Aula 1: Introdução ao Controle de Versão

Objetivos: - Compreender os conceitos básicos e a importância do controle de versão. - Familiarizar-se com a evolução histórica das ferramentas de controle de versão. - Introdução ao Git

Conteúdos: - Definição e importância do controle de versão em desenvolvimento de software. - Visão geral histórica das ferramentas de controle de versão: do CVS ao Git. - Terminologia básica: commit, branch, merge, conflict. - Laboratório

Pré-requisitos

Antes de prosseguir com este conteúdo, é recomendado que você tenha conhecimento básico de:

- Desenvolvimento de software
- Terminal ou linha de comando

Se você não possui conhecimento prévio sobre esses tópicos, recomendo que você estude-os antes de continuar com este conteúdo.

Visão Geral

1. Qual é a definição e importância do controle de versão em desenvolvimento de software?
2. Quais são os problemas de desenvolvimento de software resolvidos pelo controle de versão?
3. Quais são as vantagens do Git em relação a outras ferramentas de controle de versão?
4. Quais são as desvantagens do Git?
5. Como o Git evoluiu ao longo do tempo em relação a outras ferramentas de controle de versão?
6. Quais são os comandos básicos do Git e como eles são usados?
7. O que é um commit hash e qual é a sua importância no controle de versão?
8. Como você trabalharia com o Git em um laboratório prático?
9. Quais são as principais diferenças entre um sistema de controle de versão centralizado e um sistema de controle de versão distribuído?
10. Como o Git permite o trabalho offline e fornece redundância em caso de falha do servidor central?

Problemas de Desenvolvimento de Software Resolvidos pelo Controle de Versão

Video: https://www.youtube.com/watch?v=8MsYOAnU1eY

controle-versao

1. **Perda de Código:** O controle de versão permite que você salve diferentes versões do seu código, evitando a perda de trabalho se algo der errado.

2. **Colaboração:** O controle de versão permite que várias pessoas trabalhem no mesmo projeto sem sobrescrever o trabalho umas das outras.
3. **Rastreamento de Alterações:** O controle de versão permite que você veja quem fez uma alteração, quando ela foi feita e por que ela foi feita.
4. **Reverter Alterações:** Se uma alteração introduzir um bug, o controle de versão permite que você reverta para uma versão anterior do código.
5. **Testando Novas Funcionalidades:** O controle de versão permite que você crie uma "branch" para desenvolver e testar novas funcionalidades sem afetar o código principal.
6. **Gerenciamento de Lançamentos:** O controle de versão permite que você mantenha diferentes versões do seu código para diferentes versões do seu software, facilitando o gerenciamento de lançamentos.

Evolução das Ferramentas de Controle de Versão

controle-versao

1. **Controle de Versão Local:** No início, os desenvolvedores mantinham versões locais de arquivos de código. Isso era propenso a erros, pois era fácil sobrescrever ou perder versões.

local-vcs

2. **Controle de Versão Centralizado (CVCS):** Ferramentas como CVS, Subversion e Perforce introduziram um servidor central que mantinha todas as versões do código. Isso facilitou a colaboração, mas a perda do servidor central poderia resultar na perda de todo o histórico de versões.

local-vcs

3. **Controle de Versão Distribuído (DVCS):** Ferramentas como Git, Mercurial e Bazaar permitiram que cada desenvolvedor tivesse uma cópia completa do histórico de versões. Isso tornou possível trabalhar offline e deu aos desenvolvedores a capacidade de ter várias "branches" de trabalho.

local-vcs

> O Git foi criado por Linus Torvalds em 2005 para gerenciar o desenvolvimento do kernel do Linux. Ele introduziu muitos conceitos inovadores em DVCS, como um modelo de dados que garante a integridade do código e a capacidade de manipular facilmente a história do código.

Introdução ao Git

git

Video: https://youtu.be/e9lnsKot_SQ

Git Book Pro https://git-scm.com/book/en/v2

Historia do Git https://git-scm.com/book/pt-br/v2/Come%C3%A7ando-Uma-Breve-Hist%C3%B3ria-do-Git

Git é um sistema de controle de versão distribuído (DVCS) criado por Linus Torvalds em 2005. Ele foi projetado para lidar com tudo, desde pequenos a grandes projetos com velocidade e eficiência.

