Pi Cheat Sheet für Windows 11

Kurz, praxisnah und auf schnelle Produktivität ausgelegt.

---

1. Was ist Pi?

Pi ist ein minimaler terminalbasierter Coding-Agent.
Er kann mit Modellen sprechen und standardmäßig in deinem Projekt:

- Dateien **lesen**
- Dateien **schreiben**
- Dateien **patchen**
- Shell-Kommandos **ausführen**

Pi arbeitet direkt im aktuellen Projektordner.

---

2. Schnellstart unter Windows 11

Voraussetzungen

Du brauchst:

- **Node.js / npm**
- **Git Bash** oder einen anderen `bash` unter Windows
  Pi braucht auf Windows **bash**.

Laut Doku sucht Pi bash in dieser Reihenfolge:

1. `shellPath` aus `~/.pi/agent/settings.json`
2. Git Bash: `C:\Program Files\Git\bin\bash.exe`
3. `bash.exe` im PATH

Installation

```
npm install -g --ignore-scripts @earendil-works/pi-coding-agent
```

Dann in dein Projekt:

```
cd /pfad/zu/deinem/projekt
pi
```

---

3. Authentifizierung

Pi kann mit **Subscription-Logins** oder **API Keys** arbeiten.

Option A: Login in Pi

In Pi:

```
/login
```

Dann Provider auswählen.

Unterstützt laut Doku u. a.:

- ChatGPT Plus/Pro (Codex)
- Claude Pro/Max
- GitHub Copilot

Option B: API Key per Umgebungsvariable

Beispiel Anthropic in Git Bash:

```
export ANTHROPIC_API_KEY=sk-ant-...
pi
```

Oder OpenAI:

```
export OPENAI_API_KEY=sk-...
pi
```

---

4. Windows-11-Empfehlung: Das beste Setup

Empfohlen

- **Windows Terminal**
- **Git Bash**
- optional: **VS Code** für Editieren
- optional: API-Key oder `/login`

Wichtige Windows-Terminal-Konfiguration

Damit `Shift+Enter` und `Alt+Enter` sauber an Pi weitergegeben werden, in `settings.json` von **Windows Terminal** ergänzen:

```
{
  "actions": [
    {
      "command": { "action": "sendInput", "input": "\u001b[13;2u" },
      "keys": "shift+enter"
    },
    {
      "command": { "action": "sendInput", "input": "\u001b[13;3u" },
      "keys": "alt+enter"
    }
  ]
}
```

Das bringt dir:

- `Shift+Enter` = neue Zeile im Editor
- `Alt+Enter` = Follow-up-Message queueen

> Ohne das klaut Windows Terminal oft `Alt+Enter` für Fullscreen.

Falls Git Bash nicht gefunden wird

Datei:

```
~/.pi/agent/settings.json
```

Beispiel:

```
{
  "shellPath": "C:\\Program Files\\Git\\bin\\bash.exe"
}
```

---

5. Erste produktive Nutzung

In deinem Projektordner:

```
pi
```

Dann einfach schreiben:

```
Analysiere dieses Repository und sag mir, wie ich die Tests starte.
```

Oder:

```
Finde die wichtigsten Einstiegspunkte im Code und erkläre die Architektur.
```

---

6. Die wichtigsten Built-in-Tools

Standardmäßig hat Pi:

- `read` – Dateien lesen
- `write` – Dateien neu erstellen/überschreiben
- `edit` – Dateien patchen
- `bash` – Shell-Befehle ausführen

Zusätzliche read-only Tools:

- `grep`
- `find`
- `ls`

---

7. Die wichtigsten Tastenkürzel

Eingabe

- `Enter` → absenden
- `Shift+Enter` → neue Zeile
- `Tab` → Autocomplete / Pfadvervollständigung
- `Ctrl+G` → externen Editor öffnen (`$VISUAL` oder `$EDITOR`)

Steuerung

- `Escape` → abbrechen
- `Ctrl+D` → beenden (wenn Editor leer)
- `Ctrl+C` → Editor leeren / Auswahl abbrechen

Modell / Thinking

- `Ctrl+L` → Modell wählen
- `Ctrl+P` → nächstes Modell
- `Shift+Ctrl+P` → vorheriges Modell
- `Shift+Tab` → Thinking-Level wechseln
- `Ctrl+T` → Thinking-Blocks ein-/ausblenden

Queue / Verlauf

- `Alt+Enter` → Follow-up-Nachricht queueen
- `Alt+Up` → queuete Nachrichten zurück in den Editor holen

