
Certbot Documentation

Release 1.0.0.dev0

Certbot Project

Nov 12, 2019

CONTENTS

| | | |
|----------|---|-----------|
| 1 | Introduction | 1 |
| 1.1 | Contributing | 1 |
| 1.2 | How to run the client | 1 |
| 1.3 | Understanding the client in more depth | 2 |
| 2 | What is a Certificate? | 3 |
| 2.1 | Certificates and Lineages | 3 |
| 3 | Get Certbot | 5 |
| 3.1 | About Certbot | 5 |
| 3.2 | System Requirements | 6 |
| 3.3 | Alternate installation methods | 6 |
| 4 | User Guide | 11 |
| 4.1 | Certbot Commands | 12 |
| 4.2 | Getting certificates (and choosing plugins) | 12 |
| 4.3 | Managing certificates | 17 |
| 4.4 | Changing a Certificate's Domains | 18 |
| 4.5 | Where are my certificates? | 22 |
| 4.6 | Pre and Post Validation Hooks | 23 |
| 4.7 | Changing the ACME Server | 25 |
| 4.8 | Lock Files | 25 |
| 4.9 | Configuration file | 26 |
| 4.10 | Log Rotation | 26 |
| 4.11 | Certbot command-line options | 27 |
| 4.12 | Getting help | 39 |
| 5 | Developer Guide | 41 |
| 5.1 | Getting Started | 42 |
| 5.2 | Code components and layout | 44 |
| 5.3 | Coding style | 46 |
| 5.4 | Use <code>certbot.compat.os</code> instead of <code>os</code> | 46 |
| 5.5 | Mypy type annotations | 47 |
| 5.6 | Submitting a pull request | 47 |
| 5.7 | Asking for help | 48 |
| 5.8 | Updating certbot-auto and letsencrypt-auto | 48 |
| 5.9 | Updating the documentation | 49 |
| 5.10 | Running the client with Docker | 49 |
| 5.11 | Notes on OS dependencies | 49 |
| 6 | Packaging Guide | 51 |

| | | |
|----------|---|------------|
| 6.1 | Releases | 51 |
| 6.2 | Notes for package maintainers | 51 |
| 7 | Resources | 53 |
| 8 | API Documentation | 55 |
| 8.1 | <code>certbot.achallenges</code> | 55 |
| 8.2 | <code>certbot.cli</code> | 56 |
| 8.3 | <code>certbot.configuration</code> | 60 |
| 8.4 | <code>certbot.crypto_util</code> | 61 |
| 8.5 | <code>certbot.display</code> | 65 |
| 8.6 | <code>certbot.errors</code> | 72 |
| 8.7 | <code>certbot.hooks</code> | 74 |
| 8.8 | <code>certbot</code> | 76 |
| 8.9 | <code>certbot.interfaces</code> | 76 |
| 8.10 | <code>certbot.plugins.common</code> | 85 |
| 8.11 | <code>certbot.plugins.dns_common</code> | 88 |
| 8.12 | <code>certbot.plugins.dns_common_lexicon</code> | 90 |
| 8.13 | <code>certbot.plugins.util</code> | 91 |
| 8.14 | <code>certbot.reverter</code> | 91 |
| 8.15 | <code>certbot.util</code> | 94 |
| 9 | Indices and tables | 99 |
| | Python Module Index | 101 |
| | Index | 103 |

INTRODUCTION

Note: To get started quickly, use the [interactive installation guide](#).

Certbot is part of EFF's effort to encrypt the entire Internet. Secure communication over the Web relies on HTTPS, which requires the use of a digital certificate that lets browsers verify the identity of web servers (e.g., is that really google.com?). Web servers obtain their certificates from trusted third parties called certificate authorities (CAs). Certbot is an easy-to-use client that fetches a certificate from Let's Encrypt—an open certificate authority launched by the EFF, Mozilla, and others—and deploys it to a web server.

Anyone who has gone through the trouble of setting up a secure website knows what a hassle getting and maintaining a certificate is. Certbot and Let's Encrypt can automate away the pain and let you turn on and manage HTTPS with simple commands. Using Certbot and Let's Encrypt is free, so there's no need to arrange payment.

How you use Certbot depends on the configuration of your web server. The best way to get started is to use our [interactive guide](#). It generates instructions based on your configuration settings. In most cases, you'll need [root or administrator access](#) to your web server to run Certbot.

Certbot is meant to be run directly on your web server, not on your personal computer. If you're using a hosted service and don't have direct access to your web server, you might not be able to use Certbot. Check with your hosting provider for documentation about uploading certificates or using certificates issued by Let's Encrypt.

Certbot is a fully-featured, extensible client for the Let's Encrypt CA (or any other CA that speaks the [ACME](#) protocol) that can automate the tasks of obtaining certificates and configuring webservers to use them. This client runs on Unix-based operating systems.

To see the changes made to Certbot between versions please refer to our [changelog](#).

Until May 2016, Certbot was named simply `letsencrypt` or `letsencrypt-auto`, depending on install method. Instructions on the Internet, and some pieces of the software, may still refer to this older name.

1.1 Contributing

If you'd like to contribute to this project please read [Developer Guide](#).

This project is governed by [EFF's Public Projects Code of Conduct](#).

1.2 How to run the client

The easiest way to install and run Certbot is by visiting [certbot.eff.org](#), where you can find the correct instructions for many web server and OS combinations. For more information, see [Get Certbot](#).

1.3 Understanding the client in more depth

To understand what the client is doing in detail, it's important to understand the way it uses plugins. Please see the [explanation of plugins](#) in the User Guide.

1.3.1 Links

Documentation: <https://certbot.eff.org/docs>

Software project: <https://github.com/certbot/certbot>

Notes for developers: <https://certbot.eff.org/docs/contributing.html>

Main Website: <https://certbot.eff.org>

Let's Encrypt Website: <https://letsencrypt.org>

Community: <https://community.letsencrypt.org>

ACME spec: <http://ietf-wg-acme.github.io/acme/>

ACME working area in github: <https://github.com/ietf-wg-acme/acme>



1.3.2 System Requirements

See <https://certbot.eff.org/docs/install.html#system-requirements>.

WHAT IS A CERTIFICATE?

A public key or digital *certificate* (formerly called an SSL certificate) uses a public key and a private key to enable secure communication between a client program (web browser, email client, etc.) and a server over an encrypted SSL (secure socket layer) or TLS (transport layer security) connection. The certificate is used both to encrypt the initial stage of communication (secure key exchange) and to identify the server. The certificate includes information about the key, information about the server identity, and the digital signature of the certificate issuer. If the issuer is trusted by the software that initiates the communication, and the signature is valid, then the key can be used to communicate securely with the server identified by the certificate. Using a certificate is a good way to prevent “man-in-the-middle” attacks, in which someone in between you and the server you think you are talking to is able to insert their own (harmful) content.

You can use Certbot to easily obtain and configure a free certificate from Let’s Encrypt, a joint project of EFF, Mozilla, and many other sponsors.

2.1 Certificates and Lineages

Certbot introduces the concept of a *lineage*, which is a collection of all the versions of a certificate plus Certbot configuration information maintained for that certificate from renewal to renewal. Whenever you renew a certificate, Certbot keeps the same configuration unless you explicitly change it, for example by adding or removing domains. If you add domains, you can either add them to an existing lineage or create a new one.

See also: *Re-creating and Updating Existing Certificates*

GET CERTBOT

Table of Contents

- *About Certbot*
- *System Requirements*
- *Alternate installation methods*
 - *Certbot-Auto*
 - *Problems with Python virtual environment*
 - *Running with Docker*
 - *Operating System Packages*
 - *Installing from source*

3.1 About Certbot

Certbot is meant to be run directly on a web server, normally by a system administrator. In most cases, running Certbot on your personal computer is not a useful option. The instructions below relate to installing and running Certbot on a server.

System administrators can use Certbot directly to request certificates; they should *not* allow unprivileged users to run arbitrary Certbot commands as `root`, because Certbot allows its user to specify arbitrary file locations and run arbitrary scripts.

Certbot is packaged for many common operating systems and web servers. Check whether `certbot` (or `letsencrypt`) is packaged for your web server's OS by visiting certbot.eff.org, where you will also find the correct installation instructions for your system.

Note: Unless you have very specific requirements, we kindly suggest that you use the Certbot packages provided by your package manager (see certbot.eff.org). If such packages are not available, we recommend using `certbot-auto`, which automates the process of installing Certbot on your system.

3.2 System Requirements

Certbot currently requires Python 2.7 or 3.4+ running on a UNIX-like operating system. By default, it requires root access in order to write to `/etc/letsencrypt`, `/var/log/letsencrypt`, `/var/lib/letsencrypt`; to bind to port 80 (if you use the `standalone` plugin) and to read and modify webserver configurations (if you use the `apache` or `nginx` plugins). If none of these apply to you, it is theoretically possible to run without root privileges, but for most users who want to avoid running an ACME client as root, either `letsencrypt-nosudo` or `simp_le` are more appropriate choices.

The Apache plugin currently requires an OS with `augeas` version 1.0; currently it supports modern OSes based on Debian, Ubuntu, Fedora, SUSE, Gentoo and Darwin.

Additional integrity verification of `certbot-auto` script can be done by verifying its digital signature. This requires a local installation of `gpg2`, which comes packaged in many Linux distributions under name `gnupg` or `gnupg2`.

Installing with `certbot-auto` requires 512MB of RAM in order to build some of the dependencies. Installing from pre-built OS packages avoids this requirement. You can also temporarily set a swap file. See “Problems with Python virtual environment” below for details.

3.3 Alternate installation methods

If you are offline or your operating system doesn’t provide a package, you can use an alternate method for installing certbot.

3.3.1 Certbot-Auto

The `certbot-auto` wrapper script installs Certbot, obtaining some dependencies from your web server OS and putting others in a python virtual environment. You can download and run it as follows:

```
user@webserver:~$ wget https://dl.eff.org/certbot-auto
user@webserver:~$ sudo mv certbot-auto /usr/local/bin/certbot-auto
user@webserver:~$ sudo chown root /usr/local/bin/certbot-auto
user@webserver:~$ chmod 0755 /usr/local/bin/certbot-auto
user@webserver:~$ /usr/local/bin/certbot-auto --help
```

To check the integrity of the `certbot-auto` script, you can use these steps:

```
user@webserver:~$ wget -N https://dl.eff.org/certbot-auto.asc
user@webserver:~$ gpg2 --keyserver pool.sks-keyservers.net --recv-key_
↪A2CFB51FA275A7286234E7B24D17C995CD9775F2
user@webserver:~$ gpg2 --trusted-key 4D17C995CD9775F2 --verify certbot-auto.asc /usr/
↪local/bin/certbot-auto
```

The output of the last command should look something like:

```
gpg: Signature made Wed 02 May 2018 05:29:12 AM IST
gpg:
gpg: using RSA key A2CFB51FA275A7286234E7B24D17C995CD9775F2
gpg: key 4D17C995CD9775F2 marked as ultimately trusted
gpg: checking the trustdb
gpg: marginals needed: 3 completes needed: 1 trust model: pgp
gpg: depth: 0 valid: 2 signed: 2 trust: 0-, 0q, 0n, 0m, 0f, 2u
gpg: depth: 1 valid: 2 signed: 0 trust: 2-, 0q, 0n, 0m, 0f, 0u
gpg: next trustdb check due at 2027-11-22
gpg: Good signature from "Let's Encrypt Client Team <letsencrypt-client@eff.org>"_
↪[ultimate]
```

(continues on next page)

(continued from previous page)

The `certbot-auto` command updates to the latest client release automatically. Since `certbot-auto` is a wrapper to `certbot`, it accepts exactly the same command line flags and arguments. For more information, see [Certbot command-line options](#).

For full command line help, you can type:

```
/usr/local/bin/certbot-auto --help all
```

3.3.2 Problems with Python virtual environment

On a low memory system such as VPS with less than 512MB of RAM, the required dependencies of Certbot will fail to build. This can be identified if the pip outputs contains something like `internal compiler error: Killed (program cc1)`. You can workaround this restriction by creating a temporary swapfile:

```
user@webserver:~$ sudo fallocation -l 1G /tmp/swapfile
user@webserver:~$ sudo chmod 600 /tmp/swapfile
user@webserver:~$ sudo mkswap /tmp/swapfile
user@webserver:~$ sudo swapon /tmp/swapfile
```

Disable and remove the swapfile once the virtual environment is constructed:

```
user@webserver:~$ sudo swapoff /tmp/swapfile
user@webserver:~$ sudo rm /tmp/swapfile
```

3.3.3 Running with Docker

[Docker](#) is an amazingly simple and quick way to obtain a certificate. However, this mode of operation is unable to install certificates or configure your webserver, because our installer plugins cannot reach your webserver from inside the Docker container.

Most users should use the operating system packages (see instructions at certbot.eff.org) or, as a fallback, `certbot-auto`. You should only use Docker if you are sure you know what you are doing and have a good reason to do so.

You should definitely read the [Where are my certificates?](#) section, in order to know how to manage the certs manually. [Our ciphersuites page](#) provides some information about recommended ciphersuites. If none of these make much sense to you, you should definitely use the `certbot-auto` method, which enables you to use installer plugins that cover both of those hard topics.

If you're still not convinced and have decided to use this method, from the server that the domain you're requesting a certificate for resolves to, [install Docker](#), then issue a command like the one found below. If you are using Certbot with the [Standalone](#) plugin, you will need to make the port it uses accessible from outside of the container by including something like `-p 80:80` or `-p 443:443` on the command line before `certbot/certbot`.

```
sudo docker run -it --rm --name certbot \
    -v "/etc/letsencrypt:/etc/letsencrypt" \
    -v "/var/lib/letsencrypt:/var/lib/letsencrypt" \
    certbot/certbot certonly
```

Running Certbot with the `certonly` command will obtain a certificate and place it in the directory `/etc/letsencrypt/live` on your system. Because Certonly cannot install the certificate from within Docker, you must install the certificate manually according to the procedure recommended by the provider of your webserver.

There are also Docker images for each of Certbot's DNS plugins available at <https://hub.docker.com/u/certbot> which automate doing domain validation over DNS for popular providers. To use one, just replace `certbot/certbot` in the command above with the name of the image you want to use. For example, to use Certbot's plugin for Amazon Route 53, you'd use `certbot/dns-route53`. You may also need to add flags to Certbot and/or mount additional directories to provide access to your DNS API credentials as specified in the [DNS plugin documentation](#). If you would like to obtain a wildcard certificate from Let's Encrypt's ACMEv2 server, you'll need to include `--server https://acme-v02.api.letsencrypt.org/directory` on the command line as well.

For more information about the layout of the `/etc/letsencrypt` directory, see [Where are my certificates?](#).

3.3.4 Operating System Packages

Arch Linux

```
sudo pacman -S certbot
```

Debian

If you run Debian Buster or Debian testing/Sid, you can easily install certbot packages through commands like:

```
sudo apt-get update
sudo apt-get install certbot
```

If you run Debian Stretch, we recommend you use the packages in Debian backports repository. First you'll have to follow the instructions at <https://backports.debian.org/Instructions/> to enable the Stretch backports repo, if you have not already done so. Then run:

```
sudo apt-get install certbot -t stretch-backports
```

In all of these cases, there also packages available to help Certbot integrate with Apache, nginx, or various DNS services. If you are using Apache or nginx, we strongly recommend that you install the `python-certbot-apache` or `python-certbot-nginx` package so that Certbot can fully automate HTTPS configuration for your server. A full list of these packages can be found through a command like:

```
apt search 'python-certbot*'
```

They can be installed by running the same installation command above but replacing `certbot` with the name of the desired package.

There are no Certbot packages available for Debian Jessie and Jessie users should instead use [certbot-auto](#).

Ubuntu

If you run Ubuntu Trusty, Xenial, or Bionic, certbot is available through the official PPA, that can be installed as followed:

```
sudo apt-get update
sudo apt-get install software-properties-common
sudo add-apt-repository universe
sudo add-apt-repository ppa:certbot/certbot
sudo apt-get update
```

Then, certbot can be installed using:

```
sudo apt-get install certbot
```

Optionally to install the Certbot Apache plugin, you can use:

```
sudo apt-get install python-certbot-apache
```

Fedora

```
sudo dnf install certbot python2-certbot-apache
```

FreeBSD

- Port: `cd /usr/ports/security/py-certbot && make install clean`
- Package: `pkg install py27-certbot`

Gentoo

The official Certbot client is available in Gentoo Portage. If you want to use the Apache plugin, it has to be installed separately:

```
emerge -av app-crypt/certbot
emerge -av app-crypt/certbot-apache
```

When using the Apache plugin, you will run into a “cannot find an SSLCertificateFile directive” or “cannot find an SSLCertificateKeyFile directive for certificate” error if you’re sporting the default Gentoo `httpd.conf`. You can fix this by commenting out two lines in `/etc/apache2/httpd.conf` as follows:

Change

```
<IfDefine SSL>
LoadModule ssl_module modules/mod_ssl.so
</IfDefine>
```

to

```
#<IfDefine SSL>
LoadModule ssl_module modules/mod_ssl.so
#</IfDefine>
```

For the time being, this is the only way for the Apache plugin to recognise the appropriate directives when installing the certificate. Note: this change is not required for the other plugins.

NetBSD

- Build from source: `cd /usr/pkgsrc/security/py-certbot && make install clean`
- Install pre-compiled package: `pkg_add py27-certbot`

OpenBSD

- Port: `cd /usr/ports/security/letsencrypt/client && make install clean`
- Package: `pkg_add letsencrypt`

Other Operating Systems

OS packaging is an ongoing effort. If you’d like to package Certbot for your distribution of choice please have a look at the [Packaging Guide](#).

3.3.5 Installing from source

Installation from source is only supported for developers and the whole process is described in the [Developer Guide](#).

Warning: Please do **not** use `python setup.py install`, `python pip install .`, or `easy_install ..`. Please do **not** attempt the installation commands as superuser/root and/or without virtual environment, e.g. `sudo python setup.py install`, `sudo pip install`, `sudo ./venv/bin/..`. These modes of operation might corrupt your operating system and are **not supported** by the Certbot team!

USER GUIDE**Table of Contents**

- *Certbot Commands*
- *Getting certificates (and choosing plugins)*
 - *Apache*
 - *Webroot*
 - *Nginx*
 - *Standalone*
 - *DNS Plugins*
 - *Manual*
 - *Combining plugins*
 - *Third-party plugins*
- *Managing certificates*
 - *Re-creating and Updating Existing Certificates*
- *Changing a Certificate's Domains*
 - *Revoking certificates*
 - *Renewing certificates*
 - *Modifying the Renewal Configuration File*
 - *Automated Renewals*
- *Where are my certificates?*
- *Pre and Post Validation Hooks*
- *Changing the ACME Server*
- *Lock Files*
- *Configuration file*
- *Log Rotation*
- *Certbot command-line options*
- *Getting help*

4.1 Certbot Commands

Certbot uses a number of different commands (also referred to as “subcommands”) to request specific actions such as obtaining, renewing, or revoking certificates. The most important and commonly-used commands will be discussed throughout this document; an exhaustive list also appears near the end of the document.

The `certbot` script on your web server might be named `letsencrypt` if your system uses an older package, or `certbot-auto` if you used an alternate installation method. Throughout the docs, whenever you see `certbot`, swap in the correct name as needed.

4.2 Getting certificates (and choosing plugins)

The Certbot client supports two types of plugins for obtaining and installing certificates: authenticators and installers.

Authenticators are plugins used with the `certonly` command to obtain a certificate. The authenticator validates that you control the domain(s) you are requesting a certificate for, obtains a certificate for the specified domain(s), and places the certificate in the `/etc/letsencrypt` directory on your machine. The authenticator does not install the certificate (it does not edit any of your server’s configuration files to serve the obtained certificate). If you specify multiple domains to authenticate, they will all be listed in a single certificate. To obtain multiple separate certificates you will need to run Certbot multiple times.

Installers are Plugins used with the `install` command to install a certificate. These plugins can modify your web-server’s configuration to serve your website over HTTPS using certificates obtained by certbot.

Plugins that do both can be used with the `certbot run` command, which is the default when no command is specified. The `run` subcommand can also be used to specify a *combination* of distinct authenticator and installer plugins.

| Plugin | Auth | Inst | Notes | Challenge types (and port) |
|---|------|------|--|-------------------------------|
| <i>apache</i> | Y | Y | Automates obtaining and installing a certificate with Apache. | http-01 (80) |
| <i>nginx</i> | Y | Y | Automates obtaining and installing a certificate with Nginx. | http-01 (80) |
| <i>webroot</i> | Y | N | Obtains a certificate by writing to the webroot directory of an already running webserver. | http-01 (80) |
| <i>standalone</i> | Y | N | Uses a “standalone” webserver to obtain a certificate. Requires port 80 to be available. This is useful on systems with no webserver, or when direct integration with the local webserver is not supported or not desired. | http-01 (80) |
| <i>DNS plugins</i> | Y | N | This category of plugins automates obtaining a certificate by modifying DNS records to prove you have control over a domain. Doing domain validation in this way is | dns-01 (53) |
| 4.2. Getting certificates (and choosing plugins) | | | the only way to obtain wildcard certificates from Let’s | 13 |

Under the hood, plugins use one of several ACME protocol [challenges](#) to prove you control a domain. The options are [http-01](#) (which uses port 80) and [dns-01](#) (requiring configuration of a DNS server on port 53, though that's often not the same machine as your webserver). A few plugins support more than one challenge type, in which case you can choose one with `--preferred-challenges`.

There are also many [third-party-plugins](#) available. Below we describe in more detail the circumstances in which each plugin can be used, and how to use it.

4.2.1 Apache

The Apache plugin currently [supports](#) modern OSes based on Debian, Fedora, SUSE, Gentoo and Darwin. This automates both obtaining *and* installing certificates on an Apache webserver. To specify this plugin on the command line, simply include `--apache`.

4.2.2 Webroot

If you're running a local webserver for which you have the ability to modify the content being served, and you'd prefer not to stop the webserver during the certificate issuance process, you can use the webroot plugin to obtain a certificate by including `certonly` and `--webroot` on the command line. In addition, you'll need to specify `--webroot-path` or `-w` with the top-level directory ("web root") containing the files served by your webserver. For example, `--webroot-path /var/www/html` or `--webroot-path /usr/share/nginx/html` are two common webroot paths.

If you're getting a certificate for many domains at once, the plugin needs to know where each domain's files are served from, which could potentially be a separate directory for each domain. When requesting a certificate for multiple domains, each domain will use the most recently specified `--webroot-path`. So, for instance,

```
certbot certonly --webroot -w /var/www/example -d www.example.com -d example.com -w /
↳ var/www/other -d other.example.net -d another.other.example.net
```

would obtain a single certificate for all of those names, using the `/var/www/example` webroot directory for the first two, and `/var/www/other` for the second two.

The webroot plugin works by creating a temporary file for each of your requested domains in `${webroot-path}/.well-known/acme-challenge`. Then the Let's Encrypt validation server makes HTTP requests to validate that the DNS for each requested domain resolves to the server running certbot. An example request made to your web server would look like:

```
66.133.109.36 - - [05/Jan/2016:20:11:24 -0500] "GET /.well-known/acme-challenge/
↳ HGr8U1IeTW4kY_Z6UIyaakzOkyQgPr_7ArlLgtZE8SX HTTP/1.1" 200 87 "-" "Mozilla/5.0
↳ (compatible; Let's Encrypt validation server; +https://www.letsencrypt.org)"
```

Note that to use the webroot plugin, your server must be configured to serve files from hidden directories. If `/.well-known` is treated specially by your webserver configuration, you might need to modify the configuration to ensure that files inside `/.well-known/acme-challenge` are served by the webserver.

4.2.3 Nginx

The Nginx plugin should work for most configurations. We recommend backing up Nginx configurations before using it (though you can also revert changes to configurations with `certbot --nginx rollback`). You can use it by providing the `--nginx` flag on the commandline.

```
certbot --nginx
```

4.2.4 Standalone

Use standalone mode to obtain a certificate if you don't want to use (or don't currently have) existing server software. The standalone plugin does not rely on any other server software running on the machine where you obtain the certificate.

To obtain a certificate using a “standalone” webserver, you can use the standalone plugin by including `certonly` and `--standalone` on the command line. This plugin needs to bind to port 80 in order to perform domain validation, so you may need to stop your existing webserver.

It must still be possible for your machine to accept inbound connections from the Internet on the specified port using each requested domain name.

By default, Certbot first attempts to bind to the port for all interfaces using IPv6 and then bind to that port using IPv4; Certbot continues so long as at least one bind succeeds. On most Linux systems, IPv4 traffic will be routed to the bound IPv6 port and the failure during the second bind is expected.

Use `--<challenge-type>-address` to explicitly tell Certbot which interface (and protocol) to bind.

4.2.5 DNS Plugins

If you'd like to obtain a wildcard certificate from Let's Encrypt or run `certbot` on a machine other than your target webserver, you can use one of Certbot's DNS plugins.

These plugins are not included in a default Certbot installation and must be installed separately. While the DNS plugins cannot currently be used with `certbot-auto`, they are available in many OS package managers and as Docker images. Visit <https://certbot.eff.org> to learn the best way to use the DNS plugins on your system.

Once installed, you can find documentation on how to use each plugin at:

- [certbot-dns-cloudflare](#)
- [certbot-dns-cloudxns](#)
- [certbot-dns-digitalocean](#)
- [certbot-dns-dnssimple](#)
- [certbot-dns-dnsmadeeasy](#)
- [certbot-dns-google](#)
- [certbot-dns-linode](#)
- [certbot-dns-luadns](#)
- [certbot-dns-nsone](#)
- [certbot-dns-ovh](#)
- [certbot-dns-rfc2136](#)
- [certbot-dns-route53](#)

4.2.6 Manual

If you'd like to obtain a certificate running `certbot` on a machine other than your target webserver or perform the steps for domain validation yourself, you can use the manual plugin. While hidden from the UI, you can use the plugin to obtain a certificate by specifying `certonly` and `--manual` on the command line. This requires you to copy and paste commands into another terminal session, which may be on a different computer.

