



Depuração e Versionamento

Unidade 3 | Capítulo 3

Prof. Jorge Wattes



Executores:



Coordenação:



Iniciativa:



Sumário

Introdução	3
Depuração	4
Versionamento	11
Em Resumo	19
Conclusão	22
Referências	23

Introdução: Depuração e versionamento

Olá, estudantes! Sejam bem-vindos ao material de apoio “Depuração e Versionamento”. O objetivo deste material é servir como complemento para o Capítulo 3 da Unidade 3 do curso EmbarcaTech.

Como vocês já sabem, as ferramentas de depuração e versionamento são essenciais no cotidiano de um desenvolvedor de software. A depuração permite corrigir defeitos em um programa por meio da execução de um código linha por linha, monitorando os valores das variáveis em tempo real. Essa técnica é crucial para a identificação e correção de erros de programação, os chamados ‘bugs’. Um bom programador utiliza a depuração para melhorar a qualidade e a estabilidade de seus programas.

Por outro lado, as ferramentas de versionamento nos permitem gerenciar de maneira eficiente as alterações nos arquivos de nossos projetos, facilitando o rastreamento das mudanças e promovendo o trabalho colaborativo.

Neste material de apoio, exploraremos mais a fundo os conceitos de depuração e versionamento, além de apresentar exemplos práticos de uso dessas ferramentas.

Começaremos com a apresentação de técnicas de depuração, seguida por um exemplo prático de uso das **ferramentas de depuração do VS Code** em um programa escrito em C.

Também abordaremos os **principais conceitos** de versionamento, introduzindo termos importantes como repositórios, **commits** e **branches**, entre outros. Por fim, apresentaremos um exemplo prático de uso do Git, a ferramenta de versionamento mais utilizada atualmente para projetos de software.

Lembre-se de revisar todo o material deste capítulo com atenção. Não se esqueça de realizar as atividades propostas na plataforma e participar dos momentos síncronos.

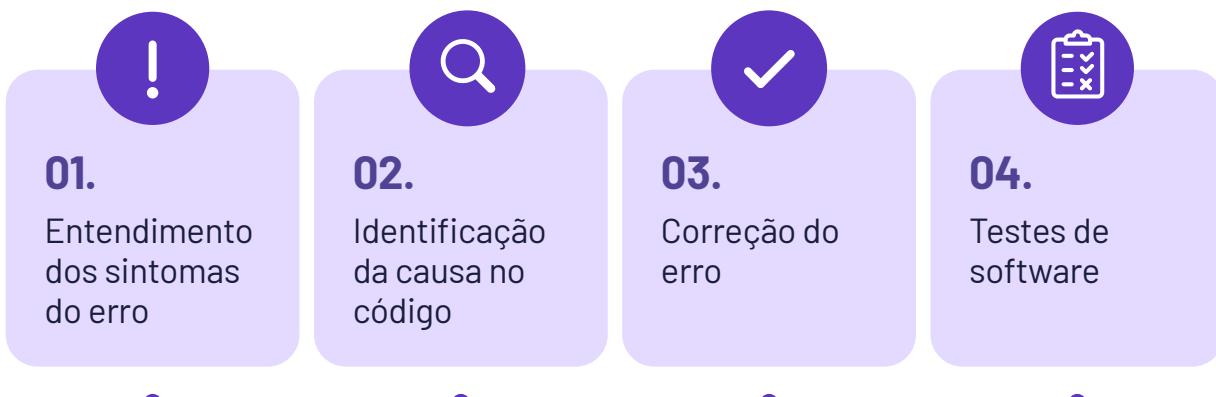
Esta é sua oportunidade de compreender o funcionamento de duas ferramentas essenciais para o desenvolvimento de software. Contamos com sua participação ativa nessa jornada. Vamos em frente!

Depuração

De maneira simples, a **depuração é um processo sistemático de identificação e remoção de defeitos encontrados em um software**. Ela é parte essencial da vida de um programador e um dos pilares para assegurar a qualidade de um produto de software.

As ferramentas de depuração permitem que um programa seja executado em um modo especial, no qual o programador consegue acompanhar a execução de uma região específica do código, enquanto monitora o valor das variáveis. Esse processo possibilita verificar o que o programa realmente está fazendo e ajustar eventuais erros.

Geralmente, ao encontrar um bug, o programador segue um roteiro simples de atividades para corrigi-lo.



O primeiro passo é analisar os sintomas do bug, o que pode ser feito a partir de relatórios de testes, logs de execução ou observações pessoais. Após entender o impacto do bug no sistema, é necessário seguir o fluxo de execução do código, utilizando ferramentas de depuração para identificar a região que está causando o comportamento indesejado. Em seguida, deve-se isolar as partes do código envolvidas até encontrar a linha ou seção

responsável pelo bug. Por fim, os ajustes são feitos no código, e o sistema é testado novamente para garantir que o erro foi corrigido.

Esse procedimento pode ser resumido em quatro etapas: entendimento dos sintomas do erro; identificação da causa no código; correção do erro; e testes de software.

Embora esse procedimento seja sistemático, nem sempre é suficiente para resolver um bug. Alguns casos são mais complexos e exigem estratégias alternativas. Por exemplo, a **depuração em pares**, na qual um colega programador é convidado a ajudar na resolução do problema.

