Certbot Documentation

Release 1.11.0.dev0

Certbot Project

Dec 27, 2020

CONTENTS

CHAPTER

ONE

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.

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: RFC 8555

ACME working area in github (archived): https://github.com/ietf-wg-acme/acme

1.3.2 System Requirements

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

CHAPTER

TWO

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*

CHAPTER

THREE

GET CERTBOT

Table of Contents

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 installation instructions for your system found at certbot.eff.org.

3.2 System Requirements

Certbot currently requires Python 2.7 or 3.6+ 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.

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 Snap

Most modern Linux distributions (basically any that use systemd) can install Certbot packaged as a snap. Snaps are available for x86_64, ARMv7 and ARMv8 architectures. The Certbot snap provides an easy way to ensure you have the latest version of Certbot with features like automated certificate renewal preconfigured.

You can find instructions for installing the Certbot snap at https://certbot.eff.org/instructions by selecting your server software and then choosing "snapd" in the "System" dropdown menu. (You should select "snapd" regardless of your operating system, as our instructions are the same across all systems.)

3.3.2 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 instructions at certbot.eff.org. 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 installation method recommended for your system at certbot.eff.org, 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 certficate 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*.

For more information about the layout of the `/etc/letsencrypt` directory, see *Where are my certificates?*.

3.3.3 Operating System Packages

Warning: While the Certbot team tries to keep the Certbot packages offered by various operating systems working in the most basic sense, due to distribution policies and/or the limited resources of distribution maintainers, Certbot OS packages often have problems that other distribution mechanisms do not. The packages are often old resulting in a lack of bug fixes and features and a worse TLS configuration than is generated by newer versions of Certbot. They also may not configure certificate renewal for you or have all of Certbot's plugins available. For reasons like these, we recommend most users follow the instructions at https://certbot.eff.org/instructions and OS packages are only documented here as an alternative.

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.

Ubuntu

If you run Ubuntu, 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. From the official Certbot plugins, three of them are also available in Portage. They need to be installed separately if you require their functionality.

```
emerge -av app-crypt/certbot
emerge -av app-crypt/certbot-apache
emerge -av app-crypt/certbot-nginx
emerge -av app-crypt/certbot-dns-nsone
```

Note: The `app-crypt/certbot-dns-nsone` package has a different maintainer than the other packages and can lag behind in version.

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*.

3.3.4 Certbot-Auto

We used to have a shell script named `certbot-auto` to help people install Certbot on UNIX operating systems, however, this script is no longer supported. If you want to uninstall `certbot-auto`, you can follow our instructions here.

Problems with Python virtual environment

When using `certbot-auto` on a low memory system such as VPS with less than 512MB of RAM, the required dependencies of Certbot may 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 fallocate -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
```

3.3.5 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 certbot/setup.py install`, `python pip install certbot`, or `easy_install certbot`. Please do **not** attempt the installation commands as superuser/root and/or without virtual environment, e.g. `sudo python certbot/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!

CHAPTER

FOUR

USER GUIDE

Table of Contents

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)
apache	Y	Y	Automates obtaining and installing a certificate with Apache.	http-01 (80)
nginx	Y	Y	Automates obtaining and installing a certificate with Nginx.	http-01 (80)
webroot	Y	N	Obtains a certificate by writing to the webroot directory of an already running webserver.	http-01 (80)
standalone	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)
DNS plugins	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 the only way to obtain wildcard certificates from Let's	dns-01 (53)

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. They are available in many OS package managers, as Docker images, and as snaps. 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-dnsimple
- certbot-dns-dnsmadeeasy
- certbot-dns-gehirn
- certbot-dns-google
- certbot-dns-linode
- certbot-dns-luadns
- certbot-dns-nsone
- certbot-dns-ovh
- certbot-dns-rfc2136
- certbot-dns-route53
- certbot-dns-sakuracloud

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
haproxy	Y	Y	Integration with the HAProxy load balancer
s3front	Y	Y	Integration with Amazon CloudFront distribution of S3 buckets
gandi	Y	N	Obtain certificates via the Gandi LiveDNS API
varnish	Y	N	Obtain certificates via a Varnish server
external-auth	Y	Y	A plugin for convenient scripting
pritunl	N	Y	Install certificates in pritunl distributed OpenVPN servers
proxmox	N	Y	Install certificates in Proxmox Virtualization servers
dns-standalone	Y	N	Obtain certificates via an integrated DNS server
dns-ispconfig	Y	N	DNS Authentication using ISPConfig as DNS server
dns-clouddns	Y	N	DNS Authentication using CloudDNS API
dns-lightsail	Y	N	DNS Authentication using Amazon Lightsail DNS API
dns-inwx	Y	Y	DNS Authentication for INWX through the XML API

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
    Key Type: RSA
    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.

4.3.2 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
```

4.3.3 Using ECDSA keys

As of version 1.10, Certbot supports two types of private key algorithms: `rsa` and `ecdsa`. The type of key used by Certbot can be controlled through the `--key-type` option. You can also use the `--elliptic-curve` option to control the curve used in ECDSA certificates.

Warning: If you obtain certificates using ECDSA keys, you should be careful not to downgrade your Certbot installation since ECDSA keys are not supported by older versions of Certbot. Downgrades like this are possible if you switch from something like the snaps or certbot-auto to packages provided by your operating system which often lag behind.

Changing existing certificates from RSA to ECDSA

Unless you are aware that you need to support very old HTTPS clients that are not supported by most sites, you can safely just transition your site to use ECDSA keys instead of RSA keys. To accomplish this if you have existing certificates managed by Certbot, you may freely change the certificate to a new private key.

If you want to use ECDSA keys for all certificates in the future, you can simply add the following line to Certbot's *configuration file*

```
key-type = ecdsa
```

After this option is set, newly obtained certificates will use ECDSA keys. This includes certificates managed by Certbot that previously used RSA keys.

If you want to change a single certificate to use ECDSA keys, you'll need to issue a new Certbot command setting `--key-type ecdsa` on the command line like

```
certbot renew --key-type ecdsa --cert-name example.com --force-renewal
```

Obtaining ECDSA certificates in addition to RSA certificates

When Certbot configures the certificates it obtains with Apache or Nginx, all HTTPS clients that we try to support can use certificates with ECDSA keys. If, however, you are aware of having a specific need to support very old TLS clients, you may want to obtain both ECDSA and RSA certificates for the same domains. Certbot can only configure Apache or Nginx to use a single certificate, however, you could manually configure your software to use the different certificates depending on your needs.

When obtaining both ECDSA and RSA certificates for the same domains with Certbot, we recommend using the `--cert-name` option to give your certificates names so that you can easily identify them. For instance, you may want to append "ecdsa" to the name of your ECDSA certificate by using a command like

```
certbot certonly --key-type ecdsa --cert-name example.com-ecdsa
```

4.3.4 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.

4.3.5 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
```

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.

4.3.6 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
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
```

4.3.7 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	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

4.4 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.

For historical reasons, the containing directories are created with permissions of `0700` meaning that certificates are accessible only to servers that run as the root user. **If you will never downgrade to an older version of Certbot**, then you can safely fix this using `chmod 0755 /etc/letsencrypt/{live,archive}`.

For servers that drop root privileges before attempting to read the private key file, you will also need to use `chgrp` and `chmod 0640` to allow the server to read `/etc/letsencrypt/live/$domain/privkey.pem`.

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 >= 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 >= 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*.

4.5 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
- `CERTBOT_TOKEN`: Resource name part of the HTTP-01 challenge (HTTP-01 only)
- `CERTBOT_REMAINING_CHALLENGES`: Number of challenges remaining after the current challenge
- `CERTBOT_ALL_DOMAINS`: A comma-separated list of all domains challenged for the current certificate

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
```

(continues on next page)

(continued from previous page)

```
# 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"

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
```

4.6 Changing the ACME Server

By default, Certbot uses Let's Encrypt's production server at https://acme-v02.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 staging server, you would add `--server https://acme-staging-v02.api.letsencrypt.org/directory` to the command line.

If you use `--server` to specify an ACME CA that implements the standardized 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*.

4.7 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.

4.8 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 ECC for the private key
key-type = ecdsa
elliptic-curve = secp384r1

# 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.

4.9 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.

4.10 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-name or --cert-path)
    delete          Delete a certificate (supply --cert-name)

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)
```

(continues on next page)

(continued from previous page)

```
  --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
                        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 webservers 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-chain PREFERRED_CHAIN
                        If the CA offers multiple certificate chains, prefer
```

(continues on next page)

(continued from previous page)

```
                        the chain with an issuer matching this Subject Common
                        Name. If no match, the default offered chain will be
                        used. (default: None)
  --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
                        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
                        "". (default: CertbotACMEClient/1.10.1
                        (certbot(-auto); OS_NAME OS_VERSION) Authenticator/XXX
                        Installer/YYY (SUBCOMMAND; flags: FLAGS)
                        Py/major.minor.patchlevel). The flags encoded in the
                        user agent are: --duplicate, --force-renew, --allow-
                        subset-of-names, -n, and whether any hooks are set.
  --user-agent-comment USER_AGENT_COMMENT
                        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 --user-agent is set.
                        (Example: Foo-Wrapper/1.0) (default: None)

automation:
  Flags for automating execution & other tweaks

  --keep-until-expiring, --keep, --reinstall
                        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: Ask)
  --expand              If an existing certificate is a strict subset of the
                        requested names, always expand and replace it with the
                        additional names. (default: Ask)
  --version             show program's version number and exit
  --force-renewal, --renew-by-default
                        If a certificate already exists for the requested
                        domains, renew it now, regardless of whether it is
                        near expiry. (Often --keep-until-expiring is more
                        appropriate). Also implies --expand. (default: False)
  --renew-with-new-domains
                        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: False)
  --reuse-key           When renewing, use the same private key as the
                        existing certificate. (default: False)
  --allow-subset-of-names
```

(continues on next page)

(continued from previous page)

```
                        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 --csr.
                        (default: False)
  --agree-tos           Agree to the ACME Subscriber Agreement (default: Ask)
  --duplicate           Allow making a certificate lineage that duplicates an
                        existing one (both can be renewed in parallel)
                        (default: False)
  --os-packages-only    (certbot-auto only) install OS package dependencies
                        and then stop (default: False)
  --no-self-upgrade     (certbot-auto only) prevent the certbot-auto script
                        from upgrading itself to newer released versions
                        (default: Upgrade automatically)
  --no-bootstrap        (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')
  --no-permissions-check
                        (certbot-auto only) skip the check on the file system
                        permissions of the certbot-auto script (default:
                        False)
  -q, --quiet           Silence all output except errors. Useful for
                        automation via cron. Implies --non-interactive.
                        (default: False)

security:
  Security parameters & server settings

  --rsa-key-size N      Size of the RSA key. (default: 2048)
  --key-type {rsa,ecdsa}
                        Type of generated private key(Only *ONE* per
                        invocation can be provided at this time) (default:
                        rsa)
  --elliptic-curve N    The SECG elliptic curve name to use. Please see RFC
                        8446 for supported values. (default: secp256r1)
  --must-staple         Adds the OCSP Must Staple extension to the
                        certificate. Autoconfigures OCSP Stapling for
                        supported setups (Apache version >= 2.3.3 ). (default:
                        False)
  --redirect            Automatically redirect all HTTP traffic to HTTPS for
                        the newly authenticated vhost. (default: redirect
                        enabled for install and run, disabled for enhance)
  --no-redirect         Do not automatically redirect all HTTP traffic to
                        HTTPS for the newly authenticated vhost. (default:
                        redirect enabled for install and run, disabled for
                        enhance)
  --hsts                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)
  --uir                 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)
  --staple-ocsp         Enables OCSP Stapling. A valid OCSP response is
```