The manual plugin can use either the `http` or the `dns` challenge. You can use the `--preferred-challenges` option to choose the challenge of your preference.

The `http` challenge will ask you to place a file with a specific name and specific content in the `/.well-known/acme-challenge/` directory directly in the top-level directory (“web root”) containing the files served by your webserver. In essence it’s the same as the *webroot* plugin, but not automated.

When using the `dns` challenge, `certbot` will ask you to place a TXT DNS record with specific contents under the domain name consisting of the hostname for which you want a certificate issued, prepended by `_acme-challenge`.

For example, for the domain `example.com`, a zone file entry would look like:

```
_acme-challenge.example.com. 300 IN TXT "gfj9Xq...Rg85nM"
```

Additionally you can specify scripts to prepare for validation and perform the authentication procedure and/or clean up after it by using the `--manual-auth-hook` and `--manual-cleanup-hook` flags. This is described in more depth in the *hooks* section.

4.2.7 Combining plugins

Sometimes you may want to specify a combination of distinct authenticator and installer plugins. To do so, specify the authenticator plugin with `--authenticator` or `-a` and the installer plugin with `--installer` or `-i`.

For instance, you could create a certificate using the *webroot* plugin for authentication and the *apache* plugin for installation.

```
certbot run -a webroot -i apache -w /var/www/html -d example.com
```

Or you could create a certificate using the *manual* plugin for authentication and the *nginx* plugin for installation. (Note that this certificate cannot be renewed automatically.)

```
certbot run -a manual -i nginx -d example.com
```

4.2.8 Third-party plugins

There are also a number of third-party plugins for the client, provided by other developers. Many are beta/experimental, but some are already in widespread use:

| Plugin | Auth | Inst | Notes |
|-----------------------|------|------|---|
| <i>haproxy</i> | Y | Y | Integration with the HAProxy load balancer |
| <i>s3front</i> | Y | Y | Integration with Amazon CloudFront distribution of S3 buckets |
| <i>gandi</i> | Y | N | Obtain certificates via the Gandi LiveDNS API |
| <i>varnish</i> | Y | N | Obtain certificates via a Varnish server |
| <i>external</i> | Y | N | A plugin for convenient scripting (See also ticket 2782) |
| <i>pritunl</i> | N | Y | Install certificates in pritunl distributed OpenVPN servers |
| <i>proxmox</i> | N | Y | Install certificates in Proxmox Virtualization servers |
| <i>dns-standalone</i> | Y | N | Obtain certificates via an integrated DNS server |
| <i>dns-ispconfig</i> | Y | N | DNS Authentication using ISPConfig as DNS server |

If you’re interested, you can also *write your own plugin*.

4.3 Managing certificates

To view a list of the certificates Certbot knows about, run the `certificates` subcommand:

```
certbot certificates
```

This returns information in the following format:

```
Found the following certs:
Certificate Name: example.com
Domains: example.com, www.example.com
Expiry Date: 2017-02-19 19:53:00+00:00 (VALID: 30 days)
Certificate Path: /etc/letsencrypt/live/example.com/fullchain.pem
Private Key Path: /etc/letsencrypt/live/example.com/privkey.pem
```

Certificate Name shows the name of the certificate. Pass this name using the `--cert-name` flag to specify a particular certificate for the `run`, `certonly`, `certificates`, `renew`, and `delete` commands. Example:

```
certbot certonly --cert-name example.com
```

4.3.1 Re-creating and Updating Existing Certificates

You can use `certonly` or run subcommands to request the creation of a single new certificate even if you already have an existing certificate with some of the same domain names.

If a certificate is requested with `run` or `certonly` specifying a certificate name that already exists, Certbot updates the existing certificate. Otherwise a new certificate is created and assigned the specified name.

The `--force-renewal`, `--duplicate`, and `--expand` options control Certbot's behavior when re-creating a certificate with the same name as an existing certificate. If you don't specify a requested behavior, Certbot may ask you what you intended.

`--force-renewal` tells Certbot to request a new certificate with the same domains as an existing certificate. Each domain must be explicitly specified via `-d`. If successful, this certificate is saved alongside the earlier one and symbolic links (the "live" reference) will be updated to point to the new certificate. This is a valid method of renewing a specific individual certificate.

`--duplicate` tells Certbot to create a separate, unrelated certificate with the same domains as an existing certificate. This certificate is saved completely separately from the prior one. Most users will not need to issue this command in normal circumstances.

`--expand` tells Certbot to update an existing certificate with a new certificate that contains all of the old domains and one or more additional new domains. With the `--expand` option, use the `-d` option to specify all existing domains and one or more new domains.

Example:

```
certbot --expand -d existing.com,example.com,newdomain.com
```

If you prefer, you can specify the domains individually like this:

```
certbot --expand -d existing.com -d example.com -d newdomain.com
```

Consider using `--cert-name` instead of `--expand`, as it gives more control over which certificate is modified and it lets you remove domains as well as adding them.

`--allow-subset-of-names` tells Certbot to continue with certificate generation if only some of the specified domain authorizations can be obtained. This may be useful if some domains specified in a certificate no longer point at this system.

Whenever you obtain a new certificate in any of these ways, the new certificate exists alongside any previously obtained certificates, whether or not the previous certificates have expired. The generation of a new certificate counts against several rate limits that are intended to prevent abuse of the ACME protocol, as described [here](#).

4.4 Changing a Certificate's Domains

The `--cert-name` flag can also be used to modify the domains a certificate contains, by specifying new domains using the `-d` or `--domains` flag. If certificate `example.com` previously contained `example.com` and `www.example.com`, it can be modified to only contain `example.com` by specifying only `example.com` with the `-d` or `--domains` flag. Example:

```
certbot certonly --cert-name example.com -d example.com
```

The same format can be used to expand the set of domains a certificate contains, or to replace that set entirely:

```
certbot certonly --cert-name example.com -d example.org,www.example.org
```

4.4.1 Revoking certificates

If your account key has been compromised or you otherwise need to revoke a certificate, use the `revoke` command to do so. Note that the `revoke` command takes the certificate path (ending in `cert.pem`), not a certificate name or domain. Example:

```
certbot revoke --cert-path /etc/letsencrypt/live/CERTNAME/cert.pem
```

You can also specify the reason for revoking your certificate by using the `reason` flag. Reasons include unspecified which is the default, as well as `keycompromise`, `affiliationchanged`, `superseded`, and `cessationofoperation`:

```
certbot revoke --cert-path /etc/letsencrypt/live/CERTNAME/cert.pem --reason_
↪keycompromise
```

Additionally, if a certificate is a test certificate obtained via the `--staging` or `--test-cert` flag, that flag must be passed to the `revoke` subcommand. Once a certificate is revoked (or for other certificate management tasks), all of a certificate's relevant files can be removed from the system with the `delete` subcommand:

```
certbot delete --cert-name example.com
```

Note: If you don't use `delete` to remove the certificate completely, it will be renewed automatically at the next renewal event.

Note: Revoking a certificate will have no effect on the rate limit imposed by the Let's Encrypt server.

4.4.2 Renewing certificates

Note: Let's Encrypt CA issues short-lived certificates (90 days). Make sure you renew the certificates at least once in 3 months.

See also:

Many of the certbot clients obtained through a distribution come with automatic renewal out of the box, such as Debian and Ubuntu versions installed through `apt`, CentOS/RHEL 7 through EPEL, etc. See [Automated Renewals](#) for more details.

As of version 0.10.0, Certbot supports a `renew` action to check all installed certificates for impending expiry and attempt to renew them. The simplest form is simply

```
certbot renew
```

This command attempts to renew any previously-obtained certificates that expire in less than 30 days. The same plugin and options that were used at the time the certificate was originally issued will be used for the renewal attempt, unless you specify other plugins or options. Unlike `certonly`, `renew` acts on multiple certificates and always takes into account whether each one is near expiry. Because of this, `renew` is suitable (and designed) for automated use, to allow your system to automatically renew each certificate when appropriate. Since `renew` only renews certificates that are near expiry it can be run as frequently as you want - since it will usually take no action.

The `renew` command includes hooks for running commands or scripts before or after a certificate is renewed. For example, if you have a single certificate obtained using the [standalone](#) plugin, you might need to stop the webserver before renewing so `standalone` can bind to the necessary ports, and then restart it after the plugin is finished. Example:

```
certbot renew --pre-hook "service nginx stop" --post-hook "service nginx start"
```

If a hook exits with a non-zero exit code, the error will be printed to `stderr` but renewal will be attempted anyway. A failing hook doesn't directly cause Certbot to exit with a non-zero exit code, but since Certbot exits with a non-zero exit code when renewals fail, a failed hook causing renewal failures will indirectly result in a non-zero exit code. Hooks will only be run if a certificate is due for renewal, so you can run the above command frequently without unnecessarily stopping your webserver.

When Certbot detects that a certificate is due for renewal, `--pre-hook` and `--post-hook` hooks run before and after each attempt to renew it. If you want your hook to run only after a successful renewal, use `--deploy-hook` in a command like this.

```
certbot renew --deploy-hook /path/to/deploy-hook-script
```

For example, if you have a daemon that does not read its certificates as the root user, a deploy hook like this can copy them to the correct location and apply appropriate file permissions.

```
/path/to/deploy-hook-script
```

```
#!/bin/sh

set -e

for domain in $RENEWED_DOMAINS; do
    case $domain in
        example.com)
            daemon_cert_root=/etc/some-daemon/certs

            # Make sure the certificate and private key files are
            # never world readable, even just for an instant while
            # we're copying them into daemon_cert_root.
```

(continues on next page)

(continued from previous page)

```

umask 077

cp "$RENEWED_LINEAGE/fullchain.pem" "$daemon_cert_root/$domain.cert"
cp "$RENEWED_LINEAGE/privkey.pem" "$daemon_cert_root/$domain.key"

# Apply the proper file ownership and permissions for
# the daemon to read its certificate and key.
chown some-daemon "$daemon_cert_root/$domain.cert" \
    "$daemon_cert_root/$domain.key"
chmod 400 "$daemon_cert_root/$domain.cert" \
    "$daemon_cert_root/$domain.key"

service some-daemon restart >/dev/null
;;
done
esac

```

You can also specify hooks by placing files in subdirectories of Certbot's configuration directory. Assuming your configuration directory is `/etc/letsencrypt`, any executable files found in `/etc/letsencrypt/renewal-hooks/pre`, `/etc/letsencrypt/renewal-hooks/deploy`, and `/etc/letsencrypt/renewal-hooks/post` will be run as pre, deploy, and post hooks respectively when any certificate is renewed with the `renew` subcommand. These hooks are run in alphabetical order and are not run for other subcommands. (The order the hooks are run is determined by the byte value of the characters in their filenames and is not dependent on your locale.)

Hooks specified in the command line, *configuration file*, or *renewal configuration files* are run as usual after running all hooks in these directories. One minor exception to this is if a hook specified elsewhere is simply the path to an executable file in the hook directory of the same type (e.g. your pre-hook is the path to an executable in `/etc/letsencrypt/renewal-hooks/pre`), the file is not run a second time. You can stop Certbot from automatically running executables found in these directories by including `--no-directory-hooks` on the command line.

More information about hooks can be found by running `certbot --help renew`.

If you're sure that this command executes successfully without human intervention, you can add the command to `crontab` (since certificates are only renewed when they're determined to be near expiry, the command can run on a regular basis, like every week or every day). In that case, you are likely to want to use the `-q` or `--quiet` quiet flag to silence all output except errors.

If you are manually renewing all of your certificates, the `--force-renewal` flag may be helpful; it causes the expiration time of the certificate(s) to be ignored when considering renewal, and attempts to renew each and every installed certificate regardless of its age. (This form is not appropriate to run daily because each certificate will be renewed every day, which will quickly run into the certificate authority rate limit.)

Note that options provided to `certbot renew` will apply to *every* certificate for which renewal is attempted; for example, `certbot renew --rsa-key-size 4096` would try to replace every near-expiry certificate with an equivalent certificate using a 4096-bit RSA public key. If a certificate is successfully renewed using specified options, those options will be saved and used for future renewals of that certificate.

An alternative form that provides for more fine-grained control over the renewal process (while renewing specified certificates one at a time), is `certbot certonly` with the complete set of subject domains of a specific certificate specified via `-d` flags. You may also want to include the `-n` or `--noninteractive` flag to prevent blocking on user input (which is useful when running the command from cron).

```
certbot certonly -n -d example.com -d www.example.com
```

All of the domains covered by the certificate must be specified in this case in order to renew and replace the old certificate rather than obtaining a new one; don't forget any `www.` domains! Specifying a subset of the domains creates a new, separate certificate containing only those domains, rather than replacing the original certificate. When

run with a set of domains corresponding to an existing certificate, the `certonly` command attempts to renew that specific certificate.

Please note that the CA will send notification emails to the address you provide if you do not renew certificates that are about to expire.

Certbot is working hard to improve the renewal process, and we apologize for any inconvenience you encounter in integrating these commands into your individual environment.

Note: `certbot renew` exit status will only be 1 if a renewal attempt failed. This means `certbot renew` exit status will be 0 if no certificate needs to be updated. If you write a custom script and expect to run a command only after a certificate was actually renewed you will need to use the `--deploy-hook` since the exit status will be 0 both on successful renewal and when renewal is not necessary.

4.4.3 Modifying the Renewal Configuration File

When a certificate is issued, by default Certbot creates a renewal configuration file that tracks the options that were selected when Certbot was run. This allows Certbot to use those same options again when it comes time for renewal. These renewal configuration files are located at `/etc/letsencrypt/renewal/CERTNAME`.

For advanced certificate management tasks, it is possible to manually modify the certificate's renewal configuration file, but this is discouraged since it can easily break Certbot's ability to renew your certificates. If you choose to modify the renewal configuration file we advise you to test its validity with the `certbot renew --dry-run` command.

Warning: Modifying any files in `/etc/letsencrypt` can damage them so Certbot can no longer properly manage its certificates, and we do not recommend doing so.

For most tasks, it is safest to limit yourself to pointing symlinks at the files there, or using `--deploy-hook` to copy / make new files based upon those files, if your operational situation requires it (for instance, combining certificates and keys in different way, or having copies of things with different specific permissions that are demanded by other programs).

If the contents of `/etc/letsencrypt/archive/CERTNAME` are moved to a new folder, first specify the new folder's name in the renewal configuration file, then run `certbot update_symlinks` to point the symlinks in `/etc/letsencrypt/live/CERTNAME` to the new folder.

If you would like the live certificate files whose symlink location Certbot updates on each run to reside in a different location, first move them to that location, then specify the full path of each of the four files in the renewal configuration file. Since the symlinks are relative links, you must follow this with an invocation of `certbot update_symlinks`.

For example, say that a certificate's renewal configuration file previously contained the following directives:

```
archive_dir = /etc/letsencrypt/archive/example.com
cert = /etc/letsencrypt/live/example.com/cert.pem
privkey = /etc/letsencrypt/live/example.com/privkey.pem
chain = /etc/letsencrypt/live/example.com/chain.pem
fullchain = /etc/letsencrypt/live/example.com/fullchain.pem
```

The following commands could be used to specify where these files are located:

```
mv /etc/letsencrypt/archive/example.com /home/user/me/certbot/example_archive
sed -i 's,/etc/letsencrypt/archive/example.com,/home/user/me/certbot/example_archive,
↪' /etc/letsencrypt/renewal/example.com.conf
```

(continues on next page)

(continued from previous page)

```
mv /etc/letsencrypt/live/example.com/*.pem /home/user/me/certbot/
sed -i 's,/etc/letsencrypt/live/example.com,/home/user/me/certbot,g' /etc/letsencrypt/
↪renewal/example.com.conf
certbot update_symlinks
```

4.4.4 Automated Renewals

Many Linux distributions provide automated renewal when you use the packages installed through their system package manager. The following table is an *incomplete* list of distributions which do so, as well as their methods for doing so.

If you are not sure whether or not your system has this already automated, refer to your distribution's documentation, or check your system's crontab (typically in `/etc/crontab/` and `/etc/cron.*/*` and systemd timers (`systemctl list-timers`).

Table 1: Distributions with Automated Renewal

| Distribution Name | Distribution Version | Automation Method |
|-------------------|----------------------|-------------------|
| CentOS | EPEL 7 | systemd |
| Debian | jessie | cron, systemd |
| Debian | stretch | cron, systemd |
| Debian | testing/sid | cron, systemd |
| Fedora | 26 | systemd |
| Fedora | 27 | systemd |
| RHEL | EPEL 7 | systemd |
| Ubuntu | 17.10 | cron, systemd |
| Ubuntu | certbot PPA | cron, systemd |

4.5 Where are my certificates?

All generated keys and issued certificates can be found in `/etc/letsencrypt/live/$domain`. In the case of creating a SAN certificate with multiple alternative names, `$domain` is the first domain passed in via `-d` parameter. Rather than copying, please point your (web) server configuration directly to those files (or create symlinks). During the *renewal*, `/etc/letsencrypt/live` is updated with the latest necessary files.

Note: `/etc/letsencrypt/archive` and `/etc/letsencrypt/keys` contain all previous keys and certificates, while `/etc/letsencrypt/live` symlinks to the latest versions.

The following files are available:

privkey.pem Private key for the certificate.

Warning: This **must be kept secret at all times!** Never share it with anyone, including Certbot developers. You cannot put it into a safe, however - your server still needs to access this file in order for SSL/TLS to work.

Note: As of Certbot version 0.29.0, private keys for new certificate default to `0600`. Any changes to the group

mode or group owner (gid) of this file will be preserved on renewals.

This is what Apache needs for `SSLCertificateKeyFile`, and Nginx for `ssl_certificate_key`.

fullchain.pem All certificates, **including** server certificate (aka leaf certificate or end-entity certificate). The server certificate is the first one in this file, followed by any intermediates.

This is what Apache $\geq 2.4.8$ needs for `SSLCertificateFile`, and what Nginx needs for `ssl_certificate`.

cert.pem and chain.pem (less common) `cert.pem` contains the server certificate by itself, and `chain.pem` contains the additional intermediate certificate or certificates that web browsers will need in order to validate the server certificate. If you provide one of these files to your web server, you **must** provide both of them, or some browsers will show “This Connection is Untrusted” errors for your site, [some of the time](#).

Apache $< 2.4.8$ needs these for `SSLCertificateFile` and `SSLCertificateChainFile`, respectively.

If you’re using OCSP stapling with Nginx $\geq 1.3.7$, `chain.pem` should be provided as the `ssl_trusted_certificate` to validate OCSP responses.

Note: All files are PEM-encoded. If you need other format, such as DER or PFX, then you could convert using `openssl`. You can automate that with `--deploy-hook` if you’re using automatic [renewal](#).

4.6 Pre and Post Validation Hooks

Certbot allows for the specification of pre and post validation hooks when run in manual mode. The flags to specify these scripts are `--manual-auth-hook` and `--manual-cleanup-hook` respectively and can be used as follows:

```
certbot certonly --manual --manual-auth-hook /path/to/http/authenticator.sh --manual-
↪ cleanup-hook /path/to/http/cleanup.sh -d secure.example.com
```

This will run the `authenticator.sh` script, attempt the validation, and then run the `cleanup.sh` script. Additionally certbot will pass relevant environment variables to these scripts:

- `CERTBOT_DOMAIN`: The domain being authenticated
- `CERTBOT_VALIDATION`: The validation string (HTTP-01 and DNS-01 only)
- `CERTBOT_TOKEN`: Resource name part of the HTTP-01 challenge (HTTP-01 only)

Additionally for cleanup:

- `CERTBOT_AUTH_OUTPUT`: Whatever the auth script wrote to stdout

Example usage for HTTP-01:

```
certbot certonly --manual --preferred-challenges=http --manual-auth-hook /path/to/
↪ http/authenticator.sh --manual-cleanup-hook /path/to/http/cleanup.sh -d secure.
↪ example.com
```

`/path/to/http/authenticator.sh`

```
#!/bin/bash
echo $CERTBOT_VALIDATION > /var/www/htdocs/.well-known/acme-challenge/$CERTBOT_TOKEN
```

`/path/to/http/cleanup.sh`

```
#!/bin/bash
rm -f /var/www/htdocs/.well-known/acme-challenge/$CERTBOT_TOKEN
```

Example usage for DNS-01 (Cloudflare API v4) (for example purposes only, do not use as-is)

```
certbot certonly --manual --preferred-challenges=dns --manual-auth-hook /path/to/dns/
↪ authenticator.sh --manual-cleanup-hook /path/to/dns/cleanup.sh -d secure.example.com
```

/path/to/dns/authenticator.sh

```
#!/bin/bash

# Get your API key from https://www.cloudflare.com/a/account/my-account
API_KEY="your-api-key"
EMAIL="your.email@example.com"

# Strip only the top domain to get the zone id
DOMAIN=$(expr match "$CERTBOT_DOMAIN" '.*\.(.*\..*)')

# Get the Cloudflare zone id
ZONE_EXTRA_PARAMS="status=active&page=1&per_page=20&order=status&direction=desc&
↪ match=all"
ZONE_ID=$(curl -s -X GET "https://api.cloudflare.com/client/v4/zones?name=$DOMAIN&
↪ $ZONE_EXTRA_PARAMS" \
    -H "X-Auth-Email: $EMAIL" \
    -H "X-Auth-Key: $API_KEY" \
    -H "Content-Type: application/json" | python -c "import sys,json;print(json.
↪ load(sys.stdin)['result'][0]['id'])")

# Create TXT record
CREATE_DOMAIN="_acme-challenge.$CERTBOT_DOMAIN"
RECORD_ID=$(curl -s -X POST "https://api.cloudflare.com/client/v4/zones/$ZONE_ID/dns_
↪ records" \
    -H "X-Auth-Email: $EMAIL" \
    -H "X-Auth-Key: $API_KEY" \
    -H "Content-Type: application/json" \
    --data '{"type":"TXT","name":"'$_CREATE_DOMAIN'", "content":"'$_CERTBOT_
↪ VALIDATION'", "ttl":120}' \
    | python -c "import sys,json;print(json.load(sys.stdin)['result']['id'])
↪ ")

# Save info for cleanup
if [ ! -d /tmp/CERTBOT_$CERTBOT_DOMAIN ];then
    mkdir -m 0700 /tmp/CERTBOT_$CERTBOT_DOMAIN
fi
echo $ZONE_ID > /tmp/CERTBOT_$CERTBOT_DOMAIN/ZONE_ID
echo $RECORD_ID > /tmp/CERTBOT_$CERTBOT_DOMAIN/RECORD_ID

# Sleep to make sure the change has time to propagate over to DNS
sleep 25
```

/path/to/dns/cleanup.sh

```
#!/bin/bash

# Get your API key from https://www.cloudflare.com/a/account/my-account
API_KEY="your-api-key"
EMAIL="your.email@example.com"
```

(continues on next page)

(continued from previous page)

```

if [ -f /tmp/CERTBOT_${CERTBOT_DOMAIN}/ZONE_ID ]; then
    ZONE_ID=$(cat /tmp/CERTBOT_${CERTBOT_DOMAIN}/ZONE_ID)
    rm -f /tmp/CERTBOT_${CERTBOT_DOMAIN}/ZONE_ID
fi

if [ -f /tmp/CERTBOT_${CERTBOT_DOMAIN}/RECORD_ID ]; then
    RECORD_ID=$(cat /tmp/CERTBOT_${CERTBOT_DOMAIN}/RECORD_ID)
    rm -f /tmp/CERTBOT_${CERTBOT_DOMAIN}/RECORD_ID
fi

# Remove the challenge TXT record from the zone
if [ -n "${ZONE_ID}" ]; then
    if [ -n "${RECORD_ID}" ]; then
        curl -s -X DELETE "https://api.cloudflare.com/client/v4/zones/${ZONE_ID}/dns_
→records/${RECORD_ID}" \
            -H "X-Auth-Email: $EMAIL" \
            -H "X-Auth-Key: $API_KEY" \
            -H "Content-Type: application/json"
    fi
fi

```

4.7 Changing the ACME Server

By default, Certbot uses Let's Encrypt's initial production server at <https://acme-v01.api.letsencrypt.org/>. You can tell Certbot to use a different CA by providing `--server` on the command line or in a *configuration file* with the URL of the server's ACME directory. For example, if you would like to use Let's Encrypt's new ACMEv2 server, you would add `--server https://acme-v02.api.letsencrypt.org/directory` to the command line. Certbot will automatically select which version of the ACME protocol to use based on the contents served at the provided URL.

If you use `--server` to specify an ACME CA that implements a newer version of the spec, you may be able to obtain a certificate for a wildcard domain. Some CAs (such as Let's Encrypt) require that domain validation for wildcard domains must be done through modifications to DNS records which means that the `dns-01` challenge type must be used. To see a list of Certbot plugins that support this challenge type and how to use them, see *plugins*.

4.8 Lock Files

When processing a validation Certbot writes a number of lock files on your system to prevent multiple instances from overwriting each other's changes. This means that by default two instances of Certbot will not be able to run in parallel.

Since the directories used by Certbot are configurable, Certbot will write a lock file for all of the directories it uses. This include Certbot's `--work-dir`, `--logs-dir`, and `--config-dir`. By default these are `/var/lib/letsencrypt`, `/var/log/letsencrypt`, and `/etc/letsencrypt` respectively. Additionally if you are using Certbot with Apache or nginx it will lock the configuration folder for that program, which are typically also in the `/etc` directory.