Em outros casos, o bug pode ser difícil de reproduzir, o que torna o processo mais demorado. De qualquer forma, para facilitar a depuração, é importante que a equipe de desenvolvimento siga boas práticas de programação, gerando um código de fácil compreensão, o chamado '**Clean Code**'. O processo de depuração começa com a aplicação adequada dessas técnicas desde o início do projeto.

Para compreendermos melhor como funciona a execução de um programa no modo de depuração, vamos, na próxima seção, abordar um exemplo prático de uso das ferramentas de depuração do VS Code em um código C. É importante que você reproduza esse exemplo em casa e tire suas dúvidas durante as aulas síncronas.

- NA PRÁTICA

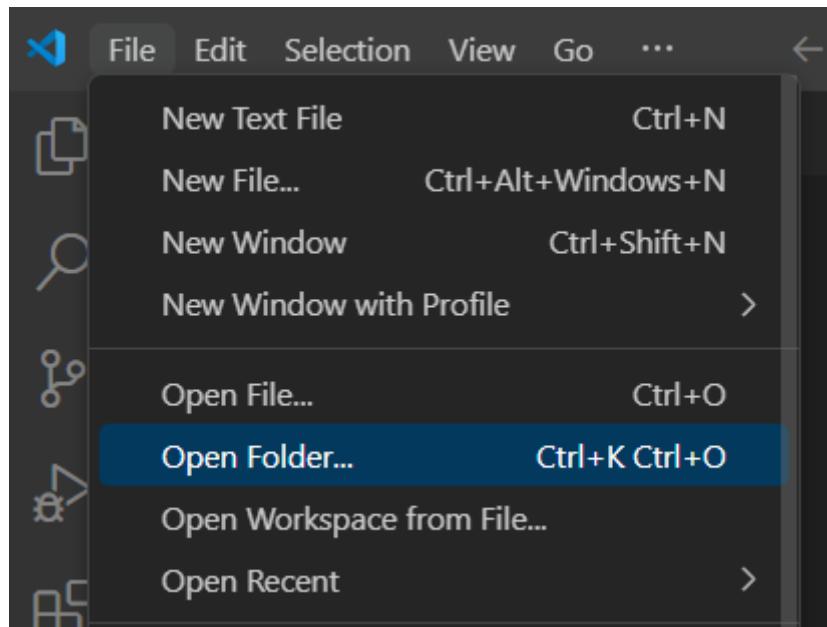
Depurando um programa C no VS Code

Vamos apresentar um exemplo prático de execução de um programa C no modo de depuração [1]. Para isso, utilizaremos o ambiente de desenvolvimento VS Code. Para seguir os passos descritos nesta seção, é necessário que você já tenha o VS Code instalado e configurado com

um ambiente de desenvolvimento para linguagem C. Caso você ainda não tenha um sistema configurado, releia o material do capítulo 2.

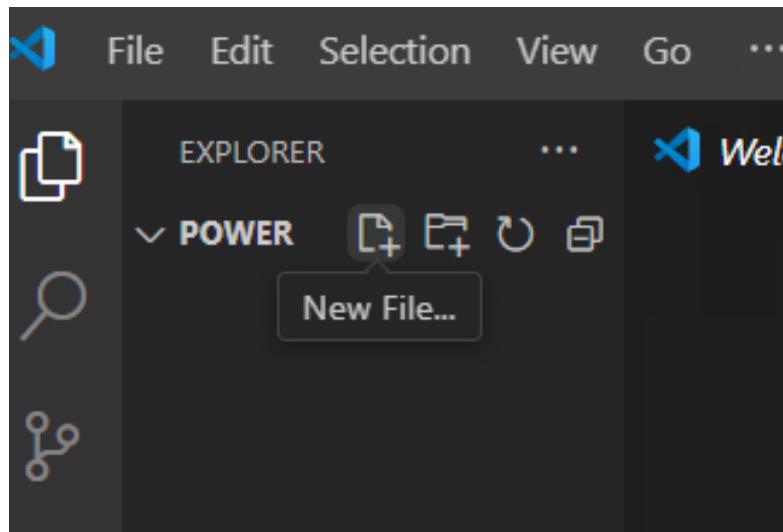
1. Selecionando um diretório de projeto

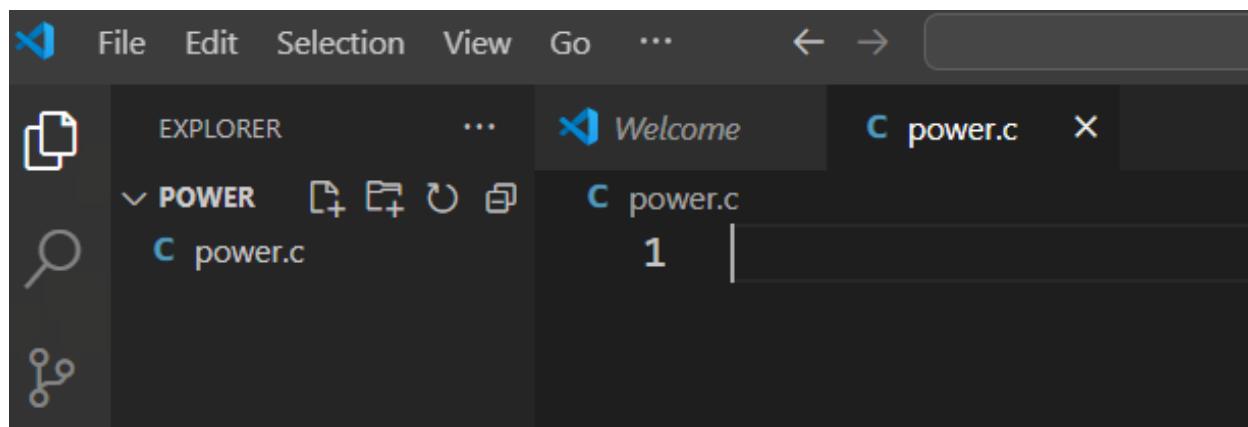
Inicie o VS Code e selecione um diretório para criar nosso projeto. Isso pode ser feito utilizando o atalho **CTRL+K CTRL+O** (pressione as teclas Ctrl, K e O simultaneamente) ou acessando o menu **File** e escolhendo a opção **Open Folder**.



