

Apostila do curso de GIT PHPConference

Introdução

Nesse curso veremos os principais pontos da utilização do **GIT** em nosso ambiente de trabalho.

Além de saber como utilizar o **GIT** aprenderemos sobre como encaixar o **GIT** no **GitHub** com utilização da metodologia **Agile** com **Scrum**, teste unitário com **PHP Unit** e no final será discutido sobre técnicas de **Deploy** com **Travis** e finalmente subindo nossa aplicação para o **Heroku**.



2 GIT

Como mencionado existem outras soluções que também podem ser utilizadas para controlar a versão do seu código, mas lembre-se, em nossa jornada o **(GIT)** será o seu maior parceiro !

Mas gostaria de, pelo menos, mencionar o nome dessas outras soluções que, sim, podem também aparecer no seu dia a dia e você tende-se a quem sabe utilizar essas soluções que são:

- CVS
- Mercurial
- SVN

2.1 Instalando o GIT no Linux

A instalação do GIT no Linux é bem simples basta que você instale ele a partir do instalador binário, para quem utiliza o gerenciamento de pacotes yum que é o caso da Fedora deverá executar o seguinte comando:

```
$ yum update  
$ yum install git-core
```

Agora caso você esteja utilizando as distribuições baseadas no Debian, como é o caso do Ubuntu você executará o seguinte comando:

```
$ apt-get update  
$ apt-get install git
```

Para verificar se a instalação ocorreu com sucesso você pode executar o comando de verificação de versão em seu terminal da seguinte maneira:

```
$ git --version
```

2.2 Instalando o GIT no Windows

Atenção caso você já tenha o GIT instalado talvez esse capítulo não seja tão interessante para você, mas caso ainda não tenha será de grande ajuda.

A instalação do GIT no Windows é bem simples basta que você entre no link:
<https://git-scm.com/download/win>

- <https://git-scm.com/download/win>

E então escolher o instalador do GIT para a arquitetura do seu sistema que pode ser de 32-bits ou 64-bits cliquem na opção que atende a arquitetura do seu sistema.

Quando terminar o Download você deverá executar e será aberta a opção como é ilustrado na Figura 2.1 você deverá apertar em next para prosseguir com a instalação.

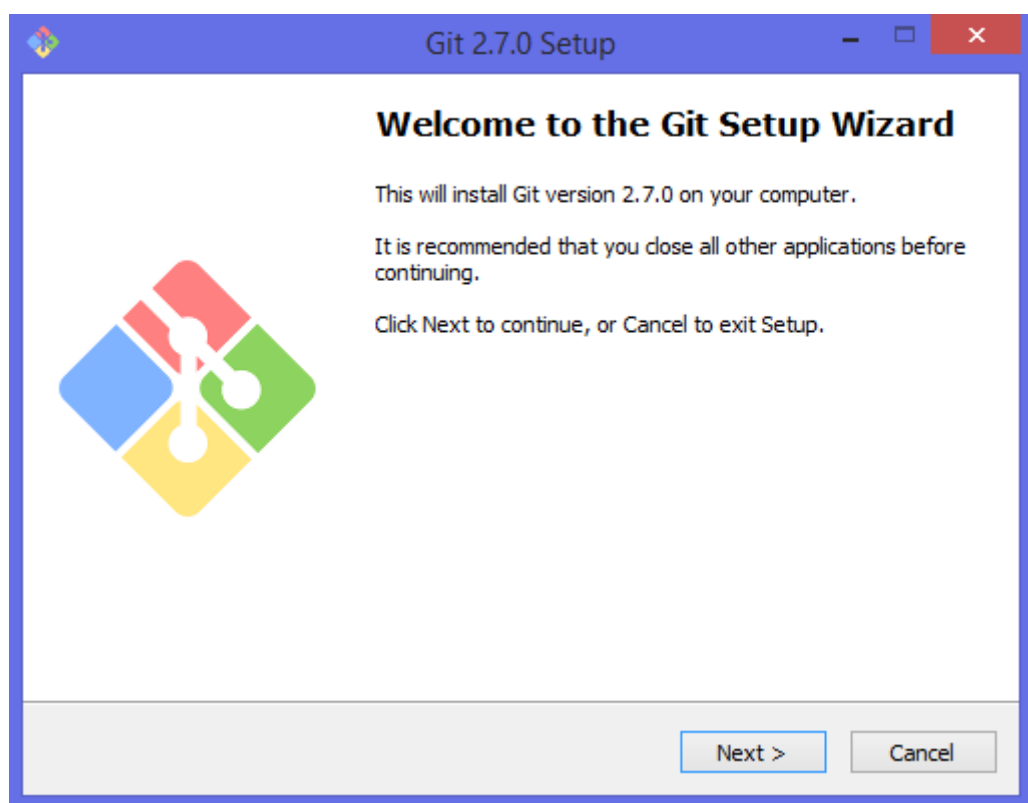


Figura 2.1 tela inicial da instalação do GIT no Windows

Agora que você já prosseguiu será exibido para você uma tela com a licença do GIT como é ilustrado na Figura 2.2, e eu sei que, muitos não irão ler essa

tela **mesmo sendo importante**, agora você deve prosseguir apertando next novamente.



Figura 2.2 tela contendo as informações de licença GNU utilizada no GIT

A próxima tela que é ilustrado na Figura 2.3 conterá os componentes que você pode selecionar em sua instalação você pode escolher entre uma opção ou mais de uma para não fica selecionado uma a uma repare que ao clicar na opção pai como ícones adicionais as filhas serão selecionadas também mas caso queira apenas uma opção filha basta clicar aquela que mais favorece ao seu gosto.

Você não é obrigado a selecionar nenhum, mas atente-se ao fato de que não selecionar, pode não trazer alguns benefícios que esses itens trazem consigo pois cada item pode conter uma funcionalidade específica em seu sistema o que fará com que facilite a utilização do GIT.

Cada item será explicado e cabe a você decidir se você concorda em deixar selecionado ou não porém a caso você opte por não selecionar você pode optar por reinstalar o GIT Bash e selecionar as opções que trarão mais benefícios e facilidades no seu dia a dia.

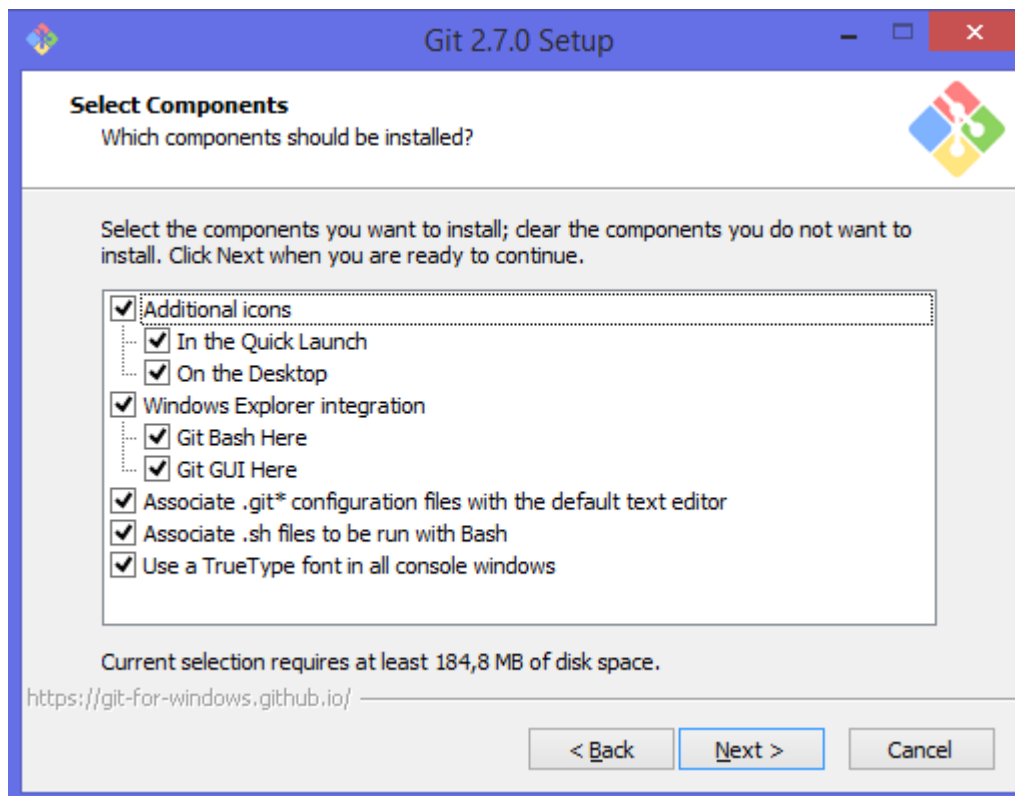


Figura 2.3 tela contendo a escolha de componentes para o seu sistema

Explicação dos itens de componentes:

- **Ícones adicionais** – *Os ícones adicionais servem para quando você iniciar o seu sistema já seja lançado o GIT Bash. - Ou para a criação de um ícone de inicialização do GIT Bash.*
 - **Quando iniciar o sistema** – Opção para inicializar o GIT junto ao sistema.
 - **Deixar na área de trabalho** – Opção que criará um ícone em sua área de trabalho
- **Integração com o Windows Explorer** – Integração com Windows Explorer para o GIT Bash e para o GIT GUI
 - **GIT Bash** – Irá integrar o GIT Bash ao Windows Explorer
 - **GIT GUI** – Irá integrar o GIT GUI ao Windows Explorer
- **Associar os arquivos com a extensão (.git*) ao editor padrão do sistema** – Essa opção habilita que os arquivos da extensão (.git) abra no editor que estiver configurado como padrão de abertura de arquivos em seu sistema, ou seja, ao clicar em um arquivo que contenha a extensão mencionada ele abre, por exemplo, no Sublime se for o editor padrão de arquivo do seu sistema.

- **Associar os arquivos com a extensão (.sh) para execução no Bash** – Essa opção habilita que ao abrir um arquivo (.sh) ele seja executado pelo Bash que você está instalando nesse momento.

Agora que você já sabe o que são as opções cabe a você escolher o que é necessário para o seu caso e não cabe ao autor dizer a você para selecionar todos ou apenas um pois cada um sabe o que é melhor para o seu caso. Em nosso exemplo foi selecionado todos pois caso seja necessário explicar algum item eu já o terei instalado e configurado em meu sistema.

Agora que você prossegui, a próxima tela que será aberta pergunta a você sobre como você gostaria de usar o GIT na linha de comando será perguntando a você sobre 3 (três) opções para escolha de execução de comandos como é ilustrado na Figura 2.4.

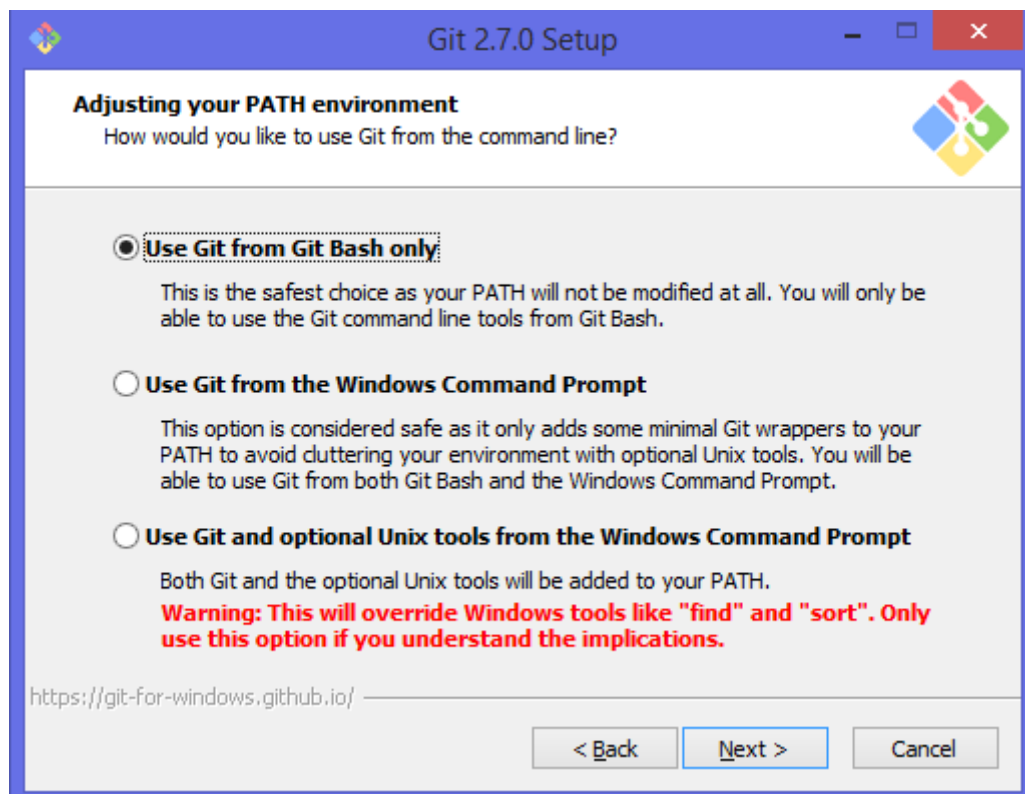


Figura 2.4 tela contendo a escolha de linha de comando

Explicação dos itens de linha de comando:

- **Usar apenas o Git Bash** – Essa opção permite que você use apenas os comandos Unix no Git Bash, ou seja, se você tentar executar os comandos Unix no Prompt Command do Windows você não conseguirá.

- **Executar o GIT a partir do Prompt Command do Windows** – Essa opção permite que você execute o GIT no Prompt Command do Windows, ou seja, você poderá utilizar o comando `Git clone` no Prompt sem nenhuma dor de cabeça. - Porém você não conseguirá executar os comandos Unix com essa opção o que pode ser ruim pois os exemplos que você encontra na internet geralmente o autor utiliza comandos Unix em seus exemplos e você pode acabar não tendo a mesma experiência que alguém que consiga executar terá.
- **Executar o GIT e incluir os comandos Unix no Prompt Command do Windows** – Essa opção permitirá você a executar os comandos Unix pelo Prompt Command do Windows, porém alguns comandos do Windows serão substituídos como você pode ver na Figura 2.5 existe um texto em vermelho informando sobre a troca de comandos no caso diz sobre os comandos `find` e `sort` que são do Windows mas tirando essas particularidades você terá um grande poder em seu Prompt Command pois agora pode executar comandos Unix sem problemas. - Em nosso exemplo será escolhido a terceira opção pois caso você precise executar os comandos que serão passados no Livro você não terá problemas para acompanhar.

A próxima opção como é ilustrado na Figura 2.5 perguntará a você qual cliente Shell você deseja utilizar para o seu GIT talvez alguns já tenham tido a experiência de ter uma instância Amazon por exemplo existe uma opção que é acessar o servidor via SSH isso no GNU/Linux basta que você execute no terminal o comando:

`ssh usuario@enderecodoservidor` – Caso você execute o comando no Linux e retorne `command not found`. - Você terá que instalar o ssh dessa forma:

- **`apt-get update` e `apt-get install ssh`**

Para você que está no Windows será instalado, por exemplo, o Open SSH caso você selecione essa opção ou você pode selecionar o Plink para, por exemplo, acessar o servidor de homologação onde encontra-se o seu código ou até mesmo conexões com o Github .

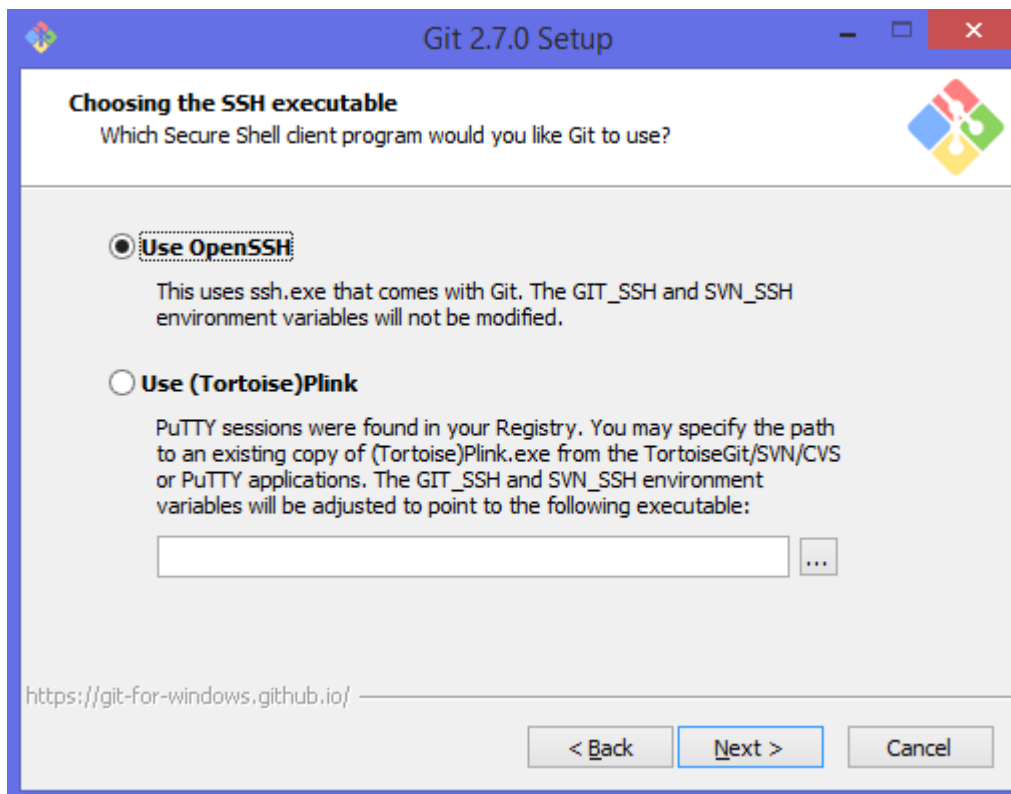


Figura 2.5 tela contendo a escolha do cliente de SSH

Agora que você já escolheu o seu cliente SSH você deverá escolher a quebra de linha que seu sistema utilizará pois como talvez você já saiba o Windows e os sistemas (Linux e Mac) utilizam a quebra de linha de forma diferenciada.

No caso de você que está utilizando o Windows e escreve seu código com quebras de linhas com o padrão Windows pode sim gerar problemas para outro usuário que utilizar esse Script, por exemplo, no Linux mas calma com a escolha correta como é exemplificado na Figura 2.6 permite que você normalize esse problema.

