Chunker (BETA)

The `chunker` overlay transparently splits large files into smaller chunks during upload to wrapped remote and transparently assembles them back when the file is downloaded. This allows to effectively overcome size limits imposed by storage providers.

To use it, first set up the underlying remote following the configuration instructions for that remote. You can also use a local pathname instead of a remote.

First check your chosen remote is working - we'll call it `remote:path` here. Note that anything inside `remote:path` will be chunked and anything outside won't. This means that if you are using a bucket based remote (e.g. S3, B2, swift) then you should probably put the bucket in the remote `s3:bucket`.

Now configure `chunker` using `rclone config`. We will call this one `overlay` to separate it from the `remote` itself.

```
No remotes found - make a new one
n) New remote
s) Set configuration password
q) Quit config
n/s/q> n
name> overlay
Type of storage to configure.
Choose a number from below, or type in your own value
[snip]
XX / Transparently chunk/split large files
   \ "chunker"
[snip]
Storage> chunker
Remote to chunk/unchunk.
Normally should contain a ':' and a path, e.g. "myremote:path/to/dir",
"myremote:bucket" or maybe "myremote:" (not recommended).
Enter a string value. Press Enter for the default ("").
remote> remote:path
Files larger than chunk size will be split in chunks.
Enter a size with suffix k,M,G,T. Press Enter for the default ("2G").
chunk_size> 100M
Choose how chunker handles hash sums. All modes but "none" require metadata.
Enter a string value. Press Enter for the default ("md5").
Choose a number from below, or type in your own value
 1 / Pass any hash supported by wrapped remote for non-chunked files, return nothing otherwis
   \ "none"
 2 / MD5 for composite files
   \ "md5"
 3 / SHA1 for composite files
   \ "sha1"
 4 / MD5 for all files
   \ "md5all"
 5 / SHA1 for all files
   \ "sha1all"
 6 / Copying a file to chunker will request MD5 from the source falling back to SHA1 if unsup
   \ "md5quick"
 7 / Similar to "md5quick" but prefers SHA1 over MD5
   \ "sha1quick"
hash_type> md5
Edit advanced config? (y/n)
y) Yes
n) No
y/n> n
Remote config
--------------------
[overlay]
type = chunker
remote = remote:bucket
chunk_size = 100M
hash_type = md5
--------------------
```

```
y) Yes this is OK
e) Edit this remote
d) Delete this remote
y/e/d> y
```

Specifying the remote

In normal use, make sure the remote has a `:` in. If you specify the remote without a `:` then rclone will use a local directory of that name. So if you use a remote of `/path/to/secret/files` then rclone will chunk stuff in that directory. If you use a remote of `name` then rclone will put files in a directory called `name` in the current directory.

Chunking

When rclone starts a file upload, chunker checks the file size. If it doesn't exceed the configured chunk size, chunker will just pass the file to the wrapped remote. If a file is large, chunker will transparently cut data in pieces with temporary names and stream them one by one, on the fly. Each data chunk will contain the specified number of bytes, except for the last one which may have less data. If file size is unknown in advance (this is called a streaming upload), chunker will internally create a temporary copy, record its size and repeat the above process.

When upload completes, temporary chunk files are finally renamed. This scheme guarantees that operations can be run in parallel and look from outside as atomic. A similar method with hidden temporary chunks is used for other operations (copy/move/rename, etc.). If an operation fails, hidden chunks are normally destroyed, and the target composite file stays intact.

When a composite file download is requested, chunker transparently assembles it by concatenating data chunks in order. As the split is trivial one could even manually concatenate data chunks together to obtain the original content.

When the `list` rclone command scans a directory on wrapped remote, the potential chunk files are accounted for, grouped and assembled into composite directory entries. Any temporary chunks are hidden.

List and other commands can sometimes come across composite files with missing or invalid chunks, e.g. shadowed by like-named directory or another file. This usually means that wrapped file system has been directly tampered with or damaged. If chunker detects a missing chunk it will by default print warning, skip the whole incomplete group of chunks but proceed with current command. You can set the `--chunker-fail-hard` flag to have commands abort with error message in such cases.

Chunk names

The default chunk name format is `*.rclone_chunk.###`, hence by default chunk names are `BIG_FILE_NAME.rclone_chunk.001`,`BIG_FILE_NAME.rclone_chunk.002` etc. You can configure another name format using the `name_format` configuration file option. The format uses asterisk `*` as a placeholder for the base file name and one or more consecutive hash characters `#` as a placeholder for sequential chunk number. There must be one and only one asterisk. The number of consecutive hash characters defines the minimum length of a string representing a chunk number. If decimal chunk number has less digits than the number of hashes, it is left-padded by zeros. If the decimal string is longer, it is left intact. By default numbering starts from 1 but there is another option that allows user to start from 0, e.g. for compatibility with legacy software.

