AULA 4: Controle de Versão

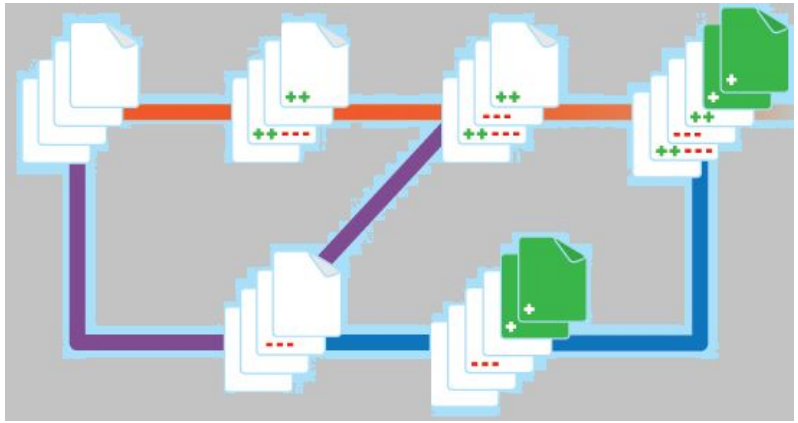
Professora: **Rosana T. Vaccare Braga**

Relembrando: Gerenciamento de Configuração

É uma disciplina da Engenharia de Software que busca identificar e controlar o acesso, as versões e as mudanças nos itens de configuração com o objetivo de garantir sua integridade

Controle de versões

Envolve o acompanhamento das várias versões dos componentes do sistema e a garantia de que as alterações feitas nos componentes por diferentes desenvolvedores não interfiram umas nas outras.



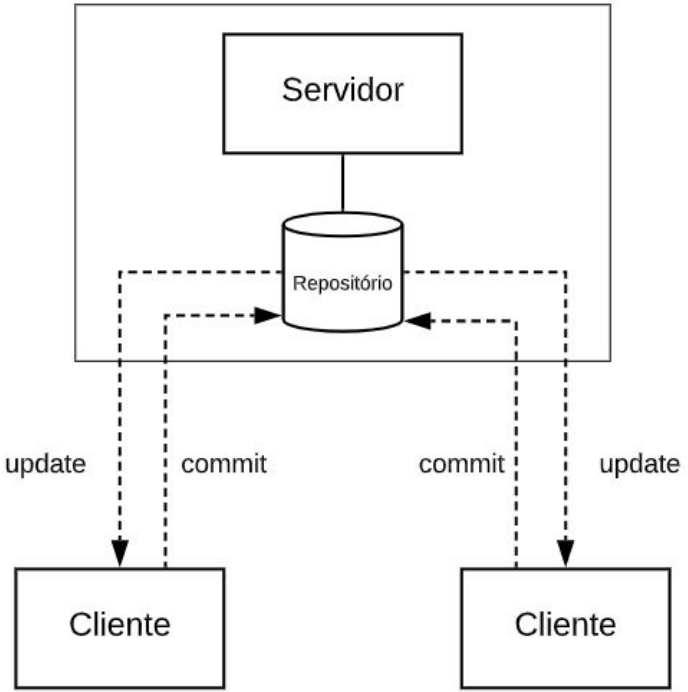
Engenharia de Software / Ian Sommerville 10ª Edição

Controle de Versões

- **Software é desenvolvido em equipe → necessidade de um servidor para armazenar o código fonte!**
- **Sistema de Controle de Versões (***VCS-Version Control System***)**
  - **Repositório**
  - **Recuperar versões mais antigas**