---

8. Die wichtigsten Slash Commands

Muss man kennen

- `/login` – anmelden
- `/logout` – abmelden

- `/model` – Modell wechseln
- `/settings` – Einstellungen
- `/resume` – alte Session öffnen
- `/new` – neue Session
- `/name <name>` – Session benennen
- `/session` – Infos zur Session
- `/tree` – Session-Baum / Branches
- `/fork` – neue Session aus früherer User-Nachricht
- `/clone` – aktuellen Branch duplizieren
- `/compact` – Kontext manuell komprimieren
- `/reload` – Extensions, Skills, Prompts etc. neu laden
- `/hotkeys` – Tastenkürzel anzeigen
- `/quit` – Pi beenden

---

9. Die wichtigsten CLI-Befehle

Interaktiv starten

```
pi
```

Mit initialem Prompt

```
pi "Finde alle wichtigen Build-Skripte"
```

One-shot / Print-Modus

```
pi -p "Fasse dieses Repository zusammen"
```

Mit Datei-Referenzen

```
pi @README.md "Fasse diese Datei zusammen"
pi @src/app.ts @src/app.test.ts "Prüfe beide Dateien zusammen"
```

Session fortsetzen

```
pi -c
```

Session-Auswahl

```
pi -r
```

Session benennen

```
pi --name "Refactor Login"
```

Bestimmte Session öffnen

```
pi --session <path|id>
```

Ohne Session-Speicherung

```
pi --no-session
```

10. Shell-Befehle in Pi

Mit Ausgabe an das Modell

Im Pi-Editor:

```
!npm test
```

Die Ausgabe wird dem Modell als Kontext gegeben.

Ohne Ausgabe an das Modell

```
!!npm test
```

Praktisch für Dinge, die du nur lokal ausführen willst.

---

11. Dateien referenzieren

Interaktiv

Im Editor `@` tippen → fuzzy Suche nach Dateien.

Direkt auf der CLI

```
pi @README.md "Erkläre mir das Projekt"
```

---

12. Unbedingt nutzen: AGENTS.md

Wenn du Pi schnell produktiv machen willst, ist das der größte Hebel.

Projektweit

Datei im Projekt:

```
AGENTS.md
```

Beispiel:

```
# Projektanweisungen


- Führe nach Codeänderungen `npm run check` aus.
- Keine produktiven Migrationen lokal ausführen.
- Halte Antworten kurz.
- Bevorzuge kleine, gezielte Änderungen.
- Nutze zuerst read/grep/find statt unnötiger bash-Kommandos.
```

Wo Pi Kontextdateien lädt

- `~/.pi/agent/AGENTS.md` → global
- `AGENTS.md` oder `CLAUDE.md` im aktuellen oder übergeordneten Verzeichnis

Nach Änderung:

```
/reload
```

oder Pi neu starten.

---

13. Die wichtigsten Settings unter Windows 11

Dateien:

- global: `~/.pi/agent/settings.json`
- projektbezogen: `.pi/settings.json`

Sehr nützliche Minimal-Konfiguration

```
{
  "theme": "dark",
  "quietStartup": false,
  "defaultThinkingLevel": "medium",
  "warnings": {
    "anthropicExtraUsage": true
  },
  "retry": {
    "enabled": true,
    "maxRetries": 3,
    "baseDelayMs": 2000,
    "provider": {
      "maxRetries": 0,
      "maxRetryDelayMs": 60000
    }
  },
  "compaction": {
    "enabled": true,
    "reserveTokens": 16384,
    "keepRecentTokens": 20000
  }
}
```

Windows-spezifisch mit Git Bash

```
{
  "shellPath": "C:\\Program Files\\Git\\bin\\bash.exe"
}
```

---

14. Modelle schnell wechseln

In Pi

```
/model
```

Auf CLI

```
pi --provider openai --model gpt-4o "Hilf mir beim Refactoring"
```

Oder:

```
pi --model openai/gpt-4o "Hilf mir beim Refactoring"
```

Oder mit Thinking-Level:

```
pi --model sonnet:high "Löse dieses komplexe Problem"
```

---

15. Thinking-Level verstehen

Mögliche Werte:

- `off`
- `minimal`

- `low`
- `medium`
- `high`
- `xhigh`

Faustregel

- **off/minimal** → schnell, günstig
- **medium** → guter Standard
- **high/xhigh** → für komplexe Architektur, Debugging, Refactoring, Planungen

Wechseln:

- `Shift+Tab`
- `/settings`
- `--thinking <level>`

Beispiel:

```
pi --thinking high
```

---

16. Session-Workflow, den du wirklich brauchst

Weitermachen

```
pi -c
```

Historie durchsuchen

```
pi -r
```

Neue Session

```
/new
```

Session benennen

```
/name Bugfix Auth Timeout
```

Baum/Navigation

```
/tree
```

Das ist extrem nützlich, um frühere Gesprächsstände wieder aufzunehmen.