O Git permite que cada desenvolvedor tenha uma cópia completa do histórico do projeto, facilitando o trabalho offline e fornecendo redundância em caso de falha do servidor central. Ele é otimizado para desempenho, garantindo que as operações sejam rápidas e eficientes.

O Git possui um sistema robusto e eficiente de branching e merging, facilitando o desenvolvimento paralelo e a experimentação. Ele também garante a integridade dos dados através de uma estrutura de dados que protege contra corrupção.

Além disso, o Git é altamente flexível e pode ser personalizado para se adequar a vários fluxos de trabalho. No entanto, essa flexibilidade e poder vêm com uma curva de aprendizado íngreme e a necessidade de entender e usar corretamente seus muitos comandos e opções.

Vantagens do Git

1. **Distribuído:** Cada desenvolvedor tem uma cópia completa do histórico do projeto, permitindo o trabalho offline e a redundância em caso de falha do servidor central.
2. **Desempenho:** O Git é otimizado para desempenho. As operações são rápidas e eficientes.
3. **Branching e Merging Eficientes:** O Git tem um sistema de branching e merging robusto e eficiente, facilitando o desenvolvimento paralelo.
4. **Integridade dos Dados:** O Git usa uma estrutura de dados que garante a integridade do código e do histórico.
5. **Flexibilidade:** O Git suporta vários fluxos de trabalho e pode ser personalizado para se adequar às necessidades de um projeto.

Desvantagens do Git

1. **Curva de Aprendizado:** O Git tem uma curva de aprendizado íngreme. Pode ser difícil para os novos usuários entenderem os conceitos e comandos.
2. **Complexidade:** O Git tem muitos comandos e opções que podem ser confusos para os usuários.
3. **Falta de Controle de Acesso Granular:** O Git não tem um sistema de controle de acesso granular integrado. Você pode controlar quem pode ler e escrever em um repositório, mas não em partes específicas do código.
4. **Binários:** O Git não lida bem com arquivos binários grandes. Eles podem inflar o tamanho do repositório e diminuir o desempenho.

Parte 2: Fundamentos do Git

Objetivos: - Instalar e configurar o Git. - Aprender comandos básicos do Git.

Conteúdos: - Instalação do Git e configuração inicial. - Repositórios Git: locais e remotos. - Comandos básicos: git init, git clone, git add, git commit, git push, git pull.

Atividades: - Criação de um novo repositório Git e clonagem de um repositório existente. - Prática dos comandos básicos com exercícios simples.

Estágios do Git

Preste atenção agora - aqui está a coisa principal para lembrar sobre o Git se você quiser que o resto do seu processo de aprendizado seja tranquilo. O Git tem três estados principais nos quais seus arquivos podem estar: modificado, preparado e confirmado:

stages

- Modificado significa que você alterou o arquivo, mas ainda não o confirmou no seu banco de dados.
- Preparado significa que você marcou um arquivo modificado em sua versão atual para entrar na próxima captura de commit.
- Confirmado significa que os dados estão armazenados com segurança no seu banco de dados local.

Isso nos leva às três principais seções de um projeto Git: a árvore de trabalho, a área de preparação e o diretório Git.

A árvore de trabalho é um único checkout de uma versão do projeto. Esses arquivos são retirados do banco de dados compactado no diretório Git e colocados no disco para você usar ou modificar.

A área de preparação é um arquivo, geralmente contido no diretório Git, que armazena informações sobre o que será incluído no seu próximo commit. Seu nome técnico na terminologia do Git é o "índice", mas a frase "área de preparação" também funciona bem.

O diretório Git é onde o Git armazena os metadados e o banco de dados de objetos do seu projeto. Esta é a parte mais importante do Git e é o que é copiado quando você clona um repositório de outro computador.

Prática

Comandos Principais do Git

1. `git init`: Inicializa um novo repositório Git.
   - Exemplo: `git init`
   - Caso de uso: Quando você quer começar a rastrear um novo projeto com o Git.

