ownCloud Desktop Client Manual

The ownCloud Team

Version: 2.8, January 16, 2022

Table of Contents

Introduction

Available for Windows, Mac OS X, and various Linux distributions, the ownCloud Desktop Sync client enables you to:

- Specify one or more directories on your computer that you want to synchronize to the ownCloud server.
- Always have the latest files synchronized, wherever they are located.

Your files are always automatically synchronized between your ownCloud server and local PC.

Desktop Sync clients older than. 2.2.1 are not allowed to connect and sync with ownCloud 8.1+ server. It is highly recommended to keep your server and client updated.

Improvements and New Features

Each release of the ownCloud Desktop Sync client has new features and improvements, for details see the complete changelog.

Installing the Desktop Synchronization Client

Introduction

You can download the latest version of the ownCloud Desktop Synchronization Client from the ownCloud download page. There are clients for *Linux*, *macOS*, and *Microsoft Windows*.

Installation on Mac OS X and Windows is the same as for any software application: download the program and then double-click it to launch the installation, and then follow the installation wizard. After it is installed and configured the sync client will automatically keep itself updated; see autoupdate for more information.

Linux users must follow the instructions on the download page to add the appropriate repository for their Linux distribution, install the signing key, and then use their package managers to install the desktop sync client. Linux users will also update their sync clients via package manager, and the client will display a notification when an update is available.

Linux users must also have a password manager enabled, such as GNOME Keyring or KWallet, so that the sync client can login automatically.

You will also find links to source code archives and older versions on the download page.

System Requirements

- Windows 7+ (x86 with 32-bit or x86-64; Native WinVFS available for Windows 10 version 1709 or later)
- macOS 10.12+ (x86-64 or Apple M in Rosetta 2 emulation; unsupported legacy builds for Mac OS X 10.10 & 10.11 available)
- CentOS 7.6+ & 8 (x86-64)
- Debian 9.0 & 10 (x86-64)
- Fedora 31 & 32 & 33 (x86-64)
- Ubuntu 18.04 & 20.04 & 20.10 & 21.04 (x86-64)
- openSUSE Leap 15.1 & 15.2 (x86-64)

For Linux distributions, we support, if technically feasible, the latest 2 versions per platform and the previous Ubuntu LTS.

Customizing the Windows Installation

If you just want to install ownCloud Desktop Synchronization Client on your local system, you can simply launch the .msi file and configure it in the wizard that pops up.

Features

The MSI installer provides several features that can be installed or removed individually, which you can also control via command-line, if you are automating the installation, then run the following command:

```
msiexec /passive /i ownCloud-x.y.z.msi.
```

The command will install the ownCloud Desktop Synchronization Client into the default location with the default features enabled. If you want to disable, e.g., desktop shortcut icons you can simply change the above command to the following:

```
msiexec /passive /i ownCloud-x.y.z.msi REMOVE=DesktopShortcut.
```

See the following table for a list of available features:

Feature	Enabled by default	Description	Property to disable.
Client	Yes, required	The actual client	
DesktopShortcut	Yes	Adds a shortcut to the desktop.	NO_DESKTOP_SHORTCUT
StartMenuShortcuts	Yes	Adds shortcuts to the start menu.	NO_START_MENU_SHORTCUTS
ShellExtensions	Yes	Adds Explorer integration	NO_SHELL_EXTENSIONS

Installation

You can also choose to only install the client itself by using the following command:

```
msiexec /passive /i ownCloud-x.y.z.msi ADDDEFAULT=Client.
```

If you for instance want to install everything but the DesktopShortcut and the ShellExtensions feature, you have two possibilities:

- You explicitly name all the features you actually want to install (whitelist) where Client is always installed anyway.

```
msiexec /passive /i ownCloud-x.y.z.msi ADDDEFAULT=StartMenuShortcuts.
```

- You pass the NO_DESKTOP_SHORTCUT and NO_SHELL_EXTENSIONS properties.

```
msiexec /passive /i ownCloud-x.y.z.msi NO_DESKTOP_SHORTCUT="1"
NO_SHELL_EXTENSIONS="1"
```

The ownCloud .msi file remembers these properties, so you don’t need to specify them on upgrades.

You cannot use these to change the installed features, if you want to do that, see the next section.

Changing Installed Features

You can change the installed features later by using REMOVE and ADDDEFAULT properties.

- If you want to add the desktop shortcut later, run the following command:

```
msiexec /passive /i ownCloud-x.y.z.msi ADDDEFAULT="DesktopShortcut"
```

- If you want to remove it, simply run the following command:

```
msiexec /passive /i ownCloud-x.y.z.msi REMOVE="DesktopShortcut"
```

Windows keeps track of the installed features and using REMOVE or ADDDEFAULT will only affect the mentioned features.

Compare REMOVE and ADDDEFAULT on the Windows Installer Guide.

You cannot specify REMOVE on initial installation as it will disable all features.

Installation Folder

You can adjust the installation folder by specifying the INSTALLDIR property like this.

```
msiexec /passive /i ownCloud-x.y.z.msi INSTALLDIR="C:\Program Files (x86)\Non Standard ownCloud Client Folder"
```

Be careful when using PowerShell instead of cmd.exe, it can be tricky to get the whitespace escaping right there. Specifying the INSTALLDIR like this only works on first installation, you cannot simply re-invoke the .msi with a different path. If you still need to change it, uninstall it first and reinstall it with the new path.

Disabling Automatic Updates.

To disable automatic updates, you can pass the SKIPAUTOUPDATE property.

```
msiexec /passive /i ownCloud-x.y.z.msi SKIPAUTOUPDATE="1"
```

Launch After Installation

To launch the client automatically after installation, you can pass the LAUNCH property.

```
msiexec /i ownCloud-x.y.z.msi LAUNCH="1"
```

This option also removes the checkbox to let users decide if they want to launch the client for non passive/quiet mode.

This option does not have any effect without GUI.

No Reboot After Installation

The ownCloud Client schedules a reboot after installation to make sure the Explorer extension is correctly (un)loaded. If you're taking care of the reboot yourself, you can set the REBOOT property.

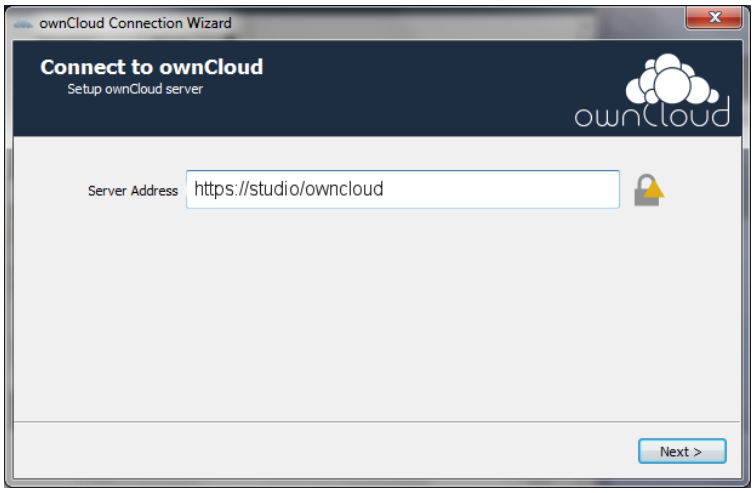
```
msiexec /i ownCloud-x.y.z.msi REBOOT=ReallySuppress.
```

This will make msiexec exit with error ERROR_SUCCESS_REBOOT_REQUIRED (3010). If your deployment tooling interprets this as an actual error and you want to avoid that, you may want to set the DO_NOT_SCHEDULE_REBOOT instead.

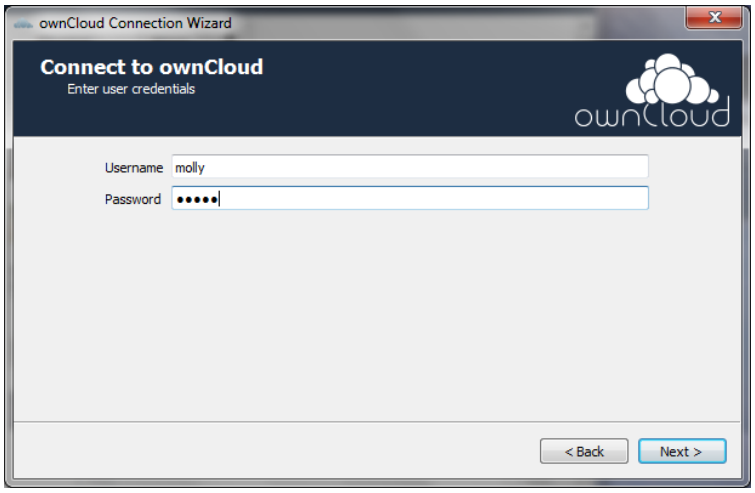
```
msiexec /i ownCloud-x.y.z.msi DO_NOT_SCHEDULE_REBOOT="1"
```

Installation Wizard

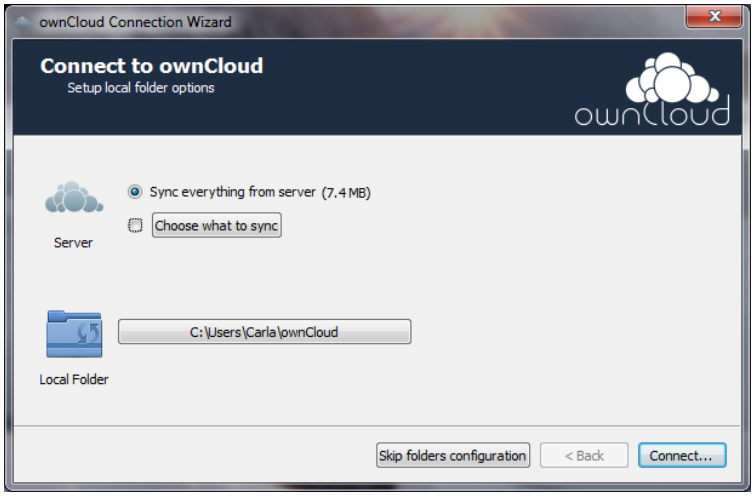
The installation wizard takes you step-by-step through configuration options and account setup. First you need to enter the URL of your ownCloud server.



Enter your ownCloud login on the next screen.



On the "*Local Folder Option*" screen you may sync all of your files on the ownCloud server, or select individual folders. The default local sync folder is ownCloud, in your home directory. You may change this as well.



When you have completed selecting your sync folders, click the "*Connect*" button at the bottom right. The client will attempt to connect to your ownCloud server, and when it is successful you'll see two buttons:

- One to connect to your ownCloud Web GUI.
- One to open your local folder.

It will also start synchronizing your files.

Using the Synchronization Client

Introduction

The ownCloud Desktop Client remains in the background and is visible as an icon in the system tray (Windows, KDE), menu bar (macOS), or notification area (Linux).

Used Client Icons

The status indicator uses icons to indicate the current status of your synchronization. The green circle with the white checkmark tells you that your synchronization is current and you are connected to your ownCloud server.

