Desenvolvimento de Sistemas

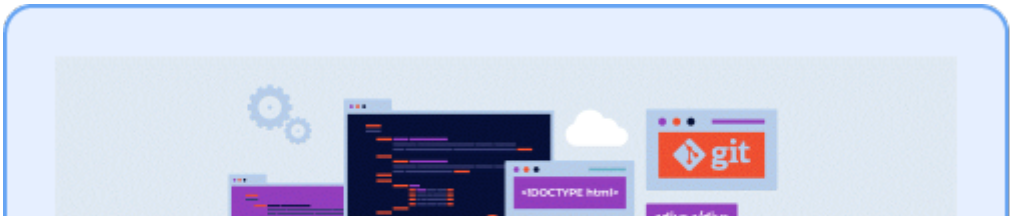
Versionamento de código: procedimentos, rotina de trabalho e aplicabilidade

O versionamento

O versionamento de código é o procedimento de controle e salvamento das diversas versões dos arquivos que compõem um sistema, ou seja, os diferentes arquivos de uma aplicação são salvos a cada alteração. Essa ação permite o controle e a recuperação dos dados a cada etapa do desenvolvimento do *software*.

Deve-se enxergar o desenvolvimento de um projeto como um processo que é dividido em várias etapas, e quando se fala em projetos relacionados a aplicações (*softwares*) não é diferente. Ao passo que um sistema vai sendo desenvolvido, diversas funcionalidades são implementadas, para cumprirmos os objetivos finais da aplicação, porém essas funcionalidades não são estáticas, ou seja, podem sofrer alterações.

Sendo assim, quando o programador executa melhorias, altera ou desenvolve uma nova funcionalidade, as versões anteriores dos arquivos do sistema devem estar guardadas de maneira organizada, prática e segura. Tal ação permite o controle e a recuperação das versões anteriores dos arquivos de um *software*, justificando, assim, a importância do versionamento de código em um projeto.



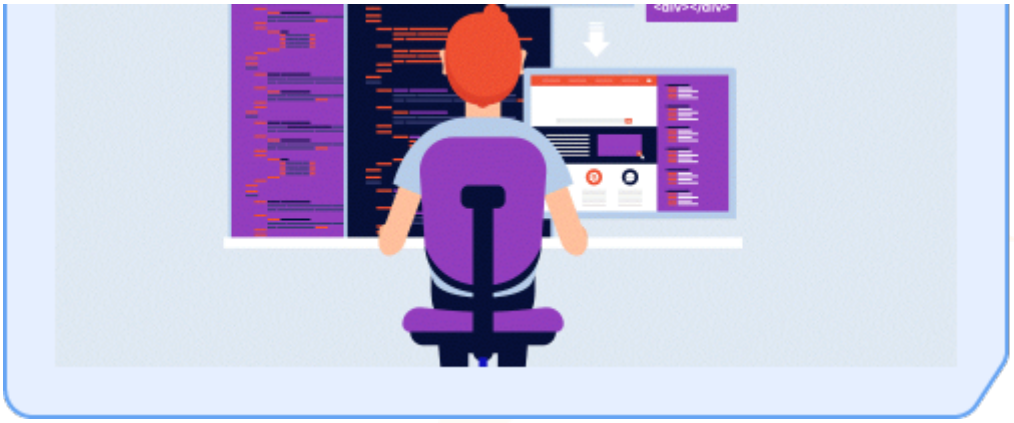


Figura 1 – Versionamento
Fonte: Ariane G. (2023)

Aplicabilidade

De maneira geral e específica, o versionamento de códigos em uma equipe evita, de forma eficiente, grande parte dos conflitos durante o desenvolvimento de uma aplicação. Nesse cenário, é possível entender **conflitos** como todos aqueles problemas que a manipulação de um mesmo arquivo por mais de uma pessoa pode causar, prejudicando de alguma maneira o processo de desenvolvimento de *software*. Porém, ainda há muitos outros benefícios obtidos pela utilização correta do versionamento do *software*.

A otimização do trabalho de uma equipe ocorre de maneira natural, a partir do momento em que os conflitos de trabalhar em um mesmo projeto simultaneamente são evitados. As alterações são mantidas de forma controlada e ágil, permitindo que um mesmo arquivo, por exemplo, seja alterado ao mesmo tempo por mais de um desenvolvedor, por meio de comandos que permitem esse controle, os quais serão vistos no decorrer deste conteúdo.

Outro ponto importante a ressaltar é a significativa melhoria na segurança do projeto, uma vez que é natural a ocorrência de erros no processo de alteração de arquivos e implementação de novas funcionalidades. Ao realizar o controle de

versão, é possível retornar o arquivo a uma versão anterior ao erro apresentado.

