Rclone v1.63.1

[This help in MD format] - [Download Rclone v1.63.1]

Usage

Rclone is a command line program to manage files on cloud storage. After download and install, continue here to learn how to use it: Initial configuration, what the basic syntax looks like, describes the various subcommands, the various options, and more.

Configure

First, you'll need to configure rclone. As the object storage systems have quite complicated authentication these are kept in a config file. (See the `--config` entry for how to find the config file and choose its location.)

The easiest way to make the config is to run rclone with the config option:

```
rclone config
```

See the following for detailed instructions for

- 1Fichier
- Akamai Netstorage
- Alias
- Amazon Drive
- Amazon S3
- Backblaze B2
- Box
- Chunker - transparently splits large files for other remotes
- Citrix ShareFile
- Compress
- Combine
- Crypt - to encrypt other remotes
- DigitalOcean Spaces
- Digi Storage
- Dropbox
- Enterprise File Fabric
- FTP
- Google Cloud Storage
- Google Drive
- Google Photos
- Hasher - to handle checksums for other remotes
- HDFS
- HiDrive
- HTTP
- Internet Archive
- Jottacloud
- Koofr
- Mail.ru Cloud

- Mega
- Memory
- Microsoft Azure Blob Storage
- Microsoft OneDrive
- OpenStack Swift / Rackspace Cloudfiles / Blomp Cloud Storage / Memset Memstore
- OpenDrive
- Oracle Object Storage
- Pcloud
- PikPak
- premiumize.me
- put.io
- QingStor
- Seafile
- SFTP
- Sia
- SMB
- Storj
- SugarSync
- Union
- Uptobox
- WebDAV
- Yandex Disk
- Zoho WorkDrive
- The local filesystem

Basic syntax

Rclone syncs a directory tree from one storage system to another.

Its syntax is like this

```
Syntax: [options] subcommand <parameters> <parameters...>
```

Source and destination paths are specified by the name you gave the storage system in the config file then the sub path, e.g. "drive:myfolder" to look at "myfolder" in Google drive.

You can define as many storage paths as you like in the config file.

Please use the `--interactive / -i` flag while learning rclone to avoid accidental data loss.

Subcommands

rclone uses a system of subcommands. For example

```
rclone ls remote:path # lists a remote
rclone copy /local/path remote:path # copies /local/path to the remote
rclone sync --interactive /local/path remote:path # syncs /local/path to the remote
```

The main rclone commands with most used first

- rclone config - Enter an interactive configuration session.
- rclone copy - Copy files from source to dest, skipping already copied.

- rclone sync - Make source and dest identical, modifying destination only.
- rclone bisync - Bidirectional synchronization between two paths.
- rclone move - Move files from source to dest.
- rclone delete - Remove the contents of path.
- rclone purge - Remove the path and all of its contents.
- rclone mkdir - Make the path if it doesn't already exist.
- rclone rmdir - Remove the path.
- rclone rmdirs - Remove any empty directories under the path.
- rclone check - Check if the files in the source and destination match.
- rclone ls - List all the objects in the path with size and path.
- rclone lsd - List all directories/containers/buckets in the path.
- rclone lsl - List all the objects in the path with size, modification time and path.
- rclone md5sum - Produce an md5sum file for all the objects in the path.
- rclone sha1sum - Produce a sha1sum file for all the objects in the path.
- rclone size - Return the total size and number of objects in remote:path.
- rclone version - Show the version number.
- rclone cleanup - Clean up the remote if possible.
- rclone dedupe - Interactively find duplicate files and delete/rename them.
- rclone authorize - Remote authorization.
- rclone cat - Concatenate any files and send them to stdout.
- rclone copyto - Copy files from source to dest, skipping already copied.
- rclone genautocomplete - Output shell completion scripts for rclone.
- rclone gendocs - Output markdown docs for rclone to the directory supplied.
- rclone listremotes - List all the remotes in the config file.
- rclone mount - Mount the remote as a mountpoint.
- rclone moveto - Move file or directory from source to dest.
- rclone obscure - Obscure password for use in the rclone.conf
- rclone cryptcheck - Check the integrity of an encrypted remote.
- rclone about - Get quota information from the remote.

See the commands index for the full list.

Copying single files

rclone normally syncs or copies directories. However, if the source remote points to a file, rclone will just copy that file. The destination remote must point to a directory - rclone will give the error `Failed to create file system for "remote:file": is a file not a directory` if it isn't.

For example, suppose you have a remote with a file in called `test.jpg`, then you could copy just that file like this

```
rclone copy remote:test.jpg /tmp/download
```

The file `test.jpg` will be placed inside `/tmp/download`.

This is equivalent to specifying

```
rclone copy --files-from /tmp/files remote: /tmp/download
```

Where `/tmp/files` contains the single line

```
test.jpg
```

It is recommended to use `copy` when copying individual files, not `sync`. They have pretty much the same effect but `copy` will use a lot less memory.

Syntax of remote paths

The syntax of the paths passed to the rclone command are as follows.

/path/to/dir

This refers to the local file system.

On Windows `\` may be used instead of `/` in local paths **only**, non local paths must use `/`. See local filesystem documentation for more about Windows-specific paths.

These paths needn't start with a leading `/` - if they don't then they will be relative to the current directory.

remote:path/to/dir

This refers to a directory `path/to/dir` on `remote:` as defined in the config file (configured with `rclone config`).

remote:/path/to/dir

On most backends this is refers to the same directory as `remote:path/to/dir` and that format should be preferred. On a very small number of remotes (FTP, SFTP, Dropbox for business) this will refer to a different directory. On these, paths without a leading `/` will refer to your "home" directory and paths with a leading `/` will refer to the root.

:backend:path/to/dir

This is an advanced form for creating remotes on the fly. `backend` should be the name or prefix of a backend (the `type` in the config file) and all the configuration for the backend should be provided on the command line (or in environment variables).

Here are some examples:

```
rclone lsd --http-url https://pub.rclone.org :http:
```

To list all the directories in the root of `https://pub.rclone.org/`.

```
rclone lsf --http-url https://example.com :http:path/to/dir
```

To list files and directories in `https://example.com/path/to/dir/`

```
rclone copy --http-url https://example.com :http:path/to/dir /tmp/dir
```

To copy files and directories in `https://example.com/path/to/dir` to `/tmp/dir`.

```
rclone copy --sftp-host example.com :sftp:path/to/dir /tmp/dir
```

To copy files and directories from `example.com` in the relative directory `path/to/dir` to `/tmp/dir` using sftp.

Connection strings {#connection-strings}

The above examples can also be written using a connection string syntax, so instead of providing the arguments as command line parameters `--http-url https://pub.rclone.org` they are provided as part of the remote specification as a kind of connection string.

```
rclone lsd ":http,url='https://pub.rclone.org':"
rclone lsf ":http,url='https://example.com':path/to/dir"
rclone copy ":http,url='https://example.com':path/to/dir" /tmp/dir
rclone copy :sftp,host=example.com:path/to/dir /tmp/dir
```

These can apply to modify existing remotes as well as create new remotes with the on the fly syntax. This example is equivalent to adding the `--drive-shared-with-me` parameter to the remote `gdrive:`.

```
rclone lsf "gdrive,shared_with_me:path/to/dir"
```

The major advantage to using the connection string style syntax is that it only applies to the remote, not to all the remotes of that type of the command line. A common confusion is this attempt to copy a file shared on google drive to the normal drive which **does not work** because the `--drive-shared-with-me` flag applies to both the source and the destination.

```
rclone copy --drive-shared-with-me gdrive:shared-file.txt gdrive:
```

However using the connection string syntax, this does work.

```
rclone copy "gdrive,shared_with_me:shared-file.txt" gdrive:
```

Note that the connection string only affects the options of the immediate backend. If for example gdriveCrypt is a crypt based on gdrive, then the following command **will not work** as intended, because `shared_with_me` is ignored by the crypt backend:

```
rclone copy "gdriveCrypt,shared_with_me:shared-file.txt" gdriveCrypt:
```

The connection strings have the following syntax

```
remote,parameter=value,parameter2=value2:path/to/dir
:backend,parameter=value,parameter2=value2:path/to/dir
```

If the `parameter` has a `:` or `,` then it must be placed in quotes `"` or `'`, so

```
remote,parameter="colon:value",parameter2="comma,value":path/to/dir
:backend,parameter='colon:value',parameter2='comma,value':path/to/dir
```

If a quoted value needs to include that quote, then it should be doubled, so

```
remote,parameter="with""quote",parameter2='with''quote':path/to/dir
```

This will make `parameter` be `with"quote` and `parameter2` be `with'quote`.

If you leave off the `=parameter` then rclone will substitute `=true` which works very well with flags. For example, to use s3 configured in the environment you could use:

```
rclone lsd :s3,env_auth:
```

Which is equivalent to

```
rclone lsd :s3,env_auth=true:
```

Note that on the command line you might need to surround these connection strings with `"` or `'` to stop the shell interpreting any special characters within them.

If you are a shell master then you'll know which strings are OK and which aren't, but if you aren't sure then enclose them in `"` and use `'` as the inside quote. This syntax works on all OSes.

```
rclone copy ":http,url='https://example.com':path/to/dir" /tmp/dir
```

On Linux/macOS some characters are still interpreted inside `"` strings in the shell (notably `\` and `$` and `"`) so if your strings contain those you can swap the roles of `"` and `'` thus. (This syntax does not work on Windows.)

```
rclone copy ':http,url="https://example.com":path/to/dir' /tmp/dir
```

Connection strings, config and logging

If you supply extra configuration to a backend by command line flag, environment variable or connection string then rclone will add a suffix based on the hash of the config to the name of the remote, eg

```
rclone -vv lsf --s3-chunk-size 20M s3:
```

Has the log message

```
DEBUG : s3: detected overridden config - adding "{Srj1p}" suffix to name
```

This is so rclone can tell the modified remote apart from the unmodified remote when caching the backends.

This should only be noticeable in the logs.

This means that on the fly backends such as

```
rclone -vv lsf :s3,env_auth:
```

Will get their own names

```
DEBUG : :s3: detected overridden config - adding "{YTu53}" suffix to name
```

Valid remote names

Remote names are case sensitive, and must adhere to the following rules:

- May contain number, letter, `_`, `-`, `.`, `+`, `@` and space.
- May not start with `-` or space.
- May not end with space.