Controle de Versões

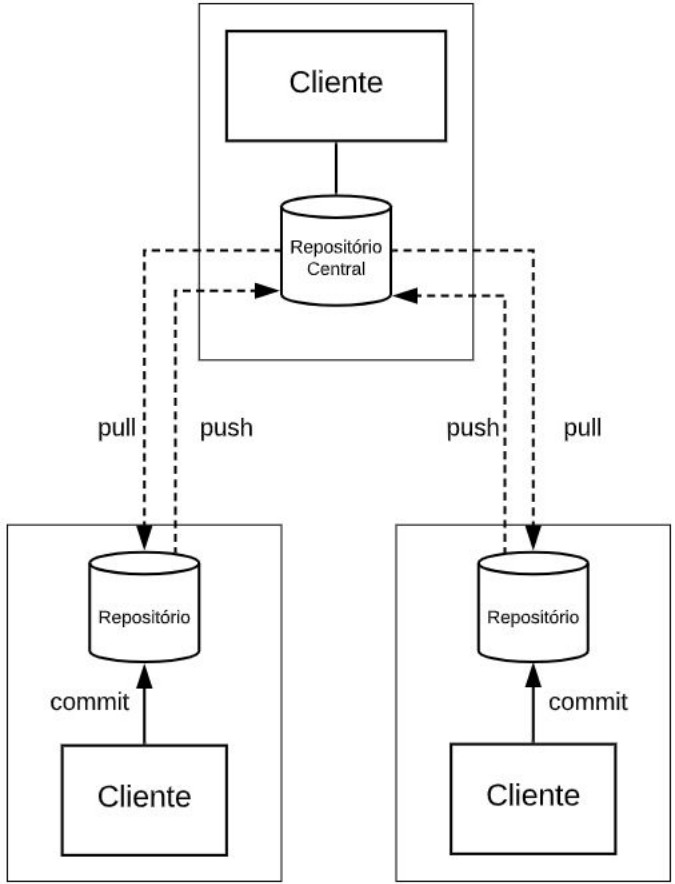
- **Início da década de 70**
  - **Source Code Control System (SCCS)**
  - **Concurrent Version System (CVS)**
  - **Subversion (SVN)**
  - **Cliente/Servidor**
  - **Commit**



VCS Centralizado. Existe um único repositório, no nodo servidor.

Controle de Versões

- **Início dos anos 2000**
  - **Sistemas de Controle de Versões Distribuídos (DVCS)**
    - **BitKeeper**
    - **Mercurial**
    - **Git**
  - **Arquitetura peer-to-peer**



VCS Distribuído (DVCS). Cada cliente possui um servidor. Logo, a arquitetura é peer-to-peer.

Controle de Versões

- **Repositório central**
- **pull e push**
- **Vantagens do DVCS sobre o VCS:**
  - **Gerenciar versões de forma offline**
  - **Realizar commits com mais frequência**
  - **Commits são executados em menos tempo**
  - **A sincronização não precisa ser sempre com o repositório central.**

Controle de Versões

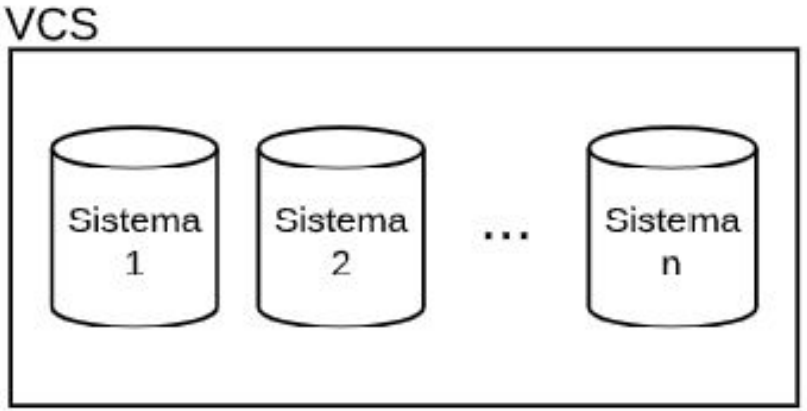
- **Git**
  - **Sistema de controle de versões distribuído**
  - **Inicialmente usava BitKeeper (licenças revogadas no Linux)**
  - **Implementação de um DVCS próprio**
  - **Sistema de código aberto**
  - **Sistema de linha de comando**
  - **Também interfaces gráficas**

