Redis

Commands for the in-memory data store, type by type.

01 Install & connect

```
brew install redis              # macOS
sudo apt install redis-server   # Ubuntu
docker run -d -p 6379:6379 redis
redis-server                    # start
redis-cli                       # connect
redis-cli -h host -p 6379 -a password
```

The default port is 6379. redis-cli PING replies PONG.

02 Data types

String	Text, numbers, counters
List	Ordered: queues, stacks
Hash	Fields of one object
Set	Unique members: tags
Sorted set	Members ranked by score
Stream	Append-only message log

Name keys with colons, such as user:1000:profile.

03 Strings & counters

```
SET name "Redis"
GET name
SET lock "1" NX EX 30   # if absent, 30s TTL
SET name "v2" XX        # only if it exists
MSET k1 "a" k2 "b"
MGET k1 k2
INCR views              # atomic +1
INCRBY views 10         # DECR, DECRBY too
APPEND name " rocks"    # STRLEN, GETRANGE
```

SET with NX, XX and EX replaces the deprecated SETNX and SETEX. INCR is atomic, so no update is lost.

04 Keys & expiry

```
EXISTS user:1           # 1 or 0
TYPE user:1             # string, hash...
DEL k1 k2               # UNLINK: non-blocking
EXPIRE session 3600     # seconds
TTL session             # -1 none, -2 missing
PERSIST session         # remove the TTL
SCAN 0 MATCH user:* COUNT 100
```

SCAN returns a new cursor; call again until it is 0. Avoid KEYS * in production: it blocks the server.

05 Lists

```
RPUSH queue "job1" "job2"  # tail
LPUSH queue "urgent"       # head
LPOP queue                 # or RPOP
LRANGE queue 0 -1          # all items
LLEN queue
LTRIM recent 0 99          # keep 100
BLPOP queue 0              # wait forever
LMOVE src dst RIGHT LEFT
```

RPUSH with LPOP is a FIFO queue; LPUSH with LPOP a stack. LMOVE replaces the deprecated RPOPLPUSH.

06 Hashes

```
HSET user:1 name "John" age 30
HGET user:1 name
HMGET user:1 name age
HGETALL user:1
HINCRBY user:1 age 1
HDEL user:1 age          # HEXISTS, HLEN
```

HSET sets many fields at once; HMSET is deprecated. Read or update one field without loading the rest.

07 Sets

```
SADD tags "redis" "cache"  # returns added
SREM tags "cache"
SISMEMBER tags "redis"     # 1 or 0
SMEMBERS tags
SCARD tags                 # count
SINTER user:1:friends user:2:friends
SUNION a b                 # SDIFF a b
```

Members are unique. SRANDMEMBER and SPOP pick random members; SINTERSTORE dest a b saves a result.

08 Sorted sets

```
ZADD board 100 "alice" 95 "bob"
ZINCRBY board 50 "bob"
ZRANGE board 0 9 REV WITHSCORES  # top 10
ZRANGE board 100 200 BYSCORE
ZSCORE board "bob"
ZRANK board "bob"                # 0-based
ZCARD board                      # count
ZREM board "bob"                 # ZPOPMAX
```

REV and BYSCORE replace the deprecated ZREVRANGE and ZRANGEBYSCORE. (100 is exclusive; -inf, +inf are open.

09 Databases & server

```
SELECT 1              # databases 0-15
DBSIZE                # keys in this DB
INFO memory           # INFO replication
CONFIG GET maxmemory
FLUSHDB               # clear this DB
FLUSHALL              # clear every DB
```

Cluster mode only has database 0. CONFIG SET changes a setting at runtime.

Redis

Messaging, atomic updates, clients and running in production.

10 Pub/Sub

```
SUBSCRIBE news             # terminal 1
PUBLISH news "Hello"       # terminal 2
PSUBSCRIBE events:order:*  # pattern
UNSUBSCRIBE news           # PUNSUBSCRIBE
PUBSUB CHANNELS            # active ones
PUBSUB NUMSUB news         # subscribers
```

Fire-and-forget: nothing is stored, so a subscriber only gets messages sent while it is connected. Use Streams when messages must persist.

11 Streams & consumer groups

```
XADD jobs * task "send_email" user 42
XRANGE jobs - + COUNT 10
XREAD BLOCK 5000 STREAMS jobs $
XGROUP CREATE jobs workers 0 MKSTREAM
XREADGROUP GROUP workers w1 STREAMS jobs >
XACK jobs workers 1700000000000-0
XPENDING jobs workers      # not acked
XTRIM jobs MAXLEN 1000
```

* generates the ID; $ waits for new entries; > reads entries not yet delivered to the group. Entries persist until trimmed.

12 Transactions & Lua

```
WATCH balance          # optimistic lock
MULTI
DECRBY balance 50
INCRBY savings 50
EXEC                   # nil if it changed
EVAL "return redis.call('GET', KEYS[1])" 1 k
```

Queued commands run together. There is no rollback: a failing command does not undo the rest. DISCARD cancels.

13 Python (redis-py)

```
import redis
r = redis.Redis(decode_responses=True)
r.set("session:abc", "data", ex=3600)
pipe = r.pipeline()       # one round trip
for i in range(1000):
    pipe.set(f"key{i}", i)
pipe.execute()
```

pip install redis. A pipeline runs as MULTI/EXEC by default. Pub/Sub: r.pubsub(), subscribe(), then listen().

14 Node.js (node-redis)

```
import { createClient } from 'redis';
const client = createClient();
client.on('error', console.error);
await client.connect();
await client.set('key', 'value');
const sub = client.duplicate();
await sub.connect();
await sub.subscribe('news', console.log);
```

npm install redis. Pass { url: 'redis://host:6379' } to connect elsewhere. A subscribed connection runs no other commands.

15 Persistence: RDB & AOF

```
BGSAVE                     # RDB snapshot
CONFIG SET appendonly yes  # enable AOF
BGREWRITEAOF               # compact AOF
INFO persistence
save 900 1                 # redis.conf
appendfsync everysec       # redis.conf
```

RDB snapshots are compact but can lose recent writes. AOF logs every write; everysec loses at most a second. Use both.

16 Memory & eviction

```
CONFIG SET maxmemory 2gb
CONFIG SET maxmemory-policy allkeys-lru
```

noeviction	Writes fail when full
allkeys-lru	Least recently used
allkeys-lfu	Least frequently used
volatile-lru	LRU, TTL keys only
volatile-ttl	Shortest TTL first

MEMORY USAGE key shows the bytes one key uses.

17 Monitoring & performance

```
SLOWLOG GET 10
CONFIG SET slowlog-log-slower-than 10000
INFO stats                 # CLIENT LIST
redis-cli --bigkeys        # --memkeys
redis-cli --latency
redis-benchmark -q -n 100000
```

The slowlog threshold is in microseconds. MONITOR prints every command, so use it only to debug.

18 Replication & cluster

```
REPLICAOF 10.0.0.1 6379  # become a replica
REPLICAOF NO ONE         # promote
INFO replication
CLUSTER INFO             # cluster_state:ok
CLUSTER KEYSLOT user:1
MSET {user:1}:a 1 {user:1}:b 2
```

Build one with redis-cli --cluster create. Multi-key commands need one slot: use {hash tags}.