Apache Cassandra commands

give all detailed exaplanination Apache cassandra all complete shell commands

Nice — I’ll give you a compact **but full** cheat-sheet: the frequently-used Apache Cassandra shell tools (`cqlsh`, `nodetool`, `sstableloader`, `cassandra-stress`, service commands, and common admin flows), what each command does, exact shell syntax/examples, and short notes/best-practices where needed. I’ll call out the most important references as I go.

Quick orientation — tools you’ll use

- **cqlsh** — the CQL interactive shell (create keyspaces/tables, read/write, run CQL shell commands like `DESCRIBE`, `COPY`, `CONSISTENCY`, `TRACING`).

- **nodetool** — node / cluster admin (status, repair, snapshot, cleanup, decommission, compaction stats, netstats, etc.). Run from any node (or specify host).
- **sstableloader / nodetool import** — bulk load SSTables to a cluster. Use when restoring or moving SSTables.
- **cassandra-stress** — bundled stress/load generator (note: Apache docs say it’s deprecated in favor of NoSQLBench). Use for quick benchmarking.
- **systemctl / service / cassandra binary** — how to start/stop Cassandra (packaged vs tarball installs).

---

1) Install / start / stop / check service

(Examples for packaged installs; tarball differs — use `cassandra -f` to run foreground.)

Install (follow official install docs for your OS / version — Debian/RPM/tarball). Example high-level steps (Ubuntu/Debian):

```bash
```

```
# ensure Java is installed (Cassandra usually needs Java 8/11 depending on version)
sudo apt update
sudo apt install openjdk-11-jdk -y

# follow the official repo instructions (see Apache docs) and then:
sudo apt update
sudo apt install cassandra -y
```

Start / stop / restart / status:

```bash
# systemd (packaged)
sudo systemctl start cassandra
sudo systemctl stop cassandra
sudo systemctl restart cassandra
sudo systemctl status cassandra   # check logs/status

# legacy sysvinit
sudo service cassandra start
sudo service cassandra stop

# tarball install (run in foreground for debugging)
# from cassandra/bin directory
./cassandra -f
```

If service fails, check `journalctl -u cassandra` and the logs `/var/log/cassandra/system.log`.

---

2) `cqlsh` — CQL shell: connect + global shell commands

Start cqlsh (default host localhost:9042):

```bash
cqlsh               # connect to localhost:9042
cqlsh 10.0.0.5 9042   # connect to remote host+port
cqlsh -u cassandra -p secret  # pass username/password
cqlsh --ssl          # connect via SSL if client-to-node encryption enabled
cqlsh --debug        # debug auth/connection issues
cqlsh -e "SELECT * FROM ks.tbl LIMIT 5;"  # run single CQL statement from shell
```

Useful cqlsh “meta” commands (these are typed inside `cqlsh` prompt):

```sql

 CONSISTENCY QUORUM;       -- set consistency level for subsequent CQL queries (ONE, QUORUM, LOCAL_QUORUM, ALL, ANY, etc.)
DESCRIBE KEYSPACES;       -- list keyspaces
DESCRIBE KEYSPACE ks_name; -- show CQL for keyspace (replication, durable_writes)
DESCRIBE TABLE ks.tbl;    -- show CREATE TABLE for a table
SHOW VERSION;             -- show connected CQL/Cassandra versions
TRACING ON;               -- enable trace for subsequent queries (TRACING OFF)
PAGING ON|OFF;            -- enable/disable paging in results
COPY ks.tbl TO 'out.csv';  -- export to CSV
COPY ks.tbl FROM 'in.csv'; -- import CSV (simple bulk import, not for huge volumes)
SOURCE 'script.cql';      -- run CQL file
HELP or ?                 -- help for cqlsh commands
EXIT;                     -- leave cqlsh
```

Create keyspace example (replication):

```sql

 -- single-DC / dev (SimpleStrategy)
CREATE KEYSPACE IF NOT EXISTS demo
  WITH REPLICATION = {'class' : 'SimpleStrategy', 'replication_factor' : 1}
  AND DURABLE_WRITES = true;

-- multi-DC / production (NetworkTopologyStrategy)
CREATE KEYSPACE IF NOT EXISTS prodks
  WITH REPLICATION = {'class' : 'NetworkTopologyStrategy', 'dc1' : 3, 'dc2' : 3}
  AND DURABLE_WRITES = true;
```