Controle de Versões

- **GitHub**
  - **Serviço de hospedagem de código que usa o sistema Git**
  - **Repositórios públicos e gratuitos**
  - **Em vez de manter internamente um DVCS, pode alugar esse serviço do GitHub**
  - **Comparação pode ser feita com serviços de e-mail**
  - **Serviços semelhantes (GitLab e BitBucket)**

Multirepos vs Monorepos

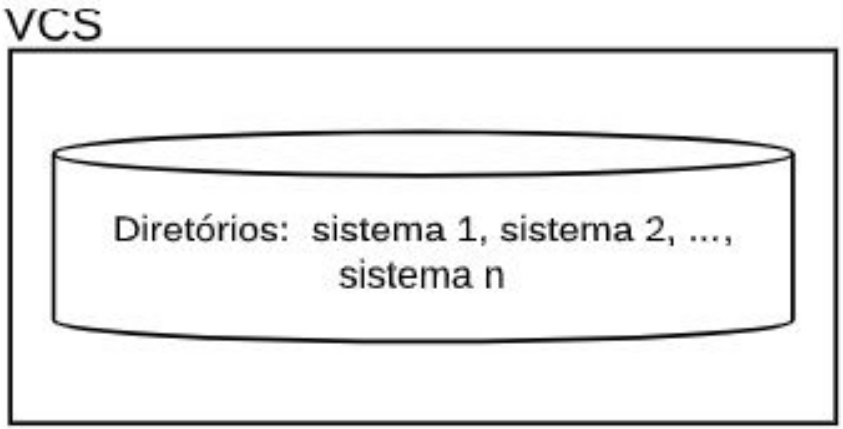
- **Multirepos é a decisão tradicional: criar um repositório para cada projeto ou sistema da organização**



Multirepos: um VCS gerencia vários repositórios. Normalmente, um repositório por projeto ou sistema.

Multirepos vs Monorepos

- **Monorepos: único repositório para toda a organização. Adotada principalmente em grandes empresas como Google, Facebook e Microsoft**



Monorepos: VCS gerencia um único repositório. Projetos são diretórios desse repositório.

Exemplo - GitHub

Multirepos

- aserg-ufmg/sistema1
- aserg-ufmg/sistema2
- aserg-ufmg/sistema3

Monorepo

- aserg-ufmg/sistemas
- Diretórios neste repo:
  - sistema1
  - sistema2
  - sistema3

Vantagens de Monorepos

- **Não há dúvida sobre qual repositório possui a versão mais atualizada**
- **Incentivam o reuso e compartilhamento de código**
- **Mudanças são sempre atômicas (1 commit pode alterar n sistemas)**
- **Facilita a execução de refactorings em larga escala**

contributed articles

DOI:10.1145/2854146

Google's monolithic repository provides a common source of truth for tens of thousands of developers around the world.

BY RACHEL POTVIN AND JOSH LEVENBERG

Why Google Stores Billions of Lines of Code in a Single Repository

This article outlines the scale of that codebase and details Google's custom-built monolithic source repository and the reasons the model was chosen. Google uses a homegrown version-control system to host one large codebase visible to, and used by, most of the software developers in the company. This centralized system is the foundation of many of Google's developer workflows. Here, we provide background on the systems and workflows that make feasible managing and working productively with such a large repository. We explain Google's "trunk-based development" strategy and the support systems that structure workflow and keep Google's codebase healthy, including software for static analysis, code cleanup, and streamlined code review.

Google-Scale

Google's monolithic software repository, which is used by 95% of its software developers worldwide, meets the definition of an ultra-large-scale[1] system, providing evidence the single-source repository model can be scaled successfully.

