se alterar o projeto principal ou até mesmo recuperar uma versão anterior, caso tenha ocorrido perda de alguma informação.

Os sistemas de controle de versão são muito utilizados no desenvolvimento de sistemas pelos iniciantes na área, que, ao serem inseridos em uma equipe de programadores, não conhecem as ferramentas utilizadas para que se consiga trabalhar em paralelo de forma eficiente, excluindo-se as possibilidades de sobreposição de alterações, utilização de versão errada, entre outras.

Neste capítulo, você conhecerá os principais tipos de controle de versão, também conhecido como versionamento. Além disso, verá quais são as principais operações utilizadas nos sistemas de versionamento. Por fim, conhecerá a diferença entre sistemas de versionamento distribuídos e centralizados.

1 Controle de versão

De maneira bem simples, os arquivos de um projeto são armazenados em um repositório, no qual o histórico de suas versões é registrado. Os programadores podem acessar e recuperar a última versão disponível e realizar uma cópia local, que possibilitará fazer alterações sobre ela. É possível submeter cada alteração executada ao repositório e recuperar as atualizações feitas pelos outros membros da equipe.

Dentro dessa arquitetura, é importante ressaltar que o controle de versão suporta o desenvolvimento em algumas formas, apresentadas a seguir:

- **Registrando o histórico:** armazena toda a evolução do projeto, ou seja, toda alteração realizada é registrada no repositório, o que possibilita a identificação de autor, data e origem das alterações. Uma característica importante é a possibilidade de reconstrução de determinada revisão específica do código-fonte, sempre que necessário.
- **Colaborando concorrentemente:** permite que mais de um programador realize alterações em paralelo sobre um mesmo código-fonte, sem sobrescrever as modificações de outro membro da equipe.
- **Variações no projeto:** proporciona a manutenção de versões diferentes de evolução do mesmo projeto. Ou seja, a versão 1.0 é a oficial, enquanto se prepara a versão 2.0.

Os controles de versões são compostos, basicamente, por duas partes: o **repositório** (servidor), que armazena todo o histórico de ajustes do projeto, registrando todas as alterações realizadas nos itens versionados; e a **estação de trabalho**, que possui uma cópia dos arquivos do projeto vinculada ao servidor para a identificação de possíveis modificações (Figura 1). Cada estação de trabalho é considerada individual e isolada das demais.

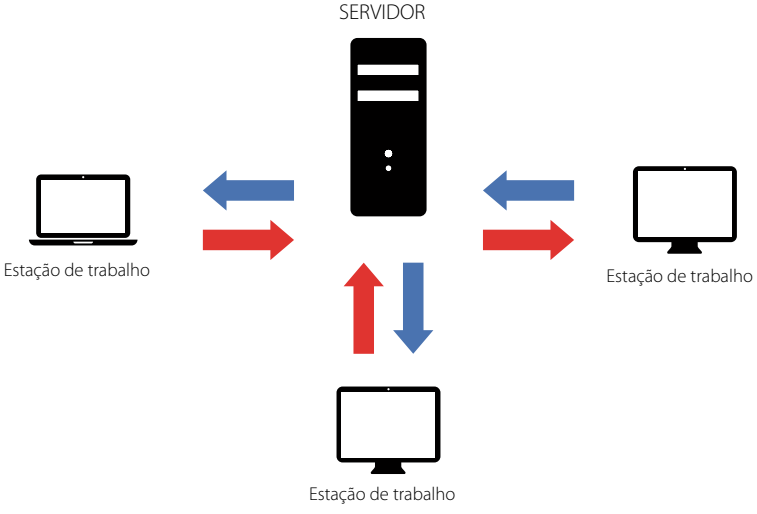

Figura 1. Arquitetura básica do controle de versão.

O processo de sincronização entre a estação de trabalho e o repositório é realizado por meio dos comandos `commit` e `update`. O comando `commit` submete um pacote com as modificações feitas pela estação de trabalho (origem) ao repositório (destino). Já o comando `update` realiza o processo inverso, ou seja, disponibiliza as alterações submetidas pelas demais estações de trabalho ao repositório (origem) para a estação de trabalho (destino), que deseja atualizar o projeto.

É importante ressaltar que todo `commit` cria uma revisão no servidor contendo as alterações, a data e o usuário responsável. O conceito de **revisão** pode ser ilustrado como uma fotografia de todos os arquivos e diretórios em um momento específico do projeto. Os registros antigos são guardados e podem ser recuperados assim que houver necessidade, e o conjunto dessas revisões é especificamente o histórico de alterações do projeto.

Existem diversas ferramentas para controle de versão no mercado, sendo algumas gratuitas e outras proprietárias. A seguir, serão apresentados os principais sistemas utilizados pelas equipes de desenvolvimento de *software* para controle de versionamento.

GIT

A ferramenta Git (Figura 2) é um dos sistemas de controle de versão e de controle de código-fonte distribuído mais conhecidos e utilizados pelas equipes de desenvolvimento de *software*. O Git foi criado por Linus Torvalds especificamente para o desenvolvimento do *kernel* Linux, porém tornou-se popular e foi adotado para outros projetos.

Figura 2. Logotipo do Git.
Fonte: Git (2020, documento *on-line*).

Uma das vantagens do Git é o fato de ele ser livre e de código aberto, além de ter sido projetado para lidar com todo tipo de projeto, desde pequenos a enormes, com eficiência e rapidez. Algumas características interessantes do Git são a garantia de dados, área de teste, ramificação e fusão.

