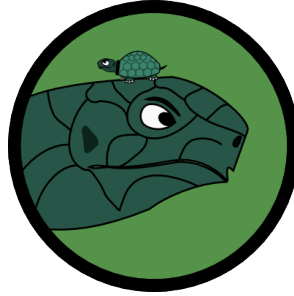

Archelon Documentation

Release 0.6.0

Carson Gee

August 23, 2015

1	Archelon Client	3
1.1	Installation	3
1.2	Web Enabled History	3
1.3	Keyboard Shortcuts	3
1.4	Within Menus	4
2	Archelon Client API	5
2.1	Data Module	5
2.2	Search Module	7
2.3	Command Module	8
3	Archelon Daemon	9
3.1	Installation and Configuration	9
3.2	Wiring Up to Elasticsearch	10
3.3	Running in Production	10
4	Archelon Daemon API	13
4.1	Abstract Base Class for Data Storage	13
4.2	In Memory Data Storage	14
4.3	Elastic Search Data Storage	15
4.4	Configuration Module	15
4.5	Log Module	16
4.6	Utility Module	16
4.7	Web Module	16
5	Release Notes	17
5.1	Version 0.6.0	17
5.2	Version 0.5.0	17
5.3	Version 0.4.1	17
5.4	Version 0.4.0	17
5.5	Version 0.3.1	18
5.6	Version 0.3.0	18
5.7	Version 0.2.1	18
5.8	Version 0.2.0	18
5.9	Version 0.1.0	18
6	Developer Information	19
7	Indices and tables	21



Do you want to share shell command history across multiple servers with a decent interface? If you have a ton of servers and lots of arcane commands with a half dozen parameters, I'm guessing you do. This is my attempt at solving this problem using a curses client for searching paired with a simple API and Web server backed by a nice search engine (Elasticsearch).

**Archelon**

python setup.py | [Reveal Token](#)

Show 25 entries

Command	Host	Time	Action
docker-compose run web python setup.py test --cov --lint	18.131.0.46	11:53.38.9am 05.26.20 15	✕
docker-compose run web python setup.py test --cov --pep8	18.131.0.46	11:53.38.8am 05.26.20 15	✕
docker-compose run web python setup.py test --cov --flakes	18.131.0.46	11:53.38.4am 05.26.20 15	✕
docker-compose run web python setup.py test --cov --pylint	18.131.0.46	11:53.38.4am 05.26.20 15	✕
python setup.py test_requirements.txt	18.131.0.46	11:16.16.7am 04.07.20 15	✕

Fig. 1: The server side Web interface showing search, sort, and delete bad entries

```
Archelon: Reverse Search
Search  python setup.py
python setup.py sdist upload
python setup.py sdist
python setup.py register
python setup.py test --cov
docker-compose run web python setup.py test --cov --lint
docker-compose run web python setup.py test --cov --pep8
docker-compose run web python setup.py test --cov --pylint
docker-compose run web python setup.py test --cov --flakes
python setup.py install
python setup.py test
python setup.py test --cov --flakes --pep8
git commit -m 'Per requirements for flask extensions, support running tests with `python setup.py test`'
python setup.py test --coverage --pep8 --flakes
docker-compose run web python setup.py test
python setup.py test --cov --flakes
python setup.py upload
coverage run python setup.py test
python setup.py tests
python setup.py test --cov --flake
python setup.py test_requirements.txt
python setup.py test --flakes --cov --pep8
python setup.py test --flakes --pep8 --coverage
python setup.py test --flakes --pep8
python setup.py test --coverage
python setup.py tes
python setup.py test | grep response_valid
- more -
Command  python setup.py sdist upload
^X: Menu  Cancel OK
```

Fig. 2: The curses client for searching and sorting history. It can be used without the server backend if you just want to try it out and not get a Web server running.

Archelon Client

Curses based application for command history that can be wired up to a Web server (archelond) for shared shell history across multiple hosts.