The Google codebase includes approximately one billion files and has a history of approximately 35 million commits spanning Google's entire 18-year existence. The repository contains 86TB[a] of data, including approximately

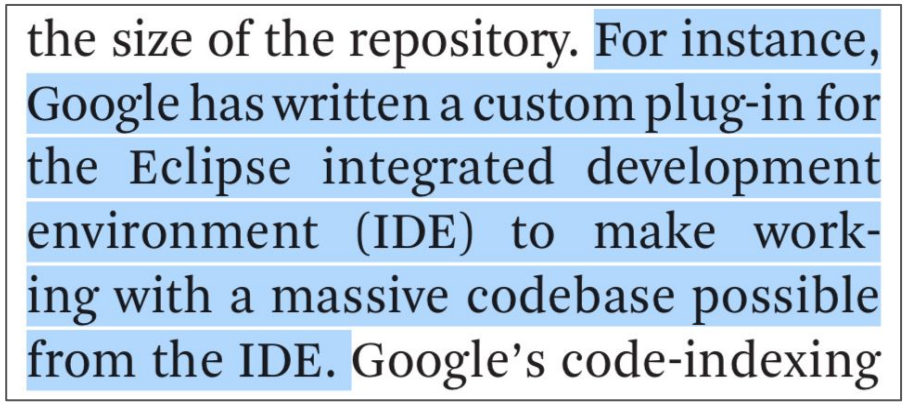
a Total size of uncompressed content, excluding release branches.

- **Monorepos são usados principalmente por grandes empresas**
- **Google: 2 bilhões de LOC e 25000 desenvolvedores em 2015 !!!**

https://research.google/pubs/pub45424/

Desvantagens de Monorepos

- **Requerem ferramentas para navegar em grandes bases de código (por exemplo uma IDE)**
  - **Desenvolvedor terá em seu repositório local todos os arquivos de todos os sistemas da organização**



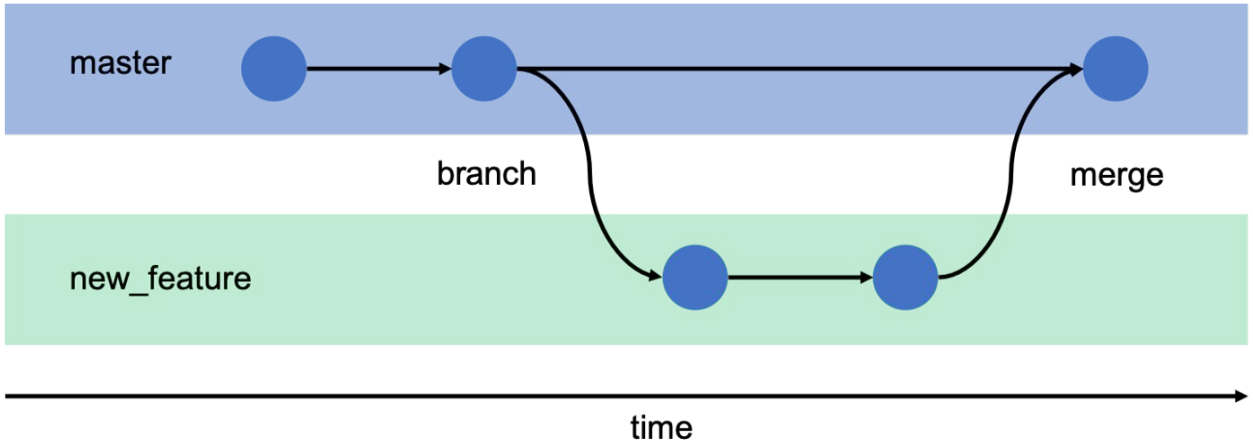
the size of the repository. For instance, Google has written a custom plug-in for the Eclipse integrated development environment (IDE) to make working with a massive codebase possible from the IDE. Google's code-indexing

Conceitos importantes (termos usados no contexto de controle de versão)

