acl-setuser

- ACL rules
  - Command rules
  - User management rules
- Examples

Create an ACL user with the specified rules or modify the rules of an existing user.

Manipulate Valkey ACL users interactively. If the username does not exist, the command creates the username without any privilege. It then reads from left to right all the rules provided as successive arguments, setting the user ACL rules as specified. If the user already exists, the provided ACL rules are simply applied *in addition* to the rules already set. For example:

```
ACL SETUSER virginia on allkeys +set
```

The above command creates a user called `virginia` who is active(the *on* rule), can access any key (*allkeys* rule), and can call the set command (*+set* rule). Then, you can use another `ACL SETUSER` call to modify the user rules:

```
ACL SETUSER virginia +get
```

The above rule applies the new rule to the user `virginia`, so other than `SET`, the user `virginia` can now also use the `GET` command.

ACL rules can also be grouped into multiple distinct sets of rules, called *selectors*. Selectors are added by wrapping the rules in parentheses and providing them just like any other rule. In order to execute a command, either the root permissions (rules defined outside of parenthesis) or any of the selectors (rules defined inside parenthesis) must match the given command. For example:

```
ACL SETUSER virginia on +GET allkeys (+SET ~app1*)
```

This sets a user with two sets of permissions, one defined on the user and one defined with a selector. The root user permissions only allow executing the get command, but can be executed on any keys. The selector then grants a secondary set of permissions: access to the `SET` command to be executed on any key that starts with `app1`. Using multiple selectors allows you to grant permissions that are different depending on what keys are being accessed.

Permissions can also be defined once in a named *role*, created with `ACL SETROLE`, and assigned to any number of users with the `role=` rule:

```
ACL SETROLE reader ~app:* +@read
ACL SETUSER virginia on >password role=reader
```

A user's own permissions are combined with those of its roles, so a user can add permissions on top of a role but cannot take away what a role grants.

When we want to be sure to define a user from scratch, without caring if it had previously defined rules associated, we can use the special rule `reset` as first rule, in order to flush all the other existing rules:

```
ACL SETUSER antirez reset [... other rules ...]
```

After resetting a user, its ACL rules revert to the default: inactive, passwordless, can't execute any command nor access any key or channel:

```
127.0.0.1:6379> ACL SETUSER antirez reset
OK
127.0.0.1:6379> ACL LIST
1) "user antirez off -@all"
```

ACL rules are either words like "on", "off", "reset", "allkeys", or are special rules that start with a special character, and are followed by another string (without any space in between), like "+SET".

The following documentation is a reference manual about the capabilities of this command, however our ACL tutorial may be a more gentle introduction to how the ACL system works in general.

ACL rules

Valkey ACL rules are split into two categories: rules that define command permissions or *command rules*, and rules that define the user state or *user management rules*. This is a list of all the supported Valkey ACL rules:

Command rules

- `~<pattern>`: Adds the specified key pattern (glob style pattern, like in the `KEYS` command), to the list of key patterns accessible by the user. This grants both read and

write permissions to keys that match the pattern. You can add multiple key patterns to the same user. Example: `~objects:*`

- `%R~<pattern>`: Adds the specified read key pattern. This behaves similar to the regular key pattern but only grants permission to read from keys that match the given pattern. See key permissions for more information.
- `%W~<pattern>`: Adds the specified write key pattern. This behaves similar to the regular key pattern but only grants permission to write to keys that match the given pattern. See key permissions for more information.
- `%RW~<pattern>`: Alias for `~<pattern>`.
- `allkeys`: Alias for `~*`, it allows the user to access all the keys.
- `resetkeys`: Removes all the key patterns from the list of key patterns the user can access.
- `&<pattern>`: Adds the specified glob style pattern to the list of Pub/Sub channel patterns accessible by the user. You can add multiple channel patterns to the same user. Example: `&chatroom:*`
- `allchannels`: Alias for `&*`, it allows the user to access all Pub/Sub channels.
- `resetchannels`: Removes all channel patterns from the list of Pub/Sub channel patterns the user can access.
- `+<command>`: Adds the command to the list of commands the user can call. Can be used with `|` for allowing subcommands (e.g "+config|get").
- `+@<category>`: Adds all the commands in the specified category to the list of commands the user is able to execute. Example: `+@string` (adds all the string commands). For a list of categories, check the `ACL CAT` command.
- `+<command>|first-arg`: Allows a specific first argument of an otherwise disabled command. It is only supported on commands with no sub-commands, and is not allowed as negative form like -SELECT|1, only additive starting with "+". This feature is deprecated and may be removed in the future.
- `allcommands`: Alias of `+@all`. Adds all the commands there are in the server, including *future commands* loaded via module, to be executed by this user.
- `-<command>`: Remove the command to the list of commands the user can call. It can be used with `|` for blocking subcommands (e.g., "-config|set").
- `-@<category>`: Like `+@<category>` but removes all the commands in the category instead of adding them.
- `nocommands`: Alias for `-@all`. Removes all the commands, and the user is no longer able to execute anything.
- `db=<id>[,<id>...]`: Sets the specified database id(s) as the databases the selector is allowed to access. Multiple ids are separated by commas, e.g. `db=0,1,2`; at least one id is required. Clears the `alldbs` flag and any previously configured database ids on the selector. See database permissions for more information.
- `alldbs`: Allows the selector to access all databases. This is the default for new selectors and is omitted from generated ACL strings.
- `resetdbs`: Removes all databases from the list of allowed databases and clears the `alldbs` flag.

