NAME

certbot - Certbot Documentation

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*.

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*.

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*.

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.

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*

Azure Pipelines CI status

System Requirements

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

WHAT IS A CERTIFICATE?

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

talking to is able to insert their own (harmful) content.

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

Certificates and Lineages

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

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

GET CERTBOT

Table of Contents

System Requirements

- Linux, macOS, BSD and Windows
- Recommended root access on Linux/BSD/Required Administrator access on Windows
- Port 80 Open

NOTE:

Certbot is most useful when run with root privileges, because it is then able to automatically configure TLS/SSL for Apache and nginx.

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.

Installation

Unless you have very specific requirements, we kindly suggest that you use the installation instructions for your system found at *https://certbot.eff.org/instructions*.

Snap (Recommended)

Our instructions are the same across all systems that use Snap. You can find instructions for installing Certbot through Snap can be found at *https://certbot.eff.org/instructions* by selecting your server software and then choosing "snapd" in the "System" dropdown menu.

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.

If you unable to use snaps, you can use an alternate method for installing **certbot**.

Alternative 1: 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 certificates 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 certificate for resolves to, *install Docker*, then issue a command like the one found below. If you are using Certbot with the *Standalone* plugin, you will need to make the port it uses

accessible from outside of the container by including something like **-p 80:80** or **-p 443:443** on the command line before **certbot/certbot**.

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

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

There are also Docker images for each of Certbot's DNS plugins available at *https://hub.docker.com/u/certbot* which automate doing domain validation over DNS for popular providers. To use one, just replace **certbot/certbot** in the command above with the name of the image you want to use. For example, to use Certbot's plugin for Amazon Route 53, you'd use **certbot/dns-route53**. You may also need to add flags to Certbot and/or mount additional directories to provide access to your DNS API credentials as specified in the *DNS plugin documentation*.

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

Alternative 2: Pip

Installing Certbot through pip is only supported on a best effort basis and when using a virtual environment. Instructions for installing Certbot through pip can be found at *https://certbot.eff.org/instructions* by selecting your server software and then choosing "pip" in the "System" dropdown menu.

Alternative 3: Third Party Distributions

Third party distributions exist for other specific needs. They often are maintained by these parties outside of Certbot and tend to rapidly fall out of date on LTS-style distributions.

Certbot-Auto [Deprecated]

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.

Please remove **certbot-auto**. To do so, you need to do three things:

1. If you added a cron job or systemd timer to automatically run certbot-auto to renew your certificates, you should delete it. If you did this by following our instructions, you can delete the entry added to **/etc/crontab** by running a command like **sudo sed -i '/certbot-auto/d' /etc/crontab**.

2. Delete the certbot-auto script. If you placed it in **/usr/local/bin‘** like we recommended, you can delete it by running **sudo rm /usr/local/bin/certbot-auto**.

3. Delete the Certbot installation created by certbot-auto by running **sudo rm -rf /opt/eff.org**.

USER GUIDE

Table of Contents

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. Throughout the docs, whenever you see **certbot**, swap in the correct name as needed.

Getting certificates (and choosing plugins)

Certbot helps you achieve two tasks:

1. Obtaining a certificate: automatically performing the required authentication steps to prove that you control the domain(s), saving the certificate to **/etc/letsencrypt/live/** and renewing it on a regular schedule.

2. Optionally, installing that certificate to supported web servers (like Apache or nginx) and other kinds of servers. This is done by automatically modifying the configuration of your server in order to use the certificate.

   To obtain a certificate and also install it, use the **certbot run** command (or **certbot**, which is the same).

   To just obtain the certificate without installing it anywhere, the **certbot certonly** ("certificate only") command can be used.

   Some example ways to use Certbot:

```
# Obtain and install a certificate:
certbot

# Obtain a certificate but don't install it:
certbot certonly

# You may specify multiple domains with -d and obtain and
# install different certificates by running Certbot multiple times:
certbot certonly -d example.com -d www.example.com
certbot certonly -d app.example.com -d api.example.com
```

   To perform these tasks, Certbot will ask you to choose from a selection of authenticator and installer plugins. The appropriate choice of plugins will depend on what kind of server software you are running and plan to use your certificates with.

Authenticators are plugins which automatically perform the required steps to prove that you control the domain names you're trying to request a certificate for. An authenticator is always required to obtain a certificate.

Installers are plugins which can automatically modify your web server's configuration to serve your website over HTTPS, using the certificates obtained by Certbot. An installer is only required if you want Certbot to install the certificate to your web server.

Some plugins are both authenticators and installers and it is possible to specify a distinct *combination* of authenticator and plugin.

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	*http-01* (80)

			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.		
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 Encrypt.	*dns-01* (53)	
manual	Y	N	Obtain a certificate by manually following instructions to perform domain validation yourself. Certificates created this way	*http-01* (80) or *dns-01* (53)	

			do not support autorenewal. Autorenewal may be enabled by providing an authentication hook script to automate the domain validation steps.		

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.

Apache

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

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 oth
```

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_Z6UIyaak

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.

Under Windows, Certbot will generate a **web.config** file, if one does not already exist, in **/.well-known/acme-challenge** in order to let IIS serve the challenge files even if they do not have an extension.

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

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.

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-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*

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"

Renewal with the manual plugin

Certificates created using **--manual do not** support automatic renewal unless combined with an *authentication hook script* via **--manual-auth-hook** to automatically set up the required HTTP and/or TXT challenges.

If you can use one of the other *plugins* which support autorenewal to create your certificate, doing so is highly recommended.

To manually renew a certificate using **--manual** without hooks, repeat the same **certbot --manual** command you used to create the certificate originally. As this will require you to copy and paste new HTTP files or DNS TXT records, the command cannot be automated with a cron job.

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

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
dns-azure	Y	N	DNS Authentication using Azure DNS
dns-godaddy	Y	N	DNS Authentication using Godaddy DNS

dns-yandexcloud	Y	N	DNS Authentication using Yandex Cloud DNS
dns-bunny	Y	N	DNS Authentication using BunnyDNS
njalla	Y	N	DNS Authentication for njalla
DuckDNS	Y	N	DNS Authentication for DuckDNS
Porkbun	Y	N	DNS Authentication for Porkbun
Infomaniak	Y	N	DNS Authentication using Infomaniak Domains API
dns-multi	Y	N	DNS authentication of 100+ providers using go-acme/lego
dns-dnsmanager	Y	N	DNS Authentication for dnsmanager.io
standalone-nfq	Y	N	HTTP Authentication that works with any webserver (Linux only)

+--------------------+--------+------+--------------------------+

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

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 certificates:
  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. The certificate name cannot contain filepath separators (i.e. '/' or '\', depending on the platform). Example:

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

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*.

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

RSA and ECDSA keys

Certbot supports two certificate private key algorithms: **rsa** and **ecdsa**.

As of version 2.0.0, Certbot defaults to ECDSA **secp256r1** (P-256) certificate private keys for all new certificates. Existing certificates will continue to renew using their existing key type, unless a key type change is requested.

The type of key used by Certbot can be controlled through the **--key-type** option. You can use the **--elliptic-curve** option to control the curve used in ECDSA certificates and the **--rsa-key-size** option to control the size of RSA keys.

WARNING:

If you obtain certificates using ECDSA keys, you should be careful not to downgrade to a Certbot version earlier than 1.10.0 where ECDSA keys were not supported. Downgrades like this are possible if you switch from something like the snaps or pip to packages provided by your operating system which often lag behind.

Changing a certificate's key type

Unless you are aware that you need to support very old HTTPS clients that are not supported by most sites, you can safely transition your site to use ECDSA keys instead of RSA keys.

If you want to change a single certificate to use ECDSA keys, you'll need to create or renew a certificate while setting **--key-type ecdsa** on the command line:

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

If you want to use ECDSA keys for all certificates in the future (including renewals of existing certificates), you can add the following line to Certbot's *configuration file*:

key-type = ecdsa

which will take effect upon the next renewal of each certificate.

Revoking certificates

If you need to revoke a certificate, use the **revoke** subcommand to do so.

A certificate may be revoked by providing its name (see **certbot certificates**) or by providing its path directly:

certbot revoke --cert-name example.com

certbot revoke --cert-path /etc/letsencrypt/live/example.com/cert.pem

If the certificate being revoked was obtained via the **--staging**, **--test-cert** or a non-default **--server** flag, that flag must be passed to the **revoke** subcommand.

NOTE:
After revocation, Certbot will (by default) ask whether you want to **delete** the certificate. Unless deleted, Certbot will try to renew revoked certificates the next time **certbot renew** runs.

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-name example.com --reason keycompromise

Revoking by account key or certificate private key

By default, Certbot will try revoke the certificate using your ACME account key. If the certificate was created from the same ACME account, the revocation will be successful.

If you instead have the corresponding private key file to the certificate you wish to revoke, use **--key-path** to perform the revocation from any ACME account:

certbot revoke --cert-path /etc/letsencrypt/live/example.com/cert.pem --key-path /etc/letsencrypt/live/example.com

Deleting certificates

If you need to delete a certificate, use the **delete** subcommand.

NOTE:
Read this and the *Safely deleting certificates* sections carefully. This is an irreversible operation and must be done with care.

Certbot does not automatically revoke a certificate before deleting it. If you're no longer using a certificate and don't plan to use it anywhere else, you may want to follow the instructions in *Revoking certificates* instead. Generally, there's no need to revoke a certificate if its private key has not been compromised, but you may still receive expiration emails from Let's Encrypt

unless you revoke.

NOTE:
Do not manually delete certificate files from inside **/etc/letsencrypt/**. Always use the **delete** subcommand.

A certificate may be deleted by providing its name with **--cert-name**. You may find its name using **certbot certificates**.

Otherwise, you will be prompted to choose one or more certificates to delete:

