**USANDO O TRYGIT**

Este artigo tem como objetivo auxiliar o entendimento da plataforma GitHub, através da tradução do inglês para o português do tutorial TryGit.

Percebeu-se a necessidade de fazer a tradução do TryGit, uma vez que muitos Técnicos da Informação, graduandos e aspirante em tecnologia não possuem o domínio do idioma inglês.

O artigo será dividido em tópicos, da mesma forma que o tutorial é dividido, assim será possível acompanhar as traduções e comando paralelamente ao terminal do TryGit. Você também encontrará alguns conceitos de alguns termos técnicos assim como explicações mais detalhas.

No final do artigo existe um índice chamado **Vocabulário**, onde você verá o significado de algumas palavras técnicas.

1. **TEM 15 MINUTOS E QUER APRENDER GIT?**

O git permite grupos de pessoas trabalharem nos mesmos documentos (geralmente códigos) ao mesmo tempo, e sem incomodar uns aos outros. É um sistema distribuído de controle de versão.

Nosso terminal abaixo está atualmente em um diretório que decidimos chamar de “octobox”. Para iniciar o repositório do Git aqui, digite o seguinte comando:

* **git init**

1. **CHECANDO O STATUS**

Bom trabalho! Como o Git nos disse, nosso [diretório](https://pt.wikipedia.org/wiki/Diret%C3%B3rio_(computa%C3%A7%C3%A3o)) “octobox” agora tem um repositório vazio em /.git/. O repositório é uma pasta oculta onde o Git opera.

...

Vamos digitar o comando **git status** para ver qual é o estado atual do nosso projeto.

