EC – Versionador de Arquivos

Para o estudo de caso, implementaremos um mecanismo de versionamento de arquivos de texto em linguagem C. A lógica de inspiração para esse projeto é o versionador geral de arquivos git.

ETAPA 1

Nesta etapa, deverá ser criado um arquivo executável, capaz de ser utilizado para adicionar arquivos de texto ao versionador de código. O versionador deverá ser executado pela command line e deverá ter os seguintes comandos (nessa nomenclatura, os elementos que estiverem entre parênteses são opcionais - podem ou não ser passados para os comandos - e, a depender de serem ou não passados, mudam o comportamento do comando):

versionador.exe iniciar

Comando que deve criar todos os arquivos necessários para começar um banco de dados de versionamento do folder em que foi executado. Nesse comando, você criará uma pasta oculta chamada .versionador, e dentro dessa pasta oculta estarão todos os arquivos necessários para que as funcionalidades do versionador sejam executadas.

versionador.exe adiciona <arquivo_1>, <arquivo_2>, ..., <arquivo_n>

Ao utilizar o comando adiciona, você deverá ir marcando arquivos que serão adicionados ao próximo snapshot de versão. O comando adiciona pode ser utilizado múltiplas vezes para adicionar múltiplos arquivos manualmente.

versionador.exe registra “Texto”

O comando registra deve criar o snapshot de todos os arquivos marcados com o comando adiciona e adicionar esse snapshot ao banco de dados de versões. Ao criar uma versão, você deve atribuir um número único (um identificador) para esse snapshot dos arquivos.

versionador.exe log (--conteudo)

O comando log deve listar todos os snapshots já feitos, com os respectivos textos. Se a flag –conteudo for passada, o comando log deve listar todos os snapshots já feitos, com os respectivos textos e, para cada snapshot apresentado, o conteúdo dos arquivos registrados naquela versão deve ser exibido.

versionador.exe mostrar <identificador>

O comando deve exibir o texto registrado para aquela versão e todos os textos de todos os arquivos versionados naquela versão.

versionador.exe mudar <identificador>

Esse comando deve passar a exibir os arquivos como estavam na versão <identificador>. Para evitar perder arquivos atuais, você deve salvar um backup dos arquivos que seriam

sobrescritos pela mudança de versão em uma pasta temporária (pois haverá comandos para voltar para o estado atual dos arquivos).

versionador.exe mudar --atual

Reverte os arquivos de uma versão específica para a versão atual dos arquivos. Estejam eles registrados em uma última versão ou não.

ETAPA 2

Antes de começar qualquer implementação a respeito da etapa 2, deve ser criada uma tag no repositório de código do grupo marcando a versão 1.0 como v1.0. Essa tag deve conter uma conversão completamente funcional da ETAPA 1.

A principal feature dessa etapa é a inclusão da possibilidade de trabalhar com branches (ramificações) no versionamento de código. O executável deverá incluir features a respeito da criação e manipulação de ramos (branches) de versionamento. Todos os comandos anteriores deverão lidar com os novos aspectos que os ramos de versionamento trazem. Por exemplo, ao exibir o log, deverá ser necessário exibir a história do ramo atual (não dos outros ramos). Para entender a lógica dos ramos, será necessário estudar em mais detalhes como os ramos funcionam no git.

Tendo isso em vista, serão necessários implementar os seguintes novos comandos:

versionador.exe ramo <nome_ramo>

Será criado um ponteiro representando um novo ramo, que aponta para o último commit deste ramo. Não deve ser possível criar um nome para um ramo repetido pois os nomes dos ramos se tornarão identificadores na árvore de versões de arquivos.

versionador.exe ramo –remover <nome_ramo>

Será necessário remover o ponteiro que representa um ramo

versionador.exe ramo --alterar <nome_ramo> <novo_nome>

Será necessário renomear o nome ponteiro que representa o ramo em questão.

