**O que é Git?**

O Git é um sistema open-source de controle de versão utilizado pela grande maioria dos desenvolvedores atualmente. Com ele podemos criar todo histórico de alterações no código do nosso projeto e facilmente voltar para qualquer ponto para saber como o código estava naquela data.

Além disso, o Git nos ajuda muito a controlar o fluxo de novas funcionalidades entre vários desenvolvedores no mesmo projeto com ferramentas para análise e resolução de conflitos quando o mesmo arquivo é editado por mais de uma pessoa em funcionalidades diferentes.

**Pontos na história**

Tudo no Git é movido através dos pontos na história do projeto que são chamados de *commits*, esses pontos são formados por **conjuntos de alterações** em um ou mais arquivos e somados a um **descritivo** que resume as alterações nesse ponto.

De forma prática, pensando que tenhamos que desenvolver um sistema de login completo, nossos *commits* podem ficar dessa forma:

1. Configuração da estrutura do projeto
2. Estrutura da página de login
3. Estilos CSS da página de login
4. Estrutura da página de cadastro
5. Resolvido problema no login
6. Estilos CSS da página de cadastro

Veja que nossos *commits* descrevem exatamente as alterações que o código sofreu e além do título podemos detalhar ainda mais com um texto maior.

**Ramificações**

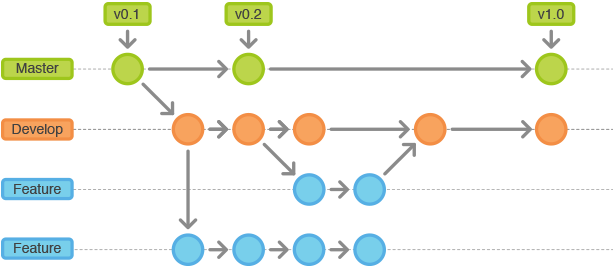
Imagine que você esteja trabalhando no meio de uma grande funcionalidade, pode levar até 2 meses para terminá-la. Em uma bela manhã de sol seu chefe resolve pedir urgentemente uma alteração na versão em produção da aplicação, ou seja, você não pode utilizar o código em que está trabalhando pois o mesmo possui features inacabadas.

As **ramificações**ou *branchs* no Git são formas de termos uma mesma versão do código sofrendo alterações e recebendo *commits* de diferentes fontes e inclusive por diferentes desenvolvedores.

Dessa forma, nós podemos manter um ramo para nossa funcionalidade que irá levar mais tempo e trabalhar em outro *branch* com a versão em produção para realizar alterações mais urgentes. E fica tranquilo, no fim de tudo o Git ainda vai nos ajudar a unir os códigos desses dois ramos de forma muito simpática.

Por padrão, você sempre está trabalhando em um ramo no Git, e mesmo quando você não cria um *branch*, o Git cria automaticamente um *branch*chamado **master** como **padrão**.

Na imagem abaixo podemos ver um exemplo de trabalho com vários ramos e *commits* aplicados. Veja que em alguns pontos da história os ramos são **unidos** para que as alterações de um ramo sejam aplicadas a outro.



Nesse caso, “master”, “Hotfix”, “Release”, “Develop” e os “Feature” são os ramos enquanto que os círculos são os *commits*. As caixas com v0.1, v0.2 e v1.0 são versões (conhecidas por *tags*) que foram pra versão em produção e **podem ser compostas por pontos na história de vários *branchs***.

**Github**

Legal, até agora falamos sobre algumas funcionalidades do Git mas tem um grande problema aí: como os outros desenvolvedores do time terão acesso a todo esse código e poderão também adicionar seus *branchs* e *commits*?

O **Github** é um serviço online de hospedagem de **repositórios** **Git** (como são chamados os projetos que utilizam Git). Com ele podemos manter todos nossos *commits* e ramos sincronizados entre os membros do time.

Além de servir como hospedagem, o Github possui muitas integrações com serviços que auxiliam no deploy da aplicação através de [integração contínua](https://blog.rocketseat.com.br/integracao-continua-ci-do-zero/).

**Na prática**

Agora que já conhecemos os principais conceitos do Git e Github, vamos entender **na prática** como eles funcionam. Inicie fazendo [download do Git](https://git-scm.com/downloads) na sua máquina.

Depois de instalado deve ser possível executar o comando git –version no seu terminal para identificar se tudo ocorreu bem.

Antes de continuar precisamos nos identificar no Git para que todos pontos na história tenham nossa assinatura. Execute os dois seguintes comandos com seu nome e e-mail respectivamente:

git config --global user.name "Seu nome"

git config --global user.email "seu-email@example.com"

Agora, em uma pasta vazia vamos indicar que utilizaremos Git nesse projeto com o comando git init. Esse comando irá criar uma pasta oculta chamada .git no seu projeto onde ficarão armazenados **todos os arquivos referentes à história** do projeto. Se você deletar essa pasta, todo histórico é **perdido**.

Você deve receber um retorno como esse:

Initialized empty Git repository in C:/Users/SEU\_USUARIO/projeto/.git/

Tudo pronto! Agora vamos criar nosso primeiro arquivo dentro dessa pasta. Abra seu editor de preferência e crie um arquivo chamado arquivo.txt na raiz do diretório.

Dentro desse arquivo, adicione o seguinte conteúdo: Primeira versão. De volta ao Terminal execute o comando git status, você deverá ter um retorno como o seguinte:

On branch master

No commits yet

Untracked files:

(use "git add <file>..." to include in what will be committed)

arquivo.txt

nothing added to commit but untracked files present (use "git add" to track)

O comando status exibe um **relatório desde o último ponto na história** e o que ele está nos dizendo agora é que estamos no *branch* master, não temos nenhum *commit* e temos um arquivo não rastreado pelo Git.

Um arquivo não rastreado pelo Git quer dizer que não estamos controlando suas versões e o mesmo não entrará no próximo *commit*. Para começarmos a rastreá-lo, vamos executar o comando git add arquivo.txt. Após isso, podemos rodar o status novamente e teremos um retorno diferente:

Changes to be committed:

(use "git rm --cached <file>..." to unstage)

new file: arquivo.txt

Agora podemos ver que o arquivo está em uma nova seção chamada “Alterações a serem comitadas”, ou seja, assim que criarmos um novo ponto na história, esse arquivo será adicionado a ele.

Mesmo se você alterar esse arquivo novamente **antes de criar** o *commit*, você precisará dar um *add* nele novamente para que essas novas alterações sejam reconhecidas pelo Git.

Agora com o arquivo adicionado, vamos criar nosso primeiro commit com o comando git commit -m "Primeira versão do projeto". Você receberá o seguinte retorno:

[master (root-commit) 646845c]: Primeira versão do projeto

1 file changed, 1 insertion (+)

create mode 100644 arquivo.txt

Pronto, **temos nosso primeiro ponto na história do projeto**!