Versionamento de código com o Git

O controle de versão, também conhecido como versionamento de código, é a prática de rastrear e gerenciar as alterações em um código de software. **É uma forma de administrar as mudanças que são feitas no código fonte do projeto e garantir mais segurança na transição de uma versão para a outra**.

1. Sistemas de Controle de Versão - VCS

Os **Sistemas de Controle de Versão (Version Control System - VCS)** são ferramentas de software que ajudam as equipes de pessoas desenvolvedoras a gerenciar as alterações no código-fonte ao longo do tempo, criando um histórico da evolução do projeto. Como os ambientes de desenvolvimento são muito dinâmicos, os sistemas de controle de versão ajudam as equipes a trabalharem de forma rápida e inteligente. Atualmente, a grande maioria das empresas trabalham dentro da cultura **DevOps**, e neste contexto, os Sistemas de Controle de Versão auxiliam a reduzir o tempo de desenvolvimento e aumentar as implementações bem-sucedidas.

DevOps é um composto de **Dev** (desenvolvimento) e **Ops** (operações), que na prática é a união de pessoas, processos e tecnologias para fornecer continuamente valor aos clientes.

Equipes que adotam a cultura, as práticas e as ferramentas de DevOps apresentam alto desempenho, criando produtos melhores, com mais rapidez, para maior satisfação do cliente.

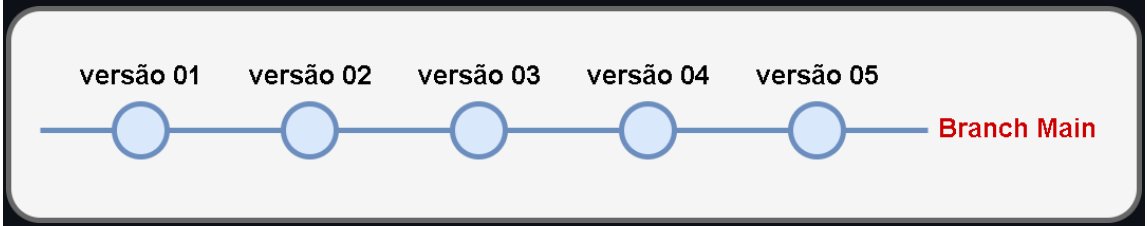
O software de controle de versão mantém o registro de todas as modificações efetuadas no código em um tipo especial de banco de dados. Se um erro for cometido, as pessoas desenvolvedoras podem **voltar no tempo** e **comparar versões anteriores do código para ajudar a corrigir o erro**, sem a necessidade de interromper o trabalhos dos demais colaboradores da equipe.

Antes dos sistemas de controle de versão, o processo de versionamento era manual e cada pessoa desenvolvedora tinha a sua forma de fazer, ou seja, não havia um padrão formal. A maioria utilizava a prática de fazer uma cópia da pasta do projeto e renomear a pasta acrescentando o numero da versão, como mostra a imagem abaixo:

O grande problema desta estratégia é que a pessoa desenvolvedora e o seu time não mantinham um histórico sobre até que ponto do desenvolvimento do projeto cada pasta contemplava, além de ter que lidar com um grande numero de pastas e arquivos repetidos. A única forma de saber o conteúdo de cada uma era abrindo uma por uma, até encontrar o que se procurava. Um processo lento, desgastante e que muitas vezes gerava mais retrabalho, principalmente quando uma das pastas eram alteradas ou apagadas acidentalmente. O controle de versão protege o código-fonte contra catástrofes e erro humano.

Outro ponto importante é que as pessoas desenvolvedoras atualmente trabalham em equipes e estão sempre escrevendo novas versões do código-fonte e modificando o código-fonte existente. Uma pessoa desenvolvedora pode estar trabalhando em um novo recurso, enquanto outra pessoa desenvolvedora corrige um bug não relacionado, e desta forma cada pessoa desenvolvedora pode fazer alterações em várias partes e em vários arquivos do projeto, inclusive alterações simultâneas em um mesmo arquivo.

