

GitHub - Boas Práticas

:github:

Nomeando suas branches

Quando estiver trabalhando em uma nova funcionalidade, o desenvolvedor deve originar seu branch à partir de um ambiente **estável**, geralmente o master, com um nome que realmente descreve sobre o que é aquela funcionalidade está sendo desenvolvida.

Algumas regras a serem seguidas ao originar seu novo branch são:

- O nome deve ser simples conciso, que represente sua tarefa;
- Um único idioma deve ser utilizado (inglês);
- Não utilize códigos que não tem sentido quando sem contexto;
- Começar com letras minúsculas;
- Palavras separadas por hífen -.

Além disso, algumas *tokens* podem ser definidas em seu projeto que vão sempre preceder os nomes de seus ramos:





















Nome do Branch	Descrição
<iniciais dos nomes>/feat-<descricao task > <i>exemplo:</i> mra/feat-env-docker	Quando desenvolvendo uma nova funcionalidade ou melhoria
<iniciais dos nomes>/bug-<descricao task >	Quando há um bug que necessita de correção e mesclagem
<iniciais dos nomes>/hotfix-<descricao task >	Quando há a necessidade de uma ação imediata em um comportamento indesejado no sistema







Padronizando os commits

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

Tipo	Descrição
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).
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 .
refactor	Commits do tipo refactor referem-se a mudanças devido a refatorações que não alterem sua funcionalidade .
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 (<i>continuous integration</i>).
raw	Commits do tipo raw indicam mudanças relacionadas a arquivos de configurações, dados, features, parametros.

Tipo do commit	Emoji	Palavra-chave

Adicionando um teste	 :white_check_mark:	test
Adicionando uma dependência	 :heavy_plus_sign:	build
Alterações de revisão de código	 :ok_hand:	style
Bugfix	 :bug:	fix
Comentários	 :bulb:	docs
Commit inicial	 :tada:	init
Configuração	 :wrench:	chore
Deploy	 :rocket:	
Documentação	 :books:	docs
Em progresso	 :construction:	
Estilização de interface	 :lipstick:	feat
Infraestrutura	 :bricks:	ci
Lista de ideias (tasks)	 :soon:	
Mover/Renomear	 :truck:	chore
Novo recurso	 :sparkles:	feat
Package.json em JS	 :package:	build
Performance	 :zap:	perf
Refatoração	 :recycle:	refactor
Removendo um arquivo	 :fire:	
Removendo uma dependência	 :heavy_minus_sign:	build

Revertendo mudanças	 :boom:	fix
Segurança	 :lock:	
Tag de versão	 :bookmark:	
Teste de aprovação	 :heavy_check_mark:	test
Testes	 :test_tube:	test
Tratamento de erros	 :goal_net:	

Exemplos de commit

Comando Git	Resultado no GitHub
git commit -m ":tada: Commit inicial"	 Commit inicial
git commit -m ":books: docs: Atualização do README"	 docs: Atualização do README
git commit -m ":bug: fix: Loop infinito na linha 50"	 fix: Loop infinito na linha 50
git commit -m ":sparkles: feat: Página de login"	 feat: Página de login
git commit -m ":bricks: ci: Modificação no Dockerfile"	 ci: Modificação no Dockerfile
git commit -m ":recycle: refactor: Passando para arrow functions"	 refactor: Passando para arrow functions
git commit -m ":zap: perf: Melhoria no tempo de resposta"	 perf: Melhoria no tempo de resposta
git commit -m ":boom: fix: Revertendo mudanças ineficientes"	 fix: Revertendo mudanças ineficientes

Pull Request

É a documentação do merge que será feito entre uma branch e outra. Nenhum merge pode ser realizado sem a abertura de um pull request, que deve ser aberto pelo GitHub.

- **Tipos de Pull Request:**

- Os pull requests podem ser separados por tipos, sendo eles, para merge ou para deploy.
- merge : quando será feito o pull request de uma branch não principal para outra do mesmo tipo, por exemplo:
- mra/feat-env-docker → develop (um pull request sendo feito de uma branch secundária para a branch de desenvolvimento).
- deploy: quando será feito o pull request de uma branch não principal ou da develop para a branch principal (main / master).



Algumas regras a serem seguidas ao criar seu pull request merge :




- Se o pull request só tiver um commit, o título deve ser o do commit.
- Se o pull request tiver mais de um commit, o título deve ser <branch_origem> → <branch_destino>.
- Pull requests com mais de um commit devem ter uma descrição com as alterações realizadas.
- Existe um padrão de título vide tabela abaixo.

-

- **Algumas regras a serem seguidas ao criar seu pull request deploy:**

- Todo pull request de deploy deve conter descrição detalhada seguindo o padrão (colocar link).
- Existe um padrão de título vide tabela abaixo.
- O pull request deploy deve ser avaliado por toda a equipe ativa do projeto.

Tipo Pull Request	Título Pull Request	Quantidade de commits	Tem descrição
merge	 [MERGE] <nome_branch/commit/task> exemplo:  [MERGE] feat: Adicionando Peso Peça REAL	1 commit	não

merge	 [MERGE] <nome_branch_origem> → <nome_branch_destino> exemplo:  [MERGE] kiataki-develop -> develop	mais de 1 commit	sim
deploy	 [DEPLOY] - Subindo Novas Alterações - <dd/mm/YYYY> exemplo:  [DEPLOY] - Subindo Novas Alterações - 15/03/2024	mais de 1 commit	sim

Observação:


Para fazer o merge, é necessário inverter as informações do campo de título com o campo de descrição.

Descrição do Pull Request para Deploy:

i Este pull request tem como objetivo integrar as mais recentes atualizações do ramo principal (main) para o ramo de desenvolvimento (develop).

Alterações Incluídas:

Integrações e Fusões:

 As últimas atualizações do desenvolvimento foram integradas ao ramo rcosta100-develop:

- Fusão das alterações do ramo rcosta100-develop para develop.

Refatorações:

 Passando os limites de engenharia para a análise e correlação:

- Refatoração para ajustar os limites de engenharia para a análise e correlação.
- Alteração de "0" para "null" nas requisições padrões.

Novas Funcionalidades:

 Limitando o range máximo de unidade de medida de datas:

- Adição de uma nova funcionalidade para limitar o range máximo de unidade de medida de datas.

Observações:

- Estas alterações foram revisadas e consideradas prontas para integração com o ramo de desenvolvimento.

- Foram realizadas refatorações importantes e adicionada uma nova funcionalidade para melhorar a aplicação.
- Este pull request está pronto para revisão e posterior implantação no ambiente de produção.