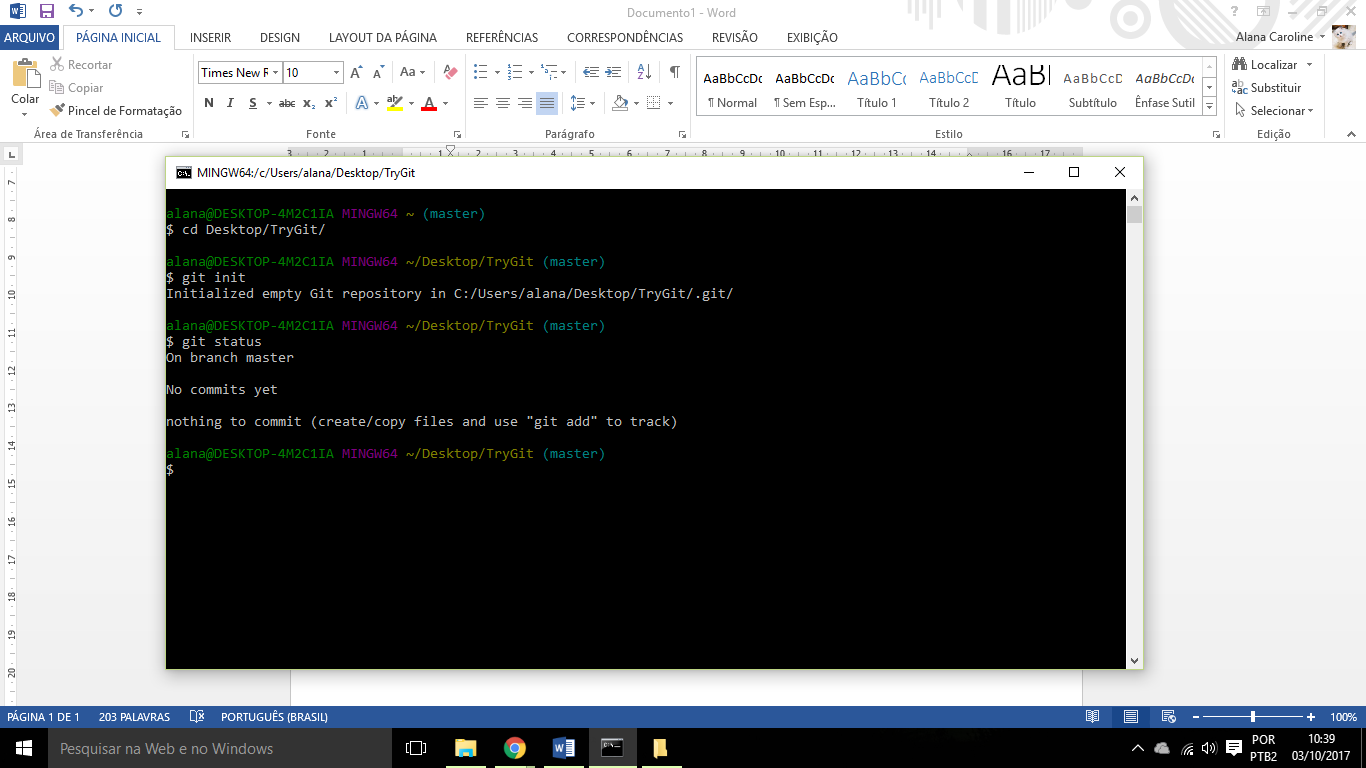
**Entendendo Melhor**

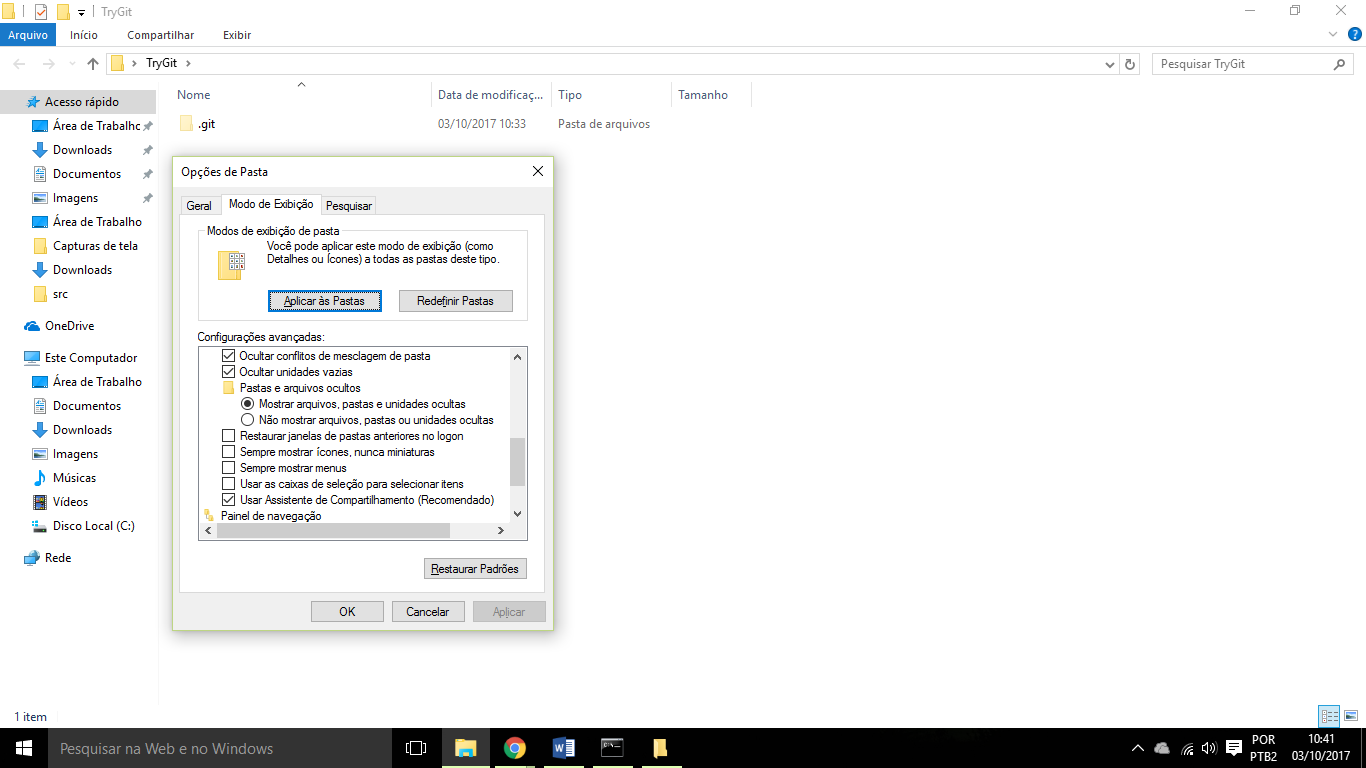
Para que você entenda melhor esse tópico, execute esses comandos em seu computador:

1. Crie um diretório em seu computador. (Para ser mais fácil, recomendo que crie na área de trabalho);
2. Abra o diretório e clique com o botão direito na tela;
3. Clique em Git Bash Here;
4. Digite o comando **git init**.
5. Mude a configuração de exibição das pastas para mostrar arquivos ocultos.
6. Agora você verá o repositório /.git/ que você acabou de criar.

