**GIT**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Desenvolva suas habilidades nos padrões da digitalimobi

**Apresentação**

A Digital Imobi é uma agência focada em estratégias de marketing digital para o mercado imobiliário.

Os produtos foram desenhados pensando nas imobiliárias, construtoras e corretores para fortalecer sua presença digital e consequentemente aumentar seus leads, conversões e negócios.

**ÍNDICE**

|  |  |  |  |
| --- | --- | --- | --- |
| Principais Comandos |  |  |  |

**LINKS**

Download Git: <http://git-scm.com>

Tourtouse Git: <https://code.google.com/p/tortoisegit>

Documentação completa: <https://git-scm.com/book/pt-br/v1/Git-Essencial-Gravando-Altera%C3%A7%C3%B5es-no-Reposit%C3%B3rio>

**COMANDOS BÁSICOS**

* git – mostra os comandos no cmd
* git init – Inicializar (abrir a pasta)
* git status – Status da pasta
* git add <!-- public\_html -->
* git commit –m “*Coment: Iniciando o versionamento da pasta*” – Comitar
* git log – Log dos arquivos
* git config user.name
* Utilizar o TAB

**PRINCIPAIS COMANDOS**

### [Identidade](https://git-scm.com/book/pt-br/v1/Primeiros-passos-Configura%C3%A7%C3%A3o-Inicial-do-Git#Sua-Identidade)

A primeira coisa que você deve fazer quando instalar o Git é definir o seu nome de usuário e endereço de e-mail. Isso é importante porque todos os commits no Git utilizam essas informações, e está imutavelmente anexado nos commits que você realiza:

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

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

### [Ferramenta de Diff](https://git-scm.com/book/pt-br/v1/Primeiros-passos-Configura%C3%A7%C3%A3o-Inicial-do-Git#Sua-Ferramenta-de-Diff)

Outra opção útil que você pode querer configurar é a ferramente padrão de diff utilizada para resolver conflitos de merge (fusão). Digamos que você queira utilizar o vimdiff:

$ git config --global merge.tool vimdiff

Git aceita kdiff3, tkdiff, meld, xxdiff, emerge, vimdiff, gvimdiff, ecmerge e opendiff como ferramentas válidas para merge.

### [Verificando Suas Configurações](https://git-scm.com/book/pt-br/v1/Primeiros-passos-Configura%C3%A7%C3%A3o-Inicial-do-Git#Verificando-Suas-Configura%C3%A7%C3%B5es)

Caso você queira verificar suas configurações, você pode utilizar o comando git config --list para listar todas as configurações que o Git encontrar naquele momento:

$ git config --list

user.name=Scott Chacon

user.email=schacon@gmail.com

color.status=auto

color.branch=auto

color.interactive=auto

color.diff=auto

...

### Ajuda

Caso você precise de ajuda usando o Git, exitem três formas de se obter ajuda das páginas de manual (manpage) para quaisquer comandos do Git:

$ git help <verb>

$ git <verb> --help

$ man git-<verb>

Por exemplo, você pode obter a manpage para o comando config executando

$ git help config

### [Inicializando um Repositório em um Diretório Existente](https://git-scm.com/book/pt-br/v1/Git-Essencial-Obtendo-um-Reposit%C3%B3rio-Git#Inicializando-um-Reposit%C3%B3rio-em-um-Diret%C3%B3rio-Existente)

Caso você esteja iniciando o monitoramento de um projeto existente com Git, você precisa ir para o diretório do projeto e digitar

$ git init

Isso cria um novo subdiretório chamado .git que contem todos os arquivos necessários de seu repositório — um esqueleto de repositório Git. Neste ponto, nada em seu projeto é monitorado.

Caso você queira começar a controlar o versionamento dos arquivos existentes (diferente de um diretório vazio), você provavelmente deve começar a monitorar esses arquivos e fazer um commit inicial. Você pode realizar isso com poucos comandos git add que especificam quais arquivos você quer monitorar, seguido de um commit:

$ git add \*.c

$ git add README

$ git commit -m 'initial project version'

### [Clonando um Repositório Existente](https://git-scm.com/book/pt-br/v1/Git-Essencial-Obtendo-um-Reposit%C3%B3rio-Git#Clonando-um-Reposit%C3%B3rio-Existente)

Caso você queira copiar um repositório Git já existente — por exemplo, um projeto que você queira contribuir — o comando necessário é git clone.

