



Guia de Commits - Sistema Acadêmico PIM II



Padrão de Commits (Conventional Commits)

Formato Básico

<tipo>(<escopo>): <descrição curta>

[corpo opcional - explicação detalhada]

[rodapé opcional - referências, breaking changes]



Tipos de Commit



feat - Nova Funcionalidade

Use quando adicionar uma **nova funcionalidade** ao sistema.

bash

feat(aluno): adicionar função cadastrarAluno

feat(turma): implementar CRUD completo de turmas

feat(aula): criar diário eletrônico

feat(relatorio): adicionar geração de relatório em TXT

feat(ia): implementar sistema de recomendação de atividades

feat(rede): adicionar servidor TCP para múltiplos clientes



fix - Correção de Bug

Use para **corrigir erros** no código.

bash

fix(aluno): corrigir segmentation fault em listarAlunos

fix(turma): resolver problema de IDs duplicados

fix(aula): corrigir validação de data DD/MM/AAAA

fix(csv): corrigir leitura de arquivos com acentos

fix(memoria): resolver vazamento de memória em buscarAlunoPorRA



docs - Documentação

Use para mudanças **apenas em documentação**.

bash

docs: adicionar README com instruções de compilação
docs: atualizar padrões de codificação
docs(api): documentar funções públicas do file_manager
docs: criar guia de instalação e execução
docs: adicionar comentários nos módulos C



style - Formatação

Use para mudanças de **formatação** (espaços, indentação, ponto-e-vírgula).

bash

style: padronizar indentação para 4 espaços
style(aluno): remover espaços em branco no final das linhas
style: aplicar convenção de nomenclatura em todas as variáveis
style: formatar código segundo padrão estabelecido



refactor - Refatoração

Use para **reestruturar código** sem mudar funcionalidade.

bash

refactor(file_manager): simplificar lógica de salvarDados
refactor(aluno): extrair função validarRA
refactor: separar funções privadas das públicas
refactor(turma): melhorar performance de buscarTurmaPorID



test - Testes

Use para adicionar ou modificar **testes**.

bash

test: adicionar testes unitários para módulo de alunos
test(turma): criar suite de testes para CRUD
test: implementar main_test.c completo
test(integracao): adicionar teste de acesso simultâneo



chore - Manutenção

Use para **tarefas de manutenção** (build, configs, dependências).

bash

chore: adicionar Makefile para automatizar compilação
chore: atualizar .gitignore para ignorar arquivos .o
chore: configurar estrutura de pastas do projeto
chore: criar script de setup inicial

perf - Performance

Use para melhorias de **performance**.

bash

perf(aluno): otimizar busca linear para busca binária
perf(csv): melhorar velocidade de leitura de arquivos grandes
perf: reduzir uso de memória em arrays globais

security - Segurança

Use para correções de **segurança**.

bash

security: adicionar validação de entrada em cadastrarAluno
security(csv): prevenir buffer overflow na leitura de strings
security: implementar sanitização de dados de entrada

build - Sistema de Build

Use para mudanças no **sistema de build**.

bash

build: adicionar flags de otimização ao Makefile
build: configurar compilação para debug
build: atualizar targets do Makefile

Escopos Sugeridos para o Projeto


Escopo	Descrição	Exemplo
aluno	Módulo de gerenciamento de alunos	feat(aluno): adicionar CRUD
turma	Módulo de gerenciamento de turmas	fix(turma): corrigir busca
aula	Módulo de diário eletrônico	feat(aula): adicionar validação de data
rede	Arquitetura cliente-servidor	feat(rede): implementar servidor TCP
ia	Módulo de inteligência artificial	feat(ia): adicionar recomendação


Escopo	Descrição	Exemplo
csv	Persistência em arquivos CSV	fix(csv): corrigir encoding UTF-8
ui	Interface do usuário (Python)	feat(ui): criar tela de login
docs	Documentação	docs: atualizar README
test	Testes	test(aluno): adicionar casos de teste