1.1 Installation

```
pip install archelonc
```

Once that is setup you can try it out by running `. archelon` this uses your existing shell history to let you try out the client. To make this work more like the bash reverse history search via `C-r` I recommend adding:

```
bind '"\033a":'. archelon\n"
```

this will launch the reverse search of archelon via Alt-A.

1.2 Web Enabled History

From here you can use archelon as is, but the cool part really start when you install archelond and wire the client up to use that project for shared and indexed shell history. To configure the client side after you have setup the server, you just need to add two environment variables to the mix.

- `ARCHELON_URL` - Web URL to your archelond installation
- `ARCHELON_TOKEN` - The API token for your user. You can get this by going to <https://your.archelond.domain/api/v1/token> and logging in with the username and password you've created.

Add those to `.bashrc`, `.profile`, or whichever shell startup you are using and it will be hooked up to the Web server. You can verify this and populate your Web history by running the `archelon_import` command which will import your current computers history.

1.3 Keyboard Shortcuts

Within the client curses application, there are a few handy keyboard shortcuts.

Alt-o This presses the Ok button and runs whatever command is in the `command` field.

Alt-c This presses the cancel button and exits out of the application without running a command.
`Ctrl-C` also works, but currently has a nasty exception message.

Ctrl-x This brings up the menu for doing things like changing the order of the search results.

1.4 Within Menus

Within the menu there are also keyboard shortcuts. And are executed emacs style, i.e. `Ctrl-x Ctrl-f` to set sorting order to oldest->newest. So far those are:

Ctrl-f Sort results from oldest to newest

Ctrl-r Default order. Sort results from newest to oldest.

Archelon Client API

For convenient reference in development, here are the Archelon client API docs.

2.1 Data Module

Data modeling for command history to be modular

exception `archelonc.data.ArcheloncAPIException`

Bases: `archelonc.data.ArcheloncException`

API exception occurred.

exception `archelonc.data.ArcheloncConnectionException`

Bases: `archelonc.data.ArcheloncException`

Connection exception class.

exception `archelonc.data.ArcheloncException`

Bases: `exceptions.Exception`

Base archelonc exception class.

class `archelonc.data.HistoryBase`

Bases: `object`

Base class of what all backend command history searches should use.

search_forward (*term*, *page=0*)

Return a list of commands that is in forward time order. i.e oldest first.

If paging is needed, the page parameter is available.

search_reverse (*term*, *page=0*)

Return a list of commands that is in reverse time order. i.e newest first.

If paging is needed, the page parameter is available.

class `archelonc.data.LocalHistory`

Bases: `archelonc.data.HistoryBase`

Use local `.bash_history` for doing searches

search_forward (*term*, *page=0*)

Return a list of commands that is in forward time order. i.e oldest first.

search_reverse (*term*, *page=0*)

Return reversed filtered list by term

class `archelonc.data.WebHistory(url, token)`
Bases: `archelonc.data.HistoryBase`
Use RESTful API to do searches against archelond.
SEARCH_URL = `u'/api/v1/history'`

add (*command*)
Post a command to the remote server using the API

Raises

- `ArcheloncConnectionException`
- `ArcheloncAPIException`

all (*page*)
Return the entire data set available, one page at a time

Raises

- `ArcheloncConnectionException`
- `ArcheloncAPIException`

bulk_add (*commands*)
Post a list of commands

Parameters **commands** (*list*) – List of commands to add to server.

Raises

- `ArcheloncConnectionException`
- `ArcheloncAPIException`

delete (*command*)
Deletes the command given on the server.

Raises

- `ArcheloncConnectionException`
- `ArcheloncAPIException`
- `ValueError`

Returns `None`

search_forward (*term, page=0*)
Return a list of commands that is in forward time order. i.e oldest first.

