ArchiveBox

Release 0.4.0

Aug 19, 2020

Contents

Welcome to ArchiveBox!

Just getting started? Check out the Quickstart guide.

Need help with something? Ping us on Twitter or Github.

Want to join the community? See our Community Wiki page.

ARCHIVEBOX

"The open-source self-hosted internet archive."

Website | Github | Source | Bug Tracker

```
mkdir my-archive; cd my-archive/
pip install archivebox

archivebox init
archivebox add https://example.com
archivebox info
```

CHAPTER 1

Documentation

1.1 Intro

Quickstart | Demo | Github | Documentation | Info & Motivation | Community | Roadmap

ArchiveBox takes a list of website URLs you want to archive, and creates a local, static, browsable HTML clone of the content from those websites (it saves HTML, JS, media files, PDFs, images and more).

You can use it to preserve access to websites you care about by storing them locally offline. ArchiveBox imports lists of URLs, renders the pages in a headless, autheticated, user-scriptable browser, and then archives the content in multiple redundant common formats (HTML, PDF, PNG, WARC) that will last long after the originals disappear off the internet. It automatically extracts assets and media from pages and saves them in easily-accessible folders, with out-of-the-box support for extracting git repositories, audio, video, subtitles, images, PDFs, and more.

1.1.1 How does it work?

```
echo 'http://example.com' | ./archive
```

After installing the dependencies, just pipe some new links into the `./archive` command to start your archive.

ArchiveBox is written in Python 3.5 and uses wget, Chrome headless, youtube-dl, pywb, and other common unix tools to save each page you add in multiple redundant formats. It doesn't require a constantly running server or backend, just open the generated `output/index.html` in a browser to view the archive. It can import and export links as JSON (among other formats), so it's easy to script or hook up to other APIs. If you run it on a schedule and import from browser history or bookmarks regularly, you can sleep soundly knowing that the slice of the internet you care about will be automatically preserved in multiple, durable long-term formats that will be accessible for decades (or longer).

1.1.2 Quickstart

ArchiveBox has 3 main dependencies beyond `python3`: `wget`, `chromium`, and `youtube-dl`. To get started, you can install them manually using your system's package manager, use the automated helper script, or use the official

Docker container. All three dependencies are optional if disabled in settings.

```
# 1. Install dependencies (use apt on ubuntu, brew on mac, or pkg on BSD)
apt install python3 python3-pip git curl wget youtube-dl chromium-browser

# 2. Download ArchiveBox
git clone https://github.com/pirate/ArchiveBox.git && cd ArchiveBox

# 3. Add your first links to your archive
echo 'https://example.com' | ./archive                  # pass URLs to archive via␣
↪stdin

./archive https://getpocket.com/users/example/feed/all  # or import an RSS/JSON/XML/
↪TXT feed
```

One you've added your first links, open `output/index.html` in a browser to view the archive. DEMO: archive.sweeting.meFor more information, see the full Quickstart guide, Usage, and Configuration docs.

(`pip install archivebox` will be available in the near future, follow our Roadmap for progress)

1.1.3 Overview

Because modern websites are complicated and often rely on dynamic content, ArchiveBox archives the sites in **several different formats** beyond what public archiving services like Archive.org and Archive.is are capable of saving. Using multiple methods and the market-dominant browser to execute JS ensures we can save even the most complex, finicky websites in at least a few high-quality, long-term data formats.

ArchiveBox imports a list of URLs from stdin, remote URL, or file, then adds the pages to a local archive folder using wget to create a browsable HTML clone, youtube-dl to extract media, and a full instance of Chrome headless for PDF, Screenshot, and DOM dumps, and more...

Running `./archive` adds only new, unique links into `output/` on each run. Because it will ignore duplicates and only archive each link the first time you add it, you can schedule it to run on a timer and re-import all your feeds multiple times a day. It will run quickly even if the feeds are large, because it's only archiving the newest links since the last run. For each link, it runs through all the archive methods. Methods that fail will save `None` and be automatically retried on the next run, methods that succeed save their output into the data folder and are never retried/overwritten by subsequent runs. Support for saving multiple snapshots of each site over time will be added soon (along with the ability to view diffs of the changes between runs).

All the archived links are stored by date bookmarked in `output/archive/<timestamp>`, and everything is indexed nicely with JSON & HTML files. The intent is for all the content to be viewable with common software in 50 - 100 years without needing to run ArchiveBox in a VM.

Can import links from many formats:

```
echo 'http://example.com' | ./archive
./archive ~/Downloads/firefox_bookmarks_export.html
./archive https://example.com/some/rss/feed.xml
```

- Browser history or bookmarks exports (Chrome, Firefox, Safari, IE, Opera, and more)
- RSS, XML, JSON, CSV, SQL, HTML, Markdown, TXT, or any other text-based format
- Pocket, Pinboard, Instapaper, Shaarli, Delicious, Reddit Saved Posts, Wallabag, Unmark.it, OneTab, and more

See the Usage: CLI page for documentation and examples.

Saves lots of useful stuff for each imported link:

```
ls output/archive/<timestamp>/
```

- **Index:** `index.html` & `index.json` HTML and JSON index files containing metadata and details
- **Title:** `title` title of the site
- **Favicon:** `favicon.ico` favicon of the site
- **WGET Clone:** `example.com/page-name.html` wget clone of the site, with .html appended if not present
- **WARC:** `warc/<timestamp>.gz` gzipped WARC of all the resources fetched while archiving
- **PDF:** `output.pdf` Printed PDF of site using headless chrome
- **Screenshot:** `screenshot.png` 1440x900 screenshot of site using headless chrome
- **DOM Dump:** `output.html` DOM Dump of the HTML after rendering using headless chrome
- **URL to Archive.org:** `archive.org.txt` A link to the saved site on archive.org
- **Audio & Video:** `media/` all audio/video files + playlists, including subtitles & metadata with youtube-dl
- **Source Code:** `git/` clone of any repository found on github, bitbucket, or gitlab links
- *More coming soon! See the Roadmap...*

It does everything out-of-the-box by default, but you can disable or tweak individual archive methods via environment variables or config file.

If you're importing URLs with secret tokens in them (e.g Google Docs, CodiMD notepads, etc), you may want to disable some of these methods to avoid leaking private URLs to 3rd party APIs during the archiving process. See the Security Overview page for more details.

Key Features

- **Free & open source**, doesn't require signing up for anything, stores all data locally
- **Few dependencies** and simple command line interface
- **Comprehensive documentation**, active development, and rich community
- **Doesn't require a constantly-running server**, proxy, or native app
- Easy to set up **scheduled importing** **from multiple sources**
- Uses common, **durable,** ***long-term formats*** like HTML, JSON, PDF, PNG, and WARC
- **Suitable for paywalled / authenticated content** (can use your cookies)
- Can **run scripts during archiving** to scroll pages, close modals, expand comment threads, etc.
- Can also **mirror content to 3rd-party archiving services** automatically for redundancy

Background & Motivation

Vast treasure troves of knowledge are lost every day on the internet to link rot. As a society, we have an imperative to preserve some important parts of that treasure, just like we preserve our books, paintings, and music in physical libraries long after the originals go out of print or fade into obscurity.

Whether it's to resist censorship by saving articles before they get taken down or edited, or just to save a collection of early 2010's flash games you love to play, having the tools to archive internet content enables to you save the stuff you care most about before it disappears.

The balance between the permanence and ephemeral nature of content on the internet is part of what makes it beautiful. I don't think everything should be preserved in an automated fashion, making all content permanent and never removable, but I do think people should be able to decide for themselves and effectively archive specific content that they care about.

Comparison to Other Projects

Check out our community page for an index of web archiving initiatives and projects.

The aim of ArchiveBox is to go beyond what the Wayback Machine and other public archiving services can do, by adding a headless browser to replay sessions accurately, and by automatically extracting all the content in multiple redundant formats that will survive being passed down to historians and archivists through many generations.

User Interface & Intended Purpose

ArchiveBox differentiates itself from similar projects by being a simple, one-shot CLI inferface for users to ingest built feeds of URLs over extended periods, as opposed to being a backend service that ingests individual, manually-submitted URLs from a web UI.

An alternative tool pywb allows you to run a browser through an always-running archiving proxy which records the traffic to WARC files. ArchiveBox intends to support this style of live proxy-archiving using `pywb` in the future, but for now it only ingests lists of links at a time via browser history, bookmarks, RSS, etc.

Private Local Archives vs Centralized Public Archives

Unlike crawler software that starts from a seed URL and works outwards, or public tools like Archive.org designed for users to manually submit links from the public internet, ArchiveBox tries to be a set-and-forget archiver suitable for archiving your entire browsing history, RSS feeds, or bookmarks, including private/authenticated content that you wouldn't otherwise share with a centralized service. Also by having each user store their own content locally, we can save much larger portions of everyone's browsing history than a shared centralized service would be able to handle.

Storage Requirements

Because ArchiveBox is designed to ingest a firehose of browser history and bookmark feeds to a local disk, it can be much more disk-space intensive than a centralized service like the Internet Archive or Archive.today. However, as storage space gets cheaper and compression improves, you should be able to use it continuously over the years without having to delete anything. In my experience, ArchiveBox uses about 5gb per 1000 articles, but your milage may vary depending on which options you have enabled and what types of sites you're archiving. By default, it archives everything in as many formats as possible, meaning it takes more space than a using a single method, but more content is accurately replayable over extended periods of time. Storage requirements can be reduced by using a compressed/deduplicated filesystem like ZFS/BTRFS, or by setting `SAVE_MEDIA=False` to skip audio & video files.

Learn more

Join out our community chat hosted on IRC freenode.net:`#ArchiveBox`!

Whether you want learn which organizations are the big players in the web archiving space, want to find a specific open source tool for your web archiving need, or just want to see where archivists hang out online, our Community Wiki page serves as an index of the broader web archiving community. Check it out to learn about some of the coolest web archiving projects and communities on the web!

- Community Wiki
  - The Master Lists*Community-maintained indexes of archiving tools and institutions.*
  - Web Archiving Software*Open source tools and projects in the internet archiving space.*
  - Reading List*Articles, posts, and blogs relevant to ArchiveBox and web archiving in general.*
  - Communities*A collection of the most active internet archiving communities and initiatives.*
- Check out the ArchiveBox Roadmap and Changelog
- Learn why archiving the internet is important by reading the "On the Importance of Web Archiving" blog post.
- Or reach out to me for questions and comments via @theSquashSH on Twitter.

1.1.4 Documentation

We use the Github wiki system and Read the Docs for documentation.

You can also access the docs locally by looking in the `ArchiveBox/docs/` folder.

You can build the docs by running:

```
cd ArchiveBox
pipenv install --dev
sphinx-apidoc -o docs archivebox
cd docs/
make html
# then open docs/_build/html/index.html
```

Getting Started

- Quickstart
- Install
- Docker

Reference

- Usage
- Configuration
- Supported Sources
- Supported Outputs
- Scheduled Archiving
- Publishing Your Archive
- Chromium Install

- Security Overview
- Troubleshooting

More Info

- Roadmap
- Changelog
- Donations
- Background & Motivation
- Web Archiving Community

1.1.5 Screenshots

1.2 Getting Started

1.2.1 Quickstart

It only takes about 5 minutes to get up and running with ArchiveBox.

ArchiveBox officially supports **macOS**, **Ubuntu/Debian**, and **BSD**, but likely runs on many other systems. You can run it on any system that supports **Docker**, including Windows.

If you want to use Docker or Docker Compose to run ArchiveBox, see the [[Docker]] page.

First, we install the ArchiveBox dependencies, then we create a folder to store the archive data, and finally, we import the list of links to the archive by running `./archive <links_file>`.

1. Set up ArchiveBox

Clone the ArchiveBox repo and install its dependencies.

```
git clone https://github.com/pirate/ArchiveBox
cd ArchiveBox/
./bin/archivebox-setup  # script prompts for user confirmation before installing␣
↪anything
```

For more detail, including the manual setup and docker instructions, see the [[Install]] page.

2. Get your list of URLs to archive

Follow the links here to find instructions for exporting a list of URLs from each service.

- Pocket
- Pinboard
- Instapaper
- Reddit Saved Posts
- Shaarli
- Unmark.it
- Wallabag
- Chrome Bookmarks
- Firefox Bookmarks
- Safari Bookmarks
- Opera Bookmarks
- Internet Explorer Bookmarks
- Chrome History: `./bin/archivebox-export-browser-history --chrome`
- Firefox History: `./bin/archivebox-export-browser-history --firefox`
- Other File or URL: (e.g. RSS feed url, text file path) pass as second argument in the next step

(If any of these links are broken, please submit an issue and I'll fix it)

3. Add your URLs to the archive

Pass in URLs to archive via stdin:

```
echo 'https://example.com' | ./archive
```

Or import a list of links from a file or feed URL:

```
./archive ~/Downloads/example_bookmarks_export.html
./archive https://getpocket.com/users/example/feed/all
```

Done!

Open `output/index.html` to view your archive. (favicons will appear next to each title once they have finished downloading)

Next Steps:

- Read [[Usage]] to learn about the various CLI and web UI functions
- Read [[Configuration]] to learn about the various archive method options
- Read [[Scheduled Archiving]] to learn how to set up automatic daily archiving
- Read [[Publishing Your Archive]] if you want to host your archive for others to access online
- Read [[Troubleshooting]] if you encounter any problems

1.2.2 Install

ArchiveBox only has a few main dependencies apart from `python3`, and they can all be installed using your normal package manager. It usually takes 1min to get up and running if you use the *helper script*, or about 5min if you install everything *manually*.

- *Supported Systems*
- *Dependencies*
- *Automatic Setup*
- *Manual Setup*
- *Docker Setup*

Supported Systems

ArchiveBox officially supports the following operating systems:

- ***macOS:*** >=10.12 (with homebrew)
- ***Linux:*** Ubuntu, Debian, etc (with apt)
- ***BSD:*** FreeBSD, OpenBSD, NetBSD etc (with pkg)

Other systems that are not officially supported but probably work to varying degrees:

- Windows: Via [[Docker]] or WSL
- Other Linux distros: Fedora, SUSE, Arch, CentOS, etc.

Platforms other than Linux, BSD, and macOS are untested, but you can probably get it working on them without too much effort.

It's recommended to use a filesystem with compression and/or deduplication abilities (e.g. ZFS or BTRFS) for maximum archive storage efficiency.

Dependencies

Not all the dependencies are required for all modes. If you disable some archive methods you can avoid those dependencies, for example, if you set `FETCH_MEDIA=False` you don't need to install `youtube-dl`, and if you set `FETCH_[PDF,SCREENSHOT,DOM]=False` you don't need `chromium`.

- `python3 >= 3.5`
- `wget >= 1.16`
- `chromium >= 59` (`google-chrome >= v59` works fine as well)
- `youtube-dl`
- `curl` (usually already on most systems)
- `git` (usually already on most systems)

More info:

- For help installing these, see the *Manual Setup*, [[Troubleshooting]] and [[Chromium Install]] pages.
- To use specific binaries for dependencies, see the Configuration: Dependencies page.
- To disable unwanted dependencies, see the Configuration: Archive Method Toggles page.

Automatic Setup

If you're on Linux with `apt`, or macOS with `brew` there is an automatic setup script provided to install all the dependencies. BSD, Windows, and other OS users should follow the *Manual Setup* or [[Docker]] instructions.

```
cd ArchiveBox/
./bin/archivebox-setup
```

The script explains what it installs beforehand, and will prompt for user confirmation before making any changes to your system.

After running the setup script, continue with the [[Quickstart]] guide...

Manual Setup

If you don't like running random setup scripts off the internet (:+1:), you can follow these manual setup instructions.

1. Install dependencies

macOS

```
brew install python3 git wget curl youtube-dl
brew cask install chromium  # Skip this if you already have Google Chrome/Chromium␣
↪installed in /Applications/
```

Ubuntu/Debian

```
apt install python3 python3-pip python3-distutils git wget curl youtube-dl
apt install chromium-browser  # Skip this if you already have Google Chrome/Chromium␣
↪installed
```

BSD

```
pkg install python3 git wget curl youtube-dl
pkg install chromium-browser  # Skip this if you already have Google Chrome/Chromium␣
↪installed
```

Check that everything worked and the versions are high enough.

```
python3 --version | head -n 1 &&
git --version | head -n 1 &&
wget --version | head -n 1 &&
curl --version | head -n 1 &&
youtube-dl --version | head -n 1 &&
echo "[] All dependencies installed."
```

If you have issues setting up Chromium / Google Chrome, see the [[Chromium Install]] page for more detailed setup instructions.

2. Get your bookmark export file

Follow the [[Quickstart]] guide to download your bookmarks export file containing a list of links to archive.

3. Run the archive script

1. Clone this repo `git clone https://github.com/pirate/ArchiveBox`
2. `cd ArchiveBox/`
3. `./archive ~/Downloads/links_list.html`

You may optionally specify a second argument to `archive.py export.html 153242424324` to resume the archive update at a specific timestamp.

Next Steps

- Read [[Usage]] to learn how to use the ArchiveBox CLI and HTML output
- Read [[Configuration]] to learn about the various archive method options
- Read [[Scheduled Archiving]] to learn how to set up automatic daily archiving
- Read [[Publishing Your Archive]] if you want to host your archive for others to access online
- Read [[Troubleshooting]] if you encounter any problems

Docker Setup

First, if you don't already have docker installed, follow the official install instructions for Linux, macOS, or Windows https://docs.docker.com/install/#supported-platforms.

Then see the [[Docker]] page for next steps.

1.2.3 Docker

Overview

Running ArchiveBox with Docker allows you to manage it in a container without exposing it to the rest of your system. Usage with Docker is similar to usage of ArchiveBox normally, with a few small differences.

Make sure you have Docker installed and set up on your machine before following these instructions. If you don't already have Docker installed, follow the official install instructions for Linux, macOS, or Windows here: https://docs.docker.com/install/#supported-platforms.

- *Setup*
- *Usage*
- *Accessing the data*
- *Configuration*

Official Docker Hub image:https://hub.docker.com/r/nikisweeting/archivebox

Usage:

```
echo 'https://example.com' | docker run -i -v ~/ArchiveBox:/data nikisweeting/
↪archivebox
```

Docker Compose

An example `docker-compose.yml` config with ArchiveBox and an Nginx server to serve the archive is included in the project root. You can edit it as you see fit, or just run it as it comes out-of-the-box.

Just make sure you have a Docker version that's new enough to support `version: 3` format:

```
docker --version
Docker version 18.09.1, build 4c52b90    # must be >= 17.04.0
```

Setup

```
git clone https://github.com/pirate/ArchiveBox && cd ArchiveBox
mkdir data && chmod 777 data
docker-compose up -d
```

Then open `http://127.0.0.1:8098` or `data/index.html` to view the archive (HTTP, not HTTPS).

Usage

First, make sure you're `cd`'ed into the same folder as your `docker-compose.yml` file (e.g. the project root) and that your containers have been started with `docker-compose up -d`.