Fork aus früherer User-Nachricht

```
/fork
```

Aktuellen Branch kopieren

```
/clone
```

---

17. Der beste Windows-Workflow im Alltag

Workflow A: Repo verstehen

```
cd projekt
pi
```

Dann:

```
Analysiere dieses Repository. Erkläre Architektur, wichtige Einstiegspunkte, Build/Test-Kommandos und typische Entwicklungsabläufe.
```

Workflow B: Bug fixen

```
Reproduziere das Problem anhand des Codes. Finde die Ursache. Schlage einen minimalen Fix vor und führe danach die relevanten Checks aus.
```

Workflow C: Refactoring

```
Analysiere das Modul src/... und schlage ein kleines, risikoarmes Refactoring vor. Erkläre erst den Plan, dann führe es aus und prüfe die Änderungen.
```

Workflow D: Testausfälle

```
Untersuche, warum die Tests fehlschlagen. Nutze bevorzugt vorhandene Test- und Lint-Kommandos. Führe danach nur die relevanten Checks erneut aus.
```

---

18. Praktische Prompts für produktives Arbeiten

Projekt-Scan

```
Fasse dieses Repository zusammen und nenne mir die wichtigsten Kommandos für Build, Test und Lint.
```

Code Review

```
Prüfe die zuletzt geänderten Dateien auf Bugs, Edge Cases und fehlende Tests.
```

Implementierung

```
Implementiere Feature X minimal-invasiv im bestehenden Stil des Projekts. Erkläre kurz deinen Plan und führe danach die passenden Checks aus.
```

Debugging

```
Untersuche die Ursache für den Fehler. Gehe hypothesengetrieben vor und validiere jede Annahme mit Code oder Kommandos.
```

Tests

```
Finde heraus, wie ich die relevanten Tests für dieses Modul ausführen kann, und führe dann nur diese aus.
```

---

19. Read-only / sicherer Modus

Wenn du Pi erstmal nur analysieren lassen willst:

```
pi --tools read,grep,find,ls -p "Reviewe den Code"
```

Das gibt Pi **keine Schreibrechte**.

Sehr empfehlenswert am Anfang.

---

20. Nützliche Umgebungsvariablen

Wichtige Standardvariablen

- `ANTHROPIC_API_KEY`
- `OPENAI_API_KEY`
- `GEMINI_API_KEY`

Pi-spezifisch

- `PI_CODING_AGENT_DIR` → anderes Config-Verzeichnis
- `PI_CODING_AGENT_SESSION_DIR` → anderes Session-Verzeichnis
- `PI_OFFLINE=1` → keine Startup-Netzwerkoperationen
- `PI_SKIP_VERSION_CHECK=1` → Versionscheck aus
- `PI_TELEMETRY=0` → Telemetrie aus

Beispiel in Git Bash:

```
export PI_SKIP_VERSION_CHECK=1
export PI_TELEMETRY=0
pi
```

---

21. Wo Pi seine Daten speichert

Standardmäßig unter:

```
~/.pi/agent/
```

Wichtig:

- `auth.json` → Zugangsdaten
- `settings.json` → globale Settings
- `keybindings.json` → Keybindings
- `AGENTS.md` → globale Anweisungen
- `sessions/` → Sessions
- `extensions/` → Extensions
- `skills/` → Skills
- `prompts/` → Prompt Templates
- `themes/` → Themes

---

22. Eigene Keybindings

Datei:

```
~/.pi/agent/keybindings.json
```

Beispiel:

```
{
  "tui.input.newLine": ["shift+enter", "ctrl+j"],
  "app.model.select": ["ctrl+l"]
}
```

Unter Windows kann `ctrl+j` als Fallback für neue Zeile nützlich sein.

Danach:

```
/reload
```

---

23. Prompt Templates, Skills, Extensions – was ist was?

Prompt Templates

Wiederverwendbare Textbausteine, die per `/name` expandieren.

Skills

Größere, wiederverwendbare Fähigkeiten, die Pi on-demand lädt.