Starting with rclone version 1.61, any Unicode numbers and letters are allowed, while in older versions it was limited to plain ASCII (0-9, A-Z, a-z). If you use the same rclone configuration from different shells, which may be configured with different character encoding, you must be cautious to use characters that are possible to write in all of them. This is mostly a problem on Windows, where the console traditionally uses a non-Unicode character set - defined by the so-called "code page".

Quoting and the shell

When you are typing commands to your computer you are using something called the command line shell. This interprets various characters in an OS specific way.

Here are some gotchas which may help users unfamiliar with the shell rules

Linux / OSX

If your names have spaces or shell metacharacters (e.g. `*`, `?`, `$`, `'`, `"`, etc.) then you must quote them. Use single quotes `'` by default.

```
rclone copy 'Important files?' remote:backup
```

If you want to send a `'` you will need to use `"`, e.g.

```
rclone copy "O'Reilly Reviews" remote:backup
```

The rules for quoting metacharacters are complicated and if you want the full details you'll have to consult the manual page for your shell.

Windows

If your names have spaces in you need to put them in `"`, e.g.

```
rclone copy "E:\folder name\folder name\folder name" remote:backup
```

If you are using the root directory on its own then don't quote it (see #464 for why), e.g.

```
rclone copy E:\ remote:backup
```

Copying files or directories with `:` in the names

rclone uses `:` to mark a remote name. This is, however, a valid filename component in non-Windows OSes. The remote name parser will only search for a `:` up to the first `/` so if you need to act on a file or directory like this then use the full path starting with a `/`, or use `./` as a current directory prefix.

So to sync a directory called `sync:me` to a remote called `remote:` use

```
rclone sync --interactive ./sync:me remote:path
```

or

```
rclone sync --interactive /full/path/to/sync:me remote:path
```

Server Side Copy

Most remotes (but not all - see the overview) support server-side copy.

This means if you want to copy one folder to another then rclone won't download all the files and re-upload them; it will instruct the server to copy them in place.

Eg

```
rclone copy s3:oldbucket s3:newbucket
```

Will copy the contents of `oldbucket` to `newbucket` without downloading and re-uploading.

Remotes which don't support server-side copy **will** download and re-upload in this case.

Server side copies are used with `sync` and `copy` and will be identified in the log when using the `-v` flag. The `move` command may also use them if remote doesn't support server-side move directly. This is done by issuing a server-side copy then a delete which is much quicker than a download and re-upload.

Server side copies will only be attempted if the remote names are the same.

This can be used when scripting to make aged backups efficiently, e.g.

```
rclone sync --interactive remote:current-backup remote:previous-backup
rclone sync --interactive /path/to/files remote:current-backup
```

Metadata support {#metadata}

Metadata is data about a file which isn't the contents of the file. Normally rclone only preserves the modification time and the content (MIME) type where possible.

Rclone supports preserving all the available metadata on files (not directories) when using the `--metadata` or `-M` flag.

Exactly what metadata is supported and what that support means depends on the backend. Backends that support metadata have a metadata section in their docs and are listed in the features table (Eg local, s3)

Rclone only supports a one-time sync of metadata. This means that metadata will be synced from the source object to the destination object only when the source object has changed and needs to be re-uploaded. If the metadata subsequently changes on the source object without changing the object itself then it won't be synced to the destination object. This is in line with the way rclone syncs `Content-Type` without the `--metadata` flag.

Using `--metadata` when syncing from local to local will preserve file attributes such as file mode, owner, extended attributes (not Windows).

Note that arbitrary metadata may be added to objects using the `--metadata-set key=value` flag when the object is first uploaded. This flag can be repeated as many times as necessary.

Types of metadata

Metadata is divided into two type. System metadata and User metadata.

Metadata which the backend uses itself is called system metadata. For example on the local backend the system metadata `uid` will store the user ID of the file when used on a unix based platform.

Arbitrary metadata is called user metadata and this can be set however is desired.

When objects are copied from backend to backend, they will attempt to interpret system metadata if it is supplied. Metadata may change from being user metadata to system metadata as objects are copied between different backends. For example copying an object from s3 sets the `content-type` metadata. In a backend which understands this (like `azureblob`) this will become the Content-Type of the object. In a backend which doesn't understand this (like the `local` backend) this will become user metadata. However should the local object be copied back to s3, the Content-Type will be set correctly.

Metadata framework

Rclone implements a metadata framework which can read metadata from an object and write it to the object when (and only when) it is being uploaded.

This metadata is stored as a dictionary with string keys and string values.

There are some limits on the names of the keys (these may be clarified further in the future).

- must be lower case
- may be `a-z` `0-9` containing `.` `-` or `_`
- length is backend dependent

Each backend can provide system metadata that it understands. Some backends can also store arbitrary user metadata.

Where possible the key names are standardized, so, for example, it is possible to copy object metadata from s3 to azureblob for example and metadata will be translated appropriately.

Some backends have limits on the size of the metadata and rclone will give errors on upload if they are exceeded.

Metadata preservation

The goal of the implementation is to

1. Preserve metadata if at all possible
2. Interpret metadata if at all possible

The consequences of 1 is that you can copy an S3 object to a local disk then back to S3 losslessly. Likewise you can copy a local file with file attributes and xattrs from local disk to s3 and back again losslessly.

The consequence of 2 is that you can copy an S3 object with metadata to Azureblob (say) and have the metadata appear on the Azureblob object also.

Standard system metadata

Here is a table of standard system metadata which, if appropriate, a backend may implement.

key	description	example
mode	File type and mode: octal, unix style	0100664

uid	User ID of owner: decimal number	500
gid	Group ID of owner: decimal number	500
rdev	Device ID (if special file) => hexadecimal	0
atime	Time of last access: RFC 3339	2006-01-02T15:04:05.999999999Z07:00
mtime	Time of last modification: RFC 3339	2006-01-02T15:04:05.999999999Z07:00
btime	Time of file creation (birth): RFC 3339	2006-01-02T15:04:05.999999999Z07:00
cache-control	Cache-Control header	no-cache
content-disposition	Content-Disposition header	inline
content-encoding	Content-Encoding header	gzip
content-language	Content-Language header	en-US
content-type	Content-Type header	text/plain

The metadata keys `mtime` and `content-type` will take precedence if supplied in the metadata over reading the `Content-Type` or modification time of the source object.

Hashes are not included in system metadata as there is a well defined way of reading those already.

Options

Rclone has a number of options to control its behaviour.

Options that take parameters can have the values passed in two ways, `--option=value` or `--option value`. However boolean (true/false) options behave slightly differently to the other options in that `--boolean` sets the option to `true` and the absence of the flag sets it to `false`. It is also possible to specify `--boolean=false` or `--boolean=true`. Note that `--boolean false` is not valid - this is parsed as `--boolean` and the `false` is parsed as an extra command line argument for rclone.

Time or duration options {#time-option}

TIME or DURATION options can be specified as a duration string or a time string.

A duration string is a possibly signed sequence of decimal numbers, each with optional fraction and a unit suffix, such as "300ms", "-1.5h" or "2h45m". Default units are seconds or the following abbreviations are valid:

- `ms` - Milliseconds
- `s` - Seconds
- `m` - Minutes

- `h` - Hours
- `d` - Days
- `w` - Weeks
- `M` - Months
- `y` - Years

These can also be specified as an absolute time in the following formats:

- RFC3339 - e.g. `2006-01-02T15:04:05Z` or `2006-01-02T15:04:05+07:00`
- ISO8601 Date and time, local timezone - `2006-01-02T15:04:05`
- ISO8601 Date and time, local timezone - `2006-01-02 15:04:05`
- ISO8601 Date - `2006-01-02` (YYYY-MM-DD)

Size options {#size-option}

Options which use SIZE use KiB (multiples of 1024 bytes) by default. However, a suffix of `B` for Byte, `K` for KiB, `M` for MiB, `G` for GiB, `T` for TiB and `P` for PiB may be used. These are the binary units, e.g. 1, 2**10, 2**20, 2**30 respectively.

--backup-dir=DIR

When using `sync`, `copy` or `move` any files which would have been overwritten or deleted are moved in their original hierarchy into this directory.

If `--suffix` is set, then the moved files will have the suffix added to them. If there is a file with the same path (after the suffix has been added) in DIR, then it will be overwritten.

The remote in use must support server-side move or copy and you must use the same remote as the destination of the sync. The backup directory must not overlap the destination directory without it being excluded by a filter rule.

For example

```
rclone sync --interactive /path/to/local remote:current --backup-dir remote:old
```

will sync `/path/to/local` to `remote:current`, but for any files which would have been updated or deleted will be stored in `remote:old`.

If running rclone from a script you might want to use today's date as the directory name passed to `--backup-dir` to store the old files, or you might want to pass `--suffix` with today's date.

See `--compare-dest` and `--copy-dest`.

--bind string

Local address to bind to for outgoing connections. This can be an IPv4 address (1.2.3.4), an IPv6 address (1234::789A) or host name. If the host name doesn't resolve or resolves to more than one IP address it will give an error.

--bwlimit=BANDWIDTH_SPEC

This option controls the bandwidth limit. For example

```
--bwlimit 10M
```

would mean limit the upload and download bandwidth to 10 MiB/s. **NB** this is **bytes** per second not **bits** per second. To use a single limit, specify the desired bandwidth in KiB/s, or use a suffix B|K|M|G|T|P. The default is `0` which means to not limit bandwidth.

The upload and download bandwidth can be specified separately, as `--bwlimit UP:DOWN`, so

```
--bwlimit 10M:100k
```

would mean limit the upload bandwidth to 10 MiB/s and the download bandwidth to 100 KiB/s. Either limit can be "off" meaning no limit, so to just limit the upload bandwidth you would use

```
--bwlimit 10M:off
```

this would limit the upload bandwidth to 10 MiB/s but the download bandwidth would be unlimited.

When specified as above the bandwidth limits last for the duration of run of the rclone binary.

It is also possible to specify a "timetable" of limits, which will cause certain limits to be applied at certain times. To specify a timetable, format your entries as `WEEKDAY-HH:MM,BANDWIDTH WEEKDAY-HH:MM,BANDWIDTH...` where: `WEEKDAY` is optional element.

- `BANDWIDTH` can be a single number, e.g.`100k` or a pair of numbers for upload:download, e.g.`10M:1M`.
- `WEEKDAY` can be written as the whole word or only using the first 3 characters. It is optional.
- `HH:MM` is an hour from 00:00 to 23:59.

