Web-Tools (z.B. NVM, NodeJS)

- NVM, NodeJS
  - NVM & Node.js Cheat Sheet
  - Node.js Update Script für Windows 11 (mit NVM)
  - Erweitertes Script - Mit Cleanup alter Versionen
- pi.dev
  - Was ist Pi.Dev?
  - Ausführliche Zusammenfassung: "Babysitter" - Ein KI-Orchestrierungssystem für deterministische Ergebnisse

NVM, NodeJS

NVM, NodeJS

NVM & Node.js Cheat Sheet

1. NVM Installation & Updates

Installation (Linux/macOS)

```
# Installation via curl
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.7/install.sh | bash

# ODER via wget
wget -qO- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.7/install.sh | bash

# Nach Installation: Terminal neu starten oder:
source ~/.bashrc        # für Bash
source ~/.zshrc         # für Zsh
```

Installation (Windows)

```
# nvm-windows verwenden (separates Projekt)
# Download: https://github.com/coreybutler/nvm-windows/releases

# Nach Installation via Installer:
nvm version
```

NVM aktualisieren

```
# Aktuelle Version prüfen
nvm --version

# Update durch Neuinstallation (neueste Version in URL einsetzen)
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.7/install.sh | bash

# Globale Pakete nach Update migrieren
```

```
nvm reinstall-packages <alte-version>
```

---

2. Node.js Versionen verwalten

Verfügbare Versionen anzeigen

```
# Alle installierbaren Versionen (sehr lange Liste)
nvm ls-remote

# Nur LTS-Versionen anzeigen
nvm ls-remote --lts

# Bestimmte Major-Version filtern
nvm ls-remote 20              # alle Node 20.x.x Versionen
nvm ls-remote --lts=iron      # LTS "Iron" (Node 20)
nvm ls-remote --lts=hydrogen  # LTS "Hydrogen" (Node 18)
```

Installierte Versionen anzeigen

```
# Alle lokalen Installationen
nvm ls
nvm list

# Aktuelle aktive Version
nvm current
node --version
```

Node.js installieren

```
# Neueste Version installieren
nvm install node

# Neueste LTS-Version installieren
nvm install --lts
```

```
# Spezifische Version installieren
nvm install 20.11.0           # exakte Version
nvm install 20                 # neueste 20.x.x
nvm install 18.19.0

# Mit Migration globaler Pakete von anderer Version
nvm install 20 --reinstall-packages-from=18

# Neueste npm-Version gleich mit installieren
nvm install 20 --latest-npm
```

Node.js Version wechseln

```
# Zu bestimmter Version wechseln (nur aktuelle Shell)
nvm use 20.11.0
nvm use 20
nvm use --lts
nvm use node                   # neueste installierte

# Standardversion festlegen (für neue Shells)
nvm alias default 20.11.0
nvm alias default lts/*
nvm alias default node

# Temporär andere Version für einen Befehl
nvm exec 18 node app.js
nvm exec 18 npm test
nvm run 18 app.js
```

Node.js deinstallieren

```
# Bestimmte Version entfernen
nvm uninstall 16.20.0

# Aktuelle Version kann nicht deinstalliert werden
# Erst wechseln, dann deinstallieren:
nvm use 20
nvm uninstall 18
```

Aliase verwalten

```
# Alias erstellen
nvm alias mein-projekt 18.19.0
nvm alias stable 20.11.0

# Alias verwenden
nvm use mein-projekt

# Alias entfernen
nvm unalias mein-projekt

# Alle Aliase anzeigen
nvm alias
```

---

3. NPM verwalten

NPM Version prüfen & aktualisieren

```
# Aktuelle Version anzeigen
npm --version
npm -v

# NPM auf neueste Version aktualisieren
npm install -g npm@latest

# Bestimmte NPM-Version installieren
npm install -g npm@10.2.0

# NPM mit Node-Installation aktualisieren
nvm install-latest-npm
```

NPM Cache verwalten

```
# Cache prüfen
npm cache verify
```

```
# Cache komplett leeren
npm cache clean --force

# Cache-Pfad anzeigen
npm config get cache
```

Globale Pakete verwalten

```
# Globale Pakete auflisten
npm list -g --depth=0

# Global installieren
npm install -g <paket>

# Global deinstallieren
npm uninstall -g <paket>

# Veraltete globale Pakete anzeigen
npm outdated -g

# Alle globalen Pakete aktualisieren
npm update -g
```

Lokale Pakete verwalten

```
# Abhängigkeiten installieren (aus package.json)
npm install
npm i

# Paket als Dependency hinzufügen
npm install <paket>
npm install <paket>@version

# Paket als DevDependency hinzufügen
npm install <paket> --save-dev
npm install <paket> -D
```

```
# Paket entfernen
npm uninstall <paket>

# Veraltete Pakete anzeigen
npm outdated

# Alle Pakete aktualisieren (nach semver)
npm update

# Sicherheitslücken prüfen
npm audit

# Sicherheitslücken automatisch beheben
npm audit fix
npm audit fix --force          # auch breaking changes
```

package-lock.json

```
# Exakte Versionen aus lock-file installieren (CI/CD)
npm ci

# Lock-file neu generieren
rm package-lock.json
npm install
```

---

4. Projekt-Konfiguration

.nvmrc Datei (automatische Node-Version)

```
# .nvmrc im Projektroot erstellen
echo "20.11.0" > .nvmrc
# ODER
echo "lts/*" > .nvmrc
# ODER
node -v > .nvmrc
```

```
# Version aus .nvmrc verwenden
nvm use
nvm install    # installiert falls nötig
```

Automatisches Wechseln (Shell-Konfiguration)

Für Bash (~/.bashrc):

```
# Am Ende der Datei hinzufügen:
cdnvm() {
    command cd "$@" || return $?
    nvm_path=$(nvm_find_up .nvmrc | tr -d '\n')
    if [[ ! $nvm_path = *[^[:space:]]* ]]; then
        declare default_version;
        default_version=$(nvm version default);
        if [[ $default_version == "N/A" ]]; then
            nvm alias default node;
            default_version=$(nvm version default);
        fi
        if [[ $(nvm current) != "$default_version" ]]; then
            nvm use default;
        fi
    elif [[ -s $nvm_path/.nvmrc && -r $nvm_path/.nvmrc ]]; then
        declare nvm_version
        nvm_version=$(<"$nvm_path"/.nvmrc)
        declare locally_resolved_nvm_version
        locally_resolved_nvm_version=$(nvm ls --no-colors "$nvm_version" | tail -1 | tr -d '\-
>*' | tr -d '[:space:]')
        if [[ "$locally_resolved_nvm_version" == "N/A" ]]; then
            nvm install "$nvm_version";
        elif [[ $(nvm current) != "$locally_resolved_nvm_version" ]]; then
            nvm use "$nvm_version";
        fi
    fi
}
alias cd='cdnvm'
cdnvm "$PWD"
```