To add new URLs, you can use docker-compose just like the normal `./archive` CLI.

To add an individual link or list of links, pass in URLs via stdin.

```
echo "https://example.com" | docker-compose exec -T archivebox /bin/archive
```

To import links from a file you can either `cat` the file and pass it via stdin like above, or move it into your data folder so that ArchiveBox can access it from within the container.

```
mv ~/Downloads/bookmarks.html data/sources/bookmarks.html
docker-compose exec archivebox /bin/archive /data/sources/bookmarks.html
```

To pull in links from a feed or remote file, pass the URL or path to the feed as an argument.

```
docker-compose exec archivebox /bin/archive https://example.com/some/feed.rss
```

Passing a URL as an argument here does not archive the specified URL, it downloads it and archives the links *inside* of it, so only use it for RSS feeds or other *lists of links* you want to add. To add an individual link you want to archive use the instruction above and pass via stdin instead of by argument.

Accessing the data

The outputted archive data is stored in `data/` (relative to the project root), or whatever folder path you specified in the `docker-compose.yml` `volumes:` section. Make sure the `data/` folder on the host has permissions initially set to `777` so that the ArchiveBox command is able to set it to the specified `OUTPUT_PERMISSIONS` config setting on the first run.

To access your archive, you can open `data/index.html` directly, or you can use the provided Nginx server running inside docker on `http://127.0.0.1:8098`.

Configuration

ArchiveBox running with docker-compose accepts all the same environment variables as normal, see the full list on the [[Configuration]] page.

The recommended way to pass in config variables is to edit the `environment:` section in `docker-compose.yml` directly or add an `env_file: ./path/to/ArchiveBox.conf` line before `environment:` to import variables from an env file.

Example of adding config options to `docker-compose.yml`:

```
...

services:
    archivebox:
        ...
        environment:
            - USE_COLOR=False
            - SHOW_PROGRESS=False
            - CHECK_SSL_VALIDITY=False
            - RESOLUTION=1900,1820
            - MEDIA_TIMEOUT=512000
        ...
```

You can also specify an env file via CLI when running compose using `docker-compose --env-file=/path/to/config.env ...` although you must specify the variables in the `environment:` section that you want to have passed down to the ArchiveBox container from the passed env file.

If you want to access your archive server with HTTPS, put a reverse proxy like Nginx or Caddy in front of `http://127.0.0.1:8098` to do SSL termination. You can find many instructions to do this online if you search "SSL reverse proxy".

Docker

Setup

Fetch and run the ArchiveBox Docker image to create your initial archive.

```
echo 'https://example.com' | docker run -i -v ~/ArchiveBox:/data nikisweeting/
↪archivebox
```

Replace `~/ArchiveBox` in the command above with the full path to a folder to use to store your archive on the host, or name of a Docker data volume.

Make sure the data folder you use host is either a new, uncreated path, or if it already exists make sure it has permissions initially set to `777` so that the ArchiveBox command is able to set it to the specified `OUTPUT_PERMISSIONS` config setting on the first run.

Usage

To add a single URL to the archive or a list of links from a file, pipe them in via stdin. This will archive each link passed in.

```
echo 'https://example.com' | docker run -i -v ~/ArchiveBox:/data nikisweeting/
↪archivebox
# or
cat bookmarks.html | docker run -i -v ~/ArchiveBox:/data nikisweeting/archivebox
```

To add a list of pages via feed URL or remote file, pass the URL of the feed as an argument.

```
docker run -v -v ~/ArchiveBox:/data nikisweeting/archivebox /bin/archive 'https://
↪example.com/some/rss/feed.xml'
```

Passing a URL as an argument here does not archive the specified URL, it downloads it and archives the links *inside* of it, so only use it for RSS feeds or other *lists of links* you want to add. To add an individual link use the instruction above and pass via stdin instead of by argument.

Accessing the data

Using a bind folder

Use the flag:

```
-v /full/path/to/folder/on/host:/data
```

This will use the folder `/full/path/to/folder/on/host` on your host to store the ArchiveBox output.

Using a named Docker data volume

```
docker volume create archivebox-data
```

Then use the flag:

```
-v archivebox-data:/data
```

You can mount your data volume using standard docker tools, or access the contents directly here:`/var/lib/docker/volumes/archivebox-data/_data` (on most Linux systems)

On a Mac you'll have to enter the base Docker Linux VM first to access the volume data:

```
screen ~/Library/Containers/com.docker.docker/Data/vms/0/tty
cd /var/lib/docker/volumes/archivebox-data/_data
```

Configuration

ArchiveBox in Docker accepts all the same environment variables as normal, see the list on the [[Configuration]] page.

To pass environment variables when running, you can use the env command.

```
echo 'https://example.com' | docker run -i -v ~/ArchiveBox:/data nikisweeting/
↪archivebox env FETCH_SCREENSHOT=False /bin/archive
```

Or you can create an `ArchiveBox.env` file (copy from the default `etc/ArchiveBox.conf.default`) and pass it in like so:

```
docker run -i -v --env-file=ArchiveBox.env nikisweeting/archivebox
```

1.3 General

1.3.1 Usage

Make sure the dependencies are fully installed before running any ArchiveBox commands.

ArchiveBox API Reference:

- *Overview*: Program structure and outline of basic archiving process.
- *CLI Usage*: Docs and examples for the ArchiveBox command line interface.
- *UI Usage*: Docs and screenshots for the outputted HTML archive interface.
- *Disk Layout*: Description of the archive folder structure and contents.

Related:

- [[Docker]]: Learn about ArchiveBox usage with Docker and Docker Compose
- [[Configuration]]: Learn about the various archive method options
- [[Scheduled Archiving]]: Learn how to set up automatic daily archiving
- [[Publishing Your Archive]]: Learn how to host your archive for others to access
- [[Troubleshooting]]: Resources if you encounter any problems
- Screenshots: See what the CLI and outputted HTML look like

Overview

The `./archive` binary is a shortcut to `bin/archivebox`. Piping RSS, JSON, Netscape, or TXT lists of links into the `./archive` command will add them to your archive folder, and create a locally-stored browsable archive for each new URL.

The archiver produces an *output folder* `output/` containing `index.html`, `index.json`, and archived copies of all the sites organized by timestamp bookmarked. It's powered by Chrome headless, good 'ol `wget`, and a few other common Unix tools.

CLI Usage

`./archive` refers to the executable shortcut in the root of the project, but you can also call ArchiveBox via `./bin/archivebox`. If you add `/path/to/ArchiveBox/bin` to your shell `$PATH` then you can call `archivebox` from anywhere on your system.

If you're using Docker, the CLI interface is similar but needs to be prefixed by `docker-compose exec ...` or `docker run ...`, for examples see the [[Docker]] page.

- *Run ArchiveBox with configuration options*
- *Import a single URL or list of URLs via stdin*
- *Import list of links exported from browser or another service*
- *Import list of URLs from a remote RSS feed or file*
- *Import list of links from browser history*

Run ArchiveBox with configuration options

You can set environment variables in your shell profile, a config file, or by using the `env` command.

```
env FETCH_MEDIA=True MEDIA_TIMEOUT=500 ./archive ...
```

See [[Configuration]] page for more details about the available options and ways to pass config.If you're using Docker, also make sure to read the Configuration section on the [[Docker]] page.

Import a single URL or list of URLs via stdin

```
echo 'https://example.com' | ./archive
# or
cat urls_to_archive.txt | ./archive
```

You can also pipe in RSS, XML, Netscape, or any of the other supported import formats via stdin.

Import list of links exported from browser or another service

```
./archive ~/Downloads/browser_bookmarks_export.html
# or
./archive ~/Downloads/pinboard_bookmarks.json
# or
./archive ~/Downloads/other_links.txt
```

Passing a file as an argument here does not archive the file, it parses it as a list of URLs and archives the links *inside of it*, so only use it for *lists of links* to archive, not HTML files or other content you want added directy to the archive.

Import list of URLs from a remote RSS feed or file

ArchiveBox will download the URL to a local file in `output/sources/` and attempt to autodetect the format and import any URLs found. Currently, Netscape HTML, JSON, RSS, and plain text links lists are supported.

```
./archive https://example.com/feed.rss
# or
./archive https://example.com/links.txt
```

Passing a URL as an argument here does not archive the specified URL, it downloads it and archives the links *inside* of it, so only use it for RSS feeds or other *lists of links* you want to add. To add an individual link use the instruction above and pass the URL via stdin instead of as an argument.

Import list of links from browser history

```
./bin/archivebox-export-browser-history --chrome
./archive output/sources/chrome_history.json
# or
./bin/archivebox-export-browser-history --firefox
./archive output/sources/firefox_history.json
```

UI Usage

To access your archive, open `output/index.html` in a browser. You should see something like this.

You can sort by column, search using the box in the upper right, and see the total number of links at the bottom.

Click the Favicon under the "Files" column to go to the details page for each link.

Disk Layout

The `output/` folder containing the UI HTML and archived data has the structure outlined here.

```
- output/
  - index.json           # Main index of all archived URLs
  - index.html

  - archive/
     - 155243135/        # Archived links are stored in folders by timestamp
        - index.json     # Index/details page for individual archived link
        - index.html

        # Archive method outputs:
        - warc/
        - media/
        - git/
        ...

  - sources/             # Each imported URL list is saved as a copy here
     - getpocket.com-1552432264.txt
```

(continues on next page)

(continued from previous page)

```
    - stdin-1552291774.txt
    ...

- static/               # Staticfiles for the archive UI
- robots.txt
```

Large Archives

I've found it takes about an hour to download 1000 articles, and they'll take up roughly 1GB.Those numbers are from running it single-threaded on my i5 machine with 50mbps down. YMMV.

Storage requirements go up immensely if you're using `FETCH_MEDIA=True` and are archiving many pages with audio & video.

You can run it in parallel by using the `resume` feature, or by manually splitting export.html into multiple files:

```
./archive export.html 1498800000 &  # second argument is timestamp to resume␣
↪downloading from
./archive export.html 1498810000 &
./archive export.html 1498820000 &
./archive export.html 1498830000 &
```

Users have reported running it with 50k+ bookmarks with success (though it will take more RAM while running).

If you already imported a huge list of bookmarks and want to import only new bookmarks, you can use the `ONLY_NEW` environment variable. This is useful if you want to import a bookmark dump periodically and want to skip broken links which are already in the index.

Python API Usage

```
from archivebox.main import add, info, remove, check_data_folder

out_dir = '~/path/to/my/data/folder'
check_data_folder(out_dir=out_dir)
add('https://example.com', index_only=True, out_dir=out_dir)
info(out_dir=out_dir)
remove('https://example.com', delete=True, yes=True, out_dir=out_dir)
```

For more information see the Python API Reference.

1.3.2 Configuration

The default ArchiveBox config file can be found here: `etc/ArchiveBox.conf.default`.

Configuration is done through environment variables. You can pass in settings using all the usual environment variable methods: e.g. by using the `env` command, exporting variables in your shell profile, or sourcing a `.env` file before running the command.

Example of passing configuration using `env` command:

```
env CHROME_BINARY=google-chrome-stable RESOLUTION=1440,900 FETCH_PDF=False ./archive ~
↪/Downloads/bookmarks_export.html
```

Available Configuration Options:

- *General Settings:* Archiving process, output format, and timing.
- *Archive Method Toggles:* On/off switches for methods.
- *Archive Method Options:* Method tunables and parameters.
- *Shell Options:* Format & behavior of CLI output.
- *Dependency Options:* Specify exact paths to dependencies.

All the available config options are described in this document below, but can also be found along with examples in `etc/ArchiveBox.conf.default`. The code that loads the config is in `archivebox/config.py`, but don't modify the defaults in `config.py` directly, as your changes there will be erased whenever you update ArchiveBox.

To create a persistent config file, see the *Creating a Config File* section.To see details on how to do configuration when using Docker, see the [[Docker]] page.

General Settings

General options around the archiving process, output format, and timing.

OUTPUT_DIR

Possible Values: [`$REPO_DIR/output`]/`/srv/www/bookmarks/...` Path to an output folder to store the archive in.

Defaults to `output/` in the root directory of the repository folder.

Note: ArchiveBox will create this folder if missing. If it already exists, make sure ArchiveBox has permission to write to it.

OUTPUT_PERMISSIONS

Possible Values: [`755`]/`644/...` Permissions to set the output directory and file contents to.

This is useful when running ArchiveBox inside Docker as root and you need to explicitly set the permissions to something that the users on the host can access.

ONLY_NEW

Possible Values: [`False`]/`True`Download files for only newly added links when running the `./archive` command.

By default, ArchiveBox will go through all links in the index and download any missing files on every run, set this to `True` to only archive the most recently added batch of links without attempting to also update older archived links.

Note: Regardless of how this is set, ArchiveBox will never re-download sites that have already succeeded previously. When this is `False` it only attempts to fix previous pages have missing archives, it does not re-archive pages that have already been archived. Set it to `True` only if you wish to skip repairing missing older archives on every run.

TIMEOUT

Possible Values: [60]/120/...Maximum allowed download time per archive method for each link in seconds. If you have a slow network connection or are seeing frequent timeout errors, you can raise this value.

Note: Do not set this to anything less than 15 seconds as it will cause Chrome to hang indefinitely and many sites to fail completely.

MEDIA_TIMEOUT

Possible Values: [3600]/120/...Maximum allowed download time for fetching media when FETCH_MEDIA=True in seconds. This timeout is separate and usually much longer than TIMEOUT because media downloaded with youtube-dl can often be quite large and take many minutes/hours to download. Tweak this setting based on your network speed and maximum media file size you plan on downloading.

Note: Do not set this to anything less than 10 seconds as it can often take 5-10 seconds for youtube-dl just to parse the page before it starts downloading media files.

Related options:FETCH_MEDIA

TEMPLATES_DIR

Possible Values: [$REPO_DIR/archivebox/templates]//path/to/custom/templates/...Path to a directory containing custom index html templates for themeing your archive output. Folder at specified path must contain the following files:

- static/
- index.html
- link_index.html
- index_row.html

You can copy the files in archivebox/templates into your own directory to start developing a custom theme, then edit TEMPLATES_DIR to point to your new custom templates directory.

Related options:FOOTER_INFO

FOOTER_INFO

Possible Values: [Content is hosted for personal archiving purposes only. Contact server owner for any takedown requests.]/Operated by ACME Co./...Some text to display in the footer of the archive index. Useful for providing server admin contact info to respond to takedown requests.

Related options:TEMPLATES_DIR

URL_BLACKLIST

Possible Values: [None]/.+\.exe$/http(s)?:\/\/(.+)?example.com\/.*'/...

A regex expression used to exclude certain URLs from the archive. You can use if there are certain domains, extensions, or other URL patterns that you want to ignore whenever they get imported. Blacklisted URLs wont be included in the index, and their page content wont be archived.

When building your blacklist, you can check whether a given URL matches your regex expression like so:

```
>>>import re
>>>URL_BLACKLIST = r'http(s)?:\/\/(.+)?(youtube\.com)|(amazon\.com)\/.*'  # replace␣
↪this with your regex to test
>>>test_url = 'https://test.youtube.com/example.php?abc=123'
>>>bool(re.compile(URL_BLACKLIST, re.IGNORECASE).match(test_url))
True
```

Related options:FETCH_MEDIA, *FETCH_GIT*, *GIT_DOMAINS*

Archive Method Toggles

High-level on/off switches for all the various methods used to archive URLs.

FETCH_TITLE

Possible Values: [True]/FalseBy default ArchiveBox uses the title provided by the import file, but not all types of imports provide titles (e.g. Plain texts lists of URLs). When this is True, ArchiveBox downloads the page (and follows all redirects), then it attempts to parse the link's title from the first <title></title> tag found in the response. It may be buggy or not work for certain sites that use JS to set the title, disabling it will lead to links imported without a title showing up with their URL as the title in the UI.

Related options:ONLY_NEW, *CHECK_SSL_VALIDITY*

FETCH_FAVICON

Possible Values: [True]/FalseFetch and save favicon for the URL from Google's public favicon service: https://www.google.com/s2/favicons?domain={domain}. Set this to FALSE if you don't need favicons.

Related options:TEMPLATES_DIR, *CHECK_SSL_VALIDITY*, *CURL_BINARY*

FETCH_WGET

Possible Values: [True]/FalseFetch page with wget, and save responses into folders for each domain, e.g. example.com/index.html, with .html appended if not present. For a full list of options used during the wget download process, see the archivebox/archive_methods.py:fetch_wget(...) function.

Related options:TIMEOUT, *FETCH_WGET_REQUISITES*, *CHECK_SSL_VALIDITY*, *COOKIES_FILE*, *WGET_USER_AGENT*, *FETCH_WARC*, *WGET_BINARY*

FETCH_WARC

Possible Values: [True]/FalseSave a timestamped WARC archive of all the page requests and responses during the wget archive process.

Related options:TIMEOUT, *FETCH_WGET_REQUISITES*, *CHECK_SSL_VALIDITY*, *COOKIES_FILE*, *WGET_USER_AGENT*, *FETCH_WGET*, *WGET_BINARY*

FETCH_PDF

Possible Values: [True]/FalsePrint page as PDF.

Related options:TIMEOUT, *CHECK_SSL_VALIDITY*, *CHROME_USER_DATA_DIR*, *CHROME_BINARY*

FETCH_SCREENSHOT

Possible Values: [True]/FalseFetch a screenshot of the page.

Related options:RESOLUTION, *TIMEOUT*, *CHECK_SSL_VALIDITY*, *CHROME_USER_DATA_DIR*, *CHROME_BINARY*

FETCH_DOM

Possible Values: [True]/FalseFetch a DOM dump of the page.

Related options:TIMEOUT, *CHECK_SSL_VALIDITY*, *CHROME_USER_DATA_DIR*, *CHROME_BINARY*

FETCH_GIT

Possible Values: [True]/FalseFetch any git repositories on the page.

Related options:TIMEOUT, *GIT_DOMAINS*, *CHECK_SSL_VALIDITY*, *GIT_BINARY*

FETCH_MEDIA

Possible Values: [True]/FalseFetch all audio, video, annotations, and media metadata on the page using youtube-dl. Warning, this can use up *a lot* of storage very quickly.

Related options:MEDIA_TIMEOUT, *CHECK_SSL_VALIDITY*, *YOUTUBEDL_BINARY*

SUBMIT_ARCHIVE_DOT_ORG

Possible Values: [True]/FalseSubmit the page's URL to be archived on Archive.org. (The Internet Archive)

Related options:TIMEOUT, *CHECK_SSL_VALIDITY*, *CURL_BINARY*

Archive Method Options

Specific options for individual archive methods above. Some of these are shared between multiple archive methods, others are specific to a single method.

CHECK_SSL_VALIDITY

Possible Values: [True]/FalseWhether to enforce HTTPS certificate and HSTS chain of trust when archiving sites. Set this to False if you want to archive pages even if they have expired or invalid certificates. Be aware that when False you cannot guarantee that you have not been man-in-the-middle'd while archiving content, so the content cannot be verified to be what's on the original site.