An example of a typical timetable to avoid link saturation during daytime working hours could be:

`--bwlimit "08:00,512k 12:00,10M 13:00,512k 18:00,30M 23:00,off"`

In this example, the transfer bandwidth will be set to 512 KiB/s at 8am every day. At noon, it will rise to 10 MiB/s, and drop back to 512 KiB/sec at 1pm. At 6pm, the bandwidth limit will be set to 30 MiB/s, and at 11pm it will be completely disabled (full speed). Anything between 11pm and 8am will remain unlimited.

An example of timetable with `WEEKDAY` could be:

`--bwlimit "Mon-00:00,512 Fri-23:59,10M Sat-10:00,1M Sun-20:00,off"`

It means that, the transfer bandwidth will be set to 512 KiB/s on Monday. It will rise to 10 MiB/s before the end of Friday. At 10:00 on Saturday it will be set to 1 MiB/s. From 20:00 on Sunday it will be unlimited.

Timeslots without `WEEKDAY` are extended to the whole week. So this example:

`--bwlimit "Mon-00:00,512 12:00,1M Sun-20:00,off"`

Is equivalent to this:

`--bwlimit "Mon-00:00,512Mon-12:00,1M Tue-12:00,1M Wed-12:00,1M Thu-12:00,1M Fri-12:00,1M Sat-12:00,1M Sun-12:00,1M Sun-20:00,off"`

Bandwidth limit apply to the data transfer for all backends. For most backends the directory listing bandwidth is also included (exceptions being the non HTTP backends, `ftp`, `sftp` and `storj`).

Note that the units are **Byte/s**, not **bit/s**. Typically connections are measured in bit/s - to convert divide by 8. For example, let's say you have a 10 Mbit/s connection and you wish rclone to use half of it - 5 Mbit/s. This is 5/8 = 0.625 MiB/s so you would use a `--bwlimit 0.625M` parameter for rclone.

On Unix systems (Linux, macOS, …) the bandwidth limiter can be toggled by sending a `SIGUSR2` signal to rclone. This allows to remove the limitations of a long running rclone transfer and to restore it back to the value specified with `--bwlimit` quickly when needed. Assuming there is only one rclone instance running, you can toggle the limiter like this:

```
kill -SIGUSR2 $(pidof rclone)
```

If you configure rclone with a remote control then you can use change the bwlimit dynamically:

```
rclone rc core/bwlimit rate=1M
```

--bwlimit-file=BANDWIDTH_SPEC

This option controls per file bandwidth limit. For the options see the `--bwlimit` flag.

For example use this to allow no transfers to be faster than 1 MiB/s

```
--bwlimit-file 1M
```

This can be used in conjunction with `--bwlimit`.

Note that if a schedule is provided the file will use the schedule in effect at the start of the transfer.

--buffer-size=SIZE

Use this sized buffer to speed up file transfers. Each `--transfer` will use this much memory for buffering.

When using `mount` or `cmount` each open file descriptor will use this much memory for buffering. See the mount documentation for more details.

Set to `0` to disable the buffering for the minimum memory usage.

Note that the memory allocation of the buffers is influenced by the --use-mmap flag.

--cache-dir=DIR

Specify the directory rclone will use for caching, to override the default.

Default value is depending on operating system:

- Windows `%LocalAppData%\rclone`, if `LocalAppData` is defined.
- macOS `$HOME/Library/Caches/rclone` if `HOME` is defined.
- Unix `$XDG_CACHE_HOME/rclone` if `XDG_CACHE_HOME` is defined, else `$HOME/.cache/rclone` if `HOME` is defined.
- Fallback (on all OS) to `$TMPDIR/rclone`, where `TMPDIR` is the value from [--temp-dir].

You can use the [config paths] command to see the current value.

Cache directory is heavily used by the [VFS File Caching] mount feature, but also by [serve], [GUI] and other parts of rclone.

--check-first

If this flag is set then in a `sync`, `copy` or `move`, rclone will do all the checks to see whether files need to be transferred before doing any of the transfers. Normally rclone would start running transfers as soon as possible.

This flag can be useful on IO limited systems where transfers interfere with checking.

It can also be useful to ensure perfect ordering when using `--order-by`.

If both `--check-first` and `--order-by` are set when doing `rclone move` then rclone will use the transfer thread to delete source files which don't need transferring. This will enable perfect ordering of the transfers and deletes but will cause the transfer stats to have more items in than expected.

Using this flag can use more memory as it effectively sets `--max-backlog` to infinite. This means that all the info on the objects to transfer is held in memory before the transfers start.

--checkers=N

Originally controlling just the number of file checkers to run in parallel, e.g. by `rclone copy`. Now a fairly universal parallelism control used by `rclone` in several places.

Note: checkers do the equality checking of files during a sync. For some storage systems (e.g. S3, Swift, Dropbox) this can take a significant amount of time so they are run in parallel.

The default is to run 8 checkers in parallel. However, in case of slow-reacting backends you may need to lower (rather than increase) this default by setting `--checkers` to 4 or less threads. This is especially advised if you are experiencing backend server crashes during file checking phase (e.g. on subsequent or top-up backups where little or no file copying is done and checking takes up most of the time). Increase this setting only with utmost care, while monitoring your server health and file checking throughput.

-c, --checksum

Normally rclone will look at modification time and size of files to see if they are equal. If you set this flag then rclone will check the file hash and size to determine if files are equal.

This is useful when the remote doesn't support setting modified time and a more accurate sync is desired than just checking the file size.

This is very useful when transferring between remotes which store the same hash type on the object, e.g. Drive and Swift. For details of which remotes support which hash type see the table in the overview section.

Eg `rclone --checksum sync s3:/bucket swift:/bucket` would run much quicker than without the `--checksum` flag.

When using this flag, rclone won't update mtimes of remote files if they are incorrect as it would normally.

--color WHEN

Specify when colors (and other ANSI codes) should be added to the output.

`AUTO` (default) only allows ANSI codes when the output is a terminal

`NEVER` never allow ANSI codes

`ALWAYS` always add ANSI codes, regardless of the output format (terminal or file)

--compare-dest=DIR

When using `sync`, `copy` or `move` DIR is checked in addition to the destination for files. If a file identical to the source is found that file is NOT copied from source. This is useful to copy just files that have changed since the last backup.

You must use the same remote as the destination of the sync. The compare directory must not overlap the destination directory.

See `--copy-dest` and `--backup-dir`.

--config=CONFIG_FILE

Specify the location of the rclone configuration file, to override the default. E.g. `rclone config --config="rclone.conf"`.

The exact default is a bit complex to describe, due to changes introduced through different versions of rclone while preserving backwards compatibility, but in most cases it is as simple as:

- `%APPDATA%/rclone/rclone.conf` on Windows
- `~/.config/rclone/rclone.conf` on other

The complete logic is as follows: Rclone will look for an existing configuration file in any of the following locations, in priority order:

1. `rclone.conf` (in program directory, where rclone executable is)
2. `%APPDATA%/rclone/rclone.conf` (only on Windows)
3. `$XDG_CONFIG_HOME/rclone/rclone.conf` (on all systems, including Windows)
4. `~/.config/rclone/rclone.conf` (see below for explanation of ~ symbol)
5. `~/.rclone.conf`

If no existing configuration file is found, then a new one will be created in the following location:

- On Windows: Location 2 listed above, except in the unlikely event that `APPDATA` is not defined, then location 4 is used instead.
- On Unix: Location 3 if `XDG_CONFIG_HOME` is defined, else location 4.

- Fallback to location 5 (on all OS), when the rclone directory cannot be created, but if also a home directory was not found then path `.rclone.conf` relative to current working directory will be used as a final resort.

The `~` symbol in paths above represent the home directory of the current user on any OS, and the value is defined as following:

- On Windows: `%HOME%` if defined, else `%USERPROFILE%`, or else `%HOMEDRIVE%\%HOMEPATH%`.
- On Unix: `$HOME` if defined, else by looking up current user in OS-specific user database (e.g. passwd file), or else use the result from shell command `cd && pwd`.

If you run `rclone config file` you will see where the default location is for you.

The fact that an existing file `rclone.conf` in the same directory as the rclone executable is always preferred, means that it is easy to run in "portable" mode by downloading rclone executable to a writable directory and then create an empty file `rclone.conf` in the same directory.

If the location is set to empty string `""` or path to a file with name `notfound`, or the os null device represented by value `NUL` on Windows and `/dev/null` on Unix systems, then rclone will keep the config file in memory only.

The file format is basic INI: Sections of text, led by a `[section]` header and followed by `key=value` entries on separate lines. In rclone each remote is represented by its own section, where the section name defines the name of the remote. Options are specified as the `key=value` entries, where the key is the option name without the `--backend-` prefix, in lowercase and with `_` instead of `-`. E.g. option `--mega-hard-delete` corresponds to key `hard_delete`. Only backend options can be specified. A special, and required, key `type` identifies the storage system, where the value is the internal lowercase name as returned by command `rclone help backends`. Comments are indicated by `;` or `#` at the beginning of a line.

Example:

```
[megaremote]
type = mega
user = you@example.com
pass = PDPcQVVjVtzFY-GTdDFozqBhTdsPg3qH
```

Note that passwords are in obscured form. Also, many storage systems uses token-based authentication instead of passwords, and this requires additional steps. It is easier, and safer, to use the interactive command `rclone config` instead of manually editing the configuration file.

The configuration file will typically contain login information, and should therefore have restricted permissions so that only the current user can read it. Rclone tries to ensure this when it writes the file. You may also choose to encrypt the file.

When token-based authentication are used, the configuration file must be writable, because rclone needs to update the tokens inside it.

To reduce risk of corrupting an existing configuration file, rclone will not write directly to it when saving changes. Instead it will first write to a new, temporary, file. If a configuration file already existed, it will (on Unix systems) try to mirror its permissions to the new file. Then it will rename the existing file to a temporary name as backup. Next, rclone will rename the new file to the correct name, before finally cleaning up by deleting the backup file.

If the configuration file path used by rclone is a symbolic link, then this will be evaluated and rclone will write to the resolved path, instead of overwriting the symbolic link. Temporary files used in the process (described above) will be written to the same parent directory as that of the resolved configuration file, but if this directory is also a symbolic link it will not be resolved and the temporary files will be written to the location of the directory symbolic link.

