📚 Apostila Didática: Git e GitHub para Trabalho em Equipe

Objetivo: Evoluir do uso individual do Git/GitHub para um fluxo de trabalho colaborativo, organizado e profissional, focado na qualidade do código e na integração segura.

Capítulo 1: Preparando o Ambiente Colaborativo

1.1. Convidando Colaboradores no GitHub

Passos (Interface GitHub):

- Acesse o repositório no GitHub.
- Vá em **Settings** (Configurações).
- Clique em **Collaborators and teams** (Colaboradores e Times).
- Clique em **Add people** (Adicionar pessoas), busque o *username* do colega e envie o convite.
- O colega deve aceitar o convite (via e-mail ou notificação).

1.2. Clonando e Sincronizando o Repositório

É essencial ter a cópia mais recente do projeto antes de começar a trabalhar.

- Clonar (Primeira vez):
  git clone
  https://docs.github.com/pt/repositories/creating-and-managing-repositories/creating-a-new-repository
- **Sincronizar (Sempre antes de começar):**
  1. Garantir que está na *branch* principal: git checkout main
  2. Puxar todas as alterações mais recentes: git pull origin main

Capítulo 2: Branches – Isolando Trabalhos

2.1. Por que usar Branches?

O **Branch** principal (**main** ou **master**) deve conter apenas código estável (pronto para produção). As *branches* de *feature* isolam seu trabalho para que você possa desenvolver ou corrigir sem comprometer a estabilidade do projeto principal.

Convenções de Nomenclatura:

- **main:** Produção/Código Estável.
- **feature/nome-da-tarefa:** Para novas funcionalidades (Ex: feature/login-de-usuario).
- **bugfix/nome-do-erro:** Para correção de bugs.

2.2. Fluxo Básico de Desenvolvimento em Branch

Este é o ciclo padrão para cada nova tarefa:

1. Atualizar a base:
   git checkout main
   git pull origin main
2. Criar e mudar para a nova branch:
   git checkout -b feature/minha-nova-feature
3. Trabalhar e Comitar:
   git add .
   git commit -m "feat: implementa validacao de formulario"
4. Publicar a branch no GitHub:
   git push -u origin feature/minha-nova-feature

2.3. Boas Práticas de Commits

- **Commits Atômicos:** Cada *commit* deve ser uma mudança lógica e pequena. Não comite "tudo de uma vez".

- **Mensagens Claras:** Use o estilo **imperativo** (comando) e, se possível, a convenção (feat, fix, refactor):
    - **Correto:** fix: corrige altura do header
    - **Incorreto:** Ajustei o header porque estava quebrado

Capítulo 3: Pull Requests (PR) — Revisão e Integração Segura

3.1. O que é uma Pull Request?

É o pedido formal para que o seu código (na *branch* de *feature*) seja **revisado, discutido e aceito** na *branch* principal (main).

3.2. Como Criar um PR (Interface GitHub)

Após o git push da sua *branch*:

1. Acesse o repositório no GitHub.
2. Clique em **Compare & pull request** ou vá na aba Pull Requests.
3. Defina o destino (base: main) e a origem (compare: sua-branch).
4. **Título:** Use o padrão (Ex: "feat: Implementa Login de Usuário").
5. **Descrição:** Explique o que foi feito, como testar, e se há alguma dependência.
    - **Dica:** Em projetos maiores, use **Templates de PR** para padronizar essa descrição com checklists.
6. Clique em **Create pull request**.

3.3. Revisão de Código (*Code Review*)

- O Revisor (colega ou professor) analisa o código no PR.
- Ele pode **aprovar** ou **solicitar mudanças** (deixando comentários construtivos).
- Se houver solicitações, o desenvolvedor faz as correções localmente, comita e dá um

novo git push. As alterações são atualizadas automaticamente no PR existente.

3.4. Fazendo o Merge (União)

Após a aprovação, o código é unido à *branch* main.

- **No GitHub**, o Revisor/Líder clica em **Merge pull request**.
- **Tipos de Merge:** A escolha depende da política do time:
  - **Merge Padrão:** Mantém o histórico completo (*commits* individuais da *feature* + *commit* de *merge*).
  - **Squash Merge:** Combina **todos** os *commits* da *feature* em **um único *commit*** antes de fazer o *merge* na main. Deixa o histórico mais limpo.
- **Limpeza:** Após o *Merge*, **sempre delete a *branch*** de *feature* no GitHub para manter o repositório organizado.

Capítulo 4: Lidando com Conflitos de Merge

4.1. Por que surgem Conflitos?

Conflitos ocorrem quando duas ou mais pessoas modificam a **mesma linha de código** em suas *branches* separadas. O Git não consegue decidir qual alteração manter.

4.2. Resolvendo Conflitos Localmente

Se o Git avisar sobre um conflito (após um git pull ou git merge), ele adiciona marcadores no arquivo afetado:

```
// arquivo.js
function calculaValor() {
<<<<<<< HEAD
  return 10 + 5; // Sua versão atual (HEAD)
=======
  return 10 * 2; // Versão do colega (branch sendo unida)
>>>>>>> feature/calculo-final
}
```

Passos para Solução:

1. **Edite o Arquivo:** Remova os marcadores (<<<<<<<, =======, >>>>>>>) e decida qual código deve permanecer (ou combine as duas alterações).
    - Exemplo de Solução:
      return 10 * 5; (Mantivemos a lógica de multiplicação, mas com um novo número).
2. Adicionar: Avise o Git que o conflito foi resolvido.
   git add arquivo.js
3. Comitar a Resolução:
   git commit -m "fix: resolve conflito de merge em calculaValor"
4. Push: Envie a resolução.
   git push

4.3. Recomendações para Evitar Conflitos

- **Sincronize com Frequência:** Faça git pull origin main no início do dia e antes de criar uma nova *branch*.
- **Branches Curta e Focadas:** *Branches* que vivem por muito tempo têm maior probabilidade de conflito.

Capítulo 5: Boas Práticas e Qualidade de Código

5.1. Qualidade e Automação (CI/CD Básico)

- **Integração Contínua (CI):** O GitHub Actions permite automatizar tarefas. Você pode configurar um **Workflow** que:
  - **Evento:** É disparado quando um Pull Request é aberto.
  - **Ação:** Roda testes automatizados ou checagens de estilo (linter).
- **Status Checks:** Configurar a main para **só permitir *merge*** se o **teste da CI passar** é a maneira mais segura de evitar *bugs* no código de produção.

5.2. Checklist Final do Fluxo de Trabalho (Guia Rápido)

1. [ ] **Atualizar[main]:** Estou na main e dei git pull origin main?
2. [ ] **Branch:** Criei uma *branch* nova e com nome claro (feature/...)?
3. [ ] **Commit:** Fiz *commits* pequenos e com mensagens claras?
4. [ ] **Atualizar[trabalho]:** Estou na branch de trabalho e dei git pull origin main?
5. [ ] **PR:** Criei o Pull Request no GitHub com título e descrição claros?
6. [ ] **Revisão:** Meu PR foi revisado e aprovado?
7. [ ] **CI:** Os testes automatizados (se houver) passaram?
8. [ ] **Merge:** O *merge* foi feito na main (Merge, Squash ou Rebase)?
9. [ ] **Limpar:** A *branch* de *feature* foi deletada após o *merge*?