```
certbot delete --cert-name example.com
# or to choose from a list:
certbot delete
```

Safely deleting certificates

Deleting a certificate without following the proper steps can result in a non-functioning server. To safely delete a certificate, follow all the steps below to make sure that references to a certificate are removed from the configuration of any installed server software (Apache, nginx, Postfix, etc) *before* deleting the certificate.

To explain further, when installing a certificate, Certbot modifies Apache or nginx's configuration to load the certificate and its private key from the **/etc/letsencrypt/live/** directory. Before deleting a certificate, it is necessary to undo that modification, by removing any references to the certificate from the webserver's configuration files.

Follow these steps to safely delete a certificate:

1. Find all references to the certificate (substitute **example.com** in the command for the name of the certificate you wish to delete):

   ```
   sudo bash -c 'grep -R live/example.com /etc/{nginx,httpd,apache2}'
   ```

   If there are no references found, skip directly to Step 4.

   If some references are found, they will look something like:

   ```
   /etc/apache2/sites-available/000-default-le-ssl.conf:SSLCertificateFile /etc/letsencrypt/live/example.com/fullch
   /etc/apache2/sites-available/000-default-le-ssl.conf:SSLCertificateKeyFile /etc/letsencrypt/live/example.com/pr
   ```

2. You will need a self-signed certificate to replace the certificate you are deleting. The following command will generate one for you, saving the certificate at **/etc/letsencrypt/self-signed-cert.pem** and its private key at **/etc/letsencrypt/self-signed-privkey.pem**:

   sudo openssl req -nodes -batch -x509 -newkey rsa:2048 -keyout /etc/letsencrypt/self-signed-privkey.pem -out /

3. For each reference found in Step 1, open the file in a text editor and replace the reference to the existing certificate with a reference to the self-signed certificate.

   Continuing from the previous example, you would open **/etc/apache2/sites-available/000-default-le-ssl.conf** in a text editor and modify the two matching lines of text to instead say:

   SSLCertificateFile /etc/letsencrypt/self-signed-cert.pem
   SSLCertificateKeyFile /etc/letsencrypt/self-signed-privkey.pem

4. It is now safe to delete the certificate. Do so by running:

   sudo certbot delete --cert-name example.com

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:

Most Certbot installations come with automatic renewal out of the box. See *Automated Renewals* for more details.

SEE ALSO:

Users of the *Manual* plugin should note that **--manual** certificates will not renew automatically, unless combined with authentication hook scripts. See *Renewal with the manual plugin*.

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.

Modifying the Renewal Configuration of Existing Certificates

When creating a certificate, Certbot will keep track of all of the relevant options chosen by the user. At renewal time, Certbot will remember these options and apply them once again.

Sometimes, you may encounter the need to change some of these options for future certificate renewals. To achieve this, you will need to perform the following steps:

Certbot v2.3.0 and newer

The **certbot reconfigure** command can be used to change a certificate's renewal options. This command will use the new renewal options to perform a test renewal against the Let's Encrypt staging server. If this is successful, the new renewal options will be saved and will apply to future renewals.

You will need to specify the **--cert-name**, which can be found by running **certbot certificates**.

A list of common options that may be updated with the **reconfigure** command can be found by running **certbot help reconfigure**.

As a practical example, if you were using the **webroot** authenticator and had relocated your website to another directory, you can change the **--webroot-path** to the new directory using the following command:

certbot reconfigure --cert-name example.com --webroot-path /path/to/new/location

Certbot v2.2.0 and older

1. Perform a *dry run renewal* with the amended options on the command line. This allows you to confirm that the change is valid and will result in successful future renewals.

2. If the dry run is successful, perform a *live renewal* of the certificate. This will persist the change for future renewals. If the certificate is not yet due to expire, you will need to force a renewal using **--force-renewal**.

NOTE:
Rate limits from the certificate authority may prevent you from performing multiple renewals in a short period of time. It is strongly recommended to perform the second step only once, when you have decided on what options should change.

As a practical example, if you were using the **webroot** authenticator and had relocated your website to another directory, you would need to change the **--webroot-path** to the new directory. Following the above advice:

1. Perform a *dry-run renewal* of the individual certificate with the amended options:

   certbot renew --cert-name example.com --webroot-path /path/to/new/location --dry-run

2. If the dry-run was successful, make the change permanent by performing a *live renewal* of the certificate with the amended options, including **--force-renewal**:

   certbot renew --cert-name example.com --webroot-path /path/to/new/location --force-renewal

   --cert-name selects the particular certificate to be modified. Without this option, all certificates will be selected.

   --webroot-path is the option intended to be changed. All other previously selected options will be kept the same and do not need to be included in the command.

For advanced certificate management tasks, it is also 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. These renewal configuration files are located at **/etc/letsencrypt/renewal/CERTNAME.conf**. If you choose to modify the renewal configuration file we advise you to make a backup of the file beforehand and test its validity with the **certbot renew --dry-run** command.

WARNING:
Manually modifying files under **/etc/letsencrypt/renewal/** can damage them if done improperly and we do not recommend doing so.

Automated Renewals

Most Certbot installations come with automatic renewals preconfigured. This is done by means of a scheduled task which runs **certbot renew** periodically.

If you are unsure whether you need to configure automated renewal:

1. Review the instructions for your system and installation method at *https://certbot.eff.org/instructions*. They will describe how to set up a scheduled task, if necessary. If no step is listed, your system comes with automated renewal pre-installed, and you should not need to take any additional actions.

2. On Linux and BSD, you can check to see if your installation method has pre-installed a timer for you. To do so, look for the **certbot renew** command in either your system's crontab (typically **/etc/crontab** or **/etc/cron.*/***) or systemd timers (**systemctl list-timers**).

3. If you're still not sure, you can configure automated renewal manually by following the steps in the next section. Certbot has been carefully engineered to handle the case where both manual automated renewal and pre-installed automated renewal are set up.

Setting up automated renewal

If you think you may need to set up automated renewal, follow these instructions to set up a scheduled task to automatically renew your certificates in the background. If you are unsure whether your system has a pre-installed scheduled task for Certbot, it is safe to follow these instructions to create one.

NOTE:

If you're using Windows, these instructions are not neccessary as Certbot on Windows comes with a scheduled task for automated renewal pre-installed.

If you are using macOS and installed Certbot using Homebrew, follow the instructions at *https://certbot.eff.org/instructions* to set up automated renewal. The instructions below are not applicable on macOS.

Run the following line, which will add a cron job to **/etc/crontab**:

SLEEPTIME=$(awk 'BEGIN{srand(); print int(rand()*(3600+1))}'); echo "0 0,12 * * * root sleep $SLEEPTIME

If you needed to stop your webserver to run Certbot, you'll want to add **pre** and **post** hooks to stop and start your webserver automatically. For example, if your webserver is HAProxy, run the following commands to create the hook files in the appropriate directory:

sudo sh -c 'printf "#!/bin/sh\nservice haproxy stop\n" > /etc/letsencrypt/renewal-hooks/pre/haproxy.sh'
sudo sh -c 'printf "#!/bin/sh\nservice haproxy start\n" > /etc/letsencrypt/renewal-hooks/post/haproxy.sh'
sudo chmod 755 /etc/letsencrypt/renewal-hooks/pre/haproxy.sh
sudo chmod 755 /etc/letsencrypt/renewal-hooks/post/haproxy.sh

Congratulations, Certbot will now automatically renew your certificates in the background.

If you are interested in learning more about how Certbot renews your certificates, see the *Renewing certificates* section above.

Where are my certificates?

All generated keys and issued certificates can be found in **/etc/letsencrypt/live/$domain**, where **$domain** is the certificate name (see the note below). Rather than copying, please point your (web) server configuration directly to those files (or create symlinks). During the *renewal*, **/etc/letsencrypt/live** is updated with the latest necessary files.

NOTE:

The certificate name **$domain** used in the path **/etc/letsencrypt/live/$domain** follows this convention:

- it is the name given to **--cert-name**,
- if **--cert-name** is not set by the user it is the first domain given to **--domains**,
- if the first domain is a wildcard domain (eg. ***.example.com**) the certificate name will be **example.com**,
- if a name collision would occur with a certificate already named **example.com**, the new certificate name will be constructed using a numerical sequence as **example.com-001**.

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**.

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*.

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/cl

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-c

/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-cl

/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_PA
    -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":"'"$CERTBOT_VALIDATION"'","ttl":1
          | python -c "import sys,json;print(json.load(sys.stdin)['result']['id'])")
# Save info for cleanup
if [ ! -d /tmp/CERTBOT_$CERTBOT_DOMAIN ];then
      mkdir -m 0700 /tmp/CERTBOT_$CERTBOT_DOMAIN
fi
echo $ZONE_ID > /tmp/CERTBOT_$CERTBOT_DOMAIN/ZONE_ID
echo $RECORD_ID > /tmp/CERTBOT_$CERTBOT_DOMAIN/RECORD_ID

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

   /path/to/dns/cleanup.sh

#!/bin/bash

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

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
```

