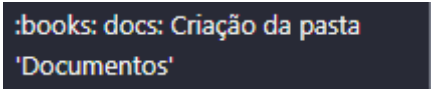
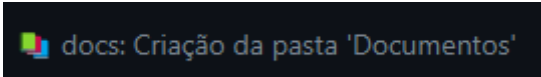


Este documento visa padronizar o Workflow e as diferentes ações referentes ao repositório do produto da Equipe 5 do Projeto Integrador no GitHub.

Commits

Os commits devem possuir:

- Cabeçalho: **Tag Emoji + Tag Tipo + Tag Task(Trello)- + Comentário**
 - Observação: caso não haja uma task válida relacionada ao *commit*, a “Tag Task” não deve ser incluída.
- Exemplo:
 - :books: docs: Task01-Implementação de classes modelo
 - 
 - 

Padrões de commits 📜

- De acordo com a documentação do [Conventional Commits](#), commits semânticos são uma convenção simples para ser utilizada nas mensagens de commit. Essa convenção define um conjunto de regras para criar um histórico de commit explícito, o que facilita a criação de ferramentas automatizadas.
- Esses commits auxiliarão você e sua equipe a entenderem de forma facilitada quais alterações foram realizadas no trecho de código que foi *commitado*.
- Essa identificação ocorre por meio de uma palavra e emoji que identifica se aquele commit realizado se trata de uma alteração de código, atualização de pacotes, documentação, alteração de visual, teste...

Tipo e descrição 🦄

O commit semântico possui os elementos estruturais abaixo (tipos), que informam a intenção do seu commit ao utilizador(a) de seu código.












- **feat**- Commits do tipo feat indicam que seu trecho de código está incluindo um novo recurso (se relaciona com o MINOR do versionamento semântico).
- **fix** - Commits do tipo fix indicam que seu trecho de código commitado está solucionando um problema (bug fix), (se relaciona com o PATCH do versionamento semântico).

- **docs** - Commits do tipo docs indicam que houveram mudanças na documentação, como por exemplo no Readme do seu repositório. (Não inclui alterações em código).
- **test** - Commits do tipo *test* são utilizados quando são realizadas alterações em testes, seja criando, alterando ou excluindo testes unitários. (Não inclui alterações em código)
- **build** - Commits do tipo build são utilizados quando são realizadas modificações em arquivos de build e dependências.
- **perf** - Commits do tipo perf servem para identificar quaisquer alterações de código que estejam relacionadas a performance.
- **style** - Commits do tipo style indicam que houveram alterações referentes a formatação de código, semicolons, trailing spaces, lint... (Não inclui alterações em código).
- **refactor** - Commits do tipo refactor referem-se a mudanças devido a refatorações que não alterem sua funcionalidade, como por exemplo, uma alteração no formato como é processada determinada parte da tela, mas que manteve a mesma funcionalidade, ou melhorias de performance devido a um code review.
- **chore** - Commits do tipo chore indicam atualizações de tarefas de build, configurações de administrador, pacotes... como por exemplo adicionar um pacote no gitignore. (Não inclui alterações em código)
- **ci** - Commits do tipo *ci* indicam mudanças relacionadas a integração contínua (continuous integration).

Recomendações 🎉

- Adicione um título consistente com o título do conteúdo.
- Recomendamos que na primeira linha deve ter no máximo 4 palavras.
- Para descrever com detalhes, usar a descrição do commit.
- Usar um emoji no início da mensagem de commit representando sobre o commit.
- Os links precisam ser adicionados em sua forma mais autêntica, ou seja: sem encurtadores de link e links afiliados.

