ArchiveBox

Release 0.4.24

Apr 06, 2021

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 is a powerful self-hosted internet archiving solution written in Python 3. You feed it URLs of pages you want to archive, and it saves them to disk in a varitety of formats depending on the configuration and the content it detects. ArchiveBox can be installed via Docker (recommended), `apt`, `brew`, or `pip`. It works on macOS, Windows, and Linux/BSD (both armv7 and amd64).

Once installed, URLs can be added via the command line `archivebox add` or the built-in Web UI `archivebox server`. It can ingest bookmarks from a service like Pocket/Pinboard, your entire browsing history, RSS feeds, or URLs one at a time.

The main index is a self-contained `data/index.sqlite3` file, and each snapshot is stored as a folder `data/archive/<timestamp>/`, with an easy-to-read `index.html` and `index.json` within. For each page, ArchiveBox auto-extracts many types of assets/media and saves them in standard formats, with out-of-the-box support for: 3 types of HTML snapshots (wget, Chrome headless, singlefile), a PDF snapshot, a screenshot, a WARC archive, git repositories, images, audio, video, subtitles, article text, and more. The snapshots are browseable and managable offline through the filesystem, the built-in webserver, or the Python API.

1.1.1 Quickstart

First, get ArchiveBox using your system package manager, Docker, or pip:

```
# You can run it with Docker or Docker Compose (recommended)
docker pull archivebox/archivebox
# https://raw.githubusercontent.com/ArchiveBox/ArchiveBox/master/docker-compose.yml

# or Ubuntu/Debian
sudo add-apt-repository -u ppa:archivebox/archivebox
apt install archivebox

```

(continues on next page)

(continued from previous page)

```
# or macOS
brew install archivebox/archivebox/archivebox

# or for the Python version only, without wget/git/chrome/etc. included
pip3 install archivebox

# If you're using an apt/brew/pip install you can run archivebox commands normally
#   archivebox [subcommand] [...args]
# If you're using Docker you'll have to run the commands like this
#   docker run -v $PWD:/data -it archivebox/archivebox [subcommand] [...args]
# And the equivalent in Docker Compose:
#   docker-compose run archivebox [subcommand] [...args]
```

Check that everything installed correctly with `archivebox --version`

To start using archivebox, you have to create a data folder and `cd` into it:

```
mkdir ~/archivebox && cd ~/archivebox    # you can put the collection dir anywhere
archivebox init
```

Then Add some URLs to your archive collection:

```
archivebox add https://github.com/ArchiveBox/ArchiveBox
archivebox add --depth=1 https://example.com
```

View the snapshots of the URLs you added via the self-hosted web UI:

```
archivebox manage createsuperuser          # create an admin acct
archivebox server 0.0.0.0:8000             # start the web server
open http://127.0.0.1:8000/                # open the interactive admin panel
ls ~/archivebox/archive/*/index.html       # or browse the snapshots on disk
```

DEMO: archivebox.zervice.io/For more information, see the full Quickstart guide, Usage, and Configuration docs.

1.1.2 Overview

ArchiveBox is a command line tool, self-hostable web-archiving server, and Python library all-in-one. It can be installed on Docker, macOS, and Linux/BSD, and Windows. You can download and install it as a Debian/Ubuntu package, Homebrew package, Python3 package, or a Docker image. No matter which install method you choose, they all provide the same CLI, Web UI, and on-disk data format.

To use ArchiveBox you start by creating a folder for your data to live in (it can be anywhere on your system), and running `archivebox init` inside of it. That will create a sqlite3 index and an `ArchiveBox.conf` file. After that, you can continue to add/export/manage/etc using the CLI `archivebox help`, or you can run the Web UI (recommended).

The CLI is considered "stable", the ArchiveBox Python API and REST APIs are in "beta", and the desktop app is in "alpha" stage.

At the end of the day, the goal is to sleep soundly knowing that the part of the internet you care about will be automatically preserved in multiple, durable long-term formats that will be accessible for decades (or longer). You can also self-host your archivebox server on a public domain to provide archive.org-style public access to your site snapshots.

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
- 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)~~ (do not do this until v0.5 is released with some security fixes)
- **Doesn't require a constantly-running daemon**, proxy, or native app
- Provides a CLI, Python API, self-hosted web UI, and REST API (WIP)
- Architected to be able to run **many varieties of scripts during archiving**, e.g. to extract media, summarize articles, scroll pages, close modals, expand comment threads, etc.
- Can also **mirror content to 3rd-party archiving services** automatically for redundancy

Input formats

ArchiveBox supports many input formats for URLs, including Pocket & Pinboard exports, Browser bookmarks, Browser history, plain text, HTML, markdown, and more!

```
echo 'http://example.com' | archivebox add
archivebox add 'https://example.com/some/page'
archivebox add < ~/Downloads/firefox_bookmarks_export.html
archivebox add < any_text_with_urls_in_it.txt
archivebox add --depth=1 'https://example.com/some/downloads.html'
archivebox add --depth=1 'https://news.ycombinator.com#2020-12-12'
```

- 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.

It also includes a built-in scheduled import feature and browser bookmarklet, so you can ingest URLs from RSS feeds, websites, or the filesystem regularly.

Output formats

All of ArchiveBox's state (including the index, snapshot data, and config file) is stored in a single folder called the "ArchiveBox data folder". All `archivebox` CLI commands must be run from inside this folder, and you first create it by running `archivebox init`.

The on-disk layout is optimized to be easy to browse by hand and durable long-term. The main index is a standard sqlite3 database (it can also be exported as static JSON/HTML), and the archive snapshots are organized by date-added timestamp in the `archive/` subfolder. Each snapshot subfolder includes a static JSON and HTML index describing its contents, and the snapshot extrator outputs are plain files within the folder (e.g. `media/example.mp4`, `git/somerepo.git`, `static/someimage.png`, etc.)

```
ls ./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.

Dependencies

You don't need to install all the dependencies, ArchiveBox will automatically enable the relevant modules based on whatever you have available, but it's recommended to use the official Docker image with everything preinstalled.

If you so choose, you can also install ArchiveBox and its dependencies directly on any Linux or macOS systems using the automated setup script or the system package manager.

ArchiveBox is written in Python 3 so it requires `python3` and `pip3` available on your system. It also uses a set of optional, but highly recommended external dependencies for archiving sites: `wget` (for plain HTML, static files, and WARC saving), `chromium` (for screenshots, PDFs, JS execution, and more), `youtube-dl` (for audio and video), `git` (for cloning git repos), and `nodejs` (for readability and singlefile), and more.

Caveats

If you're importing URLs containing secret slugs or pages with private content (e.g Google Docs, CodiMD notepads, etc), you may want to disable some of the extractor modules to avoid leaking private URLs to 3rd party APIs during the archiving process.

```
# don't do this:
archivebox add 'https://docs.google.com/document/d/12345somelongsecrethere'
archivebox add 'https://example.com/any/url/you/want/to/keep/secret/'

# without first disabling share the URL with 3rd party APIs:
archivebox config --set SAVE_ARCHIVE_DOT_ORG=False   # disable saving all URLs in␣
↪Archive.org
archivebox config --set SAVE_FAVICON=False  # optional: only the domain is leaked,␣
↪not full URL
archivebox config --get CHROME_VERSION      # optional: set this to chromium instead␣
↪of chrome if you don't like Google
```

Be aware that malicious archived JS can also read the contents of other pages in your archive due to snapshot CSRF and XSS protections being imperfect. See the Security Overview page for more details.

