Storing Data In and Accessing Data from Redis

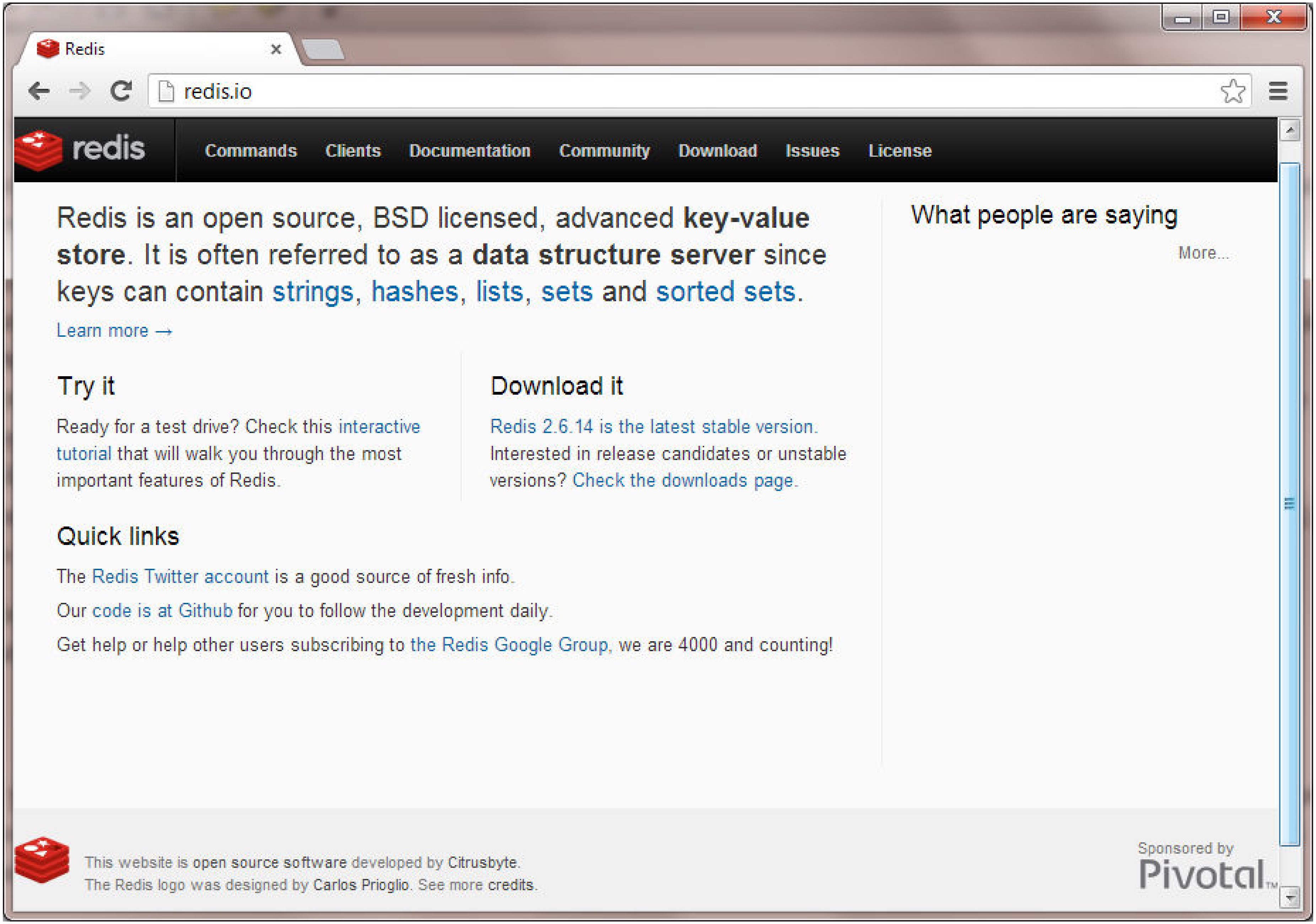
□ Redis is a persistent key/value store.

□ For efficiency it holds the database in memory and writes to disks in an asynchronous thread.

□ The values it holds can be strings, lists, hashes, sets, and sorted sets.

□ It is often referred to as a data structure server since keys can contain strings, hashes, lists, sets and sorted sets.

□ It provides a rich set of commands to manipulate its collections and insert and fetch data.



Redis is an open source, BSD licensed, advanced **key-value store**. It is often referred to as a **data structure server** since keys can contain strings, hashes, lists, sets and sorted sets.

Learn more →

Try it

Ready for a test drive? Check this interactive tutorial that will walk you through the most important features of Redis.

Download it

Redis 2.6.14 is the latest stable version. Interested in release candidates or unstable versions? Check the downloads page.

Quick links

The Redis Twitter account is a good source of fresh info.

Our code is at Github for you to follow the development daily.

Get help or help other users subscribing to the Redis Google Group, we are 4000 and counting!

What people are saying

More...

This website is open source software developed by Citrusbyte.
The Redis logo was designed by Carlos Prioglio. See more credits.

Saving a Key/Value Pair in Redis

- To save a key/value pair — { akey: “avalue” } — simply type the following command:

  ```
  redis-cli> set akey “avalue”
  ```

- If you see OK on the console in response to the command you just typed, then things look good.

- To confirm that avalue is stored for akey, simply get the value for akey like so:

  ```
  redis-cli> get akey
  ```

- You should see a value in response to this request.

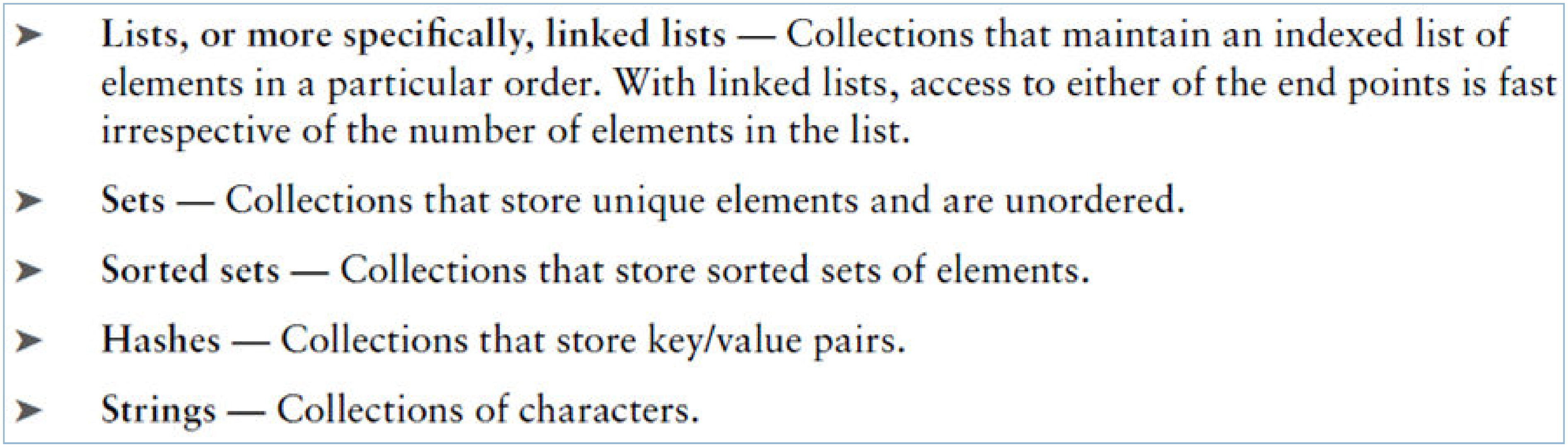
Redis Example

- In this Redis example, a database stores a list of book titles.
- Each book is tagged using an arbitrary set of tags.