User management rules

- `on`: Set the user as active, it will be possible to authenticate as this user using `AUTH <username> <password>`.
- `off`: Set user as not active, it will be impossible to log as this user. Please note that if a user gets disabled (set to off) after there are connections already authenticated with such a user, the connections will continue to work as expected. To also kill the old connections you can use `CLIENT KILL` with the user option. An alternative is to delete the user with `ACL DELUSER`, that will result in all the connections authenticated as the deleted user to be disconnected.
- `nopass`: The user is set as a *no password* user. It means that it will be possible to authenticate as such user with any password. By default, the `default` special user is set as "nopass". The `nopass` rule will also reset all the configured passwords for the user.
- `>password`: Adds the specified clear text password as a hashed password in the list of the users passwords. Every user can have many active passwords, so that password rotation will be simpler. The specified password is not stored as clear text inside the server. Example: `>mypassword`.
- `#<hashedpassword>`: Adds the specified hashed password to the list of user passwords. A hashed password is hashed with SHA256 and translated into a hexadecimal string. Example: `#c3ab8ff13720e8ad9047dd39466b3c8974e592c2fa383d4a3960714caef0c4f2`.
- `<password`: Like `>password` but removes the password instead of adding it.
- `!<hashedpassword>`: Like `#<hashedpassword>` but removes the password instead of adding it.
- `(<rule list>)`: Creates a new selector to match rules against. Selectors are evaluated after the user permissions, and are evaluated according to the order they are defined. If a command matches either the user permissions or any selector, it is allowed. See selectors for more information.
- `clearselectors`: Deletes all of the selectors attached to the user.
- `role=<role>[,<role> ...]`: Assigns the named roles to the user, replacing any role the user already had. At least one role has to be named, and every named role must already exist (see `ACL SETROLE`). The permissions of each role are combined with those of the user using OR logic, the same way selectors are. See roles for more information. Example: `role=reader,writer`. (Added in version 9.2)
- `resetroles`: Removes every role from the user. (Added in version 9.2)
- `reset`: Removes any capability from the user. They are set to off, without passwords, without roles, unable to execute any command, unable to access any key.

Examples

```
127.0.0.1:6379> ACL SETUSER alan allkeys +@string +@set -SADD >alanpassword
OK
127.0.0.1:6379> ACL SETUSER antirez heeyyyy
(error) ERR Error in ACL SETUSER modifier 'heeyyyy': Syntax error
```

Restrict a user to a subset of databases:

```
127.0.0.1:6379> ACL SETUSER alice on +@all ~* db=0,1 nopass
OK
127.0.0.1:6379> ACL LIST
1) "user alice on nopass ~* resetchannels db=0,1 +@all"
2) "user default on nopass ~* &* +@all"
```

A later `db=` rule replaces the previous database list:

```
127.0.0.1:6379> ACL SETUSER alice db=2,3
OK
127.0.0.1:6379> ACL LIST
1) "user alice on nopass ~* resetchannels db=2,3 +@all"
2) "user default on nopass ~* &* +@all"
```

Different selectors can grant access on different databases:

```
127.0.0.1:6379> ACL SETUSER bob on nopass (db=0,1 +@write +select ~*) (db=2,3 +@read +select
~*)
OK
127.0.0.1:6379> ACL LIST
1) "user bob on nopass resetchannels -@all (~* resetchannels db=0,1 -@all +@write +select) (~*
resetchannels db=2,3 -@all +@read +select)"
2) "user default on nopass ~* &* +@all"
```

The `ACL LIST` output shows three sets of permissions for `bob`: the root permissions (`-@all`, with implicit `alldbs`) and two selectors, one for each parenthesized rule set. A command is allowed when the root permissions or any selector matches it. See selectors for more information.

□□□ #1
□ □□□ □□□ 2026-10-31 15:09:29 UTC
□ □□□ □□□ 2026-10-31 15:09:30 UTC