Para enxergar de forma prática como o versionamento de *software* pode ser relevante em uma equipe, durante as etapas do desenvolvimento de um projeto, pense no exemplo a seguir.

Você fez o *download* do repositório inicial do projeto e começou a executar alterações nos arquivos, implementando funcionalidades e melhorias. Porém, nesse mesmo período de tempo, outro desenvolvedor fez alterações nos mesmos arquivos que você e atualizou o servidor com esses novos arquivos. Quando você for enviar a sua versão, a ferramenta de controle de versão emitirá um alerta sobre o ocorrido e será possível mesclar as duas versões, mostrando as alterações e onde elas ocorreram. Assim, as mudanças poderão ser executadas com total liberdade e consciência do programador.

Veja, a seguir, um segundo exemplo, porém ainda utilizando o primeiro como base.

Pense que, após as alterações serem concluídas, os arquivos foram mesclados e implementados. Na etapa de testes, verificou-se que as melhorias feitas nesse arquivo estavam ocasionando um problema de segurança dentro do sistema, havendo, assim, a necessidade de retornar à versão anterior do arquivo em questão, ou seja, antes de essa funcionalidade ser implementada. Se o controle de versão do *software* foi feito de maneira correta, essa situação não será um problema, visto que o histórico do arquivo é acessível, assim como o retorno a versões anteriores.

Como ocorre o versionamento de código (procedimentos)

Por ser um processo contínuo que deve ocorrer durante toda a construção do sistema, a cada alteração significativa, o versionamento de código muitas vezes ocorre por meio do próprio ambiente de programação (IDE, sigla para *integrated*

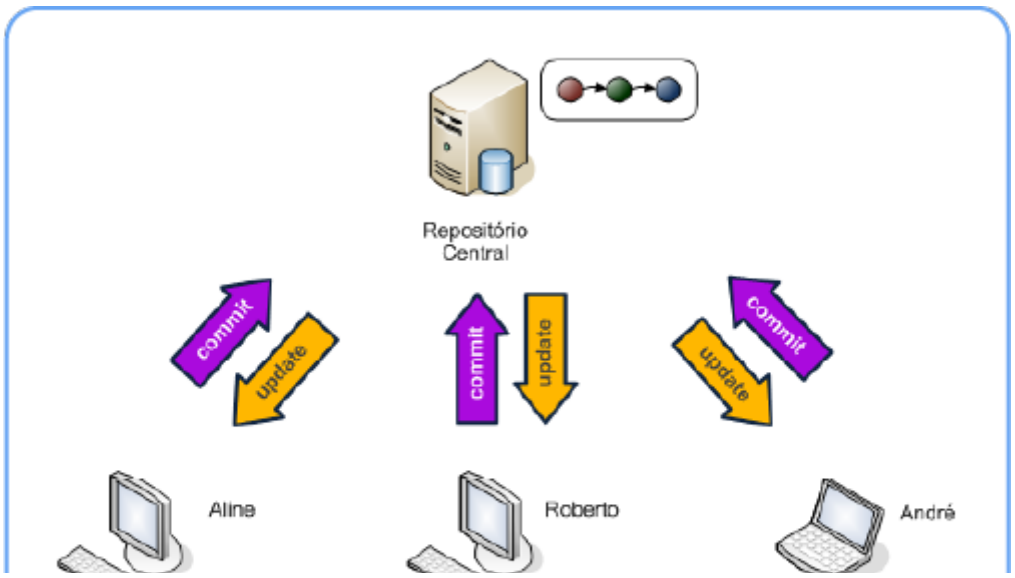
development environment, em português ambiente de desenvolvimento integrado), e grande parte das interfaces de programação mais utilizadas atualmente possibilita essa funcionalidade de versionamento integrado.

Vale ressaltar que o controle de versões também pode ser feito por meio do terminal do sistema operacional, ou por interface gráfica, se a ferramenta utilizada disponibilizar essa funcionalidade.

Porém, para realizar o versionamento de código, é necessário o uso de ferramentas específicas, as denominadas ferramentas de controle de versão de *software*. Atualmente, as ferramentas de controle de *software* podem ser classificadas em dois grupos, os sistemas centralizados e os sistemas distribuídos.

No **sistema centralizado**, o servidor ao qual o projeto está alocado é o único repositório central. É um sistema comum e popularmente utilizado em redes locais, visto que é capaz de suprir com eficiência as necessidades da maior parte das equipes. Esse sistema utiliza a arquitetura cliente-servidor, ou seja, primeiro os arquivos acessam o servidor para depois poderem comunicar-se. Costuma manter uma veloz e boa comunicação de dados.