PS: É necessário que o Git esteja instalado em sua máquina. Se você ainda não o tem instalado [clique aqui](https://git-scm.com/).

Segue abaixo algumas imagens de exemplo:





1. **ADICIONANDO E COMMITANDO**

Eu criei um arquivo chamado “**octocat.txt**” no repositório octobox para você (como você pode ver abaixo).

Você deve rodar o comando **git status** novamente para ver como o status do repositório mudou.

1. **ADICIONANDO MUDANÇAS**

Bom, parece que o nosso repositório Git está funcionando corretamente. Percebe como o Git diz **octocat.txt** não está sendo monitorado? Isso significa que o Git vê que o **octocat.txt** é um novo arquivo.

Para dizer para o Git começar monitorar mudanças feitas no **octocat.txt**,primeiro precisamos adicioná-lo à staging area (área de preparação) usando o comando:

* **git add <<nome do arquivo.extensão>>** - nesse caso o nome do arquivo é octocat.txt

1. **PROCURANDO AS MUDANÇAS**

Bom trabalho! Git está agora monitorando nosso arquivo **octocat.txt**. Vamos rodar o comando **git status** novamente para ver onde paramos.

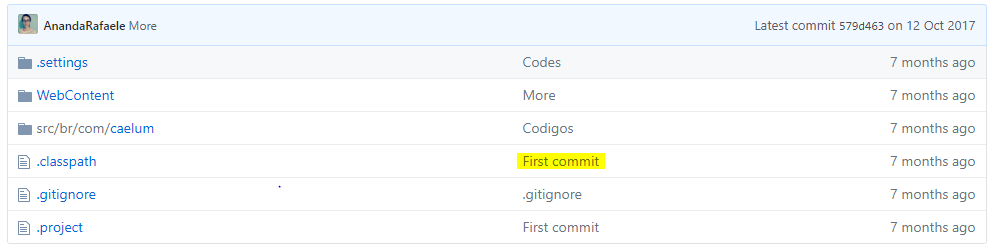
1. **COMMITANDO**

Notou como o Git diz “mudanças a serem commitadas”? Os arquivos listados aqui estão na staging area, e não estão nosso repositório ainda. Podemos adicionar ou remover arquivos da área antes de armazenarmos eles no repositório (comit).

Para gravarmos nossas mudanças devemos rodar o comando **comit** com uma mensagem descrevendo o que mudamos. Vamos fazer isso digitando:

* **git comit –m “First commit”**

O texto entre aspas no comando acima ficará como o exemplo abaixo:



1. **ADICIONANDO TODAS AS MUDAÇAS**

Ótimo! Você também pode usar *wildcards* (expressões regulares) se você quiser adicionar muitos arquivos do mesmo tipo de uma única vez. Veja que eu adicionei um monte de arquivos .txt no seu diretório abaixo.

Eu coloquei alguns arquivos no diretório chamado “octofamily” e alguns acabaram na pasta raíz **octobox**. Felizmente, podemos adicionar todos os novos arquivos usando uma *wildcards* com o comando **git add**. Não esqueça das aspas!

* Exemplo: git add ‘\*.txt’ – Neste caso a *wildcard* é o \* (asterisco)

1. **COMMITANDO TODAS AS MUDANÇAS**

Ok, você adicionou todos os arquivos na área de preparação. Sinta-se à vontade para usar o comando **git status** para ver o que você está prestes a commitar (gravar).

Se estiver tudo certo, prossiga e rode o comando.

* **git commit -m 'Add all the octocat txt files'**

**Observação:**

Para que o último comando rode, é necessário que você se identifique. Para isso use os comandos:

1. git config user.email “email”
2. git config user.name “nome”
3. Agora você pode executar o comando: git commit -m 'Add all the octocat txt files'
4. **HISTÓRICO**

Nós fizemos alguns commits. Agora vamos navegar nos commits para vermos o que mudamos.