```
# visiting an archived page with malicious JS:
https://127.0.0.1:8000/archive/1602401954/example.com/index.html

# example.com/index.js can now make a request to read everything:
https://127.0.0.1:8000/index.html
https://127.0.0.1:8000/archive/*
# then example.com/index.js can send it off to some evil server
```

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). For now ArchiveBox is designed to only archive each URL with each extractor type once. A workaround to take multiple snapshots of the same URL is to make them slightly different by adding a hash:

```
archivebox add 'https://example.com#2020-10-24'
...
archivebox add 'https://example.com#2020-10-25'
```

1.1.3 Setup

Docker Compose

This is the recommended way of running ArchiveBox.

It comes with everything working out of the box, including all extractors, a headless browser runtime, a full webserver, and CLI interface.

```
# docker-compose run archivebox <command> [args]

mkdir archivebox && cd archivebox
wget 'https://raw.githubusercontent.com/ArchiveBox/ArchiveBox/master/docker-compose.
↪yml'
docker-compose run archivebox init
docker-compose run archivebox add 'https://example.com'
docker-compose run archivebox manage createsuperuser
docker-compose up
open http://127.0.0.1:8000
```

Docker

```
# docker run -v $PWD:/data -it archivebox/archivebox <command> [args]

mkdir archivebox && cd archivebox
docker run -v $PWD:/data -it archivebox/archivebox init
docker run -v $PWD:/data -it archivebox/archivebox add 'https://example.com'
docker run -v $PWD:/data -it archivebox/archivebox manage createsuperuser

# run the webserver to access the web UI
docker run -v $PWD:/data -it -p 8000:8000 archivebox/archivebox server 0.0.0.0:8000
open http://127.0.0.1:8000
```

(continues on next page)

(continued from previous page)

```
# or export a static version of the index if you dont want to run a server
docker run -v $PWD:/data -it archivebox/archivebox list --html --with-headers > index.
↪html
docker run -v $PWD:/data -it archivebox/archivebox list --json --with-headers > index.
↪json
open ./index.html
```

Bare Metal

```
# archivebox <command> [args]

# on Debian/Ubuntu
sudo add-apt-repository -u ppa:archivebox/archivebox
apt install archivebox

# on macOS
brew install archivebox/archivebox/archivebox
```

Initialize your archive in a directory somewhere and add some links:

```
mkdir ~/archivebox && cd archivebox
npm install --prefix . 'git+https://github.com/ArchiveBox/ArchiveBox.git'
archivebox init
archivebox add 'https://example.com'  # add URLs as args pipe them in via stdin
archivebox add --depth=1 https://example.com/table-of-contents.html
# it can injest links from many formats, including RSS/JSON/XML/MD/TXT and more
curl https://getpocket.com/users/USERNAME/feed/all | archivebox add
```

Start the webserver to access the web UI:

```
archivebox manage createsuperuser
archivebox server 0.0.0.0:8000

open http://127.0.0.1:8000
```

Or export a static HTML version of the index if you don't want to run a webserver:

```
archivebox list --html --with-headers > index.html
archivebox list --json --with-headers > index.json
open ./index.html
```

To view more information about your dependencies, data, or the CLI:

```
archivebox version
archivebox status
archivebox help
```

1.1.4 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.

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.

All the archived links are stored by date bookmarked in `./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.

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 interface for users to ingest bulk feeds of URLs over extended periods, as opposed to being a backend service that ingests individual, manually-submitted URLs from a web UI. However, we also have the option to add urls via a web interface through our Django frontend.

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~~ (do not do this until v0.5 is released with some security fixes). 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

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, 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 @ArchiveBoxApp or @theSquashSH on Twitter.

1.1.5 Documentation

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

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

Getting Started

- Quickstart
- Install
- Docker

Reference

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

- Chromium Install
- Security Overview
- Troubleshooting
- Python API
- REST API (coming soon...)

More Info

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

1.1.6 ArchiveBox Development

All contributions to ArchiveBox are welcomed! Check our issues and Roadmap for things to work on, and please open an issue to discuss your proposed implementation before working on things! Otherwise we may have to close your PR if it doesn't align with our roadmap.

Setup the dev environment

First, install the system dependencies from the "Bare Metal" section above. Then you can clone the ArchiveBox repo and install

```
git clone https://github.com/ArchiveBox/ArchiveBox
cd ArchiveBox
git checkout master  # or the branch you want to test
git pull

# Install ArchiveBox + python dependencies
python3 -m venv .venv && source .venv/bin/activate && pip install -e .[dev]
# or
pipenv install --dev && pipenv shell

# Install node dependencies
npm install

# Optional: install the extractor dependencies
./bin/setup.sh

# Optional: develop via docker by mounting the code dir into the container
# if you edit e.g. ./archivebox/core/models.py on the docker host, runserver
# inside the container will reload and pick up your changes
docker build . -t archivebox
```

(continues on next page)

(continued from previous page)

```
docker run -it -p 8000:8000 \
    -v $PWD/data:/data \
    -v $PWD/archivebox:/app/archivebox \
    archivebox server 0.0.0.0:8000 --debug --reload
```

Common development tasks

See the `./bin/` folder and read the source of the bash scripts within. You can also run all these in Docker. For more examples see the Github Actions CI/CD tests that are run: `.github/workflows/*.yaml`.

Run the linters

```
./bin/lint.sh
```

(uses `flake8` and `mypy`)

Run the integration tests

```
./bin/test.sh
```

(uses `pytest -s`)

Build the docs, pip package, and docker image

```
./bin/build.sh

# or individually:
./bin/build_docs.sh
./bin/build_pip.sh
./bin/build_docker.sh
```

Roll a release

```
./bin/release.sh
```

(bumps the version, builds, and pushes a release to PyPI, Docker Hub, and Github Packages)

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 (using Docker in WSL2).

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:`archivebox add < [links_file]`

1. Set up ArchiveBox

We recommend using Docker because it has all the extractors and dependencies working out-of-the-box:

```
# first make sure you have docker: https://docs.docker.com/get-docker/

# then run this to get started with a collection in the current directory
docker run -v $PWD:/data -it archivebox/archivebox init

# alternatively, install ArchiveBox and its dependencies directly on your system␣
↪without docker
# (script prompts for user confirmation before installing anything)
curl https://raw.githubusercontent.com/ArchiveBox/ArchiveBox/master/bin/setup.sh | sh
# or follow the manual setup instructions if you don't like using curl | sh
```

(The above are shell commands to run. If you're not used to those, consult your operating system's manual for how to run a terminal emulator.)

For more details, 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/export-browser-history.sh --chrome`
- Firefox History: `./bin/export-browser-history.sh --firefox`
- Safari History: `./bin/export-browser-history.sh --safari`

- 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 directly, import a list of links from a file, or import from a feed URL. All via stdin:

```
# if using docker
docker run -v $PWD:/data -it archivebox/archivebox add 'https://example.com'

# or if not using docker
archivebox add 'https://example.com'

# any text containing links can also be passed in via stdin (works with docker as␣
↪well)
curl https://getpocket.com/users/example/feed/all | archivebox add
```

Done!

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

You can also use the interactive Web UI to view/manage/add links to your archive:

```
# with docker:
docker run -v $PWD:/data -it -p 8000:8000 archivebox/archivebox
# or without docker:
archivebox server

open http://127.0.0.1:8000
```

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.

You will also need 500MB of RAM (bare minimum), though 2GB or greater recommended. You may be able to reduce the RAM requirements if you disable all the chrome-based archiving methods with `USE_CHROME=False`.

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 >= 3.7`
- `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.