2. Adição do programa

Crie um novo arquivo e nomeie-o como `power.c`.





3. Código do programa

Adicione o código abaixo no arquivo `power.c`. Esse programa lê dois números inteiros e calcula a potência do primeiro número elevado ao segundo.

O cálculo da potência é realizado em um laço que multiplica o primeiro número por ele mesmo repetidas vezes. Por fim, o valor é impresso na tela. Leia com cuidado com o código para entender seu funcionamento.

```

#include <stdio.h>

int main()
{
    int a, b, pow = 1;

    printf("Digite o primeiro numero: ");
    scanf("%d", &a);
    printf("Digite o segundo numero: ");
    scanf("%d", &b);

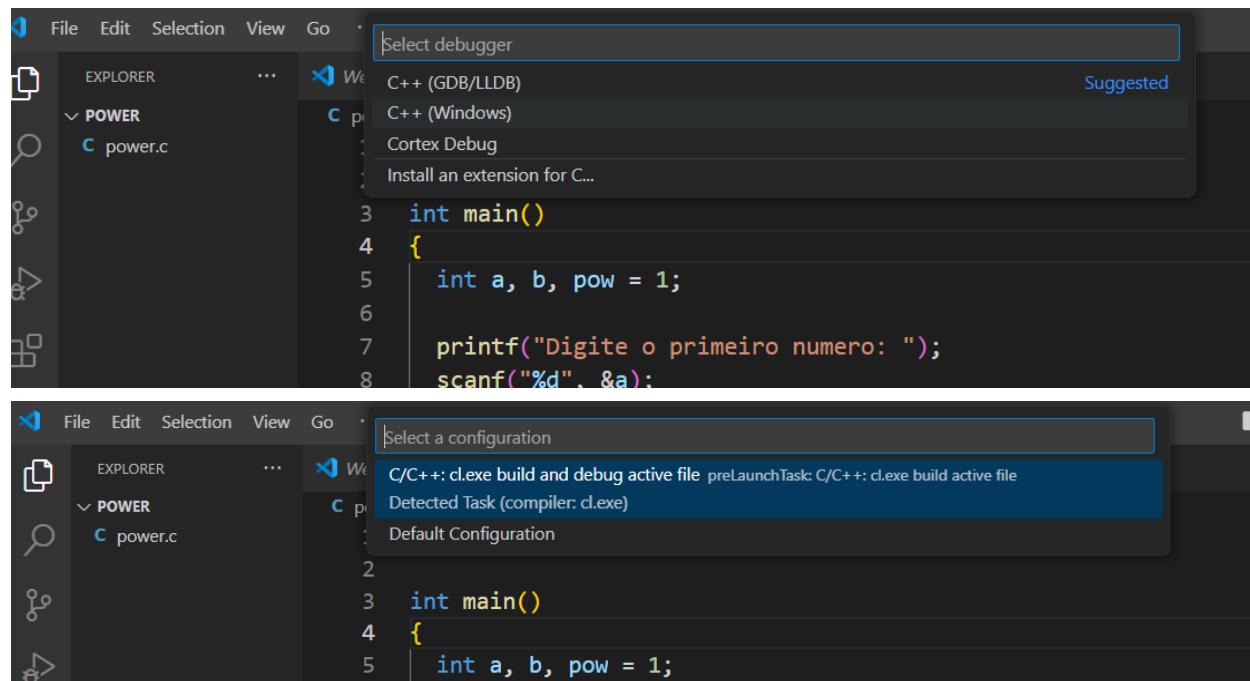
    for (int i = 1; i <= b; i++)
    {
        pow = pow * a;
    }

    printf("%d", a);
}
  
```

The screenshot shows the full content of the 'power.c' program in the code editor. The code includes #include "stdio.h", a main function that reads two integers from the user, and a loop that calculates the power by multiplying the base (a) by itself b times. The result is then printed to the screen.

4. Iniciando o ambiente de depuração

Para iniciar o ambiente de depuração, basta usar o atalho F5 e selecionar o depurador instalado. Neste exemplo, estamos utilizando o depurador do compilador MSVC para Windows, mas as etapas descritas aqui são aplicáveis a qualquer ferramenta de depuração.



