

Using the Infinispan Command Line Interface

Table of Contents

1. Getting Started with the Infinispan CLI	1
1.1. Creating and Modifying Users	1
1.1.1. Adding Credentials	1
1.1.2. Assigning Roles to Users	2
1.1.3. Adding Users to Groups	2
1.1.4. User Roles and Permissions	3
1.2. Connecting to Infinispan Servers	4
1.3. Navigating CLI Resources	5
1.3.1. CLI Resources	6
1.4. Shutting Down Infinispan Server	7
1.4.1. Restarting Infinispan Clusters	8
2. Performing Cache Operations with the Infinispan CLI	9
2.1. Creating Caches with the Infinispan Command Line Interface (CLI)	9
2.1.1. Cache Configuration	10
2.2. Adding Cache Entries	10
2.3. Clearing Caches and Deleting Entries	11
2.4. Deleting Caches	11
3. Performing Batch Operations	12
3.1. Performing Batch Operations with Files	12
3.2. Performing Batch Operations Interactively	12
4. Configuring the Infinispan CLI	14
4.1. Setting Infinispan CLI Properties and Persistent Storage	14
4.2. Creating Command Aliases	15
4.3. Trusting Infinispan Server Connections	15
4.4. Infinispan CLI Storage Directory	16
5. Working with Counters	17
5.1. Creating Counters	17
5.2. Adding Deltas to Counters	18
6. Querying Caches with Protobuf Metadata	20
6.1. Configuring Media Types	20
6.2. Registering Protobuf Schemas	21
6.3. Querying Caches with Protobuf Schemas	22
7. Performing Cross-Site Replication Operations	26
7.1. Bringing Backup Locations Offline and Online	26
7.2. Configuring Cross-Site State Transfer Modes	26
7.3. Pushing State to Backup Locations	27
8. Backing Up and Restoring Infinispan Clusters	28
8.1. Backing Up Infinispan Clusters	28

8.2. Restoring Infinispan Clusters from Backup Archives	29
9. Command Reference	31
9.1. ADD(1)	31
9.1.1. NAME	31
9.1.2. SYNOPSIS	31
9.1.3. OPTIONS	31
9.1.4. EXAMPLES	31
9.1.5. SEE ALSO	31
9.2. ALIAS(1)	31
9.2.1. NAME	31
9.2.2. SYNOPSIS	32
9.2.3. EXAMPLES	32
9.2.4. SEE ALSO	32
9.3. BACKUP(1)	32
9.3.1. NAME	32
9.3.2. SYNOPSIS	32
9.3.3. BACKUP CREATE OPTIONS	32
9.3.4. BACKUP GET OPTIONS	33
9.3.5. BACKUP RESTORE OPTIONS	33
9.3.6. EXAMPLES	33
9.3.7. SEE ALSO	34
9.4. BENCHMARK(1)	34
9.4.1. NAME	34
9.4.2. SYNOPSIS	34
9.4.3. BENCHMARK OPTIONS	35
9.4.4. EXAMPLES	35
9.5. CACHE(1)	36
9.5.1. NAME	36
9.5.2. SYNOPSIS	36
9.5.3. EXAMPLE	36
9.5.4. SEE ALSO	36
9.6. CAS(1)	36
9.6.1. NAME	36
9.6.2. SYNOPSIS	36
9.6.3. OPTIONS	36
9.6.4. EXAMPLE	37
9.6.5. SEE ALSO	37
9.7. CD(1)	37
9.7.1. NAME	37
9.7.2. DESCRIPTION	37
9.7.3. SYNOPSIS	37

9.7.4. EXAMPLE	37
9.7.5. SEE ALSO	37
9.8. CLEARCACHE(1)	37
9.8.1. NAME	37
9.8.2. SYNOPSIS	37
9.8.3. EXAMPLES	37
9.8.4. SEE ALSO	38
9.9. CONFIG(1)	38
9.9.1. NAME	38
9.9.2. SYNOPSIS	38
9.9.3. DESCRIPTION	38
9.9.4. COMMAND SYNOPSIS	38
9.9.5. COMMON OPTIONS	38
9.9.6. PROPERTIES	38
9.9.7. EXAMPLES	39
9.9.8. SEE ALSO	39
9.10. CONNECT(1)	39
9.10.1. NAME	39
9.10.2. DESCRIPTION	39
9.10.3. SYNOPSIS	39
9.10.4. OPTIONS	40
9.10.5. EXAMPLE	40
9.10.6. SEE ALSO	40
9.11. CONTAINER(1)	40
9.11.1. NAME	40
9.11.2. SYNOPSIS	40
9.11.3. EXAMPLE	40
9.11.4. SEE ALSO	40
9.12. COUNTER(1)	40
9.12.1. NAME	40
9.12.2. SYNOPSIS	40
9.12.3. EXAMPLE	41
9.12.4. SEE ALSO	41
9.13. CREATE(1)	41
9.13.1. NAME	41
9.13.2. SYNOPSIS	41
9.13.3. CREATE CACHE OPTIONS	41
9.13.4. CREATE COUNTER OPTIONS	41
9.13.5. EXAMPLES	42
9.13.6. SEE ALSO	42
9.14. CREDENTIALS(1)	42

9.14.1. NAME	42
9.14.2. SYNOPSIS	42
9.14.3. DESCRIPTION	42
9.14.4. SYNOPSIS	42
9.14.5. OPTIONS	43
9.14.6. CREDENTIALS ADD OPTIONS	43
9.14.7. EXAMPLES	43
9.15. DESCRIBE(1)	43
9.15.1. NAME	43
9.15.2. SYNOPSIS	43
9.15.3. EXAMPLES	44
9.15.4. SEE ALSO	44
9.16. DISCONNECT(1)	44
9.16.1. NAME	44
9.16.2. SYNOPSIS	44
9.16.3. EXAMPLE	44
9.16.4. SEE ALSO	44
9.17. DROP(1)	44
9.17.1. NAME	44
9.17.2. SYNOPSIS	44
9.17.3. EXAMPLES	45
9.17.4. SEE ALSO	45
9.18. ENCODING(1)	45
9.18.1. NAME	45
9.18.2. DESCRIPTION	45
9.18.3. SYNOPSIS	45
9.18.4. EXAMPLE	45
9.18.5. SEE ALSO	45
9.19. GET(1)	45
9.19.1. NAME	46
9.19.2. SYNOPSIS	46
9.19.3. OPTIONS	46
9.19.4. EXAMPLE	46
9.19.5. SEE ALSO	46
9.20. HELP(1)	46
9.20.1. NAME	46
9.20.2. SYNOPSIS	46
9.20.3. EXAMPLE	46
9.20.4. SEE ALSO	46
9.21. LOGGING(1)	46
9.21.1. NAME	46

9.21.2. SYNOPSIS	47
9.21.3. LOGGING SET OPTIONS	47
9.21.4. EXAMPLES	47
9.22. LS(1)	47
9.22.1. NAME	47
9.22.2. SYNOPSIS	47
9.22.3. EXAMPLES	47
9.22.4. SEE ALSO	48
9.23. MIGRATE(1)	48
9.23.1. NAME	48
9.23.2. SYNOPSIS	48
9.23.3. DESCRIPTION	48
9.23.4. COMMAND SYNOPSIS	48
9.23.5. COMMON OPTIONS	48
9.23.6. CLUSTER SYNCHRONIZE OPTIONS	48
9.23.7. CLUSTER DISCONNECT OPTIONS	49
9.24. PATCH(1)	49
9.24.1. NAME	49
9.24.2. DESCRIPTION	49
9.24.3. SYNOPSIS	49
9.24.4. PATCH LIST OPTIONS	49
9.24.5. PATCH INSTALL OPTIONS	49
9.24.6. PATCH DESCRIBE OPTIONS	49
9.24.7. PATCH ROLLBACK OPTIONS	50
9.24.8. PATCH CREATE OPTIONS	50
9.24.9. EXAMPLES	50
9.25. PUT(1)	50
9.25.1. NAME	50
9.25.2. DESCRIPTION	50
9.25.3. SYNOPSIS	51
9.25.4. OPTIONS	51
9.25.5. EXAMPLES	51
9.25.6. SEE ALSO	51
9.26. QUERY(1)	51
9.26.1. NAME	51
9.26.2. SYNOPSIS	52
9.26.3. OPTIONS	52
9.26.4. EXAMPLES	52
9.26.5. SEE ALSO	52
9.27. QUIT(1)	52
9.27.1. NAME	52

9.27.2. SYNOPSIS	52
9.27.3. EXAMPLE	52
9.27.4. SEE ALSO	53
9.28. REMOVE(1)	53
9.28.1. NAME	53
9.28.2. SYNOPSIS	53
9.28.3. OPTIONS	53
9.28.4. EXAMPLE	53
9.28.5. SEE ALSO	53
9.29. RESET(1)	53
9.29.1. NAME	53
9.29.2. SYNOPSIS	53
9.29.3. EXAMPLE	53
9.29.4. SEE ALSO	54
9.30. SCHEMA(1)	54
9.30.1. NAME	54
9.30.2. SYNOPSIS	54
9.30.3. OPTIONS	54
9.30.4. EXAMPLE	54
9.30.5. SEE ALSO	54
9.31. SERVER(1)	54
9.31.1. NAME	54
9.31.2. DESCRIPTION	54
9.31.3. SYNOPSIS	54
9.31.4. SERVER CONNECTOR IPFILTER OPTIONS	55
9.31.5. EXAMPLES	55
9.32. SHUTDOWN(1)	56
9.32.1. NAME	56
9.32.2. SYNOPSIS	56
9.32.3. EXAMPLES	56
9.32.4. SEE ALSO	56
9.33. SITE(1)	56
9.33.1. NAME	56
9.33.2. SYNOPSIS	56
9.33.3. OPTIONS	57
9.33.4. STATE TRANSFER MODE OPTIONS	57
9.33.5. EXAMPLES	57
9.34. STATS(1)	58
9.34.1. NAME	58
9.34.2. SYNOPSIS	58
9.34.3. EXAMPLES	58