Changing the ACME Server

By default, Certbot uses Let's Encrypt's production server at *https://acme-v02.api.letsencrypt.org/directory*. 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.

NOTE:

--dry-run uses the Let's Encrypt staging server, unless **--server** is specified on the CLI or in the *cli.ini configuration file*. Take caution when using **--dry-run** with a custom server, as it may cause real certificates to be issued and discarded.

If Certbot does not trust the SSL certificate used by the ACME server, you can use the *REQUESTS_CA_BUNDLE* environment variable to override the root certificates trusted by Certbot. Certbot uses the **requests** library, which does not use the operating system trusted root store. Make sure that **REQUESTS_CA_BUNDLE** is set globally in the environment and not only on the CLI, or scheduled renewal will not succeed.

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.

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

# Uncomment to automatically agree to the terms of service of the ACME server
# agree-tos = true

# An example of using an alternate ACME server that uses EAB credentials
# server = https://acme.sectigo.com/v2/InCommonRSAOV
# eab-kid = somestringofstuffwithoutquotes
# eab-hmac-key = yaddayaddahexhexnotquoted
```

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.

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**. Setting this flag to 0 disables log rotation entirely, causing certbot to always append to the same log file.

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.

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)
    reconfigure     Update a certificate's configuration (supply --cert-name)

manage your account:
    register        Create an ACME account
    unregister      Deactivate an ACME account
    update_account  Update an ACME account
    show_account    Display account details
   --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: 0)
--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 include. For multiple domains you can use multiple -d flags or enter a comma separated list of domains as a parameter. All domains will be included as Subject Alternative Names on the certificate. The first domain will be used as the certificate name, unless otherwise specified or if you already have a certificate with the same name. In the case of a name conflict, a number like -0001 will be appended to the certificate 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. When used in combination with the '-v' option, the challenge URLs or FQDNs and their expected return values are shown. (default: False)

--preferred-chain PREFERRED_CHAIN
Set the preferred certificate chain. If the CA offers multiple certificate chains, prefer the chain whose topmost certificate was issued from 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: [])

--issuance-timeout ISSUANCE_TIMEOUT
This option specifies how long (in seconds) Certbot will wait for the server to issue a certificate.

(default: 90)

--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/2.5.0 (certbot; 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)

--no-reuse-key When renewing, do not use the same private key as the existing certificate. Not reusing private keys is the default behavior of Certbot. This option may be used to unset --reuse-key on an existing certificate. (default: False)

--new-key When renewing or replacing a certificate, generate a new private key, even if --reuse-key is set on the existing certificate. Combining --new-key and --reuse-key will result in the private key being replaced and then reused in future renewals. (default: False)

--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 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)

-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: ecdsa)

--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 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.

--run-deploy-hooks When performing a test run using ‘--dry-run‘ or ‘reconfigure‘, run any applicable deploy hooks. This includes hooks set on the command line, saved in the certificate’s renewal configuration file, or present in the renewal-hooks directory. To exclude directory hooks, use --no-directory-hooks. The hook(s) will only be run if the dry run succeeds, and will use the current active certificate, not the temporary test certificate acquired during the dry run. This flag is

recommended when modifying the deploy hook using ‘reconfigure‘. (default: False)

--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 (default: False)

--no-verify-ssl Disable verification of the ACME server’s certificate. The root certificates trusted by Certbot can be overriden by setting the REQUESTS_CA_BUNDLE environment variable. (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 certonly --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)

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
reconfigure Update renewal configuration for a certificate specified by --cert-name

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 any certificates previously obtained if they are close to expiry, and print a summary of the results. By default, 'renew' will reuse the plugins and options used to obtain or most recently renew each certificate. You can test whether future renewals will succeed with '--dry-run'. Individual certificates can be renewed with the '--cert-name' option. 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. Unless --disable-hook-validation is used, the command's first word must be the absolute pathname of an executable or one found via the PATH environment variable. 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. Unless --disable-hook-validation is used, the command's first word must be the absolute pathname of an executable or one found via the PATH environment variable. 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. Unless --disable-hook-validation is used, the command's first word must be the absolute pathname of an executable or one found via the PATH environment variable. For this command, the shell variable $RENEWED_LINEAGE will point to the config live subdirectory (for example, "/etc/letsencrypt/live/example.com") containing the new certificates and keys; the shell variable $RENEWED_DOMAINS will contain a space-delimited list of renewed certificate domains (for example, "example.com www.example.com") (default: None)

--disable-hook-validation
Ordinarily the commands specified for --pre-hook/--post-hook/--deploy-hook will be checked for

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

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

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

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

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. (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)

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

show_account:
Options useful for the "show_account" subcommand:

reconfigure:
Common options that may be updated with the "reconfigure" subcommand:

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

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

using Cloudflare for DNS). (default: False)
--dns-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: 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-digitalocean:
Obtain certificates 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
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 certificates 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:
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: 120)
--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-sakuracloud:
Obtain certificates using a DNS TXT record (if you are using Sakura Cloud for DNS).

--dns-sakuracloud-propagation-seconds DNS_SAKURACLOUD_PROPAGATION_SECONDS
The number of seconds to wait for DNS to propagate before asking the ACME server to verify the DNS record. (default: 90)

--dns-sakuracloud-credentials DNS_SAKURACLOUD_CREDENTIALS
Sakura Cloud credentials file. (default: None)

manual:
Authenticate through manual configuration or custom shell scripts. When using shell scripts, an authenticator script must be provided. The environment variables available to this script depend on the type of challenge. $CERTBOT_DOMAIN will always contain the domain being authenticated. For HTTP-01 and DNS-01, $CERTBOT_VALIDATION is the validation string, and $CERTBOT_TOKEN is the filename of the resource requested when performing an HTTP-01 challenge. An additional cleanup script can also be provided and can use the additional variable $CERTBOT_AUTH_OUTPUT which contains the stdout output from the auth script. 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:
Runs an HTTP server locally which serves the necessary validation files under the /.well-known/acme-challenge/ request path. Suitable if there is no HTTP server already running. HTTP challenge only (wildcards not supported).

webroot:
Saves the necessary validation files to a .well-known/acme-challenge/ directory within the nominated webroot path. A seperate HTTP server must be running and serving files from the webroot path. HTTP challenge only (wildcards not supported).

--webroot-path WEBROOT_PATH, -w WEBROOT_PATH
public_html / webroot path. This can be specified multiple times to handle different domains; each domain will have the webroot path that preceded it. For instance: '-w /var/www/example -d example.com -d www.example.com -w /var/www/thing -d thing.net -d m.thing.net' (default: Ask)
--webroot-map WEBROOT_MAP
JSON dictionary mapping domains to webroot paths; this implies -d for each entry. You may need to escape this from your shell. E.g.: --webroot-map '{"eg1.is,m.eg1.is":"/www/eg1/", "eg2.is":"/www/eg2"}' This option is merged with, but takes precedence over, -w / -d entries. At present, if you put webroot-map in a config file, it needs to be on a single line, like: webroot-map = {"example.com":"/var/www"}. (default: {})

Getting help

If you're having problems, we recommend posting on the Let's Encrypt *Community Forum*.

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

- copy and paste exact command line used and the output (though mind that the latter might include

some personally identifiable information, including your email and domains)

- copy and paste logs from **/var/log/letsencrypt** (though mind they also might contain personally identifiable information)
- copy and paste **certbot --version** output
- your operating system, including specific version
- specify which installation method you've chosen

DEVELOPER GUIDE

Table of Contents

Getting Started

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.

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 running on a UNIX-like OS, you can run the following commands to install dependencies and set up a virtual environment where you can run Certbot.

Install and configure the OS system dependencies required to run Certbot.

For APT-based distributions (e.g. Debian, Ubuntu ...)