- **For example,**
- we add “The Omnivore’s Dilemma” by “Michael Pollan” to the list
  - tag it with the following: “organic,” “industrialization,” “local,” “written by a journalist,” “best seller,” and “insight”
- or add “Outliers” by “Malcolm Gladwell”
  - tag it with the following: “insight,” “best seller,” and “written by a journalist.”

- Now we can get all the books on my list or all the books “written by a journalist” or all those that relate to “organic.”
- We could also get a list of the books by a given author.

Redis supports a few different data structures



- Lists, or more specifically, linked lists — Collections that maintain an indexed list of elements in a particular order. With linked lists, access to either of the end points is fast irrespective of the number of elements in the list.
- Sets — Collections that store unique elements and are unordered.
- Sorted sets — Collections that store sorted sets of elements.
- Hashes — Collections that store key/value pairs.
- Strings — Collections of characters.

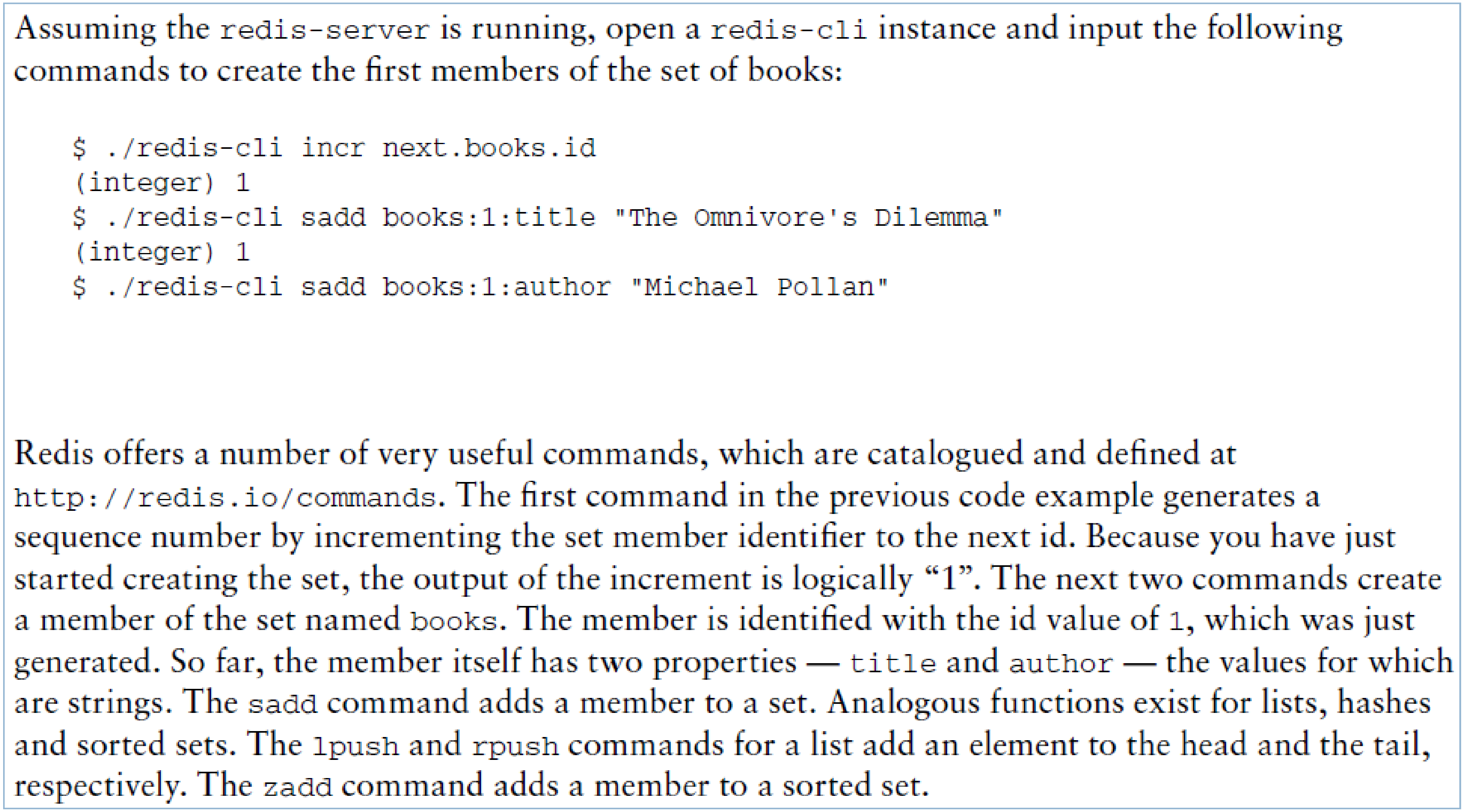
Using Set

- For the example at hand, we chose to use a set,
  - because order isn't important.

- We call this set "books".

- Each book, which is a member of the set of books, has the following properties:
  - Id
  - Title
  - Author
  - Tags (a collection)

