

Taskmaster AI - Komplettes Cheat Sheet

Strukturierte KI-Entwicklung ohne Chaos



Inhaltsverzeichnis

1. Was ist Taskmaster AI?
 2. Installation & Setup
 3. Projekt-Initialisierung
 4. Arbeitsablauf (Workflow)
 5. Wichtige Befehle
 6. PRD (Produktanforderungsdokument)
 7. Task-Management
 8. Komplexitätsanalyse
 9. Task Expansion
 10. Integration mit Claude Code
 11. Tipps & Best Practices
 12. Fehlerbehebung
-



Was ist Taskmaster AI?

Taskmaster AI ist ein MCP-Tool (Model Control Protocol), das komplexe Entwicklungsprojekte in überschaubare, logische Schritte zerlegt.

Das Problem ohne Taskmaster:

- Chaotische KI-Entwicklung
- Softwarefehler durch unstrukturierte Prompts
- Kontextverlust bei komplexen Projekten
- Ungenaue Ergebnisse

Die Lösung mit Taskmaster:

- Systematische Aufgabenzerlegung
 - Strukturierte, funktionierende Ergebnisse
 - Klare Arbeitsschritte mit Titel und Beschreibung
 - Automatische Komplexitätsanalyse
-

Installation & Setup

Globale Installation (empfohlen)

```
npm install -g task-master-ai
```

Lokale Installation

```
npm install task-master-ai
```

Warum global?

- Nutzung für alle Projekte
- Einfachere Befehlsausführung
- Einmalige Konfiguration

Projekt-Initialisierung

Grundinitialisierung

```
task-master init
```







Mit spezifischen Regeln (für Claude Code)

```
task-master init --rules claude
```

Setup-Prozess:

1. **Regelprofil auswählen** → Claude Code
2. **AI-Modelle konfigurieren:**
 - Hauptmodell: Sonnet
 - Research-Modell: Opus
 - Fallback-Modell: Sonnet
3. **API-Keys einrichten** (optional für externe APIs)

Automatisch erstellte Struktur:

```
 .taskmaster/  
   templates/  
     prd.txt           # Produktanforderungsdokument  
     .env.example      # API-Schlüssel Vorlage  
     claude.md         # Claude-Integration  
     mcp.json          # MCP-Konfiguration
```



Arbeitsablauf (Workflow)

Schritt 1: PRD erstellen

- Detaillierte Projektbeschreibung in `prd.txt`
- Je präziser das PRD, desto bessere Tasks

Schritt 2: PRD parsen

```
taskmaster parse PRD
```

Schritt 3: Tasks auflisten

```
taskmaster list
```

Schritt 4: Komplexität analysieren

```
taskmaster analyze complexity
```

Schritt 5: Tasks implementieren

- Task auswählen und starten
- KI arbeitet strukturiert ab
- Status wird automatisch aktualisiert

Wichtige Befehle

Projekt-Management

Befehl	Beschreibung
<code>task-master init</code>	Neues Projekt initialisieren
<code>task-master init --rules claude</code>	Mit Claude Code Regeln initialisieren
<code>task-master models</code>	Modelle und API-Keys anzeigen

Task-Verwaltung

Befehl	Beschreibung
<code>taskmaster parse PRD</code>	PRD in Tasks umwandeln
<code>taskmaster list</code>	Alle Tasks anzeigen
<code>taskmaster next</code>	Nächste Task anzeigen
<code>taskmaster expand [task-id]</code>	Task in Subtasks aufteilen

Analyse & Optimierung

Befehl	Beschreibung
<code>taskmaster analyze complexity</code>	Komplexitätsbewertung aller Tasks
<code>taskmaster status</code>	Aktueller Projektstatus

Task-Ausführung

Befehl	Beschreibung
<code>taskmaster start [task-id]</code>	Spezifische Task starten
<code>taskmaster complete [task-id]</code>	Task als erledigt markieren



PRD (Produktanforderungsdokument)

Was ist ein PRD?

Das PRD ist der **Bauplan** für dein gesamtes Projekt. Es beschreibt detailliert, was deine App können soll.

PRD-Struktur:

```
# Projekt Titel

## Übersicht
- Was soll die App tun?
- Wer ist die Zielgruppe?

## Funktionen
- Feature 1: Detaillierte Beschreibung
- Feature 2: Detaillierte Beschreibung
- ...

## Technische Anforderungen
- Technologie-Stack
- Datenbank-Anforderungen
- API-Schnittstellen

## Design-Anforderungen
- UI/UX Beschreibung
- Responsive Design
- Accessibility
```

PRD-Tipps:

- **Je detaillierter, desto besser** die generierten Tasks
 - Verwende **konkrete Beispiele**
 - Beschreibe **Benutzerinteraktionen**
 - Erwähne **technische Constraints**
-

Task-Management

Task-Struktur:

Jede Task enthält:

- **Titel:** Kurze Beschreibung
- **Beschreibung:** Detaillierte Anweisungen
- **Details:** Spezifische Implementierungshinweise
- **Status:** pending, in_progress, done
- **Priority:** Wichtigkeit (1-10)
- **Complexity Score:** Schwierigkeitsgrad (1-10)

Tasks anzeigen:

```
taskmaster list
```

Zeigt übersichtliche Tabelle mit allen Tasks

Nächste Task finden:

```
taskmaster next
```

Taskmaster analysiert:

- Abhängigkeiten zwischen Tasks
- Prioritätslevel
- ID-Reihenfolge

Komplexitätsanalyse

Warum Komplexitätsanalyse?