A figura a seguir demonstra o controle de versão por meio de um sistema centralizado, no qual todas as requisições e alterações dependem da passagem por um repositório central.



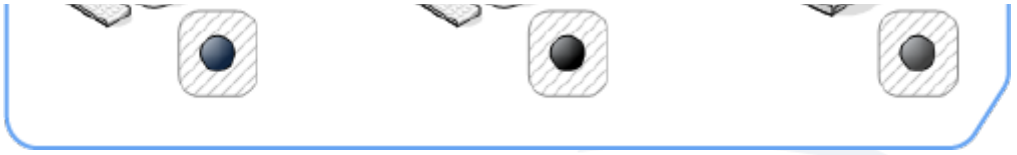


Figura 2 – Sistema centralizado
Fonte: Dias (2016)

Para conseguir realizar acesso ao repositório e outras operações pertinentes, os usuários precisam obrigatoriamente estar conectados. Cada desenvolvedor terá uma cópia em sua máquina para realizar as edições, mantendo as versões-base no servidor. Nessa estrutura, o desenvolvedor adicionará os arquivos inteiros e enviará para alterações direto no servidor. Esse tipo de sistema de controle de versão é mais indicado para equipes menores e que trabalham em uma rede local.

Já no **sistema distribuído**, os diretórios presentes nas máquinas de cada desenvolvedor trabalham de forma individual (como se cada um tivesse o próprio servidor). Cada usuário que for desenvolver código terá uma cópia completa do repositório central diretamente em seu ambiente local. Nesse sistema, os *desktops* podem fazer a comunicação entre si, apesar de, na maior parte das vezes, utilizarem um servidor central de envios para controle de fluxo. Esse servidor, muitas vezes, é oferecido diretamente pela ferramenta como um servidor remoto de hospedagem dos arquivos do sistema.

Embora seja necessária a conexão com a rede para realizar o envio das alterações e das adições para atualizar o projeto e compartilhar com outros desenvolvedores, os comandos de gerenciamento da cópia local já podem ser aplicados. Em outras palavras, cada *desktop* terá um repositório, e as operações como **commit** e **update** ocorrem de forma local. É possível também haver comunicação com outros *desktops*, para atualizações nos arquivos locais e também envio de alterações para um destino, por meio dos comandos **pull** e **push** (detalhados no decorrer deste conteúdo).

A figura a seguir mostra um exemplo de sistema distribuído em que a comunicação ocorre entre os diferentes *desktops* individualmente e também com um

servidor de controle de fluxo central.