9.34.4. SEE ALSO	58
9.35. TASK(1)	58
9.35.1. NAME	58
9.35.2. SYNOPSIS	58
9.35.3. EXAMPLES	59
9.35.4. OPTIONS	59
9.35.5. SEE ALSO	59
9.36. UNALIAS(1)	59
9.36.1. NAME	59
9.36.2. SYNOPSIS	59
9.36.3. EXAMPLES	59
9.36.4. SEE ALSO	59
9.37. USER(1)	59
9.37.1. NAME	59
9.37.2. SYNOPSIS	60
9.37.3. DESCRIPTION	60
9.37.4. COMMAND SYNOPSIS	60
9.37.5. COMMON OPTIONS	61
9.37.6. USER CREATE/MODIFY OPTIONS	61
9.37.7. USER LS OPTIONS	61
9.37.8. USER ENCRYPT-ALL OPTIONS	62
9.38. VERSION(1)	62
9.38.1. NAME	62
9.38.2. SYNOPSIS	62
9.38.3. EXAMPLE	62
9.38.4. SEE ALSO	62

Chapter 1. Getting Started with the Infinispan CLI

The command line interface (CLI) lets you remotely connect to Infinispan servers to access data and perform administrative functions.

1.1. Creating and Modifying Users

Add Infinispan user credentials and assign permissions to control access to data.

Infinispan server installations use a property realm to authenticate users for the Hot Rod and REST endpoints. This means you need to create at least one user before you can access Infinispan.

By default, users also need roles with permissions to access caches and interact with Infinispan resources. You can assign roles to users individually or add users to groups that have role permissions.

You create users and assign roles with the `user` command in the Infinispan command line interface (CLI).



Run `help user` from a CLI session to get complete command details.

1.1.1. Adding Credentials

You need an `admin` user for the Infinispan Console and full control over your Infinispan environment. For this reason you should create a user with `admin` permissions the first time you add credentials.

Procedure

1. Open a terminal in `$ISP_HOME`.
2. Create an `admin` user with the `user create` command in the CLI.

```
$ bin/cli.sh user create myuser -p changeme -g admin
```

Alternatively, the username "admin" automatically gets `admin` permissions.

```
$ bin/cli.sh user create admin -p changeme
```

3. Open `user.properties` and `groups.properties` with any text editor to verify users and groups.

```
$ cat server/conf/users.properties

#$REALM_NAME=default$
#$ALGORITHM=encrypted$
myuser=scram-sha-1\:BYGcIAwvf6b...

$ cat server/conf/groups.properties

myuser=admin
```

1.1.2. Assigning Roles to Users

Assign roles to users so they have the correct permissions to access data and modify Infinispan resources.

Procedure

1. Start a CLI session with an **admin** user.

```
$ bin/cli.sh
```

2. Assign the **deployer** role to "katie".

```
[//containers/default]> user roles grant --roles=deployer katie
```

3. List roles for "katie".

```
[//containers/default]> user roles ls katie
["deployer"]
```

1.1.3. Adding Users to Groups

Groups let you change permissions for multiple users. You assign a role to a group and then add users to that group. Users inherit permissions from the group role.

Procedure

1. Start a CLI session with an **admin** user.
2. Use the **user create** command to create a group.
 - a. Specify "developers" as the group name with the **--groups** argument.
 - b. Set a username and password for the group.

In a property realm, a group is a special type of user that also requires a username and password.

```
[//containers/default]> user create --groups=developers developers -p changeme
```

3. List groups.

```
[//containers/default]> user ls --groups  
["developers"]
```

4. Assign the **application** role to the "developers" group.

```
[//containers/default]> user roles grant --roles=application developers
```

5. List roles for the "developers" group.

```
[//containers/default]> user roles ls developers  
["application"]
```

6. Add existing users, one at a time, to the group as required.

```
[//containers/default]> user groups john --groups=developers
```

1.1.4. User Roles and Permissions

Infinispan includes a default set of roles that grant users with permissions to access data and interact with Infinispan resources.

ClusterRoleMapper is the default mechanism that Infinispan uses to associate security principals to authorization roles.



ClusterRoleMapper matches principal names to role names. A user named **admin** gets **admin** permissions automatically, a user named **deployer** gets **deployer** permissions, and so on.

Role	Permissions	Description
admin	ALL	Superuser with all permissions including control of the Cache Manager lifecycle.
deployer	ALL_READ, ALL_WRITE, LISTEN, EXEC, MONITOR, CREATE	Can create and delete Infinispan resources in addition to application permissions.

Role	Permissions	Description
<code>application</code>	ALL_READ, ALL_WRITE, LISTEN, EXEC, MONITOR	Has read and write access to Infinispan resources in addition to <code>observer</code> permissions. Can also listen to events and execute server tasks and scripts.
<code>observer</code>	ALL_READ, MONITOR	Has read access to Infinispan resources in addition to <code>monitor</code> permissions.
<code>monitor</code>	MONITOR	Can view statistics via JMX and the <code>metrics</code> endpoint.

Reference

- [org.infinispan.security.AuthorizationPermission Enumeration](#)
- [Infinispan Configuration Schema Reference](#)

1.2. Connecting to Infinispan Servers

Establish CLI connections to Infinispan.

Prerequisites

Add user credentials and have at least one running Infinispan server instance.

Procedure

1. Open a terminal in `$ISPN_HOME`.
2. Start the CLI.

- **Linux:**

```
$ bin/cli.sh
```

- **Microsoft Windows:**

```
$ bin\cli.bat
```

3. Run the `connect` command and enter your username and password when prompted.
 - Infinispan Server on the default port of `11222`:

```
[disconnected]> connect
```

- Infinispan Server with a port offset of `100`:

```
[disconnected]> connect 127.0.0.1:11322
```

1.3. Navigating CLI Resources

The Infinispan CLI exposes a navigable tree that allows you to list, describe, and manipulate Infinispan cluster resources.



Press the tab key to display available commands and options. Use the **-h** option to display help text.

When you connect to a Infinispan cluster, it opens in the context of the default cache container.

```
[//containers/default]>
```

- Use **ls** to list resources.

```
[//containers/default]> ls  
caches  
counters  
configurations  
schemas  
tasks
```

- Use **cd** to navigate the resource tree.

```
[//containers/default]> cd caches
```

- Use **describe** to view information about resources.

```
[//containers/default]> describe
{
  "name" : "default",
  "version" : "xx.x.x-FINAL",
  "cluster_name" : "cluster",
  "coordinator" : true,
  "cache_configuration_names" : [ "org.infinispan.REPL_ASYNC", "___protobuf_metadata",
  "org.infinispan.DIST_SYNC", "org.infinispan.LOCAL",
  "org.infinispan.INVALIDATION_SYNC", "org.infinispan.REPL_SYNC",
  "org.infinispan.SCATTERED_SYNC", "org.infinispan.INVALIDATION_ASYNC",
  "org.infinispan.DIST_ASYNC" ],
  "physical_addresses" : "[192.0.2.0:7800]",
  "coordinator_address" : "<hostname>",
  "cache_manager_status" : "RUNNING",
  "created_cache_count" : "1",
  "running_cache_count" : "1",
  "node_address" : "<hostname>",
  "cluster_members" : [ "<hostname1>", "<hostname2>" ],
  "cluster_members_physical_addresses" : [ "192.0.2.0:7800", "192.0.2.0:7801" ],
  "cluster_size" : 2,
  "defined_caches" : [ {
    "name" : "mycache",
    "started" : true
  }, {
    "name" : "___protobuf_metadata",
    "started" : true
  } ]
}
```

1.3.1. CLI Resources

The Infinispan CLI exposes different resources to:

- create, modify, and manage local or clustered caches.
- perform administrative operations for Infinispan clusters.

Cache Resources

```
[//containers/default]> ls
caches
counters
configurations
schemas
```

caches

Infinispan cache instances. The default cache container is empty. Use the CLI to create caches from templates or `infinispan.xml` files.

counters

Strong or **Weak** counters that record the count of objects.

configurations

Infinispan configurations.

schemas

Protocol Buffers (Protobuf) schemas that structure data in the cache.

tasks

Remote tasks creating and managing Infinispan cache definitions.

Cluster Resources

```
[hostname@cluster/]> ls
containers
cluster
server
```

containers

Cache containers on the Infinispan cluster.

cluster

Lists Infinispan servers joined to the cluster.

server

Resources for managing and monitoring Infinispan servers.

1.4. Shutting Down Infinispan Server

Stop individually running servers or bring down clusters gracefully.

Procedure

1. Create a CLI connection to Infinispan.
2. Shut down Infinispan Server in one of the following ways:
 - Stop all nodes in a cluster with the **shutdown cluster** command, for example:

```
[//containers/default]> shutdown cluster
```

This command saves cluster state to the **data** folder for each node in the cluster. If you use a cache store, the **shutdown cluster** command also persists all data in the cache.

- Stop individual server instances with the **shutdown server** command and the server hostname, for example:

```
[//containers/default]> shutdown server <my_server01>
```



The `shutdown server` command does not wait for rebalancing operations to complete, which can lead to data loss if you specify multiple hostnames at the same time.



Run `help shutdown` for more details about using the command.

