Documents/ProjetoTesteGit (master)
$ git checkout d16604 <enter>

Note: checking out 'd16604'.

You are in 'detached HEAD' state. You can look around, make experimental changes and commit them, and you can discard any commits you make in this state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may do so (now or later) by using -b with the checkout command again. Example:

git checkout -b new_branch_name

HEAD is now at d16604c... Inicio do Projeto

O git exibe que está alterando de versão, informando que sua posição passou a ser a versão que você escolheu, no nosso caso o inicio do projeto. Outro ponto importante que o git informa é o seguinte:

Que caso você queira fazer alguma alteração nesta versão na qual você acabou de voltar, que é recomendado que você crie uma nova ramificação para alterar esta versão, pois a alteração direta nesta posição atual, poderá resultar em perda das versões que você criou após esta.

Dica: só volte para as versões anteriores caso você tenha tido problemas com a última versão que criou, caso contrário não fique alternando entre as versões para evitar problemas com o projeto.

Para saber que você alterou de versão na representação onde ficava o rótulo master.

```
BrunoOliveira@Notebook-HP ~/Documents/ProjetoTesteGit ((d16604c...))
$ git checkout d16604                <enter>
```

No lugar de master é mostrado no lugar entre parênteses o os primeiros dígitos referentes a versão que você indicou para ir.

2º Forma: Tags das versões

Outra forma de alternar entre as versões é usando tags, para usar as tags, cada vez que você criar um commit, você deve criar uma tag para essa versão. As tags são como se fossem etiquetas das versões, para facilitar e você não precisar do código de identificação.

Para criar as tags nas versões você deve usar o comando git tag, sempre após fazer o commit.

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

O nome da versão não pode conter espaços e nem começar por números, para separar palavras use – ou _ e de maneira nenhuma use outros caracteres especiais como * & % $ #, pois muitos deles são usados como caracteres de comando.

Caso você queira atribuir uma tag para uma versão que não tem a tag basta voltar para a versão correspondente usando o comando git checkout, e o código de identificação e quando estiver na versão use o git tag.

Após ter as tags registradas, para alternar entre as versões basta usar o comando git checkout e o nome da tag.

```
BrunoOliveira@Notebook-HP ~/Documents/ProjetoTesteGit (master)
$ git checkout V-1.0                 <enter>
```

Feito isso a identificação da versão que você está ficará assim:

```
BrunoOliveira@Notebook-HP ~/Documents/ProjetoTesteGit ((V-1.0))
$
```

Onde ficava a indicação master e onde ficaria o código de identificação da versão, agora fica com a tag da versão o que torna a visualização mais fácil e até mesmo para alternar entre as versões.

As ramificações ajudam bastante e ter essa forma de alternar entre as versões também é de grande ajuda principalmente quando temos problemas com a atual versão, o git ajuda a nunca perdermos todo o trabalho feito, no caso de ramificações é importante que tenhamos algumas no projeto para podermos testar determinados pontos que não sabemos se funcionaram ou se são viáveis, agora na questão de mudar de versões deve-se ter cuidado, lembre-se da dica, só de mude para versões anteriores caso você esteja com problemas irreversíveis na versão atual.

Mesclando projetos com o GIT

Já abordamos quase todos os comandos essenciais do git, já sabemos os comando básicos e os comandos mais utilizados para auxiliar na execução de projetos, mais ainda faltam alguns comandos, agora veremos um dos comandos mais importantes do git, o comando git merge, o qual é utilizado para se mesclar projetos. Aproveitando o projeto exemplo vamos utilizar o comando nele.

Como fazemos isso? Bom, para podermos mesclar projetos temos que ter ao menos duas ramificações no projeto, a master que é a principal e mais uma ramificação criada pelo usuário no caso você. Vamos relembrar como estão nossas ramificações com o comando git log.