The blue icon with the white semi-circles means synchronization is in progress.

The yellow icon with the parallel lines tells you your synchronization has been paused. (Most likely by you.)

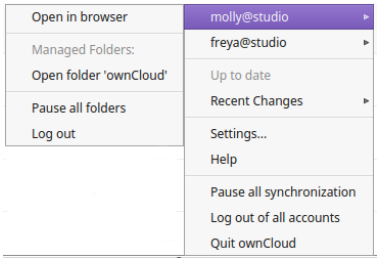
The gray icon with three white dots means your sync client has lost its connection with your ownCloud server.

When you see a white circle with the letter "i" that is the informational icon, so you should click it to see what it has to tell you.

The red circle with the white "x" indicates a configuration error, such as an incorrect login or server URL.

Systray Icon

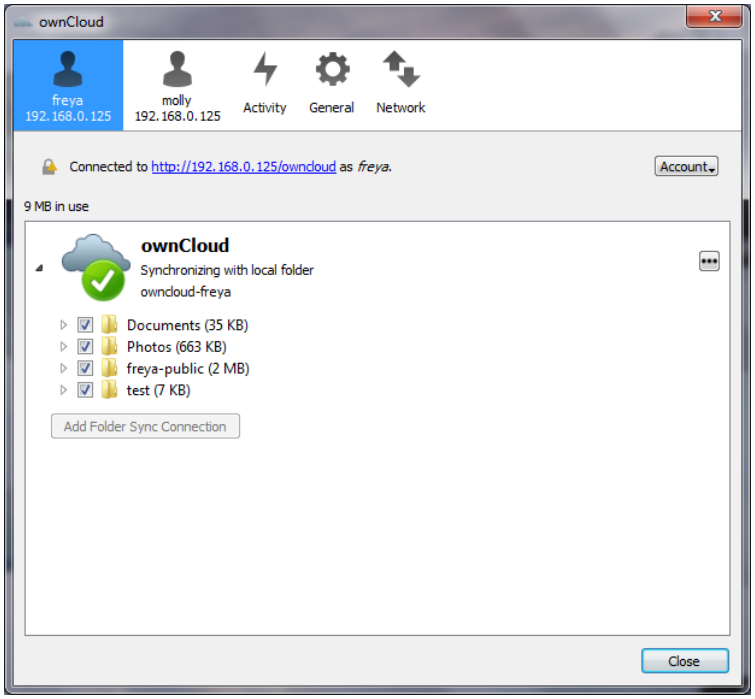
A right-click on the systray icon opens a menu for quick access to multiple operations.



This menu provides the following options:

- Quick access to your accounts.
- Sync status.
- Recent Changes, showing latest activities.
- Settings.
- Help menu.
- Pause synchronizations.
- An option to log in or log out of all of your accounts at once.
- Quit ownCloud, logging out and closing the client.

A left-click on your systray icon opens the desktop client to the account settings window.



Configuring ownCloud Account Settings

At the top of the window are tabs for each configured sync account, and three others for Activity, General and Network settings. On your account tabs you have the following features:

- Connection status, showing which ownCloud server you are connected to, and your ownCloud username.

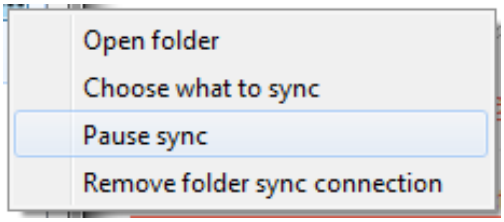
- An **Account** button, which contains a dropdown menu with **Add New**, **Log Out**, and **Remove**.
- Used and available space on the server.
- Current synchronization status.
- **Add Folder Sync Connection** button.

The little button with three dots (the overflow menu) that sits to the right of the sync status bar offers four additional options:

- Open Folder.
- Choose What to Sync (This appears only when your file tree is collapsed, and expands the file tree)
- Pause Sync / Resume Sync.
- Remove folder sync connection.

Open Folder opens your local ownCloud sync folder.

Pause Sync pauses sync operations without making any changes to your account. It will continue to update file and folder lists, without downloading or updating files. To stop all sync activity use **Remove Folder Sync Connection**.



ownCloud does not preserve the mtime (modification time) of directories, though it does update the mtimes on files. See Wrong folder date when syncing for discussion of this.

Adding New Accounts

You may configure multiple ownCloud accounts in your desktop sync client. Simply click the **Account** › **Add New** button on any account tab to add a new account, and then follow the account creation wizard. The new account will appear as a new tab in the settings dialog, where you can adjust its settings at any time. Use **Account** › **Remove** to delete accounts.

To use **Two-Factor Authentication** (2FA), ownCloud server must have the OAuth2 app installed, configured, and enabled. Please contact your ownCloud administrator for more details.

File Manager Overlay Icons

The ownCloud sync client provides overlay icons, in addition to the normal file type icons, for your system file manager (Explorer on Windows, Finder on Mac and Nautilus on Linux) to indicate the sync status of your ownCloud files.

The overlay icons are similar to the systray icons introduced above. They behave differently on files and directories according to sync status and errors.

The overlay icon of an individual file indicates its current sync state. If the file is in sync with the server version, it displays a green checkmark.

If the file is ignored from syncing, for example because it is on your exclude list, or because it is a symbolic link, it displays a warning icon.

If there is a sync error, or the file is blacklisted, it displays an eye-catching red X. If the file is waiting to be synced, or is currently syncing, the overlay icon displays a blue cycling icon.

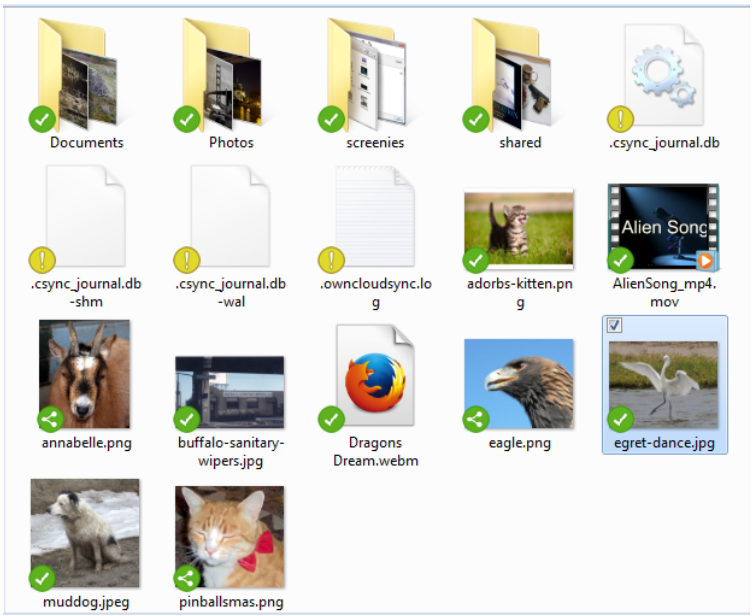
When the client is offline, no icons are shown to reflect that the folder is currently out of sync and no changes are synced to the server.

The overlay icon of a synced directory indicates the status of the files in the directory. If there are any sync errors, the directory is marked with a warning icon.

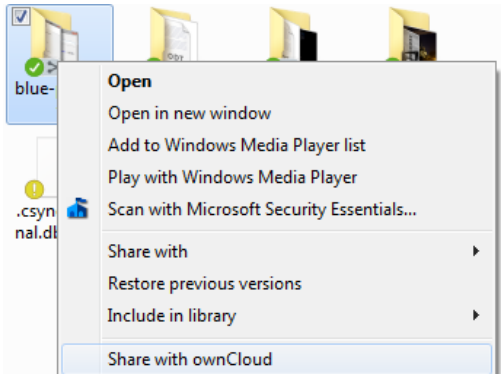
If a directory includes ignored files that are marked with warning icons that does not change the status of the parent directories.

Sharing From Your Desktop

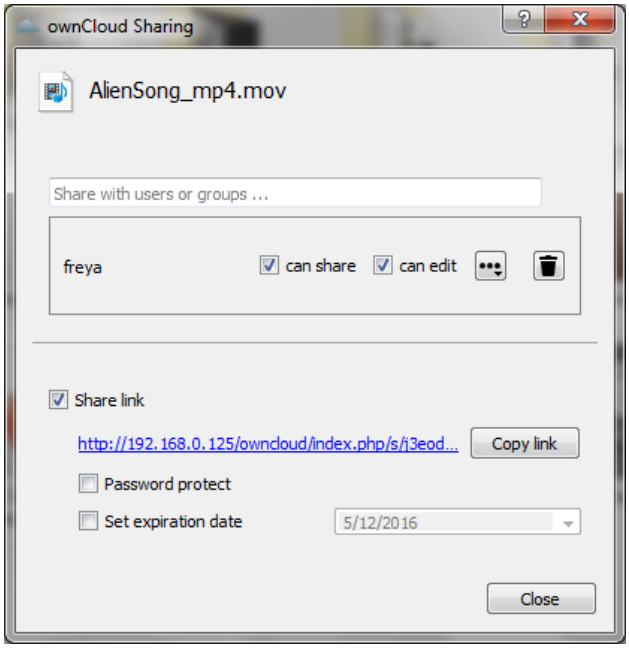
The ownCloud desktop sync client integrates with your file manager: Finder on Mac OS X, Explorer on Windows, and Nautilus on Linux. (Linux users must install the owncloud-client-nautilus plugin.) You can create share links, and share with internal ownCloud users the same way as in your ownCloud Web interface.



Right-click your systray icon, hover over the account you want to use, and left-click **Open folder › "folder name"** to quickly enter your local ownCloud folder. Right-click the file or folder you want to share to expose the share dialog, and click **Share with ownCloud**.



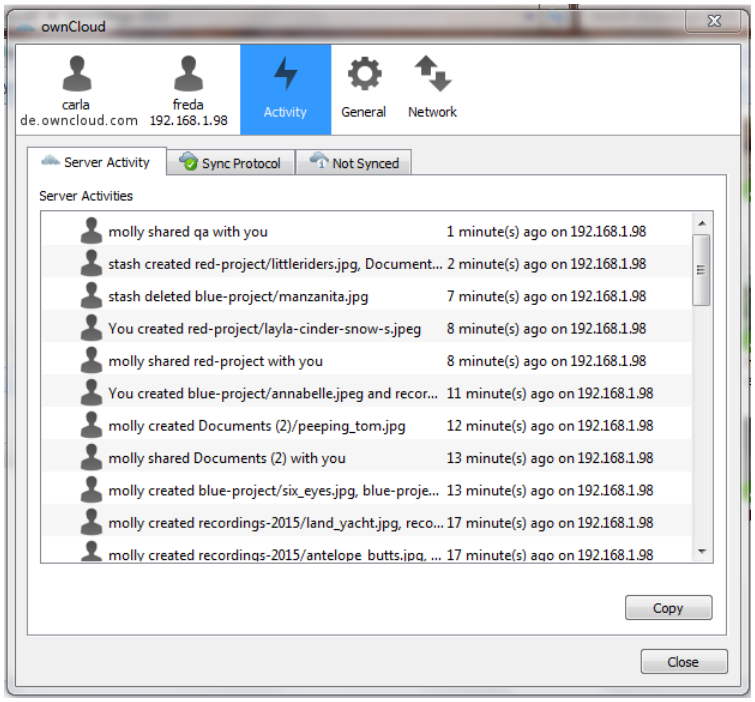
The share dialog has all the same options as your ownCloud Web interface.



