**Configurando suas informações**

A primeira coisa que você deve fazer depois de instalar o Git é definir seu usarname e email. Isso é importante por que os seus commits usarão essas informações para identificar o autor das mudanças. Pois é… Se alguém fizer alguma merda no projeto e quebrar todo o sistema, é possível saber quem, quando e qual linha foi o autor do apocalipse.

É simples, no terminal escreva:

$ git config --global user.name "John Doe"

$ git config --global user.email johndoe@example.com

**Controlando um projeto**

Pelo terminal mesmo, entre na pasta do projeto que você quer iniciar o controle e use o comando:

git init

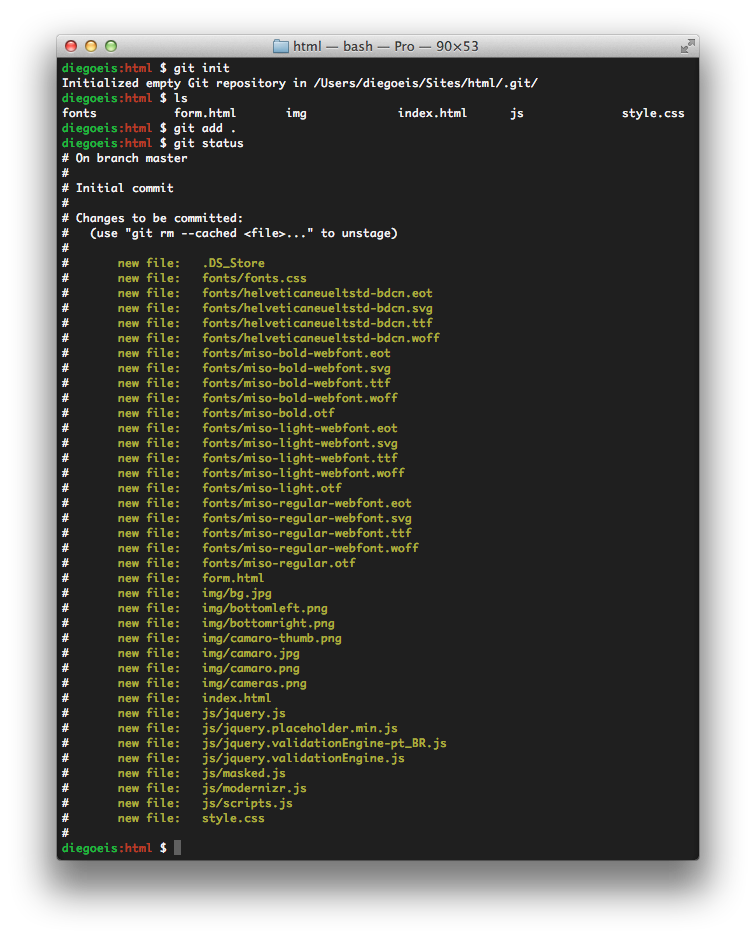
Esse comando vai criar um diretório invisível dentro do projeto chamado **.git**. Ele contém todos os arquivos necessários do seu repositório. Aqui, neste ponto, nada dos seus arquivos ainda estão sendo controlados. Você apenas criou um “lugar” (branch) para o Git colocar os arquivos.

O próximo comando vai inserir os arquivos que você quer controlar. Normalmente a gente controla TUDO o que está no projeto. Mas isso tem que ser combinado com a equipe antes. Em um projeto que envolve um CMS com o WordPress, por exemplo, é normal controlar tudo, até os arquivos do WordPress. Mas se em um projeto você guarda pastas de layouts, pastas de wireframes, protótipos e etc, é interessante não colocar isso no Git. Mas aí vai de equipe para equipe, de projeto pra projeto.

O comando para adicionar os arquivos é:

git add .

Para você ver o status, use o comando **git status**, aí você verá tudo o que foi incluído no projeto. Veja o screenshot abaixo para ter uma ideia:



Feito isso você vai precisar inserir seu primeiro commit. Vamos dar mais detalhes sobre o comando commit no próximo artigo, por agora fique com essa linha:

git commit -m "Primeiro commit - Inserindo os arquivos iniciais do projeto"

Agora você mandou uma alteração para o Git.

**Clonando um projeto**

Pode ser que já exista um projeto no Git criado e você só precise clonar para seu computador. Para isso você vai usar o comando **git clone**.

Quando você clona um projeto, o Git recebe a cópia de todos os dados que tem no servidor. Cada versão de cada arquivo da história inteira do projeto é puxada quando você roda o comando **git clone**.