(continues on next page)

(continued from previous page)

```
                        stapled to the certificate that the server offers
                        during TLS. (default: None)
  --strict-permissions  Require that all configuration files are owned by the
                        current user; only needed if your config is somewhere
                        unsafe like /tmp/ (default: False)
  --auto-hsts           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.

  --test-cert, --staging
                        Use the staging server to obtain or revoke test
                        (invalid) certificates; equivalent to --server
                        https://acme-staging-v02.api.letsencrypt.org/directory
                        (default: False)
  --debug               Show tracebacks in case of errors, and allow certbot-
                        auto execution on experimental platforms (default:
                        False)
  --no-verify-ssl       Disable verification of the ACME server's certificate.
                        (default: False)
  --http-01-port 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. (default:
                        80)
  --http-01-address HTTP01_ADDRESS
                        The address the server listens to during http-01
                        challenge. (default: )
  --https-port HTTPS_PORT
                        Port used to serve HTTPS. This affects which port
                        Nginx will listen on after a LE certificate is
                        installed. (default: 443)
  --break-my-certs      Be willing to replace or renew valid certificates with
                        invalid (testing/staging) certificates (default:
                        False)

paths:
  Flags for changing execution paths & servers

  --cert-path CERT_PATH
                        Path to where certificate is saved (with auth --csr),
                        installed from, or revoked. (default: None)
  --key-path KEY_PATH   Path to private key for certificate installation or
                        revocation (if account key is missing) (default: None)
  --fullchain-path FULLCHAIN_PATH
                        Accompanying path to a full certificate chain
                        (certificate plus chain). (default: None)
  --chain-path CHAIN_PATH
                        Accompanying path to a certificate chain. (default:
                        None)
  --config-dir CONFIG_DIR
                        Configuration directory. (default: /etc/letsencrypt)
  --work-dir WORK_DIR   Working directory. (default: /var/lib/letsencrypt)
  --logs-dir LOGS_DIR   Logs directory. (default: /var/log/letsencrypt)
  --server SERVER       ACME Directory Resource URI. (default:
                        https://acme-v02.api.letsencrypt.org/directory)
```

(continues on next page)

(continued from previous page)

```
manage:
  Various subcommands and flags are available for managing your
  certificates:

  certificates          List certificates managed by Certbot
  delete                Clean up all files related to a certificate
  renew                 Renew all certificates (or one specified with --cert-
                        name)
  revoke                Revoke a certificate specified with --cert-path or
                        --cert-name
  update_symlinks       Recreate symlinks in your /etc/letsencrypt/live/
                        directory

run:
  Options for obtaining & installing certificates

certonly:
  Options for modifying how a certificate is obtained

  --csr CSR             Path to a Certificate Signing Request (CSR) in DER or
                        PEM format. Currently --csr only works with the
                        '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
```

(continues on next page)

(continued from previous page)

```
                        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)
  --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 you will be unable to receive notice about
                        impending expiration or revocation of your
                        certificates or problems with your Certbot
                        installation that will lead to failure to renew.
```

(continues on next page)

(continued from previous page)

```
                        (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

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).
```

(continues on next page)

(continued from previous page)

```
                        (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
                        using DNS Made Easy for DNS). (default: False)
  --dns-gehirn          Obtain certificates using a DNS TXT record (if you are
                        using Gehirn Infrastructure Service for DNS).
                        (default: False)
  --dns-google          Obtain certificates using a DNS TXT record (if you are
                        using Google Cloud DNS). (default: False)
  --dns-linode          Obtain certificates using a DNS TXT record (if you are
                        using Linode for DNS). (default: False)
  --dns-luadns          Obtain certificates using a DNS TXT record (if you are
                        using LuaDNS for DNS). (default: False)
  --dns-nsone           Obtain certificates using a DNS TXT record (if you are
                        using NS1 for DNS). (default: False)
  --dns-ovh             Obtain certificates using a DNS TXT record (if you are
                        using OVH for DNS). (default: False)
  --dns-rfc2136         Obtain certificates using a DNS TXT record (if you are
                        using BIND for DNS). (default: False)
  --dns-route53         Obtain certificates using a DNS TXT record (if you are
                        using Route53 for DNS). (default: False)
  --dns-sakuracloud     Obtain certificates using a DNS TXT record (if you are
                        using Sakura Cloud for DNS). (default: False)

apache:
  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.)

  --apache-enmod APACHE_ENMOD
                        Path to the Apache 'a2enmod' binary (default: None)
  --apache-dismod APACHE_DISMOD
                        Path to the Apache 'a2dismod' binary (default: None)
  --apache-le-vhost-ext APACHE_LE_VHOST_EXT
                        SSL vhost configuration extension (default: -le-
                        ssl.conf)
  --apache-server-root APACHE_SERVER_ROOT
                        Apache server root directory (default: /etc/apache2)
  --apache-vhost-root APACHE_VHOST_ROOT
                        Apache server VirtualHost configuration root (default:
```

(continues on next page)

(continued from previous page)

```
                        None)
  --apache-logs-root APACHE_LOGS_ROOT
                        Apache server logs directory (default:
                        /var/log/apache2)
  --apache-challenge-location APACHE_CHALLENGE_LOCATION
                        Directory path for challenge configuration (default:
                        /etc/apache2)
  --apache-handle-modules APACHE_HANDLE_MODULES
                        Let installer handle enabling required modules for you
                        (Only Ubuntu/Debian currently) (default: False)
  --apache-handle-sites APACHE_HANDLE_SITES
                        Let installer handle enabling sites for you (Only
                        Ubuntu/Debian currently) (default: False)
  --apache-ctl APACHE_CTL
                        Full path to Apache control script (default:
                        apache2ctl)
  --apache-bin APACHE_BIN
                        Full path to apache2/httpd binary (default: None)

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

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

  --dns-dnsimple-propagation-seconds DNS_DNSIMPLE_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-dnsimple-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
                        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 forinformation
                        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: 120)
  --dns-linode-credentials DNS_LINODE_CREDENTIALS
                        Linode credentials INI file. (default: None)

dns-luadns:
```

(continues on next page)

(continued from previous page)

```
  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).

  --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
```

(continues on next page)

(continued from previous page)

```
                        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. For both authenticator and cleanup script, on HTTP-01 and DNS-01
  challenges, $CERTBOT_REMAINING_CHALLENGES will be equal to the number of
  challenges that remain after the current one, and $CERTBOT_ALL_DOMAINS
  contains a comma-separated list of all domains that are challenged for the
  current certificate.

  --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)

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)
  --nginx-sleep-seconds NGINX_SLEEP_SECONDS
                        Number of seconds to wait for nginx configuration
                        changes to apply when reloading. (default: 1)

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
```

(continues on next page)

(continued from previous page)

```
                        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:
                        {})
```

4.11 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

CHAPTER

FIVE

DEVELOPER GUIDE

Table of Contents

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 unit tests for Certbot and check for coverage with `tox -e py3-cover`. You should 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 Azure Pipelines 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/venv3.py` contains two CLI tools 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.

Note: Some options are available to tweak the local ACME server. You can execute `run_acme_server --help` to see the inline help of the `run_acme_server` tool.

```
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
```

Running tests in CI

Certbot uses Azure Pipelines to run continuous integration tests. If you are using our Azure setup, a branch whose name starts with `test-` will run all tests on that branch.

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 webservers, 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, Icecast 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.

5.2.5 Writing your own plugin

Note: The Certbot team is not currently accepting any new DNS plugins because we want to rethink our approach to the challenge and resolve some issues like #6464, #6503, and #6504 first.

In the meantime, you're welcome to release it as a third-party plugin. See certbot-dns-ispconfig for one example of that.

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.

Writing your own plugin snap

If you'd like your plugin to be used alongside the Certbot snap, you will also have to publish your plugin as a snap. Plugin snaps are regular confined snaps, but normally do not provide any "apps" themselves. Plugin snaps export loadable Python modules to the Certbot snap.

When the Certbot snap runs, it will use its version of Python and prefer Python modules contained in its own snap over modules contained in external snaps. This means that your snap doesn't have to contain things like an extra copy of Python, Certbot, or their dependencies, but also that if you need a different version of a dependency than is already installed in the Certbot snap, the Certbot snap will have to be updated.

Certbot plugin snaps expose their Python modules to the Certbot snap via a snap content interface where `certbot-1` is the value for the `content` attribute. The Certbot snap only uses this to find the names of connected plugin snaps and it expects to find the Python modules to be loaded under `lib/python3.8/site-packages/` in the plugin snap. This location is the default when using the `core20` base snap and the python snapcraft plugin.

The Certbot snap also provides a separate content interface which you can use to get metadata about the Certbot snap using the `content` identifier `metadata-1`.

The script used to generate the snapcraft.yaml files for our own externally snapped plugins can be found at https://github.com/certbot/certbot/blob/master/tools/snap/generate_dnsplugins_snapcraft.sh.

For more information on building externally snapped plugins, see the section on *Building the Certbot and DNS plugin snaps*.

Once you have created your own snap, if you have the snap file locally, it can be installed for use with Certbot by running:

```
snap install --classic certbot
snap set certbot trust-plugin-with-root=ok
snap install --dangerous your-snap-filename.snap
sudo snap connect certbot:plugin your-snap-name
sudo /snap/bin/certbot plugins
```

If everything worked, the last command should list your plugin in the list of plugins found by Certbot. Once your snap is published to the snap store, it will be installable through the name of the snap on the snap store without the `--dangerous` flag. If you are also using Certbot's metadata interface, you can run `sudo snap connect your-snap-name:your-plug-name-for-metadata certbot:certbot-metadata` to connect your snap to it.

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. You may consider installing a plugin for editorconfig in your editor to prevent some linting warnings.
6. Please avoid `unittest.assertTrue` or `unittest.assertFalse` when possible, and use `assertEqual` or more specific assert. They give better messages when it's failing, and are generally more correct.

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
```

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. If any documentation should be added or updated as part of the changes you have made, please include the documentation changes in your PR.
6. 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.
7. Did your tests pass on Azure Pipelines? 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 Building the Certbot and DNS plugin snaps

Instructions for how to manually build and run the Certbot snap and the externally snapped DNS plugins that the Certbot project supplies are located in the README file at https://github.com/certbot/certbot/tree/master/tools/snap.

5.9 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*!

5.9.1 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.

5.9.2 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.

5.9.3 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.

5.10 Updating the documentation

Many of the packages in the Certbot repository have documentation in a `docs/` directory. This directory is located under the top level directory for the package. For instance, Certbot's documentation is under `certbot/docs`.

To build the documentation of a package, make sure you have followed the instructions to set up a *local copy* of Certbot including activating the virtual environment. After that, `cd` to the docs directory you want to build and run the command:

```
make clean html
```

This would generate the HTML documentation in `_build/html` in your current `docs/` directory.

5.11 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
```

CHAPTER

SIX

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-dnsimple
- 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`.

Our packages are cryptographically signed and their signature can be verified using the PGP key `A2CFB51FA275A7286234E7B24D17C995CD9775F2`. This key can be found on major key servers and at https://dl.eff.org/certbot.pub.