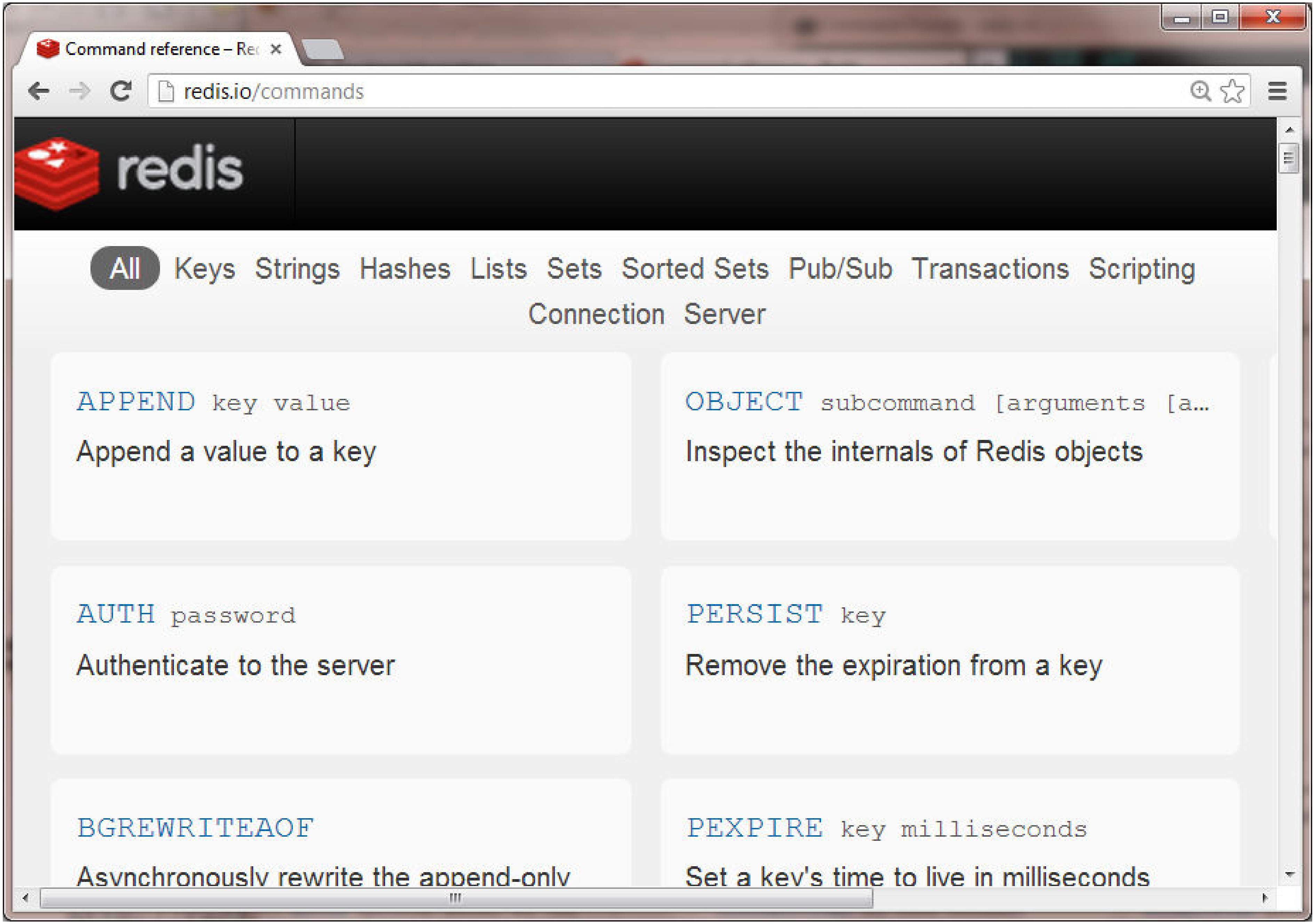
- Each tag is identified with the help of the following properties:
  - Id
  - Name



Assuming the redis-server is running, open a redis-cli instance and input the following commands to create the first members of the set of books:

```
$ ./redis-cli incr next.books.id
(integer) 1
$ ./redis-cli sadd books:1:title "The Omnivore's Dilemma"
(integer) 1
$ ./redis-cli sadd books:1:author "Michael Pollan"
```

Redis offers a number of very useful commands, which are catalogued and defined at `http://redis.io/commands`. The first command in the previous code example generates a sequence number by incrementing the set member identifier to the next id. Because you have just started creating the set, the output of the increment is logically "1". The next two commands create a member of the set named `books`. The member is identified with the id value of `1`, which was just generated. So far, the member itself has two properties — `title` and `author` — the values for which are strings. The `sadd` command adds a member to a set. Analogous functions exist for lists, hashes and sorted sets. The `lpush` and `rpush` commands for a list add an element to the head and the tail, respectively. The `zadd` command adds a member to a sorted set.



APPEND key value

Append a value to a key

OBJECT subcommand [arguments [a…

Inspect the internals of Redis objects

AUTH password

Authenticate to the server

PERSIST key

Remove the expiration from a key

BGREWRITEAOF

Asynchronously rewrite the append-only

PEXPIRE key milliseconds

Set a key's time to live in milliseconds

Next, add a bunch of tags to the member you added to the set named `books`. Here is how you do it:

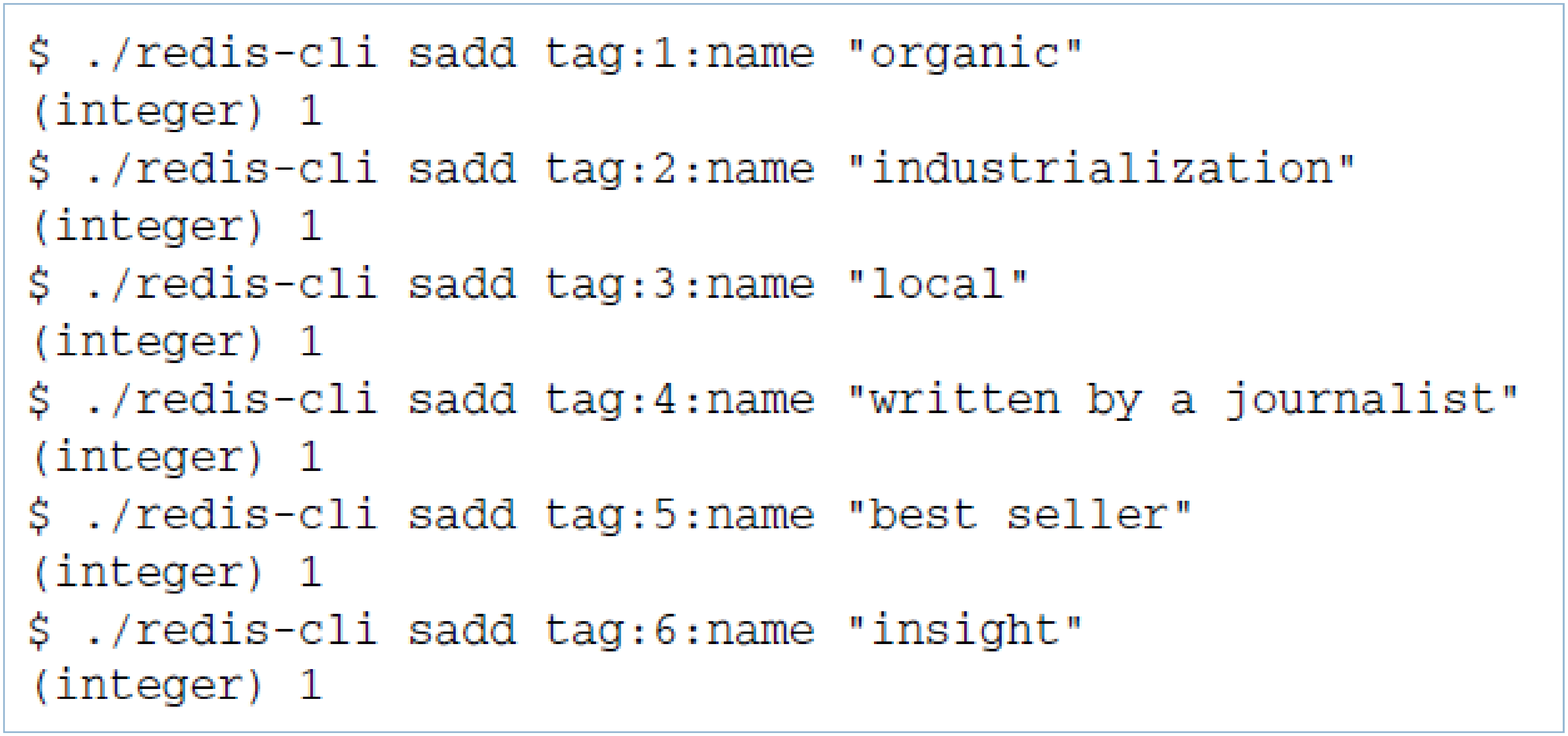
```
$ ./redis-cli sadd books:1:tags 1
(integer) 1
$ ./redis-cli sadd books:1:tags 2
(integer) 1
$ ./redis-cli sadd books:1:tags 3
(integer) 1
$ ./redis-cli sadd books:1:tags 4
(integer) 1
$ ./redis-cli sadd books:1:tags 5
(integer) 1
$ ./redis-cli sadd books:1:tags 6
(integer) 1
```

