

Redis — The AK-47 Of Post-relational Databases

Karel Minařík

Karel Minařík

- Independent web designer and developer
- Ruby, Rails, Git, CouchDB *propagandista* in .cz
- Previously: Flash Developer; Art Director; Information Architect;... (see *LinkedIn*)
- @karmi^q at Twitter

- **karmi.cz**



AK-47

noun

a type of assault rifle, originally manufactured in the Soviet Union.

ORIGIN acronym for Russian *Avtomat Kalashnikova 1947*, the designation of the original model, designed in 1947 by Mikhail T. Kalashnikov (b. 1919).

AK-47

Designed at the end of WWII by Mikhail Kalashnikov

Assault Rifle, not a “submachine gun”

Simple design

Designed for mass production & low-quality manufacturing

Extreme reliability, at the cost of accuracy



<http://www.youtube.com/watch?v=J6c3DLIM9KA#t=8m32s>

Reliability



sim•plic•i•ty | sim'plisitē |
noun

The Redis Manifesto

We're against complexity.

We believe **designing systems is a fight against complexity.**

Most of the time the best way to fight complexity is by not creating it at all.

Redis and NoSQL

What's wrong with RDBMS when used for (many) tasks that don't need all this complexity? The data model: non scalable, time complexity hard to predict, and can't model many common problems well enough.



Assault rifles as the crucial invention for new rules of warfare: fighting in shorter distances and shooting while on the run.

<http://www.youtube.com/watch?v=a1KBsqvKpXk#t=7m50s>

Memory is the new disk.
Disk is the new tape.

— Jim Gray

<http://www.infoq.com/news/2008/06/ram-is-disk>

http://research.microsoft.com/en-us/um/people/gray/talks/Flash_is_Good.ppt

Redis: Main Features

Simplicity

Predictability

Reliability

Speed

Versatility

Low Footprint



redis

Installation

```
git clone http://github.com/antirez/redis
```

```
cd redis
```

```
make
```

```
./src/redis-server
```

```
./src/redis-cli
```

The Zen of Redis

jimmybot writes:

19 Feb 10, 20:23:22

My understanding of redis is that there is no built-in easy index for *values*. No index means slow queries/searches. I think that is a very big difference as well, though not something that can't be changed.

4

antirez writes:

19 Feb 10, 20:24:45

@jimmybot: search Redis "Sorted Sets". It's our Index.

5

(...) what Redis provides are data structures (...)

Data Structures

Strings

Lists

Sets

Sorted Sets

Hashes

Strings

SET key **"value"** ~ 4 billion of keys

GET key

=> *"value"*

DEL key

Fetch multiple keys at once

```
SET key1 "value1"
```

```
SET key2 "value2"
```

```
MGET key1 key2
```

```
=> "value1"
```

```
=> "value2"
```

Expiration

```
EXPIRE key 5
```

```
GET key
```

```
=> "value"
```

```
TTL key
```

```
=> 1
```

```
GET key
```

```
=> (nil)
```

Usage

Cache

<http://antirez.com/post/redis-as-LRU-cache.html>

Sessions

<https://github.com/mattmatt/redis-session-store>

Atomic Increments

GET key

=> *nil*

INCR key

=> 1

INCR key

=> 2

GET key

=> 2

Usage

Counters (downloads, hits, votes, ...)

```
$ curl http://example.com/downloads/file1.mpg
```

```
INCR downloads:total
```

```
INCR downloads:total:today
```

```
INCR downloads:total:2011-05-10
```

```
INCR downloads:/downloads/file1.mpg:total
```

```
INCR downloads:/downloads/file1.mpg:today
```

```
INCR downloads:/downloads/file1.mpg:2011-05-10
```

Usage

Counters (downloads, hits, votes, ...)

Total downloads for server, all time

GET downloads:total

Total downloads for server, today

GET downloads:total:today

Total downloads for file

GET downloads:/downloads/file1.mpg:total

Total downloads for file today

INCR downloads:/downloads/file1.mpg:today

...

Usage

Counters (downloads, hits, votes, ...)

```
# Nightly maintenance at 23:59
RENAME downloads:total:today \
      downloads:total:yesterday
```

All this runs at super-sonic speed, with minimal overhead and resource consumption.

See implementation for Rubygems.org: <https://gist.github.com/296921>

However, you'll hit *denormalization bloat* once you start adding metrics (eg. downloads per country, per category, ...)

Usage

Variations: **Rate limiting**

```
$ curl http://api.example.com/list.json
```

```
INCR api:<TOKEN>:hits
```

```
=> 1
```

```
if INCR('api:abc123:hits') > LIMIT
  return 420 Enhance Your Calm
end
```

```
# Every hour...
```

```
DEL api:<TOKEN>:hits
```

Usage

Generating unique IDs

```
INCR global:users_ids
```

```
=> 1
```

```
SET users:1:username "john"
```

```
INCR global:users_ids
```

```
=> 2
```

```
SET users:2:username "mary"
```

Lists

LPUSH key **1**

=> **1**

LPUSH key **2**

=> **2**

LPUSH key **3**

=> **3**

LRANGE key **0 -1**

=> **"3"**

=> **"2"**

=> **"1"**

RPOP key

=> **"1"**

LRANGE key **0 -1**

=> **"3"**

=> **"2"**

LLEN key

=> **2**

LTRIM key **0 1**

=> **OK**

Usage

Indexes (list of comments, ...)

```
LPUSH article:comments <ID>
```

Timelines (of all sorts: messages, logs, ...)

```
LPUSH user:<ID>:inbox "message from Alice"
```

```
LPUSH user:<ID>:inbox "message from Bob"
```

```
# Limit the messages to 100
```

```
LTRIM user:<ID>:inbox 0 99
```

```
# Get last 10 messages
```

```
LRANGE user:<ID>:inbox 0 9
```

```
# Get next 10 messages
```

```
LRANGE user:<ID>:inbox 10 19
```