- Identifiziert zu schwierige Tasks
- Schlägt Aufteilung vor
- Optimiert Entwicklungseffizienz

Komplexität bewerten:

```
taskmaster analyze complexity
```

Complexity Scores:

- **1-3:** Einfach (direkt implementierbar)
- **4-6:** Mittel (machbar in einem Zug)
- **7-10:** Komplex (sollte aufgeteilt werden)

Was passiert bei der Analyse?

- Claude bewertet jede Task einzeln
- Generiert detaillierten Bericht
- Gibt Empfehlungen für Task-Aufteilung
- Priorisiert Tasks nach Machbarkeit



Task Expansion

Wann Tasks aufteilen?

- Complexity Score > 6
- Task zu unspezifisch
- Mehrere unabhängige Schritte

Task expandieren:

```
taskmaster expand [task-id]
```

Mit spezifischem Prompt:

```
taskmaster expand 5 --prompt "Fokus auf Sicherheitsaspekte"
```

Expansion-Prozess:

1. Task-ID auswählen
2. Expansion-Prompt definieren (optional)
3. Taskmaster generiert Subtasks
4. Originaltask wird in kleinere Schritte zerlegt
5. Jede Subtask erhält eigene Beschreibung

Beispiel Expansion:

Vorher:

- Task 5: "Benutzerauthentifizierung implementieren"

Nachher:

- Task 5.1: "Login-Form erstellen"
 - Task 5.2: "Passwort-Validation implementieren"
 - Task 5.3: "JWT-Token Generation"
 - Task 5.4: "Session-Management"
-

Integration mit Claude Code

Warum Claude Code?

- Nahtlose Integration mit Taskmaster
- Direkter Zugriff auf MCP-Server
- Strukturierte Befehlsausführung

Setup für Claude Code:

1. Taskmaster mit `--rules claude` initialisieren
2. Claude Code öffnen
3. Taskmaster-Befehle direkt verwenden

Workflow in Claude Code:

```
Benutzer: "Starte Task 1"
|
Claude:
├─ Lädt Task-Details aus tasks.json
├─ Versteht spezifische Anforderungen
├─ Schreibt strukturierten Code
├─ Führt Shell-Befehle aus
└─ Aktualisiert Task-Status auf "done"
```

Natürliche Sprache verwenden:

```
"Was sind die nächsten verfügbaren Tasks?"
"Implementiere Task 4"
"Analysiere die Komplexität unserer Tasks"
"Teile Task 5 in Subtasks auf"
```

Tipps & Best Practices

PRD-Optimierung:

- **Verwende Beispiele** für besseres Verständnis
- **Beschreibe User Stories** detailliert
- **Erwähne Edge Cases** und Fehlerfälle
- **Spezifiziere Datenstrukturen** genau



Task-Management:

- **Arbeite sequenziell** durch die Tasks
- **Verifiziere Ergebnisse** vor dem nächsten Schritt
- **Teile komplexe Tasks** rechtzeitig auf
- **Nutze Tags** für verschiedene Projektphasen

Effizienz-Steigerung:

- **Research-Modell nutzen** für bessere Qualität
- **Regelmäßige Komplexitätsanalyse** durchführen
- **Klare Commit-Messages** verwenden
- **Tests schreiben** für jede implementierte Task

Projektorganisation:

 Projekt/	
├──  .taskmaster/	# Taskmaster-Konfiguration
├──  src/	# Hauptcode
├──  tests/	# Tests für Tasks
├──  docs/	# Dokumentation
└──  README.md	# Projekt-Übersicht



Fehlerbehebung

Häufige Probleme:

"0 tools enabled" in MCP-Settings

Lösung:

```
{
  "mcpServers": {
    "task-master-ai": {
      "command": "npx",
      "args": ["-y", "task-master-ai"], // --package=task-master-ai
entfernen
      "env": {
        "ANTHROPIC_API_KEY": "your-key-here"
      }
    }
  }
}
```

PRD Parse schlägt fehl

Ursachen & Lösungen:

- PRD zu unspezifisch → Mehr Details hinzufügen
- Fehlende API-Keys → Schlüssel in .env konfigurieren
- Syntaxfehler im PRD → Markdown-Format prüfen

Tasks werden nicht generiert

Prüfe:

- PRD-Datei existiert in `.taskmaster/templates/prd.txt`
- API-Keys sind korrekt konfiguriert
- Internet-Verbindung für AI-Modelle

Komplexitätsanalyse funktioniert nicht

Lösung:






- Research-Modell konfigurieren
- Fallback-Modell definieren
- API-Limits prüfen

Debugging-Befehle:

```
task-master --version      # Version prüfen
task-master --help         # Hilfe anzeigen
task-master models         # Modell-Konfiguration anzeigen
```

Zusammenfassung

Warum Taskmaster AI nutzen?

-  **Strukturierte Entwicklung** statt chaotischer Versuche
-  **Präzise Ergebnisse** durch klare Aufgabenteilung
-  **Zeitersparnis** durch automatische Task-Generierung
-  **Bessere Code-Qualität** durch systematisches Vorgehen
-  **Skalierbarkeit** für komplexe Projekte

Der Taskmaster-Vorteil:

"Anstatt chaotischer Versuche und mehrfacher Korrekturen bekommst du strukturierte, funktionierende Ergebnisse."

Nächste Schritte:

1. Taskmaster AI installieren
 2. Erstes Projekt initialisieren
 3. PRD schreiben
 4. Tasks generieren lassen
 5. Strukturiert entwickeln
-

Weitere Ressourcen:

- GitHub: [eyaltoledano/claude-task-master](https://github.com/eyaltoledano/claude-task-master)
 - Dokumentation: docs.task-master.dev
 - NPM Package: [task-master-ai](https://www.npmjs.com/package/task-master-ai)
-