
skicka

Utility for working with files and folders stored on Google Drive.

Note: skicka is not an official Google product!

Intro

skicka makes it easy to copy files to and from Google Drive and to work with files stored on Google Drive.

For example, `skicka upload ~/Pictures /Pictures` copies the entire contents of the local `~/Pictures` directory to a folder `Pictures` in Google Drive. If you then run `skicka download /Pictures ~/Pictures2`, then the contents of your `~/Pictures2` directory will match the contents of `~/Pictures`.

More generally, skicka makes it easy to list the files in Google Drive folders, compute the space used by Drive folders, and copy files between your computer to Google Drive. If you'd like to encrypt your files before uploading them, skicka supports AES-256 encryption.

What skicka is not

skicka is not a general solution for automatically synchronizing files stored in Google Drive across multiple machines. In particular, it doesn't have logic to reconcile concurrent changes to the same file on different machines.

Furthermore, although skicka has been robust in usage so far and has no known data corruption bugs, it should for now be treated as "alpha" software. Bug reports are welcome.

Getting Started

1. You must have a Go compiler installed.
2. Download and build skicka: `go get github.com/google/skicka`
3. Either copy the `skicka` executable in `$GOPATH/bin` to a directory in your `PATH`, or add `$GOPATH/bin` to your `PATH`.
4. Run `skicka init` to create a skeleton `~/ .skicka.config` file. Various options can be set in this file to control skicka's operation; see comments in the file for more information.
5. Authorize skicka to access your Google Drive files: run `skicka ls`, and skicka will attempt to open an authorization page in your web browser. Click the "Accept" button to authorize skicka. You only need to perform this step once.

-
- Alternatively, you can authorize skicka by running `skicka -no-browser-auth ls` and skicka will ask you to visit a URL: go to the URL in a browser, log into your Google account (if needed), and give permission for the application to access your Google Drive files. After you click ‘accept’, copy the code from your browser window to the terminal with the “Enter verification code” prompt from skicka.
6. After skicka is authorized, it will download and locally cache information about the files you have stored in Google Drive. This may take a while if you have many files on Drive; a progress indicator will try to keep you posted about how this is going. Once this data is stored locally, future runs of skicka will get going much more quickly.
 7. If you’re going store encrypted files in Google Drive, create an encryption key: set the environment variable `SKICKA_PASSPHRASE` to hold your passphrase and run `skicka genkey`. Copy the lines of text that are printed to the `[encryption]` section of your `~/.skicka.config` file.
 8. Try it out: run `skicka ls -l /`. A list of the files and folders in the root directory of your Google Drive should be printed.
 9. Join the mailing list!

Usage

For a general overview of skicka’s commands, run `skicka help`.

To copy a local directory hierarchy to Google Drive, use the `upload` command. As the upload progresses, skicka periodically reports how much data has been uploaded and how much time has elapsed.

```
1 % skicka upload ~/Pictures /Pictures
2 11 / 11 [=====] 100.00
  %
3 skicka: preparation time 1s, sync time 10s
4 skicka: updated 0 Drive files, 8 local files
5 skicka: 4.52 MiB read from disk, 0 B written to disk
6 skicka: 4.52 MiB uploaded (482 kiB/s), 0 B downloaded (0 B/s)
7 skicka: 32.76 MiB peak memory used
8 %
```

If you’d like to encrypt the files, create an encryption key as described above in “Getting Started” and then use the `-encrypt` command-line option. skicka looks for your encryption passphrase in the `SKICKA_PASSPHRASE` environment variable; thus, you might use:

```
1 % env SKICKA_PASSPHRASE=mySecretPassphrase skicka upload -encrypt ~/
  Pictures /EncryptedPictures
```

To download a directory hierarchy from Google Drive to your local disk, use `download`. As the download progresses, status is periodically reported:

```
1 % skicka download /Pictures/2014 ~/Pictures.copy/2014
2 10 / 10 [=====] 100.00
   %
3 skicka: preparation time 1s, sync time 6s
4 skicka: updated 0 Drive files, 10 local files
5 skicka: 0 B read from disk, 16.18 MiB written to disk
6 skicka: 0 B uploaded (0 B/s), 16.18 MiB downloaded (2.33 MiB/s)
7 skicka: 50.23 MiB peak memory used
8 %
```

The `ls` command can be used to list files and directories in Google Drive. For example, after uploading your `~/Pictures` directory, you might run:

```
1 % skicka ls /Pictures
2 2012
3 2013
4 2014
```

To see more detail, `ls -l` can be used:

```
1 % skicka ls -l /Pictures/2013
2 -rw-r--r--  2.62 MiB  Sun Mar 10 14:41:08 2013  IMG_1127.JPG
3 -rw-r--r--  2.63 MiB  Sun Mar 10 14:41:09 2013  IMG_1128.JPG
4 -rw-r--r--  2.32 MiB  Sun Mar 10 14:41:16 2013  IMG_1129.JPG
5 -rw-r--r--  2.47 MiB  Sun Mar 10 14:43:16 2013  IMG_1130.JPG
6 [...]
```

In addition to the filename, we see the file size, the file permissions, and the local modification time that the file on Google Drive is synced to. The file permissions are based on the permissions of the file when it was uploaded to Google Drive.