```
# docker or the manual setup are preferred on all platforms now, if you want to use␣
↪the old install script you can run:
curl https://raw.githubusercontent.com/pirate/ArchiveBox/master/bin/setup.sh | sh
```

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 tap homebrew-ffmpeg/ffmpeg
brew install homebrew-ffmpeg/ffmpeg/ffmpeg --with-fdk-aac
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
```

Install ArchiveBox using pip

```
python3 -m pip install --upgrade archivebox
```

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."

archivebox version
```

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 archivebox

```
# create a new folder to hold your data and cd into it
mkdir data && cd data
archivebox init
archivebox version
archivebox add < ~/Downloads/bookmarks_export.html
```

You can also use the `update` subcommand to resume the archive update at a specific timestamp `archivebox update --resume=153242424324.123`.

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.

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

Usage:

```
docker run -v $PWD:/data archivebox/archivebox init
docker run -v $PWD:/data archivebox/archivebox add 'https://example.com'
docker run -v $PWD:/data -it archivebox/archivebox manage createsuperuser
docker run -v $PWD:/data -p 8000:8000 archivebox/archivebox server 0.0.0.0:8000
```

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

```
mkdir archivebox && cd archivebox
wget https://raw.githubusercontent.com/ArchiveBox/ArchiveBox/master/docker-compose.yml
docker-compose up -d
docker-compose run archivebox init
docker-compose run archivebox manage createsuperuser
docker-compose run archivebox add 'https://example.com'
```

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`.

Then open `http://127.0.0.1:8000` or `data/index.html` to view the archive (HTTP, not HTTPS).

To add new URLs, you can use docker-compose just like the normal `archivebox <subcommand> [args]` CLI.

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

```
echo "https://example.com" | docker-compose run archivebox add
```

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 run archivebox add /data/sources/bookmarks.html
docker-compose run archivebox add < 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 run archivebox add --depth=1 https://example.com/some/feed.rss
```

The `depth` argument controls if you want to save the links contained in that URL, or only the specified URL.

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 Django development server running inside docker on `http://127.0.0.1:8000`.

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 -it -v $PWD:/data archivebox/archivebox add
```

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 -it -v $PWD:/data archivebox/archivebox add
# or
docker run -it -v $PWD:/data archivebox/archivebox add < bookmarks.html
```

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

```
docker run -it -v $PWD:/data archivebox/archivebox add 'https://example.com/some/rss/
↪feed.xml'
```

The `depth` argument controls if you want to save the links contained in that URL, or only the specified URL.

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

(not recommended unless you know what you're doing)

```
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

The easiest way is to use the a `.env` file or add your config to your `docker-compose.yml environment:` section.

The next easiest way to get/set config is using the archivebox CLI:

```
docker-compose run archivebox config --get RESOLUTION
docker-compose run archivebox config --set RESOLUTION=1440,900
# or
docker run -it -v $PWD:/data archivebox/archivebox config --set MEDIA_TIMEOUT=120
```

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

To set environment variables for a single run, you can use the `env KEY=VAL ...` command, `-e KEY=VAL`, or `--env-file=somefile.env`.

```
echo 'https://example.com' | docker run -it -v $PWD:/data -e FETCH_SCREENSHOT=False␣
↪archivebox/archivebox add
```

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

You can also edit the `data/ArchiveBox.conf` file directly and the changes will take effect on the next run.

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

CLI Usage

All three of these ways of running ArchiveBox are equivalent and interchangeable:

- `archivebox [subcommand] [...args]` *Using the PyPI package via* `pip install archivebox`
- `archivebox run -it -v $PWD:/data archivebox/archivebox [subcommand] [...args]` *Using the official Docker image*
- `docker-compose run archivebox [subcommand] [...args]` *Using the official Docker image w/ Docker Compose*

You can share a single archivebox data directory between Docker and non-Docker instances as well, allowing you to run the server in a container but still execute CLI commands on the host for example.

For more 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.

```
# via the CLI
archivebox config --set TIMEOUT=3600

# by modifying the config file
nano ArchiveBox.conf
# TIMEOUT=3600

# or by using environment variables
env TIMEOUT=3600 archivebox add 'https://example.com'
```

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

```
archivebox add 'https://example.com'
# or
echo 'https://example.com' | archivebox add
```

You can also add `--depth=1` to any of these commands if you want to recursively archive the URLs and all URLs one hop away. (e.g. all the outlinks on a page + the page).

Import a list of URLs from a txt file

```
cat urls_to_archive.txt | archivebox add
# or
archivebox add < urls_to_archive.txt
# or
curl https://getpocket.com/users/USERNAME/feed/all | archivebox add
```

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

```
archivebox add < ~/Downloads/browser_bookmarks_export.html
# or
archivebox add < ~/Downloads/pinboard_bookmarks.json
# or
archivebox add < ~/Downloads/other_links.txt
```

Import list of links from browser history

Look in the `bin/` folder of this repo to find a script to parse your browser's SQLite history database for URLs. Specify the type of the browser as the first argument, and optionally the path to the SQLite history file as the second argument.

```
./bin/export-browser-history --chrome
archivebox add < output/sources/chrome_history.json
# or
./bin/export-browser-history --firefox
archivebox add < output/sources/firefox_history.json
# or
./bin/export-browser-history --safari
archivebox add < output/sources/safari_history.json
```

UI Usage

```
archivebox server
open http://127.0.0.1:8000
```

Or if you prefer to use the static HTML UI instead of the interactive UI provided by the server, you can open `./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_DIR` folder (usually whatever folder you run `archivebox` in), contains the UI HTML and archived data with the structure outlined below.

```
- output/
  - index.sqlite3        # Main index of all archived URLs
  - index.json           # Redundant JSON version of the same main index
  - index.html           # Redundant static HTML version of the same main index

  - 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
     - 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 manually splitting your URLs into separate chunks:

```
archivebox add < urls_chunk_1.txt &
archivebox add < urls_chunk_2.txt &
archivebox add < urls_chunk_3.txt &
```

(though this may not be faster if you have a very large collection/main index)

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 Shell Usage

Explore the Python API a bit to see whats available using the archivebox shell:

```
$ archivebox shell
[i] [2020-09-17 16:57:07] ArchiveBox v0.4.21: archivebox shell
    > /Users/squash/Documents/opt/ArchiveBox/data

# Shell Plus Model Imports
from core.models import Snapshot
from django.contrib.admin.models import LogEntry
from django.contrib.auth.models import Group, Permission, User
from django.contrib.contenttypes.models import ContentType
from django.contrib.sessions.models import Session
# Shell Plus Django Imports
from django.core.cache import cache
from django.conf import settings
from django.contrib.auth import get_user_model
from django.db import transaction
from django.db.models import Avg, Case, Count, F, Max, Min, Prefetch, Q, Sum, When
from django.utils import timezone
from django.urls import reverse
from django.db.models import Exists, OuterRef, Subquery
# ArchiveBox Imports
from archivebox.core.models import Snapshot, User
from archivebox import *
    help
    version
    init
    config
    add
    remove
    update
    list
    shell
    server
    status
    manage
    oneshot
    schedule

[i] Welcome to the ArchiveBox Shell!
    https://github.com/ArchiveBox/ArchiveBox/wiki/Usage#Shell-Usage

    Hint: Example use:
        print(Snapshot.objects.filter(is_archived=True).count())
        Snapshot.objects.get(url="https://example.com").as_json()
        add("https://example.com/some/new/url")
```

Python API Usage

```
import os
DATA_DIR = '~/some/path/containing/your/archivebox/data'
os.chdir(DATA_DIR)


from archivebox.main import check_data_folder, setup_django, add, remove, server

check_data_folder(DATA_DIR)
setup_django(DATA_DIR)
```

(continues on next page)

(continued from previous page)

```
add('https://example.com', index_only=True, out_dir=DATA_DIR)
remove(...)
server(...)
...
```

For more information see the Python API Reference.