Note that these lock files will only prevent other instances of Certbot from using those directories, not other processes. If you'd like to run multiple instances of Certbot simultaneously you should specify different directories as the `--work-dir`, `--logs-dir`, and `--config-dir` for each instance of Certbot that you would like to run.

4.9 Configuration file

Certbot accepts a global configuration file that applies its options to all invocations of Certbot. Certificate specific configuration choices should be set in the `.conf` files that can be found in `/etc/letsencrypt/renewal`.

By default no `cli.ini` file is created (though it may exist already if you installed Certbot via a package manager, for instance). After creating one it is possible to specify the location of this configuration file with `certbot --config cli.ini` (or shorter `-c cli.ini`). An example configuration file is shown below:

```
# This is an example of the kind of things you can do in a configuration file.
# All flags used by the client can be configured here. Run Certbot with
# "--help" to learn more about the available options.
#
# Note that these options apply automatically to all use of Certbot for
# obtaining or renewing certificates, so options specific to a single
# certificate on a system with several certificates should not be placed
# here.

# Use a 4096 bit RSA key instead of 2048
rsa-key-size = 4096

# Uncomment and update to register with the specified e-mail address
# email = foo@example.com

# Uncomment to use the standalone authenticator on port 443
# authenticator = standalone

# Uncomment to use the webroot authenticator. Replace webroot-path with the
# path to the public_html / webroot folder being served by your web server.
# authenticator = webroot
# webroot-path = /usr/share/nginx/html
```

By default, the following locations are searched:

- `/etc/letsencrypt/cli.ini`
- `$XDG_CONFIG_HOME/letsencrypt/cli.ini` (or `~/.config/letsencrypt/cli.ini` if `$XDG_CONFIG_HOME` is not set).

Since this configuration file applies to all invocations of certbot it is incorrect to list domains in it. Listing domains in `cli.ini` may prevent renewal from working. Additionally due to how arguments in `cli.ini` are parsed, options which wish to not be set should not be listed. Options set to false will instead be read as being set to true by older versions of Certbot, since they have been listed in the config file.

4.10 Log Rotation

By default certbot stores status logs in `/var/log/letsencrypt`. By default certbot will begin rotating logs once there are 1000 logs in the log directory. Meaning that once 1000 files are in `/var/log/letsencrypt` Certbot will delete the oldest one to make room for new logs. The number of subsequent logs can be changed by passing the desired number to the command line flag `--max-log-backups`.

Note: Some distributions, including Debian and Ubuntu, disable certbot's internal log rotation in favor of a more traditional logrotate script. If you are using a distribution's packages and want to alter the log rotation, check `/etc/logrotate.d/` for a certbot rotation script.

4.11 Certbot command-line options

Certbot supports a lot of command line options. Here's the full list, from `certbot --help all`:

```
usage:
  certbot [SUBCOMMAND] [options] [-d DOMAIN] [-d DOMAIN] ...

Certbot can obtain and install HTTPS/TLS/SSL certificates.  By default,
it will attempt to use a webserver both for obtaining and installing the
certificate. The most common SUBCOMMANDS and flags are:

obtain, install, and renew certificates:
    (default) run          Obtain & install a certificate in your current webserver
    certonly              Obtain or renew a certificate, but do not install it
    renew                 Renew all previously obtained certificates that are near expiry
    enhance               Add security enhancements to your existing configuration
    -d DOMAINS            Comma-separated list of domains to obtain a certificate for

    --apache              Use the Apache plugin for authentication & installation
    --standalone          Run a standalone webserver for authentication
    --nginx               Use the Nginx plugin for authentication & installation
    --webroot             Place files in a server's webroot folder for authentication
    --manual              Obtain certificates interactively, or using shell script hooks

    -n                   Run non-interactively
    --test-cert           Obtain a test certificate from a staging server
    --dry-run             Test "renew" or "certonly" without saving any certificates to disk

manage certificates:
    certificates          Display information about certificates you have from Certbot
    revoke               Revoke a certificate (supply --cert-path or --cert-name)
    delete              Delete a certificate

manage your account:
    register             Create an ACME account
    unregister           Deactivate an ACME account
    update_account       Update an ACME account
    --agree-tos          Agree to the ACME server's Subscriber Agreement
    -m EMAIL             Email address for important account notifications

optional arguments:
    -h, --help            show this help message and exit
    -c CONFIG_FILE, --config CONFIG_FILE
                        path to config file (default: /etc/letsencrypt/cli.ini
                        and ~/.config/letsencrypt/cli.ini)
    -v, --verbose          This flag can be used multiple times to incrementally
                        increase the verbosity of output, e.g. -vvv. (default:
                        -2)
    --max-log-backups MAX_LOG_BACKUPS
                        Specifies the maximum number of backup logs that
                        should be kept by Certbot's built in log rotation.
                        Setting this flag to 0 disables log rotation entirely,
                        causing Certbot to always append to the same log file.
                        (default: 1000)
    -n, --non-interactive, --noninteractive
                        Run without ever asking for user input. This may
                        require additional command line flags; the client will
```

(continues on next page)

(continued from previous page)

```

try to explain which ones are required if it finds one
missing (default: False)
--force-interactive Force Certbot to be interactive even if it detects
it's not being run in a terminal. This flag cannot be
used with the renew subcommand. (default: False)
-d DOMAIN, --domains DOMAIN, --domain DOMAIN
Domain names to apply. For multiple domains you can
use multiple -d flags or enter a comma separated list
of domains as a parameter. The first domain provided
will be the subject CN of the certificate, and all
domains will be Subject Alternative Names on the
certificate. The first domain will also be used in
some software user interfaces and as the file paths
for the certificate and related material unless
otherwise specified or you already have a certificate
with the same name. In the case of a name collision it
will append a number like 0001 to the file path name.
(default: Ask)
--eab-kid EAB_KID Key Identifier for External Account Binding (default:
None)
--eab-hmac-key EAB_HMAC_KEY
HMAC key for External Account Binding (default: None)
--cert-name CERTNAME Certificate name to apply. This name is used by
Certbot for housekeeping and in file paths; it doesn't
affect the content of the certificate itself. To see
certificate names, run 'certbot certificates'. When
creating a new certificate, specifies the new
certificate's name. (default: the first provided
domain or the name of an existing certificate on your
system for the same domains)
--dry-run Perform a test run of the client, obtaining test
(invalid) certificates but not saving them to disk.
This can currently only be used with the 'certonly'
and 'renew' subcommands. Note: Although --dry-run
tries to avoid making any persistent changes on a
system, it is not completely side-effect free: if used
with webserver authenticator plugins like apache and
nginx, it makes and then reverts temporary config
changes in order to obtain test certificates, and
reloads webserver to deploy and then roll back those
changes. It also calls --pre-hook and --post-hook
commands if they are defined because they may be
necessary to accurately simulate renewal. --deploy-
hook commands are not called. (default: False)
--debug-challenges After setting up challenges, wait for user input
before submitting to CA (default: False)
--preferred-challenges PREF_CHALLS
A sorted, comma delimited list of the preferred
challenge to use during authorization with the most
preferred challenge listed first (Eg, "dns" or
"http,dns"). Not all plugins support all challenges.
See https://certbot.eff.org/docs/using.html#plugins
for details. ACME Challenges are versioned, but if you
pick "http" rather than "http-01", Certbot will select
the latest version automatically. (default: [])
--user-agent USER_AGENT
Set a custom user agent string for the client. User

```

(continues on next page)

(continued from previous page)

| | |
|---|---|
| | agent strings allow the CA to collect high level statistics about success rates by OS, plugin and use case, and to know when to deprecate support for past Python versions and flags. If you wish to hide this information from the Let's Encrypt server, set this to <code>""</code> . (default: <code>CertbotACMEClient/0.40.1 (certbot(-auto); OS_NAME OS_VERSION) Authenticator/XXX Installer/YYY (SUBCOMMAND; flags: FLAGS) Py/major.minor.patchlevel</code>). The flags encoded in the user agent are: <code>--duplicate</code> , <code>--force-renew</code> , <code>--allow-subset-of-names</code> , <code>-n</code> , and whether any hooks are set. |
| <code>--user-agent-comment</code> | <code>USER_AGENT_COMMENT</code> Add a comment to the default user agent string. May be used when repackaging Certbot or calling it from another tool to allow additional statistical data to be collected. Ignored if <code>--user-agent</code> is set. (Example: <code>Foo-Wrapper/1.0</code>) (default: <code>None</code>) |
| automation: | |
| Flags for automating execution & other tweaks | |
| <code>--keep-until-expiring</code> , <code>--keep</code> , <code>--reinstall</code> | If the requested certificate matches an existing certificate, always keep the existing one until it is due for renewal (for the 'run' subcommand this means reinstall the existing certificate). (default: <code>Ask</code>) |
| <code>--expand</code> | If an existing certificate is a strict subset of the requested names, always expand and replace it with the additional names. (default: <code>Ask</code>) |
| <code>--version</code> | show program's version number and exit |
| <code>--force-renewal</code> , <code>--renew-by-default</code> | If a certificate already exists for the requested domains, renew it now, regardless of whether it is near expiry. (Often <code>--keep-until-expiring</code> is more appropriate). Also implies <code>--expand</code> . (default: <code>False</code>) |
| <code>--renew-with-new-domains</code> | If a certificate already exists for the requested certificate name but does not match the requested domains, renew it now, regardless of whether it is near expiry. (default: <code>False</code>) |
| <code>--reuse-key</code> | When renewing, use the same private key as the existing certificate. (default: <code>False</code>) |
| <code>--allow-subset-of-names</code> | When performing domain validation, do not consider it a failure if authorizations can not be obtained for a strict subset of the requested domains. This may be useful for allowing renewals for multiple domains to succeed even if some domains no longer point at this system. This option cannot be used with <code>--csr</code> . (default: <code>False</code>) |
| <code>--agree-tos</code> | Agree to the ACME Subscriber Agreement (default: <code>Ask</code>) |
| <code>--duplicate</code> | Allow making a certificate lineage that duplicates an existing one (both can be renewed in parallel) (default: <code>False</code>) |
| <code>--os-packages-only</code> | (<code>certbot-auto</code> only) install OS package dependencies and then stop (default: <code>False</code>) |
| <code>--no-self-upgrade</code> | (<code>certbot-auto</code> only) prevent the <code>certbot-auto</code> script |

(continues on next page)

(continued from previous page)

| | |
|--|---|
| | from upgrading itself to newer released versions (default: Upgrade automatically) |
| <code>--no-bootstrap</code> | (certbot-auto only) prevent the certbot-auto script from installing OS-level dependencies (default: Prompt to install OS-wide dependencies, but exit if the user says 'No') |
| <code>--no-permissions-check</code> | (certbot-auto only) skip the check on the file system permissions of the certbot-auto script (default: False) |
| <code>-q, --quiet</code> | Silence all output except errors. Useful for automation via cron. Implies <code>--non-interactive</code> . (default: False) |
| security: | |
| Security parameters & server settings | |
| <code>--rsa-key-size N</code> | Size of the RSA key. (default: 2048) |
| <code>--must-staple</code> | Adds the OCSP Must Staple extension to the certificate. Autoconfigures OCSP Stapling for supported setups (Apache version $\geq 2.3.3$). (default: False) |
| <code>--redirect</code> | Automatically redirect all HTTP traffic to HTTPS for the newly authenticated vhost. (default: Ask) |
| <code>--no-redirect</code> | Do not automatically redirect all HTTP traffic to HTTPS for the newly authenticated vhost. (default: Ask) |
| <code>--hsts</code> | Add the Strict-Transport-Security header to every HTTP response. Forcing browser to always use SSL for the domain. Defends against SSL Stripping. (default: None) |
| <code>--uir</code> | Add the "Content-Security-Policy: upgrade-insecure-requests" header to every HTTP response. Forcing the browser to use https:// for every http:// resource. (default: None) |
| <code>--staple-ocsp</code> | Enables OCSP Stapling. A valid OCSP response is stapled to the certificate that the server offers during TLS. (default: None) |
| <code>--strict-permissions</code> | Require that all configuration files are owned by the current user; only needed if your config is somewhere unsafe like /tmp/ (default: False) |
| <code>--auto-hsts</code> | Gradually increasing max-age value for HTTP Strict Transport Security security header (default: False) |
| testing: | |
| The following flags are meant for testing and integration purposes only. | |
| <code>--test-cert, --staging</code> | Use the staging server to obtain or revoke test (invalid) certificates; equivalent to <code>--server https://acme-staging-v02.api.letsencrypt.org/directory</code> (default: False) |
| <code>--debug</code> | Show tracebacks in case of errors, and allow certbot-auto execution on experimental platforms (default: False) |
| <code>--no-verify-ssl</code> | Disable verification of the ACME server's certificate. (default: False) |
| <code>--http-01-port HTTP01_PORT</code> | |

(continues on next page)

(continued from previous page)

| | |
|---|---|
| | Port used in the http-01 challenge. This only affects the port Certbot listens on. A conforming ACME server will still attempt to connect on port 80. (default: 80) |
| <code>--http-01-address HTTP01_ADDRESS</code> | The address the server listens to during http-01 challenge. (default:) |
| <code>--https-port HTTPS_PORT</code> | Port used to serve HTTPS. This affects which port Nginx will listen on after a LE certificate is installed. (default: 443) |
| <code>--break-my-certs</code> | Be willing to replace or renew valid certificates with invalid (testing/staging) certificates (default: False) |
| paths: | |
| Flags for changing execution paths & servers | |
| <code>--cert-path CERT_PATH</code> | Path to where certificate is saved (with auth <code>--csr</code>), installed from, or revoked. (default: None) |
| <code>--key-path KEY_PATH</code> | Path to private key for certificate installation or revocation (if account key is missing) (default: None) |
| <code>--fullchain-path FULLCHAIN_PATH</code> | Accompanying path to a full certificate chain (certificate plus chain). (default: None) |
| <code>--chain-path CHAIN_PATH</code> | Accompanying path to a certificate chain. (default: None) |
| <code>--config-dir CONFIG_DIR</code> | Configuration directory. (default: /etc/letsencrypt) |
| <code>--work-dir WORK_DIR</code> | Working directory. (default: /var/lib/letsencrypt) |
| <code>--logs-dir LOGS_DIR</code> | Logs directory. (default: /var/log/letsencrypt) |
| <code>--server SERVER</code> | ACME Directory Resource URI. (default: https://acme-v02.api.letsencrypt.org/directory) |
| manage: | |
| Various subcommands and flags are available for managing your certificates: | |
| <code>certificates</code> | List certificates managed by Certbot |
| <code>delete</code> | Clean up all files related to a certificate |
| <code>renew</code> | Renew all certificates (or one specified with <code>--cert-name</code>) |
| <code>revoke</code> | Revoke a certificate specified with <code>--cert-path</code> or <code>--cert-name</code> |
| <code>update_symlinks</code> | Recreate symlinks in your /etc/letsencrypt/live/ directory |
| run: | |
| Options for obtaining & installing certificates | |
| certonly: | |
| Options for modifying how a certificate is obtained | |
| <code>--csr CSR</code> | Path to a Certificate Signing Request (CSR) in DER or PEM format. Currently <code>--csr</code> only works with the |

(continues on next page)

(continued from previous page)

```

'certonly' subcommand. (default: None)

renew:
    The 'renew' subcommand will attempt to renew all certificates (or more
    precisely, certificate lineages) you have previously obtained if they are
    close to expiry, and print a summary of the results. By default, 'renew'
    will reuse the options used to create obtain or most recently successfully
    renew each certificate lineage. You can try it with `--dry-run` first. For
    more fine-grained control, you can renew individual lineages with the
    `certonly` subcommand. Hooks are available to run commands before and
    after renewal; see https://certbot.eff.org/docs/using.html#renewal for
    more information on these.

--pre-hook PRE_HOOK    Command to be run in a shell before obtaining any
                        certificates. Intended primarily for renewal, where it
                        can be used to temporarily shut down a webserver that
                        might conflict with the standalone plugin. This will
                        only be called if a certificate is actually to be
                        obtained/renewed. When renewing several certificates
                        that have identical pre-hooks, only the first will be
                        executed. (default: None)

--post-hook POST_HOOK   Command to be run in a shell after attempting to
                        obtain/renew certificates. Can be used to deploy
                        renewed certificates, or to restart any servers that
                        were stopped by --pre-hook. This is only run if an
                        attempt was made to obtain/renew a certificate. If
                        multiple renewed certificates have identical post-
                        hooks, only one will be run. (default: None)

--deploy-hook DEPLOY_HOOK
                        Command to be run in a shell once for each
                        successfully issued certificate. For this command, the
                        shell variable $RENEWED_LINEAGE will point to the
                        config live subdirectory (for example,
                        "/etc/letsencrypt/live/example.com") containing the
                        new certificates and keys; the shell variable
                        $RENEWED_DOMAINS will contain a space-delimited list
                        of renewed certificate domains (for example,
                        "example.com www.example.com" (default: None)

--disable-hook-validation
                        Ordinarily the commands specified for --pre-hook
                        /--post-hook/--deploy-hook will be checked for
                        validity, to see if the programs being run are in the
                        $PATH, so that mistakes can be caught early, even when
                        the hooks aren't being run just yet. The validation is
                        rather simplistic and fails if you use more advanced
                        shell constructs, so you can use this switch to
                        disable it. (default: False)

--no-directory-hooks    Disable running executables found in Certbot's hook
                        directories during renewal. (default: False)

--disable-renew-updates
                        Disable automatic updates to your server configuration
                        that would otherwise be done by the selected installer
                        plugin, and triggered when the user executes "certbot
                        renew", regardless of if the certificate is renewed.
                        This setting does not apply to important TLS
                        configuration updates. (default: False)

```

(continues on next page)

(continued from previous page)

```
--no-autorenew          Disable auto renewal of certificates. (default: True)

certificates:
  List certificates managed by Certbot

delete:
  Options for deleting a certificate

revoke:
  Options for revocation of certificates

  --reason {unspecified,keycompromise,affiliationchanged,superseded,
↪cessationofoperation}
                        Specify reason for revoking certificate. (default:
                        unspecified)

  --delete-after-revoke
                        Delete certificates after revoking them, along with
                        all previous and later versions of those certificates.
                        (default: None)

  --no-delete-after-revoke
                        Do not delete certificates after revoking them. This
                        option should be used with caution because the 'renew'
                        subcommand will attempt to renew undeleted revoked
                        certificates. (default: None)

register:
  Options for account registration

  --register-unsafely-without-email
                        Specifying this flag enables registering an account
                        with no email address. This is strongly discouraged,
                        because in the event of key loss or account compromise
                        you will irrevocably lose access to your account. You
                        will also be unable to receive notice about impending
                        expiration or revocation of your certificates. Updates
                        to the Subscriber Agreement will still affect you, and
                        will be effective 14 days after posting an update to
                        the web site. (default: False)

  -m EMAIL, --email EMAIL
                        Email used for registration and recovery contact. Use
                        comma to register multiple emails, ex:
                        u1@example.com,u2@example.com. (default: Ask).

  --eff-email
                        Share your e-mail address with EFF (default: None)
  --no-eff-email
                        Don't share your e-mail address with EFF (default:
                        None)

update_account:
  Options for account modification

unregister:
  Options for account deactivation.

  --account ACCOUNT_ID Account ID to use (default: None)

install:
  Options for modifying how a certificate is deployed
```

(continues on next page)

(continued from previous page)

```
rollback:
    Options for rolling back server configuration changes

    --checkpoints N          Revert configuration N number of checkpoints.
                             (default: 1)

plugins:
    Options for the "plugins" subcommand

    --init                   Initialize plugins. (default: False)
    --prepare                Initialize and prepare plugins. (default: False)
    --authenticators         Limit to authenticator plugins only. (default: None)
    --installers             Limit to installer plugins only. (default: None)

update_symlinks:
    Recreates certificate and key symlinks in /etc/letsencrypt/live, if you
    changed them by hand or edited a renewal configuration file

enhance:
    Helps to harden the TLS configuration by adding security enhancements to
    already existing configuration.

plugins:
    Plugin Selection: Certbot client supports an extensible plugins
    architecture. See 'certbot plugins' for a list of all installed plugins
    and their names. You can force a particular plugin by setting options
    provided below. Running --help <plugin_name> will list flags specific to
    that plugin.

    --configurator CONFIGURATOR
                             Name of the plugin that is both an authenticator and
                             an installer. Should not be used together with
                             --authenticator or --installer. (default: Ask)
    -a AUTHENTICATOR, --authenticator AUTHENTICATOR
                             Authenticator plugin name. (default: None)
    -i INSTALLER, --installer INSTALLER
                             Installer plugin name (also used to find domains).
                             (default: None)
    --apache                 Obtain and install certificates using Apache (default:
                             False)
    --nginx                  Obtain and install certificates using Nginx (default:
                             False)
    --standalone             Obtain certificates using a "standalone" webserver.
                             (default: False)
    --manual                 Provide laborious manual instructions for obtaining a
                             certificate (default: False)
    --webroot                Obtain certificates by placing files in a webroot
                             directory. (default: False)
    --dns-cloudflare         Obtain certificates using a DNS TXT record (if you are
                             using Cloudflare for DNS). (default: False)
    --dns-cloudxns           Obtain certificates using a DNS TXT record (if you are
                             using CloudXNS for DNS). (default: False)
    --dns-digitalocean       Obtain certificates using a DNS TXT record (if you are
                             using DigitalOcean for DNS). (default: False)
    --dns-dnsimple           Obtain certificates using a DNS TXT record (if you are
                             using DNSimple for DNS). (default: False)
    --dns-dnsmadeeasy        Obtain certificates using a DNS TXT record (if you are
```

(continues on next page)

(continued from previous page)

| | |
|---|---|
| | using DNS Made Easy for DNS). (default: False) |
| <code>--dns-gehirn</code> | Obtain certificates using a DNS TXT record (if you are using Gehirn Infrastructure Service for DNS). (default: False) |
| <code>--dns-google</code> | Obtain certificates using a DNS TXT record (if you are using Google Cloud DNS). (default: False) |
| <code>--dns-linode</code> | Obtain certificates using a DNS TXT record (if you are using Linode for DNS). (default: False) |
| <code>--dns-luadns</code> | Obtain certificates using a DNS TXT record (if you are using LuaDNS for DNS). (default: False) |
| <code>--dns-nsone</code> | Obtain certificates using a DNS TXT record (if you are using NS1 for DNS). (default: False) |
| <code>--dns-ovh</code> | Obtain certificates using a DNS TXT record (if you are using OVH for DNS). (default: False) |
| <code>--dns-rfc2136</code> | Obtain certificates using a DNS TXT record (if you are using BIND for DNS). (default: False) |
| <code>--dns-route53</code> | Obtain certificates using a DNS TXT record (if you are using Route53 for DNS). (default: False) |
| <code>--dns-sakuracloud</code> | Obtain certificates using a DNS TXT record (if you are using Sakura Cloud for DNS). (default: False) |
| <code>apache:</code> | |
| Apache Web Server plugin (Please note that the default values of the Apache plugin options change depending on the operating system Certbot is run on.) | |
| <code>--apache-enmod APACHE_ENMOD</code> | Path to the Apache 'a2enmod' binary (default: None) |
| <code>--apache-dismod APACHE_DISMOD</code> | Path to the Apache 'a2dismod' binary (default: None) |
| <code>--apache-le-vhost-ext APACHE_LE_VHOST_EXT</code> | SSL vhost configuration extension (default: -le-ssl.conf) |
| <code>--apache-server-root APACHE_SERVER_ROOT</code> | Apache server root directory (default: /etc/apache2) |
| <code>--apache-vhost-root APACHE_VHOST_ROOT</code> | Apache server VirtualHost configuration root (default: None) |
| <code>--apache-logs-root APACHE_LOGS_ROOT</code> | Apache server logs directory (default: /var/log/apache2) |
| <code>--apache-challenge-location APACHE_CHALLENGE_LOCATION</code> | Directory path for challenge configuration (default: /etc/apache2) |
| <code>--apache-handle-modules APACHE_HANDLE_MODULES</code> | Let installer handle enabling required modules for you (Only Ubuntu/Debian currently) (default: False) |
| <code>--apache-handle-sites APACHE_HANDLE_SITES</code> | Let installer handle enabling sites for you (Only Ubuntu/Debian currently) (default: False) |
| <code>--apache-ctl APACHE_CTL</code> | Full path to Apache control script (default: apache2ctl) |
| <code>dns-cloudflare:</code> | |
| Obtain certificates using a DNS TXT record (if you are using Cloudflare for DNS). | |

(continues on next page)