Für Zsh (~/.zshrc):

```
# Automatisches nvm use bei Verzeichniswechsel
autoload -U add-zsh-hook
load-nvmrc() {
  local nvmrc_path="$(nvm_find_nvmrc)"
  if [ -n "$nvmrc_path" ]; then
    local nvmrc_node_version=$(nvm version "$(cat "${nvmrc_path}")")
    if [ "$nvmrc_node_version" = "N/A" ]; then
      nvm install
    elif [ "$nvmrc_node_version" != "$(nvm version)" ]; then
      nvm use
    fi
  elif [ -n "$(PWD=$OLDPWD nvm_find_nvmrc)" ] && [ "$(nvm version)" != "$(nvm version
default)" ]; then
    echo "Reverting to nvm default version"
    nvm use default
  fi
}
add-zsh-hook chpwd load-nvmrc
load-nvmrc
```

engines in package.json

```
{
  "name": "mein-projekt",
  "engines": {
    "node": ">=18.0.0 <21.0.0",
    "npm": ">=9.0.0"
  }
}
```

---

5. LESS verwalten

Installation

```
# Global installieren (CLI überall verfügbar)
npm install -g less
```

```
# Lokal im Projekt (empfohlen)
npm install less --save-dev
```

LESS CLI verwenden

```
# Basis-Kompilierung
lessc styles.less styles.css

# Mit Source Maps
lessc styles.less styles.css --source-map

# Minifiziert (mit clean-css Plugin)
npm install -g less-plugin-clean-css
lessc styles.less styles.css --clean-css

# Bestimmte Pfade für @import
lessc --include-path=node_modules:src/styles styles.less out.css

# Variablen überschreiben
lessc --modify-var="primary-color=blue" styles.less styles.css
```

LESS mit npm Scripts

```
{
  "scripts": {
    "less:compile": "lessc src/styles/main.less dist/css/styles.css",
    "less:watch": "nodemon --watch src/styles -e less --exec 'npm run less:compile'",
    "less:minify": "lessc src/styles/main.less dist/css/styles.min.css --clean-css"
  },
  "devDependencies": {
    "less": "^4.2.0",
    "less-plugin-clean-css": "^1.5.1",
    "nodemon": "^3.0.0"
  }
}
```

LESS mit Build-Tools

Mit Webpack:

```
npm install less less-loader css-loader style-loader --save-dev
```

```
// webpack.config.js
module.exports = {
  module: {
    rules: [
      {
        test: /\.less$/,
        use: [
          'style-loader',
          'css-loader',
          'less-loader'
        ]
      }
    ]
  }
};
```

Mit Gulp:

```
npm install gulp gulp-less gulp-clean-css gulp-sourcemaps --save-dev
```

```
// gulpfile.js
const gulp = require('gulp');
const less = require('gulp-less');
const cleanCSS = require('gulp-clean-css');
const sourcemaps = require('gulp-sourcemaps');

gulp.task('less', function() {
  return gulp.src('src/styles/**/*.less')
    .pipe(sourcemaps.init())
    .pipe(less())
    .pipe(cleanCSS())
    .pipe(sourcemaps.write('.'))
    .pipe(gulp.dest('dist/css'));
});

gulp.task('watch', function() {
```

```
  gulp.watch('src/styles/**/*.less', gulp.series('less'));
});
```

LESS Version aktualisieren

```
# Global
npm update -g less

# Lokal
npm update less

# Auf neueste Major-Version
npm install less@latest --save-dev
```

---

6. SCSS/SASS verwalten

Installation

```
# Dart Sass (empfohlen, offiziell)
npm install -g sass

# Lokal im Projekt
npm install sass --save-dev

# Alternative: node-sass (deprecated, aber noch verwendet)
npm install node-sass --save-dev
```

SASS CLI verwenden

```
# Basis-Kompilierung
sass src/styles.scss dist/styles.css

# Mit Source Maps (Standard: an)
sass src/styles.scss dist/styles.css --source-map
```

```
# Ohne Source Maps
sass src/styles.scss dist/styles.css --no-source-map

# Komprimiert/Minifiziert
sass src/styles.scss dist/styles.css --style=compressed

# Watch-Modus (einzelne Datei)
sass --watch src/styles.scss:dist/styles.css

# Watch-Modus (Verzeichnisse)
sass --watch src/styles:dist/css

# Mehrere Dateien/Verzeichnisse
sass src/main.scss:dist/main.css src/admin.scss:dist/admin.css

# Load-Path für @use und @import
sass --load-path=node_modules --load-path=src/styles src/main.scss dist/main.css
```

SASS mit npm Scripts

```
{
  "scripts": {
    "sass:compile": "sass src/scss:dist/css",
    "sass:watch": "sass --watch src/scss:dist/css",
    "sass:prod": "sass src/scss:dist/css --style=compressed --no-source-map",
    "sass:lint": "stylelint 'src/scss/**/*.scss'"
  },
  "devDependencies": {
    "sass": "^1.69.0",
    "stylelint": "^15.11.0",
    "stylelint-config-standard-scss": "^11.0.0"
  }
}
```

SASS mit Build-Tools

Mit Webpack:

```
npm install sass sass-loader css-loader style-loader --save-dev
```

```
// webpack.config.js
module.exports = {
  module: {
    rules: [
      {
        test: /\.s[ac]ss$/,
        use: [
          'style-loader',
          'css-loader',
          {
            loader: 'sass-loader',
            options: {
              // Dart Sass verwenden
              implementation: require('sass'),
              sassOptions: {
                outputStyle: 'compressed',
              },
            },
          },
        ],
      },
    ],
  },
};
```

Mit Vite (modern, empfohlen):

```
npm install sass --save-dev
```

```
// vite.config.js
import { defineConfig } from 'vite';

export default defineConfig({
  css: {
    preprocessorOptions: {
      scss: {
        additionalData: `@use "@/styles/variables" as *;`
      }
    }
  }
```

```
});
```

Mit Gulp:

```
npm install gulp gulp-sass sass gulp-clean-css gulp-sourcemaps --save-dev
```

```
// gulpfile.js
const gulp = require('gulp');
const sass = require('gulp-sass')(require('sass'));
const cleanCSS = require('gulp-clean-css');
const sourcemaps = require('gulp-sourcemaps');

gulp.task('sass', function() {
  return gulp.src('src/scss/**/*.scss')
    .pipe(sourcemaps.init())
    .pipe(sass().on('error', sass.logError))
    .pipe(cleanCSS())
    .pipe(sourcemaps.write('.'))
    .pipe(gulp.dest('dist/css'));
});

gulp.task('watch', function() {
  gulp.watch('src/scss/**/*.scss', gulp.series('sass'));
});
```

SASS Linting (Stylelint)

```
# Installation
npm install stylelint stylelint-config-standard-scss --save-dev
```

```
// .stylelintrc.json
{
  "extends": "stylelint-config-standard-scss",
  "rules": {
    "selector-class-pattern": null,
    "scss/dollar-variable-pattern": null
  }
}
```

```
# Ausführen
npx stylelint "src/scss/**/*.scss"
npx stylelint "src/scss/**/*.scss" --fix
```