5. Executando a depuração

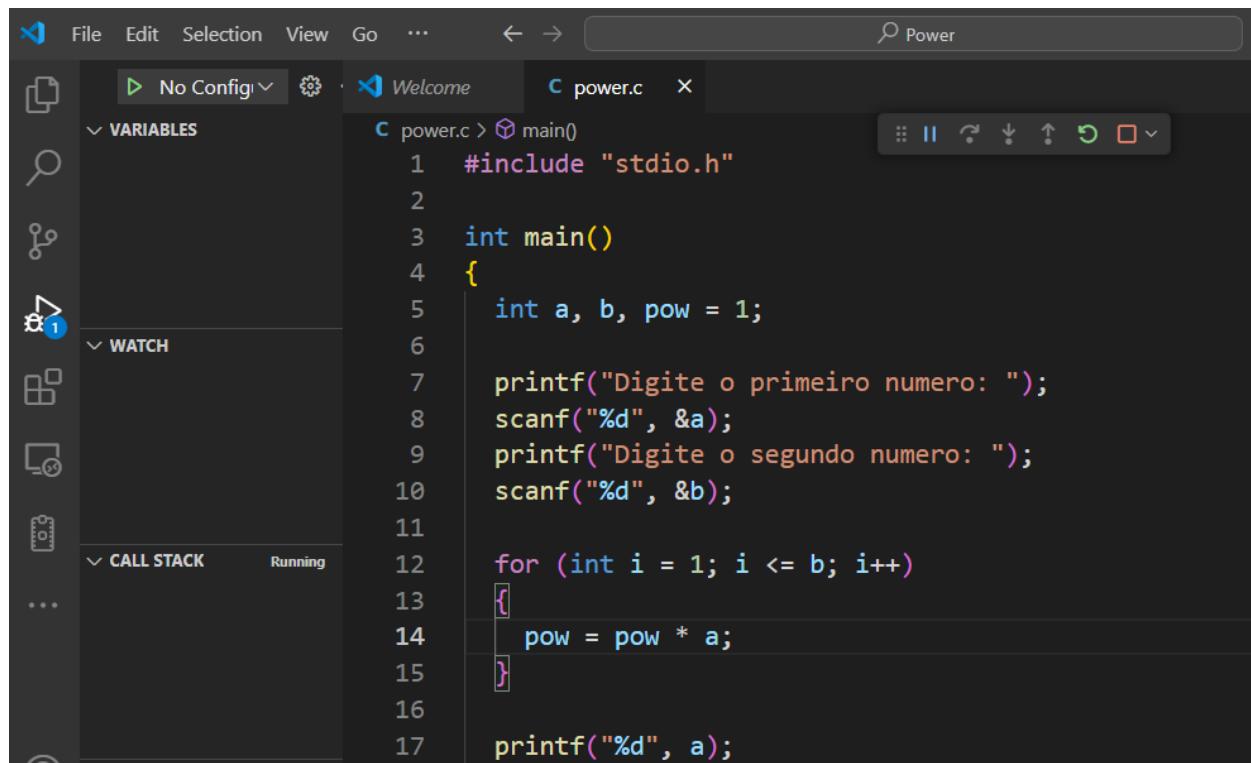
Após a seleção da ferramenta, o VS Code [2] iniciará o ambiente de depuração. Você verá novas seções na tela e uma barra com as ferramentas de depuração.

No ambiente de depuração, é possível visualizar as variáveis, criar regras de monitoramento, ver a pilha de execução de funções e gerenciar os breakpoints. Agora, vamos falar um pouco sobre as operações disponíveis para controlar a execução de um programa durante a depuração. Na Debug Toolbar, que é a barra de ferramentas de depuração, encontramos comandos para continuar ou pausar a execução. Durante a depuração, é possível executar o programa passo a passo, e, para isso, temos alguns comandos importantes.

O comando **Step Over** permite que você execute a próxima função como se fosse um comando simples, sem entrar nos detalhes da execução.

dessa função. Já o comando **Step Into** permite acompanhar a execução da próxima função linha por linha, entrando nos detalhes do código. O comando **Step Out** é utilizado quando você deseja pular o restante da execução de uma função e ir diretamente para o ponto de retorno.

Além disso, temos o comando **Restart**, que, como o nome sugere, reinicia a depuração, e o comando **Stop**, que finaliza a execução do programa.



The screenshot shows a debugger interface with a dark theme. On the left is a sidebar with icons for file operations, configuration, variables, watch, and call stack. The main area shows a file named 'power.c' with the following code:

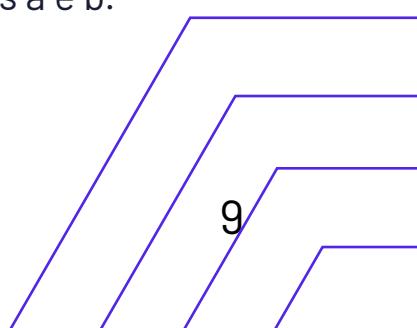
```

1 #include <stdio.h>
2
3 int main()
4 {
5     int a, b, pow = 1;
6
7     printf("Digite o primeiro numero: ");
8     scanf("%d", &a);
9     printf("Digite o segundo numero: ");
10    scanf("%d", &b);
11
12    for (int i = 1; i <= b; i++)
13    {
14        pow = pow * a;
15    }
16
17    printf("%d", a);

```

6. Adicionando um breakpoint

Um dos recursos mais importantes das ferramentas de depuração é o famoso **breakpoint**, ou ponto de parada. Imagine que você precisa depurar um código muito extenso; executar linha por linha não seria viável nesse caso. Para adicionar um breakpoint, basta clicar com o botão esquerdo do mouse próximo à numeração da linha de código na qual você deseja pausar a execução. Após inserir o breakpoint, reinicie a depuração clicando em **Restart** e verifique se a execução é pausada no ponto que você determinou. Observe, ao lado, os valores das variáveis **a** e **b**.