(continued from previous page)

```
--dns-cloudflare-propagation-seconds DNS_CLOUDFLARE_PROPAGATION_SECONDS
    The number of seconds to wait for DNS to propagate
    before asking the ACME server to verify the DNS
    record. (default: 10)
--dns-cloudflare-credentials DNS_CLOUDFLARE_CREDENTIALS
    Cloudflare credentials INI file. (default: None)

dns-cloudxns:
    Obtain certificates using a DNS TXT record (if you are using CloudXNS for
    DNS).

--dns-cloudxns-propagation-seconds DNS_CLOUDXNS_PROPAGATION_SECONDS
    The number of seconds to wait for DNS to propagate
    before asking the ACME server to verify the DNS
    record. (default: 30)
--dns-cloudxns-credentials DNS_CLOUDXNS_CREDENTIALS
    CloudXNS credentials INI file. (default: None)

dns-digitalocean:
    Obtain certs using a DNS TXT record (if you are using DigitalOcean for
    DNS).

--dns-digitalocean-propagation-seconds DNS_DIGITALOCEAN_PROPAGATION_SECONDS
    The number of seconds to wait for DNS to propagate
    before asking the ACME server to verify the DNS
    record. (default: 10)
--dns-digitalocean-credentials DNS_DIGITALOCEAN_CREDENTIALS
    DigitalOcean credentials INI file. (default: None)

dns-dnssimple:
    Obtain certificates using a DNS TXT record (if you are using DNSimple for
    DNS).

--dns-dnssimple-propagation-seconds DNS_DNSIMPLE_PROPAGATION_SECONDS
    The number of seconds to wait for DNS to propagate
    before asking the ACME server to verify the DNS
    record. (default: 30)
--dns-dnssimple-credentials DNS_DNSIMPLE_CREDENTIALS
    DNSimple credentials INI file. (default: None)

dns-dnsmadeeasy:
    Obtain certificates using a DNS TXT record (if you are using DNS Made Easy
    for DNS).

--dns-dnsmadeeasy-propagation-seconds DNS_DNSMADEEASY_PROPAGATION_SECONDS
    The number of seconds to wait for DNS to propagate
    before asking the ACME server to verify the DNS
    record. (default: 60)
--dns-dnsmadeeasy-credentials DNS_DNSMADEEASY_CREDENTIALS
    DNS Made Easy credentials INI file. (default: None)

dns-gehirn:
    Obtain certificates using a DNS TXT record (if you are using Gehirn
    Infrastructure Service for DNS).

--dns-gehirn-propagation-seconds DNS_GEHIRN_PROPAGATION_SECONDS
```

(continues on next page)

(continued from previous page)

```

        The number of seconds to wait for DNS to propagate
        before asking the ACME server to verify the DNS
        record. (default: 30)
--dns-gehirn-credentials DNS_GEHIRN_CREDENTIALS
        Gehirn Infrastructure Service credentials file.
        (default: None)

dns-google:
    Obtain certificates using a DNS TXT record (if you are using Google Cloud
    DNS for DNS).

--dns-google-propagation-seconds DNS_GOOGLE_PROPAGATION_SECONDS
        The number of seconds to wait for DNS to propagate
        before asking the ACME server to verify the DNS
        record. (default: 60)
--dns-google-credentials DNS_GOOGLE_CREDENTIALS
        Path to Google Cloud DNS service account JSON file.
        (See https://developers.google.com/identity/protocols/
        OAuth2ServiceAccount#creatinganaccount for information
        about creating a service account and
        https://cloud.google.com/dns/access-
        control#permissions\_and\_roles for information about
        therequired permissions.) (default: None)

dns-linode:
    Obtain certs using a DNS TXT record (if you are using Linode for DNS).

--dns-linode-propagation-seconds DNS_LINODE_PROPAGATION_SECONDS
        The number of seconds to wait for DNS to propagate
        before asking the ACME server to verify the DNS
        record. (default: 1200)
--dns-linode-credentials DNS_LINODE_CREDENTIALS
        Linode credentials INI file. (default: None)

dns-luadns:
    Obtain certificates using a DNS TXT record (if you are using LuaDNS for
    DNS).

--dns-luadns-propagation-seconds DNS_LUADNS_PROPAGATION_SECONDS
        The number of seconds to wait for DNS to propagate
        before asking the ACME server to verify the DNS
        record. (default: 30)
--dns-luadns-credentials DNS_LUADNS_CREDENTIALS
        LuaDNS credentials INI file. (default: None)

dns-nsone:
    Obtain certificates using a DNS TXT record (if you are using NS1 for DNS).

--dns-nsone-propagation-seconds DNS_NSONE_PROPAGATION_SECONDS
        The number of seconds to wait for DNS to propagate
        before asking the ACME server to verify the DNS
        record. (default: 30)
--dns-nsone-credentials DNS_NSONE_CREDENTIALS
        NS1 credentials file. (default: None)

dns-ovh:
    Obtain certificates using a DNS TXT record (if you are using OVH for DNS).
```

(continues on next page)

(continued from previous page)

```
--dns-ovh-propagation-seconds DNS_OVH_PROPAGATION_SECONDS
    The number of seconds to wait for DNS to propagate
    before asking the ACME server to verify the DNS
    record. (default: 30)
--dns-ovh-credentials DNS_OVH_CREDENTIALS
    OVH credentials INI file. (default: None)

dns-rfc2136:
    Obtain certificates using a DNS TXT record (if you are using BIND for
    DNS).

--dns-rfc2136-propagation-seconds DNS_RFC2136_PROPAGATION_SECONDS
    The number of seconds to wait for DNS to propagate
    before asking the ACME server to verify the DNS
    record. (default: 60)
--dns-rfc2136-credentials DNS_RFC2136_CREDENTIALS
    RFC 2136 credentials INI file. (default: None)

dns-route53:
    Obtain certificates using a DNS TXT record (if you are using AWS Route53
    for DNS).

--dns-route53-propagation-seconds DNS_ROUTE53_PROPAGATION_SECONDS
    The number of seconds to wait for DNS to propagate
    before asking the ACME server to verify the DNS
    record. (default: 10)

dns-sakuracloud:
    Obtain certificates using a DNS TXT record (if you are using Sakura Cloud
    for DNS).

--dns-sakuracloud-propagation-seconds DNS_SAKURACLOUD_PROPAGATION_SECONDS
    The number of seconds to wait for DNS to propagate
    before asking the ACME server to verify the DNS
    record. (default: 90)
--dns-sakuracloud-credentials DNS_SAKURACLOUD_CREDENTIALS
    Sakura Cloud credentials file. (default: None)

manual:
    Authenticate through manual configuration or custom shell scripts. When
    using shell scripts, an authenticator script must be provided. The
    environment variables available to this script depend on the type of
    challenge. $CERTBOT_DOMAIN will always contain the domain being
    authenticated. For HTTP-01 and DNS-01, $CERTBOT_VALIDATION is the
    validation string, and $CERTBOT_TOKEN is the filename of the resource
    requested when performing an HTTP-01 challenge. An additional cleanup
    script can also be provided and can use the additional variable
    $CERTBOT_AUTH_OUTPUT which contains the stdout output from the auth
    script.

--manual-auth-hook MANUAL_AUTH_HOOK
    Path or command to execute for the authentication
    script (default: None)
--manual-cleanup-hook MANUAL_CLEANUP_HOOK
    Path or command to execute for the cleanup script
    (default: None)
```

(continues on next page)

(continued from previous page)

```
--manual-public-ip-logging-ok
    Automatically allows public IP logging (default: Ask)

nginx:
    Nginx Web Server plugin

--nginx-server-root NGINX_SERVER_ROOT
    Nginx server root directory. (default: /etc/nginx or
    /usr/local/etc/nginx)

--nginx-ctl NGINX_CTL
    Path to the 'nginx' binary, used for 'configtest' and
    retrieving nginx version number. (default: nginx)

null:
    Null Installer

standalone:
    Spin up a temporary webserver

webroot:
    Place files in webroot directory

--webroot-path WEBROOT_PATH, -w WEBROOT_PATH
    public_html / webroot path. This can be specified
    multiple times to handle different domains; each
    domain will have the webroot path that preceded it.
    For instance: `-w /var/www/example -d example.com -d
    www.example.com -w /var/www/thing -d thing.net -d
    m.thing.net` (default: Ask)

--webroot-map WEBROOT_MAP
    JSON dictionary mapping domains to webroot paths; this
    implies -d for each entry. You may need to escape this
    from your shell. E.g.: --webroot-map
    '{"eg1.is,m.eg1.is":"/www/eg1/", "eg2.is":"/www/eg2"}'
    This option is merged with, but takes precedence over,
    -w / -d entries. At present, if you put webroot-map in
    a config file, it needs to be on a single line, like:
    webroot-map = {"example.com":"/var/www"}. (default:
    {})
```

4.12 Getting help

If you're having problems, we recommend posting on the Let's Encrypt [Community Forum](#).

If you find a bug in the software, please do report it in our [issue tracker](#). Remember to give us as much information as possible:

- copy and paste exact command line used and the output (though mind that the latter might include some personally identifiable information, including your email and domains)
- copy and paste logs from `/var/log/letsencrypt` (though mind they also might contain personally identifiable information)
- copy and paste `certbot --version` output
- your operating system, including specific version

- specify which installation method you've chosen

DEVELOPER GUIDE

Table of Contents

- *Getting Started*
 - *Running a local copy of the client*
 - *Find issues to work on*
 - *Testing*
 - * *Running automated unit tests*
 - * *Running automated integration tests*
 - * *Running manual integration tests*
- *Code components and layout*
 - *Plugin-architecture*
 - *Authenticators*
 - *Installer*
 - *Installer Development*
 - * *Writing your own plugin*
- *Coding style*
- *Use `certbot.compat.os` instead of `os`*
- *Mypy type annotations*
- *Submitting a pull request*
- *Asking for help*
- *Updating certbot-auto and letsencrypt-auto*
 - *Updating the scripts*
 - *Building letsencrypt-auto-source/letsencrypt-auto*
 - *Opening a PR*
- *Updating the documentation*
- *Running the client with Docker*
- *Notes on OS dependencies*

5.1 Getting Started

Certbot has the same *system requirements* when set up for development. While the section below will help you install Certbot and its dependencies, Certbot needs to be run on a UNIX-like OS so if you're using Windows, you'll need to set up a (virtual) machine running an OS such as Linux and continue with these instructions on that UNIX-like OS.

5.1.1 Running a local copy of the client

Running the client in developer mode from your local tree is a little different than running Certbot as a user. To get set up, clone our git repository by running:

```
git clone https://github.com/certbot/certbot
```

If you're on macOS, we recommend you skip the rest of this section and instead run Certbot in Docker. You can find instructions for how to do this [here](#). If you're running on Linux, you can run the following commands to install dependencies and set up a virtual environment where you can run Certbot.

Install the OS system dependencies required to run Certbot.

```
# For APT-based distributions (e.g. Debian, Ubuntu ...)
sudo apt update
sudo apt install python3-dev python3-venv gcc libaugeas0 libssl-dev \
    libffi-dev ca-certificates openssl
# For RPM-based distributions (e.g. Fedora, CentOS ...)
# NB1: old distributions will use yum instead of dnf
# NB2: RHEL-based distributions use python3X-devel instead of python3-devel (e.g. ↪
↪python36-devel)
sudo dnf install python3-devel gcc augeas-libs openssl-devel libffi-devel \
    redhat-rpm-config ca-certificates openssl
```

Set up the Python virtual environment that will host your Certbot local instance.

```
cd certbot
python tools/venv3.py
```

Note: You may need to repeat this when Certbot's dependencies change or when a new plugin is introduced.

You can now run the copy of Certbot from git either by executing `venv3/bin/certbot`, or by activating the virtual environment. You can do the latter by running:

```
source venv3/bin/activate
```

After running this command, `certbot` and development tools like `ipdb`, `ipython`, `pytest`, and `tox` are available in the shell where you ran the command. These tools are installed in the virtual environment and are kept separate from your global Python installation. This works by setting environment variables so the right executables are found and Python can pull in the versions of various packages needed by Certbot. More information can be found in the [virtualenv docs](#).

5.1.2 Find issues to work on

You can find the open issues in the [github issue tracker](#). Comparatively easy ones are marked [good first issue](#). If you're starting work on something, post a comment to let others know and seek feedback on your plan where appropriate.

Once you've got a working branch, you can open a pull request. All changes in your pull request must have thorough unit test coverage, pass our tests, and be compliant with the [coding style](#).

5.1.3 Testing

You can test your code in several ways:

- running the [automated unit](#) tests,
- running the [automated integration](#) tests
- running an [ad hoc manual integration](#) test

Running automated unit tests

When you are working in a file `foo.py`, there should also be a file `foo_test.py` either in the same directory as `foo.py` or in the `tests` subdirectory (if there isn't, make one). While you are working on your code and tests, run `python foo_test.py` to run the relevant tests.

For debugging, we recommend putting `import ipdb; ipdb.set_trace()` statements inside the source code.

Once you are done with your code changes, and the tests in `foo_test.py` pass, run all of the unittests for Certbot with `tox -e py27` (this uses Python 2.7).

Once all the unittests pass, check for sufficient test coverage using `tox -e py27-cover`, and then check for code style with `tox -e lint` (all files) or `pylint --rcfile=.pylintrc path/to/file.py` (single file at a time).

Once all of the above is successful, you may run the full test suite using `tox --skip-missing-interpreters`. We recommend running the commands above first, because running all tests like this is very slow, and the large amount of output can make it hard to find specific failures when they happen.

Warning: The full test suite may attempt to modify your system's Apache config if your user has sudo permissions, so it should not be run on a production Apache server.

Running automated integration tests

Generally it is sufficient to open a pull request and let Github and Travis run integration tests for you. However, you may want to run them locally before submitting your pull request. You need Docker and docker-compose installed and working.

The tox environment `integration` will setup [Pebble](#), the Let's Encrypt ACME CA server for integration testing, then launch the Certbot integration tests.

With a user allowed to access your local Docker daemon, run:

```
tox -e integration
```

Tests will be run using pytest. A test report and a code coverage report will be displayed at the end of the integration tests execution.

Running manual integration tests

You can also manually execute Certbot against a local instance of the [Pebble](#) ACME server. This is useful to verify that the modifications done to the code makes Certbot behave as expected.

To do so you need:

- Docker installed, and a user with access to the Docker client,
- an available *local copy* of Certbot.

The virtual environment set up with `python tools/venv.py` contains two commands that can be used once the virtual environment is activated:

```
run_acme_server
```

- Starts a local instance of Pebble and runs in the foreground printing its logs.
- Press CTRL+C to stop this instance.
- This instance is configured to validate challenges against certbot executed locally.

```
certbot_test [ARGS...]
```

- Execute certbot with the provided arguments and other arguments useful for testing purposes, such as: verbose output, full tracebacks in case Certbot crashes, *etc.*
- Execution is preconfigured to interact with the Pebble CA started with `run_acme_server`.
- Any arguments can be passed as they would be to Certbot (eg. `certbot_test certonly -d test.example.com`).

Here is a typical workflow to verify that Certbot successfully issued a certificate using an HTTP-01 challenge on a machine with Python 3:

```
python tools/venv3.py
source venv3/bin/activate
run_acme_server &
certbot_test certonly --standalone -d test.example.com
# To stop Pebble, launch `fg` to get back the background job, then press CTRL+C
```

5.2 Code components and layout

The following components of the Certbot repository are distributed to users:

acme contains all protocol specific code

certbot main client code

certbot-apache and certbot-nginx client code to configure specific web servers

certbot-dns-* client code to configure DNS providers

certbot-auto and letsencrypt-auto shell scripts to install Certbot and its dependencies on UNIX systems

windows installer Installs Certbot on Windows and is built using the files in windows-installer/

5.2.1 Plugin-architecture

Certbot has a plugin architecture to facilitate support for different webrowsers, other TLS servers, and operating systems. The interfaces available for plugins to implement are defined in `interfaces.py` and `plugins/common.py`.

The main two plugin interfaces are `IAuthenticator`, which implements various ways of proving domain control to a certificate authority, and `IInstaller`, which configures a server to use a certificate once it is issued. Some plugins, like the built-in Apache and Nginx plugins, implement both interfaces and perform both tasks. Others, like the built-in Standalone authenticator, implement just one interface.

There are also `IDisplay` plugins, which can change how prompts are displayed to a user.

5.2.2 Authenticators

Authenticators are plugins that prove control of a domain name by solving a challenge provided by the ACME server. ACME currently defines several types of challenges: HTTP, TLS-ALPN, and DNS, represented by classes in `acme.challenges`. An authenticator plugin should implement support for at least one challenge type.

An Authenticator indicates which challenges it supports by implementing `get_chall_pref(domain)` to return a sorted list of challenge types in preference order.

An Authenticator must also implement `perform(achalls)`, which “performs” a list of challenges by, for instance, provisioning a file on an HTTP server, or setting a TXT record in DNS. Once all challenges have succeeded or failed, Certbot will call the plugin’s `cleanup(achalls)` method to remove any files or DNS records that were needed only during authentication.

5.2.3 Installer

Installers plugins exist to actually setup the certificate in a server, possibly tweak the security configuration to make it more correct and secure (Fix some mixed content problems, turn on HSTS, redirect to HTTPS, etc). Installer plugins tell the main client about their abilities to do the latter via the `supported_enhancements()` call. We currently have two Installers in the tree, the `ApacheConfigurator`, and the `NginxConfigurator`. External projects have made some progress toward support for IIS, Iccast and Plesk.

Installers and Authenticators will oftentimes be the same class/object (because for instance both tasks can be performed by a webserver like nginx) though this is not always the case (the standalone plugin is an authenticator that listens on port 80, but it cannot install certs; a postfix plugin would be an installer but not an authenticator).

Installers and Authenticators are kept separate because it should be possible to use the `StandaloneAuthenticator` (it sets up its own Python server to perform challenges) with a program that cannot solve challenges itself (Such as MTA installers).

5.2.4 Installer Development

There are a few existing classes that may be beneficial while developing a new `IInstaller`. Installers aimed to reconfigure UNIX servers may use Augeas for configuration parsing and can inherit from `AugeasConfigurator` class to handle much of the interface. Installers that are unable to use Augeas may still find the `Reverter` class helpful in handling configuration checkpoints and rollback.

Writing your own plugin

Certbot client supports dynamic discovery of plugins through the `setuptools` entry points using the `certbot.plugins` group. This way you can, for example, create a custom implementation of `IAuthenticator` or

the `IInstaller` without having to merge it with the core upstream source code. An example is provided in `examples/plugins/` directory.

While developing, you can install your plugin into a Certbot development virtualenv like this:

```
. venv/bin/activate
pip install -e examples/plugins/
certbot_test plugins
```

Your plugin should show up in the output of the last command. If not, it was not installed properly.

Once you've finished your plugin and published it, you can have your users install it system-wide with `pip install`. Note that this will only work for users who have Certbot installed from OS packages or via `pip`. Users who run `certbot-auto` are currently unable to use third-party plugins. It's technically possible to install third-party plugins into the virtualenv used by `certbot-auto`, but they will be wiped away when `certbot-auto` upgrades.

Warning: Please be aware though that as this client is still in a developer-preview stage, the API may undergo a few changes. If you believe the plugin will be beneficial to the community, please consider submitting a pull request to the repo and we will update it with any necessary API changes.

5.3 Coding style

Please:

1. **Be consistent with the rest of the code.**
2. Read [PEP 8 - Style Guide for Python Code](#).
3. Follow the [Google Python Style Guide](#), with the exception that we use [Sphinx-style](#) documentation:

```
def foo(arg):
    """Short description.

    :param int arg: Some number.

    :returns: Argument
    :rtype: int

    """
    return arg
```

4. Remember to use `pylint`.

5.4 Use `certbot.compat.os` instead of `os`

Python's standard library `os` module lacks full support for several Windows security features about file permissions (eg. DACLS). However several files handled by Certbot (eg. private keys) need strongly restricted access on both Linux and Windows.

To help with this, the `certbot.compat.os` module wraps the standard `os` module, and forbids usage of methods that lack support for these Windows security features.

As a developer, when working on Certbot or its plugins, you must use `certbot.compat.os` in every place you would need `os` (eg. `from certbot.compat import os` instead of `import os`). Otherwise the tests will fail when your PR is submitted.

5.5 Mypy type annotations

Certbot uses the `mypy` static type checker. Python 3 natively supports official type annotations, which can then be tested for consistency using `mypy`. Python 2 doesn't, but type annotations can be [added in comments](#). `Mypy` does some type checks even without type annotations; we can find bugs in Certbot even without a fully annotated codebase.

Certbot supports both Python 2 and 3, so we're using Python 2-style annotations.

Zulip wrote a [great guide](#) to using `mypy`. It's useful, but you don't have to read the whole thing to start contributing to Certbot.

To run `mypy` on Certbot, use `tox -e mypy` on a machine that has Python 3 installed.

Note that instead of just importing `typing`, due to packaging issues, in Certbot we import from `acme.magic_typing` and have to add some comments for `pylint` like this:

```
from acme.magic_typing import Dict # pylint: disable=unused-import, no-name-in-module
```

Also note that `OpenSSL`, which we rely on, has type definitions for `crypto` but not `SSL`. We use both. Those imports should look like this:

```
from OpenSSL import crypto
from OpenSSL import SSL # type: ignore # https://github.com/python/typeshed/issues/
↪2052
```

5.6 Submitting a pull request

Steps:

1. Write your code! When doing this, you should add *mypy type annotations* for any functions you add or modify. You can check that you've done this correctly by running `tox -e mypy` on a machine that has Python 3 installed.
2. Make sure your environment is set up properly and that you're in your `virtualenv`. You can do this by following the instructions in the [Getting Started](#) section.
3. Run `tox -e lint` to check for `pylint` errors. Fix any errors.
4. Run `tox --skip-missing-interpreters` to run the entire test suite including coverage. The `--skip-missing-interpreters` argument ignores missing versions of Python needed for running the tests. Fix any errors.
5. Submit the PR. Once your PR is open, please do not force push to the branch containing your pull request to squash or amend commits. We use [squash merges](#) on PRs and rewriting commits makes changes harder to track between reviews.
6. Did your tests pass on Travis? If they didn't, fix any errors.

5.7 Asking for help

If you have any questions while working on a Certbot issue, don't hesitate to ask for help! You can do this in the Certbot channel in EFF's Mattermost instance for its open source projects as described below.

You can get involved with several of EFF's software projects such as Certbot at the [EFF Open Source Contributor Chat Platform](#). By signing up for the EFF Open Source Contributor Chat Platform, you consent to share your personal information with the Electronic Frontier Foundation, which is the operator and data controller for this platform. The channels will be available both to EFF, and to other users of EFFOSCCP, who may use or disclose information in these channels outside of EFFOSCCP. EFF will use your information, according to the [Privacy Policy](#), to further the mission of EFF, including hosting and moderating the discussions on this platform.

Use of EFFOSCCP is subject to the [EFF Code of Conduct](#). When investigating an alleged Code of Conduct violation, EFF may review discussion channels or direct messages.

5.8 Updating certbot-auto and letsencrypt-auto

Note: We are currently only accepting changes to certbot-auto that fix regressions on platforms where certbot-auto is the recommended installation method at <https://certbot.eff.org/instructions>. If you are unsure if a change you want to make qualifies, don't hesitate to *ask for help*!

5.8.1 Updating the scripts

Developers should *not* modify the certbot-auto and letsencrypt-auto files in the root directory of the repository. Rather, modify the letsencrypt-auto.template and associated platform-specific shell scripts in the letsencrypt-auto-source and letsencrypt-auto-source/pieces/bootstrappers directory, respectively.

5.8.2 Building letsencrypt-auto-source/letsencrypt-auto

Once changes to any of the aforementioned files have been made, the letsencrypt-auto-source/letsencrypt-auto script should be updated. In lieu of manually updating this script, run the build script, which lives at letsencrypt-auto-source/build.py:

```
python letsencrypt-auto-source/build.py
```

Running build.py will update the letsencrypt-auto-source/letsencrypt-auto script. Note that the certbot-auto and letsencrypt-auto scripts in the root directory of the repository will remain **unchanged** after this script is run. Your changes will be propagated to these files during the next release of Certbot.

5.8.3 Opening a PR

When opening a PR, ensure that the following files are committed:

1. letsencrypt-auto-source/letsencrypt-auto.template and
letsencrypt-auto-source/pieces/bootstrappers/*
2. letsencrypt-auto-source/letsencrypt-auto (generated by build.py)

It might also be a good idea to double check that **no** changes were inadvertently made to the `certbot-auto` or `letsencrypt-auto` scripts in the root of the repository. These scripts will be updated by the core developers during the next release.

5.9 Updating the documentation

In order to generate the Sphinx documentation, run the following commands:

```
make -C docs clean html man
```

This should generate documentation in the `docs/_build/html` directory.

Note: If you skipped the “Getting Started” instructions above, run `pip install -e ".[docs]"` to install Certbot’s docs extras modules.

5.10 Running the client with Docker

You can use Docker Compose to quickly set up an environment for running and testing Certbot. To install Docker Compose, follow the instructions at <https://docs.docker.com/compose/install/>.

Note: Linux users can simply run `pip install docker-compose` to get Docker Compose after installing Docker Engine and activating your shell as described in the *Getting Started* section.

Now you can develop on your host machine, but run Certbot and test your changes in Docker. When using `docker-compose` make sure you are inside your clone of the Certbot repository. As an example, you can run the following command to check for linting errors:

```
docker-compose run --rm --service-ports development bash -c 'tox -e lint'
```

You can also leave a terminal open running a shell in the Docker container and modify Certbot code in another window. The Certbot repo on your host machine is mounted inside of the container so any changes you make immediately take effect. To do this, run:

```
docker-compose run --rm --service-ports development bash
```

Now running the check for linting errors described above is as easy as:

```
tox -e lint
```

5.11 Notes on OS dependencies

OS-level dependencies can be installed like so:

```
./certbot-auto --debug --os-packages-only
```

In general...

- `sudo` is required as a suggested way of running privileged process

- [Python](#) 2.7 or 3.4+ is required
- [Augeas](#) is required for the Python bindings
- `virtualenv` is used for managing other Python library dependencies

5.11.1 FreeBSD

FreeBSD by default uses `tcsh`. In order to activate `virtualenv` (see above), you will need a compatible shell, e.g. `pkg install bash && bash`.

PACKAGING GUIDE

6.1 Releases

We release packages and upload them to PyPI (wheels and source tarballs).