SASS Version aktualisieren

```
# Global
npm update -g sass

# Lokal
npm update sass

# Migration von node-sass zu dart-sass
npm uninstall node-sass
npm install sass --save-dev
# Code-Anpassungen: @import → @use/@forward
```

---

7. Nützliche Kombinationen

Komplettes Projekt-Setup

```
# 1. Node-Version festlegen
nvm install 20 --lts
nvm use 20
echo "20" > .nvmrc

# 2. Projekt initialisieren
npm init -y

# 3. SCSS einrichten
npm install sass --save-dev

# 4. package.json Scripts
```

```
{
  "scripts": {
```

```
    "dev": "npm run sass:watch",
    "build": "npm run sass:prod",
    "sass:compile": "sass src/scss:dist/css",
    "sass:watch": "sass --watch src/scss:dist/css",
    "sass:prod": "sass src/scss:dist/css --style=compressed --no-source-map"
  }
}
```

Globale Pakete bei Node-Update migrieren

```
# Aktuelle globale Pakete auflisten (zum Dokumentieren)
npm list -g --depth=0 > global-packages.txt

# Neue Node-Version mit Paket-Migration
nvm install 22 --reinstall-packages-from=20

# Oder manuell nach Update
nvm install 22
nvm use 22
npm install -g less sass typescript nodemon
```

Troubleshooting

```
# NVM nicht gefunden nach Installation
source ~/.bashrc   # oder ~/.zshrc

# Berechtigungsprobleme
# NVM installiert in $HOME, sollte keine sudo-Rechte brauchen

# node/npm nicht gefunden
nvm use default
# oder
nvm alias default node

# Paket-Konflikte
rm -rf node_modules package-lock.json
npm cache clean --force
npm install
```

```
# SASS Kompilierungsfehler nach Update
npm rebuild sass

# node-gyp Fehler (bei nativen Modulen)
npm install -g node-gyp
node-gyp rebuild
```

? Schnellreferenz

Aufgabe	Befehl
NVM Version	`nvm --version`
Node installieren	`nvm install 20`
Node wechseln	`nvm use 20`
Default setzen	`nvm alias default 20`
NPM aktualisieren	`npm install -g npm@latest`
LESS installieren	`npm install -g less`
SASS installieren	`npm install -g sass`
LESS kompilieren	`lessc input.less output.css`
SASS kompilieren	`sass input.scss output.css`
SASS watch	`sass --watch src:dist`

Erstellt: Mai 2026 | NVM v0.39.x | Node.js 18-22 | SASS 1.x | LESS 4.x

NVM, NodeJS

Node.js Update Script für Windows 11 (mit NVM)

Hier ist ein PowerShell-Script, das alles automatisch erledigt:

? `update-node.ps1`

```
# ============================================
# Node.js Update Script für Windows 11 (NVM)
# ============================================

Write-Host "`n□□Node.js Update Script gestartet...`n" -ForegroundColor Cyan

# 1. NVM-Liste anzeigen und neueste Version finden
Write-Host "□□Suche neueste Node.js Version..." -ForegroundColor Yellow

# NVM list available ausgeben und analysieren
$nvmOutput = cmd /c "nvm list available" 2>&1

Write-Host "`nVerfügbare Versionen:" -ForegroundColor Gray
Write-Host $nvmOutput -ForegroundColor DarkGray

# Die erste LTS Version aus der Tabelle extrahieren (erste Zeile nach Header, erste Spalte)
$lines = $nvmOutput -split "`n"
$latestLTS = $null

foreach ($line in $lines) {
    # Suche nach Zeilen die mit einer Versionsnummer beginnen (nach |)
    if ($line -match '\|\s*(\d+\.\d+\.\d+)\s*\|') {
        $latestLTS = $matches[1]
        break
    }
}
```

```
# Fallback: Versuche alternative Parsing-Methode
if (-not $latestLTS) {
    foreach ($line in $lines) {
        if ($line -match '(\d+\.\d+\.\d+)') {
            $latestLTS = $matches[1]
            break
        }
    }
}

# Wenn immer noch nichts gefunden, manuelle Eingabe
if (-not $latestLTS) {
    Write-Host "`n⚠  Konnte Version nicht automatisch ermitteln." -ForegroundColor Red
    Write-Host "   Bitte schaue oben in der Tabelle nach der neuesten LTS Version.`n" -
ForegroundColor Yellow
    $latestLTS = Read-Host "   Gib die gewünschte Version ein (z.B. 22.16.0)"
}

if (-not $latestLTS) {
    Write-Host " Keine Version angegeben. Abbruch." -ForegroundColor Red
    exit 1
}

Write-Host "`n   →  Ausgewählte Version: $latestLTS" -ForegroundColor Green

# 2. Neueste Version installieren
Write-Host "`n Installiere Node.js $latestLTS..." -ForegroundColor Yellow
cmd /c "nvm install $latestLTS"

# 3. Neue Version aktivieren
Write-Host "`n Aktiviere Node.js $latestLTS..." -ForegroundColor Yellow
cmd /c "nvm use $latestLTS"

# Kurze Pause damit NVM die Änderung übernimmt
Start-Sleep -Seconds 2

# 4. Überprüfen
Write-Host "`n Aktuelle Versionen:" -ForegroundColor Yellow
```

```
try {
    $nodeVersion = cmd /c "node -v" 2>&1
    $npmVersion = cmd /c "npm -v" 2>&1
    Write-Host "   Node.js: $nodeVersion" -ForegroundColor Green
    Write-Host "   NPM: $npmVersion" -ForegroundColor Green
} catch {
    Write-Host "   ⚠  Konnte Versionen nicht abfragen - ggf. neues Terminal öffnen" -
ForegroundColor Yellow
}

# 5. NPM selbst aktualisieren
Write-Host "`n Aktualisiere NPM auf neueste Version..." -ForegroundColor Yellow
cmd /c "npm install -g npm@latest"

# 6. Globale Pakete installieren/aktualisieren
Write-Host "`n Installiere globale Pakete..." -ForegroundColor Yellow

$packages = @(
    "less",
    "sass",
    "typescript",
    "npm-check-updates",
    "nodemon",
    "live-server"
)

foreach ($package in $packages) {
    Write-Host "   → Installiere $package..." -ForegroundColor Gray
    cmd /c "npm install -g $package" 2>$null
}

# 7. Zusammenfassung
Write-Host "`n" -NoNewline
Write-Host ("=" * 50) -ForegroundColor Cyan
Write-Host " FERTIG! Zusammenfassung:" -ForegroundColor Green
Write-Host ("=" * 50) -ForegroundColor Cyan

$finalNode = cmd /c "node -v" 2>&1
$finalNpm = cmd /c "npm -v" 2>&1
```

```
Write-Host "`n   Node.js Version: $finalNode" -ForegroundColor White
Write-Host "   NPM Version: $finalNpm" -ForegroundColor White
Write-Host "`n   Installierte globale Pakete:" -ForegroundColor White
cmd /c "npm list -g --depth=0"

Write-Host "`n□□Alles erledigt!`n" -ForegroundColor Green
Write-Host "□□Tipp: Öffne ein NEUES Terminal damit alle Änderungen aktiv sind.`n" -
ForegroundColor Yellow
```