Extensions

TypeScript-Erweiterungen für Tools, Commands, Events, UI, Provider etc.

Für schnellen Start unter Windows

Am Anfang brauchst du meist nur:

- gutes `AGENTS.md`
- sinnvolle `settings.json`
- optional ein paar Prompt Templates

---

24. Nützliche One-shot-Kommandos

README zusammenfassen

```
pi -p @README.md "Fasse das prägnant zusammen"
```

Code + Test gemeinsam prüfen

```
pi -p @src/app.ts @src/app.test.ts "Prüfe Implementierung und Tests zusammen"
```

Stdin pipen

```
cat README.md | pi -p "Fasse den Text zusammen"
```

Unter Windows/Git Bash ebenfalls nutzbar.

---

25. Fehlerbehebung unter Windows 11

Problem: Pi startet nicht wegen bash

Lösung:

- Git Bash installieren
- `shellPath` setzen

```
{
  "shellPath": "C:\\Program Files\\Git\\bin\\bash.exe"
}
```

Problem: `Shift+Enter` geht nicht

Lösung:

- Windows Terminal `sendInput`-Mapping setzen
- alternativ anderes Terminal testen

Problem: `Alt+Enter` geht nicht

Lösung:

- Windows Terminal remappen, sonst nutzt es Fullscreen

Problem: Auth klappt nicht

Check:

- API-Key korrekt gesetzt?
- richtiges Terminal?
- `/login` statt ENV probieren
- `auth.json` prüfen

Problem: Pi arbeitet zu aggressiv im Dateisystem

Lösung:

- read-only starten:

```
pi --tools read,grep,find,ls
```

- Git verwenden
- vorher committen

---

26. Meine klare Empfehlung für dein Start-Setup

`~/.pi/agent/settings.json`

```
{
  "shellPath": "C:\\Program Files\\Git\\bin\\bash.exe",
  "theme": "dark",
  "defaultThinkingLevel": "medium",
  "compaction": {
    "enabled": true,
    "reserveTokens": 16384,
    "keepRecentTokens": 20000
  },
  "retry": {
    "enabled": true,
    "maxRetries": 3,
    "baseDelayMs": 2000,
    "provider": {
      "maxRetries": 0,
      "maxRetryDelayMs": 60000
    }
  }
}
```

`~/.pi/agent/AGENTS.md`

```
# Global Pi Instructions

- Be concise and practical.
- Prefer minimal, low-risk code changes.
- Before changing code, inspect the surrounding files and existing conventions.
- After code changes, suggest or run the most relevant checks.
- On Windows, prefer commands that work in Git Bash.
```

Projekt-`AGENTS.md`

```
# Project Instructions

- Run `npm test` only when relevant.
- Prefer targeted tests over full-suite runs.
- Keep changes small and easy to review.
- Do not modify CI, release, or deployment config unless explicitly requested.
```

---

27. Die 15 wichtigsten Dinge auf einen Blick

1. Installieren:

```
npm install -g --ignore-scripts @earendil-works/pi-coding-agent
```

2. Starten:

```
pi
```

3. Unter Windows brauchst du **bash**.
4. Bestes Setup: **Windows Terminal + Git Bash**.
5. Login:

```
/login
```

6. API-Key alternativ via ENV.
7. Neue Zeile:
   `Shift+Enter`
8. Modell wechseln:
   `Ctrl+L`
9. Shell-Befehl mit Kontext:

```
!npm test
```

10. Shell-Befehl ohne Kontext:

```
!!npm test
```

11. Datei referenzieren:
    `@`
12. Alte Session:

```
pi -c
```

13. Session-Baum:

```
/tree
```

14. Projektanweisungen in:

```
AGENTS.md
```

15. Sicher analysieren:

```
pi --tools read,grep,find,ls
```

---

28. Mein empfohlener erster echter Einsatz

Schritt 1

```
cd dein-projekt
pi
```

Schritt 2

```
Analysiere dieses Repository. Erkläre mir:
1. Architektur
2. wichtige Einstiegspunkte
3. Build/Test/Lint-Kommandos
4. typische Entwickler-Workflows
5. Risiken oder Stolperfallen
```

Schritt 3

Dann:

```
Erstelle mir einen konkreten Plan, wie wir in diesem Repository produktiv mit dir arbeiten sollten.
```

Revision #1
Created 2026-06-02 18:47:57 UTC by art10m
Updated 2026-06-02 18:49:54 UTC by art10m