1.3.2 Configuration

The full ArchiveBox config file definition with defaults can be found here: archivebox/config.py.

Configuration of ArchiveBox is done by using the `archivebox config` command, modifying the `ArchiveBox.conf` file in the data folder, or by using environment variables. All three methods work equivalently when using Docker as well.

Some equivalent examples of setting some configuration options:

```
archivebox config --set CHROME_BINARY=google-chrome-stable
# OR
echo "CHROME_BINARY=google-chrome-stable" >> ArchiveBox.conf
# OR
env CHROME_BINARY=google-chrome-stable archivebox add ~/Downloads/bookmarks_export.
↪html
```

Environment variables take precedence over the config file, which is useful if you only want to use a certain option temporarily during a single run.

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/__init__.py`.

General Settings

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

OUTPUT_DIR

Possible Values: [`.`]/`~/archivebox`/...Path to an output folder to store the archive in.

Defaults to the current folder you're in `./` (`$PWD`) when you run the `archivebox` command.

Note: make sure the user running ArchiveBox has permissions set to allow writing to this folder!

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: [`True`]/`False`Toggle whether or not to attempt rechecking old links when adding new ones, or leave old incomplete links alone and only archive the new links.

By default, ArchiveBox will only archive new links on each import. If you want it to go back through all links in the index and download any missing files on every run, set this to `False`.

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 *archive extractor outputs, it does not re-archive pages that have already been successfully archived.*

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 `SAVE_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:`SAVE_MEDIA`

TEMPLATES_DIR

Possible Values: [$REPO_DIR/archivebox/templates]/path/to/custom/templates/...Path to a directory containing custom index html templates for theming your archive output. Files found in the folder at the specified path can override any of the defaults in the archivebox/themes directory. If you've used django before, this works exactly the same way that django template overrides work (because it uses django under the hood).

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:SAVE_MEDIA, SAVE_GIT, GIT_DOMAINS

Archive Method Toggles

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

SAVE_TITLE

Possible Values: [True]/False By 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*

SAVE_FAVICON

Possible Values: [True]/False Fetch 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*

SAVE_WGET

Possible Values: [True]/False Fetch 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:save_wget(...) function.

Related options: *TIMEOUT*, *SAVE_WGET_REQUISITES*, *CHECK_SSL_VALIDITY*, *COOKIES_FILE*, *WGET_USER_AGENT*, *SAVE_WARC*, *WGET_BINARY*

SAVE_WARC

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

Related options: *TIMEOUT*, *SAVE_WGET_REQUISITES*, *CHECK_SSL_VALIDITY*, *COOKIES_FILE*, *WGET_USER_AGENT*, *SAVE_WGET*, *WGET_BINARY*

SAVE_PDF

Possible Values: [True]/False Print page as PDF.

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

SAVE_SCREENSHOT

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

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

SAVE_DOM

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

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

SAVE_SINGLEFILE

Possible Values: [True]/FalseFetch an HTML file with all assets embedded using Single File.

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

SAVE_READABILITY

Possible Values: [True]/FalseExtract article text, summary, and byline using Mozilla's Readability library. Unlike the other methods, this does not download any additional files, so it's practically free from a disk usage perspective. It works by using any existing downloaded HTML version (e.g. wget, DOM dump, singlefile) and piping it into readability.

Related options:TIMEOUT, *SAVE_WGET*, *SAVE_DOM*, *SAVE_SINGLEFILE*

SAVE_GIT

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

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

SAVE_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`]/`False`Submit 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`]/`False`Whether 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.

SAVE_WGET_REQUISITES

Possible Values: [`True`]/`False`Fetch images/css/js with wget. (True is highly recommended, otherwise your won't download many critical assets to render the page, like images, js, css, etc.)

Related options: *TIMEOUT*, *SAVE_WGET*, *SAVE_WARC*, *WGET_BINARY*

RESOLUTION

Possible Values: [`1440,2000`]/`1024,768`/...Screenshot resolution in pixels width,height.

Related options: *SAVE_SCREENSHOT*

CURL_USER_AGENT

Possible Values: [`Curl/1.19.1`]/`"Mozilla/5.0 ..."`/...This is the user agent to use during curl 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: *USE_CURL*, *SAVE_TITLE*, *CHECK_SSL_VALIDITY*, *CURL_BINARY*, *WGET_USER_AGENT*, *CHROME_USER_AGENT*

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:SAVE_WGET, *SAVE_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:SAVE_PDF, *SAVE_SCREENSHOT*, *SAVE_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:SAVE_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:SAVE_WGET, *SAVE_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: *SAVE_PDF*, *SAVE_SCREENSHOT*, *SAVE_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: *SAVE_PDF*, *SAVE_SCREENSHOT*, *SAVE_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: *SAVE_PDF*, *SAVE_SCREENSHOT*, *SAVE_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: *SAVE_PDF*, *SAVE_SCREENSHOT*, *SAVE_DOM*, *SAVE_SINGLEFILE*, *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: *SAVE_WGET*, *SAVE_WARC*

YOUTUBEDL_BINARY

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

Related options:SAVE_MEDIA

GIT_BINARY

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

Related options:SAVE_GIT

CURL_BINARY

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

Related options:SAVE_FAVICON, SUBMIT_ARCHIVE_DOT_ORG

SINGLEFILE_BINARY

Possible Values: [single-file]/usr/local/bin/single-file/...Path or name of the SingleFile binary to use.

This can be installed using `npm install -g git+https://github.com/gildas-lormeau/SingleFile.git`.

Related options:SAVE_SINGLEFILE, CHROME_BINARY, CHROME_USER_DATA_DIR, CHROME_HEADLESS, CHROME_SANDBOX

READABILITY_BINARY

Possible Values: [readability-extractor]/usr/local/bin/readability-extractor/...Path or name of the Readability extrator binary to use.

This can be installed using `npm install -g git+https://github.com/pirate/readability-extractor.git`.

Related options:SAVE_READABILITY

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.

```
- ------------------------------------------------------------------------
```

1.3.5 IMPORTANT: Don't use ArchiveBox for private archived content right now as we're in the middle of resolving some security issues with how JS is executed in archived content.

```
- ------------------------------------------------------------------------
```

~~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 archivebox add ...
```

~~If you absolutely must run it as root for some reason, a footgun is provided: you can set `ALLOW_ROOT=True` via environment variable or in your ArchiveBox.conf file.~~ It was removed.

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.

1.3.6 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.

1.3.7 Scheduled Archiving

Using Cron

To schedule regular archiving you can use any task scheduler like `cron`, `at`, `systemd`, 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
archivebox add < ./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 every 12 hours:

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
curl https://getpocket.com/users/yourusernamegoeshere/feed/all | archivebox add  >> /
↪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
```

1.3.8 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_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_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

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: Union[pathlib.Path, str, None] = 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.**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.**status**(*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.**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.**oneshot**(*args: Optional[List[str]] = None*, *stdin: Optional[IO] = None*, *pwd: Optional[str] = None*) → None
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.