Verification

Infinispan logs the following messages when you shut down servers:

```
ISPN080002: Infinispan Server stopping
ISPN000080: Disconnecting JGroups channel cluster
ISPN000390: Persisted state, version=<$version> timestamp=YYYY-MM-DDTHH:MM:SS
ISPN080003: Infinispan Server stopped
```

1.4.1. Restarting Infinispan Clusters

When you bring Infinispan clusters back online after shutting them down, you should wait for the cluster to be available before adding or removing nodes or modifying cluster state.

If you shutdown clustered nodes with the `shutdown server` command, you must restart each server in reverse order.

For example, if you shutdown `server1` and then shutdown `server2`, you should first start `server2` and then start `server1`.

If you shutdown a cluster with the `shutdown cluster` command, clusters become fully operational only after all nodes rejoin.

You can restart nodes in any order but the cluster remains in DEGRADED state until all nodes that were joined before shutdown are running.

Chapter 2. Performing Cache Operations with the Infinispan CLI

The command line interface (CLI) lets you remotely connect to Infinispan servers to access data and perform administrative functions.

2.1. Creating Caches with the Infinispan Command Line Interface (CLI)

Use the Infinispan CLI to add caches from templates or configuration files in XML or JSON format.

Prerequisites

Create a user and start at least one Infinispan server instance.

Procedure

1. Create a CLI connection to Infinispan.
2. Add cache definitions with the `create cache` command.
 - Add a cache definition from an XML or JSON file with the `--file` option.

```
[//containers/default]> create cache --file=configuration.xml mycache
```

- Add a cache definition from a template with the `--template` option.

```
[//containers/default]> create cache --template=org.infinispan.DIST_SYNC mycache
```



Press the tab key after the `--template=` argument to list available cache templates.

3. Verify the cache exists with the `ls` command.

```
[//containers/default]> ls caches  
mycache
```

4. Retrieve the cache configuration with the `describe` command.

```
[//containers/default]> describe caches/mycache
```

Reference

- [Creating Infinispan CLI Connections](#)
- [Performing Cache Operations with the Infinispan CLI](#)

2.1.1. Cache Configuration

You can provide cache configuration in XML or JSON format.

XML

```
<distributed-cache name="myCache" mode="SYNC">
  <encoding media-type="application/x-protostream"/>
  <memory max-count="1000000" when-full="REMOVE"/>
</distributed-cache>
```

JSON

```
{
  "distributed-cache": {
    "name": "myCache",
    "mode": "SYNC",
    "encoding": {
      "media-type": "application/x-protostream"
    },
    "memory": {
      "max-count": 1000000,
      "when-full": "REMOVE"
    }
  }
}
```

JSON format

Cache configuration in JSON format must follow the structure of an XML configuration. * XML elements become JSON objects. * XML attributes become JSON fields.

2.2. Adding Cache Entries

Create **key:value** pair entries in the data container.

Prerequisites

Create an Infinispan cache that can store your data.

Procedure

1. Create a CLI connection to Infinispan.
2. Add entries into your cache as follows:
 - Use the **put** command from the context of a cache:

```
[//containers/default/caches/mycache]> put hello world
```

- Use the **--cache=** with the **put** command:

```
[//containers/default]> put --cache=mycache hello world
```

3. Use the **get** command to verify entries.

```
[//containers/default/caches/mycache]> get hello  
world
```

2.3. Clearing Caches and Deleting Entries

Remove data from caches with the Infinispan CLI.

Procedure

1. Create a CLI connection to Infinispan.
2. Do one of the following:
 - Delete all entries with the **clearcache** command.

```
[//containers/default]> clearcache mycache
```

- Remove specific entries with the **remove** command.

```
[//containers/default]> remove --cache=mycache hello
```

2.4. Deleting Caches

Drop caches to remove them and delete all data they contain.

Procedure

1. Create a CLI connection to Infinispan.
2. Remove caches with the **drop** command.

```
[//containers/default]> drop cache mycache
```

Chapter 3. Performing Batch Operations

Process operations in groups, either interactively or using batch files.

Prerequisites

- A running Infinispan cluster.

3.1. Performing Batch Operations with Files

Create files that contain a set of operations and then pass them to the Infinispan CLI.

Procedure

1. Create a file that contains a set of operations.

For example, create a file named `batch` that creates a cache named `mybatch`, adds two entries to the cache, and disconnects from the CLI.

```
connect --username=<username> --password=<password> <hostname>:11222
create cache --template=org.infinispan.DIST_SYNC mybatch
put --cache=mybatch hello world
put --cache=mybatch hola mundo
ls caches/mybatch
disconnect
```



Configure the CLI with the `autoconnect-url` property instead of using the `connect` command directly in your batch files.

2. Run the CLI and specify the file as input.

```
$ bin/cli.sh -f batch
```



CLI batch files support system property expansion. Strings that use the `${property}` format are replaced with the value of the `property` system property.

3.2. Performing Batch Operations Interactively

Use the standard input stream, `stdin`, to perform batch operations interactively.

Procedure

1. Start the Infinispan CLI in interactive mode.

```
$ bin/cli.sh -c localhost:11222 -f -
```



You can configure the CLI connection with the `autoconnect-url` property instead of using the `-c` argument.

2. Run batch operations, for example:

```
create cache --template=org.infinispan.DIST_SYNC mybatch
put --cache=mybatch hello world
put --cache=mybatch hola mundo
disconnect
quit
```



Use `echo` to add commands in interactive mode.

The following example shows how to use `echo describe` to get cluster information:

```
$ echo describe|bin/cli.sh -c localhost:11222 -f -
{
  "name" : "default",
  "version" : "10.0.0-SNAPSHOT",
  "coordinator" : false,
  "cache_configuration_names" : [ "org.infinispan.REPL_ASYNC", "___protobuf_metadata",
"org.infinispan.DIST_SYNC", "qcache", "org.infinispan.LOCAL", "dist_cache_01",
"org.infinispan.INVALIDATION_SYNC", "org.infinispan.REPL_SYNC",
"org.infinispan.SCATTERED_SYNC", "mycache", "org.infinispan.INVALIDATION_ASYNC",
"mybatch", "org.infinispan.DIST_ASYNC" ],
  "cluster_name" : "cluster",
  "physical_addresses" : "[192.168.1.7:7800]",
  "coordinator_address" : "thundercat-34689",
  "cache_manager_status" : "RUNNING",
  "created_cache_count" : "4",
  "running_cache_count" : "4",
  "node_address" : "thundercat-47082",
  "cluster_members" : [ "thundercat-34689", "thundercat-47082" ],
  "cluster_members_physical_addresses" : [ "10.36.118.25:7801", "192.168.1.7:7800" ],
  "cluster_size" : 2,
  "defined_caches" : [ {
    "name" : "___protobuf_metadata",
    "started" : true
  }, {
    "name" : "mybatch",
    "started" : true
  } ]
}
```

Chapter 4. Configuring the Infinispan CLI

Define configuration properties for the Infinispan CLI.

4.1. Setting Infinispan CLI Properties and Persistent Storage

Configure Infinispan CLI startup operations and customize the location for persistent storage.

Prerequisites

Create at least one Infinispan user.

Procedure

1. Optionally set a custom path to the Infinispan CLI storage directory in one of the following ways:

- Using the `cli.dir` system property:

```
$ bin/cli.sh -Dcli.dir=/path/to/cli/storage ...
```

- Using the `ISPN_CLI_DIR` environment variable:

```
export ISPN_CLI_DIR=/path/to/cli/storage
$ bin/cli.sh ...
```

2. Set values for configuration properties with the `config set` command.

For example, set the `autoconnect-url` property so that the CLI automatically connects to that URL.



For remote connections, specify the URL and provide credentials:

- `http[s]://<username>:<password>@<hostname>:<port>` for basic authentication.
- `http[s]://<token>@<hostname>:<port>` for OAuth authentication.

```
$ bin/cli.sh config set autoconnect-url
http://<username>:<password>@<hostname>:11222
```

3. Verify configuration properties with the `config get` command.



Run `help config` to review available configuration properties and get example usage.

4.2. Creating Command Aliases

Create aliases for Infinispan CLI commands to define custom shortcuts.

Procedure

1. Create aliases with the `alias <alias>=<command>` command.

For example, set `q` as an alias for the `quit` command:

```
[//containers/default]> alias q=quit
```

2. Run the `alias` command to check the defined aliases.

```
[//containers/default]> alias  
alias q='quit'
```

3. Delete aliases with the `unalias` command, for example:

```
[//containers/default]> unalias q
```

4.3. Trusting Infinispan Server Connections

Secure Infinispan CLI connections to Infinispan Server with SSL/TLS certificates. If you create a key store as an SSL identity for Infinispan Server, the CLI can validate server certificates to verify the identity.

Prerequisites

- Set up an SSL identity for Infinispan Server.
- Create at least one Infinispan user.

Procedure

1. Specify the location of the server key store, as in the following example:

```
$ bin/cli.sh config set truststore /home/user/my-trust-store.jks
```

2. Define the key store password, if necessary, as follows:

```
$ bin/cli.sh config set truststore-password secret
```

3. Verify your CLI configuration.

```
$ bin/cli.sh config get truststore
truststore=/home/user/my-trust-store.jks

$ bin/cli.sh config get truststore-password
truststore-password=secret
```

Additional resources

- [Setting Up SSL Identities for Infinispan Server](#)

4.4. Infinispan CLI Storage Directory

Infinispan CLI stores configuration in the following default directory:

Operating System	Default Path
Linux/Unix	<code>\$HOME/.config/{brandshortname}</code>
Microsoft Windows	<code>%APPDATA%/Sun/Java/{brandshortname}</code>
Mac OS	<code>\$HOME/Library/Java/{brandshortname}</code>

This directory contains the following files:

cli.properties

Stores values for CLI configuration properties.

aliases

Stores command aliases.

history

Stores CLI history.