Regras de Escrita

Boas Práticas

1. Use o **imperativo** ("adicionar" não "adicionado")
2. **Primeira letra minúscula** após o tipo
3. **Sem ponto final** na descrição curta
4. **Máximo 50 caracteres** na primeira linha
5. **Corpo opcional** com até 72 caracteres por linha
6. **Seja específico** mas conciso

```
bash
#  Correto
feat(aluno): adicionar função cadastrarAluno

#  Incorreto
Feat(aluno): Adicionada função cadastrarAluno.
```

Exemplos Práticos do Projeto

Desenvolvimento dos Módulos C

```
bash
```

Estruturas

feat(structs): criar estruturas Aluno, Turma e Aula

File Manager

feat(file): implementar salvarDados e carregarDados

feat(file): adicionar suporte para múltiplos tipos de dados

fix(file): corrigir problema ao abrir arquivo inexistente

Aluno Manager

feat(aluno): implementar CRUD completo de alunos

feat(aluno): adicionar função buscarAlunoPorRA

fix(aluno): corrigir validação de RA duplicado

refactor(aluno): extrair função de validação

Turma Manager

feat(turma): implementar CRUD de turmas

feat(turma): adicionar associação aluno-turma

feat(turma): implementar geração de próximo ID

fix(turma): corrigir segmentation fault em listarTurmas

Aula Manager

feat(aula): implementar registro de aulas

feat(aula): adicionar validação de data DD/MM/AAAA

feat(aula): criar função de geração de relatórios

fix(aula): corrigir formato de data no CSV

Arquitetura e Infraestrutura

bash

Compilação

chore: adicionar Makefile com targets all, clean, run

build: configurar flags de compilação do GCC

chore: criar estrutura de diretórios do projeto

Versionamento

chore: adicionar .gitignore para arquivos C

chore: criar README com instruções de setup

Testes

test: criar main_test.c com menu interativo

test(aluno): adicionar testes de CRUD

test(turma): adicionar testes de associação

fix(test): corrigir segmentation fault nos testes

Documentação

bash

docs: adicionar guia de padronização de código

docs: criar documentação de funções no cabeçalho

docs: atualizar README com exemplos de uso

docs(abnt): adicionar referências bibliográficas

docs: criar manual de usuário **do** sistema

Rede e Python (futuro)

bash

feat(rede): implementar servidor TCP com sockets

feat(rede): adicionar suporte para múltiplos clientes

feat(cliente): criar módulo de comunicação via socket

feat(ui): implementar tela de login com Tkinter

feat(ui): adicionar tela de cadastro de alunos

feat(ia): implementar análise de dados com Pandas

Exemplos Reais para Você Usar AGORA

Para o Trabalho Atual

bash

Commit inicial

chore: configurar estrutura inicial do projeto

Após criar structs.h

feat(structs): definir estruturas Aluno, Turma e Aula

Após criar file_manager

feat(file): implementar persistência de dados em CSV

Após criar aluno_manager

feat(aluno): implementar CRUD completo de alunos

Após criar turma_manager

feat(turma): implementar CRUD de turmas e associações

Após criar aula_manager

feat(aula): implementar diário eletrônico

Após criar main_test.c

test: adicionar programa de testes interativo

Após adicionar Makefile

build: adicionar Makefile para automação de compilação

Se corrigiu o segmentation fault

fix(turma): corrigir segmentation fault ao atualizar turma

Após criar documentação

docs: adicionar guia de padronização de código

Commit com Múltiplas Linhas

Use quando a mudança for complexa:

bash

git commit -m "feat(rede): implementar servidor TCP multi-cliente

- Adicionar listener de conexões TCP na porta 5000
- Implementar threading para múltiplos clientes
- Criar protocolo de comunicação baseado em JSON
- Adicionar handler para requisições GET_ALUNOS

Closes #12"

Ou use o editor:

```
bash
```

```
git commit
```

```
# Abre o editor de texto para escrever mensagem completa
```