Exemplos de commits:

Comando Git	Resultado no GitHub
<code>git commit -m ":tada: Commit inicial"</code>	 Commit inicial
<code>git commit -m ":books: docs: Atualização do README"</code>	 docs: Atualização do README
<code>git commit -m ":bug: fix: Loop infinito na linha 50"</code>	 fix: Loop infinito na linha 50
<code>git commit -m ":sparkles: feat: Pagina de login"</code>	 feat: Pagina de login
<code>git commit -m ":bricks: ci: Modificação no Dockerfile"</code>	 ci: Modificação no Dockerfile
<code>git commit -m ":recycle: refactor: Passando para arrow functions"</code>	 refactor: Passando para arrow functions
<code>git commit -m ":zap: perf: Melhoria no tempo de resposta"</code>	 perf: Melhoria no tempo de resposta
<code>git commit -m ":boom: fix: Revertendo mudanças ineficientes"</code>	 fix: Revertendo mudanças ineficientes
<code>git commit -m ":lipstick: feat: Estilização CSS do formulario"</code>	 feat: Estilização CSS do formulario
<code>git commit -m ":test_tube: test: Criando novo teste"</code>	 test: Criando novo teste
<code>git commit -m ":bulb: docs: Comentários sobre a função LoremIpsum()"</code>	 docs: Comentários sobre a função LoremIpsum()

Branches

- Novas *branches* devem ser criadas somente a partir da *branch* “**develop**”(salvo exceções).
- Uma nova *branch* deve ser nomeada no formato “**feature/0x-nova_branch**” em caso de nova **implementação** de **funcionalidade**.
 - Exemplo:
 - feature/01-implementacao_classes_modelo
- Uma nova *branch* deve ser nomeada no formato “**bugfix/0x-#tagIssue-correcao_novo_bug**” em caso de correção de **bug** e/ou resolução de **Issue**.
 - Exemplo:
 - bugfix/01-#01-correcao_bug_excessao_inventario
- Em geral, as *branches* devem ser criadas a partir da “**main**”, exceto casos específicos.
- Um “**pull request**” só pode ser realizado partindo a *branch* “develop” com destino a *branch* “main” (exceto casos específicos). Um *merge* entre as *branches* criadas e a *branch* “develop” são necessárias ao fim de implementações, antes do PR ocorrer.

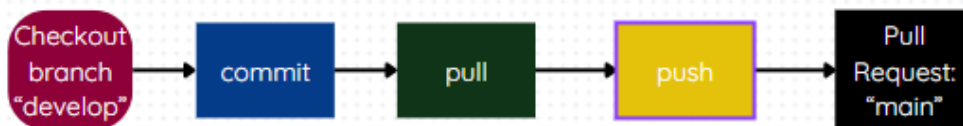
Workflow

- **ATENÇÃO!**
 - A *branch* “**main**” não deve, teoricamente, estar *commits* a frente da *branch* “**develop**”.
 - Novas *branches* devem ser criadas somente a partir da *branch* principal “**main**”(salvo exceções).
 - Após análises realizadas pelo Analista de Qualidade, *Issues* serão criadas para solução de problemas e/ou realização de respectivas correções.

Documentação

- Em caso de documentação, os *commits* devem ser feitos diretamente na *branch* “**develop**”, em suas respectivas pastas.
- Posteriormente, após realização dos *commits*, um “**pull request**” deve ser criado para *branch* “**main**”. O Analista de Qualidade e/ou outro membro responsável deve ser comunicado para verificação e análise do “**pull request**”.