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

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[[archivebox.config_stubs.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[archivebox.config_stubs.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]]]]

parse bool, int, and str key=value pairs from env

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*) → archivebox.config_stubs.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[archivebox.config_stubs.ConfigDict] = None, out_dir: Optional[str] = None, env_vars: Optional[os._Environ] = None, config_file_vars: Optional[Dict[str, str]] = None*) → archivebox.config_stubs.ConfigDict

archivebox.config.**stdout**(**args, color: Optional[str] = None, prefix: str = '', config: Optional[archivebox.config_stubs.ConfigDict] = None*) → None

archivebox.config.**stderr**(**args, color: Optional[str] = None, prefix: str = '', config: Optional[archivebox.config_stubs.ConfigDict] = None*) → None

archivebox.config.**hint**(*text: Union[Tuple[str, ...], List[str], str], prefix=' ', config: Optional[archivebox.config_stubs.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: archivebox.config_stubs.ConfigDict*) → Dict[str, Union[str, bool, int, None, Pattern[AnyStr], Dict[str, Any]]]

archivebox.config.**get_external_locations**(*config: archivebox.config_stubs.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: archivebox.config_stubs.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: archivebox.config_stubs.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: archivebox.config_stubs.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: archivebox.config_stubs.ConfigDict = {'ACTIVE_THEME': 'default', 'ALLOWED_HOSTS': '*', 'ANSI': {'black': '', 'blue': '', 'green': '', 'lightblue': '', 'lightred': '', 'lightyellow': '', 'red': '', 'reset': '', 'white': ''}, 'ARCHIVEBOX_BINARY': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/envs/v0.4.24/li packages/sphinx/__main__.py', 'ARCHIVE_DIR': PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkout 'BIND_ADDR': '127.0.0.1:8000', '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': {'PACKAGE_DIR': {'enabled': True, 'is_valid': True, 'path': PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkout 'TEMPLATES_DIR': {'enabled': True, 'is_valid': True, 'path': PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkout 'CONFIG_FILE': PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkout 'COOKIES_FILE': None, 'CURL_ARGS': ['–silent', '–location', '–compressed'], 'CURL_BINARY': 'curl', 'CURL_USER_AGENT': 'ArchiveBox/0.4.24 (+https://github.com/ArchiveBox/ArchiveBox/) curl/curl 7.58.0 (x86_64-pc-linux-gnu)', 'CURL_VERSION': 'curl 7.58.0 (x86_64-pc-linux-gnu)', 'DATA_LOCATIONS': {'ARCHIVE_DIR': {'enabled': True, 'is_valid': False, 'path': PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkout 'CONFIG_FILE': {'enabled': True, 'is_valid': False, 'path': PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkout 'LOGS_DIR': {'enabled': True, 'is_valid': False, 'path': PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkout 'OUTPUT_DIR': {'enabled': True, 'is_valid': False, 'path': PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkout 'SOURCES_DIR': {'enabled': True, 'is_valid': False, 'path': PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkout 'SQL_INDEX': {'enabled': True, 'is_valid': False, 'path': PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkout*

archivebox.config.**check_dependencies**(*config: archivebox.config_stubs.ConfigDict = {'ACTIVE_THEME': 'default', 'ALLOWED_HOSTS': '*', 'ANSI': {'black': '', 'blue': '', 'green': '', 'lightblue': '', 'lightred': '', 'lightyellow': '', 'red': '', 'reset': '', 'white': ''}, 'ARCHIVEBOX_BINARY': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/envs/v0.4.24/lib/ packages/sphinx/__main__.py', 'ARCHIVE_DIR': PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/ 'BIND_ADDR': '127.0.0.1:8000', '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': {'PACKAGE_DIR': {'enabled': True, 'is_valid': True, 'path': PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/ 'TEMPLATES_DIR': {'enabled': True, 'is_valid': True, 'path': PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/ 'CONFIG_FILE': PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/ 'COOKIES_FILE': None, 'CURL_ARGS': ['–silent', '–location', '–compressed'], 'CURL_BINARY': 'curl', 'CURL_USER_AGENT': 'ArchiveBox/0.4.24 (+https://github.com/ArchiveBox/ArchiveBox/) curl/curl 7.58.0 (x86_64-pc-linux-gnu)', 'CURL_VERSION': 'curl 7.58.0 (x86_64-pc-linux-gnu)', 'DATA_LOCATIONS': {'ARCHIVE_DIR': {'enabled': True, 'is_valid': False, 'path': PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/ 'CONFIG_FILE': {'enabled': True, 'is_valid': False, 'path': PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/ 'LOGS_DIR': {'enabled': True, 'is_valid': False, 'path': PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/ 'OUTPUT_DIR': {'enabled': True, 'is_valid': False, 'path': PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/ 'SOURCES_DIR': {'enabled': True, 'is_valid': False, 'path': PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/ 'SQL_INDEX': {'enabled': True, 'is_valid': False, 'path': PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/ 'DEBUG': False, 'DEPENDENCIES': {'ARCHIVE-*

archivebox.config.**check_data_folder**(*out_dir: Optional[str] = None, config: archivebox.config_stubs.ConfigDict = {'ACTIVE_THEME': 'default', 'ALLOWED_HOSTS': '*', 'ANSI': {'black': '', 'blue': '', 'green': '', 'lightblue': '', 'lightred': '', 'lightyellow': '', 'red': '', 'reset': '', 'white': ''}, 'ARCHIVEBOX_BINARY': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/envs/v0.4.24/lib/p packages/sphinx/__main__.py', 'ARCHIVE_DIR': PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v 'BIND_ADDR': '127.0.0.1:8000', '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': {'PACKAGE_DIR': {'enabled': True, 'is_valid': True, 'path': PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v 'TEMPLATES_DIR': {'enabled': True, 'is_valid': True, 'path': PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v 'CONFIG_FILE': PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v 'COOKIES_FILE': None, 'CURL_ARGS': ['–silent', '–location', '–compressed'], 'CURL_BINARY': 'curl', 'CURL_USER_AGENT': 'ArchiveBox/0.4.24 (+https://github.com/ArchiveBox/ArchiveBox/) curl/curl 7.58.0 (x86_64-pc-linux-gnu)', 'CURL_VERSION': 'curl 7.58.0 (x86_64-pc-linux-gnu)', 'DATA_LOCATIONS': {'ARCHIVE_DIR': {'enabled': True, 'is_valid': False, 'path': PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v 'CONFIG_FILE': {'enabled': True, 'is_valid': False, 'path': PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v 'LOGS_DIR': {'enabled': True, 'is_valid': False, 'path': PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v 'OUTPUT_DIR': {'enabled': True, 'is_valid': False, 'path': PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v 'SOURCES_DIR': {'enabled': True, 'is_valid': False, 'path': PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v 'SQL_INDEX': {'enabled': True, 'is_valid': False, 'path': PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v 'DEBUG': False, 'DEPENDENCIES': {'ARCHIVEBOX_BINARY': {'enabled': True, 'hash':*