6.2 Notes for package maintainers

0. Please use our tagged releases, not `master`!

1. Do not package `certbot-compatibility-test` as it's only used internally.
2. To run tests on our packages, you should use pytest by running the command `python -m pytest`. Running `pytest` directly may not work because PYTHONPATH is not handled the same way and local modules may not be found by the test runner.
3. If you'd like to include automated renewal in your package:

- `certbot renew -q` should be added to crontab or systemd timer.
- A random per-machine time offset should be included to avoid having a large number of your clients hit Let's Encrypt's servers simultaneously.
- `--preconfigured-renewal` should be included on the CLI or in `cli.ini` for all invocations of Certbot, so that it can adjust its interactive output regarding automated renewal (Certbot >= 1.9.0).

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.

CHAPTER

SEVEN

BACKWARDS COMPATIBILITY

All Certbot components including acme, Certbot, and *non-third party plugins* follow Semantic Versioning both for its Python *API* and for the application itself. This means that we will not change behavior in a backwards incompatible way except in a new major version of the project.

Note: None of this applies to the behavior of Certbot distribution mechanisms such as *our snaps* or OS packages whose behavior may change at any time. Semantic versioning only applies to the common Certbot components that are installed by various distribution methods.

For Certbot as an application, the command line interface and non-interactive behavior can be considered stable with two exceptions. The first is that no aspects of Certbot's console or log output should be considered stable and it may change at any time. The second is that Certbot's behavior should only be considered stable with certain files but not all. Files with which users should expect Certbot to maintain its current behavior with are:

- `/etc/letsencrypt/live/<domain>/{cert,chain,fullchain,privkey}.pem` where `<domain>` is the name given to `--cert-name`. If `--cert-name` is not set by the user, it is the first domain given to `--domains`.
- *CLI configuration files*
- Hook directories in `/etc/letsencrypt/renewal-hooks`

Certbot's behavior with other files may change at any point.

Another area where Certbot should not be considered stable is its behavior when not run in non-interactive mode which also may change at any point.

In general, if we're making a change that we expect will break some users, we will bump the major version and will have warned about it in a prior release when possible. For our Python API, we will issue warnings using Python's warning module. For application level changes, we will print and log warning messages.

CHAPTER

EIGHT

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: RFC 8555

ACME working area in github (archived): https://github.com/ietf-wg-acme/acme

CHAPTER

NINE

API DOCUMENTATION

9.1 certbot package

Certbot client.

9.1.1 Subpackages

certbot.compat package

Compatibility layer to run certbot both on Linux and Windows.

This package contains all logic that needs to be implemented specifically for Linux and for Windows. Then the rest of certbot code relies on this module to be platform agnostic.

Submodules

certbot.compat.filesystem module

Compat module to handle files security on Windows and Linux

certbot.compat.filesystem.**chmod**(*file_path*, *mode*)

Apply a POSIX mode on given file_path:

- for Linux, the POSIX mode will be directly applied using chmod,
- for Windows, the POSIX mode will be translated into a Windows DACL that make sense for Certbot context, and applied to the file using kernel calls.

The definition of the Windows DACL that correspond to a POSIX mode, in the context of Certbot, is explained at https://github.com/certbot/certbot/issues/6356 and is implemented by the method `_generate_windows_flags()`.

Parameters

- **file_path** (*str*) – Path of the file
- **mode** (*int*) – POSIX mode to apply

certbot.compat.filesystem.**umask**(*mask*)

Set the current numeric umask and return the previous umask. On Linux, the built-in umask method is used. On Windows, our Certbot-side implementation is used.

Parameters **mask** (*int*) – The user file-creation mode mask to apply.

Return type int

Returns The previous umask value.

certbot.compat.filesystem.**copy_ownership_and_apply_mode**(*src*, *dst*, *mode*, *copy_user*, *copy_group*)

Copy ownership (user and optionally group on Linux) from the source to the destination, then apply given mode in compatible way for Linux and Windows. This replaces the os.chown command.

Parameters

- **src** (*str*) – Path of the source file
- **dst** (*str*) – Path of the destination file
- **mode** (*int*) – Permission mode to apply on the destination file
- **copy_user** (*bool*) – Copy user if `True`
- **copy_group** (*bool*) – Copy group if `True` on Linux (has no effect on Windows)

certbot.compat.filesystem.**copy_ownership_and_mode**(*src*, *dst*, *copy_user=True*, *copy_group=True*)

Copy ownership (user and optionally group on Linux) and mode/DACL from the source to the destination.

Parameters

- **src** (*str*) – Path of the source file
- **dst** (*str*) – Path of the destination file
- **copy_user** (*bool*) – Copy user if `True`
- **copy_group** (*bool*) – Copy group if `True` on Linux (has no effect on Windows)

certbot.compat.filesystem.**check_mode**(*file_path*, *mode*)

Check if the given mode matches the permissions of the given file. On Linux, will make a direct comparison, on Windows, mode will be compared against the security model.

Parameters

- **file_path** (*str*) – Path of the file
- **mode** (*int*) – POSIX mode to test

Return type bool

Returns True if the POSIX mode matches the file permissions

certbot.compat.filesystem.**check_owner**(*file_path*)

Check if given file is owned by current user.

Parameters **file_path** (*str*) – File path to check

Return type bool

Returns True if given file is owned by current user, False otherwise.

certbot.compat.filesystem.**check_permissions**(*file_path*, *mode*)

Check if given file has the given mode and is owned by current user.

Parameters

- **file_path** (*str*) – File path to check
- **mode** (*int*) – POSIX mode to check

Return type bool

Returns True if file has correct mode and owner, False otherwise.

certbot.compat.filesystem.**open**(*file_path*, *flags*, *mode=511*)

Wrapper of original os.open function, that will ensure on Windows that given mode is correctly applied.

Parameters

- **file_path** (*str*) – The file path to open
- **flags** (*int*) – Flags to apply on file while opened
- **mode** (*int*) – POSIX mode to apply on file when opened, Python defaults will be applied if `None`

Returns the file descriptor to the opened file

Return type int

Raise OSError(errno.EEXIST) if the file already exists and os.O_CREAT & os.O_EXCL are set, OSError(errno.EACCES) on Windows if the file already exists and is a directory, and os.O_CREAT is set.

`certbot.compat.filesystem.`**`makedirs`**(*file_path*, *mode=511*)

Rewrite of original os.makedirs function, that will ensure on Windows that given mode is correctly applied.

Parameters

- **file_path** (*str*) – The file path to open
- **mode** (*int*) – POSIX mode to apply on leaf directory when created, Python defaults will be applied if `None`

`certbot.compat.filesystem.`**`mkdir`**(*file_path*, *mode=511*)

Rewrite of original os.mkdir function, that will ensure on Windows that given mode is correctly applied.

Parameters

- **file_path** (*str*) – The file path to open
- **mode** (*int*) – POSIX mode to apply on directory when created, Python defaults will be applied if `None`

`certbot.compat.filesystem.`**`replace`**(*src*, *dst*)

Rename a file to a destination path and handles situations where the destination exists.

Parameters

- **src** (*str*) – The current file path.
- **dst** (*str*) – The new file path.

`certbot.compat.filesystem.`**`realpath`**(*file_path*)

Find the real path for the given path. This method resolves symlinks, including recursive symlinks, and is protected against symlinks that creates an infinite loop.

Parameters **file_path** (*str*) – The path to resolve

Returns The real path for the given path

Return type str

`certbot.compat.filesystem.`**`is_executable`**(*path*)

Is path an executable file?

Parameters **path** (*str*) – path to test

Returns True if path is an executable file

Return type bool

`certbot.compat.filesystem.`**`has_world_permissions`**(*path*)

Check if everybody/world has any right (read/write/execute) on a file given its path.

Parameters **path** (*str*) – path to test

Returns True if everybody/world has any right to the file

Return type bool

certbot.compat.filesystem.**compute_private_key_mode**(*old_key*, *base_mode*)
Calculate the POSIX mode to apply to a private key given the previous private key.

Parameters

- **old_key** (*str*) – path to the previous private key
- **base_mode** (*int*) – the minimum modes to apply to a private key

Returns the POSIX mode to apply

Return type int

certbot.compat.filesystem.**has_same_ownership**(*path1*, *path2*)
Return True if the ownership of two files given their respective path is the same. On Windows, ownership is checked against owner only, since files do not have a group owner.

Parameters

- **path1** (*str*) – path to the first file
- **path2** (*str*) – path to the second file

Returns True if both files have the same ownership, False otherwise

Return type bool

certbot.compat.filesystem.**has_min_permissions**(*path*, *min_mode*)
Check if a file given its path has at least the permissions defined by the given minimal mode. On Windows, group permissions are ignored since files do not have a group owner.

Parameters

- **path** (*str*) – path to the file to check
- **min_mode** (*int*) – the minimal permissions expected

Returns True if the file matches the minimal permissions expectations, False otherwise

Return type bool

certbot.compat.misc module

This compat module handles various platform specific calls that do not fall into one particular category.

certbot.compat.misc.**raise_for_non_administrative_windows_rights**()
On Windows, raise if current shell does not have the administrative rights. Do nothing on Linux.

Raises ***errors.Error*** – If the current shell does not have administrative rights on Windows.

certbot.compat.misc.**readline_with_timeout**(*timeout*, *prompt*)
Read user input to return the first line entered, or raise after specified timeout.

Parameters

- **timeout** (*float*) – The timeout in seconds given to the user.
- **prompt** (*str*) – The prompt message to display to the user.

Returns The first line entered by the user.

Return type str

certbot.compat.misc.**get_default_folder**(*folder_type*)
Return the relevant default folder for the current OS

Parameters **folder_type** (*str*) – The type of folder to retrieve (config, work or logs)

Returns The relevant default folder.

Return type str

certbot.compat.misc.**underscores_for_unsupported_characters_in_path**(*path*)
Replace unsupported characters in path for current OS by underscores. :param str path: the path to normalize :return: the normalized path :rtype: str

certbot.compat.misc.**execute_command**(*cmd_name*, *shell_cmd*, *env=None*)

Run a command:

- on Linux command will be run by the standard shell selected with Popen(shell=True)
- on Windows command will be run in a Powershell shell

Parameters

- **cmd_name** (*str*) – the user facing name of the hook being run
- **shell_cmd** (*str*) – shell command to execute
- **env** (*dict*) – environ to pass into Popen

Returns tuple (str stderr, str stdout)

certbot.compat.os module

This compat modules is a wrapper of the core os module that forbids usage of specific operations (e.g. chown, chmod, getuid) that would be harmful to the Windows file security model of Certbot. This module is intended to replace standard os module throughout certbot projects (except acme).