```
sudo apt update
sudo apt install python3-venv libaugeas0
# For RPM-based distributions (e.g. Fedora, CentOS ...)
# NB1: old distributions will use yum instead of dnf
# NB2: RHEL-based distributions use python3X instead of python3 (e.g. python38)
sudo dnf install python3 augeas-libs
# For macOS installations with Homebrew already installed and configured
# NB: If you also run 'brew install python' you don't need the ~/lib
#     directory created below, however, Certbot's Apache plugin won't work
#     if you use Python installed from other sources such as pyenv or the
#     version provided by Apple.
brew install augeas
mkdir ~/lib
ln -s $(brew --prefix)/lib/libaugeas* ~/lib
```

NOTE:
If you have trouble creating the virtual environment below, you may need to install additional dependencies. See the *cryptography project's site* for more information.

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

```
cd certbot
python tools/venv.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 **venv/bin/certbot**, or by activating the virtual environment. You can do the latter by running:

```
source venv/bin/activate
```

After running this command, **certbot** and development tools like **ipdb3**, **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*.

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*.

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

> **NOTE:**
> Running integration tests does not currently work on macOS. See *https://github.com/certbot/certbot/issues/6959*. In the meantime, we recommend developers on macOS open a PR to run integration tests.

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 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/venv.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/venv.py
source venv/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.

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

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

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 *Authenticator*, which implements various ways of proving domain control to a certificate authority, and *Installer*, 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.

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.

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 certificates; 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).

Installer Development

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

Writing your own plugin

NOTE:

The Certbot team is not currently accepting any new 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 *Authenticator* or the *Installer* 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.

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.

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.

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. Mypy does some type checks even without type annotations; we can find bugs in Certbot even without a fully annotated codebase.

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.

