**EvidenceRescue — Complete CLI command reference (prototype)**

Below are **every command** in the current EvidenceRescue prototype (the CLI you have), plus a few highly recommended extra commands you may add later (marked). For each command I show:

* **syntax**
* **purpose / short description**
* **important options**
* **example usage**
* **suggested next steps** you’d usually run after it

Use these from the project directory (where evidence\_rescue.py lives). If you installed into a venv, activate it first (source venv/bin/activate) and run python evidence\_rescue.py <command> ....

**General usage**

python3 evidence\_rescue.py <command> [options] <input>

Run python3 evidence\_rescue.py --help for a brief summary.

**1) analyze**

**Syntax**

python3 evidence\_rescue.py analyze <input> [--output-dir OUTDIR] [--detailed]

**What it does**  
Quickly inspects a raw file or image to detect partition table hints (MBR/GPT), read the first bytes, and produce a small JSON summary (entropy sample, MBR signature). Good first step.

**Important options**

* --output-dir, -o — where analysis artifacts are written (default out\_scan)
* --detailed — run slower deeper checks (if implemented)

**Example**

python3 evidence\_rescue.py analyze /path/to/disk.img --output-dir /tmp/scan1

**Suggested next steps**

* If it detects filesystem/partitions → parse-fs.
* If not → entropy-scan or carve.

**2) parse-fs *(requires pytsk3 to do real work; otherwise creates a stub map)***

**Syntax**

python3 evidence\_rescue.py parse-fs <image> [--partition N] [--output-dir OUTDIR] [--deleted]

**What it does**  
Parses a disk image’s filesystem (NTFS/FAT/ext, etc.) and writes a map.json (file map). If pytsk3 not present the prototype writes a stub map.

**Important options**

* --partition — partition index to parse (0 = auto)
* --deleted — include deleted entries (where recoverable)
* --output-dir — output folder for map.json (default out\_fs)

**Example**

python3 evidence\_rescue.py parse-fs /tmp/kali\_disk.raw --partition 1 --output-dir /tmp/fsmap

**Suggested next steps**

* list-map (not implemented in prototype — see “extra commands” below) to search the map
* extract (by path or inode) to pull specific files

**3) carve**

**Syntax**

python3 evidence\_rescue.py carve <input> [--output-dir OUTDIR] [--types TYPE,TYPE] [--chunk-size BYTES] [--overlap BYTES] [--max-carve BYTES] [--signatures PATH]

**What it does**  
Signature-based streaming carver. Scans the input for headers/footers defined in signatures.json (or custom file) and writes carved files + index.json with metadata (offset, size, sha256, entropy).

**Important options**

* --output-dir, -o — where carved files and index.json go (default out\_carved)
* --types — comma-separated subset of signature ids/exts to limit scan (e.g. jpg,pdf)
* --chunk-size — read chunk size (default 1 MiB)
* --overlap — overlap between chunk reads (default 512 bytes)
* --max-carve — maximum bytes to write per carved item (default 100 MiB)
* --signatures — path to a custom signatures JSON

**Example**

python3 evidence\_rescue.py carve disk.img --output-dir /tmp/carved --types jpg,pdf --chunk-size 65536 --overlap 256

**Suggested next steps**

* list-carved to inspect index.json
* detect-encrypted to triage high-entropy items
* identify-carved (recommended extra) — run python-magic/TrID on carved files

**4) list-carved**

**Syntax**

python3 evidence\_rescue.py list-carved <carved/index.json>

**What it does**  
Prints a human list of carved items from the index.json created by carve (ID, type, path, offset, size, entropy).

**Example**

python3 evidence\_rescue.py list-carved /tmp/carved/index.json

**Suggested next steps**

* extract-carved to copy selected carved items out for review
* detect-encrypted to flag likely encrypted/compressed items

**5) extract-carved**

**Syntax**

python3 evidence\_rescue.py extract-carved <carved/index.json> [--id ID] [--type TYPE] [--limit N] [--output-dir OUTDIR]

**What it does**  
Extracts carved files from the carve output by carved id (e.g., carve\_000001) or by type (e.g., jpg) into OUTDIR.

