Memcached Cheatsheet

A quick reference for Memcached, covering basic commands, data management, and common use cases. Ideal for developers needing a fast, distributed memory object caching system.

Core Concepts & Commands

Basic Operations

`set <key> <flags> <exptime> <bytes>\r\n<data>\r\n`	Stores data under the specified key. Flags are arbitrary 16-bit integer, exptime is expiration time in seconds (0 means never expire), bytes is data length. **Example:** `set mykey 0 3600 5\r\nvalue\r\n`
`get <key>\r\n`	Retrieves data associated with the specified key. **Example:** `get mykey\r\n`
`add <key> <flags> <exptime> <bytes>\r\n<data>\r\n`	Stores data under the specified key, but only if the server *doesn't* already hold data for this key. **Example:** `add newkey 0 3600 5\r\nvalue\r\n`
`replace <key> <flags> <exptime> <bytes>\r\n<data>\r\n`	Stores data under the specified key, but only if the server *does* already hold data for this key. **Example:** `replace existingkey 0 3600 5\r\nvalue\r\n`
`delete <key> <time>\r\n`	Deletes data associated with the specified key. Time is an optional delay before deletion (in seconds). **Example:** `delete mykey\r\n`
`incr <key> <value>\r\n`	Increments the value of the key by value. The value must be an integer. **Example:** `incr counter 1\r\n`
`decr <key> <value>\r\n`	Decrements the value of the key by value. The value must be an integer. **Example:** `decr counter 1\r\n`

Flags

Flags are an arbitrary 16-bit integer that the client stores along with the data. It is opaque to the server.
Clients can use flags to indicate the type of data being stored (e.g., serialized object, compressed data).
When data is retrieved via `get`, the flags are also returned.

Advanced Features

CAS (Check and Set)

`gets <key>\r\n`	Retrieves data with a unique CAS identifier. **Example:** `gets mykey\r\n`
`cas <key> <flags> <exptime> <bytes> <cas unique>\r\n<data>\r\n`	Stores data only if the CAS identifier matches the current value. Prevents race conditions. **Example:** `cas mykey 0 3600 5 12345\r\nvalue\r\n`

Multi-Get

`get <key1> <key2> ... <keyN>\r\n`

Retrieves multiple keys in a single request, reducing network overhead.

Example:
`get key1 key2 key3\r\n`

Expiration

Expiration time is specified in seconds. A value of 0 means the item never expires (although it may be evicted from the cache if memory is needed).

If the expiration time is greater than 30 days (2592000 seconds), it is treated as a Unix timestamp.

LRU (Least Recently Used)

Memcached uses an LRU algorithm to evict items from the cache when it runs out of memory. Least recently used items are removed first.

Configuration & Management

Starting Memcached

`memcached -m <memory> -p <port> -u <user> -d`

Starts Memcached with specified memory allocation, port, user, and as a daemon.

Example:
`memcached -m 64 -p 11211 -u memcache -d`

Configuration Options

`-m <memory>`	Sets the maximum memory to use for cache items, in MB.
`-p <port>`	Sets the port number to listen on (default: 11211).
`-u <user>`	Runs the daemon as the specified user.
`-d`	Runs Memcached as a daemon.
`-l <ip_addres s>`	Bind Memcached to a specific IP address. Useful for multi-homed servers.
`-c <connectio ns>`	Sets the maximum number of concurrent connections.

Stats

`stats\r\n`

Displays various statistics about the Memcached server, such as uptime, cache hits, misses, and memory usage.

Example Output:

`STAT pid 12345\r\nSTAT uptime 1234\r\nSTAT bytes 123456\r\nEND`

Client Libraries

Popular Libraries

PHP	`memcached`, `Memcache`
Python	`pymemcache`, `python-memcached`
Java	`spymemcached`, `xmemcached`
Ruby	`dalli`, `memcache`
Node.js	`memcached`, `node-cache`

Basic Usage (Python - pymemcache)

```python
from pymemcache.client.base import Client

client = Client('127.0.0.1:11211')

client.set('my_key', 'my_value')
value = client.get('my_key')

print(value)
```