--contimeout=TIME

Set the connection timeout. This should be in go time format which looks like `5s` for 5 seconds, `10m` for 10 minutes, or `3h30m`.

The connection timeout is the amount of time rclone will wait for a connection to go through to a remote object storage system. It is `1m` by default.

--copy-dest=DIR

When using `sync`, `copy` or `move` DIR is checked in addition to the destination for files. If a file identical to the source is found that file is server-side copied from DIR to the destination. This is useful for incremental backup.

The remote in use must support server-side copy and you must use the same remote as the destination of the sync. The compare directory must not overlap the destination directory.

See `--compare-dest` and `--backup-dir`.

--dedupe-mode MODE

Mode to run dedupe command in. One of `interactive`, `skip`, `first`, `newest`, `oldest`, `rename`. The default is `interactive`. See the dedupe command for more information as to what these options mean.

--default-time TIME

If a file or directory does have a modification time rclone can read then rclone will display this fixed time instead.

The default is `2000-01-01 00:00:00 UTC`. This can be configured in any of the ways shown in the time or duration options.

For example `--default-time 2020-06-01` to set the default time to the 1st of June 2020 or `--default-time 0s` to set the default time to the time rclone started up.

--disable FEATURE,FEATURE,...

This disables a comma separated list of optional features. For example to disable server-side move and server-side copy use:

```
--disable move,copy
```

The features can be put in any case.

To see a list of which features can be disabled use:

```
--disable help
```

The features a remote has can be seen in JSON format with:

```
rclone backend features remote:
```

See the overview features and optional features to get an idea of which feature does what.

Note that some features can be set to `true` if they are `true` / `false` feature flag features by prefixing them with `!`. For example the `CaseInsensitive` feature can be forced to `false` with `--disable CaseInsensitive` and forced to `true` with `--disable '!CaseInsensitive'`. In general it isn't a good idea doing this but it may be useful in extremis.

(Note that `!` is a shell command which you will need to escape with single quotes or a backslash on unix like platforms.)

This flag can be useful for debugging and in exceptional circumstances (e.g. Google Drive limiting the total volume of Server Side Copies to 100 GiB/day).

--disable-http2

This stops rclone from trying to use HTTP/2 if available. This can sometimes speed up transfers due to a problem in the Go standard library.

--dscp VALUE

Specify a DSCP value or name to use in connections. This could help QoS system to identify traffic class. BE, EF, DF, LE, CSx and AFxx are allowed.

See the description of differentiated services to get an idea of this field. Setting this to 1 (LE) to identify the flow to SCAVENGER class can avoid occupying too much bandwidth in a network with DiffServ support (RFC 8622).

For example, if you configured QoS on router to handle LE properly. Running:

```
rclone copy --dscp LE from:/from to:/to
```

would make the priority lower than usual internet flows.

This option has no effect on Windows (see golang/go#42728).

-n, --dry-run

Do a trial run with no permanent changes. Use this to see what rclone would do without actually doing it. Useful when setting up the `sync` command which deletes files in the destination.

--expect-continue-timeout=TIME

This specifies the amount of time to wait for a server's first response headers after fully writing the request headers if the request has an "Expect: 100-continue" header. Not all backends support using this.

Zero means no timeout and causes the body to be sent immediately, without waiting for the server to approve. This time does not include the time to send the request header.

The default is `1s`. Set to `0` to disable.

--error-on-no-transfer

By default, rclone will exit with return code 0 if there were no errors.

This option allows rclone to return exit code 9 if no files were transferred between the source and destination. This allows using rclone in scripts, and triggering follow-on actions if data was copied, or skipping if not.

NB: Enabling this option turns a usually non-fatal error into a potentially fatal one - please check and adjust your scripts accordingly!

--fs-cache-expire-duration=TIME

When using rclone via the API rclone caches created remotes for 5 minutes by default in the "fs cache". This means that if you do repeated actions on the same remote then rclone won't have to build it again from scratch, which makes it more efficient.

This flag sets the time that the remotes are cached for. If you set it to `0` (or negative) then rclone won't cache the remotes at all.

Note that if you use some flags, eg `--backup-dir` and if this is set to `0` rclone may build two remotes (one for the source or destination and one for the `--backup-dir` where it may have only built one before.

--fs-cache-expire-interval=TIME

This controls how often rclone checks for cached remotes to expire. See the `--fs-cache-expire-duration` documentation above for more info. The default is 60s, set to 0 to disable expiry.

--header

Add an HTTP header for all transactions. The flag can be repeated to add multiple headers.

If you want to add headers only for uploads use `--header-upload` and if you want to add headers only for downloads use `--header-download`.

This flag is supported for all HTTP based backends even those not supported by `--header-upload` and `--header-download` so may be used as a workaround for those with care.

```
rclone ls remote:test --header "X-Rclone: Foo" --header "X-LetMeIn: Yes"
```

--header-download

Add an HTTP header for all download transactions. The flag can be repeated to add multiple headers.

```
rclone sync --interactive s3:test/src ~/dst --header-download "X-Amz-Meta-Test: Foo" --header-download "X-Amz-Meta-Test2: Bar"
```

See the GitHub issue here for currently supported backends.

--header-upload

Add an HTTP header for all upload transactions. The flag can be repeated to add multiple headers.

```
rclone sync --interactive ~/src s3:test/dst --header-upload "Content-Disposition:
attachment; filename='cool.html'" --header-upload "X-Amz-Meta-Test: FooBar"
```

See the GitHub issue here for currently supported backends.

--human-readable

Rclone commands output values for sizes (e.g. number of bytes) and counts (e.g. number of files) either as *raw* numbers, or in *human-readable* format.

In human-readable format the values are scaled to larger units, indicated with a suffix shown after the value, and rounded to three decimals. Rclone consistently uses binary units (powers of 2) for sizes and decimal units (powers of 10) for counts. The unit prefix for size is according to IEC standard notation, e.g. `Ki` for kibi. Used with byte unit, `1 KiB` means 1024 Byte. In list type of output, only the unit prefix appended to the value (e.g. `9.762Ki`), while in more textual output the full unit is shown (e.g. `9.762 KiB`). For counts the SI standard notation is used, e.g. prefix `k` for kilo. Used with file counts, `1k` means 1000 files.

The various list commands output raw numbers by default. Option `--human-readable` will make them output values in human-readable format instead (with the short unit prefix).

The about command outputs human-readable by default, with a command-specific option `--full` to output the raw numbers instead.

Command size outputs both human-readable and raw numbers in the same output.

The tree command also considers `--human-readable`, but it will not use the exact same notation as the other commands: It rounds to one decimal, and uses single letter suffix, e.g. `K` instead of `Ki`. The reason for this is that it relies on an external library.

The interactive command ncdu shows human-readable by default, and responds to key `u` for toggling human-readable format.

--ignore-case-sync

Using this option will cause rclone to ignore the case of the files when synchronizing so files will not be copied/synced when the existing filenames are the same, even if the casing is different.

--ignore-checksum

Normally rclone will check that the checksums of transferred files match, and give an error "corrupted on transfer" if they don't.

You can use this option to skip that check. You should only use it if you have had the "corrupted on transfer" error message and you are sure you might want to transfer potentially corrupted data.

--ignore-existing

Using this option will make rclone unconditionally skip all files that exist on the destination, no matter the content of these files.

While this isn't a generally recommended option, it can be useful in cases where your files change due to encryption. However, it cannot correct partial transfers in case a transfer was interrupted.

When performing a `move`/`moveto` command, this flag will leave skipped files in the source location unchanged when a file with the same name exists on the destination.

--ignore-size

Normally rclone will look at modification time and size of files to see if they are equal. If you set this flag then rclone will check only the modification time. If `--checksum` is set then it only checks the checksum.

It will also cause rclone to skip verifying the sizes are the same after transfer.

This can be useful for transferring files to and from OneDrive which occasionally misreports the size of image files (see #399 for more info).

-I, --ignore-times

Using this option will cause rclone to unconditionally upload all files regardless of the state of files on the destination.

Normally rclone would skip any files that have the same modification time and are the same size (or have the same checksum if using `--checksum`).

--immutable

Treat source and destination files as immutable and disallow modification.

With this option set, files will be created and deleted as requested, but existing files will never be updated. If an existing file does not match between the source and destination, rclone will give the error `Source and destination exist but do not match: immutable file modified`.

Note that only commands which transfer files (e.g. `sync`, `copy`, `move`) are affected by this behavior, and only modification is disallowed. Files may still be deleted explicitly (e.g. `delete`, `purge`) or implicitly (e.g. `sync`, `move`). Use `copy --immutable` if it is desired to avoid deletion as well as modification.

This can be useful as an additional layer of protection for immutable or append-only data sets (notably backup archives), where modification implies corruption and should not be propagated.

--inplace {#inplace}

The `--inplace` flag changes the behaviour of rclone when uploading files to some backends (backends with the `PartialUploads` feature flag set) such as:

- local
- ftp
- sftp

Without `--inplace` (the default) rclone will first upload to a temporary file with an extension like this where `XXXXXX` represents a random string.

```
original-file-name.XXXXXX.partial
```

(rclone will make sure the final name is no longer than 100 characters by truncating the `original-file-name` part if necessary).

When the upload is complete, rclone will rename the `.partial` file to the correct name, overwriting any existing file at that point. If the upload fails then the `.partial` file will be deleted.

This prevents other users of the backend from seeing partially uploaded files in their new names and prevents overwriting the old file until the new one is completely uploaded.

If the `--inplace` flag is supplied, rclone will upload directly to the final name without creating a `.partial` file.

This means that an incomplete file will be visible in the directory listings while the upload is in progress and any existing files will be overwritten as soon as the upload starts. If the transfer fails then the file will be deleted. This can cause data loss of the existing file if the transfer fails.

Note that on the local file system if you don't use `--inplace` hard links (Unix only) will be broken. And if you do use `--inplace` you won't be able to update in use executables.

Note also that versions of rclone prior to v1.63.0 behave as if the `--inplace` flag is always supplied.

-i, --interactive {#interactive}

This flag can be used to tell rclone that you wish a manual confirmation before destructive operations.

It is **recommended** that you use this flag while learning rclone especially with `rclone sync`.