(For even more detail, `ls -ll` can be used, which also prints the MD5 checksums of the files on Google Drive.)

The contents of a single file can be downloaded using `cat`:

```
1 % skicka cat /Pictures/2013/IMG_1129.JPG > img.jpg
```

Google Drive folders can be created with `mkdir`. (As with the Unix command, the `-p` option can be specified to indicate that the intermediate directories in the path should be created).

```
1 % skicka mkdir /Pictures/2015
```

The amount of space used by the children of a given Google Drive folder can be reported by `du`:

```
1 % skicka du /Pictures
```

```
2 81.56 MiB /Pictures/2012
3 92.14 MiB /Pictures/2013
4 121.02 MiB /Pictures/2014
5 294.72 MiB /Pictures
```

You can remove a file with `rm`; by default, files are moved to the Trash in Google Drive:

```
1 % skicka rm /Pictures/2014/embarassing.jpg
```

If you'd like to delete the file immediately with no chance to recover it, use the `-s` command-line option to `rm`. To remove a folder and everything inside it, use `-r`.

Finally, there is a `fsck` command that checks the file system on Google Drive for problems and verifies that the local cache of file metadata is in-sync with the files stored on Drive.

FAQs

Can skicka work with Google Drive files that it didn't create itself?

Yes. The only limitations are that regular Google Drive files don't store the Unix permissions or the local modification of the original file when it was uploaded. Therefore, in this case `skicka download` uses 644 permissions for files it creates and 755 permissions for directories.

`skicka ls -l` indicates that a file has world-readable permissions on Google Drive; does this mean anyone can access it?

No. Those permissions are only used to set the local file permissions when the file is downloaded with `skicka download`. The access permissions for the files stored on Drive are handled with Drive's regular mechanisms.

How can I speed up uploads?

There's a fixed per-file overhead for each file uploaded to Google Drive that limits `skicka` to creating roughly five files a second; if files are relatively small, this overhead will be more of a limit than the time spent transferring the contents of the files.

If the uploaded files don't need to be accessed individually, creating a `tar` or `zip` archive of them before uploading may help in this case.

I occasionally see “operation timed out” or “broken pipe” errors when uploading; what’s going on?

A variety of transient errors can happen when using RESTful APIs like the Google Drive API. When these errors are encountered, skicka makes a number of attempts to retry the operation before giving up.

It may be that skicka should make more attempts before giving up or that there are better error handling strategies; one trade-off is that if there is a serious error (like the internet connection is lost), then it’s useful for the user to know this sooner rather than later.

If you do see these errors, re-run the operation you were performing; any files that weren’t transferred the first time should be taken care of with a second run.

Does skicka support rate-limited uploads and downloads?

Yes. By default, skicka doesn’t try to limit its bandwidth usage. However, if you add a line `bytes-per-second-limit=...` to the `[upload]` or `[download]` section of your `.skicka.config` file, you can specify a maximum number of bytes per second to transfer for uploads or downloads, respectively.

If this line isn’t present (or has a value of zero), then bandwidth won’t be limited.

Can an http proxy be used with skicka?

Yes—just set the `HTTP_PROXY` environment variable appropriately.

“skicka”?!?

Swedish, “to send”.

Implementation notes

How files are stored in Google Drive

When a directory hierarchy is uploaded, Google Drive file is created for each local file and a Google Drive folder is created for each local directory. skicka stores the time the local copy file was last modified in the “modifiedDate” Google Drive File resource. The Unix file permissions of the file or directory are stored in using a custom “Permissions” file property, stored as a string with the octal file permissions.

See the discussion of encryption below for details about how encrypted files are represented.

Synchronization algorithm

When deciding if a local file needs to be uploaded to Google Drive, skicka performs the following checks.

1. If there is no corresponding file on Drive, the local file will be uploaded.
2. Otherwise, if there is a corresponding file on Drive and the sizes of the files are different, the local file will be uploaded.
3. Otherwise, if the local file's modification time is after the modification time of the file the last time it was synced to Drive, then an MD5 checksum of the file contents is computed. If this checksum differs from the checksum of the file stored on Google Drive, the file will be uploaded. (Thus, if `touch` is run on a file to update its modification time but the file's contents aren't modified, skicka won't unnecessarily re-upload the file.)