Sobre os comandos anteriores, eles deverão lidar com esses ponteiros que representam os ramos das árvores de registros de versões. Portanto, os comandos anteriores deverão ter o escopo descrito abaixo. Nessa nomenclatura, os elementos que estiverem entre parênteses são opcionais - podem ou não ser passados para os comandos - e, a depender de serem ou não passados, mudam o comportamento do comando:

versionador.exe iniciar

Esse comando deve, a partir de agora, criar um ramo padrão, que todo repositório do versionador de vocês irá conter, que é o ramo cujo ponteiro será chamado “principal”.

versionador.exe mudar (<identificador>)

Além das funcionalidades anteriores, esse comando deve aceitar como <identificador> um nome de um ramo. Ao receber um nome de ponteiro que representa um ramo, as versões de arquivos devem mudar para a última versão do ramo passado como <identificador>. Essa é uma característica do git também. As branches representam o último commit do ramo para os quais elas apontam. Se o <identificador> não for passado, o código deve mudar para o ramo padrão, chamado “principal”.

versionador.exe log (--conteudo) (<identificador>)

O comando deve passar a ter um comportamento voltado para os ramos. Se algum <identificador> for passado, será necessário mostrar toda a história de versões do ramo representado pelo <identificador>. Se não for passado, será necessário mostrar toda a história de versões do ramo atualmente selecionado. A flag –conteudo terá o mesmo comportamento anterior, se passada, mas também seguirá a lógica orientada ao ramo especificado.

REQUISITOS

- **Para compreender a lógica desse versionador, é necessário entender o git**. Para tal, faça um estudo do livro do git, o Git Pro, que é um livro de leitura muito fácil. Aprenda a usar o versionador com base nos tutoriais dos capítulos 1, 2, 3 e 7. Crie um repositório vazio com o git e experimente o versionador em vários arquivos de texto gerando múltiplas versões com commits, experimentando versionamentos paralelos com branches. Experimente todas as features relatadas nos tutoriais dos capítulos 1, 2, 3 e 7 do libro Git Pro.

- **O código vai ter que estar totalmente modularizado**: cada funcionalidade tem que estar implementada em uma função separada e funções só realizam o que elas foram designadas para realizar. Por exemplo, uma função de leitura de vetor, só deve ler o vetor. Uma função de cálculo da média de um vetor deve calcular a média de um vetor que já foi lido, e assim sucessivamente. Eventualmente pode ser necessário criar interfaces de funções, caso o código fique muito extenso.

- **O código deve estar todo documentando**: cada função deve ter uma descrição breve (como comentário) do que ela realiza além de uma descrição do que cada argumento da função significa. Seguir o padrão doxigen de documentação e gerar uma documentação no final.

- **O código deve ser legível**: os nomes das variáveis terão de ser significativos do papel delas em cada uma das funções e os nomes das funções devem representar bem o que as funções realizam.

- **O banco de dados do versionador será composto por um conjunto arquivos**:

  - Um **arquivo de versões** guardará informações sobre a versão e quais os arquivos que foram registrados em cada versão. O arquivo de versões deve guardar informação a respeito de como ler os dados do conteúdo dos arquivos correspondentes àquela versão. Todavia, não deve guardar o conteúdo dos arquivos. Na versão final
  - **Um arquivo de conteúdo das versões**. Este outro arquivo guardará o conteúdo dos arquivos versionados. O arquivo de versões deve guardar informação a respeito de como ler os dados do conteúdo dos arquivos correspondentes àquela versão.
  - **Os arquivos que compõe as referências do banco de dados são esses**. **Todavia, o grupo terá de lidar com a manipulação de ramos (branches), o que pode impor a**

necessidade de mais arquivos de metadados para estruturar o banco de dados do versionador. Isso para referenciar todos os ramos existentes, quais ramos da árvore de commits são apontados por esses ponteiros, de onde os commits nascem, para onde atualmente estamos olhando na árvores (que é similar ao comportamento do ponteiro HEAD do git). Além disso, o arquivo de versões pode ter que ser reestruturado também para o novo conjunto de features. Fica à escolha do grupo o design interno da implementação do versionador.

- **O projeto deve utilizar explicitamente estruturas de dados:**

- Na primeira etapa foi exercitado o conceito de listas encadeadas. Assim, ao realizar cada comando, antes de qualquer coisa, era necessário criar uma lista encadeada das versões, com os ponteiros para o conteúdo dos arquivos. Ao terminar de executar o comando, a lista deveria ser desalocada. Ou seja, todas as operações deveriam ser feitas com base na análise da lista encadeada de versões com ponteiros para os conteúdos dos arquivos.
- Na segunda etapa, será necessário estender o conceito de estruturas de dados com a utilização de árvores de commits. A funcionalidade de branches é basicamente a extensão da lista de versões para a noção de árvore de versões que o git tem. Portanto, aplicam-se os mesmos requisitos da ETAPA 1 para a ETAPA 2. Todavia, dessa vez, será necessário manipular árvores.

