Agent Bot

macOS Dev, Docker & AI

Ultimate Cheat Sheet

zsh · Homebrew · Git · Node/TS · pnpm · Vite/React
Docker · Playwright · LM Studio · CI/CD

Hannes Schwede

agent-bot.de · 26. Maerz 2026

Inhaltsverzeichnis

1. Navigation & Datei-Operationen
2. Dateien lesen, suchen & bearbeiten
3. macOS Spezifika & System
4. Prozesse, Netzwerk & Ports
5. Git — Basics
6. Git — Fortgeschritten
7. Homebrew & Package Management
8. Node.js, pnpm & Vite
9. Playwright
10. Docker — Images & Container
11. Docker Compose — Orchestrierung
12. CI/CD — GitHub Actions (Referenz)
13. Local AI & LLM Workflow (LM Studio)
14. Power-User — Pipes, Aliases & .zshrc

+ Environment-Strategie (Tabelle)

1. Navigation & Datei-Operationen

```
pwd
# Aktuelles Verzeichnis anzeigen
ls -la
# Alle Dateien inkl. versteckter, mit Rechten
cd ~/Projects/agent-bot
# In Projektordner wechseln
cd -
# Zum vorherigen Verzeichnis springen
mkdir -p src/components/ui
# Verschachtelte Ordner anlegen
touch .env.local
# Leere Datei erstellen / Zeitstempel setzen
cp -r src/ src-backup/
# Ordner rekursiv kopieren
mv old.ts new.ts
# Datei umbenennen oder verschieben
rm -rf dist
# Ordner mit Inhalt loeschen (Vorsicht!)
find . -name "*.ts" -not -path "*/node_modules/*"
# TS-Dateien finden, node_modules ignorieren
```

2. Dateien lesen, suchen & bearbeiten

```
cat package.json
# Kurze Datei im Terminal ausgeben
less long.log
# Lange Datei scrollbar oeffnen (q = Beenden)
head -n 20 app.log
# Erste 20 Zeilen anzeigen
tail -f app.log
# Neue Logzeilen live streamen
grep -rn "API_KEY" ./src
# Rekursiv suchen mit Zeilennummern
rg "API_KEY"
# Schnellere Alternative (brew install ripgrep)
sed -i '' 's/old/new/g' config.ts
# Suchen & Ersetzen in Datei (macOS)
wc -l src/**/*.ts
# Zeilen pro TypeScript-Datei zaehlen
diff file-a.ts file-b.ts
# Unterschiede zwischen Dateien anzeigen
chmod +x setup-project.sh
# Skript ausfuehrbar machen
```

3. macOS Spezifika & System

```
open .
# Aktuellen Ordner im Finder oeffnen
open -a "Google Chrome" http://localhost:5173
# URL in Chrome oeffnen
cat ~/.ssh/id_rsa.pub | pbcopy
# SSH-Key in Zwischenablage kopieren
pbpaste > output.txt
# Zwischenablage in Datei schreiben
qlmanage -p design-draft.png >& /dev/null
# QuickLook-Vorschau aus dem Terminal
caffeinate -d npm run build
# Sleep-Mode verhindern waehrend Build
mdfind -name "briefing"
# Spotlight-Suche im Terminal
defaults write com.apple.finder AppleShowAllFiles YES
# Versteckte Dateien im Finder anzeigen
say "Build fertig"
# Text ueber Lautsprecher ausgeben
xcode-select --install
# Apple CLI-Tools installieren
ipconfig getifaddr en0
# Lokale IPv4-Adresse fuer mobile Tests
```

4. Prozesse, Netzwerk & Ports

```
htop
# CPU/RAM-Auslastung (brew install htop)
ps aux | grep node
# Laufende Node-Prozesse filtern
pkill -9 node
# Alle Node-Prozesse sofort beenden
lsof -i :5173
# Welcher Prozess belegt Port 5173?
kill -9 $(lsof -ti :3000)
# Prozess auf Port 3000 direkt killen
df -h
# Festplattenauslastung anzeigen
du -sh node_modules
# Ordnergroesse pruefen
ping -c 5 google.com
# 5 Pakete senden, dann stoppen
traceroute google.com
# Netzwerk-Route zum Ziel anzeigen
curl -I http://localhost:1234/v1/models
# HTTP-Header eines Endpoints pruefen
nc -vz 127.0.0.1 1234
# Port-Erreichbarkeit testen (TCP)
dig example.com +short
# DNS-Aufloesung pruefen (kompakt)
```

