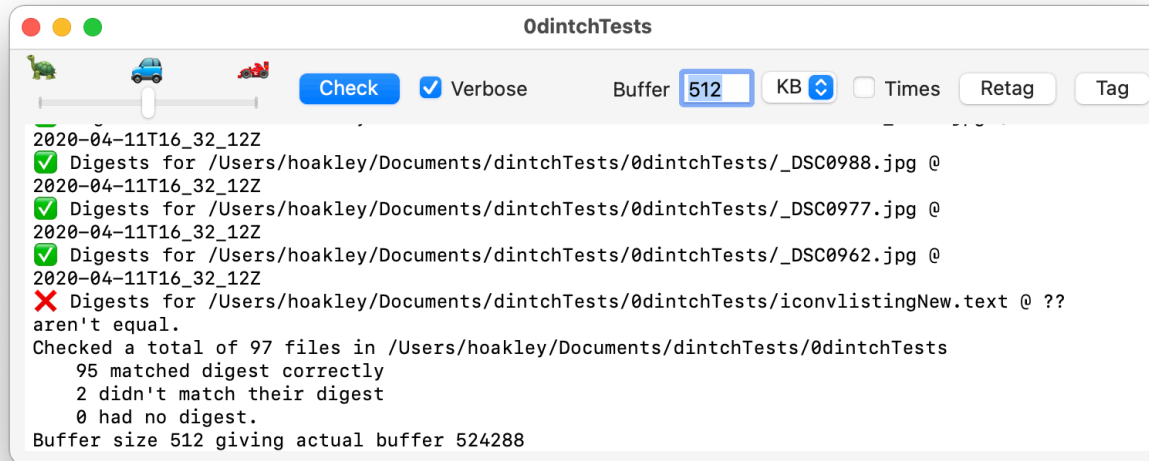


# Start



Dintch is a utility for tagging files with their checksum (actually SHA-256 digests), and checking whether previously tagged files have changed. This lets you determine whether files have changed as the result of ‘bit rot’, errors, or malicious activity.

To tag all the files in a folder, click the **Tag** button and select the folder.

To check the checksums of files in a previously tagged folder, click the **Check** button and select the folder.

To refresh the tags in a previously tagged folder, click the **Retag** button and select the folder.

The **Verbose** checkbox gives fuller detail in reports, and **Times** saves timestamps with each tag. Save a report to a text file using the **Save...** command in the **File** menu.

→ [Speed](#) → [Tag](#) → [Retag](#) → [Check](#) → [Times](#) → [Buffer size](#) → [Updates](#) → [Technical Information](#)

# Speed

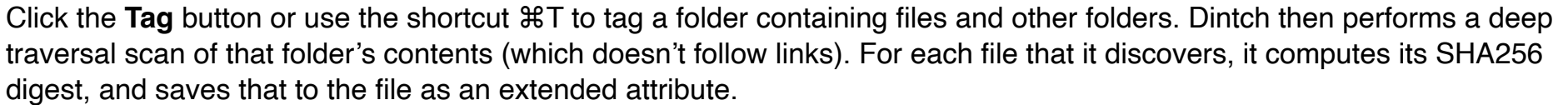


Dintch's tasks are run at one of three speeds. Although these have lesser effects on Intel Macs, they're most useful on Apple silicon models, where they control which processor cores are used. The slider at the left of the window has three speed settings:

- 🐢 a green turtle, which runs threads in the background, using only efficiency (E) cores on Apple silicon.
- 🚗 a blue pickup, which runs threads at high priority (userInitiated), using up to four performance (P) cores on Apple silicon.
- 🏎️ a red racing car, which runs threads at high priority (userInitiated), using up to twenty cores of both types on Apple silicon.

Because of the nature of its tasks, the big difference is between the turtle and the two vehicles. Running these threads on the E cores slows them down considerably, but leaves your apps the full power of the P cores to carry on with their work.

If you want Dintch's tasks completed as quickly as possible, then either of the other two settings will be much quicker, but could slow down other apps that your Mac is running.