FETCH_WGET_REQUISITES

Possible Values: [True]/FalseFetch images/css/js with wget. (True is highly recommended, otherwise your wont download many critical assets to render the page, like images, js, css, etc.)

Related options:TIMEOUT, *FETCH_WGET*, *FETCH_WARC*, *WGET_BINARY*

RESOLUTION

Possible Values: [1440,900]/1024,768/...Screenshot resolution in pixels width,height.

Related options:FETCH_SCREENSHOT

WGET_USER_AGENT

Possible Values: [`Wget/1.19.1`]/`"Mozilla/5.0 ..."`/...This is the user agent to use during wget archiving. You can set this to impersonate a more common browser like Chrome or Firefox if you're getting blocked by servers for having an unknown/blacklisted user agent.

Related options:`FETCH_WGET`, `FETCH_WARC`, `CHECK_SSL_VALIDITY`, `WGET_BINARY`, `CHROME_USER_AGENT`

CHROME_USER_AGENT

Possible Values: [`"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) HeadlessChrome/73.0.3683.75 Safari/537.36"`]/`"Mozilla/5.0 ..."`/...

This is the user agent to use during Chrome headless archiving. If you're experiencing being blocked by many sites, you can set this to hide the `Headless` string that reveals to servers that you're using a headless browser.

Related options:`FETCH_PDF`, `FETCH_SCREENSHOT`, `FETCH_DOM`, `CHECK_SSL_VALIDITY`, `CHROME_USER_DATA_DIR`, `CHROME_HEADLESS`, `CHROME_BINARY`, `WGET_USER_AGENT`

GIT_DOMAINS

Possible Values: [`github.com,bitbucket.org,gitlab.com`]/`git.example.com`/...Domains to attempt download of git repositories on using `git clone`.

Related options:`FETCH_GIT`, `CHECK_SSL_VALIDITY`

COOKIES_FILE

Possible Values: [`None`]/`/path/to/cookies.txt`/...Cookies file to pass to wget. To capture sites that require a user to be logged in, you can specify a path to a netscape-format `cookies.txt` file for wget to use. You can generate this file by using a browser extension to export your cookies in this format, or by using wget with `--save-cookies`.

Related options:`FETCH_WGET`, `FETCH_WARC`, `CHECK_SSL_VALIDITY`, `WGET_BINARY`

CHROME_USER_DATA_DIR

Possible Values: [`~/.config/google-chrome`]/`/tmp/chrome-profile`/...Path to a Chrome user profile directory. To capture sites that require a user to be logged in, you can specify a path to a chrome user profile (which loads the cookies needed for the user to be logged in). If you don't have an existing Chrome profile, create one with `chromium-browser --user-data-dir=/tmp/chrome-profile`, and log into the sites you need. Then set `CHROME_USER_DATA_DIR=/tmp/chrome-profile` to make ArchiveBox use that profile.

Note: Make sure the path does not have `Default` *at the end (it should the the parent folder of* `Default`*), e.g. set it to* `CHROME_USER_DATA_DIR=~/.config/chromium` *and not* `CHROME_USER_DATA_DIR=~/.config/chromium/Default`*.*

By default when set to `None`, ArchiveBox tries all the following User Data Dir paths in order:https://chromium.googlesource.com/chromium/src/+/HEAD/docs/user_data_dir.md

Related options:`FETCH_PDF`, `FETCH_SCREENSHOT`, `FETCH_DOM`, `CHECK_SSL_VALIDITY`, `CHROME_HEADLESS`, `CHROME_BINARY`

CHROME_HEADLESS

Possible Values: [`True`]/`False`Whether or not to use Chrome/Chromium in `--headless` mode (no browser UI displayed). When set to `False`, the full Chrome UI will be launched each time it's used to archive a page, which greatly slows down the process but allows you to watch in real-time as it saves each page.

Related options:`FETCH_PDF`, `FETCH_SCREENSHOT`, `FETCH_DOM`, `CHROME_USER_DATA_DIR`, `CHROME_BINARY`

CHROME_SANDBOX

Possible Values: [`True`]/`False`Whether or not to use the Chrome sandbox when archiving.

If you see an error message like this, it means you are trying to run ArchiveBox as root:

```
:ERROR:zygote_host_impl_linux.cc(89)] Running as root without --no-sandbox is not␣
↪supported. See https://crbug.com/638180
```

*Note: **Do not run ArchiveBox as root!** The solution to this error is not to override it by setting `CHROME_SANDBOX=False`, it's to use create another user (e.g. `www-data`) and run ArchiveBox under that new, less privileged user. This is a security-critical setting, only set this to `False` if you're running ArchiveBox inside a container or VM where it doesn't have access to the rest of your system!

Related options:`FETCH_PDF`, `FETCH_SCREENSHOT`, `FETCH_DOM`, `CHECK_SSL_VALIDITY`, `CHROME_USER_DATA_DIR`, `CHROME_HEADLESS`, `CHROME_BINARY`

Shell Options

Options around the format of the CLI output.

USE_COLOR

Possible Values: [`True`]/`False`Colorize console output. Defaults to `True` if stdin is a TTY (interactive session), otherwise `False` (e.g. if run in a script or piped into a file).

SHOW_PROGRESS

Possible Values: [`True`]/`False`Show real-time progress bar in console output. Defaults to `True` if stdin is a TTY (interactive session), otherwise `False` (e.g. if run in a script or piped into a file).

Dependency Options

Options for defining which binaries to use for the various archive method dependencies.

CHROME_BINARY

Possible Values: [`chromium-browser`]/`/usr/local/bin/google-chrome`/...Path or name of the Google Chrome / Chromium binary to use for all the headless browser archive methods.

Without setting this environment variable, ArchiveBox by default look for the following binaries in `$PATH` in this order:

- `chromium-browser`
- `chromium`
- `google-chrome`
- `google-chrome-stable`
- `google-chrome-unstable`
- `google-chrome-beta`
- `google-chrome-canary`
- `google-chrome-dev`

You can override the default behavior to search for any available bin by setting the environment variable to your preferred Chrome binary name or path.

The chrome/chromium dependency is *optional* and only required for screenshots, PDF, and DOM dump output, it can be safely ignored if those three methods are disabled.

Related options:`FETCH_PDF`, `FETCH_SCREENSHOT`, `FETCH_DOM`, `CHROME_USER_DATA_DIR`, `CHROME_HEADLESS`, `CHROME_SANDBOX`

WGET_BINARY

Possible Values: [`wget`]/`/usr/local/bin/wget`/...Path or name of the wget binary to use.

Related options:`FETCH_WGET`, `FETCH_WARC`

YOUTUBEDL_BINARY

Possible Values: [youtube-dl]/usr/local/bin/youtube-dl/...Path or name of the youtube-dl binary to use.

Related options:FETCH_MEDIA

GIT_BINARY

Possible Values: [git]/usr/local/bin/git/...Path or name of the git binary to use.

Related options:FETCH_GIT

CURL_BINARY

Possible Values: [curl]/usr/local/bin/curl/...Path or name of the curl binary to use.

Related options:FETCH_FAVICON, SUBMIT_ARCHIVE_DOT_ORG

1.3.3 Troubleshooting

If you need help or have a question, you can open an issue or reach out on Twitter.

What are you having an issue with?:

- *Installing*
- Configuration
- *Archiving Process*
- *Hosting the Archive*

Installing

Make sure you've followed the Manual Setup guide in the [[Install]] instructions first. Then check here for help depending on what component you need help with:

Python

On some Linux distributions the python3 package might not be recent enough. If this is the case for you, resort to installing a recent enough version manually.

```
add-apt-repository ppa:fkrull/deadsnakes && apt update && apt install python3.6
```

If you still need help, the official Python docs are a good place to start.

Chromium/Google Chrome

For more info, see the [[Chromium Install]] page.

`archive.py` depends on being able to access a `chromium-browser`/`google-chrome` executable. The executable used defaults to `chromium-browser` but can be manually specified with the environment variable `CHROME_BINARY`:

```
env CHROME_BINARY=/usr/local/bin/chromium-browser ./archive ~/Downloads/bookmarks_
↪export.html
```

1. Test to make sure you have Chrome on your `$PATH` with:

```
which chromium-browser || which google-chrome
```

If no executable is displayed, follow the setup instructions to install and link one of them.

1. If a path is displayed, the next step is to check that it's runnable:

```
chromium-browser --version || google-chrome --version
```

If no version is displayed, try the setup instructions again, or confirm that you have permission to access chrome.

1. If a version is displayed and it's `<59`, upgrade it:

```
apt upgrade chromium-browser -y
# OR
brew cask upgrade chromium-browser
```

1. If a version is displayed and it's `>=59`, make sure `archive.py` is running the right one:

```
env CHROME_BINARY=/path/from/step/1/chromium-browser ./archive bookmarks_export.html ␣
↪ # replace the path with the one you got from step 1
```

Wget & Curl

If you're missing `wget` or `curl`, simply install them using `apt` or your package manager of choice. See the "Manual Setup" instructions for more details.

If wget times out or randomly fails to download some sites that you have confirmed are online, upgrade wget to the most recent version with `brew upgrade wget` or `apt upgrade wget`. There is a bug in versions `<=1.19.1_1` that caused wget to fail for perfectly valid sites.

Archiving

No links parsed from export file

Please open an issue with a description of where you got the export, and preferrably your export file attached (you can redact the links). We'll fix the parser to support your format.

Lots of skipped sites

If you ran the archiver once, it wont re-download sites subsequent times, it will only download new links. If you haven't already run it, make sure you have a working internet connection and that the parsed URLs look correct. You

can check the `archive.py` output or `index.html` to see what links it's downloading.

If you're still having issues, try deleting or moving the `output/archive` folder (back it up first!) and running `./archive` again.

Lots of errors

Make sure you have all the dependencies installed and that you're able to visit the links from your browser normally. Open an issue with a description of the errors if you're still having problems.

Lots of broken links from the index

Not all sites can be effectively archived with each method, that's why it's best to use a combination of `wget`, PDFs, and screenshots. If it seems like more than 10-20% of sites in the archive are broken, open an issue with some of the URLs that failed to be archived and I'll investigate.

Removing unwanted links from the index

If you accidentally added lots of unwanted links into index and they slow down your archiving, you can use the `bin/purge` script to remove them from your index, which removes everything matching python regexes you pass into it. E.g: `bin/purge -r 'amazon\.com' -r 'google\.com'`. It would prompt before removing links from index, but for extra safety you might want to back up `index.json` first (or put in undex version control).

Hosting the Archive

If you're having issues trying to host the archive via nginx, make sure you already have nginx running with SSL. If you don't, google around, there are plenty of tutorials to help get that set up. Open an issue if you have problem with a particular nginx config.

1.3.4 Security Overview

Usage Modes

ArchiveBox has three common usage modes outlined below.

Public Mode [Default]

This is the default (lax) mode, intended for archiving public (non-secret) URLs without authenticating the headless browser. This is the mode used if you're archiving news articles, audio, video, etc. browser bookmarks to a folder published on your webserver. This allows you to access and link to content on `http://your.archive.com/archive...` after the originals go down.

This mode should not be used for archiving entire browser history or authenticated private content like Google Docs, paywalled content, invite-only subreddits, etc.

Private Mode

ArchiveBox is designed to be able to archive content that requires authentication or cookies. This includes paywalled content, private forums, LAN-only content, etc.

To get started, set `CHROME_USER_DATA_DIR` and `COOKIES_FILE` to point to a Chrome user folder that has your sessions and a wget `cookies.txt` file respectively.

If you're importing private links or authenticated content, you probably don't want to share your archive folder publicly on a webserver, so don't follow the [[Publishing Your Archive]] instructions unless you are only serving it on a trusted LAN or have some sort of authentication in front of it. Make sure to point ArchiveBox to an output folder with conservative permissions, as it may contain archived content with secret session tokens or pieces of your user data. You may also wish to encrypt the archive using an encrypted disk image or filesystem like ZFS as it will contain all requests and response data, including session keys, user data, usernames, etc.

Stealth Mode

If you want ArchiveBox to be less noisy and avoid leaking any URLs to 3rd-party APIs during archiving, you can disable the options below. Disabling these are recommended if you plan on archiving any sites that use secret tokens in the URL to grant access to private content without authentication, e.g. Google Docs, CodiDM notepads, etc.

- `https://web.archive.org/save/{url}` when `SUBMIT_ARCHIVE_DOT_ORG` is `True`, full URLs are submitted to the Wayback Machine for archiving, but no cookies or content from the local authenticated archive are shared
- `https://www.google.com/s2/favicons?domain={domain}` when `FETCH_FAVICON` is `True`, the domains for each link are shared in order to get the favicon, but not the full URL

Do not run as root

Do not run ArchiveBox as root for a number of reasons:

- Chrome will execute as root and fail immediately because Chrome sandboxing is pointless when the data directory is opened as root (do not set `CHROME_SANDBOX=False` just to bypass that error!)
- All dependencies will be run as root, if any of them have a vulnerability that's exploited by sites you're archiving you're opening yourself up to full system compromise
- ArchiveBox does lots of HTML parsing, filesystem access, and shell command execution. A bug in any one of those subsystems could potentially lead to deleted/damaged data on your hard drive, or full system compromise unless restricted to a user that only has permissions to access the directories needed
- Do you really trust a project created by a Github user called `@pirate` ? Why give a random program off the internet root access to your entire system? (I don't have malicious intent, I'm just saying in principle you should not be running random Github projects as root)

Instead, you should run ArchiveBox as your normal user, or create a user with less privileged access:

```
useradd -r -g archivebox -G audio,video archivebox
mkdir -p /home/archivebox/data
chown -R archivebox:archivebox /home/archivebox
...
sudo -u archivebox ./archive ...
```

Output Folder

Permissions

What are the permissions on the archive folder? Limit access to the fewest possible users by checking folder ownership and setting `OUTPUT_PERMISSIONS` accordingly.

Filesystem

How much are you planning to archive? Only a few bookmarked articles, or thousands of pages of browsing history a day? If it's only 1-50 pages a day, you can probably just stick it in a normal folder on your hard drive, but if you want to go over 100 pages a day, you will likely want to put your archive on a compressed/deduplicated/encrypted disk image or filesystem like ZFS.

Publishing

Are you publishing your archive? If so, make sure you're only serving it as HTML and not accidentally running it as php or cgi, and put it on its own domain not shared with other services. This is done in order to avoid cookies leaking between your main domain and domains hosting content you don't control. Many companies put user provided files on separate domains like googleusercontent.com and github.io to avoid this problem.

Published archives automatically include a `robots.txt` `Dissallow: /` to block search engines from indexing them. You may still wish to publish your contact info in the index footer though using `FOOTER_INFO` so that you can respond to any DMCA and copyright takedown notices if you accidentally rehost copyrighted content.

1.3.5 Publishing Your Archive

The archive produced by `./archive` is suitable for serving on any provider that can host static html (e.g. github pages!).

You can also serve it from a home server or VPS by uploading the outputted `output` folder to your web directory, e.g. `/var/www/ArchiveBox` and configuring your webserver. If you're using docker-compose, an Nginx server serving the archive via HTTP is provided right out of the box! See the [[Docker]] page for details.

Here's a sample nginx configuration that works to serve archive folders:

```
location / {
    alias       /path/to/ArchiveBox/output/;
    index       index.html;
    autoindex   on;               # see directory listing upon clicking "The Files"␣
↪links
    try_files   $uri $uri/ =404;
}
```

Make sure you're not running any content as CGI or PHP, you only want to serve static files!

Urls look like: `https://archive.example.com/archive/1493350273/en.wikipedia.org/wiki/Dining_philosophers_problem.html`

Security Concerns

Re-hosting other people's content has security implications for any other sites sharing your hosting domain. Make sure you understand the dangers of hosting unknown archived CSS & JS files on your shared domain. Due to the security

risk of serving some malicious JS you archived by accident, it's best to put this on a domain or subdomain of its own to keep cookies separate and slightly mitigate CSRF attacks and other nastiness.

Copyright Concerns

Be aware that some sites you archive may not allow you to rehost their content publicly for copyright reasons, it's up to you to host responsibly and respond to takedown requests appropriately.

You may also want to blacklist your archive in `/robots.txt` if you don't want to be publicly assosciated with all the links you archive via search engine results.

Please modify the `FOOTER_INFO` config variable to add your contact info to the footer of your index.

1.3.6 Scheduled Archiving

Using Cron

To schedule regular archiving you can use any task scheduler like `cron`, `at`, `sytsemd`, etc.

ArchiveBox ignores links that are imported multiple times (keeping the earliest version that it's seen). This means you can add cron jobs that regularly poll the same file or URL for new links, adding only new ones as necessary.

For some example configs, see the `etc/cron.d` and `etc/supervisord` folders.

Examples

Example: Import Firefox browser history every 24 hours

This example exports your browser history and archives it once a day:

Create `/opt/ArchiveBox/bin/firefox_custom.sh`:

```
#!/bin/bash

cd /opt/ArchiveBox
./bin/archivebox-export-browser-history --firefox ./output/sources/firefox_history.
↪json
./bin/archivebox ./output/sources/firefox_history.json  >> /var/log/ArchiveBox.log
```

Then create a new file `/etc/cron.d/ArchiveBox-Firefox` to tell cron to run your script every 24 hours:

```
0 24 * * * www-data /opt/ArchiveBox/bin/firefox_custom.sh
```

Example: Import an RSS feed from Pocket every 12 hours

This example imports your Pocket bookmark feed and archives any new links once a day:

First, set your Pocket RSS feed to "public" under https://getpocket.com/privacy_controls.

Create `/opt/ArchiveBox/bin/pocket_custom.sh`:

```
#!/bin/bash

cd /opt/ArchiveBox
./bin/archivebox https://getpocket.com/users/yourusernamegoeshere/feed/all  >> /var/
↪log/ArchiveBox.log
```

Then create a new file `/etc/cron.d/ArchiveBox-Pocket` to tell cron to run your script every 12 hours:

```
0 12 * * * www-data /opt/ArchiveBox/bin/pocket_custom.sh
```

1.3.7 Chromium Install

By default, ArchiveBox looks for any existing installed version of Chrome/Chromium and uses it if found. You can optionally install a specific version and set the environment variable `CHROME_BINARY` to force ArchiveBox to use that one, e.g.:

- `CHROME_BINARY=google-chrome-beta`
- `CHROME_BINARY=/usr/bin/chromium-browser`
- `CHROME_BINARY='/Applications/Chromium.app/Contents/MacOS/Chromium'`

If you don't already have Chrome installed, I recommend installing Chromium instead of Google Chrome, as it's the open-source fork of Chrome that doesn't send as much tracking data to Google.