This module has the same API as the os module in the Python standard library except for the functions defined below.

certbot.compat.os.**chmod**(**unused_args*, ***unused_kwargs*)
Method os.chmod() is forbidden

certbot.compat.os.**umask**(**unused_args*, ***unused_kwargs*)
Method os.chmod() is forbidden

certbot.compat.os.**chown**(**unused_args*, ***unused_kwargs*)
Method os.chown() is forbidden

certbot.compat.os.**open**(**unused_args*, ***unused_kwargs*)
Method os.open() is forbidden

certbot.compat.os.**mkdir**(**unused_args*, ***unused_kwargs*)
Method os.mkdir() is forbidden

certbot.compat.os.**makedirs**(**unused_args*, ***unused_kwargs*)
Method os.makedirs() is forbidden

certbot.compat.os.**rename**(**unused_args*, ***unused_kwargs*)
Method os.rename() is forbidden

certbot.compat.os.**replace**(**unused_args*, ***unused_kwargs*)
Method os.replace() is forbidden

certbot.compat.os.**access**(**unused_args*, ***unused_kwargs*)
Method os.access() is forbidden

certbot.compat.os.**stat**(**unused_args*, ***unused_kwargs*)
Method os.stat() is forbidden

certbot.compat.os.**fstat**(**unused_args*, ***unused_kwargs*)
Method os.stat() is forbidden

certbot.compat.os.**fsencode**(*filename*)
Encode filename (an os.PathLike, bytes, or str) to the filesystem encoding with 'surrogateescape' error handler, return bytes unchanged. On Windows, use 'strict' error handler if the file system encoding is 'mbcs' (which is the default encoding).

certbot.compat.os.**fsdecode**(*filename*)
Decode filename (an os.PathLike, bytes, or str) from the filesystem encoding with 'surrogateescape' error handler, return str unchanged. On Windows, use 'strict' error handler if the file system encoding is 'mbcs' (which is the default encoding).

certbot.compat.os.**get_exec_path**(*env=None*)
Returns the sequence of directories that will be searched for the named executable (similar to a shell) when launching a process.

env must be an environment variable dict or None. If *env* is None, os.environ will be used.

certbot.compat.os.**fdopen**(*fd*, **args*, ***kwargs*)

certbot.compat.os.**popen**(*cmd*, *mode='r'*, *buffering=-1*)

class certbot.compat.os.**DirEntry**
Bases: object

inode()
Return inode of the entry; cached per entry.

is_dir()
Return True if the entry is a directory; cached per entry.

is_file()
Return True if the entry is a file; cached per entry.

is_symlink()
Return True if the entry is a symbolic link; cached per entry.

name
the entry's base filename, relative to scandir() "path" argument

path
the entry's full path name; equivalent to os.path.join(scandir_path, entry.name)

stat()
Return stat_result object for the entry; cached per entry.

certbot.compat.os.**WCOREDUMP**()
Return True if the process returning status was dumped to a core file.

certbot.compat.os.**WEXITSTATUS**()
Return the process return code from status.

certbot.compat.os.**WIFCONTINUED**()
Return True if a particular process was continued from a job control stop.

Return True if the process returning status was continued from a job control stop.

certbot.compat.os.**WIFEXITED**()
Return True if the process returning status exited via the exit() system call.

certbot.compat.os.**WIFSIGNALED**()
Return True if the process returning status was terminated by a signal.

certbot.compat.os.**WIFSTOPPED**()
Return True if the process returning status was stopped.

certbot.compat.os.**WSTOPSIG**()
Return the signal that stopped the process that provided the status value.

certbot.compat.os.**WTERMSIG**()
Return the signal that terminated the process that provided the status value.

certbot.compat.os.**abort**()
Abort the interpreter immediately.

This function 'dumps core' or otherwise fails in the hardest way possible on the hosting operating system. This function never returns.

certbot.compat.os.**chdir**()
Change the current working directory to the specified path.

path may always be specified as a string. On some platforms, path may also be specified as an open file descriptor.

If this functionality is unavailable, using it raises an exception.

certbot.compat.os.**chroot**()
Change root directory to path.

certbot.compat.os.**close**()
Close a file descriptor.

certbot.compat.os.**closerange**()
Closes all file descriptors in [fd_low, fd_high), ignoring errors.

certbot.compat.os.**confstr**()
Return a string-valued system configuration variable.

certbot.compat.os.**copy_file_range**()
Copy count bytes from one file descriptor to another.

src Source file descriptor.

dst Destination file descriptor.

count Number of bytes to copy.

offset_src Starting offset in src.

offset_dst Starting offset in dst.

If offset_src is None, then src is read from the current position; respectively for offset_dst.

certbot.compat.os.**cpu_count**()
Return the number of CPUs in the system; return None if indeterminable.

This number is not equivalent to the number of CPUs the current process can use. The number of usable CPUs can be obtained with `len(os.sched_getaffinity(0))`

certbot.compat.os.**ctermid**()
Return the name of the controlling terminal for this process.

certbot.compat.os.**device_encoding**()
Return a string describing the encoding of a terminal's file descriptor.

The file descriptor must be attached to a terminal. If the device is not a terminal, return None.

certbot.compat.os.**dup**()
Return a duplicate of a file descriptor.

certbot.compat.os.**dup2**()
Duplicate file descriptor.

certbot.compat.os.**error**
alias of builtins.OSError

certbot.compat.os.**execv**()
Execute an executable path with arguments, replacing current process.

path Path of executable file.

argv Tuple or list of strings.

certbot.compat.os.**execve**()
Execute an executable path with arguments, replacing current process.

path Path of executable file.

argv Tuple or list of strings.

env Dictionary of strings mapping to strings.

certbot.compat.os.**fchdir**()
Change to the directory of the given file descriptor.

fd must be opened on a directory, not a file. Equivalent to os.chdir(fd).

certbot.compat.os.**fchmod**()
Change the access permissions of the file given by file descriptor fd.

Equivalent to os.chmod(fd, mode).

certbot.compat.os.**fchown**()
Change the owner and group id of the file specified by file descriptor.

Equivalent to os.chown(fd, uid, gid).

certbot.compat.os.**fdatasync**()
Force write of fd to disk without forcing update of metadata.

certbot.compat.os.**fork**()
Fork a child process.

Return 0 to child process and PID of child to parent process.

certbot.compat.os.**forkpty**()
Fork a new process with a new pseudo-terminal as controlling tty.

Returns a tuple of (pid, master_fd). Like fork(), return pid of 0 to the child process, and pid of child to the parent process. To both, return fd of newly opened pseudo-terminal.

certbot.compat.os.**fpathconf**()
Return the configuration limit name for the file descriptor fd.

If there is no limit, return -1.

certbot.compat.os.**fspath**()
Return the file system path representation of the object.

If the object is str or bytes, then allow it to pass through as-is. If the object defines __fspath__(), then return the result of that method. All other types raise a TypeError.

certbot.compat.os.**fstatvfs**()
Perform an fstatvfs system call on the given fd.

Equivalent to statvfs(fd).

certbot.compat.os.**fsync**()
Force write of fd to disk.

certbot.compat.os.**ftruncate**()
Truncate a file, specified by file descriptor, to a specific length.

certbot.compat.os.**get_blocking**()
Get the blocking mode of the file descriptor.

Return False if the O_NONBLOCK flag is set, True if the flag is cleared.

certbot.compat.os.**get_inheritable**()
Get the close-on-exe flag of the specified file descriptor.

certbot.compat.os.**get_terminal_size**()
Return the size of the terminal window as (columns, lines).

The optional argument fd (default standard output) specifies which file descriptor should be queried.

If the file descriptor is not connected to a terminal, an OSError is thrown.

This function will only be defined if an implementation is available for this system.

shutil.get_terminal_size is the high-level function which should normally be used, os.get_terminal_size is the low-level implementation.

certbot.compat.os.**getcwd**()
Return a unicode string representing the current working directory.

certbot.compat.os.**getcwdb**()
Return a bytes string representing the current working directory.

certbot.compat.os.**getegid**()
Return the current process's effective group id.

certbot.compat.os.**geteuid**()
Return the current process's effective user id.

certbot.compat.os.**getgid**()
Return the current process's group id.

certbot.compat.os.**getgrouplist**(*user*, *group*) → list of groups to which a user belongs
Returns a list of groups to which a user belongs.

user: username to lookup group: base group id of the user

certbot.compat.os.**getgroups**()
Return list of supplemental group IDs for the process.

certbot.compat.os.**getloadavg**()
Return average recent system load information.

Return the number of processes in the system run queue averaged over the last 1, 5, and 15 minutes as a tuple of three floats. Raises OSError if the load average was unobtainable.

certbot.compat.os.**getlogin**()
Return the actual login name.

certbot.compat.os.**getpgid**()
Call the system call getpgid(), and return the result.

certbot.compat.os.**getpgrp**()
Return the current process group id.

certbot.compat.os.**getpid**()
Return the current process id.

certbot.compat.os.**getppid**()
Return the parent's process id.

If the parent process has already exited, Windows machines will still return its id; others systems will return the id of the 'init' process (1).

certbot.compat.os.**getpriority**()
Return program scheduling priority.

certbot.compat.os.**getrandom**()
Obtain a series of random bytes.

certbot.compat.os.**getresgid**()
Return a tuple of the current process's real, effective, and saved group ids.

certbot.compat.os.**getresuid**()
Return a tuple of the current process's real, effective, and saved user ids.

certbot.compat.os.**getsid**()
Call the system call getsid(pid) and return the result.

certbot.compat.os.**getuid**()
Return the current process's user id.

certbot.compat.os.**getxattr**()
Return the value of extended attribute attribute on path.

path may be either a string, a path-like object, or an open file descriptor. If follow_symlinks is False, and the last element of the path is a symbolic

link, getxattr will examine the symbolic link itself instead of the file the link points to.

certbot.compat.os.**initgroups**(*username*, *gid*) → None
Call the system initgroups() to initialize the group access list with all of the groups of which the specified username is a member, plus the specified group id.

certbot.compat.os.**isatty**()
Return True if the fd is connected to a terminal.

Return True if the file descriptor is an open file descriptor connected to the slave end of a terminal.

certbot.compat.os.**kill**()
Kill a process with a signal.

certbot.compat.os.**killpg**()
Kill a process group with a signal.

certbot.compat.os.**lchown**()
Change the owner and group id of path to the numeric uid and gid.

This function will not follow symbolic links. Equivalent to os.chown(path, uid, gid, follow_symlinks=False).

certbot.compat.os.**link**()
Create a hard link to a file.

If either src_dir_fd or dst_dir_fd is not None, it should be a file descriptor open to a directory, and the respective path string (src or dst) should be relative; the path will then be relative to that directory.

If follow_symlinks is False, and the last element of src is a symbolic link, link will create a link to the symbolic link itself instead of the file the link points to.

src_dir_fd, dst_dir_fd, and follow_symlinks may not be implemented on your platform. If they are unavailable, using them will raise a NotImplementedError.

certbot.compat.os.**listdir**()
Return a list containing the names of the files in the directory.

path can be specified as either str, bytes, or a path-like object. If path is bytes, the filenames returned will also be bytes; in all other circumstances the filenames returned will be str.

If path is None, uses the path='.'. On some platforms, path may also be specified as an open file descriptor;

the file descriptor must refer to a directory. If this functionality is unavailable, using it raises NotImplementedError.

The list is in arbitrary order. It does not include the special entries '.' and '..' even if they are present in the directory.

certbot.compat.os.**listxattr**()
Return a list of extended attributes on path.

path may be either None, a string, a path-like object, or an open file descriptor. if path is None, listxattr will examine the current directory. If follow_symlinks is False, and the last element of the path is a symbolic

link, listxattr will examine the symbolic link itself instead of the file the link points to.

certbot.compat.os.**lockf**()
Apply, test or remove a POSIX lock on an open file descriptor.

fd An open file descriptor.

command One of F_LOCK, F_TLOCK, F_ULOCK or F_TEST.

length The number of bytes to lock, starting at the current position.

certbot.compat.os.**lseek**()
Set the position of a file descriptor. Return the new position.

Return the new cursor position in number of bytes relative to the beginning of the file.

certbot.compat.os.**lstat**()
Perform a stat system call on the given path, without following symbolic links.