Create table example:

```sql

 CREATE TABLE IF NOT EXISTS demo.users (
  user_id uuid,
  created_at timestamp,
  name text,
  email text,
  PRIMARY KEY ((user_id), created_at)
) WITH CLUSTERING ORDER BY (created_at DESC);
```

CRUD CQL examples:

```sql
 INSERT INTO demo.users (user_id, created_at, name, email) VALUES (uuid(),
toTimestamp(now()), 'Alice','a@x.com');
SELECT * FROM demo.users WHERE user_id = <uuid> LIMIT 10;
UPDATE demo.users SET email='new@x.com' WHERE user_id=<uuid> AND created_at=<ts>;
DELETE FROM demo.users WHERE user_id=<uuid> AND created_at=<ts>;
```

Batch:

```sql
 BEGIN BATCH
  INSERT INTO ks.t1 (...) VALUES (...);
  UPDATE ks.t2 ...;
APPLY BATCH;
```

(Use batches only for atomic multi-partition needs carefully — not for bulk inserts.)
Reference for cqlsh commands and `CONSISTENCY` / `COPY` / `DESCRIBE` : official CQL/cqlsh docs.

---

3) `nodetool` — cluster/node admin (run as same OS user that runs Cassandra or via sudo)

`nodetool` connects to JMX on the node (default 7199) — many commands:

Basic status / info

```bash
 nodetool status        # cluster status: Up/Down, tokens, owns, load
nodetool ring          # older alias; shows token ring info (use status with vnodes)
nodetool info          # node info (uptime, load, heap, tokens)
nodetool version        # cassandra version on the node
```

Diagnostics / metrics

```bash
nodetool tpstats          # thread pool stats (pending tasks)
nodetool compactionstats  # compaction activity
nodetool netstats         # streaming/network stats (useful during repair/bootstrap)
nodetool cfstats          # table statistics (renamed in newer versions; use table stats equivalents)
nodetool cfhistograms ks tbl  # per-table histograms (older name; check version)
nodetool proxyhistograms
```

Repair / consistency maintenance

```bash
nodetool repair                 # run anti-entropy repair on node (all ranges/tables)
nodetool repair -pr             # repair only primary ranges on this node (used when running -pr on all nodes sequentially)
nodetool repair -local          # restrict to local datacenter
# More flags exist for parallelism, incremental, hosts, etc.
```

Data maintenance

```bash
nodetool cleanup   ks [table]     # remove data no longer belonging to node after topology changes
nodetool drain                    # stop accepting writes and flush memtables (before shutdown)
nodetool flush   [keyspace] [table] # flush memtables to SSTables
nodetool scrub   ks table         # scrub (rebuild) SSTables to fix corruption
nodetool snapshot -t name ks [tbl] # take snapshot (point-in-time backup)
nodetool clearsnapshot -t name    # remove named snapshot
```

Node lifecycle

```bash
nodetool decommission   # gracefully remove node (stream data to replicas)
nodetool removenode <host-id>  # remove dead node from ring (after proper steps)
nodetool assassinate <ip>     # forcefully remove dead node (last-resort)
nodetool bootstrap <options>  # monitor bootstrap process when adding node
nodetool move <new-token>     # move node token (avoid on vnodes unless necessary)
nodetool rebuild [options]    # rebuild data by streaming from other nodes (use when replacing node)
```

Examples

```bash
# show cluster status for keyspace `prodks`
nodetool status prodks

# compact all tables in a keyspace (force compaction)
nodetool compact prodks

# take snapshot of keyspace 'prodks'
nodetool snapshot -t pre-upgrade prodks
```

`nodetool` has many flags and subcommands — run `nodetool help` on your node. See the Apache nodetool docs for full list and semantics.

---

4) Bulk load / export / restore

COPY (cqlsh) — simple CSV import/export (for small-medium datasets):

```sql
COPY ks.tbl TO 'out.csv';
COPY ks.tbl FROM 'in.csv' WITH DELIMITER=',' AND HEADER=TRUE;
```

sstableloader (bulk load SSTables) — recommended for large bulk loads or restoring snapshots:

```bash
# load SSTables (point to host(s) to stream into)
sstableloader -d 10.0.0.2,10.0.0.3 /path/to/keyspace/table-<sstabledir>

# if SSL enabled, pass --conf-path to read encryption options from cassandra.yaml
sstableloader --conf-path /etc/cassandra/cassandra.yaml -d 10.0.0.2 /path/to/sstables
```

nodetool import — another option (works with SSTables too). For details, see Bulk Loading docs.