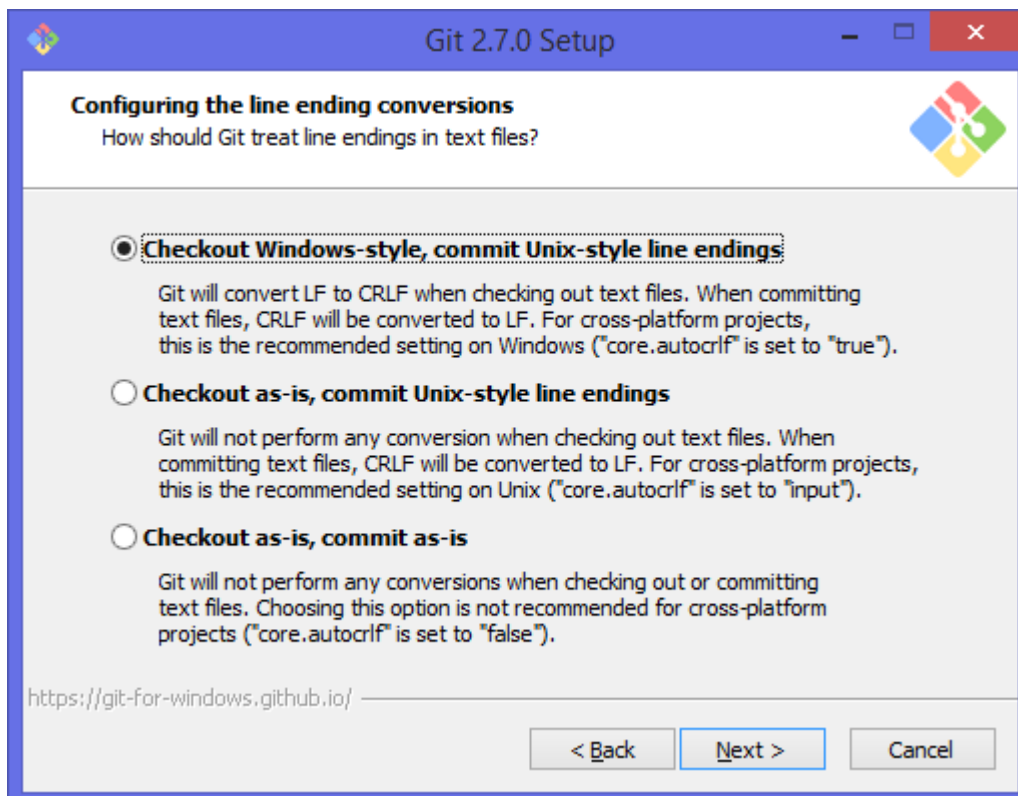


Figura 2.6 tela contendo a escolha da quebra de linhas.

Agora que você continuou com a instalação será pergunta a você sobre qual emulador de terminal você gostaria de trabalhar pois sabemos que o padrão atual em sua máquina é o prompt command do Windows a escolha é ilustrada na Figura 2.7.

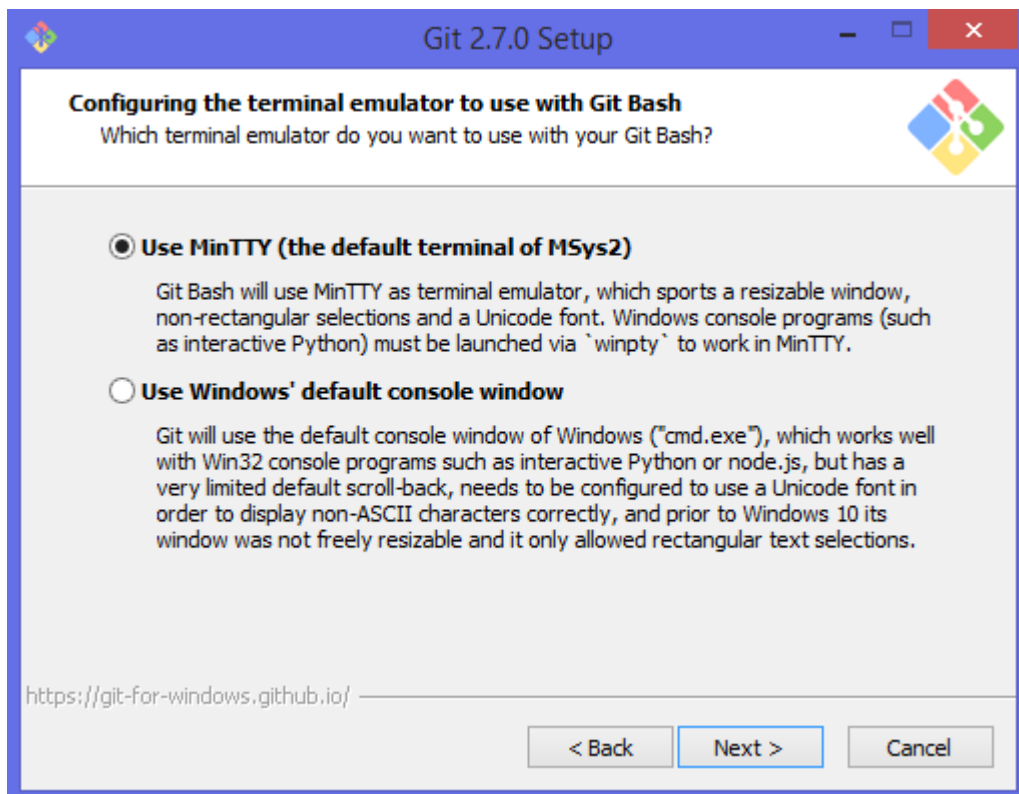


Figura 2.7 tela contendo a escolha emulador de terminal

Explicação dos itens de escolha do emulador de terminal:

- **Usar Mintty como emulador de terminal** – Essa opção permite você a instalar o programa que emula terminal conhecido como Mintty e você utilizará sua interface para executar seus comandos.
- **Usar o console padrão do Windows** – Essa opção usa como padrão o CMD (Command Prompt Windows) como programa para executar os seus comandos.

Em nosso exemplo foi escolhido a primeira opção mas caso você gostei do CMD da Microsoft não terá nenhum problema para executar os comandos junto a ele.

Agora que você já decidiu qual emulador de terminal você utilizará, você será perguntado se deseja ativar uma configuração experimental de performance que é o Tweaks e a opção que ele trará para ser marcada é a de ativar o cache de arquivos como é ilustrado na Figura 2.8 em nosso exemplo essa opção ficará ativa.

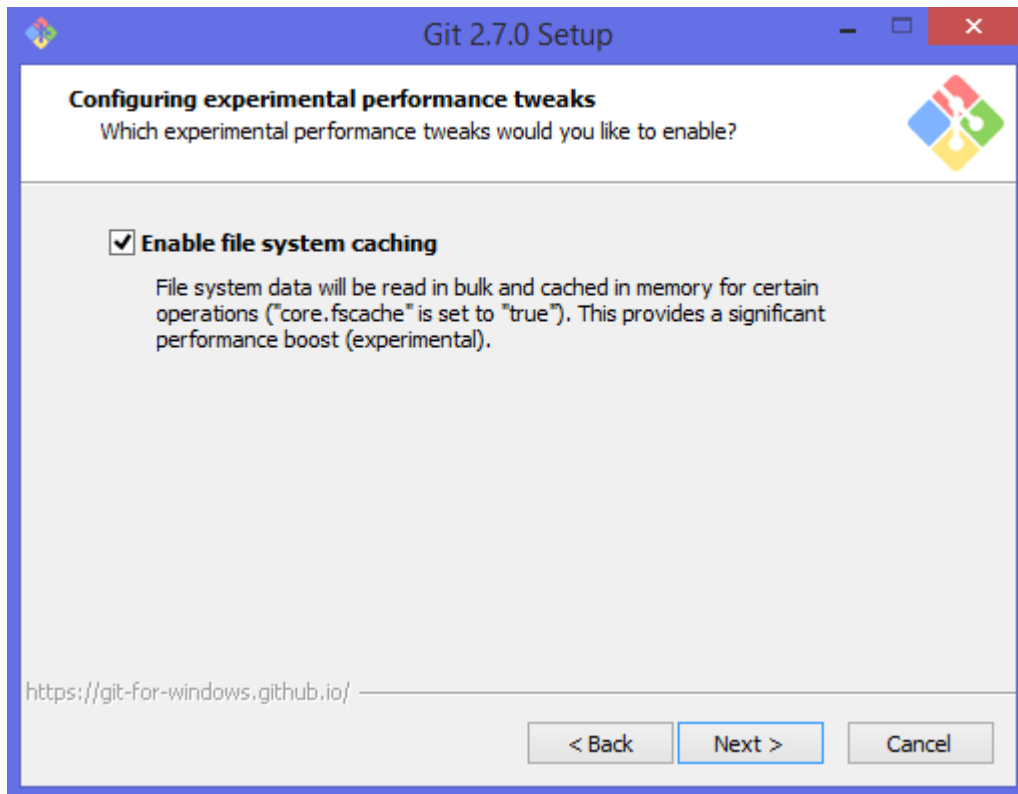


Figura 2.8 tela contendo a escolha para ativar o cache de arquivos

E finalmente após tantas telas e “Next” você verá que o GIT começa a ser instalado em seu sistema sei que alguns podem ter apertado o Next sem ler as opções e caso você queira ajustar algo que leu posteriormente não se preocupe pois ao chamar o executável novamente você poderá instalar o GIT novamente pois quando chegar nessa tela que é ilustrada na Figura 2.8 ele abrirá uma janela nova informando que está desinstalando o GIT e instalará novamente e a partir desse momento você pode trabalhar com o GIT e executar todos os comandos que serão explicados no livro sem grandes problemas mesmo executando eles no ambiente Windows.

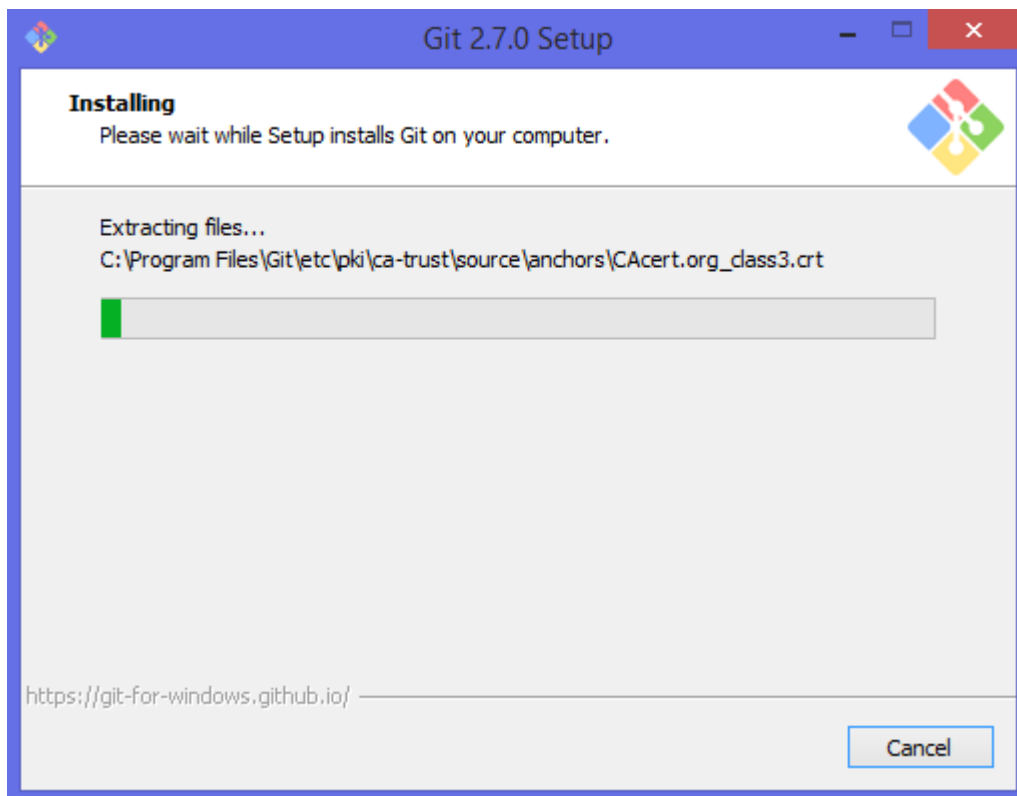


Figura 2.8 que ilustra a etapa final de instalação do GIT no Windows

Quando terminar essa etapa será exibido uma tela a você contendo a informação que o GIT foi instalado como sucesso caso você não queira ler as notas de lançamento basta que você desmarque a opção e aperte em concluir como é ilustrado na Figura 2.9.

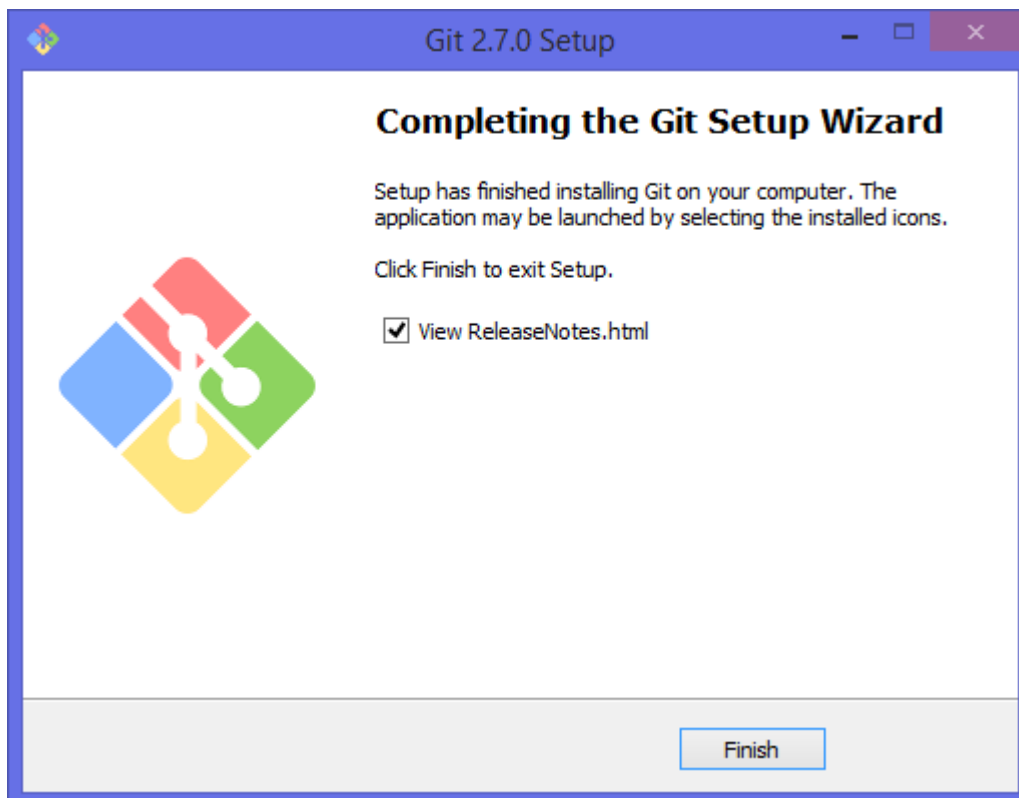


Figura 2.9 que ilustra a etapa de conclusão de instalação do GIT

Feita a instalação você pode agora chamar o GIT a partir do ícone dele, dependendo de como foi instalado em sua máquina você verá que ao ser chamado o GIT abre a tela que é ilustrada na Figura 2.10 se for apresentado essa tela a partir de agora você poderá executar os comandos que serão apresentados mais a frente em nossa jornada do curso.

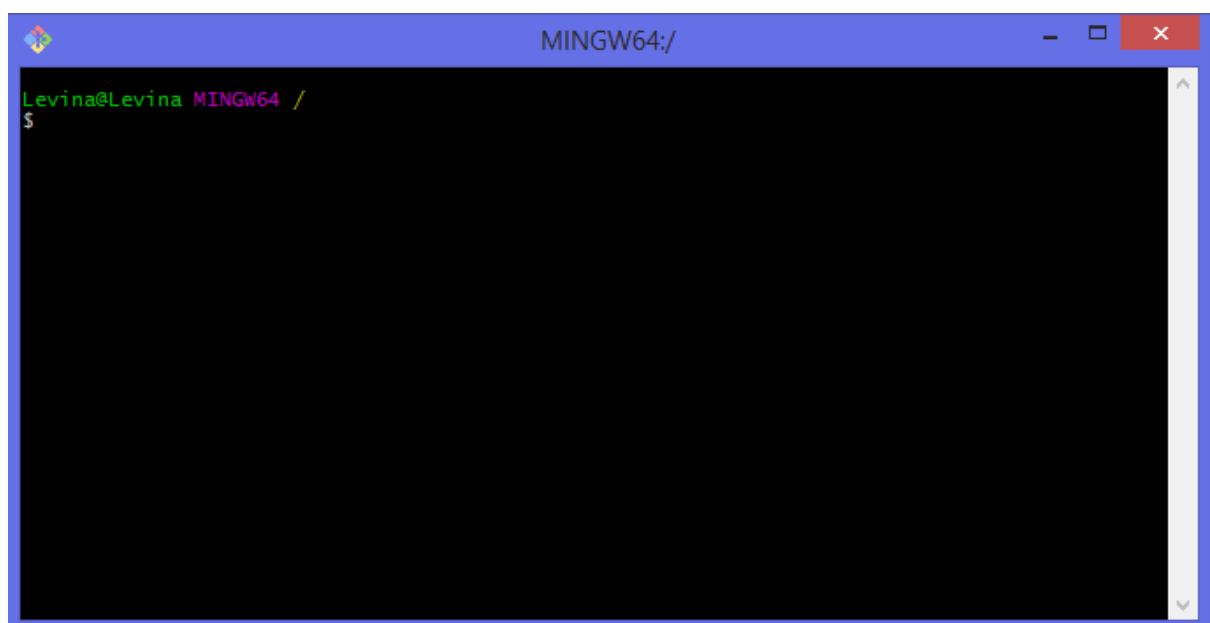


Figura 2.10 ilustra que o GIT está instalado com sucesso.

2.3 GIT GUI

Git Gui é uma interface gráfica que é baseada no Tcl\Tk para o GIT, ou seja, para quem está acostumado a utilizar uma interface gráfica para trabalhar o GIT GUI é uma mão na roda.

Mas lembre-se de que estamos sob o ambiente o Windows então a interface que será mostrada na Figura 2.15 refere-se ao ambiente do GIT GUI para Windows.

Para você que trabalha com Linux e Mac pode utilizar algumas das seguintes interfaces gráficas:

- **Git-Cola** – Uma interface desenvolvida em Python muito fácil de utilizar e instalar.
 - Pode ser instalado no Linux da seguinte forma:
 - \$ sudo apt-get install git-cola
 - Em Mac pode ser instalado da seguinte forma:
 - Entre no link: <http://www.collab.net/downloads/giteye> e selecione Mac Osx e então realize o Download.
- **Gitg** – Assim como Git-Cola temos o Gitg que é uma interface de fácil utilização e de instalação também.
 - Poder ser instalado no Linux da seguinte forma:
 - \$ sudo apt-get install gitg
- **Git Gui** – Também existe suporte para o Linux e funciona com quase toda a experiência de tela do Windows.
 - Pode ser instalado no Linux da seguinte forma:
 - \$ sudo apt-get install git-gui
- **GitUp** – Assim como apresentado nas outras interfaces gráfica GitUp facilita a utilização do GIT por meio de sua interface amigável e de fácil utilização porém só tem suporte para Mac.
 - Pode ser instalado da seguinte forma:
 - Entre no link: <http://gitup.co> e realize o Download do ZIP do GitUp.

Caso queira ver mais interfaces gráficas você pode escolher a partir do link: <https://git-scm.com/download/gui/linux> e ver qual das opções agrada mais e que seja mais útil no seu dia a dia.