O controle de versão ajuda as equipes a solucionarem esses tipos de problemas, rastreando cada alteração individual feita por cada pessoa desenvolvedora, ajudando a evitar conflitos com trabalhos simultâneos. O software de controle de versão é uma parte essencial no dia a dia das práticas profissionais para qualquer equipe de pessoas desenvolvedoras nos dias de hoje.



Observe que cada versão do código (commit) é um marco no processo de desenvolvimento, ou seja, um commit adiciona as alterações mais recentes do código-fonte no repositório, tornando essas alterações parte da revisão principal do repositório. Cada bolinha na imagem acima, representa uma versão consolidada do sistema, que já foi testada e validada.

É importante ressaltar que os commits nos sistemas de controle de versão são mantidos no repositório **indefinidamente**. Assim, quando outros usuários fizerem uma atualização ou check-out no repositório, eles receberão a última versão da aplicação. Os sistemas de controle de versão permitem reverter facilmente para versões anteriores. Nesse contexto, **um commit dentro de um sistema de controle de versão é protegido**, pois é facilmente revertido, mesmo após o commit ter sido aplicado.

1.2. Benefícios do Sistema de Controle de Versão

1. **Um histórico de alterações completo e a longo prazo de todos os arquivos.** Esse histórico mantém as alterações realizadas no código e também inclui o autor, a data e as notas escritas sobre o objetivo de cada alteração. Ter o histórico completo permite voltar às versões anteriores para ajudar na análise da causa raiz de bugs e é crucial para corrigir problemas nas versões mais antigas do software. Se o software estiver sempre sendo trabalhado, quase tudo poderá ser considerado uma "versão mais antiga" do software.
2. **Ramificação e mescla**. O trabalho simultâneo da equipe é certo, mas mesmo os indivíduos que trabalham sozinhos podem se beneficiar da capacidade de trabalhar em fluxos independentes de mudanças. Criar uma "ramificação" nas ferramentas do VCS mantém vários fluxos de trabalho independentes uns dos outros, além de oferecer a facilidade de mesclar esse trabalho de novo, permitindo que os desenvolvedores verifiquem se as alterações em cada ramificação não estão em conflito.
3. **Rastreabilidade**. Ser capaz de rastrear cada alteração feita no software e conectar ao software de gestão de projetos e rastreamento de bugs, além de ser capaz de anotar cada alteração com uma mensagem descrevendo o objetivo da mudança.

2. O que é o Git?

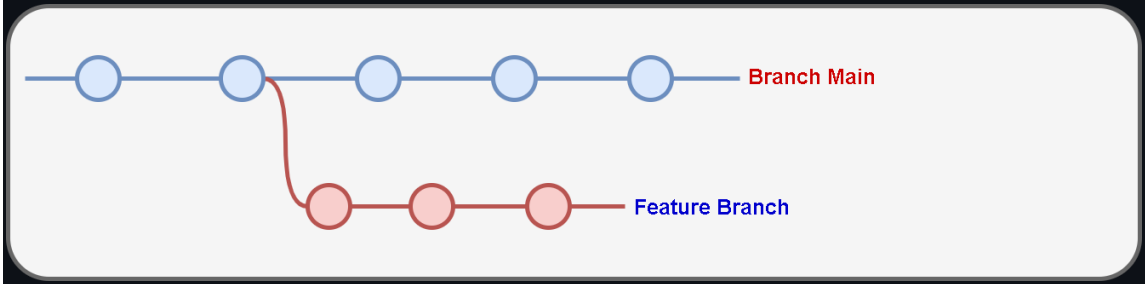
O **Git** é o Sistema de Controle de Versões mais popular e utilizado no mundo do Desenvolvimento, que possui a capacidade de documentar e armazenar todas as alterações que são feitas no código durante o processo de desenvolvimento e atualização do projeto.