- **Master**
- **Branching**
- **Merge**
- **Delta**

Master

- **Definição de "Master"**
  - Termo comumente utilizado para se referir à branch principal de um repositório de código-fonte.
  - A branch "master" é a branch principal e padrão de um repositório Git, onde as versões mais recentes e estáveis do código-fonte são mantidas



Master

- **Uso de "Master"**
  - É comum encontrar a última versão do software lançada ou em produção
  - O uso da branch "master" vem sendo questionado em algumas comunidades de tecnologia, devido a possíveis conotações racistas e escravocratas associadas ao termo.
  - A maioria das comunidades optaram por renomear a branch principal de seus repositórios para termos neutros, como "main" ou "default".

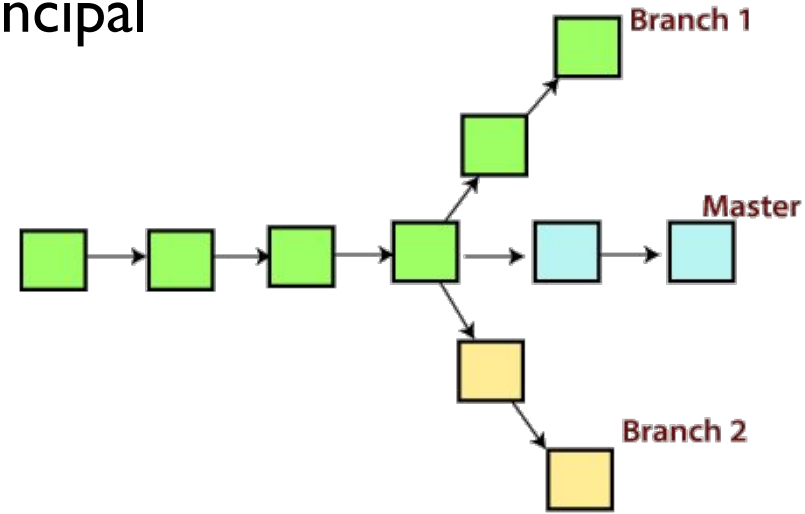
Trunk x Master

- "Trunk",, "master" e “main” são termos que se referem à ramificação principal de um projeto de software. A diferença entre eles pode variar de acordo com o sistema de controle de versão utilizado, mas geralmente se refere ao nome padrão da ramificação principal.
- No SVN (Subversion), o "trunk" é o termo utilizado para a ramificação principal do projeto, enquanto no Git, o termo "main" é usado para a mesma finalidade. Em ambos os casos, essa é a ramificação principal onde as mudanças são feitas e onde as novas funcionalidades são implementadas antes de serem mescladas em outras ramificações.

É importante notar que o uso de um ou outro termo não afeta a funcionalidade ou capacidade de um sistema de controle de versão, e é mais uma convenção de nomenclatura.

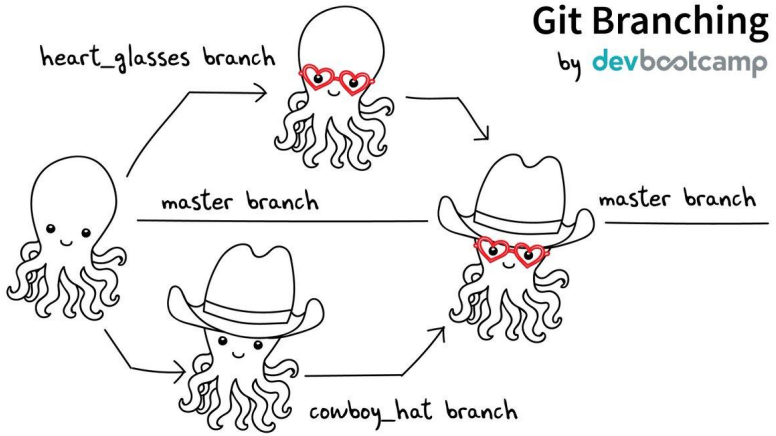
Branching

- **Definição de "branching"**
  - Processo de criar uma cópia independente de um conjunto de arquivos em um repositório de controle de versões
  - É uma linha de desenvolvimento isolada que permite aos desenvolvedores trabalhar em novos recursos ou correções de bugs sem afetar o código principal



