

nanocode

ncfc

Manual for the nanocode file compressor and archiver

Contents

1	Copyright	3
2	List of files	3
3	Overview	4
3.1	Purpose and Advantages	4
3.2	Key Safety Features	4
4	System Requirements	5
5	Support Contact	5
6	Command-Line Options Reference	6
6.1	Basic Usage	6
6.2	The <code>add</code> Subcommand	6
6.3	The <code>get</code> Subcommand	7
6.4	The <code>list</code> Subcommand	8
6.5	The <code>daemon</code> Subcommand	8
6.6	Backends	8
7	File Filtering (Smart Glob)	9
7.1	Pattern Syntax and Translation	9
7.2	Examples	9
8	Data Security and the Daemon	10
8.1	Daemon Lifecycle	10
8.2	Using Encryption	10
9	Recipes	11
9.1	High-Throughput Data Ingestion	11
9.2	Secure Cloud Archival (S3 Cold Storage)	11
9.3	Efficient Cloud Retrieval	12
9.4	Safe and Robust Backups	12
9.5	Advanced Filtering with Regular Expressions	12

1 Copyright

Copyright ©, 2026 by nanocode GmbH, Germany.

This software is provided under a license and may be used and copied only in accordance with the terms and conditions of that license, and only with the inclusion of the above copyright notice. This software, or any copies thereof, may not be provided or otherwise made available to any third party. No title to or ownership of the software is transferred by this agreement. The information contained in this software is subject to change without notice and does not constitute a commitment by the copyright holder.

2 List of files

The package comes with the following files:

- `ncfc` : compiled executable for Linux
- `ncfc_manual.pdf` : this manual

3 Overview

`ncfc` (nanocode file compressor and archiver) is a high-performance synchronization and archival tool with on-the-fly compression and encryption. It automatically selects the optimal compression scheme for each file by examining file content. It is specifically designed for efficiently managing large-scale data, such as raw camera streams and sensor data from ADAS development vehicles, across local and cloud storage.

3.1 Purpose and Advantages

The primary purpose of `ncfc` is to efficiently connect high-bandwidth data acquisition systems and cost-effective cloud storage. Traditional compression tools like `tar` or `zip` are often too slow for multi-terabyte datasets or fail to exploit the specific structure of binary sensor data.

Key Advantages:

- **Domain-Specific Compression:** Unlike general-purpose tools, `ncfc` includes specialized algorithms optimized for multi-channel 8/10/12/16-bit and packed raw data.
- **Massive Parallelism:** The entire pipeline—from file scanning and compression to encryption and cloud upload—is fully multi-threaded, maximizing throughput on modern multi-core systems.
- **On-the-fly Processing:** Data are compressed and encrypted in memory before being sent to the cloud, eliminating the need for large temporary local storage.
- **Authenticated Encryption:** Security is built-in using a high-performance, secure Authenticated Encryption with Associated Data (AEAD) algorithm (XChaCha20-Poly1305), ensuring both confidentiality and data integrity without significant performance overhead.
- **Storage Efficiency:** Typically achieves 30-60% reduction in storage footprint for raw data, significantly lowering cloud storage costs.
- **Cost Optimization:** Most cloud providers charge per request (PUT/GET). Unlike traditional backups that might store millions of small files as individual objects, `ncfc` bundles data into large 1 GB parts. This significantly reduces the number of objects in cloud storage buckets, making metadata listing faster and retrieval much cheaper, as only a few large requests are needed instead of thousands of small ones.

3.2 Key Safety Features

`ncfc` is designed to be a robust backup solution for large-scale data:

- **Automatic Format Detection:** There is no need to specify what kind of data is being archived. `ncfc` analyzes the bit-patterns and automatically chooses the most efficient compressor for every 64 MB segment.
- **Fast Metadata Queries:** The archive index is stored separately from the data. This allows `ncfc ls` to answer queries about millions of files instantly without having to touch the actual multi-terabyte data blocks.

