**Iniciando o Git**

Entre no diretório que deseja controlar a versão e inicie o Git assim:

1. git init

Feito isso, seus arquivos ainda não estão sendo versionados, mas eles estão esperando para serem adicionados no estágio de controle. Para fazer isso digite o comando

1. git add nome-do-arquivo-incluindo-extensão

Se você precisa adicionar todos os arquivos do diretório, basta digitar:

1. git add .

Saber o status do projeto é importante. Com o comando abaixo você consegue ver quais arquivos estão fora do controle, quais foram modificados e estão esperando por uma descrição de modificação etc:

1. git status

Voltando ao estágio anterior do adicionamento:

1. git reset HEAD nome-do-arquivo

Commit – Comitando:

1. git commit -m "Mensagem do commit"

Adicionando e comitando ao mesmo tempo:

1. git commit -a -m "Mensagem do commit"

**Voltando commits a versões anteriores**

Voltar um commit:

1. git reset HEAD~1

Voltar dois commits:

1. git reset HEAD~2

Voltando um commit e deixando o arquivo no estagio anterior:

1. git reset HEAD~1 --soft

Voltando um commit e excluindo o arquivo, deixando no estágio anterior:

1. git reset HEAD~1 --hard

Verificando o histórico de commits:

1. git log

Verificando o que foi mudado, diferença entre um arquivo e outro:

1. git log -p

Verificando os 2 últimos commits:

1. git log -p -2

Mostrando as estatísticas de todos os commits:

1. git log --stat

Mostrando todos os commits, cada um em uma linha:

1. git log --pretty=oneline

Mostrando todos os commits dos últimos 2 dias até o momento atual

1. git log --since=2.days

Criando um branch – uma ramificação

1. git checkout -b nome-do-branch

Verificando em que branch você está

1. git branch

Voltando para o branch master

1. git checkout master

**Jogando o branch criado no branch master**

Entre como branch master:

1. git merge nome-do-branch-que-foi-criado

**Grudando o branch criado no branch master sem o commit**

Somente localmente – localhost, entre como branch master:

1. git rebase nome-do-branch-que-foi-criado

Removendo um branch:

1. git branch -D nome-do-branch

Vendo branchs remotos:

1. git branch -a

Mostrando o início do hash, quem comitou, quanto tempo atrás, mensagem: descrição do commit:

1. git log --pretty=format: "%h - %an, %ar : %s"

Deletando arquivos:

1. git rm nome-do-arquivo

Deletando todos os aquivos removidos ao mesmo tempo:

1. git ls-files --deleted | xargs git rm

**Ignorando arquivos**

Existem alguns arquivos que muito provavelmente você não vai precisar versionar, como por exemplo os arquivos de cache do SASS, arquivos de configuração e etc. Nesse caso você precisa fazer com que o controle de versão ignore estes arquivos. Para tanto, crie um arquivo chamado **.gitignore**. Feito isso, dentro deste arquivo, digite o nome ou o endereço das pastas que você quer ignorar. Um exemplo:

1. # See http://help.github.com/ignore-files/ for more about ignoring files.
2. #
3. # If you find yourself ignoring temporary files generated by your text editor
4. # or operating system, you probably want to add a global ignore instead:
5. # git config --global core.excludesfile ~/.gitignore\_global
7. # Ignore bundler config
8. /.bundle
10. # Ignore the build directory
11. /build
13. # Ignore Sass' cache
14. /.sass-cache
16. # Ignore .DS\_store file
17. .DS\_Store
18. .cache
19. .rvmrc
21. vendor/\*
23. .DS\_Store
25. # Vim
26. \*.swp
27. \*.swo
29. Gemfile.lock
30. .vagrant
31. Vagrantfile
33. # rbenv
34. .ruby-version
36. # Ignore deploy related files
37. deploy
39. Gemfile.lock

O arquivo **.gitignore** fica na raiz do projeto.

**Clonando e puxando alterações de projetos**

Clonando um projeto remoto:

1. git clone url-do-projeto

Fazendo um clone de outros branchs:

1. git checkout -b nome-do-branch origin/ nome-do-branch

Trazendo, puxando as alterações feitas por outros usuários:

1. git pull origin master

Sincronizando tudo que está no repositório remoto:

1. git pull

Enviando o(s) projeto(s), arquivo(s) para o repositório:

1. git push origin master

Enviando um branch para o repositório:

1. git push origin nome-do-branch

**Tags**

As tags servem para marcar uma etapa. Imagine que você vai lançar uma versão, que resolve uma série de problemas. Você pode marcar aquela etapa criando uma tag. Assim fica simples de fazer qualquer rollback do projeto para uma tag específica em vez de voltar para um commit. Você sabe que tudo o que foi feito até aquela tag está funcionando.

Criando tags:

1. git tag versão-da-tag

Listando tags:

1. git tag -l

Enviando a tag para o repositório

1. git push origin master --tags

Removendo as tags criadas localmente:

1. git tag -d versão-da-tag

Removendo tag no repositório remoto:

1. git push origin :refs/tags/versão-da-tag

**Concluindo**

Se você quer continuar ou iniciar seus estudos com Git, indico o link do livro citado acima, é um ótimo começo, se tiver problemas com o inglês, encontrará várias versões em português.

O Akita fez um [screencast para quem está começando com Git](http://www.akitaonrails.com/2010/08/17/screencast-comecando-com-git). Vale a pena ver.

O pessoal da CodeSchool juntamente com o GitHub fizeram uma página exclusivamente para ensinar Git na prática. [Visite aqui](https://try.github.io/levels/1/challenges/1).

Há também a [documentação do Git](http://www.git-scm.com/) que é bastante completa.