Chapter 5. Working with Counters

Counters provide atomic increment and decrement operations that record the count of objects.

Prerequisites

- Start the Infinispan CLI.
- Connect to a running Infinispan cluster.

5.1. Creating Counters

Create strong and weak counters with the Infinispan CLI.

Procedure

1. Create a CLI connection to Infinispan.
2. Run the `create counter` command with the appropriate arguments.
 - a. Create `my-weak-counter`.

```
[//containers/default]> create counter --concurrency-level=1 --initial-value=5 -  
-storage=PERSISTENT --type=weak my-weak-counter
```

- b. Create `my-strong-counter`.

```
[//containers/default]> create counter --initial-value=3 --storage=PERSISTENT --  
type=strong my-strong-counter
```

3. List available counters.

```
[//containers/default]> ls counters  
my-strong-counter  
my-weak-counter
```

4. Verify counter configurations.
 - a. Describe `my-weak-counter`.

```
[//containers/default]> describe counters/my-weak-counter

{
  "weak-counter":{
    "initial-value":5,
    "storage":"PERSISTENT",
    "concurrency-level":1
  }
}
```

b. Describe **my-strong-counter**.

```
[//containers/default]> describe counters/my-strong-counter

{
  "strong-counter":{
    "initial-value":3,
    "storage":"PERSISTENT",
    "upper-bound":5
  }
}
```

5.2. Adding Deltas to Counters

Increment or decrement counters with arbitrary values.

Procedure

1. Select a counter.

```
[//containers/default]> counter my-weak-counter
```

2. List the current count.

```
[//containers/default/counters/my-weak-counter]> ls
5
```

3. Increment the counter value by **2**.

```
[//containers/default/counters/my-weak-counter]> add --delta=2
```

4. Decrement the counter value by **-4**.

```
[//containers/default/counters/my-weak-counter]> add --delta=-4
```



Strong counters return values after the operation is applied. Use `--quiet=true` to hide the return value.

For example, `my-strong-counter]> add --delta=3 --quiet=true`.

Weak counters return empty responses.

Chapter 6. Querying Caches with Protobuf Metadata

Infinispan supports using Protocol Buffers (Protobuf) to structure data in the cache so that you can query it.

Prerequisites

- Start the Infinispan CLI.
- Connect to a running Infinispan cluster.

6.1. Configuring Media Types

Encode cache entries with different media types to store data in a format that best suits your requirements.

For example, the following procedure shows you how to configure the `application/x-protostream` media type.

Procedure

1. Create a Infinispan configuration file that adds a distributed cache named `pcache` and configures the media type, for example:

```
<distributed-cache name="pcache">
  <encoding>
    <key media-type="application/x-protostream"/>
    <value media-type="application/x-protostream"/>
  </encoding>
</distributed-cache>
```

2. Create `pcache` from `pcache.xml` with the `--file=` option.

```
[//containers/default]> create cache --file=pcache.xml pcache
```

3. Verify `pcache`.

```

[//containers/default]> ls caches
pcache
__protobuf_metadata
[//containers/default]> describe caches/pcache
{
  "distributed-cache" : {
    "mode" : "SYNC",
    "encoding" : {
      "key" : {
        "media-type" : "application/x-protostream"
      },
      "value" : {
        "media-type" : "application/x-protostream"
      }
    },
    "transaction" : {
      "mode" : "NONE"
    }
  }
}

```

4. Add an entry to **pcache** and check the encoding.

```

[//containers/default]> put --cache=pcache good morning
[//containers/default]> cd caches/pcache
[//containers/default/caches/pcache]> get good
{
  "_type" : "string",
  "_value" : "morning"
}

```

6.2. Registering Protobuf Schemas

Protobuf schemas contain data structures known as messages in **.proto** definition files.

Procedure

1. Create a schema file named **person.proto** with the following messages:

```

package org.infinispan.rest.search.entity;

message Address {
    required string street = 1;
    required string postCode = 2;
}

message PhoneNumber {
    required string number = 1;
}

message Person {
    optional int32 id = 1;
    required string name = 2;
    required string surname = 3;
    optional Address address = 4;
    repeated PhoneNumber phoneNumbers = 5;
    optional uint32 age = 6;
    enum Gender {
        MALE = 0;
        FEMALE = 1;
    }

    optional Gender gender = 7;
}

```

2. Register `person.proto`.

```
[//containers/default]> schema --upload=person.proto person.proto
```

3. Verify `person.proto`.

```

[//containers/default]> cd caches/___protobuf_metadata
[//containers/default/caches/___protobuf_metadata]> ls
person.proto
[//containers/default/caches/___protobuf_metadata]> get person.proto

```

6.3. Querying Caches with Protobuf Schemas

Infinispan automatically converts JSON to Protobuf so that you can read and write cache entries in JSON format and use Protobuf schemas to query them.

For example, consider the following JSON documents:

lukecage.json

```
{
  "_type": "org.infinispan.rest.search.entity.Person",
  "id": 2,
  "name": "Luke",
  "surname": "Cage",
  "gender": "MALE",
  "address": {"street": "38th St", "postCode": "NY 11221"},
  "phoneNumbers": [{"number": 4444}, {"number": 5555}]
}
```

jessicajones.json

```
{
  "_type": "org.infinispan.rest.search.entity.Person",
  "id": 1,
  "name": "Jessica",
  "surname": "Jones",
  "gender": "FEMALE",
  "address": {"street": "46th St", "postCode": "NY 10036"},
  "phoneNumbers": [{"number": 1111}, {"number": 2222}, {"number": 3333}]
}
```

matthewmurdock.json

```
{
  "_type": "org.infinispan.rest.search.entity.Person",
  "id": 3,
  "name": "Matthew",
  "surname": "Murdock",
  "gender": "MALE",
  "address": {"street": "57th St", "postCode": "NY 10019"},
  "phoneNumbers": []
}
```

Each of the preceding JSON documents contains:

- A **_type** field that identifies the Protobuf message to which the JSON document corresponds.
- Several fields that correspond to datatypes in the **person.proto** schema.

Procedure

1. Navigate to the **pcache** cache.

```
[//containers/default/caches]> cd pcache
```

2. Add each JSON document as an entry to the cache, for example:

```
[//containers/default/caches/pcache]> put --encoding=application/json  
--file=jessicajones.json jessicajones  
[//containers/default/caches/pcache]> put --encoding=application/json  
--file=matthewmurdock.json matthewmurdock  
[//containers/default/caches/pcache]> put --encoding=application/json  
--file=lukecage.json lukecage
```

3. Verify that the entries exist.

```
[//containers/default/caches/pcache]> ls  
lukecage  
matthewmurdock  
jessicajones
```

4. Query the cache to return entries from the Protobuf **Person** entity where the gender datatype is **MALE**.


```

[//containers/default/caches/pcache]> query "from
org.infinispan.rest.search.entity.Person p where p.gender = 'MALE'"
{
  "total_results" : 2,
  "hits" : [ {
    "hit" : {
      "_type" : "org.infinispan.rest.search.entity.Person",
      "id" : 2,
      "name" : "Luke",
      "surname" : "Cage",
      "gender" : "MALE",
      "address" : {
        "street" : "38th St",
        "postCode" : "NY 11221"
      },
      "phoneNumbers" : [ {
        "number" : "4444"
      }, {
        "number" : "5555"
      } ]
    }
  }, {
    "hit" : {
      "_type" : "org.infinispan.rest.search.entity.Person",
      "id" : 3,
      "name" : "Matthew",
      "surname" : "Murdock",
      "gender" : "MALE",
      "address" : {
        "street" : "57th St",
        "postCode" : "NY 10019"
      }
    }
  } ]
}

```

Chapter 7. Performing Cross-Site Replication Operations

Infinispan clusters running in different locations can discover and communicate with each other to backup data.

Prerequisites

- Start the Infinispan CLI.
- Connect to a running Infinispan cluster.

7.1. Bringing Backup Locations Offline and Online

Take backup locations offline manually and bring them back online.

Procedure

1. Create a CLI connection to Infinispan.
2. Check if backup locations are online or offline with the `site status` command:

```
[//containers/default]> site status --cache=cacheName --site=NYC
```



`--site` is an optional argument. If not set, the CLI returns all backup locations.

3. Manage backup locations as follows:
 - Bring backup locations online with the `bring-online` command:

```
[//containers/default]> site bring-online --cache=customers --site=NYC
```

- Take backup locations offline with the `take-offline` command:

```
[//containers/default]> site take-offline --cache=customers --site=NYC
```

For more information and examples, run the `help site` command.

7.2. Configuring Cross-Site State Transfer Modes

You can configure cross-site state transfer operations to happen automatically when Infinispan detects that backup locations come online. Alternatively you can use the default mode, which is to manually perform state transfer through the CLI or via JMX or REST.

Procedure

1. Create a CLI connection to Infinispan.

2. Use the **site** command to configure state transfer modes, as in the following examples:

- Retrieve the current state transfer mode.

```
[//containers/default]> site state-transfer-mode get --cache=cacheName  
--site=NYC  
"MANUAL"
```

- Configure automatic state transfer operations for a cache and backup location.

```
[//containers/default]> site state-transfer-mode set --cache=cacheName --site=NYC  
--mode=AUTO
```



Run the **help site** command for more information and examples.

7.3. Pushing State to Backup Locations

Transfer cache state to remote backup locations.