Use **Share with ownCloud** to see who you have shared with, and to modify their permissions, or to delete the share.

Activity Window

The Activity window contains the log of your recent activities, organized over three tabs: **Server Activities**, which includes new shares and files downloaded and deleted, **Sync Protocol**, which displays local activities such as which local folders your files went into, and. **Not Synced** shows errors such as files not synced. Double clicking an entry pointing to an existing file in **Server Activities** or **Sync Protocol** will open the folder containing the file and highlight it.

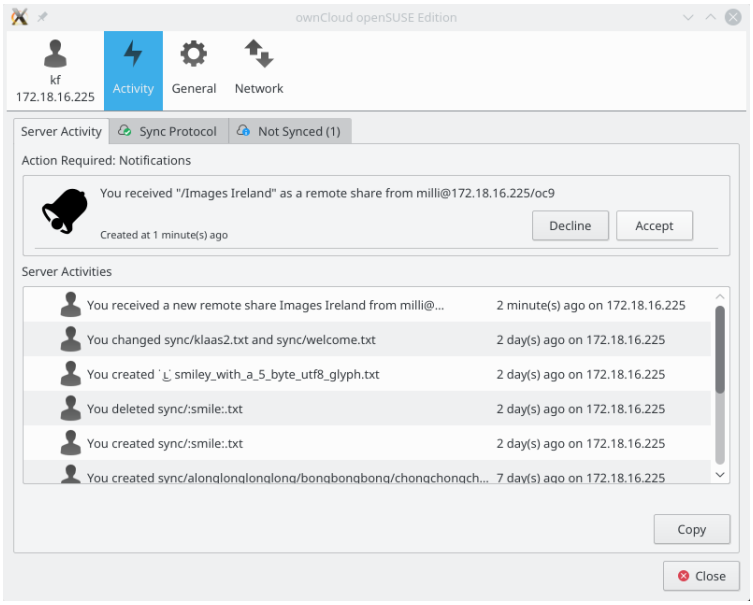


Server Notifications

Starting with version 2.2.0, the client will display notifications from your ownCloud server that require manual interaction by you. For example, when a user on a remote ownCloud creates a new Federated share for you, you can accept it from your desktop

client.

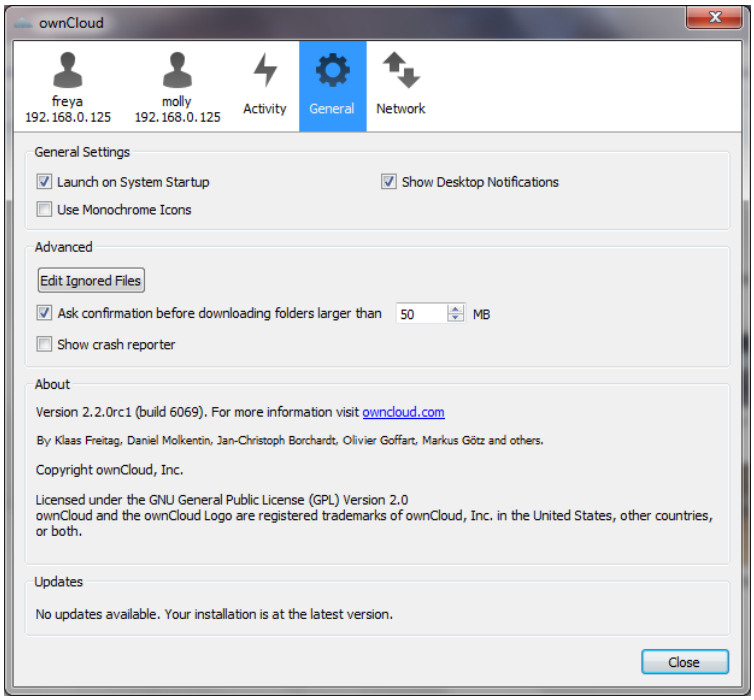
The desktop client automatically checks for available notifications automatically on a regular basis. Notifications are displayed in the Server Activity tab, and if you have **Show Desktop Notifications**. enabled (General tab) you'll also see a systray notification.



This also displays notifications sent to users by the ownCloud admin via the Announcements app.

General Window

The General window has configuration options such as "*Launch on System Startup*", "*Use Monochrome Icons*", and "*Show Desktop Notifications*". This is where you will find the "*Edit Ignored Files*" button, to launch the ignored files editor, and "*Ask confirmation before downloading folders larger than [folder size]*".

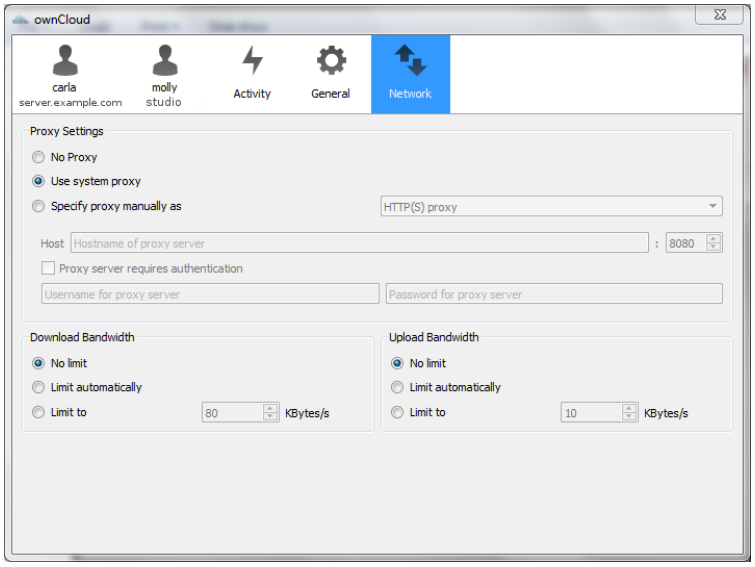


While you can elect whether to show or hide the crash reporter, from the General window, you can also configure whether to show or hide it from the general section of the configuration file as well. Doing so can help with debugging on-startup-crashes.

Using the Network Window

proxy settings, SOCKS, bandwith, throttling, limiting.

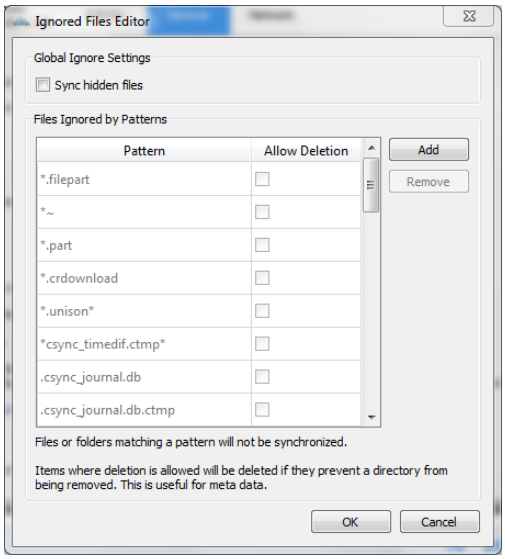
The Network settings window enables you to define network proxy settings, and also to limit download and upload bandwidth.



Using the Ignored Files Editor

Ignored files, exclude files, patterns.

You might have some local files or directories that you do not want to backup and store on the server. To identify and exclude these files or directories, you can use the **General Tab › Ignored Files Editor**

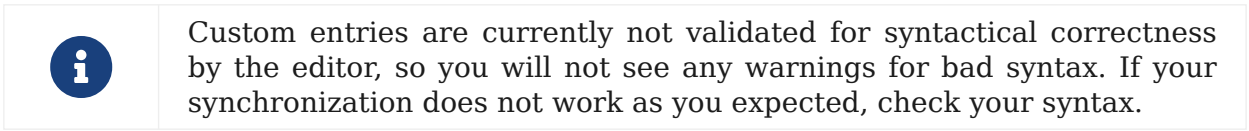


For your convenience, the editor is pre-populated with a default list of typical ignore patterns. These patterns are contained in a system file. (typically sync-exclude.lst) located in the ownCloud Client application directory. You cannot modify these pre-populated patterns directly from the editor. However, if necessary, you can hover over

any pattern in the list to show the path and filename associated with that pattern, locate the file, and edit the sync-exclude.lst file.

Each line in the editor contains an ignore pattern string. When creating custom patterns, in addition to being able to use normal characters to define an ignore pattern, you can use wildcards characters for matching values. As an example, you can use an asterisk (*) to identify an arbitrary number of characters or a question mark (?) to identify a single character.

Patterns that end with a slash character (/) are applied to only directory components of the path being checked.



Each pattern string in the list is preceded by a checkbox. When the check box contains a check mark, in addition to ignoring the file or directory component matched by the pattern, any matched files are also deemed "fleeting metadata" and removed by the client.

In addition to excluding files and directories that use patterns defined in this list:

- The ownCloud Client always excludes files containing characters that cannot be synchronized to other file systems.
- Files are removed that cause individual errors three times during a synchronization. However, the client provides the option of retrying a synchronization three additional times on files that produce errors.

For more detailed information see the Ignored Files section.

Using the Virtual Filesystem

Introduction

ownCloud offers the possibility for users to enable a virtual file system (VFS) when synchronizing data. This has the big advantage that all files and folders are visible to the client, but the files are not downloaded until the user requests to do so. Here are some of the key benefits:

- Full access to files and folders without having to download them all first
- Selectively sync folders and files based on user requirements
- Optimize space usage on the client

The quote below gives you a brief overview of what a virtual file system is about.

> A virtual file system (VFS) or virtual filesystem switch is an abstract layer on top of a more concrete file system. The purpose of a VFS is to allow client applications to access different types of concrete file systems in a uniform way. A VFS can, for example, be used to access local and network storage devices transparently without the client application noticing the difference. It can be used to bridge the differences in Windows, classic Mac OS/macOS and Unix filesystems, so that applications can access files on local file systems of those types without having to know what type of file system they are accessing.
>
> — https://en.wikipedia.org/wiki/Virtual_file_system

Microsoft VFS Implementation

Background

> A sync engine is a service that syncs files, typically between a remote host and a local client. Sync engines on Windows often present those files to the user through the Windows file system and File Explorer.
>
> — https://docs.microsoft.com/en-us/windows/win32/cfapi/build-a-cloud-file-sync-engine

Files can exist in three states:

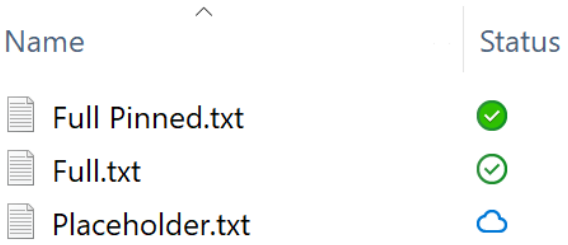
Full pinned file
: The file has been *hydrated* explicitly by the user through File Explorer and is guaranteed to be available offline.