archivebox.config.**setup_django**(*out_dir: pathlib.Path = None, check_db=False, config: archivebox.config_stubs.ConfigDict = {'ACTIVE_THEME': 'default', 'ALLOWED_HOSTS': '*', 'ANSI': {'black': '', 'blue': '', 'green': '', 'lightblue': '', 'lightred': '', 'lightyellow': '', 'red': '', 'reset': '', 'white': ''}, 'ARCHIVEBOX_BINARY': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/envs/v0.4.24/lib/python3.*
packages/sphinx/__main__.py', 'ARCHIVE_DIR': PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.24/a
'BIND_ADDR': '127.0.0.1:8000', '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': {'PACKAGE_DIR': {'enabled': True, 'is_valid': True, 'path': PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.24/a
'TEMPLATES_DIR': {'enabled': True, 'is_valid': True, 'path': PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.24/a
'CONFIG_FILE': PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archive
'COOKIES_FILE': None, 'CURL_ARGS': ['–silent', '–location', '–compressed'], 'CURL_BINARY': 'curl', 'CURL_USER_AGENT': 'ArchiveBox/0.4.24 (+https://github.com/ArchiveBox/ArchiveBox/) curl/curl 7.58.0 (x86_64-pc-linux-gnu)', 'CURL_VERSION': 'curl 7.58.0 (x86_64-pc-linux-gnu)', 'DATA_LOCATIONS': {'ARCHIVE_DIR': {'enabled': True, 'is_valid': False, 'path': PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.24/a
'CONFIG_FILE': {'enabled': True, 'is_valid': False, 'path': PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.24/a
'LOGS_DIR': {'enabled': True, 'is_valid': False, 'path': PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.24/a
'OUTPUT_DIR': {'enabled': True, 'is_valid': False, 'path': PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.24/a
'SOURCES_DIR': {'enabled': True, 'is_valid': False, 'path': PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.24/a
'SQL_INDEX': {'enabled': True, 'is_valid': False, 'path': PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.24/a
'DEBUG': False, 'DEPENDENCIES': {'ARCHIVEBOX_BINARY': {'enabled': True, 'hash': 'md5:dcfa2704a5f1dd5098ddc9081e29d235', 'is_valid': True, 'path': '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/envs/v0.4.24/lib/p
packages/sphinx/__main__.py', 'version': '0.4.24'}, 'CHROME_BINARY': {'enabled': False, 'hash': None, 'is_valid': False, 'path': None, 'version': None}, 'CURL_BINARY': {'enabled': True, 'hash':

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

Module contents

Submodules

archivebox.core.admin module

archivebox.core.admin.**update_snapshots**(*modeladmin*, *request*, *queryset*)

archivebox.core.admin.**update_titles**(*modeladmin*, *request*, *queryset*)

archivebox.core.admin.**overwrite_snapshots**(*modeladmin*, *request*, *queryset*)

archivebox.core.admin.**verify_snapshots**(*modeladmin*, *request*, *queryset*)

archivebox.core.admin.**delete_snapshots**(*modeladmin*, *request*, *queryset*)

class archivebox.core.admin.**SnapshotAdminForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *instance=None*, *use_required_attribute=None*, *renderer=None*)

Bases: django.forms.models.ModelForm

class Meta

Bases: object

model

alias of core.models.Snapshot

fields = '__all__'

save(*commit=True*)
Save this form's self.instance object if commit=True. Otherwise, add a save_m2m() method to the form which can be called after the instance is saved manually at a later time. Return the model instance.

base_fields = {'tags': <core.forms.TagField object>, 'timestamp': <django.forms.fiel

declared_fields = {'tags': <core.forms.TagField object>}

media

class archivebox.core.admin.**SnapshotAdmin**(*model*, *admin_site*)
Bases: django.contrib.admin.options.ModelAdmin

list_display = ('added', 'title_str', 'url_str', 'files', 'size')

sort_fields = ('title_str', 'url_str', 'added')

readonly_fields = ('id', 'url', 'timestamp', 'num_outputs', 'is_archived', 'url_hash',

search_fields = ['url', 'timestamp', 'title', 'tags__name']

fields = ('id', 'url', 'timestamp', 'num_outputs', 'is_archived', 'url_hash', 'added',

list_filter = ('added', 'updated', 'tags')

ordering = ['-added']

actions = [<function delete_snapshots>, <function overwrite_snapshots>, <function upda

actions_template = 'admin/actions_as_select.html'

form
alias of *SnapshotAdminForm*

get_queryset(*request*)
Return a QuerySet of all model instances that can be edited by the admin site. This is used by changelist_view.

tag_list(*obj*)

id_str(*obj*)

title_str(*obj*)

files(*obj*)

size(*obj*)

url_str(*obj*)

media

class archivebox.core.admin.**TagAdmin**(*model*, *admin_site*)
Bases: django.contrib.admin.options.ModelAdmin

list_display = ('slug', 'name', 'id')

sort_fields = ('id', 'name', 'slug')

readonly_fields = ('id',)

search_fields = ('id', 'name', 'slug')

fields = ('id', 'name', 'slug')

media

class archivebox.core.admin.**ArchiveBoxAdmin**(*name='admin'*)
Bases: django.contrib.admin.sites.AdminSite

site_header = 'ArchiveBox'

index_title = 'Links'

site_title = 'Index'

get_urls()

add_view(*request*)

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

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

get(*request*, *path*)

class archivebox.core.views.**PublicArchiveView**(***kwargs*)
Bases: django.views.generic.list.ListView

template = 'snapshot_list.html'

model
alias of core.models.Snapshot

paginate_by = 100

ordering = ['title']

get_queryset(***kwargs*)
Return the list of items for this view.

The return value must be an iterable and may be an instance of *QuerySet* in which case *QuerySet* specific behavior will be enabled.

get(**args*, ***kwargs*)

class archivebox.core.views.**AddView**(***kwargs*)
Bases: django.contrib.auth.mixins.UserPassesTestMixin, django.views.generic.edit.FormView

template_name = 'add_links.html'

form_class
alias of core.forms.AddLinkForm

get_initial()
Prefill the AddLinkForm with the 'url' GET parameter

test_func()

get_context_data(**args*, ***kwargs*)
Insert the form into the context dict.

form_valid(*form*)
If the form is valid, redirect to the supplied URL.

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[pathlib.Path] = None*) → bool

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

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

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

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

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

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

take screenshot of site using chrome –headless

archivebox.extractors.title module

class archivebox.extractors.title.**TitleParser**(**args*, ***kwargs*)

Bases: html.parser.HTMLParser

title

handle_starttag(*tag*, *attrs*)

handle_data(*data*)

handle_endtag(*tag*)

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

archivebox.extractors.title.**extract_title_with_regex**(*html*)

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

archivebox.extractors.wget.**save_wget**(*link: archivebox.index.schema.Link*, *out_dir: Optional[pathlib.Path] = 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.**get_default_archive_methods**()

archivebox.extractors.**ignore_methods**(*to_ignore: List[str]*)

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

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

archivebox.extractors.**archive_links**(*all_links: Union[Iterable[archivebox.index.schema.Link], django.db.models.query.QuerySet]*, *overwrite: bool = False*, *methods: Optional[Iterable[str]] = None*, *out_dir: Optional[pathlib.Path] = None*) → List[archivebox.index.schema.Link]

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.**parse_html_main_index**(*out_dir: pathlib.Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/*) → 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: pathlib.Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/*, *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*, *template: str = '/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkou*) → 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: pathlib.Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/*) → 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: pathlib.Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/*) → 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: Union[pathlib.Path, str]*, *guess: Optional[bool] = False*) → Optional[archivebox.index.schema.Link]

load the json link index from a given directory

archivebox.index.json.**parse_json_links_details**(*out_dir: Union[pathlib.Path, 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)
```

(continues on next page)

(continued from previous page)

```
    # 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 guess_ts(*dict_info*)

classmethod from_json(*json_info*, *guess=False*)

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*, *guess=False*)