Você clona um repositório com git clone [url]. Por exemplo, caso você queria clonar a biblioteca Git do Ruby chamada Grit, você pode fazê-lo da seguinte forma:

$ git clone git://github.com/schacon/grit.git

Isso cria um diretório chamado grit, inicializa um diretório .gitdentro deste, obtém todos os dados do repositório e verifica a cópia atual da última versão. Se você entrar no novo diretório grit, você verá todos os arquivos do projeto nele, pronto para serem editados ou utilizados. Caso você queira clonar o repositório em um diretório diferente de grit, é possível especificar esse diretório utilizando a opção abaixo:

$ git clone git://github.com/schacon/grit.git mygrit

Este comando faz exatamente a mesma coisa que o anterior, mas o diretório alvo será chamado mygrit.

O Git possui diversos protocolos de transferência que você pode utilizar. O exemplo anterior utiliza o protocolo git://, mas você também pode ver http(s):// ou user@server:/path.git, que utilizam o protocolo de transferência SSH.

### [Verificando o Status de Seus Arquivos](https://git-scm.com/book/pt-br/v1/Git-Essencial-Gravando-Altera%C3%A7%C3%B5es-no-Reposit%C3%B3rio#Verificando-o-Status-de-Seus-Arquivos)

A principal ferramenta utilizada para determinar quais arquivos estão em quais estados é o comando git status. Se você executar este comando diretamente após uma clonagem, você deverá ver algo similar a isso:

$ git status

# On branch master

nothing to commit, working directory clean

Isso significa que você tem um diretório de trabalho limpo — em outras palavras, não existem arquivos monitorados e modificados.

### [Monitorando Novos Arquivos](https://git-scm.com/book/pt-br/v1/Git-Essencial-Gravando-Altera%C3%A7%C3%B5es-no-Reposit%C3%B3rio#Monitorando-Novos-Arquivos)

Para passar a monitorar um novo arquivo, use o comando git add. Para monitorar o arquivo README, você pode rodar isso:

$ git add README

Se você rodar o comando status novamente, você pode ver que o seu arquivo README agora está sendo monitorado e está selecionado:

$ git status

# On branch master

# Changes to be committed:

# (use "git reset HEAD <file>..." to unstage)

#

# new file: README

#

### [Ignorando Arquivos](https://git-scm.com/book/pt-br/v1/Git-Essencial-Gravando-Altera%C3%A7%C3%B5es-no-Reposit%C3%B3rio#Ignorando-Arquivos)

Muitas vezes, você terá uma classe de arquivos que não quer que o Git automaticamente adicione ou mostre como arquivos não monitorados. Normalmente estes arquivos são gerados automaticamente como arquivos de log ou produzidos pelo seu sistema de build. Nestes casos, você pode criar um arquivo contendo uma lista de padrões a serem checados chamado .gitignore. Eis um exemplo de arquivo .gitignore:

$ cat .gitignore

\*.[oa]

\*~

A primeira linha fala para o Git ignorar qualquer arquivo finalizado em .o ou .a — arquivos objetos e archive (compactados) que devem ter produto da construção (build) de seu código. A segunda linha fala para o Git ignorar todos os arquivos que terminam com um til (~), os quais são utilizados por muitos editores de texto como o Emacs para marcar arquivos temporários. Você também pode incluir um diretório log, tmp ou pid; documentação gerada automaticamente; e assim por diante. Configurar um arquivo .gitignore antes de começar a trabalhar, normalmente é uma boa ideia, pois evita que você commite acidentalmente arquivos que não deveriam ir para o seu repositório Git.

As regras para os padrões que você pode pôr no arquivo .gitignore são as seguintes:

* Linhas em branco ou iniciando com # são ignoradas.
* Padrões glob comuns funcionam.
* Você pode terminar os padrões com uma barra (/) para especificar diretórios.
* Você pode negar um padrão ao iniciá-lo com um ponto de exclamação (!).

Padrões glob são como expressões regulares simples que os shells usam. Um asterísco (\*) significa zero ou mais caracteres; [abc] condiz com qualquer um dos caracteres de dentro dos colchetes (nesse caso, a, b, ou c); um ponto de interrogação (?) condiz com um único caractere; e os caracteres separados por hífen dentro de colchetes ([0-9]) condizem à qualquer um dos caracteres entre eles (neste caso, de 0 à 9).