This is safe to perform even on files within signed bundles such as apps, as extended attributes aren't included in signature checks. That tag should remain associated with that file on all file systems such as HFS+ and APFS which respect extended attributes.

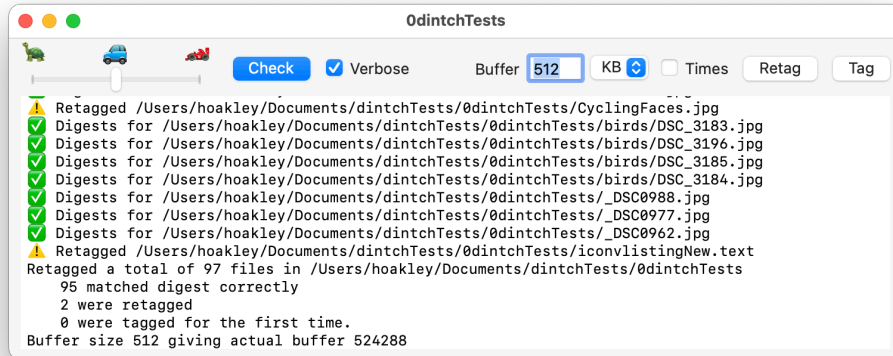
⚠ Writing tags to files can count as changing them, and will result in some backup software, notably Time Machine, making a new backup of each tagged file. If you tag many files, this can increase the size of your next backup very substantially.

The **Verbose** checkbox provides additional detail in the report. Save the contents of that text view using the **Save...** command in the **File** menu. You can also change the size of the output text using ⌘+ to enlarge and ⌘– to reduce the size.

The **Times** checkbox saves a timestamp to each file as it is tagged.

→ Retag → Check → Times → Buffer size → Technical information

# Retag



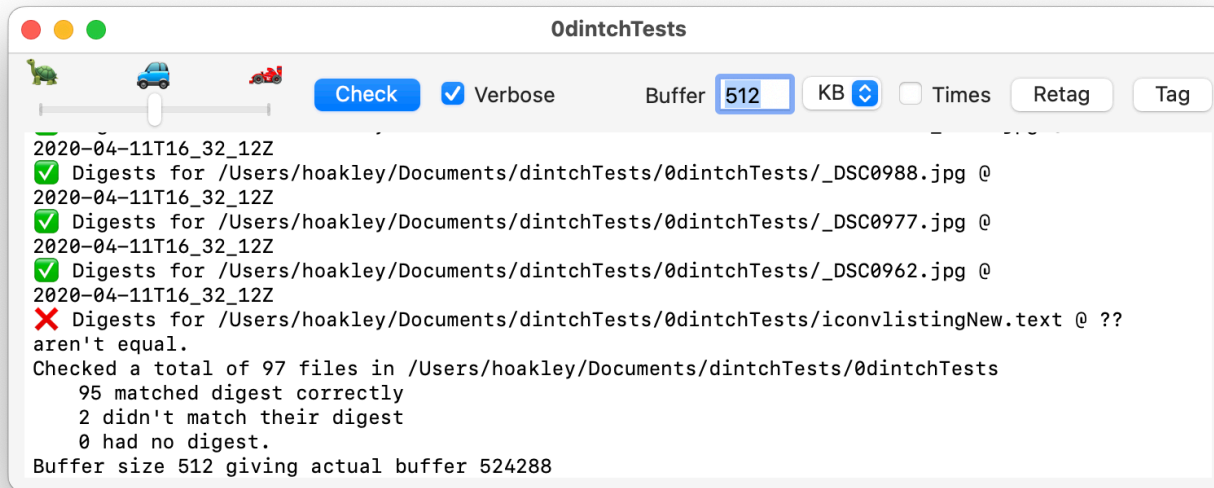
Click the **Retag** button or use the shortcut ⌘R to retag a folder containing files and other folders. Dintch then performs a deep traversal scan of that folder's contents (which doesn't follow links). For each file that it discovers, it computes its SHA256 digest and compares that to any already attached to that file. If they're the same, it doesn't alter the existing tag. If they're different, it writes the new digest as the file's tag. If the file doesn't have an existing tag, it adds one containing the new digest. This minimises the number of changed tags, and any subsequent Time Machine backup.