---

5) Stress & benchmarking

cassandra-stress (simple example — creates schema, writes, reads):

```bash
# run default stress profile (writes)
cassandra-stress write n=100000 -node 10.0.0.5

# run mixed read/write for duration
cassandra-stress mixed duration=1m -rate threads=50 -node 10.0.0.5
```

Note: Apache docs mark `cassandra-stress` as deprecated and recommend **NoSQLBench** for modern benchmarking; but `cassandra-stress` is still commonly used for quick testing.

---

6) Common admin flows & example sequences

Add a new node (high-level):

1. install Cassandra on new machine, configure `cassandra.yaml` (`seeds`, `cluster_name`, `listen_address`, `rpc_address`, datacenter/rack via snitch).
2. start node (`sudo systemctl start cassandra` or `./cassandra -f`) — it will bootstrap (stream data) from other nodes.
3. monitor with `nodetool status`, `nodetool netstats`, `nodetool compactionstats`.

Decommission a node (graceful remove):

```bash
# on node to remove
nodetool decommission
# verify from another node:
nodetool status
```

Remove a dead node (forceful cleanup):

```bash
```

```
# from a remaining node:
nodetool removenode <host-id>
# if removenode impossible, use assassinate as last resort (dangerous)
nodetool assassinate <ip>
```

Take backup (snapshot) and restore:

```bash
# take snapshot
nodetool snapshot -t mybackup prodks

# list snapshots (filesystem)
ls /var/lib/cassandra/data/<keyspace>/<table>-*/snapshots/mybackup

# restore: typically copy SSTables back to data dir on target node and run sstableloader or nodetool refresh/import
sstableloader -d destination-node /path/to/snapshot/sstables
```

Repair schedule (important): run `nodetool repair` regularly to prevent tombstone/consistency problems — for multi-DC production use `-pr` carefully and/or DC-aware options. See docs for recommended frequency for your setup.

---

7) Useful tips & gotchas (short)

- **Replication strategy**: use `NetworkTopologyStrategy` for production multi-DC; `SimpleStrategy` is for single-DC/dev only. Don't use `SimpleStrategy` for multi-DC.
- **Consistency levels**: reads/writes are controlled by consistency (ONE, QUORUM, LOCAL_QUORUM, ALL). Set via drivers or `CONSISTENCY` in `cqlsh`.

- **Never stop a node abruptly in some maintenance operations** — use `nodetool drain` before shutting down to avoid long recoveries.
- **Snapshots are not enough** — snapshot files are hard links to SSTables; for full restore you may need `sstableloader` or careful file copy + `nodetool refresh`.

- **Large batch statements are anti-pattern** — avoid huge batches for bulk inserts.

8) Quick printable cheat-sheet (most used commands)

Service

```bash
sudo systemctl start cassandra
sudo systemctl stop cassandra
sudo systemctl status cassandra
```

cqlsh

```bash
cqlsh
cqlsh -u user -p pass
CONSISTENCY QUORUM;
DESCRIBE KEYSPACES;
DESCRIBE KEYSPACE ks;
DESCRIBE TABLE ks.tbl;
TRACING ON;
COPY ks.tbl TO 'out.csv';
COPY ks.tbl FROM 'in.csv';
SOURCE 'script.cql';
```

nodetool

```bash
nodetool status
nodetool ring
nodetool info
nodetool snapshot -t snapname ks
nodetool clearsnapshot -t snapname
nodetool repair -pr
nodetool cleanup
nodetool flush
nodetool decommission
nodetool removenode <host-id>
nodetool netstats
nodetool compactionstats
nodetool tpstats
nodetool help
```

bulk / other

```bash
 sstableloader -d 10.0.0.2 /path/to/sstabledir
cassandra-stress write n=100000 -node 10.0.0.5
```

References (official docs — read these for exact flags & version specifics)

- CQL shell / cqlsh commands reference (DESCRIBE, COPY, CONSISTENCY).

- Nodetool commands & usage (status, repair, snapshot, etc.).
- sstableloader / bulk loading docs.
- Cassandra Stress / benchmarking (note deprecation suggestion).
- Create keyspace & replication strategy guidance.

If you want, I can now:

- produce a single **copy-pasteable shell script** that installs Cassandra (Ubuntu example) + common `nodetool` checks and `cqlsh` sanity queries for a fresh VM, **or**
- give a single-page printable PDF cheat-sheet, or
- walk step-by-step through **adding a node** or **recovering a node** with exact commands for your Cassandra version.

Which of those would help you most next?

Sources