The screenshot shows the Clion IDE interface. On the left, the 'WATCH' pane is open, displaying a logical expression `i==4 = false`. In the center, the code editor shows a C program named 'power.c' with the following code:

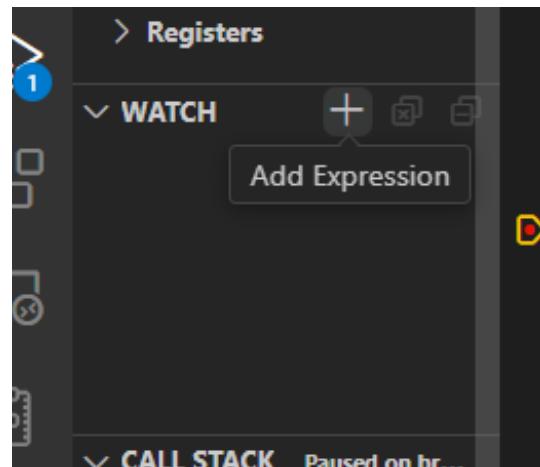
```

1 #include <stdio.h>
2
3 int main()
4 {
5     int a, b, pow = 1;
6
7     printf("Digite o primeiro numero: ");
8     scanf("%d", &a);
9     printf("Digite o segundo numero: ");
10    scanf("%d", &b);

```

7. Adicionando uma expressão no Watch

A seção Watch permite que adicionemos expressões lógicas que são monitoradas ao longo da depuração. Vamos adicionar uma expressão para identificar quando a variável i assume o valor 4. Adicione um breakpoint na linha 14 para acompanhar a execução do laço for passo a passo. Perceba que a expressão lógica é automaticamente avaliada, permitindo que rapidamente saibamos sobre condições importantes do estado de execução do programa.



The screenshot shows the Clion IDE during step-by-step execution. The Watch pane now contains the expression `i==4 = false`. The code editor highlights the line `for (int i = 1; i <= b; i++)`, which is currently being executed. The Call Stack pane indicates the program is 'Paused on step'.

As ferramentas que apresentamos aqui são muito importantes no processo de depuração. A execução passo a passo do programa e o monitoramento das variáveis permitem que tenhamos ciência de cada etapa da execução. O código que usamos como exemplo nesta seção contém um erro. Você foi capaz de encontrá-lo? Tente seguir o procedimento apresentado na seção anterior para identificá-lo.

Você lembra o que significa...

O ‘Clean Code’ é uma filosofia de desenvolvimento de software que consiste em aplicar técnicas simples que facilitam a escrita e a leitura de um código. Dessa forma, o código torna-se mais fácil de entender, manter e evoluir. Procure se aprofundar nessa técnica para produzir códigos melhores e mais fáceis de depurar.

• SAIBA MAIS

O ‘Clean Code’ é uma filosofia de desenvolvimento de software que consiste em aplicar técnicas simples que facilitam a escrita e a leitura de um código. Dessa forma, o código torna-se mais fácil de entender, manter e evoluir. Procure se aprofundar nessa técnica para produzir códigos melhores e mais fáceis de depurar.

O teste unitário é uma parte essencial da estratégia de testes de software. Basicamente, ele verifica, de forma automática, o funcionamento de uma pequena região de código isolada do sistema. Esses testes devem ser rápidos e independentes. O uso de testes unitários ajuda na identificação precoce de erros e contribui para a qualidade final do software.

Versionamento

Um sistema de versionamento **registra as mudanças ocorridas em um arquivo ou grupo de arquivos ao longo do tempo, permitindo que versões específicas sejam recuperadas.**

Muitas pessoas optam por adotar um “sistema manual de versionamento”, no qual elas mesmas salvam, periodicamente, seus arquivos em diretórios nomeados como “Versão 1”, “Versão 2”, e assim por diante. No entanto, essa estratégia não é adequada para os dias de hoje.

Atualmente, existem softwares que automatizam o versionamento de arquivos. O software mais utilizado nessa área é o **Git [3]**, amplamente adotado na maioria dos projetos de software. Nos próximos parágrafos, vamos definir alguns conceitos-chave desse sistema.



Para trabalhar com o Git, é essencial compreender alguns conceitos básicos. O primeiro deles é o conceito de **repositório**, que é um diretório onde armazenamos os arquivos de um projeto.



Normalmente, esse repositório é hospedado na nuvem em serviços como GitHub, GitLab, entre outros. Entretanto, não trabalhamos diretamente no repositório remoto; precisamos de uma cópia local, chamada repositório local, onde podemos fazer nossas modificações.

No repositório local, podemos **alterar, criar e remover** arquivos à medida que o projeto evolui. Para salvar essas mudanças, utilizamos o **commit**, que funciona como um ponto de salvamento do nosso projeto [4].

Os commits são realizados no repositório local. Quando queremos sincronizar as alterações entre os repositórios, usamos dois comandos principais: o **push**, que envia as modificações locais para o repositório remoto, e o **pull**, que traz as mudanças do repositório remoto para o local.