5. Git — Basics

```
git init
# Neues Repository initialisieren
git clone git@github.com:user/repo.git
# Remote-Repo klonen via SSH
git status
# Geaenderte/ungetrackte Dateien anzeigen
git add .
# Alle Aenderungen stagen
git commit -m "feat: implement LLM endpoint"
# Commit mit Conventional-Commit-Message
git push origin main
# Lokale Commits nach Remote pushen
git pull origin main --rebase
# Remote-Aenderungen ohne Merge-Commit
git log --oneline --graph --all
# Kompakte Historie mit Branch-Graph
git branch feature/scanner
# Neuen Branch erstellen
git switch feature/scanner
# Zu Branch wechseln (modern)
```

6. Git — Fortgeschritten

```
git stash
# Aenderungen temporaer parken
git stash pop
# Geparkte Aenderungen wiederherstellen
git diff --staged
# Gestagete Aenderungen vor Commit pruefen
git reset --soft HEAD~1
# Letzten Commit rueckgaengig, Code behalten
git reset --hard HEAD
# ALLES zuruecksetzen (Datenverlust!)
git rebase -i HEAD~5
# Letzte 5 Commits interaktiv bearbeiten
git cherry-pick <hash>
# Einzelnen Commit uebernehmen
git bisect start
# Binaere Suche nach Bug-Commit
git reflog
# Geloeschte Commits wiederfinden
git clean -fd
# Ungetrackte Dateien loeschen
git tag -a v1.0.0 -m "Release 1.0"
# Release-Tag erstellen
```

7. Homebrew & Package Management

```
brew update && brew upgrade
# Paketlisten + alles upgraden
brew install ripgrep fd bat jq
# CLI-Tools in einem Rutsch
brew install --cask visual-studio-code
# GUI-App installieren
brew list --formulae
# Installierte CLI-Pakete auflisten
brew services start postgresql
# Hintergrunddienst starten
brew services list
# Verwaltete Dienste anzeigen
brew doctor
# Homebrew auf Probleme pruefen
brew cleanup --prune=30
# Alte Versionen entfernen (>30 Tage)
brew deps --tree node
# Abhaengigkeitsbaum anzeigen
```

8. Node.js, pnpm & Vite

```
node -v && pnpm -v
# Installierte Versionen pruefen
npm create vite@latest app -- --template react-ts
# React/TS-Projekt scaffolden
pnpm install
# Dependencies installieren
pnpm add react react-dom
# Produktions-Dependency hinzufuegen
pnpm add -D typescript @types/react
# Dev-Dependency hinzufuegen
pnpm remove lodash
# Paket entfernen
pnpm run dev
# Vite-Dev-Server starten
pnpm run build
# Produktions-Build erstellen
pnpm run preview
# Prod-Build lokal testen
pnpm dlx npm-check-updates -u
# Alle Deps auf neueste Version
npx tsc --noEmit
# TypeScript-Typecheck ohne Output
```

9. Playwright

```
npx playwright install
# Browser-Engines herunterladen
npx playwright test
# Alle Tests ausfuehren
npx playwright test tests/app.spec.ts
# Einzelne Testdatei ausfuehren
npx playwright test --headed
# Tests sichtbar im Browser
npx playwright test --project=chromium
# Nur in Chromium testen
npx playwright test --debug
# Debug-Modus mit Inspector
npx playwright codegen http://localhost:5173
# Testcode durch Aufnahme generieren
npx playwright show-report
# HTML-Testreport oeffnen
```

10. Docker — Images & Container

```
docker build -t my-app:latest --target production .
# Image bauen (Build-Stage)
docker images my-app --format "{{.Tag}}\t{{.Size}}"
# Image-Groesse checken
docker run -d -p 8080:80 --name web my-app:latest
# Container starten
docker run -it --rm my-app:latest sh
# Container interaktiv debuggen
docker ps
# Laufende Container anzeigen
docker ps -a
# Alle Container inkl. gestoppter
docker logs -f web
# Container-Logs live streamen
docker exec -it web sh
# Shell in laufendem Container
docker stop web && docker rm web
# Container stoppen + entfernen
docker system prune -af --volumes
# ALLES Ungenutzte entfernen
```