O **Git** é um projeto de código aberto, maduro e com manutenção ativa e constante, que foi desenvolvido em 2005 por Linus Torvalds, o famoso criador do kernel do sistema operacional Linux. Um número impressionante de projetos de software depende do Git para controle de versão, incluindo projetos comerciais e de código-fonte aberto.

O Git é um sistema de versionamento com arquitetura distribuída (**DVCS - Distributed Version Control System**. Em vez de ter apenas um único repositório armazenando o histórico completo da versão do software, muito comum nos sistemas de controle de versão com arquitetura simultânea (**CVCS - Concurrent Version Control System**), como o **Subversion** por exemplo, no Git a cópia do repositório de cada pessoa desenvolvedora do código, também é um repositório local que pode conter o histórico completo de todas as alterações (locais e remotas).

2.1. Fluxo de trabalho de ramificação de recurso

Uma das maiores vantagens do Git são seus recursos de **branch (ramificação)**. Ao contrário dos sistemas de controle de versão centralizados, as branches do Git são simples e o processo de merge (fusão) entre as branches é muito fácil. Essas características facilitam o fluxo de trabalho e é um dos principais motivos da popularidade do Git. Na imagem abaixo, vemos um exemplo de um repositório composto por 2 branches:

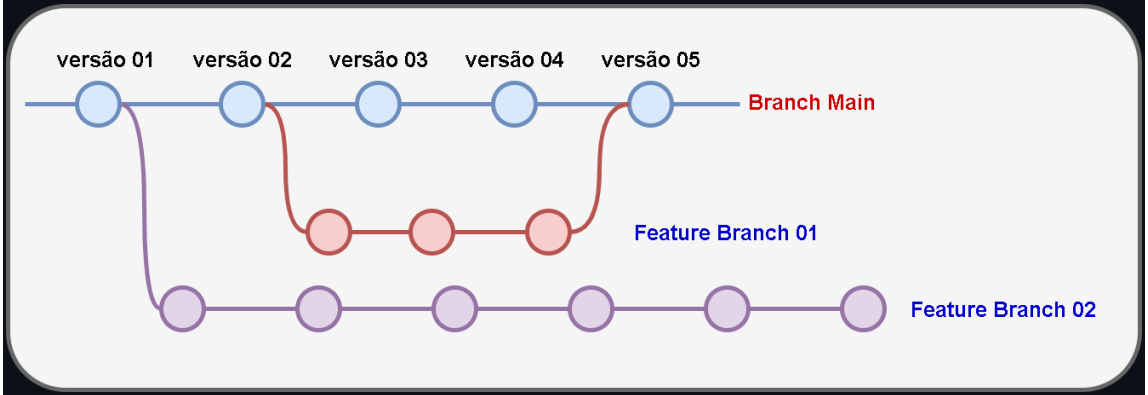


Ao criar um novo repositório, automaticamente é criada a **Branch Principal (Branch Main)**, que na prática é uma branch igual a qualquer outra. A única diferença é que ela é padrão em qualquer repositório git.

ATENÇÃO: ***Antiga mente, a Branch Main era nomeada como Branch Master (Mestre). O Github, buscando remover termos racistas, renomeou a "Branch Master" para "Branch Main". Observe que o Git ainda não adotou esta prática por padrão, mas existem formas de criar esta configuração na sua máquina local, como veremos na sessão prática.***

Branches de Recursos (Feature Branch) oferecem um ambiente isolado para cada alteração na base do código. Quando uma pessoa desenvolvedora inicia a implementação de um código, independente do tamanho do projeto, ele cria uma nova branch (feature branch), isolando o código em desenvolvimento do código testado e aprovado. Assim, é garantido que a branch principal (branch main) sempre contenha código de produção.