2. `git clone`: Clona um repositório Git existente.
    - Exemplo: `git clone https://github.com/usuario/projeto.git`
    - Caso de uso: Quando você quer trabalhar em um projeto que já está sendo rastreado pelo Git.
3. `git add`: Adiciona arquivos ao índice do Git para serem rastreados.
    - Exemplo: `git add .` (adiciona todos os arquivos modificados) ou `git add arquivo.txt` (adiciona um arquivo específico)
    - Caso de uso: Quando você fez alterações em um ou mais arquivos e quer que essas alterações sejam rastreadas pelo Git.
4. `git commit`: Cria um novo commit com as alterações rastreadas.
    - Exemplo: `git commit -m "Mensagem do commit"`
    - Caso de uso: Quando você quer salvar um ponto específico no histórico do projeto.
5. `git branch`: Lista, cria ou deleta branches.
    - Exemplo: `git branch` (lista branches), `git branch nova_branch` (cria nova branch), `git branch -d branch_antiga` (deleta branch)
    - Caso de uso: Quando você quer trabalhar em uma nova funcionalidade ou corrigir um bug sem afetar o código principal.
6. `git merge`: Une as alterações de uma branch em outra.
    - Exemplo: `git merge branch_fonte`
    - Caso de uso: Quando você terminou de trabalhar em uma branch e quer unir as alterações na branch principal.
7. `git status`: Mostra o estado atual do repositório.
    - Exemplo: `git status`
    - Caso de uso: Quando você quer ver quais arquivos foram modificados e quais alterações estão prontas para serem commitadas.

Trabalhando com histórico

git-log

O Git mantém um histórico completo de todas as alterações feitas em um projeto. Cada alteração é registrada como um commit, que contém informações sobre as alterações feitas, o autor, a data e um identificador único chamado commit hash.

Para visualizar o histórico de commits, você pode usar o comando `git log`. Isso mostrará uma lista de todos os commits, começando pelo mais recente. Cada commit é identificado pelo seu commit hash.

Além disso, o Git permite que você navegue pelo histórico usando comandos como `git checkout` e `git revert`. O `git checkout` permite que você volte no tempo e trabalhe em versões anteriores do código, enquanto o `git revert` cria um novo commit que desfaz as alterações feitas em um commit anterior.

Outra funcionalidade importante do histórico do Git é o branching. Um branch é uma linha de desenvolvimento separada que permite que você trabalhe em novas funcionalidades ou correções de bugs sem afetar o código principal. Você pode criar um novo branch usando o comando `git branch` e alternar entre branches

usando o comando `git checkout`.

Ao finalizar o trabalho em um branch, você pode unir as alterações na branch principal usando o comando `git merge`. Isso combinará as alterações feitas no branch atual com o branch principal.

Em resumo, o histórico do Git permite que você acompanhe todas as alterações feitas em um projeto, navegue pelo histórico, trabalhe em branches separadas e una as alterações de diferentes branches. Isso facilita o desenvolvimento colaborativo, a experimentação e o gerenciamento de lançamentos.

Commit Hash

hash

Um commit hash é uma sequência única de caracteres gerada pelo sistema de controle de versão para identificar de forma exclusiva um commit específico. É uma combinação de letras e números que representa o estado do código em um determinado momento. O commit hash é usado para referenciar commits ao realizar operações como merge, revert e checkout de versões específicas do código.

Cada commit hash é único e é gerado com base nas alterações feitas no código. Isso significa que mesmo uma pequena alteração no código resultará em um commit hash diferente. O commit hash é uma forma eficiente de identificar e rastrear alterações no código ao longo do tempo.

Ao visualizar o histórico de commits em um repositório, você pode ver o commit hash associado a cada commit. Isso permite que você identifique facilmente commits específicos e acompanhe as alterações feitas no código ao longo do tempo.

O commit hash é uma parte fundamental do sistema de controle de versão e é usado para garantir a integridade do código e rastrear as alterações feitas no projeto.

