***Objetivo:*** Utilização de uma ferramenta de Inteligência Artificial Generativa como apoio a atividades de autoaprendizagem.

***Procedimentos:***

1. Aceda a ***duas*** ferramentas de IA, sendo que uma terá que ser a *Gemini* e outra será à sua escolha;
2. Copie e cole as questões que se seguem abaixo em cada uma dessas ferramentas;
3. Copie e cole, neste documento, as respetivas respostas indicadas pelas ferramentas;
4. Se o texto gerado pelas ferramentas for demasiado “extenso”, “aperte” com as ferramentas de forma a que as respostas sejam mais resumidas (*veja um exemplo no final deste documento*).
5. Guarde este documento em formato ***pdf*** e submeta-o na tarefa assíncrona na Moodle.

Bom trabalho!

Questões a colocar:

**1.** Qual a principal diferença entre uma gestão de versões *centralizada* e uma *descentralizada*, como o *Git*, e quais as vantagens e desvantagens de cada uma das abordagens?

**2.** Explica o conceito de "ramo" (*branch*) no *Git*e indica quais as principais utilizações dos ramos num fluxo de trabalho típico no desenvolvimento de *software*.

**3.** O que é uma "confirmação" (*commit*) no *Git* e qual a importância de escrever mensagens de confirmação claras e concisas?

**4.** Descreve o fluxo de trabalho típico de um pedido de incorporação no *GitHub* e indica quais são as etapas envolvidas e qual o seu papel na colaboração em projetos de software.

**5.** Quais as principais vantagens de utilizar o *GitHub* em comparação com outras plataformas de hospedagem de repositórios do tipo *Git*?

**6.** Como utilizarias o *Git* para fazer a gestão de uma grande aplicação (*software*) com vários programadores a trabalhar em diferentes funcionalidades dessa aplicação?

Exemplo, após resposta da *Gemini* à primeira pergunta.

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

A resposta mais simples foi a seguinte:

Uma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Ou seja, esta resposta é *muito* mais *simples* (e compreensível) do que a resposta original dada pela IA! 😊

**1.** Qual a principal diferença entre uma gestão de versões *centralizada* e uma *descentralizada*, como o *Git*, e quais as vantagens e desvantagens de cada uma das abordagens?

## Resposta (Gemini):

## Gestão de Versões: Centralizada vs. Descentralizada (Git)

**Centralizada:**

* **Um único repositório:** Todas as versões do código estão em um só lugar.
* **Vantagens:** Fácil de usar, controle centralizado.
* **Desvantagens:** Ponto único de falha, necessidade de conexão constante.

**Descentralizada (Git):**

* **Múltiplos repositórios:** Cada desenvolvedor tem uma cópia completa.
* **Vantagens:** Alta disponibilidade, trabalho offline, flexibilidade.
* **Desvantagens:** Curva de aprendizado mais íngreme, complexidade.

**Escolha:**

* **Centralizada:** Projetos pequenos, controle rígido, facilidade de uso.
* **Descentralizada (Git):** Projetos grandes, equipes distribuídas, alta disponibilidade.

**Em resumo:** Git é ideal para projetos modernos, oferecendo maior flexibilidade e segurança.

## Resposta (ChatGPT):

**Centralizado** (ex.: SVN):

* **Diferença**: Um repositório central acessado por todos.
* **Vantagens**: Simplicidade, controle centralizado.
* **Desvantagens**: Depende do servidor central, sem trabalho offline.

**Descentralizado** (ex.: Git):

* **Diferença**: Cada desenvolvedor tem uma cópia completa do repositório.
* **Vantagens**: Trabalho offline, backup local, melhor colaboração.
* **Desvantagens**: Curva de aprendizado maior, coordenação mais complexa.

**2.** Explica o conceito de "ramo" (*branch*) no *Git*e indica quais as principais utilizações dos ramos num fluxo de trabalho típico no desenvolvimento de *software*.

## Resposta (Gemini):

## Resumo: Ramos no Git

**O que é um ramo?**

* É como um galho em uma árvore: uma linha do tempo independente das alterações feitas no código principal.
* Permite trabalhar em diferentes funcionalidades ou versões sem afetar o código principal.

**Para que servem os ramos?**

* **Desenvolver novas funcionalidades:** Isolar mudanças e experimentar novas ideias sem risco.
* **Corrigir bugs:** Criar um ramo específico para cada bug e testar a correção isoladamente.
* **Gerenciar diferentes versões:** Manter versões anteriores do software sem afetar a versão atual.
* **Colaborar em equipe:** Cada membro pode trabalhar em um ramo diferente.

**Fluxo de trabalho típico:**

1. **Criar um novo ramo.**
2. **Desenvolver as alterações.**
3. **Testar as alterações.**
4. **Mesclar o ramo com o ramo principal.**

**Em resumo:**

Os ramos no Git são essenciais para organizar o trabalho, facilitar a colaboração e evitar conflitos no desenvolvimento de software.