Para clonar um projeto você precisa ter a URL do Git daquele projeto em específico. O comando completo fica mais ou menos assim:

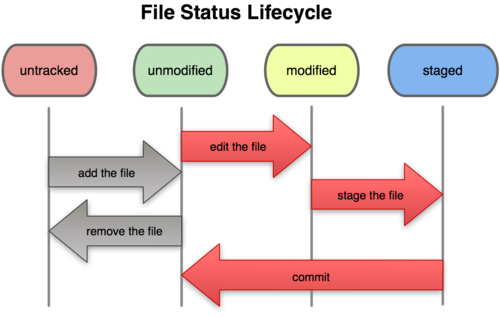
git clone https://github.com/tableless/exemplos.git

Pode testar com o endereço acima. Ele é nosso diretório do Git de exemplos no GitHub.

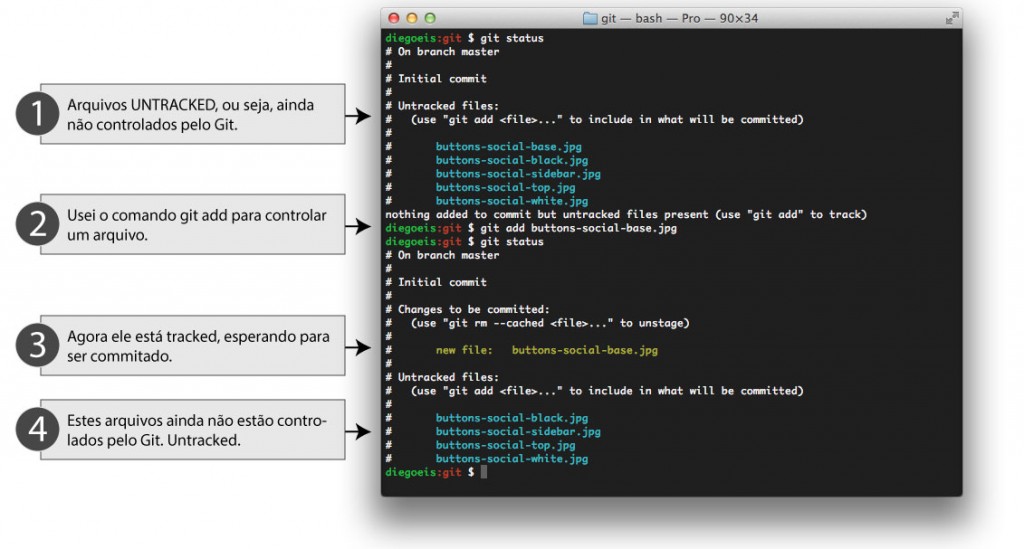
## Status

Antes de tudo você precisa entender em qual status os arquivos se encontram. Você pode modificar um arquivo, mas não commita-lo. Veja abaixo uma imagem direto do site do Git que mostra os diversos status dos arquivos.

Você já clonou ou iniciou seu projeto no Git e agora vai fazer modificações nos arquivos e enviar essas modificações para o repositório. Lembre-se que os arquivos em seu Work Directory podem estar **traked** ou **untracked**. Vou manter os termos em inglês para você se familiarizar melhor. Arquivos com status **tracked** são arquivos que já estão inseridos no repositório. Eles podem ser **unmodified** (que não foram modificados por você), **modified** (que foram modificados por você) ou **staged** (que são os arquivos que acabaram de ser mudados).



Esse ciclo é repetido diversas e diversas vezes. Veja abaixo um exemplo:

[](https://raw.githubusercontent.com/diegoeis/tableless-static-images/master/2012/11/git2.jpg)

## 10 Principais Comandos do Git

### Git Clone

O Git clone é um comando para baixar o código-fonte existente de um repositório remoto (como o Github, por exemplo).

Existem algumas maneiras de baixar o código-fonte, mas principalmente eu prefiro o clone com o modo https:

git clone <https://nome-do-link>

### Git Branch

Com branches, vários desenvolvedores podem trabalhar em paralelo no mesmo projeto simultaneamente.

Pode-se usar o comando **git branch** para criar, listar e excluir branches.

Criando uma nova branch:

git branch <nome-do-branch>

Este comando criará uma branch localmente. Para enviar a nova branch para o repositório remoto, você precisa usar o seguinte comando:

git push -u <remote> <nome-da-branch>

Para ver as branches:

git branch or git branch --list

Deletando uma branch:

git branch -d <nome-da-branch>

### Git Checkout

Este é um dos comandos Git mais usados. Para trabalhar em uma branch, primeiro você precisa mudar para ela.