11. Docker Compose — Orchestrierung

```
docker compose up --build
# Services bauen und starten
docker compose up -d
# Im Hintergrund starten (detached)
docker compose -f *.yml -f *.prod.yml up -d
# Production-Override verwenden
docker compose down -v --remove-orphans
# Stoppen, Volumes + Waisen entfernen
docker compose restart api
# Einzelnen Service neustarten
docker compose logs -f --tail=100 api
# Service-Logs live verfolgen
docker compose exec api sh
# Shell in laufendem Service
docker compose exec db psql -U app -d myproject
# Direkt in PostgreSQL einloggen
docker compose build --no-cache
# Komplett neu bauen ohne Cache
docker compose pull
# Alle Images aktualisieren
```

12. CI/CD — GitHub Actions (Referenz)

```
jobs: quality > npm run lint + typecheck
# Code-Qualitaet pruefen
jobs: test > npm test -- --coverage
# Tests mit Coverage ausfuehren
services: postgres:16-alpine
# Test-DB als GitHub Service
docker/build-push-action@v5
# Image bauen + GHCR pushen
cache-from: type=gha
# GitHub Actions Build-Cache nutzen
concurrency: cancel-in-progress: true
# Parallele Runs auf Branch canceln
```

13. Local AI & LLM Workflow (LM Studio)

```
lms server start
# Lokalen Inferenz-Server starten
lms status
# Geladene Modelle + Server-Status
lms ls
# Heruntergeladene Modelle auflisten
lms load mistral-7b-instruct
# Modell in den Server laden
curl http://localhost:1234/v1/models
# OpenAI-kompatible API testen
curl -s localhost:1234/v1/chat/completions ...
# Chat-Completion testen
python3 -m venv venv && source venv/bin/activate
# Python-Environment erstellen
pip install -r requirements.txt
# Python-Dependencies installieren
tail -f ~/.cache/lm-studio/logs/lmstudio.log
# LM Studio Logs live verfolgen
```

Environment-Strategie

Scope	Datei	Git?	Beschreibung
Defaults	.env.example	Ja	Template mit Platzhaltern
Lokal	.env	Nein	Lokale Overrides
CI/CD	GitHub Secrets	—	Sensitive Werte + Build-Vars
Runtime	Docker Env	—	Injected via Compose

14. Power-User — Pipes, Aliases & .zshrc

```
history | grep "docker" | tail -20
# Letzte 20 Docker-Befehle aus History
ps aux | grep node | awk '{print $2}'
# Nur PIDs von Node-Prozessen
find . -name "*.log" -mtime +7 | xargs rm
# Alte Logs loeschen (>7 Tage)
curl -s https://api.example.com | jq '.data[]'
# JSON-API mit jq parsen
docker compose logs api 2>&1 | grep "ERROR"
# Fehler in Docker-Logs filtern

# -- Aliases (in ~/.zshrc) --
alias ll="ls -lah"
alias gs="git status"
alias dev="pnpm run dev"
alias ports="lsof -i -P -n | grep LISTEN"
alias killnode="pkill -9 node"
alias dc="docker compose"
alias dcu="docker compose up --build"
alias update="brew update && brew upgrade && brew cleanup"

# -- Exports (in ~/.zshrc) --
export EDITOR="code --wait"
export PATH="$HOME/.local/bin:$PATH"
export PNPM_HOME="$HOME/Library/pnpm"

# -- Funktionen (in ~/.zshrc) --
function killport() { kill -9 $(lsof -ti :$1); }
# Port freigeben
function project() { cd ~/Projects/$1 && code .; }
# Projekt oeffnen
function dsh() { docker compose exec $1 sh; }
# Docker Shell oeffnen
```

```
source ~/.zshrc
# Shell-Config neu laden
sudo !!
# Letzten Befehl mit sudo wiederholen
```