Na ramificação master temos os seguintes commits e o projeto está da seguinte forma:

commit 2b10eb2a96fd61e9882b2109f8aa726397a8312c
Author: Bruno Oliveira bruni.nho_oliveira@hotmail.com
Date: Thu Aug 29 21:05:04 2013 -0300

Invertendo o lado dos botoes

commit 22e85b33686175c6be00ae43a44364b52d516db8
Author: Bruno Oliveira bruni.nho_oliveira@hotmail.com
Date: Wed Aug 28 09:00:58 2013 -0300

Criando layout inicial do formulario de cadastro

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

Inicio do Projeto

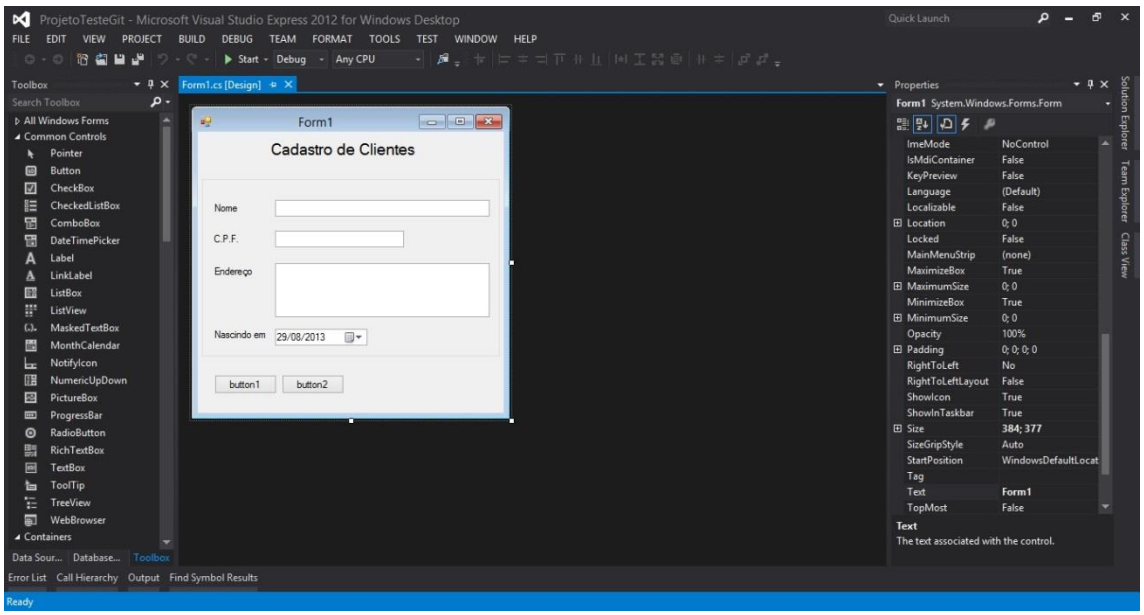

Figura 8 - Estado atual da ramificação master

E na ramificação teste temos o seguinte:

commit 2ac0292e23d65dea888f13d56b47fc1f6cc89bd0
Author: Bruno Oliveira bruni.nho_oliveira@hotmail.com
Date: Thu Aug 29 20:23:30 2013 -0300

Adicionado o botao limpar

commit 22e85b33686175c6be00ae43a44364b52d516db8
Author: Bruno Oliveira bruni.nho_oliveira@hotmail.com
Date: Wed Aug 28 09:00:58 2013 -0300