? Einrichtung & Nutzung

Option 1: Direkt ausführen

```
# PowerShell als Administrator öffnen, dann:
Set-ExecutionPolicy -Scope CurrentUser RemoteSigned
.\update-node.ps1
```

Option 2: Als Batch-Datei (Doppelklick)

Erstelle `update-node.bat`:

```
@echo off
PowerShell -ExecutionPolicy Bypass -File "%~dp0update-node.ps1"
pause
```

?? Pakete anpassen

Bearbeite diese Liste im Script nach deinen Wünschen:

```
$packages = @(
    "less",           # LESS Compiler
    "sass",           # SCSS/SASS Compiler
    "typescript",     # TypeScript
    "npm-check-updates",  # Dependency Updates checken
    "nodemon",        # Auto-Restart bei Änderungen
```

```
    "live-server"     # Dev-Server mit Hot-Reload
    # Füge hier weitere hinzu:
    # "gulp-cli",
    # "webpack-cli",
    # "eslint",
    # "prettier"
)
```

?? Wichtig

- **Als Administrator ausführen** (Rechtsklick → "Als Administrator ausführen")
- NVM für Windows muss bereits installiert sein
- Falls nicht: https://github.com/coreybutler/nvm-windows/releases

Soll ich das Script noch anpassen oder weitere Pakete hinzufügen? □□

NVM, NodeJS

Erweitertes Script - Mit Cleanup alter Versionen

Hier ist das erweiterte Script, das nach der Installation alle anderen Node.js Versionen entfernt:

? `update-node.ps1` (Mit Cleanup)

```
# ============================================
# Node.js Update Script für Windows 11 (NVM)
# LTS VERSION + CLEANUP
# ============================================

Write-Host "`n□□Node.js Update Script gestartet...`n" -ForegroundColor Cyan

# 1. Aktuelle installierte Versionen anzeigen
Write-Host "□□Aktuell installierte Node.js Versionen:" -ForegroundColor Yellow
$installedBefore = cmd /c "nvm list" 2>&1
Write-Host $installedBefore -ForegroundColor DarkGray

# 2. NVM-Liste anzeigen
Write-Host "`n□□Suche neueste Node.js LTS Version..." -ForegroundColor Yellow

$nvmOutput = cmd /c "nvm list available" 2>&1

Write-Host "`nVerfügbare Versionen:" -ForegroundColor Gray
Write-Host $nvmOutput -ForegroundColor DarkGray

# ============================================
# NVM LIST AVAILABLE Format:
# |   CURRENT    |     LTS      |  OLD STABLE  | OLD UNSTABLE |
# |--------------|--------------|--------------|--------------
# |    24.1.0    |   22.16.0    |   0.12.18    |   0.11.16    |
# ============================================
```

```
# Die LTS Version ist in der ZWEITEN Spalte!
$lines = $nvmOutput -split "`n"
$latestLTS = $null

foreach ($line in $lines) {
    # Suche nach Zeilen mit dem Tabellenformat und extrahiere die ZWEITE Versionsnummer (LTS)
    if ($line -match '^\s*\|\s*[\d.]+\s*\|\s*([\d]+\.[\d]+\.[\d]+)\s*\|') {
        $latestLTS = $matches[1]
        break
    }
}

# Fallback: Alternative Methode
if (-not $latestLTS) {
    foreach ($line in $lines) {
        $versions = [regex]::Matches($line, '(\d+\.\d+\.\d+)')
        if ($versions.Count -ge 2) {
            $latestLTS = $versions[1].Value
            break
        }
    }
}

# Wenn immer noch nichts gefunden, manuelle Eingabe
if (-not $latestLTS) {
    Write-Host "`n⚠  Konnte LTS Version nicht automatisch ermitteln." -ForegroundColor Red
    Write-Host "   Bitte schaue oben in der Tabelle - LTS ist die ZWEITE Spalte.`n" -
ForegroundColor Yellow
    $latestLTS = Read-Host "   Gib die gewünschte LTS Version ein (z.B. 22.16.0)"
}

if (-not $latestLTS) {
    Write-Host " Keine Version angegeben. Abbruch." -ForegroundColor Red
    exit 1
}

Write-Host "`n   ➡  Ausgewählte LTS Version: $latestLTS" -ForegroundColor Green
```

```
# Bestätigung
Write-Host ""
$confirm = Read-Host "   Fortfahren mit Version $latestLTS? (J/n)"
if ($confirm -eq "n" -or $confirm -eq "N") {
    $latestLTS = Read-Host "   Gib die gewünschte Version ein"
}

# 3. LTS Version installieren
Write-Host "`n□□Installiere Node.js $latestLTS (LTS)..." -ForegroundColor Yellow
cmd /c "nvm install $latestLTS"

# 4. LTS Version aktivieren
Write-Host "`n□ Aktiviere Node.js $latestLTS..." -ForegroundColor Yellow
cmd /c "nvm use $latestLTS"

# Kurze Pause damit NVM die Änderung übernimmt
Start-Sleep -Seconds 3

# 5. Überprüfen
Write-Host "`n□□Aktuelle Versionen:" -ForegroundColor Yellow
$nodeVersion = cmd /c "node -v" 2>&1
$npmVersion = cmd /c "npm -v" 2>&1
Write-Host "   Node.js: $nodeVersion" -ForegroundColor Green
Write-Host "   NPM: $npmVersion" -ForegroundColor Green

# ============================================
# 6. CLEANUP - Alte Versionen entfernen
# ============================================
Write-Host "`n□□Cleanup: Entferne alte Node.js Versionen..." -ForegroundColor Yellow

# Installierte Versionen abrufen
$nvmList = cmd /c "nvm list" 2>&1
$installedVersions = @()

foreach ($line in ($nvmList -split "`n")) {
    # Versionsnummern extrahieren (Format: "  * 22.16.0 (Currently using...)" oder "
20.10.0")
    if ($line -match '(\d+\.\d+\.\d+)') {
        $version = $matches[1]
```

```
        if ($version -ne $latestLTS) {
            $installedVersions += $version
        }
    }
}

# Duplikate entfernen
$installedVersions = $installedVersions | Select-Object -Unique

if ($installedVersions.Count -eq 0) {
    Write-Host "   ✓ Keine alten Versionen gefunden - alles sauber!" -ForegroundColor Green
} else {
    Write-Host "`n   Folgende alte Versionen werden entfernt:" -ForegroundColor Yellow
    foreach ($version in $installedVersions) {
        Write-Host "   • $version" -ForegroundColor DarkGray
    }

    Write-Host ""
    $confirmDelete = Read-Host "   Alle alten Versionen löschen? (J/n)"

    if ($confirmDelete -ne "n" -and $confirmDelete -ne "N") {
        foreach ($version in $installedVersions) {
            Write-Host "   □□□ Entferne Node.js $version..." -ForegroundColor Gray
            cmd /c "nvm uninstall $version" 2>&1 | Out-Null
        }
        Write-Host "`n   ✓ Alle alten Versionen entfernt!" -ForegroundColor Green
    } else {
        Write-Host "   □□  Cleanup übersprungen." -ForegroundColor Yellow
    }
}

# 7. NPM selbst aktualisieren
Write-Host "`n□□Aktualisiere NPM auf neueste Version..." -ForegroundColor Yellow
cmd /c "npm install -g npm@latest"

# 8. NPM Cache leeren (optional aber empfohlen)
Write-Host "`n□□Leere NPM Cache..." -ForegroundColor Yellow
cmd /c "npm cache clean --force" 2>&1 | Out-Null
Write-Host "   ✓ Cache geleert" -ForegroundColor Green
```

```
# 9. Globale Pakete installieren/aktualisieren
Write-Host "`n□□Installiere globale Pakete..." -ForegroundColor Yellow

$packages = @(
    "less",
    "sass",
    "typescript",
    "npm-check-updates",
    "nodemon",
    "live-server"
)

foreach ($package in $packages) {
    Write-Host "   → Installiere $package..." -ForegroundColor Gray
    cmd /c "npm install -g $package" 2>$null
}

# 10. Zusammenfassung
Write-Host "`n" -NoNewline
Write-Host ("=" * 50) -ForegroundColor Cyan
Write-Host "□ FERTIG! Zusammenfassung:" -ForegroundColor Green
Write-Host ("=" * 50) -ForegroundColor Cyan

$finalNode = cmd /c "node -v" 2>&1
$finalNpm = cmd /c "npm -v" 2>&1

Write-Host "`n   Node.js Version: $finalNode (LTS)" -ForegroundColor White
Write-Host "   NPM Version: $finalNpm" -ForegroundColor White

Write-Host "`n   Installierte Node.js Versionen:" -ForegroundColor White
cmd /c "nvm list"

Write-Host "`n   Installierte globale Pakete:" -ForegroundColor White
cmd /c "npm list -g --depth=0"

Write-Host "`n□□Alles erledigt!" -ForegroundColor Green
Write-Host "□□Tipp: Öffne ein NEUES Terminal damit alle Änderungen aktiv sind.`n" -
ForegroundColor Yellow
```