Para o gerenciamento dos repositórios na *web*, os desenvolvedores utilizam, em sua maioria, o *github* ou o *bitbucket*.

Redmine

A ferramenta Redmine (Figura 3) consiste em uma aplicação *web* desenvolvida por meio do uso do *framework* Ruby on Rails, além de ser multiplataforma e utilizar um banco de dados cruzado.

Figura 3. Logotipo do Redmine.
Fonte: Redmine (2020, documento *on-line*).

Dentre as características do Redmine, destacam-se as apresentadas a seguir:

- suporte para múltiplos projetos;
- controle flexível de acesso baseado em função;
- sistema de rastreamento de problemas flexíveis;
- rastreamento do tempo;
- suporte a vários bancos de dados;
- integração com diversos repositórios (SVN, CVS, Git, Mercurial e Bazaar).

IBM Rational ClearCase

O IBM Rational ClearCase (Figura 4) disponibiliza acesso controlado aos ativos de *software*, incluindo código, requisitos, documentos de projeto, modelos, planos e resultados de testes. Além disso, ele oferece suporte a desenvolvimento paralelo, gerenciamento automatizado do espaço de trabalho, gerenciamento de linha de base, gerenciamento seguro de versões, auditoria confiável de compilação e acesso flexível.

Figura 4. Logotipo do IBM Rational ClearCase.
Fonte: IBM (2020, documento *on-line*).

Dentre as características do IBM Rational ClearCase, destacam-se as apresentadas a seguir:

- gerenciamento automatizado do espaço de trabalho;
- acesso transparente e em tempo real a arquivos e diretórios;
- suporte a desenvolvimento paralelo;
- gerenciamento avançado de criação e lançamento;
- acesso local, remoto e de cliente à *web*.

Subversion

O Subversion (Figura 5) é um sistema de controle de versão de código aberto, centralizado e caracterizado, em virtude de sua confiabilidade, como um repositório seguro para projetos e dados importantes. Alguns pontos de destaque são: a simplicidade do modelo e seu fácil uso e a capacidade de suportar as demandas de uma variedade de usuários e projetos para operações corporativas de grande escala.

Figura 5. Logotipo do Subversion.
Fonte: IBM (2020, documento *on-line*).

Dentre as características do Subversion, destacam-se as apresentadas a seguir:

- versionamento de diretórios;
- registro de cópias, exclusões e renomeação;
- metadados versionados de forma livre;
- *commits* únicos e individuais;
- bloqueio de arquivos;
- resolução de conflitos interativos.

Mercurial

O Mercurial (Figura 6) é um sistema de controle de versão distribuído e gratuito que gerencia de forma eficaz projetos de qualquer tamanho, além de oferecer uma interface fácil e intuitiva. Cada cópia possui todo o histórico do projeto, com isso, a maioria das ações é realizada localmente, de forma rápida e conveniente.

Figura 6. Logotipo do Mercurial.
Fonte: Mercurial (2020, documento *on-line*).

Dentre as características do Mercurial, destacam-se as apresentadas a seguir:

- suporte a vários fluxos de trabalho;
- arquitetura distribuída;
- facilidade de uso;
- extensível;
- código aberto;
- plataforma independente.

Os principais objetivos do desenvolvimento do Mercurial incluem alta *performance* e escalabilidade, descentralização, desenvolvimento colaborativo distribuído, controle de arquivos textuais e binários de forma robusta e operações avançadas de ramos e mesclagem. Além disso, o Mercurial inclui, de forma integrada, um sistema de visualização dos repositórios via *web* e facilitação na transição de usuários do Subversion.

Darcs

O Darcs (Figura 7) é um sistema de controle de versão de código aberto e gratuito e de plataforma cruzada, como Git, Mercurial ou Subversion. No entanto, uma característica diferenciada é o seu foco em mudanças, em vez de em *snapshots*. O Darcs permite, ainda, uma forma de trabalho mais livre e interface de usuário simplista. Outro ponto a ser considerado como diferencial é que ele não exige um servidor centralizado, funcionando perfeitamente no modo *off-line*.

Figura 7. Logotipo do Darcs.
Fonte: Darcs (2020, documento *on-line*).

Dentre as características do Darcs, destacam-se as apresentadas a seguir:

- modo *off-line*;
- preparação local;
- fácil ramificação e fusão;
- fácil colaboração por *e-mail*;
- desenvolvimento paralelo;
- interatividade;
- hospedagem própria.

BAZAAR

O Bazaar (Figura 8) é um sistema de controle de versão que tem a capacidade de rastrear o histórico do projeto ao longo do tempo, além de possibilitar a colaboração entre desenvolvedores.

Figura 8. Logotipo do Bazaar.
Fonte: Bazaar (2020, documento *on-line*).

Dentre as características do Bazaar, destacam-se as apresentadas a seguir:

- trabalho *off-line*;
- suporte a diversos fluxos de trabalho;
- suporte multiplataforma;
- rastreamento de renomeação e mesclagem inteligente;
- alta eficiência e velocidade de armazenamento.

O Git e o Mercurial são muito flexíveis em relação aos fluxos de trabalho, porém o Bazaar é a única dessa ferramenta que suporta ramificações vinculadas, sendo uma forma mais fácil e segura de implementar um fluxo de trabalho centralizado.