Criando layout inicial do formulario de cadastro

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

Inicio do Projeto

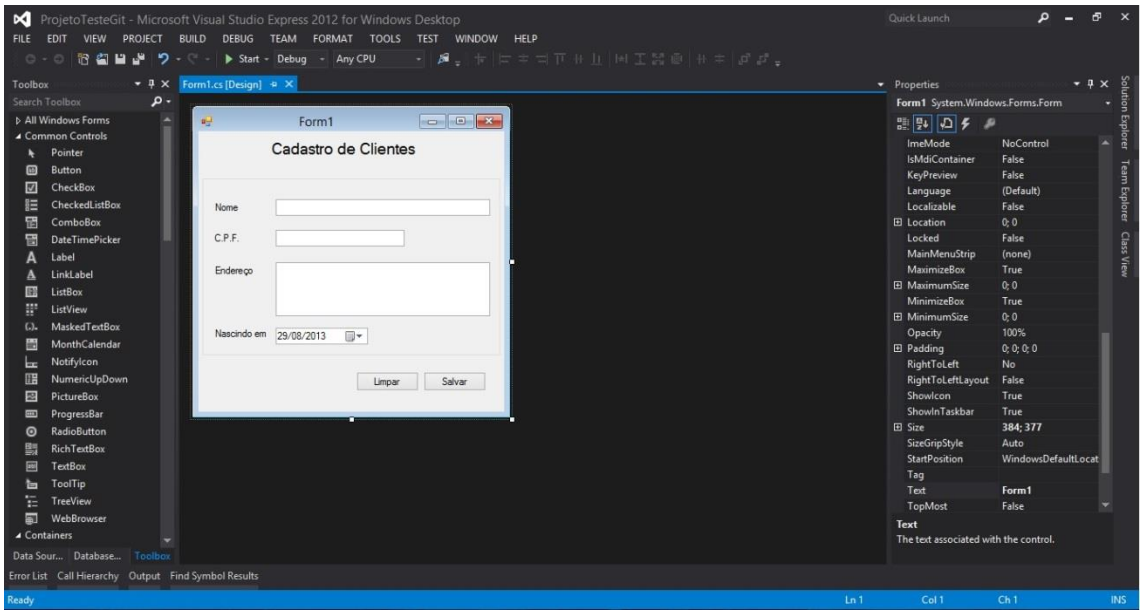

Figura 9 - Estado atual na ramificação teste

Podemos perceber que temos algumas pequenas diferenças, no caso a posição dos botões, vamos aproveitar essas pequenas diferenças e vamos mesclar os projetos.

Para mesclar devemos estar na ramificação que vai receber as alterações da outra ramificação, por exemplo, para mesclarmos a ramificação teste para a master temos obrigatoriamente que esta na ramificação master.

Olhe na indicação ao lado do endereço da pasta do projeto para saber em qual ramificação você está ou use o comando git branch e veja qual está com o asterisco e em verde.

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

No caso estou na ramificação master e irei mesclar com a ramificação teste. Detalhe para juntar as ramificações não deve ter nenhuma alteração no projeto, caso haja alterações em qualquer uma das ramificações, antes deverá ser feito os commits para depois mescla-las.

Quando for mesclar um projeto, essa operação pode te fornecer três mensagens diferentes.

1º Everything up to date

É quando ambas as ramificações continham as mesmas alterações, ou seja, eram iguais. Dificilmente aparecerá.

2º Fast-Forward

Mesclagem direta, o próprio git organiza os arquivos e mescla todas as alterações sem problemas. Esse caso acontece quando as alterações não são no mesmo arquivo, por exemplo quando estamos trabalhando com formulários diferentes e ramificações diferentes, ou como vamos ver mais a frente, quando estamos trabalhando em equipe e cada um está trabalhando em um formulário do Visual Studio.

3º Conflicts (ATENÇÃO NESTE PONTO)

São os conflitos, que exigem a você que os resolva, ou seja, que arrume o código ou documento com as alterações que são de fato as mais importantes e mais atualizadas. Acontecem constantemente em projetos desenvolvidos em grupo.

As duas primeiras formas, que acontecem com o comando git merge não entraremos em detalhes, pois acontecem de forma automática, o que é bem diferente dos conflitos, que precisam de maior atenção do desenvolvedor.

Para mesclar usamos o seguinte comando:

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

Então como estamos na ramificação master, temos que usar git merge teste, se estivéssemos na teste, usaríamos git merge master. Essa definição serve para qualquer outra ramificação.