For example, if name format is `big_*-##.part` and original file name is `data.txt` and numbering starts from 0, then the first chunk will be named `big_data.txt-00.part`, the 99th chunk will be `big_data.txt-98.part` and the 302nd chunk will become `big_data.txt-301.part`.

Note that `list` assembles composite directory entries only when chunk names match the configured format and treats non-conforming file names as normal non-chunked files.

Metadata

Besides data chunks chunker will by default create metadata object for a composite file. The object is named after the original file. Chunker allows user to disable metadata completely (the `none` format). Note that metadata is normally not created for files smaller than the configured chunk size. This may change in future rclone releases.

Simple JSON metadata format

This is the default format. It supports hash sums and chunk validation for composite files. Meta objects carry the following fields:

- `ver` - version of format, currently `1`
- `size` - total size of composite file
- `nchunks` - number of data chunks in file
- `md5` - MD5 hashsum of composite file (if present)
- `sha1` - SHA1 hashsum (if present)

There is no field for composite file name as it's simply equal to the name of meta object on the wrapped remote. Please refer to respective sections for details on hashsums and modified time handling.

No metadata

You can disable meta objects by setting the meta format option to `none`. In this mode chunker will scan directory for all files that follow configured chunk name format, group them by detecting chunks with the same base name and show group names as virtual composite files. This method is more prone to missing chunk errors (especially missing last chunk) than format with metadata enabled.

Hashsums

Chunker supports hashsums only when a compatible metadata is present. Hence, if you choose metadata format of `none`, chunker will report hashsum as `UNSUPPORTED`.

Please note that by default metadata is stored only for composite files. If a file is smaller than configured chunk size, chunker will transparently redirect hash requests to wrapped remote, so support depends on that. You will see the empty string as a hashsum of requested type for small files if the wrapped remote doesn't support it.

Many storage backends support MD5 and SHA1 hash types, so does chunker. With chunker you can choose one or another but not both. MD5 is set by default as the most supported type. Since chunker keeps hashes for composite files and falls back to the wrapped remote hash for non-chunked ones, we advise you to choose the same hash type as supported by wrapped remote so that your file listings look coherent.

If your storage backend does not support MD5 or SHA1 but you need consistent file hashing, configure chunker with `md5all` or `sha1all`. These two modes guarantee given hash for all files. If wrapped remote doesn't support it, chunker will then add metadata to all files, even small. However, this can double the amount of small files in storage and incur additional service charges. You can even use chunker to force md5/sha1 support in any other remote at expense of sidecar meta objects by setting e.g. `chunk_type=sha1all` to force hashsums and `chunk_size=1P` to effectively disable chunking.

Normally, when a file is copied to chunker controlled remote, chunker will ask the file source for compatible file hash and revert to on-the-fly calculation if none is found. This involves some CPU overhead but provides a guarantee that given hashsum is available. Also, chunker will reject a server-side copy or move operation if source and destination hashsum types are different resulting in the extra network bandwidth, too. In some rare cases this may be undesired, so chunker provides two optional choices: `sha1quick` and `md5quick`. If the source does not support primary hash type and the quick mode is enabled, chunker will try to fall back to the secondary type. This will save CPU and bandwidth but can result in empty hashsums at destination. Beware of consequences: the `sync` command will revert (sometimes silently) to time/size comparison if compatible hashsums between source and target are not found.

Modified time

Chunker stores modification times using the wrapped remote so support depends on that. For a small non-chunked file the chunker overlay simply manipulates modification time of the wrapped remote file. For a composite file with metadata chunker will get and set modification time of the metadata object on the wrapped remote. If file is chunked but metadata format is `none` then chunker will use modification time of the first data chunk.

Migrations

The idiomatic way to migrate to a different chunk size, hash type or chunk naming scheme is to:

- Collect all your chunked files under a directory and have your chunker remote point to it.
- Create another directory (most probably on the same cloud storage) and configure a new remote with desired metadata format, hash type, chunk naming etc.
- Now run `rclone sync -i oldchunks: newchunks:` and all your data will be transparently converted in transfer. This may take some time, yet chunker will try server-side copy if possible.
- After checking data integrity you may remove configuration section of the old remote.

