Exercícios Conceitos - CI/CD e Git

O que é CI/CD e qual é o seu objetivo?
CI/CD é a sigla para Integração Contínua (Continuous Integration) e Entrega Contínua ou Implantação Contínua (Continuous Delivery/Deployment).
CI (Integração Contínua):

- Prática em que os desenvolvedores integram frequentemente seu código ao repositório principal.
- Cada integração é verificada por builds automáticos e testes.
- Objetivo: Detectar erros rapidamente, garantir que o código esteja sempre funcional.

CD (Entrega Contínua ou Implantação Contínua):

- Entrega Contínua: Garante que o software esteja sempre em um estado pronto para ser implantado, mas a liberação para produção ainda pode depender de aprovação manual.
- Implantação Contínua: Vai além, automatizando o processo de colocar as mudanças em produção automaticamente, sem intervenção humana.

Objetivo do CI/CD:

- Automatizar e otimizar o processo de desenvolvimento, testes e entrega de software.
- Reduzir erros manuais.
- Aumentar a qualidade do código.
- Entregar atualizações mais rapidamente e de forma segura.
- Facilitar o feedback contínuo sobre o estado do sistema.

Quais são os benefícios do CI/CD?
Os principais benefícios do CI/CD são: Entregas mais rápidas, Redução de erros, Maior qualidade do software, Feedback contínuo, Menos retrabalho, Processo padronizado e automatizado e Facilidade na escalabilidade

Quais são as diferenças entre integração contínua (CI) e entrega contínua (CD)?
CI garante que o código novo funcione bem com o código existente.
CD garante que esse código testado possa ser entregue ao usuário facilmente.

Como o CI/CD pode melhorar a qualidade do software?
O CI/CD aumenta a confiança no código, reduz erros e melhora a estabilidade do software entregue ao usuário final.

Como o CI/CD pode melhorar a colaboração entre os desenvolvedores?
O CI/CD promove mais comunicação, transparência e agilidade, tornando o trabalho em equipe mais eficiente e colaborativo.

Como o controle de versão está relacionado ao CI/CD?
O controle de versão organiza o código e rastreia mudanças, enquanto o CI/CD automatiza testes e entrega com base nessas mudanças. Um depende do outro para garantir qualidade, velocidade e colaboração no desenvolvimento de software.

Quais ferramentas e tecnologias são comumente usadas em uma implementação de CI/CD?
CI: Jenkins, GitHub Actions, GitLab CI...

CD: Spinnaker, ArgoCD, Flux...

Testes: JUnit, Selenium, Postman...

Cloud e containers: Docker, Kubernetes, AWS/GCP/Azure...

O que é o Git e por que é usado em desenvolvimento de software?
O Git é um sistema de controle de versão distribuído. Ele permite que os desenvolvedores registrem, acompanhem e gerenciem mudanças no código-fonte ao longo do tempo.

Como o Git difere de outros sistemas de controle de versão?

O Git se diferencia de outros sistemas de controle de versão principalmente por ser distribuído, o que significa que cada desenvolvedor tem uma cópia completa do projeto e seu histórico, podendo trabalhar mesmo sem internet. Isso é diferente dos sistemas centralizados, onde só o servidor principal guarda o histórico e os desenvolvedores precisam estar conectados para fazer mudanças.

Além disso, o Git é muito rápido nas operações porque tudo é feito localmente. Ele também facilita o uso de branches (ramificações) de forma leve e eficiente, permitindo que desenvolvedores trabalhem em funcionalidades separadas sem complicações.

Outra vantagem importante é que o Git usa uma forma de garantir a integridade do código, impedindo que o histórico seja corrompido ou alterado sem registro. Ele é muito

flexível para diferentes estilos de trabalho e tem grande suporte em plataformas e ferramentas modernas.

O que é um repositório do Git?
Um repositório Git é onde o código-fonte do projeto é armazenado junto com todo o seu histórico de versões. É uma pasta especial que guarda não só os arquivos do projeto, mas também todas as mudanças feitas ao longo do tempo. Permite que você acompanhe, reverta e compare versões anteriores do código. Pode ser local (no seu computador) ou remoto (em serviços como GitHub, GitLab, Bitbucket).