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
```

Submitting a pull request

Steps:

0. We recommend you talk with us in a GitHub issue or *Mattermost* before writing a pull request to ensure the changes you're making is something we have the time and interest to review.

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 all the tests we recommend developers run locally. 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.

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.

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*.

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.

Certbot's dependencies

We attempt to pin all of Certbot's dependencies whenever we can for reliability and consistency. Some of the places we have Certbot's dependencies pinned include our snaps, Docker images, Windows installer, CI, and our development environments.

In most cases, the file where dependency versions are specified is **tools/requirements.txt**. The one exception to this is our "oldest" tests where **tools/oldest_constraints.txt** is used instead. The purpose of the "oldest" tests is to ensure Certbot continues to work with the oldest versions of our dependencies which we claim to support. The oldest versions of the dependencies we support should also be declared in our setup.py files to communicate this information to our users.

The choices of whether Certbot's dependencies are pinned and what file is used if they are should be automatically handled for you most of the time by Certbot's tooling. The way it works though is **tools/pip_install.py** (which many of our other tools build on) checks for the presence of environment variables. If **CERTBOT_OLDEST** is set to 1, **tools/oldest_constraints.txt** will be used as constraints for **pip**, otherwise, **tools/requirements.txt** is used as constraints.

Updating dependency versions

tools/requirements.txt and **tools/oldest_constraints.txt** can be updated using **tools/pinning/current/repin.sh** and **tools/pinning/oldest/repin.sh** respectively. This works by using **poetry** to generate pinnings based on a Poetry project defined by the **pyproject.toml** file in the same directory as the script. In many cases, you can just run the script to generate updated dependencies, however, if you need to pin back packages or unpin packages that were previously restricted to an older version, you will need to modify the **pyproject.toml** file. The syntax used by this file is described at *https://python-poetry.org/docs/pyproject/* and how dependencies are specified in this file is further described at *https://python-poetry.org/docs/dependency-specification/*.

If you want to learn more about the design used here, see **tools/pinning/DESIGN.md** in the Certbot repo.

PACKAGING GUIDE

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-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**.

Since version 1.21.0, our packages are cryptographically signed by one of four PGP keys:

- **BF6BCFC89E90747B9A680FD7B6029E8500F7DB16**
- **86379B4F0AF371B50CD9E5FF3402831161D1D280**
- **20F201346BF8F3F455A73F9A780CC99432A28621**
- **F2871B4152AE13C49519111F447BF683AA3B26C3‘**

These keys can be found on major key servers and at *https://dl.eff.org/certbot.pub*.

Releases before 1.21.0 were signed by the PGP key **A2CFB51FA275A7286234E7B24D17C995CD9775F2** which can still be found on major key servers.

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.

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 certificate name (see *Where are my certificates?* for more details)

- *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.

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*

Azure Pipelines CI status

API DOCUMENTATION

certbot package

Certbot client.

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: str, mode: int) -> None

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: int) -> int

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.temp_umask(mask: int) -> Generator[None, None, None]

Apply a umask temporarily, meant to be used in a **with** block. Uses the Certbot implementation of

umask.

Parameters

mask (*int*) -- The user file-creation mode mask to apply temporarily

certbot.compat.filesystem.copy_ownership_and_apply_mode(src: str, dst: str, mode: int, copy_user: bool, copy_group: bool) -> None

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: str, dst: str, copy_user: bool = True, copy_group: bool = True) -> None

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: str, mode: int) -> bool

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: str) -> bool
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: str, mode: int) -> bool
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: str, flags: int, mode: int = 511) -> int

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: str, mode: int = 511) -> None

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: str, mode: int = 511) -> None

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: str, dst: str) -> None

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: str) -> str

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.readlink(link_path: str) -> str

Return a string representing the path to which the symbolic link points.

Parameters

link_path (*str*) -- The symlink path to resolve

Returns

The path the symlink points to

Returns

str

Raise

ValueError if a long path (260> characters) is encountered on Windows

certbot.compat.filesystem.is_executable(path: str) -> bool

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: str) -> bool

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: str, base_mode: int) -> int

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: str, path2: str) -> bool

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: str, min_mode: int) -> bool

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() -> None

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.prepare_virtual_console() -> None

On Windows, ensure that Console Virtual Terminal Sequences are enabled.

certbot.compat.misc.readline_with_timeout(timeout: float, prompt: Optional[str]) -> str

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: str) -> str

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: str) -> str

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_status(cmd_name: str, shell_cmd: str, env: Optional[dict] = None) -> Tuple[int, str, str]

Run a command:

- on Linux command will be run by the standard shell selected with

subprocess.run(shell=True)

- on Windows command will be run in a Powershell shell

This function returns the exit code, and does not log the result and output of the command.

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 subprocess.run

Returns

tuple (**int** returncode, **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.

isort:skip_file

certbot.compat.os.access(*unused_args, **unused_kwargs)

Method os.access() is forbidden

certbot.compat.os.chmod(*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.fstat(*unused_args, **unused_kwargs)

Method os.stat() is forbidden

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

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

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

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

certbot.display package

Certbot display utilities.

Submodules

certbot.display.ops module

Contains UI methods for LE user operations.

certbot.display.ops.get_email(invalid: bool = False, optional: bool = True) -> str
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: List[Account]) -> Optional[Account]
Choose an account.

Parameters
accounts (*list*) -- Containing at least one **Account**

certbot.display.ops.choose_values(values: List[str], question: Optional[str] = None) -> List[str]
Display screen to let user pick one or multiple values from the provided list.

Parameters

- **values** (*list*) -- Values to select from
- **question** (*str*) -- Question to ask to user while choosing values

Returns
List of selected values

Return type
list

certbot.display.ops.choose_names(installer: Optional[*Installer*], question: Optional[str] = None) -> List[str]
Display screen to select domains to validate.

Parameters

- **installer** (*certbot.interfaces.Installer*) -- 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: Iterable[str]) -> List[str]

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: List[str]) -> None
Display a box confirming the installation of HTTPS.

Parameters
domains (*list*) -- domain names which were enabled

certbot.display.ops.success_renewal(unused_domains: List[str]) -> None
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: str) -> None
Display a message confirming a certificate has been revoked.

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

certbot.display.ops.report_executed_command(command_name: str, returncode: int, stdout: str, stderr: str) -> None
Display a message describing the success or failure of an executed process (e.g. hook).

Parameters

- **command_name** (*str*) -- Human-readable description of the executed command
- **returncode** (*int*) -- The exit code of the executed command
- **stdout** (*str*) -- The stdout output of the executed command
- **stderr** (*str*) -- The stderr output of the executed command

certbot.display.ops.validated_input(validator: Callable[[str], Any], *args: Any, **kwargs: Any) -> Tuple[str, str]

Like *input_text*, 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_text*.
- ****kwargs** (*dict*) -- Arguments to be passed to *input_text*.

Returns

as *input_text*

Return type

tuple

certbot.display.ops.validated_directory(validator: Callable[[str], Any], *args: Any, **kwargs: Any) -> Tuple[str, str]

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.notify(msg: str) -> None
Display a basic status message.

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

certbot.display.util.notification(message: str, pause: bool = True, wrap: bool = True, force_interactive: bool = False, decorate: bool = True) -> None
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

certbot.display.util.menu(message: str, choices: Union[List[str], List[Tuple[str, str]]], default: Optional[int] = None, cli_flag: Optional[str] = None, force_interactive: bool = False) -> Tuple[str, int]

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 interaction is not possible
- **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

certbot.display.util.input_text(message: str, default: Optional[str] = None, cli_flag: Optional[str] = None, force_interactive: bool = False) -> Tuple[str, str]

Accept input from the user.

Parameters

- **message** (*str*) -- message to display to the user
- **default** -- default value to return, if interaction is not possible
- **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

certbot.display.util.yesno(message: str, yes_label: str = 'Yes', no_label: str = 'No', default: Optional[bool] = None, cli_flag: Optional[str] = None, force_interactive: bool = False) -> bool

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 interaction is not possible
- **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

certbot.display.util.checklist(message: str, tags: List[str], default: Optional[List[str]] = None, cli_flag: Optional[str] = None, force_interactive: bool = False) -> Tuple[str, List[str]]

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 interaction is not possible
- **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

certbot.display.util.directory_select(message: str, default: Optional[str] = None, cli_flag: Optional[str] = None, force_interactive: bool = False) -> Tuple[str, str]