If rclone gets killed during a long operation on a big composite file, hidden temporary chunks may stay in the directory. They will not be shown by the `list` command but will eat up your account quota. Please note that the `deletefile` command deletes only active chunks of a file. As a workaround, you can use remote of the wrapped file system to see them. An easy way to get rid of hidden garbage is to copy littered directory somewhere using the chunker remote and purge the original directory. The `copy` command will copy only active chunks while the `purge` will remove everything including garbage.

Caveats and Limitations

Chunker requires wrapped remote to support server-side `move` (or `copy` + `delete`) operations, otherwise it will explicitly refuse to start. This is because it internally renames temporary chunk files to their final names when an operation completes successfully.

Chunker encodes chunk number in file name, so with default `name_format` setting it adds 17 characters. Also chunker adds 7 characters of temporary suffix during operations. Many file systems limit base file name without path by 255 characters. Using rclone's crypt remote as a base file system limits file name by 143 characters. Thus, maximum name length is 231 for most files and 119 for chunker-over-crypt. A user in need can change name format to e.g. `*.rcc##` and save 10 characters (provided at most 99 chunks per file).

Note that a move implemented using the copy-and-delete method may incur double charging with some cloud storage providers.

Chunker will not automatically rename existing chunks when you run `rclone config` on a live remote and change the chunk name format. Beware that in result of this some files which have been treated as chunks before the change can pop up in directory listings as normal files and vice versa. The same warning holds for the chunk size. If you desperately need to change critical chunking settings, you should run data migration as described above.

If wrapped remote is case insensitive, the chunker overlay will inherit that property (so you can't have a file called "Hello.doc" and "hello.doc" in the same directory).

Chunker included in rclone releases up to `v1.54` can sometimes fail to detect metadata produced by recent versions of rclone. We recommend users to keep rclone up-to-date to avoid data corruption.

Standard Options

Here are the standard options specific to chunker (Transparently chunk/split large files).

--chunker-remote

Remote to chunk/unchunk. Normally should contain a ':' and a path, e.g. "myremote:path/to/dir", "myremote:bucket" or maybe "myremote:" (not recommended).

- Config: remote
- Env Var: RCLONE_CHUNKER_REMOTE
- Type: string
- Default: ""

--chunker-chunk-size

Files larger than chunk size will be split in chunks.

- Config: chunk_size
- Env Var: RCLONE_CHUNKER_CHUNK_SIZE
- Type: SizeSuffix
- Default: 2G

--chunker-hash-type

Choose how chunker handles hash sums. All modes but "none" require metadata.

- Config: hash_type
- Env Var: RCLONE_CHUNKER_HASH_TYPE
- Type: string
- Default: "md5"
- Examples:
    - "none"
        - Pass any hash supported by wrapped remote for non-chunked files, return nothing otherwise
    - "md5"
        - MD5 for composite files
    - "sha1"
        - SHA1 for composite files
    - "md5all"
        - MD5 for all files
    - "sha1all"
        - SHA1 for all files
    - "md5quick"
        - Copying a file to chunker will request MD5 from the source falling back to SHA1 if unsupported
    - "sha1quick"
        - Similar to "md5quick" but prefers SHA1 over MD5

Advanced Options

Here are the advanced options specific to chunker (Transparently chunk/split large files).

--chunker-name-format

String format of chunk file names. The two placeholders are: base file name (*) and chunk number (#...). There must be one and only one asterisk and one or more consecutive hash characters. If chunk number has less digits than the number of hashes, it is left-padded by zeros. If there are more digits in the number, they are left as is. Possible chunk files are ignored if their name does not match given format.

- Config: name_format
- Env Var: RCLONE_CHUNKER_NAME_FORMAT
- Type: string
- Default: "*.rclone_chunk.###"

--chunker-start-from

Minimum valid chunk number. Usually 0 or 1. By default chunk numbers start from 1.

- Config: start_from
- Env Var: RCLONE_CHUNKER_START_FROM
- Type: int

- Default: 1

--chunker-meta-format

Format of the metadata object or "none". By default "simplejson". Metadata is a small JSON file named after the composite file.

- Config: meta_format
- Env Var: RCLONE_CHUNKER_META_FORMAT
- Type: string
- Default: "simplejson"
- Examples:
    - "none"
        - Do not use metadata files at all. Requires hash type "none".
    - "simplejson"
        - Simple JSON supports hash sums and chunk validation.
        - It has the following fields: ver, size, nchunks, md5, sha1.

--chunker-fail-hard

Choose how chunker should handle files with missing or invalid chunks.

- Config: fail_hard
- Env Var: RCLONE_CHUNKER_FAIL_HARD
- Type: bool
- Default: false
- Examples:
    - "true"
        - Report errors and abort current command.
    - "false"
        - Warn user, skip incomplete file and proceed.