- **Versioning and Append-Only Logic:** Files in `ncfc` are versioned. When a file is “deleted” or updated, a new version is created, but the original data remains safely in the archive parts (unless specified otherwise).
- **Interruption Resilience:** If the internet connection drops or the program is killed (e.g., `kill -9` or a power failure), the archive remains consistent:
 - `ncfc` syncs the index to the backend every time a 1 GB part is completed.
 - If a crash occurs during uploading part 5 (the 5th 1GB data block), the index still correctly points to parts 0-4.
 - When the `add` command is run again, `ncfc` detects the incomplete part 5, removes it, and resumes the upload exactly where it left off.

4 System Requirements

- Linux operating system (64-bit)
- Processor supporting AVX2 instruction set (required for high-performance compression)
- 200 MB of RAM per worker thread

5 Support Contact

E-Mail: support@nanocode.com

Phone: +49 6226/7876840

6 Command-Line Options Reference

ncfc supports automatic file versioning: when a file is added to an archive multiple times, each version is retained and can be accessed individually (unless deletion is explicitly configured). Encryption is available using XChaCha20-Poly1305 AEAD and requires a master key stored in a file (which itself can be passphrase-protected). The encryption daemon manages keys in memory to avoid repeated passphrase entry.

6.1 Basic Usage

The ncfc tool follows a subcommand-based interface:

```
# Add files to an archive
ncfc <archive_url> add [options]

# List files in an archive
ncfc <archive_url> ls [options]

# Download/Extract files from an archive
ncfc <archive_url> get [options]

# Manage the encryption daemon
ncfc daemon [--stop|--status]
```

6.2 The add Subcommand

Used to upload files into a new or existing archive. If the archive does not exist, it is created automatically. The full directory tree starting from the base directory (or matched by include patterns) is archived.

Options:

- `--base-directory <dir>`, `-b <dir>`: The root directory for the local scan.
- `--compression-level <0-2>`, `-cl <0-2>`: Compression level to use:
 - 0: No compression, maximum speed.
 - 1: Fast compression, recommended for real-time offloading.
 - 2: Best compression, recommended for long-term storage (default).
- `--encrypt`, `-x`: Enable XChaCha20-Poly1305 encryption for all new files.
- `--endpoint <url>`, `-E <url>`: Custom S3-compatible endpoint URL (e.g., `https://minio.example.com`). Use this to connect to MinIO, Ceph, or other S3-compatible storage systems instead of AWS.
- `--exclude <pattern>`, `-e <pattern>`: Pattern for files to skip (see Section 7).
- `--format <types>`, `-f <types>`: Skip automatic format detection and use the specified data format. Types can be a comma-separated list of data types (u8, i8, u16, i16, u32, i32, u64, i64, float16, float32, float64) or a packed format name (mipicsi2bayer12bit, mipicsi2bayer10bit). Example: `--format u16,u16` for 2-channel 16-bit data. This saves significant CPU time when the data format is known in advance.

- `--include <pattern>, -i <pattern>`: Pattern for files to include (see Section 7). Note: The order of `-i` and `-e` matters: later patterns override earlier ones, allowing selective inclusion and exclusion.
- `--keyfile <path>, -k <path>`: Path to a master key file (32 bytes). If not provided, `ncfc` checks the daemon or `~/.ncfc/key`.
- `--no-verify-ssl`: Disable SSL/TLS certificate verification for cloud backends.
- `--pretend, -P`: Simulate the operation but do not write anything to the archive.
- `--profile <name>, -p <name>`: Cloud provider profile name (e.g., AWS profile).
- `--quiet, -q`: Suppress file list output; only errors are shown.
- `--region <name>, -r <name>`: Cloud region (e.g., eu-central-1).
- `--summary, -s`: Show a summary of the transfer after completion.
- `--tag <text>, -T <text>`: Metadata tag (max 15 chars) stored with each file in this operation.
- `--threads <n>, -t <n>`: Number of parallel worker threads for compression, encryption, and I/O (default: 16); each thread requires 130 MB of RAM.
- `--width <n>, -w <n>`: Image row width in samples. Skips automatic width detection for all channels. Use together with `--format` when the image dimensions are known (e.g., `--format u16,u16 --width 3856`).