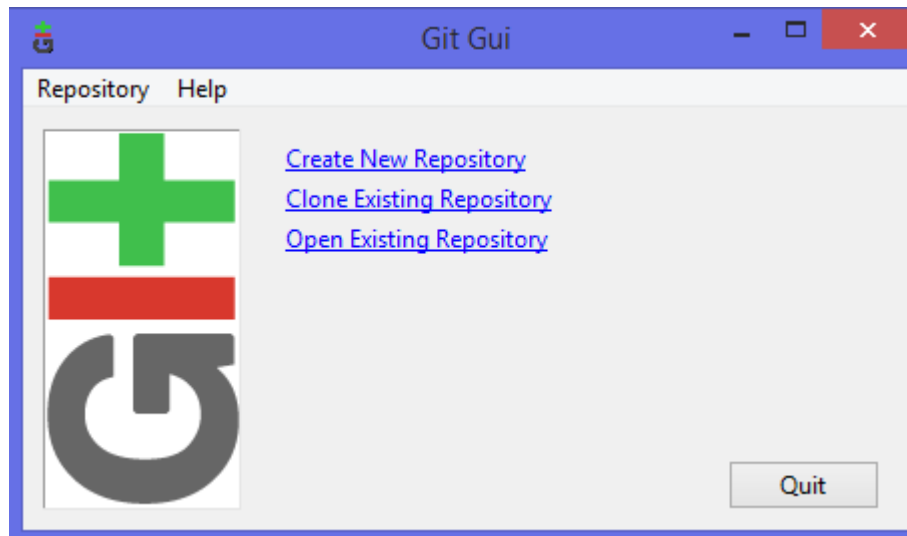


Figura 2.15 ilustra a tela inicial do GIT GUI

Como você pode reparar você possui três escolhas para trabalhar com o GIT a partir do GIT GUI que são:

- **Criar novo repositório** – Essa opção permite que você inicie um repositório GIT novo.
- **Clonar um repositório existente** – Essa opção permite que você clone um repositório GIT já existente.
- **Abrir repositório existente** – Essa opção permite que você abra um repositório GIT já existente.

Iremos então criar um novo repositório para o nosso projeto do curso após escolher a opção de criar um novo repositório será aberto a tela de seleção caminho do projeto como é ilustrado na Figura 2.16.

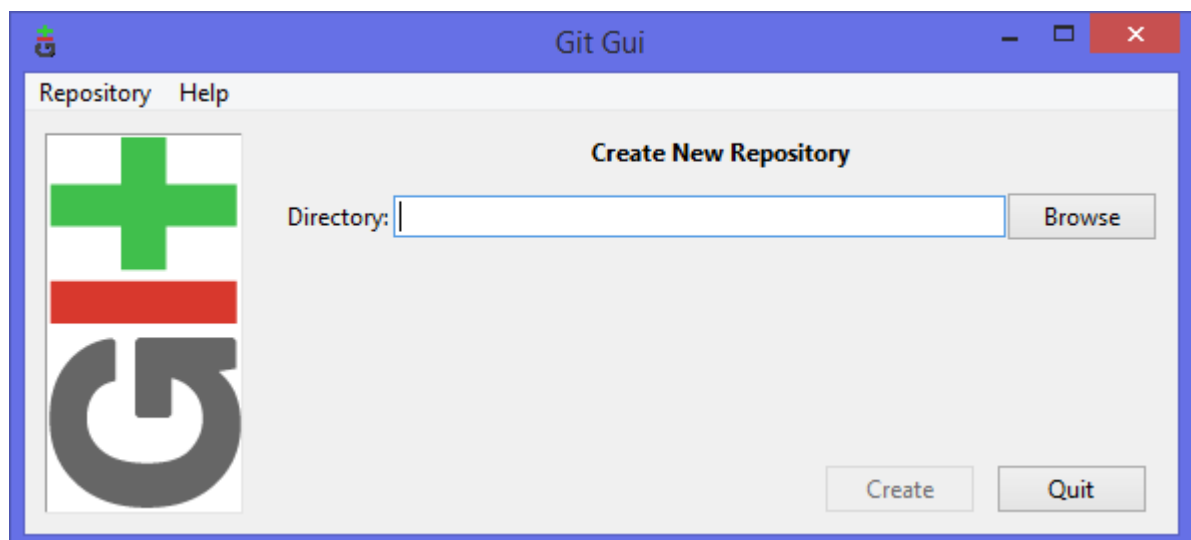


Figura 2.16 ilustra a tela para criação do novo repositório

Para você que instalou o **Wamp** nos colocaremos o arquivo do livro no caminho: **C:/wamp/www/livro** como é ilustrado na Figura 2.17

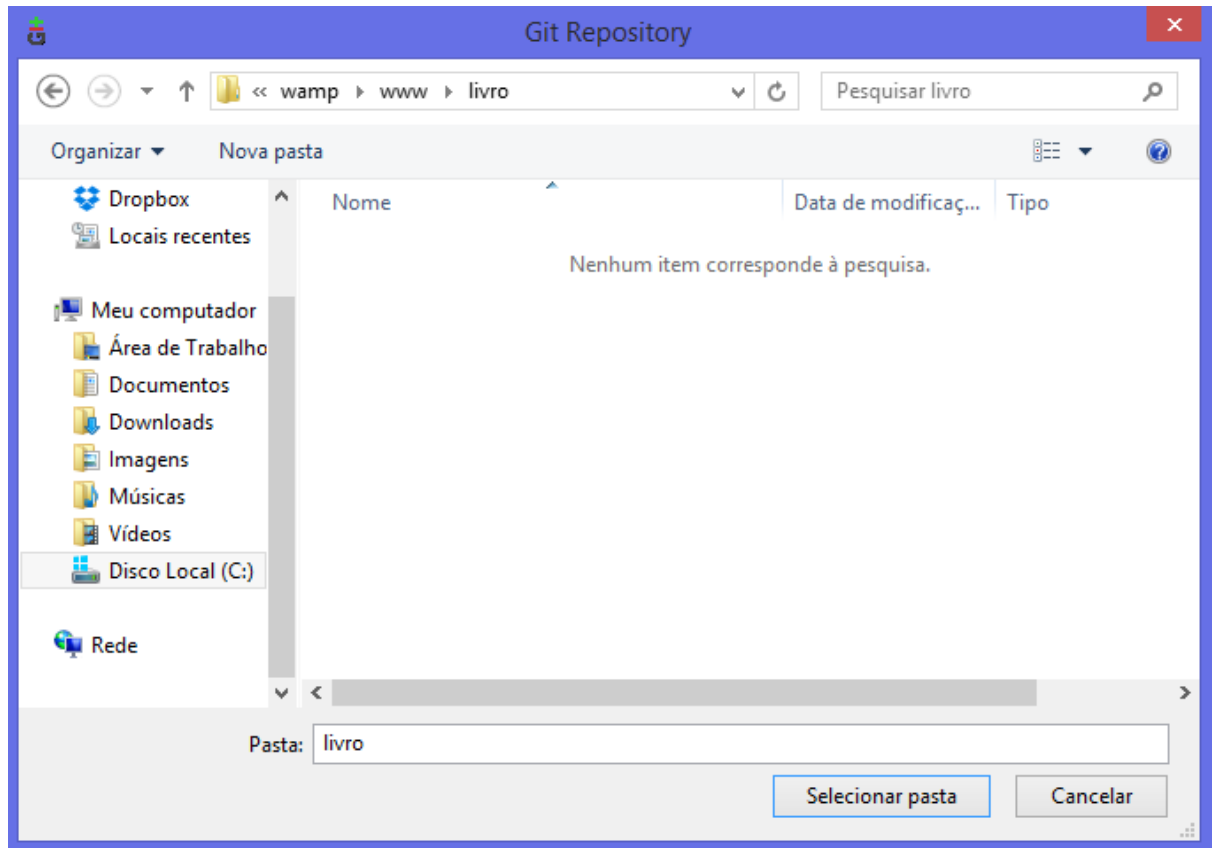


Figura 2.17 ilustra a tela para criação do novo repositório

Agora que você já selecionou o destino do projeto será exibido a tela com a escolha de destino do projeto como é ilustrado na Figura 2.18 basta que você cliquem em criar (Create).

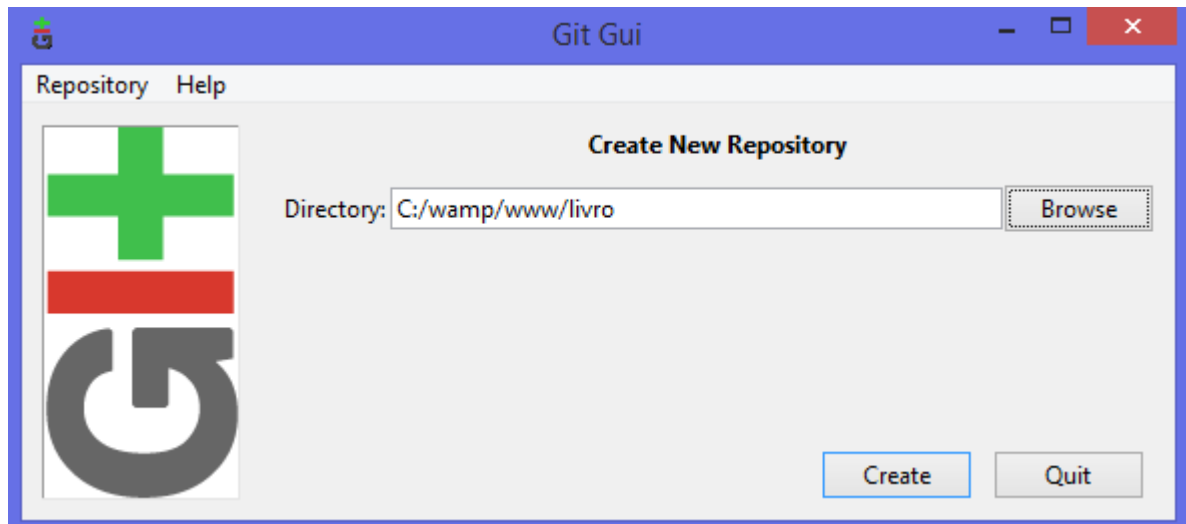


Figura 2.18 ilustra a tela de escolha do destino do projeto

Agora que você prosseguiu com a escolha de destino do seu novo repositório será exibida a tela da Figura 2.19 que é o seu local de trabalho GIT, nesse repositório foi criado um arquivo **index.php** sem conteúdo só para ilustrar a exibição de arquivos com o Git Gui.

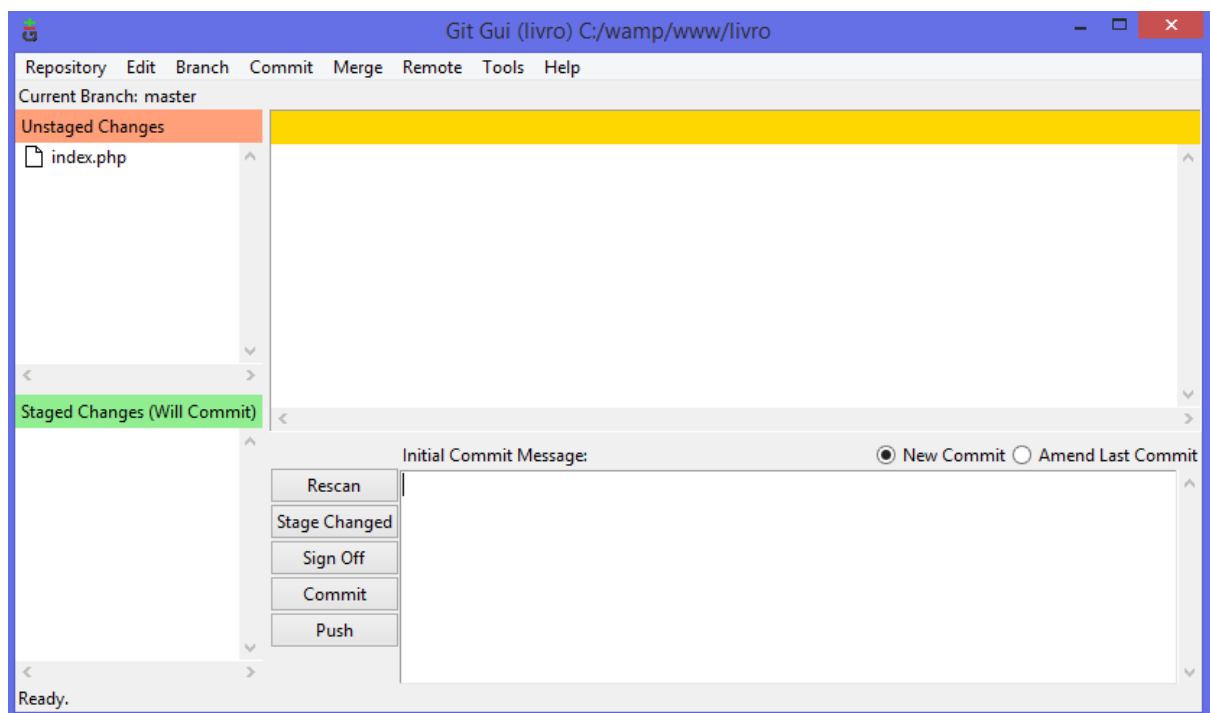


Figura 2.19 ilustra a tela de trabalho com GIT GUI

Como você pode perceber a partir de agora você tem 2 (duas) escolhas ou trabalhar no terminal com o GIT ou a partir de agora trabalhar utilizando o GIT GUI que você terá os mesmos comandos do terminal só que de forma

gráfica podendo até ser mais produtivo por não ter que ficar digitando os comandos GIT no terminal e mudar isso por um botão que fará a mesma ação.

2.4 Instalando o GIT no Mac

A instalação do GIT é bem simples no Mac assim como é Windows podemos utilizar a interface gráfica para instalar essa ferramenta que é a Git OSX pode ser baixada no link:

<http://sourceforge.net/projects/git-osx-installer>

Ao baixar e executar o arquivo será aberta a tela de instalação do GIT para o Mac como é ilustrado na Figura 2.20.

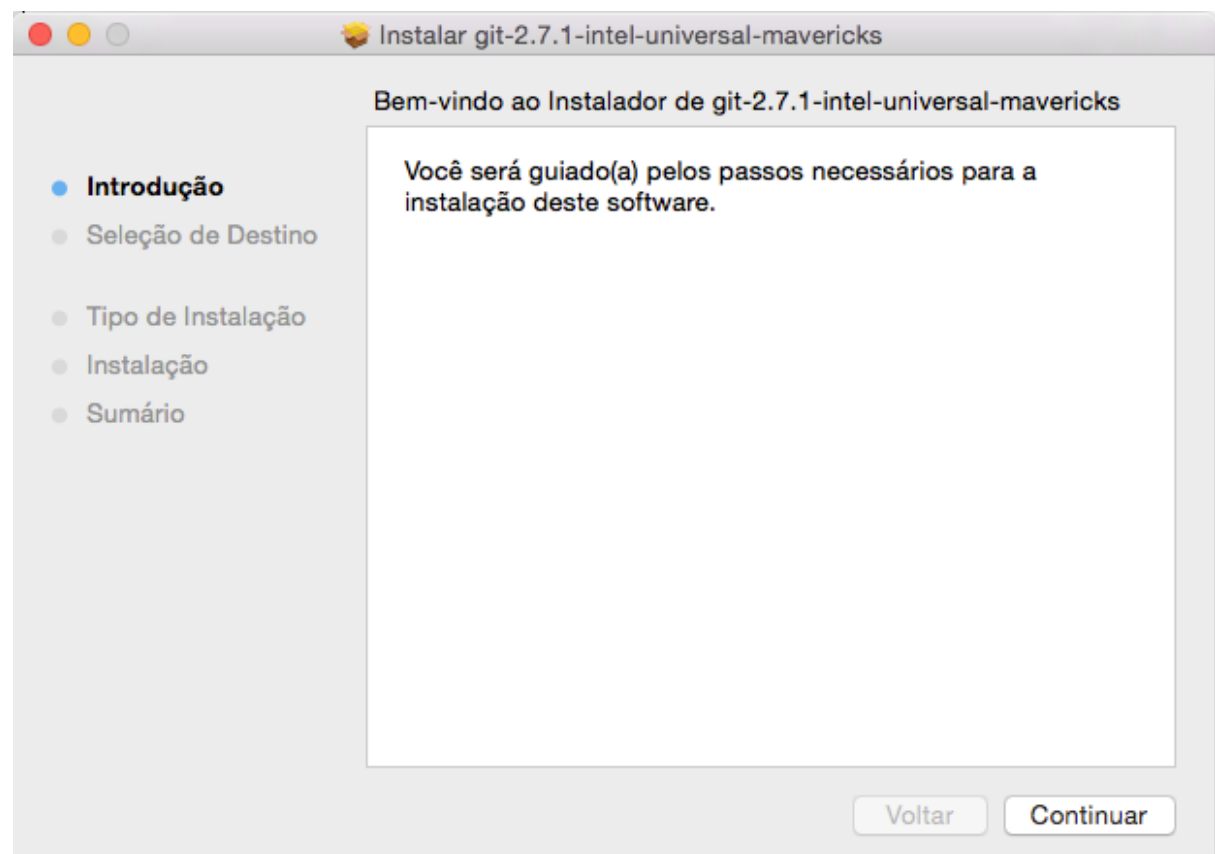


Figura 2.20 que ilustra a etapa inicial de instalação do GIT no Mac.

Na próxima tela será perguntado a você sobre qual localização a instalação deve utilizar no caso a padrão é utilizar no disco “MACINTOSH” como é ilustrado na Figura 2.21.

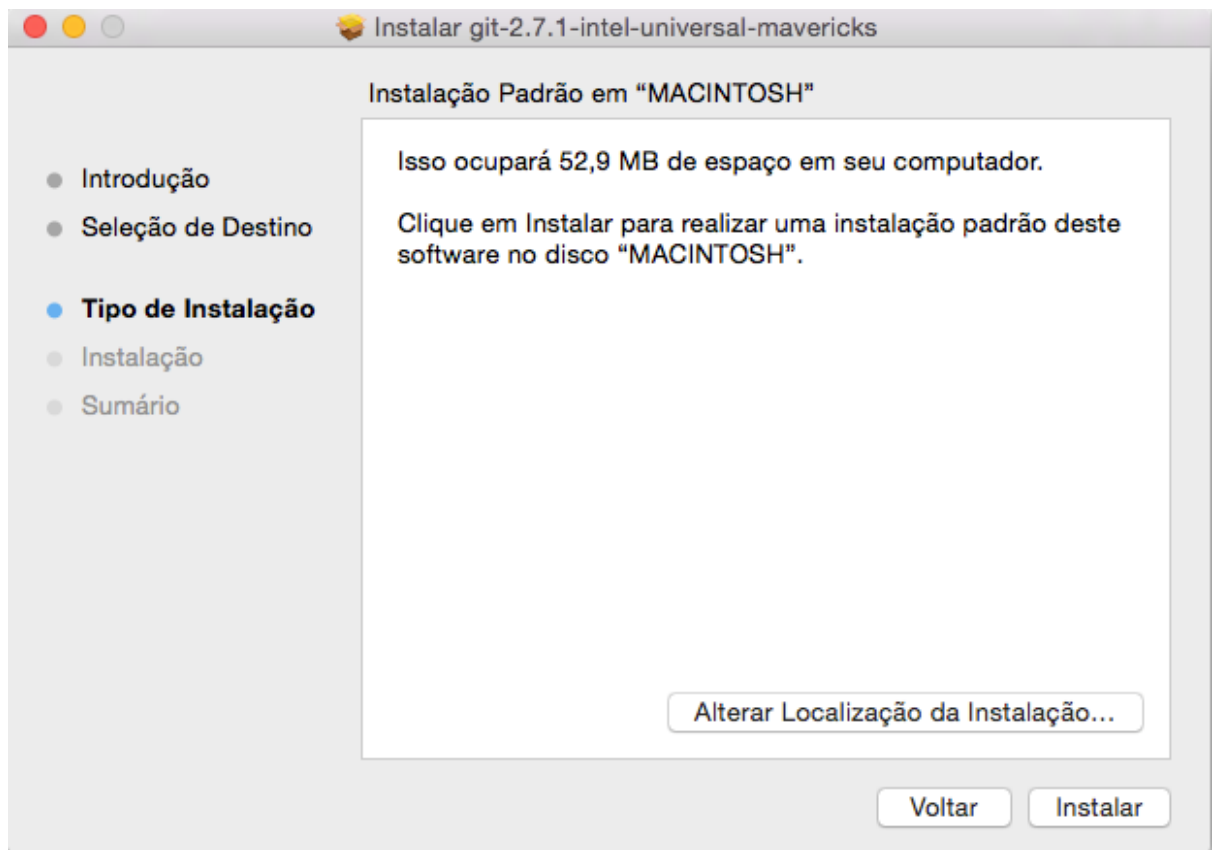


Figura 2.21 que ilustra a etapa de escolha de disco no Mac.