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

warning: Cannot merge binary files:
ProjetoTesteGit/obj/Debug/ProjetoTesteGit.pdb (HEAD vs. teste)

warning: Cannot merge binary files:
ProjetoTesteGit/obj/Debug/ProjetoTesteGit.exe (HEAD vs. teste)

warning: Cannot merge binary files:
ProjetoTesteGit/obj/Debug/ProjetoTesteGit.csproj.GenerateResource.Cache
(HEAD vs. teste)

warning: Cannot merge binary files:
ProjetoTesteGit/bin/Debug/ProjetoTesteGit.pdb (HEAD vs. teste)

warning: Cannot merge binary files:
ProjetoTesteGit/bin/Debug/ProjetoTesteGit.exe (HEAD vs. teste)
```

warning: Cannot merge binary files: ProjetoTesteGit.v11.suo (HEAD vs. teste)

Auto-merging ProjetoTesteGit/obj/Debug/ProjetoTesteGit.pdb
CONFLICT (content): Merge conflict in
ProjetoTesteGit/obj/Debug/ProjetoTesteGit.pdb

Auto-merging ProjetoTesteGit/obj/Debug/ProjetoTesteGit.exe
CONFLICT (content): Merge conflict in
ProjetoTesteGit/obj/Debug/ProjetoTesteGit.exe

Auto-merging
ProjetoTesteGit/obj/Debug/ProjetoTesteGit.csproj.GenerateResource.Cache
CONFLICT (content): Merge conflict in
ProjetoTesteGit/obj/Debug/ProjetoTesteGit.csproj.GenerateResource.Cache

Auto-merging ProjetoTesteGit/bin/Debug/ProjetoTesteGit.pdb
CONFLICT (content): Merge conflict in
ProjetoTesteGit/bin/Debug/ProjetoTesteGit.pdb

Auto-merging ProjetoTesteGit/bin/Debug/ProjetoTesteGit.exe
CONFLICT (content): Merge conflict in
ProjetoTesteGit/bin/Debug/ProjetoTesteGit.exe

