Uma característica importante dos controles de versão é a possibilidade de separar modificações em um caminho diferente para cada desenvolvimento, conforme a Figura 9. Esse caminho, conhecido como ***branch*** (seta verde), é utilizado especialmente para a implementação de novas funcionalidades, sem comprometer o caminho principal da implementação, denominado ***trunk***, (seta azul), com erros de compilação e *bugs*. A *branch* só será integrada ao *trunk* quando ela se tornar estável. Há também a possibilidade de congelamento de revisão, denominada ***tag*** (seta cor de laranja), isto é, é um estado fixo do produto que possui um conjunto de funcionalidades estáveis que não sofrerão mais nenhuma alteração.

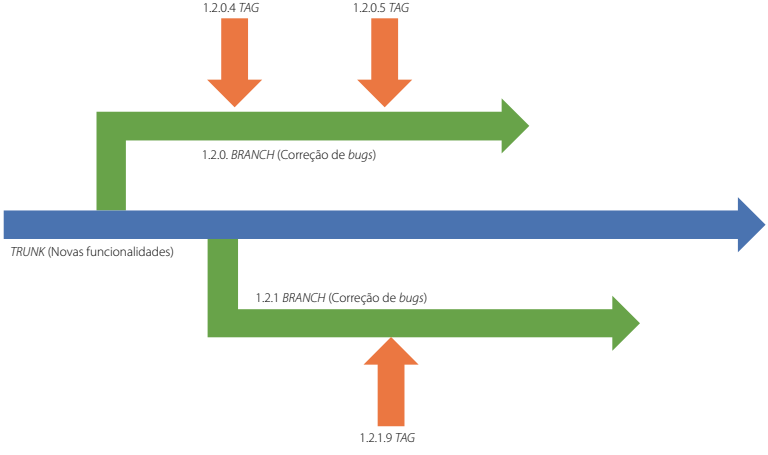

Figura 9. Diferenças entre *trunk*, *branch* e *tag*.
Fonte: Adaptada de Guelfi (2015).

3 Versionamentos distribuído e centralizado

Existem dois tipos de sistemas de controles de versão: centralizado e distribuído. Ambos possuem repositórios e estações de trabalho, porém a diferença entre eles está em como cada um está estruturado e organizado. A seguir, são apresentados os detalhes e as diferenças entre esses dois tipos de versionamento.

Versionamento centralizado

O sistema de controle de versão centralizado é composto por um único servidor central e várias estações de trabalho, com base no conceito de arquitetura cliente–servidor. Por ser um sistema centralizado, as estações de trabalho necessitam consultar o servidor para a realização da comunicação. Esse modelo atende à maioria das equipes de desenvolvimento de sistemas de médio porte para a realização de implementação por meio de uma rede local, além de não necessitar de velocidade para o envio e o recebimento dos dados. Um dos sistemas mais comuns com esse tipo de controle de versão centralizado é o Subversion.

Os sistemas de controle de versão centralizados possuem uma topologia em forma de estrela, ou seja, há um único repositório central com diversas estações de trabalho, uma para cada desenvolvedor. A comunicação entre as estações de trabalho passa, obrigatoriamente, pelo repositório central, conforme a Figura 10.

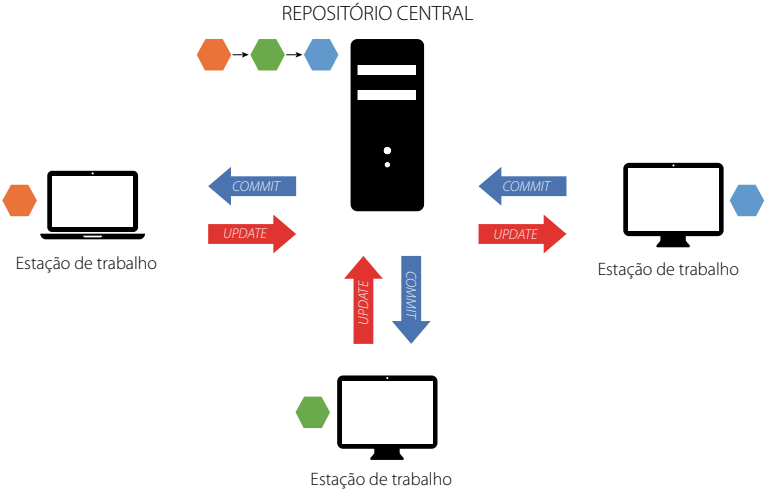

Figura 10. Sistema de controle de versão centralizado.

Algumas vantagens do uso do versionamento centralizado são: maior controle do projeto, imposição de segurança de acesso com facilidade e possibilidade de bloqueio de arquivos específicos, sendo ideal para equipes pequenas. Com relação às desvantagens, pode-se considerar: baixa escalabilidade e necessidade constante de conexão com a internet.

Versionamento distribuído

Os sistemas de controle de versão distribuídos podem ser definidos como diversos repositórios autônomos e independentes, um para cada programador. Cada repositório tem a sua estação de trabalho acoplada, onde as operações *commit* e *update* ocorrem localmente, conforme a Figura 11.