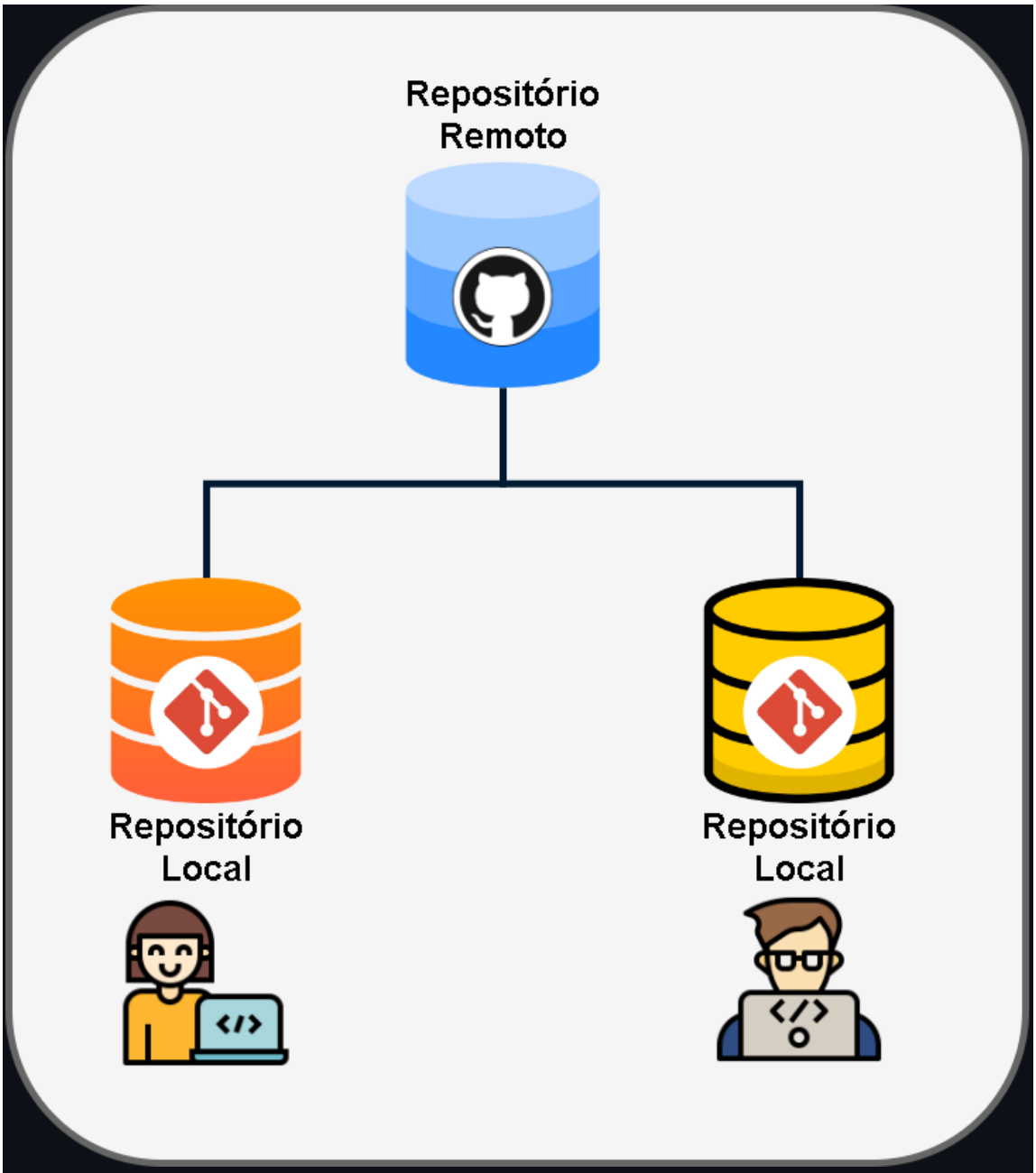
Uma branch de recurso é como se fosse um universo paralelo, que caminha de forma independente, sem interferir no ramo principal, a branch main. O merge, seria um commit onde a branch de recurso passa a fazer parte da branch principal, criando uma fusão entre as duas, como vemos na imagem abaixo:



Observe que a Feature Branch 01 foi criada a partir do commit versão 02 da Branch Main e foi unida (merge) com a Branch Main apenas no commit versão 05. Tudo o que aconteceu na Feature Branch 01 desde a sua criação até a fusão, a Branch Main desconhece, ou seja, a Feature Branch 01 é um universo paralelo desconhecido pela Branch Main até o merge.