Display a directory selection screen.

Parameters

- **message** (*str*) -- prompt to give the user
- **default** -- default value to return, if interaction is not possible
- **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: str, default: str, cli_flag: str, force_interactive: bool) -> None

Verify that provided arguments is a valid display 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

certbot.plugins package

Certbot plugins.

Submodules

certbot.plugins.common module

Plugin common functions.

certbot.plugins.common.option_namespace(name: str) -> str

ArgumentParser options namespace (prefix of all options).

certbot.plugins.common.dest_namespace(name: str) -> str

ArgumentParser dest namespace (prefix of all destinations).

class certbot.plugins.common.Plugin(config: NamespaceConfig, name: str)

Bases: *Plugin*

Generic plugin.

abstract classmethod add_parser_arguments(add: Callable[[...], None]) -> None

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: ArgumentParser, name: str) -> None
Inject parser options.

See *inject_parser_options* for docs.

property option_namespace: str
ArgumentParser options namespace (prefix of all options).

option_name(name: str) -> str
Option name (include plugin namespace).

property dest_namespace: str
ArgumentParser dest namespace (prefix of all destinations).

dest(var: str) -> str
Find a destination for given variable **var**.

conf(var: str) -> Any
Find a configuration value for variable **var**.

auth_hint(failed_achalls: List[*AnnotatedChallenge***]) -> str**
Human-readable string to help the user troubleshoot the authenticator.

Shown to the user if one or more of the attempted challenges were not a success.

Should describe, in simple language, what the authenticator tried to do, what went wrong and what the user should try as their "next steps".

TODO: auth_hint belongs in Authenticator but can't be added until the next major version of Certbot. For now, it lives in .Plugin and auth_handler will only call it on authenticators that subclass .Plugin. For now, inherit from *Plugin* to implement and/or override the method.

Parameters
failed_achalls (*list*) -- List of one or more failed challenges (**achallenges.AnnotatedChallenge** subclasses).

Rtype str

class certbot.plugins.common.Installer(*args: Any, **kwargs: Any)

Bases: *Installer*, *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: Set[str], save_notes: str, temporary: bool = False) -> None

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: str) -> None

Timestamp and save changes made through the reverter.

Parameters

title (*str*) -- Title describing checkpoint

Raises

.errors.PluginError -- when an error occurs

recovery_routine() -> None

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() -> None
Rollback temporary checkpoint.

Raises
.errors.PluginError -- when unable to revert config

rollback_checkpoints(rollback: int = 1) -> None
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

property ssl_dhparams: str
Full absolute path to ssl_dhparams file.

property updated_ssl_dhparams_digest: str
Full absolute path to digest of updated ssl_dhparams file.

install_ssl_dhparams() -> None
Copy Certbot's ssl_dhparams file into the system's config dir if required.

class certbot.plugins.common.Configurator(*args: Any, **kwargs: Any)
Bases: *Installer*, *Authenticator*

A plugin that extends certbot.plugins.common.Installer and implements certbot.interfaces.Authenticator

class certbot.plugins.common.Addr(tup: Tuple[str, str], ipv6: bool = 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: str) -> Optional[GenericAddr]
Initialize Addr from string.

normalized_tuple() -> Tuple[str, str]
Normalized representation of addr/port tuple

get_addr() -> str
Return addr part of Addr object.

get_port() -> str
Return port.

get_addr_obj(port: str) -> GenericAddr
Return new address object with same addr and new port.

get_ipv6_exploded() -> str
Return IPv6 in normalized form

class certbot.plugins.common.ChallengePerformer(configurator: *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: *KeyAuthorizationAnnotatedChallenge***, idx: Optional[int] = None) -> 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() -> List[KeyAuthorizationChallengeResponse]

Perform all added challenges.

Returns

challenge responses

Return type

list of **acme.challenges.KeyAuthorizationChallengeResponse**

certbot.plugins.common.install_version_controlled_file(dest_path: str, digest_path: str, src_path: str, all_hashes: Iterable[str]) -> None

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: str, pkg: str) -> Tuple[str, str, str]

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: NamespaceConfig, name: str)

Bases: *Plugin*, *Authenticator*

Base class for DNS Authenticators

classmethod add_parser_arguments(add: Callable[[...], None], default_propagation_seconds: int = 10) -> None

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.

auth_hint(failed_achalls: List[*AnnotatedChallenge***]) -> str**

See certbot.plugins.common.Plugin.auth_hint.

get_chall_pref(unused_domain: str) -> Iterable[Type[Challenge]]

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

prepare() -> None

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() -> str

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

perform(achalls: List[*AnnotatedChallenge***]) -> List[ChallengeResponse]**

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

list of ACME **ChallengeResponse** instances corresponding to each provided **Challenge**.

Return type

collections.List 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: List[*AnnotatedChallenge***]) -> None**

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

class certbot.plugins.dns_common.CredentialsConfiguration(filename: str, mapper: ~typing.Callable[[str], str] = <function CredentialsConfiguration.<lambda>>)

Bases: **object**

Represents a user-supplied filed which stores API credentials.

require(required_variables: Mapping[str, str]) -> None

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: str) -> Optional[str]

Find a configuration value for variable **var**, as transformed by **mapper**.

Parameters

var (*str*) -- The variable to get.

Returns

The value of the variable, if it exists.

Return type

str or None

certbot.plugins.dns_common.validate_file(filename: str) -> None

Ensure that the specified file exists.

certbot.plugins.dns_common.validate_file_permissions(filename: str) -> None

Ensure that the specified file exists and warn about unsafe permissions.

certbot.plugins.dns_common.base_domain_name_guesses(domain: str) -> List[str]

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: str, record_name: str, record_content: str) -> None
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: str, record_name: str, record_content: str) -> None
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: str, lexicon_options: Mapping[str, Any], provider_options: Mapping[str, Any]) -> Union[None, Dict[str, Any]]

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'17817c66b60ce2e4012dfad92657527a'), domain='example.com', account_key=JWKRSA(key=<ComparableRSAKey(<cryptography.hazmat.backends.openssl.rsa._RSA

object>)>))

test_more_info() -> None

test_get_chall_pref() -> None

test_parser_arguments() -> None

certbot.plugins.dns_test_common.write(values: Mapping[str, Any], path: str) -> None
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: *BaseAuthenticatorTest*

test_perform(unused_mock_get_utility: Any) -> None

test_cleanup() -> None

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

DOMAIN_NOT_FOUND = Exception('No domain found')

GENERIC_ERROR
alias of **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() -> None

test_add_txt_record_try_twice_to_find_domain() -> None

test_add_txt_record_fail_to_find_domain() -> None

test_add_txt_record_fail_to_authenticate() -> None

test_add_txt_record_fail_to_authenticate_with_unknown_error() -> None

test_add_txt_record_error_finding_domain() -> None

test_add_txt_record_error_adding_record() -> None

test_del_txt_record() -> None

test_del_txt_record_fail_to_find_domain() -> None

test_del_txt_record_fail_to_authenticate() -> None

test_del_txt_record_fail_to_authenticate_with_unknown_error() -> None

test_del_txt_record_error_finding_domain() -> None

test_del_txt_record_error_deleting_record() -> None

certbot.plugins.enhancements module

New interface style Certbot enhancements

certbot.plugins.enhancements.ENHANCEMENTS = ['redirect', 'ensure-http-header', 'ocsp-stapling']

List of possible *certbot.interfaces.Installer* 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: NamespaceConfig) ->

Generator[Dict[str, Any], None, None]

Generator to yield the enabled new style enhancements.

Parameters

config (*certbot.configuration.NamespaceConfig*) -- Configuration.

certbot.plugins.enhancements.are_requested(config: NamespaceConfig) -> bool

Checks if one or more of the requested enhancements are those of the new enhancement interfaces.

Parameters

config (*certbot.configuration.NamespaceConfig*) -- Configuration.

certbot.plugins.enhancements.are_supported(config: NamespaceConfig, installer: Optional[*Installer***]) -> bool**

Checks that all of the requested enhancements are supported by the installer.

Parameters

- **config** (*certbot.configuration.NamespaceConfig*) -- Configuration.
- **installer** (*interfaces.Installer*) -- Installer object

Returns

If all the requested enhancements are supported by the installer

Return type

bool

certbot.plugins.enhancements.enable(lineage: Optional[*RenewableCert***], domains: Iterable[str], installer: Optional[***Installer***], config: NamespaceConfig) -> None**

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.Installer*) -- Installer object

- **config** (*certbot.configuration.NamespaceConfig*) -- Configuration.

certbot.plugins.enhancements.populate_cli(add: Callable[[...], None]) -> None

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.

abstract update_autohsts(lineage: *RenewableCert***, *args: Any, **kwargs: Any) -> None**

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.Plugin** might need to be called manually within implementation of this interface method to finalize the plugin initialization.

abstract deploy_autohsts(lineage: *RenewableCert***, *args: Any, **kwargs: Any) -> None**
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

abstract enable_autohsts(lineage: Optional[*RenewableCert***], domains: Iterable[str], *args: Any, **kwargs: Any) -> None**
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: NamespaceConfig, classkey: str)
Bases: **object**

Class implementing storage functionality for plugins

save() -> None
Saves PluginStorage content to disk

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

put(key: str, value: Any) -> None
Put configuration value to PluginStorage

Parameters

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

- **value** -- Data to store

fetch(key: str) -> Any

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: str) -> List[str]

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: str) -> bool

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.chall_to_challb(chall: Challenge, status: Status) -> ChallengeBody

Return ChallengeBody from Challenge.

certbot.tests.acme_util.gen_authzr(authz_status: Status, domain: str, challs: Iterable[Challenge], statuses: Iterable[Status]) -> AuthorizationResource

Generate an authorization resource.

Parameters

- **authz_status** (**acme.messages.Status**) -- Status object