? Neue Features:

Feature	Beschreibung
□□ **Vorher-Übersicht**	Zeigt installierte Versionen vor dem Update
□□ **Auto-Cleanup**	Findet und listet alle alten Versionen
□ **Bestätigung**	Fragt vor dem Löschen nach Bestätigung
□□ **Batch-Deinstallation**	Entfernt alle alten Versionen automatisch
□□ **NPM Cache**	Leert auch den NPM Cache
□□ **Nachher-Übersicht**	Zeigt am Ende nur noch die aktive Version

?? Ausführen:

```
.\update-node.ps1
```

? Beispiel-Ablauf:

```
□□Node.js Update Script gestartet...

□□Aktuell installierte Node.js Versionen:
    20.10.0
    18.19.0
  * 22.11.0 (Currently using 64-bit executable)

□□Suche neueste Node.js LTS Version...
   →□  Ausgewählte LTS Version: 22.16.0
   Fortfahren mit Version 22.16.0? (J/n): J

□□Installiere Node.js 22.16.0 (LTS)...
□ Aktiviere Node.js 22.16.0...

□□Cleanup: Entferne alte Node.js Versionen...

   Folgende alte Versionen werden entfernt:
   • 20.10.0
   • 18.19.0
```

```
   • 22.11.0

   Alle alten Versionen löschen? (J/n): J
   □□□ Entferne Node.js 20.10.0...
   □□□ Entferne Node.js 18.19.0...
   □□□ Entferne Node.js 22.11.0...

   ✓ Alle alten Versionen entfernt!

□□Alles erledigt!
```

Jetzt hast du ein sauberes System mit nur einer Node.js Version! □□□

pi.dev

KI-Agent

Was ist Pi.Dev?

Einleitung und Kontext

Das Video stellt **Pi** vor (auch bekannt als "Shitty Coding Agent" oder offiziell unter pi.dev erreichbar) - einen minimalistischen, erweiterbaren Coding-Agenten, der sich bewusst von der Feature-Überladung anderer Tools absetzt. Der Ersteller des Videos präsentiert Pi als eine erfrischende Alternative zu etablierten Agenten wie Claude Code, Codex oder Open Code.

https://youtu.be/OMFIPv8a4qA

Der Schöpfer: Mario Zechner (BadLogic)

Pi wurde von **Mario Zechner** entwickelt, bekannt unter seinem Online-Pseudonym "BadLogic". Er ist ein erfahrener Open-Source-Enthusiast, der in der Vergangenheit **libGDX** entwickelt hat - eines der populärsten Game-Development-Frameworks für Java. Mario war frustriert von allen existierenden Coding-Agenten und entschied sich, einen eigenen zu bauen, der seiner Philosophie entspricht.

Zitat von Mario:

> "Ich hasse alle existierenden Coding-Agenten. Pi's Philosophie ist: Passe den Coding-Agenten an deine Bedürfnisse an, anstatt umgekehrt."

Die Kernphilosophie: Was Pi NICHT hat

Das Besondere an Pi ist ironischerweise das, was es **nicht** mitbringt. Der Video-Ersteller betont wiederholt, dass die Stärke in der Abwesenheit von Features liegt:

Bewusst weggelassene Features:

- **Keine MCPs** (Model Context Protocol Server)
- **Keine Sub-Agenten**
- **Kein Plan-Modus**
- **Kein Background-Bash**
- **Keine Permission-Pop-ups**
- **Keine übermäßig komplexe UI**

Kritik an Claude Code:

Mario beschreibt Claude Code als "Raumschiff" mit einer exzessiven Menge an Features:

> "Füge dieses Feature hinzu und jenes Feature... irgendwann ist Claude Code zu einem Raumschiff geworden. Es macht so viele Dinge, dass du wahrscheinlich nur 5% davon jemals nutzt. Du kennst vielleicht 10% insgesamt, und der Rest - die übrigen 90% - das ist wie die dunkle Materie der KI."

Kritik an Open Code:

- Häufige Session-Kompaktierung macht den Agenten "dümmer" während Gesprächen
- Stilles Löschen von Tool-Outputs nach einer bestimmten Token-Menge
- LSP-Unterstützung (Language Server Protocol) kann problematisch sein, da der Agent während der Implementierung sofortiges (falsches) Feedback erhält, bevor die Änderungen fertig sind

---

Technische Details und Installation

Installation:

```
npm install
```

Pi ist in **TypeScript** geschrieben - was der Video-Ersteller humorvoll als "die Sprache für KI" bezeichnet.