Felizmente, existe o **git log**. Pense no **git log** como um diário que lembra-se de todas as mudanças que commitamos até agora, na ordem em que commitamos. Tente roda-lo agora

* **git log**

1. **REPOSITÓRIOS REMOTOS**

Bom trabalho! Nós criamos um novo e vazio repositório no GitHub para você usar com o TryGit no https://github.com/try-git/try\_git.git. Para colocar nosso repositório local no servidor do GitHub nós precisaremos adicionar um repositório remoto.

Este comando pega um *nome remoto* e o *URL de um repositório*, o que nesse caso será: [**https://github.com/try-git/try\_git.git**](https://github.com/try-git/try_git.git).

Prossiga rodando o comando **git remote add** com as opções abaixo:

* **git remote add origin** [**https://github.com/try-git/try\_git.git**](https://github.com/try-git/try_git.git)

Antes de seguir com a tradução do TryGit, eu sugiro que você vá em vocabulário, no final do artigo, e veja o significado de “push”, “pull” e “branch”, pois essas palavras serão bastante usadas daqui para frente.

1. **PUSHING REMOTAMENTE**

Este comando *push* diz ao **Git** onde colocar nossos commits quando estivermos prontos, e agora nós estamos. Então vamos “empurrar” nossas mudanças locais para nosso repositório **origin** no GitHub.

A sintaxe para o comando **push** normalmente é:

* **git push <remote> <branch>.**

Onde **remote** é um endereço de um repositório clonado que tem os mesmos dados e histórico e está apenas esperando para receber atualizações. Normalmente a maioria dos projetos trabalham com um remote sendo **origin** e o branch sendo **master** por padrão.

Informações tiradas do site **http://pt-br.gitready.com**

1. **PULLING REMOTAMENTE**

Vamos fingir que algum tempo se passou. Nós convidamos outras pessoas para o nosso projeto no GitHub que fizeram deram um *pull* em suas mudanças, fizeram as mudanças deles, commitaram e deram um *push* do que fizeram.

Nós podemos checar as mudanças no nosso repositório do GitHub e dar um push em qualquer mudança usando o comando:

* **git pull origin máster**

1. **DIFERENÇAS**

Parece que houveram algumas adições e mudanças na octocat family. Vamos dar uma olhada no que difere do nosso último commit usando o comando:

* **git diff**.

Nesse caso nós queremos as mudanças do nosso mais recente commit, o qual podemos o referenciar usando o ponteiro HEAD.

* **git diff HEAD –** serão descritas as diferenças que ocorreram desde a última vez que você commitou algo

1. **DIFERENÇAS EM STAGED**

Outro uso para o comando **diff** é procurar por mudanças dentro de arquivos que já estão sendo monitorados. Lembre-se, arquivos monitorados são arquivos que dissemos ao Git que estão prontos para serem commitados.

Vamos usar o **git add** para registrar **octofamily/octodog.txt**, o qual eu acabei de adicionar à família para você.

* **git add octofamily/octodog.txt**

1. **DIFERENÇA EM STAGED (CONTINUAÇÃO)**

Agora vamos rodar o comando **git diff** com a opção **--staged** para ver as mudanças que você acabou de registrar. Você verá que o **octodog.txt** foi criado.

* **git diff -- staged**

1. **RESETANDO O STAGE**

Agora que octodog é parte da família, o octocat está deprimido. Já que gostamos mais do octocat do que do octodog, nós iremos deixá-lo feliz removendo o **octodog.txt**.

Você pode deixar de monitorar arquivos usando o comando **git reset**. Agora remova **octofamily/octodog.txt**.

* **git reset octofamily/octodog.txt**

1. **DESFAZER**

O **git reset** fez um ótimo trabalho tirando octodog.txt do staged, mas você vai notar que ele continua lá. Ele só não está mais na staged area. Seria ótimo se nós pudéssemos voltar para como estava antes do octodog aparecer e acabar com a festa.

Arquivos podem ser alterados para como estavam no último commit usando o comando:

* **git checkout -- <arquivo>**

Agora se livre de todas as mudanças desde o último commit.

1. **RAMIFICANDO-SE**

Quando desenvolvedores estão trabalhando em funcionalidades ou bugs, eles geralmente criam uma cópia (aka. branch) do código ao qual podem realizar outros commits serapadamente. Então quando terminam, eles podem mesclar esse branch dentro do branch principal, ou seja, o **master**.

