Banco de Dados

Aula 1 - Controle de Versão com Git

O que é controle de versão?

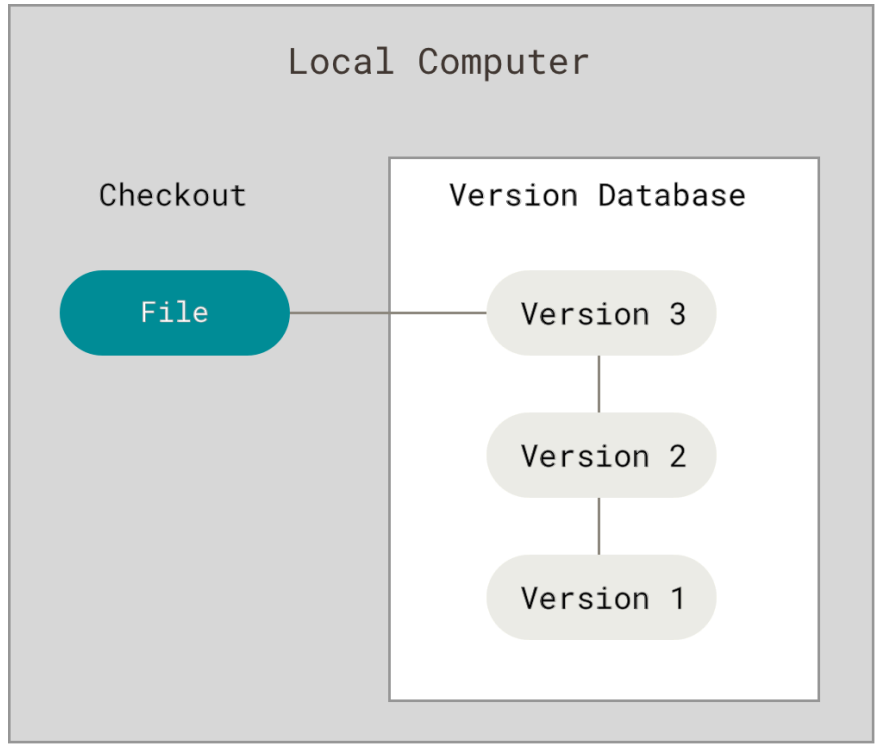
O controle de versão é um sistema que grava alterações em um arquivo, ou conjunto de arquivos, ao longo do tempo para que seja possível recuperar uma versão específica no futuro. Ou seja, é uma sistema que ajuda na maneira como controlamos a adição de funcionalidades, correções de bugs e mudanças que alteram o comportamento do software e a conexão com outros sistemas. Existem diferentes ferramentas para controle de software, tais como `GIT`, `CVS`, `Subversion`, `Mercurial` e etc. Nesta disciplina vamos trabalhar com o `GIT`.

O `GIT` é uma das ferramentas de controle de versão de software mais populares, principalmente em projetos open source. Isso se deve, principalmente, pela popularidade do GitHub, uma plataforma para hospedagem de códigos. Além disso, o `GIT` é uma ferramenta de controle de versão distribuída, o que significa que é adequado para a utilização em grandes equipes, nas quais os desenvolvedores não estão localizados geograficamente no mesmo local.

Sistemas de Controle de Versão Local

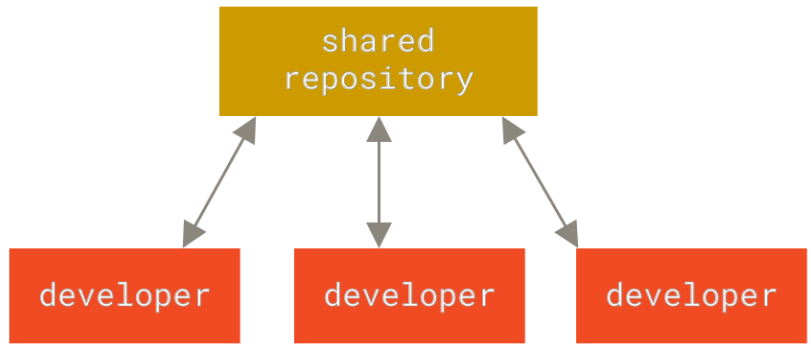
O método de controle de versão escolhido por muitas pessoas é copiar arquivos em outro diretório. Essa abordagem é muito comun porque é muito simples, mas também é muito sucetível a erros. É fácil esquecer em qual diretório você se encontra e acidentalmente sobrescrever o arquivo errado.