Like stat(), but do not follow symbolic links. Equivalent to stat(path, follow_symlinks=False).

certbot.compat.os.**major**()
Extracts a device major number from a raw device number.

certbot.compat.os.**makedev**()
Composes a raw device number from the major and minor device numbers.

certbot.compat.os.**memfd_create**()

certbot.compat.os.**minor**()
Extracts a device minor number from a raw device number.

certbot.compat.os.**mkfifo**()
Create a "fifo" (a POSIX named pipe).

If dir_fd is not None, it should be a file descriptor open to a directory, and path should be relative; path will then be relative to that directory.

dir_fd may not be implemented on your platform. If it is unavailable, using it will raise a NotImplementedError.

certbot.compat.os.**mknod**()
Create a node in the file system.

Create a node in the file system (file, device special file or named pipe) at path. mode specifies both the permissions to use and the type of node to be created, being combined (bitwise OR) with one of S_IFREG, S_IFCHR, S_IFBLK, and S_IFIFO. If S_IFCHR or S_IFBLK is set on mode, device defines the newly created device special file (probably using os.makedev()). Otherwise device is ignored.

If dir_fd is not None, it should be a file descriptor open to a directory, and path should be relative; path will then be relative to that directory.

dir_fd may not be implemented on your platform. If it is unavailable, using it will raise a NotImplementedError.

certbot.compat.os.**nice**()
Add increment to the priority of process and return the new priority.

certbot.compat.os.**openpty**()
Open a pseudo-terminal.

Return a tuple of (master_fd, slave_fd) containing open file descriptors for both the master and slave ends.

certbot.compat.os.**pathconf**()
Return the configuration limit name for the file or directory path.

If there is no limit, return -1. On some platforms, path may also be specified as an open file descriptor.

If this functionality is unavailable, using it raises an exception.

certbot.compat.os.**pipe**()
Create a pipe.

Returns a tuple of two file descriptors: (read_fd, write_fd)

certbot.compat.os.**pipe2**()
Create a pipe with flags set atomically.

Returns a tuple of two file descriptors: (read_fd, write_fd)

flags can be constructed by ORing together one or more of these values: O_NONBLOCK, O_CLOEXEC.

certbot.compat.os.**posix_fadvise**()
Announce an intention to access data in a specific pattern.

Announce an intention to access data in a specific pattern, thus allowing the kernel to make optimizations. The advice applies to the region of the file specified by fd starting at offset and continuing for length bytes. advice is one of POSIX_FADV_NORMAL, POSIX_FADV_SEQUENTIAL, POSIX_FADV_RANDOM, POSIX_FADV_NOREUSE, POSIX_FADV_WILLNEED, or POSIX_FADV_DONTNEED.

certbot.compat.os.**posix_fallocate**()
Ensure a file has allocated at least a particular number of bytes on disk.

Ensure that the file specified by fd encompasses a range of bytes starting at offset bytes from the beginning and continuing for length bytes.

certbot.compat.os.**posix_spawn**()
Execute the program specified by path in a new process.

path Path of executable file.

argv Tuple or list of strings.

env Dictionary of strings mapping to strings.

file_actions A sequence of file action tuples.

setpgroup The pgroup to use with the POSIX_SPAWN_SETPGROUP flag.

resetids If the value is `true` the POSIX_SPAWN_RESETIDS will be activated.

setsid If the value is `true` the POSIX_SPAWN_SETSID or POSIX_SPAWN_SETSID_NP will be activated.

setsigmask The sigmask to use with the POSIX_SPAWN_SETSIGMASK flag.

setsigdef The sigmask to use with the POSIX_SPAWN_SETSIGDEF flag.

scheduler A tuple with the scheduler policy (optional) and parameters.

certbot.compat.os.**posix_spawnp**()
Execute the program specified by path in a new process.

path Path of executable file.

argv Tuple or list of strings.

env Dictionary of strings mapping to strings.

file_actions A sequence of file action tuples.

setpgroup The pgroup to use with the POSIX_SPAWN_SETPGROUP flag.

resetids If the value is `True` the POSIX_SPAWN_RESETIDS will be activated.

setsid If the value is `True` the POSIX_SPAWN_SETSID or POSIX_SPAWN_SETSID_NP will be activated.

setsigmask The sigmask to use with the POSIX_SPAWN_SETSIGMASK flag.

setsigdef The sigmask to use with the POSIX_SPAWN_SETSIGDEF flag.

scheduler A tuple with the scheduler policy (optional) and parameters.

certbot.compat.os.**pread**()
Read a number of bytes from a file descriptor starting at a particular offset.

Read length bytes from file descriptor fd, starting at offset bytes from the beginning of the file. The file offset remains unchanged.

certbot.compat.os.**preadv**()
Reads from a file descriptor into a number of mutable bytes-like objects.

Combines the functionality of readv() and pread(). As readv(), it will transfer data into each buffer until it is full and then move on to the next buffer in the sequence to hold the rest of the data. Its fourth argument, specifies the file offset at which the input operation is to be performed. It will return the total number of bytes read (which can be less than the total capacity of all the objects).

The flags argument contains a bitwise OR of zero or more of the following flags:

- RWF_HIPRI
- RWF_NOWAIT

Using non-zero flags requires Linux 4.6 or newer.

certbot.compat.os.**putenv**()
Change or add an environment variable.

certbot.compat.os.**pwrite**()
Write bytes to a file descriptor starting at a particular offset.

Write buffer to fd, starting at offset bytes from the beginning of the file. Returns the number of bytes writte. Does not change the current file offset.

certbot.compat.os.**pwritev**()
Writes the contents of bytes-like objects to a file descriptor at a given offset.

Combines the functionality of writev() and pwrite(). All buffers must be a sequence of bytes-like objects. Buffers are processed in array order. Entire contents of first buffer is written before proceeding to second, and so on. The operating system may set a limit (sysconf() value SC_IOV_MAX) on the number of buffers that can be used. This function writes the contents of each object to the file descriptor and returns the total number of bytes written.

The flags argument contains a bitwise OR of zero or more of the following flags:

- RWF_DSYNC
- RWF_SYNC

Using non-zero flags requires Linux 4.7 or newer.

certbot.compat.os.**read**()
Read from a file descriptor. Returns a bytes object.

certbot.compat.os.**readlink**()
Return a string representing the path to which the symbolic link points.

If dir_fd is not None, it should be a file descriptor open to a directory, and path should be relative; path will then be relative to that directory.

dir_fd may not be implemented on your platform. If it is unavailable, using it will raise a NotImplementedError.

certbot.compat.os.**readv**()
Read from a file descriptor fd into an iterable of buffers.

The buffers should be mutable buffers accepting bytes. readv will transfer data into each buffer until it is full and then move on to the next buffer in the sequence to hold the rest of the data.

readv returns the total number of bytes read, which may be less than the total capacity of all the buffers.

certbot.compat.os.**register_at_fork**()
Register callables to be called when forking a new process.

before A callable to be called in the parent before the fork() syscall.

after_in_child A callable to be called in the child after fork().

after_in_parent A callable to be called in the parent after fork().

'before' callbacks are called in reverse order. 'after_in_child' and 'after_in_parent' callbacks are called in order.

certbot.compat.os.**remove**()
Remove a file (same as unlink()).

If dir_fd is not None, it should be a file descriptor open to a directory, and path should be relative; path will then be relative to that directory.

dir_fd may not be implemented on your platform. If it is unavailable, using it will raise a NotImplemented-Error.

certbot.compat.os.**removexattr**()
Remove extended attribute attribute on path.

path may be either a string, a path-like object, or an open file descriptor. If follow_symlinks is False, and the last element of the path is a symbolic

link, removexattr will modify the symbolic link itself instead of the file the link points to.

certbot.compat.os.**rmdir**()
Remove a directory.

If dir_fd is not None, it should be a file descriptor open to a directory, and path should be relative; path will then be relative to that directory.

dir_fd may not be implemented on your platform. If it is unavailable, using it will raise a NotImplementedError.

certbot.compat.os.**scandir**()
Return an iterator of DirEntry objects for given path.

path can be specified as either str, bytes, or a path-like object. If path is bytes, the names of yielded DirEntry objects will also be bytes; in all other circumstances they will be str.

If path is None, uses the path='.'.

certbot.compat.os.**sched_get_priority_max**()
Get the maximum scheduling priority for policy.

certbot.compat.os.**sched_get_priority_min**()
Get the minimum scheduling priority for policy.

certbot.compat.os.**sched_getaffinity**()
Return the affinity of the process identified by pid (or the current process if zero).

The affinity is returned as a set of CPU identifiers.

certbot.compat.os.**sched_getparam**()
Returns scheduling parameters for the process identified by pid.

If pid is 0, returns parameters for the calling process. Return value is an instance of sched_param.

certbot.compat.os.**sched_getscheduler**()
Get the scheduling policy for the process identifiedy by pid.

Passing 0 for pid returns the scheduling policy for the calling process.

class certbot.compat.os.**sched_param**
Bases: tuple

Current has only one field: sched_priority");

sched_priority A scheduling parameter.

n_fields = 1

n_sequence_fields = 1

n_unnamed_fields = 0

sched_priority
the scheduling priority

certbot.compat.os.**sched_rr_get_interval**()
Return the round-robin quantum for the process identified by pid, in seconds.

Value returned is a float.

certbot.compat.os.**sched_setaffinity**()
Set the CPU affinity of the process identified by pid to mask.

mask should be an iterable of integers identifying CPUs.

certbot.compat.os.**sched_setparam**()
Set scheduling parameters for the process identified by pid.

If pid is 0, sets parameters for the calling process. param should be an instance of sched_param.

certbot.compat.os.**sched_setscheduler**()
Set the scheduling policy for the process identified by pid.

If pid is 0, the calling process is changed. param is an instance of sched_param.

certbot.compat.os.**sched_yield**()
Voluntarily relinquish the CPU.

certbot.compat.os.**sendfile**(*out*, *in*, *offset*, *count*) → byteswritten

sendfile(out, in, offset, count[, headers][, trailers], flags=0) -> byteswritten

Copy count bytes from file descriptor in to file descriptor out.

certbot.compat.os.**set_blocking**()
Set the blocking mode of the specified file descriptor.

Set the O_NONBLOCK flag if blocking is False, clear the O_NONBLOCK flag otherwise.

certbot.compat.os.**set_inheritable**()
Set the inheritable flag of the specified file descriptor.

certbot.compat.os.**setegid**()
Set the current process's effective group id.

certbot.compat.os.**seteuid**()
Set the current process's effective user id.

certbot.compat.os.**setgid**()
Set the current process's group id.

certbot.compat.os.**setgroups**()
Set the groups of the current process to list.

certbot.compat.os.**setpgid**()
Call the system call setpgid(pid, pgrp).

certbot.compat.os.**setpgrp**()
Make the current process the leader of its process group.

certbot.compat.os.**setpriority**()
Set program scheduling priority.

certbot.compat.os.**setregid**()
Set the current process's real and effective group ids.

certbot.compat.os.**setresgid**()
Set the current process's real, effective, and saved group ids.

certbot.compat.os.**setresuid**()
Set the current process's real, effective, and saved user ids.

certbot.compat.os.**setreuid**()
Set the current process's real and effective user ids.

certbot.compat.os.**setsid**()
Call the system call setsid().

certbot.compat.os.**setuid**()
Set the current process's user id.

certbot.compat.os.**setxattr**()
Set extended attribute attribute on path to value.

path may be either a string, a path-like object, or an open file descriptor. If follow_symlinks is False, and the last element of the path is a symbolic

link, setxattr will modify the symbolic link itself instead of the file the link points to.

class certbot.compat.os.**stat_result**
Bases: tuple

stat_result: Result from stat, fstat, or lstat.