Procedure

1. Create a CLI connection to Infinispan.
2. Use the **site** command to push state transfer, as in the following example:

```
[//containers/default]> site push-site-state --cache=cacheName --site=NYC
```

For more information and examples, run the **help site** command.

Chapter 8. Backing Up and Restoring Infinispan Clusters

Create archives of Infinispan resources that include cached entries, cache configurations, Protobuf schemas, and server scripts. You can then use the backup archives to restore Infinispan Server clusters after a restart or migration.

Prerequisites

- Start the Infinispan CLI.
- Connect to a running Infinispan cluster.

8.1. Backing Up Infinispan Clusters

Create backup archives in `.zip` format that you can download or store on Infinispan Server.

Prerequisites

Backup archives should reflect the most recent cluster state. For this reason you should ensure the cluster is no longer accepting write requests before you create backup archives.

Procedure

1. Create a CLI connection to Infinispan.
2. Run the `backup create` command with the appropriate options, for example:
 - Back up all resources with an automatically generated name.

```
[//containers/default]> backup create
```

- Back up all resources in a backup archive named `example-backup`.

```
[//containers/default]> backup create -n example-backup
```

- Back up all resources to the `/some/server/dir` path on the server.

```
[//containers/default]> backup create -d /some/server/dir
```

- Back up only caches and cache templates.

```
[//containers/default]> backup create --caches=* --templates=*
```

- Back up named Protobuf schemas only.

```
[//containers/default]> backup create --proto-schemas=schema1,schema2
```

3. List available backup archives on the server.

```
[//containers/default]> backup ls
```

4. Download the **example-backup** archive from the server.

If the backup operation is still in progress, the command waits for it to complete.

```
[//containers/default]> backup get example-backup
```

5. Optionally delete the **example-backup** archive from the server.

```
[//containers/default]> backup delete example-backup
```

8.2. Restoring Infinispan Clusters from Backup Archives

Apply the content of backup archives to Infinispan clusters to restore them to the backed up state.

Prerequisites

- Create a backup archive that is either local to the Infinispan CLI or stored on Infinispan Server.
- Ensure that the target container matches the container name in the backup archive. You cannot restore backups if the container names do not match.

Procedure

1. Create a CLI connection to Infinispan.
2. Run the **backup restore** command with the appropriate options.
 - Restore all content from a backup archive accessible on the server.

```
[//containers/default]> backup restore /some/path/on/the/server
```

- Restore all content from a local backup archive.

```
[//containers/default]> backup restore -u /some/local/path
```

- Restore only cache content from a backup archive on the server.

```
[//containers/default]> backup restore /some/path/on/the/server --caches=*
```

Chapter 9. Command Reference

Review manual pages for Infinispan CLI commands.



Use `help` command to access manual pages directly from your CLI session.

For example, to view the manual page for the `get` command do the following:

```
$ help get
```

9.1. ADD(1)

9.1.1. NAME

`add` - increments and decrements counters with arbitrary values.

9.1.2. SYNOPSIS

```
add ['OPTIONS'] ['COUNTER_NAME']
```

9.1.3. OPTIONS

`--delta='nnn'`

Sets a delta to increment or decrement the counter value. Defaults to `1`.

`-q, --quiet='[true|false]'`

Hides return values for strong counters. The default is `false`.

9.1.4. EXAMPLES

```
add --delta=10 cnt_a
```

Increments the value of `cnt_a` by `10`.

```
add --delta=-5 cnt_a
```

Decrements the value of `cnt_a` by `5`.

9.1.5. SEE ALSO

`cas(1)`, `reset(1)`

9.2. ALIAS(1)

9.2.1. NAME

`alias` - creates or displays aliases.

9.2.2. SYNOPSIS

alias ['ALIAS-NAME']='COMMAND']

9.2.3. EXAMPLES

alias q=quit

Creates **q** as an alias for the **quit** command.

alias

Lists all defined aliases.

9.2.4. SEE ALSO

config(1), unalias(1)

9.3. BACKUP(1)

9.3.1. NAME

backup - manage container backup creation and restoration.

9.3.2. SYNOPSIS

backup create ['OPTIONS']

backup delete ['OPTIONS'] **BACKUP_NAME**

backup get ['OPTIONS'] **BACKUP_NAME**

backup ls

backup restore ['OPTIONS'] **BACKUP_PATH**

9.3.3. BACKUP CREATE OPTIONS

-d, --dir='PATH'

Specifies a directory on the server to create and store the backup archive.

-n, --name='NAME'

Defines a name for the backup archive.

--caches='cache1,cache2,...'

Lists caches to back up. Use '*' to back up all caches.

--templates='template1,template2,...'

Lists cache templates to back up. Use '*' to back up all templates.

--counters='counter1,counter2,...'

Lists of counters to back up. Use '*' to back up all counters.

--proto-schemas='schema1,schema2,...'

Lists Protobuf schemas to back up. Use '*' to back up all schemas.

--tasks='task1,task2,...'

Lists server tasks to back up. Use '*' to back up all tasks.

9.3.4. BACKUP GET OPTIONS

--no-content

Does not download content. The command returns only when the backup operation is complete.

9.3.5. BACKUP RESTORE OPTIONS

-u, --upload

Defines the path to a local backup archive that is uploaded to the server.

-n, --name='NAME'

Defines a name for the restore request.

--caches='cache1,cache2,...'

Lists caches to restore. Use '*' to restore all caches from the backup archive.

--templates='template1,template2,...'

Lists cache templates to restore. Use '*' to restore all templates from the backup archive.

--counters='counter1,counter2,...'

Lists counters to restore. Use '*' to restore all counters from the backup archive.

--proto-schemas='schema1,schema2,...'

Lists Protobuf schemas to restore. Use '*' to restore all schemas from the backup archive.

--tasks='task1,task2,...'

Lists server tasks to restore. Use '*' to restore all tasks from the backup archive.

9.3.6. EXAMPLES

```
backup create -n example-backup
```

Initiates a backup of all container content with name `example-backup`.

```
backup create -d /some/server/dir
```

Initiates a backup of all container content and stores it on the server at path `/some/server/dir`.

```
backup create --caches=* --templates=*
```

Initiates a backup that contains only cache and cache configuration resources.

```
backup create --proto-schemas=schema1,schema2
```

Initiates a backup that contains the named schema resources only.

backup ls

Lists all backups available on the server.

backup get example-backup

Downloads the **example-backup** archive from the server. If the backup operation is in progress, the command waits for it to complete.

backup restore /some/path/on/the/server

Restores all content from a backup archive on the server.

backup restore -u /some/local/path

Restores all content from a local backup archive that is uploaded to the server.

backup restore /some/path/on/the/server --caches=*

Restores only cache content from a backup archive on the server.

backup restore /some/path/on/the/server --proto-schemas=schema1,schema2

Restores only the named schema resources from a backup archive on the server.

backup delete example-backup

Deletes the **example-backup** archive from the server.

9.3.7. SEE ALSO

drop(1)

9.4. BENCHMARK(1)

9.4.1. NAME

benchmark - runs a performance benchmark against a cache.

You can run performance benchmarks for the following HTTP and Hot Rod protocols: **http**, **https**, **hotrod**, and **hotrods**. You specify the protocol for the benchmark with a URI. If you do not specify a protocol, the benchmark uses the URI of the current CLI connection.

Benchmarks for Hot Rod URIs connect to the entire cluster. For HTTP URIs, benchmarks connect to a single node only.

Benchmarks test performance against an existing cache. Before you run a benchmark, you should create a cache with the capabilities you want to measure. For example, if you want to evaluate the performance of cross-site replication, you should create a cache that has backup locations. If you want to test the performance of persistence, create a cache that uses an appropriate cache store.

9.4.2. SYNOPSIS

benchmark ['OPTIONS'] [**uri**]

9.4.3. BENCHMARK OPTIONS

-t, --threads='num'

Specifies the number of threads to create. Defaults to 10.

--cache='cache'

Names the cache against which the benchmark is performed. Defaults to `benchmark`. You must create the cache before running the benchmark if it does not already exist.

***--key-size='num'**

Sets the size, in bytes, of the key. Defaults to 16 bytes.

***--value-size='num'**

Sets the size, in bytes, of the value. Defaults to 1000 bytes.

***--keyset-size='num'**

Defines the size, in bytes, of the test key set. Defaults to 1000.

--verbosity=['SILENT', 'NORMAL', 'EXTRA']

Specifies the verbosity level of the output. Possible values, from least to most verbose, are `SILENT`, `NORMAL`, and `EXTRA`. The default is `NORMAL`.

-c, --count='num'

Specifies how many measurement iterations to perform. Defaults to 5.

--time='time'

Sets the amount of time, in seconds, that each iteration takes. Defaults to 10.

--warmup-count='num'

Specifies how many warmup iterations to perform. Defaults to 5.

--warmup-time='time'

Sets the amount of time, in seconds, that each warmup iteration takes. Defaults to 1.

--mode='mode'

Specifies the benchmark mode. Possible values are `Throughput`, `AverageTime`, `SampleTime`, `SingleShotTime`, and `All`. The default is `Throughput`.

--time-unit='unit'

Specifies the time unit for results in the benchmark report. Possible values are `NANOSECONDS`, `MICROSECONDS`, `MILLISECONDS`, and `SECONDS`. The default is `MICROSECONDS`.

9.4.4. EXAMPLES

```
benchmark hotrod://localhost:11222
```

Performs a benchmark test with the Hot Rod protocol.

```
benchmark --value-size=10000 --cache=largecache hotrod://localhost:11222
```

Performs a benchmark test with the Hot Rod protocol against the **largecache** cache using test values that are 10000 bytes in size.

benchmark **--mode=All** **--threads=20** <https://user:password@server:11222>

Performs a benchmark test with the HTTPS protocol using 20 threads and includes all modes in the report.

