**Como Usar o GIT**

Agora que o GIT está instalado e configurado no seu dispositivo, vamos explorar os conceitos básicos do GIT e como começar a usar o GIT.

**Criar/configurar/verificar um repositório**

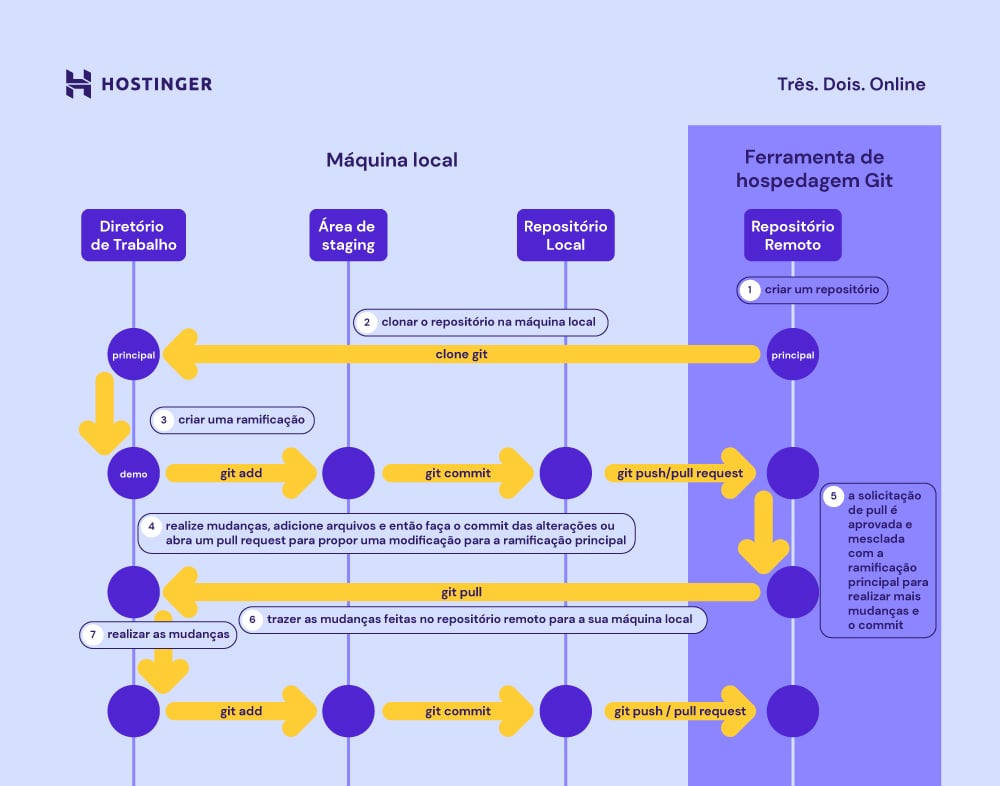
Um repositório é o maior bem de qualquer projeto controlado por versão. Para transformar qualquer diretório em um repositório GIT, o simples comando git init <directory> pode ser utilizado. Uma pasta chamada .git também deve começar a existir no diretório em que o comando foi executado.

Por outro lado, se você já tem um diretório e deseja verificar (clone-lo), você pode usar o comando git clone. Se você estiver tentando verificar um repositório local, use o seguinte comando:

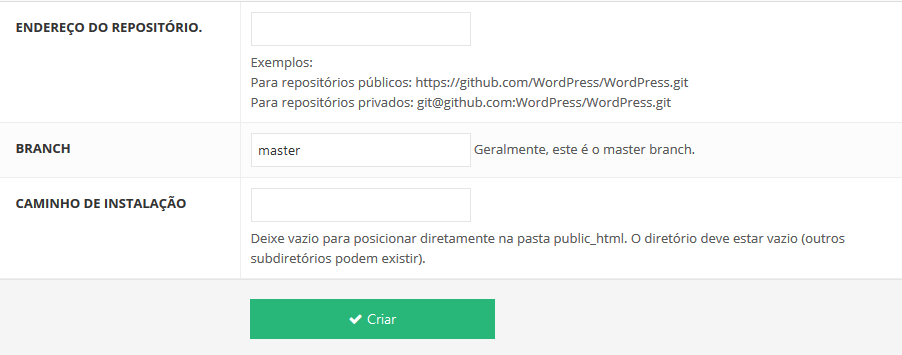
git clone /path/to/local/repository

Se você pretende verificar um repositório armazenado remotamente, use:

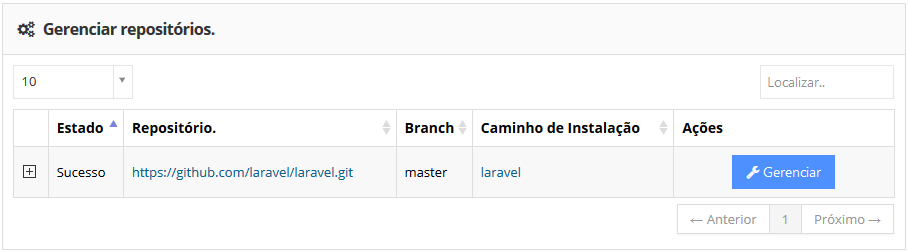
git clone user.name@host:/path/to/remote/repository



Se você tem uma conta na Hostinger, você pode facilmente clonar e gerenciar repositórios via **Painel de Controle**>**GIT**. Por exemplo, se você quer clonar um repositório GIT, basta digitar seu endereço, escolher um ramo e instalar o caminho, e clicar no botão criar.



Após a conclusão da criação, você poderá gerenciar seu repositório na mesma seção.



**Fluxo de trabalho – Tutorial Git**

Agora que o repositório está pronto, vamos falar sobre a estrutura que é mantida pelo GIT.

Cada repositório local consiste em três árvores: o **diretório de trabalho** que contém os arquivos reais; O **índice** que desempenha o papel de uma área de teste e o **HEAD** que é um ponteiro para o último *commit* feito pelo usuário.

Então, é assim que o fluxo de trabalho pode ser explicado: o usuário adiciona um arquivo ou alterações do diretório de trabalho para o índice (a área de teste) e uma vez revistos, o arquivo ou as alterações são finalmente comprometidos com o **HEAD**.

**Os comandos *Add* e *Commit:***

Alterações ou adições de arquivos propostas são adicionadas ao índice usando o comando *add*. Para adicionar qualquer arquivo, o comando é:

git add <nome\_do\_arquivo>

Se você está realmente confiante o suficiente para fazer essas mudanças no ***HEAD***, então você pode usar o comando *commit*:

git commit –m “Adicionar qualquer mensagem sobre o commit aqui”

Nota: Uma vez que o comando *commit* é executado (a partir do diretório de trabalho), o arquivo fica comprometido com o ***HEAD***, mas ainda não é enviado para o repositório remoto.

**Dando continuidade com as mudanças**

Depois de confirmar as alterações (e acreditar que elas estão prontas para serem enviadas para o repositório original), você pode usar o comando push.

Uma vez que o git push origin master é executado de dentro do diretório de trabalho, as mudanças presentes no **HEAD** são enviadas para o repositório remoto. No comando acima mencionado, o **master** pode ser alterado para o nome do *branch* ao qual você deseja que as alterações sejam comprometidas.

Se, no entanto, um repositório existente ainda não tiver sido clonado e você pretende estabelecer uma ligação entre o seu repositório e um servidor remoto, execute o seguinte comando:

git remote add origin <servidor>

Nota: Substitua <servidor> pelo endereço do servidor remoto.

Uma vez clonado, quaisquer alterações feitas serão aplicadas para o servidor pertinente.

***Branches***

Outra característica brilhante (mas avançada) do GIT é sua capacidade de permitir que desenvolvedores e gerentes de projeto criem vários ramos (*branches*) independentes dentro de um único projeto.

O objetivo principal de um *branch* é desenvolver novas funcionalidades, mantendo-os isolados uns dos outros. O *branch* padrão em qualquer projeto é sempre o ***master branch***. Tantos *branches* quanto necessários podem ser criados e eventualmente mesclados ao *master branch*.

Um novo *branch* pode ser criado usando o seguinte comando:

git checkout -b feature\_n \*

**feature\_n** é o nome do *branch.*

Se você deseja retornar ao *master* branch, o seguinte comando pode ser usado:

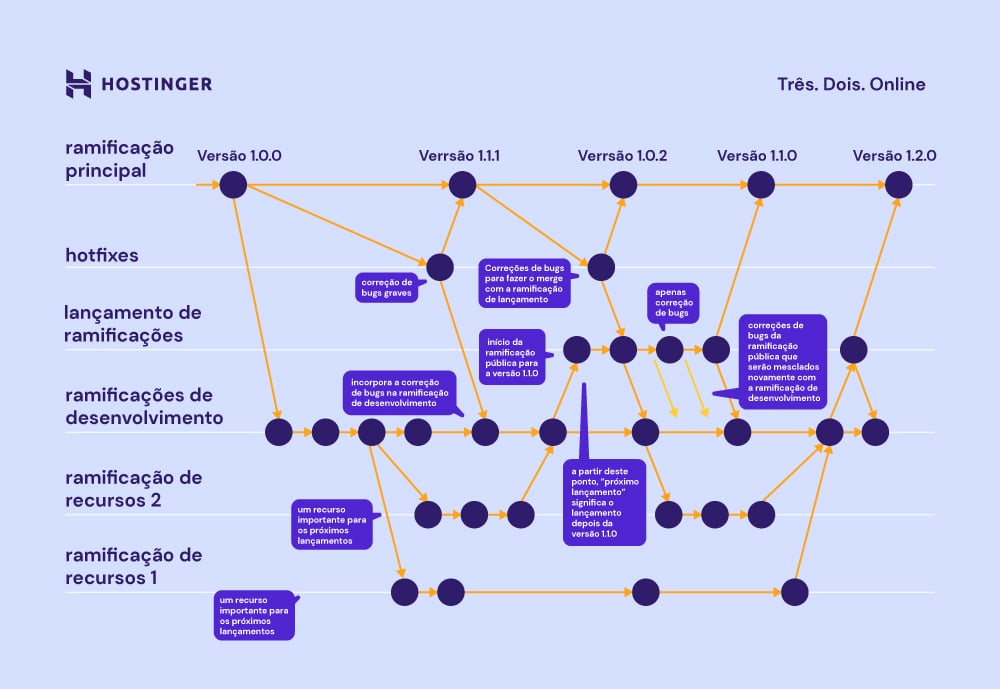
git checkout master

Qualquer *branch* pode ser excluído usando o seguinte comando:

git checkout -b feature\_n

Para tornar o *branch* disponível para outros usuários, você terá que movê-lo para o repositório remoto. Para fazer isso, use o seguinte comando:

git push origin feature\_n



**Atualizando e dando *merge***

Caso você queira atualizar seu diretório de trabalho local para o mais recente do repositório remoto, o simples comando git pull pode ser usado.

Para mesclar outro branch (dar um *merge*) no atualmente ativo, use: git merge feature\_n.

Se você der um *merge* ou *pull*, o GIT sempre tenta lidar com os conflitos por conta própria, mas as vezes não consegue. Em caso de falha devido a conflitos, o usuário tem que resolver os conflitos manualmente. Depois de editar os arquivos (para erradicar conflitos), marque-os como *merged* usando:

git add <nome.arquivo>

Se antes do *merge* você desejar visualizar as alterações, o seguinte comando pode ser executado:

git diff <nome\_branch\_origem> <nome\_branch\_alvo>

***Tagging***

Antes de lançar atualizações/alterações de software, é sempre recomendado criar tags. Para fazer isso no GIT, use o seguinte comando:

git tag 1.1.0 1c2d2d56fa

O **1c2d2d56fa** no comando acima refere-se aos primeiros 10 caracteres do **commit-id** que é referenciado com a tag. O ID de *commit* pode ser encontrado no log.

***Log***

O histórico do repositório pode ser estudado através do log. O comando git log recupera as informações. Para recuperar os *commits* feitos por um único usuário, você pode usar:

git log --author =Smith

Uma versão compactada do log (um *commit* por linha) pode ser visualizada usando:

git log --pretty=oneline

Para exibir somente os arquivos que foram alterados:

git log --name-status

**Substituindo alterações locais**

Se você acabou fazendo bagunça e precisa reverter as alterações feitas em qualquer arquivo, faça isso usando o seguinte comando:

git checkout -- <nomedoarquivo>

Isso substituirá as alterações da árvore de trabalho pelos últimos dados presentes no **HEAD**. Quaisquer alterações que já tenham sido adicionadas ao índice não serão prejudicadas.

Por outro lado, se todas as alterações/*commits* locais devem ser eliminados e o *master branch* local for necessário para apontar para o histórico mais recente do servidor, execute os seguintes comandos:

git fetch origin

git reset --hard origin/master

**Conclusão – Tutorial Git**

No mundo de projetos de software, é sempre reconfortante saber que alguém está cuidando de toda a gestão de código para você.

Nosso **GIT Tutorial para iniciantes** irá ajudar qualquer desenvolvedor para começar a utilizar o GIT, que é um rigoroso (e muito útil) sistema de controle de versão com uma infinidade de recursos. Se quiser saber mais sobre comandos GIT, consulte nosso [**tutorial**](https://www.hostinger.com.br/tutoriais/comandos-basicos-de-git/#gref).