2.2. Desenvolvimento distribuído

No Sistemas de controle de versão centralizados, cada desenvolvedor recebe uma cópia do repositório, que aponta para um único repositório central. O Git, por sua vez, é um sistema de controle de versão distribuído. Em vez de uma cópia do repositório, cada pessoa desenvolvedora tem o seu próprio repositório local, com o seu próprio histórico completo de commits. A imagem abaixo, exemplifica o Modelo Distribuído:



Ter um histórico local completo torna o Git muito mais rápido, já que você não precisa de uma conexão de rede para criar seus próprios commits e inspecionar as versões anteriores de um arquivo entre os commits.

O desenvolvimento distribuído também facilita a escalabilidade do projeto e do time de pessoas desenvolvedoras. Se alguém quebrar o branch de produção no sistema centralizado, ninguém consegue inserir suas alterações até a correção ser realizada. No Git, esse tipo de bloqueio não existe. Todos podem continuar trabalhando em seus próprios repositórios locais e depois enviar as alterações para o repositório remoto, assim que as correções forem aplicadas.

Assim como nas branches de recursos (feature branches), o desenvolvimento distribuído cria um ambiente mais confiável. Mesmo que um desenvolvedor

destrua seu repositório local, ele pode simplesmente clonar (copiar) o de outra pessoa ou do repositório remoto e começar de novo.

[Site Oficial: *Git*]

3. O que é o Github?

O GitHub é um serviço baseado em nuvem que hospeda um sistema de controle de versão distribuído (DVCS) chamado Git. Ele permite que os desenvolvedores colaborem e façam mudanças em projetos compartilhados enquanto mantêm um registro detalhado do seu progresso. O Github é uma espécie de rede social de projetos e código fonte, que permite hospedar e compartilhar os seus projetos.

Além do Github, existem outras alternativas como o **GitLab, o Bitbucket e o Subversion**:

Gitlab **Bitbucket** **Subversion**

O GitHub é uma excelente ferramenta para o trabalho em equipe, porque a plataforma online facilita a gestão do projeto. Essa ferramenta também está disponível para empresas: é conhecida como GitHub Enterprise. Nela você encontra uma forma inteligente de gerenciar o trabalho de uma equipe.

Além disso, a segurança é levada muito a sério, algo fundamental quando se trata de projetos digitais. Porém, o que chama realmente a atenção é o fato da equipe poder trabalhar ao mesmo tempo, de diversos lugares do mundo. Hoje em dia, para qualquer negócio, a automatização dos fluxos de trabalho é fundamental e o GitHub faz isso possível. Os recursos encontrados na plataforma auxiliam no desenvolvimento dos projetos, facilitando o crescimento da empresa como um todo.