Full file
: The file has been *hydrated* implicitly and could be *dehydrated* by the system if space is needed.

Placeholder file
: An empty representation of the file and only available if the sync service is available.

The following image demonstrates how the *full pinned*, *full* and *placeholder* file states

are shown in File Explorer.



Limitations and Restrictions

Limitations

A virtual file system needs a root folder all synchronization items will be stored in. The following locations are **not** allowed as synchronization root:

- The root of a disk like D:\
- A non-NTFS Filesystem
- Mounted network shares
- Symbolic links or junction points
- Assigned drives

Restrictions

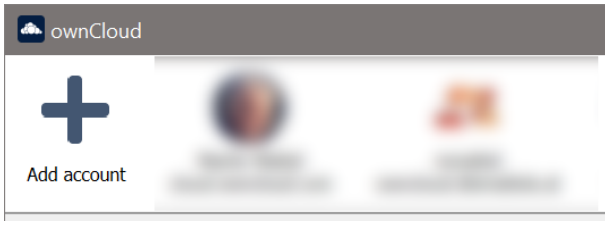
Similar to OneDrive as it also uses Microsoft's virtual file system, there are some additional restrictions which should be considered like the *maximum file size*, *invalid file or folder names*, etc. See the Restrictions and limitations in OneDrive and SharePoint for more information.

ownCloud VFS Implementation

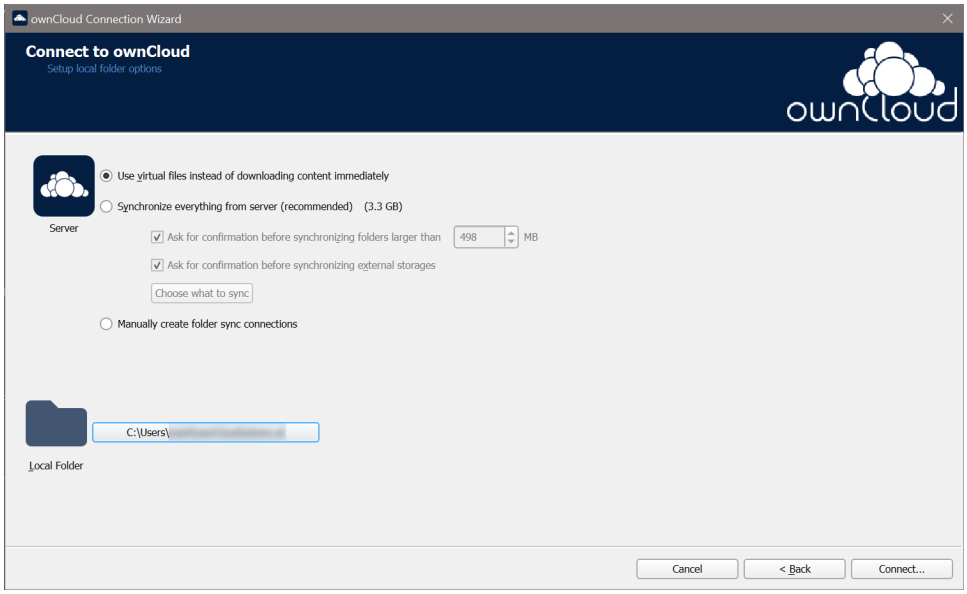
New Sync with VFS enabled

To set up a new synchronization with virtual file system enabled, perform the following steps:

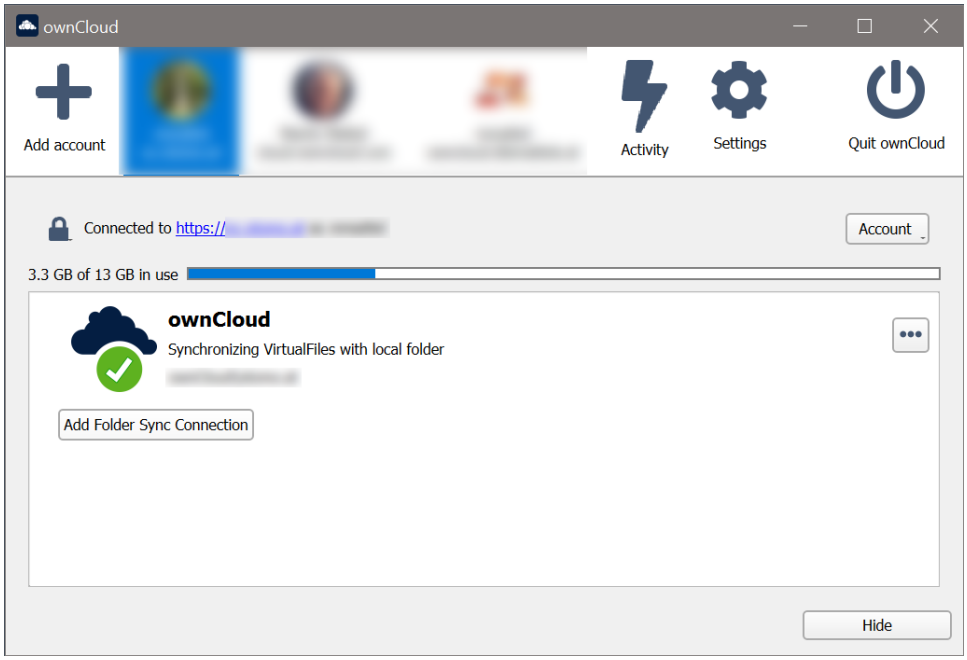
1. Add a new synchronization by clicking the **`[+ Add account]`** button.



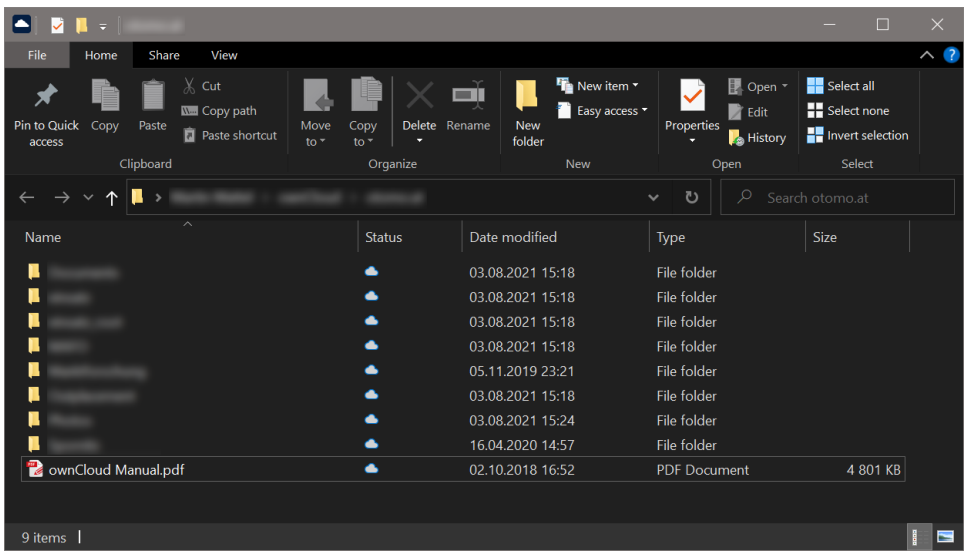
2. Enter the server address and your credentials in the following dialogs.
3. Select the radio button **`[Use virtual files]`** and set the local folder where your synchronization data will reside.



4. When everything is done, you should see a similar screen as below, showing that the setup completed successfully.



5. After the first sync, your synchronization folder will show your items with the *Placeholder* icon.

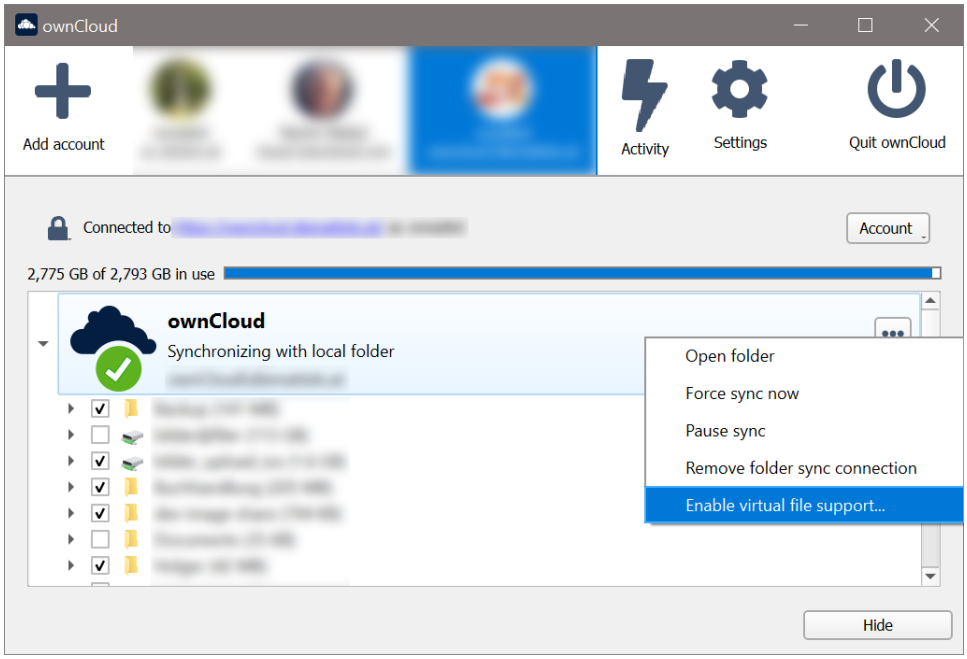


6. When opening a file, the file gets downloaded and its synchronization icon changes to *Full*.

Convert Full Sync to VFS

If you have full synchronization enabled, you can change to a virtual file system at any time.