Branching

- **Uso comum do branching**
  - Permitir que várias pessoas trabalhem em diferentes recursos do projeto ao mesmo tempo
  - Cada pessoa pode criar um branch separado e fazer suas alterações sem afetar a versão principal do projeto
  - Quando as alterações estiverem concluídas, o branch pode ser mesclado (merged) novamente com a versão principal



Branching

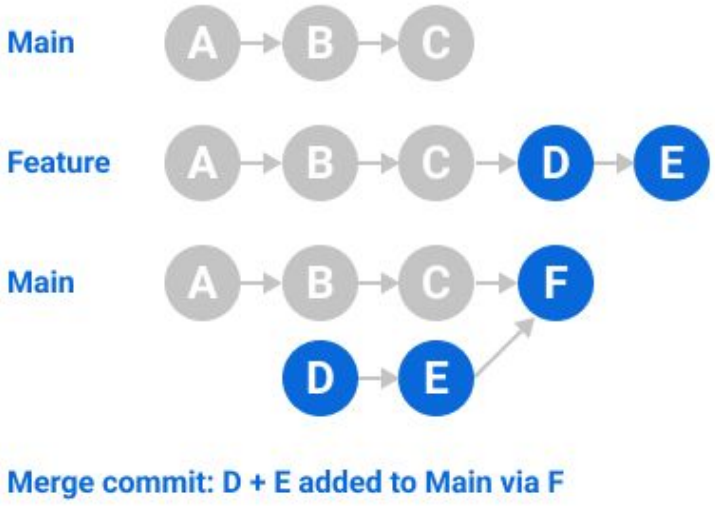
- **Uso do branching para testes**
  - Testar novos recursos ou correções de bugs sem afetar o desenvolvimento principal do projeto
  - Novo branch pode ser criado para testar uma nova funcionalidade e, se for bem-sucedido, pode ser mesclado novamente na versão principal

Branching

- **Importância do uso adequado do branching**
  - Ajuda a organizar e facilitar o desenvolvimento de projetos de grande porte
  - Cuidado ao criar muitos branches, pois pode levar à complexidade e dificuldade de gerenciamento.

Merge

- **Definição de "merge"**
  - Combinação de duas ou mais versões de um arquivo ou conjunto de arquivos em uma única versão
  - Objetivo: juntar as alterações feitas em diferentes versões de um arquivo para criar uma nova versão que contenha todas as mudanças

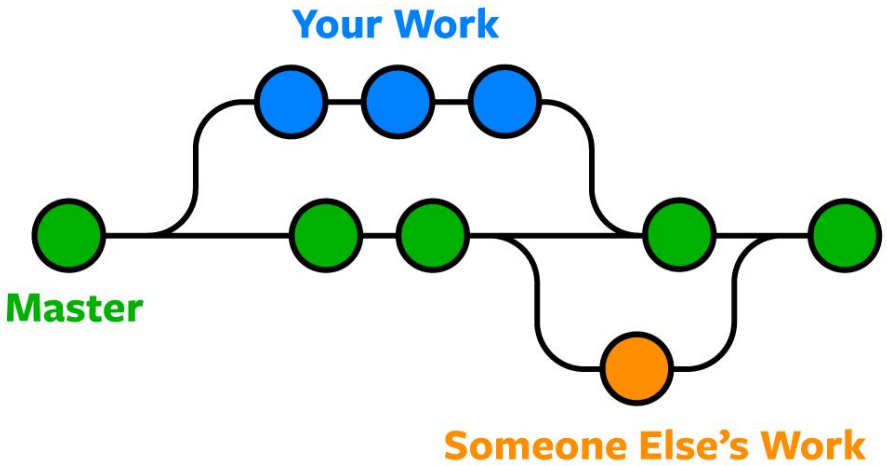


Merge

- **Quando ocorre o processo de merge?**
  - Quando duas ou mais pessoas trabalham em diferentes partes do mesmo projeto e fazem alterações separadamente
  - Quando essas alterações são enviadas para um repositório de controle de versões, elas precisam ser combinadas para criar uma nova versão que inclua todas as mudanças