Usamos o git checkout principalmente para alternar de um branch para outro. Também podemos usá-lo para verificar arquivos e commits:

git checkout <nome-da-branch>

Há também um comando de atalho que permite criar e trocar para um branch ao mesmo tempo:

git checkout -b <nome-da-branch>

### Git Status

O comando status do Git fornece todas as informações necessárias sobre o branch atual.

git status

### Git Add

Quando criamos, modificamos ou excluímos um arquivo, essas alterações ocorrerão em nosso ambiente local e não serão incluídas no próximo commit (a menos que alteremos as configurações).

Precisamos usar o comando **git add**para incluir as alterações de um arquivo em nosso próximo commit.

Para adicionar um arquivo:

git add <arquivo>

Para adicionar tudo de uma vez:

git add -A

O comando **git add** não altera o repositório e as alterações não são salvas até usarmos o git commit.

### Git Commit

Este comando é como definir um ponto de verificação no processo de desenvolvimento, para o qual você pode voltar mais tarde, se necessário.

git commit -m "mensagem explicando a mudança no código"

### Git Push

Após confirmar as alterações, a próxima coisa que você deseja fazer é enviar as alterações para o servidor remoto.

O comando **git push** envia e salva suas confirmações no repositório remoto.

git push <remote> <nome-do-branch>

No entanto, se seu branch for criado recentemente, você também precisará fazer upload do branch com o seguinte comando:

git push -u origin <nome-do-branch>

### Git Pull

O comando **git pull** é usado para obter atualizações do repositório remoto.

Este comando é uma combinação de git fetch e git merge, o que significa que, quando usamos o **git pull**, ele recebe as atualizações do repositório remoto **git fetch** e aplica imediatamente as alterações mais recentes no seu local **git merge**.

git pull <remote>

### Git Revert

Existem várias maneiras de desfazer nossas alterações local ou remotamente (depende do que precisamos), mas devemos usar esses comandos com cuidado para evitar problemas.

Uma maneira segura de desfazer os commits é usando **git revert**.

git revert 'número do hash'

O número do hash pode ser conseguido pelo comando:

git log -- oneline

### Git Merge

Quando você conclui o desenvolvimento em sua branch e tudo funciona bem, a etapa final é mesclar as branches, isso é feito com o comando **git merge**.

git merge <nome-da-branch>

Lembrando que existem diversas outras opções de comandos e configurações, que podem ser verificadas na [documentação oficial do Git](https://git-scm.com/docs/git-config).

Agora podemos identificar seu nome de usuário e e-mail para o git. Essas informações são importantes, pois cada commit efetuado por um usuário utiliza esses dados. Os comandos são os seguintes:

git config --global user.name “fulano”

git config --global user.email [fulano@email.com](mailto:fulano@email.com)

Para selecionar o editor de texto padrão do Git, como o Nano, por exemplo, é

necessário entrar com o seguinte comando:

git config --global core.editor nano

Conforme mostrado na Figura 5, para criar começar a realizar os registros do seu projeto, basta se deslocar via terminal até o diretório onde os arquivos do projeto estão localizados e entrar com o seguinte comando:

git init

git add oimundo.c

O comando acima especifica que o arquivo example.c irá ser usado em um commit

posterior. Com o arquivo oimundo.c na lista de arquivos que serão consolidados e, sendo apenas ele necessário na mesma lista, só é preciso efetuar um commit:

git commit -m “Alguma mensagem”

É importante lembrar que os arquivos presentes no diretório podem estar em dois estados distintos: monitorado ou não monitorado. Os arquivos monitorados são os arquivos presentes no último commit, podendo estar inalterados, modificados ou selecionados para um outro commit. Os arquivos não monitorados são os que não foram consolidados e também não estão presentes em nenhuma seleção pré commit.

Para visualizar o status dos seus arquivos, é usado o comando:

git status

O comando status tem como saída o estado em que se encontra cada arquivo modificado. Você pode efetuar o comando para um arquivo específico da seguinte maneira:

git status example.c

É possível observar as mudanças feitas em arquivos monitorados modificados através do comando:

git diff

Para remover arquivos presentes no seu repositório .git, é preciso removê-lo não só do diretório em que ele se encontra, mas também é removê-lo da área de seleção e então efetuar o commit. A seguir, um exemplo:

git rm “arquivo destino”

Para verificar o histórico dos commits efetuados em determinado repositório, existe o comando:

git log

Pode ser necessário desfazer alguma modificação que gerou um erro no projeto.