- <https://pypi.python.org/pypi/acme>
- <https://pypi.python.org/pypi/certbot>
- <https://pypi.python.org/pypi/certbot-apache>
- <https://pypi.python.org/pypi/certbot-nginx>
- <https://pypi.python.org/pypi/certbot-dns-cloudflare>
- <https://pypi.python.org/pypi/certbot-dns-cloudxns>
- <https://pypi.python.org/pypi/certbot-dns-digitalocean>
- <https://pypi.python.org/pypi/certbot-dns-dnssimple>
- <https://pypi.python.org/pypi/certbot-dns-dnsmadeeasy>
- <https://pypi.python.org/pypi/certbot-dns-google>
- <https://pypi.python.org/pypi/certbot-dns-linode>
- <https://pypi.python.org/pypi/certbot-dns-luadns>
- <https://pypi.python.org/pypi/certbot-dns-nsone>
- <https://pypi.python.org/pypi/certbot-dns-ovh>
- <https://pypi.python.org/pypi/certbot-dns-rfc2136>
- <https://pypi.python.org/pypi/certbot-dns-route53>

The following scripts are used in the process:

- <https://github.com/certbot/certbot/blob/master/tools/release.sh>

We use git tags to identify releases, using Semantic Versioning. For example: `v0.11.1`.

6.2 Notes for package maintainers

0. Please use our tagged releases, not `master`!
1. Do not package `certbot-compatibility-test` or `letshelp-certbot` - it's only used internally.

2. To run tests on our packages, you should use `python setup.py test`. Doing things like running `pytest` directly on our package files may not work because Certbot relies on `setuptools` to register and find its plugins.
3. If you'd like to include automated renewal in your package `certbot renew -q` should be added to `crontab` or `systemd` timer. Additionally you should include a random per-machine time offset to avoid having a large number of your clients hit Let's Encrypt's servers simultaneously.
4. `jws` is an internal script for `acme` module and it doesn't have to be packaged - it's mostly for debugging: you can use it as `echo foo | jws sign | jws verify`.
5. Do get in touch with us. We are happy to make any changes that will make packaging easier. If you need to apply some patches don't do it downstream - make a PR [here](#).

RESOURCES

Documentation: <https://certbot.eff.org/docs>

Software project: <https://github.com/certbot/certbot>

Notes for developers: <https://certbot.eff.org/docs/contributing.html>

Main Website: <https://certbot.eff.org>

Let's Encrypt Website: <https://letsencrypt.org>

Community: <https://community.letsencrypt.org>

ACME spec: <http://ietf-wg-acme.github.io/acme/>

ACME working area in github: <https://github.com/ietf-wg-acme/acme>



API DOCUMENTATION

8.1 certbot.achallenges

Client annotated ACME challenges.

Please use names such as `achall` to distinguish from variables “of type” `acme.challenges.Challenge` (denoted by `chall`) and `ChallengeBody` (denoted by `challb`):

```
from acme import challenges
from acme import messages
from certbot import achallenges

chall = challenges.DNS(token='foo')
challb = messages.ChallengeBody(chall=chall)
achall = achallenges.DNS(chall=challb, domain='example.com')
```

Note, that all annotated challenges act as a proxy objects:

```
achall.token == challb.token
```

class `certbot.achallenges.AnnotatedChallenge` (***kwargs*)

Bases: `josepy.util.ImmutableMap`

Client annotated challenge.

Wraps around server provided challenge and annotates with data useful for the client.

Variables `challb` – Wrapped `ChallengeBody`.

class `certbot.achallenges.KeyAuthorizationAnnotatedChallenge` (***kwargs*)

Bases: `certbot.achallenges.AnnotatedChallenge`

Client annotated `KeyAuthorizationChallenge` challenge.

response_and_validation (**args, **kwargs*)

Generate response and validation.

class `certbot.achallenges.DNS` (***kwargs*)

Bases: `certbot.achallenges.AnnotatedChallenge`

Client annotated “dns” ACME challenge.

acme_type

alias of `acme.challenges.DNS`

8.2 certbot.cli

Certbot command line argument & config processing.

`certbot.cli.report_config_interaction(modified, modifiers)`

Registers config option interaction to be checked by `set_by_cli`.

This function can be called by during the `__init__` or `add_parser_arguments` methods of plugins to register interactions between config options.

Parameters

- **modified** (*iterable or str (string_types)*) – config options that can be modified by modifiers
- **modifiers** (*iterable or str (string_types)*) – config options that modify modified

class `certbot.cli._Default`

Bases: `object`

A class to use as a default to detect if a value is set by a user

`certbot.cli.set_by_cli(var)`

Return True if a particular config variable has been set by the user (CLI or config file) including if the user explicitly set it to the default. Returns False if the variable was assigned a default value.

`certbot.cli.has_default_value(option, value)`

Does option have the default value?

If the default value of option is not known, False is returned.

Parameters

- **option** (*str*) – configuration variable being considered
- **value** – value of the configuration variable named option

Returns True if option has the default value, otherwise, False

Return type `bool`

`certbot.cli.option_was_set(option, value)`

Was option set by the user or does it differ from the default?

Parameters

- **option** (*str*) – configuration variable being considered
- **value** – value of the configuration variable named option

Returns True if the option was set, otherwise, False

Return type `bool`

`certbot.cli.argparse_type(variable)`

Return our argparse type function for a config variable (default: `str`)

`certbot.cli.read_file(filename, mode='rb')`

Returns the given file's contents.

Parameters

- **filename** (*str*) – path to file
- **mode** (*str*) – open mode (see `open`)

Returns absolute path of filename and its contents

Return type `tuple`

Raises `argparse.ArgumentTypeError` – File does not exist or is not readable.

`certbot.cli.flag_default` (*name*)

Default value for CLI flag.

`certbot.cli.config_help` (*name*, *hidden=False*)

Extract the help message for an IConfig attribute.

class `certbot.cli.HelpfulArgumentGroup` (*helpful_arg_parser*, *topic*)

Bases: `object`

Emulates an argparse group for use with HelpfulArgumentParser.

This class is used in the `add_group` method of HelpfulArgumentParser. Command line arguments can be added to the group, but help suppression and default detection is applied by HelpfulArgumentParser when necessary.

add_argument (**args*, ***kwargs*)

Add a new command line argument to the argument group.

class `certbot.cli.CustomHelpFormatter` (*prog*, *indent_increment=2*, *max_help_position=24*, *width=None*)

Bases: `argparse.HelpFormatter`

This is a clone of ArgumentDefaultsHelpFormatter, with bugfixes.

In particular we fix <https://bugs.python.org/issue28742>

class `certbot.cli.HelpfulArgumentParser` (*args*, *plugins*, *detect_defaults=False*)

Bases: `object`

Argparse Wrapper.

This class wraps argparse, adding the ability to make `-help` less verbose, and request help on specific subcategories at a time, eg ‘certbot `-help` security’ for security options.

_usage_string (*plugins*, *help_arg*)

Make usage strings late so that plugins can be initialised late

Parameters

- **plugins** – all discovered plugins
- **help_arg** – False for none; True for `-help`; “TOPIC” for `-help TOPIC`

Return type `str`

Returns a short usage string for the top of `-help TOPIC`)

remove_config_file_domains_for_renewal (*parsed_args*)

Make “certbot renew” safe if domains are set in cli.ini.

parse_args ()

Parses command line arguments and returns the result.

Returns parsed command line arguments

Return type `argparse.Namespace`

set_test_server (*parsed_args*)

We have `-staging/-dry-run`; perform sanity check and set config.server

handle_csr (*parsed_args*)

Process a `-csr` flag.

determine_verb()

Determines the verb/subcommand provided by the user.

This function works around some of the limitations of argparse.

prescan_for_flag(flag, possible_arguments)

Checks cli input for flags.

Check for a flag, which accepts a fixed set of possible arguments, in the command line; we will use this information to configure argparse's help correctly. Return the flag's argument, if it has one that matches the sequence @possible_arguments; otherwise return whether the flag is present.

add(topics, *args, **kwargs)

Add a new command line argument.

Parameters

- **topics** – str or [str] help topic(s) this should be listed under, or None for options that don't fit under a specific topic which will only be shown in “–help all” output. The first entry determines where the flag lives in the “–help all” output (None -> “optional arguments”).
- ***args** (*list*) – the names of this argument flag
- ****kwargs** (*dict*) – various argparse settings for this argument

modify_kwargs_for_default_detection(kwargs)**

Modify an arg so we can check if it was set by the user.

Changes the parameters given to argparse when adding an argument so we can properly detect if the value was set by the user.

Parameters **kwargs** (*dict*) – various argparse settings for this argument

Returns a modified versions of kwargs

Return type *dict*

add_deprecated_argument(argument_name, num_args)

Adds a deprecated argument with the name argument_name.

Deprecated arguments are not shown in the help. If they are used on the command line, a warning is shown stating that the argument is deprecated and no other action is taken.

Parameters

- **argument_name** (*str*) – Name of deprecated argument.
- **num_args** (*int*) – Number of arguments the option takes.

add_group(topic, verbs=(), **kwargs)

Create a new argument group.

This method must be called once for every topic, however, calls to this function are left next to the argument definitions for clarity.

Parameters

- **topic** (*str*) – Name of the new argument group.
- **verbs** (*str*) – List of subcommands that should be documented as part of this help group / topic

Returns The new argument group.

Return type *HelpfulArgumentGroup*

add_plugin_args (*plugins*)

Let each of the plugins add its own command line arguments, which may or may not be displayed as help topics.

determine_help_topics (*chosen_topic*)

The user may have requested help on a topic, return a dict of which topics to display. @chosen_topic has prescan_for_flag's return type

Returns dict

`certbot.cli.prepare_and_parse_args` (*plugins, args, detect_defaults=False*)

Returns parsed command line arguments.

Parameters

- **plugins** (*PluginsRegistry*) – available plugins
- **args** (*list*) – command line arguments with the program name removed

Returns parsed command line arguments

Return type `argparse.Namespace`

class `certbot.cli.CaseInsensitiveList`

Bases: `list`

A list that will ignore case when searching.

This class is passed to the `choices` argument of `argparse.add_arguments` through the helpful wrapper. It is necessary due to special handling of command line arguments by `set_by_cli` in which the `type_func` is not applied.

class `certbot.cli._EncodeReasonAction` (*option_strings, dest, nargs=None, const=None, default=None, type=None, choices=None, required=False, help=None, metavar=None*)

Bases: `argparse.Action`

Action class for parsing revocation reason.

class `certbot.cli._DomainsAction` (*option_strings, dest, nargs=None, const=None, default=None, type=None, choices=None, required=False, help=None, metavar=None*)

Bases: `argparse.Action`

Action class for parsing domains.

`certbot.cli.add_domains` (*args_or_config, domains*)

Registers new domains to be used during the current client run.

Domains are not added to the list of requested domains if they have already been registered.

Parameters

- **args_or_config** (*argparse.Namespace* or *configuration.NamespaceConfig*) – parsed command line arguments
- **domain** (*str*) – one or more comma separated domains

Returns domains after they have been normalized and validated

Return type `list` of `str`

class `certbot.cli._PrefChallAction` (*option_strings, dest, nargs=None, const=None, default=None, type=None, choices=None, required=False, help=None, metavar=None*)

Bases: `argparse.Action`

Action class for parsing preferred challenges.

`certbot.cli.parse_preferred_challenges` (*pref_challs*)

Translate and validate preferred challenges.

Parameters `pref_challs` (*list of str*) – list of preferred challenge types

Returns validated list of preferred challenge types

Return type *list of str*

Raises `errors.Error` – if `pref_challs` is invalid

class `certbot.cli._DeployHookAction` (*option_strings, dest, nargs=None, const=None, default=None, type=None, choices=None, required=False, help=None, metavar=None*)

Bases: `argparse.Action`

Action class for parsing deploy hooks.

class `certbot.cli._RenewHookAction` (*option_strings, dest, nargs=None, const=None, default=None, type=None, choices=None, required=False, help=None, metavar=None*)

Bases: `argparse.Action`

Action class for parsing renew hooks.

`certbot.cli.nonnegative_int` (*value*)

Converts value to an int and checks that it is not negative.

This function should be used as the type parameter for argparse arguments.

Parameters `value` (*str*) – value provided on the command line

Returns integer representation of value

Return type *int*

Raises `argparse.ArgumentTypeError` – if value isn't a non-negative integer

8.3 certbot.configuration

Certbot user-supplied configuration.

class `certbot.configuration.NamespaceConfig` (*namespace*)

Bases: `object`

Configuration wrapper around `argparse.Namespace`.

For more documentation, including available attributes, please see `certbot.interfaces.IConfig`. However, note that the following attributes are dynamically resolved using `work_dir` and relative paths defined in `certbot._internal.constants`:

- `accounts_dir`
- `csr_dir`
- `in_progress_dir`
- `key_dir`
- `temp_checkpoint_dir`

And the following paths are dynamically resolved using `config_dir` and relative paths defined in `certbot._internal.constants`:

- `default_archive_dir`
- `live_dir`
- `renewal_configs_dir`

Variables `namespace` – Namespace typically produced by `argparse.ArgumentParser.parse_args()`.

server_path

File path based on `server`.

accounts_dir_for_server_path (*server_path*)

Path to accounts directory based on `server_path`

renewal_hooks_dir

Path to directory with hooks to run with the `renew` subcommand.

renewal_pre_hooks_dir

Path to the pre-hook directory for the `renew` subcommand.

renewal_deploy_hooks_dir

Path to the deploy-hook directory for the `renew` subcommand.

renewal_post_hooks_dir

Path to the post-hook directory for the `renew` subcommand.

`certbot.configuration.check_config_sanity` (*config*)

Validate command line options and display error message if requirements are not met.

Parameters `config` – `IConfig` instance holding user configuration

8.4 certbot.crypto_util

Certbot client crypto utility functions.

Todo: Make the transition to use PSS rather than PKCS1_v1_5 when the server is capable of handling the signatures.

`certbot.crypto_util.init_save_key` (*key_size*, *key_dir*, *keyname*=`'key-certbot.pem'`)

Initializes and saves a privkey.

Init's key and saves it in PEM format on the filesystem.

Note: `keyname` is the attempted filename, it may be different if a file already exists at the path.

Parameters

- **key_size** (*int*) – RSA key size in bits
- **key_dir** (*str*) – Key save directory.
- **keyname** (*str*) – Filename of key

Returns `Key`

Return type `certbot.util.Key`

Raises `ValueError` – If unable to generate the key given `key_size`.

`certbot.crypto_util.init_save_csr(privkey, names, path)`

Initialize a CSR with the given private key.

Parameters

- **privkey** (`certbot.util.Key`) – Key to include in the CSR
- **names** (`set`) – `str` names to include in the CSR
- **path** (`str`) – Certificate save directory.

Returns CSR

Return type `certbot.util.CSR`

`certbot.crypto_util.valid_csr(csr)`

Validate CSR.

Check if `csr` is a valid CSR for the given domains.

Parameters **csr** (`str`) – CSR in PEM.

Returns Validity of CSR.

Return type `bool`

`certbot.crypto_util.csr_matches_pubkey(csr, privkey)`

Does private key correspond to the subject public key in the CSR?

Parameters

- **csr** (`str`) – CSR in PEM.
- **privkey** (`str`) – Private key file contents (PEM)

Returns Correspondence of private key to CSR subject public key.

Return type `bool`

`certbot.crypto_util.import_csr_file(csrfile, data)`

Import a CSR file, which can be either PEM or DER.

Parameters

- **csrfile** (`str`) – CSR filename
- **data** (`str`) – contents of the CSR file

Returns (`crypto.FILETYPE_PEM`, `util.CSR` object representing the CSR, list of domains requested in the CSR)

Return type `tuple`

`certbot.crypto_util.make_key(bits)`

Generate PEM encoded RSA key.

Parameters **bits** (`int`) – Number of bits, at least 1024.

Returns new RSA key in PEM form with specified number of bits

Return type `str`

`certbot.crypto_util.valid_privkey(privkey)`

Is valid RSA private key?

Parameters **privkey** (`str`) – Private key file contents in PEM

Returns Validity of private key.

Return type `bool`

`certbot.crypto_util.verify_renewable_cert(renewable_cert)`

For checking that your certs were not corrupted on disk.

Several things are checked:

1. Signature verification for the cert.
2. That fullchain matches cert and chain when concatenated.
3. Check that the private key matches the certificate.

Parameters `renewable_cert` (`storage.RenewableCert`) – cert to verify

Raises `errors.Error` – If verification fails.

`certbot.crypto_util.verify_renewable_cert_sig(renewable_cert)`

Verifies the signature of a `storage.RenewableCert` object.

Parameters `renewable_cert` (`storage.RenewableCert`) – cert to verify

Raises `errors.Error` – If signature verification fails.

`certbot.crypto_util.verify_signed_payload(public_key, signature, payload, signature_hash_algorithm)`

Check the signature of a payload.

Parameters

- **public_key** (`RSAPublicKey/EllipticCurvePublicKey`) – the public_key to check signature
- **signature** (`bytes`) – the signature bytes
- **payload** (`bytes`) – the payload bytes

:param `cryptography.hazmat.primitives.hashes.HashAlgorithm` `signature_hash_algorithm`: algorithm used to hash the payload

Raises

- **InvalidSignature** – If signature verification fails.
- `errors.Error` – If public key type is not supported

`certbot.crypto_util.verify_cert_matches_priv_key(cert_path, key_path)`

Verifies that the private key and cert match.

Parameters

- **cert_path** (`str`) – path to a cert in PEM format
- **key_path** (`str`) – path to a private key file

Raises `errors.Error` – If they don't match.

`certbot.crypto_util.verify_fullchain(renewable_cert)`

Verifies that fullchain is indeed cert concatenated with chain.

Parameters `renewable_cert` (`storage.RenewableCert`) – cert to verify

Raises `errors.Error` – If cert and chain do not combine to fullchain.

`certbot.crypto_util.pyopenssl_load_certificate(data)`

Load PEM/DER certificate.

Raises `errors.Error` –

`certbot.crypto_util.get_sans_from_cert(cert, typ=1)`
 Get a list of Subject Alternative Names from a certificate.

Parameters

- **cert** (*str*) – Certificate (encoded).
- **typ** – `crypto.FILETYPE_PEM` or `crypto.FILETYPE_ASN1`

Returns A list of Subject Alternative Names.

Return type `list`

`certbot.crypto_util.get_names_from_cert(csr, typ=1)`
 Get a list of domains from a cert, including the CN if it is set.

Parameters

- **cert** (*str*) – Certificate (encoded).
- **typ** – `crypto.FILETYPE_PEM` or `crypto.FILETYPE_ASN1`

Returns A list of domain names.

Return type `list`

`certbot.crypto_util.dump_pyopenssl_chain(chain, filetype=1)`
 Dump certificate chain into a bundle.

Parameters **chain** (*list*) – List of `crypto.X509` (or wrapped in `josepy.util.ComparableX509`).

`certbot.crypto_util.notBefore(cert_path)`
 When does the cert at `cert_path` start being valid?

Parameters **cert_path** (*str*) – path to a cert in PEM format

Returns the `notBefore` value from the cert at `cert_path`

Return type `datetime.datetime`

`certbot.crypto_util.notAfter(cert_path)`
 When does the cert at `cert_path` stop being valid?

Parameters **cert_path** (*str*) – path to a cert in PEM format

Returns the `notAfter` value from the cert at `cert_path`

Return type `datetime.datetime`

`certbot.crypto_util._notAfterBefore(cert_path, method)`
 Internal helper function for finding `notbefore`/`notafter`.

Parameters

- **cert_path** (*str*) – path to a cert in PEM format
- **method** (*function*) – one of `crypto.X509.get_notBefore` or `crypto.X509.get_notAfter`

Returns the `notBefore` or `notAfter` value from the cert at `cert_path`

Return type `datetime.datetime`

`certbot.crypto_util.sha256sum(filename)`

Compute a sha256sum of a file.

NB: In given file, platform specific newlines characters will be converted into their equivalent unicode counterparts before calculating the hash.

Parameters `filename` (*str*) – path to the file whose hash will be computed

Returns sha256 digest of the file in hexadecimal

Return type *str*

`certbot.crypto_util.cert_and_chain_from_fullchain(fullchain_pem)`

Split fullchain_pem into cert_pem and chain_pem

Parameters `fullchain_pem` (*str*) – concatenated cert + chain

Returns tuple of string cert_pem and chain_pem

Return type *tuple*

8.5 certbot.display

Certbot display utilities.

8.5.1 certbot.display.util

Certbot display.

`certbot.display.util.OK = 'ok'`

Display exit code indicating user acceptance.

`certbot.display.util.CANCEL = 'cancel'`

Display exit code for a user canceling the display.

`certbot.display.util.HELP = 'help'`

Display exit code when for when the user requests more help. (UNUSED)

`certbot.display.util.ESC = 'esc'`

Display exit code when the user hits Escape (UNUSED)

`certbot.display.util.SIDE_FRAME = '- - - - -'`

Display boundary (alternates spaces, so when copy-pasted, markdown doesn't interpret it as a heading)

`certbot.display.util._wrap_lines(msg)`

Format lines nicely to 80 chars.

Parameters `msg` (*str*) – Original message

Returns Formatted message respecting newlines in message

Return type *str*

`certbot.display.util.input_with_timeout(prompt=None, timeout=36000.0)`

Get user input with a timeout.

Behaves the same as `six.moves.input`, however, an error is raised if a user doesn't answer after timeout seconds. The default timeout value was chosen to place it just under 12 hours for users following our advice and running Certbot twice a day.

Parameters

- **prompt** (*str*) – prompt to provide for input
- **timeout** (*float*) – maximum number of seconds to wait for input

Returns user response

Return type *str*

:raises errors.Error if no answer is given before the timeout

class `certbot.display.util.FileDisplay` (*outfile*, *force_interactive*)

Bases: `object`

File-based display.

notification (*message*, *pause=True*, *wrap=True*, *force_interactive=False*)

Displays a notification and waits for user acceptance.

Parameters

- **message** (*str*) – Message to display
- **pause** (*bool*) – Whether or not the program should pause for the user’s confirmation
- **wrap** (*bool*) – Whether or not the application should wrap text
- **force_interactive** (*bool*) – True if it’s safe to prompt the user because it won’t cause any workflow regressions

menu (*message*, *choices*, *ok_label=None*, *cancel_label=None*, *help_label=None*, *default=None*, *cli_flag=None*, *force_interactive=False*, ***unused_kwargs*)

Display a menu.

Todo: This doesn’t enable the help label/button (I wasn’t sold on any interface I came up with for this). It would be a nice feature

Parameters

- **message** (*str*) – title of menu
- **choices** (*list of tuples (tag, item) or list of descriptions (tags will be enumerated)*) – Menu lines, len must be > 0
- **default** – default value to return (if one exists)
- **cli_flag** (*str*) – option used to set this value with the CLI
- **force_interactive** (*bool*) – True if it’s safe to prompt the user because it won’t cause any workflow regressions

Returns tuple of (*code*, *index*) where *code* - str display exit code *index* - int index of the user’s selection

Return type *tuple*

input (*message*, *default=None*, *cli_flag=None*, *force_interactive=False*, ***unused_kwargs*)

Accept input from the user.

Parameters

- **message** (*str*) – message to display to the user
- **default** – default value to return (if one exists)
- **cli_flag** (*str*) – option used to set this value with the CLI

- **force_interactive** (*bool*) – True if it’s safe to prompt the user because it won’t cause any workflow regressions

Returns tuple of (*code*, *input*) where *code* - str display exit code *input* - str of the user’s input

Return type tuple

yesno (*message*, *yes_label*=‘Yes’, *no_label*=‘No’, *default*=None, *cli_flag*=None, *force_interactive*=False, ***unused_kwargs*)
Query the user with a yes/no question.

Yes and No label must begin with different letters, and must contain at least one letter each.

Parameters

- **message** (*str*) – question for the user
- **yes_label** (*str*) – Label of the “Yes” parameter
- **no_label** (*str*) – Label of the “No” parameter
- **default** – default value to return (if one exists)
- **cli_flag** (*str*) – option used to set this value with the CLI
- **force_interactive** (*bool*) – True if it’s safe to prompt the user because it won’t cause any workflow regressions

Returns True for “Yes”, False for “No”

Return type bool

checklist (*message*, *tags*, *default*=None, *cli_flag*=None, *force_interactive*=False, ***unused_kwargs*)
Display a checklist.

Parameters

- **message** (*str*) – Message to display to user
- **tags** (*list*) – *str* tags to select, len(tags) > 0
- **default** – default value to return (if one exists)
- **cli_flag** (*str*) – option used to set this value with the CLI
- **force_interactive** (*bool*) – True if it’s safe to prompt the user because it won’t cause any workflow regressions

Returns tuple of (*code*, *tags*) where *code* - str display exit code *tags* - list of selected tags

Return type tuple

_return_default (*prompt*, *default*, *cli_flag*, *force_interactive*)
Should we return the default instead of prompting the user?

Parameters

- **prompt** (*str*) – prompt for the user
- **default** – default answer to prompt
- **cli_flag** (*str*) – command line option for setting an answer to this question
- **force_interactive** (*bool*) – if interactivity is forced by the IDisplay call

Returns True if we should return the default without prompting

Return type bool

`_can_interact` (*force_interactive*)

Can we safely interact with the user?

Parameters **`force_interactive`** (*bool*) – if interactivity is forced by the IDisplay call

Returns True if the display can interact with the user

Return type *bool*

`directory_select` (*message*, *default=None*, *cli_flag=None*, *force_interactive=False*, ***unused_kwargs*)

Display a directory selection screen.

Parameters

- **`message`** (*str*) – prompt to give the user
- **`default`** – default value to return (if one exists)
- **`cli_flag`** (*str*) – option used to set this value with the CLI
- **`force_interactive`** (*bool*) – True if it's safe to prompt the user because it won't cause any workflow regressions

Returns tuple of the form (*code*, *string*) where *code* - display exit code *string* - input entered by the user

`_scrub_checklist_input` (*indices*, *tags*)

Validate input and transform indices to appropriate tags.

