

Migration Guide

Migração de Regras Monolíticas para Cursor Rules v2

Este guia ajuda você a migrar de um arquivo grande de regras (como o “User Rules 3.0”) para o sistema modular do Cursor Rules v2.

Visão Geral da Migração

Antes (Monolítico)

```
user-rules-3.0.md (3000+ linhas)
├── Contexto e identidade
├── Instruções fundamentais
├── Controles operacionais
├── Princípios de desenvolvimento
├── Padrões de nomenclatura
├── Sistema de Gates
├── Protocolos de segurança
├── Tratamento de erros
├── Diretrizes por tecnologia
├── Segurança e compliance
├── Docker e containerização
└── CI/CD e deployment
```

Depois (Modular)

```
User Rules (texto puro, ~200 linhas)
├── Estilo e preferências básicas
├── Controles fundamentais
└── Guard-rails de segurança

.cursor/rules/ (múltiplos arquivos .mdc)
├── 00-core-guardrails.mdc
├── 10-gates-system.mdc
├── 20-io-contracts.mdc
├── 30-testing.mdc
├── 40-security.mdc
├── 50-frontend-standards.mdc
├── 51-python-fastapi.mdc
├── 60-docker-compose.mdc
└── 90-troubleshooting.mdc
```

Processo de Migração Passo a Passo

Passo 1: Análise do Conteúdo Existente

Categorize suas regras atuais:

Para User Rules (Global):

- ☒ Estilo de comunicação
- ☒ Controles fundamentais (step-by-step, patch-only)

- ✓ Protocolos básicos (ASSUMPTION_REQUEST, RISK_ALERT)
- ✓ Princípios gerais (DRY, KISS, SOLID)

Para Project Rules (Específico):

- ✓ Padrões de linguagem/framework
- ✓ Configurações de ferramentas
- ✓ Processos complexos (Gates)
- ✓ Checklists detalhados
- ✓ Exemplos de código

Identifique o que remover:

- ✗ Conteúdo duplicado
- ✗ Regras muito específicas para User Rules
- ✗ Exemplos extensos
- ✗ Documentação que pode ser referenciada

Passo 2: Criar User Rules Essenciais

Extraia apenas o essencial para User Rules:

Você é um agente sênior. **Prioridades:** DRY ☐ KISS ☐ YAGNI ☐ SOLID ☐ DDD/Clean.

Execução:

- Faça apenas o próximo passo e espere "Go/No-Go".
- Patch-**only** ☐ máx. 5 arquivos por ciclo, 200 linhas/arquivo, 500 linhas **no** total.
- Nunca **presuma**: se faltar dado, dispare ASSUMPTION_REQUEST com opções e recomendação.
- **RISCO**: se envolver secrets, migrações, deleção em massa, auth, config de produção ☐ pare e peça validação (RISK_ALERT).
- Qualquer trabalho fora do escopo atual ☐ SCOPE_CHANGE com impacto e alternativas.

Qualidade:

- Sem artefatos/dummies. Tudo precisa ter uso **real**.
- Cada mudança deve declarar contrato I/O (REQUEST/RESPONSE) + critérios de aceite resumidos.
- Sem testes, sem **merge**. Cobertura alvo ☐ 80% **no** módulo tocado.

Operação:

- Consulte MCP/Context e docs relevantes antes de decisões.
- Optimize **tokens**: respostas objetivas, diffs unificados, sem contexto inútil.

Passo 3: Mapear Conteúdo para Project Rules

Mapeamento Direto

Seção Original	Arquivo Project Rule	Tipo de Ativação
Instruções fundamentais [A.1-A.3, A.6-A.9]	00-core-guardrails.mdc	Always Apply
Sistema de Gates G1-G5	10-gates-system.mdc	Agent Requested
Contratos I/O	20-io-contracts.mdc	Manual
Testing/coverage/performance	30-testing.mdc	Auto Attached
Security (OWASP, headers, secrets)	40-security.mdc	Always Apply
Frontend Next/React padrões	50-frontend-standards.mdc	Auto Attached
Python/FastAPI padrões	51-python-fastapi.mdc	Auto Attached
Docker/Compose	60-docker-compose.mdc	Auto Attached
CI/CD	70-cicd.mdc	Auto Attached
Observabilidade	80-observability.mdc	Agent Requested
Troubleshooting/escalação	90-troubleshooting.mdc	Manual

Passo 4: Criar Arquivos Project Rules

Template para cada arquivo:

```
---
description: [Descrição clara e concisa]
globs:
  - "[padrão de arquivo 1]"
  - "[padrão de arquivo 2]"
alwaysApply: [true/false]
---

# [Título da Regra]

## [Categoria 1]
- Regra específica 1
- Regra específica 2

## [Categoria 2]
- Regra específica 1
- Regra específica 2

## Anti-Patterns (Forbidden)
- ❌ Anti-pattern: Por que é ruim e alternativa

## Examples
```language
// Exemplo de código bom
```

## References

- @ref:other-rule#section
- @docs:<https://official-docs.com>

### Passo 5: Configurar Globs Apropriados

#### Por Tecnologia:

```
Frontend (TypeScript/React):
```yaml
globs:
  - "**/*.tsx"
  - "**/*.ts"
  - "**/components/**"
  - "**/app/**"
  - "**/pages/**"
```

Backend (Python/FastAPI):

```
globs:
  - "**/*.py"
  - "**/api/**"
  - "**/models/**"
  - "**/requirements*.txt"
  - "**/pyproject.toml"
```