- **O projeto deverá ser versionado em um repositório git no Github (colocar o repositório privado):**

- Cada commit deverá conter uma versão funcionando do seu projeto.
- Para facilitar a análise, cada commit deverá conter uma versão funcionando com uma nova funcionalidade da command line. Por exemplo, o primeiro commit do seu repositório de código deverá implementar uma versão do "versionador" que executa corretamente o comando "iniciar". O próximo commit deverá ter uma versão funcionando do seu "versionador" com o comando "adiciona". E assim sucessivamente.
- A ETAPA 1 deve ser marcada com uma tag no repositório de código, conforme explicitado no início desta seção.
- Para entrega do projeto, será necessário entregar esse código versionado no github (compartilhar com o professor).

AVALIAÇÃO

- Ao término da versão para a ETAPA 2, deverá ser criada uma nova tag no Github do grupo chamada v2.0 para marcar a nova grande versão do código de vocês.
- A avaliação dos códigos será feita por apresentação/entrevista aos grupos. Os integrantes deverão apresentar todo o design escolhido para o código, quais caminhos foram seguidos para a implementação do versionador, como o código foi modularizado, quais arquivos tem quais papeis, qual o papel das funções de cada um dos arquivos, como o processo de compilação e geração do executável é realizado, como estão as dependências entre os arquivos do código, dentre outros aspectos técnicos;

    - Todos os integrantes do grupo devem mostrar COMPLETO DOMÍNIO do código, tanto na v1.0 quanto na v2.0;
    - Poderá ser solicitado que integrantes debuggem o código com o gdb para mostrar cenários específicos de funcionamento do código e comportamento das funções, das estruturas de dados durante a entrevista.
    - Os integrantes deverão ter a capacidade analítica de dissertar sobre possíveis variações do código, e quais os impactos no comportamento do código (demonstrando o referido domínio sobre o código).
- A avaliação poderá ser feita em grupo ou individualmente;
    - Se o grupo optar por avaliação individual, cada elemento do grupo será entrevistado e sabatinado individualmente;
    - Se o grupo optar por avaliação em conjunto, apenas uma entrevista com todos será feita. Todavia, todos os elementos do grupo serão entrevistados em pontos aleatórios do código e se um elemento do grupo mostrar desconhecimento do código, a pontuação geral do grupo irá ser penalizada.
    - No caso de avaliação em conjunto, a avaliação durará 30 minutos para o grupo todo. No caso de avaliação individual, cada avaliação individual durará 20 minutos com cada pessoa.

ETAPA BÔNUS

Aplicam-se todos os mecanismos de avaliação e requisitos para esta etapa. Caso o grupo tenha uma versão COMPLETAMENTE FUNCIONAL da ETAPA 2, os integrantes poderão optar por implementar uma interface gráfica do versionador, utilizando o QTCreator (C++) ou alguma outra ferramenta gráfica de interesse do grupo e que interaja com códigos C. O grupo terá liberdade para as escolhas do design do versionador, que deverá funcionar tanto pela command line, quanto pela interface. Na interface serão observados os seguintes elementos:

- ***Todas as funcionalidades devem ser facilmente utilizáveis com uma interface amigável, demandando pouco ou nenhum conhecimento da command line para quem optar por utilizar a interface;***
- ***O design da interface deve ser pensado de forma a fazer com que a utilização do versionador seja o mais simples possível;***
- ***Caso o grupo escolha pela versão completa e gráfica do sistema, será estendido o prazo de 10 minutos na apresentação em conjunto para que as funcionalidades da interface sejam apresentadas. No caso de avaliação individual, cada integrante terá adicionalmente 5 minutos de entrevista para mostrar a usabilidade da interface (além do conhecimento do código da interface);***
- ***O repositório de código deve ser marcado com uma tag v3.0 nesse caso. O grupo que apresentar uma interface amigável, com design bem trabalhado para o versionador da v2.0 terá 2 pontos extras de nota no estudo de caso. Caso a nota do A2 do aluno alcance os 10 pontos com a nota extra a diferença será transferida para o primeiro bimestre;***
- ***Vale destacar que além de ter pontuação extra, um sistema completo desses facilmente pode compor o portfólio individual de projetos de cada um de vocês. Algo***

bastante interessante para quem quer demonstrar domínio de programação e de design de interfaces.