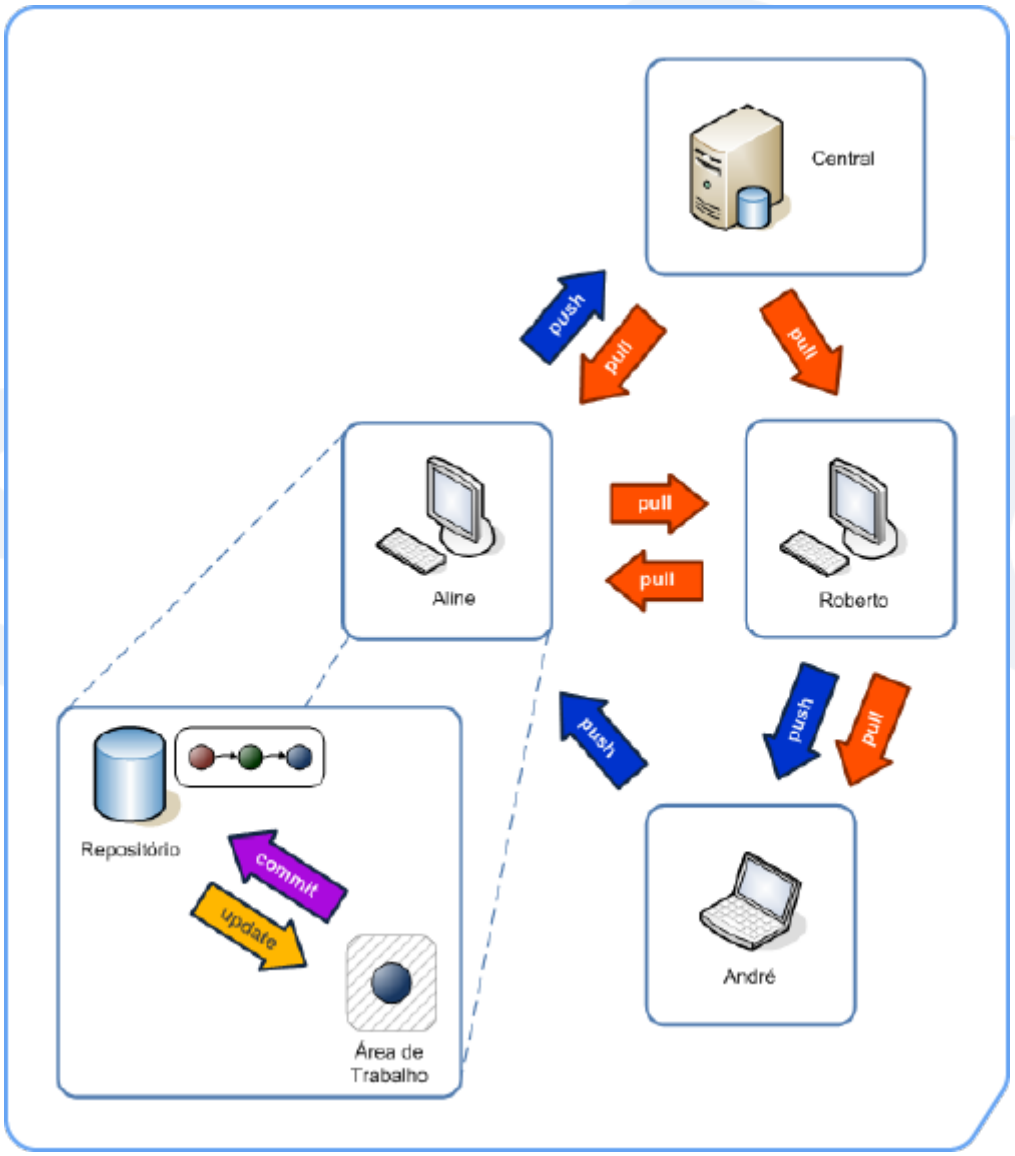


Figura 3 – Sistema distribuído

Fonte: Dias (2016)

Esse tipo de sistema é mais indicado para equipes com uma quantidade maior de desenvolvedores ou que não trabalhem em uma rede local.

Fluxo de trabalho com versionamento de *software*

Aqui você verá, de maneira prática, como ocorrem e quais são as etapas para

o desenvolvimento de um projeto com controle de versões.

1. Inicialmente, um novo repositório é criado (comando **init**), o qual servirá de ponto central para o armazenamento dos arquivos do projeto.
2. Em seguida, cada membro da equipe deve fazer uma cópia desse repositório para seu dispositivo local (comando **clone**). A partir desse momento, as alterações são feitas de forma local até serem enviadas para o servidor central (comando **push**).
3. Quando um desenvolvedor conclui uma tarefa, uma nova *branch* (ramificação) deve ser criada no seu repositório local.
4. Sempre que necessário, o desenvolvedor deve guardar as modificações com uma mensagem de alteração, salvando esse histórico em seu repositório local (comando **commit**).
5. Antes do início de cada nova tarefa, realização dos **commits** necessários, o desenvolvedor deve atualizar o repositório local, com os arquivos mais recentes (comando **pull**).
6. Na etapa final do processo, após concluir uma determinada tarefa, o desenvolvedor deve enviar as alterações realizadas para o servidor central (comando **push**).

Ferramentas de controle de versão

Como já citado anteriormente, o Git é a ferramenta de controle de versão mais utilizada e difundida mundialmente, porém é preciso mencionar que existem outras ferramentas, como CVS (Concurrent Version System), Subversion, entre outras, que apresentam proposta semelhante ao Git.

Aqui serão abordados recursos de versionamento de códigos, presentes nos comandos do Git, porém as funcionalidades desses recursos são comuns à maior parte das ferramentas de controle de versão.

Figura 4 – Ferramentas de versionamento de *software*
Fonte: Cortuk (2019)

Git

Lançado em 2005, o Git é um sistema gratuito e *open source* de controle de versão distribuído. Dotado de uma vasta comunidade, o Git atualmente é a mais difundida ferramenta de controle de versão disponível.

Essa ferramenta tem algumas peculiaridades. As principais são: necessidade de instalação para o funcionamento local; e não ter interface gráfica de forma nativa, assim a aplicação de seus recursos deve ser feita por meio do terminal do sistema operacional ou pelo recurso disponibilizado pelo IDE utilizado pelo programador.

Ao longo de sua carreira como programador, você poderá deparar-se com os termos Git e GitHub. Assim sendo, é interessante dissertar brevemente sobre a diferença entre eles.