Segue um outro exemplo de arquivo .gitignore:

# um comentário - isto é ignorado

# sem arquivos terminados em .a

\*.a

# mas rastreie lib.a, mesmo que você tenha ignorado arquivos terminados em .a acima

!lib.a

# apenas ignore o arquivo TODO na raiz, não o subdiretório TODO

/TODO

# ignore todos os arquivos no diretório build/

build/

# ignore doc/notes.txt mas, não ignore doc/server/arch.txt

doc/\*.txt

### [Visualizando Suas Mudanças Selecionadas e Não Selecionadas](https://git-scm.com/book/pt-br/v1/Git-Essencial-Gravando-Altera%C3%A7%C3%B5es-no-Reposit%C3%B3rio#Visualizando-Suas-Mudan%C3%A7as-Selecionadas-e-N%C3%A3o-Selecionadas)

Se o comando git status for muito vago — você quer saber exatamente o que você alterou, não apenas quais arquivos foram alterados — você pode utilizar o comando git diff. Nós trataremos o comando git diff em mais detalhes posteriormente; mas provavelmente você vai utilizá-lo com frequência para responder estas duas perguntas: O que você alterou, mas ainda não selecionou (stage)? E o que você selecionou, que está para ser commitado? Apesar do comando git status responder essas duas perguntas de maneira geral, o git diff mostra as linhas exatas que foram adicionadas e removidas — o patch, por assim dizer.

Este comando compara o que está no seu diretório de trabalho com o que está na sua área de seleção (staging). O resultado te mostra as mudanças que você fez que ainda não foram selecionadas.

Se você quer ver o que selecionou que irá no seu próximo commit, pode utilizar git diff --cached. (Nas versões do Git 1.6.1 e superiores, você também pode utilizar git diff --staged, que deve ser mais fácil de lembrar.) Este comando compara as mudanças selecionadas com o seu último commit

### [Fazendo Commit de Suas Mudanças](https://git-scm.com/book/pt-br/v1/Git-Essencial-Gravando-Altera%C3%A7%C3%B5es-no-Reposit%C3%B3rio#Fazendo-Commit-de-Suas-Mudan%C3%A7as)

Agora que a sua área de seleção está do jeito que você quer, você pode fazer o commit de suas mudanças. Lembre-se que tudo aquilo que ainda não foi selecionado — qualquer arquivo que você criou ou modificou que você não tenha rodado o comando git add desde que editou — não fará parte deste commit. Estes arquivos permanecerão como arquivos modificados em seu disco. Neste caso, a última vez que você rodou git status, viu que tudo estava selecionado, portanto você está pronto para fazer o commit de suas mudanças. O jeito mais simples é digitar git commit:

$ git commit

Você pode digitar sua mensagem de commit junto ao comanto commit ao especificá-la após a flag -m, assim:

$ git commit -m "Story 182: Fix benchmarks for speed"

### Visualizando o Histórico de Commits

Depois que você tiver criado vários commits, ou se clonou um repositório com um histórico de commits existente, você provavelmente vai querer ver o que aconteceu. A ferramente mais básica e poderosa para fazer isso é o comando git log.

Estes exemplos usam um projeto muito simples chamado simplegit, que eu frequentemente uso para demonstrações. Para pegar o projeto, execute:

git clone git://github.com/schacon/simplegit-progit.git

Uma das opções mais úteis é -p, que mostra o diff introduzido em cada commit. Você pode ainda usar -2, que limita a saída somente às duas últimas entradas.

## Trabalhando com Remotos

Para ser capaz de colaborar com qualquer projeto no Git, você precisa saber como gerenciar seus repositórios remotos. Repositórios remotos são versões do seu projeto que estão hospedados na Internet ou em uma rede em algum lugar. Você pode ter vários deles, geralmente cada um é somente leitura ou leitura/escrita pra você. Gerenciar repositórios remotos inclui saber como adicionar repositório remoto, remover remotos que não são mais válidos, gerenciar vários branches remotos e defini-los como monitorados ou não, e mais. Nesta seção, vamos cobrir essas habilidades de gerenciamento remoto.

### [Exibindo Seus Remotos](https://git-scm.com/book/pt-br/v1/Git-Essencial-Trabalhando-com-Remotos#Exibindo-Seus-Remotos)

Para ver quais servidores remotos você configurou, você pode executar o comando git remote. Ele lista o nome de cada remoto que você especificou. Se você tiver clonado seu repositório, você deve pelo menos ver um chamado origin — esse é o nome padrão que o Git dá ao servidor de onde você fez o clone:

$ git clone git://github.com/schacon/ticgit.git

Initialized empty Git repository in /private/tmp/ticgit/.git/

remote: Counting objects: 595, done.

remote: Compressing objects: 100% (269/269), done.

remote: Total 595 (delta 255), reused 589 (delta 253)

Receiving objects: 100% (595/595), 73.31 KiB | 1 KiB/s, done.