Check for existing Chrome/Chromium install:

```
google-chrome --version | chromium-browser --version
Google Chrome 73.0.3683.75 beta     # should be >v59
```

Installing Chromium

macOS

If you already have `/Applications/Chromium.app`, you don't need to run this.

```
brew cask install chromium-browser
```

Ubuntu/Debian

If you already have `chromium-browser` >= v59 installed (run `chromium-browser --version`, you don't need to run this.

```
apt update
apt install chromium-browser
```

Installing Google Chrome

macOS

If you already have `/Applications/Google Chrome.app`, you don't need to run this.

```
brew cask install google-chrome
```

Ubuntu/Debian

If you already have `google-chrome` >= v59 installed (run `google-chrome --version`, you don't need to run this.

```
wget -q -O - https://dl-ssl.google.com/linux/linux_signing_key.pub | sudo apt-key add␣
↪-
sudo sh -c 'echo "deb [arch=amd64] http://dl.google.com/linux/chrome/deb/ stable main
↪" >> /etc/apt/sources.list.d/google-chrome.list'
apt update
apt install google-chrome-beta
```

Troubleshooting

If you encounter problems setting up Google Chrome or Chromium, see the Troubleshooting page.

1.4 API Reference

1.4.1 archivebox

archivebox package

Subpackages

archivebox.cli package

Submodules

archivebox.cli.archivebox module

archivebox.cli.archivebox.**main**(*args: Optional[List[str]] = None*, *stdin: Optional[IO] = None*, *pwd: Optional[str] = None*) → None

archivebox.cli.archivebox_add module

archivebox.cli.archivebox_add.**main**(*args: Optional[List[str]] = None*, *stdin: Optional[IO] = None*, *pwd: Optional[str] = None*) → None

Add a new URL or list of URLs to your archive

archivebox.cli.archivebox_config module

archivebox.cli.archivebox_config.**main**(*args: Optional[List[str]] = None*, *stdin: Optional[IO] = None*, *pwd: Optional[str] = None*) → None

Get and set your ArchiveBox project configuration values

archivebox.cli.archivebox_help module

archivebox.cli.archivebox_help.**main**(*args: Optional[List[str]] = None*, *stdin: Optional[IO] = None*, *pwd: Optional[str] = None*) → None

Print the ArchiveBox help message and usage

archivebox.cli.archivebox_info module

archivebox.cli.archivebox_info.**main**(*args: Optional[List[str]] = None*, *stdin: Optional[IO] = None*, *pwd: Optional[str] = None*) → None

Print out some info and statistics about the archive collection

archivebox.cli.archivebox_init module

archivebox.cli.archivebox_init.**main**(*args: Optional[List[str]] = None*, *stdin: Optional[IO] = None*, *pwd: Optional[str] = None*) → None

Initialize a new ArchiveBox collection in the current directory

archivebox.cli.archivebox_list module

archivebox.cli.archivebox_list.**main**(*args: Optional[List[str]] = None*, *stdin: Optional[IO] = None*, *pwd: Optional[str] = None*) → None

List, filter, and export information about archive entries

archivebox.cli.archivebox_manage module

archivebox.cli.archivebox_manage.**main**(*args: Optional[List[str]] = None*, *stdin: Optional[IO] = None*, *pwd: Optional[str] = None*) → None

Run an ArchiveBox Django management command

archivebox.cli.archivebox_remove module

archivebox.cli.archivebox_remove.**main**(*args: Optional[List[str]] = None*, *stdin: Optional[IO] = None*, *pwd: Optional[str] = None*) → None

Remove the specified URLs from the archive

archivebox.cli.archivebox_schedule module

archivebox.cli.archivebox_schedule.**main**(*args: Optional[List[str]] = None*, *stdin: Optional[IO] = None*, *pwd: Optional[str] = None*) → None

Set ArchiveBox to regularly import URLs at specific times using cron

archivebox.cli.archivebox_server module

archivebox.cli.archivebox_server.**main**(*args: Optional[List[str]] = None*, *stdin: Optional[IO] = None*, *pwd: Optional[str] = None*) → None

Run the ArchiveBox HTTP server

archivebox.cli.archivebox_shell module

archivebox.cli.archivebox_shell.**main**(*args: Optional[List[str]] = None*, *stdin: Optional[IO] = None*, *pwd: Optional[str] = None*) → None

Enter an interactive ArchiveBox Django shell

archivebox.cli.archivebox_update module

archivebox.cli.archivebox_update.**main**(*args: Optional[List[str]] = None*, *stdin: Optional[IO] = None*, *pwd: Optional[str] = None*) → None

Import any new links from subscriptions and retry any previously failed/skipped links

archivebox.cli.archivebox_version module

archivebox.cli.archivebox_version.**main**(*args: Optional[List[str]] = None*, *stdin: Optional[IO] = None*, *pwd: Optional[str] = None*) → None

Print the ArchiveBox version and dependency information

archivebox.cli.logging module

class archivebox.cli.logging.**RuntimeStats**(*skipped: int = 0*, *succeeded: int = 0*, *failed: int = 0*, *parse_start_ts: Optional[datetime.datetime] = None*, *parse_end_ts: Optional[datetime.datetime] = None*, *index_start_ts: Optional[datetime.datetime] = None*, *index_end_ts: Optional[datetime.datetime] = None*, *archiving_start_ts: Optional[datetime.datetime] = None*, *archiving_end_ts: Optional[datetime.datetime] = None*)

Bases: object

mutable stats counter for logging archiving timing info to CLI output

skipped = 0

succeeded = 0

failed = 0

parse_start_ts = None

parse_end_ts = None

index_start_ts = None

index_end_ts = None

archiving_start_ts = None

archiving_end_ts = None

class archivebox.cli.logging.**SmartFormatter**(*prog*, *indent_increment=2*, *max_help_position=24*, *width=None*)

Bases: argparse.HelpFormatter

Patched formatter that prints newlines in argparse help strings

archivebox.cli.logging.**reject_stdin**(*caller: str*, *stdin: Optional[IO] = <_io.TextIOWrapper name='<stdin>' mode='r' encoding='UTF-8'>*) → None

Tell the user they passed stdin to a command that doesn't accept it

archivebox.cli.logging.**accept_stdin**(*stdin: Optional[IO] = <_io.TextIOWrapper name='<stdin>' mode='r' encoding='UTF-8'>*) → Optional[str]

accept any standard input and return it as a string or None

class archivebox.cli.logging.**TimedProgress**(*seconds*, *prefix=''*)

Bases: object

Show a progress bar and measure elapsed time until .end() is called

end()

immediately end progress, clear the progressbar line, and save end_ts

archivebox.cli.logging.**progress_bar**(*seconds: int*, *prefix: str = ''*) → None

show timer in the form of progress bar, with percentage and seconds remaining

archivebox.cli.logging.**log_parsing_started**(*source_file: str*)

archivebox.cli.logging.**log_parsing_finished**(*num_parsed: int*, *num_new_links: int*, *parser_name: str*)

archivebox.cli.logging.**log_indexing_process_started**(*num_links: int*)

archivebox.cli.logging.**log_indexing_process_finished**()

archivebox.cli.logging.**log_indexing_started**(*out_path: str*)

archivebox.cli.logging.**log_indexing_finished**(*out_path: str*)

archivebox.cli.logging.**log_archiving_started**(*num_links: int*, *resume: Optional[float] = None*)

archivebox.cli.logging.**log_archiving_paused**(*num_links: int*, *idx: int*, *timestamp: str*)

archivebox.cli.logging.**log_archiving_finished**(*num_links: int*)

archivebox.cli.logging.**log_link_archiving_started**(*link: archivebox.index.schema.Link*, *link_dir: str*, *is_new: bool*)

archivebox.cli.logging.**log_link_archiving_finished**(*link: archivebox.index.schema.Link*, *link_dir: str*, *is_new: bool*, *stats: dict*)

archivebox.cli.logging.**log_archive_method_started**(*method: str*)

archivebox.cli.logging.**log_archive_method_finished**(*result: archivebox.index.schema.ArchiveResult*)

quote the argument with whitespace in a command so the user can copy-paste the outputted string directly to run the cmd

archivebox.cli.logging.**log_list_started**(*filter_patterns: Optional[List[str]]*, *filter_type: str*)

archivebox.cli.logging.**log_list_finished**(*links*)

archivebox.cli.logging.**log_removal_started**(*links: List[archivebox.index.schema.Link]*, *yes: bool*, *delete: bool*)

archivebox.cli.logging.**log_removal_finished**(*all_links: int*, *to_keep: int*)

archivebox.cli.logging.**log_shell_welcome_msg**()

archivebox.cli.logging.**pretty_path**(*path: str*) → str
convert paths like .../ArchiveBox/archivebox/../output/abc into output/abc

archivebox.cli.logging.**printable_filesize**(*num_bytes: Union[int, float]*) → str

archivebox.cli.logging.**printable_folders**(*folders: Dict[str, Optional[archivebox.index.schema.Link]], json: bool = False, csv: Optional[str] = None*) → str

archivebox.cli.logging.**printable_config**(*config: importlib._bootstrap.ConfigDict, prefix: str = ''*) → str

archivebox.cli.logging.**printable_folder_status**(*name: str, folder: Dict[KT, VT]*) → str

archivebox.cli.logging.**printable_dependency_version**(*name: str, dependency: Dict[KT, VT]*) → str

archivebox.cli.tests module

archivebox.cli.tests.**output_hidden**(*show_failing=True*)

class archivebox.cli.tests.**TestInit**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()
Hook method for setting up the test fixture before exercising it.

tearDown()
Hook method for deconstructing the test fixture after testing it.

test_basic_init()

test_conflicting_init()

test_no_dirty_state()

class archivebox.cli.tests.**TestAdd**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()
Hook method for setting up the test fixture before exercising it.

tearDown()
Hook method for deconstructing the test fixture after testing it.

test_add_arg_url()

test_add_arg_file()

test_add_stdin_url()

class archivebox.cli.tests.**TestRemove**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()
Hook method for setting up the test fixture before exercising it.

test_remove_exact()

test_remove_regex()

test_remove_domain()

test_remove_none()

Module contents

archivebox.cli.**list_subcommands**() → Dict[str, str]
find and import all valid archivebox_<subcommand>.py files in CLI_DIR

archivebox.cli.**run_subcommand**(*subcommand: str*, *subcommand_args: List[str] = None*, *stdin: Optional[IO] = None*, *pwd: Optional[str] = None*) → None
Run a given ArchiveBox subcommand with the given list of args

archivebox.cli.**help**(*args: Optional[List[str]] = None*, *stdin: Optional[IO] = None*, *pwd: Optional[str] = None*) → None
Print the ArchiveBox help message and usage

archivebox.cli.**version**(*args: Optional[List[str]] = None*, *stdin: Optional[IO] = None*, *pwd: Optional[str] = None*) → None
Print the ArchiveBox version and dependency information

archivebox.cli.**init**(*args: Optional[List[str]] = None*, *stdin: Optional[IO] = None*, *pwd: Optional[str] = None*) → None
Initialize a new ArchiveBox collection in the current directory

archivebox.cli.**info**(*args: Optional[List[str]] = None*, *stdin: Optional[IO] = None*, *pwd: Optional[str] = None*) → None
Print out some info and statistics about the archive collection

archivebox.cli.**config**(*args: Optional[List[str]] = None*, *stdin: Optional[IO] = None*, *pwd: Optional[str] = None*) → None
Get and set your ArchiveBox project configuration values

archivebox.cli.**add**(*args: Optional[List[str]] = None*, *stdin: Optional[IO] = None*, *pwd: Optional[str] = None*) → None
Add a new URL or list of URLs to your archive

archivebox.cli.**remove**(*args: Optional[List[str]] = None*, *stdin: Optional[IO] = None*, *pwd: Optional[str] = None*) → None
Remove the specified URLs from the archive

archivebox.cli.**update**(*args: Optional[List[str]] = None*, *stdin: Optional[IO] = None*, *pwd: Optional[str] = None*) → None
Import any new links from subscriptions and retry any previously failed/skipped links

archivebox.cli.**list**(*args: Optional[List[str]] = None*, *stdin: Optional[IO] = None*, *pwd: Optional[str] = None*) → None
List, filter, and export information about archive entries

archivebox.cli.**shell**(*args: Optional[List[str]] = None*, *stdin: Optional[IO] = None*, *pwd: Optional[str] = None*) → None
Enter an interactive ArchiveBox Django shell

archivebox.cli.**manage**(*args: Optional[List[str]] = None*, *stdin: Optional[IO] = None*, *pwd: Optional[str] = None*) → None
Run an ArchiveBox Django management command

archivebox.cli.**server**(*args: Optional[List[str]] = None*, *stdin: Optional[IO] = None*, *pwd: Optional[str] = None*) → None
Run the ArchiveBox HTTP server

archivebox.cli.**schedule**(*args: Optional[List[str]] = None*, *stdin: Optional[IO] = None*, *pwd: Optional[str] = None*) → None
Set ArchiveBox to regularly import URLs at specific times using cron

archivebox.config package

Submodules

archivebox.config.stubs module

class archivebox.config.stubs.**ConfigDefault**
Bases: dict

Module contents

archivebox.config.**get_real_name**(*key: str*) → str

archivebox.config.**load_config_val**(*key: str, default: Union[str, bool, int, None, Pattern[AnyStr], Dict[str, Any], Dict[str, Union[str, bool, int, None, Pattern[AnyStr], Dict[str, Any]]], Callable[[], Union[str, bool, int, None, Pattern[AnyStr], Dict[str, Any]]], Callable[[importlib._bootstrap.ConfigDict], Union[str, bool, int, None, Pattern[AnyStr], Dict[str, Any], Dict[str, Union[str, bool, int, None, Pattern[AnyStr], Dict[str, Any]]], Callable[[], Union[str, bool, int, None, Pattern[AnyStr], Dict[str, Any]]]]]] = None, type: Optional[Type[CT_co]] = None, aliases: Optional[Tuple[str, ...]] = None, config: Optional[importlib._bootstrap.ConfigDict] = None, env_vars: Optional[os._Environ] = None, config_file_vars: Optional[Dict[str, str]] = None*) → Union[str, bool, int, None, Pattern[AnyStr], Dict[str, Any], Dict[str, Union[str, bool, int, None, Pattern[AnyStr], Dict[str, Any]]], Callable[[], Union[str, bool, int, None, Pattern[AnyStr], Dict[str, Any]]]]

archivebox.config.**load_config_file**(*out_dir: str = None*) → Optional[Dict[str, str]]
load the ini-formatted config file from OUTPUT_DIR/Archivebox.conf

archivebox.config.**write_config_file**(*config: Dict[str, str], out_dir: str = None*) → importlib._bootstrap.ConfigDict
load the ini-formatted config file from OUTPUT_DIR/Archivebox.conf

archivebox.config.**load_config**(*defaults: Dict[str, archivebox.config.stubs.ConfigDefault], config: Optional[importlib._bootstrap.ConfigDict] = None, out_dir: Optional[str] = None, env_vars: Optional[os._Environ] = None, config_file_vars: Optional[Dict[str, str]] = None*) → importlib._bootstrap.ConfigDict

archivebox.config.**stderr**(**args, color: Optional[str] = None, config: Optional[importlib._bootstrap.ConfigDict] = None*) → None

archivebox.config.**bin_version**(*binary: Optional[str]*) → Optional[str]
check the presence and return valid version line of a specified binary

archivebox.config.**bin_path**(*binary: Optional[str]*) → Optional[str]

archivebox.config.**bin_hash**(*binary: Optional[str]*) → Optional[str]

archivebox.config.**find_chrome_binary**() → Optional[str]
find any installed chrome binaries in the default locations

archivebox.config.**find_chrome_data_dir**() → Optional[str]
find any installed chrome user data directories in the default locations

archivebox.config.**wget_supports_compression**(*config*)

archivebox.config.**get_code_locations**(*config: importlib._bootstrap.ConfigDict*) → Dict[str, Union[str, bool, int, None, Pattern[AnyStr], Dict[str, Any]]]

archivebox.config.**get_external_locations**(*config: importlib._bootstrap.ConfigDict*) → Union[str, bool, int, None, Pattern[AnyStr], Dict[str, Any], Dict[str, Union[str, bool, int, None, Pattern[AnyStr], Dict[str, Any]]], Callable[[], Union[str, bool, int, None, Pattern[AnyStr], Dict[str, Any]]]]

archivebox.config.**get_data_locations**(*config: importlib._bootstrap.ConfigDict*) → Union[str, bool, int, None, Pattern[AnyStr], Dict[str, Any], Dict[str, Union[str, bool, int, None, Pattern[AnyStr], Dict[str, Any]]], Callable[[], Union[str, bool, int, None, Pattern[AnyStr], Dict[str, Any]]]]

archivebox.config.**get_dependency_info**(*config: importlib._bootstrap.ConfigDict*) → Union[str, bool, int, None, Pattern[AnyStr], Dict[str, Any], Dict[str, Union[str, bool, int, None, Pattern[AnyStr], Dict[str, Any]]], Callable[[], Union[str, bool, int, None, Pattern[AnyStr], Dict[str, Any]]]]

archivebox.config.**get_chrome_info**(*config: importlib._bootstrap.ConfigDict*) → Union[str, bool, int, None, Pattern[AnyStr], Dict[str, Any], Dict[str, Union[str, bool, int, None, Pattern[AnyStr], Dict[str, Any]]], Callable[[], Union[str, bool, int, None, Pattern[AnyStr], Dict[str, Any]]]]

archivebox.config.**load_all_config**()

archivebox.config.**check_system_config**(*config: importlib._bootstrap.ConfigDict = {'ANSI': {'black': '', 'blue': '', 'green': '', 'lightblue': '', 'lightred': '', 'lightyellow': '', 'red': '', 'reset': '', 'white': ''}, 'ARCHIVEBOX_BINARY': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/envs/v0.4.0/bin/archivebox', 'ARCHIVE_DIR': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0/archive', 'CHECK_SSL_VALIDITY': True, 'CHROME_BINARY': None, 'CHROME_HEADLESS': True, 'CHROME_OPTIONS': {'CHECK_SSL_VALIDITY': True, 'CHROME_BINARY': None, 'CHROME_HEADLESS': True, 'CHROME_SANDBOX': True, 'CHROME_USER_AGENT': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/73.0.3683.75 Safari/537.36', 'CHROME_USER_DATA_DIR': None, 'RESOLUTION': '1440,2000', 'TIMEOUT': 60}, 'CHROME_SANDBOX': True, 'CHROME_USER_AGENT': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/73.0.3683.75 Safari/537.36', 'CHROME_USER_DATA_DIR': None, 'CHROME_VERSION': None, 'CODE_LOCATIONS': {'PYTHON_DIR': {'enabled': True, 'is_valid': True, 'path': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0/archivebox'}, 'REPO_DIR': {'enabled': True, 'is_valid': True, 'path': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0'}, 'TEMPLATES_DIR': {'enabled': True, 'is_valid': True, 'path': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0/archivebox/templates'}, 'CONFIG_FILE': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0/archive/ArchiveBox.conf', 'COOKIES_FILE': None, 'CURL_BINARY': 'curl', 'CURL_VERSION': 'curl 7.58.0 (x86_64-pc-linux-gnu)', 'DATA_LOCATIONS': {'ARCHIVE_DIR': {'enabled': True, 'is_valid': False, 'path': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0/archive'}, 'CONFIG_FILE': {'enabled': True, 'is_valid': False, 'path': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0/archive/ArchiveBox.conf'}, 'HTML_INDEX': {'enabled': True, 'is_valid': False, 'path': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0/archive/index.html'}, 'JSON_INDEX': {'enabled': True, 'is_valid': False, 'path': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0/archive/index.json'}, 'LOGS_DIR': {'enabled': True, 'is_valid': False, 'path': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0/archive/logs'}, 'OUTPUT_DIR': {'enabled': True, 'is_valid': False, 'path': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0/archive'}, 'SOURCES_DIR': {'enabled': True, 'is_valid': False, 'path': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0/archive/sources'}, 'SQL_INDEX': {'enabled': True, 'is_valid': False, 'path':*