Parameters

- **`indices`** (*list*) – input
- **`tags`** (*list*) – Original tags of the checklist

Returns valid tags the user selected

Return type *list of str*

`_print_menu` (*message*, *choices*)

Print a menu on the screen.

Parameters

- **`message`** (*str*) – title of menu
- **`choices`** (*list of tuples (tag, item) or list of descriptions (tags will be enumerated)*) – Menu lines

`_get_valid_int_ans` (*max_*)

Get a numerical selection.

Parameters **`max`** (*int*) – The maximum entry (len of choices), must be positive

Returns tuple of the form (*code*, *selection*) where *code* - str display exit code ('ok' or cancel') *selection* - int user's selection

Return type *tuple*

`certbot.display.util.assert_valid_call` (*prompt*, *default*, *cli_flag*, *force_interactive*)

Verify that provided arguments is a valid IDisplay call.

Parameters

- **`prompt`** (*str*) – prompt for the user
- **`default`** – default answer to prompt

- **cli_flag** (*str*) – command line option for setting an answer to this question
- **force_interactive** (*bool*) – if interactivity is forced by the IDisplay call

class certbot.display.util.**NoninteractiveDisplay** (*outfile*, **unused_args*, ***unused_kwargs*)

Bases: `object`

An iDisplay implementation that never asks for interactive user input

_interaction_fail (*message*, *cli_flag*, *extra=""*)

Error out in case of an attempt to interact in noninteractive mode

notification (*message*, *pause=False*, *wrap=True*, ***unused_kwargs*)

Displays a notification without waiting for user acceptance.

Parameters

- **message** (*str*) – Message to display to stdout
- **pause** (*bool*) – The NoninteractiveDisplay waits for no keyboard
- **wrap** (*bool*) – Whether or not the application should wrap text

menu (*message*, *choices*, *ok_label=None*, *cancel_label=None*, *help_label=None*, *default=None*, *cli_flag=None*, ***unused_kwargs*)

Avoid displaying a menu.

Parameters

- **message** (*str*) – title of menu
- **choices** (*list of tuples (tag, item) or list of descriptions (tags will be enumerated)*) – Menu lines, len must be > 0
- **default** (*int*) – the default choice
- **kwargs** (*dict*) – absorbs various irrelevant labelling arguments

Returns tuple of (*code*, *index*) where *code* - str display exit code *index* - int index of the user's selection

Return type `tuple`

Raises `errors.MissingCommandlineFlag` – if there was no default

input (*message*, *default=None*, *cli_flag=None*, ***unused_kwargs*)

Accept input from the user.

Parameters **message** (*str*) – message to display to the user

Returns tuple of (*code*, *input*) where *code* - str display exit code *input* - str of the user's input

Return type `tuple`

Raises `errors.MissingCommandlineFlag` – if there was no default

yesno (*message*, *yes_label=None*, *no_label=None*, *default=None*, *cli_flag=None*, ***unused_kwargs*)

Decide Yes or No, without asking anybody

Parameters

- **message** (*str*) – question for the user
- **kwargs** (*dict*) – absorbs yes_label, no_label

Raises `errors.MissingCommandlineFlag` – if there was no default

Returns True for “Yes”, False for “No”

Return type `bool`

checklist (*message*, *tags*, *default=None*, *cli_flag=None*, ***unused_kwargs*)

Display a checklist.

Parameters

- **message** (*str*) – Message to display to user
- **tags** (*list*) – *str* tags to select, len(tags) > 0
- **kwargs** (*dict*) – absorbs default_status arg

Returns tuple of (*code*, *tags*) where *code* - str display exit code *tags* - list of selected tags

Return type `tuple`

directory_select (*message*, *default=None*, *cli_flag=None*, ***unused_kwargs*)

Simulate prompting the user for a directory.

This function returns default if it is not None, otherwise, an exception is raised explaining the problem. If *cli_flag* is not None, the error message will include the flag that can be used to set this value with the CLI.

Parameters

- **message** (*str*) – prompt to give the user
- **default** – default value to return (if one exists)
- **cli_flag** (*str*) – option used to set this value with the CLI

Returns tuple of the form (*code*, *string*) where *code* - int display exit code *string* - input entered by the user

`certbot.display.util.separate_list_input` (*input_*)

Separate a comma or space separated list.

Parameters **input** (*str*) – input from the user

Returns strings

Return type `list`

`certbot.display.util._parens_around_char` (*label*)

Place parens around first character of label.

Parameters **label** (*str*) – Must contain at least one character

8.5.2 `certbot.display.ops`

Contains UI methods for LE user operations.

`certbot.display.ops.get_email` (*invalid=False*, *optional=True*)

Prompt for valid email address.

Parameters

- **invalid** (*bool*) – True if an invalid address was provided by the user
- **optional** (*bool*) – True if the user can use `--register-unsafely-without-email` to avoid providing an e-mail

Returns e-mail address

Return type `str`

Raises `errors.Error` – if the user cancels

`certbot.display.ops.choose_account(accounts)`
Choose an account.

Parameters `accounts` (*list*) – Containing at least one Account

`certbot.display.ops.choose_values(values, question=None)`
Display screen to let user pick one or multiple values from the provided list.

Parameters `values` (*list*) – Values to select from

Returns List of selected values

Return type *list*

`certbot.display.ops.choose_names(installer, question=None)`
Display screen to select domains to validate.

Parameters

- **installer** (`certbot.interfaces.IInstaller`) – An installer object
- **question** (*str*) – Overriding dialog question to ask the user if asked to choose from domain names.

Returns List of selected names

Return type *list of str*

`certbot.display.ops.get_valid_domains(domains)`

Helper method for choose_names that implements basic checks on domain names

Parameters `domains` (*list*) – Domain names to validate

Returns List of valid domains

Return type *list*

`certbot.display.ops._sort_names(FQDNs)`
Sort FQDNs by SLD (and if many, by their subdomains)

Parameters `FQDNs` (*list*) – list of domain names

Returns Sorted list of domain names

Return type *list*

`certbot.display.ops._filter_names(names, override_question=None)`
Determine which names the user would like to select from a list.

Parameters `names` (*list*) – domain names

Returns tuple of the form (`code`, `names`) where `code` - str display exit code `names` - list of names selected

Return type *tuple*

`certbot.display.ops._choose_names_manually(prompt_prefix=)`
Manually input names for those without an installer.

Parameters `prompt_prefix` (*str*) – string to prepend to prompt for domains

Returns list of provided names

Return type *list of str*

`certbot.display.ops.success_installation(domains)`

Display a box confirming the installation of HTTPS.

Parameters `domains` (*list*) – domain names which were enabled

`certbot.display.ops.success_renewal(domains)`

Display a box confirming the renewal of an existing certificate.

Parameters `domains` (*list*) – domain names which were renewed

`certbot.display.ops.success_revocation(cert_path)`

Display a box confirming a certificate has been revoked.

Parameters `cert_path` (*list*) – path to certificate which was revoked.

`certbot.display.ops._gen_ssl_lab_urls(domains)`

Returns a list of urls.

Parameters `domains` (*list*) – Each domain is a 'str'

`certbot.display.ops._gen_https_names(domains)`

Returns a string of the https domains.

Domains are formatted nicely with `https://` prepended to each.

Parameters `domains` (*list*) – Each domain is a 'str'

`certbot.display.ops.validated_input(validator, *args, **kwargs)`

Like `input`, but with validation.

Parameters

- **validator** (*callable*) – A method which will be called on the supplied input. If the method raises a `errors.Error`, its text will be displayed and the user will be re-prompted.
- ***args** (*list*) – Arguments to be passed to `input`.
- ****kwargs** (*dict*) – Arguments to be passed to `input`.

Returns as `input`

Return type `tuple`

`certbot.display.ops.validated_directory(validator, *args, **kwargs)`

Like `directory_select`, but with validation.

Parameters

- **validator** (*callable*) – A method which will be called on the supplied input. If the method raises a `errors.Error`, its text will be displayed and the user will be re-prompted.
- ***args** (*list*) – Arguments to be passed to `directory_select`.
- ****kwargs** (*dict*) – Arguments to be passed to `directory_select`.

Returns as `directory_select`

Return type `tuple`

8.6 certbot.errors

Certbot client errors.

exception `certbot.errors.Error`
 Bases: `exceptions.Exception`
 Generic Certbot client error.

exception `certbot.errors.AccountStorageError`
 Bases: `certbot.errors.Error`
 Generic *AccountStorage* error.

exception `certbot.errors.AccountNotFound`
 Bases: `certbot.errors.AccountStorageError`
 Account not found error.

exception `certbot.errors.ReverterError`
 Bases: `certbot.errors.Error`
 Certbot Reverter error.

exception `certbot.errors.SubprocessError`
 Bases: `certbot.errors.Error`
 Subprocess handling error.

exception `certbot.errors.CertStorageError`
 Bases: `certbot.errors.Error`
 Generic CertStorage error.

exception `certbot.errors.HookCommandNotFound`
 Bases: `certbot.errors.Error`
 Failed to find a hook command in the PATH.

exception `certbot.errors.SignalExit`
 Bases: `certbot.errors.Error`
 A Unix signal was received while in the ErrorHandler context manager.

exception `certbot.errors.OverlappingMatchFound`
 Bases: `certbot.errors.Error`
 Multiple lineages matched what should have been a unique result.

exception `certbot.errors.LockError`
 Bases: `certbot.errors.Error`
 File locking error.

exception `certbot.errors.AuthorizationError`
 Bases: `certbot.errors.Error`
 Authorization error.

exception `certbot.errors.FailedChallenges` (*failed_achalls*)
 Bases: `certbot.errors.AuthorizationError`
 Failed challenges error.

Variables `failed_achalls` (*set*) – Failed *AnnotatedChallenge* instances.

exception `certbot.errors.PluginError`
 Bases: `certbot.errors.Error`
 Certbot Plugin error.

exception `certbot.errors.PluginEnhancementAlreadyPresent`

Bases: `certbot.errors.Error`

Enhancement was already set

exception `certbot.errors.PluginSelectionError`

Bases: `certbot.errors.Error`

A problem with plugin/configurator selection or setup

exception `certbot.errors.NoInstallationError`

Bases: `certbot.errors.PluginError`

Certbot No Installation error.

exception `certbot.errors.MisconfigurationError`

Bases: `certbot.errors.PluginError`

Certbot Misconfiguration error.

exception `certbot.errors.NotSupportedError`

Bases: `certbot.errors.PluginError`

Certbot Plugin function not supported error.

exception `certbot.errors.PluginStorageError`

Bases: `certbot.errors.PluginError`

Certbot Plugin Storage error.

exception `certbot.errors.StandaloneBindError` (*socket_error, port*)

Bases: `certbot.errors.Error`

Standalone plugin bind error.

exception `certbot.errors.ConfigurationError`

Bases: `certbot.errors.Error`

Configuration sanity error.

exception `certbot.errors.MissingCommandlineFlag`

Bases: `certbot.errors.Error`

A command line argument was missing in noninteractive usage

8.7 certbot.hooks

Facilities for implementing hooks that call shell commands.

`certbot.hooks.validate_hooks` (*config*)

Check hook commands are executable.

`certbot.hooks._prog` (*shell_cmd*)

Extract the program run by a shell command.

Parameters `shell_cmd` (*str*) – command to be executed

Returns basename of command or None if the command isn't found

Return type `str` or None

`certbot.hooks.validate_hook` (*shell_cmd, hook_name*)

Check that a command provided as a hook is plausibly executable.

Raises `errors.HookCommandNotFound` – if the command is not found

`certbot.hooks.pre_hook (config)`

Run pre-hooks if they exist and haven't already been run.

When Certbot is running with the `renew` subcommand, this function runs any hooks found in the `config.renewal_pre_hooks_dir` (if they have not already been run) followed by any pre-hook in the config. If hooks in `config.renewal_pre_hooks_dir` are run and the pre-hook in the config is a path to one of these scripts, it is not run twice.

Parameters `config` (`configuration.NamespaceConfig`) – Certbot settings

`certbot.hooks._run_pre_hook_if_necessary (command)`

Run the specified pre-hook if we haven't already.

If we've already run this exact command before, a message is logged saying the pre-hook was skipped.

Parameters `command` (`str`) – pre-hook to be run

`certbot.hooks.post_hook (config)`

Run post-hooks if defined.

This function also registers any executables found in `config.renewal_post_hooks_dir` to be run when Certbot is used with the `renew` subcommand.

If the verb is `renew`, we delay executing any post-hooks until `run_saved_post_hooks()` is called. In this case, this function registers all hooks found in `config.renewal_post_hooks_dir` to be called followed by any post-hook in the config. If the post-hook in the config is a path to an executable in the post-hook directory, it is not scheduled to be run twice.

Parameters `config` (`configuration.NamespaceConfig`) – Certbot settings

`certbot.hooks._run_eventually (command)`

Registers a post-hook to be run eventually.

All commands given to this function will be run exactly once in the order they were given when `run_saved_post_hooks()` is called.

Parameters `command` (`str`) – post-hook to register to be run

`certbot.hooks.run_saved_post_hooks ()`

Run any post hooks that were saved up in the course of the 'renew' verb

`certbot.hooks.deploy_hook (config, domains, lineage_path)`

Run post-issuance hook if defined.

Parameters

- **config** (`configuration.NamespaceConfig`) – Certbot settings
- **domains** (`list of str`) – domains in the obtained certificate
- **lineage_path** (`str`) – live directory path for the new cert

`certbot.hooks.renew_hook (config, domains, lineage_path)`

Run post-renewal hooks.

This function runs any hooks found in `config.renewal_deploy_hooks_dir` followed by any `renew-hook` in the config. If the `renew-hook` in the config is a path to a script in `config.renewal_deploy_hooks_dir`, it is not run twice.

If Certbot is doing a dry run, no hooks are run and messages are logged saying that they were skipped.

Parameters

- **config** (`configuration.NamespaceConfig`) – Certbot settings

- **domains** (*list of str*) – domains in the obtained certificate
- **lineage_path** (*str*) – live directory path for the new cert

`certbot.hooks._run_deploy_hook` (*command, domains, lineage_path, dry_run*)

Run the specified deploy-hook (if not doing a dry run).

If `dry_run` is `True`, `command` is not run and a message is logged saying that it was skipped. If `dry_run` is `False`, the hook is run after setting the appropriate environment variables.

Parameters

- **command** (*str*) – command to run as a deploy-hook
- **domains** (*list of str*) – domains in the obtained certificate
- **lineage_path** (*str*) – live directory path for the new cert
- **dry_run** (*bool*) – `True` iff Certbot is doing a dry run

`certbot.hooks._run_hook` (*cmd_name, shell_cmd*)

Run a hook command.

Parameters

- **cmd_name** (*str*) – the user facing name of the hook being run
- **shell_cmd** (*list of str or str*) – shell command to execute

Returns `stderr` if there was any

`certbot.hooks.execute` (*cmd_name, shell_cmd*)

Run a command.

Parameters

- **cmd_name** (*str*) – the user facing name of the hook being run
- **shell_cmd** (*list of str or str*) – shell command to execute

Returns `tuple (str stderr, str stdout)`

`certbot.hooks.list_hooks` (*dir_path*)

List paths to all hooks found in `dir_path` in sorted order.

Parameters **dir_path** (*str*) – directory to search

Returns *list of str*

Return type sorted list of absolute paths to executables in `dir_path`

8.8 certbot

Certbot client.

8.9 certbot.interfaces

Certbot client interfaces.

class `certbot.interfaces.AccountStorage`

Bases: `object`

Accounts storage interface.

find_all()

Find all accounts.

Returns All found accounts.

Return type `list`

load(account_id)

Load an account by its id.

Raises

- **AccountNotFound** – if account could not be found
- **AccountStorageError** – if account could not be loaded

save(account, client)

Save account.

Raises **AccountStorageError** – if account could not be saved

interface `certbot.interfaces.IPluginFactory`

IPlugin factory.

Objects providing this interface will be called without satisfying any entry point “extras” (extra dependencies) you might have defined for your plugin, e.g (excerpt from `setup.py` script):

```
setup(
    ...
    entry_points={
        'certbot.plugins': [
            'name=example_project.plugin[plugin_deps]',
        ],
    },
    extras_require={
        'plugin_deps': ['dep1', 'dep2'],
    }
)
```

Therefore, make sure such objects are importable and usable without extras. This is necessary, because CLI does the following operations (in order):

- loads an entry point,
- calls `inject_parser_options`,
- requires an entry point,
- creates plugin instance (`__call__`).

description

Short plugin description

__call__(config, name)

Create new *IPlugin*.

Parameters

- **config** (*IConfig*) – Configuration.
- **name** (*str*) – Unique plugin name.

inject_parser_options(parser, name)

Inject argument parser options (flags).

1. Be nice and prepend all options and destinations with `option_namespace` and `dest_namespace`.

2. Inject options (flags) only. Positional arguments are not allowed, as this would break the CLI.

Parameters

- **parser** (*ArgumentParser*) – (Almost) top-level CLI parser.
- **name** (*str*) – Unique plugin name.

interface `certbot.interfaces.IPlugin`

Certbot plugin.

prepare ()

Prepare the plugin.

Finish up any additional initialization.

Raises

- **PluginError** – when full initialization cannot be completed.
- **MisconfigurationError** – when full initialization cannot be completed. Plugin will be displayed on a list of available plugins.
- **NoInstallationError** – when the necessary programs/files cannot be located. Plugin will NOT be displayed on a list of available plugins.
- **NotSupportedError** – when the installation is recognized, but the version is not currently supported.

more_info ()

Human-readable string to help the user.

Should describe the steps taken and any relevant info to help the user decide which plugin to use.

Rtype str

interface `certbot.interfaces.IAuthenticator`

Extends: `certbot.interfaces.IPlugin`

Generic Certbot Authenticator.

Class represents all possible tools processes that have the ability to perform challenges and attain a certificate.

get_chall_pref (*domain*)

Return `collections.Iterable` of challenge preferences.

Parameters **domain** (*str*) – Domain for which challenge preferences are sought.

Returns `collections.Iterable` of challenge types (subclasses of `acme.challenges.Challenge`) with the most preferred challenges first. If a type is not specified, it means the Authenticator cannot perform the challenge.

Return type `collections.Iterable`

perform (*achalls*)

Perform the given challenge.

Parameters **achalls** (*list*) – Non-empty (guaranteed) list of `AnnotatedChallenge` instances, such that it contains types found within `get_chall_pref()` only.

Returns `collections.Iterable` of ACME `ChallengeResponse` instances corresponding to each provided `Challenge`.

Return type `collections.Iterable` of `acme.challenges.ChallengeResponse`, where responses are required to be returned in the same order as corresponding input challenges

Raises *PluginError* – If some or all challenges cannot be performed

cleanup (*achalls*)

Revert changes and shutdown after challenges complete.

This method should be able to revert all changes made by perform, even if perform exited abnormally.

Parameters *achalls* (*list*) – Non-empty (guaranteed) list of *AnnotatedChallenge* instances, a subset of those previously passed to *perform()*.

Raises *PluginError* – if original configuration cannot be restored

interface `certbot.interfaces.IConfig`

Certbot user-supplied configuration.

Warning: The values stored in the configuration have not been filtered, stripped or sanitized.

server

ACME Directory Resource URI.

email

Email used for registration and recovery contact. Use comma to register multiple emails, ex: `u1@example.com,u2@example.com`. (default: Ask).

rsa_key_size

Size of the RSA key.

must_staple

Adds the OCSP Must Staple extension to the certificate. Autoconfigures OCSP Stapling for supported setups (Apache version $\geq 2.3.3$).

config_dir

Configuration directory.

work_dir

Working directory.

accounts_dir

Directory where all account information is stored.

backup_dir

Configuration backups directory.

csr_dir

Directory where newly generated Certificate Signing Requests (CSRs) are saved.

in_progress_dir

Directory used before a permanent checkpoint is finalized.

key_dir

Keys storage.

temp_checkpoint_dir

Temporary checkpoint directory.

no_verify_ssl

Disable verification of the ACME server's certificate.

http01_port

Port used in the http-01 challenge. This only affects the port Certbot listens on. A conforming ACME server will still attempt to connect on port 80.

http01_address

The address the server listens to during http-01 challenge.

https_port

Port used to serve HTTPS. This affects which port Nginx will listen on after a LE certificate is installed.

pref_challs

Sorted user specified preferred challengestype strings with the most preferred challenge listed first

allow_subset_of_names

When performing domain validation, do not consider it a failure if authorizations can not be obtained for a strict subset of the requested domains. This may be useful for allowing renewals for multiple domains to succeed even if some domains no longer point at this system. This is a boolean

strict_permissions

Require that all configuration files are owned by the current user; only needed if your config is somewhere unsafe like /tmp/. This is a boolean

disable_renew_updates

If updates provided by installer enhancements when Certbot is being run with “renew” verb should be disabled.

interface `certbot.interfaces.IInstaller`

Extends: `certbot.interfaces.IPlugin`

Generic Certbot Installer Interface.

Represents any server that an X509 certificate can be placed.

It is assumed that `save()` is the only method that finalizes a checkpoint. This is important to ensure that checkpoints are restored in a consistent manner if requested by the user or in case of an error.

Using `certbot.reverter.Reverter` to implement checkpoints, rollback, and recovery can dramatically simplify plugin development.

get_all_names()

Returns all names that may be authenticated.

Return type `collections.Iterable of str`

deploy_cert (*domain*, *cert_path*, *key_path*, *chain_path*, *fullchain_path*)

Deploy certificate.

Parameters

- **domain** (*str*) – domain to deploy certificate file
- **cert_path** (*str*) – absolute path to the certificate file
- **key_path** (*str*) – absolute path to the private key file
- **chain_path** (*str*) – absolute path to the certificate chain file
- **fullchain_path** (*str*) – absolute path to the certificate fullchain file (cert plus chain)

Raises `PluginError` – when cert cannot be deployed

enhance (*domain*, *enhancement*, *options=None*)

Perform a configuration enhancement.

Parameters

- **domain** (*str*) – domain for which to provide enhancement
- **enhancement** (*str*) – An enhancement as defined in `ENHANCEMENTS`

- **options** – Flexible options parameter for enhancement. Check documentation of `ENHANCEMENTS` for expected options for each enhancement.

Raises `PluginError` – If Enhancement is not supported, or if an error occurs during the enhancement.

supported_enhancements()

Returns a `collections.Iterable` of supported enhancements.

Returns supported enhancements which should be a subset of `ENHANCEMENTS`

Return type `collections.Iterable` of `str`

save(title=None, temporary=False)

Saves all changes to the configuration files.

Both title and temporary are needed because a save may be intended to be permanent, but the save is not ready to be a full checkpoint.

It is assumed that at most one checkpoint is finalized by this method. Additionally, if an exception is raised, it is assumed a new checkpoint was not finalized.

Parameters

- **title** (`str`) – The title of the save. If a title is given, the configuration will be saved as a new checkpoint and put in a timestamped directory. `title` has no effect if `temporary` is true.
- **temporary** (`bool`) – Indicates whether the changes made will be quickly reversed in the future (challenges)

Raises `PluginError` – when save is unsuccessful

rollback_checkpoints(rollback=1)

Revert `rollback` number of configuration checkpoints.

Raises `PluginError` – when configuration cannot be fully reverted

recovery_routine()

Revert configuration to most recent finalized checkpoint.

Remove all changes (temporary and permanent) that have not been finalized. This is useful to protect against crashes and other execution interruptions.

Raises `errors.PluginError` – If unable to recover the configuration

config_test()

Make sure the configuration is valid.

Raises `MisconfigurationError` – when the config is not in a usable state

restart()

Restart or refresh the server content.

Raises `PluginError` – when server cannot be restarted

interface `certbot.interfaces.IDisplay`

Generic display.

notification(message, pause, wrap=True, force_interactive=False)

Displays a string message

Parameters

- **message** (`str`) – Message to display

- **pause** (*bool*) – Whether or not the application should pause for confirmation (if available)
- **wrap** (*bool*) – Whether or not the application should wrap text
- **force_interactive** (*bool*) – True if it’s safe to prompt the user because it won’t cause any workflow regressions

menu (*message*, *choices*, *ok_label=None*, *cancel_label=None*, *help_label=None*, *default=None*, *cli_flag=None*, *force_interactive=False*)
 Displays a generic menu.

When not setting `force_interactive=True`, you must provide a default value.

Parameters

- **message** (*str*) – message to display
- **choices** (*list* of *tuple()* or *str*) – choices
- **ok_label** (*str*) – label for OK button (UNUSED)
- **cancel_label** (*str*) – label for Cancel button (UNUSED)
- **help_label** (*str*) – label for Help button (UNUSED)
- **default** (*int*) – default (non-interactive) choice from the menu
- **cli_flag** (*str*) – to automate choice from the menu, eg “--keep”
- **force_interactive** (*bool*) – True if it’s safe to prompt the user because it won’t cause any workflow regressions

Returns *tuple* of (*code*, *index*) where *code* - *str* display exit code *index* - *int* index of the user’s selection

Raises `errors.MissingCommandlineFlag` – if called in non-interactive mode without a default set

input (*message*, *default=None*, *cli_args=None*, *force_interactive=False*)
 Accept input from the user.

When not setting `force_interactive=True`, you must provide a default value.