Raises

- `ArcheloncConnectionException`
- `ArcheloncAPIException`

search_reverse (*term, page=0*)
Make request to API with sort order specified and return the results as a list.

Raises

- `ArcheloncConnectionException`
- `ArcheloncAPIException`

2.2 Search Module

npyscreen based application for searching shell history

```
class archelonc.search.CommandBox(screen, **kwargs)
```

Bases: npyscreen.wgttitlefield.TitleText

Command Box widget

```
when_value_edited()
```

Mark myself as having been edited

```
class archelonc.search.Search
```

Bases: npyscreen.apNPSApplicationManaged.NPSAppManaged

Search application. Determines which form to show.

```
more = True
```

```
onStart()
```

Startup routine for the search application

```
page = 0
```

```
class archelonc.search.SearchBox(screen, begin_entry_at=16, field_width=None, value=None,  
use_two_lines=None, hidden=False, labelColor='LABEL', al-  
low_override_begin_entry_at=True, **keywords)
```

Bases: npyscreen.wgttitlefield.TitleText

Search box command, updates trigger deeper searching.

```
search(page=0)
```

Do the search and return the results

```
when_value_edited()
```

Do the search and filter the search result list based on what is returned

```
class archelonc.search.SearchForm(*args, **kwargs)
```

Bases: npyscreen.fmFormWithMenus.ActionFormWithMenus

Command history search form

```
afterEditing()
```

This is the only form to display, so set next to None

```
beforeEditing()
```

Set the edit index to the search box and tell it to preserve the value

```
create()
```

Build the form for searching

```
forward_order()
```

Change sort order to forward

```
on_cancel(*)
```

Drop out with a non 0 exit code so the wrapper doesn't execute anything

```
on_ok(*)
```

We just drop the command into a known file for the wrapper to pick up

```
reverse_order()
```

Change sort order to forward

```
class archelonc.search.SearchResult (screen, value='', highlight_color='CURSOR', highlight_whole_widget=False, invert_highlight_color=True,
                                     **keywords)
    Bases: npyscreen.wgtextbox.Textfield
    Search result item

    update (clear=True)
        Update search results include getting the next page.

class archelonc.search.SearchResults (*args, **keywords)
    Bases: npyscreen.wgmultiline.MultiLineAction
    MultiLine widget for displaying search results.

    actionHighlighted (act_on_this, key_press)
```

2.3 Command Module

Command line entry points for the archelon client.

```
archelonc.command.export_history()
    Pull all remote history down into a file specified or stdout if none is specified

archelonc.command.import_history()
    Import current shell's history into server

archelonc.command.search_form()
    Entry point to search history

archelonc.command.update()
    Capture diff between stored history and new history and upload the delta.
```

Archelon Daemon

This is the Web server side of archelon. Once it is all configured and wired up to archelonc it can be used to store your shell history from all your hosts.

It is a simple Flask app that is generally designed to be wired up to an elasticsearch host to provide a nicely indexed shell history, and should be deployable for free on heroku using an elasticsearch addon.

3.1 Installation and Configuration

```
pip install archelond
```

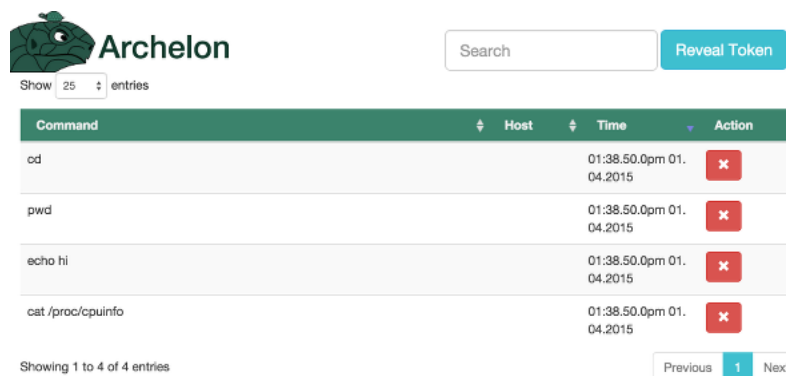
Security is obviously important for shell history, and to setup authentication we use basic authentication using apache htpasswd files as the user database. To add one for yourself and configure archelond to use it, run something like:

```
htpasswd -c ~/.htpasswd username
export ARCHELOND_HTPASSWD_PATH=~/.htpasswd
```