Usage

Queues

Publisher

```
RPUSH queue "task-1"
```

```
RPUSH queue "task-2"
```

Worker (blocks and waits for tasks)

```
BLPOP queue 0
```

Usage

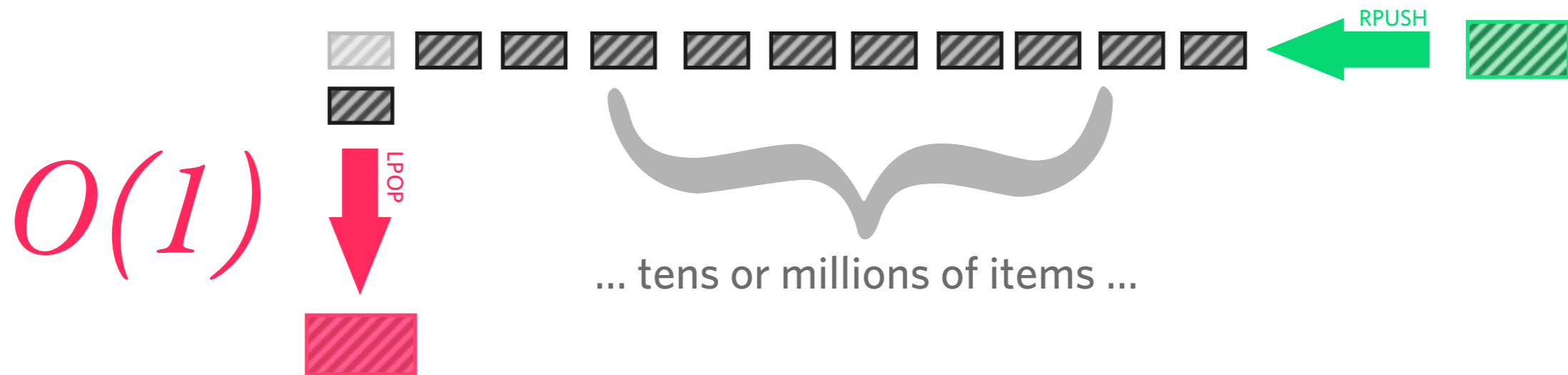
Queues

```
# publisher.sh
for i in {1..10}; do
  redis-cli RPush "queue" "task-$i"
done
```

```
# worker.sh
while true; do
  redis-cli BLPop "queue" 0
done
```

[Demo](#)

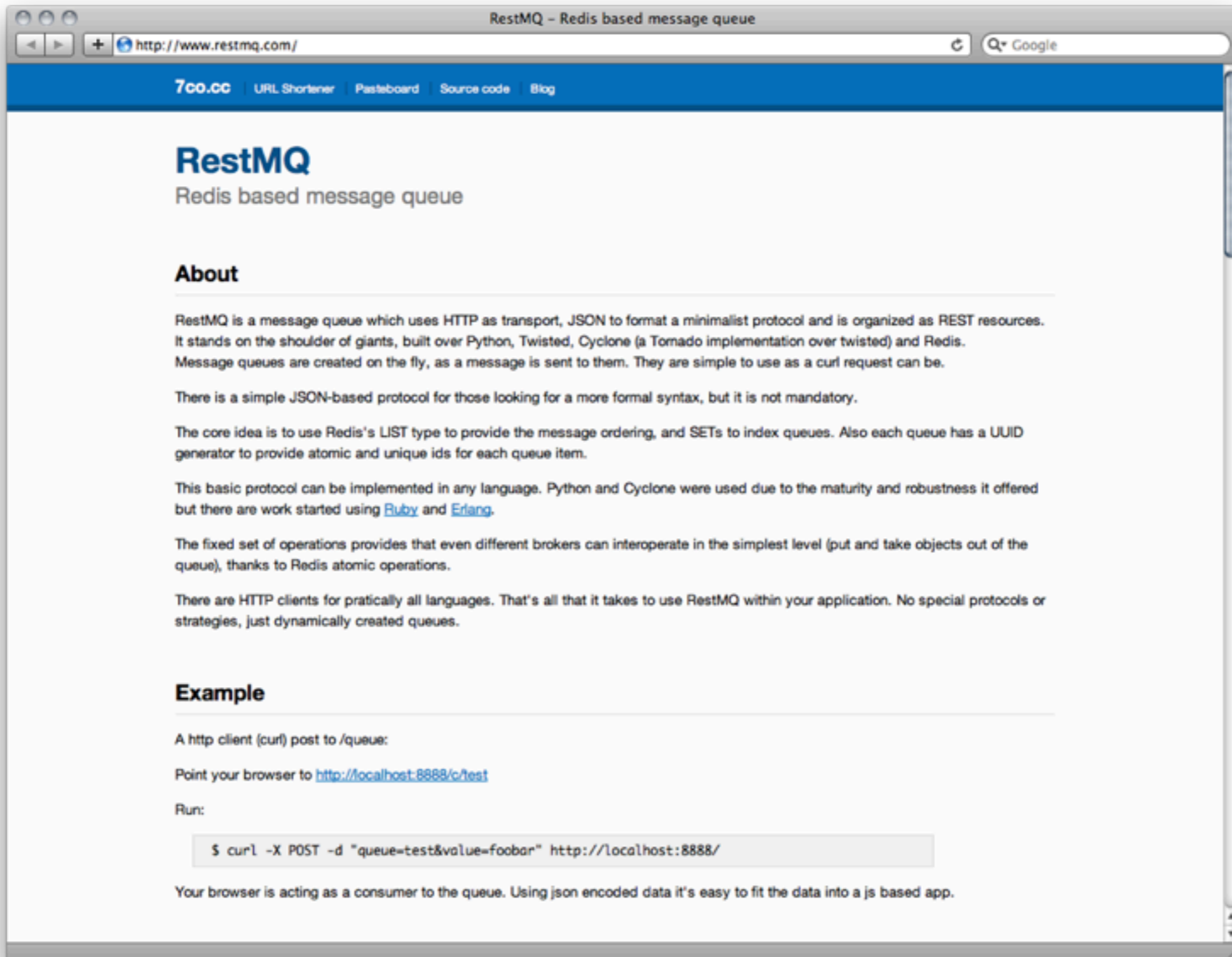
Resque: Background Processing from Github



<https://github.com/defunkt/resque/blob/v1.13.0/lib/resque.rb#L133-138>

```

133 # Pops a job off a queue. Queue name should be a string.
134 #
135 # Returns a Ruby object.
136 def pop(queue)
137   decode redis.lpop("queue:#{queue}")
138 end
    
```



The Code of the Forking Paths

The screenshot shows a presentation slide titled "REDIS Jak to funguje?". The slide features a diagram of a list of "Millions of items" represented by a horizontal row of small squares. A green arrow labeled "PUSH" points to the right, indicating the direction of data flow. Below the list, a red arrow labeled "POP" points downwards, indicating the removal of an item. The text $O(1)$ is written in red next to the POP arrow. The slide is part of a presentation on asynchronous processing with Resque and AMQP.