1. Open your existing synchronization, click the **[…]** button and **Enable virtual file support**.

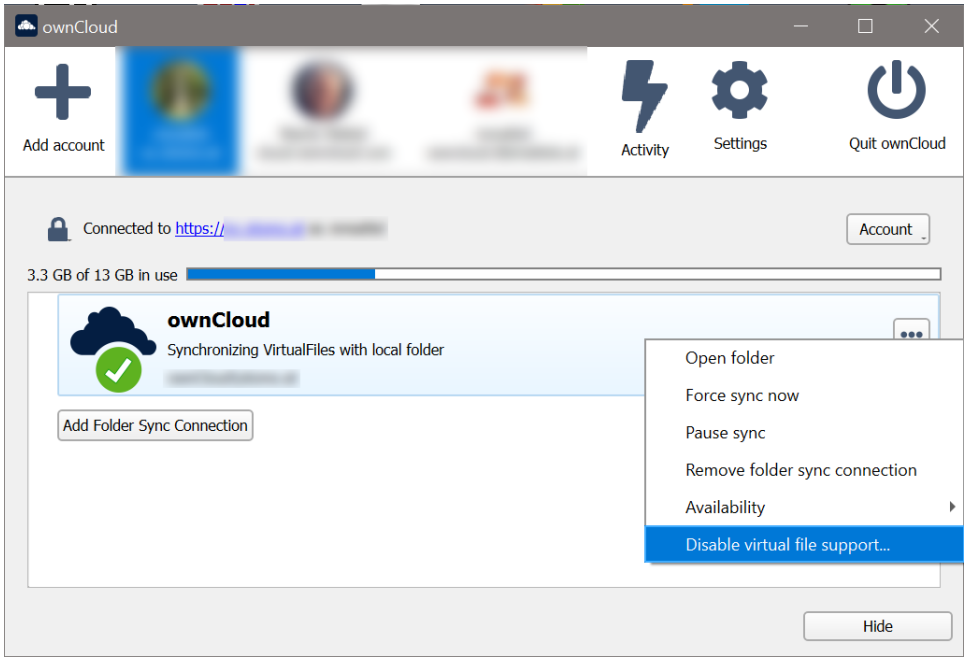


2. Your local files will get replaced by *placeholders*, thus freeing up the space previously occupied.

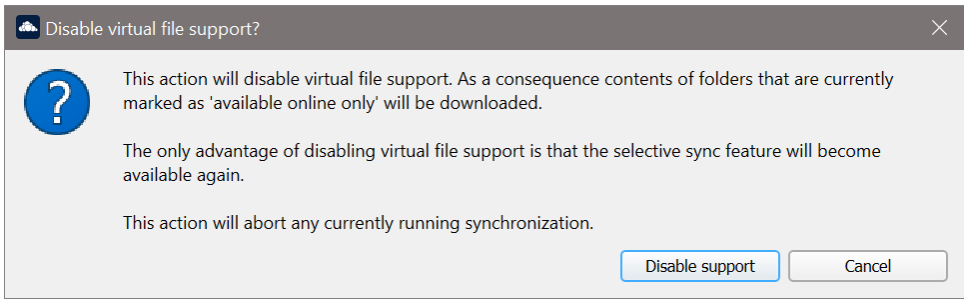
Convert VFS to Full Sync

You can also change the synchronization setting from virtual file system to full sync.

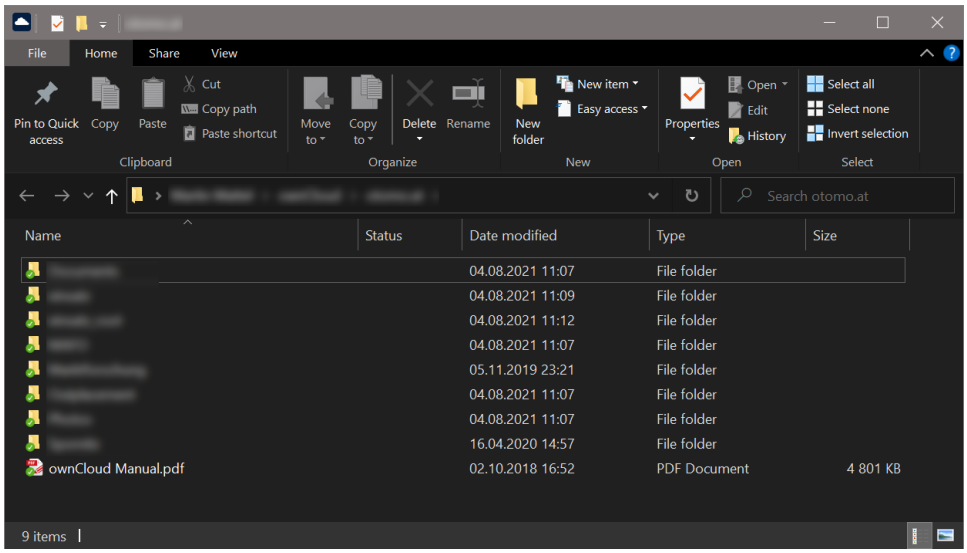
1. Open your existing synchronization, click the **[…]** button and **Disable virtual file support**.



2. A notification window will ask you to confirm before completing the conversion.



3. When done, your files will be fully downloaded, which you can tell by the sync icons, see the example image below. Depending on the quantity and size of the files, this may take a while.

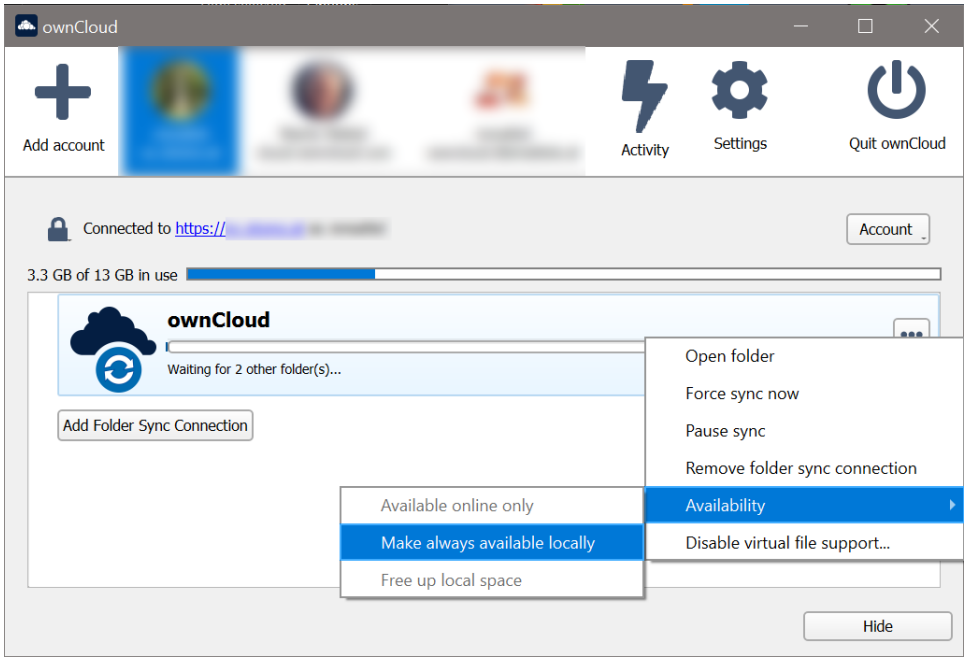


Manage VFS from ownCloud

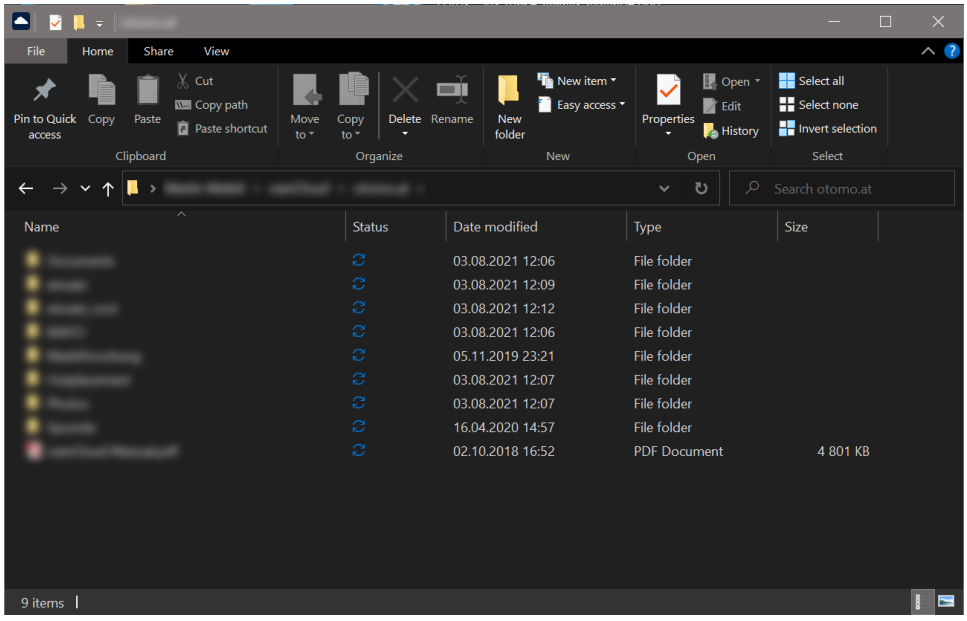
You can manage the synchronization for all files and folders via ownCloud. This can be beneficial if you e.g. want all files to be downloaded (pinned) without you having to open every single one, or if you want to free up all space at once.

Make All Files *Full Pinned*

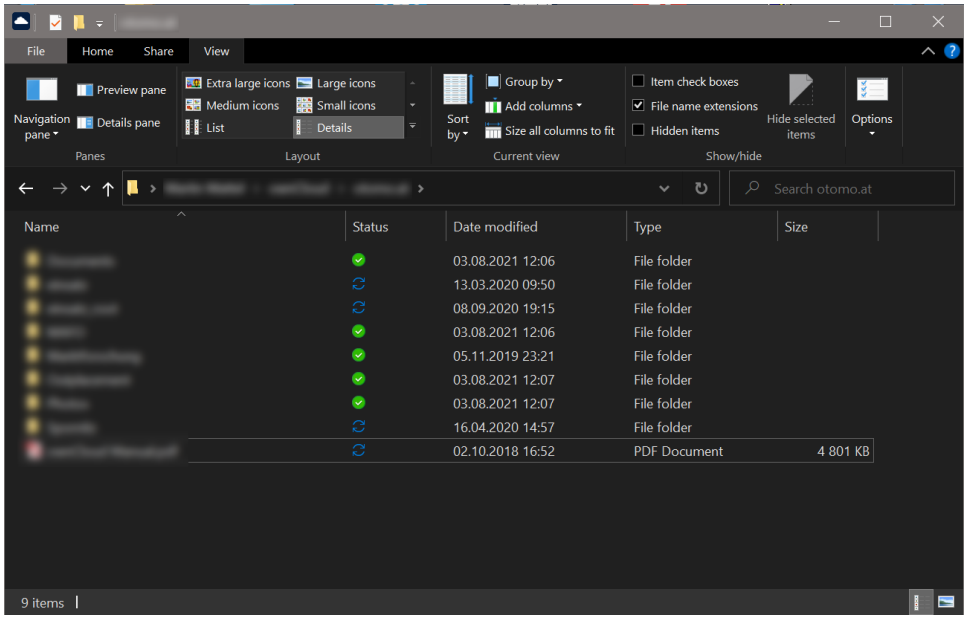
1. Open your existing synchronization, click the **[…]** button and **Availability › Make always available locally**.



