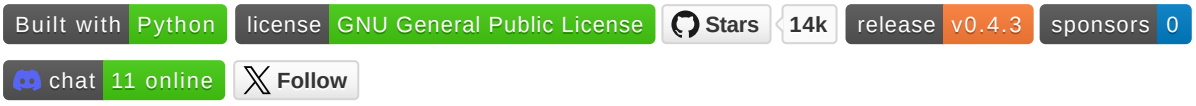
GitHub - maurosoria/dirsearch: Web path scanner

By maurosoria

Archived: 2026-04-06 00:09:01 UTC

dirsearch - Web path discovery



> An advanced web path brute-forcer

dirsearch is being actively developed by @maurosoria and @shelld3v

Reach to our Discord server to communicate with the team at best

Table of Contents

Supported Platforms

dirsearch runs on multiple platforms and can be used either via Python or standalone binaries:

Platform	Python	Standalone Binary
Linux (x86_64)	Python 3.9+	`dirsearch-linux-amd64`
Windows (x64)	Python 3.9+	`dirsearch-windows-x64.exe`
macOS (Intel)	Python 3.9+	`dirsearch-macos-intel`
macOS (Apple Silicon)	Python 3.9+	`dirsearch-macos-silicon`

Standalone binaries are self-contained executables that don't require Python installation.

Installation & Usage

Requirement: python 3.9 or higher

Choose one of these installation options:

- Install with **git**: `git clone https://github.com/maurosoria/dirsearch.git --depth 1` (**RECOMMENDED**)
- Install with ZIP file: Download here
- Install with Docker: `docker build -t "dirsearch:v0.4.3" .` (more information can be found here)
- Install with PyPi: `pip3 install dirsearch` or `pip install dirsearch`
- Install with Kali Linux: `sudo apt-get install dirsearch` (deprecated)

Standalone Binaries

Pre-built standalone binaries are available for all major platforms. These don't require Python to be installed.

Download from Releases

Platform	Binary Name	Architecture
Linux	`dirsearch-linux-amd64`	x86_64
Windows	`dirsearch-windows-x64.exe`	x64
macOS Intel	`dirsearch-macos-intel`	x86_64
macOS Apple Silicon	`dirsearch-macos-silicon`	ARM64

Usage:

```
# Linux/macOS - make executable first
chmod +x dirsearch-linux-amd64
./dirsearch-linux-amd64 -u https://target

# Windows
dirsearch-windows-x64.exe -u https://target
```

Note: Standalone binaries include bundled `db/` wordlists and `config.ini`. Session files are stored in `$HOME/.dirsearch/sessions/` when using bundled builds.

Wordlists (IMPORTANT)

Summary:

- Wordlist is a text file, each line is a path.
- About extensions, unlike other tools, dirsearch only replaces the `%EXT%` keyword with extensions from **-e** flag.
- For wordlists without `%EXT%` (like SecLists), **-f | --force-extensions** switch is required to append extensions to every word in wordlist, as well as the `/`.
- To apply your extensions to wordlist entries that have extensions already, use **-O** | **--overwrite-extensions** (Note: some extensions are excluded from being overwritten such as *.log*, *.json*, *.xml*, ... or media extensions like *.jpg*, *.png*)
- To use multiple wordlists, you can separate your wordlists with commas. Example: `wordlist1.txt,wordlist2.txt`.
- Bundled wordlist categories live in `db/categories/` and can be selected with **--wordlist-categories**. Available: `extensions`, `conf`, `vcs`, `backups`, `db`, `logs`, `keys`, `web`, `common` (use `all` to include everything).