9.5. CACHE(1)

9.5.1. NAME

cache - selects the default cache for subsequent commands.

9.5.2. SYNOPSIS

cache ['CACHE_NAME']

9.5.3. EXAMPLE

cache mycache

Selects **mycache** and is the same as navigating the resource tree using **cd caches/mycache**.

9.5.4. SEE ALSO

cd(1), **clear(1)**, **container(1)**, **get(1)**, **put(1)**, **remove(1)**

9.6. CAS(1)

9.6.1. NAME

cas - performs 'compare-and-swap' operations on strong counters.

9.6.2. SYNOPSIS

cas ['OPTIONS'] ['COUNTER_NAME']

9.6.3. OPTIONS

--expect='nnn'

Specifies the expected value of the counter.

--value='nnn'

Sets a new value for the counter.

-q, --quiet='[true|false]'

Hides return values. The default is false.

9.6.4. EXAMPLE

```
cas --expect=10 --value=20 cnt_a
```

Sets the value of `cnt_a` to `20` only if the current value is `10`

9.6.5. SEE ALSO

`add(1)`, `cas(1)`, `reset(1)`

9.7. CD(1)

9.7.1. NAME

`cd` - navigates the server resource tree.

9.7.2. DESCRIPTION

`PATH` can be absolute or relative to the current resource. `../` specifies parent resources.

9.7.3. SYNOPSIS

```
cd ['PATH']
```

9.7.4. EXAMPLE

```
cd caches
```

Changes to the `caches` path in the resource tree.

9.7.5. SEE ALSO

`cache(1)`, `ls(1)`, `container(1)`

9.8. CLEARCACHE(1)

9.8.1. NAME

`clearcache` - removes all entries from a cache.

9.8.2. SYNOPSIS

```
clearcache ['CACHE_NAME']
```

9.8.3. EXAMPLES

```
clearcache mycache
```

Removes all entries from `mycache`.

9.8.4. SEE ALSO

cache(1), drop(1), remove(1)

9.9. CONFIG(1)

9.9.1. NAME

config - manages CLI configuration properties.

9.9.2. SYNOPSIS

config

config set 'name' 'value'

config get 'name'

9.9.3. DESCRIPTION

Manage (list, set, get) CLI configuration properties.

9.9.4. COMMAND SYNOPSIS

config

Lists all configuration properties that are set.

config set 'name' ['value']

Sets the value of a specific property. If you do not specify a value, the property is not set.

config get 'name'

Retrieves the value of a specific property.

9.9.5. COMMON OPTIONS

These options apply to all commands:

-h, --help

Displays a help page for the command or sub-command.

9.9.6. PROPERTIES

autoconnect-url

Specifies the URL to which the CLI automatically connects on startup.

autoexec

Specifies the path of a CLI batch file to execute on startup.

trustall

Specifies whether to trust all server certificates. Values are **false** (default) and **true**.

truststore

Defines the path to a keystore that contains a certificate chain that verifies server identity.

truststore-password

Specifies a password to access the keystore.

9.9.7. EXAMPLES

```
config set autoconnect-url http://192.0.2.0:11222
```

Connects to a server at a custom IP address when you start the CLI.

```
config get autoconnect-url
```

Returns the value for the **autoconnect-url** configuration property.

```
config set autoexec /path/to/mybatchfile
```

Runs a batch file named "mybatchfile" when you start the CLI.

```
config set trustall true
```

Trusts all server certificates.

```
config set truststore /home/user/my-trust-store.jks
```

Specifies the path of a keystore named "my-trust-store.jks".

```
config set truststore-password secret
```

Sets the keystore password, if required.

9.9.8. SEE ALSO

alias(1), unalias(1)

9.10. CONNECT(1)

9.10.1. NAME

connect - connects to running \${infinispan.brand.name} servers.

9.10.2. DESCRIPTION

Defaults to <http://localhost:11222> and prompts for credentials if authentication is required.

9.10.3. SYNOPSIS

```
connect ['OPTIONS'] ['SERVER_LOCATION']
```

9.10.4. OPTIONS

-u, --username='USERNAME'

Specifies a username to authenticate with `${infinispan.brand.name}` servers.

-p, --password='PASSWORD'

Specifies passwords.

9.10.5. EXAMPLE

```
connect 127.0.0.1:11322 -u test -p changeme
```

Connects to a locally running server using a port offset of `100` and example credentials.

9.10.6. SEE ALSO

disconnect(1)

9.11. CONTAINER(1)

9.11.1. NAME

container - selects the container for running subsequent commands.

9.11.2. SYNOPSIS

container ['CONTAINER_NAME']

9.11.3. EXAMPLE

```
container default
```

Selects the default container and is the same as navigating the resource tree using `cd containers/default`.

9.11.4. SEE ALSO

cd(1), clear(1), container(1), get(1), put(1), remove(1)

9.12. COUNTER(1)

9.12.1. NAME

counter - selects the default counter for subsequent commands.

9.12.2. SYNOPSIS

counter ['COUNTER_NAME']

9.12.3. EXAMPLE

`counter cnt_a`

Selects `cnt_a` and is the same as navigating the resource tree using `cd counters/cnt_a`.

9.12.4. SEE ALSO

`add(1)`, `cas(1)`

9.13. CREATE(1)

9.13.1. NAME

`create` - creates caches and counters on `${infinispan.brand.name}` servers.

9.13.2. SYNOPSIS

`create cache` [`OPTIONS`] `CACHE_NAME`

`create counter` [`OPTIONS`] `COUNTER_NAME`

9.13.3. CREATE CACHE OPTIONS

`-f, --file='FILE'`

Specifies a configuration file in JSON or XML format.

`-t, --template='TEMPLATE'`

Specifies a configuration template. Use tab autocompletion to see available templates.

`-v, --volatile='[true | false]'`

Specifies whether the cache is persistent or volatile. The default is false.

9.13.4. CREATE COUNTER OPTIONS

`-t, --type='[weak | strong]'`

Specifies if the counter is weak or strong.

`-s, --storage='[PERSISTENT | VOLATILE]'`

Specifies whether the counter is persistent or volatile.

`-c, --concurrency-level='nnn'`

Sets the concurrency level of the counter.

`-i, --initial-value='nnn'`

Sets the initial value of the counter.

`-l, --lower-bound='nnn'`

Sets the lower bound of a **strong** counter.

-u, --upper-bound='nnn'

Sets the upper bound of a **strong** counter.

9.13.5. EXAMPLES

```
create cache --template=org.infinispan.DIST_SYNC mycache
```

Creates a cache named **mycache** from the **DIST_SYNC** template.

```
create counter --initial-value=3 --storage=PERSISTENT --type=strong cnt_a
```

Creates a strong counter named **cnt_a**.

9.13.6. SEE ALSO

drop(1)

9.14. CREDENTIALS(1)

9.14.1. NAME

credentials - manages keystores that contain \${infinispan.brand.name} Server credentials

9.14.2. SYNOPSIS

credentials ls

credentials add 'alias'

credentials remove 'alias'

9.14.3. DESCRIPTION

List, create, and remove credentials inside a keystore. By default, commands manage the **credentials.pfx** keystore in the server configuration directory.

9.14.4. SYNOPSIS

credentials ls

Lists credential aliases stored in the keystore.

Add a credential

credentials add 'alias'

Adds an alias and corresponding credential to the keystore.

Remove a credential

credentials remove 'alias'

Deletes an alias and corresponding credential from the keystore.

9.14.5. OPTIONS

-h, --help

Prints command help.

-s, --server-root='path-to-server-root'

Specifies the path to the server root directory. Defaults to `server`.

--path='credentials.pfx'

Specifies the path to the credential keystore. Defaults to the server configuration directory, `server/conf`.

-p, --password='password'

Specifies a password for the credential keystore.

-t, --type='PKCS12'

Specifies the type of keystore that contains credentials. Supported types are `PKCS12` or `JCEKS`. Defaults to `PKCS12`.

9.14.6. CREDENTIALS ADD OPTIONS

-c, --credential='credential'

Specifies the credential to store.

9.14.7. EXAMPLES

```
credentials add dbpassword -c changeme -p "secret1234!"
```

Creates a new default credential keystore, if does not already exist, and adds an alias of "dbpassword" for a password of "changeme". This command also sets "secret1234!" as the password for the credential keystore, which must match the password in the server configuration: `<clear-text-credential clear-text="secret1234!"/>`

```
credentials ls -p "secret1234!"
```

Lists all aliases in the default credential keystore.

```
credentials add ldappassword -t JCEKS -p "secret1234!"
```

Creates a credential keystore in JCEKS format and adds an alias "ldappassword". This command prompts you to specify the password that corresponds to the alias.

9.15. DESCRIBE(1)

9.15.1. NAME

`describe` - displays information about resources.

9.15.2. SYNOPSIS

```
describe [PATH]
```

9.15.3. EXAMPLES

`describe //containers/default`

Displays information about the default container.

`describe //containers/default/caches/mycache`

Displays information about the `mycache` cache.

`describe //containers/default/caches/mycache/k1`

Displays information about the `k1` key.

`describe //containers/default/counters/cnt1`

Displays information about the `cnt1` counter.

9.15.4. SEE ALSO

`cd(1)`, `ls(1)`

9.16. DISCONNECT(1)

9.16.1. NAME

`disconnect` - ends CLI sessions with `$(infinispan.brand.name)` servers.

9.16.2. SYNOPSIS

`disconnect`

9.16.3. EXAMPLE

`disconnect`

Ends the current CLI session.