6.3 The `get` Subcommand

Used to extract files from an archive.

- `--endpoint <url>, -E <url>`: Custom S3-compatible endpoint URL (e.g., `https://minio.example.com`).
- `--exclude <pattern>, -e <pattern>`: Pattern for files to skip.
- `--include <pattern>, -i <pattern>`: Pattern for files to include.
- `--keyfile <path>, -k <path>`: Path to a master key file (32 bytes). If not provided, `ncfc` checks the daemon or `~/.ncfc/key`.
- `--no-verify-ssl`: Disable SSL/TLS certificate verification for cloud backends.
- `--output-directory <dir>, -o <dir>`: Local path where files will be restored.
- `--overwrite-always`: Overwrite existing files on disk.
- `--overwrite-if-newer, -u`: Overwrite existing files on disk if extracted files are newer.
- `--pipe`: Write file content directly to `stdout`. Multiple files are concatenated.
- `--pretend, -P`: Simulate the operation but do not write anything to disk.
- `--profile <name>, -p <name>`: Cloud provider profile name (e.g., AWS profile).
- `--quiet, -q`: Suppress file list output; only errors are shown.
- `--region <name>, -r <name>`: Cloud region (e.g., eu-central-1).
- `--summary, -s`: Show a summary of the transfer after completion.
- `--tag <text>, -T <text>`: Get only files with this tag.
- `--threads <n>, -t <n>`: Number of parallel worker threads for decompression, decryption, and I/O (default: 16); each thread requires 130 MB of RAM.
- `--version <n>, -v <n>`: Extract a specific version of a file. Version numbers are assigned automatically by `ncfc` starting from 1 (default: latest).

6.4 The list Subcommand

- `--cap, -c`: Truncate long filenames to fit the terminal width.
- `--endpoint <url>, -E <url>`: Custom S3-compatible endpoint URL (e.g., `https://minio.example.com`).
- `--exclude <pattern>, -e <pattern>`: Pattern for files to skip.
- `--include <pattern>, -i <pattern>`: Pattern for files to include.
- `--json, -j`: Output file list in JSON format for scripting.
- `--keyfile <path>, -k <path>`: Path to a master key file (32 bytes). If not provided, `ncfc` checks the daemon or `~/.ncfc/key`.
- `--long, -l [cols]`: Detailed listing. `cols` can be a comma-separated list: `perms,path,size,ratio,mtime,version,tag,encrypted`.
- `--no-verify-ssl`: Disable SSL/TLS certificate verification for cloud backends.
- `--profile <name>, -p <name>`: Cloud provider profile name (e.g., AWS profile).
- `--region <name>, -r <name>`: Cloud region (e.g., `eu-central-1`).
- `--segments, -S`: List all archive chunks (segments) for each file. When used with `--long`, it shows detailed chunk information (part file, offset).
- `--summary, -s`: Show aggregate statistics for the entire archive.
- `--tag <text>, -T <text>`: List only files with this tag.
- `--version <n>, -v <n>`: List a specific version of a file.

6.5 The daemon Subcommand

Manages the background key-caching process.

- `ncfc daemon [start]`: Starts the daemon. Will prompt for a passphrase if the key is encrypted.
- `ncfc daemon --stop`: Signals the running daemon to exit and wipe the key from memory.
- `ncfc daemon --status`: Shows if the daemon is running and whether it is locked (key not available, passphrase required) or unlocked (key loaded in memory and available for use).

6.6 Backends

`ncfc` supports the following storage backends (storage destinations where archives can be created and accessed):

- **Local File System**: `path/to/archive.ncfc`
- **AWS S3**: `s3://bucket-name/prefix/archive.ncfc`
- **Azure Blob**: `az://container/prefix/archive.ncfc` (on request)
- **Google Cloud**: `gs://bucket/prefix/archive.ncfc` (on request)
- **SFTP**: `sftp://user@host:port/path/archive.ncfc` (on request)

The backend is automatically detected from the archive URL.