2. When opening the Explorer, you will see that all files get the sync icon .



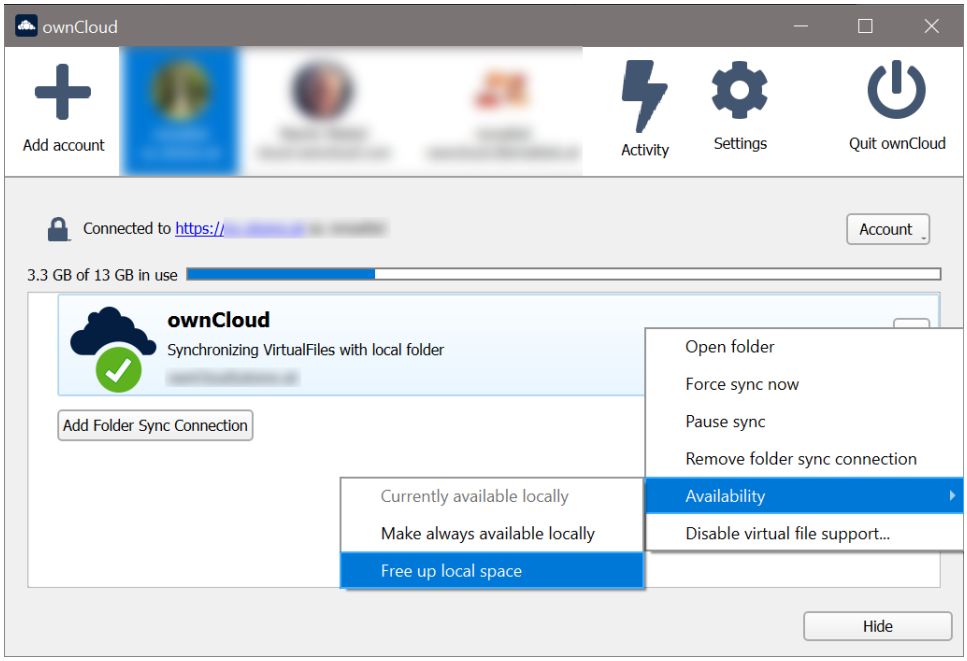
3. The icon will change to *Full Pinned* when downloaded.



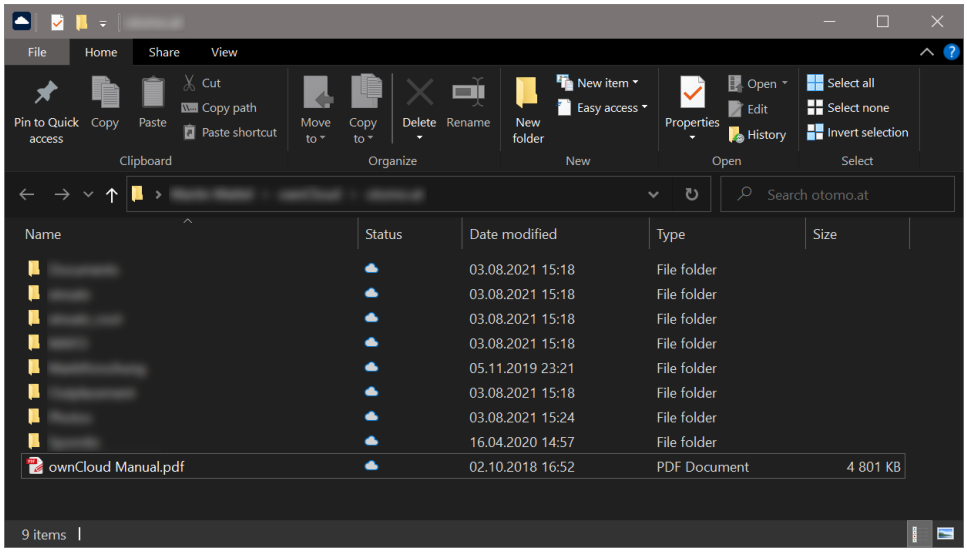
Make All Files *Placeholders*

You can free up space by unpinning all files at once and making them placeholders with only a view clicks.

1. Open your existing synchronization, click the **[…]** button and **Availability** › **Free up local space**.



2. When done, Explorer will show the files in *Placeholder* state.

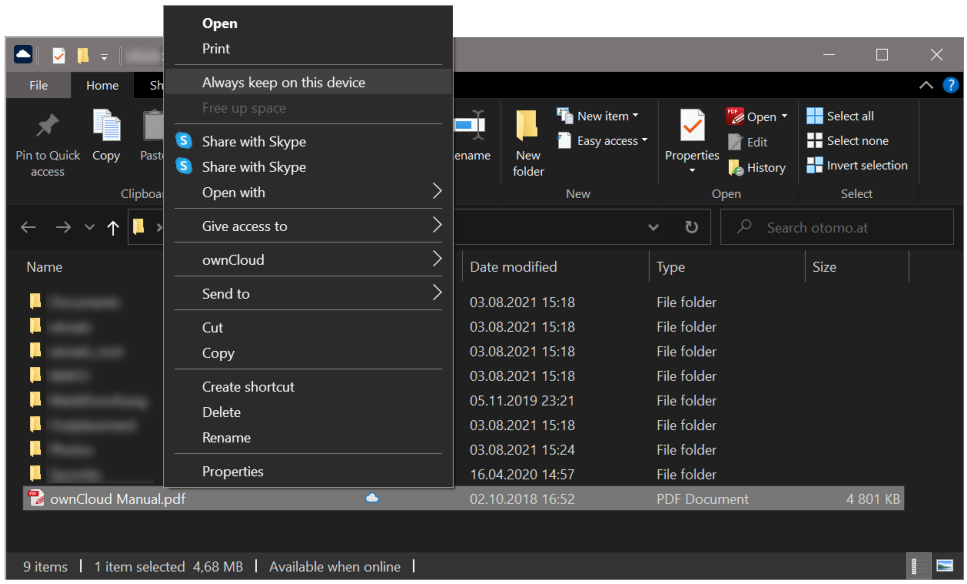


Manage VFS from Windows Explorer

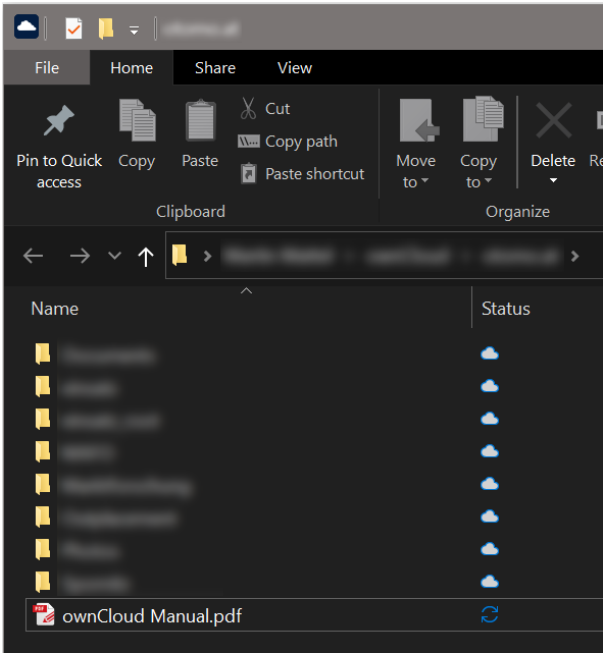
You can manage individual files in the Explorer window by **right-clicking** on them. This opens a drop-down menu of actions that can be performed on a specific file.

Create a Local Copy of a File

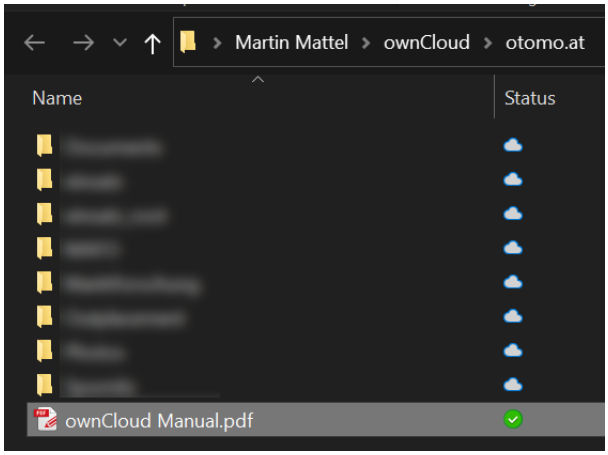
1. To create a Full Pinned file (have a local copy of it), use the action **`[Always keep on this device]`**.



The state of the file will change to synchronizing.

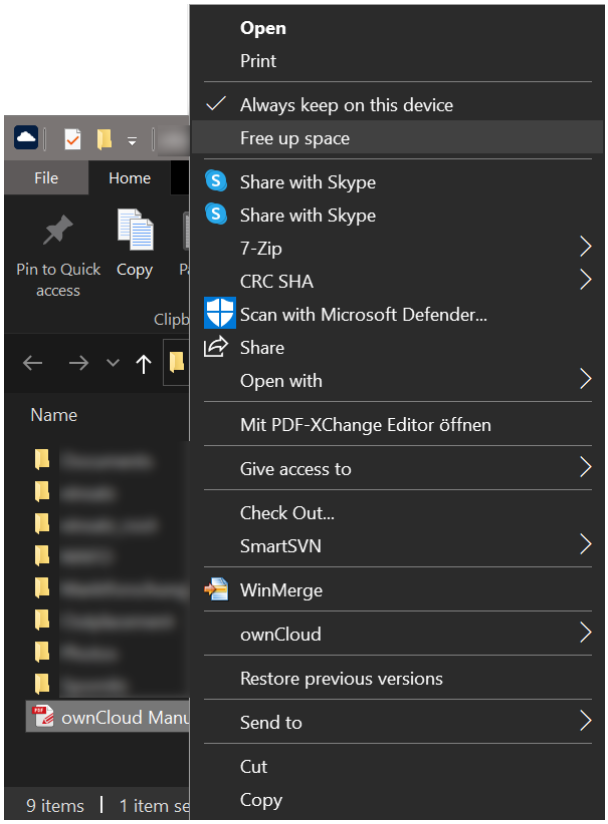


When the local copy has been created, the state (icon) changes to *Full Pinned*.