Numeric Tag Idetifiers

- A bunch of numeric tag identifiers were added to the member identified by the id value of 1.
- Tags themselves have not been defined any more than having been assigned an id so far.
- It may be worthwhile to break down the constituents of *books:1:tags* a bit further and explain how the key naming systems work in Redis.
- Any string, except those containing whitespace and special characters, are good choices for a key in Redis.
- Avoid very long or short keys. Keys can be structured in a manner where a hierarchical relationship can be established and nesting of objects and their properties can be established.
- It is a suggested practice and convention to use a scheme like

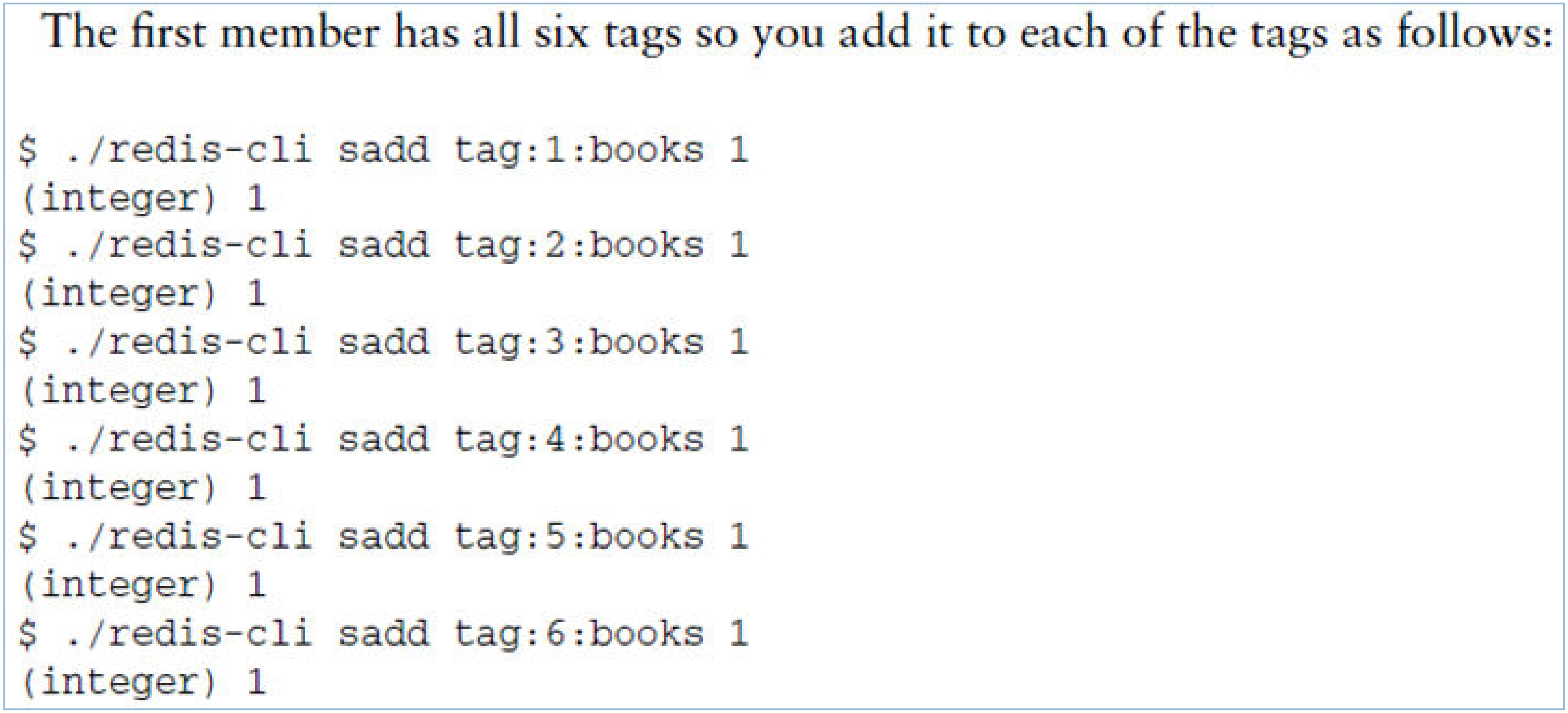
  object-type:id:field for key names.
- Therefore, a key such as *books:1:tags* implies a tag collection for a member identified by the id 1 within a set named "books."
- Similarly, *books:1:title* means title field of a member, identified by an id value of 1, within the set of books.

After adding a bunch of tags to the first member of the set of books, you can define the tags themselves like so:



```
$ ./redis-cli sadd tag:1:name "organic"
(integer) 1
$ ./redis-cli sadd tag:2:name "industrialization"
(integer) 1
$ ./redis-cli sadd tag:3:name "local"
(integer) 1
$ ./redis-cli sadd tag:4:name "written by a journalist"
(integer) 1
$ ./redis-cli sadd tag:5:name "best seller"
(integer) 1
$ ./redis-cli sadd tag:6:name "insight"
(integer) 1
```