Na próxima tela você verá que é exibido a você o progresso de instalação como é exibido na Figura 2.22.

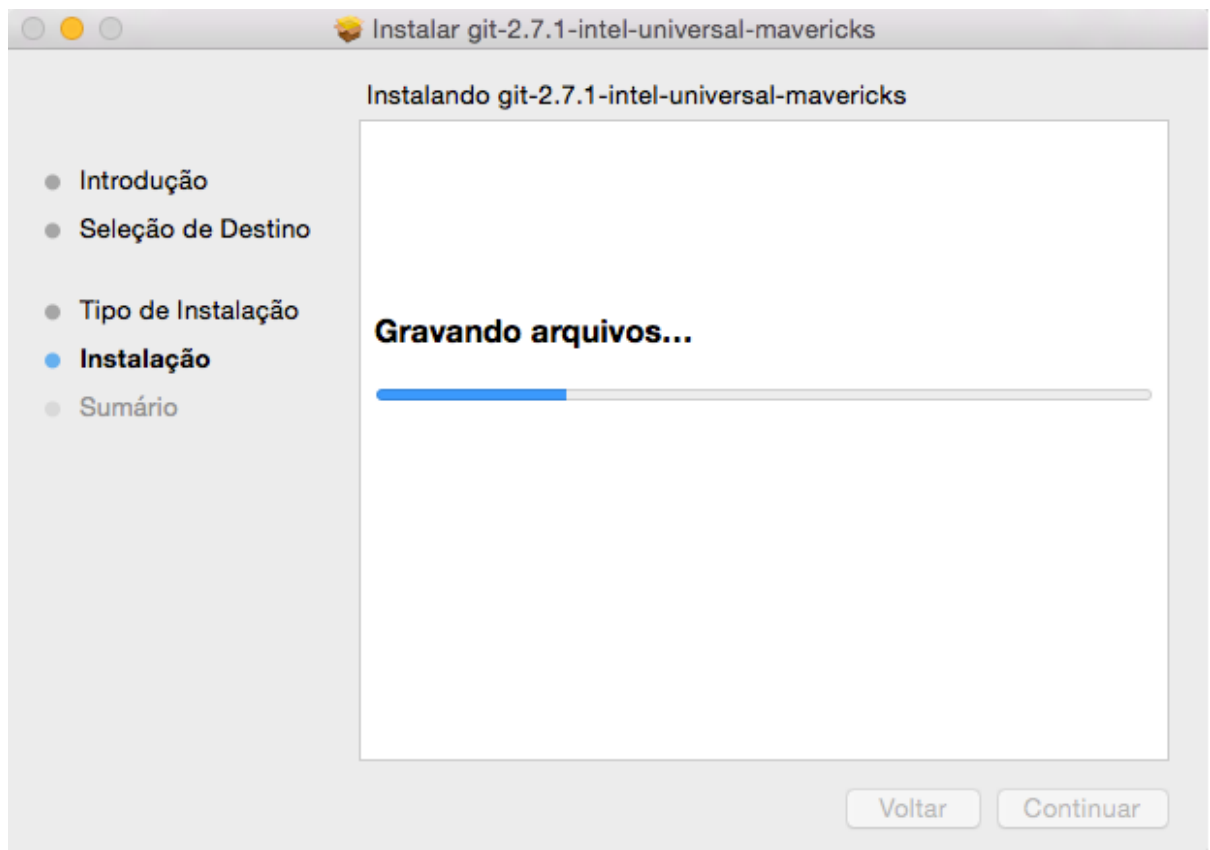


Figura 2.22 que ilustra a etapa de escolha de disco no Mac.

Agora se tudo ocorreu sem nenhum problema você verá a última tela que confirma a instalação do GIT em seu Mac como é ilustrado na Figura 2.23.



Figura 2.23 ilustra a finalização da instalação do GIT.

Existe uma outra maneira de instalar o GIT no Mac que é via MacPorts (<http://www.macports.org>) execute o comando em seu terminal:

```
$ sudo port install git-core +gitweb
```

Para confirmar a instalação você pode executar no terminal o comando para verificação da versão do GIT:

```
$ git --version
```

2.5 Criando um novo repositório

Em nossos exemplos utilizaremos o sistema operacional **GNU/Linux** mas não se preocupe pois para quem utiliza Windows por exemplo não terá problemas pois os comandos do GIT funcionarão em seu sistema operacional graças ao GIT Bash.

Para você criar um repositório você deverá criar uma pasta, você pode realizar os seguintes comandos para criar a pasta:

```
$ cd /var/www
$ mkdir cursogit
$ cd cursogit
```

Feito isso agora você pode executar seu primeiro comando para iniciar sua vida com o GIT:

```
$ git init
```

O comando Git init é responsável por criar um repositório, ou seja, ele criará um repositório (.git) que conterá um subdiretório para os objetos:

- Objects
- Refs/heads
- Refs/tags

Ou seja, a partir desse momento você já iniciou sua vida com o Git e com controles de versões baseados no padrão Git, lembrando que você executa um comando desse na raiz do seu projeto e o Git já se encarrega de criar a referência em todos os arquivos e então adiciona, editado ou excluído o Git avisa para você como veremos mais a frente.

2.5 Obtendo um repositório

Para obter um repositório o que você precisa aprender e como clonar ele de sua origem que pode ser tanto um repositório local, ou seja, se quisermos criar uma cópia do projeto cursogit que acabamos de criar você deverá executar em seu terminal o seguinte comando:

```
$ git clone /caminho/cursogit/ cursogit2
```

Repare que foi necessário inserir um espaço e o nome do clone caso você tente clonar sem colocar o nome e estiver na mesma pasta será gerado o erro:

fatal: destination path 'cursogit' already exists and is not an empty directory.

Esse erro foi gerado por que ao tentar realizar o clone em um local que já existe o projeto não será possível pois gera erro de duplicidade de nome de pasta, pois já existe uma pasta com o nome: cursogit.

Seguindo essa mesma ideia você talvez já conheça o Git Hub famoso por conter um perfil do programador e também seus repositórios de código que

podem estar abertos para suporte da comunidade ao código como também podem estar privados para utilização apenas do dono do projeto.

Usaremos o Git Hub como exemplo para clonar um projeto que encontra-se em outro servidor, mas veja, não necessariamente existe apenas o Git Hub como solução para armazenamento de códigos outras soluções também podem ser vistas e até instaladas em seu ambiente e alguns deles são:

- <https://about.gitlab.com>
- <https://bitbucket.org/>

Manteremos o foco no Git Hub por ser mais famoso e também por levar em consideração que você já o tenha utilizado.

Assim como clonar o projeto local podemos também clonar projetos terceiros com o mesmo comando porém a única diferença é que você utiliza a URL externa do projeto, por exemplo:

```
$ git clone https://github.com/michaeldouglas/laravel-pagseguro.git
```

Repare que no termino da execução do comando você terá em sua máquina os códigos do projeto Laravel PagSeguro, sendo assim, você pode navegar no Git Hub buscar por bibliotecas de outros desenvolvedores e clonar para a sua máquina sem grandes novidades.

2.6 Fluxo de trabalho com GIT

Agora que você já aprendeu sobre como clonar um projeto tanto local quanto em um link externo você deve aprender sobre o fluxo de trabalho que você acaba criando ao utilizar o GIT.

Basicamente esse fluxo se resume que em seus repositórios locais você terá três árvores que, quem toma conta nesse caso é o GIT.

A primeira é a **Working Directory** que contém os arquivos em que você esteja trabalhando nesse momento, ou seja, depois que você iniciar seu trabalho na pasta e for alterando os arquivos esse diretório que fica responsável por manter tudo o que está acontecendo.

Existe também a árvore **Index** que é responsável por armazenar uma área temporária, ou seja, basicamente contém o estado do seu próximo **commit**, uma vez que o **commit** que falaremos mais a frente é uma referência a um

objeto da árvore, que contém todas as informações para gerar o objeto da árvore ao realizar o comando **git commit**.

E por último temos o **HEAD** que aponta para o último commit que você fez. Talvez você já o tenha visto pois muitos tutoriais na internet você acaba encontrado algum tipo de referência ao comando **GIT** com o **HEAD** em suas composições, por exemplo:

```
$ git reset HEAD  
$ git reset --hard HEAD
```

2.6 Adicionando arquivos

Agora que você já aprendeu bastante sobre o GIT pode estar se perguntando como você mantém a versão com o GIT, ou seja, como manter a versão dos arquivos após editar, criar ou excluir arquivos ?

No diretório: **cursoGit** você irá criar um arquivo index.php que não irá conter nenhum conteúdo por enquanto para criar o arquivo você deve executar os comandos:

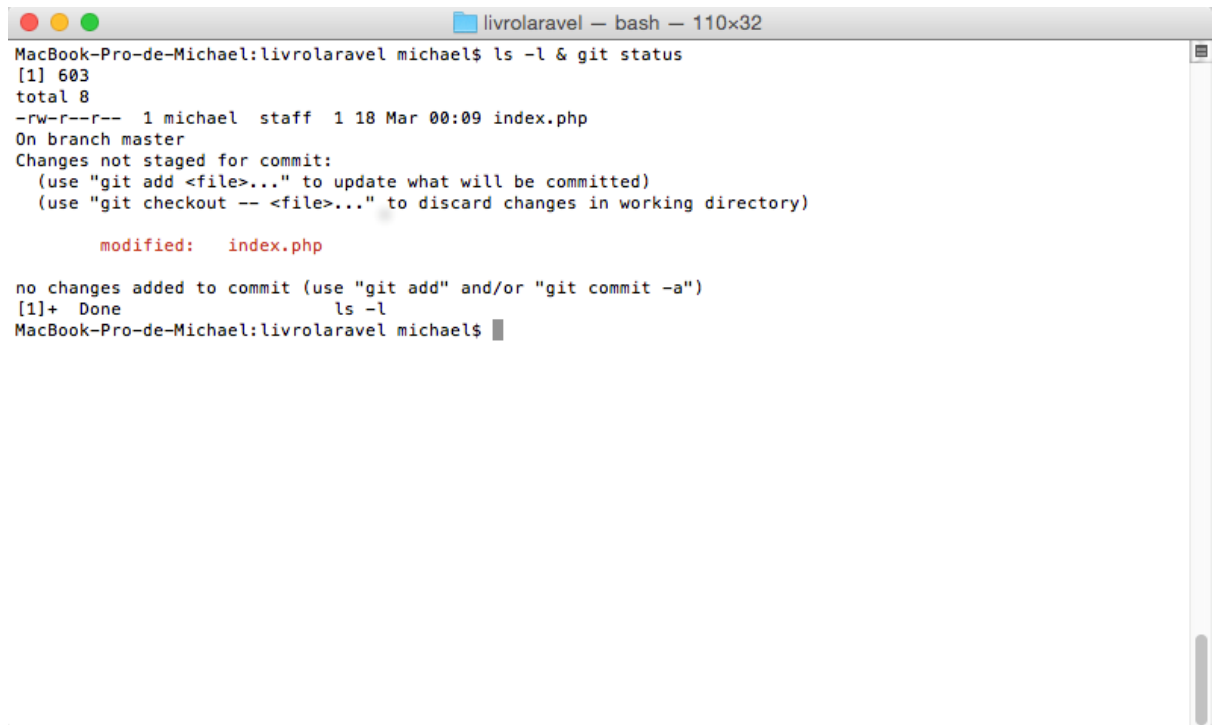
```
$ echo "" >> index.php
```

– Caso você queira alterar e inserir algo no arquivo index.php deve ser feito entre as aspas, por exemplo, “Curso GIT”

Agora que você efetuou a criação do arquivo você pode verificar se o arquivo está criado e também seu status GIT com os comandos:

```
$ ls -l & git status
```

Um exemplo de saída do comando pode ser visto na Figura 2.6.

A screenshot of a macOS terminal window titled "livrolaravel - bash - 110x32". The terminal shows the execution of the command "ls -l & git status". The output includes file permissions, ownership, size, date, and filename for "index.php". It also shows the current branch as "master" and indicates that changes are not staged for commit. A red "modified:" status is shown for "index.php". The prompt "MacBook-Pro-de-Michael:livrolaravel michael\$" is visible at the bottom.

```
MacBook-Pro-de-Michael:livrolaravel michael$ ls -l & git status
[1] 603
total 8
-rw-r--r--  1 michael  staff   1 18 Mar 00:09 index.php
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   index.php

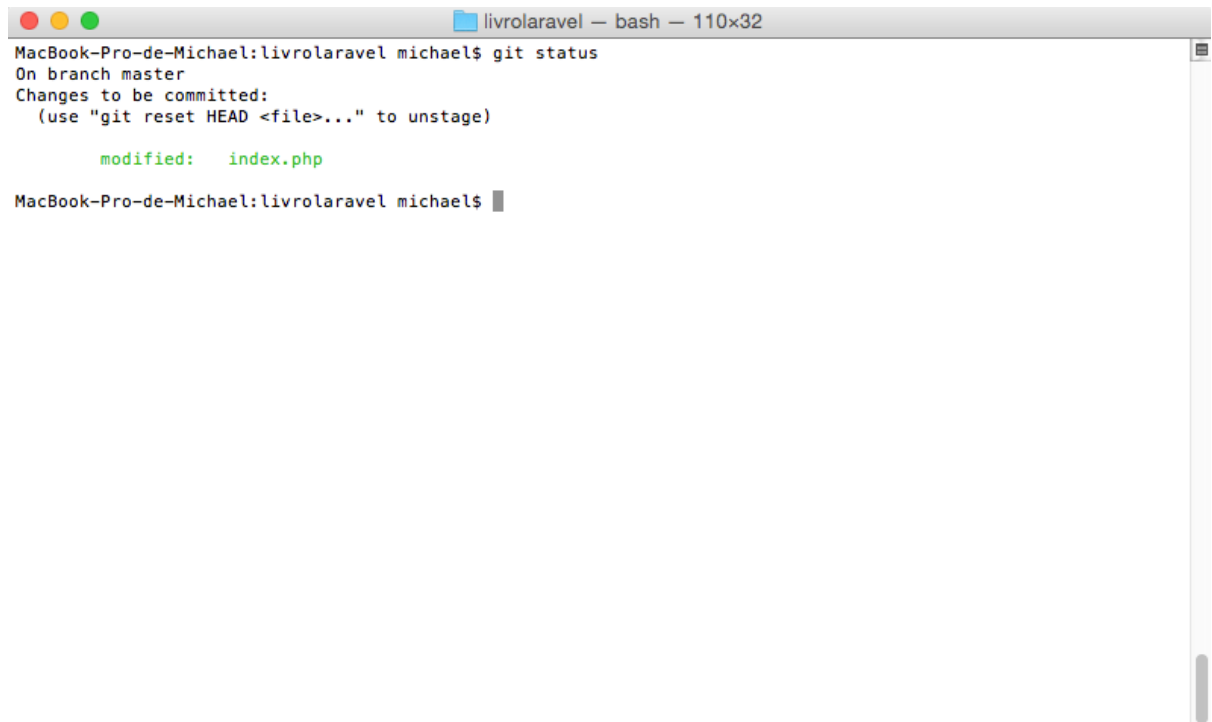
no changes added to commit (use "git add" and/or "git commit -a")
[1]+  Done                  ls -l
MacBook-Pro-de-Michael:livrolaravel michael$
```

Figura 2.6 ilustra a execução do comando para exibir a criação do arquivo e o status

Agora que você já percebeu que o status do documento é modificado precisamos adicionar essa mudança ao (INDEX) para isso você deverá executar o comando:

\$ git add index.php – Ou você pode utilizar * que irá adicionar todas as modificações e mais de uma modificação de arquivo, ou seja, caso contivesse mais arquivos todos seriam adicionados.

Agora você deve executar novamente o comando de status do git e irá perceber que ao adicionar o arquivo a (INDEX) a coloração e o texto mudam como você pode ver na Figura 2.7.

A screenshot of a macOS terminal window titled 'livrolaravel - bash - 110x32'. The terminal shows the output of the 'git status' command. The output indicates that the user is on the 'master' branch and that there are changes to be committed. Specifically, it shows 'modified: index.php'. The prompt 'MacBook-Pro-de-Michael:livrolaravel michael\$' is visible at the bottom of the terminal output.

```
MacBook-Pro-de-Michael:livrolaravel michael$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   index.php

MacBook-Pro-de-Michael:livrolaravel michael$
```

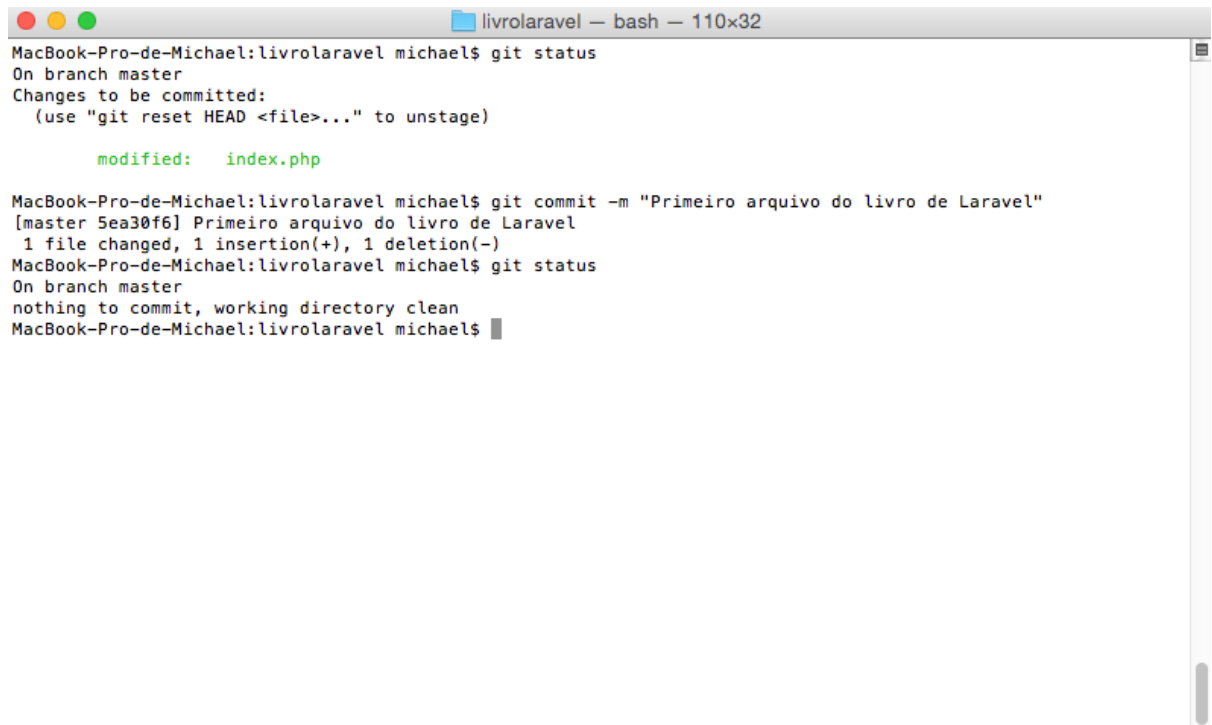
Figura 2.7 ilustra a execução do comando para exibir a alteração de status após o comando de adição.