É possível entender o Git como um *software* de controle de versão. Já o GitHub é visto como uma plataforma de versionamento, com alguns aspectos de redes sociais, em que os programadores podem criar perfis e publicar seus projetos, sejam eles de forma pública, sejam eles de forma privativa. Vale ressaltar a interação entre o Git e o GitHub, que ocorre quando as versões dos arquivos locais controladas pelo

Git são enviadas para o repositório remoto associado do GitHub.

Figura 5 – Logos do Git e do GitHub

Fonte: GitHub (c2023)

CVS

O CVS é uma plataforma de controle de versão consolidada no mercado há anos. É considerado um sistema de qualidade de produção, fazendo uso de uma arquitetura cliente-servidor, em que os desenvolvedores têm acesso a cópias controladas dos arquivos do projeto nos quais estão trabalhando. Um destaque do CVS é a sua simplicidade, facilitando o aprendizado por parte da equipe.

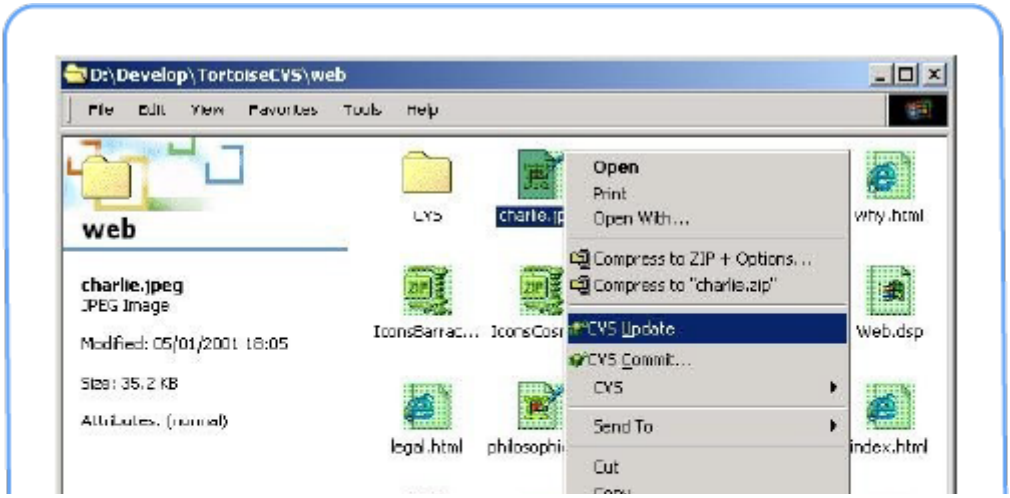


Figura 6 – Interface de opções integradas do CVS

Fonte: TortoiseCVS ([s. d.])

Subversion

O Subversion é uma ferramenta bastante utilizada no meio corporativo, principalmente por sua confiabilidade e sua segurança. Englobando a maior parte dos recursos do CVS, o Subversion é muito utilizado em projetos corporativos de grande porte. A figura a seguir demonstra o funcionamento da ferramenta Subversion, apresentando um processo simples de interação dos desenvolvedores com o repositório central (sistema centralizado), por meio de **commits** e **updates**.