## Resposta (ChatGPT):

No **Git**, um **ramo** (branch) é uma linha independente de desenvolvimento, que permite trabalhar em alterações sem afetar o código principal.

### Principais utilizações:

* **Desenvolvimento de novas funcionalidades**: Criar branches para adicionar funcionalidades sem interferir no código estável.
* **Correção de bugs**: Usar branches para isolar e corrigir problemas, especialmente em produção.
* **Branches principais**:
  + **Main/Master**: Código estável e pronto para produção.
  + **Develop**: Onde o desenvolvimento principal ocorre antes de ser integrado ao main.
* **Trabalho colaborativo**: Permite que múltiplos desenvolvedores trabalhem em diferentes partes do projeto simultaneamente.

**3.** O que é uma "confirmação" (*commit*) no *Git* e qual a importância de escrever mensagens de confirmação claras e concisas?

## Resposta (Gemini):

## O que é um commit e a importância das mensagens de commit no Git

**O que é um commit?**

* É como um "instantâneo" do seu código em um determinado momento.
* É uma espécie de marcação que registra as alterações feitas no seu projeto.
* Cada commit possui um código único e está ligado ao commit anterior, formando um histórico do projeto.

**Por que as mensagens de commit são importantes?**

* **Documentação:** As mensagens explicam o que foi alterado em cada commit.
* **Histórico claro:** Facilitam a compreensão da evolução do projeto.
* **Facilidade na busca:** Ajudam a encontrar rapidamente alterações específicas.
* **Colaboração:** Melhoram a comunicação entre os desenvolvedores.
* **Depuração:** Facilitam a identificação da origem de problemas.

**Dicas para escrever boas mensagens de commit:**

* **Seja conciso e claro:** Explique o objetivo da alteração de forma direta.
* **Use verbos no imperativo:** Por exemplo, "Adiciona nova funcionalidade".
* **Seja específico:** Evite mensagens vagas como "Corrige alguns bugs".
* **Use o presente do indicativo:** Por exemplo, "Atualiza a documentação".

**Exemplo:**

feat(login): Adiciona campo de recuperação de senha

**Em resumo:** As mensagens de commit são a documentação do seu código. Escrever mensagens claras e concisas é fundamental para manter um histórico organizado e facilitar a colaboração em projetos.

## Resposta (ChatGPT):

Uma **confirmação** (commit) no Git é um registro de alterações no repositório, salvando o estado atual do código. Cada commit cria um ponto no histórico do projeto.

### Importância de mensagens claras e concisas:

* **Facilitam a compreensão** do histórico de mudanças.
* **Ajudam na colaboração**, permitindo que outros desenvolvedores entendam o propósito de cada alteração.
* **Facilitam a depuração**, ajudando a identificar quando e por que uma mudança foi feita.

**4.** Descreve o fluxo de trabalho típico de um pedido de incorporação no *GitHub* e indica quais são as etapas envolvidas e qual o seu papel na colaboração em projetos de software.

## Resposta (Gemini):

## Fluxo de Trabalho Típico de um Pull Request no GitHub

Um **pull request** (PR) no GitHub é uma solicitação formal para que as alterações feitas em um ramo sejam incorporadas a outro ramo, geralmente o ramo principal. Ele serve como um ponto central para discussão, revisão e aprovação de código.

**Etapas envolvidas em um pull request:**

1. **Criação de um ramo:** O desenvolvedor cria um novo ramo a partir do ramo principal para trabalhar em uma nova funcionalidade ou correção de bug.
2. **Desenvolvimento e testes:** As alterações são feitas no novo ramo e testadas localmente.
3. **Criação do pull request:** O desenvolvedor cria um PR, especificando o ramo base e o ramo de destino.
4. **Revisão:** Outros membros da equipe revisam o código, sugerem alterações e fazem perguntas.
5. **Discussão:** O autor do PR e os revisores discutem as mudanças e respondem aos comentários.
6. **Aprovação:** Se o código estiver conforme os padrões da equipe, os revisores aprovam o PR.
7. **Mesclagem:** Após a aprovação, o PR é mesclado ao ramo de destino, incorporando as alterações ao projeto principal.

**Papel do pull request na colaboração:**

* **Visibilidade:** Torna as alterações visíveis para toda a equipe, facilitando a revisão e o feedback.
* **Discussão:** Permite que os desenvolvedores discutam as mudanças e cheguem a um consenso.
* **Qualidade:** Garante que o código seja revisado e aprovado antes de ser mesclado ao projeto principal, aumentando a qualidade do software.
* **Histórico:** Cria um histórico das alterações, facilitando o rastreamento de bugs e a compreensão da evolução do projeto.