Parameters

- **message** (*str*) – message to display to the user
- **default** (*str*) – default (non-interactive) response to prompt
- **force_interactive** (*bool*) – True if it’s safe to prompt the user because it won’t cause any workflow regressions

Returns *tuple* of (*code*, *input*) where *code* - *str* display exit code *input* - *str* of the user’s input

Return type *tuple*

Raises `errors.MissingCommandlineFlag` – if called in non-interactive mode without a default set

yesno (*message*, *yes_label='Yes'*, *no_label='No'*, *default=None*, *cli_args=None*, *force_interactive=False*)
 Query the user with a yes/no question.

Yes and No label must begin with different letters.

When not setting `force_interactive=True`, you must provide a default value.

Parameters

- **message** (*str*) – question for the user
- **default** (*str*) – default (non-interactive) choice from the menu
- **cli_flag** (*str*) – to automate choice from the menu, eg “--redirect / --no-redirect”
- **force_interactive** (*bool*) – True if it’s safe to prompt the user because it won’t cause any workflow regressions

Returns True for “Yes”, False for “No”

Return type *bool*

Raises *errors.MissingCommandlineFlag* – if called in non-interactive mode without a default set

checklist (*message, tags, default=None, cli_args=None, force_interactive=False*)

Allow for multiple selections from a menu.

When not setting *force_interactive=True*, you must provide a default value.

Parameters

- **message** (*str*) – message to display to the user
- **tags** (*list*) – where each is of type *str* *len(tags) > 0*
- **default** (*str*) – default (non-interactive) state of the checklist
- **cli_flag** (*str*) – to automate choice from the menu, eg “--domains”
- **force_interactive** (*bool*) – True if it’s safe to prompt the user because it won’t cause any workflow regressions

Returns tuple of the form (*code, list_tags*) where *code* - int display exit code *list_tags* - list of str tags selected by the user

Return type *tuple*

Raises *errors.MissingCommandlineFlag* – if called in non-interactive mode without a default set

directory_select (*self, message, default=None, cli_flag=None, force_interactive=False*)

Display a directory selection screen.

When not setting *force_interactive=True*, you must provide a default value.

Parameters

- **message** (*str*) – prompt to give the user
- **default** – the default value to return, if one exists, when using the *NoninteractiveDisplay*
- **cli_flag** (*str*) – option used to set this value with the CLI, if one exists, to be included in error messages given by *NoninteractiveDisplay*
- **force_interactive** (*bool*) – True if it’s safe to prompt the user because it won’t cause any workflow regressions

Returns tuple of the form (*code, string*) where *code* - int display exit code *string* - input entered by the user

interface *certbot.interfaces.IReporter*

Interface to collect and display information to the user.

HIGH_PRIORITY

Used to denote high priority messages

MEDIUM_PRIORITY

Used to denote medium priority messages

LOW_PRIORITY

Used to denote low priority messages

add_message (*self*, *msg*, *priority*, *on_crash=True*)

Adds msg to the list of messages to be printed.

Parameters

- **msg** (*str*) – Message to be displayed to the user.
- **priority** (*int*) – One of HIGH_PRIORITY, MEDIUM_PRIORITY, or LOW_PRIORITY.
- **on_crash** (*bool*) – Whether or not the message should be printed if the program exits abnormally.

print_messages (*self*)

Prints messages to the user and clears the message queue.

class certbot.interfaces.GenericUpdater

Bases: `object`

Interface for update types not currently specified by Certbot.

This class allows plugins to perform types of updates that Certbot hasn't defined (yet).

To make use of this interface, the installer should implement the interface methods, and `interfaces.GenericUpdater.register(InstallerClass)` should be called from the installer code.

The plugins implementing this enhancement are responsible of handling the saving of configuration checkpoints as well as other calls to interface methods of `interfaces.IInstaller` such as `prepare()` and `restart()`

generic_updates (*lineage*, **args*, ***kwargs*)

Perform any update types defined by the installer.

If an installer is a subclass of the class containing this method, this function will always be called when “certbot renew” is run. If the update defined by the installer should be run conditionally, the installer needs to handle checking the conditions itself.

This method is called once for each lineage.

Parameters lineage (*storage.RenewableCert*) – Certificate lineage object

class certbot.interfaces.RenewDeployer

Bases: `object`

Interface for update types run when a lineage is renewed

This class allows plugins to perform types of updates that need to run at lineage renewal that Certbot hasn't defined (yet).

To make use of this interface, the installer should implement the interface methods, and `interfaces.RenewDeployer.register(InstallerClass)` should be called from the installer code.

renew_deploy (*lineage*, **args*, ***kwargs*)

Perform updates defined by installer when a certificate has been renewed

If an installer is a subclass of the class containing this method, this function will always be called when a certificate has been renewed by running “certbot renew”. For example if a plugin needs to copy a certificate over, or change configuration based on the new certificate.

This method is called once for each lineage renewed

Parameters `lineage` (`storage.RenewableCert`) – Certificate lineage object

8.10 certbot.plugins.common

Internal class delegating to a module, and displaying warnings when attributes related to TLS-SNI-01 are accessed.

`certbot.plugins.common.option_namespace` (*name*)

ArgumentParser options namespace (prefix of all options).

`certbot.plugins.common.dest_namespace` (*name*)

ArgumentParser dest namespace (prefix of all destinations).

class `certbot.plugins.common.Plugin` (*config*, *name*)

Bases: `object`

Generic plugin.

classmethod `add_parser_arguments` (*add*)

Add plugin arguments to the CLI argument parser.

NOTE: If some of your flags interact with others, you can use `cli.report_config_interaction` to register this to ensure values are correctly saved/overridable during renewal.

Parameters `add` (*callable*) – Function that proxies calls to `argparse.ArgumentParser.add_argument` prepending options with unique plugin name prefix.

classmethod `inject_parser_options` (*parser*, *name*)

Inject parser options.

See `inject_parser_options` for docs.

option_namespace

ArgumentParser options namespace (prefix of all options).

option_name (*name*)

Option name (include plugin namespace).

dest_namespace

ArgumentParser dest namespace (prefix of all destinations).

dest (*var*)

Find a destination for given variable *var*.

conf (*var*)

Find a configuration value for variable *var*.

class `certbot.plugins.common.Installer` (**args*, ***kwargs*)

Bases: `certbot.plugins.common.Plugin`

An installer base class with reverter and `ssl_dhparam` methods defined.

Installer plugins do not have to inherit from this class.

add_to_checkpoint (*save_files*, *save_notes*, *temporary=False*)

Add files to a checkpoint.

Parameters

- **save_files** (*set*) – set of filepaths to save
- **save_notes** (*str*) – notes about changes during the save
- **temporary** (*bool*) – True if the files should be added to a temporary checkpoint rather than a permanent one. This is usually used for changes that will soon be reverted.

Raises **errors.PluginError** – when unable to add to checkpoint

finalize_checkpoint (*title*)

Timestamp and save changes made through the reverter.

Parameters **title** (*str*) – Title describing checkpoint

Raises **errors.PluginError** – when an error occurs

recovery_routine ()

Revert all previously modified files.

Reverts all modified files that have not been saved as a checkpoint

Raises **errors.PluginError** – If unable to recover the configuration

revert_temporary_config ()

Rollback temporary checkpoint.

Raises **errors.PluginError** – when unable to revert config

rollback_checkpoints (*rollback=1*)

Rollback saved checkpoints.

Parameters **rollback** (*int*) – Number of checkpoints to revert

Raises **errors.PluginError** – If there is a problem with the input or the function is unable to correctly revert the configuration

ssl_dhparams

Full absolute path to ssl_dhparams file.

updated_ssl_dhparams_digest

Full absolute path to digest of updated ssl_dhparams file.

install_ssl_dhparams ()

Copy Certbot's ssl_dhparams file into the system's config dir if required.

class certbot.plugins.common.**Addr** (*tup, ipv6=False*)

Bases: **object**

Represents an virtual host address.

Parameters

- **addr** (*str*) – addr part of vhost address
- **port** (*str*) – port number or *, or ""

classmethod **fromstring** (*str_addr*)

Initialize Addr from string.

normalized_tuple ()

Normalized representation of addr/port tuple

get_addr ()

Return addr part of Addr object.

get_port()
Return port.

get_addr_obj(port)
Return new address object with same addr and new port.

_normalize_ipv6(addr)
Return IPv6 address in normalized form, helper function

get_ipv6_exploded()
Return IPv6 in normalized form

_explode_ipv6(addr)
Explode IPv6 address for comparison

class certbot.plugins.common.ChallengePerformer(*configurator*)

Bases: `object`

Abstract base for challenge performers.

Variables

- **configurator** – Authenticator and installer plugin
- **achalls** (list of `KeyAuthorizationAnnotatedChallenge`) – Annotated challenges
- **indices** (list of `int`) – Holds the indices of challenges from a larger array so the user of the class doesn't have to.

add_chall(achall, idx=None)
Store challenge to be performed when perform() is called.

Parameters

- **achall** (`KeyAuthorizationAnnotatedChallenge`) – Annotated challenge.
- **idx** (`int`) – index to challenge in a larger array

perform()
Perform all added challenges.

Returns challenge responses

Return type list of `acme.challenges.KeyAuthorizationChallengeResponse`

certbot.plugins.common.install_version_controlled_file(*dest_path, digest_path, src_path, all_hashes*)

Copy a file into an active location (likely the system's config dir) if required.

Parameters

- **dest_path** (`str`) – destination path for version controlled file
- **digest_path** (`str`) – path to save a digest of the file in
- **src_path** (`str`) – path to version controlled file found in distribution
- **all_hashes** (list) – hashes of every released version of the file

certbot.plugins.common.dir_setup(*test_dir, pkg*)
Setup the directories necessary for the configurator.

class certbot.plugins.common._TLSSNI01DeprecationModule(*module*)

Bases: `object`

Internal class delegating to a module, and displaying warnings when attributes related to TLS-SNI-01 are accessed.

8.11 certbot.plugins.dns_common

Common code for DNS Authenticator Plugins.

class certbot.plugins.dns_common.DNSAuthenticator(*config, name*)

Bases: *certbot.plugins.common.Plugin*

Base class for DNS Authenticators

classmethod add_parser_arguments(*add, default_propagation_seconds=10*)

Add plugin arguments to the CLI argument parser.

NOTE: If some of your flags interact with others, you can use `cli.report_config_interaction` to register this to ensure values are correctly saved/overridable during renewal.

Parameters *add* (*callable*) – Function that proxies calls to `argparse.ArgumentParser.add_argument` prepending options with unique plugin name prefix.

_setup_credentials ()

Establish credentials, prompting if necessary.

_perform (*domain, validation_name, validation*)

Performs a dns-01 challenge by creating a DNS TXT record.

Parameters

- **domain** (*str*) – The domain being validated.
- **validation_domain_name** (*str*) – The validation record domain name.
- **validation** (*str*) – The validation record content.

Raises *errors.PluginError* – If the challenge cannot be performed

_cleanup (*domain, validation_name, validation*)

Deletes the DNS TXT record which would have been created by `_perform_achallenge`.

Fails gracefully if no such record exists.

Parameters

- **domain** (*str*) – The domain being validated.
- **validation_domain_name** (*str*) – The validation record domain name.
- **validation** (*str*) – The validation record content.

_configure (*key, label*)

Ensure that a configuration value is available.

If necessary, prompts the user and stores the result.

Parameters

- **key** (*str*) – The configuration key.
- **label** (*str*) – The user-friendly label for this piece of information.

`_configure_file` (*key*, *label*, *validator=None*)
 Ensure that a configuration value is available for a path.
 If necessary, prompts the user and stores the result.

Parameters

- **`key`** (*str*) – The configuration key.
- **`label`** (*str*) – The user-friendly label for this piece of information.

`_configure_credentials` (*key*, *label*, *required_variables=None*, *validator=None*)
 As `_configure_file`, but for a credential configuration file.

If necessary, prompts the user and stores the result.
 Always stores absolute paths to avoid issues during renewal.

Parameters

- **`key`** (*str*) – The configuration key.
- **`label`** (*str*) – The user-friendly label for this piece of information.
- **`required_variables`** (*dict*) – Map of variable which must be present to error to display.
- **`validator`** (*callable*) – A method which will be called to validate the `CredentialsConfiguration` resulting from the supplied input after it has been validated to contain the `required_variables`. Should throw a `PluginError` to indicate any issue.

`static _prompt_for_data` (*label*)
 Prompt the user for a piece of information.

Parameters **`label`** (*str*) – The user-friendly label for this piece of information.

Returns The user’s response (guaranteed non-empty).

Return type *str*

`static _prompt_for_file` (*label*, *validator=None*)
 Prompt the user for a path.

Parameters

- **`label`** (*str*) – The user-friendly label for the file.
- **`validator`** (*callable*) – A method which will be called to validate the supplied input after it has been validated to be a non-empty path to an existing file. Should throw a `PluginError` to indicate any issue.

Returns The user’s response (guaranteed to exist).

Return type *str*

`class certbot.plugins.dns_common.CredentialsConfiguration` (*filename*, *map_per=<function <lambda>>*)

Bases: *object*

Represents a user-supplied file which stores API credentials.

`require` (*required_variables*)
 Ensures that the supplied set of variables are all present in the file.

Parameters `required_variables` (*dict*) – Map of variable which must be present to error to display.

Raises `errors.PluginError` – If one or more are missing.

conf (*var*)

Find a configuration value for variable *var*, as transformed by mapper.

Parameters `var` (*str*) – The variable to get.

Returns The value of the variable.

Return type *str*

`certbot.plugins.dns_common.validate_file(filename)`

Ensure that the specified file exists.

`certbot.plugins.dns_common.validate_file_permissions(filename)`

Ensure that the specified file exists and warn about unsafe permissions.

`certbot.plugins.dns_common.base_domain_name_guesses(domain)`

Return a list of progressively less-specific domain names.

One of these will probably be the domain name known to the DNS provider.

Example

```
>>> base_domain_name_guesses('foo.bar.baz.example.com')
['foo.bar.baz.example.com', 'bar.baz.example.com', 'baz.example.com', 'example.com',
 'com']
```

Parameters `domain` (*str*) – The domain for which to return guesses.

Returns The a list of less specific domain names.

Return type *list*

8.12 certbot.plugins.dns_common_lexicon

Common code for DNS Authenticator Plugins built on Lexicon.

class `certbot.plugins.dns_common_lexicon.LexiconClient`

Bases: `object`

Encapsulates all communication with a DNS provider via Lexicon.

add_txt_record (*domain, record_name, record_content*)

Add a TXT record using the supplied information.

Parameters

- **domain** (*str*) – The domain to use to look up the managed zone.
- **record_name** (*str*) – The record name (typically beginning with ‘_acme-challenge.’).
- **record_content** (*str*) – The record content (typically the challenge validation).

Raises `errors.PluginError` – if an error occurs communicating with the DNS Provider API

del_txt_record (*domain, record_name, record_content*)

Delete a TXT record using the supplied information.

Parameters

- **domain** (*str*) – The domain to use to look up the managed zone.
- **record_name** (*str*) – The record name (typically beginning with ‘_acme-challenge.’).
- **record_content** (*str*) – The record content (typically the challenge validation).

Raises `errors.PluginError` – if an error occurs communicating with the DNS Provider API

_find_domain_id (*domain*)

Find the domain_id for a given domain.

Parameters **domain** (*str*) – The domain for which to find the domain_id.

Raises `errors.PluginError` – if the domain_id cannot be found.

`certbot.plugins.dns_common_lexicon.build_lexicon_config` (*lexicon_provider_name*,
lexicon_options,
provider_options)

Convenient function to build a Lexicon 2.x/3.x config object. :param str lexicon_provider_name: the name of the lexicon provider to use :param dict lexicon_options: options specific to lexicon :param dict provider_options: options specific to provider :return: configuration to apply to the provider :rtype: ConfigurationResolver or dict

8.13 certbot.plugins.util

Plugin utilities.

`certbot.plugins.util.get_prefixes` (*path*)

Retrieves all possible path prefixes of a path, in descending order of length. For instance,

(linux) /a/b/c returns [‘/a/b/c’, ‘/a/b’, ‘/a’, ‘/’] (windows) C:abc returns [‘C:abc’, ‘C:ab’, ‘C:a’, ‘C:’]

Parameters **path** (*str*) – the path to break into prefixes

Returns all possible path prefixes of given path in descending order

Return type `list` of `str`

`certbot.plugins.util.path_surgery` (*cmd*)

Attempt to perform PATH surgery to find cmd

Mitigates <https://github.com/certbot/certbot/issues/1833>

Parameters **cmd** (*str*) – the command that is being searched for in the PATH

Returns True if the operation succeeded, False otherwise

8.14 certbot.reverter

Reverter class saves configuration checkpoints and allows for recovery.

class `certbot.reverter.Reverter` (*config*)

Bases: `object`

Reverter Class - save and revert configuration checkpoints.

This class can be used by the plugins, especially Installers, to undo changes made to the user’s system. Modifications to files and commands to do undo actions taken by the plugin should be registered with this class before the action is taken.

Once a change has been registered with this class, there are three states the change can be in. First, the change can be a temporary change. This should be used for changes that will soon be reverted, such as config changes for the purpose of solving a challenge. Changes are added to this state through calls to `add_to_temp_checkpoint()` and reverted when `revert_temporary_config()` or `recovery_routine()` is called.

The second state a change can be in is in progress. These changes are not temporary, however, they also have not been finalized in a checkpoint. A change must become in progress before it can be finalized. Changes are added to this state through calls to `add_to_checkpoint()` and reverted when `recovery_routine()` is called.

The last state a change can be in is finalized in a checkpoint. A change is put into this state by first becoming an in progress change and then calling `finalize_checkpoint()`. Changes in this state can be reverted through calls to `rollback_checkpoints()`.

As a final note, creating new files and registering undo commands are handled specially and use the methods `register_file_creation()` and `register_undo_command()` respectively. Both of these methods can be used to create either temporary or in progress changes.

Note: Consider moving everything over to CSV format.

Parameters `config` (`certbot.interfaces.IConfig`) – Configuration.

revert_temporary_config()

Reload users original configuration files after a temporary save.

This function should reinstall the users original configuration files for all saves with temporary=True

Raises `ReverterError` – when unable to revert config

rollback_checkpoints (`rollback=1`)

Revert ‘rollback’ number of configuration checkpoints.

Parameters `rollback` (`int`) – Number of checkpoints to reverse. A str num will be cast to an integer. So “2” is also acceptable.

Raises `ReverterError` – if there is a problem with the input or if the function is unable to correctly revert the configuration checkpoints

add_to_temp_checkpoint (`save_files`, `save_notes`)

Add files to temporary checkpoint.

Parameters

- `save_files` (`set`) – set of filepaths to save
- `save_notes` (`str`) – notes about changes during the save

add_to_checkpoint (`save_files`, `save_notes`)

Add files to a permanent checkpoint.

Parameters

- `save_files` (`set`) – set of filepaths to save
- `save_notes` (`str`) – notes about changes during the save

`_add_to_checkpoint_dir` (*cp_dir*, *save_files*, *save_notes*)

Add save files to checkpoint directory.

Parameters

- **`cp_dir`** (*str*) – Checkpoint directory filepath
- **`save_files`** (*set*) – set of files to save
- **`save_notes`** (*str*) – notes about changes made during the save

Raises

- **`IOError`** – if unable to open `cp_dir` + `FILEPATHS` file
- **`ReverterError`** – if unable to add checkpoint

`_read_and_append` (*filepath*)

Reads the file lines and returns a file obj.

Read the file returning the lines, and a pointer to the end of the file.

`_recover_checkpoint` (*cp_dir*)

Recover a specific checkpoint.

Recover a specific checkpoint provided by `cp_dir` Note: this function does not reload Augeas.

Parameters **`cp_dir`** (*str*) – checkpoint directory file path

Raises **`errors.ReverterError`** – If unable to recover checkpoint

`_run_undo_commands` (*filepath*)

Run all commands in a file.

`_check_tempfile_saves` (*save_files*)

Verify save isn't overwriting any temporary files.

Parameters **`save_files`** (*set*) – Set of files about to be saved.

Raises **`certbot.errors.ReverterError`** – when save is attempting to overwrite a temporary file.

`register_file_creation` (*temporary*, **files*)

Register the creation of all files during certbot execution.

Call this method before writing to the file to make sure that the file will be cleaned up if the program exits unexpectedly. (Before a save occurs)

Parameters

- **`temporary`** (*bool*) – If the file creation registry is for a temp or permanent save.
- **`*files`** – file paths (*str*) to be registered

Raises **`certbot.errors.ReverterError`** – If call does not contain necessary parameters or if the file creation is unable to be registered.

`register_undo_command` (*temporary*, *command*)

Register a command to be run to undo actions taken.

Warning: This function does not enforce order of operations in terms of file modification vs. command registration. All undo commands are run first before all normal files are reverted to their previous state. If you need to maintain strict order, you may create checkpoints before and after the the command registration. This function may be improved in the future based on demand.

Parameters

- **temporary** (*bool*) – Whether the command should be saved in the IN_PROGRESS or TEMPORARY checkpoints.
- **command** (*list of str*) – Command to be run.

_get_cp_dir (*temporary*)

Return the proper reverter directory.

recovery_routine ()

Revert configuration to most recent finalized checkpoint.

Remove all changes (temporary and permanent) that have not been finalized. This is useful to protect against crashes and other execution interruptions.

Raises *errors.ReverterError* – If unable to recover the configuration

_remove_contained_files (*file_list*)

Erase all files contained within file_list.

Parameters **file_list** (*str*) – file containing list of file paths to be deleted

Returns Success

Return type *bool*

Raises *certbot.errors.ReverterError* – If all files within file_list cannot be removed

finalize_checkpoint (*title*)

Finalize the checkpoint.

Timestamps and permanently saves all changes made through the use of *add_to_checkpoint()* and *register_file_creation()*

Parameters **title** (*str*) – Title describing checkpoint

Raises *certbot.errors.ReverterError* – when the checkpoint is not able to be finalized.

_checkpoint_timestamp ()

Determine the timestamp of the checkpoint, enforcing monotonicity.

_timestamp_progress_dir ()

Timestamp the checkpoint.

8.15 certbot.util

Utilities for all Certbot.

class *certbot.util.Key* (*file, pem*)

Bases: *tuple*

_asdict ()

Return a new OrderedDict which maps field names to their values

classmethod **_make** (*iterable, new=<built-in method __new__ of type object at 0x8f9920>, len=<built-in function len>*)

Make a new Key object from a sequence or iterable

_replace (***kws*)

Return a new Key object replacing specified fields with new values

file
Alias for field number 0

pem
Alias for field number 1

class `certbot.util.CSR` (*file, data, form*)

Bases: `tuple`

__asdict ()
Return a new OrderedDict which maps field names to their values

classmethod **__make** (*iterable, new=<built-in method __new__ of type object at 0x8f9920>, len=<built-in function len>*)
Make a new CSR object from a sequence or iterable

__replace (***kws*)
Return a new CSR object replacing specified fields with new values

data
Alias for field number 1

file
Alias for field number 0

form
Alias for field number 2

`certbot.util.run_script` (*params, log=<bound method Logger.error of <logging.Logger object>>*)
Run the script with the given params.

Parameters

- **params** (*list*) – List of parameters to pass to Popen
- **log** (*callable*) – Logger method to use for errors

`certbot.util.exe_exists` (*exe*)
Determine whether path/name refers to an executable.

Parameters **exe** (*str*) – Executable path or name

Returns If exe is a valid executable

Return type `bool`

`certbot.util.lock_dir_until_exit` (*dir_path*)
Lock the directory at dir_path until program exit.

Parameters **dir_path** (*str*) – path to directory

Raises `errors.LockError` – if the lock is held by another process

`certbot.util.set_up_core_dir` (*directory, mode, strict*)
Ensure directory exists with proper permissions and is locked.

Parameters

- **directory** (*str*) – Path to a directory.
- **mode** (*int*) – Directory mode.
- **strict** (*bool*) – require directory to be owned by current user

Raises

- `errors.LockError` – if the directory cannot be locked

- **`errors.Error`** – if the directory cannot be made or verified

`certbot.util.make_or_verify_dir(directory, mode=493, strict=False)`
 Make sure directory exists with proper permissions.

Parameters

- **`directory`** (*str*) – Path to a directory.
- **`mode`** (*int*) – Directory mode.
- **`strict`** (*bool*) – require directory to be owned by current user

Raises

- **`errors.Error`** – if a directory already exists, but has wrong permissions or owner
- **`OSError`** – if invalid or inaccessible file names and paths, or other arguments that have the correct type, but are not accepted by the operating system.

`certbot.util.safe_open(path, mode='w', chmod=None)`
 Safely open a file.

Parameters

- **`path`** (*str*) – Path to a file.
- **`mode`** (*str*) – Same os mode for `open`.
- **`chmod`** (*int*) – Same as mode for `filesystem.open`, uses Python defaults if `None`.

`certbot.util.unique_file(path, chmod=511, mode='w')`
 Safely finds a unique file.

Parameters

- **`path`** (*str*) – path/filename.ext
- **`chmod`** (*int*) – File mode
- **`mode`** (*str*) – Open mode

Returns tuple of file object and file name

`certbot.util.unique_lineage_name(path, filename, chmod=420, mode='w')`
 Safely finds a unique file using lineage convention.

Parameters

- **`path`** (*str*) – directory path
- **`filename`** (*str*) – proposed filename
- **`chmod`** (*int*) – file mode
- **`mode`** (*str*) – open mode

Returns tuple of file object and file name (which may be modified from the requested one by appending digits to ensure uniqueness)

Raises **`OSError`** – if writing files fails for an unanticipated reason, such as a full disk or a lack of permission to write to specified location.

`certbot.util.safely_remove(path)`
 Remove a file that may not exist.

`certbot.util.get_filtered_names(all_names)`
 Removes names that aren't considered valid by Let's Encrypt.