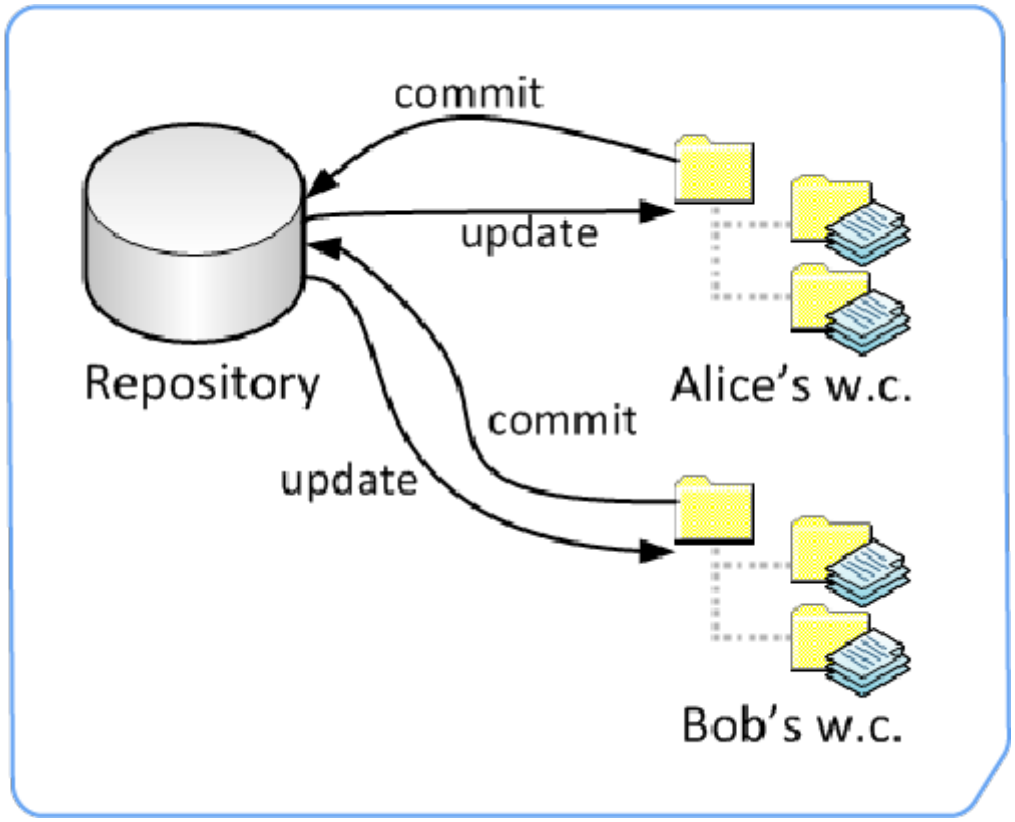


Figura 7 – Funcionamento básico do Subversion

Fonte: WKU Physics ([s. d.])

Embora o Subversion não seja um dos lançamentos mais recentes, ainda se mantém no mercado entre os mais utilizados.

Mercurial

O Mercurial merece destaque nesta lista por ser uma ferramenta de controle de versão utilizada por grandes empresas do setor de tecnologia, como Facebook e Google. A agilidade de uso e a facilidade em trabalhar com grandes equipes tornam a abordagem distribuída do Mercurial um grande destaque para essa ferramenta.

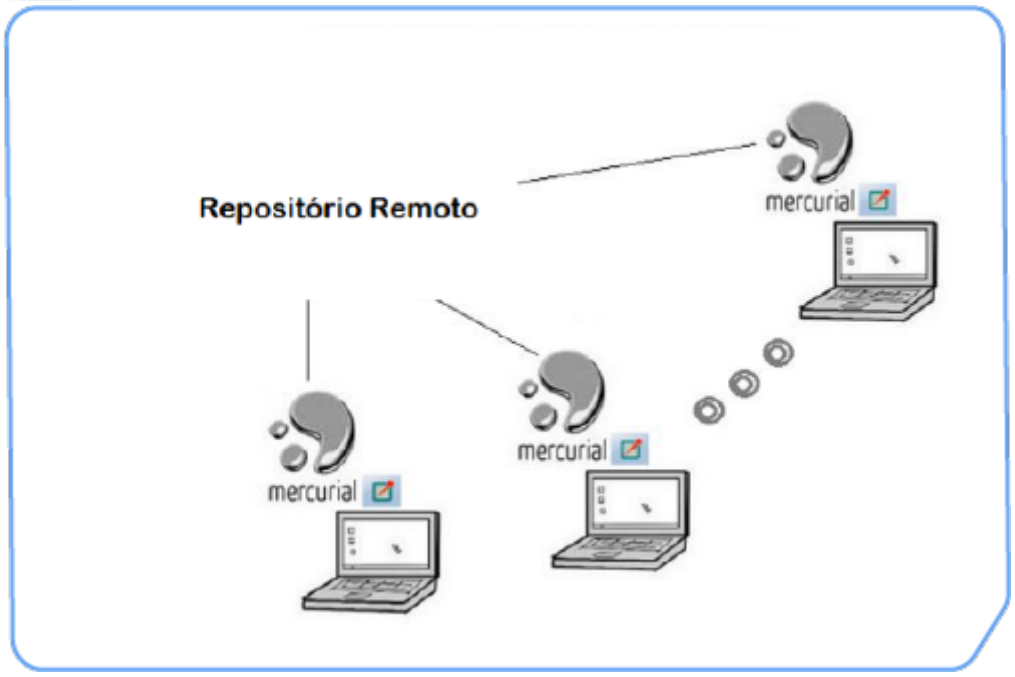


Figura 8 – Sistema distribuído do Mercurial

Fonte: adaptado de LabTel (c2023)

Em comparação ao Git, há uma pequena diferença em *performance* na qual o Git se destaca, entretanto o Mercurial também apresenta respostas rápidas.

TFS/Azure DevOps Server

O TFS (Team Foundation Server) era o nome do controle de versão centralizado não gratuito da Microsoft lançado em 2005. No ano de 2018, com diversas atualizações relevantes no sistema, foi renomeado para Azure DevOps Services. Além de realizar o gerenciamento de versões, essa ferramenta também conta com suas próprias ferramentas colaborativas de desenvolvimento e integra-se ao Git.