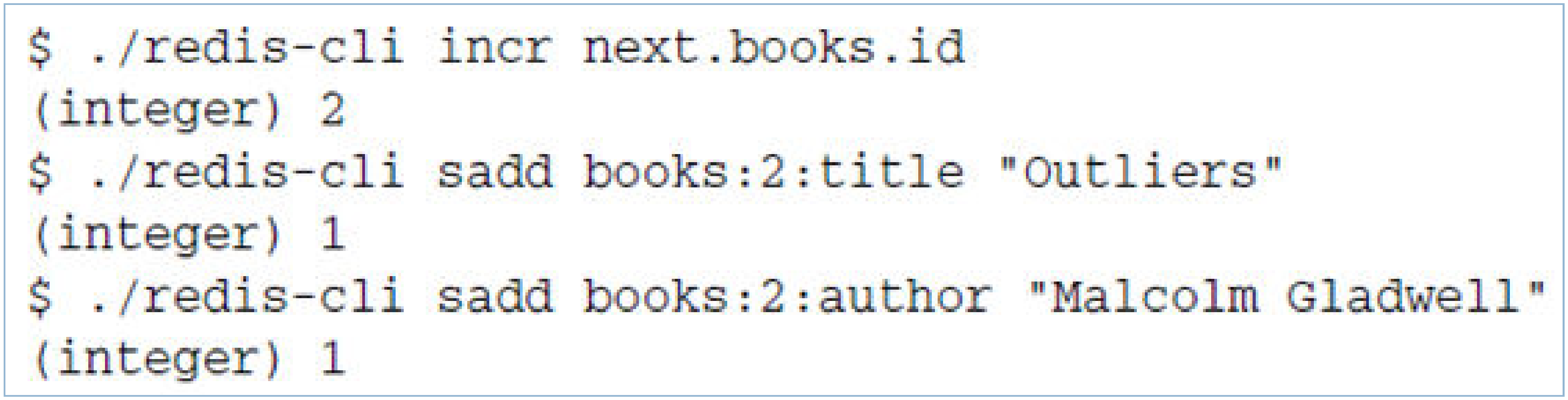
With tags defined, you establish the reciprocal relationship to associate books that have the particular tags.



The first member has all six tags so you add it to each of the tags as follows:

```
$ ./redis-cli sadd tag:1:books 1
(integer) 1
$ ./redis-cli sadd tag:2:books 1
(integer) 1
$ ./redis-cli sadd tag:3:books 1
(integer) 1
$ ./redis-cli sadd tag:4:books 1
(integer) 1
$ ./redis-cli sadd tag:5:books 1
(integer) 1
$ ./redis-cli sadd tag:6:books 1
(integer) 1
```

After the cross-relationships are established, you would create a second member of the set like so:



```
$ ./redis-cli incr next.books.id
(integer) 2
$ ./redis-cli sadd books:2:title "Outliers"
(integer) 1
$ ./redis-cli sadd books:2:author "Malcolm Gladwell"
(integer) 1
```

incr function

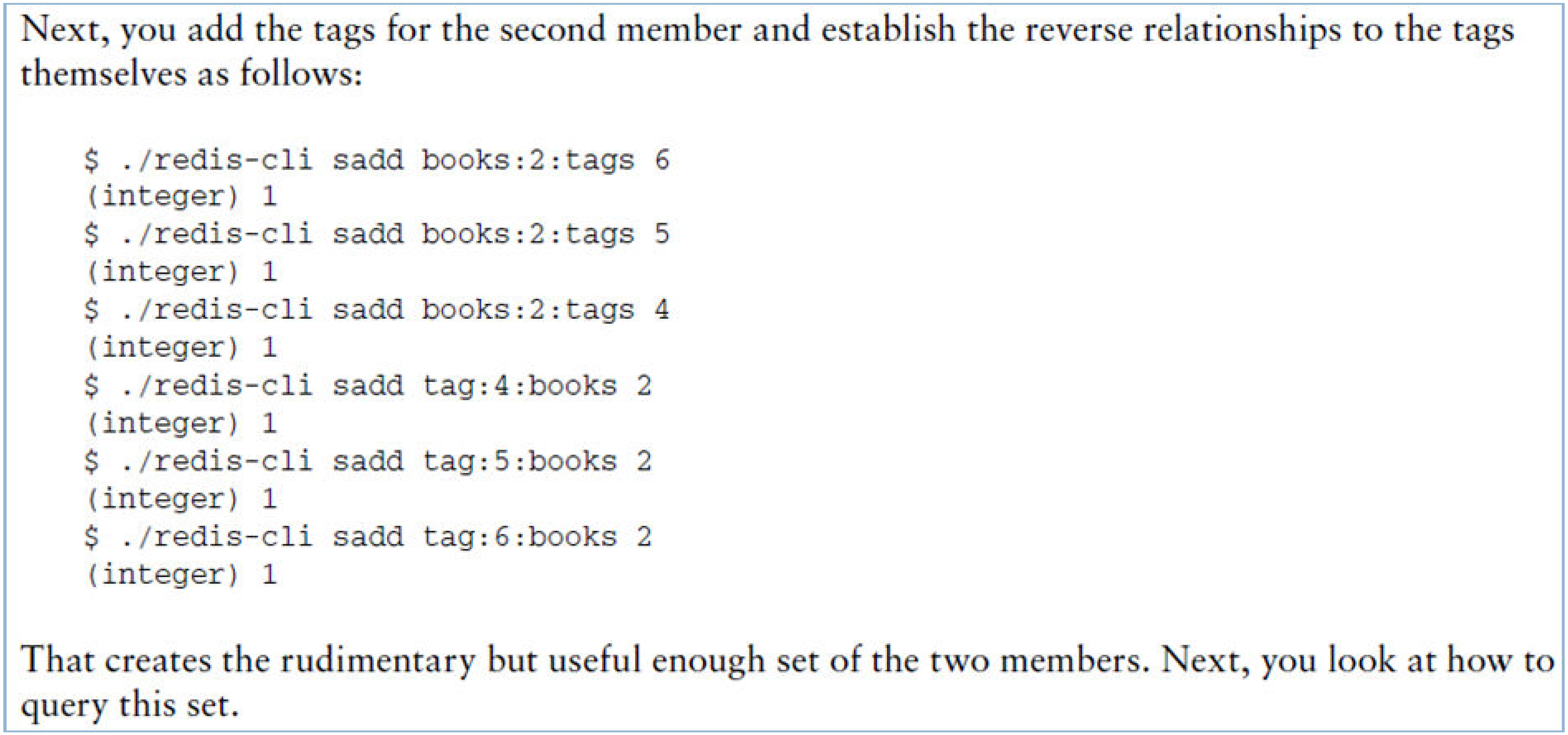
- The incr function is used to generate the id for the second member of the set.

- Functions like
  - ***incrby***, which allows increment by a defined step;
  - ***decr***, which allows you to decrement; and
  - ***decrby***, which allows you to decrement by a defined step

  are also available as useful utility functions whenever sequence number generation is required.

- You can choose the appropriate function and define the step as required.

- For now incr does the job.



Next, you add the tags for the second member and establish the reverse relationships to the tags themselves as follows:

```
$ ./redis-cli sadd books:2:tags 6
(integer) 1
$ ./redis-cli sadd books:2:tags 5
(integer) 1
$ ./redis-cli sadd books:2:tags 4
(integer) 1
$ ./redis-cli sadd tag:4:books 2
(integer) 1
$ ./redis-cli sadd tag:5:books 2
(integer) 1
$ ./redis-cli sadd tag:6:books 2
(integer) 1
```

That creates the rudimentary but useful enough set of the two members. Next, you look at how to query this set.

Querying Redis

- Continuing with the redis-cli session, you can first list the title and author of member 1, identified by the id 1, of the set of books as follows:
  - $ ./redis-cli smembers books:1:title
  - 1. “The Omnivore\xe2\x80\x99s Dilemma”
  - $ ./redis-cli smembers books:1:author
  - 1. “Michael Pollan”

- The special characters in the title string represent the apostrophe that was introduced in the string value.