archivebox.config.**check_dependencies**(*config: importlib._bootstrap.ConfigDict = {'ANSI': {'black': '', 'blue': '', 'green': '', 'lightblue': '', 'lightred': '', 'lightyellow': '', 'red': '', 'reset': '', 'white': ''}, 'ARCHIVEBOX_BINARY': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/envs/v0.4.0/bin/ build', 'ARCHIVE_DIR': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0 'CHECK_SSL_VALIDITY': True, 'CHROME_BINARY': None, 'CHROME_HEADLESS': True, 'CHROME_OPTIONS': {'CHECK_SSL_VALIDITY': True, 'CHROME_BINARY': None, 'CHROME_HEADLESS': True, 'CHROME_SANDBOX': True, 'CHROME_USER_AGENT': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/73.0.3683.75 Safari/537.36', 'CHROME_USER_DATA_DIR': None, 'RESOLUTION': '1440, 2000', 'TIMEOUT': 60}, 'CHROME_SANDBOX': True, 'CHROME_USER_AGENT': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/73.0.3683.75 Safari/537.36', 'CHROME_USER_DATA_DIR': None, 'CHROME_VERSION': None, 'CODE_LOCATIONS': {'PYTHON_DIR': {'enabled': True, 'is_valid': True, 'path': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0 'REPO_DIR': {'enabled': True, 'is_valid': True, 'path': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0 'TEMPLATES_DIR': {'enabled': True, 'is_valid': True, 'path': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0 'CONFIG_FILE': '/home/docs/checkouts/readthedocs.org/user_builds/archiveb 'COOKIES_FILE': None, 'CURL_BINARY': 'curl', 'CURL_VERSION': 'curl 7.58.0 (x86_64-pc-linux-gnu)', 'DATA_LOCATIONS': {'ARCHIVE_DIR': {'enabled': True, 'is_valid': False, 'path': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0 'CONFIG_FILE': {'enabled': True, 'is_valid': False, 'path': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0 'HTML_INDEX': {'enabled': True, 'is_valid': False, 'path': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0 'JSON_INDEX': {'enabled': True, 'is_valid': False, 'path': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0 'LOGS_DIR': {'enabled': True, 'is_valid': False, 'path': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0 'OUTPUT_DIR': {'enabled': True, 'is_valid': False, 'path': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0 'SOURCES_DIR': {'enabled': True, 'is_valid': False, 'path': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0 'SQL_INDEX': {'enabled': True, 'is_valid': False, 'path': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0*

archivebox.config.**check_data_folder**(*out_dir: Optional[str] = None*, *config: importlib._bootstrap.ConfigDict = {'ANSI': {'black': '', 'blue': '', 'green': '', 'lightblue': '', 'lightred': '', 'lightyellow': '', 'red': '', 'reset': '', 'white': ''}, 'ARCHIVEBOX_BINARY': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/envs/v0.4.0/bin/sp
build', 'ARCHIVE_DIR': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0/
'CHECK_SSL_VALIDITY': True, 'CHROME_BINARY': None, 'CHROME_HEADLESS': True, 'CHROME_OPTIONS': {'CHECK_SSL_VALIDITY': True, 'CHROME_BINARY': None, 'CHROME_HEADLESS': True, 'CHROME_SANDBOX': True, 'CHROME_USER_AGENT': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/73.0.3683.75 Safari/537.36', 'CHROME_USER_DATA_DIR': None, 'RESOLUTION': '1440, 2000', 'TIMEOUT': 60}, 'CHROME_SANDBOX': True, 'CHROME_USER_AGENT': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/73.0.3683.75 Safari/537.36', 'CHROME_USER_DATA_DIR': None, 'CHROME_VERSION': None, 'CODE_LOCATIONS': {'PYTHON_DIR': {'enabled': True, 'is_valid': True, 'path': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0/
'REPO_DIR': {'enabled': True, 'is_valid': True, 'path': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0'
'TEMPLATES_DIR': {'enabled': True, 'is_valid': True, 'path': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0/
'CONFIG_FILE': '/home/docs/checkouts/readthedocs.org/user_builds/archivebo
'COOKIES_FILE': None, 'CURL_BINARY': 'curl', 'CURL_VERSION': 'curl 7.58.0 (x86_64-pc-linux-gnu)', 'DATA_LOCATIONS': {'ARCHIVE_DIR': {'enabled': True, 'is_valid': False, 'path': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0/
'CONFIG_FILE': {'enabled': True, 'is_valid': False, 'path': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0/
'HTML_INDEX': {'enabled': True, 'is_valid': False, 'path': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0/
'JSON_INDEX': {'enabled': True, 'is_valid': False, 'path': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0/
'LOGS_DIR': {'enabled': True, 'is_valid': False, 'path': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0/
'OUTPUT_DIR': {'enabled': True, 'is_valid': False, 'path': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0/
'SOURCES_DIR': {'enabled': True, 'is_valid': False, 'path': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0/
'SQL_INDEX': {'enabled': True, 'is_valid': False, 'path': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0/
'DEPENDENCIES': {'CHROME_BINARY': {'enabled':*

archivebox.config.**setup_django**(*out_dir: str = None*, *check_db=False*, *config: importlib._bootstrap.ConfigDict = {'ANSI': {'black': '', 'blue': '', 'green': '', 'lightblue': '', 'lightred': '', 'lightyellow': '', 'red': '', 'reset': '', 'white': ''}, 'ARCHIVEBOX_BINARY': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/envs/v0.4.0/bin/sphinx-build', 'ARCHIVE_DIR': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox*
'CHECK_SSL_VALIDITY': True, 'CHROME_BINARY': None, 'CHROME_HEADLESS': True, 'CHROME_OPTIONS': {'CHECK_SSL_VALIDITY': True, 'CHROME_BINARY': None, 'CHROME_HEADLESS': True, 'CHROME_SANDBOX': True, 'CHROME_USER_AGENT': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/73.0.3683.75 Safari/537.36', 'CHROME_USER_DATA_DIR': None, 'RESOLUTION': '1440, 2000', 'TIMEOUT': 60}, 'CHROME_SANDBOX': True, 'CHROME_USER_AGENT': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/73.0.3683.75 Safari/537.36', 'CHROME_USER_DATA_DIR': None, 'CHROME_VERSION': None, 'CODE_LOCATIONS': {'PYTHON_DIR': {'enabled': True, 'is_valid': True, 'path': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0/archive
'REPO_DIR': {'enabled': True, 'is_valid': True, 'path': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0'}, 'TEMPLATES_DIR': {'enabled': True, 'is_valid': True, 'path': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0/archive
'CONFIG_FILE': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkc
'COOKIES_FILE': None, 'CURL_BINARY': 'curl', 'CURL_VERSION': 'curl 7.58.0 (x86_64-pc-linux-gnu)', 'DATA_LOCATIONS': {'ARCHIVE_DIR': {'enabled': True, 'is_valid': False, 'path': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0/docs/ar
'CONFIG_FILE': {'enabled': True, 'is_valid': False, 'path': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0/docs/Ar
'HTML_INDEX': {'enabled': True, 'is_valid': False, 'path': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0/docs/in
'JSON_INDEX': {'enabled': True, 'is_valid': False, 'path': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0/docs/in
'LOGS_DIR': {'enabled': True, 'is_valid': False, 'path': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0/docs/lo
'OUTPUT_DIR': {'enabled': True, 'is_valid': False, 'path': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0/docs'}, 'SOURCES_DIR': {'enabled': True, 'is_valid': False, 'path': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0/docs/so
'SQL_INDEX': {'enabled': True, 'is_valid': False, 'path': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0/docs/in
'DEPENDENCIES': {'CHROME_BINARY': {'enabled': False, 'hash': None, 'is_valid': False, 'path': None, 'version': None}, 'CURL_BINARY': {'enabled': True, 'hash': 'md5:9962d21616045a8ad3d2fb10f5fc82cb', 'is_valid': True, 'path': '/usr/bin/curl', 'version': 'curl 7.58.0 (x86_64-pc-linux-gnu)'}, 'DJANGO_BINARY': {'enabled': True, 'hash': 'md5:d678482566f7cae731ae9d5e4a4125c5', 'is_valid': True, 'path': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/envs/v0.4.0/lib/py
packages/Django-2.2-py3.7.egg/django/bin/django-admin.py', 'version': '2.2.0 final (0)'}, 'GIT_BINARY': {'enabled': True, 'hash': 'md5:04170955ce66697b20587dd4e1813c80', 'is_valid': True, 'path': '/usr/bin/git', 'version': 'git version 2.17.1'}, 'PYTHON_BINARY': {'enabled': True, 'hash': 'md5:9962d21616045a8ad3d2fb10f5fc82cb', 'is_valid': True,

archivebox.config.**TERM_WIDTH**()

archivebox.core package

Subpackages

archivebox.core.migrations package

Submodules

archivebox.core.migrations.0001_initial module

class archivebox.core.migrations.0001_initial.**Migration**(*name*, *app_label*)
Bases: django.db.migrations.migration.Migration

initial = True

dependencies = []

operations = [<CreateModel name='Snapshot', fields=[('id', <django.db.models.fields.UU

archivebox.core.migrations.0002_auto_20190417_0739 module

Module contents

Submodules

archivebox.core.admin module

archivebox.core.apps module

class archivebox.core.apps.**CoreConfig**(*app_name*, *app_module*)
Bases: django.apps.config.AppConfig

name = 'core'

archivebox.core.models module

archivebox.core.settings module

archivebox.core.tests module

archivebox.core.urls module

archivebox.core.urls.**path**(*route*, *view*, *kwargs=None*, *name=None*, ***, *Pattern=<class 'django.urls.resolvers.RoutePattern'>*)

archivebox.core.views module

class archivebox.core.views.**MainIndex**(***kwargs*)
Bases: django.views.generic.base.View

template = 'main_index.html'

get(*request*)

class archivebox.core.views.**AddLinks**(***kwargs*)
Bases: django.views.generic.base.View

template = 'add_links.html'

get(*request*)

post(*request*)

class archivebox.core.views.**LinkDetails**(***kwargs*)
Bases: django.views.generic.base.View

get(*request*, *path*)

archivebox.core.welcome_message module

archivebox.core.wsgi module

WSGI config for archivebox project.

It exposes the WSGI callable as a module-level variable named `application`.

For more information on this file, see https://docs.djangoproject.com/en/2.1/howto/deployment/wsgi/

Module contents

archivebox.extractors package

Submodules

archivebox.extractors.archive_org module

archivebox.extractors.archive_org.**should_save_archive_dot_org**(*link: archivebox.index.schema.Link*, *out_dir: Optional[str] = None*) → bool

archivebox.extractors.archive_org.**save_archive_dot_org**(*link: archivebox.index.schema.Link*, *out_dir: Optional[str] = None*, *timeout: int = 60*) → archivebox.index.schema.ArchiveResult

submit site to archive.org for archiving via their service, save returned archive url

archivebox.extractors.archive_org.**parse_archive_dot_org_response**(*response: bytes*) → Tuple[List[str], List[str]]

archivebox.extractors.dom module

archivebox.extractors.dom.**should_save_dom**(*link: archivebox.index.schema.Link*, *out_dir: Optional[str] = None*) → bool

archivebox.extractors.dom.**save_dom**(*link: archivebox.index.schema.Link*, *out_dir: Optional[str] = None*, *timeout: int = 60*) → archivebox.index.schema.ArchiveResult

print HTML of site to file using chrome –dump-html

archivebox.extractors.favicon module

archivebox.extractors.favicon.**should_save_favicon**(*link: archivebox.index.schema.Link*, *out_dir: Optional[str] = None*) → bool

archivebox.extractors.favicon.**save_favicon**(*link: archivebox.index.schema.Link*, *out_dir: Optional[str] = None*, *timeout: int = 60*) → archivebox.index.schema.ArchiveResult

download site favicon from google's favicon api

archivebox.extractors.git module

archivebox.extractors.git.**should_save_git**(*link: archivebox.index.schema.Link*, *out_dir: Optional[str] = None*) → bool

archivebox.extractors.git.**save_git**(*link: archivebox.index.schema.Link*, *out_dir: Optional[str] = None*, *timeout: int = 60*) → archivebox.index.schema.ArchiveResult

download full site using git

archivebox.extractors.media module

archivebox.extractors.media.**should_save_media**(*link: archivebox.index.schema.Link*, *out_dir: Optional[str] = None*) → bool

archivebox.extractors.media.**save_media**(*link: archivebox.index.schema.Link*, *out_dir: Optional[str] = None*, *timeout: int = 3600*) → archivebox.index.schema.ArchiveResult

Download playlists or individual video, audio, and subtitles using youtube-dl

archivebox.extractors.pdf module

archivebox.extractors.pdf.**should_save_pdf**(*link: archivebox.index.schema.Link*, *out_dir: Optional[str] = None*) → bool

archivebox.extractors.pdf.**save_pdf**(*link: archivebox.index.schema.Link*, *out_dir: Optional[str] = None*, *timeout: int = 60*) → archivebox.index.schema.ArchiveResult

print PDF of site to file using chrome –headless

archivebox.extractors.screenshot module

archivebox.extractors.screenshot.**should_save_screenshot**(*link: archivebox.index.schema.Link*, *out_dir: Optional[str] = None*) → bool

archivebox.extractors.screenshot.**save_screenshot**(*link: archivebox.index.schema.Link*, *out_dir: Optional[str] = None*, *timeout: int = 60*) → archivebox.index.schema.ArchiveResult

take screenshot of site using chrome –headless

archivebox.extractors.title module

archivebox.extractors.title.**should_save_title**(*link: archivebox.index.schema.Link*, *out_dir: Optional[str] = None*) → bool

archivebox.extractors.title.**save_title**(*link: archivebox.index.schema.Link*, *out_dir: Optional[str] = None*, *timeout: int = 60*) → archivebox.index.schema.ArchiveResult

try to guess the page's title from its content

archivebox.extractors.wget module

archivebox.extractors.wget.**should_save_wget**(*link: archivebox.index.schema.Link*, *out_dir: Optional[str] = None*) → bool

archivebox.extractors.wget.**save_wget**(*link: archivebox.index.schema.Link*, *out_dir: Optional[str] = None*, *timeout: int = 60*) → archivebox.index.schema.ArchiveResult

download full site using wget

archivebox.extractors.wget.**wget_output_path**(*link: archivebox.index.schema.Link*) → Optional[str]

calculate the path to the wgetted .html file, since wget may adjust some paths to be different than the base_url path.

See docs on wget –adjust-extension (-E)

Module contents

archivebox.extractors.**archive_link**(*link: archivebox.index.schema.Link*, *overwrite: bool = False*, *out_dir: Optional[str] = None*) → archivebox.index.schema.Link

download the DOM, PDF, and a screenshot into a folder named after the link's timestamp

archivebox.index package

Submodules

archivebox.index.csv module

archivebox.index.csv.**links_to_csv**(*links: List[archivebox.index.schema.Link], cols: Optional[List[str]] = None, header: bool = True, separator: str = ',', ljust: int = 0*) → str

archivebox.index.csv.**to_csv**(*obj: Any, cols: List[str], separator: str = ',', ljust: int = 0*) → str

archivebox.index.html module

archivebox.index.html.**join**(**paths*)

archivebox.index.html.**parse_html_main_index**(*out_dir: str = '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/check*) → Iterator[str]

parse an archive index html file and return the list of urls

archivebox.index.html.**write_html_main_index**(*links: List[archivebox.index.schema.Link], out_dir: str = '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/check finished: bool = False*) → None

write the html link index to a given path

archivebox.index.html.**main_index_template**(*links: List[archivebox.index.schema.Link], finished: bool = True*) → str

render the template for the entire main index

archivebox.index.html.**main_index_row_template**(*link: archivebox.index.schema.Link*) → str

render the template for an individual link row of the main index

archivebox.index.html.**write_html_link_details**(*link: archivebox.index.schema.Link, out_dir: Optional[str] = None*) → None

archivebox.index.html.**link_details_template**(*link: archivebox.index.schema.Link*) → str

archivebox.index.html.**render_legacy_template**(*template_path: str, context: Mapping[str, str]*) → str

render a given html template string with the given template content

archivebox.index.json module

archivebox.index.json.**parse_json_main_index**(*out_dir: str = '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/check*) → Iterator[archivebox.index.schema.Link]

parse an archive index json file and return the list of links

archivebox.index.json.**write_json_main_index**(*links: List[archivebox.index.schema.Link], out_dir: str = '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/check*) → None

write the json link index to a given path

archivebox.index.json.**write_json_link_details**(*link: archivebox.index.schema.Link, out_dir: Optional[str] = None*) → None

write a json file with some info about the link

archivebox.index.json.**parse_json_link_details**(*out_dir: str*) → Optional[archivebox.index.schema.Link]

load the json link index from a given directory

archivebox.index.json.**parse_json_links_details**(*out_dir: str*) → Iterator[archivebox.index.schema.Link]

read through all the archive data folders and return the parsed links

class archivebox.index.json.**ExtendedEncoder**(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *default=None*)

Bases: json.encoder.JSONEncoder

Extended json serializer that supports serializing several model fields and objects

default(*obj*)