Quais são os principais comandos do Git que são usados com mais frequência?

git init — cria um novo repositório Git local
git clone [url] — copia um repositório remoto para seu computador
git status — mostra o estado atual dos arquivos
git add [arquivo] — adiciona arquivos para o próximo commit
git commit -m "mensagem" — salva as mudanças com uma mensagem
git push — envia as mudanças para o repositório remoto
git pull — atualiza o repositório local com mudanças do remoto
git branch — lista ou cria ramificações (branches)
git checkout [branch/nome_arquivo] — muda de branch ou restaura arquivo
git merge [branch] — une mudanças de outra branch na atual

O que é um commit no Git?
Um commit no Git é como um "salvamento" do estado atual do seu projeto.

O que é um branch no Git e como ele é usado?
O branch permite que você trabalhe em paralelo, testando e desenvolvendo mudanças de forma isolada, garantindo um fluxo de trabalho seguro e colaborativo.
Você cria um branch para desenvolver uma nova funcionalidade ou corrigir um bug. Depois de finalizar e testar, você integra esse branch de volta ao branch principal usando um comando chamado `merge`.
Isso mantém o código organizado, facilita o trabalho em equipe e evita conflitos diretos no código principal.

O que é um merge no Git e como é realizado?
Um **merge** no Git é o processo de **unir as mudanças de um branch** com outro, geralmente para integrar o trabalho feito em uma ramificação separada de volta à branch principal.
Você primeiro **muda para a branch de destino** (ex: `main`):
git checkout main
Depois, executa o comando de merge para trazer as mudanças da outra branch:
git merge (nome da branch)

O que é um conflito de merge no Git e como ele é resolvido?

Um conflito de merge no Git acontece quando você tenta unir duas branches que fizeram mudanças diferentes nas mesmas linhas de um arquivo, e o Git não consegue decidir automaticamente qual versão deve prevalecer.

Como resolver:

1. Abra o arquivo com conflito.
2. Analise as diferenças entre as versões.
3. Escolha qual código manter (ou combine as partes manualmente).
4. Remova as marcações (<<<<<<<, =======, >>>>>>>).
5. Depois, adicione o arquivo corrigido ao stage:

   git add nome_do_arquivo

6. Finalize o merge com um commit:
   git commit

O que é o GitHub e como ele se relaciona com o Git?
O **GitHub** é uma plataforma online que hospeda repositórios Git na nuvem, facilitando o compartilhamento e a colaboração em projetos de software.

Como ele se relaciona com o Git?

- O Git é a ferramenta que gerencia o controle de versão localmente no seu computador.
- O GitHub funciona como um repositório remoto, onde você pode enviar (push) e baixar (pull) seus projetos usando Git.
- Além de armazenar o código, o GitHub oferece recursos extras, como:
    - Pull requests (para revisar e discutir mudanças antes de integrar)
    - Issues (para gerenciar bugs e tarefas)
    - Ações (GitHub Actions) para automação de CI/CD
    - Interface gráfica para explorar código, histórico e colaboração

Quais são algumas melhores práticas para trabalhar com o Git em equipes de desenvolvimento?

Use branches para cada tarefa ou funcionalidade
Crie uma branch separada para cada nova funcionalidade, correção de bug ou experimento. Isso evita misturar mudanças e facilita a revisão.

Faça commits pequenos e frequentes
Commits menores e com mensagens claras tornam o histórico mais fácil de entender e ajudam a identificar problemas rapidamente.

Escreva mensagens de commit claras e descritivas
Explique o que foi feito e por quê, para facilitar a comunicação com a equipe.

Atualize sua branch frequentemente com o branch principal
Use git pull ou git merge para manter seu branch sincronizado e evitar conflitos grandes no futuro.

Revise o código antes de integrar
Utilize pull requests para que outros membros da equipe revisem as mudanças antes de unir ao branch principal.

Resolva conflitos com cuidado
Quando houver conflitos, analise as mudanças com atenção para não perder código importante.

Use tags para marcar versões importantes
Isso ajuda a identificar releases estáveis ou pontos de referência no projeto.

Evite commitar arquivos desnecessários
Utilize um arquivo .gitignore para excluir arquivos temporários, binários ou configurações locais que não devem ir para o repositório.

Automatize processos com CI/CD
Integre testes e deploy automático para garantir que o código que entra na branch principal esteja sempre funcionando.