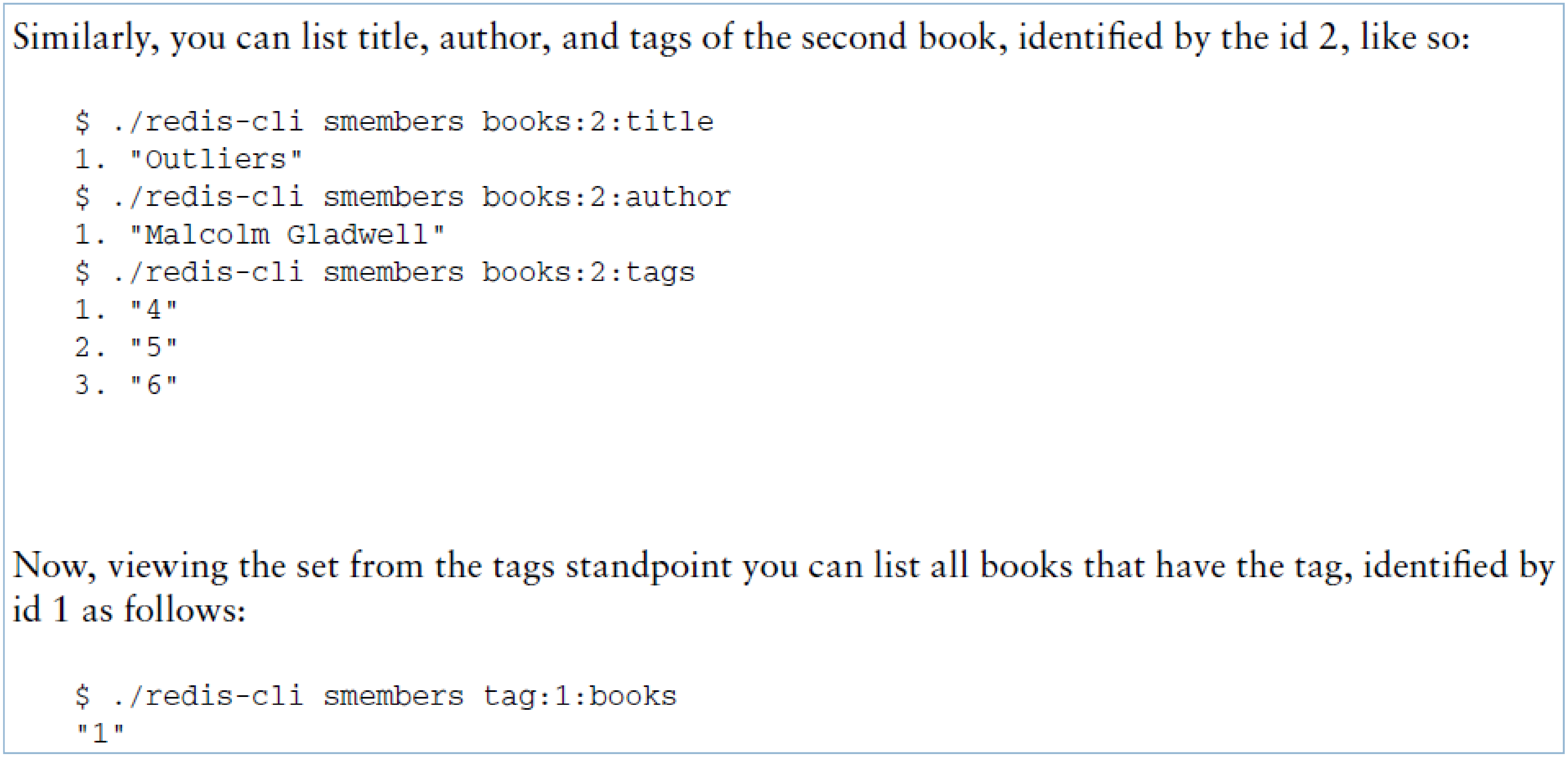
You can list all the tags for this book like so:

```
$ ./redis-cli smembers books:1:tags
    1. “4”
    2. “1”
    3. “2”
    4. “3”
    5. “5”
    6. “6”
```

- Notice that the list of tag ids is not ordered in the same way as they were entered.

- Question:
  - Why do you think that members are not ordered?

- Question:
  - If you need an ordered set, what would you do?