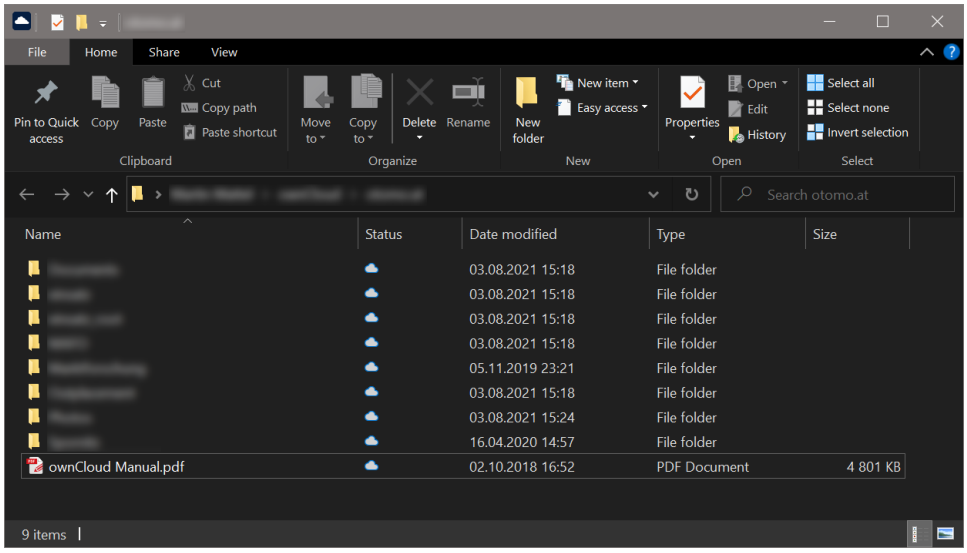


Free up Space of a File

1. To free up the space the file occupied, use the action **`[Free up space]`**.



2. When done, Explorer will show the file in *Placeholder* state.



Filename Considerations

Introduction

When using the client, depending the operating system (OS) you are using, file and folder names can have different restrictions. Creating files and folders with allowed names on one OS, may have issues or even can´t be synced because of different rules in another OS. This page gives you a brief overview of limitations of different OS for file and folder names.

This is not an ownCloud rule but an OS dependency

Here are some rules of thumb

1. Do not use any of the mentioned characters or words in any OS when using the Desktop Sync Client.
2. When the sync client is on Linux/Unix and the target mount to sync on is on SMB, file and folder names on Linux/Unix must comply with the Windows rules for successful syncing.
3. When the sync client is on Linux/Unix and the target mount to sync on is on SMB and you want to just rename the file with different casings, rename the file to a total different name, let it sync and then rename it again to the name that you want.

Forbidden Printable ASCII Characters

Linux/Unix

/ (forward slash)

Windows

< (less than)
> (greater than)
: (colon - sometimes works, but is actually NTFS Alternate Data Streams)
" (double quote)
/ (forward slash)
\ (backslash)
| (vertical bar or pipe)
? (question mark)
* (asterisk)

Non-Printable Characters

If your files are created via a program, do not use non-printable characters. See the Wikipedia "Control code chart" section for more information on ASCII control characters.

Linux/Unix

0 (NULL byte)

While it is legal under Linux/Unix file systems to create files with control characters in the filename, they might be inaccessible and/or unsyncable.

Windows

0-31 (ASCII control characters)

Reserved File Names

The following file names are reserved:

Windows

CON, PRN, AUX, NUL COM1, COM2, COM3, COM4, COM5, COM6, COM7, COM8, COM9, LPT1, LPT2, LPT3, LPT4, LPT5, LPT6, LPT7, LPT8, LPT9

Other Rules

Linux/Unix

When the sync client is on Linux/Unix and the target mount to sync on is on SMB, you cannot have the same file or folder name but with different casings. A cross icon will be shown that indicates that the file can't be synced. Files on Linux/Unix must comply with the Windows rules for successful syncing.

Windows

Filenames cannot end in a space or dot

Examples and Pitfalls

1. When creating a file in Linux/Unix like my-filename. (see the dot at the end) or my-filename.LPT1 (see the reserved name LPT1), you can sync the file to your ownCloud if the mount target is Linux/Unix. When a Windows user tries to sync these files, Windows rejects the file. Comparing the file list in both environments shows that one side has more files than the other. There will be **no notification** as this is an OS dependency.
2. When renaming an existing file in Linux/Unix by just changing the casing like owncloud → ownCloud, you might get issues on the windows sync side as for Windows the file looks the same.

Manage Synchronisation Conflicts

Introduction

The ownCloud desktop client uploads local changes and downloads remote changes. When a file has changed on the local and on the remote side between synchronization runs, the client will be unable to resolve the situation on its own. It will create a conflict file with the local version, downloads the remote version and notifies the user that a conflict occurred which needs attention.

Example Situation

Imagine there is a file called mydata.txt your synchronized folder. It has not changed for a while and contains the text "contents" locally and remotely. Now, nearly at the same time you update it locally to say "*local contents*" while the file on the server gets updated to contain "*remote contents*" by someone else.

When attempting to upload your local changes the desktop client will notice that the server version has also changed. It creates a conflict and you will now have two files on your local machine:

- mydata.txt containing "*remote contents*"
- mydata (conflicted copy 2018-04-10 093612).txt containing "*local contents*"

In this situation the file mydata.txt has the remote changes (and will continue to be updated with further remote changes when they happen), but your local adjustments have not been sent to the server (unless the server enables conflict uploading, see below).

The desktop client notifies you of this situation via system notifications, the system tray icon and a yellow "unresolved conflicts" badge in the account settings window. Clicking this badge shows a list that includes the unresolved conflicts and clicking one of them opens an explorer window pointing at the relevant file.

To resolve this conflict, open both files, compare the differences and copy your local changes from the "conflicted copy" file into the base file where applicable. In this example you might change mydata.txt to say "local and remote contents" and delete the file with "conflicted copy" in its name. With that, the conflict is resolved.

Uploading Conflicts (experimental)

By default the conflict file (the file with "conflicted copy" in its name that contains your local conflicting changes) is not uploaded to the server. The idea is that you, the author of the changes, are the best person for resolving the conflict and showing the conflict to other users might create confusion.

However, in some scenarios it makes a lot of sense to upload these conflicting changes such that local work can become visible even if the conflict won't be resolved immediately.

In the future there might be a server-wide switch for this behavior. For now it can already be tested by setting the environment variable: OWNCLOUD_UPLOAD_CONFLICT_FILES = 1.

Automatic Updating of the Desktop Client

Introduction

The Automatic Updater ensures that you always have the latest features and bug fixes for your ownCloud Desktop synchronization client. The Automatic Updater updates only on Mac OS X and Windows computers; Linux users only need to use their normal package managers. However, on Linux systems the Updater will check for updates and notify you when a new version is available.

Basic Workflow

The following sections describe how to use the Automatic Updater on different operating systems.

Windows

The ownCloud client checks for updates and downloads them when available. You can view the update status under **Settings › General › Updates** in the ownCloud client.

If an update is available, and has been successfully downloaded, the ownCloud client starts a silent update prior to its next launch and then restarts itself. Should the silent update fail, the client offers a manual download.

Administrative privileges are required to perform the update.

Mac OS X

If a new update is available, the ownCloud client initializes a pop-up dialog to alert you of the update and requesting that you update to the latest version. Due to their use of the Sparkle frameworks, this is the default process for Mac OS X applications.

Linux

Linux distributions provide their own update tools, so ownCloud clients that use the Linux operating system do not perform any updates on their own. The client will inform you (**Settings › General › Updates**) when an update is available.

Preventing Automatic Updates

In controlled environments, such as companies or universities, you might not want to enable the auto-update mechanism, as it interferes with controlled deployment tools and policies. To address this case, it is possible to disable the auto-updater entirely. The following sections describe how to disable the auto-update mechanism for different operating systems.

Preventing Automatic Updates in Windows Environments

Users may disable automatic updates by adding this line to the [General] section of their owncloud.cfg files:

owncloud.cfg is usually located in C:\Users\<USERNAME>\AppData\Roaming\ownCloud\owncloud.cfg.

```
skipUpdateCheck=true
```

Windows administrators have more options for preventing automatic updates in Windows environments by using one of two methods. The first method allows users to override the automatic update check mechanism, whereas the second method prevents any manual overrides.

To prevent automatic updates, but allow manual overrides:

1. Edit these Registry keys:

```
a. (32-bit-Windows) HKEY_LOCAL_MACHINE\Software\ownCloud\ownCloud
b. (64-bit-Windows)
HKEY_LOCAL_MACHINE\Software\Wow6432Node\ownCloud\ownCloud
```

2. Add the key skipUpdateCheck (of type DWORD).
3. Specify a value of 1 to the machine.

To manually override this key, use the same value in HKEY_CURRENT_USER. To prevent automatic updates and disallow manual overrides:

This is the preferred method of controlling the updater behavior using Group Policies.

1. Edit this Registry key:

```
HKEY_LOCAL_MACHINE\Software\Policies\ownCloud\ownCloud
```

2. Add the key skipUpdateCheck (of type DWORD).
3. Specify a value of 1 to the machine.

Enterprise branded clients (Building Branded ownCloud Clients) have different key names, which are set in ownBrander using the Application Vendor and Application Name fields.

Your key names look like this:

```
HKEY_LOCAL_MACHINE\Software\Policies\myCompanyName\myAppName
```

Preventing Automatic Updates in Mac OS X Environments

You can disable the automatic update mechanism, in the Mac OS X operating system, by copying the file owncloud.app/Contents/Resources/deny_autoupdate_com.owncloud.desktopclient.plist to /Library/Preferences/com.owncloud.desktopclient.plist.

Preventing Automatic Updates in Linux Environments

Because the Linux client does not provide automatic updating functionality, there is no need to remove the automatic-update check. However, if you want to disable it edit your desktop client configuration file, $HOME/.config/ownCloud/owncloud.cfg, add this line to the [General] section:

```
skipUpdateCheck=true
```

Removing the Desktop Synchronization Client

Removing the Configuration File

When you remove the client - the configuration file remains on your system. If you then decide to install the client again, you won't need to re-enter the connection information.

In case you want a clean removal of the client, you manually have to delete the configuration file. The location of the configuration file is operating system dependent and can be found in the Configuration File description.

Windows Navigation Sidebar

If you have removed the Windows Synchronization Client but still have the ownCloud shortcut or symbol in the Windows Navigation Side Bar, here is how you to remove it:

1. Open regedit. (press the Windows Key and type regedit)
2. Search (CTL+F) for the name of the client, in this case `owncloud `.
3. Press F3 for next search result.
4. Look for the key System.IsPinnedToNameSpaceTree.
5. Right click on the key name, change, set to 0 instead of 1.

You have to do this twice. Once for the x64 and x32 system settings.

Once this is done, you don't need a reboot, just open and close your explorer - the sidebar is clean.

FAQ

Some Files Are Continuously Uploaded to the Server, Even When They Are Not Modified

It is possible that another program is changing the modification date of the file. If the file has an .eml extension, Windows automatically and continually changes the file, unless you remove it. \HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\PropertySystem\PropertyHandlers from the windows registry. See http://petersteier.wordpress.com/2011/10/22/windows-indexer-changes-modification-dates-of-eml-files/ for more information.