▶ **Wordlist Examples (click to expand)**

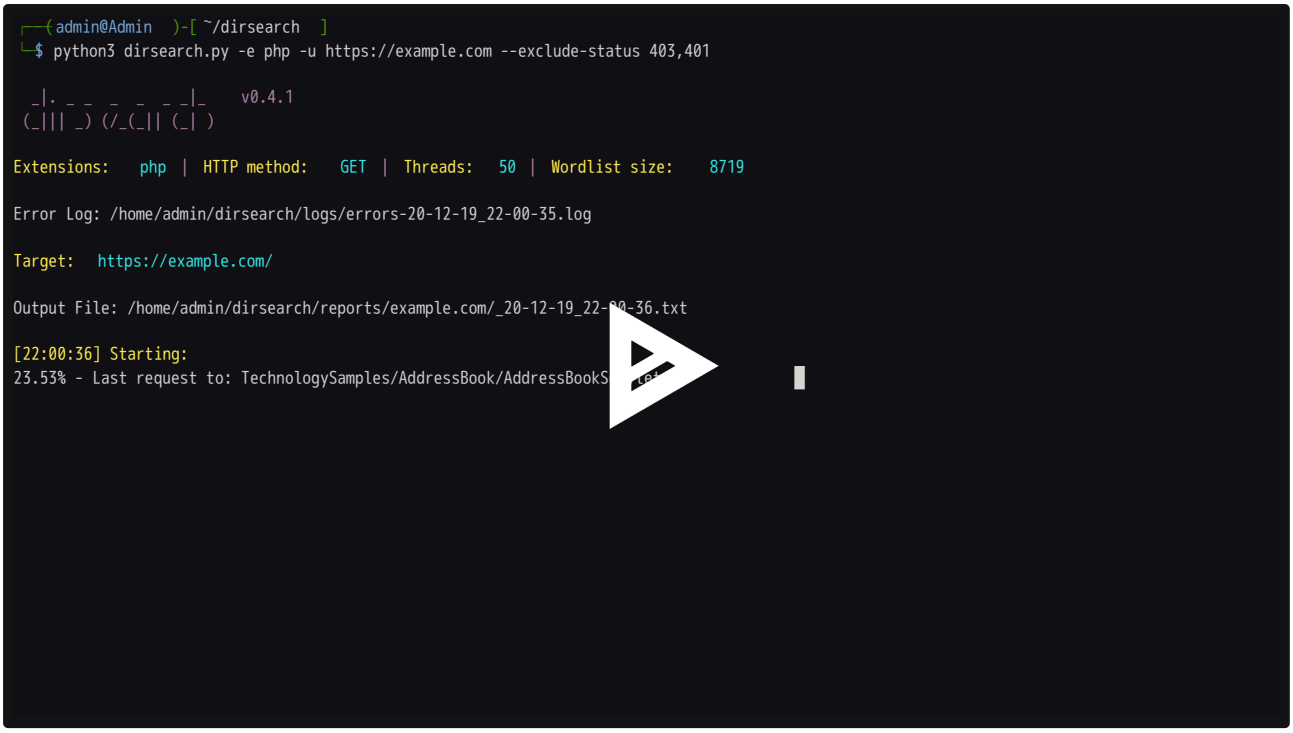
Options

▶ **Full Options List (click to expand)**

Configuration

▶ **Configuration File Reference (click to expand)**

How to use



Some examples for how to use dirsearch - those are the most common arguments. If you need all, just use the **-h** argument.

Simple usage

```
python3 dirsearch.py -u https://target
```

```
python3 dirsearch.py -e php,html,js -u https://target
```

```
python3 dirsearch.py -e php,html,js -u https://target -w /path/to/wordlist
```

▶ **More Usage Examples (click to expand)**

Session Management

dirsearch supports saving and resuming scan sessions, allowing you to pause a long-running scan and continue it later.

Session Format

Sessions are stored in **JSON format** (directory-based structure) for human readability and easy inspection. Legacy `.pickle` / `.pkl` session files are no longer supported.

Session directory structure:

```
session_name/
├─── meta.json        # Version, timestamps, output history
├─── controller.json  # Scan state (URLs, directories, progress)
├─── dictionary.json  # Wordlist state and position
└─── options.json     # Command-line options used
```

Saving a Session

When you pause a scan with **CTRL+C**, you'll be prompted to save the session:

```
python3 dirsearch.py -u https://target -e php
# Press CTRL+C during scan
# Select "save" and provide a session name
```

Resuming a Session

Resume a saved session with the **-s** / **--session** flag:

```
python3 dirsearch.py -s sessions/my_session
```

Listing Available Sessions

View all resumable sessions with **--list-sessions**:

```
python3 dirsearch.py --list-sessions
```

This displays:

- Session path
- Target URL
- Remaining targets and directories
- Jobs processed
- Error count
- Last modified time

Custom Sessions Directory

Specify a custom directory to search for sessions:

```
python3 dirsearch.py --list-sessions --sessions-dir /path/to/sessions
```

Default session locations:

- **Source install:** `<dirsearch>/sessions/`
- **Bundled binary:** `$HOME/.dirsearch/sessions/`

Output History

Sessions maintain a history of previous scan outputs, allowing you to review results from interrupted scans. Each resume appends to the output history with timestamps.

Support Docker

▶ **Docker Installation & Usage (click to expand)**

Building from Source

You can build standalone executables using PyInstaller. This creates a single binary file that includes all dependencies.