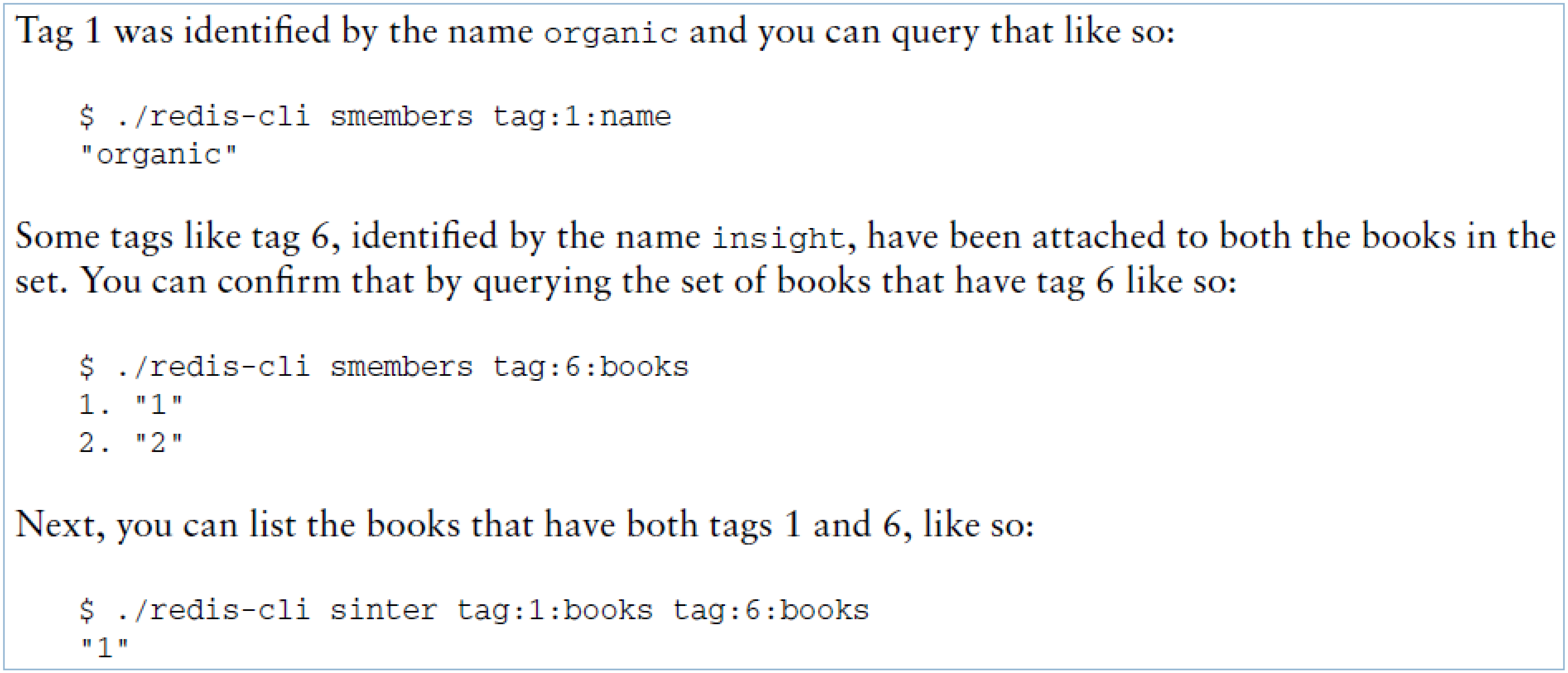


Similarly, you can list title, author, and tags of the second book, identified by the id 2, like so:

```
    $ ./redis-cli smembers books:2:title
    1. "Outliers"
    $ ./redis-cli smembers books:2:author
    1. "Malcolm Gladwell"
    $ ./redis-cli smembers books:2:tags
    1. "4"
    2. "5"
    3. "6"
```

Now, viewing the set from the tags standpoint you can list all books that have the tag, identified by id 1 as follows:

```
    $ ./redis-cli smembers tag:1:books
    "1"
```



Tag 1 was identified by the name `organic` and you can query that like so:

```
$ ./redis-cli smembers tag:1:name
"organic"
```

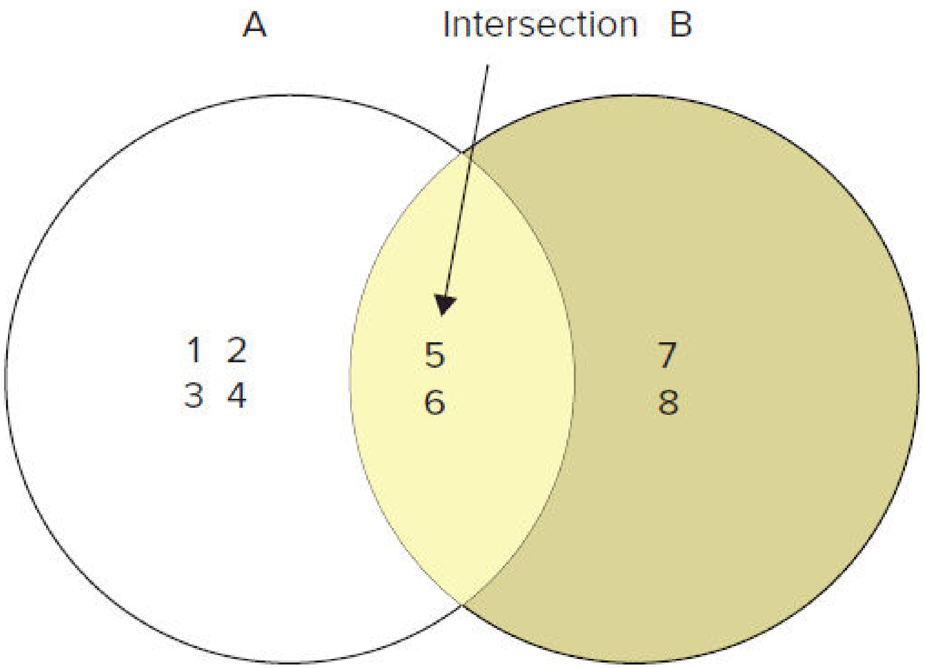
Some tags like tag 6, identified by the name `insight`, have been attached to both the books in the set. You can confirm that by querying the set of books that have tag 6 like so:

```
$ ./redis-cli smembers tag:6:books
1. "1"
2. "2"
```

Next, you can list the books that have both tags 1 and 6, like so:

```
$ ./redis-cli sinter tag:1:books tag:6:books
"1"
```

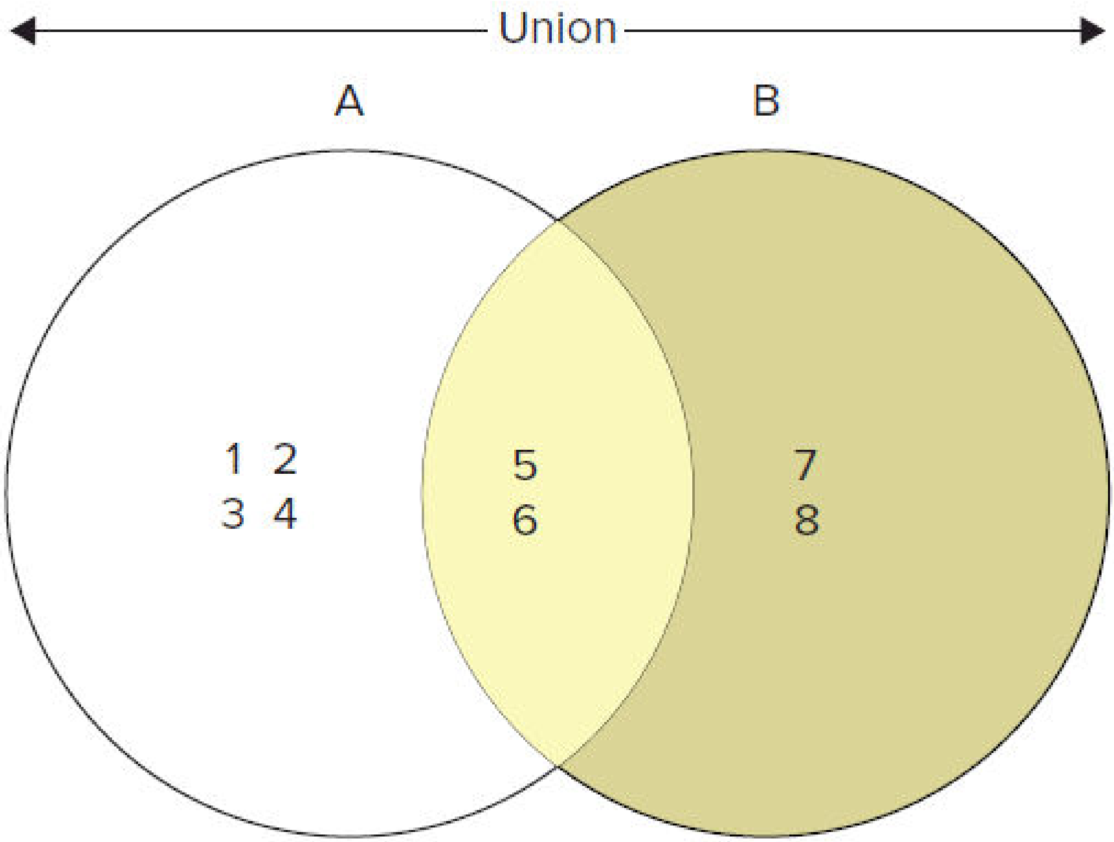
The sinter command allows you to query for the intersection of two or more sets. If the word "intersection" has befuddled you, then review the Venn diagram in Figure 3-2 to set things back to normal.