Merge

- **Complicações do merge**
  - As alterações em diferentes versões do arquivo podem se sobrepor ou entrar em conflito
  - O uso de ferramentas de controle de versões adequadas e uma boa comunicação entre os membros da equipe é essencial para realizar merges de forma eficaz.



Delta

- **Definição de Delta**
  - Diferença entre duas versões consecutivas de um arquivo ou conjunto de arquivos
  - É calculada comparando o conteúdo de cada arquivo em ambas as versões e identificando as adições, remoções ou alterações que ocorreram.
    - Delta Negativo: armazena-se integralmente a versão mais recente e as diferenças (deltas) existentes até então.
    - Delta Positivo: armazena-se a versão mais antiga e, para montar as versões mais recentes, processam-se as diferenças (deltas) armazenadas.

Uso do Delta Positivo

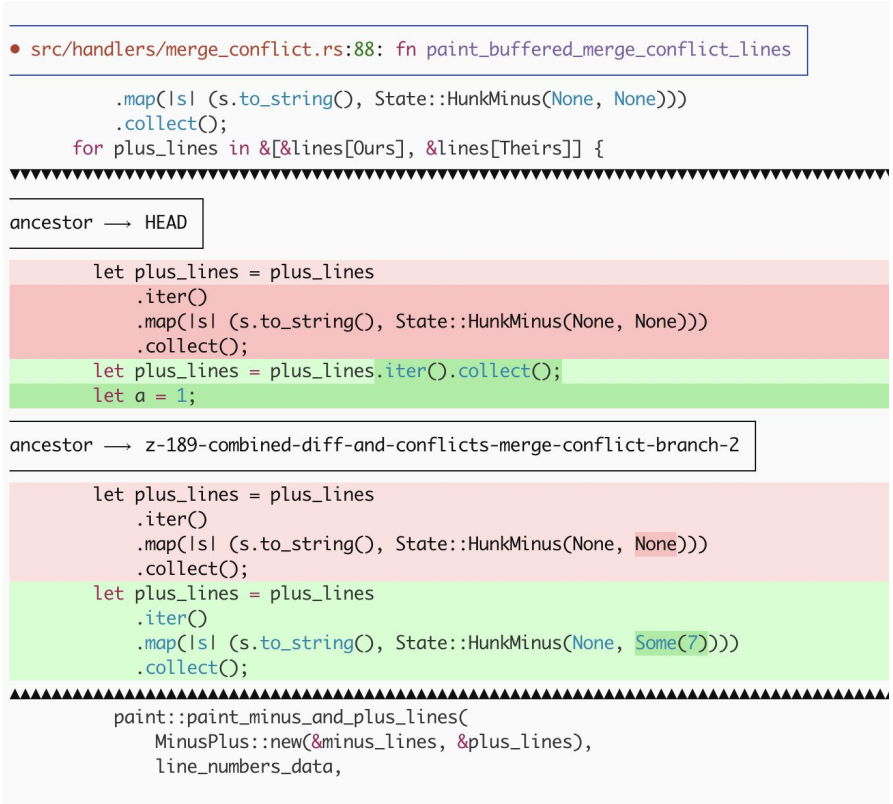
- Quando uma nova versão de um arquivo é salva em um repositório de controle de versões, o sistema compara o novo arquivo com a versão anterior e calcula o delta.
  - Armazena-se apenas as alterações feitas em um arquivo em vez de armazenar a versão completa do arquivo a cada alteração.
- Permite que o repositório de controle de versões seja mais eficiente em termos de espaço de armazenamento e transferência de dados.
  - usuários acessam e baixam apenas as alterações mais recentes em vez de baixar todo o arquivo novamente.
  - No entanto, o cálculo do delta pode ser computacionalmente intenso, especialmente em arquivos grandes ou quando há muitas alterações, o que pode afetar o desempenho do sistema.