Principais funcionalidades de uma ferramenta de controle de versão (com base em comandos Git)

As funcionalidades das ferramentas de controle de versão de *software* podem ser explicadas de maneira prática, à medida que se conhecem seus comandos, os quais resumem e tornam possível o processo de versionamento de código.

Aqui, você verá um pouco mais sobre os comandos comentados ao longo deste conteúdo.

Commit

Este comando realizará a adição das alterações mais recentes dos arquivos para o repositório. Tais alterações devem ser revisadas no repositório principal e, posteriormente, confirmadas. Com isso, quando outros desenvolvedores solicitarem esse arquivo ao servidor central, receberão a versão mais atualizada dele.

Os diferentes **commits** realizados em arquivos de um projeto, dentro do repositório da ferramenta de controle de versão de *software*, são mantidos de forma constante, proporcionando, assim, o fácil acesso às diferentes versões de um arquivo.

Push

Este comando é responsável por enviar arquivos do repositório local para o repositório remoto. Dessa forma, o comando **push** envia todos os **commits** que foram produzidos pelo desenvolvedor para a confirmação no repositório remoto do servidor central. Isso permite que o resto da equipe tenha acesso a esse arquivo atualizado.

Pull

De forma oposta ao comando **push**, o comando **pull** é utilizado para trazer os arquivos do repositório remoto para o repositório local, garantindo que o desenvolvedor está trabalhando com a versão mais atualizada desses dados.

A figura a seguir apresenta a interação entre um repositório local com o repositório remoto, por meio dos comandos **pull** e **push**, permitindo atualizar e acessar os **commits** do repositório remoto.

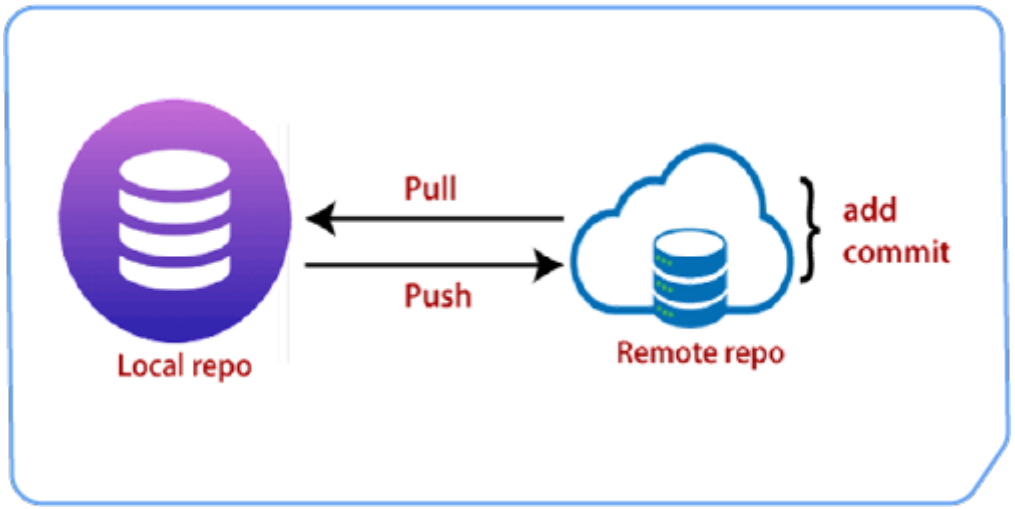


Figura 9 – Demonstração de comandos e funcionamento
Fonte: Java T Point (c2021)

Rotina de trabalho

Neste tópico, você verá como o versionamento de código se aplica dentro da rotina de trabalho de uma equipe. Neste cenário apresentado, você trabalhará com uma equipe de três programadores: Luís, Ana, Camila.

Como primeiro passo, utilizando o Git, os membros da equipe criam um repositório, o qual armazenará os arquivos do projeto que está sendo desenvolvido. Camila realiza um primeiro **commit** nos arquivos do projeto, denominado “versão inicial”.

A partir desse momento, todos os programadores devem trabalhar de maneira alinhada, sempre utilizando as funcionalidades da ferramenta de controle de versão para manter os arquivos do projeto atualizados, tanto em seu repositório local quanto no repositório remoto, fazendo com que todos os membros da equipe tenham acesso às últimas versões e alterações dos arquivos.

Realizam-se **commits** que descrevem as alterações realizadas e levam-se essas atualizações ao repositório central (comando **push**) para que os outros programadores possam também acessar essas modificações e trazê-las para o seu repositório local também (comando **pull**).