Slides
<http://www.slideshare.net/karmi/the-code-of-the-forking-paths-asynchronous-processing-with-resque-and-amqp>

Video (in Czech)
<http://multimedia.vse.cz/media/Viewer/?peid=51c06c512f4645289c4e9c749dc85acc1d>

Sets

SADD key 1

=> 1

SADD key 2

=> 2

SADD key 3

=> 3

SMEMBERS key

=> "3"

=> "1"

=> "2"

SISMEMBER key 1

=> "1"

SISMEMBER key 5

=> "0"

SRANDMEMBER key

=> "<RAND>"

SREM key 3

=> 1

Set Operations

```
SADD A 1
```

```
SADD A 2
```

```
SMEMBERS A
```

```
=> "1"
```

```
=> "2"
```

```
SADD B 1
```

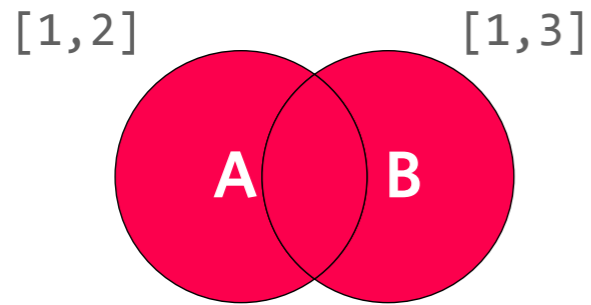
```
SADD B 3
```

```
SMEMBERS B
```

```
=> "1"
```

```
=> "3"
```

Set Operations



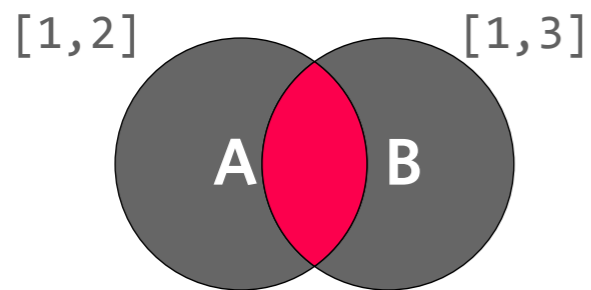
Union

SUNION A B

=> 1

=> 2

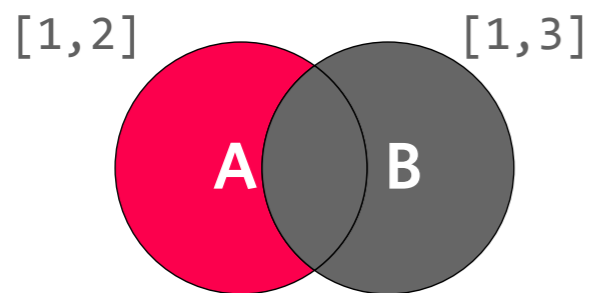
=> 3



Intersection

SINTER A B

=> 1



Difference

SDIFF A B

=> 2

Usage

Ad serving

```
SADD ads:cars "Check out Toyota!"
```

```
SADD ads:cars "Check out Ford!"
```

```
...
```

```
SADD ads:movies "Check out Winter's Bone!"
```

```
...
```

```
SRANDMEMBER ads:cars
```

```
SRANDMEMBER ads:movies
```

Note: Time complexity is $O(1)$. "Check out ORDER BY RAND()!"

Usage

Relations (Friends/followers)

```
SADD users:A:follows B
```

```
SADD users:B:follows C
```

```
SADD users:B:follows D
```

```
SADD users:C:follows A
```

```
SADD users:C:follows D
```

Usage

Relations (Friends/followers)

Joint network of A and B

SUNION users:A:follows users:B:follows

1) "C"

2) "D"

3) "B"

Usage

Relations (Friends/followers)

Common for A and B

```
SINTER users:A:follows users:B:follows
```

[]

Common for B and C

```
SINTER users:B:follows users:C:follows
```

1) "D"

Unique to B compared to C

```
SDIFF users:B:follows users:C:follows
```

1) "C"

Usage

Relations (Friends/followers)

Whom I follow...

```
SADD friends A
```

```
SADD friends B
```

```
SMEMBERS friends
```

```
1) "A"
```

```
2) "B"
```

Who follows me...

```
SADD followers B
```

```
SADD followers C
```

```
SMEMBERS followers
```

```
1) "C"
```

```
2) "B"
```

Usage

Relations (Friends/followers)

Mutual relationships

SINTER friends followers

1) "B"

Who does not follow me back?