Nós queremos remover todos esses malditos octocats, então vamos criar um branch chamado **clean\_up**, onde nós vamos fazer todo o trabalho.

1. **TROCANDO DE BRANCHES**

Agora se você digitar **git branch** você verá dois branches locais: um branch principal chamado de **master** e seu novo branch chamado clean\_up.

Você pode trocar de branch usando o comando **git checkout <branch>**.

1. **REMOVENDO TUDO**

Então, você está no branch **clean\_up**. Você finalmente pode remover todos os malditos octocats usando o comando **git rm**. Ele não vai só remover o arquivo do HD, como vai também organizar a remoção dos arquivos para nós.

Você vai querer usar um wildcard de novo para pegar todos os octocats de uma única vez.

* **git rm ‘\*.txt”**

1. **COMMITANDO AS MUDANÇAS DO BRANCH**

Agora que vocês já removeram os gatos, você precisará commitar suas mudanças.

Sinta-se à vontade para rodar o **git status** checar as mudanças que estás prestes a commitar.

1. **VOLTANDO AO BRANCH PRINCIPAL (MASTER)**

Ótimo, você está quando terminando com os gatos... er de ajeitar os bugs, você só precisa voltar para o branch **master** para você copiar (ou mesclar) suas mudanças do branch **clean\_up** para o branch **master**.

Vá em frente e cheque o branch **máster**:

* **git checkout master**

1. **PREPARANDO PARA MESCLAR**

Certo, o momento de você mesclar suas mudanças do branch **clean\_up** para o branch **master** chegou. Respire fundo, não é tão assustador assim.

Nós já estamos no branch **master**, só precisamos dizer ao Git para mesclar o **clean\_up** no **master**.

* **git merge clean\_up**

1. **MANTENDO AS COISAS LIMPAS**

Parabéns! Você acabou de consertar os bugs e fazer a mesclagem com sucesso. Tudo que falta agora é limpar.

Agora que você já usou o **clean\_up** você não precisará mais dele.

Você pode usar o comando **git branch –d <brach name>** para deletar uma branch. Agora delete o a branch **clean\_up**.

1. **O *PUSH* FINAL**

Aqui estamos, no último passo. Estou orgulhoso por você ter chegado tão longe, e tem sido ótimo aprender git com você. Tudo que resta você fazer é o *push* em tudo que você veio trabalhando no seu repositório remoto, e você está livre.

* **git push**

**VOCABULÁRIO**

**Diretório** – diretório ou pasta é uma estrutura utilizada para organizar arquivos em um computador ou um arquivo que contém referências a outros arquivos.

**Repositório** – repositório de software é um local de armazenamento de onde pacotes de software podem ser recuperados e instalados em um computador.

**Staging** **area** - A área de preparação é um simples arquivo, geralmente contido no seu diretório Git, que armazena informações sobre o que irá em seu próximo commit.

**Commitar** – ato de gravar alterações no repositório.

**Push** – ato de enviar/empurrar as mudanças que foram feitas no seu repositório local para o repositório remoto do GitHub.

**Pull** – incorpora mudanças de um repositório remoto para o branch local.

**Branch** - é simplesmente um leve ponteiro móvel para um desses commits. O nome do branch padrão no Git é master. Como você inicialmente fez commits, você tem um branch principal (master branch) que aponta para o último commit que você fez. Cada vez que você faz um commit ele avança automaticamente.

**REFERÊNCIAS**

* <https://pt.wikipedia.org/wiki/Diret%C3%B3rio_(computa%C3%A7%C3%A3o)>
* <https://pt.wikipedia.org/wiki/Reposit%C3%B3rio_(software)>
* <https://git-scm.com/book/pt-br/v1/Primeiros-passos-No%C3%A7%C3%B5es-B%C3%A1sicas-de-Git>
* <https://git-scm.com/book/pt-br/v1/Git-Essencial-Gravando-Altera%C3%A7%C3%B5es-no-Reposit%C3%B3rio>
* <https://pt-br.gitready.com/iniciante/2009/01/21/pushing-and-pulling.html>
* <http://pt.stackoverflow.com/questions/3231/qual-a-diferença-entre-os-comandos-git-pull-e-git-fetch>
* <https://git-scm.com/book/pt-br/v1/Ramifica%C3%A7%C3%A3o-Branching-no-Git-O-que-%C3%A9-um-Branch>