This object may be accessed either as a tuple of (mode, ino, dev, nlink, uid, gid, size, atime, mtime, ctime)

or via the attributes st_mode, st_ino, st_dev, st_nlink, st_uid, and so on.

Posix/windows: If your platform supports st_blksize, st_blocks, st_rdev, or st_flags, they are available as attributes only.

See os.stat for more information.

n_fields = 19

n_sequence_fields = 10

n_unnamed_fields = 3

st_atime
time of last access

st_atime_ns
time of last access in nanoseconds

st_blksize
blocksize for filesystem I/O

st_blocks
number of blocks allocated

st_ctime
time of last change

st_ctime_ns
time of last change in nanoseconds

st_dev
device

st_gid
group ID of owner

st_ino
inode

st_mode
protection bits

st_mtime
time of last modification

st_mtime_ns
time of last modification in nanoseconds

st_nlink
number of hard links

st_rdev
device type (if inode device)

st_size
total size, in bytes

st_uid
user ID of owner

certbot.compat.os.**statvfs**()
Perform a statvfs system call on the given path.

path may always be specified as a string. On some platforms, path may also be specified as an open file descriptor.

If this functionality is unavailable, using it raises an exception.

class certbot.compat.os.**statvfs_result**
Bases: tuple

statvfs_result: Result from statvfs or fstatvfs.

This object may be accessed either as a tuple of (bsize, frsize, blocks, bfree, bavail, files, ffree, favail, flag, namemax),

or via the attributes f_bsize, f_frsize, f_blocks, f_bfree, and so on.

See os.statvfs for more information.

f_bavail

f_bfree

f_blocks

f_bsize

f_favail

f_ffree

f_files

f_flag

f_frsize

f_fsid

f_namemax

n_fields = 11

n_sequence_fields = 10

n_unnamed_fields = 0

certbot.compat.os.**strerror**()
Translate an error code to a message string.

certbot.compat.os.**symlink**()
Create a symbolic link pointing to src named dst.

target_is_directory is required on Windows if the target is to be interpreted as a directory. (On Windows, symlink requires Windows 6.0 or greater, and raises a NotImplementedError otherwise.) target_is_directory is ignored on non-Windows platforms.

If dir_fd is not None, it should be a file descriptor open to a directory, and path should be relative; path will then be relative to that directory.

dir_fd may not be implemented on your platform. If it is unavailable, using it will raise a NotImplementedError.

certbot.compat.os.**sync**()
Force write of everything to disk.

certbot.compat.os.**sysconf**()
Return an integer-valued system configuration variable.

certbot.compat.os.**system**()
Execute the command in a subshell.

certbot.compat.os.**tcgetpgrp**()
Return the process group associated with the terminal specified by fd.

certbot.compat.os.**tcsetpgrp**()
Set the process group associated with the terminal specified by fd.

class certbot.compat.os.**terminal_size**
Bases: tuple

A tuple of (columns, lines) for holding terminal window size

columns
width of the terminal window in characters

lines
height of the terminal window in characters

n_fields = 2

n_sequence_fields = 2

n_unnamed_fields = 0

certbot.compat.os.**times**()
Return a collection containing process timing information.

The object returned behaves like a named tuple with these fields: (utime, stime, cutime, cstime, elapsed_time)

All fields are floating point numbers.

class certbot.compat.os.**times_result**
Bases: tuple

times_result: Result from os.times().

This object may be accessed either as a tuple of (user, system, children_user, children_system, elapsed),

or via the attributes user, system, children_user, children_system, and elapsed.

See os.times for more information.

children_system
system time of children

children_user
user time of children

elapsed
elapsed time since an arbitrary point in the past

n_fields = 5

n_sequence_fields = 5

n_unnamed_fields = 0

system
system time

user
user time

certbot.compat.os.**truncate**()
Truncate a file, specified by path, to a specific length.

On some platforms, path may also be specified as an open file descriptor. If this functionality is unavailable, using it raises an exception.

certbot.compat.os.**ttyname**()
Return the name of the terminal device connected to 'fd'.

fd Integer file descriptor handle.

certbot.compat.os.**uname**()
Return an object identifying the current operating system.

The object behaves like a named tuple with the following fields: (sysname, nodename, release, version, machine)

class certbot.compat.os.**uname_result**
Bases: tuple

uname_result: Result from os.uname().

This object may be accessed either as a tuple of (sysname, nodename, release, version, machine),

or via the attributes sysname, nodename, release, version, and machine.

See os.uname for more information.

machine
hardware identifier

n_fields = 5

n_sequence_fields = 5

n_unnamed_fields = 0

nodename
name of machine on network (implementation-defined)

release
operating system release

sysname
operating system name

version
operating system version

certbot.compat.os.**unlink**()
Remove a file (same as remove()).

If dir_fd is not None, it should be a file descriptor open to a directory, and path should be relative; path will then be relative to that directory.

dir_fd may not be implemented on your platform. If it is unavailable, using it will raise a NotImplementedError.

certbot.compat.os.**unsetenv**()
Delete an environment variable.

certbot.compat.os.**urandom**()
Return a bytes object containing random bytes suitable for cryptographic use.

certbot.compat.os.**utime**()
Set the access and modified time of path.

path may always be specified as a string. On some platforms, path may also be specified as an open file descriptor.

If this functionality is unavailable, using it raises an exception.

If times is not None, it must be a tuple (atime, mtime); atime and mtime should be expressed as float seconds since the epoch.

If ns is specified, it must be a tuple (atime_ns, mtime_ns); atime_ns and mtime_ns should be expressed as integer nanoseconds since the epoch.

If times is None and ns is unspecified, utime uses the current time. Specifying tuples for both times and ns is an error.

If dir_fd is not None, it should be a file descriptor open to a directory, and path should be relative; path will then be relative to that directory.

If follow_symlinks is False, and the last element of the path is a symbolic link, utime will modify the symbolic link itself instead of the file the link points to.

It is an error to use dir_fd or follow_symlinks when specifying path as an open file descriptor.

dir_fd and follow_symlinks may not be available on your platform. If they are unavailable, using them will raise a NotImplementedError.

certbot.compat.os.**wait**()
Wait for completion of a child process.

Returns a tuple of information about the child process: (pid, status)

certbot.compat.os.**wait3**()
Wait for completion of a child process.

Returns a tuple of information about the child process: (pid, status, rusage)

certbot.compat.os.**wait4**()
Wait for completion of a specific child process.

Returns a tuple of information about the child process: (pid, status, rusage)

certbot.compat.os.**waitid**()
Returns the result of waiting for a process or processes.

idtype Must be one of be P_PID, P_PGID or P_ALL.

id The id to wait on.

options Constructed from the ORing of one or more of WEXITED, WSTOPPED or WCONTINUED and additionally may be ORed with WNOHANG or WNOWAIT.

Returns either waitid_result or None if WNOHANG is specified and there are no children in a waitable state.

class certbot.compat.os.**waitid_result**
Bases: tuple

waitid_result: Result from waitid.

This object may be accessed either as a tuple of (si_pid, si_uid, si_signo, si_status, si_code),

or via the attributes si_pid, si_uid, and so on.

See os.waitid for more information.

n_fields = 5

n_sequence_fields = 5

n_unnamed_fields = 0

si_code

si_pid

si_signo

si_status

si_uid

certbot.compat.os.**waitpid**()
Wait for completion of a given child process.

Returns a tuple of information regarding the child process: (pid, status)

The options argument is ignored on Windows.

certbot.compat.os.**write**()
Write a bytes object to a file descriptor.

certbot.compat.os.**writev**()
Iterate over buffers, and write the contents of each to a file descriptor.

Returns the total number of bytes written. buffers must be a sequence of bytes-like objects.

certbot.compat.os.**removedirs**(*name*)
Super-rmdir; remove a leaf directory and all empty intermediate ones. Works like rmdir except that, if the leaf directory is successfully removed, directories corresponding to rightmost path segments will be pruned away until either the whole path is consumed or an error occurs. Errors during this latter phase are ignored – they generally mean that a directory was not empty.

certbot.compat.os.**renames**(*old*, *new*)
Super-rename; create directories as necessary and delete any left empty. Works like rename, except creation of any intermediate directories needed to make the new pathname good is attempted first. After the rename, directories corresponding to rightmost path segments of the old name will be pruned until either the whole path is consumed or a nonempty directory is found.

Note: this function can fail with the new directory structure made if you lack permissions needed to unlink the leaf directory or file.

certbot.compat.os.**walk**(*top*, *topdown=True*, *onerror=None*, *followlinks=False*)
Directory tree generator.

For each directory in the directory tree rooted at top (including top itself, but excluding '.' and '..'), yields a 3-tuple

dirpath, dirnames, filenames

dirpath is a string, the path to the directory. dirnames is a list of the names of the subdirectories in dirpath (excluding '.' and '..'). filenames is a list of the names of the non-directory files in dirpath. Note that the names in the lists are just names, with no path components. To get a full path (which begins with top) to a file or directory in dirpath, do os.path.join(dirpath, name).

If optional arg 'topdown' is true or not specified, the triple for a directory is generated before the triples for any of its subdirectories (directories are generated top down). If topdown is false, the triple for a directory is generated after the triples for all of its subdirectories (directories are generated bottom up).

When topdown is true, the caller can modify the dirnames list in-place (e.g., via del or slice assignment), and walk will only recurse into the subdirectories whose names remain in dirnames; this can be used to prune the search, or to impose a specific order of visiting. Modifying dirnames when topdown is false has no effect on the behavior of os.walk(), since the directories in dirnames have already been generated by the time dirnames itself is generated. No matter the value of topdown, the list of subdirectories is retrieved before the tuples for the directory and its subdirectories are generated.

By default errors from the os.scandir() call are ignored. If optional arg 'onerror' is specified, it should be a function; it will be called with one argument, an OSError instance. It can report the error to continue with the walk, or raise the exception to abort the walk. Note that the filename is available as the filename attribute of the exception object.

By default, os.walk does not follow symbolic links to subdirectories on systems that support them. In order to get this functionality, set the optional argument 'followlinks' to true.

Caution: if you pass a relative pathname for top, don't change the current working directory between resumptions of walk. walk never changes the current directory, and assumes that the client doesn't either.

Example:

import os from os.path import join, getsize for root, dirs, files in os.walk('python/Lib/email'):

print(root, "consumes", end="") print(sum(getsize(join(root, name)) for name in files), end="") print("bytes in", len(files), "non-directory files") if 'CVS' in dirs:

dirs.remove('CVS') # don't visit CVS directories

certbot.compat.os.**fwalk**(*top='.'*, *topdown=True*, *onerror=None*, ***, *follow_symlinks=False*, *dir_fd=None*)

Directory tree generator.

This behaves exactly like walk(), except that it yields a 4-tuple

dirpath, dirnames, filenames, dirfd

`dirpath`, `dirnames` and `filenames` are identical to walk() output, and `dirfd` is a file descriptor referring to the directory `dirpath`.

The advantage of fwalk() over walk() is that it's safe against symlink races (when follow_symlinks is False).

If dir_fd is not None, it should be a file descriptor open to a directory, and top should be relative; top will then be relative to that directory. (dir_fd is always supported for fwalk.)

Caution: Since fwalk() yields file descriptors, those are only valid until the next iteration step, so you should dup() them if you want to keep them for a longer period.