Konfiguration:

- Einstellungen unter: `~/.pi/agent_settings.json`
- Sessions werden als JSON-Dateien gespeichert (im Gegensatz zu Open Code, das kürzlich auf SQLite umgestellt hat)
- System-Prompt ist minimal und vollständig anpassbar

Provider-Unterstützung:

Pi ist **modell-agnostisch** und unterstützt:

- Anthropic (Claude)
- OpenAI
- Ollama (lokale Modelle)
- Andere lokale Provider
- "Big Pickle" (kostenloser Anbieter für einfache Anfragen)

Wichtige Funktionen und Shortcuts

Session-Management:

Befehl/Shortcut	Funktion
`pi`	Agent starten
`pi -r`	Session fortsetzen
`--no-session`	Ohne Session-Speicherung ausführen
`Ctrl+D`	Session beenden, zurück zur CLI
`Ctrl+G`	Prompt zum Editor senden
`/tree`	Hierarchie der Session-Branches anzeigen
`!`	Ad-hoc Bash-Befehle ausführen

Session-Branching:

Eine besonders nützliche Funktion ist die Möglichkeit, **Sessions zu forken** und von jedem Punkt in der Konversation abzuzweigen. Dies hilft dabei:

- "Side Quests" zu vermeiden, die das Context Window verschmutzen
- Halluzinationen zu reduzieren
- Die Fehlerrate zu senken

Gist-Export:

Sessions können exportiert werden als:

- Durchsuchbare HTML-Dateien
- GitHub Private Gists

File-Tagging:

Ähnlich wie bei Cursor können Dateien getaggt werden, um das Context-Management zu verbessern und präzisere Antworten zu erhalten.

Bild-Unterstützung:

Bilder können durch Einfügen des Pfads verwendet werden (die Drag-and-Drop-Funktionalität ist laut Video-Ersteller etwas unklar dokumentiert).

Erweiterbarkeit: Das Pi-Ökosystem

Obwohl Pi minimalistisch ist, bietet es umfangreiche Erweiterungsmöglichkeiten:

Erweiterungstypen:

- **Tools**: Zusätzliche Werkzeuge
- **Commands**: Benutzerdefinierte Befehle
- **Shortcuts**: Tastenkürzel
- **Themes**: Visuelle Anpassungen
- **Prompts/Templates**: Slash-Befehle für häufige Aktionen

Prompt-Templates:

Unter dem Ordner `prompts` können eigene Slash-Befehle erstellt werden, z.B.:

- `/review` - Code auf Bugs und Sicherheitsprobleme prüfen

Packages installieren:

```
pi install <package-path>
```

Bekannte Erweiterungen:

- **Babysitter**: Reduziert Halluzinationen (verlangsamt aber den Prozess)
- **Sub-Agent-Erweiterungen**: Für diejenigen, die Sub-Agenten möchten
- Themes und Prompts auf der offiziellen Package-Website

Skills-Integration:

Pi erkennt automatisch bereits installierte Skills von Claude Code oder Open Code und macht diese verfügbar.

Inline-Ausführung und Aliase

One-Shot-Queries:

Mit `-p` kann Pi für einzelne Anfragen verwendet werden:

```
pi -p "Was belegt Port 8080?"
```

Modell-Liste anzeigen:

```
pi list models
```

Eigene Aliase erstellen:

```
alias q='pi -p --model <günstiges-modell>'
```

Dann einfach:

```
q "Wie spät ist es in Tokyo?"
```

Context ist König

Der Video-Ersteller betont wiederholt die Bedeutung des **Context Windows**:

> "Wenn es eine Sache gibt, auf die wir uns einigen können, dann ist es, dass Context König ist. Oder genauer gesagt: Context, sowohl seine Qualität als auch seine Größe, wird bestimmen, wie glücklich oder frustriert du sein wirst."

Probleme mit überladenem Context:

- Halluzinationen häufen sich
- Fehlerfrequenz steigt
- Agent wird "dümmer"

Pi's Lösungsansätze:

- Session-Branching und Forking
- Minimaler System-Prompt
- Keine automatische Session-Kompaktierung wie bei anderen Tools

---

Vergleich mit anderen Tools

Feature	Pi	Claude Code	Open Code
MCPs	(erweiterbar)		
Sub-Agenten	(erweiterbar)		
Plan-Modus			
Session-Kompaktierung			
Erweiterbarkeit	Hoch	Gering	⚠ Mittel
Footprint	Sehr klein	Groß	Mittel
Session-Branching		⚠	⚠

Praktische Anwendung des Video-Erstellers

Der Ersteller beschreibt seinen eigenen Workflow:

1. **Home-Server-Setup**: Pi läuft auf einem neuen Home-Server

2. **Kontextdatei**: Eine gut gestaltete MD-Datei mit Instruktionen und Context
3. **Soft Guardrails**: Eine Datei mit grundlegenden Richtlinien, um sich nicht ständig wiederholen zu müssen
4. **Tmux-Integration**: Statt Sub-Agenten werden Tmux-Sessions und Fork-Sessions verwendet

---

Zusätzliches Wissen über Pi

Aus der Community und Dokumentation:

1. **Awesome-Pi**: Es gibt eine community-gepflegte "Awesome Pi"-Seite mit Ressourcen und Erweiterungen
2. **Thorsten Ball's Artikel**: Der Ersteller referenziert einen Artikel, der zeigt, dass man mit nur ~400 Zeilen Code einen funktionierenden Agenten bauen kann - beschrieben als "Der Kaiser hat keine Kleider"
3. **Anerkennung von Experten**: Pi wird gelobt von:
   - Peter Steinberg (Ersteller von Open Claw)
   - Einem der Unity-Gründer
   - Dem Ersteller von Flask und Jinja (Armin Ronacher), der kürzlich den Entwickler hinter Pi eingestellt hat
4. **GitHub-Erfolg**: Pi wird als Agent beschrieben, der eines der am schnellsten wachsenden Projekte in der GitHub-Geschichte antreibt
5. **Philosophie der Einfachheit**:

   > "Alle diese Agenten sind im Grunde nur Schleifen über den Modellen."

Fazit des Video-Erstellers

Positiv:

- Schlank und schnell
- Einfach zu installieren
- Geringe Einstiegshürde
- Hohe Erweiterbarkeit bei Bedarf
- Session-Branching sehr nützlich
- Transparente Funktionsweise

Neutral/Abwägend:

- Noch keine SQLite-basierte Session-Verwaltung (JSON-Dateien)
- Bild-Unterstützung könnte klarer dokumentiert sein
- Für "Vibe Coding" und LinkedIn-Posts über Agent-Schwärme nicht geeignet

Endurteil:

> "Es ist noch zu früh, um es sicher zu sagen, aber Pi ist ein Keeper."

Der Ersteller betont, dass er trotz seiner Begeisterung für Open Code (mit all seinen Features) die Schlankheit von Pi sehr schätzt. Die Abwesenheit von Features ist nicht als Mangel zu verstehen, sondern als bewusste Designentscheidung, die Entwicklern ermöglicht, genau das hinzuzufügen, was sie brauchen - nicht mehr und nicht weniger.