Parameters `all_names` (*set*) – all names found in the configuration

Returns all found names that are considered valid by LE

Return type *set*

`certbot.util.get_os_info()`

Get OS name and version

Returns (`os_name`, `os_version`)

Return type *tuple of str*

`certbot.util.get_os_info_ua()`

Get OS name and version string for User Agent

Returns `os_ua`

Return type *str*

`certbot.util.get_systemd_os_like()`

Get a list of strings that indicate the distribution likeness to other distributions.

Returns List of distribution acronyms

Return type *list of str*

`certbot.util.get_var_from_file(varname, filepath='/etc/os-release')`

Get single value from a file formatted like systemd /etc/os-release

Parameters

- **varname** (*str*) – Name of variable to fetch
- **filepath** (*str*) – File path of os-release file

Returns requested value

Return type *str*

`certbot.util._normalize_string(orig)`

Helper function for `get_var_from_file()` to remove quotes and whitespaces

`certbot.util.get_python_os_info(pretty=False)`

Get Operating System type/distribution and major version using python platform module

Parameters **pretty** (*bool*) – If the returned OS name should be in longer (pretty) form

Returns (`os_name`, `os_version`)

Return type *tuple of str*

`certbot.util.safe_email(email)`

Scrub email address before using it.

class `certbot.util._ShowWarning` (*option_strings*, *dest*, *nargs=None*, *const=None*, *default=None*, *type=None*, *choices=None*, *required=False*, *help=None*, *metavar=None*)

Bases: `argparse.Action`

Action to log a warning when an argument is used.

`certbot.util.add_deprecated_argument(add_argument, argument_name, nargs)`

Adds a deprecated argument with the name `argument_name`.

Deprecated arguments are not shown in the help. If they are used on the command line, a warning is shown stating that the argument is deprecated and no other action is taken.

Parameters

- **add_argument** (*callable*) – Function that adds arguments to an argument parser/group.
- **argument_name** (*str*) – Name of deprecated argument.
- **nargs** – Value for nargs when adding the argument to argparse.

`certbot.util.enforce_le_validity(domain)`

Checks that Let's Encrypt will consider domain to be valid.

Parameters **domain** (*str* or *unicode*) – FQDN to check

Returns The domain cast to *str*, with ASCII-only contents

Return type *str*

Raises *ConfigurationError* – for invalid domains and cases where Let's Encrypt currently will not issue certificates

`certbot.util.enforce_domain_sanity(domain)`

Method which validates domain value and errors out if the requirements are not met.

Parameters **domain** (*str* or *unicode*) – Domain to check

Raises *ConfigurationError* – for invalid domains and cases where Let's Encrypt currently will not issue certificates

Returns The domain cast to *str*, with ASCII-only contents

Return type *str*

`certbot.util.is_wildcard_domain(domain)`

“Is domain a wildcard domain?”

Parameters **domain** (*bytes* or *str* or *unicode*) – domain to check

Returns True if domain is a wildcard, otherwise, False

Return type *bool*

`certbot.util.get_strict_version(normalized)`

Converts a normalized version to a strict version.

Parameters **normalized** (*str*) – normalized version string

Returns An equivalent strict version

Return type `distutils.version.StrictVersion`

`certbot.util.is_staging(srv)`

Determine whether a given ACME server is a known test / staging server.

Parameters **srv** (*str*) – the URI for the ACME server

Returns True iff srv is a known test / staging server

Rtype *bool*

`certbot.util.atexit_register(func, *args, **kwargs)`

Sets func to be called before the program exits.

Special care is taken to ensure func is only called when the process that first imports this module exits rather than any child processes.

Parameters **func** (*function*) – function to be called in case of an error

INDICES AND TABLES

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

PYTHON MODULE INDEX

C

- `certbot`, [76](#)
- `certbot.achallenges`, [55](#)
- `certbot.cli`, [56](#)
- `certbot.configuration`, [60](#)
- `certbot.crypto_util`, [61](#)
- `certbot.display`, [65](#)
- `certbot.display.ops`, [70](#)
- `certbot.display.util`, [65](#)
- `certbot.errors`, [72](#)
- `certbot.hooks`, [74](#)
- `certbot.interfaces`, [76](#)
- `certbot.plugins.common`, [85](#)
- `certbot.plugins.dns_common`, [88](#)
- `certbot.plugins.dns_common_lexicon`, [90](#)
- `certbot.plugins.util`, [91](#)
- `certbot.reverter`, [91](#)
- `certbot.util`, [94](#)

Symbols

- `_Default` (class in `certbot.cli`), 56
- `_DeployHookAction` (class in `certbot.cli`), 60
- `_DomainsAction` (class in `certbot.cli`), 59
- `_EncodeReasonAction` (class in `certbot.cli`), 59
- `_PrefChallAction` (class in `certbot.cli`), 59
- `_RenewHookAction` (class in `certbot.cli`), 60
- `_ShowWarning` (class in `certbot.util`), 97
- `_TLSSNI01DeprecationModule` (class in `certbot.plugins.common`), 87
- `__call__()` (`certbot.interfaces.IPluginFactory` method), 77
- `_add_to_checkpoint_dir()` (`certbot.reverter.Reverter` method), 92
- `_asdict()` (`certbot.util.CSR` method), 95
- `_asdict()` (`certbot.util.Key` method), 94
- `_can_interact()` (`certbot.display.util.FileDisplay` method), 67
- `_check_tempfile_saves()` (`certbot.reverter.Reverter` method), 93
- `_checkpoint_timestamp()` (`certbot.reverter.Reverter` method), 94
- `_choose_names_manually()` (in module `certbot.display.ops`), 71
- `_cleanup()` (`certbot.plugins.dns_common.DNSAuthenticator` method), 88
- `_configure()` (`certbot.plugins.dns_common.DNSAuthenticator` method), 88
- `_configure_credentials()` (`certbot.plugins.dns_common.DNSAuthenticator` method), 89
- `_configure_file()` (`certbot.plugins.dns_common.DNSAuthenticator` method), 88
- `_explode_ipv6()` (`certbot.plugins.common.Addr` method), 87
- `_filter_names()` (in module `certbot.display.ops`), 71
- `_find_domain_id()` (`certbot.plugins.dns_common_lexicon.LexiconClient` method), 91
- `_gen_https_names()` (in module `certbot.display.ops`), 72
- `_gen_ssl_lab_urls()` (in module `certbot.display.ops`), 72
- `_get_cp_dir()` (`certbot.reverter.Reverter` method), 94
- `_get_valid_int_ans()` (`certbot.display.util.FileDisplay` method), 68
- `_interaction_fail()` (`certbot.display.util.NoninteractiveDisplay` method), 69
- `_make()` (`certbot.util.CSR` class method), 95
- `_make()` (`certbot.util.Key` class method), 94
- `_normalize_ipv6()` (`certbot.plugins.common.Addr` method), 87
- `_normalize_string()` (in module `certbot.util`), 97
- `_notAfterBefore()` (in module `certbot.crypto_util`), 64
- `_parens_around_char()` (in module `certbot.display.util`), 70
- `_perform()` (`certbot.plugins.dns_common.DNSAuthenticator` method), 88
- `_print_menu()` (`certbot.display.util.FileDisplay` method), 68
- `_prog()` (in module `certbot.hooks`), 74
- `_prompt_for_data()` (`certbot.plugins.dns_common.DNSAuthenticator` static method), 89
- `_prompt_for_file()` (`certbot.plugins.dns_common.DNSAuthenticator` static method), 89
- `_read_and_append()` (`certbot.reverter.Reverter` method), 93
- `_recover_checkpoint()` (`certbot.reverter.Reverter` method), 93
- `_remove_contained_files()` (`certbot.reverter.Reverter` method), 94
- `_replace()` (`certbot.util.CSR` method), 95
- `_replace()` (`certbot.util.Key` method), 94
- `_return_default()` (`certbot.display.util.FileDisplay` method), 67
- `_run_deploy_hook()` (in module `certbot.hooks`), 76
- `_run_eventually()` (in module `certbot.hooks`), 75
- `_run_hook()` (in module `certbot.hooks`), 76
- `_run_pre_hook_if_necessary()` (in module `certbot.hooks`), 75
- `_run_undo_commands()` (`certbot.reverter.Reverter` method), 93
- `_scrub_checklist_input()` (`certbot.display.util.FileDisplay` method), 68

`_setup_credentials()` (certbot.plugins.dns_common.DNSAuthenticator method), 88
`_sort_names()` (in module certbot.display.ops), 71
`_timestamp_progress_dir()` (certbot.reverter.Reverter method), 94
`_usage_string()` (certbot.cli.HelpfulArgumentParser method), 57
`_wrap_lines()` (in module certbot.display.util), 65

A

`AccountNotFound`, 73
`accounts_dir` (certbot.interfaces.IConfig attribute), 79
`accounts_dir_for_server_path()` (certbot.configuration.NamespaceConfig method), 61
`AccountStorage` (class in certbot.interfaces), 76
`AccountStorageError`, 73
`acme_type` (certbot.achallenges.DNS attribute), 55
`add()` (certbot.cli.HelpfulArgumentParser method), 58
`add_argument()` (certbot.cli.HelpfulArgumentGroup method), 57
`add_chall()` (certbot.plugins.common.ChallengePerformer method), 87
`add_deprecated_argument()` (certbot.cli.HelpfulArgumentParser method), 58
`add_deprecated_argument()` (in module certbot.util), 97
`add_domains()` (in module certbot.cli), 59
`add_group()` (certbot.cli.HelpfulArgumentParser method), 58
`add_message()` (certbot.interfaces.IReporter method), 84
`add_parser_arguments()` (certbot.plugins.common.Plugin class method), 85
`add_parser_arguments()` (certbot.plugins.dns_common.DNSAuthenticator class method), 88
`add_plugin_args()` (certbot.cli.HelpfulArgumentParser method), 58
`add_to_checkpoint()` (certbot.plugins.common.Installer method), 85
`add_to_checkpoint()` (certbot.reverter.Reverter method), 92
`add_to_temp_checkpoint()` (certbot.reverter.Reverter method), 92
`add_txt_record()` (certbot.plugins.dns_common_lexicon.LexiconClient method), 90
`Addr` (class in certbot.plugins.common), 86
`allow_subset_of_names` (certbot.interfaces.IConfig attribute), 80
`AnnotatedChallenge` (class in certbot.achallenges), 55
`argparse_type()` (in module certbot.cli), 56
`assert_valid_call()` (in module certbot.display.util), 68
`atexit_register()` (in module certbot.util), 98

`AuthorizationError`, 73

B

`backup_dir` (certbot.interfaces.IConfig attribute), 79
`base_domain_name_guesses()` (in module certbot.plugins.dns_common), 90
`build_lexicon_config()` (in module certbot.plugins.dns_common_lexicon), 91

C

`CANCEL` (in module certbot.display.util), 65
`CaseInsensitiveList` (class in certbot.cli), 59
`cert_and_chain_from_fullchain()` (in module certbot.crypto_util), 65
`certbot` (module), 76
`certbot.achallenges` (module), 55
`certbot.cli` (module), 56
`certbot.configuration` (module), 60
`certbot.crypto_util` (module), 61
`certbot.display` (module), 65
`certbot.display.ops` (module), 70
`certbot.display.util` (module), 65
`certbot.errors` (module), 72
`certbot.hooks` (module), 74
`certbot.interfaces` (module), 76
`certbot.plugins.common` (module), 85
`certbot.plugins.dns_common` (module), 88
`certbot.plugins.dns_common_lexicon` (module), 90
`certbot.plugins.util` (module), 91
`certbot.reverter` (module), 91
`certbot.util` (module), 94
`CertStorageError`, 73
`ChallengePerformer` (class in certbot.plugins.common), 87
`check_config_sanity()` (in module certbot.configuration), 61
`checklist()` (certbot.display.util.FileDisplay method), 67
`checklist()` (certbot.display.util.NoninteractiveDisplay method), 70
`checklist()` (certbot.interfaces.IDisplay method), 83
`choose_account()` (in module certbot.display.ops), 71
`choose_names()` (in module certbot.display.ops), 71
`choose_values()` (in module certbot.display.ops), 71
`cleanup()` (certbot.interfaces.IAuthenticator method), 79
`conf()` (certbot.plugins.common.Plugin method), 85
`ConfClient` (certbot.plugins.dns_common.CredentialsConfiguration method), 90
`config_dir` (certbot.interfaces.IConfig attribute), 79
`config_help()` (in module certbot.cli), 57
`config_test()` (certbot.interfaces.IInstaller method), 81
`ConfigurationError`, 74
`CredentialsConfiguration` (class in certbot.plugins.dns_common), 89
`CSR` (class in certbot.util), 95

csr_dir (certbot.interfaces.IConfig attribute), 79
 csr_matches_pubkey() (in module certbot.crypto_util), 62
 CustomHelpFormatter (class in certbot.cli), 57

D

data (certbot.util.CSR attribute), 95
 del_txt_record() (certbot.plugins.dns_common_lexicon.Lexicon method), 90
 deploy_cert() (certbot.interfaces.IInstaller method), 80
 deploy_hook() (in module certbot.hooks), 75
 description (certbot.interfaces.IPluginFactory attribute), 77
 dest() (certbot.plugins.common.Plugin method), 85
 dest_namespace (certbot.plugins.common.Plugin attribute), 85
 dest_namespace() (in module certbot.plugins.common), 85
 determine_help_topics() (certbot.cli.HelpfulArgumentParser method), 59
 determine_verb() (certbot.cli.HelpfulArgumentParser method), 57
 dir_setup() (in module certbot.plugins.common), 87
 directory_select() (certbot.display.util.FileDisplay method), 68
 directory_select() (certbot.display.util.NoninteractiveDisplay method), 70
 directory_select() (certbot.interfaces.IDisplay method), 83
 disable_renew_updates (certbot.interfaces.IConfig attribute), 80
 DNS (class in certbot.achallenges), 55
 DNSAuthenticator (class in certbot.plugins.dns_common), 88
 dump_pyopenssl_chain() (in module certbot.crypto_util), 64

E

email (certbot.interfaces.IConfig attribute), 79
 enforce_domain_sanity() (in module certbot.util), 98
 enforce_le_validity() (in module certbot.util), 98
 enhance() (certbot.interfaces.IInstaller method), 80
 Error, 72
 ESC (in module certbot.display.util), 65
 exe_exists() (in module certbot.util), 95
 execute() (in module certbot.hooks), 76

F

FailedChallenges, 73
 file (certbot.util.CSR attribute), 95
 file (certbot.util.Key attribute), 94
 FileDisplay (class in certbot.display.util), 66

finalize_checkpoint() (certbot.plugins.common.Installer method), 86
 finalize_checkpoint() (certbot.reverter.Reverter method), 94
 find_all() (certbot.interfaces.AccountStorage method), 76
 flag_default() (in module certbot.cli), 57
 form (certbot.util.CSR attribute), 95
 fromstring() (certbot.plugins.common.Addr class method), 86

G

generic_updates() (certbot.interfaces.GenericUpdater method), 84
 GenericUpdater (class in certbot.interfaces), 84
 get_addr() (certbot.plugins.common.Addr method), 86
 get_addr_obj() (certbot.plugins.common.Addr method), 87
 get_all_names() (certbot.interfaces.IInstaller method), 80
 get_chall_pref() (certbot.interfaces.IAuthenticator method), 78
 get_email() (in module certbot.display.ops), 70
 get_filtered_names() (in module certbot.util), 96
 get_ipv6_explored() (certbot.plugins.common.Addr method), 87
 get_names_from_cert() (in module certbot.crypto_util), 64
 get_os_info() (in module certbot.util), 97
 get_os_info_ua() (in module certbot.util), 97
 get_port() (certbot.plugins.common.Addr method), 86
 get_prefixes() (in module certbot.plugins.util), 91
 get_python_os_info() (in module certbot.util), 97
 get_sans_from_cert() (in module certbot.crypto_util), 64
 get_strict_version() (in module certbot.util), 98
 get_systemd_os_like() (in module certbot.util), 97
 get_valid_domains() (in module certbot.display.ops), 71
 get_var_from_file() (in module certbot.util), 97

H

handle_csr() (certbot.cli.HelpfulArgumentParser method), 57
 has_default_value() (in module certbot.cli), 56
 HELP (in module certbot.display.util), 65
 HelpfulArgumentGroup (class in certbot.cli), 57
 HelpfulArgumentParser (class in certbot.cli), 57
 HIGH_PRIORITY (certbot.interfaces.IReporter attribute), 83
 HookCommandNotFound, 73
 http01_address (certbot.interfaces.IConfig attribute), 79
 http01_port (certbot.interfaces.IConfig attribute), 79
 https_port (certbot.interfaces.IConfig attribute), 80

I

IAuthenticator (interface in certbot.interfaces), 78
 IConfig (interface in certbot.interfaces), 79

IDisplay (interface in certbot.interfaces), 81
 IInstaller (interface in certbot.interfaces), 80
 import_csr_file() (in module certbot.crypto_util), 62
 in_progress_dir (certbot.interfaces.IConfig attribute), 79
 init_save_csr() (in module certbot.crypto_util), 62
 init_save_key() (in module certbot.crypto_util), 61
 inject_parser_options() (certbot.interfaces.IPluginFactory method), 77
 inject_parser_options() (certbot.plugins.common.Plugin class method), 85
 input() (certbot.display.util.FileDisplay method), 66
 input() (certbot.display.util.NoninteractiveDisplay method), 69
 input() (certbot.interfaces.IDisplay method), 82
 input_with_timeout() (in module certbot.display.util), 65
 install_ssl_dhparams() (certbot.plugins.common.Installer method), 86
 install_version_controlled_file() (in module certbot.plugins.common), 87
 Installer (class in certbot.plugins.common), 85
 IPlugin (interface in certbot.interfaces), 78
 IPluginFactory (interface in certbot.interfaces), 77
 IReporter (interface in certbot.interfaces), 83
 is_staging() (in module certbot.util), 98
 is_wildcard_domain() (in module certbot.util), 98

K

Key (class in certbot.util), 94
 key_dir (certbot.interfaces.IConfig attribute), 79
 KeyAuthorizationAnnotatedChallenge (class in certbot.achallenges), 55

L

LexiconClient (class in certbot.plugins.dns_common_lexicon), 90
 list_hooks() (in module certbot.hooks), 76
 load() (certbot.interfaces.AccountStorage method), 77
 lock_dir_until_exit() (in module certbot.util), 95
 LockError, 73
 LOW_PRIORITY (certbot.interfaces.IReporter attribute), 84

M

make_key() (in module certbot.crypto_util), 62
 make_or_verify_dir() (in module certbot.util), 96
 MEDIUM_PRIORITY (certbot.interfaces.IReporter attribute), 84
 menu() (certbot.display.util.FileDisplay method), 66
 menu() (certbot.display.util.NoninteractiveDisplay method), 69
 menu() (certbot.interfaces.IDisplay method), 82
 MisconfigurationError, 74
 MissingCommandlineFlag, 74

modify_kwargs_for_default_detection() (certbot.cli.HelpfulArgumentParser method), 58
 more_info() (certbot.interfaces.IPlugin method), 78
 must_staple (certbot.interfaces.IConfig attribute), 79

N

NamespaceConfig (class in certbot.configuration), 60
 no_verify_ssl (certbot.interfaces.IConfig attribute), 79
 NoInstallationError, 74
 NoninteractiveDisplay (class in certbot.display.util), 69
 nonnegative_int() (in module certbot.cli), 60
 normalized_tuple() (certbot.plugins.common.Addr method), 86
 notAfter() (in module certbot.crypto_util), 64
 notBefore() (in module certbot.crypto_util), 64
 notification() (certbot.display.util.FileDisplay method), 66
 notification() (certbot.display.util.NoninteractiveDisplay method), 69
 notification() (certbot.interfaces.IDisplay method), 81
 NotSupportedError, 74

O

OK (in module certbot.display.util), 65
 option_name() (certbot.plugins.common.Plugin method), 85
 option_namespace (certbot.plugins.common.Plugin attribute), 85
 option_namespace() (in module certbot.plugins.common), 85
 option_was_set() (in module certbot.cli), 56
 OverlappingMatchFound, 73

P

parse_args() (certbot.cli.HelpfulArgumentParser method), 57
 parse_preferred_challenges() (in module certbot.cli), 60
 path_surgery() (in module certbot.plugins.util), 91
 pem (certbot.util.Key attribute), 95
 perform() (certbot.interfaces.IAuthenticator method), 78
 perform() (certbot.plugins.common.ChallengePerformer method), 87
 Plugin (class in certbot.plugins.common), 85
 PluginEnhancementAlreadyPresent, 73
 PluginError, 73
 PluginSelectionError, 74
 PluginStorageError, 74
 post_hook() (in module certbot.hooks), 75
 pre_hook() (in module certbot.hooks), 75
 pref_challs (certbot.interfaces.IConfig attribute), 80
 prepare() (certbot.interfaces.IPlugin method), 78
 prepare_and_parse_args() (in module certbot.cli), 59

prescan_for_flag() (certbot.cli.HelpfulArgumentParser method), 58
 print_messages() (certbot.interfaces.IReporter method), 84
 pyopenssl_load_certificate() (in module certbot.crypto_util), 63

R

read_file() (in module certbot.cli), 56
 recovery_routine() (certbot.interfaces.IInstaller method), 81
 recovery_routine() (certbot.plugins.common.Installer method), 86
 recovery_routine() (certbot.reverter.Reverter method), 94
 register_file_creation() (certbot.reverter.Reverter method), 93
 register_undo_command() (certbot.reverter.Reverter method), 93
 remove_config_file_domains_for_renewal() (certbot.cli.HelpfulArgumentParser method), 57
 renew_deploy() (certbot.interfaces.RenewDeployer method), 84
 renew_hook() (in module certbot.hooks), 75
 renewal_deploy_hooks_dir (certbot.configuration.NamespaceConfig attribute), 61
 renewal_hooks_dir (certbot.configuration.NamespaceConfig attribute), 61
 renewal_post_hooks_dir (certbot.configuration.NamespaceConfig attribute), 61
 renewal_pre_hooks_dir (certbot.configuration.NamespaceConfig attribute), 61
 RenewDeployer (class in certbot.interfaces), 84
 report_config_interaction() (in module certbot.cli), 56
 require() (certbot.plugins.dns_common.CredentialsConfiguration method), 89
 response_and_validation() (certbot.achallenges.KeyAuthorizationAnnotatedChallenge method), 55
 restart() (certbot.interfaces.IInstaller method), 81
 revert_temporary_config() (certbot.plugins.common.Installer method), 86
 revert_temporary_config() (certbot.reverter.Reverter method), 92
 Reverter (class in certbot.reverter), 91
 ReverterError, 73
 rollback_checkpoints() (certbot.interfaces.IInstaller method), 81
 rollback_checkpoints() (certbot.plugins.common.Installer method), 86

rollback_checkpoints() (certbot.reverter.Reverter method), 92
 rsa_key_size (certbot.interfaces.IConfig attribute), 79
 run_saved_post_hooks() (in module certbot.hooks), 75
 run_script() (in module certbot.util), 95

S

safe_email() (in module certbot.util), 97
 safe_open() (in module certbot.util), 96
 safely_remove() (in module certbot.util), 96
 save() (certbot.interfaces.AccountStorage method), 77
 save() (certbot.interfaces.IInstaller method), 81
 separate_list_input() (in module certbot.display.util), 70
 server (certbot.interfaces.IConfig attribute), 79
 server_path (certbot.configuration.NamespaceConfig attribute), 61
 set_by_cli() (in module certbot.cli), 56
 set_test_server() (certbot.cli.HelpfulArgumentParser method), 57
 set_up_core_dir() (in module certbot.util), 95
 sha256sum() (in module certbot.crypto_util), 64
 SIDE_FRAME (in module certbot.display.util), 65
 SignalExit, 73
 ssl_dhparams (certbot.plugins.common.Installer attribute), 86
 StandaloneBindError, 74
 strict_permissions (certbot.interfaces.IConfig attribute), 80
 SubprocessError, 73
 success_installation() (in module certbot.display.ops), 71
 success_renewal() (in module certbot.display.ops), 72
 success_revocation() (in module certbot.display.ops), 72
 supported_enhancements() (certbot.interfaces.IInstaller method), 81

T

temp_checkpoint_dir (certbot.interfaces.IConfig attribute), 79

U

unique_file() (in module certbot.util), 96
 unique_lineage_name() (in module certbot.util), 96
 updated_ssl_dhparams_digest (certbot.plugins.common.Installer attribute), 86

V

valid_csr() (in module certbot.crypto_util), 62
 valid_privkey() (in module certbot.crypto_util), 62
 validate_file() (in module certbot.plugins.dns_common), 90
 validate_file_permissions() (in module certbot.plugins.dns_common), 90
 validate_hook() (in module certbot.hooks), 74

`validate_hooks()` (in module `certbot.hooks`), [74](#)
`validated_directory()` (in module `certbot.display.ops`), [72](#)
`validated_input()` (in module `certbot.display.ops`), [72](#)
`verify_cert_matches_priv_key()` (in module `certbot.crypto_util`), [63](#)
`verify_fullchain()` (in module `certbot.crypto_util`), [63](#)
`verify_renewable_cert()` (in module `certbot.crypto_util`), [63](#)
`verify_renewable_cert_sig()` (in module `certbot.crypto_util`), [63](#)
`verify_signed_payload()` (in module `certbot.crypto_util`), [63](#)

W

`work_dir` (`certbot.interfaces.IConfig` attribute), [79](#)

Y

`yesno()` (`certbot.display.util.FileDisplay` method), [67](#)
`yesno()` (`certbot.display.util.NoninteractiveDisplay` method), [69](#)
`yesno()` (`certbot.interfaces.IDisplay` method), [82](#)