For example

```
$ rclone delete --interactive /tmp/dir
rclone: delete "important-file.txt"?
y) Yes, this is OK (default)
n) No, skip this
s) Skip all delete operations with no more questions
!) Do all delete operations with no more questions
q) Exit rclone now.
y/n/s/!/q> n
```

The options mean

- `y` : **Yes**, this operation should go ahead. You can also press Return for this to happen. You'll be asked every time unless you choose `s` or `!`.
- `n` : **No**, do not do this operation. You'll be asked every time unless you choose `s` or `!`.

- `s` : **Skip** all the following operations of this type with no more questions. This takes effect until rclone exits. If there are any different kind of operations you'll be prompted for them.
- `!` : **Do all** the following operations with no more questions. Useful if you've decided that you don't mind rclone doing that kind of operation. This takes effect until rclone exits . If there are any different kind of operations you'll be prompted for them.
- `q` : **Quit** rclone now, just in case!

--leave-root

During rmdirs it will not remove root directory, even if it's empty.

--log-file=FILE

Log all of rclone's output to FILE. This is not active by default. This can be useful for tracking down problems with syncs in combination with the `-v` flag. See the [Logging section](#logging) for more info.

If FILE exists then rclone will append to it.

Note that if you are using the `logrotate` program to manage rclone's logs, then you should use the `copytruncate` option as rclone doesn't have a signal to rotate logs.

--log-format LIST

Comma separated list of log format options. Accepted options are `date`, `time`, `microseconds`, `pid`, `longfile`, `shortfile`, `UTC`. Any other keywords will be silently ignored. `pid` will tag log messages with process identifier which useful with `rclone mount --daemon`. Other accepted options are explained in the [go documentation](#). The default log format is "`date`,`time`".

--log-level LEVEL

This sets the log level for rclone. The default log level is `NOTICE`.

`DEBUG` is equivalent to `-vv`. It outputs lots of debug info - useful for bug reports and really finding out what rclone is doing.

`INFO` is equivalent to `-v`. It outputs information about each transfer and prints stats once a minute by default.

`NOTICE` is the default log level if no logging flags are supplied. It outputs very little when things are working normally. It outputs warnings and significant events.

`ERROR` is equivalent to `-q`. It only outputs error messages.

--use-json-log

This switches the log format to JSON for rclone. The fields of json log are level, msg, source, time.

--low-level-retries NUMBER

This controls the number of low level retries rclone does.

A low level retry is used to retry a failing operation - typically one HTTP request. This might be uploading a chunk of a big file for example. You will see low level retries in the log with the `-v` flag.

This shouldn't need to be changed from the default in normal operations. However, if you get a lot of low level retries you may wish to reduce the value so rclone moves on to a high level retry (see the `--retries` flag) quicker.

Disable low level retries with `--low-level-retries 1`.

--max-backlog=N

This is the maximum allowable backlog of files in a sync/copy/move queued for being checked or transferred.

This can be set arbitrarily large. It will only use memory when the queue is in use. Note that it will use in the order of N KiB of memory when the backlog is in use.

Setting this large allows rclone to calculate how many files are pending more accurately, give a more accurate estimated finish time and make `--order-by` work more accurately.

Setting this small will make rclone more synchronous to the listings of the remote which may be desirable.

Setting this to a negative number will make the backlog as large as possible.

--max-delete=N

This tells rclone not to delete more than N files. If that limit is exceeded then a fatal error will be generated and rclone will stop the operation in progress.

--max-delete-size=SIZE

Rclone will stop deleting files when the total size of deletions has reached the size specified. It defaults to off.

If that limit is exceeded then a fatal error will be generated and rclone will stop the operation in progress.

--max-depth=N

This modifies the recursion depth for all the commands except purge.

So if you do `rclone --max-depth 1 ls remote:path` you will see only the files in the top level directory. Using `--max-depth 2` means you will see all the files in first two directory levels and so on.

For historical reasons the `lsd` command defaults to using a `--max-depth` of 1 - you can override this with the command line flag.

You can use this command to disable recursion (with `--max-depth 1`).

Note that if you use this with `sync` and `--delete-excluded` the files not recursed through are considered excluded and will be deleted on the destination. Test first with `--dry-run` if you are not sure what will happen.

--max-duration=TIME

Rclone will stop scheduling new transfers when it has run for the duration specified.

Defaults to off.

When the limit is reached any existing transfers will complete.

Rclone won't exit with an error if the transfer limit is reached.

--max-transfer=SIZE

Rclone will stop transferring when it has reached the size specified. Defaults to off.

When the limit is reached all transfers will stop immediately.

Rclone will exit with exit code 8 if the transfer limit is reached.

-M, --metadata

Setting this flag enables rclone to copy the metadata from the source to the destination. For local backends this is ownership, permissions, xattr etc. See the [#metadata](metadata section) for more info.

--metadata-set key=value

Add metadata `key` = `value` when uploading. This can be repeated as many times as required. See the [#metadata](metadata section) for more info.

--cutoff-mode=hard|soft|cautious

This modifies the behavior of `--max-transfer` Defaults to `--cutoff-mode=hard`.

Specifying `--cutoff-mode=hard` will stop transferring immediately when Rclone reaches the limit.

Specifying `--cutoff-mode=soft` will stop starting new transfers when Rclone reaches the limit.

Specifying `--cutoff-mode=cautious` will try to prevent Rclone from reaching the limit.

--modify-window=TIME

When checking whether a file has been modified, this is the maximum allowed time difference that a file can have and still be considered equivalent.

The default is `1ns` unless this is overridden by a remote. For example OS X only stores modification times to the nearest second so if you are reading and writing to an OS X filing system this will be `1s` by default.

This command line flag allows you to override that computed default.

--multi-thread-write-buffer-size=SIZE

When downloading with multiple threads, rclone will buffer SIZE bytes in memory before writing to disk for each thread.

This can improve performance if the underlying filesystem does not deal well with a lot of small writes in different positions of the file, so if you see downloads being limited by disk write speed, you might want to experiment with different values. Specially for magnetic drives and remote file systems a higher value can be useful.

Nevertheless, the default of `128k` should be fine for almost all use cases, so before changing it ensure that network is not really your bottleneck.

As a final hint, size is not the only factor: block size (or similar concept) can have an impact. In one case, we observed that exact multiples of 16k performed much better than other values.

--multi-thread-cutoff=SIZE

When downloading files to the local backend above this size, rclone will use multiple threads to download the file (default 250M).

Rclone preallocates the file (using `fallocate(FALLOC_FL_KEEP_SIZE)` on unix or `NTSetInformationFile` on Windows both of which takes no time) then each thread writes directly into the file at the correct place. This means that rclone won't create fragmented or sparse files and there won't be any assembly time at the end of the transfer.

The number of threads used to download is controlled by `--multi-thread-streams`.

Use `-vv` if you wish to see info about the threads.

This will work with the `sync`/`copy`/`move` commands and friends `copyto`/`moveto`. Multi thread downloads will be used with `rclone mount` and `rclone serve` if `--vfs-cache-mode` is set to `writes` or above.

NB that this **only** works for a local destination but will work with any source.

NB that multi thread copies are disabled for local to local copies as they are faster without unless `--multi-thread-streams` is set explicitly.

NB on Windows using multi-thread downloads will cause the resulting files to be sparse. Use `--local-no-sparse` to disable sparse files (which may cause long delays at the start of downloads) or disable multi-thread downloads with `--multi-thread-streams 0`

--multi-thread-streams=N

When using multi thread downloads (see above `--multi-thread-cutoff`) this sets the maximum number of streams to use. Set to `0` to disable multi thread downloads (Default 4).

Exactly how many streams rclone uses for the download depends on the size of the file. To calculate the number of download streams Rclone divides the size of the file by the `--multi-thread-cutoff` and rounds up, up to the maximum set with `--multi-thread-streams`.

So if `--multi-thread-cutoff 250M` and `--multi-thread-streams 4` are in effect (the defaults):

- 0..250 MiB files will be downloaded with 1 stream
- 250..500 MiB files will be downloaded with 2 streams
- 500..750 MiB files will be downloaded with 3 streams
- 750+ MiB files will be downloaded with 4 streams

--no-check-dest

The `--no-check-dest` can be used with `move` or `copy` and it causes rclone not to check the destination at all when copying files.

This means that:

- the destination is not listed minimising the API calls
- files are always transferred
- this can cause duplicates on remotes which allow it (e.g. Google Drive)
- `--retries 1` is recommended otherwise you'll transfer everything again on a retry

This flag is useful to minimise the transactions if you know that none of the files are on the destination.

This is a specialized flag which should be ignored by most users!

--no-gzip-encoding

Don't set `Accept-Encoding: gzip`. This means that rclone won't ask the server for compressed files automatically. Useful if you've set the server to return files with `Content-Encoding: gzip` but you uploaded compressed files.

There is no need to set this in normal operation, and doing so will decrease the network transfer efficiency of rclone.

--no-traverse

The `--no-traverse` flag controls whether the destination file system is traversed when using the `copy` or `move` commands. `--no-traverse` is not compatible with `sync` and will be ignored if you supply it with `sync`.

If you are only copying a small number of files (or are filtering most of the files) and/or have a large number of files on the destination then `--no-traverse` will stop rclone listing the destination and save time.

However, if you are copying a large number of files, especially if you are doing a copy where lots of the files under consideration haven't changed and won't need copying then you shouldn't use `--no-traverse`.

See rclone copy for an example of how to use it.

--no-unicode-normalization

Don't normalize unicode characters in filenames during the sync routine.

Sometimes, an operating system will store filenames containing unicode parts in their decomposed form (particularly macOS). Some cloud storage systems will then recompose the unicode, resulting in duplicate files if the data is ever copied back to a local filesystem.

Using this flag will disable that functionality, treating each unicode character as unique. For example, by default é and é will be normalized into the same character. With `--no-unicode-normalization` they will be treated as unique characters.

--no-update-modtime

When using this flag, rclone won't update modification times of remote files if they are incorrect as it would normally.

This can be used if the remote is being synced with another tool also (e.g. the Google Drive client).

--order-by string

The `--order-by` flag controls the order in which files in the backlog are processed in `rclone sync`, `rclone copy` and `rclone move`.