Agora para que você confirme que esse arquivo pertence ao projeto você deve executar o famoso comando de commit que irá finalmente adicionar nosso arquivo ao (HEAD) e para isso use você deve inserir a seguinte instrução:

\$ git commit -m "Primeiro arquivo do Livro" – Você também pode executar o commando da seguinte omitindo o -m "".

\$ git commit – Dessa maneira será aberto o editor, por exemplo, vim ou vi para que você escreva a mensagem de confirmação de adição do arquivo.

Agora que você confirmou a adição do arquivo você verá pela primeira vez a sua branch máster como é ilustrado na Figura 2.8.

A screenshot of a macOS terminal window titled 'livrolaravel — bash — 110x32'. The terminal shows the following commands and output:

```
MacBook-Pro-de-Michael:livrolaravel michael$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   index.php

MacBook-Pro-de-Michael:livrolaravel michael$ git commit -m "Primeiro arquivo do livro de Laravel"
[master Sea30f6] Primeiro arquivo do livro de Laravel
1 file changed, 1 insertion(+), 1 deletion(-)
MacBook-Pro-de-Michael:livrolaravel michael$ git status
On branch master
nothing to commit, working directory clean
MacBook-Pro-de-Michael:livrolaravel michael$
```

Figura 2.8 ilustra a confirmação do arquivo e também a exibição da branch master.

2.6 Repositório remoto

Para trabalharmos com um repositório remoto utilizaremos o GitHub que basicamente é um serviço de Web Hosting compartilhado que acaba utilizando o GIT como controle de versionamento.

É muito utilizado por grandes projeto tais como:

- AngularJS
- Linux
- JQuery
- Laravel
- Ruby on Rails.

Para explicação de repositórios remotos irei utilizar a organização casa do PHP, essa organização foi criada para o Livro descomplicando a certificação PHP e também para o nosso curso.

O repositório que iremos é o que criamos agora para o nosso projeto:

<https://github.com/nossorepositorio>

Esse repositório irá conter os códigos que serão usados no livro e também materiais de apoio para acompanhamento do nosso livro e a partir desse repositório que continuaremos nossos estudos de GIT remoto.

Para você prosseguir você deve criar sua conta no GitHub e criar seu repositório para executar os próximos passos, não se preocupe irei demonstrar para você como criar sua conta e repositório GitHub caso você já tenha sua conta ou já conheça esse procedimento de criação de conta e repositório talvez os próximos passos não sejam uteis para você.

2.6 Criando sua conta no GitHub

Para criar sua conta primeiramente você deverá entrar no site do GitHub no link: <https://github.com> e então realizar o seu cadastro no formulário que é ilustrado na Figura 2.24.

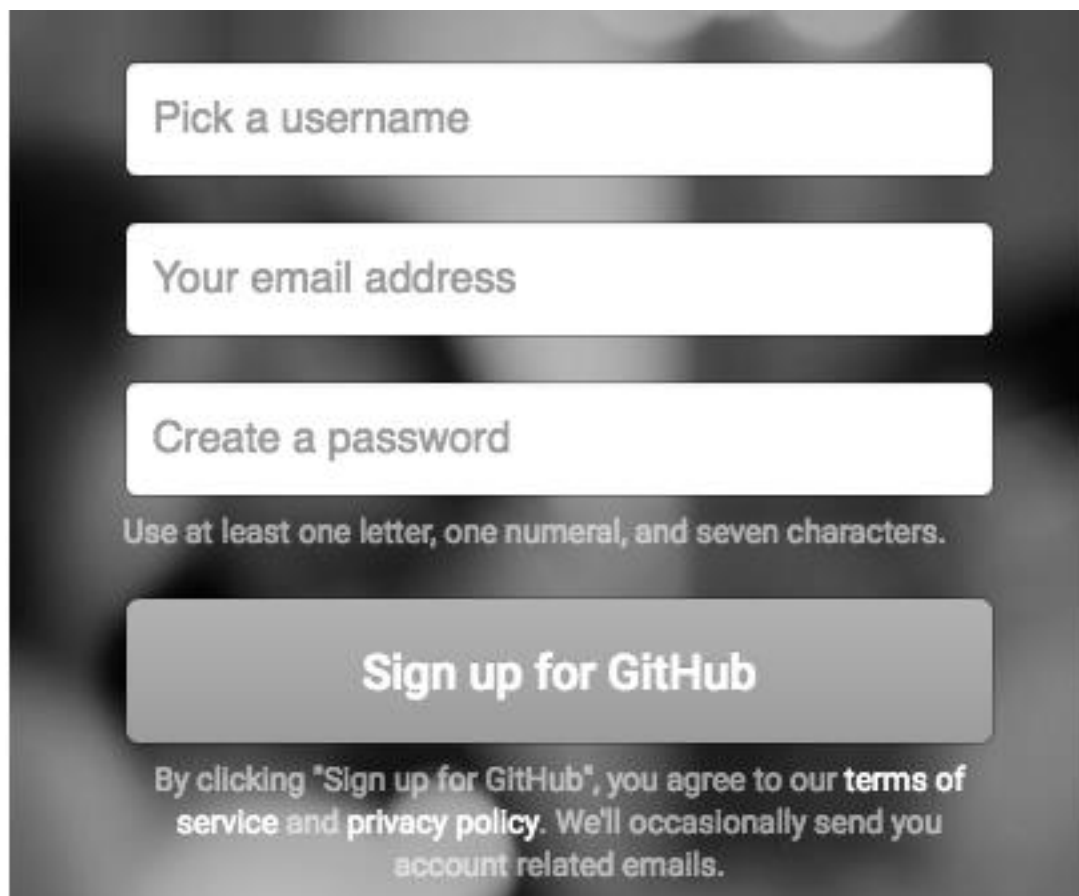
The image shows a screenshot of the GitHub sign-up form. It features three input fields: 'Pick a username', 'Your email address', and 'Create a password'. Below the password field is a note: 'Use at least one letter, one numeral, and seven characters.' At the bottom is a large button labeled 'Sign up for GitHub'. Below the button is a disclaimer: 'By clicking "Sign up for GitHub", you agree to our terms of service and privacy policy. We'll occasionally send you account related emails.'

Figura 2.24 ilustra o formulário de cadastro para o GitHub.

Irei criar uma conta utilizando dados fictícios de usuário e e-mail, o usuário que será utilizado é o cursogit o e-mail foi criado pela ferramenta:

<http://br.getairmail.com/zaipuyif/azm4> que gera e-mail temporária e permite que você leia normalmente os e-mails gerados pela ferramenta.

Depois que você cria sua conta pelo formulário exibido na Figura 2.24 você receberá no seu e-mail um link para confirmar a criação da sua conta no GitHub com isso você já tem acesso ao mundo do GitHub e poderá trabalhar com repositórios remotos normalmente.

Irei então subir o nosso primeiro commit no repositório do livro e você saberá os passos necessários para realizar seu primeiro comando de push para a branch master.

2.7 Enviar alterações com GIT

Nós já aprendemos sobre como: Adicionar e confirmar nossas alterações em nossa cópia da área de trabalho que estarão no **HEAD** caso tenha curiosidade de verificar o diretório mencionado você pode encontrar ele, por exemplo, no caminho: **caminhodascript/.git**

Agora precisamos aprender sobre como enviar nossas alterações para o repositório remoto no caso do livro como mencionado anteriormente será utilizado o repositório: <https://github.com/nossoprojeto>

Para realizar a tarefa de enviar os dados para o servidor remoto nós devemos então executar o comando de push, por exemplo, no caso do repositório do “cursogit” executaremos os passos a seguir que se baseiam no sistema Debian:


```
$ cd /var/www/cursogit  
$ git remote add origin https://github.com/nossorepositorio
```

Repare que eu executei um comando novo com GIT, no caso o comando é para adicionar uma origin para o push (**Envio**) e também serve para o próximo tópico que veremos que é o de pull (**Recebimento**), ou seja, eu criei uma origin de código que ao executar o comando de envio tudo que tiver sofrido alterações no código local estarão também no meu repositório externo.

Mas para que isso seja possível você precisa criar em sua conta do GitHub um repositório e isso é muito fácil basta que você clique na opção de criação de novo repositório ao fazer isso será aberta a tela que é ilustrada na Figura 2.25.

Create a new repository

A repository contains all the files for your project, including the revision history.


Owner
 **casadophp** ▼


Repository name

/


Great repository names are short and memorable. Need inspiration? How about **musical-dollop**.

Description (optional)

☒  **Public**
Anyone can see this repository. You choose who can commit.

☐  **Private**
You choose who can see and commit to this repository.

☐ **Initialize this repository with a README**
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

| 

Create repository

Figura 2.25 ilustra o formulário para criação de novo repositório

Basta que você coloque o nome do seu repositório e se você quiser também pode inserir uma descrição do seu repositório explicando o que mantém nesse seu repositórios códigos da sua aplicação, em nosso caso será armazenado os códigos utilizados no livro.

Depois que você inserir o repositório remoto você pode verificar qual esta utilizando com o comando:

```
$ git remote -v
```

Será retornado o **origin** que adicionamos anteriormente, ou seja, se você parar para pensar quando executamos o comando de adicionar uma origem poderíamos não colocar no comando de adição a palavra **origin** poderia ser no caso: **repositorioemoto, veja os exemplos com o origin e também o repositorioemoto adicionado:**

```
$ origin https://github.com/nossorepositorio (fetch)
$ origin https://github.com/ nossorepositorio (push)
$ repositorioemoto https://github.com/nossorepositorio (fetch)
$ repositorioemoto https://github.com/nossorepositorio (push)
```

Agora que você já conhece sobre como criar o repositório remoto iremos executar nosso comando para envio dos arquivos desenvolvidos no repositório local para o remoto com ajuda do comando:

```
$ git push origin master
```

Nesse comando nos realizamos nosso primeiro envio dos códigos para nosso repositório remoto e utilizamos o **origin** porém nada impede de você executar o mesmo comando utilizando o **repositorioemoto**, por exemplo:

```
$ git push repositorioemoto master
```

Caso você o retornado mostrado a baixo seus códigos estarão no Github como é ilustrado na Figura 2.26.

```
$ Counting objects: 3, done.  
Writing objects: 100% (3/3), 275 bytes | 0 bytes/s, done.  
Total 3 (delta 0), reused 0 (delta 0)  
To https://github.com/nossorepositorio  
* [new branch]    master -> master
```

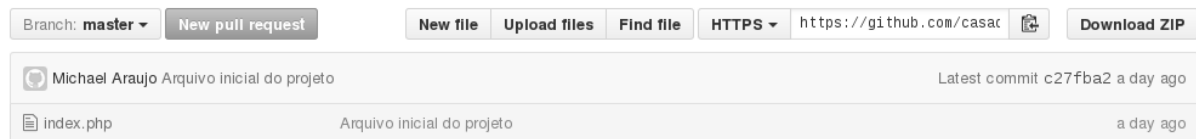


Figura 2.26 ilustra a primeira subida de arquivos para o repositório remoto.

2.8 GIT utilizando o Github com SSH

Essa configuração do GIT para o seu GitHub é interessante de ser feita pois fará com que você consiga ter acesso ao seu repositório remoto sem que se tenha a necessidade de inserir seu usuário e senha do GitHub a todo momento que você executar o comando: PUSH e PULL.

Porém se atente ao detalhe de que nessa configuração não será mais perguntando usuário e senha por medida de segurança sempre deixe bloqueado seu computador!

Para criar sua chave basta executar o comando:

```
$ ssh-keygen -t rsa -b 4096 -C "michaeldouglas@phpconference.com.br"
```

Você receberá como retorno o seguinte:

```
Generating public/private rsa key pair.
```

Os próximos questionamentos são importantes pois ele irá perguntar o local onde sua chave de acesso ficará por padrão ele irá colocar na pasta do usuário que está executando atualmente o comando:

```
Enter a file in which to save the key (/Users/michaeldouglas/.ssh/id_rsa): [aperte enter]
```

O próximo questionamento é se o acesso a ramificação externa possui uma frase de acesso em nosso caso como é nosso GitHub basta apertar enter:

```
Enter passphrase (empty for no passphrase): [Type a passphrase]  
Enter same passphrase again: [Type passphrase again]
```

Em nossa conta GitHub precisamos então associar essa chave criado a nossa conta e para isso você precisa logar, ir até sua configurações pessoais clicar em nova chave de SSH e então atribuir um nome e colocar o conteúdo da chave que foi criado em seu computador.

Com isso ao realizar um comando, por exemplo, de push ou pull em seu repositório externo não será mais perguntando sobre usuário e senha pois o GitHub já associa a sua conta a chave local.

2.9 Ramificando o seu projeto GIT

Os ramos ou branches são utilizados para que você seja capaz de desenvolver funcionalidades que estejam isoladas umas das outras.

Imagine o seguinte cenário: Você trabalha em uma equipe que são compostas por você e outro desenvolvedor e chega até vocês uma demanda para que seja realizado uma alteração no código da empresa. Dependendo da arquitetura do sistema o arquivo que compoem a alteração pode estar no mesmo script, ou por exemplo pode ser o famoso script index.php.

Outro cenário dessa mesma alteração pode ser a alteração da classe principal do sistema e vocês dois chegam a conclusão de que precisam trabalhar nessa alteração juntos devido ao prazo que foi passado a vocês dois ? - A questão que fica é: E se precisar alterar o mesmo script,, se precisar criar novos arquivos ? - Como realizar essa tarefa com 2 pessoas alterando o código ?

Agora que você já conhece o GIT talvez pense que é fácil a solução, por que ? - Você pode pensar em ajustar essa questão com um pensamento ruim porém simples que é: Fácil ajustar isso dos dois desenvolvedores pois cada um possui um repositório único em sua máquina, ou seja, ele trabalha na máquina dele e eu na minha e depois juntamos nossos desenvolvimentos, fácil não ?

Não é tão simples assim por que você pode até realizar esse procedimento mas veja que no momento da junção desses arquivos pode chegar-se na conclusão que só o ajuste do desenvolvedor 1 (um) faça sentido para a tarefa e você já mesclou todos os ramos no ramo principal que é a master para subida do desenvolvimento em produção para melhor entender esse cenário vamos criar um exemplo e então execute o seguinte código no projeto cursogit:

```
$ echo "Teste de branch" >> testbranch.php
$ git status
$ git add testbranch.php
$ git commit -m "Teste de branch"
$ git branch
```

O código que executamos é para criar um script php novo com o nome testbranch e em seguida adicionamos ele ao nosso repositório local assim como aprendemos anteriormente porém o que não estávamos dando atenção é sobre o que é a branch master que acaba sendo exibido em alguns momentos.

A branch master é a padrão de quando você cria o seu repositório git em nosso caso foi criado no momento que fizemos nosso primeiro comando de confirmação (commit), por ser a ramificação principal não devemos trabalhar na master e sim usar outras branches para desenvolver e também mesclar nossas alterações, iremos aprender agora sobre como criar uma nova ramificação e posteriormente mesclar nosso desenvolvimento a ramificação principal que é a master, veja o exemplo:

```
$ git branch
$ git checkout -b novatarefa
```

```
$ git checkout master
$ git checkout novatarefa
```

Esses códigos que acabamos de executar fazem o seguinte:

- `git branch` – Responsável por mostrar a visão de ramo atual no caso **master**
- `git checkout -b novatarefa` – Responsável por criar a nova branch **novatarefa** e também troca a visão do ramo para novatarefa, ou seja, o que você codificar no ramo novatarefa só ficará nessa ramificação e só será possível mesclar se executar o comando merge que veremos a seguir.
- `git checkout master` – Responsável apenas por trocar a visão para **master** novamente.
- `git checkout novatarefa` – Responsável apenas por retornar a visão para **novatarefa**.

Caso você crie errado ou até mesmo já tenha atendido a solicitação e deseja excluir uma ramificação você pode fazer dessa forma:

```
$ git checkout -b removerbranch
$ git branch
$ git checkout novatarefa
$ git branch -d removerbranch
```

O que existe de novo em nosso código é a adição da instrução para remover a branch do projeto, ou seja, executar a exclusão de uma ramificação com o nome: **removerbranch** o que você precisa executar para excluir é inserir **-d** e o nome da **branch** cuidado com a execução desse comando pois como você já pode perceber se excluir uma ramificação do projeto o que foi desenvolvido nela será perdido.

E por último caso você crie uma nova ramificação e esteja trabalhando com o seu projeto em um repositório remoto você precisa enviar essa ramificação para o repositório remoto utilizando o comando de push da seguinte maneira:

```
$ git push origin novatarefa
```

2.10 Git Blame

Em nosso dia a dia pode-se em algum momento haver a necessidade de saber quem fez uma última alteração de um arquivo ou até mesmo saber quando o arquivo foi modificado pela última vez.

A técnica de utilizar o comando `git blame` ajuda a descobrir que características foram adicionados a um release e o termo técnico para esse tipo de verificação é mais conhecido como: **blamestorming**.

A utilização do blame no Git é muito fácil basta você executar o comando git blame contra um arquivo e receberá como retorno as informações de:

- Timestamp
- Autor

O exemplo de execução do comando seria algo como:

```
$ git blame phpconference
```

```
56120453 (Michael Araujo 2016-01-07 09:43:39 +0100 1) First line  
e360ad22 (Michael Araujo 2016-01-07 09:43:51 +0100 2) Second line  
1361v631 (Michael Araujo 2016-01-07 09:44:06 +0100 3) Third line  
df1g2a33 (Michael Araujo 2016-01-07 09:43:51 +0100 4)
```

Existem casos que você pode apenas realizar um commit com espaço em branco e isso pode gerar ruídos com seu gerente ou até mesmo para você que está buscando a informação para evitar isso execute juntamente ao comando a flag **-e**.