Auto-merging ProjetoTesteGit/Form1.Designer.cs
CONFLICT (content): Merge conflict in ProjetoTesteGit/Form1.Designer.cs

Auto-merging ProjetoTesteGit.v11.suo
CONFLICT (content): Merge conflict in ProjetoTesteGit.v11.suo

Automatic merge failed; fix conflicts and then commit the result.

Vários conflitos foram detectados, isso geralmente vai acontecer e muito, mas não se assuste os conflitos são indicações de que há mudanças que precisam ser analisadas pelo programador, o git faz indicações no código de forma a ficar mais fácil encontrar onde estão os problemas.

As indicações são feitas assim:

<<<<<< HEAD

Versão que você possui ou que estão na ramificação que você está.

======

Versão da ramificação que você quer mesclar, ou parte do projeto que estava no servidor.

>>>>>> Nome_Ramificação

Vamos ver no nosso código como ficou:

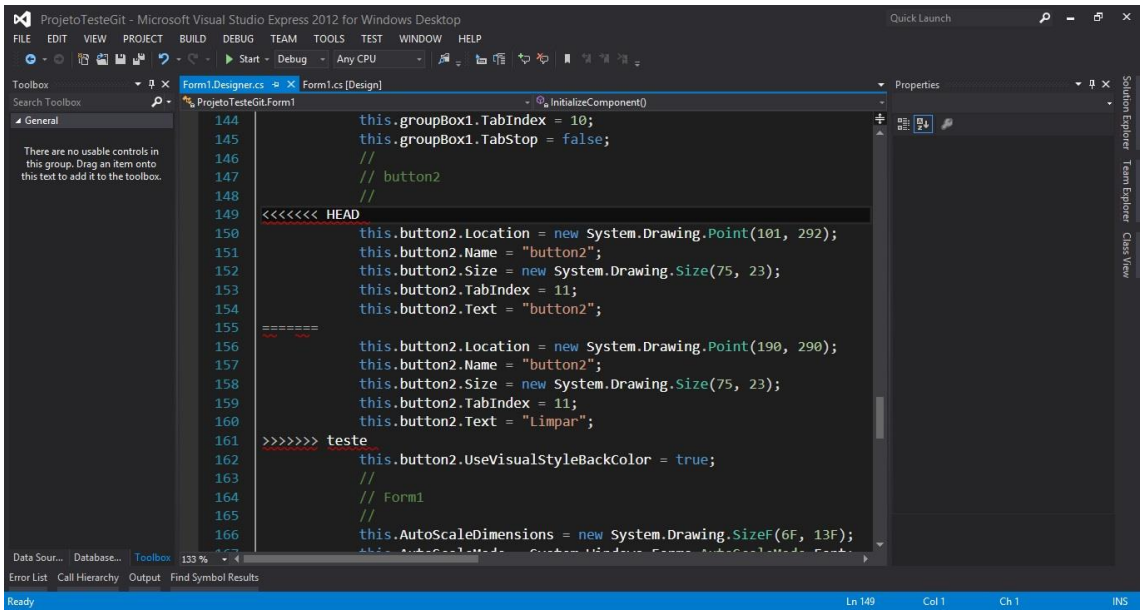

Figura 10 - Indicação de conflitos que o git coloca no código

Percebam que o git informa em dois blocos, o primeiro como foi dito são as alterações da sua versão atual ou da ramificação que você está neste momento (entre o HEAD e os símbolos de igualdade), seguindo o raciocínio, o segundo bloco indica as alterações que você quer mesclar, no caso o que esta na outra ramificação (entre os símbolos de igualdade e o nome da ramificação), você deve verificar neste momento qual dos dois blocos de fato é a versão que você vai deixar na ultima versão.