Note that skicka trusts that file modification times are meaningful: if a file's contents are modified leaving the file size is unchanged and if the modification time is set to be in the past, then skicka won't compute an MD5 checksum and won't know that the file should be uploaded. To override this behavior, run `skicka upload` with the `-ignore-times` flag; if this flag is provided, then the MD5 checksum check in the third step will be applied regardless of the file modification time.

Note also that this algorithm is an algorithm to efficiently mirror the contents of a set of local files on Google Drive; it's not a general bidirectional synchronization algorithm. For example, if a file is modified both on Drive and on the local filesystem, a `skicka upload` run will clobber the file contents on Drive. In other words, the assumption is that the source directory hierarchy is by definition the canonical one and the destination directory's role is to perfectly reflect the source.

When downloading from Google Drive, skicka follows the same general approach: only files that don't yet exist, have different sizes, or different MD5 checksums from the corresponding local file will be downloaded. The `-ignore-times` option can also be used to bypass the file modification time check and to force a comparison of file contents to decide whether to download.

Encryption

If the `-encrypt` flag is provided to the `upload` command, skicka will encrypt the contents of each file before uploading it to Google Drive. Conversely, the `download` and `cat` commands transparently decrypt encrypted files when downloading them (if the encryption key and passphrase are available!)

Encryption requires both an encryption key and a passphrase; your passphrase should be stored in the `SKICKA_PASSPHRASE` environment variable. To generate an encryption key, run `skicka genkey`. For example:

```
1 % env SKICKA_PASSPHRASE=mySecretPassphrase skicka genkey
2 [...]
```

`skicka` will generate a few lines of output that are your encrypted encryption key; to save your key, edit your `~/ .skicka.config` file and add the printed text to the `[encryption]` section of the file. You can now upload and encrypt files:

```
1 % env SKICKA_PASSPHRASE=mySecretPassphrase skicka upload -encrypt ~/
   backups /backups
```

If you lose your encryption key or forget your passphrase, your encrypted data is lost forever. If you use encryption, please be very careful with this information.

`skicka` only encrypts file contents: it doesn't encrypt filenames or hide file sizes, for example. If this is problematic for your usage, you should probably use `tar` or `zip` to put the files you want to encrypt into a single file before uploading it.

Key generation and storage When `skicka genkey` is executed, an encryption key is generated as follows:

1. `skicka` generates a random 32-byte salt using Go's `rand.Reader`, which returns cryptographically secure pseudo-random numbers. The hex-encoded salt is printed out, and should be recorded in the `salt` field of the `[encryption]` section of the config file.
2. The user's passphrase, read from the `SKICKA_PASSPHRASE` environment variable, is run through the PBKDF2 key derivation function, using 65536 iterations and the SHA-256 hash function to derive a 64-byte hash.
3. The first 32 bytes of the hash are hex encoded and printed out; they should be copied to the `passphrase-hash` field of the `[encryption]` section of the config file. These bytes are later used only to validate that the user has provided the correct passphrase on subsequent runs of `skicka`.
4. A random 32-byte encryption key is generated (again with `rand.Reader`). This is the key that will actually be used for encryption and decryption of file contents.
5. A random 16-byte initialization vector is generated with `rand.Reader`. It is hex encoded and printed out, and should be copied to the `encrypted-key-iv` configuration file field.
6. The encryption key from #4 is encrypted using the initialization vector from #5, using the second 32 bytes of the hash computed in #2 as the encryption key. The result should be copied to the `encrypted-key` field of the config file.

Upon subsequent runs of skicka, the salt is loaded from the config file so that PBKDF2 can be used as in #2 above to hash the user's passphrase. If the first 32 bytes of the passphrase hash match the stored bytes, then the second 32 bytes of the hash and the stored IV are used to decrypt the encrypted encryption key.

Given the encryption key, when a file is to be encrypted before being uploaded to Google Drive, skicka uses the key along with a fresh 16-byte initialization vector for each file to encrypt the file using AES-256. The initialization vector is prepended to the file contents before upload. (Thus, encrypted files are 16 bytes larger on Google Drive than they are locally.)

The initialization vector is also stored hex-encoded as a Google Drive file Property with the name "IV". We store the initialization vector redundantly so that if one downloads the encrypted file contents, it's possible to decrypt the file using the file contents (and the key!) alone. Conversely, also having the IV available in a property makes it possible to encrypt the contents of a local version of a file without needing to download any of the contents of the corresponding file from Google Drive.