Para resolver este problema, programadores desenvolveram muito tempo atrás um sistema de controle de versão local baseado em um simples banco de dados que mantinha todas as alterações em arquivos mantidos em controle de revisão.



Sistemas de Controle de Versão Centralizado

O próximo problema que as pessoas tiverem que lidar foi a colaboração entre pessoas em diferentes sistemas. Para atacar este problemas foi criado o Sistemas de Controle de Versão Centralizado. Sistemas como CVS, Subversion e Perforce utilizavam um único servidor que contém todos os arquivos versionados, e um número determinados de clientes (pessoas) que podiam fazer `check out` dos arquivos direto daquele servidor central.



Vantagens

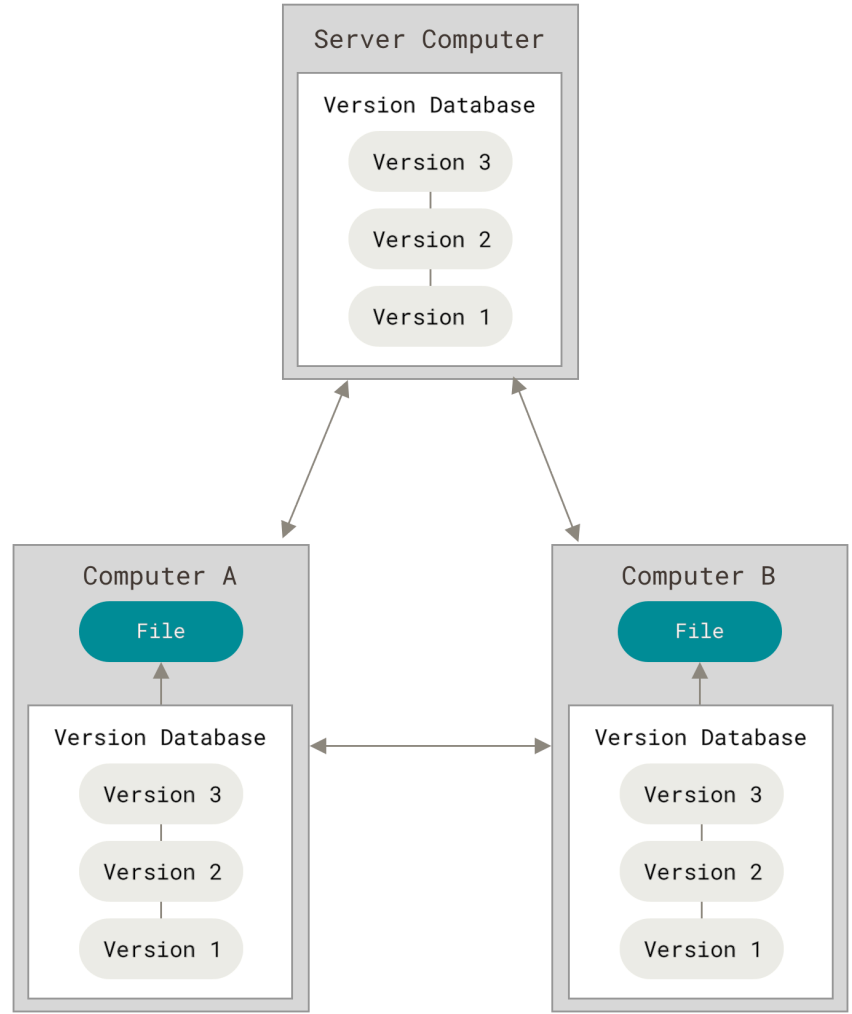
- Colaboração, controle de acesso pelos administradores, e visibilidade para as alterações

Desvantagens

- Disponibilidade e Backups

Sistemas de Controle de Versão Distribuído

Neste sistema, ao invés dos clientes pegarem apenas o último `snapshot` dos arquivos, eles espelham completamente o repositório, inclusive todo o histórico de modificações. Desta forma, cada repositório cliente se torna um `clone` do repositório no servidor remoto, resolvendo problemas de backup. O sistemas Git, Mercurial, Bazaar e Darcs funcionam de forma distribuída.