Essa arquitetura é recomendada para equipes com uma grande quantidade de desenvolvedores que estão remotamente distantes. O funcionamento do sistema de controle de versão distribuído ocorre da seguinte forma: cada estação de trabalho possui o seu próprio repositório, ou seja, as operações de *checkin* e *checkout* são realizadas de forma local. Ao contrário da arquitetura centralizada, essas estações de trabalho podem se comunicar entre si, porém é recomendável que se utilize um servidor responsável pelo envio dos arquivos para que se organize o fluxo e se evite ramificações do projeto e a perda do controle. Na maioria das vezes, o sistema oferece um servidor remoto para que o projeto seja hospedado. O processo de comunicação entre o servidor principal e as estações de trabalho funciona por meio de duas operações: uma para atualizar (puxar) e outra para mesclar o projeto (empurrar), conhecidas como ***pull*** e ***push***, respectivamente.

Considerando-se que o processo é local, o sistema de controle distribuído possui maior rapidez, porém exige um maior conhecimento da ferramenta e, inicialmente, pode confundir o programador. Por exemplo, o sistema de mesclagem em alterações concorrentes torna-se distinto por trabalhar em um sistema de arquivos binários, que, em determinadas situações, não possibilita a comparação entre essas atualizações ao mesmo tempo. Já os sistemas de controle de versão centralizados utilizam arquivos de texto, o que permite a comparação em alterações concorrentes, apresentando ao programador a possibilidade de escolher a solução ideal.

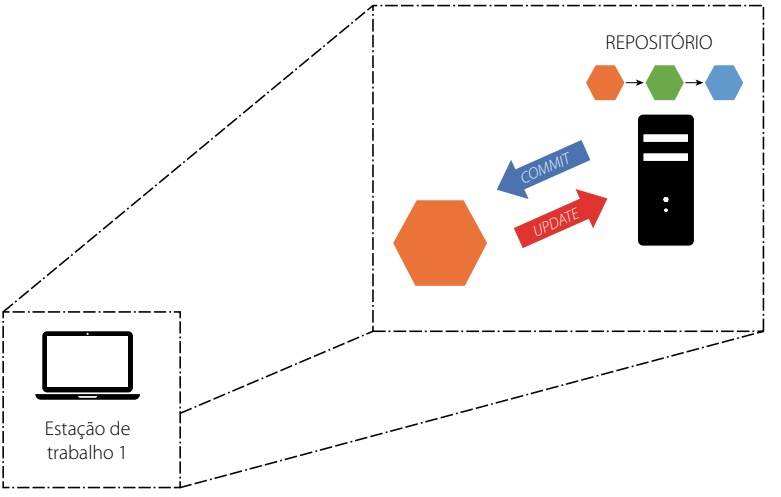

Figura 11. Sistema de controle de versão distribuído.

Com relação ao sistema de controle de versão distribuído, algumas das vantagens do seu uso são replicação do repositório, maior rapidez e autonomia, pois ele permite a realização de alteração *off-line*. No que se refere às desvantagens, pode-se citar a complexidade do fluxo de trabalho e a dificuldade em bloqueio de arquivos específicos.

Por fim, observa-se que os sistemas de controle de versão resolvem muitos problemas relacionados diretamente ao desenvolvimento de *software*. Atualmente, é prática comum a utilização dessa forma de trabalho, e existem inúmeras ferramentas disponíveis no mercado, conforme apresentado. É importante ressaltar que, antes de escolher qual sistema utilizar, deve-se analisar as opções e identificar a solução que melhor atende às necessidades da equipe de desenvolvimento.

Referências

BAZAAR. [*Site*]. [2020]. Disponível em: https://bazaar.canonical.com/en/. Acesso em: 26 ago. 2020.

DARCS. [*Site*]. [2020]. Disponível em: http://darcs.net/. Acesso em: 26 ago. 2020.

GIT. [*Site*]. [2020]. Disponível em: https://git-scm.com/. Acesso em: 26 ago. 2020.

GUELFI, E. *Conceitos do controle de versão*: criando branchs e tags utilizando Tortoise SVN. 2015. Disponível em: https://tsdn.tecnospeed.com.br/blog-do-desenvolvimento-tecnospeed/post/conceitos-do-controle-de-versao-criando-branchs-e-tags-utilizando-tortoise-svn#:~:text=Logo%20de%20cara%20podemos%20dizer,funcionalidades%20que%20n%C3%A3o%20ser%C3%A3o%20mais. Acesso em: 26 ago. 2020.

IBM. *What can IBM Rational ClearCase do for my business?* [2020]. Disponível em: https://www.ibm.com/us-en/marketplace/rational-clearcase. Acesso em: 26 ago. 2020.

KIM, G. *et al. Manual de DevOps*: como obter agilidade, confiabilidade e segurança em organizações tecnológicas. Rio de Janeiro: Alta Books, 2018.

MERCURIAL. [*Site*]. [2020]. Disponível em: https://www.mercurial-scm.org/. Acesso em: 26 ago. 2020.

REDMINE. [*Site*]. [2020]. Disponível em: https://www.redmine.org/projects/redmine/wiki/Logo. Acesso em: 26 ago. 2020.

Leituras recomendadas

ARUNDEL, J.; DOMINGUS, J. *DevOps nativo de nuvem com Kubernetes*: como construir, implantar e escalar aplicações modernas na nuvem. São Paulo: Novatec, 2019.

MUNIZ, A. *et al. Jornada DevOps*: unindo cultura ágil, Lean e tecnologia para entregar software com qualidade. 2. ed. Rio de Janeiro: Brasport, 2020.

Fique atento

Os *links* para *sites* da *web* fornecidos neste capítulo foram todos testados, e seu funcionamento foi comprovado no momento da publicação do material. No entanto, a rede é extremamente dinâmica; suas páginas estão constantemente mudando de local e conteúdo. Assim, os editores declaram não ter qualquer responsabilidade sobre qualidade, precisão ou integralidade das informações referidas em tais *links*.