Essas práticas evitam diversas situações de conflito e mantêm a organização do projeto mesmo com os membros da equipe trabalhando de forma simultânea nos mesmos arquivos.

Agora serão simuladas, na prática, ações de membros da equipe de desenvolvimento, demonstrando o desfecho de cada situação, com e sem a aplicação do versionamento de código.

Os programadores Ana e Luís estão trabalhando no mesmo projeto. Em determinado momento, acabam trabalhando no mesmo arquivo de forma simultânea. Nesse caso, Ana realizou um **commit** no arquivo, porém Luís já havia atualizado esse arquivo anteriormente.

Tendo em vista essa situação, veja os dois cenários possíveis:

1. A equipe não utiliza nenhum tipo de ferramenta para o controle de versão de *software*

Neste caso, acontecerá uma situação de conflito, visto que Ana está realizando um **commit** em um arquivo desatualizado, pois não tem a última versão desenvolvida por Luís.

2. A equipe utiliza uma ferramenta de controle de versão de *software*

Nesta situação, Ana teria sempre os arquivos atualizados em sua máquina. Mesmo que Luís houvesse realizado mais atualizações no arquivo em questão, Ana estaria ciente, e seu **commit** seria sobre a última versão do arquivo citado. Assim, situações de conflito seriam evitadas.

Numerações de uma versão de *software*

O versionamento de código permite à equipe decidir a forma como ele será aplicado durante o desenvolvimento de um projeto. Após definida essa etapa, cada versão do projeto que for gerada terá uma hierarquia com base em sua numeração: a numeração principal (primeiro número) muda quando houver alguma alteração completa na aplicação ou questões de incompatibilidade; a numeração secundária identifica novas funcionalidade ou complementos; e, por fim, a terceira numeração demonstra pequenos ajuste ou correção de pequenas falhas.

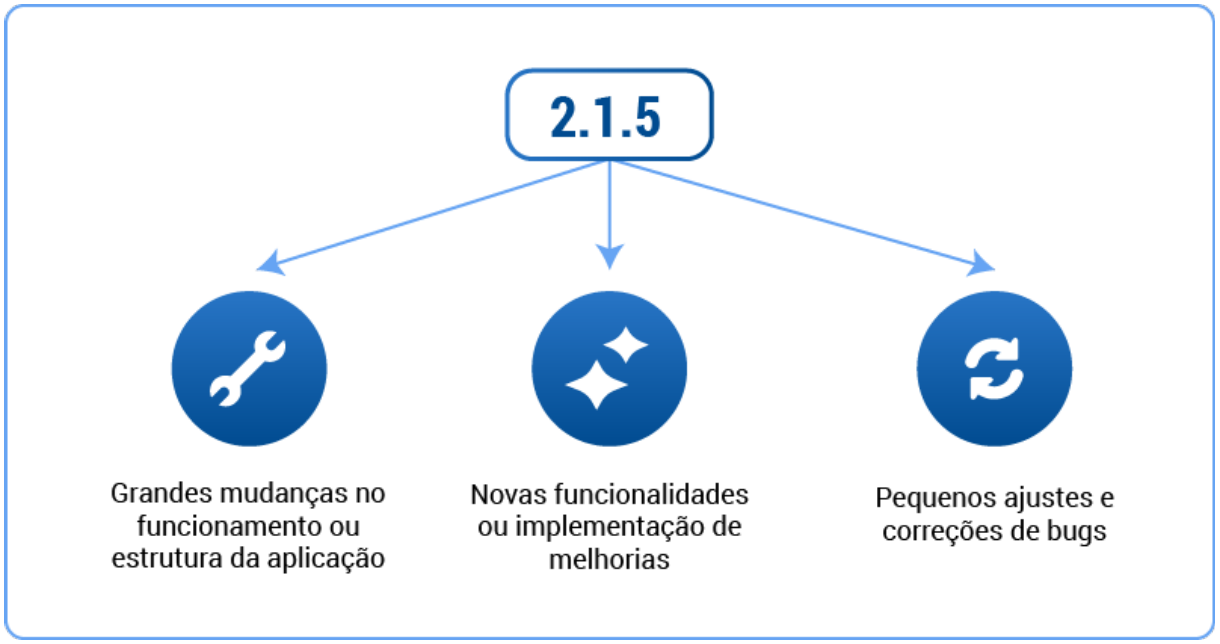


Figura 10 – Entendendo a numeração de uma versão de *software*
Fonte: Senac EAD (2023)

É preciso perceber que essa numeração aplicada à versão do *software* deve seguir uma sequência lógica. É essa sequência que permitirá encontrar onde ocorreu determinada falha, evitando, assim, o comprometimento da estrutura do sistema em questão.