Resolving deltas: 100% (255/255), done.

$ cd ticgit

$ git remote

origin

Você também pode especificar -v, que mostra a URL que o Git armazenou para o nome do remoto:

$ git remote -v

origin git://github.com/schacon/ticgit.git (fetch)

origin git://github.com/schacon/ticgit.git (push)

Se você tem mais de um remoto, o comando lista todos. Por exemplo, meu repositório Grit se parece com isso.

$ cd grit

$ git remote -v

bakkdoor git://github.com/bakkdoor/grit.git

cho45 git://github.com/cho45/grit.git

defunkt git://github.com/defunkt/grit.git

koke git://github.com/koke/grit.git

origin git@github.com:mojombo/grit.git

Isso significa que podemos puxar contribuições de qualquer um desses usuários muito facilmente. Mas note que somente o remoto origin é uma URL SSH, sendo o único pra onde eu posso fazer o push (vamos ver o motivo disso no Capítulo 4).

### [Adicionando Repositórios Remotos](https://git-scm.com/book/pt-br/v1/Git-Essencial-Trabalhando-com-Remotos#Adicionando-Reposit%C3%B3rios-Remotos)

Eu mencionei e dei algumas demonstrações de adição de repositórios remotos nas seções anteriores, mas aqui está como fazê-lo explicitamente. Para adicionar um novo repositório remoto no Git com um nome curto, para que você possa fazer referência facilmente, execute git remote add [nomecurto] [url]:

$ git remote

origin

$ git remote add pb git://github.com/paulboone/ticgit.git

$ git remote -v

origin git://github.com/schacon/ticgit.git

pb git://github.com/paulboone/ticgit.git

Agora você pode usar a string pb na linha de comando em lugar da URL completa. Por exemplo, se você quer fazer o fetch de todos os dados de Paul que você ainda não tem no seu repositório, você pode executar git fetch pb:

$ git fetch pb

remote: Counting objects: 58, done.

remote: Compressing objects: 100% (41/41), done.

remote: Total 44 (delta 24), reused 1 (delta 0)

Unpacking objects: 100% (44/44), done.

From git://github.com/paulboone/ticgit

\* [new branch] master -> pb/master

\* [new branch] ticgit -> pb/ticgit

O branch master de Paul é localmente acessível como pb/master — você pode fazer o merge dele em um de seus branches, ou fazer o check out de um branch local a partir deste ponto se você quiser inspecioná-lo.

### [Adicionando Repositórios Remotos](https://git-scm.com/book/pt-br/v1/Git-Essencial-Trabalhando-com-Remotos#Adicionando-Reposit%C3%B3rios-Remotos)

Eu mencionei e dei algumas demonstrações de adição de repositórios remotos nas seções anteriores, mas aqui está como fazê-lo explicitamente. Para adicionar um novo repositório remoto no Git com um nome curto, para que você possa fazer referência facilmente, execute git remote add [nomecurto] [url]:

$ git remote

origin

$ git remote add pb git://github.com/paulboone/ticgit.git

$ git remote -v

origin git://github.com/schacon/ticgit.git

pb git://github.com/paulboone/ticgit.git

### [Pushing Para Seus Remotos](https://git-scm.com/book/pt-br/v1/Git-Essencial-Trabalhando-com-Remotos#Pushing-Para-Seus-Remotos)

Quando o seu projeto estiver pronto para ser compartilhado, você tem que enviá-lo para a fonte. O comando para isso é simples: git push [nome-remoto] [branch]. Se você quer enviar o seu branch master para o servidor origin (novamente, clonando normalmente define estes dois nomes para você automaticamente), então você pode rodar o comando abaixo para enviar o seu trabalho para o sevidor:

$ git push origin master

Este comando funciona apenas se você clonou de um servidor que você têm permissão para escrita, e se mais ninguém enviou dados no meio tempo.

### [Removendo e Renomeando Remotos](https://git-scm.com/book/pt-br/v1/Git-Essencial-Trabalhando-com-Remotos#Removendo-e-Renomeando-Remotos)

Se você quiser renomear uma referência, em versões novas do Git você pode rodar git remote rename para modificar um apelido de um remoto. Por exemplo, se você quiser renomear pb para paul, você pode com git remote rename:

$ git remote rename pb paul

$ git remote

origin

paul

É válido mencionar que isso modifica também os nomes dos branches no servidor remoto. O que costumava ser referenciado como pb/master agora é paul/master.