After that minimal setup we can try things out with just a simple command of:

```
archelond
```

Which will fire up the debug/development server using an in memory bash history data store that is very forgetful. Once it is up, you should be able to go <http://localhost:8580/>, login with the username and password you created in your htpasswd file, and see a lovely Web interface for searching and deleting your shell history similar to:



. It also provides a simple button to reveal the token you need in archelonc to connect the two together. To access the RESTful API side directly, you can check out the sample commands by visiting <http://localhost:8580/api/v1/history> or get your token for use with archelonc <http://localhost:8580/api/v1/token>.

3.2 Wiring Up to Elasticsearch

In order to have your history survive start ups we can use Elasticsearch. You can either install it locally, or grab it from an add-on on Heroku. Once you have the connection URL, we just need to add a couple environment variables to point at the service and set the storage provider class with something like:

```
export ARCHELOND_ELASTICSEARCH_URL='http://localhost:9200'
export ARCHELOND_ELASTICSEARCH_INDEX='history'
export ARCHELOND_DATABASE='ElasticData'
```

The index can be changed as desired, but it is the index in elasticsearch that will be used to store the history.

Note: archelond with the ElasticData can support multiple users as it uses the user in the document type

3.3 Running in Production

Running the archelond command is good for testing out, but to run it in production you will want to run it through a proper wsgi application server. As an example, we've added uwsgi in the requirements and it can be run in production with something like:

```
uwsgi --http :8580 -w archelond.web:app
```

and then a Web server like nginx proxying over https in order to further secure your shell history.

3.3.1 Running in Heroku

For heroku, it is very easy to setup the application part. Just create a requirements.txt file in the root of your repo with at least one line:

```
archelond
```

Setup a Procfile with:

```
web: uwsgi uwsgi.ini
```

and a uwsgi.ini that looks something like:

```
[uwsgi]
http-socket = :$(PORT)
master = true
processes = 10
die-on-term = true
module = archelond.web:app
memory-report = true
```

You also need to setup your secrets using heroku config:set commands. The vars that need to be set minimally for an elasticsearch version are:

```
ARCHELOND_DATABASE="ElasticData"
ARCHELOND_ELASTICSEARCH_INDEX="my_index"
ARCHELOND_ELASTICSEARCH_URL="http://example.com/elastic_search"
ARCHELOND_FLASK_SECRET="a_very_long_randomized_string"
ARCHELOND_HTTPASSWD="username:hashfromhttpasswd"
ARCHELOND_HTTPASSWD_PATH="htpasswd"
```

Note: I had to also add `-e git+https://github.com/elasticsearch/elasticsearch-py.git@master#egg=elasticsearch` to my requirements file because my elasticsearch server needed to specify https, username, and password. Currently the release version 1.2.0 didn't have that feature, but it is available in their master branch

Archelon Daemon API

For convenient reference in development, here are the Archelon Daemon API docs.

4.1 Abstract Base Class for Data Storage

Abstract base class for data stores

class `archelond.data.abstract.HistoryData` (*config*)

Bases: `object`

Abstract Data storage for command history

Abstract class implementation of a database for use with command history. Generally a command data item needs just two things, an ID and the command itself. It also needs order. See the `archelond.data.MemoryData` class as the simplest structure using an `collections.OrderedDict`.

An ID can be any string, and the concrete implementation of *HistoryData* is responsible for type casting it if needed.