Empfehlung

Pi eignet sich besonders für:

- Entwickler, die Kontrolle über ihre Tools schätzen
- Minimalisten, die Feature-Bloat vermeiden wollen
- Diejenigen, die ihre Agenten selbst erweitern möchten
- Terminal-Power-User
- Entwickler, die mit verschiedenen LLM-Providern arbeiten

Pi eignet sich weniger für:

- Entwickler, die "out of the box" alles haben wollen
- Diejenigen, die komplexe Multi-Agenten-Setups ohne eigene Konfiguration erwarten
- "Vibe Coders", die maximale Automatisierung suchen

Ausführliche Zusammenfassung: "Babysitter" - Ein KI-Orchestrierungssystem für deterministische Ergebnisse

Einleitung und Problemstellung

Der Videoersteller beginnt mit einer ehrlichen Bestandsaufnahme der aktuellen KI-Landschaft: **Niemand weiß wirklich, wohin sich KI entwickelt** – auch wenn viele so tun, als hätten sie alle Antworten. Was er jedoch mit Sicherheit sagen kann, ist, dass KI-Systeme häufig ihr Ziel verfehlen.

Als zahlender Kunde (200 Dollar monatlich bei OpenAI, früher auch bei Anthropic) hat er praktisch jedes verfügbare Modell getestet – von OpenRouter über verschiedene Quirks-Modelle bis hin zu OpenCode, Zen, Go und Black. Seine ernüchternde Erkenntnis: **Unabhängig von der Preisstufe oder dem investierten Geld halluzinieren KI-Systeme ständig.**

Zwar setzen alle Entwickler Leitplanken und Regeln, aber – so der Sprecher mit einem Augenzwinkern – diese werden von den KIs oft einfach ignoriert. Die Systeme halten sich nicht wirklich an Vorgaben.

https://youtu.be/xNDxh35Dva4

Die Lösung: Babysitter von A5C.AI

Das Versprechen

Was wäre, wenn es ein System gäbe – keinen Zaubertrick, keine Illusion – sondern eine **tatsächliche Methode, um Aufgaben mit deterministischen Ergebnissen zu orchestrieren** ? Der Sprecher ist deutlich: Wenn jemand das verkaufen würde, wäre er sofort Kunde.

Die gute Nachricht: **Niemand verkauft es** – denn es ist Open Source. Das Projekt namens "Babysitter" von A5C.AI soll dabei helfen, "Gehorsam durchzusetzen" und halluzinationsfreie

Ausgaben anzustreben.

Die Website und das Team

Die Website A5C.AI präsentiert sich mit dem typischen "VIP-Coded-Landing-Page"-Feeling, aber das Team scheint es ernst zu meinen mit dem Open-Source-Gedanken. Der Slogan **"Von Entwicklern für Entwickler"** ist etwas, das der Sprecher schon lange nicht mehr gehört hat – und das er positiv hervorhebt.

Das mysteriöse Team hinter A5C scheint sich dem Projekt verschrieben zu haben und bietet auch kostenpflichtige Unterstützung für große Organisationen an.

Die Kernprobleme, die Babysitter adressiert

1. **Alles ist eine Blackbox** – es gibt keine echte Prüfmöglichkeit
2. **Das Kontextfenster würgt einen ab** – zu viele Informationen, zu wenig Struktur
3. **Wer entscheidet, wann etwas fertig ist?** – Spoiler: Nicht der Nutzer

Die Lösung durch Babysitter

Babysitter implementiert:

- **Quality Gates** (Qualitätsschranken): Schleifen, die laufen, bis bestimmte Kriterien erfüllt sind
- **Definition of Done**: Ein echtes Konzept, nicht nur eine Metapher
- **Deterministische Workflows**: Vorhersagbare Ergebnisse durch strukturierte Prozesse

Die vier Betriebsmodi

Das Projekt bietet vier verschiedene Optionen:

1. **Call**: Der Standardaufruf für normale Aufgaben
2. **Plan Only**: Nur Planung ohne Ausführung
3. **YOLO-Modus**: "Forever Mode" – das System läuft autonom (riskanter)
4. Ein weiterer Modus (im Video nicht vollständig spezifiziert)

Technische Details

Kompatibilität

Babysitter unterstützt:

- CodeX
- Cursor
- Gemini
- GitHub Copilot
- OpenCode
- **Pi** (der im Video verwendete Agent)

Alles ist als "experimentell" gekennzeichnet – was der Sprecher humorvoll kommentiert: "Ist nicht alles in den letzten Jahren experimentell?"

Installation und Ausführung

Babysitter kann auf zwei Arten verwendet werden:

1. **Als Paket/Extension** für den bevorzugten Agenten
2. **Als eigenständiges Harness** (Wrapper)

Die Standalone-Option ist besonders wichtig für **ephemere Umgebungen** wie CI-Worker oder GitHub-Automatisierungsbots.

Installation:

```
npm install @a5c-ai/babysitter-sdk
```

Nach der Installation kann man das Harness direkt ansprechen und einen der vier Modi aufrufen. Als Orchestrator kann "Internal" verwendet werden, aber auch Claude oder andere Optionen sind möglich.

Der Prozess

Ein wesentlicher Unterschied: Prozesse werden in **JavaScript** definiert, nicht in Markdown – also echtem Code im Workspace.

Der Einrichtungsprozess

Schritt 1: Nutzer-Setup

```
babysitter user install
```

Hier lernt das System den Nutzer kennen. Man kann eigene Details festlegen:

- Name
- Spezialisierungen
- Erfahrung
- Verbosity-Präferenzen

Wichtiger Tipp: Kurz und spezifisch halten – und bitte **keine Emojis**!

Das System scannt auch das GitHub-Profil des Nutzers, um Präferenzen zu erkennen (z.B. ob man eher Go/Shell oder JavaScript/Perl bevorzugt).

Schritt 2: Projekt-Setup

Mit jedem neuen Projekt durchläuft man den **Project Install Process**. Dabei:

1. Liest das System den Code
2. Analysiert Dependencies
3. Durchforstet die Git-Historie
4. Baut ein mentales Modell des Projekts
5. Erstellt Konfigurationsdateien unter `.a5c`

Wichtige Fragen während des Setups:

- Was optimierst du? (Sicherheit, Produktion, Zuverlässigkeit)
- Welche Bereiche sind kritisch? (z.B. Datenintegrationen)
- Wie ist dein Workflow? (Sind PRs kritisch? Tools wie WorkTrunk?)

Zeitaufwand

Der Sprecher ist ehrlich: **Das dauert!** Etwa 15 Minuten für den initialen Setup, plus Zeit für Nutzerfragen. Er verwendete GPT 5.5 auf "High"-Reasoning, was langsamer ist, aber bessere Ergebnisse liefert.

Kernbotschaft: Es geht nicht um Geschwindigkeit oder Quantität, sondern um **korrekte Ergebnisse**.

Integrierte Methodologien

Babysitter hat verschiedene Methodologien "eingebrannt":

- **GSD (Getting Stuff Done)**: Eine Produktivitätsmethode
- Debugging-Prozesse
- Testing-Frameworks
- Und viele mehr...

Die Dokumentation listet Prozesse für:

- TDD (Test-Driven Development)
- Web- und Mobile-Entwicklung
- DevOps
- Security
- Geschäftsbereiche
- Sogar Wissenschaft

Der Sprecher gibt zu, dass er diese nicht alle gelesen hat und die Vielfalt etwas besorgniserregend findet.