The order by string is constructed like this. The first part describes what aspect is being measured:

- `size` - order by the size of the files
- `name` - order by the full path of the files
- `modtime` - order by the modification date of the files

This can have a modifier appended with a comma:

- `ascending` or `asc` - order so that the smallest (or oldest) is processed first
- `descending` or `desc` - order so that the largest (or newest) is processed first
- `mixed` - order so that the smallest is processed first for some threads and the largest for others

If the modifier is `mixed` then it can have an optional percentage (which defaults to `50`), e.g. `size,mixed,25` which means that 25% of the threads should be taking the smallest items and 75% the largest. The threads which take the smallest first will always take the smallest first and likewise the largest first threads. The `mixed` mode can be useful to minimise the transfer time when you are transferring a mixture of large and small files - the large files are guaranteed upload threads and bandwidth and the small files will be processed continuously.

If no modifier is supplied then the order is `ascending`.

For example

- `--order-by size,desc` - send the largest files first
- `--order-by modtime,ascending` - send the oldest files first
- `--order-by name` - send the files with alphabetically by path first

If the `--order-by` flag is not supplied or it is supplied with an empty string then the default ordering will be used which is as scanned. With `--checkers 1` this is mostly alphabetical, however with the default `--checkers 8` it is somewhat random.

Limitations

The `--order-by` flag does not do a separate pass over the data. This means that it may transfer some files out of the order specified if

- there are no files in the backlog or the source has not been fully scanned yet
- there are more than --max-backlog files in the backlog

Rclone will do its best to transfer the best file it has so in practice this should not cause a problem. Think of `--order-by` as being more of a best efforts flag rather than a perfect ordering.

If you want perfect ordering then you will need to specify --check-first which will find all the files which need transferring first before transferring any.

--password-command SpaceSepList

This flag supplies a program which should supply the config password when run. This is an alternative to rclone prompting for the password or setting the `RCLONE_CONFIG_PASS` variable.

The argument to this should be a command with a space separated list of arguments. If one of the arguments has a space in then enclose it in `"`, if you want a literal `"` in an argument then enclose the argument in `"` and double the `"`. See [CSV encoding](https://godoc.org/encoding/csv) for more info.

Eg

```
--password-command echo hello
--password-command echo "hello with space"
--password-command echo "hello with ""quotes"" and space"
```

See the [Configuration Encryption](#configuration-encryption) for more info.

See a [Windows PowerShell example on the Wiki](https://github.com/rclone/rclone/wiki/Windows-Powershell-use-rclone-password-command-for-Config-file-password).

-P, --progress

This flag makes rclone update the stats in a static block in the terminal providing a realtime overview of the transfer.

Any log messages will scroll above the static block. Log messages will push the static block down to the bottom of the terminal where it will stay.

Normally this is updated every 500mS but this period can be overridden with the `--stats` flag.

This can be used with the `--stats-one-line` flag for a simpler display.

Note: On Windows until [this bug](https://github.com/Azure/go-ansiterm/issues/26) is fixed all non-ASCII characters will be replaced with `.` when `--progress` is in use.

--progress-terminal-title

This flag, when used with `-P/--progress`, will print the string `ETA: %s` to the terminal title.

-q, --quiet

This flag will limit rclone's output to error messages only.

--refresh-times

The `--refresh-times` flag can be used to update modification times of existing files when they are out of sync on backends which don't support hashes.

This is useful if you uploaded files with the incorrect timestamps and you now wish to correct them.

This flag is **only** useful for destinations which don't support hashes (e.g. `crypt`).

This can be used any of the sync commands `sync`, `copy` or `move`.

To use this flag you will need to be doing a modification time sync (so not using `--size-only` or `--checksum`). The flag will have no effect when using `--size-only` or `--

`checksum`.

If this flag is used when rclone comes to upload a file it will check to see if there is an existing file on the destination. If this file matches the source with size (and checksum if available) but has a differing timestamp then instead of re-uploading it, rclone will update the timestamp on the destination file. If the checksum does not match rclone will upload the new file. If the checksum is absent (e.g. on a `crypt` backend) then rclone will update the timestamp.

Note that some remotes can't set the modification time without re-uploading the file so this flag is less useful on them.

Normally if you are doing a modification time sync rclone will update modification times without `--refresh-times` provided that the remote supports checksums **and** the checksums match on the file. However if the checksums are absent then rclone will upload the file rather than setting the timestamp as this is the safe behaviour.

--retries int

Retry the entire sync if it fails this many times it fails (default 3).

Some remotes can be unreliable and a few retries help pick up the files which didn't get transferred because of errors.

Disable retries with `--retries 1`.

--retries-sleep=TIME

This sets the interval between each retry specified by `--retries`

The default is `0`. Use `0` to disable.

--server-side-across-configs

Allow server-side operations (e.g. copy or move) to work across different configurations.

This can be useful if you wish to do a server-side copy or move between two remotes which use the same backend but are configured differently.

Note that this isn't enabled by default because it isn't easy for rclone to tell if it will work between any two configurations.

--size-only

Normally rclone will look at modification time and size of files to see if they are equal. If you set this flag then rclone will check only the size.

This can be useful transferring files from Dropbox which have been modified by the desktop sync client which doesn't set checksums of modification times in the same way as rclone.

--stats=TIME

Commands which transfer data (`sync`, `copy`, `copyto`, `move`, `moveto`) will print data transfer stats at regular intervals to show their progress.

This sets the interval.

The default is `1m`. Use `0` to disable.

If you set the stats interval then all commands can show stats. This can be useful when running other commands, `check` or `mount` for example.

Stats are logged at `INFO` level by default which means they won't show at default log level `NOTICE`. Use `--stats-log-level NOTICE` or `-v` to make them show. See the Logging section for more info on log levels.

Note that on macOS you can send a SIGINFO (which is normally ctrl-T in the terminal) to make the stats print immediately.

--stats-file-name-length integer

By default, the `--stats` output will truncate file names and paths longer than 40 characters. This is equivalent to providing `--stats-file-name-length 40`. Use `--stats-file-name-length 0` to disable any truncation of file names printed by stats.

--stats-log-level string

Log level to show `--stats` output at. This can be `DEBUG`, `INFO`, `NOTICE`, or `ERROR`. The default is `INFO`. This means at the default level of logging which is `NOTICE` the stats won't show - if you want them to then use `--stats-log-level NOTICE`. See the Logging section for more info on log levels.

--stats-one-line

When this is specified, rclone condenses the stats into a single line showing the most important stats only.

--stats-one-line-date

When this is specified, rclone enables the single-line stats and prepends the display with a date string. The default is `2006/01/02 15:04:05 -`

--stats-one-line-date-format

When this is specified, rclone enables the single-line stats and prepends the display with a user-supplied date string. The date string MUST be enclosed in quotes. Follow golang specs for date formatting syntax.

--stats-unit=bits|bytes

By default, data transfer rates will be printed in bytes per second.

This option allows the data rate to be printed in bits per second.

Data transfer volume will still be reported in bytes.

The rate is reported as a binary unit, not SI unit. So 1 Mbit/s equals 1,048,576 bit/s and not 1,000,000 bit/s.

The default is `bytes`.

--suffix=SUFFIX

When using `sync`, `copy` or `move` any files which would have been overwritten or deleted will have the suffix added to them. If there is a file with the same path (after the suffix has been added), then it will be overwritten.

The remote in use must support server-side move or copy and you must use the same remote as the destination of the sync.

This is for use with files to add the suffix in the current directory or with `--backup-dir`. See `--backup-dir` for more info.

For example

```
rclone copy --interactive /path/to/local/file remote:current --suffix .bak
```

will copy `/path/to/local` to `remote:current`, but for any files which would have been updated or deleted have .bak added.

If using `rclone sync` with `--suffix` and without `--backup-dir` then it is recommended to put a filter rule in excluding the suffix otherwise the `sync` will delete the backup files.

```
rclone sync --interactive /path/to/local/file remote:current --suffix .bak --exclude
"*.bak"
```

--suffix-keep-extension

When using `--suffix`, setting this causes rclone put the SUFFIX before the extension of the files that it backs up rather than after.

So let's say we had `--suffix -2019-01-01`, without the flag `file.txt` would be backed up to `file.txt-2019-01-01` and with the flag it would be backed up to `file-2019-01-01.txt`. This can be helpful to make sure the suffixed files can still be opened.

If a file has two (or more) extensions and the second (or subsequent) extension is recognised as a valid mime type, then the suffix will go before that extension. So `file.tar.gz` would be backed up to `file-2019-01-01.tar.gz` whereas `file.badextension.gz` would be backed up to `file.badextension-2019-01-01.gz`.

--syslog

On capable OSes (not Windows or Plan9) send all log output to syslog.

This can be useful for running rclone in a script or `rclone mount`.

--syslog-facility string

If using `--syslog` this sets the syslog facility (e.g. `KERN`, `USER`). See `man syslog` for a list of possible facilities. The default facility is `DAEMON`.

--temp-dir=DIR

Specify the directory rclone will use for temporary files, to override the default. Make sure the directory exists and have accessible permissions.

By default the operating system's temp directory will be used:

- On Unix systems, `$TMPDIR` if non-empty, else `/tmp`.
- On Windows, the first non-empty value from `%TMP%`, `%TEMP%`, `%USERPROFILE%`, or the Windows directory.

When overriding the default with this option, the specified path will be set as value of environment variable `TMPDIR` on Unix systems and `TMP` and `TEMP` on Windows.

You can use the config paths command to see the current value.

--tpslimit float

Limit transactions per second to this number. Default is 0 which is used to mean unlimited transactions per second.

A transaction is roughly defined as an API call; its exact meaning will depend on the backend. For HTTP based backends it is an HTTP PUT/GET/POST/etc and its response. For FTP/SFTP it is a round trip transaction over TCP.

For example, to limit rclone to 10 transactions per second use `--tpslimit 10`, or to 1 transaction every 2 seconds use `--tpslimit 0.5`.

Use this when the number of transactions per second from rclone is causing a problem with the cloud storage provider (e.g. getting you banned or rate limited).

This can be very useful for `rclone mount` to control the behaviour of applications using it.

This limit applies to all HTTP based backends and to the FTP and SFTP backends. It does not apply to the local backend or the Storj backend.

See also `--tpslimit-burst`.

--tpslimit-burst int

Max burst of transactions for `--tpslimit` (default `1`).

Normally `--tpslimit` will do exactly the number of transaction per second specified. However if you supply `--tps-burst` then rclone can save up some transactions from when it was idle giving a burst of up to the parameter supplied.