7 File Filtering (Smart Glob)

The `--include (-i)` and `--exclude (-e)` options use a “Smart Glob” system—a custom pattern matching system specifically designed for `ncfc`. Under the hood, these patterns are converted to powerful Regular Expressions.

7.1 Pattern Syntax and Translation

Unlike standard shell globs, `ncfc` patterns are designed for deep directory structures:

- **Literal Dot:** `.` is always treated as a literal dot (internally `\.`).
- **Single Level Wildcard:** `*` matches any character *except* a forward slash. It is used to match files or directories within a single level.
- **Recursive Wildcard:** `**` matches any character *including* forward slashes. This is the primary way to perform recursive searches.
- **Alternation:** `{option1,option2,option3}` allows matching multiple alternatives (e.g., `*.{raw,log,txt}` matches files ending in `.raw`, `.log`, or `.txt`).
- **Root Marker:** `//` is a special marker. The part of the path *before* `//` is used for scanning on disk but is **removed** from the path stored inside the archive. Note: The double slash in the backend URL (e.g. `s3://`) is parsed separately.

7.2 Examples

Using absolute paths:

```
# Archive the entire /data/recordings folder
# Files are stored as: data/recordings/file1.raw, ...
ncfc archive.ncfc add -i /data/recordings
```

This archives all files and subdirectories within `/data/recordings`.

```
# With wildcards - archive all .raw files recursively
# Files are stored as: data/recordings/subdir/file.raw, ...
ncfc archive.ncfc add -i '/data/recordings/**/*.raw'
```

This recursively finds and archives only `.raw` files within `/data/recordings` and all subdirectories.

Stripping path prefix with double-slash:

```
# Files are stored starting from 'src/...'
ncfc <url> add -i '/home/user/projects/myproj//src/**'
```

This strips `/home/user/projects/myproj/` from the stored paths, so files are stored as `src/...` in the archive.

```
# Store recordings with only the 'recordings' folder
# Files are stored as: recordings/cam1/file.raw, ...
ncfc archive.ncfc add -i '/data//recordings'
```

This strips `/data/` from the paths, storing files as `recordings/cam1/file.raw` instead of `data/recordings/cam1/file.raw`.

Including multiple file types:

```
# Uses regex alternation
ncfc <url> add -i '**/*.{raw,log,txt}'
```

This recursively matches files with extensions .raw, .log, or .txt using the alternation syntax.

Complex exclusion:

```
# Include all data but skip 'tmp' folders and hidden files
ncfc <url> add -i 'data/**' -e '**/tmp/**' -e '**/*.*' 
```

This includes all files under data/ but excludes any files in tmp directories and any hidden files (starting with a dot).

8 Data Security and the Daemon

ncfc provides strong encryption using XChaCha20-Poly1305. To manage encryption keys securely without repeated passphrase entry, ncfc uses a background daemon.

8.1 Daemon Lifecycle

The daemon holds the master key in memory and provides it to the ncfc client upon request.

```
# Start the daemon (prompts for passphrase if key is encrypted)
ncfc daemon
```

```
# Stop the daemon
ncfc daemon --stop
```

```
# Check daemon status
ncfc daemon --status
```

When ncfc daemon is run, it checks for a key (plaintext) or key.enc (encrypted) in the ~/.ncfc/ directory. If neither exists, it offers to generate a new master key.

8.2 Using Encryption

To enable encryption for an archive, simply use the -x or --encrypt flag during the add operation.

```
ncfc <url> add -i 'data/**' -x
```

Note: The daemon must be running and “unlocked” (successfully started with the correct passphrase) for encryption and decryption to work.

9 Recipes

This chapter provides practical examples for common workflows and advanced use cases of ncfc.

9.1 High-Throughput Data Ingestion

Scenario: Data has been recorded on an external SSD and needs to be transferred to a NAS for quick analysis multiple times.

