

cs-scripts

Handbuch

Build- und Hilfsskripte für die napp-it CS Tools - OmniOS/illumos- und Solaris-Builds, Ollama-Library-Parser, ZFS-Pool-Benchmark

Version 1.0 · 13. Oktober 2026

Inhalt

1. Überblick	3
2. illumos / OmniOS Builds	3
3. Solaris Builds	3
4. ZFS-Pool-Benchmark	3
5. Ollama-Library-Parser	4
5.1 Optionen	4
5.2 Ausgabesatz	4
6. Lizenz	5

1. Überblick

cs-scripts sammelt die Build- und Hilfsskripte der napp-it-CS-Projektfamilie: die OmniOS/illumos- und Solaris-Build-Skripte für die Rust- und Go-cs-tools, den eigenständigen Ollama-Library-Parser der napp-it-CS-Download-Seite und den eigenständigen ZFS-Pool-Benchmark.

Aufbau des Repositories:

```
illumos/
  rustfs_omnios_1.0_release.sh   RustFS-Build auf OmniOS/illumos
  cs-imageindex_omnios_1a.sh    cs-imageindex-Build auf OmniOS/illumos
  build_llamacpp_omnios.sh      llama.cpp llama-server auf OmniOS
  build.rc.sh                   cargo install rustfs-cli + ipadm-Pufferkorrektur
  needed_ip_modification_for_rustfs.txt
  build_restic_omnios.txt       manuelle restic-Build-Notizen (Go)
solaris/
  build_rclone_solaris.sh       rclone aus dem Quellcode auf Solaris 11.4
  build_restic_solaris.sh       restic aus dem Quellcode auf Solaris 11.4
ollama/
  ollama-library.pl             Ollama-Library-Katalogparser
benchmark/
  benchmark_worker.pl           eigenständiger ZFS-Pool-Benchmark
```

2. illumos / OmniOS Builds

Für illumos gibt es keinen GitHub-Actions-Runner. Die illumos-Binaries werden daher mit diesen Skripten von Hand auf echter OmniOS-Hardware gebaut und in das passende GitHub-Release hochgeladen.

Skript	Zweck
rustfs_omnios_1.0_release.sh	kanonischer RustFS-Build (am 2026-09-20 von rustfs_omnios_1a.sh umbenannt; die Historie der 2a5..2a12-Linie steht im Skript-Header)
cs-imageindex_omnios_1a.sh	cs-imageindex-Build inklusive mitgeliefertem statischem LGPL-ffmpeg; das Ergebnis ist eigenständig
build_llamacpp_omnios.sh	llama.cpp llama-server als lokales OpenAI-kompatibles LLM; 10 illumos-Patches, auf r151058 verifiziert
build.rc.sh	cargo install rustfs-cli plus ipadm-Pufferkorrektur
build_restic_omnios.txt	manuelle restic-Build-Notizen (Go)

Die Dateien illumos/readme.txt und needed_ip_modification_for_rustfs.txt beschreiben Details und die Netzwerkpuffer-Anpassung, die RustFS benötigt.

3. Solaris Builds

build_rclone_solaris.sh und build_restic_solaris.sh bauen rclone und restic aus dem Quellcode auf Solaris 11.4. Als normaler Benutzer auf einem Host mit installierter Go-Toolchain ausführen.

4. ZFS-Pool-Benchmark

benchmark/benchmark_worker.pl ist ein eigenständiger Pool-Benchmark zum Vergleichen von Pools ("was liefert genau DIESER Pool wirklich?") auf jedem ZFS-Host: TrueNAS CORE/SCALE, FreeBSD, illumos/Solaris, OpenZFS unter Linux und Windows. Er ist reines Perl und braucht nur perl, zfs/zpool, df und optional smartctl - kein fio. Das Testmedium ist ein Scratch-Dataset, das auf dem getesteten Pool angelegt und am Ende wieder gelöscht wird.

Es ist dieselbe Datei (gleicher Name, gleicher Inhalt) wie der napp-it-Worker in data/menues/_lib/scripts/bench/benchmark_worker.pl.

```
perl benchmark/benchmark_worker.pl profile=quick pool=tank
perl benchmark/benchmark_worker.pl profile=mailserver syncwrite=yes load=balanced
```

```
perl benchmark/benchmark_worker.pl check=yes pool=tank
perl benchmark/benchmark_worker.pl help=yes
```

- profile=quick dauert etwa 1-2 Minuten.
- check=yes ist ein Trockenlauf ohne jede I/O.
- Die Werte sind per Design cache-sicher: primarycache=metadata (Daten nicht im ARC, Metadaten weiter gecacht), sync=always für die Sync-Write-Phase, ein paralleles zpool-iostat -v-Sample je Phase und recordsize=4K für die 4k-Testdatei.
- Jedes Ergebnis ist als storage-bound, cache, cache-influenced oder tool-limited gekennzeichnet.
- Das Log endet mit einer BENCH_DONE-Marke.

Details und die gemessenen Einschränkungen stehen in benchmark/readme.txt.

5. Ollama-Library-Parser

ollama/ollama-library.pl liest die öffentliche Ollama-Modellbibliothek (ollama.com/library) per curl und liefert bei Bedarf den vollständigen Modell- und Tag-Katalog: Modellnamen, Download-Größe je Tag, Kontextfenster, Eingabefähigkeiten (Text/Bild) und Quantisierung. Eine GUI kann damit eine gefilterte Modellauswahl für ollama pull anbieten.

Das Skript wird bei Bedarf aus dem Repository geladen (raw.githubusercontent) und lokal gecacht. Der Schnittstellenvertrag ist im Skript-Header festgeschrieben (CONTRACT cs-ollama-catalog-v1); so bleibt die aufrufende GUI stabil, auch wenn Ollama das Markup seiner Webseite ändert - nur die Abruf-/Parse-Internia unterhalb des Vertrags-Headers dürfen sich ändern.

```
perl ollama/ollama-library.pl --list --json
perl ollama/ollama-library.pl --json --vision --param-max 14 --size-max 10
perl ollama/ollama-library.pl --json --vision --newer 12
perl ollama/ollama-library.pl --version
```

5.1 Optionen

Option	Bedeutung
--list	Modelle auflisten (Name, Fähigkeiten, Parametergrößen)
--tags [--model NAME]	Katalog je Tag (alle Modelle oder ein Modell)
--json	maschinenlesbares JSON (Standard; --tsv für Text)
--vision	nur Modelle mit Bildeingabe
--newer MONTHS	nur Modelle, die innerhalb von MONTHS aktualisiert wurden (ungefähr)
--param-max B	nur Modelle mit höchstens B Milliarden Parametern
--size-max GB	nur Tags mit höchstens GB Download-Größe
--cache SECS	Cache-Lebensdauer des Katalogs (Standard 2592000 = 30 Tage)
--refresh	Katalog neu aufbauen, Cache ignorieren
--version	Vertragsversion ausgeben

5.2 Ausgabesatz

Ein Satz pro Tag:

```
{ "model": "llama3.2-vision",
  "tag": "llama3.2-vision:11b-instruct-q4_K_M",
  "size_gb": 7.8, "context": 128000,
  "input": ["text","image"], "quant": "q4_K_M",
  "param_b": 11, "age_days": 300 }
```

6. Lizenz

BSD 2-Clause. Copyright (c) 2026 Guenther Alka / napp-it.org. Projekt: napp-it / csweb-gui, gepflegt von gea-napp-it, KI-unterstützt durch Claude (Anthropic).