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

archive_size

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: pathlib.Path = Posix-Path('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/che*
→ Iterator[archivebox.index.schema.Link]

archivebox.index.sql.**remove_from_sql_main_index**(*snapshots: django.db.models.query.QuerySet, out_dir: pathlib.Path = Posix-Path('/home/docs/checkouts/readthedocs.org/user_builds/archiv*
→ None

archivebox.index.sql.**write_link_to_sql_index**(*link: archivebox.index.schema.Link*)

archivebox.index.sql.**write_sql_main_index**(*links: List[archivebox.index.schema.Link], out_dir: pathlib.Path = Posix-Path('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/che*
→ None

archivebox.index.sql.**write_sql_link_details**(*link: archivebox.index.schema.Link, out_dir: pathlib.Path = Posix-Path('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/*
→ None

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

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

archivebox.index.sql.**get_admins**(*out_dir: pathlib.Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.24/*
→ 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.**fix_duplicate_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: pathlib.Path*)

archivebox.index.**write_main_index**(*links: List[archivebox.index.schema.Link], out_dir: pathlib.Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.*
finished: bool = False) → None
Writes links to sqlite3 file for a given list of links

archivebox.index.**get_empty_snapshot_queryset**(*out_dir: pathlib.Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebo*

archivebox.index.**load_main_index**(*out_dir: pathlib.Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.2*
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: pathlib.Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkout*
→ Optional[dict]

archivebox.index.**parse_links_from_source**(*source_path: str*, *root_url: Optional[str] = None*) → Tuple[List[archivebox.index.schema.Link], List[archivebox.index.schema.Link]]

archivebox.index.**fix_duplicate_links_in_index**(*snapshots: django.db.models.query.QuerySet*, *links: Iterable[archivebox.index.schema.Link]*) → Iterable[archivebox.index.schema.Link]

Given a list of in-memory Links, dedupe and merge them with any conflicting Snapshots in the DB.

archivebox.index.**dedupe_links**(*snapshots: django.db.models.query.QuerySet*, *new_links: List[archivebox.index.schema.Link]*) → List[archivebox.index.schema.Link]

The validation of links happened at a different stage. This method will focus on actual deduplication and timestamp fixing.

archivebox.index.**write_link_details**(*link: archivebox.index.schema.Link*, *out_dir: Optional[str] = None*, *skip_sql_index: bool = False*) → 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.**snapshot_filter**(*snapshots: django.db.models.query.QuerySet*, *filter_patterns: List[str]*, *filter_type: str = 'exact'*) → django.db.models.query.QuerySet

archivebox.index.**get_indexed_folders**(*snapshots*, *out_dir: pathlib.Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/*
→ Dict[str, Optional[archivebox.index.schema.Link]]

indexed links without checking archive status or data directory validity

archivebox.index.**get_archived_folders**(*snapshots*, *out_dir: pathlib.Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkout*
→ Dict[str, Optional[archivebox.index.schema.Link]]

indexed links that are archived with a valid data directory

archivebox.index.**get_unarchived_folders**(*snapshots*, *out_dir: pathlib.Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/check*
→ 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**(*snapshots*, *out_dir: pathlib.Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/*
→ Dict[str, Optional[archivebox.index.schema.Link]]

dirs that actually exist in the archive/ folder

archivebox.index.**get_valid_folders**(*snapshots*, *out_dir: pathlib.Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.*
→ 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**(*snapshots*, *out_dir: pathlib.Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/*
→ Dict[str, Optional[archivebox.index.schema.Link]]

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

archivebox.index.**get_duplicate_folders**(*snapshots*, *out_dir: pathlib.Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkou*) → 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**(*snapshots*, *out_dir: pathlib.Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkout*) → 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**(*snapshots*, *out_dir: pathlib.Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkou*) → 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**(*snapshots*, *out_dir: pathlib.Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/che*) → 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: pathlib.Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archiveb*) → 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]*, ***_kwargs*) → 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]*, ***_kwargs*) → 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]*, ***_kwargs*) → 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]*, ***_kwargs*) → 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]*, ***_kwargs*) → 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]*, ***_kwargs*) → 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]*, ***_kwargs*) → 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]*, ***_kwargs*) → 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_memory**(*urls: List[str]*, *root_url: Optional[str] = None*)

parse a list of URLS without touching the filesystem

archivebox.parsers.**parse_links**(*source_file: str*, *root_url: Optional[str] = None*) → 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.**run_parser_functions**(*to_parse: IO[str]*, *timer*, *root_url: Optional[str] = None*) → Tuple[List[archivebox.index.schema.Link], Optional[str]]

archivebox.parsers.**save_text_as_source**(*raw_text: str*, *filename: str = '{ts}-stdin.txt'*, *out_dir: pathlib.Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkou*
→ str

archivebox.parsers.**save_file_as_source**(*path: str*, *timeout: int = 60*, *filename: str = '{ts}-{basename}.txt'*, *out_dir: pathlib.Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkou*
→ 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: pathlib.Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/check*
→ None
Print the ArchiveBox help message and usage

archivebox.main.**version**(*quiet: bool = False*, *out_dir: pathlib.Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.24/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: pathlib.Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.24/docs')*)
→ None
Run a given ArchiveBox subcommand with the given list of args

archivebox.main.**init**(*force: bool = False*, *out_dir: pathlib.Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.24/docs')*)
→ None
Initialize a new ArchiveBox collection in the current directory

archivebox.main.**status**(*out_dir: pathlib.Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/ch*
→ None
Print out some info and statistics about the archive collection

archivebox.main.**oneshot**(*url: str*, *out_dir: pathlib.Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.24/docs')*)
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.

archivebox.main.**add**(*urls: Union[str, List[str]]*, *depth: int = 0*, *update_all: bool = False*, *index_only: bool = False*, *overwrite: bool = False*, *init: bool = False*, *out_dir: pathlib.Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.24/docs')*, *extractors: str = ''*) → 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'*, *snapshots: Optional[django.db.models.query.QuerySet] = None*, *after: Optional[float] = None*, *before: Optional[float] = None*, *yes: bool = False*, *delete: bool = False*, *out_dir: pathlib.Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.24/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: pathlib.Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.24/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*, *html: bool = False*, *with_headers: bool = False*, *out_dir: pathlib.Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.24/docs')*) → Iterable[archivebox.index.schema.Link]

List, filter, and export information about archive entries

archivebox.main.**list_links**(*snapshots: Optional[django.db.models.query.QuerySet] = None*, *filter_patterns: Optional[List[str]] = None*, *filter_type: str = 'exact'*, *after: Optional[float] = None*, *before: Optional[float] = None*, *out_dir: pathlib.Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.24/docs')*) → Iterable[archivebox.index.schema.Link]

archivebox.main.**list_folders**(*links: List[archivebox.index.schema.Link]*, *status: str*, *out_dir: pathlib.Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.24/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: pathlib.Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.24/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*, *depth: int = 0*, *import_path: Optional[str] = None*, *out_dir: pathlib.Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.24/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*, *init: bool = False*, *out_dir: pathlib.Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.24/docs')*) → None

Run the ArchiveBox HTTP server

archivebox.main.**manage**(*args: Optional[List[str]] = None*, *out_dir: pathlib.Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/checkouts/v0.4.24/docs')*) → None
Run an ArchiveBox Django management command

archivebox.main.**shell**(*out_dir: pathlib.Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/archivebox/chec*) → None
Enter an interactive ArchiveBox Django shell

archivebox.manage module

archivebox.system module

archivebox.system.**run**(**args*, *input=None*, *capture_output=True*, *text=False*, ***kwargs*)
Patched of subprocess.run to fix blocking io making timeout=innefective

archivebox.system.**atomic_write**(*path: Union[pathlib.Path, str], contents: Union[dict, str, bytes], overwrite: bool = True*) → None
Safe atomic write to filesystem by writing to temp file + atomic rename

archivebox.system.**chmod_file**(*path: str*, *cwd: str = '.'*, *permissions: str = '755'*) → None
chmod -R <permissions> <cwd>/<path>

archivebox.system.**copy_and_overwrite**(*from_path: Union[str, pathlib.Path], to_path: Union[str, pathlib.Path]*)
copy a given file or directory to a given path, overwriting the destination

archivebox.system.**get_dir_size**(*path: Union[str, pathlib.Path], 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

class archivebox.system.**suppress_output**(*stdout=True*, *stderr=True*)
Bases: object

A context manager for doing a "deep suppression" of stdout and stderr in Python, i.e. will suppress all print, even if the print originates in a compiled C/Fortran sub-function.