To maximize throughput and ensure data can be identified by the recording campaign:

```
# Ingest data with fast compression and a campaign tag
ncfc /mnt/nas/analysis.ncfc add -i /mnt/ssd/data/ -cl 1 -T 'campaign_A'
```

The fastest compression level (-cl 1) ensures that the CPU does not become a bottleneck while offloading from the SSD. The tag campaign_A allows you to manage multiple ingestion rounds in the same archive.

To list only files from a specific campaign:

```
ncfc /mnt/nas/analysis.ncfc ls --tag 'campaign_A'
```

To extract only the data from a specific campaign:

```
ncfc /mnt/nas/analysis.ncfc get --tag 'campaign_A' -o ./campaign_data/
```

To extract files for analysis and pipe them directly into a processing tool:

```
ncfc /mnt/nas/analysis.ncfc get --tag 'campaign_A' -i '**/cam0/*.raw' --pipe |
analyze_raw --stdin
```

This extracts only files from camera 0 with tag “campaign_A”, writes them to stdout (via --pipe), and pipes them directly into the analyze_raw tool for processing without creating temporary files.

9.2 Secure Cloud Archival (S3 Cold Storage)

Scenario: Datasets need to be uploaded to AWS S3 Glacier or Deep Archive for long-term storage, requiring maximum compression and high security.

```
# Start and unlock the encryption daemon first
ncfc daemon
```

```
# Upload with high compression and encryption
ncfc s3://my-bucket/archive.ncfc add -i ./local_data/ -cl 2 -x --profile my-aws
```

The --profile my-aws option specifies which AWS credentials profile to use (as configured in ~/.aws/credentials), allowing access to the S3 bucket.

Encryption Details:

- ncfc uses XChaCha20-Poly1305, a state-of-the-art AEAD cipher.
- Data is encrypted **before** it leaves the local machine.

- **Security Warning:** The master key (usually in `~/.ncfc/key` or managed by the daemon) is the only way to recover the data. If the key or the passphrase to an encrypted key is lost, the data on S3 is **permanently irrecoverable**. Keep backups of the master key in a safe, offline location (e.g., a hardware security module or a physical safe).

9.3 Efficient Cloud Retrieval

Scenario: A specific subset of data needs to be downloaded from a large S3 archive.

```
# Restore only the logs from a specific month
ncfc s3://my-bucket/archive.ncfc get -i 'logs/2025/jan/**' -o ./restored_logs/
```

This retrieves only the specified subset of files from the archive, leveraging `ncfc`'s efficient storage structure (see Section 3.1).

9.4 Safe and Robust Backups

`ncfc` is designed to be a robust backup solution for large-scale data (see Section 3 for key safety features).

Verifying Archive Integrity:

```
# Verify archive integrity and see all versions
ncfc s3://my-bucket/archive.ncfc ls -l version --summary
```

This command displays file version information for all files in the archive and provides summary statistics about the archive's contents and integrity.

9.5 Advanced Filtering with Regular Expressions

The “Smart Glob” system makes complex file selection easy by combining standard wildcards with path-aware truncation.

Example: Selective ingestion with path stripping

```
# 1. Use ** for recursive matching
# 2. Use // to strip the leading project path from the archive
# 3. Use -e to skip bulky log files
ncfc backup.ncfc add \
  -i '/home/user/projects/my_app//src/**' \
  -i '/home/user/projects/my_app//docs/**/*.*md' \
  -e '**/tests/data/**' \
  -e '**/*.*log'
```

How it works:

- The `//` marker tells `ncfc` to start the path inside the archive **after** that point. In the example above, files will be stored as `src/main.cpp` instead of `home/user/projects/my_app/src/main.cpp`.
- The `**` ensures that all subdirectories are traversed.

- The order of `-i` and `-e` matters: later patterns override earlier ones, allowing selective inclusion with specific exclusions (e.g., “include everything but exclude these specific parts”).