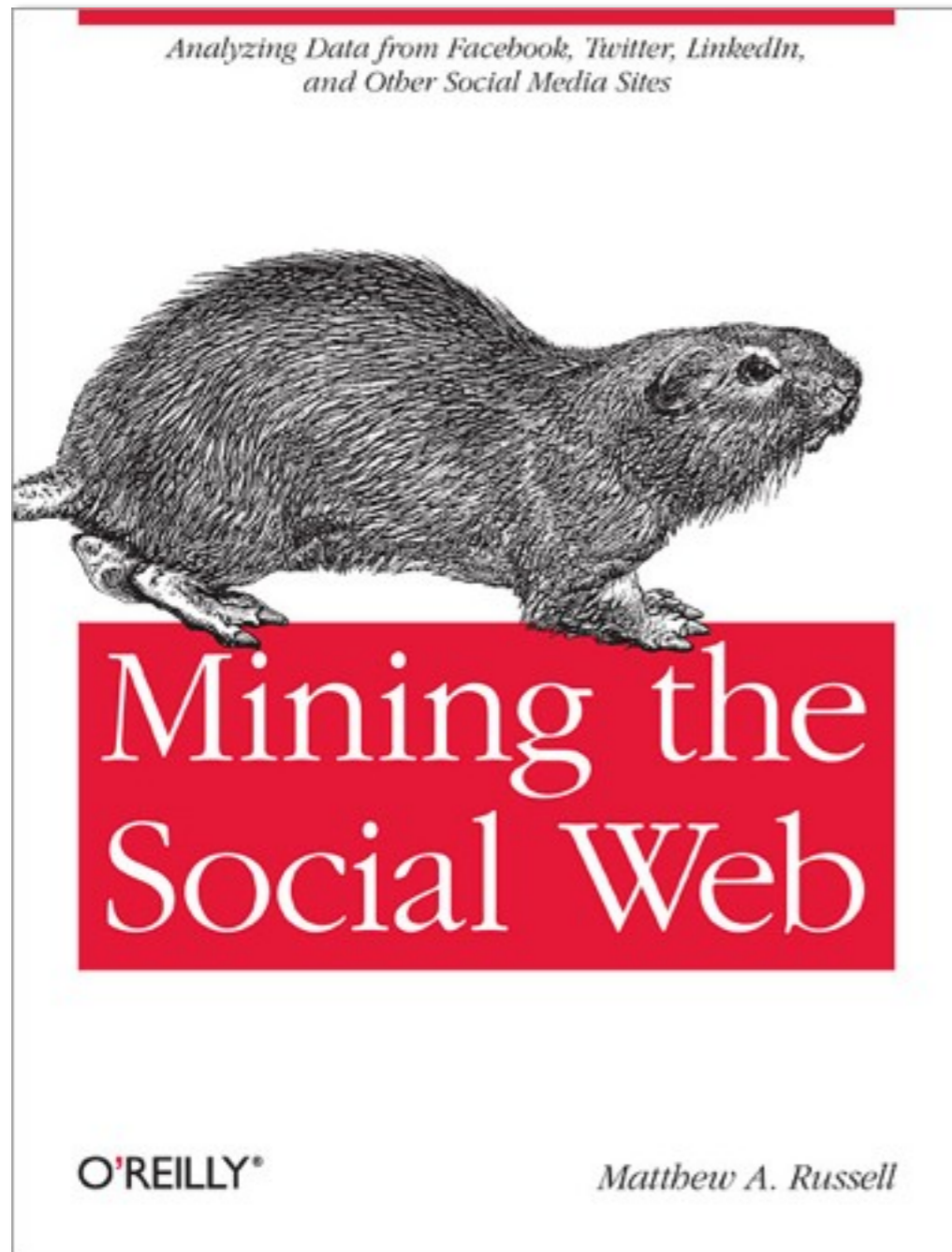
SDIFF friends followers

1) "A"

Who am I not following back?

SDIFF followers friends

1) "C"



Mining the Social Web

Analyzing Data from Facebook, Twitter, LinkedIn, and Other Social Media Sites

Matthew A. Russell

<http://books.google.com/books?id=SYM1rOdrdsC&lpg=PP1&pg=PA94>

Usage

Relations (Article tags/categories)

```
SADD tags:ruby article-1
```

```
SADD tags:java article-2
```

```
SADD tags:web article-1
```

```
SADD tags:web article-2
```

Usage

Relations (Article tags/categories)

ruby OR java

SUNION tags:ruby tags:java

1) *"article-2"*

2) *"article-1"*

ruby AND java

SINTER tags:ruby tags:java

[]

web AND NOT ruby

SDIFF tags:web tags:ruby

1) *"article-2"*

Usage

Friends Online

My friends

SADD friends A

SADD friends B

SADD friends C

Friend A connects

SADD online:fresh A

SUNIONSTORE **online** online:fresh online:stale

Who's online now?

SINTER friends **online**

[A]

Usage

Friends Online

Every minute, rename the "fresh" to "stale" ...

```
RENAME online:fresh online:stale
```

... and update the "online" set

```
SUNIONSTORE online online:fresh online:stale
```

Friend B connects

```
SADD online:fresh B
```

```
SUNIONSTORE online online:fresh online:stale
```

Who's online now?

```
SINTER friends online
```

```
[A,B]
```

Usage

Friends Online

Time passes ...

Rename the "fresh" to "stale", every minute ...

```
RENAME online:fresh online:stale
```

... and update the "online" set

```
SUNIONSTORE online online:fresh online:stale
```

Who's online now?

```
SINTER friends online
```

[B]

Sorted Sets

ZADD key 100 A

ZADD key 10 C

ZADD key 80 B

ZREVRANGE key 0 -1

1) "A"

2) "B"

3) "C"

ZRANGE key 0 -1

1) "C"

2) "B"

3) "A"

ZINCRBY key 10 C

"20"

Sorted Sets

ZREVRANGE key 0 -1
WITHSCORES

- 1) "A"
- 2) "100"
- 3) "B"
- 4) "80"
- 5) "C"
- 6) "20"

ZREVRANGEBYSCORE
 key 100 50

- 1) "A"
- 2) "B"

Usage

Leaderboards

User A got 10 points

ZINCRBY scores 10 A

User B got 15 points

ZINCRBY scores 15 B

User A got *another* 10 points

ZINCRBY scores 10 A

Display scores

ZREVRANGE scores 0 -1 WITHSCORES

1) "A"

2) "20"

3) "B"

4) "15"

Usage

Inverted Index

Index document A

```
ZINCRBY index:foo 1 document-A
```

```
ZINCRBY index:foo 1 document-A
```

```
ZINCRBY index:bar 1 document-A
```

Index document B

```
ZINCRBY index:foo 1 document-B
```

```
ZINCRBY index:baz 1 document-B
```

Search for token foo, sort by occurrences

```
ZREVRANGE index:foo 0 -1 WITHSCORES
```

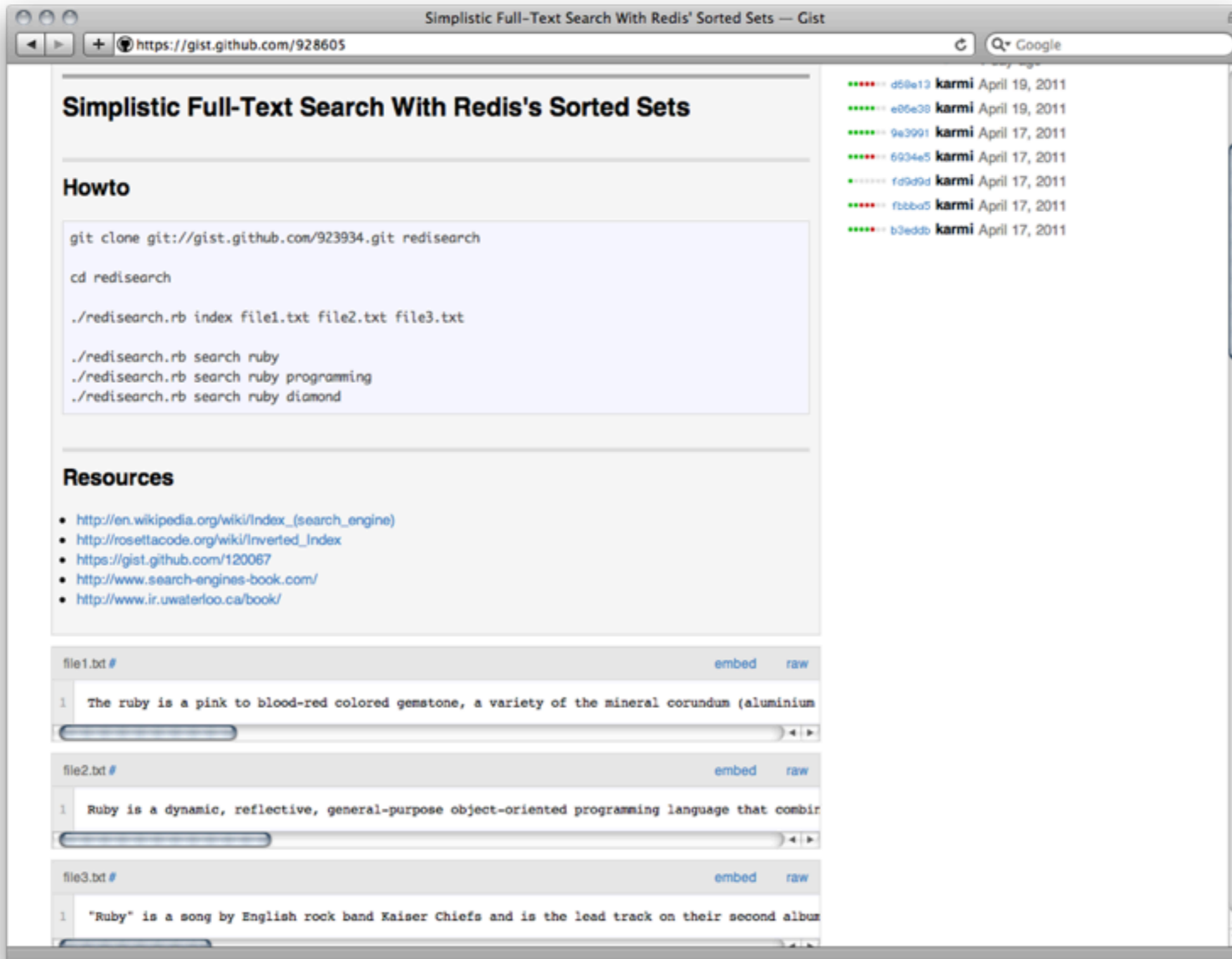
1) "document-A"