DevOps (Docker/K8s):

```

globs:
- "Dockerfile*"
- "**/docker/**"
- "**/docker-compose*.yaml"
- "**/k8s/**"
- "**/*.yaml"

```

Testing:

```


globs:
- "**/tests/**"
- "**/*.test.*"
- "**/*.spec.*"
- "**/cypress/**"
- "**/playwright/**"

```

Exemplo Prático de Migração

Antes: Seção Monolítica

```

##  Diretrizes Específicas por Tecnologia

### Python/FastAPI

```json
{
 "python_standards": {
 "project_structure": {
 "src": {
 "domain/": "entidades, value objects, eventos",
 "application/": "use cases, DTOs, interfaces",
 "infrastructure/": "repositories, external services",
 "presentation/": "controllers, serializers"
 },
 // ... 500+ linhas de configuração detalhada
 },
 }
}

```

```

Depois: Project Rule Focada
```markdown
---
description: Python and FastAPI standards following Clean Architecture and DDD
principles
globs:
- "**/*.py"
- "**/requirements*.txt"
- "**/pyproject.toml"
- "**/Pipfile"
alwaysApply: false
---

# Python & FastAPI Standards

## Project Structure (Clean Architecture)

```




```
src/
├─ domain/ # Entities, Value Objects, Domain Events
├─ application/ # Use Cases, DTOs, Interfaces
├─ infrastructure/ # Repositories, External Services
└─ presentation/ # Controllers, Serializers
```

```
## FastAPI Patterns

### Dependency Injection
`python
from fastapi import Depends

def get_db() -> Session:
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()
```

Anti-Patterns (Forbidden)

-  Bare except clauses
-  Mutable default arguments
-  Global variables

@ref:testing#unit-tests

@ref:security#input-validation

Validação da Migração

Checklist de Validação

User Rules

- [] Menos de 1000 caracteres
- [] Apenas preferências globais
- [] Sem detalhes técnicos específicos
- [] Controles fundamentais incluídos

Project Rules

- [] Cada arquivo < 500 linhas
- [] Front-matter correto
- [] Globs apropriados para ativação
- [] Referências cruzadas funcionando
- [] Exemplos práticos incluídos

Funcionalidade

- [] Regras ativam no contexto correto
- [] Sem conflitos entre regras
- [] Performance adequada
- [] Comportamento consistente

Teste da Migração

1. **Backup**: Salve suas regras originais
2. **Implementação gradual**: Migre uma seção por vez
3. **Teste isolado**: Teste cada regra individualmente
4. **Teste integrado**: Verifique funcionamento conjunto
5. **Ajuste fino**: Refine baseado no comportamento observado

Problemas Comuns e Soluções

Token Bloat

Problema: Muitas regras **always** apply causando uso excessivo de tokens

Solução:

- Mova regras para auto attached
- Reduza conteúdo das regras **always** apply
- Use referências ao invés de duplicar conteúdo

Regras Não Ativam

Problema: Regras auto attached não ativam quando esperado

Solução:

- Verifique padrões glob
- Teste com arquivos reais
- Considere usar **always** apply para regras críticas

Conflitos entre Regras

Problema: User Rules e Project Rules conflitando

Solução:

- Mantenha User Rules genéricas
- Torne Project Rules específicas
- Remova duplicações

Performance Degradada

Problema: Sistema mais lento após migração

Solução:

- Reduza número de regras **always** apply

```
- Otimize padrões glob
- Divida regras muito grandes

## Ferramentas de Apoio

### Script de Análise
❏ bash
#!/bin/bash
# Analisa tamanho das regras atuais
echo "Analisando regras atuais..."
wc -l user-rules-*.md
echo "Recomendação: User Rules devem ter < 50 linhas"
```

Validador de Front-matter

```
import yaml
import os

def validate_frontmatter(file_path):
    with open(file_path, 'r') as f:
        content = f.read()

    if not content.startswith('---'):
        return False, "Missing front-matter"

    try:
        _, frontmatter, _ = content.split('---', 2)
        data = yaml.safe_load(frontmatter)

        required_fields = ['description', 'globs', 'alwaysApply']
        for field in required_fields:
            if field not in data:
                return False, f"Missing required field: {field}"

        return True, "Valid"
    except Exception as e:
        return False, f"Invalid YAML: {e}"

# Uso
for file in os.listdir('.cursor/rules/'):
    if file.endswith('.mdc'):
        valid, message = validate_frontmatter(f'.cursor/rules/{file}')
        print(f"{file}: {message}")
```

Cronograma Sugerido

Semana 1: Preparação

- [] Análise do conteúdo atual
- [] Categorização das regras
- [] Criação do plano de migração

Semana 2: User Rules

- [] Extração das regras essenciais
- [] Criação das User Rules otimizadas
- [] Teste inicial

Semana 3: Project Rules Core

- [] Migração das regras fundamentais (00-40)
- [] Teste de funcionalidade básica
- [] Ajustes iniciais

Semana 4: Project Rules Específicas

- [] Migração das regras específicas (50-90)
- [] Configuração de globs
- [] Teste completo

Semana 5: Refinamento

- [] Otimização de performance
- [] Ajuste de referências cruzadas
- [] Documentação final

Referências

- [User Rules Guide](#) (user-rules-guide.md)
- [Project Rules Guide](#) (project-rules-guide.md)
- [Templates](#) (../templates/)
- [Cursor Rules Documentation](https://docs.cursor.com/en/context/rules) (https://docs.cursor.com/en/context/rules)