9.16.4. SEE ALSO

`connect(1)`

9.17. DROP(1)

9.17.1. NAME

`drop` - deletes caches and counters.

9.17.2. SYNOPSIS

`drop cache` `CACHE_NAME`

`drop counter` `COUNTER_NAME`

9.17.3. EXAMPLES

`drop cache mycache`

Deletes the `mycache` cache.

`drop counter cnt_a`

Deletes the `cnt_a` counter.

9.17.4. SEE ALSO

`create(1)`, `clearcache(1)`

9.18. ENCODING(1)

9.18.1. NAME

`encoding` - displays and sets the encoding for cache entries.

9.18.2. DESCRIPTION

Sets a default encoding for **put** and **get** operations on a cache. If no argument is specified, the **encoding** command displays the current encoding.

Valid encodings use standard MIME type (IANA media types) naming conventions, such as the following:

- `text/plain`
- `application/json`
- `application/xml`
- `application/octet-stream`

9.18.3. SYNOPSIS

encoding ['ENCODING']

9.18.4. EXAMPLE

`encoding application/json`

Configures the currently selected cache to encode entries as `application/json`.

9.18.5. SEE ALSO

`get(1)`, `put(1)`

9.19. GET(1)

9.19.1. NAME

get - retrieves entries from a cache.

9.19.2. SYNOPSIS

get ['OPTIONS'] **KEY**

9.19.3. OPTIONS

-c, --cache='NAME'

Specifies the cache from which to retrieve entries. Defaults to the currently selected cache.

9.19.4. EXAMPLE

get **hello** **-c mycache**

Retrieves the value of the key named **hello** from **mycache**.

9.19.5. SEE ALSO

query(1), put(1)

9.20. HELP(1)

9.20.1. NAME

help - prints manual pages for commands.

9.20.2. SYNOPSIS

help ['COMMAND']

9.20.3. EXAMPLE

help **get**

Prints the manual page for the **get** command.

9.20.4. SEE ALSO

version(1)

9.21. LOGGING(1)

9.21.1. NAME

logging - inspects and manipulates the `${infinispan.brand.name}` server runtime logging configuration.

9.21.2. SYNOPSIS

logging list-loggers

logging list-appenders

logging set ['OPTIONS'] [LOGGER_NAME]

logging remove **LOGGER_NAME**

9.21.3. LOGGING SET OPTIONS

-l, --level='OFF|TRACE|DEBUG|INFO|WARN|ERROR|ALL'

Specifies the logging level for the specific logger.

-a, --appender='APPENDER'

Specifies an appenders to set on the specific logger. The option can be repeated for multiple appenders.



calling **logging set** without a logger name will modify the root logger.

9.21.4. EXAMPLES

logging list-loggers

Lists all available loggers

logging set --level=DEBUG --appenders=FILE org.infinispan

Sets the log level for the **org.infinispan** logger to **DEBUG** and configures it to use the **FILE** appender.

9.22. LS(1)

9.22.1. NAME

ls - lists resources for the current path or a given path.

9.22.2. SYNOPSIS

ls ['PATH']

9.22.3. EXAMPLES

ls caches

Lists the available caches.

ls ../

Lists parent resources.

9.22.4. SEE ALSO

cd(1)

9.23. MIGRATE(1)

9.23.1. NAME

migrate - migrates data from one version of `${infinispan.brand.name}` to another.

9.23.2. SYNOPSIS

migrate cluster synchronize

migrate cluster disconnect

9.23.3. DESCRIPTION

Use the `migrate` command to migrate data from one version of `${infinispan.brand.name}` to another.

9.23.4. COMMAND SYNOPSIS

Migrate clusters

migrate cluster synchronize

Synchronize data between the source cluster and the target cluster.

migrate cluster disconnect

Disconnects the target cluster from the source cluster.

9.23.5. COMMON OPTIONS

These options apply to all commands:

-h, --help

Displays a help page for the command or sub-command.

9.23.6. CLUSTER SYNCHRONIZE OPTIONS

-c, --cache='name'

The name of the cache to synchronize.

-b, --read-batch='num'

The amount of entries to process in a batch. Defaults to 10000.

-t, --threads='num'

The number of threads to use. Defaults to the number of cores on the server.

9.23.7. CLUSTER DISCONNECT OPTIONS

-c, --cache='name'

The name of the cache to disconnect from the source.

9.24. PATCH(1)

9.24.1. NAME

patch - manages server patches.

9.24.2. DESCRIPTION

List, describe, install, rollback, and create server patches.

Patches are zip archive files that contain artifacts to upgrade servers and resolve issues or add new features. Patches can apply target versions to multiple server installations with different versions.

9.24.3. SYNOPSIS

patch ls

patch install 'patch-file'

patch describe 'patch-file'

patch rollback

patch create 'patch-file' 'target-server' 'source-server-1' ['source-server-2'...]

9.24.4. PATCH LIST OPTIONS

--server='path/to/server'

Sets the path to a target server outside the current server home directory.

-v, --verbose

Shows the content of each installed patch, including information about individual files.

9.24.5. PATCH INSTALL OPTIONS

--dry-run

Shows the operations that the patch performs without applying any changes.

--server='path/to/server'

Sets the path to a target server outside the current server home directory.

9.24.6. PATCH DESCRIBE OPTIONS

-v, --verbose

Shows the content of the patch, including information about individual files

9.24.7. PATCH ROLLBACK OPTIONS

--dry-run

Shows the operations that the patch performs without applying any changes.

--server='path/to/server'

Sets the path to a target server outside the current server home directory.

9.24.8. PATCH CREATE OPTIONS

-q, --qualifier='name'

Specifies a descriptive qualifier string for the patch; for example, 'one-off for issue nnnn'.

9.24.9. EXAMPLES

`patch ls`

Lists the patches currently installed on a server in order of installation.

`patch install mypatch.zip`

Installs "mypatch.zip" on a server in the current directory.

`patch install mypatch.zip --server=/path/to/server/home`

Installs "mypatch.zip" on a server in a different directory.

`patch describe mypatch.zip`

Displays the target version and list of source versions for "mypatch.zip".

`patch create mypatch.zip 'target-server' 'source-server-1' ['source-server-2'...]`

Creates a patch file named "mypatch.zip" that uses the version of the target server and applies to the source server versions.

`patch rollback`

Rolls back the last patch that was applied to a server and restores the previous version.

9.25. PUT(1)

9.25.1. NAME

put - adds or updates cache entries.

9.25.2. DESCRIPTION

Creates entries for new keys. Replaces values for existing keys.

9.25.3. SYNOPSIS

put ['OPTIONS'] **KEY** [VALUE]

9.25.4. OPTIONS

-c, --cache='NAME'

Specifies the name of the cache. Defaults to the currently selected cache.

-e, --encoding='ENCODING'

Sets the media type for the value.

-f, --file='FILE'

Specifies a file that contains the value for the entry.

-l, --ttl='TTL'

Sets the number of seconds before the entry is automatically deleted (time-to-live). Defaults to the value for **lifespan** in the cache configuration if **0** or not specified. If you set a negative value, the entry is never deleted.

-i, --max-idle='MAXIDLE'

Sets the number of seconds that the entry can be idle. If a read or write operation does not occur for an entry after the maximum idle time elapses, the entry is automatically deleted. Defaults to the value for **maxIdle** in the cache configuration if **0** or not specified. If you set a negative value, the entry is never deleted.

-a, --if-absent=[true | false]

Puts an entry only if it does not exist.

9.25.5. EXAMPLES

```
put -c mycache hello world
```

Adds the **hello** key with a value of **world** to the **mycache** cache.

```
put -c mycache -f myfile -i 500 hola
```

Adds the **hola** key with the value from the contents of **myfile**. Also sets a maximum idle of **500** seconds.

9.25.6. SEE ALSO

`get(1)`, `remove(1)`

9.26. QUERY(1)

9.26.1. NAME

query - retrieves entries that match Ickle query strings.

9.26.2. SYNOPSIS

query ['OPTIONS'] **QUERY_STRING**

9.26.3. OPTIONS

-c, --cache='NAME'

Specifies the cache to query. Defaults to the currently selected cache.

--max-results='MAX_RESULTS'

Sets the number of results to return. The default is **10**.

-o, --offset='OFFSET'

Specifies the index of the first result to return. The default is **0**.

--query-mode='QUERY_MODE'

Specifies how the server executes the query. Values are **FETCH** and **BROADCAST**. The default is **FETCH**.

9.26.4. EXAMPLES

```
query "from org.infinispan.rest.search.entity.Person p where p.gender = 'MALE'"
```

Queries the currently selected cache to return entries from a Protobuf **Person** entity where the gender datatype is **MALE**.

9.26.5. SEE ALSO

schema(1)

9.27. QUIT(1)

9.27.1. NAME

quit - exits the command line interface.

9.27.2. SYNOPSIS

quit

exit and **bye** are command aliases.

9.27.3. EXAMPLE

```
quit
```

Ends the CLI session.

```
exit
```

Ends the CLI session.

bye

Ends the CLI session.

9.27.4. SEE ALSO

disconnect(1), shutdown(1)

9.28. REMOVE(1)

9.28.1. NAME

remove - deletes entries from a cache.

9.28.2. SYNOPSIS

remove **KEY** ['OPTIONS']

9.28.3. OPTIONS

--cache='NAME'

Specifies the cache from which to remove entries. Defaults to the currently selected cache.

9.28.4. EXAMPLE

remove --cache=mycache hola

Deletes the **hola** entry from the **mycache** cache.

9.28.5. SEE ALSO

cache(1), drop(1), clearcache(1)

9.29. RESET(1)

9.29.1. NAME

reset - restores the initial values of counters.