Implement this method in a subclass such that it returns a serializable object for `o`, or calls the base implementation (to raise a `TypeError`).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return JSONEncoder.default(self, o)
```

archivebox.index.json.**to_json**(*obj: Any*, *indent: Optional[int] = 4*, *sort_keys: bool = True*, *cls=<class 'archivebox.index.json.ExtendedEncoder'>*) → str

archivebox.index.schema module

exception archivebox.index.schema.**ArchiveError**(*message*, *hints=None*)

Bases: Exception

class archivebox.index.schema.**ArchiveResult**(*cmd: List[str]*, *pwd: Union[str, NoneType]*, *cmd_version: Union[str, NoneType]*, *output: Union[str, Exception, NoneType]*, *status: str*, *start_ts: datetime.datetime*, *end_ts: datetime.datetime*, *schema: str = 'ArchiveResult'*)

Bases: object

schema = 'ArchiveResult'

typecheck() → None

classmethod from_json(*json_info*)

to_dict(**keys*) → dict

to_json(*indent=4*, *sort_keys=True*) → str

to_csv(*cols: Optional[List[str]] = None*, *separator: str = ','*, *ljust: int = 0*) → str

classmethod field_names()

duration

class archivebox.index.schema.**Link**(*timestamp: str, url: str, title: Union[str, NoneType], tags: Union[str, NoneType], sources: List[str], history: Dict[str, List[archivebox.index.schema.ArchiveResult]] = <factory>, updated: Union[datetime.datetime, NoneType] = None, schema: str = 'Link'*)

Bases: object

updated = None

schema = 'Link'

overwrite(***kwargs*)
pure functional version of dict.update that returns a new instance

typecheck() → None

classmethod from_json(*json_info*)

to_json(*indent=4*, *sort_keys=True*) → str

to_csv(*cols: Optional[List[str]] = None*, *separator: str = ','*, *ljust: int = 0*) → str

classmethod field_names()

link_dir

archive_path

url_hash

scheme

extension

domain

path

basename

base_url

bookmarked_date

updated_date

archive_dates

oldest_archive_date

newest_archive_date

num_outputs

num_failures

is_static

is_archived

latest_outputs(*status: str = None*) → Dict[str, Union[str, Exception, None]]
get the latest output that each archive method produced for link

canonical_outputs() → Dict[str, Optional[str]]
predict the expected output paths that should be present after archiving

archivebox.index.sql module

archivebox.index.sql.**parse_sql_main_index**(*out_dir: str = '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkou*) → Iterator[archivebox.index.schema.Link]

archivebox.index.sql.**write_sql_main_index**(*links: List[archivebox.index.schema.Link], out_dir: str = '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkou*) → None

archivebox.index.sql.**list_migrations**(*out_dir: str = '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/ch*) → List[Tuple[bool, str]]

archivebox.index.sql.**apply_migrations**(*out_dir: str = '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/*) → List[str]

archivebox.index.sql.**get_admins**(*out_dir: str = '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts*) → List[str]

Module contents

archivebox.index.**merge_links**(*a: archivebox.index.schema.Link, b: archivebox.index.schema.Link*) → archivebox.index.schema.Link

deterministially merge two links, favoring longer field values over shorter, and "cleaner" values over worse ones.

archivebox.index.**validate_links**(*links: Iterable[archivebox.index.schema.Link]*) → List[archivebox.index.schema.Link]

archivebox.index.**archivable_links**(*links: Iterable[archivebox.index.schema.Link]*) → Iterable[archivebox.index.schema.Link]

remove chrome://, about:// or other schemed links that cant be archived

archivebox.index.**uniquefied_links**(*sorted_links: Iterable[archivebox.index.schema.Link]*) → Iterable[archivebox.index.schema.Link]

ensures that all non-duplicate links have monotonically increasing timestamps

archivebox.index.**sorted_links**(*links: Iterable[archivebox.index.schema.Link]*) → Iterable[archivebox.index.schema.Link]

archivebox.index.**links_after_timestamp**(*links: Iterable[archivebox.index.schema.Link], resume: Optional[float] = None*) → Iterable[archivebox.index.schema.Link]

archivebox.index.**lowest_uniq_timestamp**(*used_timestamps: collections.OrderedDict, timestamp: str*) → str

resolve duplicate timestamps by appending a decimal 1234, 1234 -> 1234.1, 1234.2

archivebox.index.**timed_index_update**(*out_path: str*)

archivebox.index.**write_main_index**(*links: List[archivebox.index.schema.Link], out_dir: str = '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0/doc, finished: bool = False*) → None

create index.html file for a given list of links

archivebox.index.**load_main_index**(*out_dir: str = '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkou, warn: bool = True*) → List[archivebox.index.schema.Link]

parse and load existing index with any new links from import_path merged in

archivebox.index.**load_main_index_meta**(*out_dir: str = '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/*) → Optional[dict]

archivebox.index.**import_new_links**(*existing_links: List[archivebox.index.schema.Link], import_path: str, out_dir: str = '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0/doc*) → Tuple[List[archivebox.index.schema.Link], List[archivebox.index.schema.Link]]

archivebox.index.**patch_main_index**(*link: archivebox.index.schema.Link, out_dir: str = '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0/doc*) → None

hack to in-place update one row's info in the generated index files

archivebox.index.**write_link_details**(*link: archivebox.index.schema.Link, out_dir: Optional[str] = None*) → None

archivebox.index.**load_link_details**(*link: archivebox.index.schema.Link, out_dir: Optional[str] = None*) → archivebox.index.schema.Link

check for an existing link archive in the given directory, and load+merge it into the given link dict

archivebox.index.**link_matches_filter**(*link: archivebox.index.schema.Link, filter_patterns: List[str], filter_type: str = 'exact'*) → bool

archivebox.index.**get_indexed_folders**(*links, out_dir: str = '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0*) → Dict[str, Optional[archivebox.index.schema.Link]]

indexed links without checking archive status or data directory validity

archivebox.index.**get_archived_folders**(*links, out_dir: str = '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4*) → Dict[str, Optional[archivebox.index.schema.Link]]

indexed links that are archived with a valid data directory

archivebox.index.**get_unarchived_folders**(*links, out_dir: str = '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v*) → Dict[str, Optional[archivebox.index.schema.Link]]

indexed links that are unarchived with no data directory or an empty data directory

archivebox.index.**get_present_folders**(*links, out_dir: str = '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0*) → Dict[str, Optional[archivebox.index.schema.Link]]

dirs that actually exist in the archive/ folder

archivebox.index.**get_valid_folders**(*links, out_dir: str = '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0/do*) → Dict[str, Optional[archivebox.index.schema.Link]]

dirs with a valid index matched to the main index and archived content

archivebox.index.**get_invalid_folders**(*links, out_dir: str = '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0*) → Dict[str, Optional[archivebox.index.schema.Link]]

dirs that are invalid for any reason: corrupted/duplicate/orphaned/unrecognized

archivebox.index.**get_duplicate_folders**(*links, out_dir: str = '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0*) → Dict[str, Optional[archivebox.index.schema.Link]]

dirs that conflict with other directories that have the same link URL or timestamp

archivebox.index.**get_orphaned_folders**(*links, out_dir: str = '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4*) → Dict[str, Optional[archivebox.index.schema.Link]]

dirs that contain a valid index but aren't listed in the main index

archivebox.index.**get_corrupted_folders**(*links*, *out_dir: str = '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0*) → Dict[str, Optional[archivebox.index.schema.Link]]

dirs that don't contain a valid index and aren't listed in the main index

archivebox.index.**get_unrecognized_folders**(*links*, *out_dir: str = '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkou*) → Dict[str, Optional[archivebox.index.schema.Link]]

dirs that don't contain recognizable archive data and aren't listed in the main index

archivebox.index.**is_valid**(*link: archivebox.index.schema.Link*) → bool

archivebox.index.**is_corrupt**(*link: archivebox.index.schema.Link*) → bool

archivebox.index.**is_archived**(*link: archivebox.index.schema.Link*) → bool

archivebox.index.**is_unarchived**(*link: archivebox.index.schema.Link*) → bool

archivebox.index.**fix_invalid_folder_locations**(*out_dir: str = '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/ch*) → Tuple[List[str], List[str]]

archivebox.parsers package

Submodules

archivebox.parsers.generic_json module

archivebox.parsers.generic_json.**parse_generic_json_export**(*json_file: IO[str]*) → Iterable[archivebox.index.schema.Link]

Parse JSON-format bookmarks export files (produced by pinboard.in/export/, or wallabag)

archivebox.parsers.generic_rss module

archivebox.parsers.generic_rss.**parse_generic_rss_export**(*rss_file: IO[str]*) → Iterable[archivebox.index.schema.Link]

Parse RSS XML-format files into links

archivebox.parsers.generic_txt module

archivebox.parsers.generic_txt.**parse_generic_txt_export**(*text_file: IO[str]*) → Iterable[archivebox.index.schema.Link]

Parse raw links from each line in a text file

archivebox.parsers.medium_rss module

archivebox.parsers.medium_rss.**parse_medium_rss_export**(*rss_file: IO[str]*) → Iterable[archivebox.index.schema.Link]

Parse Medium RSS feed files into links

archivebox.parsers.netscape_html module

archivebox.parsers.netscape_html.**parse_netscape_html_export**(*html_file: IO[str]*) → Iterable[archivebox.index.schema.Link]

Parse netscape-format bookmarks export files (produced by all browsers)

archivebox.parsers.pinboard_rss module

archivebox.parsers.pinboard_rss.**parse_pinboard_rss_export**(*rss_file: IO[str]*) → Iterable[archivebox.index.schema.Link]

Parse Pinboard RSS feed files into links

archivebox.parsers.pocket_html module

archivebox.parsers.pocket_html.**parse_pocket_html_export**(*html_file: IO[str]*) → Iterable[archivebox.index.schema.Link]

Parse Pocket-format bookmarks export files (produced by getpocket.com/export/)

archivebox.parsers.shaarli_rss module

archivebox.parsers.shaarli_rss.**parse_shaarli_rss_export**(*rss_file: IO[str]*) → Iterable[archivebox.index.schema.Link]

Parse Shaarli-specific RSS XML-format files into links

Module contents

Everything related to parsing links from input sources.

For a list of supported services, see the README.md. For examples of supported import formats see tests/.

archivebox.parsers.**parse_links**(*source_file: str*) → Tuple[List[archivebox.index.schema.Link], str]

parse a list of URLs with their metadata from an RSS feed, bookmarks export, or text file

archivebox.parsers.**save_stdin_to_sources**(*raw_text: str*, *out_dir: str = '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts*) → str

archivebox.parsers.**save_file_to_sources**(*path: str*, *timeout: int = 60*, *out_dir: str = '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v*) → str

download a given url's content into output/sources/domain-<timestamp>.txt

archivebox.parsers.**check_url_parsing_invariants**() → None

Check that plain text regex URL parsing works as expected

Submodules

archivebox.main module

archivebox.main.**help**(*out_dir: str = '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0/docs'*) → None

Print the ArchiveBox help message and usage

archivebox.main.**version**(*quiet: bool = False*, *out_dir: str = '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0/docs'*) → None

Print the ArchiveBox version and dependency information

archivebox.main.**run**(*subcommand: str*, *subcommand_args: Optional[List[str]]*, *stdin: Optional[IO] = None*, *out_dir: str = '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0/docs'*) → None

Run a given ArchiveBox subcommand with the given list of args

archivebox.main.**init**(*force: bool = False*, *out_dir: str = '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0/docs'*) → None

Initialize a new ArchiveBox collection in the current directory

archivebox.main.**info**(*out_dir: str = '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0/docs'*) → None

Print out some info and statistics about the archive collection

archivebox.main.**add**(*import_str: Optional[str] = None*, *import_path: Optional[str] = None*, *update_all: bool = False*, *index_only: bool = False*, *out_dir: str = '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0/docs'*) → List[archivebox.index.schema.Link]

Add a new URL or list of URLs to your archive

archivebox.main.**remove**(*filter_str: Optional[str] = None*, *filter_patterns: Optional[List[str]] = None*, *filter_type: str = 'exact'*, *after: Optional[float] = None*, *before: Optional[float] = None*, *yes: bool = False*, *delete: bool = False*, *out_dir: str = '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0/docs'*) → List[archivebox.index.schema.Link]

Remove the specified URLs from the archive

archivebox.main.**update**(*resume: Optional[float] = None*, *only_new: bool = True*, *index_only: bool = False*, *overwrite: bool = False*, *filter_patterns_str: Optional[str] = None*, *filter_patterns: Optional[List[str]] = None*, *filter_type: Optional[str] = None*, *status: Optional[str] = None*, *after: Optional[str] = None*, *before: Optional[str] = None*, *out_dir: str = '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0/docs'*) → List[archivebox.index.schema.Link]

Import any new links from subscriptions and retry any previously failed/skipped links

archivebox.main.**list_all**(*filter_patterns_str: Optional[str] = None*, *filter_patterns: Optional[List[str]] = None*, *filter_type: str = 'exact'*, *status: Optional[str] = None*, *after: Optional[float] = None*, *before: Optional[float] = None*, *sort: Optional[str] = None*, *csv: Optional[str] = None*, *json: bool = False*, *out_dir: str = '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0/docs'*) → Iterable[archivebox.index.schema.Link]

List, filter, and export information about archive entries

archivebox.main.**list_links**(*filter_patterns: Optional[List[str]] = None, filter_type: str = 'exact', after: Optional[float] = None, before: Optional[float] = None, out_dir: str = '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0/docs'*) → Iterable[archivebox.index.schema.Link]

archivebox.main.**list_folders**(*links: List[archivebox.index.schema.Link], status: str, out_dir: str = '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0/docs'*) → Dict[str, Optional[archivebox.index.schema.Link]]

archivebox.main.**config**(*config_options_str: Optional[str] = None, config_options: Optional[List[str]] = None, get: bool = False, set: bool = False, reset: bool = False, out_dir: str = '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0/docs'*) → None

Get and set your ArchiveBox project configuration values

archivebox.main.**schedule**(*add: bool = False, show: bool = False, clear: bool = False, foreground: bool = False, run_all: bool = False, quiet: bool = False, every: Optional[str] = None, import_path: Optional[str] = None, out_dir: str = '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0/docs'*)

Set ArchiveBox to regularly import URLs at specific times using cron

archivebox.main.**server**(*runserver_args: Optional[List[str]] = None, reload: bool = False, debug: bool = False, out_dir: str = '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0/docs'*) → None

Run the ArchiveBox HTTP server

archivebox.main.**manage**(*args: Optional[List[str]] = None, out_dir: str = '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0/docs'*) → None

Run an ArchiveBox Django management command

archivebox.main.**shell**(*out_dir: str = '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.0/docs'*) → None

Enter an interactive ArchiveBox Django shell

archivebox.manage module

archivebox.system module

archivebox.system.**run**(**popenargs, input=None, capture_output=False, timeout=None, check=False, **kwargs*)

Patched of subprocess.run to fix blocking io making timeout=innefective

archivebox.system.**atomic_write**(*contents: Union[dict, str, bytes], path: str*) → None

Safe atomic write to filesystem by writing to temp file + atomic rename

archivebox.system.**chmod_file**(*path: str, cwd: str = '.', permissions: str = '755', timeout: int = 30*) → None

chmod -R <permissions> <cwd>/<path>

archivebox.system.**copy_and_overwrite**(*from_path: str, to_path: str*)

copy a given file or directory to a given path, overwriting the destination

archivebox.system.**get_dir_size**(*path: str, recursive: bool = True, pattern: Optional[str] = None*) → Tuple[int, int, int]

get the total disk size of a given directory, optionally summing up recursively and limiting to a given filter list

archivebox.system.**dedupe_cron_jobs**(*cron: crontab.CronTab*) → crontab.CronTab

archivebox.util module

archivebox.util.**scheme**(*url*)

archivebox.util.**without_scheme**(*url*)

archivebox.util.**without_query**(*url*)

archivebox.util.**without_fragment**(*url*)

archivebox.util.**without_path**(*url*)

archivebox.util.**path**(*url*)

archivebox.util.**basename**(*url*)

archivebox.util.**domain**(*url*)

archivebox.util.**query**(*url*)

archivebox.util.**fragment**(*url*)

archivebox.util.**extension**(*url*)

archivebox.util.**base_url**(*url*)

archivebox.util.**without_www**(*url*)

archivebox.util.**without_trailing_slash**(*url*)

archivebox.util.**hashurl**(*url*)

archivebox.util.**is_static_file**(*url*)

archivebox.util.**urlencode**(*s*)

archivebox.util.**urldecode**(*s*)

archivebox.util.**htmlencode**(*s*)

archivebox.util.**htmldecode**(*s*)

archivebox.util.**short_ts**(*ts*)

archivebox.util.**ts_to_date**(*ts*)

archivebox.util.**ts_to_iso**(*ts*)

archivebox.util.**enforce_types**(*func*)
Enforce function arg and kwarg types at runtime using its python3 type hints

archivebox.util.**docstring**(*text: Optional[str]*)
attach the given docstring to the decorated function

archivebox.util.**str_between**(*string: str*, *start: str*, *end: str = None*) → str
(<abc>12345</def>, <abc>, </def>) -> 12345

archivebox.util.**parse_date**(*date: Any*) → Optional[datetime.datetime]
Parse unix timestamps, iso format, and human-readable strings

archivebox.util.**download_url**(*url: str*, *timeout: int = 60*) → str
Download the contents of a remote url and return the text

archivebox.util.**chrome_args**(***options*) → List[str]
helper to build up a chrome shell command with arguments

class archivebox.util.**ExtendedEncoder**(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *default=None*)

Bases: json.encoder.JSONEncoder

Extended json serializer that supports serializing several model fields and objects

default(*obj*)

Implement this method in a subclass such that it returns a serializable object for o, or calls the base implementation (to raise a TypeError).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return JSONEncoder.default(self, o)
```

Module contents

1.5 Meta

1.5.1 Roadmap

Comment here to discuss the contribution roadmap:Official Roadmap Discussion.

If you feel like contributing a PR, some of these tasks are pretty easy. Feel free to open an issue if you need help getting started in any way!

IMPORTANT: *Please don't work on any of these major long-term tasks without contacting me first, work is already in progress for many of these, and I may have to reject your PR if it doesn't align with the existing work!*

Planned Specification

ArchiveBox is going to migrate towards this design spec over the next 3 months bit by bit as functionality gets implemented and refactors are released.

To see how much of this spec is scheduled / implemented / released so far, read these pull requests:

- v0.3.0
- v0.4.0

API:

- *pip install archivebox*
- *archivebox version*
- *archivebox help*

- *archivebox init*
- *archivebox info*
- *archivebox add*
- *archivebox remove*
- *archivebox schedule*
- *archivebox config*
- *archivebox update*
- *archivebox list*
- *archivebox oneshot*
- *archivebox server*
- *archivebox proxy*
- *archivebox shell*
- *archivebox manage*
- *from archivebox import ...*
- *from archivebox.component import ...*

Design:

- *Overview*
- *Dependencies*
- *Dependencies*
- *Code Layout*
- *Data Layout*
- *Export Layout*

CLI Usage

Note, these ways to run ArchiveBox are all equivalent:

- `archivebox [subcommand] [...args]`
- `python3 -m archivebox [subcommand] [...args]`
- `python3 archivebox/__main__.py [subcommand] [...args]`
- `python3 archivebox/manage.py archivebox [subcommand] [...args]`

`$ pip install archivebox`

```
...
Installing collected packages: archivebox
  Running setup.py install for archivebox ... done
Successfully installed archivebox-0.4.0
```

Developers who are working on the ArchiveBox codebase should install the project in "linked" mode for development using: `pipenv install; pip install -e .`.