Vamos nos aprofundar um pouco mais no conceito de **commit no Git**. O Git armazena versões comprimidas dos arquivos chamadas **snapshots**, que são salvas quando realizamos um commit.

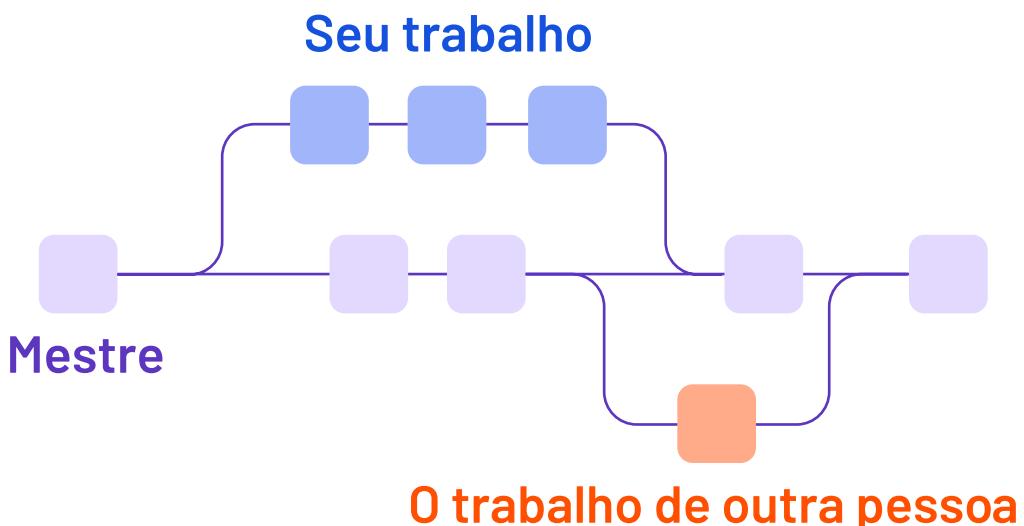
Diferentemente de outros sistemas de versionamento que armazenavam apenas as diferenças entre as versões, o Git é realmente eficiente, pois salva apenas os arquivos que sofreram mudanças entre commits.

Outro conceito importante é a capacidade de criar linhas de desenvolvimento paralelas, conhecidas como **branches**.

Grosso modo, os branches permitem a criação de várias cadeias de commits dentro de um projeto, como ilustrado na figura abaixo [5]. Os branches possibilitam que vários programadores trabalhem simultaneamente no mesmo projeto, cada um em sua branch.

Em determinados pontos do desenvolvimento, os programadores podem fundir suas modificações com a branch principal (geralmente chamada de **master** ou **main**) usando o comando **git merge**.

É uma boa prática de projeto submeter uma requisição de merge (**Merge Request**) para o responsável pelo repositório, permitindo que as modificações sejam avaliadas antes de serem incorporadas ao projeto principal.



Comandos do Git

Agora, vamos falar sobre alguns comandos do Git que usamos no dia a dia. Você encontrará mais detalhes sobre esses comandos na folha de dicas do **GitHub**, disponível na seção de referências.

Quando iniciamos um projeto, precisamos criar nosso repositório. Existem duas formas de fazer isso: com o comando **git init**, criamos um

repositório novo em folha. Já se quisermos criar um repositório a partir de um existente em um repositório remoto, usamos o comando **git clone**.

Para salvar as mudanças no nosso repositório, utilizamos o comando **git commit**, que cria nossos **savepoints**. Antes disso, para adicionar novos arquivos ao controle de versão, usamos o comando **git add**. Esse comando coloca os arquivos no **staging environment** (ambiente de preparação), que é uma área intermediária onde as mudanças ficam aguardando para serem confirmadas.

É importante lembrar que as alterações só são efetivamente salvas no histórico do Git após a execução de um `git commit`, que move as mudanças do ambiente de preparação para o repositório local.

Continuando nosso panorama sobre comandos do Git, vamos explorar como gerenciar branches e sincronizar repositórios locais e remotos.

O comando **git branch** lista as branches existentes em seu repositório. Para criar uma nova branch, basta usar `git branch <nome-da-branch>`, substituindo `<nome-da-branch>` pelo nome desejado.

Para fazer um merge entre a branch atual e outra, utilize o comando `git merge <nome-da-branch>`, substituindo `<nome-da-branch>` pelo nome da branch que você deseja integrar.

O comando `git push` atualiza o repositório remoto com as mudanças do repositório local. Já o **git pull** faz o oposto: ele atualiza o repositório local com as mudanças do remoto. Esses comandos são essenciais para gerenciar a evolução do projeto e garantir que todos os colaboradores estejam trabalhando com a versão mais atualizada do código.

- NA PRÁTICA

Versionamento com Git

Nesta seção, vamos apresentar um exemplo prático de uso dos comandos do Git. Tente seguir esses passos em seu computador. Lembre-se

de tirar suas dúvidas com o seu mentor durante os encontros síncronos. Vamos lá!

01.

Instale o Git

O primeiro passo é instalar o Git em sua máquina e criar uma conta no GitHub. Deixamos um guia de instalação nas referências.

02.

Crie um novo repositório

Ao iniciar um novo projeto em sua máquina usando Git, o primeiro passo é criar um novo repositório. Utilize o prompt de comando do Windows ou o terminal do Linux, navegue até a pasta onde deseja criar o repositório e execute o comando `git init`.