O Que NÃO Fazer

```
bash
```

```
#  Muito genérico
```

```
git commit -m "alterações"
```

```
git commit -m "fix"
```

```
git commit -m "update"
```

```
#  Sem tipo
```

```
git commit -m "corrigir bug"
```

```
git commit -m "adicionar função"
```

```
#  Muito longo na primeira linha
```

```
git commit -m "feat: adicionar função de cadastrar aluno com validação completa de RA e email"
```

```
#  Misturar tipos
```

```
git commit -m "feat: adicionar CRUD e corrigir bugs e atualizar docs"
```

```
#  Commit gigante
```

```
# (Evite commits com 50+ arquivos alterados)
```

O Que Fazer

```
bash
```


 *Específico e claro*

```
git commit -m "feat(aluno): adicionar função cadastrarAluno"
```

 *Tipo correto*

```
git commit -m "fix(csv): corrigir encoding UTF-8"
```

 *Escopo apropriado*

```
git commit -m "docs: atualizar README com instruções"
```

 *Commits pequenos e focados*

(Um commit = uma mudança lógica)

Fluxo de Trabalho Recomendado

1. Antes de Começar a Trabalhar

bash

Atualizar branch

```
git pull origin main
```

2. Durante o Desenvolvimento

bash

Fazer mudanças pequenas e commits frequentes

Trabalhou 30min? Faça um commit!

```
git add c_modules/aluno_manager.c
```

```
git commit -m "feat(aluno): adicionar função cadastrarAluno"
```

```
git add c_modules/aluno_manager.c
```

```
git commit -m "feat(aluno): adicionar validação de RA"
```

```
git add c_modules/aluno_manager.c c_modules/aluno_manager.h
```

```
git commit -m "feat(aluno): adicionar função buscarAlunoPorRA"
```

3. Verificar Status

bash

Ver o que foi modificado

`git status`

Ver diferenças

`git diff`

Ver histórico

`git log --oneline`

4. Enviar para o GitHub

bash

Enviar commits

`git push origin main`



Checklist Antes de Commitar

- ☐ O código compila sem erros?
- ☐ Testei a mudança?
- ☐ O commit tem apenas uma mudança lógica?
- ☐ A mensagem descreve claramente o que foi feito?
- ☐ Usei o tipo correto (feat/fix/docs/etc)?
- ☐ A mensagem está no imperativo?
- ☐ Não há arquivos desnecessários (.o, executáveis)?



Comandos Git Úteis

Ver Histórico Bonito

bash

`git log --oneline --graph --decorate --all`

Desfazer Último Commit (mantém mudanças)

bash

`git reset --soft HEAD~1`

Desfazer Mudanças Não Commitadas

```
bash
git checkout -- arquivo.c
```

Ver Diferenças de um Arquivo

```
bash
git diff c_modules/aluno_manager.c
```

Exemplo de Histórico Ideal

- feat(ia): adicionar sistema de recomendação
- test(ia): adicionar testes do módulo de IA
- docs: atualizar capítulo 10 com evidências de IA
- feat(relatorio): implementar geração de PDF
- feat(aula): adicionar contagem de aulas por turma
- fix(turma): corrigir busca por ID
- refactor(file): simplificar função salvarDados
- feat(turma): implementar associação aluno-turma
- feat(aula): criar módulo de diário eletrônico
- feat(turma): implementar CRUD completo
- feat(aluno): implementar CRUD completo
- feat(file): adicionar persistência em CSV
- feat(structs): definir estruturas do sistema
- chore: configurar estrutura inicial do projeto

Resumo Rápido

Situação	Comando
Nova funcionalidade	feat(escopo): descrição
Correção de bug	fix(escopo): descrição
Documentação	docs: descrição
Formatação	style: descrição
Refatoração	refactor(escopo): descrição
Testes	test(escopo): descrição
Build/Config	chore: descrição

Dica Final

Commits pequenos e frequentes são melhores que commits gigantes!

É melhor ter 10 commits pequenos e claros do que 1 commit enorme com mensagem genérica.

Este padrão vai impressionar qualquer avaliador e mostrar profissionalismo! 🚀