2.11 Git Log

O comando de Log no GIT irá retornar o histórico de commits realizado no projeto ao passar do tempo você provavelmente vai querer saber o que anda acontecendo no seu projeto. E a ferramenta mais básica para isso no Git é o Git Log, ao obter um projeto que já tenha bastante commits ao executar o comando Git Log você terá como saída o seguinte:

```
$ git log
```

```
commit ca82a6dff817ec66f44342007202690a93763949  
Author: Michael Araujo <michaeldouglas@phpconference.com>  
Date: Mon Mar 17 21:52:11 2016 -0700
```

```
changed the verison number
```

```
commit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7  
Author: Michael Araujo <michaeldouglas@phpconference.com>  
Date: Mon Mar 17 21:52:11 2016 -0700
```

```
removed unnecessary test code
```

```
commit a11bef06a3f659402fe7563abf99ad00de2209e6  
Author: Michael Araujo <michaeldouglas@phpconference.com>  
Date: Mon Mar 17 21:52:11 2016 -0700
```

first commit

Esse comando ainda pode ser melhorado ao introduzir a flag: -p irá retornar um diff de cada commit no log e parar melhorar você pode introduzir -2 que irá limitar a saída em somente duas entradas.

```
$ git log -p -2
commit ca82a6dff817ec66f44342007202690a93763949
Author: Michael Araujo <michaeldouglas@phpconference.com>
Date: Mon Mar 17 21:52:11 2008 -0700
```

changed the verison number

```
diff --git a/Rakefile b/Rakefile
index a874b73..8f94139 100644
--- a/Rakefile
+++ b/Rakefile
@@ -5,7 +5,7 @@ require 'rake/gempackagetask'
spec = Gem::Specification.new do |s|
-  s.version = "0.1.0"
+  s.version = "0.1.1"
  s.author = "Scott Chacon"
```

```
commit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7
Author: Michael Araujo <michaeldouglas@phpconference.com>
Date: Sat Mar 15 16:40:33 2008 -0700
```

removed unnecessary test code

```
diff --git a/lib/simplegit.rb b/lib/simplegit.rb
index a0a60ae..47c6340 100644
--- a/lib/simplegit.rb
+++ b/lib/simplegit.rb
@@ -18,8 +18,3 @@ class SimpleGit
  end

end

-
-if $0 == __FILE__
-  git = SimpleGit.new
-  puts git.show
-end
\ No newline at end of file
```

É sempre bom você verificar o log do seu projeto para saber o andamento de tudo que anda entrar de novo, sendo editado e etc.

2.12 Desfazendo commits

Para desfazer um commit em seu projeto ou até mesmo para voltar ao estado em que sua branch se encontrava anteriormente as alterações você pode utilizar o comando: reset do git.

Porém esse comando deve ser utilizado com muito cuidado pois como já é bem sugestivo ele irá refazer ou voltar o estado da branch ou desfazer um commit para desfazer alterações feitas na branch você irá executar:

\$ git reset --hard – **MUITO CUIDADO AO EXECUTAR ESSE COMANDO!**

E para desfazer um commit em sua branch basta passar o ID do commit da seguinte maneira:

\$ git reset b24e5d595769dfe348c63ab18e3154f379e61370

E novamente o aviso de que tome cuidado a desfazer os **commits** do seu projeto pois se havia algum arquivo de código importante que você refez pode se dizer que dará bastante trabalho para ser recuperado!

2.13 Git stash

Depois de aprendermos muito sobre nosso fluxo de trabalho com o controle de versão do nosso programa chegamos a alguns problemas quando trabalhamos em equipe e/ou até mesmo em nosso dia a dia e um deles é o seguinte: Imagina que você iniciou o trabalho em um arquivo e seu chefe chega com uma demanda que precisa ser atendida naquele momento e precisa que você realize agora esse commit para produção ?

Podemos resolver de várias maneiras uma delas seria pegar o arquivo que estamos trabalhando agora e salvar localmente, outra maneira seria criar um branch do ajuste e trabalhar depois e entre outras.

Bom e para isso que existe o comando Stash do git ele permite que você salve temporariamente o arquivo que você está trabalhando para depois voltar a trabalhar nele e utilizar o stash é bem simples.

Para listar os arquivos que ficam no stash você pode utilizar o comando:

\$ git stash list

Como ainda não possuímos nenhum arquivo em stash não irá retornar nada vamos então simular a criação do arquivo e colocar ele em stash.

Para isso então execute os comandos:

```
$ touch phpconference
```

```
$ git add example
```

```
$ git stash
```

Lembra que não existia nada em nossa lista do stash ? – Agora execute novamente o comando de listagem de stash:

```
$ git stash list
```

Porém como ele “não” faz parte do seu projeto ainda ao listar os arquivos na pasta não é retornada nada o arquivo **phpconference**.

Agora imagine que sim você já pode obter o arquivo que está em stash para continuar o trabalho que parou, para isso execute então o comando:

```
$ git stash pop
```

Com isso seu arquivo estará relacionado em seu projeto e pronto para pertencer ao projeto!

2.14 *Git TAG*

A criação de rótulos do seu projeto é muito importante pois com ela você consegue definir versões do seu software, ou seja, diferente das suas primas branches que permitem alterações da ramificação e subir novos releases as tags já são um pouco diferente pois com elas definimos que será apontado para aquele conjunto de commits ou alterações que a partir daquele ponto são imutáveis, ou seja, serve muito bem para o conceito de versões do seu software e para utilizar também é bem simples para definir nosso primeiro release utilizamos, por exemplo, o comando:

```
$ git tag -a v1 -m "Version 1 release"
```

Esse comando irá criar a tag do momento atual do seu projeto e para verificar as tags existentes ou até mesmo para saber se o comando funcionou você pode executar o comando:

```
$ git tag
```

E caso você deseje excluir uma tag que você criou basta executar o comando:

```
$ git tag -d v1
```

E finalmente se desejar que esse release se torne oficial no seu projeto para o mundo, ou seja, no Github basta subir essa versão do seu software com o comando:

```
$ git push --tags
```

2.15 Git Config

O git é recheado de configurações que podemos alterar para trabalhar com ele em linha de comando não iremos focar em todos os comandos de configuração do GIT em seu computador porém existe duas configurações que são de extrema importância para você que é definir o e-mail do autor do projeto e também o nome, ou seja, ao realizar um commit de alteração é importante saber que é a pessoa que o fez.

Para configurar isso basta você executar o comando:

```
$ git config --global user.name "Michael Douglas Barbosa Araujo"  
$ git config --global user.email michaeldouglas010790@gmail.com
```

Repare que foi utilizado a flag: --global isso indica que você deseja que essa configuração se mantenha a qualquer projeto git em seu computador, caso você deseje manter seu nome de usuário e e-mail por projeto GIT basta retirar a flag que essa configuração só se manterá no repositório atual que você executou o comando.

Para verificar as configurações que você tem em seu GIT basta executar o comando:

```
$ ~/.gitconfig ou /etc/gitconfig
```

2.16 Fork

O processo de fork para quem trabalha com desenvolvimento de software open-source é de extrema importância pois existem casos em que o projeto que você mais gostava parou de ter suporte e você matinha um versão dele em seu projeto e você precisa realizar novos desenvolvimento que a biblioteca em si não possui !

E nesse momento que o fork irá salvar sua vida pois basta você realizar o fork do projeto e você terá uma versão desse software que pode ser editável por você!

Para realizar o fork de um projeto basta você entrar no GitHub e apertar o botão de fork que será copiado em seu usuário aquele projeto e você terá total acesso de edição em uma versão que é sua daquele projeto.

2.17 Pull Request

O processo de pull request é um pouco parecido com a explicação do Fork porém com pouca diferença, mas voltamos a imaginar o cenário do projeto que você encontrou no GitHub e tem vontade de contribuir pois ajuda você e muito no seu dia a dia e ao utilizar você encontrou um bug que você conseguiu corrigir e agora como subir essa solução para o projeto de outra pessoa ou empresa ?

E nesse momento que entra nosso processo de Pull Request, ou seja, você irá realizar uma solicitação para o dono do projeto com o ajuste que você realizou e caso seja aprovado seu ajuste no projeto ele entra como parte do projeto da biblioteca.

Além desse cenário mencionado o Pull request para o seu dia a dia é importante pois é sempre interessante que se tenha alguém responsável por validar os códigos que nós desenvolvedores criamos, ou seja, não é nada legal ficar realizando Push a torto e a direita no ramo do projeto sem que haja a devida validação.

3 Scrum

Referencia bibliográfica: <http://www.mindmaster.com.br/scrum>

Veja nós não iremos aprofundar muito o assunto é apenas introdutório!

O Scrum não é um processo padronizado onde metodicamente você segue uma série de etapas sequenciais e que vão garantir que você produza, no prazo e no orçamento, um produto de alta qualidade e que encanta os seus clientes.

Em vez disso, o Scrum é um framework para organizar e gerenciar trabalhos complexos, tal como projetos de desenvolvimento de software.

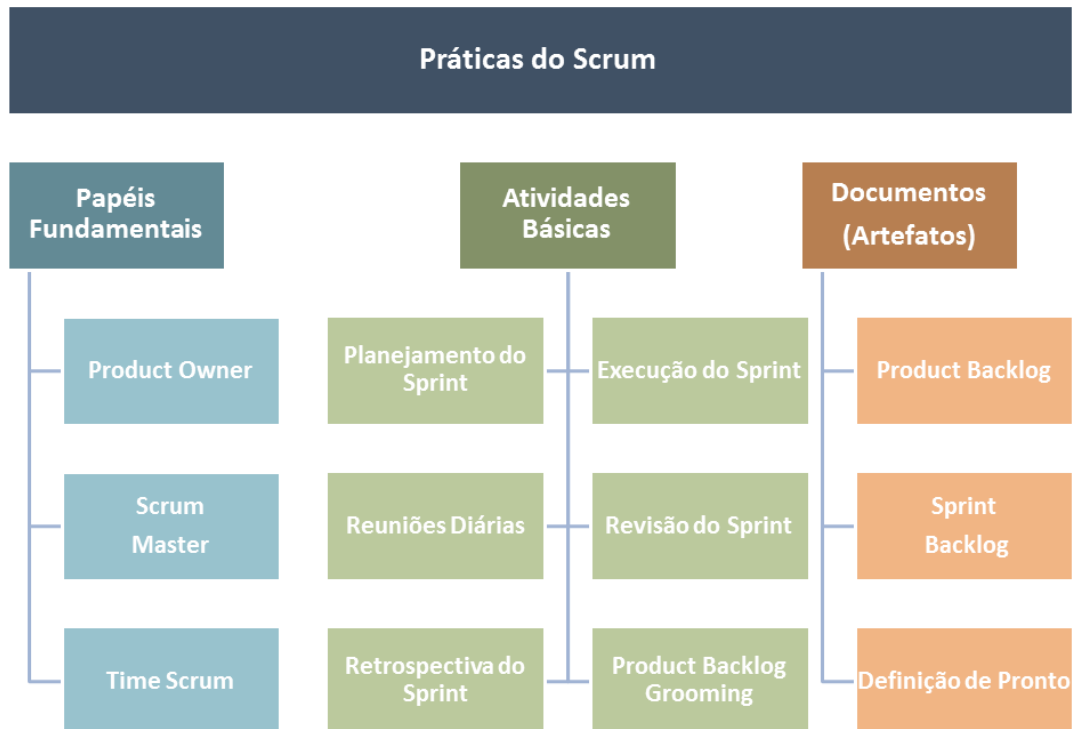
Alguém aqui sabe por que o nome da metodologia é Scrum ?

Esse nome veio de uma jogada do Rugby, onde é exigido do atleta que se tenha força de trabalho em equipe no caso do Rugby o objetivo final é avançar a bola oval no campo adversário o máximo possível. E para isso novamente precisamos realizar um trabalho em equipe.

Mas Michael por que Scrum e GIT no mesmo curso ? – Simples o Git deve acompanhar o andamento do seu projeto e o Scrum deve acompanhar o que é feito em seu repositório do projeto mas antes de continuar vejamos a tal jogada:



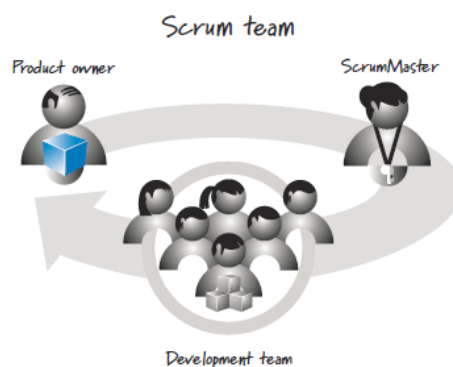
A base fundamental do Scrum segue nessa imagem:



3.1 Papéis Fundamentais

Os esforços de desenvolvimento utilizando Scrum consiste em uma ou mais equipes Scrum, cada uma composta basicamente de três papéis:

**Product Owner,
ScrumMaster e
Time de Desenvolvimento.**



3.2 Product Owner

Product Owner é o ponto central com poderes de liderança sobre o produto. Ele é o único responsável por decidir quais recursos e funcionalidades serão construídos e qual a ordem que devem ser feitos.

É responsabilidade dele manter e comunicar a todos os outros participantes uma visão clara do que a equipe Scrum está buscando alcançar no projeto. Como tal, ele é responsável pelo sucesso global da solução.

Para garantir que a equipe construa rapidamente o que o Product Owner precisa, ele deve colaborar ativamente com o ScrumMaster e equipe de desenvolvimento e deve estar disponível para responder às perguntas tão logo estas são feitas.

3.2 Scrum Master

O ScrumMaster é responsável por ajudar a todos os envolvidos a entender e abraçar os valores, princípios e práticas do Scrum.

Ela age como um Coach, executando a liderança do processo e ajudando a equipe Scrum (e o resto da organização) a desenvolver sua própria abordagem do Scrum, que tenha a melhor performance, respeitando as particularidades da organização.

O ScrumMaster também tem um papel de facilitador. Ele deve ajudar a equipe a resolver problemas e fazer melhorias no uso do Scrum. Ele também é responsável por proteger a equipe contra interferências externas e assume um papel de liderança na remoção de impedimentos que podem atrapalhar a produtividade.

3.2 Time Scrum

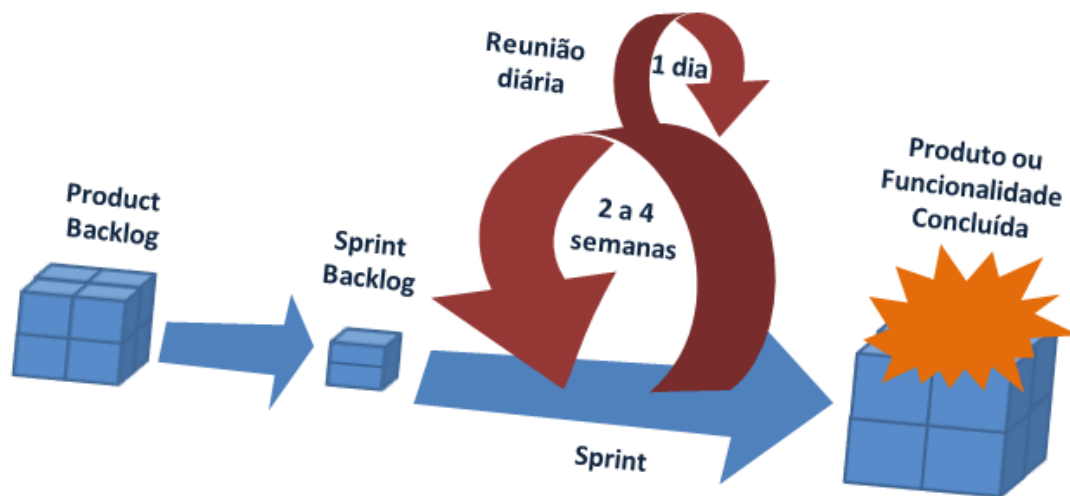
No desenvolvimento tradicional de software são abordados vários tipos de trabalho, tais como: arquiteto, programador, testador, administrador de banco de dados, Designer, e assim por diante.

No Scrum é definido o papel do Time de Desenvolvimento, que é simplesmente a junção de todas essas pessoas em uma equipe multidisciplinar, e que são responsáveis pela concepção, construção e testes do produto.

A idéia principal é que a equipe de desenvolvimento se auto-organiza para determinar a melhor maneira de realizar o trabalho para atingir a meta estabelecida pelo Product Owner.

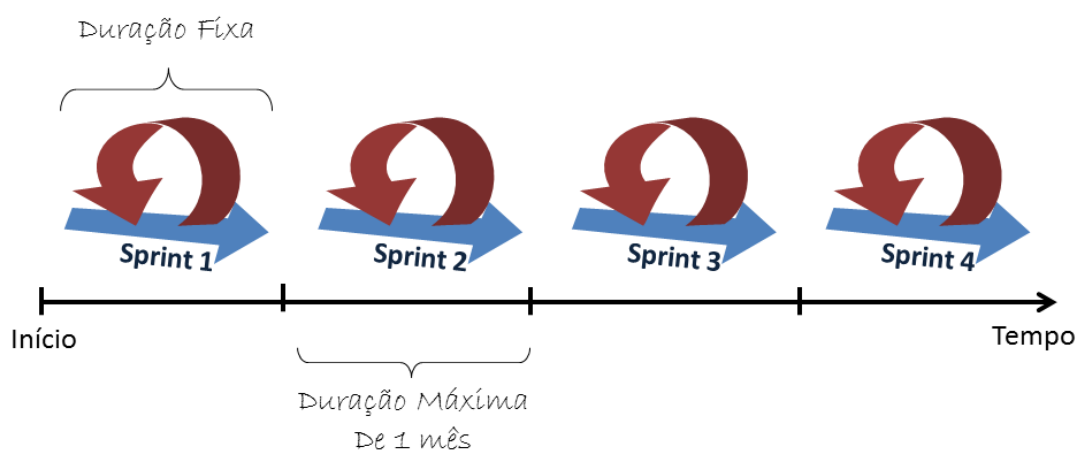
3.3 Atividades e Artefatos Principais

Não iremos focar em todos os artefatos do Scrum e sim apenas em explicar o Sprint que é o que interessa a nós para trabalhar com o Git juntamente com o Scrum veja a imagem:



Como mencionado vamos focar no artefato de Sprint que no Scrum é realizado em iterações ou ciclos de até um mês de calendário chamado de Sprints.

O trabalho realizado em cada sprint deve criar algo de valor tangível para o cliente ou usuário. Sprints são timeboxed (duração fixa) para que tenham sempre um início e fim data fixa, e, geralmente, todos eles devem estar com a mesma duração.