It is also required implicitly that there is only one entry per command. Thus `add` ing the same command multiple times should result in the return of just one command when filtered by a term equal to that command.

add (*command*, *username*, *host*, ***kwargs*)

Add or update a command

Save (update or create) a command to the data store.

Parameters

- **command** (*str*) – The command to store
- **username** (*str*) – The username of the person adding it
- **host** (*str*) – The IP address of API caller

Returns **Command ID** – The id of the command stored

Return type `str`

all (*order*, *username*, *host*, ***kwargs*)

Unfiltered but ordered command history

Return the full data set as a list of dict structures in the specified order.

Parameters

- **order** (*str*) – An ordering from `ORDER_TYPES`
- **username** (*str*) – The username of the person adding it
- **host** (*str*) – The IP address of API caller

Returns A list of dictionaries where each dictionary must have at least a `command` key and an `id` key.

Return type list

delete (*command_id*, *username*, *host*, ***kwargs*)

Delete a command

Remove a command from the data store, raise a `KeyError` if it is not available to be deleted.

Parameters

- **command_id** (*str*) – Unique command identifier
- **username** (*str*) – The username of the person adding it
- **host** (*str*) – The IP address of API caller

filter (*term*, *order*, *username*, *host*, ***kwargs*)

Get a filtered by term and ordered command history

Parameters

- **term** (*str*) – The term being searched for/filtered by.
- **order** (*str*) – An ordering from `ORDER_TYPES`
- **username** (*str*) – The username of the person adding it
- **host** (*str*) – The IP address of API caller

Returns A list of dictionaries where each dictionary must have at least a `command` key and an `id` key.

Return type list

get (*command_id*, *username*, *host*, ***kwargs*)

Get a single command

Retrieve a single command by username and id. Raise a `KeyError` if the command does not exist.

Parameters

- **command_id** (*str*) – Unique command identifier
- **username** (*str*) – The username of the person adding it
- **host** (*str*) – The IP address of API caller

Returns **Command** – Dictionary with at least the keys `id` and `command`

Return type dict

4.2 In Memory Data Storage

In memory data store implementation for development and testing

```
class archelond.data.memory.MemoryData (config)
    Bases: archelond.data.abstract.HistoryData

    A quick in memory deduplicated structure for standalone testing and development.

    INITIAL_DATA = [u'cd', u'pwd', u'echo hi', u'cat /proc/cpuinfo']

    add (command, username, host, **kwargs)
        Append item to data list

    all (order, username, host, page=0, **kwargs)
        Simply rewrap the data structure, order, and return

    delete (command_id, username, host, **kwargs)
        Remove key from internal dictionary

    filter (term, order, username, host, page=0, **kwargs)
        Return filtered and reversed OrderedDict.

    get (command_id, username, host, **kwargs)
        Pull the specified command out of the data store.
```

4.3 Elastic Search Data Storage

ElasticSearch implementation of the data store. Currently the recommended default data store.

```
class archelond.data.elastic.ElasticData (config)
    Bases: archelond.data.abstract.HistoryData

    An ElasticSearch implementation of HistoryData. This is what should be used in production

    DOC_TYPE = u'history'

    NUM_RESULTS = 50

    add (command, username, host, **kwargs)
        Add the command to the index with a time stamp and id by hash of the command and append username to
        doc type for user separation of data.

    all (order, username, host, page=0, **kwargs)
        Just build a body with match all and return filter

    delete (command_id, username, host, **kwargs)
        Remove item from elasticsearch

    filter (term, order, username, host, body=None, page=0, **kwargs)
        Return filtered search that is ordered

    get (command_id, username, host, **kwargs)
        Pull one command out of elasticsearch
```

4.4 Configuration Module

Configure the flask application

4.5 Log Module

Configure logging

`archelond.log.configure_logging(app)`
Set the log level for the application

4.6 Utility Module

Classic utility module for removing repetitive tasks and such

`archelond.util.jsonify_code(src_object, status_code)`
Wrap jsonify with a status code option for jsonifying non-200 responses.