- **challs** (*list*) -- Challenge objects
- **statuses** (*list*) -- status of each challenge object

certbot.tests.util module

Test utilities.

class certbot.tests.util.DummyInstaller(*args: Any, **kwargs: Any)

Bases: *Installer*

Dummy installer plugin for test purpose.

get_all_names() -> Iterable[str]

Returns all names that may be authenticated.

Return type

collections.Iterable of **str**

deploy_cert(domain: str, cert_path: str, key_path: str, chain_path: str, fullchain_path: str) -> None

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: str, enhancement: str, options: Optional[Union[List[str], str]] = None) -> 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() -> List[str]

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: Optional[str] = None, temporary: bool = False) -> None
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

config_test() -> None
Make sure the configuration is valid.

Raises
.MisconfigurationError -- when the config is not in a usable state

restart() -> None
Restart or refresh the server content.

Raises
.PluginError -- when server cannot be restarted

classmethod add_parser_arguments(add: Callable[[...], None]) -> None
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.

prepare() -> None

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() -> str

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

certbot.tests.util.vector_path(*names: str) -> str

Path to a test vector.

certbot.tests.util.load_vector(*names: str) -> bytes

Load contents of a test vector.

certbot.tests.util.load_cert(*names: str) -> X509

Load certificate.

certbot.tests.util.load_csr(*names: str) -> X509Req

Load certificate request.

certbot.tests.util.load_comparable_csr(*names: str) -> ComparableX509
Load ComparableX509 certificate request.

certbot.tests.util.load_rsa_private_key(*names: str) -> ComparableRSAKey
Load RSA private key.

certbot.tests.util.load_pyopenssl_private_key(*names: str) -> PKey
Load pyOpenSSL private key.

certbot.tests.util.make_lineage(config_dir: str, testfile: str, ec: bool = True) -> str
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
- **ec** (*bool*) -- True if we generate the lineage with an ECDSA key

Returns
path to the renewal conf file for the created lineage

Return type
str

certbot.tests.util.patch_display_util() -> MagicMock
Patch certbot.display.util to use a special mock display utility.

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

The mock created by this patch mocks out Certbot internals. That is, the mock object will be called by the certbot.display.util functions and the mock returned by that call will be used as the display utility. This was done to simplify the transition from zope.component and mocking certbot.display.util functions directly in test code should be preferred over using this function in

the future.

See *https://github.com/certbot/certbot/issues/8948*

Returns
patch on the function used internally by certbot.display.util to get a display utility instance

Return type
mock.MagicMock

certbot.tests.util.patch_display_util_with_stdout(stdout: Optional[IO] = None) -> MagicMock
Patch certbot.display.util to use a special mock display utility.

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

The mock created by this patch mocks out Certbot internals. That is, the mock object will be called by the certbot.display.util functions and the mock returned by that call will be used as the display utility. This was done to simplify the transition from zope.component and mocking certbot.display.util functions directly in test code should be preferred over using this function in the future.

See *https://github.com/certbot/certbot/issues/8948*

The **message** argument passed to the display utility methods is passed to stdout's write method.

Parameters
stdout (*object*) -- object to write standard output to; it is expected to have a **write** method

Returns
patch on the function used internally by certbot.display.util to get a display utility instance

Return type
mock.MagicMock

class certbot.tests.util.FreezableMock(frozen: bool = False, func: Optional[Callable[[...], Any]] = None, return_value: Any = 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() -> None
Freeze object preventing further changes.

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

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

setUp() -> None
Execute before test

tearDown() -> None
Execute after test

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

Test class which sets up a NamespaceConfig object.

setUp() -> None
Execute before test

certbot.tests.util.lock_and_call(callback: Callable[[], Any], path_to_lock: str) -> None
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: str) -> Callable[[Callable[[...], Any]], Callable[[...], Any]]
Decorator to skip permanently a test on Windows. A reason is required.

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

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: Any)**

Bases: **ImmutableMap**

Client annotated challenge.

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

Variables

~.challb -- Wrapped **ChallengeBody**.

challb

class certbot.achallenges.KeyAuthorizationAnnotatedChallenge(kwargs: Any)**

Bases: *AnnotatedChallenge*

Client annotated **KeyAuthorizationChallenge** challenge.

response_and_validation(*args: Any, **kwargs: Any) -> Any

Generate response and validation.

challb

domain

account_key

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

Client annotated "dns" ACME challenge.

acme_type
alias of **DNS**

challb

domain

class certbot.achallenges.Other(kwargs: Any)**
Bases: *AnnotatedChallenge*

Client annotated ACME challenge of an unknown type.

acme_type
alias of **Challenge**

challb

domain

certbot.crypto_util module
Certbot client crypto utility functions.

certbot.crypto_util.generate_key(key_size: int, key_dir: Optional[str], key_type: str = 'rsa', elliptic_curve: str = 'secp256r1', keyname: str = 'key-certbot.pem', strict_permissions: bool = True) -> *Key*
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*) -- Optional 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
- **strict_permissions** (*bool*) -- If true and key_dir exists, an exception is raised if the directory doesn't have 0700 permissions or isn't owned by the current user.

Returns

Key

Return type

certbot.util.Key

Raises

ValueError -- If unable to generate the key given key_size.

certbot.crypto_util.generate_csr(privkey: *Key***, names: Union[List[str], Set[str]], path: Optional[str], must_staple: bool = False, strict_permissions: bool = True)** -> *CSR*

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*) -- Optional certificate save directory.
- **must_staple** (*bool*) -- If true, include the TLS Feature extension "OCSP Must-Staple"
- **strict_permissions** (*bool*) -- If true and path exists, an exception is raised if the

directory doesn't have 0755 permissions or isn't owned by the current user.

Returns
CSR

Return type
certbot.util.CSR

certbot.crypto_util.valid_csr(csr: bytes) -> bool
Validate CSR.

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

Parameters
csr (*bytes*) -- CSR in PEM.

Returns
Validity of CSR.

Return type
bool

certbot.crypto_util.csr_matches_pubkey(csr: bytes, privkey: bytes) -> bool
Does private key correspond to the subject public key in the CSR?

Parameters

- **csr** (*bytes*) -- CSR in PEM.
- **privkey** (*bytes*) -- 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: str, data: bytes) -> Tuple[int, *CSR*, List[str]]
Import a CSR file, which can be either PEM or DER.

Parameters

- **csrfile** (*str*) -- CSR filename
- **data** (*bytes*) -- 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: int = 1024, key_type: str = 'rsa', elliptic_curve: Optional[str] = None) -> bytes

Generate PEM encoded RSA|EC key.

Parameters

- **bits** (*int*) -- Number of bits if key_type=rsa. At least 1024 for RSA.
- **key_type** (*str*) -- The type of key to generate, but be rsa or ecdsa
- **elliptic_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: Union[str, bytes]) -> bool

Is valid RSA private key?

Parameters

privkey -- Private key file contents in PEM

Returns

Validity of private key.

Return type
bool

certbot.crypto_util.verify_renewable_cert(renewable_cert: *RenewableCert***) -> None**
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: *RenewableCert***) -> None**
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: Union[DSAPublicKey, Ed25519PublicKey, Ed448PublicKey, EllipticCurvePublicKey, RSAPublicKey, X25519PublicKey, X448PublicKey], signature: bytes, payload: bytes, signature_hash_algorithm: HashAlgorithm) -> None
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
- **signature_hash_algorithm** (*hashes.HashAlgorithm*) -- 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: str, key_path: str) -> None

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: *RenewableCert*) -> None

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: bytes) -> Tuple[X509, int]

Load PEM/DER certificate.

Raises

errors.Error --

certbot.crypto_util.get_sans_from_cert(cert: bytes, typ: int = 1) -> List[str]

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(cert: bytes, typ: int = 1) -> List[str]
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.get_names_from_req(csr: bytes, typ: int = 1) -> List[str]
Get a list of domains from a CSR, including the CN if it is set.

Parameters

- **csr** (*str*) -- CSR (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: Union[List[X509], List[ComparableX509]], filetype: int = 1) -> bytes

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: str) -> datetime

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: str) -> datetime

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: str) -> str

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: str) -> Tuple[str, str]
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: str) -> int
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: List[str], issuer_cn: str, warn_on_no_match: bool = False) -> str
Chooses the first certificate chain from fullchains whose topmost intermediate has an Issuer Common Name matching issuer_cn (in other words the first chain which chains to a root whose name matches 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: *Error*

Generic *AccountStorage* error.

exception certbot.errors.AccountNotFound

Bases: *AccountStorageError*

Account not found error.

exception certbot.errors.ReverterError

Bases: *Error*

Certbot Reverter error.

exception certbot.errors.SubprocessError

Bases: *Error*

Subprocess handling error.

exception certbot.errors.CertStorageError
Bases: *Error*

Generic **CertStorage** error.

exception certbot.errors.HookCommandNotFound
Bases: *Error*

Failed to find a hook command in the PATH.

exception certbot.errors.SignalExit
Bases: *Error*

A Unix signal was received while in the ErrorHandler context manager.

exception certbot.errors.OverlappingMatchFound
Bases: *Error*

Multiple lineages matched what should have been a unique result.

exception certbot.errors.LockError
Bases: *Error*

File locking error.

exception certbot.errors.AuthorizationError
Bases: *Error*

Authorization error.

exception certbot.errors.FailedChallenges(failed_achalls: Set[*AnnotatedChallenge***])**
Bases: *AuthorizationError*

Failed challenges error.

Variables
failed_achalls (*set*) -- Failed *AnnotatedChallenge* instances.

exception certbot.errors.PluginError
Bases: *Error*

Certbot Plugin error.

exception certbot.errors.PluginEnhancementAlreadyPresent
Bases: *Error*

Enhancement was already set