Example:

import os for root, dirs, files, rootfd in os.fwalk('python/Lib/email'):

print(root, "consumes", end="") print(sum(os.stat(name, dir_fd=rootfd).st_size for name in files),

end="")

print("bytes in", len(files), "non-directory files") if 'CVS' in dirs:

dirs.remove('CVS') # don't visit CVS directories

certbot.compat.os.**execl**(*file*, **args*)

Execute the executable file with argument list args, replacing the current process.

certbot.compat.os.**execle**(*file*, **args*, *env*)
Execute the executable file with argument list args and environment env, replacing the current process.

certbot.compat.os.**execlp**(*file*, **args*)
Execute the executable file (which is searched for along $PATH) with argument list args, replacing the current process.

certbot.compat.os.**execlpe**(*file*, **args*, *env*)
Execute the executable file (which is searched for along $PATH) with argument list args and environment env, replacing the current process.

certbot.compat.os.**execvp**(*file*, *args*)
Execute the executable file (which is searched for along $PATH) with argument list args, replacing the current process. args may be a list or tuple of strings.

certbot.compat.os.**execvpe**(*file*, *args*, *env*)
Execute the executable file (which is searched for along $PATH) with argument list args and environment env, replacing the current process. args may be a list or tuple of strings.

certbot.compat.os.**getenv**(*key*, *default=None*)
Get an environment variable, return None if it doesn't exist. The optional second argument can specify an alternate default. key, default and the result are str.

certbot.compat.os.**getenvb**(*key*, *default=None*)
Get an environment variable, return None if it doesn't exist. The optional second argument can specify an alternate default. key, default and the result are bytes.

certbot.compat.os.**spawnv**(*mode*, *file*, *args*) → integer
Execute file with arguments from args in a subprocess. If mode == P_NOWAIT return the pid of the process. If mode == P_WAIT return the process's exit code if it exits normally; otherwise return -SIG, where SIG is the signal that killed it.

certbot.compat.os.**spawnve**(*mode*, *file*, *args*, *env*) → integer
Execute file with arguments from args in a subprocess with the specified environment. If mode == P_NOWAIT return the pid of the process. If mode == P_WAIT return the process's exit code if it exits normally; otherwise return -SIG, where SIG is the signal that killed it.

certbot.compat.os.**spawnvp**(*mode*, *file*, *args*) → integer
Execute file (which is looked for along $PATH) with arguments from args in a subprocess. If mode == P_NOWAIT return the pid of the process. If mode == P_WAIT return the process's exit code if it exits normally; otherwise return -SIG, where SIG is the signal that killed it.

certbot.compat.os.**spawnvpe**(*mode*, *file*, *args*, *env*) → integer
Execute file (which is looked for along $PATH) with arguments from args in a subprocess with the supplied environment. If mode == P_NOWAIT return the pid of the process. If mode == P_WAIT return the process's exit code if it exits normally; otherwise return -SIG, where SIG is the signal that killed it.

certbot.compat.os.**spawnl**(*mode*, *file*, **args*) → integer
Execute file with arguments from args in a subprocess. If mode == P_NOWAIT return the pid of the process. If mode == P_WAIT return the process's exit code if it exits normally; otherwise return -SIG, where SIG is the signal that killed it.

certbot.compat.os.**spawnle**(*mode*, *file*, **args*, *env*) → integer
Execute file with arguments from args in a subprocess with the supplied environment. If mode == P_NOWAIT return the pid of the process. If mode == P_WAIT return the process's exit code if it exits normally; otherwise return -SIG, where SIG is the signal that killed it.

certbot.compat.os.**spawnlp**(*mode*, *file*, **args*) → integer
Execute file (which is looked for along $PATH) with arguments from args in a subprocess with the supplied

environment. If mode == P_NOWAIT return the pid of the process. If mode == P_WAIT return the process's exit code if it exits normally; otherwise return -SIG, where SIG is the signal that killed it.

certbot.compat.os.**spawnlpe**(*mode*, *file*, **args*, *env*) → integer
Execute file (which is looked for along $PATH) with arguments from args in a subprocess with the supplied environment. If mode == P_NOWAIT return the pid of the process. If mode == P_WAIT return the process's exit code if it exits normally; otherwise return -SIG, where SIG is the signal that killed it.

certbot.display package

Certbot display utilities.

Submodules

certbot.display.ops module

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 default 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.**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 message confirming a certificate has been revoked.

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

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 an 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 an 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

certbot.display.util module

Certbot display.

This module (*certbot.display.util*) or its companion *certbot.display.ops* should be used whenever:

- Displaying status information to the user on the terminal
- Collecting information from the user via prompts

Other messages can use the logging module. See log.py.

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.**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

certbot.display.util.**notify**(*msg*)
Display a basic status message.

Parameters **msg** (*str*) – message to display

class certbot.display.util.**FileDisplay**(*outfile*, *force_interactive*)
Bases: object

File-based display.

notification(*message*, *pause=True*, *wrap=True*, *force_interactive=False*, *decorate=True*)
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
- **decorate** (*bool*) – Whether to surround the message with a decorated frame

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.

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

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

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

notification(*message*, *pause=False*, *wrap=True*, *decorate=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
- **decorate** (*bool*) – Whether to apply a decorated frame to the message

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.`**`summarize_domain_list`**(*domains*)

Summarizes a list of domains in the format of: example.com.com and N more domains

or if there is are only two domains: example.com and www.example.com

or if there is only one domain: example.com

Parameters **`domains`** (*`list`*) – `str` list of domains

Returns the domain list summary

Return type str

certbot.plugins package

Certbot plugins.

Submodules

certbot.plugins.common module

Plugin common functions.

`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.

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.

get_ipv6_exploded()
Return IPv6 in normalized form

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.

certbot.plugins.dns_common module

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.

Parameters **add** (*callable*) – Function that proxies calls to argparse.ArgumentParser.add_argument prepending options with unique plugin name prefix.

get_chall_pref(*unused_domain*)

prepare()

perform(*achalls*)

cleanup(*achalls*)

class certbot.plugins.dns_common.**CredentialsConfiguration**(*filename*, *mapper=<function CredentialsConfiguration.<lambda>>*)
Bases: object

Represents a user-supplied filed 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

certbot.plugins.dns_common_lexicon module

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

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

certbot.plugins.dns_test_common module

Base test class for DNS authenticators.

class certbot.plugins.dns_test_common.**BaseAuthenticatorTest**
Bases: object

A base test class to reduce duplication between test code for DNS Authenticator Plugins.

Assumes:

- That subclasses also subclass unittest.TestCase
- That the authenticator is stored as self.auth

achall = KeyAuthorizationAnnotatedChallenge(challb=DNS01(token=b'17817c66b60ce2e4012df

test_more_info()

test_get_chall_pref()

test_parser_arguments()

certbot.plugins.dns_test_common.**write**(*values*, *path*)
Write the specified values to a config file.

Parameters

- **values** (*dict*) – A map of values to write.
- **path** (*str*) – Where to write the values.

certbot.plugins.dns_test_common_lexicon module

Base test class for DNS authenticators built on Lexicon.

class certbot.plugins.dns_test_common_lexicon.**BaseLexiconAuthenticatorTest**
Bases: *certbot.plugins.dns_test_common.BaseAuthenticatorTest*

test_perform()

test_cleanup()

class certbot.plugins.dns_test_common_lexicon.**BaseLexiconClientTest**
Bases: object

DOMAIN_NOT_FOUND = Exception('No domain found')

GENERIC_ERROR
alias of requests.exceptions.RequestException

LOGIN_ERROR = HTTPError('400 Client Error: ...')

UNKNOWN_LOGIN_ERROR = HTTPError('500 Surprise! Error: ...')

record_prefix = '_acme-challenge'

record_name = '_acme-challenge.example.com'

record_content = 'bar'

test_add_txt_record()

test_add_txt_record_try_twice_to_find_domain()

test_add_txt_record_fail_to_find_domain()

test_add_txt_record_fail_to_authenticate()

test_add_txt_record_fail_to_authenticate_with_unknown_error()

test_add_txt_record_error_finding_domain()

test_add_txt_record_error_adding_record()

test_del_txt_record()

test_del_txt_record_fail_to_find_domain()

test_del_txt_record_fail_to_authenticate()

test_del_txt_record_fail_to_authenticate_with_unknown_error()

test_del_txt_record_error_finding_domain()

test_del_txt_record_error_deleting_record()

certbot.plugins.enhancements module

New interface style Certbot enhancements

certbot.plugins.enhancements.**ENHANCEMENTS = ['redirect', 'ensure-http-header', 'ocsp-stapli**
List of possible *certbot.interfaces.IInstaller* enhancements.

List of expected options parameters: - redirect: None - ensure-http-header: name of header (i.e. Strict-Transport-Security) - ocsp-stapling: certificate chain file path

certbot.plugins.enhancements.**enabled_enhancements**(*config*)
Generator to yield the enabled new style enhancements.

Parameters **config** (*certbot.interfaces.IConfig*) – Configuration.

certbot.plugins.enhancements.**are_requested**(*config*)
Checks if one or more of the requested enhancements are those of the new enhancement interfaces.

Parameters **config** (*certbot.interfaces.IConfig*) – Configuration.

certbot.plugins.enhancements.**are_supported**(*config*, *installer*)
Checks that all of the requested enhancements are supported by the installer.

Parameters

- **config** (*certbot.interfaces.IConfig*) – Configuration.
- **installer** (*interfaces.IInstaller*) – Installer object

Returns If all the requested enhancements are supported by the installer

Return type bool

certbot.plugins.enhancements.**enable**(*lineage*, *domains*, *installer*, *config*)
Run enable method for each requested enhancement that is supported.

Parameters

- **lineage** (certbot.interfaces.RenewableCert) – Certificate lineage object
- **domains** (*str*) – List of domains in certificate to enhance
- **installer** (*interfaces.IInstaller*) – Installer object
- **config** (*certbot.interfaces.IConfig*) – Configuration.

certbot.plugins.enhancements.**populate_cli**(*add*)
Populates the command line flags for certbot._internal.cli.HelpfulParser

Parameters **add** (*func*) – Add function of certbot._internal.cli.HelpfulParser

class certbot.plugins.enhancements.**AutoHSTSEnhancement**
Bases: object

Enhancement interface that installer plugins can implement in order to provide functionality that configures the software to have a 'Strict-Transport-Security' with initially low max-age value that will increase over time.

The plugins implementing new style enhancements are responsible of handling the saving of configuration checkpoints as well as calling possible restarts of managed software themselves. For update_autohsts method, the installer may have to call prepare() to finalize the plugin initialization.

Methods: enable_autohsts is called when the header is initially installed using a low max-age value.

update_autohsts is called every time when Certbot is run using 'renew' verb. The max-age value should be increased over time using this method.

deploy_autohsts is called for every lineage that has had its certificate renewed. A long HSTS max-age value should be set here, as we should be confident that the user is able to automatically renew their certificates.

update_autohsts(*lineage*, **args*, ***kwargs*)
Gets called for each lineage every time Certbot is run with 'renew' verb. Implementation of this method should increase the max-age value.

Parameters **lineage** (certbot.interfaces.RenewableCert) – Certificate lineage object

---

Note: prepare() method inherited from interfaces.IPlugin might need to be called manually within implementation of this interface method to finalize the plugin initialization.

---

deploy_autohsts(*lineage*, **args*, ***kwargs*)
Gets called for a lineage when its certificate is successfully renewed. Long max-age value should be set in implementation of this method.