Parameters

- **src_object** (*serializable object*) – data structure to jsonify
- **status_code** (*int*) – Status code to send

Returns werkzeug response object with json MIME-type

4.7 Web Module

Main entry point for flask application

`archelond.web.history()`
POST=Add entry GET=Get entries with query

`archelond.web.history_item(cmd_id)`
Actions for individual command history items.

Updates, gets, or deletes a command from the active data store.

PUT: Takes a payload in either form or JSON request, and runs the add routine by passing the dictinoary minus command, username, and host as kwargs to the data stores add routine.

`archelond.web.index()`
Simple index view for documentation and navigation.

`archelond.web.run_server()`
If started from command line, rebuild object in debug mode and run directly

`archelond.web.token()`
Return the user token for API auth that is based off the flask secret and user password

`archelond.web.wsgi_app()`
Start flask application runtime

Release Notes

5.1 Version 0.6.0

- Converted to using tox with py.test as test runner.
- Documentation updates
- Added release notes page
- Added docker containers for client and server
- Python 3 compatibility for both client and server
- Switched to using flask-htpasswd for authentication
- All views protected with before_request
- Full test coverage in client and server
- Required at least Elasticsearch client 1.3.0 or greater

5.2 Version 0.5.0

- Results paging in both client and server
- Added unit tests for server
- Documentation corrections
- Allowed for empty htpasswd file, but logged as critical
- Pylint fixes

5.3 Version 0.4.1

- Corrected bad tar on pypi

5.4 Version 0.4.0

- Delete command option

- Favicon and style improvements
- Sphinx/RTD docs

5.5 Version 0.3.1

- Fixed an incorrectly cased javascript file include.

5.6 Version 0.3.0

5.6.1 Archelon Client

- Optimized interface
- Menus added for sorting

5.6.2 Archelon Server

- New index page with searchable history

5.7 Version 0.2.1

- PyPi version bump only

5.8 Version 0.2.0

- Added default value for httpasswd path
- Removed elastic search sniffing

5.9 Version 0.1.0

- Initial Release

Developer Information

For better or worse, archelon is two projects in one. To help with that though, you can still run tests with just `tox` after having `tox` installed, build just the docs with `tox -e docs`, or just run `archelond` with `tox -e archelond`.

If you want to develop in a nice containerized environment so you don't have to run `ElasticSearch` locally for example, there is also `docker-compose`. After running `pip install docker-compose` locally, you should be able to run `docker-compose up` and `ElasticSearch` and `archelond` will be running on port 8580 with the default credentials of `admin` and `pass`. Additionally, there is a secondary `docker-compose` file for the client that is connected to the server. This is a little more awkward because `docker-compose` isn't really setup to run interactive containers. To get this going, just run: `docker-compose -f docker-client.yml run archelonc`.

Indices and tables

- `genindex`
- `modindex`
- `search`

a

- `archelonc.command`, 8
- `archelonc.data`, 5
- `archelonc.search`, 7
- `archelond.config`, 15
- `archelond.data.abstract`, 13
- `archelond.data.elastic`, 15
- `archelond.data.memory`, 14
- `archelond.log`, 16
- `archelond.util`, 16
- `archelond.web`, 16

A

`actionHighlighted()` (archelonc.search.SearchResults method), 8
`add()` (archelonc.data.WebHistory method), 6
`add()` (archelonc.data.abstract.HistoryData method), 13
`add()` (archelonc.data.elastic.ElasticData method), 15
`add()` (archelonc.data.memory.MemoryData method), 15
`afterEditing()` (archelonc.search.SearchForm method), 7
`all()` (archelonc.data.WebHistory method), 6
`all()` (archelonc.data.abstract.HistoryData method), 13
`all()` (archelonc.data.elastic.ElasticData method), 15
`all()` (archelonc.data.memory.MemoryData method), 15
`archelonc.command` (module), 8
`archelonc.data` (module), 5
`archelonc.search` (module), 7
`ArcheloncAPIException`, 5
`ArcheloncConnectionException`, 5
`ArcheloncException`, 5
`archelonc.config` (module), 15
`archelonc.data.abstract` (module), 13
`archelonc.data.elastic` (module), 15
`archelonc.data.memory` (module), 14
`archelonc.log` (module), 16
`archelonc.util` (module), 16
`archelonc.web` (module), 16