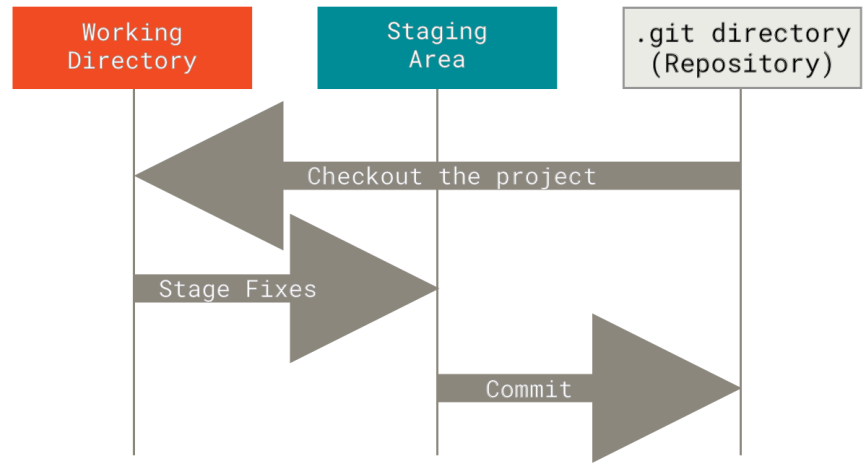
GIT

Git trabalha com seus dados pensando em uma série de snapshots de um pequeno sistema de arquivos. Assim, toda vez que fazemos um `commit` ou salvamos o estado do nosso projeto, o Git basicamente tira uma foto do estado de todos os arquivos naquele momento e armazena uma referência pra esse `snapshot`. Para ser eficiente, o Git não armazena novamente os arquivos que não foram modificados, mas cria uma referência apontando para a versão anterior

Os Três Estados

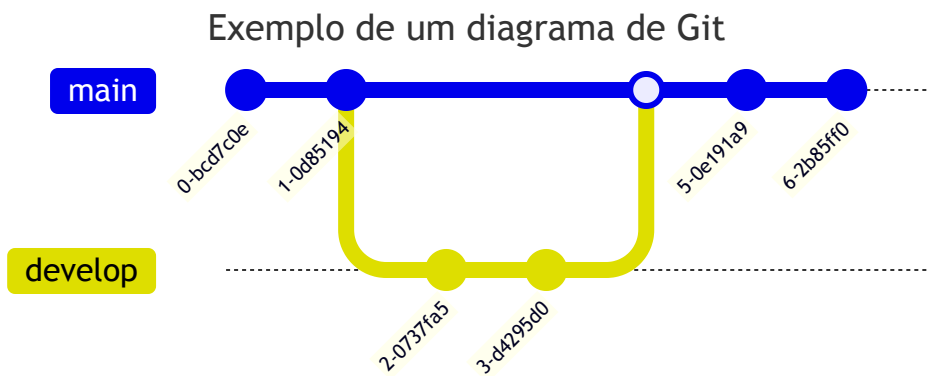
Com Git, os arquivos podem se encontrar em três principais estados:

- **Modificado**: significa que você alterou o arquivo mas ainda não commitou para o banco de dados;
- **Staged**: significa que você marcou o arquivo modificado para fazer parte do próximo snapshot no seu controle de versão;
- **Commitado**: significa que o seu arquivo foi gravado com sucesso no seu banco de dados local.



Git Braching

Uma Branch (Ramificação) nada mais é que abrir um novo fluxo de trabalho para continuar o desenvolvimento sem bagunçar o código que se encontra funcional no momento.



Comandos Importantes

```
$ git init # inicia um novo repositório local
$ git status # verifica as modificações do repositório local
$ git log # mostra histórico de alterações gravadas no banco de dados local
$ git add [nome-do-arquivo] # marca o arquivo para gravar no próximo commit
$ git commit # efetua a gravção das modificações no banco de dados local
$ git push [nome-do-remote] [nome-da-branch] # envia alterações para o repositório remoto
$ git pull [nome-do-remote]  [nome-da-branch] # baixa alterações do repositório remoto para o repositório local
$ git branch [nome-da-branch] # cria uma nova branch no repositório local
$ git checkout [nome-da-branch] # troca de branch para fazer novas alterações
$ git remote # configura informações do repositório remoto
$ git fetch # busca informações no repositório remoto mas não sobreescreve o banco de dados local
$ git stash # salva modificações em uma memória temporária e não registra no banco de dados
$ git merge [nome-da-branch] # Faz o merge da branch atual com a branch indicada no comando
```

GitFlow

O GitFlow é uma recomendação de organização de branchs e commits quando se está trabalhando em equipe. Isso ajuda a manter um estado funcional do código, recuperar versões anteriores e ajuda a manter o controle de qual versão será lançada para o usuário final.