DICA: antes de sair excluindo tudo, no caso de códigos comente todos os blocos HEAD que foram detectados no programa, teste as funções e em seguida retire os comentários do bloco HEAD e insira no bloco da Ramificação, e teste novamente para ver qual função você quer deixar. Exemplo.

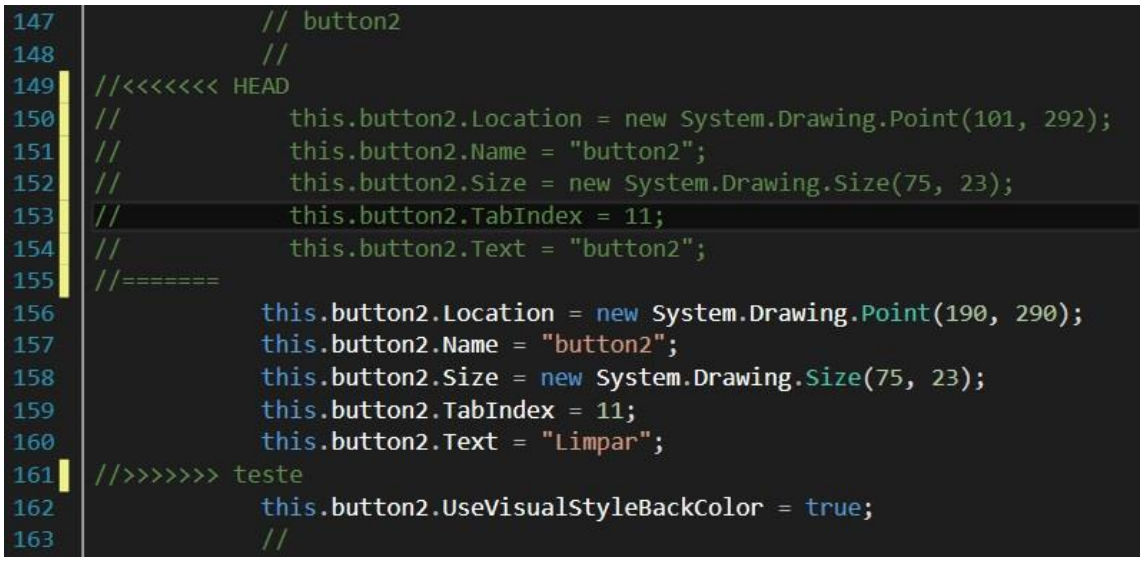

Figura 11 - Comente o trecho HEAD e execute o programa

Obs. Faça isso e todas as indicações HEAD que estiverem no código, em todos os arquivos que estiver, pois no caso do Visual Studio isso pode acontecer em vários arquivos diferentes.

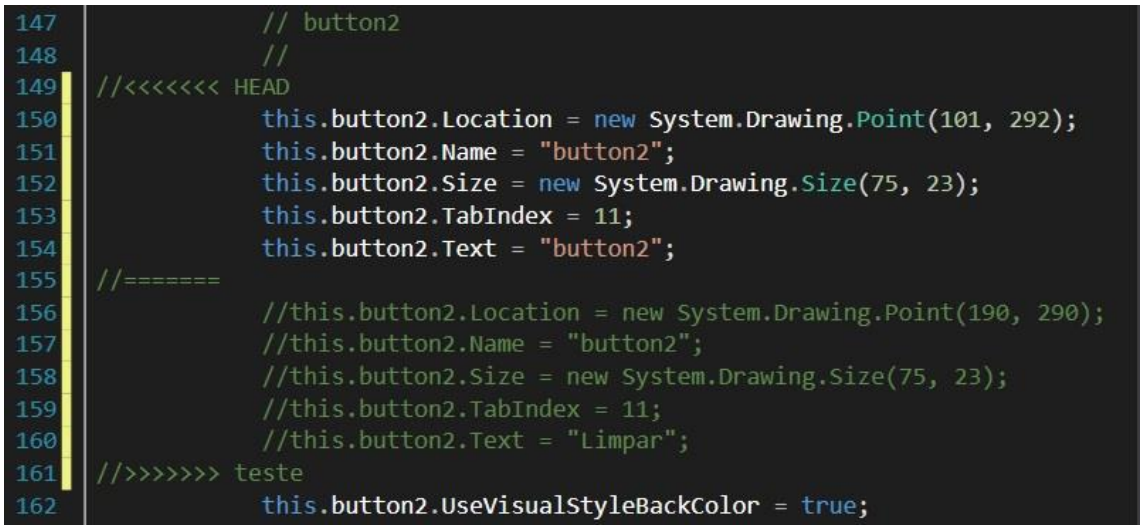

Figura 12 - Tire os comentário do bloco e HEAD e comente o outro bloco e execute o programa

Feito isso você conseguirá testar cada bloco e assim saberá exatamente o que deve ficar e o que deve sair, é uma dica fica a seu critério, se souber o que tem que sair fique a vontade para retirar.

E atenção às vezes o git indica um bloco de códigos e na segunda parte fica em branco, às vezes pode ser apenas uma chave que ou mesmo um espaço m branco que tinha em um arquivo e não tinha no outro, então analise bem as indicações para ver se o que você for apagar não vai afetar o seu código.

Ainda não acabou, você arrumou todo o código? Tirou os problemas? Seu programa está rodando? Se todas as respostas forem verdadeiras, agora você deve voltar ao git e concluir o merge, ou seja, consolidar a mescla das ramificações.