B

`beforeEditing()` (archelonc.search.SearchForm method), 7
`bulk_add()` (archelonc.data.WebHistory method), 6

C

`CommandBox` (class in archelonc.search), 7
`configure_logging()` (in module archelonc.log), 16
`create()` (archelonc.search.SearchForm method), 7

D

`delete()` (archelonc.data.WebHistory method), 6
`delete()` (archelonc.data.abstract.HistoryData method), 14
`delete()` (archelonc.data.elastic.ElasticData method), 15

`delete()` (archelonc.data.memory.MemoryData method), 15
`DOC_TYPE` (archelonc.data.elastic.ElasticData attribute), 15

E

`ElasticData` (class in archelonc.data.elastic), 15
`export_history()` (in module archelonc.command), 8

F

`filter()` (archelonc.data.abstract.HistoryData method), 14
`filter()` (archelonc.data.elastic.ElasticData method), 15
`filter()` (archelonc.data.memory.MemoryData method), 15
`forward_order()` (archelonc.search.SearchForm method), 7

G

`get()` (archelonc.data.abstract.HistoryData method), 14
`get()` (archelonc.data.elastic.ElasticData method), 15
`get()` (archelonc.data.memory.MemoryData method), 15

H

`history()` (in module archelonc.web), 16
`history_item()` (in module archelonc.web), 16
`HistoryBase` (class in archelonc.data), 5
`HistoryData` (class in archelonc.data.abstract), 13

I

`import_history()` (in module archelonc.command), 8
`index()` (in module archelonc.web), 16
`INITIAL_DATA` (archelonc.data.memory.MemoryData attribute), 15

J

`jsonify_code()` (in module archelonc.util), 16

L

`LocalHistory` (class in archelonc.data), 5

M

MemoryData (class in archelond.data.memory), 14
more (archelonc.search.Search attribute), 7

N

NUM_RESULTS (archelond.data.elastic.ElasticData attribute), 15

O

on_cancel() (archelonc.search.SearchForm method), 7
on_ok() (archelonc.search.SearchForm method), 7
onStart() (archelonc.search.Search method), 7

P

page (archelonc.search.Search attribute), 7

R

reverse_order() (archelonc.search.SearchForm method), 7
run_server() (in module archelond.web), 16

S

Search (class in archelonc.search), 7
search() (archelonc.search.SearchBox method), 7
search_form() (in module archelonc.command), 8
search_forward() (archelonc.data.HistoryBase method), 5
search_forward() (archelonc.data.LocalHistory method), 5
search_forward() (archelonc.data.WebHistory method), 6
search_reverse() (archelonc.data.HistoryBase method), 5
search_reverse() (archelonc.data.LocalHistory method), 5
search_reverse() (archelonc.data.WebHistory method), 6
SEARCH_URL (archelonc.data.WebHistory attribute), 6
SearchBox (class in archelonc.search), 7
SearchForm (class in archelonc.search), 7
SearchResult (class in archelonc.search), 7
SearchResults (class in archelonc.search), 8

T

token() (in module archelond.web), 16

U

update() (archelonc.search.SearchResult method), 8
update() (in module archelonc.command), 8

W

WebHistory (class in archelonc.data), 6
when_value_edited() (archelonc.search.CommandBox method), 7
when_value_edited() (archelonc.search.SearchBox method), 7
wsgi_app() (in module archelond.web), 16