**Em resumo:**

O pull request é uma ferramenta fundamental para a colaboração em projetos de software, pois promove a revisão do código, a comunicação entre os desenvolvedores e a garantia da qualidade do software.

## Resposta (ChatGPT):

O fluxo de trabalho de um **pedido de incorporação** (pull request) no GitHub envolve:

1. **Criar branch**: Desenvolvedor cria uma branch para novas funcionalidades ou correções.
2. **Fazer commits**: Realiza e salva as alterações no branch.
3. **Abrir pull request**: Solicita a incorporação das mudanças no branch principal (main/develop).
4. **Revisão de código**: Outros desenvolvedores revisam, sugerem melhorias ou aprovam.
5. **Merge**: Após aprovação, as mudanças são incorporadas ao branch principal.

Este processo facilita a colaboração, garante controle de qualidade e mantém o histórico organizado.

**5.** Quais as principais vantagens de utilizar o *GitHub* em comparação com outras plataformas de hospedagem de repositórios do tipo *Git*?

## Resposta (Gemini):

## Principais Vantagens do GitHub

O GitHub se destaca como a plataforma de hospedagem de repositórios Git mais popular por diversas razões:

* **Comunidade:** Possui a maior comunidade de desenvolvedores, o que facilita a colaboração, o aprendizado e a busca por soluções.
* **Recursos:** Oferece uma ampla gama de ferramentas integradas, como gerenciamento de issues, wikis, code review, integração contínua e deploy.
* **Interface amigável:** A interface do GitHub é intuitiva, facilitando o uso tanto para iniciantes quanto para desenvolvedores experientes.
* **Integrações:** Se integra facilmente com outras ferramentas e serviços populares, como ferramentas de CI/CD, plataformas de nuvem e editores de código.
* **Popularidade:** Sendo a plataforma mais utilizada, há uma grande quantidade de recursos, tutoriais e suporte disponíveis.
* **Versões públicas e privadas:** Permite criar tanto repositórios públicos para projetos open-source quanto privados para projetos comerciais.

**Em resumo:**

O GitHub oferece uma combinação única de comunidade, recursos e facilidade de uso, tornando-o a escolha preferencial para a maioria dos desenvolvedores.

## Resposta (ChatGPT):

As principais vantagens do **GitHub** são:

* **Popularidade**: Grande comunidade e integração com muitos projetos open source.
* **Integrações nativas**: Com GitHub Actions (CI/CD), GitHub Pages, e GitHub Packages.
* **Colaboração**: Ferramentas de revisão de código, pull requests e comentários.
* **Interface amigável**: UI intuitiva para gestão de repositórios e issues.
* **Ecosistema de extensões**: Suporte a ferramentas de terceiros e automações.

Esses fatores tornam o GitHub uma escolha popular para colaboração e desenvolvimento de software.

**6.** Como utilizarias o *Git* para fazer a gestão de uma grande aplicação (*software*) com vários programadores a trabalhar em diferentes funcionalidades dessa aplicação?

## Resposta (Gemini):

## Gestão de Grandes Aplicações com Git: Em Poucas Palavras

**O Git é a ferramenta ideal para gerenciar projetos de software com múltiplos desenvolvedores.**

**Como funciona:**

* **Ramos:** Cada desenvolvedor trabalha em um ramo isolado, evitando conflitos.
* **Pull requests:** Alterações são revisadas antes de serem incorporadas ao projeto principal.
* **Fluxo de trabalho:** Um processo definido guia o desenvolvimento (ex: Gitflow, GitHub Flow).
* **Histórico:** Todas as mudanças são registradas, permitindo rastrear e reverter alterações.

**Benefícios:**

* **Colaboração eficiente:** Desenvolvedores trabalham em paralelo.
* **Qualidade:** Revisão de código garante alta qualidade.
* **Flexibilidade:** Adaptação fácil a mudanças.
* **Histórico completo:** Facilita manutenção e resolução de problemas.

**Em resumo:**

O Git é a espinha dorsal de muitos projetos, proporcionando um ambiente de desenvolvimento colaborativo, organizado e eficiente.

## Resposta (ChatGPT):

Para gerir uma grande aplicação com vários programadores:

1. **Branching model**: Usaria o modelo **Git Flow**:
   * **Main**: Para código de produção.
   * **Develop**: Para integração de funcionalidades antes de ir para produção.
   * **Feature branches**: Para cada nova funcionalidade, isolando o trabalho de cada desenvolvedor.
   * **Release branches**: Para preparar versões estáveis.
   * **Hotfix branches**: Para correções rápidas no código de produção.
2. **Pull requests**: Cada funcionalidade ou correção passa por **revisão de código** antes de ser integrada.
3. **Commits frequentes**: Encorajaria commits pequenos e frequentes para facilitar o merge e rastreamento de problemas.
4. **CI/CD**: Integração contínua (CI) para testar alterações automaticamente antes de serem incorporadas.