9.29.2. SYNOPSIS

reset ['COUNTER_NAME']

9.29.3. EXAMPLE

reset cnt_a

Resets the **cnt_a** counter.

9.29.4. SEE ALSO

`add(1)`, `cas(1)`, `drop(1)`

9.30. SCHEMA(1)

9.30.1. NAME

`schema` - uploads and registers protobuf schemas.

9.30.2. SYNOPSIS

schema ['OPTIONS'] *SCHEMA_NAME*

9.30.3. OPTIONS

-u, --upload='FILE'

Uploads a file as a protobuf schema with the given name.

9.30.4. EXAMPLE

schema --upload=person.proto person.proto

Registers a *person.proto* Protobuf schema.

9.30.5. SEE ALSO

`query(1)`

9.31. SERVER(1)

9.31.1. NAME

`server` - server configuration and state management.

9.31.2. DESCRIPTION

The **server** command describes and manages server endpoint connectors and datasources and retrieves aggregated diagnostic reports about both the server and host.

Reports provide details about CPU, memory, open files, network sockets and routing, threads, in addition to configuration and log files.

9.31.3. SYNOPSIS

server report

server connector ls

server connector describe 'connector-name'

server connector start 'connector-name'

server connector stop 'connector-name'

server connector ipfilter ls 'connector-name'

server connector ipfilter set 'connector-name' --rules='[ACCEPT|REJECT]/cidr',...

server connector ipfilter clear 'connector-name'

server datasource ls

server datasource test 'datasource-name'

9.31.4. SERVER CONNECTOR IPFILTER OPTIONS

--rules='[ACCEPT|REJECT]/cidr',...

One or more IP filtering rules.

9.31.5. EXAMPLES

server report

Obtains a server report, including information about network, threads, memory, etc.

server connector ls

Lists all available connectors on the server.

server connector describe endpoint-default

Shows information about the specified connector, including host, port, local and global connections, IP filtering rules.

server connector stop my-hotrod-connector

Stops a connector dropping all established connections across the cluster. This command will be refused if attempting to stop the connector which is handling the request.

server connector start my-hotrod-connector

Starts a connector so that it can accept connections across the cluster.

server connector ipfilter ls my-hotrod-connector

Lists all IP filtering rules active on a connector across the cluster.

server connector ipfilter set my-hotrod-connector --rules=ACCEPT/192.168.0.0/16,REJECT/10.0.0.0/8 Sets IP filtering rules on a connector across the cluster. Replaces all existing rules. This command will be refused if one of the rejection rules matches the address of the connection on which it is invoked.

server connector ipfilter clear my-hotrod-connector

Removes all IP filtering rules on a connector across the cluster.

`server datasource ls`

Lists all available datasources on the server.

`server datasource test my-datasource`

Performs a test connection on the datasource.

9.32. SHUTDOWN(1)

9.32.1. NAME

shutdown - stops running servers or brings clusters down gracefully.

9.32.2. SYNOPSIS

shutdown server ['SERVERS']

shutdown cluster

9.32.3. EXAMPLES

`shutdown server`

Stops the server to which the CLI is connected.

`shutdown server my_server01`

Stops the server with hostname `my_server01`.

`shutdown cluster`

Stores cluster state, persists entries if you use a cache store, and stops all nodes.

9.32.4. SEE ALSO

connect(1), disconnect(1), quit(1)

9.33. SITE(1)

9.33.1. NAME

site - manages backup locations and performs cross-site replication operations.

9.33.2. SYNOPSIS

site status ['OPTIONS']

site bring-online ['OPTIONS']

site take-offline ['OPTIONS']

site push-site-state ['OPTIONS']

site cancel-push-state ['OPTIONS']

site cancel-receive-state ['OPTIONS']

site push-site-status ['OPTIONS']

site state-transfer-mode get|set ['OPTIONS']

site name

site view

9.33.3. OPTIONS

--cache='CACHE_NAME'

Specifies a cache.

--site='SITE_NAME'

Specifies a backup location.

9.33.4. STATE TRANSFER MODE OPTIONS

--mode='MODE'

Sets the state transfer mode. Values are **MANUAL** (default) or **AUTO**.

9.33.5. EXAMPLES

site status --cache=mycache

Returns the status of all backup locations for **mycache**.

site status --cache=mycache --site=NYC

Returns the status of **NYC** for **mycache**.

site bring-online --cache=mycache --site=NYC

Brings the site **NYC** online for **mycache**.

site take-offline --cache=mycache --site=NYC

Takes the site **NYC** offline for **mycache**.

site push-site-state --cache=mycache --site=NYC

Backs up caches to remote backup locations.

site push-site-status --cache=mycache

Displays the status of the operation to backup **mycache**.

site cancel-push-state --cache=mycache --site=NYC

Cancels the operation to backup **mycache** to **NYC**.

site cancel-receive-state --cache=mycache --site=NYC

Cancels the operation to receive state from **NYC**.

`site clear-push-state-status --cache=myCache`

Clears the status of the push state operation for `mycache`.

`site state-transfer-mode get --cache=myCache --site=NYC`

Retrieves the state transfer mode for `mycache` to `NYC`.

`site state-transfer-mode set --cache=myCache --site=NYC --mode=AUTO`

Configures automatic state transfer for `mycache` to `NYC`.

`site name`

Returns the name of the local site. If cross-site replication is not configured, the name of the local site is always "local".

`site view`

Returns a list of names for all sites or an empty list ("[]") if cross-site replication is not configured.

9.34. STATS(1)

9.34.1. NAME

`stats` - displays statistics about resources.

9.34.2. SYNOPSIS

`stats` [`PATH`]

9.34.3. EXAMPLES

`stats //containers/default`

Displays statistics about the default container.

`stats //containers/default/caches/mycache`

Displays statistics about the `mycache` cache.

9.34.4. SEE ALSO

`cd(1)`, `ls(1)`, `describe(1)`

9.35. TASK(1)

9.35.1. NAME

`task` - executes and uploads server-side tasks and scripts

9.35.2. SYNOPSIS

`task upload --file='script' 'TASK_NAME'`

`task exec` [`TASK_NAME`]

9.35.3. EXAMPLES

```
task upload --file=hello.js hello
```

Uploads a script from a `hello.js` file and names it `hello`.

```
task exec @@cache@names
```

Runs a task that returns available cache names.

```
task exec hello -Pgreetee=world
```

Runs a script named `hello` and specifies the `greetee` parameter with a value of `world`.

9.35.4. OPTIONS

-P, --parameters='PARAMETERS'

Passes parameter values to tasks and scripts.

-f, --file='FILE'

Uploads script files with the given names.

9.35.5. SEE ALSO

ls(1)

9.36. UNALIAS(1)

9.36.1. NAME

unalias - deletes aliases.

9.36.2. SYNOPSIS

unalias 'ALIAS-NAME'

9.36.3. EXAMPLES

```
unalias q
```

Deletes the `q` alias.

9.36.4. SEE ALSO

config(1), alias(1)

9.37. USER(1)

9.37.1. NAME

user - manages `${infinispan.brand.name}` users in property security realms.

9.37.2. SYNOPSIS

user ls

user create 'username'

user describe 'username'

user remove 'username'

user password 'username'

user groups 'username'

user encrypt-all

user roles ls 'principal'

user roles grant --roles='role1'[, 'role2'...] 'principal'

user roles deny --roles='role1'[, 'role2'...] 'principal'

9.37.3. DESCRIPTION

Manage users in property realms with the **ls**, **create**, **describe**, **remove**, **password**, **groups** and **encrypt-all** subcommands. List and modify principal to role mappings with the **roles** subcommand when using the cluster role mapper for authorization.

9.37.4. COMMAND SYNOPSIS

user ls

Lists the users or groups which are present in the property file.

user create 'username'

Creates a user after prompting for a password.

user describe 'username'

Describes a user, including its username, realm and any groups it belongs to.

user remove 'username'

Removes the specified user from the property file.

user password 'username'

Changes the password for a user.

user groups 'username'

Sets the groups to which a user belongs.

user encrypt-all

Encrypt all passwords in a plain-text user property file.

user roles ls 'principal'

Lists all roles of the specified principal (user or group).

user roles grant --roles='role1'[, 'role2'...] 'principal'

Grants one or more roles to a principal.

user roles deny --roles='role1'[, 'role2'...] 'principal'

Denies one or more roles to a principal.

9.37.5. COMMON OPTIONS

These options apply to all commands:

-h, --help

Displays a help page for the command or sub-command.

-s, --server-root='path-to-server-root'

The path to the server root. Defaults to `server`.

-f, --users-file='users.properties'

The name of the property file which contains the user passwords. Defaults to `users.properties`.

-w, --groups-file='groups.properties'

The name of the property file which contains the user to groups mapping. Defaults to `groups.properties`.

9.37.6. USER CREATE/MODIFY OPTIONS

-a, --algorithms

Specifies the algorithms used to hash the password.

-g, --groups='group1,group2,...'

Specifies the groups to which the user belongs.

-p, --password='password'

Specifies the user's password.

-r, --realm='realm'

Specifies the realm name.

--plain-text

Whether passwords should be stored in plain-text (not recommended).

9.37.7. USER LS OPTIONS

--groups

Shows a list of groups instead of the users.

9.37.8. USER ENCRYPT-ALL OPTIONS

-a, --algorithms

Specifies the algorithms used to hash the password.

9.38. VERSION(1)

9.38.1. NAME

version - displays the server version and CLI version.

9.38.2. SYNOPSIS

version

9.38.3. EXAMPLE

`version`

Returns the version for the server and the CLI.

9.38.4. SEE ALSO

`help(1)`