2) "2"

3) "document-B"

4) "1"

Inverted Index



Hashes

```
HMSET users:1 username j name John
```

```
HMSET users:2 username m name Mary
```

```
HGETALL users:1
```

```
1) "username"
```

```
2) "j"
```

```
...
```

```
HKEYS users:1
```

```
1) "username"
```

```
2) "name"
```

```
HSET users:1 score 100
```

```
HGET users:1 score
```

```
1) "100"
```

Usage

Structured data (Articles, users, ...)

```
HMSET articles:1 title "Redis is cool!" \  
content "I recently ..." \  
published "2011-05-10"
```

HGETALL articles:1

- 1) "title"
- 2) "Redis is cool!"
- 3) "content"
- 4) "I recently ..."
- 5) "published"
- 6) "2011-05-10"

```
HSET articles:1 title "Redis is very cool!"
```

```
HGET articles:1 title
```

```
"Redis is very cool!"
```

Usage

User Preferences (No login)

Save preferences from <FORM>

```
HMSET prefs:<COOKIE HASH> background #ccc color #333
```

Keep it for one year

```
EXPIRE prefs:<COOKIE HASH> 31556926
```

Retrieve preferences

```
HGETALL prefs:<COOKIE HASH>
```


Publish/Subscribe

SUBSCRIBE `log.error`
PUBLISH `log.error "ERROR"`

PSUBSCRIBE `log.*` ← "AMQP"
PUBLISH `log.error "ERROR"`

=> "ERROR"
 => "ERROR"

PUBLISH `log.info "INFO"`
 => "INFO"

Demo

Durability

BGSAVE

```
/usr/local/var/db/redis/dump.rdb
```

BGREWRITEAOF

```
# redis.conf
```

```
appendonly yes
```

Virtual Memory

```
# redis.conf  
vm-enabled yes
```

Allows to work with data sets bigger than available RAM.
Swaps less often used *values* to disk.
Keys must still fit into RAM.

```
# Make Redis disk-bound database  
vm-max-memory 0
```

The future? *Redis Diskstore* (disk-bound by default, cache in between server and disk)

Replication

```
# slave.conf  
slaveof 127.0.0.1 6379  
$ redis-server slave.conf
```

The future? Redis Cluster (*Dynamo-like, Distributed hash table*)

Scripting (experimental)

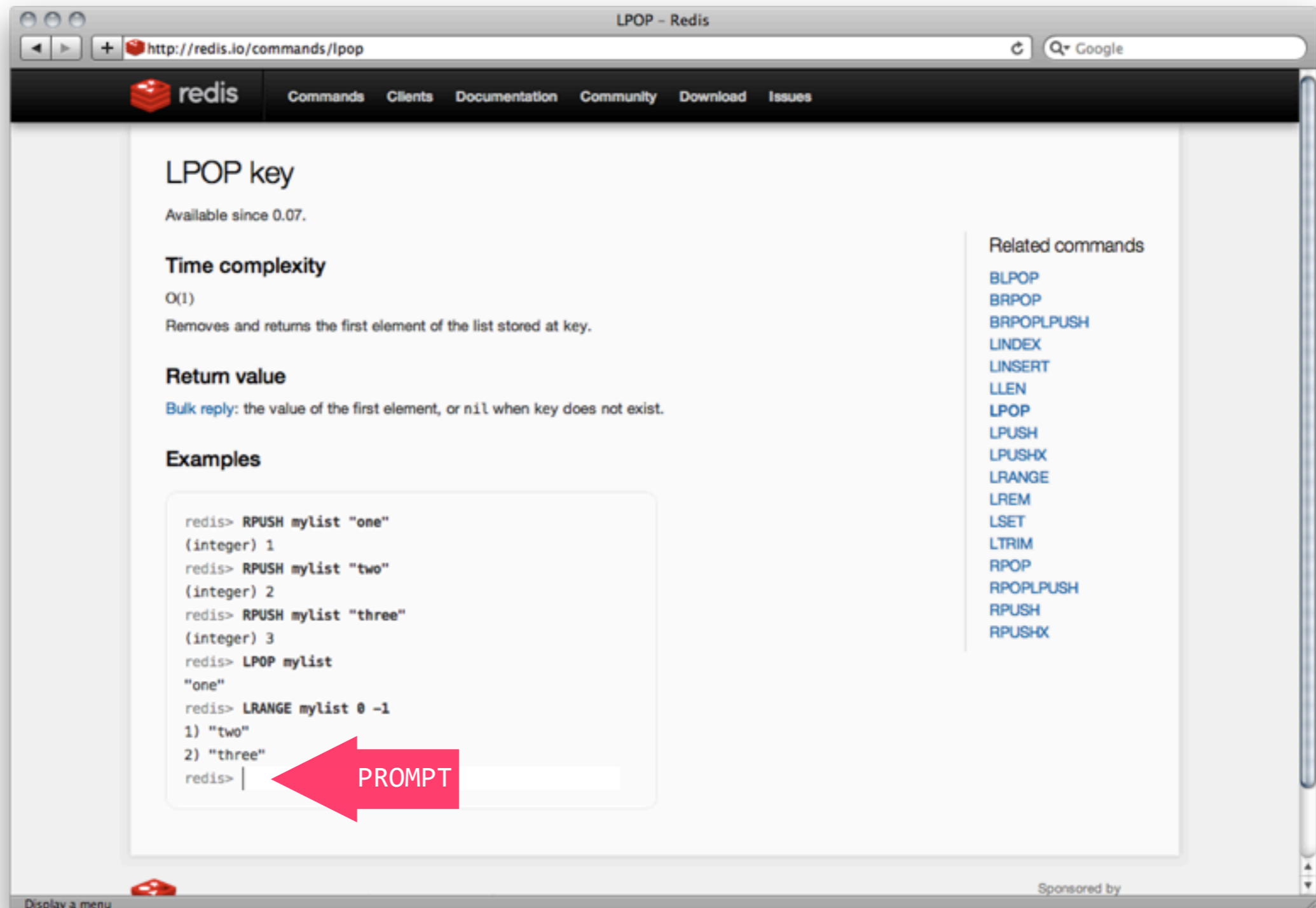
```
$ git clone https://github.com/antirez/redis -b scripting redis-scripting
$ cd redis-scripting
$ make
$ echo port 7397 | ./src/redis-server -
$ ./src/redis-cli
```

```
SET myscript "return 'HELLO'"
```