**Important options**

* --id — exact carved id to extract
* --type — filetype (signature id or ext) to extract in bulk
* --limit — max number of items to extract
* --output-dir — destination directory

**Example**

python3 evidence\_rescue.py extract-carved /tmp/carved/index.json --type jpg --limit 50 --output-dir /tmp/extracted\_jpgs

**Suggested next steps**

* run validators (Pillow/PyPDF2) on extracted files
* report to include selected items in a case report

**6) entropy-scan**

**Syntax**

python3 evidence\_rescue.py entropy-scan <input> [--window BYTES] [--step BYTES] [--output-dir OUTDIR]

**What it does**  
Scans the input with sliding windows and computes Shannon entropy per window; writes entropy.json (offset, size, entropy) and helps identify high-entropy (likely compressed/encrypted) regions.

**Important options**

* --window — window size in bytes (default 1 MiB)
* --step — step size (default 512 KiB)
* --output-dir — where to place entropy.json (default out\_entropy)

**Example**

python3 evidence\_rescue.py entropy-scan disk.img --window 1048576 --step 524288 --output-dir /tmp/entropy

**Suggested next steps**

* carve targeted high-priority offset ranges
* detect-encrypted on carved index

**7) detect-encrypted**

**Syntax**

python3 evidence\_rescue.py detect-encrypted <file\_or\_index> [--threshold FLOAT]

**What it does**  
Runs simple entropy heuristics (and small sample compression tests) to classify files/entries as likely\_plain, likely\_compressed, or likely\_encrypted. If passed a carved index.json, it iterates entries and writes <index>.encrypted\_report.json.

**Important options**

* --threshold — entropy cutoff for likely\_encrypted (default 7.9)

**Example**

python3 evidence\_rescue.py detect-encrypted /tmp/carved/index.json --threshold 7.8

python3 evidence\_rescue.py detect-encrypted /tmp/extracted/file.bin

**Suggested next steps**

* escalate encrypted items (legal/operational)
* try targeted repair for partially recoverable files

**8) report**

**Syntax**

python3 evidence\_rescue.py report <index\_or\_map.json> [--format json|csv] [--output PATH]

**What it does**  
Generates a case-ready JSON or CSV report from carved index or file map. Outputs summaries and per-file metadata.

**Important options**

* --format — json or csv (default json)
* --output — base output path (without extension)

**Example**

python3 evidence\_rescue.py report /tmp/carved/index.json --format json --output /tmp/case123\_report

**Suggested next steps**

* include report in export bundle (manual zip in prototype)
* sign/copy to evidence vault

**9) make-test-blob**

**Syntax**

python3 evidence\_rescue.py make-test-blob [--output PATH]

**What it does**  
Creates a small synthetic binary blob containing a fake JPEG, PDF and ZIP headers mixed with noise — useful for testing the carver.

**Example**

python3 evidence\_rescue.py make-test-blob --output /tmp/test\_blob.bin

**Suggested next steps**

* analyze and carve the generated blob to validate the tool

**10) version**

**Syntax**

python3 evidence\_rescue.py version

**What it does**  
Prints the tool version string (prototype v0.1).

**Quick common workflows (copy-paste)**

**Filesystem present**

python3 evidence\_rescue.py analyze disk.img --output-dir out\_scan

python3 evidence\_rescue.py parse-fs disk.img --partition 1 --output-dir out\_fs

# (then search map.json and extract)

**No filesystem / carving fallback**

python3 evidence\_rescue.py analyze disk.img --output-dir out\_scan

python3 evidence\_rescue.py entropy-scan disk.img --output-dir out\_entropy

python3 evidence\_rescue.py carve disk.img --output-dir out\_carved --chunk-size 65536 --overlap 256

python3 evidence\_rescue.py list-carved out\_carved/index.json

python3 evidence\_rescue.py extract-carved out\_carved/index.json --type jpg --limit 50 --output-dir out\_extracted

python3 evidence\_rescue.py detect-encrypted out\_carved/index.json --threshold 7.9

python3 evidence\_rescue.py report out\_carved/index.json --format json --output out\_report

**Notes & tips**

* The prototype uses a **simple JSON signatures file** (signatures.json). You can extend it with more headers/footers and min\_size thresholds.
* Some features (real parse-fs and advanced identification) require optional packages: pytsk3, python-magic, Pillow, PyPDF2, trid CLI, etc. If not installed the tool falls back to simple behavior or stub outputs.
* Use --chunk-size and --overlap to tune performance for very large images — smaller chunk sizes and overlaps catch boundaries at the cost of speed.
* Always work on **copies** of evidence; EvidenceRescue never modifies input, but good forensic practice is to keep originals safe.