Requirements

- Python 3.9+
- PyInstaller 6.3.0+
- All dependencies from `requirements.txt`

Quick Build

```
# Install dependencies
pip install -r requirements.txt
pip install pyinstaller==6.3.0

# Build using the spec file
pyinstaller pyinstaller/dirsearch.spec

# Binary will be in dist/dirsearch
./dist/dirsearch --version
```

Manual Build (Linux/macOS)

```
pyinstaller \
  --onefile \
  --name dirsearch \
  --paths=. \
  --collect-submodules=lib \
  --add-data "db:db" \
  --add-data "config.ini:." \
  --add-data "lib/report:lib/report" \
```

```
  --hidden-import=requests \
  --hidden-import=httpx \
  --hidden-import=urllib3 \
  --hidden-import=jinja2 \
  --hidden-import=colorama \
  --strip \
  --clean \
  dirsearch.py
```

Manual Build (Windows)

```
pyinstaller `
  --onefile `
  --name dirsearch `
  --paths=. `
  --collect-submodules=lib `
  --add-data "db;db" `
  --add-data "config.ini;." `
  --add-data "lib/report;lib/report" `
  --hidden-import=requests `
  --hidden-import=httpx `
  --hidden-import=urllib3 `
  --hidden-import=jinja2 `
  --hidden-import=colorama `
  --clean `
  dirsearch.py
```

Note: Windows uses `;` instead of `:` as the path separator in `--add-data`.

Build Output

After building:

- **Linux/macOS:** `dist/dirsearch`
- **Windows:** `dist/dirsearch.exe`

The binary includes:

- All Python dependencies
- `db/` directory (wordlists, blacklists)
- `config.ini` (default configuration)
- `lib/report/` (Jinja2 templates for reports)

CI/CD & GitHub Workflows

dirsearch uses GitHub Actions for continuous integration and automated builds.

Available Workflows

Workflow	Trigger	Description
Inspection (CI)	Push, PR	Runs tests, linting, and codespell on Python 3.9/3.11 across Ubuntu and Windows
PyInstaller Linux	Manual, Workflow call	Builds `dirsearch-linux-amd64` binary
PyInstaller Windows	Manual, Workflow call	Builds `dirsearch-windows-x64.exe` binary
PyInstaller macOS Intel	Manual, Workflow call	Builds `dirsearch-macos-intel` binary
PyInstaller macOS Silicon	Manual, Workflow call	Builds `dirsearch-macos-silicon` binary
PyInstaller Draft Release	Manual	Builds all platforms and creates a draft GitHub release
Docker Image	Push, PR	Builds and tests Docker image
CodeQL Analysis	Push, PR, Schedule	Security scanning with GitHub CodeQL
Semgrep Analysis	Push, PR	Static analysis with Semgrep

Running Workflows Manually

PyInstaller builds can be triggered manually from the GitHub Actions tab:

1. Go to **Actions** > Select workflow (e.g., "PyInstaller Linux")
2. Click **Run workflow**
3. Download artifacts from the completed run

Creating a Release

To create a new release with all platform binaries:

1. Go to **Actions** > **PyInstaller Draft Release**
2. Click **Run workflow**
3. Enter the tag (e.g., `v0.4.4`)
4. Select target branch
5. Optionally mark as prerelease
6. Review and publish the draft release

Build Matrix

The CI workflow tests on:

- **Python versions:** 3.9, 3.11
- **Operating systems:** Ubuntu (latest), Windows (latest)

References

▶ **Articles & Tutorials (click to expand)**

Tips

- The server has requests limit? That's bad, but feel free to bypass it, by randomizing proxy with `--proxy-list`
- Want to find out config files or backups? Try `--suffixes ~` and `--prefixes .`
- Want to find only folders/directories? Why not combine `--remove-extensions` and `--suffixes /`!
- The mix of `--cidr`, `-F`, `-q` and will reduce most of noises + false negatives when brute-forcing with a CIDR
- Scan a list of URLs, but don't want to see a 429 flood? `--skip-on-status 429` will help you to skip a target whenever it returns 429
- The server contains large files that slow down the scan? You *might* want to use `HEAD` HTTP method instead of `GET`
- Brute-forcing CIDR is slow? Probably you forgot to reduce request timeout and request retries. Suggest: `--timeout 3 --retries 1`

Contribution

We have been receiving a lot of helps from many people around the world to improve this tool. Thanks so much to everyone who have helped us so far! See CONTRIBUTORS.md to know who they are.

Pull requests and feature requests are welcomed

License