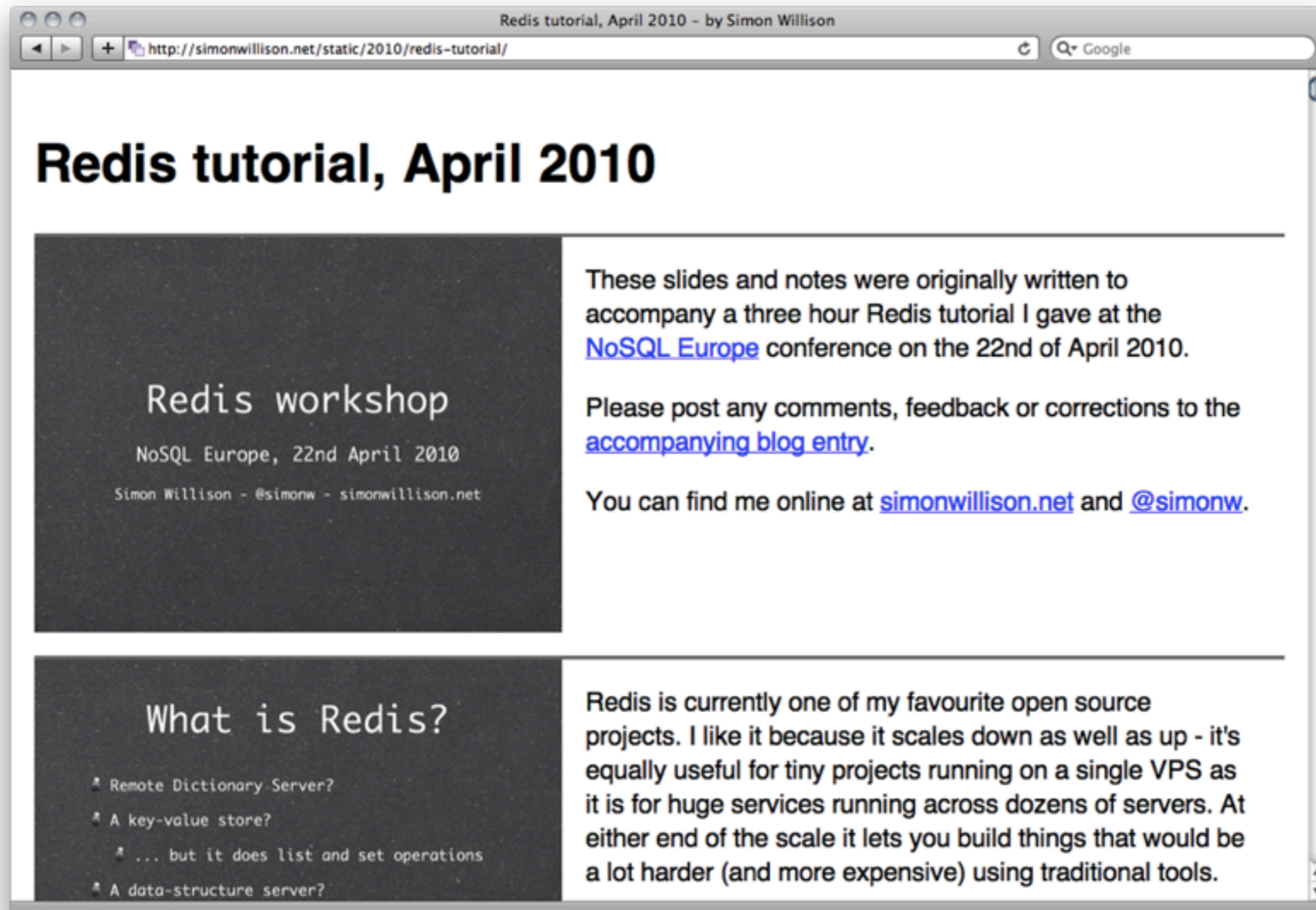
```
EVAL "return loadstring(redis('get', KEYS[1]))()" \
      1 myscript
```

```
=> "HELLO"
```

The Interactive Documentation



Simon Willison Tutorial



The screenshot shows a web browser window with the title "Redis tutorial, April 2010 - by Simon Willison". The address bar contains the URL "http://simonwillison.net/static/2010/redis-tutorial/". The page content is as follows:

Redis tutorial, April 2010

Redis workshop

NoSQL Europe, 22nd April 2010

Simon Willison - @simonw - simonwillison.net

These slides and notes were originally written to accompany a three hour Redis tutorial I gave at the [NoSQL Europe](#) conference on the 22nd of April 2010.

Please post any comments, feedback or corrections to the [accompanying blog entry](#).