03.

Adicione um novo arquivo no repositório

Crie um novo arquivo na pasta, por exemplo, `embarca.txt`. Em seguida, use o comando `git add` para adicionar esse arquivo ao ambiente de preparação (staging environment).

04.

Vamos ‘comitar’ nosso arquivo

Execute o comando `git commit -m "Meu primeiro commit"`. Normalmente, a mensagem de um commit deve estar relacionada ao seu objetivo, como a implementação de uma nova funcionalidade, uma correção, entre outros.

05.

Primeiro branch

Para criar uma nova branch e mudar imediatamente para ela, execute o comando `git checkout -b minha_branch`, onde `minha_branch` é o nome da branch que você acabou de criar. Você pode confirmar a criação da branch executando o comando `git branch`, que listará todas as branches do repositório.

06.

Crie um repositório no GitHub

Acesse o GitHub, faça login e crie um novo repositório.

The screenshot shows the GitHub desktop application interface. At the top is a search bar with placeholder text "Type ⌘ to search". To its right are several icons: a plus sign for creating new items, a circular arrow for repository status, a double arrow for cloning, a folder for organization, and a user icon. Below the search bar is a "Filter" button. On the left, there's a sidebar with a "Star" button and a dropdown menu. A central modal window is open, titled "UNIVERSE'24" in large bold letters. It contains the text "Your GitHub Copilot questions, answered." and "Learn how to ship more and context switch less at the world's fair of software. Get 20% off your tickets to Universe, only until September 3." At the bottom of the modal is a "Get tickets" button. To the right of the modal is a vertical sidebar with options: "New repository", "Import repository", "New codespace", "New gist", "New organization", and "New project".

Preencham o formulário com as informações do seu novo repositório.

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk ().*

Owner *

glober75 /

Great repository names are short and memorable. Need inspiration? How about [literate-waddle](#) ?

Description (optional)

Public

Anyone on the internet can see this repository. You choose who can commit.

Private

You choose who can see and commit to this repository.

Initialize this repository with:

Add a README file

This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

.gitignore template: None ▾

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

License: None ▾

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

You are creating a public repository in your personal account.

Create repository

O GitHub perguntará se você deseja criar um repositório do zero ou adicionar um repositório local. Neste caso, escolha a segunda opção. Execute os seguintes comandos: git remote add origin https://github.com/repo, onde https://github.com/repo é o endereço do seu repositório. Em seguida, execute git push -u origin master para que o conteúdo do seu repositório local seja enviado ao repositório remoto.

07.

Envie sua branch para o GitHub

Você pode enviar o conteúdo da branch que criou para o repositório remoto usando o comando git push origin minhabranch, onde minhabranch é o nome da branch que você criou.

08.

Criando um Merge Request

No GitHub, os Merge Requests são chamados de Pull Requests (PR). Eles permitem que você notifique o proprietário do repositório sobre a atualização que deseja fazer. Preencha o formulário e faça a requisição.

Open a pull request

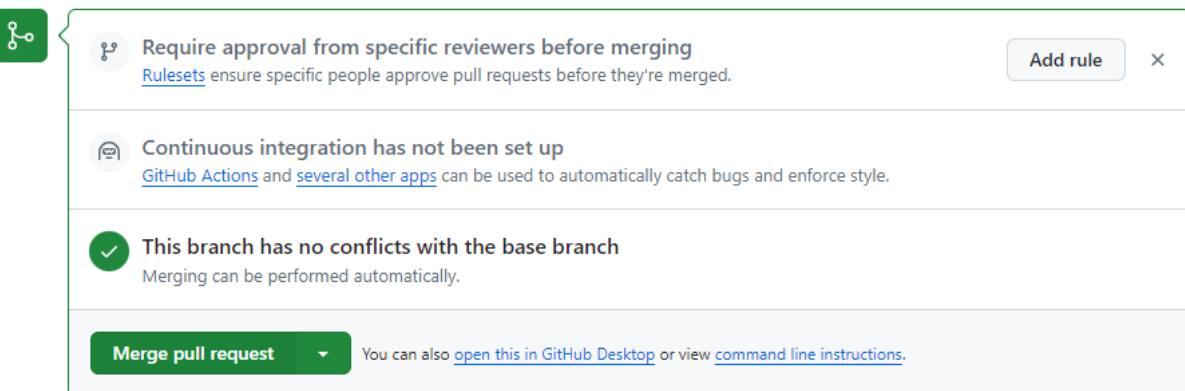
Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks. Learn more about diff con

The screenshot shows the GitHub interface for creating a pull request. At the top, there are dropdown menus for 'base: main' and 'compare: develop'. A green checkmark indicates 'Able to merge. These branches can be automatically merged.' Below this, there's a 'Add a title' field containing 'Develop' and a 'Add a description' field with a rich text editor. The rich text editor has tabs for 'Write' and 'Preview', and a toolbar with various formatting icons like bold, italic, and code blocks. A placeholder text 'Add your description here...' is visible in the editor area.

09.

Aceitando um Merge Request

Após fazer a requisição, como proprietário do repositório, você pode avaliar o pedido e decidir se as mudanças serão integradas à sua branch principal. Quando estiver pronto, aceite a solicitação clicando em ‘Merge pull request’.