This is safe to perform even on files within signed bundles such as apps, as extended attributes aren't included in signature checks. That tag should remain associated with that file on all file systems such as HFS+ and APFS which respect extended attributes.

The **Verbose** checkbox provides additional detail in the report. Save the contents of that text view using the **Save...** command in the **File** menu. You can also change the size of the output text using ⌘+ to enlarge and ⌘- to reduce the size.

→ [Tag](#) → [Check](#) → [Times](#) → [Buffer size](#) → [Technical information](#)

# Check

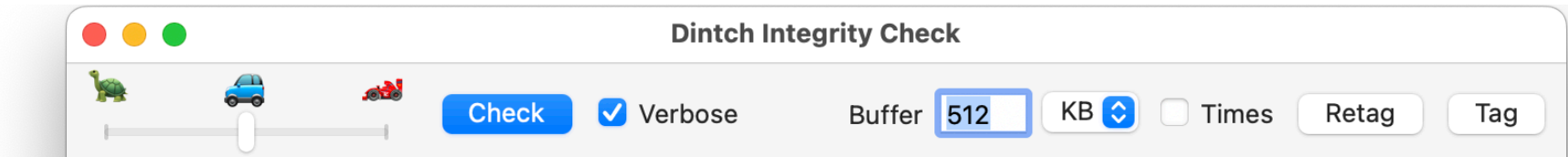


Click the **Check** button or use the shortcut ⌘↵ to check a folder containing files and other folders. Dintch then performs a deep traversal scan of that folder's contents (which doesn't follow links). For each file that it discovers, it compares its current SHA256 digest with any saved to that file already.

The **Verbose** checkbox provides additional detail in the report. Save the contents of that text view using the **Save...** command in the **File** menu. You can also change the size of the output text using ⌘+ to enlarge and ⌘- to reduce the size.

→ [Tag](#)    → [Retag](#)    → [Times](#)    → [Buffer size](#)    → [Technical information](#)

# Times



Tagging a file *doesn't* change its main timestamps. If you want each file to keep a record of when it was last tagged, tick the **Times** checkbox when tagging and retagging them. This saves an additional extended attribute containing the time of its tagging to that file. Any timestamps found when checking in verbose mode are reported. These can be used for audit and forensic purposes, although because they are in text format, they are relatively easy for another app to modify.

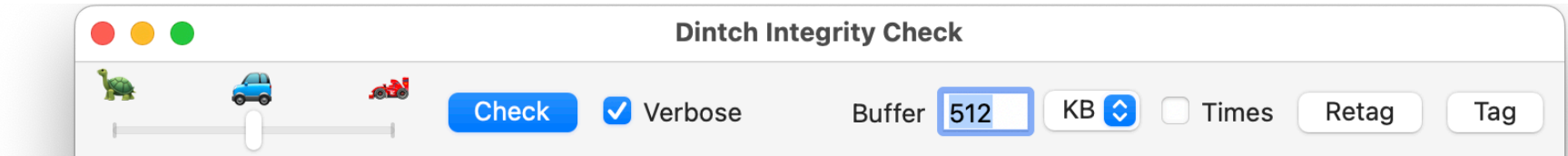
With the **Times** checkbox ticked, each time Dintch writes a tag to a file, it writes a timestamp. When you use the **Tag** button, each file is given a fresh tag and a new timestamp. When you use the **Retag** button, new timestamps are only written to those files which are given new tags, either because they no longer matched their existing tag, or because they didn't have a tag in the first place.

⚠ For any given folder, you should be consistent in your use of timestamps. For example, if you tag a folder with timestamps, then tag it again with **Times** *unticked*, fresh tags will be written to all the files but the original timestamps will not be removed or updated, and will still show the time of their original tagging.

⚠ Adding timestamps to files makes the process of tagging significantly slower.