Para poder refazer o último commit efetuado, é utilizada a diretiva --amend do comando git commit como no exemplo a seguir:

git commit –amend

Vamos começar exibindo os remotos do seu repositório. Para isso entre com o

comando:

git remote

É possível ainda, utilizar a diretiva “-v” para mostrar a url completa do seu remoto. Para adicionar remotos, usa-se o comando “git remote add” como demonstrado:

git remote add “apelido/nome curto” “url”

Quando o seu trabalho está pronto e já lhe é de interesse enviar os arquivos

modificados ao remoto do projeto, usa-se o comando “push” para efetuar a tarefa. É possível enviar o arquivo para o remoto de origem ou para um remoto em específico

git push origin master

**Configurando suas informações**

A primeira coisa que você deve fazer depois de instalar o Git é definir seu usarname e email. Isso é importante por que os seus commits usarão essas informações para identificar o autor das mudanças. Pois é… Se alguém fizer alguma merda no projeto e quebrar todo o sistema, é possível saber quem, quando e qual linha foi o autor do apocalipse.

É simples, no terminal escreva:

$ git config --global user.name "John Doe"

$ git config --global user.email [johndoe@example.com](mailto:johndoe@example.com)

$ git config –list (ver todas configurações)

**Gravando alterações**

$ git init (inicializa repositório)

$ git add nome.extensaoarquivo (adiciona na área de stage, assim é rastreado)

$ git commit -m “mensagem/comentário” (envia para o repositório)

$ git status ou git status -s

É possível rastrear as mudanças e comitá-las de uma vez com a opção -a:

$ git commit -am " mensagem/comentário " ou git commit -a -m " mensagem/comentário "

É possível observar as mudanças feitas em arquivos monitorados/modificados, ou seja, antes de fazer commit, através do comando:

$ git diff

$ git diff --name-only (vê somente arquivos modificados)

É possível mostrar as diferenças entre os arquivos na área de stage e a última versão que foi comitada utilizando a opção --staged:

$ git diff --staged



**Se encontram modified,** caso os arquivos tenham sido modificados e, no momento, se encontram diferentes do que está committed no diretório Git.

**Se encontra staged,** quando o usuário já terminou todas suas mudanças no arquivo e, lhe enviou pra área de staged para que as mudanças efetuadas sejam registradas como committed.

**Se encontram committed,** quando já estão assegurados na base de dados do diretório Git.



Lembre-se que cada arquivo em seu diretório de trabalho pode estar em um dos seguintes estados: rastreado e não-rastreado.

Arquivos rastreados são arquivos que foram incluídos no último snapshot;

eles podem ser não modificados, modificados ou preparados (adicionados ao stage).

Em resumo, arquivos rastreados são os arquivos que o Git conhece (Unmodifie, Modified, Staged)

Arquivos não rastreados são todos os outros - quaisquer arquivos em seu diretório de trabalho que não foram incluídos em seu último snapshot e não estão na área de stage. Quando você clona um repositório pela primeira vez, todos os seus arquivos serão rastreados e não modificados já que o Git acabou de obtê-los e você ainda não editou nada.

Assim que você edita alguns arquivos, Git os considera modificados, porque você os editou desde o seu último commit. Você preparar os arquivos editados e então faz commit das suas alterações, e o ciclo se repete.

**Histórico**

Para verificar o histórico das alterações gravadas no repositório, podemos executar o comando:

$ git log

$ git log -p

$ git log --stat (mostrar um resumo dos arquivos alterados)

$ git log --oneline (resumo bem conciso dos commits)

**Ignorando arquivos**

Basta criar arquivo .gitignore e descrever nele o que deve ser ignorado (lembrar de add e comitar o .gitignore)

**Desfazendo mudanças**

$ git commit --amend ;

**Renomeando arquivos do repositório**

$ git mv nome.extensão novonome.extensão

Porém, não basta deletarmos o arquivo. Precisamos deletá-lo e adicionar a deleção na stage, para só então efetuarmos um commit no repositório.

**Removendo arquivos do repositório**

$ git rm nomeEextensãoarquivo

Porém, não basta deletarmos o arquivo. Precisamos deletá-lo e adicionar a deleção na stage, para só então efetuarmos um commit no repositório.