This will not suppress raised exceptions, since exceptions are printed

to stderr just before a script exits, and after the context manager has exited (at least, I think that is why it lets exceptions through).

with suppress_stdout_stderr(): rogue_function()

archivebox.util module

archivebox.util.**detect_encoding**(*rawdata*)

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.**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.**is_static_file**(*url: str*)

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 = None*) → str
Download the contents of a remote url and return the text

archivebox.util.**get_headers**(*url: str*, *timeout: int = None*) → str
Download the contents of a remote url and return the headers

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

archivebox.util.**ansi_to_html**(*text*)
Based on: https://stackoverflow.com/questions/19212665/python-converting-ansi-color-codes-to-html

class archivebox.util.**AttributeDict**(**args*, ***kwargs*)
Bases: dict

Helper to allow accessing dict values via Example.key or Example['key']

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.

Planned Specification

(this is not set in stone, just a rough estimate)

v0.5: Remove live-updated JSON & HTML index in favor of archivebox export

- use SQLite as the main db and export staticfile indexes once at the *end* of the whole process instead of live-updating them during each extractor run (i.e. remove patch_main_index)
- create archivebox export command
- we have to create a public view to replace index.html / old.html used for non-logged in users

v0.6: Code cleanup / refactor

- move config loading logic into settings.py
- move all the extractors into "plugin" style folders that register their own config

- right now, the paths of the extractor output are scattered all over the codebase, e.g. `output.pdf` (should be moved to constants at the top of the plugin config file)
- make out_dir, link_dir, extractor_dir, naming consistent across codebase
- convert all `os.path` calls and raw string paths to `Pathlib`

v0.7: Schema improvements

- remove `timestamps` as primary keys in favor of hashes, UUIDs, or some other slug
- create a migration system for folder layout independent of the index (`mv` is atomic at the FS level, so we just need a `transaction.atomic(): move(oldpath, newpath); snap.data_dir = newpath; snap.save()`)
- make `Tag` a real model `ManyToMany` with Snapshots
- allow multiple Snapshots of the same site over time + CLI / UI to manage those, + migration from old style `#2020-01-01` hack to proper versioned snapshots

v0.8: Security

- Add CSRF/CSP/XSS protection to rendered archive pages
- Provide secure reverse proxy in front of archivebox server in docker-compose.yml
- Create UX flow for users to setup session cookies / auth for archiving private sites
  - cookies for wget, curl, etc low-level commands
  - localstorage, cookies, indexedb setup for chrome archiving methods

v0.9: Performance

- setup huey, break up archiving process into tasks on a queue that a worker pool executes
- setup pyppeteer2 to wrap chrome so that it's not open/closed during each extractor

v1.0: Full headless browser control

- run user-scripts / extensions in the context of the page during archiving
- community userscripts for unrolling twitter threads, reddit threads, youtube comment sections, etc.
- pywb-based headless browser session recording and warc replay
- archive proxy support
  - support sending upstream requests through an external proxy
  - support for exposing a proxy that archives all downstream traffic

...

v2.0 Federated or distributed archiving + paid hosted service offering

- merkel tree for storing archive output subresource hashes
- DHT for assigning merkel tree hash:file shards to nodes
- tag system for tagging certain hashes with human-readable names, e.g. title, url, tags, filetype etc.
- distributed tag lookup system

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!*

Past Releases

To see how this spec has been scheduled / implemented / released so far, read these pull requests:

- v0.2.x
- v0.3.x
- v0.4.x
- v0.5.x

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!

- v0.4.9 released
  - `pip install archivebox` https://pypi.org/project/archivebox/
  - `docker run archivebox/archivebox` https://hub.docker.com/r/archivebox/archivebox
  - https://archivebox.readthedocs.io/en/latest/
  - https://github.com/ArchiveBox/ArchiveBox/releases
- easy migration from previous versions

```
cd path/to/your/archive/folder
archivebox init
archviebox add 'https://example.com'
archviebox add 'https://getpocket.com/users/USERNAME/feed/all' --depth=1
```

- full transition to Django Sqlite DB with migrations (making upgrades between versions much safer now)
- maintains an intuitive and helpful CLI that's backwards-compatible with all previous archivebox data versions
- uses argparse instead of hand-written CLI system: see `archivebox/cli/archivebox.py`
- new subcommands-based CLI for `archivebox` (see below)
- new Web UI with pagination, better search, filtering, permissions, and more
- 30+ assorted bugfixes, new features, and tickets closed
- for more info, see: https://github.com/ArchiveBox/ArchiveBox/releases/tag/v0.4.9

- 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 (sshhh 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-Specific Posts, Tutorials, and Guides*
  - *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**
- 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 Rhizome.org/WebRecorder.io/Conifer team

- **Conifer by Rhizome.org An open-source personal archiving server that uses pywb under the hood** previously known as *Webrecorder.io*
- **Webrecorder.net Suite of open source projects and tools, led by Ilya Kreymer, to capture interactive websites and replay them at a later time as accurately as possible**
- **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

- **OpenWayback Open source project developing core Wayback-Machine components**
- **awesome-web-archiving Large list of archiving projects and orgs**
- 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**
- Archivy Python-based self-hosted knowledge base embedded into your filesystem
- 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)
- TiddlyWiki Non-linear bookmark and note-taking tool with archiving support

---

Smaller Utilities

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

- https://github.com/gildas-lormeau/SingleFile/ Web Extension for Firefox and Chrome to save a web page as a single HTML file
- https://github.com/vrtdev/save-page-state A Chrome extension for saving the state of a page in multiple formats
- 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://github.com/Lifesgood123/prevent-link-rot Replace any broken URLs in some content with Wayback machine URL equivalents
- 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://theconversation.com/your-internet-data-is-rotting-115891
- 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-Specific Posts, Tutorials, and Guides

- "How to install ArchiveBox to preserve websites you care about" https://blog.sleeplessbeastie.eu/2019/06/19/how-to-install-archivebox-to-preserve-websites-you-care-about/
- "How to remotely archive websites using ArchiveBox" https://blog.sleeplessbeastie.eu/2019/06/26/how-to-remotely-archive-websites-using-archivebox/
- "How to use CutyCapt inside ArchiveBox" https://blog.sleeplessbeastie.eu/2019/07/10/how-to-use-cutycapt-inside-archivebox/
- "Automate ArchiveBox with Google Spreadsheet to Backup your internet" https://manfred.life/archivebox
- "♪ConoHaArchiveBox" https://qiita.com/CloudRemix/items/691caf91efa3ef19a7ad
- "WEB-ARCHIV TEIL 8: WALLABAG UND ARCHIVEBOX" http://webermartin.net/blog/web-archiv-teil-8-wallabag-und-archivebox/

- https://metaxyntax.neocities.org/entries/7.html

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)
- **Rhizome.org** The digital preservation group that works on Webrecorder.io (USA)
- **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

L

M

N

T

U

V

W