→ [Tag](#) → [Retag](#) → [Check](#) → [Buffer size](#) → [Technical information](#)

# Buffer size



When Dintch calculates a SHA256 digest, it does so by streaming the file rather than reading it all into memory at once. The size of the buffer used during streaming affects its performance: if that's very small, then Dintch will take a long time to read large files; if that's too large, then small files won't be read as efficiently.

You can set the buffer size used by Dintch in the box labelled Buffer and the popup menu next to it. The value in the box should be from 1 to 1024, and the menu can make that KB or MB as you wish. Together, these let you set the buffer size to anything from 1 KB to 1024 MB (or 1 GB).

Suggested starting values are:

- for small files, 4 KB
- for larger files, 512 KB
- for very large files, 512 MB.

When you quit Dintch, it remembers the last buffer size settings, which are made the default when you open it again.

[→ Technical information](#)

# Updates

Whenever you open Dintch, it may check to see if an update is available. This *doesn't* use the popular Sparkle mechanism for updating in place, but works as detailed here.

Once Dintch has successfully completed its integrity check, it checks whether update checking has been turned off in its preferences file. If that has, it abandons any attempt to check for updates. If checking is allowed, it then checks when it last checked for updates. If that was more than 12 hours ago, it continues to perform the check. It then connects to my GitHub server, from where it downloads a list of current versions of my apps. It doesn't upload any data to the GitHub server at all, and no statistics beyond GitHub normal connection figures are collected either: no personal identifiers are recorded. If there is an update available, Dintch then checks that its location is on this WordPress blog, and posts a dialog which invites you to download the update.

If you click on the **Download** button, it then points your default browser at that update, which should trigger the update to be downloaded to your normal downloads folder. The update is received as a regular Zip archive, and is exactly the same as you would download from the Downloads page here. It also carries a quarantine flag, so that when you unZip it and install the app inside, it undergoes normal first run 'Gatekeeper' security checks. If you click on the **Ignore** button, Dintch won't remind you about it again for another 12 hours.

An additional item at the end of the **Help** menu explains the update status. If no update check is performed, or the check fails, the last item reads **Update not checked**. If the check is performed and update information is obtained, even when no update is available or you decline to download it, that menu item reads **Checked for update** and is ticked (but still disabled).

You can customise this behaviour by changing Dintch's preferences. The keys to use are:

- `noUpdateCheck`, a Boolean. When set to `true`, this disables all update checking. Default is `false`.
- `updateCheckInt`, a real number (Double). When set to a value greater than 1.0, the minimum time interval between checks, in seconds. Default is 43200, which is 12 hours. If you set it to any value less than 1, Dintch will reset it automatically to that default.

To change either of these, use a Terminal command of the form

```
defaults write co.electiclight.Dintch updateCheckInt '10'
```

which works properly through the preferences server `cfprefsd`.

# Technical Information

Dintch uses macOS CryptoKit's SHA-256 support for calculating digest on files, or its equivalent Common Crypto when running on Mojave and earlier. Digests written in different versions of macOS should be identical, so you can check digests in Big Sur which were originally written in El Capitan.

When files are tagged with their digest, this is written to an extended attribute named `co.electiclight.dintch.hash`, with the flag `#S` in an attempt to preserve that xattr as much as possible. However, apps which use certain save-in-place schemes don't copy the xattr when saving. This effectively strips the xattr from the saved file.

Timestamps are saved in an extended attribute named `co.electiclight.dintch.time`, also with the flag `#S`. This contains the time at which the digest was saved to that file as UTF-8 text, in the format YYYY-MM-DDTHH\_MM\_SSZ (UTC).

## Change list

- 1.4
  - Added speed control.
- 1.3:
  - Universal App.
- 1.2:
  - fixed a memory leak in digests.
- 1.1:
  - added timestamps
  - saves Verbose and Times settings to Preferences.

*1.0:*

- fixed bug in saving buffer sizes when quitting the app
- added Retag feature.

*1.0b2:*

- now built to run on El Capitan and later
- added control over buffer size
- extended reporting.

*1.0b1:*

- initial release.

14 June 2022.