For example if you provide `--tpslimit-burst 10` then if rclone has been idle for more than 10*`--tpslimit` then it can do 10 transactions very quickly before they are limited again.

This may be used to increase performance of `--tpslimit` without changing the long term average number of transactions per second.

--track-renames

By default, rclone doesn't keep track of renamed files, so if you rename a file locally then sync it to a remote, rclone will delete the old file on the remote and upload a new copy.

An rclone sync with `--track-renames` runs like a normal sync, but keeps track of objects which exist in the destination but not in the source (which would normally be deleted), and which objects exist in the source but not the destination (which would normally be transferred). These objects are then candidates for renaming.

After the sync, rclone matches up the source only and destination only objects using the `--track-renames-strategy` specified and either renames the destination object or transfers the source and deletes the destination object. `--track-renames` is stateless like all of rclone's syncs.

To use this flag the destination must support server-side copy or server-side move, and to use a hash based `--track-renames-strategy` (the default) the source and the destination must have a compatible hash.

If the destination does not support server-side copy or move, rclone will fall back to the default behaviour and log an error level message to the console.

Encrypted destinations are not currently supported by `--track-renames` if `--track-renames-strategy` includes `hash`.

Note that `--track-renames` is incompatible with `--no-traverse` and that it uses extra memory to keep track of all the rename candidates.

Note also that `--track-renames` is incompatible with `--delete-before` and will select `--delete-after` instead of `--delete-during`.

--track-renames-strategy (hash,modtime,leaf,size)

This option changes the file matching criteria for `--track-renames`.

The matching is controlled by a comma separated selection of these tokens:

- `modtime` - the modification time of the file - not supported on all backends
- `hash` - the hash of the file contents - not supported on all backends
- `leaf` - the name of the file not including its directory name
- `size` - the size of the file (this is always enabled)

The default option is `hash`.

Using `--track-renames-strategy modtime,leaf` would match files based on modification time, the leaf of the file name and the size only.

Using `--track-renames-strategy modtime` or `leaf` can enable `--track-renames` support for encrypted destinations.

Note that the `hash` strategy is not supported with encrypted destinations.

--delete-(before,during,after)

This option allows you to specify when files on your destination are deleted when you sync folders.

Specifying the value `--delete-before` will delete all files present on the destination, but not on the source *before* starting the transfer of any new or updated files. This uses two passes through the file systems, one for the deletions and one for the copies.

Specifying `--delete-during` will delete files while checking and uploading files. This is the fastest option and uses the least memory.

Specifying `--delete-after` (the default value) will delay deletion of files until all new/updated files have been successfully transferred. The files to be deleted are collected in the copy pass then deleted after the copy pass has completed successfully. The files to be deleted are held in memory so this mode may use more memory. This is the safest mode as it will only delete files if there have been no errors subsequent to that. If there have been errors before the deletions start then you will get the message `not deleting files as there were IO errors`.

--fast-list

When doing anything which involves a directory listing (e.g. `sync`, `copy`, `ls` - in fact nearly every command), rclone normally lists a directory and processes it before using more directory lists to process any subdirectories. This can be parallelised and works very quickly using the least amount of memory.

However, some remotes have a way of listing all files beneath a directory in one (or a small number) of transactions. These tend to be the bucket-based remotes (e.g. S3, B2, GCS, Swift).

If you use the `--fast-list` flag then rclone will use this method for listing directories. This will have the following consequences for the listing:

- It **will** use fewer transactions (important if you pay for them)
- It **will** use more memory. Rclone has to load the whole listing into memory.
- It *may* be faster because it uses fewer transactions
- It *may* be slower because it can't be parallelized

rclone should always give identical results with and without `--fast-list`.

If you pay for transactions and can fit your entire sync listing into memory then `--fast-list` is recommended. If you have a very big sync to do then don't use `--fast-list` otherwise you will run out of memory.

If you use `--fast-list` on a remote which doesn't support it, then rclone will just ignore it.

--timeout=TIME

This sets the IO idle timeout. If a transfer has started but then becomes idle for this long it is considered broken and disconnected.

The default is `5m`. Set to `0` to disable.

--transfers=N

The number of file transfers to run in parallel. It can sometimes be useful to set this to a smaller number if the remote is giving a lot of timeouts or bigger if you have lots of bandwidth and a fast remote.

The default is to run 4 file transfers in parallel.

Look at --multi-thread-streams if you would like to control single file transfers.

-u, --update

This forces rclone to skip any files which exist on the destination and have a modified time that is newer than the source file.

This can be useful in avoiding needless transfers when transferring to a remote which doesn't support modification times directly (or when using `--use-server-modtime` to avoid extra API calls) as it is more accurate than a `--size-only` check and faster than using `--checksum`. On such remotes (or when using `--use-server-modtime`) the time checked will be the uploaded time.

If an existing destination file has a modification time older than the source file's, it will be updated if the sizes are different. If the sizes are the same, it will be updated if the checksum is different or not available.

If an existing destination file has a modification time equal (within the computed modify window) to the source file's, it will be updated if the sizes are different. The checksum will not be checked in this case unless the `--checksum` flag is provided.

In all other cases the file will not be updated.

Consider using the `--modify-window` flag to compensate for time skews between the source and the backend, for backends that do not support mod times, and instead use uploaded times. However, if the backend does not support checksums, note that syncing or copying within the time skew window may still result in additional transfers for safety.

--use-mmap

If this flag is set then rclone will use anonymous memory allocated by mmap on Unix based platforms and VirtualAlloc on Windows for its transfer buffers (size controlled by `--buffer-size`). Memory allocated like this does not go on the Go heap and can be returned to the OS immediately when it is finished with.

If this flag is not set then rclone will allocate and free the buffers using the Go memory allocator which may use more memory as memory pages are returned less aggressively to the OS.

It is possible this does not work well on all platforms so it is disabled by default; in the future it may be enabled by default.

--use-server-modtime

Some object-store backends (e.g, Swift, S3) do not preserve file modification times (modtime). On these backends, rclone stores the original modtime as additional metadata on the object. By default it will make an API call to retrieve the metadata when the modtime is needed by an operation.

Use this flag to disable the extra API call and rely instead on the server's modified time. In cases such as a local to remote sync using `--update`, knowing the local file is newer than the time it was last uploaded to the remote is sufficient. In those cases, this flag can speed up the process and reduce the number of API calls necessary.

Using this flag on a sync operation without also using `--update` would cause all files modified at any time other than the last upload time to be uploaded again, which is probably not what you want.

-v, -vv, --verbose

With `-v` rclone will tell you about each file that is transferred and a small number of significant events.

With `-vv` rclone will become very verbose telling you about every file it considers and transfers. Please send bug reports with a log with this setting.

When setting verbosity as an environment variable, use `RCLONE_VERBOSE=1` or `RCLONE_VERBOSE=2` for `-v` and `-vv` respectively.

-V, --version

Prints the version number

SSL/TLS options

The outgoing SSL/TLS connections rclone makes can be controlled with these options. For example this can be very useful with the HTTP or WebDAV backends. Rclone HTTP servers have their own set of configuration for SSL/TLS which you can find in their documentation.

--ca-cert stringArray

This loads the PEM encoded certificate authority certificates and uses it to verify the certificates of the servers rclone connects to.

If you have generated certificates signed with a local CA then you will need this flag to connect to servers using those certificates.

--client-cert string

This loads the PEM encoded client side certificate.

This is used for mutual TLS authentication.

The `--client-key` flag is required too when using this.

--client-key string

This loads the PEM encoded client side private key used for mutual TLS authentication. Used in conjunction with `--client-cert`.

--no-check-certificate=true/false

`--no-check-certificate` controls whether a client verifies the server's certificate chain and host name. If `--no-check-certificate` is true, TLS accepts any certificate presented by the server and any host name in that certificate. In this mode, TLS is susceptible to man-in-the-middle attacks.

This option defaults to `false`.

This should be used only for testing.

Configuration Encryption

Your configuration file contains information for logging in to your cloud services. This means that you should keep your `rclone.conf` file in a secure location.

If you are in an environment where that isn't possible, you can add a password to your configuration. This means that you will have to supply the password every time you start rclone.

To add a password to your rclone configuration, execute `rclone config`.

```
>rclone config
Current remotes:

e) Edit existing remote
```

```
n) New remote
d) Delete remote
s) Set configuration password
q) Quit config
e/n/d/s/q>
```

Go into `s`, Set configuration password:

```
e/n/d/s/q> s
Your configuration is not encrypted.
If you add a password, you will protect your login information to cloud services.
a) Add Password
q) Quit to main menu
a/q> a
Enter NEW configuration password:
password:
Confirm NEW password:
password:
Password set
Your configuration is encrypted.
c) Change Password
u) Unencrypt configuration
q) Quit to main menu
c/u/q>
```

Your configuration is now encrypted, and every time you start rclone you will have to supply the password. See below for details. In the same menu, you can change the password or completely remove encryption from your configuration.

There is no way to recover the configuration if you lose your password.

rclone uses nacl secretbox which in turn uses XSalsa20 and Poly1305 to encrypt and authenticate your configuration with secret-key cryptography. The password is SHA-256 hashed, which produces the key for secretbox. The hashed password is not stored.

While this provides very good security, we do not recommend storing your encrypted rclone configuration in public if it contains sensitive information, maybe except if you use a very strong password.

If it is safe in your environment, you can set the `RCLONE_CONFIG_PASS` environment variable to contain your password, in which case it will be used for decrypting the configuration.

You can set this for a session from a script. For unix like systems save this to a file called `set-rclone-password`:

```
#!/bin/echo Source this file don't run it

read -s RCLONE_CONFIG_PASS
export RCLONE_CONFIG_PASS
```

Then source the file when you want to use it. From the shell you would do `source set-rclone-password`. It will then ask you for the password and set it in the environment variable.

An alternate means of supplying the password is to provide a script which will retrieve the password and print on standard output. This script should have a fully specified path name and not rely on any environment variables. The script is supplied either via `--password-command="..."` command line argument or via the `RCLONE_PASSWORD_COMMAND` environment variable.

One useful example of this is using the `passwordstore` application to retrieve the password:

```
export RCLONE_PASSWORD_COMMAND="pass rclone/config"
```

If the `passwordstore` password manager holds the password for the rclone configuration, using the script method means the password is primarily protected by the `passwordstore` system, and is never embedded in the clear in scripts, nor available for examination using the standard commands available. It is quite possible with long running rclone sessions for copies of passwords to be innocently captured in log files or terminal scroll buffers, etc. Using the script method of supplying the password enhances the security of the config password considerably.