Site Oficial: *Github*

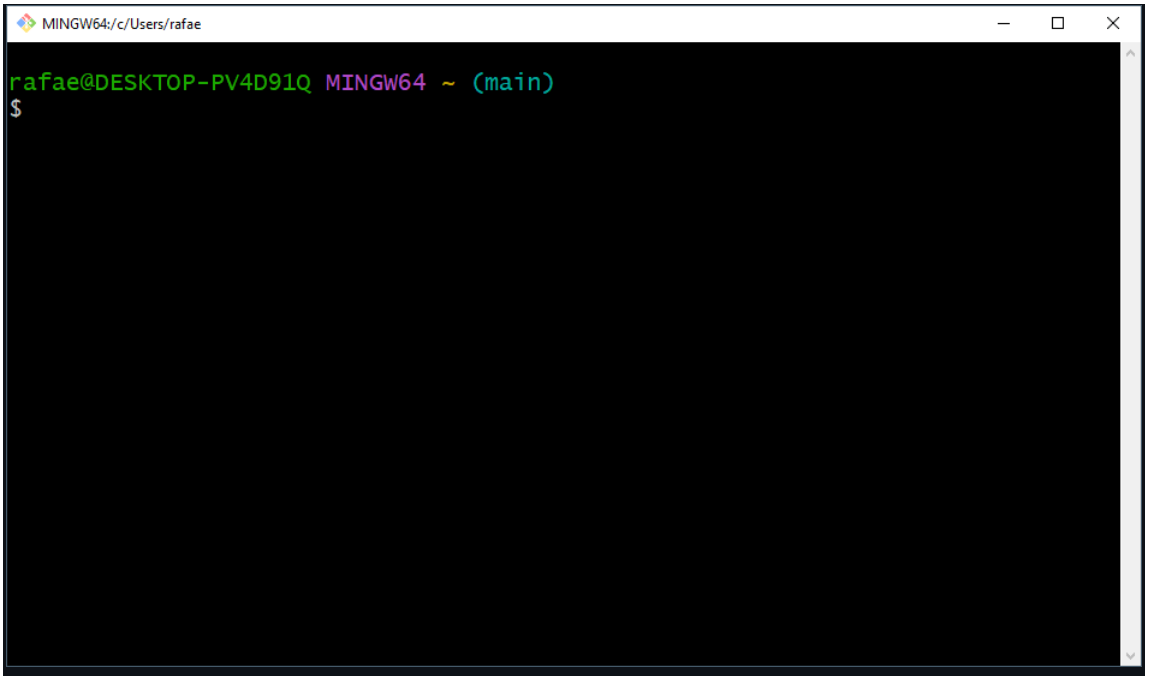
3.1. Portfólio no Github

Uma das formas de mostrar o seu trabalho para outras pessoas, em especial, empresas, é pelo seu portfólio. Pelo Github, você pode montar o seu portfólio com projetos que demonstrem as suas habilidades com código.

Não há nada melhor do que provar suas habilidades através do código. É como diria Linus Torvalds, criador do Linux: *"falar é fácil, me mostre o código"*.

4. Git Bash

O Git for Windows fornece uma emulação **BASH - Terminal do Linux**, o **Git Bash**, para executar o Git a partir da linha de comando do Linux. A emulação do BASH, no Git Bash, se comporta exatamente como o comando "git" em ambientes LINUX. Na imagem abaixo, temos a tela do Git Bash:



Observe que o prompt do Git Bash é diferente do CMD.

4.1. Comandos equivalentes no Gitbash

Como o **Git Bash** emula o BASH, ele utiliza os comandos padrão do Terminal Linux. Na tabela abaixo, vamos conhecer os comandos utilizados pelo Git Bash, que são equivalentes ao comandos do Terminal do Windows - CMD:

Comando Windows	Comando Gitbash	Exemplo
date **time**	date	*date*
cls	clear	*clear*
dir	ls	*ls* *ls -a* *ls -la*
cd	cd	*cd pasta01*
cd %USERPROFILE%	cd ~	*cd ~*
md	mkdir	*mkdir pasta02*
notepad	touch	*touch arquivo1.txt arquivo2.txt*

Comando Windows	Comando Gitbash	Exemplo
copy	cp	*cp arquivo1.txt pasta2*
xcopy	cp -r	*cp pasta1 pasta2*
ren	mv	*mv arquivo1.txt arquivo3.txt*
move	mv	*mv arquivo3.txt pasta2*
del	rm	*rm arquivo2.txt*
rd	rmdir	*rmdir pasta1*
rd /s	rm -rf	*rm -rf pasta2*