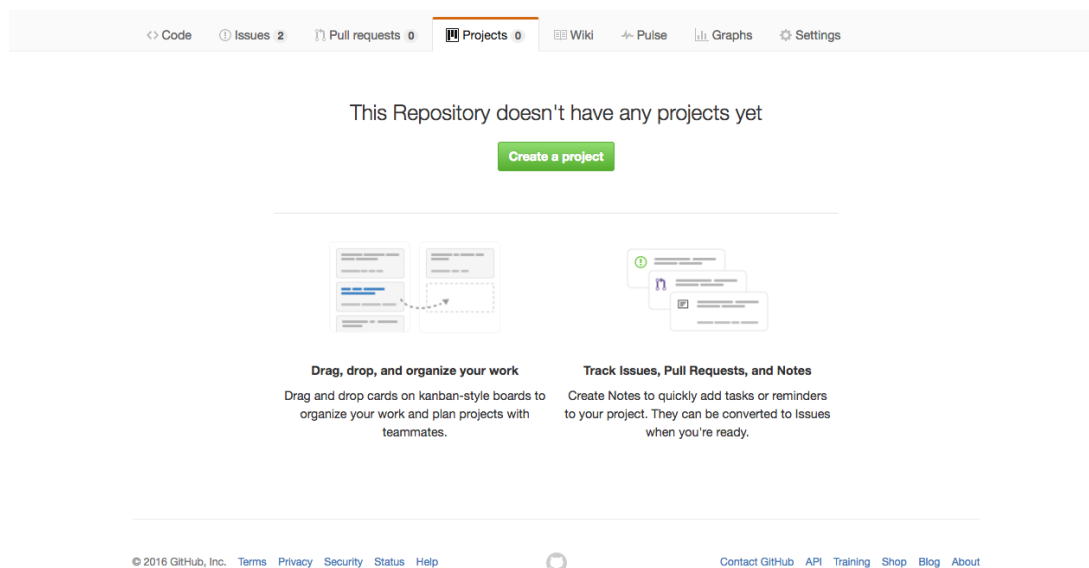
3.4 Uso do Kanban Github

Referencia bibliográfica: <https://www.significados.com.br/kanban>

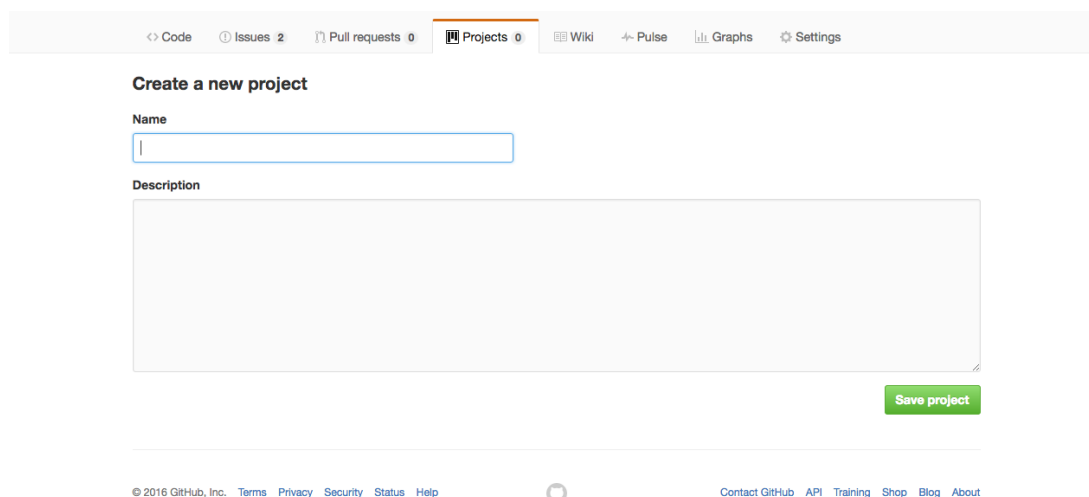
Kanban é um termo de origem japonesa e significa literalmente “cartão” ou “sinalização”.

Este é um conceito relacionado com a utilização de cartões (post-it e outros) para indicar o andamento dos fluxos de produção em empresas de fabricação em série.

E como estamos trabalhando com o Git e GitHub iremos utilizar um novo recurso da ferramenta que permite criar um projeto e inserir a lógica de Kanban para isso basta entrar no Github no repositório do projeto e então clicar na aba: **Projects**



Agora iremos clicar em: **Create a Project** e será perguntando sobre o nome do projeto e também uma descrição do mesmo:



Com isso já podemos então criar nosso board com nossas tarefas e manter o controle que aprendemos com o Scrum para os nossos futuros projetos ou até mesmo para nosso projeto atual que não segue as regras ensinadas.

3.4 *Exercício Scrum*

É bem simples o que faremos mas exige de vocês um verdadeiro trabalho em equipe a partir de agora eu serei o: Paulo Almeida que preciso da seguinte demanda:

“Desejo obter uma frota de aviões de papel que seja barata e sustentável esses aviões precisam voar por pelo menos 2 segundos e eu irei realizar esses testes com o Product Owner de cada equipe as empresa que irão licitar nesse projeto serão determinadas por vocês conto com o empenho de todos!”

4 PHP Unit

Uma das etapas mais importantes no seu dia a dia como desenvolvedor é manter os testes do seu projeto infelizmente não iremos nos aprofundar muito em testes porém não deixe de estudar pois isso é uma etapa muito importante do fluxo do seu projeto.

O PHP Unit é uma ferramenta de teste de unidade para PHP, ou seja, quando você cria um script no PHP você precisa garantir que aquilo esteja funcionando da melhor maneira possível e eu sei que você sabe que é fácil de garantir isso pois basta você realizar um refresh na página e você não recebeu nenhum crash da aplicação significa que está tudo rodando perfeitamente e sem nenhum problema.

Isso no inicio parece uma ideia bem interessante não é ? – É bem mais simples você garantir como criador que sua cria está rodando mas e quando essa cria se torna maior e maior e cada vez maior ?



Será que você irá lembrar de todos os locais que eventualmente se quebrariam em seu projeto ? – Talvez você pense: Talvez eu lembre e essa coisa de teste é chata mesmo !

Sim, posso garantir que no início realizar a tarefa de criar um script de teste não é nada agradável mas depois que um script desse salvar sua sexta-feira com um feriado na terça com emenda talvez você ame mais testes!

4.1 Instalação do PHP Unit

iremos usar um projeto como base para nossos estudos que contém uma pasta src e um arquivo composer.json nesse arquivo iremos especificar o desejo de utilizar o PHP Unit em nosso projeto e a estrutura do Json é a seguinte:

composer.json

```
{
  "name": "phpconference/curso-git",
  "description": "Contém os exemplos do curso de GIT",
  "keywords": ["curso", "phpconference"],
  "require": {
    "php": ">=5.4.0"
  },
  "require-dev": {
    "phpunit/phpunit": "4.0.*"
  },
  "autoload": {
    "psr-0": {
      "php\\conference\\": "src/"
    }
  },
  "authors": [
    {
      "name": "Michael Douglas Barbosa Araujo",
      "email": "michaeldouglas010790@gmail.com"
    }
  ],
  "scripts": {
    "test:unit": "phpunit -c phpunit.xml tests/unit --coverage-html=./tests/phpunit"
  },
  "config": {
    "bin-dir": "bin/"
  },
  "minimum-stability": "dev"
}
```


O que faz com que nosso PHP Unit seja instalado é a linha:

```
"require-dev": {  
    "phpunit/phpunit": "4.0.*"  
},
```

E a outra configuração que irá ajudar a executar o cli do php unit é a seguinte linha:

```
"config":{  
    "bin-dir": "bin/"  
},
```

Nela nós especificamos que o composer jogue nosso arquivo binário do php unit nessa pasta dando poder a você de executar o php unit da seguinte maneira:

```
$ php bin/phpunit
```

Iremos então criar uma class de calculadora simples que contém um método de soma e o corpo da nossa classe ficará da seguinte maneira:

Soma.php

```
namespace Php\Conference\Calculadora;  
  
class Soma  
{  
    private $valor1;  
    private $valor2;  
  
    public function getResultado($valor1, $valor2)  
    {  
        return $valor1 + $valor2;  
    }  
}
```

Uma classe simples que contém um método de obter o resultado de uma soma porém veja quem garante que esse resultado realmente esta realizando uma soma ?

Sendo assim a qualquer momento você ou sua equipe pode alterar a variável, alterar a estrutura da classe e entre outros problemas que podem surgir por isso motivo que criamos os testes que servem como garantia do estado da classe e seus métodos naquele momento do teste sendo assim se o seu software possuir uma boa

quantidade de testes o seu php unit ao rodar irá avisar sobre a ausência de uma lógica que existia no momento em que aquele teste foi criado.

Para utilizar o PHP Unit de forma definitiva em nosso projeto precisamos especificar o local de mapeamento e configurações do nosso projeto e para realizar isso criamos o arquivo phpunit.xml que contém as configurações para os testes do nosso projeto:

phpunit.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<phpunit backupGlobals="false"
    backupStaticAttributes="false"
    bootstrap="tests/unit/bootstrap.php"
    colors="true"
    convertErrorsToExceptions="true"
    convertNoticesToExceptions="true"
    convertWarningsToExceptions="true"
    processIsolation="false"
    stopOnFailure="false"
    syntaxCheck="false"
    >
    <testsuites>
        <testsuite name="Package Test Suite">
            <directory suffix=".php">./tests/</directory>
        </testsuite>
    </testsuites>
    <filter>
        <blacklist>
            <directory suffix=".php">./vendor</directory>
        </blacklist>
    </filter>
    <logging>
        <log type="coverage-html" target="./coverage/" />
    </logging>
</phpunit>
```

Agora então criaremos nossa classe de teste na seguinte estrutura:
tests\unit\Calculadora

E a nossa classe que irá testar a soma ficará da seguinte maneira:

SomaTest.php

```

<?php

namespace Php\Conference\Calculadora\Unit\Calculadora;

use Php\Conference\Calculadora\Soma;

class SomaTest extends \PHPUnit_Framework_TestCase
{
    protected $soma;
    protected function setUp()
    {
        $this->soma = new Soma;
    }

    public function testSomar()
    {
        $val1 = 1;
        $val2 = 2;
        $resultadoEsperado = 3;
        $total = $this->soma->getResultado($val1, $val2);
        $this->assertEquals($resultadoEsperado, $total);
    }

    public function testBoolSomar()
    {
        $val1 = 'a';
        $val2 = 2;
        $val3 = 2;
        $val4 = 'a';

        $resultado = $this->soma->getResultado($val1, $val2);
        $resultado2 = $this->soma->getResultado($val3, $val4);

        $this->assertEquals(false, $resultado);
        $this->assertEquals(false, $resultado2);
    }
}

```

Como nosso próximo passo é configurar o Travis para rodar nossos testes e posteriormente subir o deploy para produção deixaremos então nossos testes junto com o projeto mas se você possuir outra técnica que não envolva subir os scripts de teste e o projeto não é para o público de devs e sim para a sua empresa talvez subir os testes juntamente com o projeto não se faça necessário e lembre-se o teste deve fazer parte do seu Sprint e no Kanban a

coluna teste deve ser realmente utilizada para validar todos os testes do deploy que você irá realizar.

6 Gulp

O **gulp** é uma ferramenta de **automação de tarefas** feito com JavaScript e roda em cima do **NodeJS**.

Iremos aprender um pouco também de Gulp pois em nossos projetos PHP o Javascript, css e HTML andam juntos e precisamos automatizar essa etapa.

Caso você ainda não utilize automatizadores de tarefas deveria pois é uma ferramenta que ajuda nós programadores a realizar tarefas repetitivas porém essenciais para os nossos arquivos estáticos tais como: concatenação de arquivos, minificação, testes e entre outros!

6.1 Gulp X Grunt

Basicamente a escolha do Gulp para o seu automatizador de arquivos estáticos parece ser uma boa alternativa por que é mais rápido que o GRUNT e também muito mais simples de utilizar!

O Gulp acaba ganhando um pouco a batalha contra o Grunt por que em seu Core utiliza a lógica de Stream do Node JS e com isso dispensa intermediário para processamento, ou seja, quando falamos de intermediário de processamento significa que o Grunt depende de bibliotecas que ajudam na performance do processo de abertura do arquivo e minificação por exemplo.

6.2 Verificação de instalação do Nodejs

Como explicado o Gulp roda no node então antes de instalar o Gulp precisamos confirmar se o Node e o NPM estão instalados corretamente para isso execute em seu terminal:

```
$ node -v
```

E para confirmar se o NPM também está instalado execute em seu terminal:

```
$ npm -v
```

Basicamente o NPM é um gerenciador de pacotes para Node!

6.3 Instalação do Gulp

Agora que já foi confirmado a instalação do Node e NPM instalaremos o gerenciador de tarefas Gulp em nosso sistema operacional e para isso execute o comando:

```
$ npm install -g gulp
```

E para garantir que o Gulp foi instalado basta você executar em seu terminal o comando:

```
$ gulp -v
```

6.4 Estrutura para teste do Gulp

Iremos criar uma estrutura básica para nosso estudo de Gulp que ficará da seguinte maneira:

```
|cursogit/  
  |--dist/  
  |--src/  
    |--js  
    |--source.js  
    |--Gulpfile.js
```

6.5 Instalação dos plug-ins Gulp

O gulp sozinho não faz muita coisa ele precisa de alguns plug-ins para tornar as tarefas do nosso dia a dia mais fáceis de manter.

Iremos instalar alguns plug-ins bem úteis para nosso desenvolvimento que são eles:

- gulp-util – Serve para utilitário do Gulp
- gulp-uglify – Minificação de arquivos JS
- gulp-watch – Server para esperar mudanças em nossos arquivos.

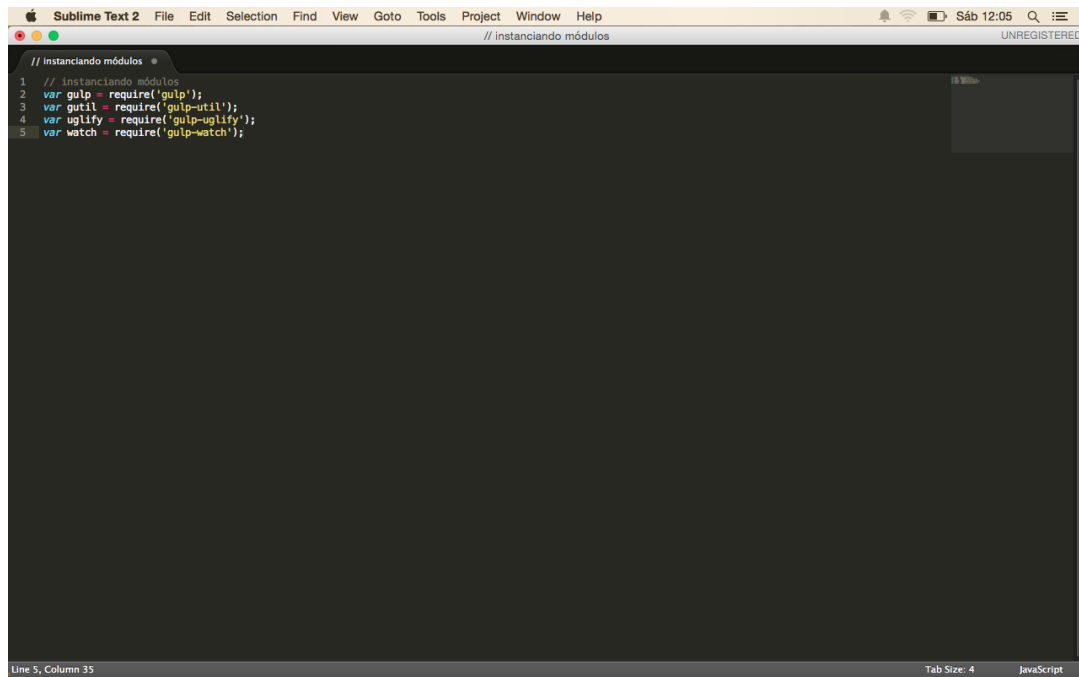
Lembre-se que o Gulp roda junto com o Node é justamente por isso que iremos criar um arquivo de gerenciamento de dependências de pacotes Node que basicamente chama-se: **package.json**.

Para criar o arquivo basta você executar em seu terminal na raiz do projeto os seguintes comandos:

```
$ npm init  
$ npm install gulp --save-dev  
$ npm install gulp-util --save-dev  
$ npm install gulp-uglify --save-dev  
$ npm install gulp-watch --save-dev  
$ npm install
```

6.6 Primeira tarefa

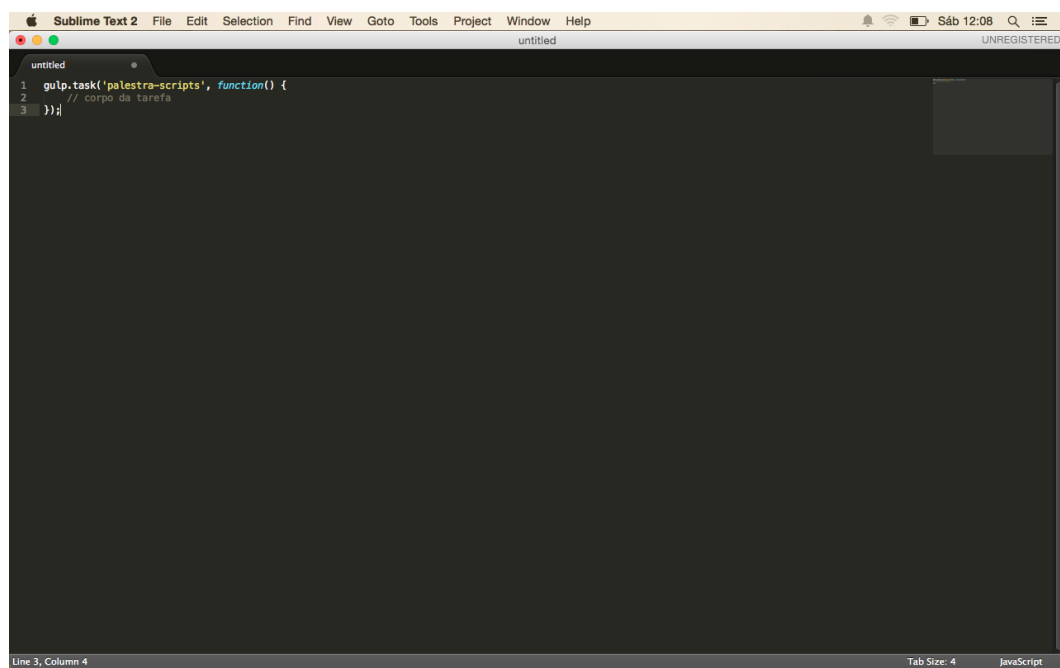
Para criar nossa primeira tarefa iremos abrir nosso arquivo Gulpfile.js e então definir uma tarefa que será responsável por minificar nossos arquivos de JavaScript.



The screenshot shows the Sublime Text 2 editor with a file named "Gulpfile.js". The code in the file is as follows:

```
// instanciando módulos
1 // instanciando módulos
2 var gulp = require('gulp');
3 var gutil = require('gulp-util');
4 var uglify = require('gulp-uglify');
5 var watch = require('gulp-watch');
```

The status bar at the bottom indicates "Line 5, Column 35", "Tab Size: 4", and "JavaScript".



The screenshot shows the Sublime Text 2 editor with a file named "Gulpfile.js". The code in the file is as follows:

```
1 gulp.task('palestra-scripts', function() {
2   // corpo da tarefa
3 });
```

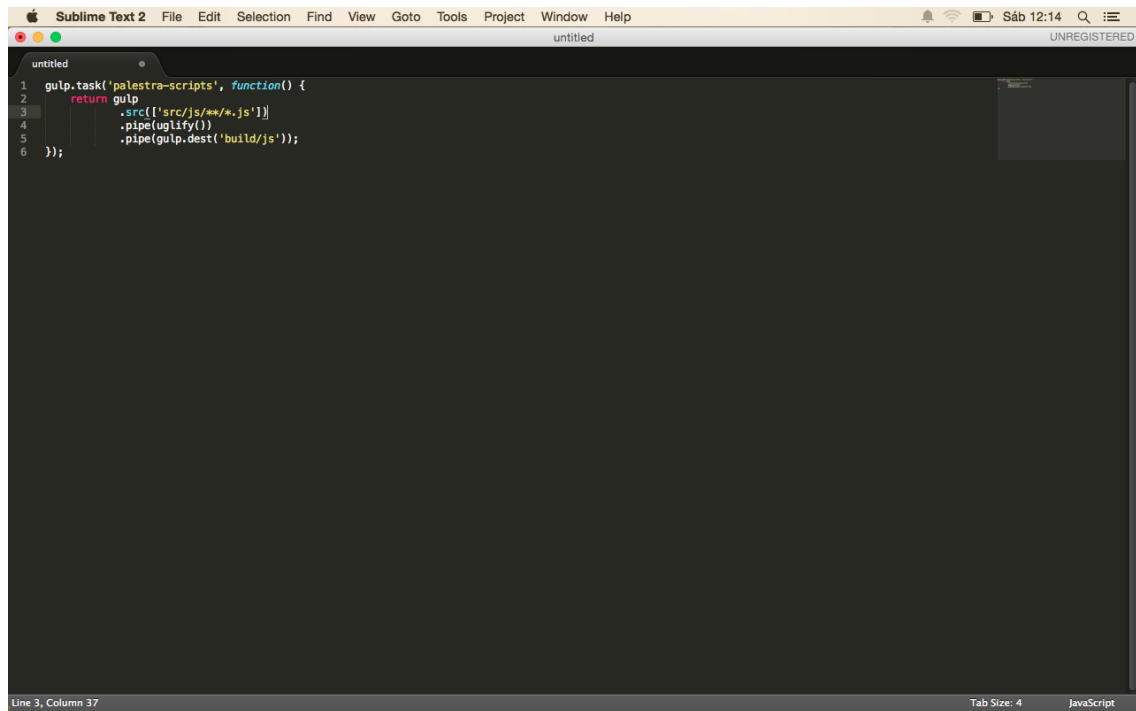
The status bar at the bottom indicates "Line 3, Column 4", "Tab Size: 4", and "JavaScript".

Até o momento nossa tarefa não tem nenhuma poder a seguir iremos atribuir para essa tarefa uma responsabilidade que será composta pelo seguinte:

- Obter os arquivos de JS
- Minificar os arquivos que obteve

- E colocar os arquivos na build

Que basicamente ficará da seguinte maneira:

A screenshot of the Sublime Text 2 editor interface. The window title is 'Sublime Text 2' and the menu bar includes 'File', 'Edit', 'Selection', 'Find', 'View', 'Goto', 'Tools', 'Project', 'Window', and 'Help'. The status bar at the top right shows 'Sáb 12:14' and 'UNREGISTERED'. The editor has a single tab titled 'untitled'. The code in the editor is as follows:

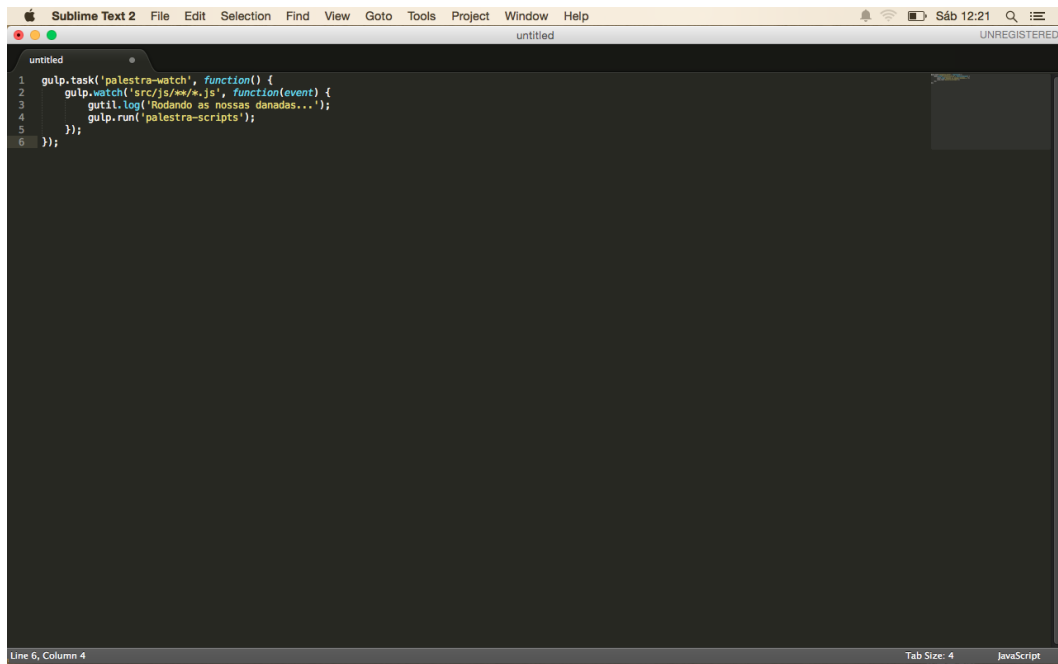
```
1 gulp.task('palestra-scripts', function() {  
2   return gulp  
3     .src(['src/js/**/*.js'])  
4     .pipe(uglify())  
5     .pipe(gulp.dest('build/js'));  
6 });
```

The status bar at the bottom left shows 'Line 3, Column 37' and the bottom right shows 'Tab Size: 4' and 'JavaScript'.

E para rodar essa nossa tarefa execute no terminal o seguinte comando:

```
$ gulp palestra-scripts
```

E por último criaremos um watch que ficará esperando que ocorra alterações em nosso script e em seguida ele roda nossa tarefa de minificação de arquivos Javascript e gera nossa build final do arquivo para isso basta criar essa regra em Gulpfile.js



```
1 gulp.task('palestra-watch', function() {
2   gulp.watch('src/js/**/*.js', function(event) {
3     gutil.log('Rodando as nossas danadas...');
4     gulp.run('palestra-scripts');
5   });
6 });
```

E para rodar nosso watch basta executar o comando:

```
$ gulp palestra-watch
```

Basicamente sobre a nossa estrutura de arquivos estáticos e isso em seguida falaremos sobre as nossas etapas finais para deixar nosso projeto realmente integrável e com deploy automático.

5 Travis CI

Segundo a Wikipedia: Deploy é a fase do ciclo de vida de um software (programa computacional, documentação e dados), no contexto de um Sistema de Informação, que corresponde textualmente à passagem do software para a produção.

O processo de implantação universal consiste de várias atividades intercaladas como possíveis transições entre elas. Estas atividades podem ocorrer no ambiente de produção e ou no ambiente de desenvolvimento ou em ambos. Pelo o fato de cada software ser único, o processo preciso ou procedimentos a serem seguidos são difíceis de definir. Além disto, a implantação pode ser interpretado como um processo universal que tem de ser customizado de acordo com requerimentos específicos ou características.

A ferramenta que iremos utilizar para criar nossa regra de deploy é o Travis CI que é um serviço de Integração Contínua na nuvem que pode ser conectado a repositórios no GitHub. Ele é gratuito para repositórios públicos e pago para repositórios privados.

É um serviço excelente, amplamente usado em projetos no GitHub. Possui suporte a diversas linguagens, como: C, C++, Clojure, Erlang, Go, Groovy, Haskell, Java, JavaScript (com Node.js), Objective-C, Perl, PHP, Python, Ruby, Scala, etc.. Vale lembrar que a documentação também é muito boa.

Em nosso caso iremos tirar proveito de tudo que aprendemos até aqui e iremos utilizar como etapa final do nosso novo fluxo de trabalho a ferramenta Travis e o Servidor em nuvem Heroku.

Por que a escolha dos dois ? – Simples por que são simples de configurar e também conseguimos utilizar os recursos de deploy e nuvem free obviamente com algumas limitações mas para nós que estamos aprendendo isso já é o suficiente.

5.1 Configurações de conta

Inicialmente você precisa criar as suas contas no Travis e também no Heroku vamos iniciar pelo Travis para criar sua conta entre no link: <https://travis-ci.org>

Então iremos criar nosso login a partir da nossa conta do GitHub clicando no botão: Sign in with GitHub.

Agora que você está conectado a sua conta Travis precisa informar a ele qual repositório será utilizado para o deploy da aplicação para isso você deve entrar em seu profile a partir do link: <https://travis-ci.org/profile/michaeldouglas>

Será exibida uma lista de repositórios e nós iremos escolher o nosso repositório do curso e marcar como ativo para integração continua.

5.2 Configurações *.travis no projeto*

Como o travis é por projeto ele compõem em sua lógicas que você utilize um arquivo chamado: **.travis.yml** esse irá conter toda a sua regra de deploy tais como:

- O projeto funciona na versão: php 5.6
- E deve executar o teste unitário

Em nosso projeto nosso arquivo ficará da seguinte maneira:

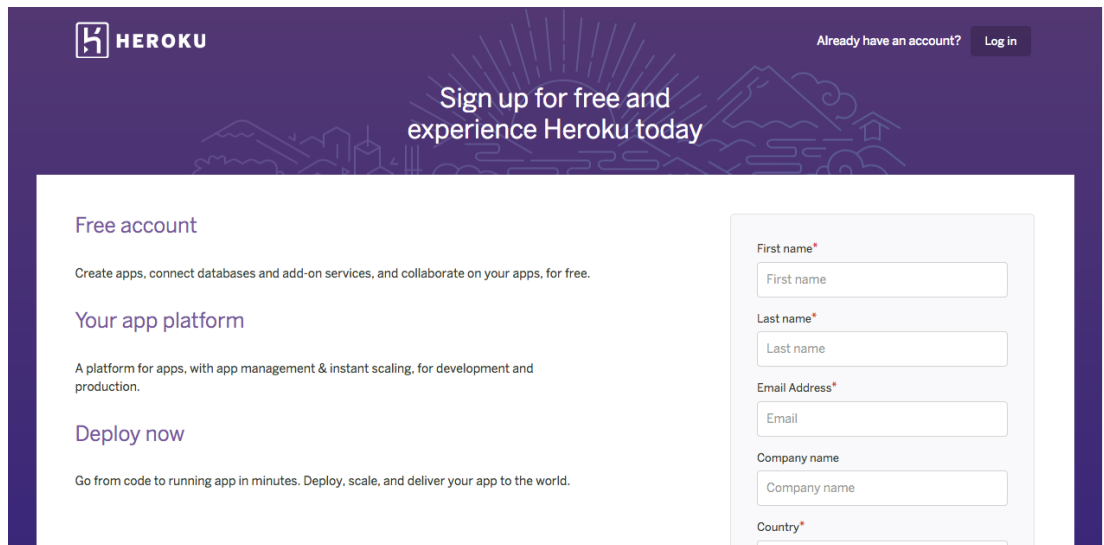
.travis.yml

```
language: php
php:
- 5.6
before_script:
- travis_retry composer self-update
- travis_retry composer install --prefer-source --no-interaction
deploy:
  provider: heroku
  api_key:
    secure:
NOIFDHjC0jRXgjliTGJrQGjpuw4m+kT8hFeQ663t13OC11XpzWlQyxMlhEKHXNinOg
yBtHYycu4clYKcs4BRU9KkYksT//fT74NsEUrQctfm8uLlafVE6hm5wKiZ5jr4b8f1BMo
s/kc7hJFpLNUeQYf4LPcekQ0Y88ExvCYeAC1eOYWwfAiNhJ1uP086NBy+pmcAJzL
eAZ2m49VzTmoggAdxX5RRvk2GqjXsKGNOj4qvVW3LS24iVk43Kd9dGZ8b8ER3drTI
F3oL2wY+a9GOBJFnXCIAReXT9HEXRjgwcuaCumh6EvT9fluMvg+g90fkfzABpFGp
c8zB9ei73DG+/lRrD0tkCvFeMgb3A6cb7PEm5ZLlgfvQnB1qkf8QSLft8VFfb/mnb8pbx
iHlIGte1zx/j506cqi4u/CIRZXJkpm8OFXzEmsNjf/OKNE29rFzT92IEs1xwvkSNMbWJ
OaOlxoMjvKh1zNZWxq95ExBel9QAWEcbTKQzVqqLTxbO96i0I7e/INahCkLE9vLS+
TI1+3Wi4+vgl8g72nff7QYGnQvMnI3BE5iL8nrAS4meSjJWxZkQYcsRQ5UQu446ioro
jWtlgYQEeqewJrSI9VT1BxKyvlpshJSF0jm/TfE1Af4PdHbNtM0YBC3BH/Cov2A30wjxd
SHGTIJDuvNxs+luAE=
  app: phpconference
  skip_cleanup: true
script: bin/phpunit
```

5.3 Configurações Heroku

Para iniciar a utilização de Heroku para hospedar seus softwares você inicialmente precisa criar sua conta e para isso basta você acessar o link:

<https://signup.heroku.com/?c=70130000001x9jFAAQ>



The image shows the Heroku sign-up page. At the top, there's a purple header with the Heroku logo on the left and links for 'Already have an account?' and 'Log in' on the right. The main heading says 'Sign up for free and experience Heroku today'. Below this, there's a white box containing the sign-up details. On the left side of this box, it says 'Free account' followed by 'Create apps, connect databases and add-on services, and collaborate on your apps, for free.' Below that, 'Your app platform' is followed by 'A platform for apps, with app management & instant scaling, for development and production.' Then 'Deploy now' is followed by 'Go from code to running app in minutes. Deploy, scale, and deliver your app to the world.' On the right side of the white box, there's a form with five fields: 'First name*', 'Last name*', 'Email Address*', 'Company name', and 'Country*'. Each field has a corresponding input box.

HEROKU

Already have an account? Log in

Sign up for free and experience Heroku today

Free account

Create apps, connect databases and add-on services, and collaborate on your apps, for free.

Your app platform

A platform for apps, with app management & instant scaling, for development and production.

Deploy now

Go from code to running app in minutes. Deploy, scale, and deliver your app to the world.

First name*

First name

Last name*

Last name

Email Address*

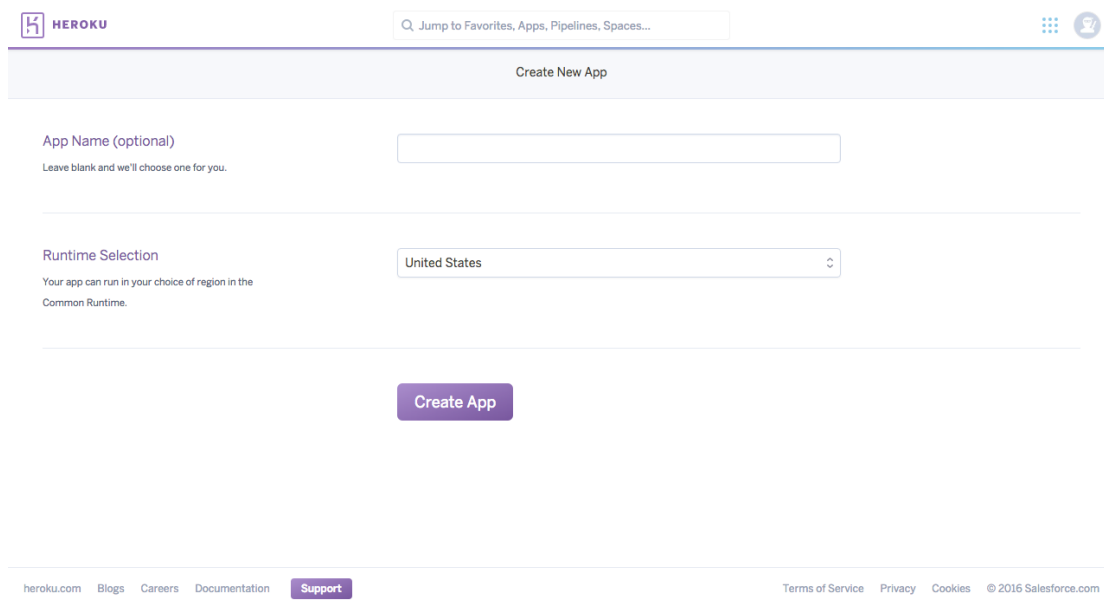
Email

Company name

Company name

Country*

Agora que você possui sua conta e esta devidamente logado precisa então criar seu APP no Heroku e isso é bem simples basta clicar em: new -> create new APP em seguida será aberto a você a seguinte tela:



The image shows the 'Create New App' page on Heroku. At the top, there's a purple header with the Heroku logo on the left, a search bar with the text 'Jump to Favorites, Apps, Pipelines, Spaces...', and a user profile icon on the right. Below the header, there's a light blue bar with the text 'Create New App'. The main content area is white and contains two sections. The first section is 'App Name (optional)' with a text input box and a note 'Leave blank and we'll choose one for you.' The second section is 'Runtime Selection' with a dropdown menu showing 'United States' and a note 'Your app can run in your choice of region in the Common Runtime.' At the bottom of the form, there's a purple button labeled 'Create App'. The footer of the page is a light blue bar with links for 'heroku.com', 'Blogs', 'Careers', 'Documentation', 'Support', 'Terms of Service', 'Privacy', 'Cookies', and '© 2016 Salesforce.com'.

HEROKU

Jump to Favorites, Apps, Pipelines, Spaces...

Create New App

App Name (optional)

Leave blank and we'll choose one for you.

Runtime Selection

Your app can run in your choice of region in the Common Runtime.

United States

Create App

heroku.com Blogs Careers Documentation Support

Terms of Service Privacy Cookies © 2016 Salesforce.com

Agora basta você definir um nome para a sua aplicação e em seguida clicar em criar seu novo app.

Feito isso precisamos então instalar o Heroku-CLI pois é com ele que iremos gerar nossa autenticação local com o servidor de produção.

Sendo assim baixe ele no link: <https://devcenter.heroku.com/articles/heroku-cli>

E para testar e já deixar a sessão ativa no seu terminal execute o comando:

```
$ heroku login
```

E por último precisamos instalar o travis-cli para que seja possível executar os comando de vinculo do travis com o Heroku e para isso basta executar o comando:

```
$ gem install travis-cli
```

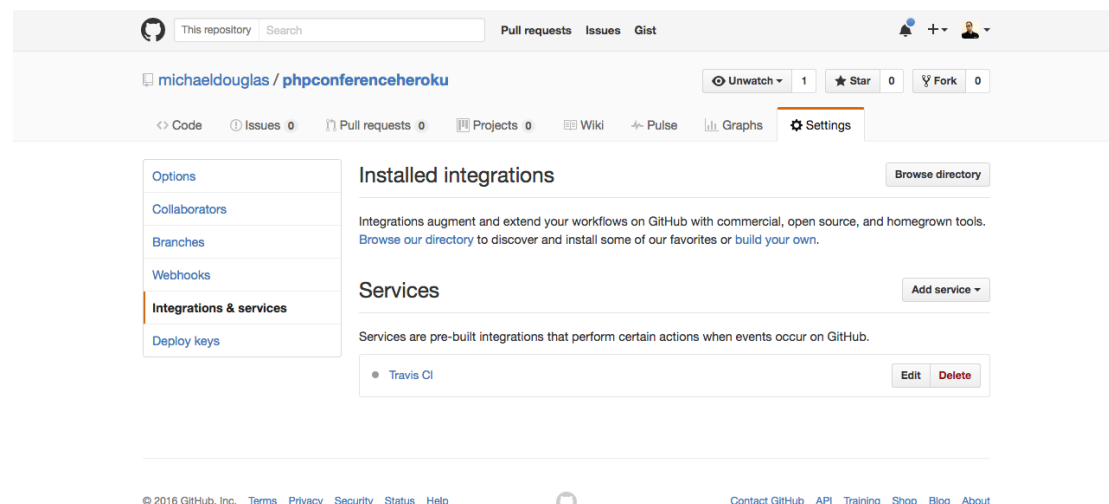
Agora na raiz do projeto execute o comando de vinculo do travis com o Heroku:

```
travis encrypt $(heroku auth:token) --add deploy.api_key
```

Esse comando fará com que o Travis gere a partir da sessão de login atual um token de acesso até o APP.

5.3 Configurações *GitHub* *travis*

Agora para deixar nosso processo funcionando com a integração continua que desejamos basta você entrar no repositório do projeto e configurar uma integração do GitHub com o Travis isso fará com que a cada push no projeto ele verifique essa release e se todos os teste passarem com sucesso então será criado um novo release do nosso projeto a partir desse commit e para isso basta você entrar em configurações e inserir o Travis CI:



Bom galera com tudo isso espero que no próximo Sprint da sua empresa você realmente possa afirmar:

**FTP não preciso mais
disso !**