$ archivebox [version|--version]

```
ArchiveBox v0.4.0

[i] Dependency versions:
   PYTHON_BINARY            /optArchiveBox/.venv/bin/python3.7            v3.7
↪    valid
   DJANGO_BINARY            /optArchiveBox/.venv/lib/python3.7/site-packages/django/
↪bin/django-admin.py v2.2.0          valid
   CURL_BINARY              /usr/bin/curl
↪             v7.54.0         valid
   WGET_BINARY              /usr/local/bin/wget
↪             v1.20.1         valid
   GIT_BINARY               /usr/local/bin/git
↪             v2.20.1         valid
   YOUTUBEDL_BINARY         /optArchiveBox/.venv/bin/youtube-dl           v2019.04.17
↪    valid
   CHROME_BINARY            /Applications/Google Chrome.app/Contents/MacOS/Google
↪Chrome           v74.0.3729.91   valid

[i] Folder locations:
   REPO_DIR                 /optArchiveBox                                28 files
↪    valid
   PYTHON_DIR               /optArchiveBox/archivebox                     14 files
↪    valid
   LEGACY_DIR               /optArchiveBox/archivebox/legacy              15 files
↪    valid
   TEMPLATES_DIR            /optArchiveBox/archivebox/legacy/templates    7 files
↪    valid
   OUTPUT_DIR               /optArchiveBox/archivebox/data                10 files
↪    valid
   SOURCES_DIR              /optArchiveBox/archivebox/data/sources        1 files
↪    valid
   LOGS_DIR                 /optArchiveBox/archivebox/data/logs           0 files
↪    valid
   ARCHIVE_DIR              /optArchiveBox/archivebox/data/archive        2 files
↪    valid
   CHROME_USER_DATA_DIR     /Users/squash/Library/Application Support/Chromium
↪             2 files         valid
 -  COOKIES_FILE
↪              -               disabled
↪                                      -               disabled
```

$ archivebox [help|-h|--help]

```
ArchiveBox: The self-hosted internet archive.

Documentation:
    https://github.com/pirate/ArchiveBox/wiki

UI Usage:
    Open output/index.html to view your archive.

CLI Usage:
    mkdir data; cd data/
```

(continues on next page)

(continued from previous page)

```
    archivebox init

    echo 'https://example.com/some/page' | archivebox add
    archivebox add https://example.com/some/other/page
    archivebox add --depth=1 ~/Downloads/bookmarks_export.html
    archivebox add --depth=1 https://example.com/feed.rss
    archivebox update --resume=15109948213.123
```

$ archivebox init

Initialize a new "collection" folder, aka a complete archive containing an ArchiveBox.conf config file, an index of all the archived pages, and the archived content for each page.

```
$ mkdir ~/my-archive && ~/my-archive
$ archivebox init
[+] Initializing a new ArchiveBox collection in this folder...
    ~/my-archive
------------------------------------------------------------------

[+] Building archive folder structure...
     ~/my-archive/sources
     ~/my-archive/archive
     ~/my-archive/logs

[+] Building main SQL index and running migrations...
     ~/my-archive/index.sqlite3

    Operations to perform:
      Apply all migrations: admin, auth, contenttypes, core, sessions
    Running migrations:
    Applying contenttypes.0001_initial... OK
    Applying auth.0001_initial... OK
    Applying admin.0001_initial... OK
    ...

[*] Collecting links from any existing index or archive folders...
     Loaded 30 links from existing main index...
    ! Skipped adding 2 orphaned link data directories that would have overwritten␣
↪existing data.
    ! Skipped adding 2 corrupted/unrecognized link data directories that could not be␣
↪read.
        For more information about the link data directories that were skipped, run:
            archivebox info
            archivebox list --status=invalid
            archivebox list --status=orphaned
            archivebox list --status=duplicate

[*] [2019-04-24 15:41:11] Writing 30 links to main index...
     ~/my-archive/index.sqlite3
     ~/my-archive/index.json
     ~/my-archive/index.html

------------------------------------------------------------------
[] Done. A new ArchiveBox collection was initialized (30 links).
```

(continues on next page)

(continued from previous page)

```
    To view your archive index, open:
        ~/my-archive/index.html

    To add new links, you can run:
        archivebox add 'https://example.com'

    For more usage and examples, run:
        archivebox help
```

$ archivebox info

Print out some info and statistics about the archive collection.

```
$ archivebox info
[*] Scanning archive collection main index...
    /Users/squash/Documents/Code/ArchiveBox/data/*
    Size: 209.3 KB across 3 files

    > JSON Main Index: 30 links      (found in index.json)
    > SQL Main Index: 30 links       (found in index.sqlite3)
    > HTML Main Index: 30 links      (found in index.html)
    > JSON Link Details: 1 links     (found in archive/*/index.json)
    > Admin: 0 users                 (found in index.sqlite3)

    Hint: You can create an admin user by running:
        archivebox manage createsuperuser

[*] Scanning archive collection link data directories...
    /Users/squash/Documents/Code/ArchiveBox/data/archive/*
    Size: 1.6 MB across 46 files in 50 directories

    > indexed: 30                    (indexed links without checking archive status␣
↪or data directory validity)
      > archived: 1                  (indexed links that are archived with a valid␣
↪data directory)
      > unarchived: 29               (indexed links that are unarchived with no data␣
↪directory or an empty data directory)

    > present: 1                     (dirs that are expected to exist based on the␣
↪main index)
      > valid: 1                     (dirs with a valid index matched to the main␣
↪index and archived content)
      > invalid: 0                   (dirs that are invalid for any reason: corrupted/
↪duplicate/orphaned/unrecognized)
        > duplicate: 0               (dirs that conflict with other directories that␣
↪have the same link URL or timestamp)
        > orphaned: 0                (dirs that contain a valid index but aren't␣
↪listed in the main index)
        > corrupted: 0               (dirs that don't contain a valid index and aren
↪'t listed in the main index)
        > unrecognized: 0            (dirs that don't contain recognizable archive␣
↪data and aren't listed in the main index)

    Hint: You can list link data directories by status like so:
        archivebox list --status=<status>  (e.g. indexed, corrupted, archived, etc.)
```

$ archivebox add

--only-new

Controls whether to only add new links or also retry previously failed/skipped links.

--index-only

Pass this to only add the links to the main index without archiving them.

--mirror

Archive an entire site (finding all linked pages below it on the same domain)

--depth

Controls how far to follow links from the given url. `0` sets it to only archive the page, and not follow any outlinks. `1` sets it to archive the page and follow one link outwards and archive those pages. `2` sets it to follow a maximum of two hops outwards, and so on...

--crawler=[type]

Controls which crawler to use in order to find outlinks in a given page.

url

Is the page you want to archive

< stdin

URLs to be added can also be piped in via stdin instead of passed as an argument

```
$ archivebox add --depth=1 https://example.com
[+] [2019-03-30 18:36:41] Adding 1 new url and all pages 1 hop out: https://example.
↪com
[*] [2019-03-30 18:36:42] Saving main index files...
     ./index.json
     ./index.html
[] [2019-03-30 18:36:42] Updating archive content...
[+] [2019-03-30 18:36:42] "Using Geolocation Data to Understand Consumer Behavior␣
↪During Severe Weather Events"
    https://orbitalinsight.com/using-geolocation-data-understand-consumer-behavior-
↪severe-weather-events
    > ./archive/1553789823
        > wget
        > warc
        > media
        > screenshot
```

(continues on next page)

(continued from previous page)

```
[] [2019-03-30 18:39:00] Update of 37 pages complete (2.08 sec)
    - 35 links skipped
    - 0 links updated
    - 2 links had errors
[*] [2019-03-30 18:39:00] Saving main index files...
     ./index.json
     ./index.html

    To view your archive, open:
        /Users/example/ArchiveBox/index.html
```

$ archivebox schedule

Use `python-crontab` to add, remove, and edit regularly scheduled archive update jobs.

--run-all

Run all the scheduled jobs once immediately, independent of their configured schedules

--foreground

Launch ArchiveBox as a long-running foreground task instead of using cron.

--show

Print a list of currently active ArchiveBox cron jobs

--clear

Stop all ArchiveBox scheduled runs, clear it completely from cron

--add

Add a new scheduled ArchiveBox update job to cron

--quiet

Don't warn about many jobs potentially using up storage space.

--every=[schedule]

The schedule to run the command can be either:

- `minute/hour/day/week/month/year`
- or a cron-formatted schedule like `"0/2 * * * *"/"* 0/10 * * * *"/...`

import_path

Specify the path as the path to a local file or remote URL to check for new links.

```
$ archivebox schedule --show
@hourly cd /optArchiveBox/data && /opt/ArchiveBox/.venv/bin/archivebox add "https://
↪getpocket.com/users/nikisweeting/feed/all" 2>&1 > /opt/ArchiveBox/data/logs/
↪archivebox.log # archivebox_schedule
```

```
$ archivebox schedule --add --every=hour https://getpocket.com/users/nikisweeting/
↪feed/all

[] Scheduled new ArchiveBox cron job for user: squash (1 jobs are active).
  > @hourly cd /Users/squash/Documents/Code/ArchiveBox/data && /Users/squash/
↪Documents/Code/ArchiveBox/.venv/bin/archivebox add "https://getpocket.com/users/
↪nikisweeting/feed/all" 2>&1 > /Users/squash/Documents/Code/ArchiveBox/data/logs/
↪archivebox.log # archivebox_schedule

[!] With the current cron config, ArchiveBox is estimated to run >365 times per year.
    Congrats on being an enthusiastic internet archiver!

    Make sure you have enough storage space available to hold all the data.
    Using a compressed/deduped filesystem like ZFS is recommended if you plan on␣
↪archiving a lot.
```

$ archivebox config

(no args)

Print the entire config to stdout.

--get KEY

Get the given config key:value and print it to stdout.

--set KEY=VALUE

Set the given config key:value in the current collection's config file.

< stdin

```
$ archviebox config
OUTPUT_DIR="output"
OUTPUT_PERMISSIONS=755
ONLY_NEW=False
...
```

```
$ archviebox --get CHROME_VERSION
Google Chrome 74.0.3729.40 beta
```

```
$ archviebox --set USE_CHROME=False
USE_CHROME=False
```

$ archivebox update

Check all subscribed feeds for new links, archive them and retry any previously failed pages.

(no args)

Update the index and go through each page, retrying any that failed previously.

--only-new

By default it always retries previously failed/skipped pages, pass this flag to only archive newly added links without going through the whole archive and attempting to fix previously failed links.

--resume=[timestamp]

Resume the update process from a specific URL timestamp.

--snapshot

[TODO] by default ArchiveBox never re-archives pages after the first successful archive, if you want to take a new snapshot of every page even if there's an existing version, pass this option.

$ archivebox list

--csv=COLUMNS

Print the output in CSV format, with the specified columns, e.g. `--csv=timestamp,base_url,is_archived`

--json

Print the output in JSON format (with all the link attributes included in the JSON output).

--filter=REGEX

Print only URLs matching a specified regex, e.g. `--filter='.*github.com.*'`

--before=TIMESTAMP / --after=TIMESTAMP

Print only URLs before or after a given timestamp, e.g. `--before=1554263415.2` or `--after=1554260000`

```
$ archivebox list --sort=timestamp
http://www.iana.org/domains/example
https://github.com/pirate/ArchiveBox/wiki
https://github.com/pirate/ArchiveBox/commit/0.4.0
https://github.com/pirate/ArchiveBox
https://archivebox.io
```

```
$ archivebox list --sort=timestamp --csv=timestamp,url
timestamp,url
1554260947,http://www.iana.org/domains/example
1554263415,https://github.com/pirate/ArchiveBox/wiki
1554263415.0,https://github.com/pirate/ArchiveBox/commit/0.4.0
1554263415.1,https://github.com/pirate/ArchiveBox
1554263415.2,https://archivebox.io
```

```
$ archivebox list --sort=timestamp --csv=timestamp,url --after=1554263415.0
timestamp,url
1554263415,https://github.com/pirate/ArchiveBox/wiki
1554263415.0,https://github.com/pirate/ArchiveBox/commit/0.4.0
1554263415.1,https://github.com/pirate/ArchiveBox
1554263415.2,https://archivebox.io
```

$ archivebox remove

--yes

Proceed with removal without prompting the user for confirmation.

--delete

Also delete all the matching links snapshot data folders and content files.

--filter-type

Defaults to `exact`, but can be set to any of `exact`, `substring`, `domain`, or `regex`.

pattern

The filter pattern used to match links in the index. Matching links are removed.

--before=TIMESTAMP / --after=TIMESTAMP

Remove any URLs bookmarked before/after the given timestamp, e.g. `--before=1554263415.2` or `--after=1554260000`.

```
$ archivebox remove --delete --filter-type=regex 'http(s)?:\\/\\/(.+)?(demo\\.
↪dev|example\\.com)\\/?.*'
[*] Finding links in the archive index matching these regex patterns:
    http(s)?:\/\/(.+)?(youtube\.com|example\.com)\/?.*

-------------------------------------------------------------------------------
↪-------------
timestamp        | is_archived      | num_outputs      | url
"1554984695"     | true             | 7                | "https://example.com"
-------------------------------------------------------------------------------
↪-------------

[i] Found 1 matching URLs to remove.
    1 Links will be de-listed from the main index, and their archived content folders␣
↪will be deleted from disk.
    (1 data folders with 7 archived files will be deleted!)

[?] Do you want to proceed with removing these 1 links?
    y/[n]: y

[*] [2019-04-11 08:11:57] Saving main index files...
     /opt/ArchiveBox/data/index.json
     /opt/ArchiveBox/data/index.html

[] Removed 1 out of 1 links from the archive index.
    Index now contains 0 links.
```

```
$ archivebox remove --yes --delete --filter-type=domain example.com
...
```

$ archivebox manage

Run a Django management command in the context of the current archivebox data directory.

[command] [...args]

The name of the management command to run, e.g.: `help`, `migrate`, `changepassword`, `createsuperuser`, etc.

```
$ archivebox manage help
Type 'archivebox manage help <subcommand>' for help on a specific subcommand.

Available subcommands:

[auth]
    changepassword
    createsuperuser

[contenttypes]
    remove_stale_contenttypes

[core]
    archivebox
```

(continues on next page)

(continued from previous page)

```
...
```

$ archivebox server

[ip:port]

The address:port combo to run the server on, defaults to `127.0.0.1:8000`.

```
$ archivebox server
[+] Starting ArchiveBox webserver...
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
April 23, 2019 - 01:40:52
Django version 2.2, using settings 'core.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

$ archivebox proxy

Run a live HTTP/HTTPS proxy for recording and replaying WARCs using pywb.

--bind=[ip:port]

The address:port combo to run the proxy on, defaults to `127.0.0.1:8010`.

--record

Save all traffic visited through the proxy to the archive.

--replay

Attempt to serve all pages visited through the proxy from the archive.

$ archivebox shell

Drop into an ArchiveBox Django shell with access to all models and data.

```
$ archivebox shell
Loaded archive data folder ~/example_collection...
Python 3.7.2 (default, Feb 12 2019, 08:15:36)

In [1]: url_to_archive = Link.objects.filter(is_archived=True).values_list('url',␣
↪flat=True)
...
```

`$ archivebox oneshot`

Create a single URL archive folder with an index.json and index.html, and all the archive method outputs. You can run this to archive single pages without needing to create a whole collection with `archivebox init`.

`--out-dir=[path]`

Path to save the single archive folder to, e.g. `./example.com_archive`.

`[--all|--media|--wget|...]`

Which archive methods to use when saving the URL.

Python Usage

API for normal ArchiveBox usage

```
from archivebox import add, subscribe, update

add('https://example.com', depth=2)
subscribe('https://example.com/some/feed.rss')
update(only_new=True)
```

API for All Useful Subcomponents

```
from archivebox import oneshot
from archivebox.crawl import rss
from archivebox.extract import media

links = crawl_rss(open('feed.rss', 'r').read())
assets = media.extract('https://youtube.com/watch?v=example')
oneshot('https://example.com', depth=2, out_dir='~/Desktop/example.com_archive')
```

Design

As of v0.4.0 ArchiveBox also writes the index to a `sqlite3` file using the Django ORM (in addition to the usual `json` and `html` formats, those aren't going away). To an end user, it will still appear to be a single CLI application, and none of the django complexity will be exposed. Django is used primarily because it allows for safe migrations of a sqlite database. As the schema gets updated in the future I don't want to break people's archives with every new version. It also allows us to have the GUI server start with many safe defaults and share much of the same codebase with the CLI and library components, including maintaining the archive database and managing a worker pool.

There will be 3 primary use cases for archivebox, and all three will be served by the pip package:

- simple CLI operation: `archivebox.cli import add --depth=1 ./path/to/export.html` (similar to current `archivebox` CLI)

- use of individual components as a library: `from archivebox.extract import screenshot` or `archivebox oneshot --screenshot ...`
- usage in server mode with a GUI to add/remove links and create exports: `archivebox server`

Dependencies:

- django (required)
- sqlite (required)
- headless chrome (required)
- wget (required)
- redis (optional, for web GUI only)
- dramatiq (optinal, for web GUI only)

When launched in webserver mode, archivebox will automatically spawn a pool of workers (dramatiq) as big as the number of CPUs available to use for crawling, archiving, and publishing.

When launched in CLI mode it will use normal subprocesses to do multithreading without redis/dramatiq.

Code Folder Layout

- archivebox/
  - core/
    - models.py Archive = Dict[Page, Dict[Archiver, List[Asset]]] # A collection of archived pages Crawl = List[Page] # list of links to add to an archive Page # an archived page with unique url Asset # a file archived from a page
    - util.py
    - settings.py
  - crawl/ impl: detect_crawlable(Import) -> bool crawl(Import) -> List[Page]
    - txt.py
    - rss.py
    - netscape.py
    - pocket.py
    - pinboard.py
    - html.py
  - extract/ impl: detect_extractable(Page) -> bool extract(Page) -> List[Asset]
    - wget.py
    - screenshot.py
    - pdf.py
    - dom.py
    - youtubedl.py
    - waybackmachine.py

        * solana.py
    - publish/ impl: publish(Archive, output_format)
        * html.py
        * json.py
        * csv.py
        * sql.py

Collection Data Folder Layout

- ArchiveBox.conf
- database/
    - sqlite.db
- archive
    - assets/<hash>/
- logs/
    - server.log
    - crawl.log
    - archive.log

Exported Folder Layout

For publishing the archive as static html/json/csv/sql.

- index.html,json,csv,sql
- archive/
    - <timestamp>/
        * index.html
        * <url>/
            · index.html,json,csv,sql
        * assets/
            · hash.mp4
            · hash.txt
            · hash.mp3

The server will be runnable with docker / docker-compose as well:

```
version: '3'

services:
    archivebox:
        image: archivebox
```

(continues on next page)

(continued from previous page)