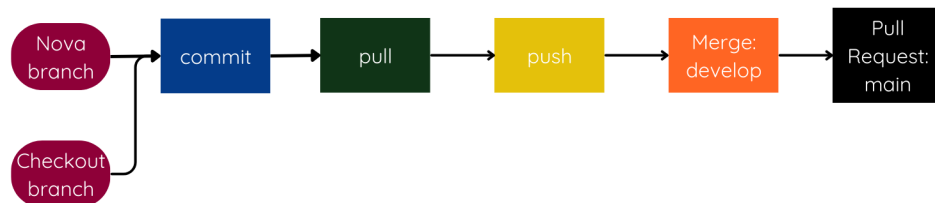
Workflow para Documentos/Artefatos



Implementações

- Cada nova funcionalidade ou implementação, que não está relacionada a uma *branch* já criada, deve ser feita em uma **nova branch**.
- A nova *branch* deve ser nomeada nos formatos definidos acima.
- Os *commits* devem ser realizados e padronizados conforme definido no título “*Commits*” acima.
- Sempre realizar um “**git pull**” **antes** de um “**git push**”.
- “*Pushes*” regulares devem ser realizados, para manter o repositório do github atualizado.
- Ao final das implementações da respectiva *branch*, um “**git merge**” com a *branch* develop deve ser realizado.
- Por fim, após testagens e demais verificações, deve ser criado um “**pull request**” deve ser criado para a *branch* “**main**”. Aceito posteriormente pelo “**Desenvolvedor Sênior**”.

Workflow para Implementações

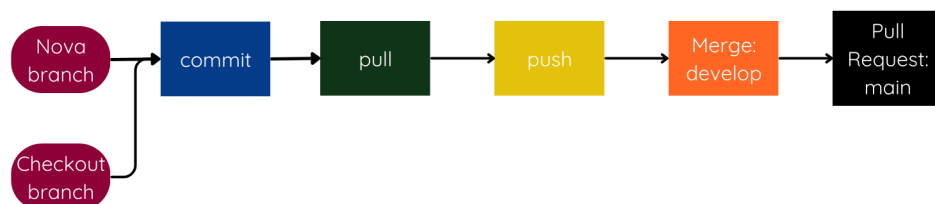


•

Resolução de Issues/Bugs

- Inicialmente uma **Issue** deve ser criada para documentar um problema ou relatar uma tarefa/atividade.
- Uma nova *branch* deve ser criada no formato padrão.
- Os *commits* devem ser realizados e padronizados conforme definido no título “Commits” acima.
- Sempre realizar um “**git pull**” antes de um “**git push**”.
- “*Pushes*” regulares devem ser realizados, para manter o repositório do github atualizado.
- Ao final das implementações da respectiva *branch*, um “**git merge**” com a *branch* develop deve ser realizado.
- Por fim, após testagens e demais verificações, deve ser criado um “**pull request**” para a *branch* “**main**”. Aceito posteriormente pelo “**Desenvolvedor Sênior**”.

Workflow para Implementações




•

Pull Requests


É uma maneira de contribuir com um projeto, sugerindo modificações, correções ou adições ao código existente. A principal ideia é permitir que outros membros da equipe revisem as alterações antes de serem incorporadas ao código principal (também conhecido como "branch principal" ou "branch principal").

Vídeo explicativo e introdutório:

“  Como criar seu primeiro pull request no GitHub #hacktoberfest ”.

Padrão

OBS: Um template foi criado e adicionado no repositório do projeto. Qualquer pull request que será criado pode ter ele como base. Todos devem conter o texto abaixo em seu corpo e o *checklist* deve ser realizado. Segue o formato:

```
### Sua checklist para esse pull request
🔥 Por favor, revise as diretrizes para contribuir para este
repositório...(Check with: )

- [ ] Certifique-se de estar solicitando o pull request da branch
**develop** (lado direito).
- [ ] Certifique-se que a branch de destino seja a **main** (lado
esquerdo).
- [ ] Verifique se os estilos de mensagem do commit ou mesmo de todos
os commits correspondem à nossa estrutura solicitada.
- [ ] Verifique se suas adições de código não falharão nas verificações
de linting de código nem no teste de unidade.
- [ ] Verifique se a aplicação pode ser atualizada.

### Sprint:

### Membros responsáveis pela verificação:
@gutox2001 e @Flankerstein

### Descrição:
Por favor descreva seu pull request.

### Observações:
```

Adicione suas observações.

Pre-merge checklist(Analista de Qualidade):

- [] Verifique se as Branches estão corretas.
- [] Verifique se não há erros ortográficos.
- [] Verifique a padronização de código e implementação.
- [] Verifique a padronização de commits.
- [] Verifique a relação com casos de uso e requisitos.

💖 Obrigado!