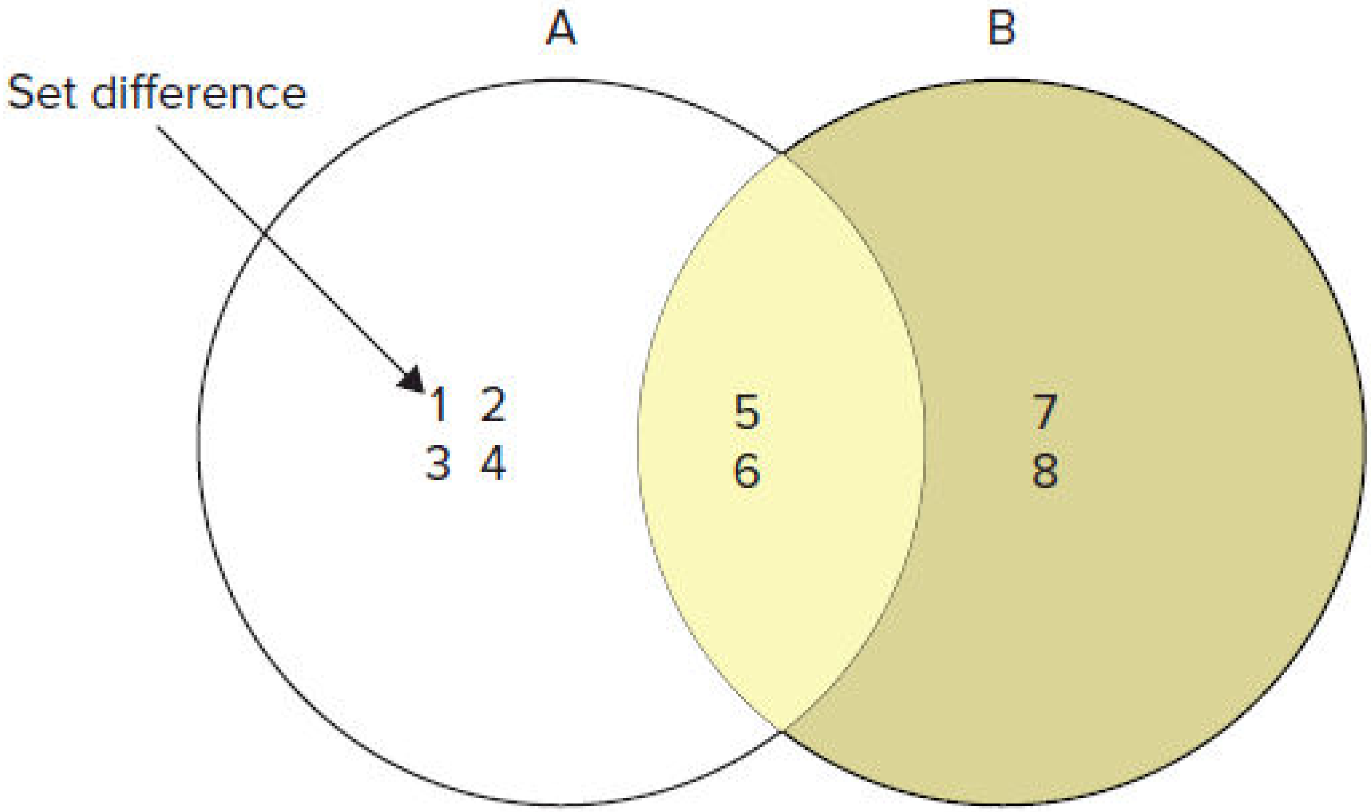
5 and 6 belong to both sets A and B

You know both books 1 and 2 have tags 5 and 6, so a sinter between the books of tags 5 and 6 should list both books. You can run the sinter command to confirm this. The command and output are as follows:

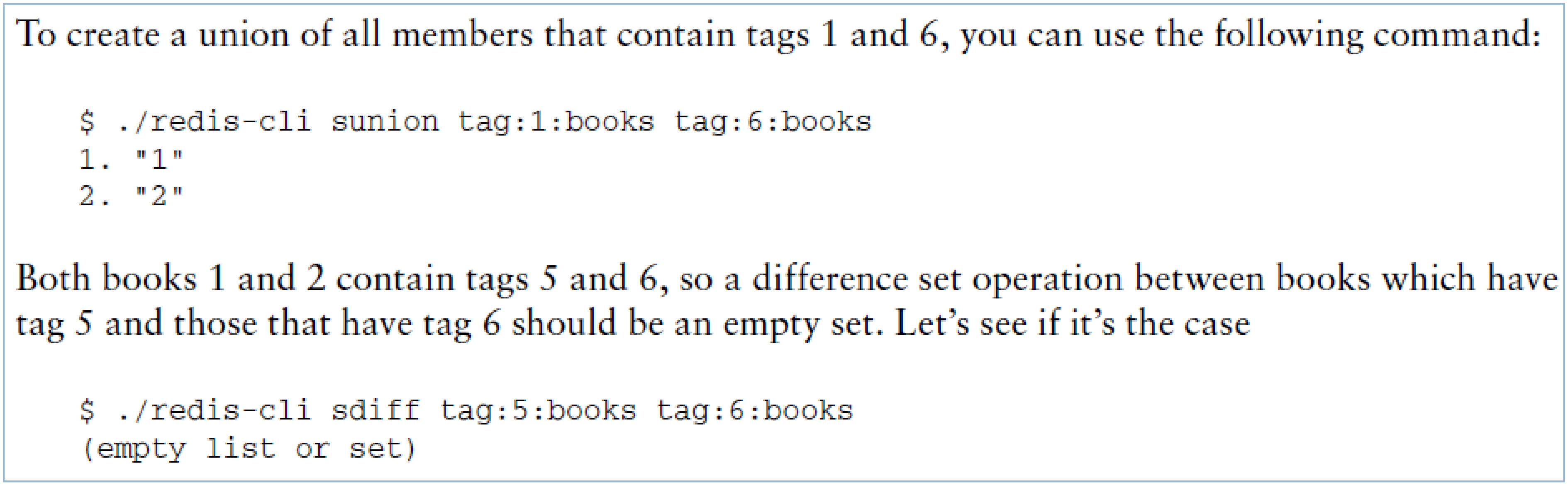
```
$ ./redis-cli sinter tag:5:books tag:6:books
1. "1"
2. "2"
```



1, 2, 3, 4, 5, 6, 7, and 8 (all members in both A and B)
belong to the union of sets A and B



The difference A-B contains members that belong to A but not B.
Thus, A-B contains 1, 2, 3 and 4.



To create a union of all members that contain tags 1 and 6, you can use the following command:

```
$ ./redis-cli sunion tag:1:books tag:6:books
1. "1"
2. "2"
```

Both books 1 and 2 contain tags 5 and 6, so a difference set operation between books which have tag 5 and those that have tag 6 should be an empty set. Let's see if it's the case

```
$ ./redis-cli sdiff tag:5:books tag:6:books
(empty list or set)
```