```
        ports:
            - '8098:80'
        volumes:
            - ./data/:/data
```

Major long-term changes

- release **`pip`, `apt`, `pkg`, and `brew` packaged distributions** for installing ArchiveBox
- add an **optional web GUI** for managing sources, adding new links, and viewing the archive
- switch to django + **sqlite db with migrations system** & json/html export for managing archive schema changes and persistence
- modularize internals to allow importing individual components
- switch to sha256 of URL as unique link ID
- support **storing multiple snapshots** of pages over time
- support **custom user puppeteer scripts to run while archiving** (e.g. for expanding reddit threads, scrolling thread on twitter, etc)
- support named collections of archived content with different user access permissions
- support sharing archived assets via DHT + torrent / ipfs / ZeroNet / other sharing system

Smaller planned features

- support pushing pages to multiple 3rd party services using ArchiveNow instead of just archive.org
- body text extraction to markdown (using fathom?)
- featured image / thumbnail extraction
- auto-tagging links based on important/frequent keywords in extracted text (like pocket)
- automatic article summary paragraphs from extracted text with nlp summarization library
- full-text search of extracted text with elasticsearch/elasticlunr/ag
- download closed-caption subtitles from Youtube and other video sites for full-text indexing of video content
- try pulling dead sites from archive.org and other sources if original is down (https://github.com/hartator/wayback-machine-downloader)
- And more in the issues list...

IMPORTANT: *Please don't work on any of these major long-term tasks without contacting me first, work is already in progress for many of these, and I may have to reject your PR if it doesn't align with the existing work!*

1.5.2 Changelog

If you're having an issue with a breaking change, or migrating your data between versions, open an issue to get help.

`ArchiveBox` was previously named `Pocket Archive Stream` and then `Bookmark Archiver`.

See the releases page for versioned source downloads and full changelog. Many thanks to our 30+ contributors and everyone in the web archiving community!

- v0.2.4 released
- better archive corruption guards (check structure invariants on every parse & save)
- remove title prefetching in favor of new FETCH_TITLE archive method
- slightly improved CLI output for parsing and remote url downloading
- re-save index after archiving completes to update titles and urls
- remove redundant derivable data from link json schema
- markdown link parsing support
- faster link parsing and better symbol handling using a new compiled URL_REGEX

- v0.2.3 released
- fixed issues with parsing titles including trailing tags
- fixed issues with titles defaulting to URLs instead of attempting to fetch
- fixed issue where bookmark timestamps from RSS would be ignored and current ts used instead
- fixed issue where ONLY_NEW would overwrite existing links in archive with only new ones
- fixed lots of issues with URL parsing by using `urllib.parse` instead of hand-written lambdas
- ignore robots.txt when using wget (sssshhh don't tell anyone)
- fix RSS parser bailing out when there's whitespace around XML tags
- fix issue with browser history export trying to run ls on wrong directory

---

- v0.2.2 released
- Shaarli RSS export support
- Fix issues with plain text link parsing including quotes, whitespace, and closing tags in URLs
- add USER_AGENT to archive.org submissions so they can track archivebox usage
- remove all icons similar to archive.org branding from archive UI
- hide some of the noisier youtubedl and wget errors
- set permissions on youtubedl media folder
- fix chrome data dir incorrect path and quoting
- better chrome binary finding
- show which parser is used when importing links, show progress when fetching titles

---

- v0.2.1 released with new logo
- ability to import plain lists of links and almost all other raw filetypes
- WARC saving support via wget
- Git repository downloading with git clone
- Media downloading with youtube-dl (video, audio, subtitles, description, playlist, etc)

- v0.2.0 released with new name
- renamed from **Bookmark Archiver** -> **ArchiveBox**

- v0.1.0 released
- support for browser history exporting added with `./bin/archivebox-export-browser-history`
- support for chrome `--dump-dom` to output full page HTML after JS executes

---

- v0.0.3 released
- support for chrome `--user-data-dir` to archive sites that need logins
- fancy individual html & json indexes for each link
- smartly append new links to existing index instead of overwriting

---

- v0.0.2 released
- proper HTML templating instead of format strings (thanks to https://github.com/bardisty!)
- refactored into separate files, wip audio & video archiving

---

- v0.0.1 released
- Index links now work without nginx url rewrites, archive can now be hosted on github pages
- added setup.sh script & docstrings & help commands
- made Chromium the default instead of Google Chrome (yay free software)
- added env-variable configuration (thanks to https://github.com/hannah98!)
- renamed from **Pocket Archive Stream** -> **Bookmark Archiver**
- added Netscape-format export support (thanks to https://github.com/ilvar!)
- added Pinboard-format export support (thanks to https://github.com/sconeyard!)
- front-page of HN, oops! apparently I have users to support now :grin:?
- added Pocket-format export support

- v0.0.0 released: created Pocket Archive Stream 2017/05/05

1.5.3 Donations

Patreon: https://www.patreon.com/theSquashSH

Paypal: https://paypal.me/NicholasSweeting

I develop this project solely in my spare time right now. If you want to help me keep it alive and flourishing, donate to support more development!

If you have any questions or want to partner with this project, contact me at: archivebox-hello@sweeting.me

1.5.4 Web Archiving Community

Just getting started and want to learn more about why Web Archiving is important? Check out this article: On the Importance of Web Archiving.

The internet archiving community is surprisingly far-reaching and almost universally friendly!

Whether you want to learn which organizations are the big players in the web archiving space, want to find a specific open source tool for your web archiving need, or just want to see where archivists hang out online, this is my attempt at an index of the entire web archiving community.

- *The Master ListsCommunity-maintained indexes of web archiving tools and groups by IIPC, COPTR, ArchiveTeam, Wikipedia, & the ASA.*
- *Web Archiving SoftwareOpen source tools and projects in the internet archiving space.*
  - *Bookmarking Services*
  - *Well-Known Open Source Projects*
  - *Public Archiving Services*
  - *ArchiveBox Alternatives*
  - *Smaller Utilities*
- *Reading ListArticles, posts, and blogs relevant to ArchiveBox and web archiving in general.*
  - *Blogs*
  - *Articles*
  - *ArchiveBox Discussions in News & Social Media*
- *CommunitiesA collection of the most active internet archiving communities and initiatives.*
  - *Most Active Web-Archiving Communities*
  - *Other Web Archiving Communities*
  - *General Archiving Foundations, Coalitions, Initiatives, and Institutes*

The Master Lists

Indexes of archiving institutions and software maintained by other people. If there's anything archivists love doing, it's making lists.

- **COPTR Wiki of Web Archiving Tools (COPTR)**
- **Awesome Web Archiving Tools (IIPC)**
- Awesome Web Crawling Tools
- Awesome Web Scraping Tools
- ArchiveTeam's List of Software (ArchiveTeam.org)
- List of Web Archiving Initiatives (Wikipedia.org)
- Directory of Archiving Organizations (American Society of Archivists)

Web Archiving Projects

Bookmarking Services

- **Pocket Premium Bookmarking tool that provides an archiving service in their paid version, run by Mozilla**
- **Pinboard Bookmarking tool that provides archiving in a paid version, run by a single independent developer**
- Instapaper Bookmarking alternative to Pocket/Pinboard (with no archiving)
- Wallabag / Wallabag.it Self-hostable web archiving server that can import via RSS
- Shaarli Self-hostable bookmark tagging, archiving, and sharing service

---

From the Archive.org & Archive-It teams

- **Archive.org The O.G. wayback machine provided publicly by the Internet Archive (Archive.org)**
- **Archive.it commercial Wayback-Machine solution**
- **Heretrix The king of internet archiving crawlers, powers the Wayback Machine**
- **Brozzler chrome headless crawler + WARC archiver maintained by Archive.org**
- OpenWayback Toolkit of major open-source wayback-machine components
- WarcProx warc proxy recording and playback utility
- WarcTools utilities for dealing with WARCs
- Grab-Site An easy preconfigured web crawler designed for backing up websites
- WPull A pure python implementation of wget with WARC saving
- More on their Github...

From the WebRecorder.io team

- **Webrecorder.io An open-source personal archiving server that uses pywb under the hood**
- **pywb The python wayback machine, the codebase forked off archive.org that powers webrecorder**
- warcit Create a warc file out of a folder full of assets
- WebArchivePlayer A tool for replaying web archives
- warcio fast streaming asynchronous WARC reader and writer
- node-warc Parse And Create Web ARChive (WARC) files with node.js
- WAIL Web archiver GUI using Heritrix and OpenWayback
- squidwarc User-scriptable, archival crawler using Chrome
- More on their Github...

From the Old Dominion University: Web Science Team

- **ipwb A distributed web archiving solution using pywb with ipfs for storage**
- **archivenow tool that pushes urls into all the online archive services like Archive.is and Archive.org**
- WAIL Electron app version of the original wail for creating and interacting with web archives
- **warcreate a Chrome extension for creating WARCs from any webpage**
- More on their Github...

From the Archives Unleashed Team

- AUT Archives Unleashed Toolkit for analyzing web archives (formerly WarcBase)
- Warclight A Rails engine for finding and searching web archives
- More on their Github...

From the IIPC team

- **awesome-web-archiving Large list of archiving projects and orgs**
- OpenWayback Toolkit of major open-source wayback-machine components
- JWARC A Java library for reading and writing WARC files.
- More on their Github...

Other Public Archiving Services

- https://archive.is / https://archive.today
- https://archive.st
- http://theoldnet.com
- https://timetravel.mementoweb.org/
- https://freezepage.com/
- https://webcitation.org/archive
- https://archiveofourown.org/
- https://megalodon.jp/
- https://github.com/HelloZeroNet/ZeroNet (super cool project)
- Google, Bing, DuckDuckGo, and other search engine caches

Other ArchiveBox Alternatives

- **Memex by Worldbrain.io a beautiful, user-friendly browser extension that archives all history with full-text search, annotation support, and more**
- **Hypothes.is a web/pdf/ebook annotation tool that also archives content**
- **Reminiscence extremely similar to ArchiveBox, uses a Django backend + UI and provides auto-tagging and summary features with NLTK**
- **Shaarchiver very similar project that archives Firefox, Shaarli, or Delicious bookmarks and all linked media, generating a markdown/HTML index**
- **Polarized a desktop application for bookmarking, annotating, and archiving articles offline**
- **Photon a fast crawler with archiving and asset extraction support**
- ReadableWebProxy A proxying archiver that downloads content from sites and can snapshot multiple versions of sites over time
- Perkeep "Perkeep lets you permanently keep your stuff, for life."
- Fetching.io A personal search engine/archiver that lets you search through all archived websites that you've bookmarked
- Fossilo A commercial archiving solution that appears to be very similar to ArchiveBox
- Archivematica web GUI for institutional long-term archiving of web and other content
- Headless Chrome Crawler distributed web crawler built on puppeteer with screenshots
- WWWofle old proxying recorder software similar to ArchiveBox
- Erised Super simple CLI utility to bookmark and archive webpages
- Zotero collect, organize, cite, and share research (mainly for technical/scientific papers & citations)

Smaller Utilities

Random helpful utilities for web archiving, WARC creation and replay, and more...

- https://github.com/jsvine/waybackpack command-line tool that lets you download the entire Wayback Machine archive for a given URL
- https://github.com/hartator/wayback-machine-downloader Download an entire website from the Internet Archive Wayback Machine.
- https://en.archivarix.com download an archived page or entire site from the Wayback Machine
- https://proofofexistence.com prove that a certain file existed at a given time using the blockchain
- https://github.com/chfoo/warcat for merging, extracting, and verifying WARC files
- https://github.com/mozilla/readability tool for extracting article contents and text
- https://github.com/mholt/timeliner All your digital life on a single timeline, stored locally
- https://github.com/wkhtmltopdf/wkhtmltopdf Webkit HTML to PDF archiver/saver
- Sheetsee-Pocket project that provides a pretty auto-updating index of your Pocket links (without archiving them)
- Pocket -> IFTTT -> Dropbox Post by Christopher Su on his Pocket saving IFTTT recipe
- http://squidman.net/squidman/index.html
- https://wordpress.org/plugins/broken-link-checker/
- https://github.com/ArchiveTeam/wpull
- http://freedup.org/
- https://en.wikipedia.org/wiki/Furl
- *And many more on the other lists...*

Reading List

A collection of blog posts and articles about internet archiving, contact me / open an issue if you want to add a link here!

Blogs

- https://blog.archive.org
- https://netpreserveblog.wordpress.com
- https://blog.webrecorder.io/
- https://ws-dl.blogspot.com
- https://siarchives.si.edu/blog
- https://parameters.ssrc.org
- https://sr.ithaka.org/publications
- https://ait.blog.archive.org

- https://brewster.kahle.org
- https://ianmilligan.ca
- https://medium.com/@giovannidamiola

Articles

- https://parameters.ssrc.org/2018/09/on-the-importance-of-web-archiving/
- https://www.bbc.com/future/story/20190401-why-theres-so-little-left-of-the-early-internet
- https://sr.ithaka.org/publications/the-state-of-digital-preservation-in-2018/
- https://gizmodo.com/delete-never-the-digital-hoarders-who-collect-tumblrs-1832900423
- https://siarchives.si.edu/blog/we-are-not-alone-progress-digital-preservation-community
- https://www.gwern.net/Archiving-URLs
- http://brewster.kahle.org/2015/08/11/locking-the-web-open-a-call-for-a-distributed-web-2/
- https://lwn.net/Articles/766374/
- https://en.wikipedia.org/wiki/List_of_Web_archiving_initiatives
- https://medium.com/@giovannidamiola/making-the-internet-archives-full-text-search-faster-30fb11574ea9
- https://xkcd.com/1909/
- https://samsaffron.com/archive/2012/06/07/testing-3-million-hyperlinks-lessons-learned#comment-31366
- https://www.gwern.net/docs/linkrot/2011-muflax-backup.pdf
- https://thoughtstreams.io/higgins/permalinking-vs-transience/
- http://ait.blog.archive.org/files/2014/04/archiveit_life_cycle_model.pdf
- https://blog.archive.org/2016/05/26/web-archiving-with-national-libraries/
- https://blog.archive.org/2014/10/28/building-libraries-together/
- https://ianmilligan.ca/2018/03/27/ethics-and-the-archived-web-presentation-the-ethics-of-studying-geocities/
- https://ianmilligan.ca/2018/05/22/new-article-if-these-crawls-could-talk-studying-and-documenting-web-archives-provenance/
- https://ws-dl.blogspot.com/2019/02/2019-02-08-google-is-being-shuttered.html

If any of these links are dead, you can find an archived version on https://archive.sweeting.me.

ArchiveBox Discussions in News & Social Media

- **Aggregators:** **ProductHunt**, **AlternativeTo**, SteemHunt, Recurse Center: The Joy of Computing, Github Changelog, Dev.To Ultra List, O'Reilly 4 Short Links, JaxEnter
- **Blog Posts & Podcasts:** Korben.info, Defining Desktop Linux Podcast #296 (0:55:00), Binärgewitter Podcast #221, Schrankmonster.de, La Ferme Du Web
- **Hacker News:** #1, #2, #3, **#4**

- **Reddit r/DataHoarder**:#1, #2, #3, #4, #5 , **#6**
- **Reddit r/SelfHosted:**#1, #2
- **Twitter:**Python Trending, PyCoder's Weekly, Python Hub, Smashing Magazine
- **More on:**Twitter, Reddit, HN, Google...

Communities

Most Active Communities

- **The Internet Archive (Archive.org)** (USA)
- **International Internet Preservation Consortium (IIPC)** (International)
- **The Archive Team, URL Team, r/ArchiveTeam** (International)
- **r/DataHoarder, r/Archivists, r/DHExchange** (International)
- The Eye Non-profit working on content archival and long-term preservation (Europe)
- Digital Preservation Coalition & their Software Tool Registry (COPTR) (UK & Wales)
- Archives Unleashed Project and UAP Github (Canada)
- Old Dominion University: Web Science and Digital Libraries (WS-DL @ ODU) (Virginia, USA)

Web Archiving Communities

- Canadian Web Archiving Coalition (Canada)
- Web Archives for Historical Research Group (Canada)
- Smithsonian Institution Archives: Digital Curation (Washington D.C., USA)
- National Digital Stewardship Alliance (NDSA) (USA)
- Digital Library Federation (DLF) (USA)
- Council on Library and Information Resources (CLIR) (USA)
- Digital Curation Centre (DCC) (UK)
- ArchiveMatica & their Community Wiki (International)
- Professional Development Institutes for Digital Preservation (POWRR) (USA)
- Institute of Museum and Library Services (IMLS) (USA)
- Stanford Libraries Web Archiving (USA)
- Society of American Archivists: Electronic Records (SAA) (USA)
- BitCurator Consortium (BCC) (USA)
- Ethics & Archiving the Web Conference (Rhizome/Webrecorder.io) (USA)
- Archivists Round Table of NYC (USA)

General Archiving Foundations, Coalitions, Initiatives, and Institutes

- Community Archives and Heritage Group (UK & Ireland)
- Open Preservation Foundation (OPF) (UK & Europe)
- Software Preservation Network (International)
- ITHAKA, Portico, JSTOR, ARTSTOR, S+R (USA)
- Archives and Records Association (UK & Ireland)
- Arkivrådet AAS (Sweden)
- Asociación Española de Archiveros, Bibliotecarios, Museologos y Documentalistas (ANABAD) (Spain)
- Associação dos Arquivistas Brasileiros (AAB) (Brazil)
- Associação Portuguesa de Bibliotecários, Archivistas e Documentalistas (BAD) (Portugal)
- Association des archivistes français (AAF) (France)
- Associazione Nazionale Archivistica Italiana (ANAI) (Italy)
- Australian Society of Archivists Inc. (Australia)
- International Council on Archives (ICA)
- International Records Management Trust (IRMT)
- Irish Society for Archives (Ireland)
- Koninklijke Vereniging van Archivarissen in Nederland (Netherlands)
- State Archives Administration of the People's Republic of China (China)
- Academy of Certified Archivists
- Archivists and Librarians in the History of the Health Sciences
- Archivists for Congregations of Women Religious
- Archivists of Religious Institutions
- Association of Catholic Diocesan Archivists
- Association of Moving Image Archivists
- Council of State Archivists
- National Association of Government Archives and Records Administrators
- National Episcopal Historians and Archivists
- Archival Education and Research Institute
- Archives Leadership Institute
- Georgia Archives Institute
- Modern Archives Institute
- Western Archives Institute
- Association des archivistes du Québec
- Association of Canadian Archivists
- Canadian Council of Archives/Conseil canadien des archives
- Archives Association of British Columbia

- Archives Association of Ontario
- Archives Council of Prince Edward Island
- Archives Society of Alberta
- Association for Manitoba Archives
- Association of Newfoundland and Labrador Archives
- Council of Nova Scotia Archives
- Réseau des services d'archives du Québec
- Saskatchewan Council for Archives and Archivists

You can find more organizations and initiatives on these other lists:

- Wikipedia.org List of Web Archiving Initiatives
- SAA List of USA & Canada Based Archiving Organizations
- SAA List of International Archiving Organizations
- Digital Preservation Coalition's Member List

---

^ Back to Top ^

Python Module Index

a

Index

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W