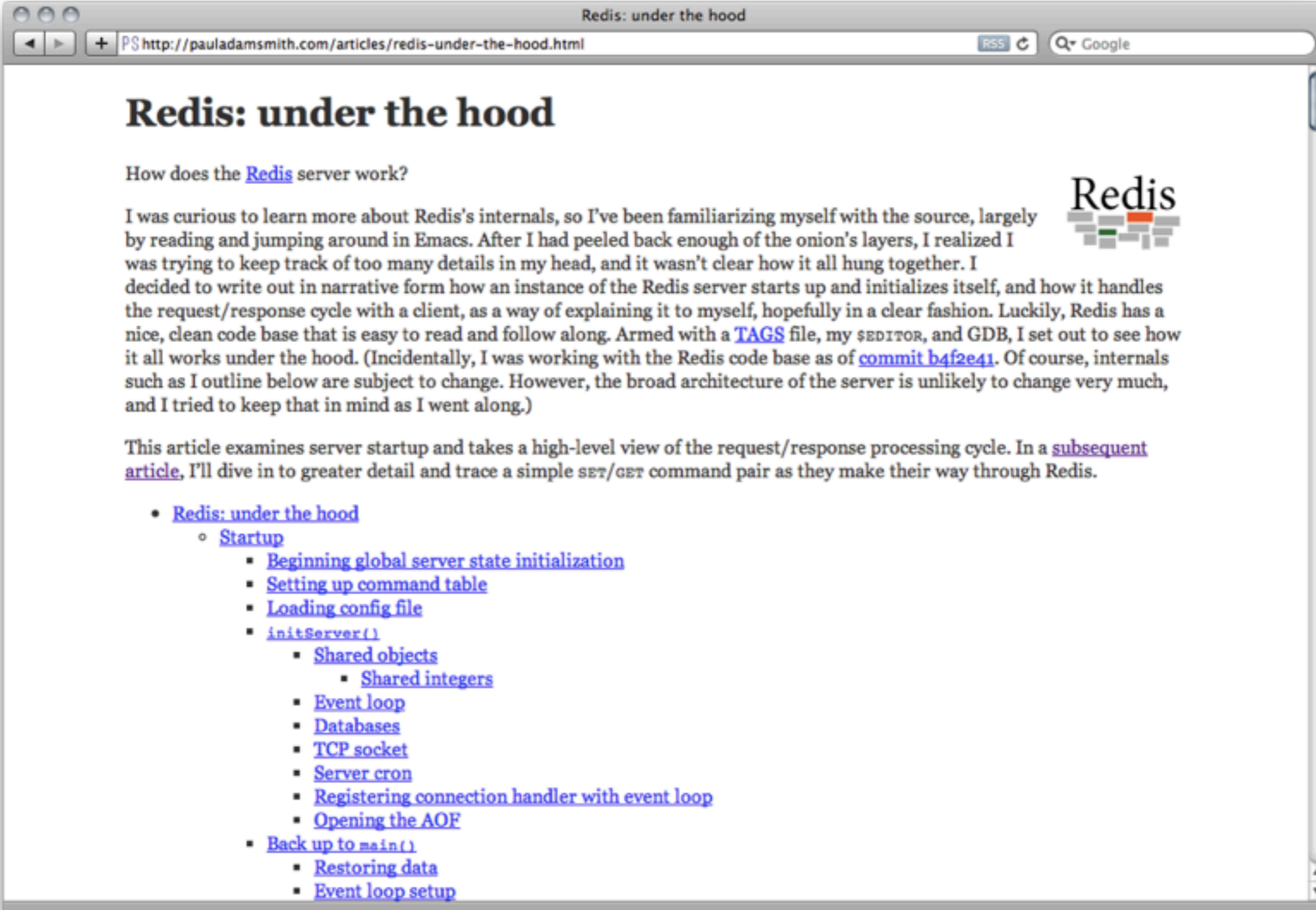
You can find me online at [simonwillison.net](#) and [@simonw](#).

What is Redis?

- Remote Dictionary Server?
- A key-value store?
- ... but it does list and set operations
- A data-structure server?

Redis is currently one of my favourite open source projects. I like it because it scales down as well as up - it's equally useful for tiny projects running on a single VPS as it is for huge services running across dozens of servers. At either end of the scale it lets you build things that would be a lot harder (and more expensive) using traditional tools.

Redis Implementation Details



The screenshot shows a web browser window with the title "Redis: under the hood". The address bar contains the URL "http://pauladamsmith.com/articles/redis-under-the-hood.html". The page content includes the title "Redis: under the hood", a sub-header "How does the Redis server work?", and a paragraph of text. To the right of the text is the Redis logo. Below the text is a list of links, including "Redis: under the hood" and a sub-list under "Startup" with items like "Beginning global server state initialization", "Setting up command table", "Loading config file", "initServer()", "Shared objects", "Event loop", "Databases", "TCP socket", "Server cron", "Registering connection handler with event loop", "Opening the AOF", "Back up to main()", "Restoring data", and "Event loop setup".

Redis: under the hood

How does the [Redis](#) server work?

I was curious to learn more about Redis's internals, so I've been familiarizing myself with the source, largely by reading and jumping around in Emacs. After I had peeled back enough of the onion's layers, I realized I was trying to keep track of too many details in my head, and it wasn't clear how it all hung together. I decided to write out in narrative form how an instance of the Redis server starts up and initializes itself, and how it handles the request/response cycle with a client, as a way of explaining it to myself, hopefully in a clear fashion. Luckily, Redis has a nice, clean code base that is easy to read and follow along. Armed with a [TAGS](#) file, my `EDITOR`, and GDB, I set out to see how it all works under the hood. (Incidentally, I was working with the Redis code base as of [commit b4f2e41](#). Of course, internals such as I outline below are subject to change. However, the broad architecture of the server is unlikely to change very much, and I tried to keep that in mind as I went along.)

This article examines server startup and takes a high-level view of the request/response processing cycle. In a [subsequent article](#), I'll dive in to greater detail and trace a simple `SET/GET` command pair as they make their way through Redis.

- [Redis: under the hood](#)
 - [Startup](#)
 - [Beginning global server state initialization](#)
 - [Setting up command table](#)
 - [Loading config file](#)
 - [initServer\(\)](#)
 - [Shared objects](#)
 - [Shared integers](#)
 - [Event loop](#)
 - [Databases](#)
 - [TCP socket](#)
 - [Server cron](#)
 - [Registering connection handler with event loop](#)
 - [Opening the AOF](#)
 - [Back up to main\(\)](#)
 - [Restoring data](#)
 - [Event loop setup](#)

Redis Use Cases

High Scalability - High Scalability - What the heck are you actually using NoSQL for?

http://highscalability.com/blog/2010/12/6/what-the-heck-are-you-actually-using-nosql-for.html

High Scalability
Building bigger, faster, more reliable websites.