```
BrunoOliv@Notebook-HP ~/Documents/ProjetoTesteGit (master | MERGING)
$
```

Quando você voltar ao git, verá ao lado da indicação master, a indicação MERGING, isso quer dizer que você está no meio de uma junção que ainda não foi concluída, depois de ter arrumado o código, executado e salvo as alterações, você deve terminar o merge, fazendo o processo de um commit. Isso mesmo que você acabou de ler, você deve fazer um commit para concluir o merge, seguindo os passos já aprendidos.

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

BrunoOliv@Notebook-HP ~/Documents/ProjetoTesteGit (master | MERGING)
$ git commit –m “Resolvendo Conflitos – Posicao dos botoes”    <enter>
```

Quando fizer isso se tudo estiver certo, a indicação MERGING saíra e você de fato mesclou e criou a nova versão atualizada.

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

Com isso vimos todos os comandos que podemos utilizar em um repositório local, já temos uma lista grande de comandos, o que permite trabalhar com o git, crie arquivos e repositórios no seu computador com esses comandos e pratique todas as possibilidade e principalmente as junções (Merges), para que você adquira um melhor entendimento, pois o git facilita e muito desde que você saiba usar a ferramenta e para que isso seja possível à prática deve ser feita.

Trabalhando com o GIT e o GITHUB (Repositório em Nuvem)

Já vimos como trabalhar na forma local com o git, aprendemos como fazer os commits, como trabalhar com ramificações, como criar as tags, como alternar entre as versões e ramificações e como fazer as junções, 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.

Feito o cadastro, execute os procedimentos de criação de perfil, e o principal de tudo, confirme sua conta de e-mail para poder usar todos os recursos do GITHUB.

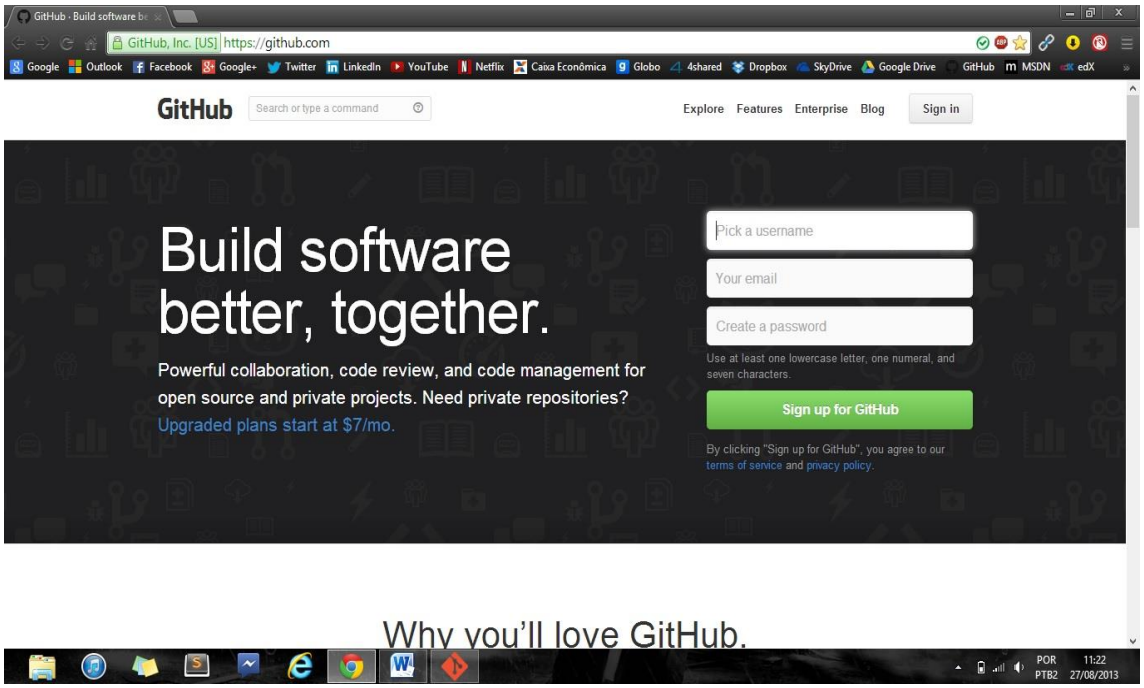

Figura 13 - Tela inicial do GITHUB - Faça o cadastro caso não tenha.