Praktische Demonstration

Der Doctor-Befehl

Nach dem Setup empfiehlt sich:

```
babysitter doctor
```

Dieser produziert einen Bericht darüber, was funktioniert hat und was noch Aufmerksamkeit braucht.

Beispiel 1: REST-API mit TDD (Call-Modus)

Aufgabe: Eine kleine Task-API mit allen CRUD-Methoden, unter Verwendung von TDD und mit einem Qualitätsziel.

Der Prozess:

1. Babysitter ruft den Orchestrator und die Skill-Anweisungen auf
2. Es druckt einen Plan mit Request-Endpoints und Quality Gates aus
3. **Interessant:** Es generiert eigenen Orchestrierungscode (TypeScript!)
4. Das System fragt um **Genehmigung** bevor es fortfährt (Approval Gate)
5. **Erster generierter Code:** Die Tests!
6. Erst danach der Code, der die Tests besteht

Ergebnis:

- Endpoints funktionieren
- Tests laufen mit Coverage
- Spec-Code vorhanden
- Alles wie angewiesen

Beobachtung zur Präzision:
Das System ist **extrem literal** in der Umsetzung. Es gab keine Logs – weil keine angefordert wurden. Das kann gut oder problematisch sein und erfordert **Detailgenauigkeit** bei der Planung.

Test der API:

- `GET /tasks` → Leeres Array ✓
- `POST` mit leeren Daten → Korrekte Fehlerbehandlung ✓
- `POST` mit korrektem JSON → Task erscheint ✓
- `DELETE` → Funktioniert ✓
- `GET /tasks` → Wieder leer ✓

Beispiel 2: YOLO-Modus mit generischer Aufgabe

Parallel wurde eine vage Aufgabe im YOLO-Modus gestartet. Das Ergebnis:

- Es existiert etwas (Login, Aktionen)
- Keine echte Struktur
- Es läuft... irgendwie
- Fehler, die der Sprecher nicht debuggen wollte

Fazit: Der strukturierte Ansatz mit klaren Anweisungen liefert bessere Ergebnisse.

Beobachtungen zu TDD

Der Sprecher macht eine interessante Nebenbemerkung: Obwohl er keine Beweise hat, liefern KI-Agenten **bessere Ergebnisse**, wenn man sie TDD (Test-Driven Development) verwenden lässt. Das ist keine isolierte Beobachtung – andere haben Ähnliches bemerkt.

Ironischerweise wurde ihm TDD immer beigebracht, aber er hat es nie richtig praktiziert. Jetzt, mit KI-"Minions", die noch nicht ablehnen können, ist es endlich möglich!

Über Pi – Der verwendete Agent

Was ist Pi?

Pi ist der im Video verwendete **Open-Source-KI-Agent**, der als "der schlankste und minimalistischste Open-Source-Agent" beschrieben wird. Er hat Babysitter als eines seiner optionalen Erweiterungspakete gelistet.

Installation mit Pi

```
pi install
```

Nach der Installation zeigt Pi neue Optionen für Babysitter. Der Sprecher empfiehlt:

```
babysitter help
```

...um alle verfügbaren Optionen zu sehen (es sind viele!).

Pi's Philosophie

Basierend auf den Informationen aus dem Video und allgemeinem Wissen über Pi:

1. **Minimalismus:** Pi verfolgt einen schlanken Ansatz ohne Bloat
2. **Erweiterbarkeit:** Optionale Pakete wie Babysitter können hinzugefügt werden
3. **Open Source:** Vollständig quelloffen und community-getrieben
4. **Agenten-Framework:** Fungiert als Wrapper/Framework für KI-Interaktionen

Nutzung mit Babysitter

```
# Ad-hoc Session
pi -p "dein prompt hier"

# Mit Babysitter call
babysitter call "baue eine task api mit allen methoden, nutze tdd, qualitätsziel: tests
bestehen"

# Mit YOLO-Modus (autonomer)
babysitter yolo "generische aufgabe"
```

Kernerkenntnisse und Fazit

Was Babysitter NICHT ist:

- Kein Zauberer
- Keine Wunderlösung
- Kein System, das alles alleine macht

Was Babysitter IST:

- Ein **Orchestrator**
- Ein System für **deterministische Workflows**
- Ein Tool, das **Quality Gates** erzwingt
- **Modell-agnostisch** (funktioniert mit verschiedenen LLMs)
- **Open Source** und **datenschutzfreundlich** (keine Daten werden gesendet)

Die Rolle des Nutzers

Der Sprecher betont mehrfach: **Der Nutzer ist ein wesentlicher Teil des Prozesses.** Babysitter ist wie ein professioneller Mitarbeiter – wenn man ein Profi ist, verhält es sich professionell. Aber man muss:

- Klare Anweisungen geben
- Detailliert sein in der Planung
- Quality Gates definieren
- Genehmigungen erteilen

Abschließende Bewertung

Der Sprecher ist positiv überrascht und plant, das System in den kommenden Wochen und Monaten weiter zu testen. Er bittet die Community, ihre Erfahrungen zu teilen.

Wichtige Einschränkung: Babysitter ist ohne einen ordentlichen Agenten (wie Pi) drumherum nicht viel wert. Es ist ein Orchestrierungslayer, kein eigenständiges Tool.

Ergänzende Informationen zu Pi (aus externem Wissen)

Pi Agent – Technischer Hintergrund

Pi ist ein Terminal-basierter KI-Agent, der:

1. **CLI-First Design:** Entwickelt für die Kommandozeile
2. **Lightweight:** Minimaler Ressourcenverbrauch
3. **Plugin-Architektur:** Erweiterbar durch Pakete wie Babysitter
4. **Multi-Model Support:** Kann mit verschiedenen LLM-Backends arbeiten

Warum Pi für Babysitter?

Die Kombination macht Sinn weil:

- Pi bietet die **Agenten-Infrastruktur** (Tool-Aufrufe, Kontext-Management)
- Babysitter bietet die **Orchestrierungslogik** (Quality Gates, Prozesse)
- Zusammen ermöglichen sie **strukturierte, deterministische Workflows**

Typische Pi-Workflows

```
# Einfache Interaktion
pi "erkläre mir diesen code"

# Mit Datei-Kontext
pi -f main.py "refactore diese funktion"

# Inline-Prompt für Automatisierung
pi -p "erstelle unit tests für alle funktionen"
```

Persönliche Empfehlung des Videoerstellers

1. **Für Einsteiger:** Erst Pi kennenlernen, dann Babysitter hinzufügen
2. **Für Profis:** Quality Gates und TDD-Workflows nutzen
3. **Für alle:** Geduld haben – gute Ergebnisse brauchen Zeit
4. **Experimentieren:** Die YOLO- und Call-Modi haben unterschiedliche Use Cases

---

Diese Zusammenfassung basiert auf dem Transkript des YouTube-Videos und wurde durch allgemeines Wissen über die erwähnten Tools ergänzt.