If you are running rclone inside a script, unless you are using the `--password-command` method, you might want to disable password prompts. To do that, pass the parameter `--ask-password=false` to rclone. This will make rclone fail instead of asking for a password if `RCLONE_CONFIG_PASS` doesn't contain a valid password, and `--password-command` has not been supplied.

Whenever running commands that may be affected by options in a configuration file, rclone will look for an existing file according to the rules described [above](), and load any it finds. If an encrypted file is found, this includes decrypting it, with the possible consequence of a password prompt. When executing a command line that you know are not actually using anything from such a configuration file, you can avoid it being loaded by overriding the location, e.g. with one of the documented special values for memory-only configuration. Since only backend options can be stored in configuration files, this is normally unnecessary for commands that do not operate on backends, e.g. `genautocomplete`. However, it will be relevant for commands that do operate on backends in general, but are used without referencing a stored remote, e.g. listing local filesystem paths, or [connection strings](): `rclone --config="" ls .`

Developer options

These options are useful when developing or debugging rclone. There are also some more remote specific options which aren't documented here which are used for testing. These start with remote name e.g. `--drive-test-option` - see the docs for the remote in question.

--cpuprofile=FILE

Write CPU profile to file. This can be analysed with `go tool pprof`.

--dump flag,flag,flag

The `--dump` flag takes a comma separated list of flags to dump info about.

Note that some headers including `Accept-Encoding` as shown may not be correct in the request and the response may not show `Content-Encoding` if the go standard libraries auto gzip encoding was in effect. In this case the body of the request will be gunzipped before showing it.

The available flags are:

--dump headers

Dump HTTP headers with `Authorization:` lines removed. May still contain sensitive info. Can be very verbose. Useful for debugging only.

Use `--dump auth` if you do want the `Authorization:` headers.

--dump bodies

Dump HTTP headers and bodies - may contain sensitive info. Can be very verbose. Useful for debugging only.

Note that the bodies are buffered in memory so don't use this for enormous files.

--dump requests

Like `--dump bodies` but dumps the request bodies and the response headers. Useful for debugging download problems.

--dump responses

Like `--dump bodies` but dumps the response bodies and the request headers. Useful for debugging upload problems.

--dump auth

Dump HTTP headers - will contain sensitive info such as `Authorization:` headers - use `--dump headers` to dump without `Authorization:` headers. Can be very verbose. Useful for debugging only.

--dump filters

Dump the filters to the output. Useful to see exactly what include and exclude options are filtering on.

--dump goroutines

This dumps a list of the running go-routines at the end of the command to standard output.

--dump openfiles

This dumps a list of the open files at the end of the command. It uses the `lsof` command to do that so you'll need that installed to use it.

--memprofile=FILE

Write memory profile to file. This can be analysed with `go tool pprof`.

Filtering

For the filtering options

- `--delete-excluded`
- `--filter`

- `--filter-from`
- `--exclude`
- `--exclude-from`
- `--exclude-if-present`
- `--include`
- `--include-from`
- `--files-from`
- `--files-from-raw`
- `--min-size`
- `--max-size`
- `--min-age`
- `--max-age`
- `--dump filters`
- `--metadata-include`
- `--metadata-include-from`
- `--metadata-exclude`
- `--metadata-exclude-from`
- `--metadata-filter`
- `--metadata-filter-from`

See the filtering section.

Remote control

For the remote control options and for instructions on how to remote control rclone

- `--rc`
- and anything starting with `--rc-`

See the remote control section.

Logging

rclone has 4 levels of logging, `ERROR`, `NOTICE`, `INFO` and `DEBUG`.

By default, rclone logs to standard error. This means you can redirect standard error and still see the normal output of rclone commands (e.g. `rclone ls`).

By default, rclone will produce `Error` and `Notice` level messages.

If you use the `-q` flag, rclone will only produce `Error` messages.

If you use the `-v` flag, rclone will produce `Error`, `Notice` and `Info` messages.

If you use the `-vv` flag, rclone will produce `Error`, `Notice`, `Info` and `Debug` messages.

You can also control the log levels with the `--log-level` flag.

If you use the `--log-file=FILE` option, rclone will redirect `Error`, `Info` and `Debug` messages along with standard error to FILE.

If you use the `--syslog` flag then rclone will log to syslog and the `--syslog-facility` control which facility it uses.

Rclone prefixes all log messages with their level in capitals, e.g. INFO which makes it easy to grep the log file for different kinds of information.

Exit Code

If any errors occur during the command execution, rclone will exit with a non-zero exit code. This allows scripts to detect when rclone operations have failed.

During the startup phase, rclone will exit immediately if an error is detected in the configuration. There will always be a log message immediately before exiting.

When rclone is running it will accumulate errors as it goes along, and only exit with a non-zero exit code if (after retries) there were still failed transfers. For every error counted there will be a high priority log message (visible with `-q`) showing the message and which file caused the problem. A high priority message is also shown when starting a retry so the user can see that any previous error messages may not be valid after the retry. If rclone has done a retry it will log a high priority message if the retry was successful.

List of exit codes

- `0` - success
- `1` - Syntax or usage error
- `2` - Error not otherwise categorised
- `3` - Directory not found
- `4` - File not found
- `5` - Temporary error (one that more retries might fix) (Retry errors)
- `6` - Less serious errors (like 461 errors from dropbox) (NoRetry errors)
- `7` - Fatal error (one that more retries won't fix, like account suspended) (Fatal errors)
- `8` - Transfer exceeded - limit set by --max-transfer reached
- `9` - Operation successful, but no files transferred

Environment Variables

Rclone can be configured entirely using environment variables. These can be used to set defaults for options or config file entries.

Options

Every option in rclone can have its default set by environment variable.

To find the name of the environment variable, first, take the long option name, strip the leading `--`, change `-` to `_`, make upper case and prepend `RCLONE_`.

For example, to always set `--stats 5s`, set the environment variable `RCLONE_STATS=5s`. If you set stats on the command line this will override the environment variable setting.

Or to always use the trash in drive `--drive-use-trash`, set `RCLONE_DRIVE_USE_TRASH=true`.

Verbosity is slightly different, the environment variable equivalent of `--verbose` or `-v` is `RCLONE_VERBOSE=1`, or for `-vv`, `RCLONE_VERBOSE=2`.

The same parser is used for the options and the environment variables so they take exactly the same form.

The options set by environment variables can be seen with the `-vv` flag, e.g. `rclone version -vv`.

Config file

You can set defaults for values in the config file on an individual remote basis. The names of the config items are documented in the page for each backend.

To find the name of the environment variable, you need to set, take `RCLONE_CONFIG_` + name of remote + `_` + name of config file option and make it all uppercase.

For example, to configure an S3 remote named `mys3:` without a config file (using unix ways of setting environment variables):

```
$ export RCLONE_CONFIG_MYS3_TYPE=s3
$ export RCLONE_CONFIG_MYS3_ACCESS_KEY_ID=XXX
$ export RCLONE_CONFIG_MYS3_SECRET_ACCESS_KEY=XXX
$ rclone lsd mys3:
          -1 2016-09-21 12:54:21        -1 my-bucket
$ rclone listremotes | grep mys3
mys3:
```

Note that if you want to create a remote using environment variables you must create the `..._TYPE` variable as above.

Note that the name of a remote created using environment variable is case insensitive, in contrast to regular remotes stored in config file as documented above. You must write the name in uppercase in the environment variable, but as seen from example above it will be listed and can be accessed in lowercase, while you can also refer to the same remote in uppercase:

```
$ rclone lsd mys3:
          -1 2016-09-21 12:54:21        -1 my-bucket
$ rclone lsd MYS3:
          -1 2016-09-21 12:54:21        -1 my-bucket
```

Note that you can only set the options of the immediate backend, so RCLONE_CONFIG_MYS3CRYPT_ACCESS_KEY_ID has no effect, if myS3Crypt is a crypt remote based on an S3 remote. However RCLONE_S3_ACCESS_KEY_ID will set the access key of all remotes using S3, including myS3Crypt.

Note also that now rclone has connection strings, it is probably easier to use those instead which makes the above example

```
rclone lsd :s3,access_key_id=XXX,secret_access_key=XXX:
```

Precedence

The various different methods of backend configuration are read in this order and the first one with a value is used.

- Parameters in connection strings, e.g. `myRemote,skip_links:`
- Flag values as supplied on the command line, e.g. `--skip-links`
- Remote specific environment vars, e.g. `RCLONE_CONFIG_MYREMOTE_SKIP_LINKS` (see above).

- Backend-specific environment vars, e.g. `RCLONE_LOCAL_SKIP_LINKS`.
- Backend generic environment vars, e.g. `RCLONE_SKIP_LINKS`.
- Config file, e.g. `skip_links = true`.
- Default values, e.g. `false` - these can't be changed.

So if both `--skip-links` is supplied on the command line and an environment variable `RCLONE_LOCAL_SKIP_LINKS` is set, the command line flag will take preference.

The backend configurations set by environment variables can be seen with the `-vv` flag, e.g. `rclone about myRemote: -vv`.

For non backend configuration the order is as follows:

- Flag values as supplied on the command line, e.g. `--stats 5s`.
- Environment vars, e.g. `RCLONE_STATS=5s`.
- Default values, e.g. `1m` - these can't be changed.

Other environment variables

- `RCLONE_CONFIG_PASS` set to contain your config file password (see Configuration Encryption section)
- `HTTP_PROXY`, `HTTPS_PROXY` and `NO_PROXY` (or the lowercase versions thereof).
  - `HTTPS_PROXY` takes precedence over `HTTP_PROXY` for https requests.
  - The environment values may be either a complete URL or a "host[:port]" for, in which case the "http" scheme is assumed.
- `USER` and `LOGNAME` values are used as fallbacks for current username. The primary method for looking up username is OS-specific: Windows API on Windows, real user ID in /etc/passwd on Unix systems. In the documentation the current username is simply referred to as `$USER`.
- `RCLONE_CONFIG_DIR` - rclone **sets** this variable for use in config files and sub processes to point to the directory holding the config file.

The options set by environment variables can be seen with the `-vv` and `--log-level=DEBUG` flags, e.g. `rclone version -vv`.