Resumo

1. A definição do controle de versão em desenvolvimento de software é a prática de rastrear e gerenciar alterações no código-fonte ao longo do tempo. Ele é importante porque permite que os desenvolvedores trabalhem de forma colaborativa, revertam alterações, testem novas funcionalidades e gerenciem lançamentos.
2. Os problemas de desenvolvimento de software resolvidos pelo controle de versão incluem a perda de código, a colaboração entre desenvolvedores, o rastreamento de alterações, a reversão de alterações, o teste de novas funcionalidades e o gerenciamento de lançamentos.

3. As vantagens do Git em relação a outras ferramentas de controle de versão incluem a distribuição, o desempenho, o sistema eficiente de branching e merging, a integridade dos dados e a flexibilidade.

4. As desvantagens do Git incluem a curva de aprendizado íngreme, a complexidade dos comandos, a falta de controle de acesso granular e a dificuldade de lidar com arquivos binários grandes.

5. O Git evoluiu ao longo do tempo em relação a outras ferramentas de controle de versão, introduzindo o modelo de dados que garante a integridade do código e a capacidade de manipular facilmente a história do código.

6. Os comandos básicos do Git incluem git init, git clone, git add, git commit, git push, git pull, git branch, git merge e git status. Eles são usados para inicializar um novo repositório, clonar um repositório existente, adicionar arquivos ao índice, criar um novo commit, enviar alterações para um repositório remoto, obter alterações de um repositório remoto, gerenciar branches, unir alterações de uma branch em outra e verificar o estado atual do repositório.

7. Um commit hash é uma sequência única de caracteres gerada pelo sistema de controle de versão para identificar de forma exclusiva um commit específico. Ele é importante no controle de versão porque permite referenciar commits ao realizar operações como merge, revert e checkout de versões específicas do código.

8. Para trabalhar com o Git em um laboratório prático, você pode seguir os seguintes passos:

    - Crie um novo diretório em seu computador e navegue até ele no terminal.
    - Inicialize um novo repositório Git com `git init`.
    - Crie um novo arquivo ou adicione arquivos existentes ao repositório com `git add`.
    - Crie um novo commit com as alterações rastreadas usando `git commit`.
    - Crie branches para desenvolver novas funcionalidades ou corrigir bugs com `git branch`.
    - Una as alterações de uma branch em outra com `git merge`.
    - Verifique o estado atual do repositório com `git status`.

9. As principais diferenças entre um sistema de controle de versão centralizado e um sistema de controle de versão distribuído são:

    - No sistema centralizado, há um servidor central que mantém todas as versões do código, enquanto no sistema distribuído, cada desenvolvedor tem uma cópia completa do histórico de versões.
    - No sistema centralizado, é necessário estar conectado ao servidor para realizar operações, enquanto no sistema distribuído, é possível trabalhar offline e sincronizar as alterações posteriormente.

- No sistema centralizado, a perda do servidor central pode resultar na perda de todo o histórico de versões, enquanto no sistema distribuído, cada cópia do repositório contém todo o histórico.
- No sistema centralizado, as operações são mais lentas, pois dependem da conexão com o servidor, enquanto no sistema distribuído, as operações são mais rápidas e eficientes.

10. O Git permite o trabalho offline e fornece redundância em caso de falha do servidor central através do seu modelo de dados distribuído. Cada desenvolvedor tem uma cópia completa do histórico de versões, o que permite trabalhar offline e realizar operações localmente. Além disso, as alterações podem ser sincronizadas entre os repositórios locais e remotos, fornecendo redundância em caso de falha do servidor central.

Nota: Este laboratório pressupõe que você está trabalhando localmente e não está fazendo push das alterações para um repositório remoto.

Histórico do Git

histórico-git

O Git mantém um histórico completo de todas as alterações feitas em um projeto. Cada alteração é registrada como um commit, que contém informações sobre as alterações feitas, o autor, a data e um identificador único chamado commit hash. Isso permite que você acompanhe todas as alterações feitas no código ao longo do tempo e navegue pelo histórico usando comandos como `git log`, `git checkout` e `git revert`.