HOME START HERE REAL LIFE ARCHITECTURES ALL POSTS ADVERTISING CONTACT BOOK STORE RSS

« Sponsored Post: Joyent, Membase, Appirio, CloudSigma, ManageEngine, Site24x7 | Main | GPU vs CPU Smackdown : The Rise of Throughput-Oriented Architectures »

What The Heck Are You Actually Using NoSQL For?

MONDAY, DECEMBER 6, 2010 AT 9:34AM

It's a truism that we should choose the *right tool for the job*. Everyone says that. And who can disagree? The problem is this is not helpful advice without being able to answer more specific questions like: What jobs are the tools good at? Will they work on jobs like mine? Is it worth the risk to try something new when all my people know something else and we have a deadline to meet? How can I make all the tools work together?

In the NoSQL space this kind of real-world data is still a bit vague. When asked, vendors tend to give very general answers like NoSQL is good for BigData or key-value access. What does that mean for the developer in the trenches faced with the task of solving a specific problem and there are a dozen confusing choices and no obvious winner? Not a lot. It's often hard to take that next step and imagine how their specific problems could be solved in a way that's worth taking the trouble and risk.

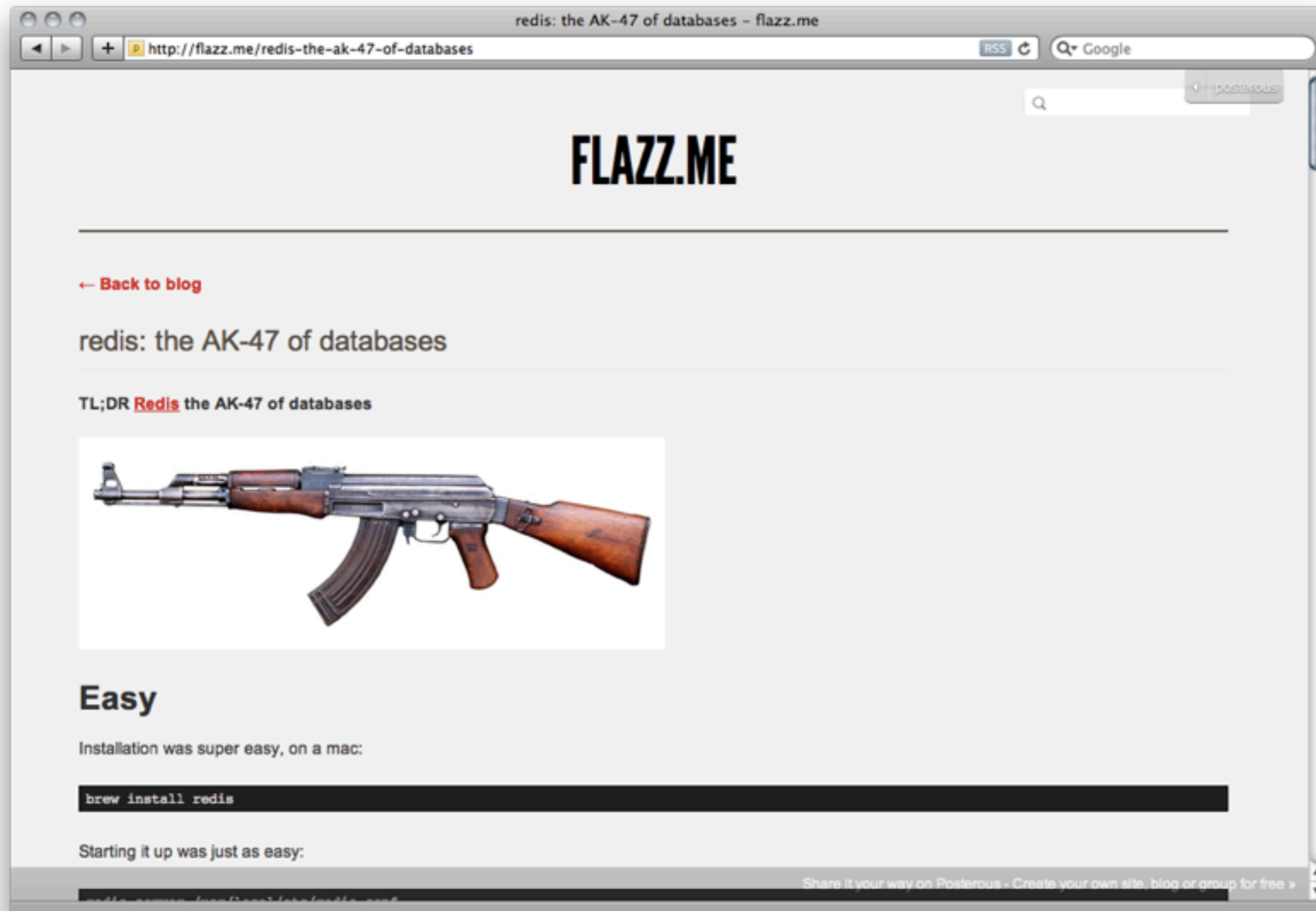
Let's change that. What problems are you using NoSQL to solve? Which product are you using? How is it helping you? Yes, this is part the research for my [webinar on December 14th](#), but I'm a huge

Redis Use Cases

Redis is unique in the repertoire as it is a data structure server, with many fascinating use cases that people are excited to share.

- Calculating whose friends are online using sets.
- Memcached on steroids.
- Distributed lock manager for process coordination.
- Full text inverted index lookups.
- Tag clouds.
- Leaderboards. Sorted sets for maintaining high score tables.
- Circular log buffers.
- Database for university course availability information. If the set contains the course ID it has an open seat. Data is scraped and processed continuously and there are ~7200 courses.
- Server for backed sessions. A random cookie value which is then associated with a larger chunk of serialized data on the server) are a very poor fit for relational databases. They are often created for every visitor, even those who stumble in from Google and then leave, never to return again. They then hang around for weeks taking up valuable database space. They are never queried by anything other than their primary key.
- Fast, atomically incremented counters are a great fit for offering real-time statistics.
- Polling the database every few seconds. Cheap in a key-value store. If you're sharding your data you'll need a central lookup service for quickly determining which shard is being used for a specific user's data. A replicated Redis cluster is a great solution here - GitHub use exactly that to manage sharding their many repositories between different backend file servers.
- Transient data. Any transient data used by your application is also a good fit for Redis. **CSRF tokens** (to prove a POST submission came from a form you served up, and not a form on a malicious third party site, need to be stored for a short while, as does handshake data for various security protocols.
- Incredibly easy to set up and ridiculously fast (30,000 read or writes a second on a laptop with

The Original Metaphore...



Thanks!