Delta no Git (negativo no tronco)

- Em Git, o delta é calculado automaticamente sempre que uma nova versão de um arquivo é salva no repositório.
- O Git compara o conteúdo do arquivo alterado com a versão anterior e calcula o delta, que é armazenado como parte do commit que inclui a nova versão do arquivo.
- O comando Git mais comum para salvar uma nova versão de um arquivo no repositório é o `` `git commit` ``, que automaticamente calcula e armazena o delta das alterações no commit.

Delta

- O comando `git diff` pode ser usado para comparar as diferenças entre duas versões de um arquivo, mostrando as alterações linha por linha.
- O `git diff` não calcula o delta em si, mas usa o delta calculado pelo Git para mostrar as diferenças entre as versões.



Ferramentas

- **Controle de Versão**
  - **SVN:** O Apache Subversion (também conhecido como SVN) é um sistema de controle de versão de código-fonte aberto e centralizado, que permite rastrear mudanças em arquivos e diretórios ao longo do tempo, bem como colaborar em projetos de software com outras pessoas. O SVN é projetado para lidar com grandes volumes de dados e pode gerenciar qualquer tipo de arquivo, incluindo código-fonte, documentos e multimídia.

Ferramentas

- **Controle de Versão**
  - **CVS:** O Concurrent Versions System (CVS) é um sistema de controle de versão de código-fonte aberto e centralizado, que permite que várias pessoas trabalhem em um mesmo conjunto de arquivos simultaneamente. O CVS armazena os arquivos de um projeto em um repositório central, mantendo um histórico de todas as versões anteriores e permitindo que os usuários recuperem versões anteriores do código.

Ferramentas

- **Controle de Versão**
  - **GIT:** É um sistema de controle de versão de software livre e distribuído. O Git é capaz de lidar com projetos de qualquer tamanho com rapidez e eficiência, e é amplamente utilizado em diversos tipos de projetos de software. O Git se diferencia dos sistemas de controle de versão centralizados, como CVS e SVN, porque armazena todo o histórico de alterações em um repositório local de cada desenvolvedor, permitindo que as alterações sejam gerenciadas de forma independente, mesmo em projetos grandes e complexos.

Ferramentas

- **Controle de Versão**
  - **ClearCase:** ClearCase é um software comercial de gerenciamento de configuração de software da IBM. Ele é um sistema centralizado que armazena todas as versões e alterações em um repositório central e oferece recursos avançados de gerenciamento de ramificação e mesclagem. É usado principalmente em grandes empresas e oferece recursos de rastreamento de alterações e integração com outras ferramentas de desenvolvimento de software, como IDEs e sistemas de gerenciamento de bug.

Ferramentas

Outras ferramentas para controle de versão

- **Mercurial:** sistema de controle de versão distribuído que enfatiza o desempenho e a facilidade de uso;
- **Perforce:** sistema de controle de versão centralizado que é amplamente utilizado na indústria de jogos e em outros setores que trabalham com arquivos grandes;
- **Bitbucket:** plataforma de hospedagem de repositórios Git e Mercurial que inclui recursos de gerenciamento de código-fonte e colaboração em equipe.

Próxima aula: DevOps; Integração e entrega contínua

Se possível, leiam com antecedência:

Seções 10.1, 10.3 e 10.4 do livro Engenharia de Software Moderna
https://engsoftmoderna.info/cap10.html

Leitura interessante:

https://research.google/pubs/pub45424/