Parameters **lineage** (certbot.interfaces.RenewableCert) – Certificate lineage object

enable_autohsts(*lineage*, *domains*, **args*, ***kwargs*)
Enables the AutoHSTS enhancement, installing Strict-Transport-Security header with a low initial value to be increased over the subsequent runs of Certbot renew.

Parameters

- **lineage** (certbot.interfaces.RenewableCert) – Certificate lineage object
- **domains** (list of str) – List of domains in certificate to enhance

certbot.plugins.storage module

Plugin storage class.

class certbot.plugins.storage.**PluginStorage**(*config*, *classkey*)
Bases: object

Class implementing storage functionality for plugins

save()
Saves PluginStorage content to disk

Raises ***errors.PluginStorageError*** – when unable to serialize the data or write it to the filesystem

put(*key*, *value*)
Put configuration value to PluginStorage

Parameters

- **key** (*str*) – Key to store the value to
- **value** – Data to store

fetch(*key*)
Get configuration value from PluginStorage

Parameters **key** (*str*) – Key to get value from the storage

Raises **KeyError** – If the key doesn't exist in the storage

certbot.plugins.util module

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

certbot.tests package

Utilities for running Certbot tests

Submodules

certbot.tests.acme_util module

ACME utilities for testing.

certbot.tests.acme_util.**gen_combos**(*challbs*)
Generate natural combinations for challbs.

certbot.tests.acme_util.**chall_to_challb**(*chall*, *status*)
Return ChallengeBody from Challenge.

certbot.tests.acme_util.**gen_authzr**(*authz_status*, *domain*, *challs*, *statuses*, *combos=True*)
Generate an authorization resource.

Parameters

- **authz_status** (acme.messages.Status) – Status object
- **challs** (*list*) – Challenge objects

- **statuses** (*list*) – status of each challenge object
- **combos** (*bool*) – Whether or not to add combinations

certbot.tests.util module

Test utilities.

certbot.tests.util.**vector_path**(**names*)
Path to a test vector.

certbot.tests.util.**load_vector**(**names*)
Load contents of a test vector.

certbot.tests.util.**load_cert**(**names*)
Load certificate.

certbot.tests.util.**load_csr**(**names*)
Load certificate request.

certbot.tests.util.**load_comparable_csr**(**names*)
Load ComparableX509 certificate request.

certbot.tests.util.**load_rsa_private_key**(**names*)
Load RSA private key.

certbot.tests.util.**load_pyopenssl_private_key**(**names*)
Load pyOpenSSL private key.

certbot.tests.util.**make_lineage**(*config_dir*, *testfile*, *ec=False*)
Creates a lineage defined by testfile.

This creates the archive, live, and renewal directories if necessary and creates a simple lineage.

Parameters

- **config_dir** (*str*) – path to the configuration directory
- **testfile** (*str*) – configuration file to base the lineage on

Returns path to the renewal conf file for the created lineage

Return type str

certbot.tests.util.**patch_get_utility**(*target='zope.component.getUtility'*)
Patch zope.component.getUtility to use a special mock IDisplay.

The mock IDisplay works like a regular mock object, except it also also asserts that methods are called with valid arguments.

Parameters **target** (*str*) – path to patch

Returns mock zope.component.getUtility

Return type mock.MagicMock

certbot.tests.util.**patch_get_utility_with_stdout**(*target='zope.component.getUtility'*, *stdout=None*)
Patch zope.component.getUtility to use a special mock IDisplay.

The mock IDisplay works like a regular mock object, except it also also asserts that methods are called with valid arguments.

The `message` argument passed to the IDisplay methods is passed to stdout's write method.

Parameters

- **target** (*str*) – path to patch
- **stdout** (*object*) – object to write standard output to; it is expected to have a `write` method

Returns mock zope.component.getUtility

Return type mock.MagicMock

class certbot.tests.util.**FreezableMock**(*frozen=False*, *func=None*, *return_value=sentinel.DEFAULT*)

Bases: `object`

Mock object with the ability to freeze attributes.

This class works like a regular mock.MagicMock object, except attributes and behavior set before the object is frozen cannot be changed during tests.

If a func argument is provided to the constructor, this function is called first when an instance of FreezableMock is called, followed by the usual behavior defined by MagicMock. The return value of func is ignored.

freeze()
Freeze object preventing further changes.

class certbot.tests.util.**TempDirTestCase**(*methodName='runTest'*)

Bases: `unittest.case.TestCase`

Base test class which sets up and tears down a temporary directory

setUp()
Execute before test

tearDown()
Execute after test

class certbot.tests.util.**ConfigTestCase**(*methodName='runTest'*)

Bases: *certbot.tests.util.TempDirTestCase*

Test class which sets up a NamespaceConfig object.

setUp()
Execute before test

certbot.tests.util.**lock_and_call**(*callback*, *path_to_lock*)
Grab a lock on path_to_lock from a foreign process then execute the callback. :param callable callback: object to call after acquiring the lock :param str path_to_lock: path to file or directory to lock

certbot.tests.util.**skip_on_windows**(*reason*)
Decorator to skip permanently a test on Windows. A reason is required.

certbot.tests.util.**temp_join**(*path*)
Return the given path joined to the tempdir path for the current platform Eg.: 'cert' => /tmp/cert (Linux) or 'C:UserscurrentuserAppDataTempcert' (Windows)

9.1.2 Submodules

certbot.achallenges module

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.

acme_type = NotImplemented

challb

class certbot.achallenges.**KeyAuthorizationAnnotatedChallenge**(***kwargs*)
Bases: *certbot.achallenges.AnnotatedChallenge*

Client annotated KeyAuthorizationChallenge challenge.

response_and_validation(**args*, ***kwargs*)
Generate response and validation.

account_key

challb

domain

class certbot.achallenges.**DNS**(***kwargs*)
Bases: *certbot.achallenges.AnnotatedChallenge*

Client annotated "dns" ACME challenge.

acme_type
alias of acme.challenges.DNS

challb

domain

certbot.crypto_util module

Certbot client crypto utility functions.

certbot.crypto_util.**init_save_key**(*key_size*, *key_dir*, *key_type='rsa'*, *elliptic_curve='secp256r1'*, *keyname='key-certbot.pem'*)
Initializes and saves a privkey.

Inits 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*) – key size in bits if key size is rsa.
- **key_dir** (*str*) – Key save directory.
- **key_type** (*str*) – Key Type [rsa, ecdsa]
- **elliptic_curve** (*str*) – Name of the elliptic curve if key type is ecdsa.
- **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=1024*, *key_type='rsa'*, *elliptic_curve=None*)
Generate PEM encoded RSA|EC key.

Parameters

- **bits** (*int*) – Number of bits if key_type=rsa. At least 1024 for RSA.
- **ec_curve** (*str*) – The elliptic curve to use.

Returns new RSA or ECDSA key in PEM form with specified number of bits or of type ec_curve when key_type ecdsa is used.

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** (certbot.interfaces.RenewableCert) – cert to verify

Raises ***errors.Error*** – If verification fails.

certbot.crypto_util.**verify_renewable_cert_sig**(*renewable_cert*)
Verifies the signature of a RenewableCert object.

Parameters **renewable_cert** (certbot.interfaces.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** (certbot.interfaces.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.**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

Raises ***errors.Error*** – If there are less than 2 certificates in the chain.

certbot.crypto_util.**get_serial_from_cert**(*cert_path*)
Retrieve the serial number of a certificate from certificate path

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

Returns serial number of the certificate

Return type int

certbot.crypto_util.**find_chain_with_issuer**(*fullchains*, *issuer_cn*, *warn_on_no_match=False*)
Chooses the first certificate chain from fullchains which contains an Issuer Subject Common Name matching issuer_cn.

Parameters

- **fullchains** (list of str) – The list of fullchains in PEM chain format.
- **issuer_cn** (*str*) – The exact Subject Common Name to match against any issuer in the certificate chain.

Returns The best-matching fullchain, PEM-encoded, or the first if none match.

Return type str

certbot.errors module

Certbot client errors.

exception certbot.errors.**Error**
Bases: 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

certbot.interfaces module

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.

elliptic_curve
The SECG elliptic curve name to use. Please see RFC 8446 for supported values.

key_type
Type of generated private key(Only *ONE* per invocation can be provided at this time)

must_staple
Adds the OCSP Must Staple extension to the certificate. Autoconfigures OCSP Stapling for supported setups (Apache version >= 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.

preferred_chain
If the CA offers multiple certificate chains, prefer the chain with an issuer matching this Subject Common Name. If no match, the default offered chain will be used.

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.**RenewableCert**
Bases: object

Interface to a certificate lineage.

cert_path
Path to the certificate file.

Return type str

key_path
Path to the private key file.

Return type str

chain_path
Path to the certificate chain file.

Return type str

fullchain_path
Path to the full chain file.

The full chain is the certificate file plus the chain file.

Return type str

lineagename
Name given to the certificate lineage.

Return type str

names()
What are the subject names of this certificate?

Returns the subject names

Return type list of str

Raises ***CertStorageError*** – if could not find cert file.

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** (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 certficate 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** (RenewableCert) – Certificate lineage object

certbot.main module

Certbot main public entry point.

certbot.main.**main**(*cli_args=None*)
Run Certbot.

Parameters **cli_args** (list of str) – command line to Certbot, defaults to `sys.argv[1:]`

Returns value for `sys.exit` about the exit status of Certbot

Return type str or int or None

certbot.ocsp package

Tools for checking certificate revocation.

class certbot.ocsp.**RevocationChecker**(*enforce_openssl_binary_usage=False*)
Bases: object

This class figures out OCSP checking on this system, and performs it.

ocsp_revoked(*cert*)
Get revoked status for a particular cert version.

Parameters **cert** (interfaces.RenewableCert) – Certificate object

Returns True if revoked; False if valid or the check failed or cert is expired.

Return type bool

ocsp_revoked_by_paths(*cert_path*, *chain_path*, *timeout=10*)

Performs the OCSP revocation check

Parameters

- **cert_path** (*str*) – Certificate filepath
- **chain_path** (*str*) – Certificate chain
- **timeout** (*int*) – Timeout (in seconds) for the OCSP query

Returns True if revoked; False if valid or the check failed or cert is expired.

Return type bool

certbot.reverter module

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

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.

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

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.

certbot.util module

Utilities for all Certbot.

class certbot.util.**Key**(*file*, *pem*)
Bases: tuple

file
Alias for field number 0

pem
Alias for field number 1

class certbot.util.**CSR**(*file*, *data*, *form*)
Bases: tuple

data
Alias for field number 1

file
Alias for field number 0

form
Alias for field number 2

certbot.util.**env_no_snap_for_external_calls**()
When Certbot is run inside a Snap, certain environment variables are modified. But Certbot sometimes calls out to external programs, since it uses classic confinement. When we do that, we must modify the env to remove our modifications so it will use the system's libraries, since they may be incompatible with the versions of libraries included in the Snap. For example, apachectl, Nginx, and anything run from inside a hook should call this function and pass the results into the `env` argument of `subprocess.Popen`.

Returns A modified copy of os.environ ready to pass to Popen

Return type dict

certbot.util.**run_script**(*params*, *log=<bound method Logger.error of <Logger certbot.util (WARNING)>>*)
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.**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.**DeprecatedArgumentAction**(*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

CHAPTER

TEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

C

INDEX

Symbols

A

B

C

D

E

F

G

H

I

K

O

P

R

S

T

U

V

W

Y