exception certbot.errors.PluginSelectionError
Bases: *Error*

A problem with plugin/configurator selection or setup

exception certbot.errors.NoInstallationError
Bases: *PluginError*

Certbot No Installation error.

exception certbot.errors.MisconfigurationError
Bases: *PluginError*

Certbot Misconfiguration error.

exception certbot.errors.NotSupportedError
Bases: *PluginError*

Certbot Plugin function not supported error.

exception certbot.errors.PluginStorageError
Bases: *PluginError*

Certbot Plugin Storage error.

exception certbot.errors.StandaloneBindError(socket_error: OSError, port: int)
Bases: *Error*

Standalone plugin bind error.

exception certbot.errors.ConfigurationError
Bases: *Error*

Configuration sanity error.

exception certbot.errors.MissingCommandlineFlag
Bases: *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.

abstract find_all() -> List[Account]
Find all accounts.

Returns
All found accounts.

Return type
list

abstract load(account_id: str) -> Account
Load an account by its id.

Raises

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

Returns
The account loaded

Return type
.Account

abstract save(account: Account, client: ClientV2) -> None

Save account.

Raises

.AccountStorageError -- if account could not be saved

class certbot.interfaces.Plugin(config: Optional[NamespaceConfig], name: str)

Bases: **object**

Certbot plugin.

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: str = NotImplemented

Short plugin description

name: str = NotImplemented

Unique name of the plugin

abstract prepare() -> None
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.

abstract more_info() -> str
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

abstract classmethod inject_parser_options(parser: ArgumentParser, name: str) -> None
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.

class certbot.interfaces.Authenticator(config: Optional[NamespaceConfig], name: str)

Bases: *Plugin*

Generic Certbot Authenticator.

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

abstract get_chall_pref(domain: str) -> Iterable[Type[Challenge]]

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

abstract perform(achalls: List[*AnnotatedChallenge***]) -> List[ChallengeResponse]**

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

list of ACME **ChallengeResponse** instances corresponding to each provided **Challenge**.

Return type

collections.List 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

abstract cleanup(achalls: List[*AnnotatedChallenge*]) -> None
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

class certbot.interfaces.Installer(config: Optional[NamespaceConfig], name: str)
Bases: *Plugin*

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.

abstract get_all_names() -> Iterable[str]
Returns all names that may be authenticated.

Return type
collections.Iterable of **str**

abstract deploy_cert(domain: str, cert_path: str, key_path: str, chain_path: str, fullchain_path: str) -> None
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

abstract enhance(domain: str, enhancement: str, options: Optional[Union[List[str], str]] = None) -> 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.

abstract supported_enhancements() -> List[str]

Returns a **collections.Iterable** of supported enhancements.

Returns

supported enhancements which should be a subset of *ENHANCEMENTS*

Return type

collections.Iterable of **str**

abstract save(title: Optional[str] = None, temporary: bool = False) -> None

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

abstract rollback_checkpoints(rollback: int = 1) -> None

Revert **rollback** number of configuration checkpoints.

Raises

.PluginError -- when configuration cannot be fully reverted

abstract recovery_routine() -> None

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

abstract config_test() -> None

Make sure the configuration is valid.

Raises

.MisconfigurationError -- when the config is not in a usable state

abstract restart() -> None
Restart or refresh the server content.

Raises
.PluginError -- when server cannot be restarted

class certbot.interfaces.RenewableCert
Bases: **object**

Interface to a certificate lineage.

abstract property cert_path: str
Path to the certificate file.

Return type
str

abstract property key_path: str
Path to the private key file.

Return type
str

abstract property chain_path: str
Path to the certificate chain file.

Return type
str

abstract property fullchain_path: str
Path to the full chain file.

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

Return type
str

abstract property lineagename: str

Name given to the certificate lineage.

Return type
str

abstract names() -> List[str]
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.Installer** such as prepare() and restart()

abstract generic_updates(lineage: *RenewableCert***, *args: Any, **kwargs: Any) -> None**
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.

abstract renew_deploy(lineage: *RenewableCert***, *args: Any, **kwargs: Any) -> None**
Perform updates defined by installer when a certificate has been renewed

If an installer is a subclass of the class containing this method, this function will always be called when a certificate has been renewed by running "certbot renew". For example if a plugin needs to copy a certificate over, or change configuration based on the new certificate.

This method is called once for each lineage renewed

Parameters
lineage (*RenewableCert*) -- Certificate lineage object

class certbot.interfaces.IPluginFactory
Bases: **object**

Compatibility shim for plugins that still use Certbot's old zope.interface classes.

class certbot.interfaces.IPlugin
Bases: **object**

Compatibility shim for plugins that still use Certbot's old zope.interface classes.

class certbot.interfaces.IAuthenticator
Bases: *IPlugin*

Compatibility shim for plugins that still use Certbot's old zope.interface classes.

class certbot.interfaces.IInstaller

Bases: *IPlugin*

Compatibility shim for plugins that still use Certbot's old zope.interface classes.

certbot.main module

Certbot main public entry point.

certbot.main.main(cli_args: Optional[List[str]] = None) -> Optional[Union[str, int]]

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: bool = False)

Bases: **object**

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

ocsp_revoked(cert: *RenewableCert***) -> bool**

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: str, chain_path: str, timeout: int = 10) -> bool

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: NamespaceConfig)

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.configuration.NamespaceConfig**) -- Configuration.

revert_temporary_config() -> None

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: int = 1) -> None

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: Set[str], save_notes: str) -> None

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: Set[str], save_notes: str) -> None

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: bool, *files: str) -> None

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: bool, command: Iterable[str]) -> None

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() -> None

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: str) -> None

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: Optional[str], pem: bytes)

Bases: **NamedTuple**

Container for an optional file path and contents for a PEM-formated private key.

file: Optional[str]

Alias for field number 0

pem: bytes

Alias for field number 1

class certbot.util.CSR(file: Optional[str], data: bytes, form: str)

Bases: **NamedTuple**

Container for an optional file path and contents for a PEM or DER-formatted CSR.

file: Optional[str]

Alias for field number 0

data: bytes

Alias for field number 1

form: str

Alias for field number 2

certbot.util.env_no_snap_for_external_calls() -> Dict[str, str]

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: ~typing.List[str], log: ~typing.Callable[[str], None] = <bound method Logger.error of <Logger certbot.util (WARNING)>>) -> Tuple[str, str]

Run the script with the given params.

Parameters

- **params** (*list*) -- List of parameters to pass to subprocess.run
- **log** (*callable*) -- Logger method to use for errors

certbot.util.exe_exists(exe: str) -> bool

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: str) -> None

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: str, mode: int, strict: bool) -> None

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: str, mode: int = 493, strict: bool = False) -> None

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: str, mode: str = 'w', chmod: Optional[int] = None) -> IO

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: str, chmod: int = 511, mode: str = 'w') -> Tuple[IO, str]

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: str, filename: str, chmod: int = 420, mode: str = 'w') -> Tuple[IO, str]

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: str) -> None

Remove a file that may not exist.

certbot.util.get_filtered_names(all_names: Set[str]) -> Set[str]

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() -> Tuple[str, str]

Get OS name and version

Returns
(os_name, os_version)

Return type
tuple of **str**

certbot.util.get_os_info_ua() -> str
Get OS name and version string for User Agent

Returns
os_ua

Return type
str

certbot.util.get_systemd_os_like() -> List[str]
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: str, filepath: str = '/etc/os-release') -> str
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: bool = False) -> Tuple[str, str]

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: str) -> bool
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: **Action**

Action to log a warning when an argument is used.

certbot.util.add_deprecated_argument(add_argument: Callable[[...], None], argument_name: str, nargs: Union[str, int]) -> None
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: str) -> str
Checks that Let’s Encrypt will consider domain to be valid.

Parameters

domain (**str**) -- 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: Union[str, bytes]) -> str
Method which validates domain value and errors out if the requirements are not met.

Parameters
domain (**str** or **bytes**) -- 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_ipaddress(address: str) -> bool
Is given address string form of IP(v4 or v6) address?

Parameters
address (**str**) -- address to check

Returns
True if address is valid IP address, otherwise return False.

Return type
bool

certbot.util.is_wildcard_domain(domain: Union[str, bytes]) -> bool

"Is domain a wildcard domain?

Parameters

domain (**bytes** or **str**) -- domain to check

Returns

True if domain is a wildcard, otherwise, False

Return type

bool

certbot.util.is_staging(srv: str) -> bool

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: Callable, *args: Any, **kwargs: Any) -> None

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

certbot.util.parse_loose_version(version_string: str) -> List[Union[int, str]]

Parses a version string into its components.

This code and the returned tuple is based on the now deprecated distutils.version.LooseVersion class from the Python standard library. Two LooseVersion classes and two lists as returned by this function should compare in the same way. See *https://github.com/python/cpython/blob/v3.10.0/Lib/distutils/version.py#L205-L347*.

Parameters
version_string (*str*) -- version string

Returns
list of parsed version string components

Return type
list

- *Index*
- *Module Index*
- *Search Page*

AUTHOR
Certbot