- **Todos os PR devem conter** o texto apresentado acima. Esse formato pode ser copiado e colado em caso de erro no template.
- Os campos “**Reviewers**”, “**Assignees**” e “**Labels**” devem ser preenchidos adequadamente! Utilize os padrões de *commits* para estes campos.

Issues

O que são "issues"? São pontos de **discussão** usados para rastrear **tarefas**, **melhorias**, **bugs** ou qualquer outra coisa relacionada a um projeto de software. Eles servem como um mecanismo central para a colaboração, **comunicação** e **gerenciamento** de trabalho em equipes que utilizam o GitHub para desenvolver software ou projetos relacionados.

Vídeo explicativo e introdutório: “  Introdução às issues no Github ”.

Padrão de Issue “Bug report”

title: “[Bug]: Descrição resumida”

labels: “bug”

assignees: marcar membro que se responsabilizará pela tarefa.

Write:

“TASK 0X (caso haja um task relacionada)

Cargo/ID do membro requisitado: Em geral o membro a ser comunicado será o Desenvolvedor Sênior: “Desenvolvedor Sênior/@Flankerstein”

Descrição detalhada do bug. Pode ser feita em tópicos ou em um parágrafo.

Máquina em que ocorreu o bug: ...

Condições para que o bug ocorra: ...

OBS: informações extras”

Padrão de Issue “Documentation”

title: “[Doc]: Descrição resumida”

labels: “documentation”

assignees: marcar membro que se responsabilizará pela tarefa.

Write:

“TASK 0X (caso haja um task relacionada-null em caso de não existir)

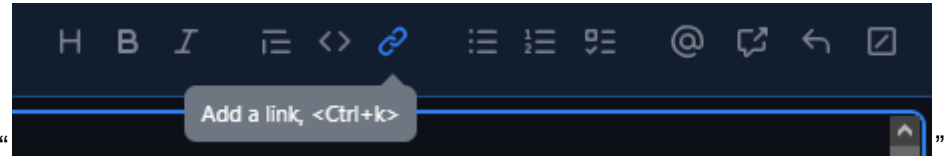
Cargo/ID do membro requisitado: Exemplo: “Analista de Qualidade/@gutox2001”

Descrição detalhada da tarefa a ser realizada ou da questão a ser resolvida.

Documentos relacionados: Nomes e/ou links para arquivos devem ser atribuídos.

OBS: Ao incluir um link na descrição da Issue siga os seguintes passos:

1- Clique na opção



2- Preencha o campo "(url)" com o link adequado o o campo "[]" com o nome que quer atribuir ao link. Exemplo:

[(url)]

[DCU_Arasy](https://github.com/ProjetoIntegradorUFV2023/Equipe5/blob/main/Documentos/Design%20de%20Projeto/Casos%20de%20Uso/DCU_Arasy.asta)

Exemplo de Issue

A screenshot of a GitHub issue page. The title is "[Doc]: Análise do DCU corrigido com base no Doc. de Análise de Artefato #2". The issue is open, created by gutox2001 5 minutes ago, with 0 comments. The main content is a comment from gutox2001 stating: "TASK null. Cargo/ID do membro requisitado: Designer de Projeto/@Flankerstein. Solicito que o responsável verifique as observações e erros a serem tratados informados no 'Documento de Análise de Artefato-DCU_Arasy', com base no Diagrama atualizado do commit 'commit_DCU'." Below the comment, it says "Documentos relacionados: DCU_Arasy, Documento de Análise de Artefato-DCU_Arasy". On the right side, there are settings for Assignees (Flankerstein), Labels (documentation), Projects (None yet), Milestone (No milestone), and Development (Create a branch for this issue or link a pull request.). At the bottom, there are notifications: "gutox2001 added the documentation label 5 minutes ago" and "gutox2001 assigned Flankerstein 5 minutes ago".

Referências

[1] ChatGPT. Pode ser acessado em: <<https://chat.openai.com>>. Último acesso em: 20/10/2023.

[2] Padrão de *commits* - Repositório GitHub. Pode ser acessado em: <<https://github.com/iuricode/padroes-de-commits>>. Último acesso em: 20/10/2023.