**10.**

Atualizando seu repositório local

Após as mudanças serem aceitas, o repositório remoto estará diferente do seu repositório local. Você pode sincronizar seu repositório local usando o comando git pull origin master.

Esse exemplo prático nos oferece uma visão geral de como utilizar o Git. Contudo, apenas o uso contínuo da ferramenta permitirá que você comprehenda plenamente seu funcionamento. Em suas próximas atividades práticas neste curso, adote o uso do Git e do GitHub para melhorar o controle das versões de seus projetos.

Em resumo

Clean Code

Clean Code se refere a um código que é fácil de entender, manter e modificar. Geralmente é escrito de maneira clara e concisa, seguindo boas práticas de programação.

Teste Unitário

O teste unitário é uma técnica de teste de software onde partes individuais de código, chamadas unidades, são testadas isoladamente para garantir que funcionam corretamente.

GitHub

GitHub é uma plataforma de hospedagem de código-fonte que utiliza o sistema de controle de versão Git. Permite que desenvolvedores colaborem em projetos de software de forma eficiente.

git init

O comando git init inicializa um novo repositório Git vazio ou re-inicializa um repositório existente em um diretório.

git clone

Git clone é um comando que cria uma cópia de um repositório remoto em sua máquina local, incluindo todos os arquivos, commits e branches.

git commit

O comando git commit é usado para salvar as mudanças na história do repositório. Um commit captura o estado do projeto em um momento específico.

Savepoints

Savepoints são pontos de salvamento no desenvolvimento de software, que permitem restaurar o estado do projeto a um ponto anterior, similar a commits no Git.

git add

O comando git add adiciona mudanças no diretório de trabalho ao staging area, preparando-as para o próximo commit.

Staging Environment

Um staging environment (ambiente de teste) é uma cópia do ambiente de produção onde novos códigos ou mudanças são testados antes de serem implementados definitivamente.

git branch

O comando git branch cria, lista ou exclui branches no seu repositório. Branches permitem o desenvolvimento paralelo de diferentes funcionalidades.

git pull

Git pull é um comando que atualiza o repositório local com mudanças do repositório remoto, combinando os commits remotos com o branch local.

Git

Git é um sistema de controle de versão distribuído que permite a vários desenvolvedores trabalharem no mesmo projeto sem conflitos.

Repositório

Um repositório é um local onde todos os arquivos de um projeto são armazenados, incluindo o histórico de revisões e as mudanças feitas ao longo do tempo.

Commit

Commit é um instantâneo das mudanças feitas no código, incluindo uma mensagem descritiva sobre o que foi alterado.

Push

Push é o comando que envia commits feitos localmente para um repositório remoto.

Pull

Pull é o comando que busca e integra mudanças do repositório remoto no repositório local.

Commit no Git

Commit no Git é um instantâneo do repositório em um momento específico, usado para registrar as mudanças feitas no código.

Snapshots

Snapshots são capturas instantâneas do estado de um repositório em um momento específico, similares a commits no Git.

Branches

Branches são versões paralelas e independentes do repositório, permitindo o desenvolvimento de diferentes funcionalidades ao mesmo tempo.

Master/Main

Master ou Main são os branches principais de um repositório Git, onde geralmente a versão estável e funcional do projeto é mantida.

git merge

Git merge é o comando que combina mudanças de um branch com outro, unificando os históricos de commits.

Merge Request

Merge Request é um pedido para integrar mudanças feitas em um branch com o branch principal do repositório. No GitHub, é conhecido como Pull Request.

Conclusão

Ao refletirmos sobre as técnicas de depuração e versionamento, fica claro que elas representam um enorme potencial para melhorar nosso trabalho como desenvolvedores de software embarcado.

Ao dominarmos essas tecnologias, podemos construir aplicações mais estáveis e eficientes. Espero que este material de apoio seja útil e enriquecedor para o seu aprendizado. Por favor, aprofunde seus estudos por meio das referências deste texto, que servem como material suplementar. Vamos explorar juntos o fascinante mundo do software embarcado!

Referências

[1] MICROSOFT. Debug C++ in Visual Studio Code. Visual Studio Code Documentation, 2023. Disponível em: <https://code.visualstudio.com/docs/cpp/cpp-debug>. Acesso em: 15 ago. 2024.

[2] MICROSOFT. Configure C++ Debugging with launch.json in Visual Studio Code. Visual Studio Code Documentation, 2023. Disponível em: <https://code.visualstudio.com/docs/cpp/launch-json-reference>. Acesso em: 15 ago. 2024.

CHACON, Scott; STRAUB, Ben. Pro Git. 2. ed. Apress, 2014. Disponível em: <https://git-scm.com/book/en/v2>. Acesso em: 15 ago. 2024.

[3] GITHUB. GitHub Cheat Sheet. Disponível em: https://training.github.com/downloads/pt_BR/github-git-cheat-sheet/. Acesso em: 15 ago. 2024.

[4] MOORE, Lindsay. Git and GitHub Tutorial for Beginners. HubSpot, 2023. Disponível em: <https://product.hubspot.com/blog/git-and-github-tutorial-for-beginners>. Acesso em: 15 ago. 2024.

[5] BERKSHIRE, Seth Kenlon. An Introduction to Important Git Concepts. Opensource.com, 2022. Disponível em: <https://opensource.com/article/22/11/git-concepts>. Acesso em: 15 ago. 2024.

Obrigado