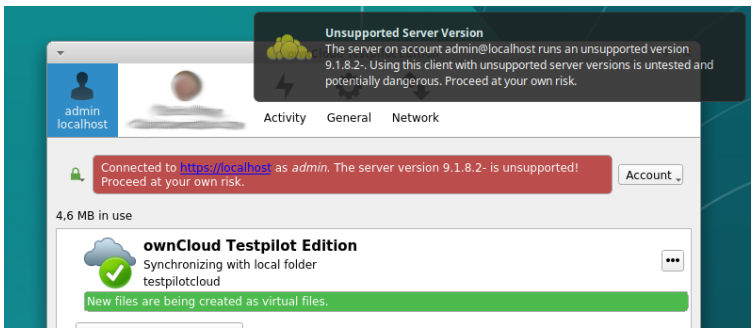
Syncing Stops When Attempting to Sync Deeper Than 100 Sub-Directories

The sync client has been intentionally limited to sync no deeper than 100 sub-directories. The hard limit exists to guard against bugs with cycles like symbolic link loops. When a deeply nested directory is excluded from synchronization it will be listed with other ignored files and directories in the "Not synced" tab of the "Activity" pane.

I See a Warning Message for Unsupported Versions

Keeping software up to date is crucial for file integrity and security – if software is outdated, there can be unfixed bugs. That's why you should always upgrade your software when there is a new version.

The ownCloud Desktop Client talks to a server, e.g. the ownCloud server, so you don't only have to upgrade your client when there is a new version for it, also the server has to be kept up-to-date by your sysadmin. Starting with version 2.5.0, the client will show a warning message if you connect to an outdated or unsupported server:



Because Earlier Versions Are Not Maintained Anymore, Only ownCloud 10.0.0 or Higher Is Supported

So if you encounter such a message, you should ask your administrator to upgrade ownCloud to a secure version. An important feature of the ownCloud Client is checksumming – each time you download or upload a file, the client and the server both check if the file was corrupted during the sync. This way you can be sure that you don't lose any files.

There are servers out there which don't have checksumming implemented on their side, or which are not tested by ownCloud's QA team. They can't ensure file integrity, they have potential security issues, and we can't guarantee that they are compatible with the ownCloud Desktop Client.

We Care About Your Data and Want It to Be Safe

That's why you see this warning message, so you can evaluate your data security. Don't worry – you can still use the client with an unsupported server, but do so at your own risk.

There Was a Warning About Changes in Synchronized Folders Not Being Tracked Reliably

On Linux, when the synchronized folder contains a high number of subfolders, the operating system may not allow for enough inotify watches to monitor the changes in all of them.

In this case the client will not be able to immediately start the synchronization process when a file in one of the unmonitored folders changes. Instead, the client will show the warning and manually scan folders for changes in a regular interval (two hours by default).

This problem can be solved by setting the fs.inotify.max_user_watches sysctl to a higher value and can usually be done either temporarily:

```
echo 524288 > /proc/sys/fs/inotify/max_user_watches.
```

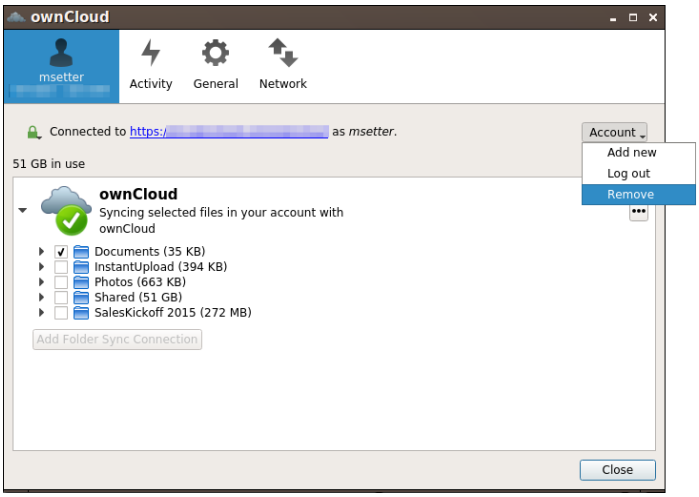
or permanently by adjusting /etc/sysctl.conf.

I Want to Move My Local Sync Folder

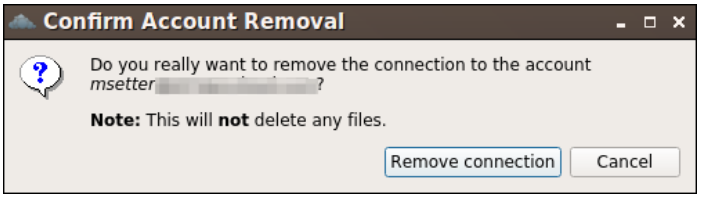
The ownCloud desktop client does not provide a way to change the local sync folder directly. However, it can be done in two ways:

1. Copy the folder and avoid a full re-sync:
   a. Stop the client and edit the localPath= line in the configuration file according your needs.
   b. Copy (or move) all your data from the current to the new location manually and start the client.
2. Create a new sync connection with a new location:
   a. Remove the existing connection which syncs to the old directory.

      To do so, in the client UI, which you can see below, click the drop down menu **Account › Remove**.



This will display a "**Confirm Account Removal**" dialog window. If you're sure, click **[Remove connection]**.



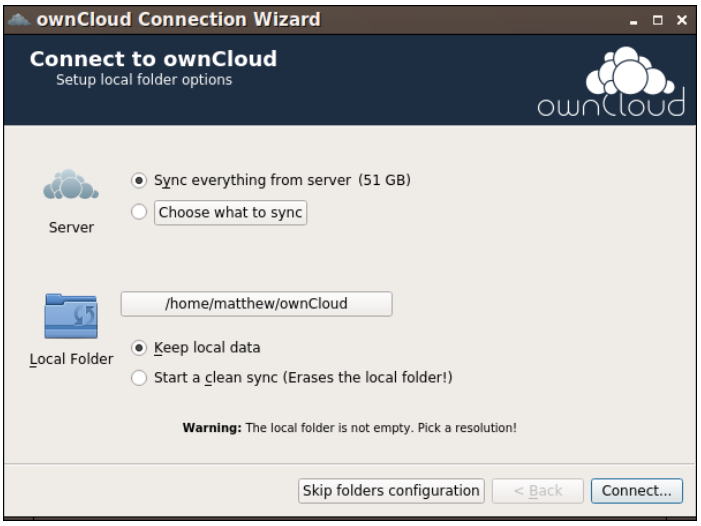
b. Add a new connection which syncs to the desired directory.

Click the drop-down menu **Account** › **Add new**.

This opens the ownCloud Connection Wizard, which you can see below, *but* with an extra option. This option provides the ability to either keep the existing data *(synced by the previous connection)* or to start a clean sync *(erasing the existing data)*.

Be careful before choosing the "Start a clean sync" option. The old sync folder *may* contain a considerable amount of data, ranging into the gigabytes or terabytes. If it does, after the client creates the new connection, it will have to download **all** of that information again.

Instead, first move or copy the old local sync folder, containing a copy of the existing files, to the new location. Then, when creating the new connection choose "*keep existing data*" instead. The ownCloud client will check the files in the newly-added sync folder and find that they match what is on the server and not need to download anything.



Make your choice and click **[Connect…]** This will then lead you through the Connection Wizard, just like when you set up the previous sync connection, but giving you the opportunity to choose a new sync directory.

My Sync Folder Displays a Different Quota Than the Web Interface

When other users share data with you, it's downloaded to the sync folder and counted as space used by the desktop client although it doesn't affect your quota for storage usage. There are more factors taken into account when calculating the quota status.

For more information, see the User Manual on quota: https://doc.owncloud.com/server/next/user_manual/files/webgui/quota.html

I Want to Change My Server URL

Since changing server urls is a potentially dangerous operation the ownCloud desktop client does not provide a user interface for this change. Typically, server url changes should be implemented by serving a permanent redirect to the new location on the old url. The client will then permanently update the server url the next time it queries the old url.

For situations where arranging for a redirect is impossible, url changes can be done by editing the config file. Before doing so make sure that the new url does indeed point to the same server, with the same users and the same data. Then go through these steps:

1. Shut down the ownCloud client.
2. Locate the configuration file
3. Open it with a text editor.
4. Find your old server url and adjust it.
5. Save the file and start the ownCloud client again.

Advanced Usage

In this section, you find information about advanced usage.

Command Line Options

Introduction

Depending on your operating system, you can start the ownCloud client from the command line by typing owncloud or owncloud.exe. You may need to change to the directory of the binary first. When starting owncloud manually, you can add options to this command.

List Options

To get the list of options, run the following example command:

owncloud -h
or
owncloud --help

Use Options

Option	Description
--logwindow	Opens a window displaying log output.
-s --showsettings	Show the settings dialog while starting
-q --quit	Quit the running instance
--logfile <filename>	Write log output to the file specified. To write to stdout, specify - as the filename.
--logdir <name>	Writes each synchronization log output in a new file in the specified directory.
--logexpire <hours>	Removes logs older than the value specified (in hours). This command is used with --logdir.
--logflush	Clears (flushes) the log file after each write action.
--logdebug	Also output debug-level messages in the log equivalent to setting the environment variable QT_LOGGING_RULES = "qt.=*true;*.debug=true".
--confdir <dirname>	Uses the specified configuration directory.

Configuration File

Introduction

The ownCloud Client uses a configuration file. It has several sections for particular settings. You will find more sections in the configuration file than described here. Do not change any of those settings except support advises you to do so.

Location of the Configuration File

The location of the configuration file depends on the operating system used. You can locate this configuration file as follows:

System	Location
Linux	$HOME/.config/ownCloud/owncloud.cfg
Microsoft Windows	%APPDATA%\ownCloud\owncloud.cfg
macOS	$HOME/Library/Preferences/ownCloud/owncloud.cfg

The configuration file contains settings using the Microsoft Windows .ini file format. You can overwrite changes using the ownCloud configuration dialog.

Use caution when making changes to the ownCloud Client configuration file. Incorrect settings can produce unintended results.

Section [ownCloud]

Variable	Default	Meaning
remotePollInterval	30000	Specifies the poll time for the remote repository in milliseconds.
forceSyncInterval	7200000	The duration of no activity after which a synchronization run shall be triggered automatically.
fullLocalDiscoveryInterval	3600000	The interval after which the next synchronization will perform a full local discovery.
notificationRefreshInterval	300000	Specifies the default interval of checking for new server notifications in milliseconds.

Section [General]

Variable	Default	Meaning
chunkSize	10000000 (or 10 MB)	Specifies the initial chunk size of uploaded files in bytes. The client will dynamically adjust this size within the maximum and minimum bounds (see below).
maxChunkSize	100000000 (or 100 MB)	Specifies the maximum chunk size of uploaded files in bytes.
minChunkSize	1000000 (or 1 MB)	Specifies the minimum chunk size of uploaded files in bytes.
targetChunkUploadDuration	60000 (1 minute)	Target duration in milliseconds for chunk uploads. The client adjusts the chunk size until each chunk upload takes approximately this long. Set to 0 to disable dynamic chunk sizing.
promptDeleteAllFiles	true	If a UI prompt should ask for confirmation if it was detected that all files and folders were deleted.
crashReporter	true	Whether to show the crash reporter when a crash occurs.
timeout	300	The timeout for network connections in seconds.

Variable	Default	Meaning
moveToTrash	false	If non-locally deleted files should be moved to trash instead of deleting them completely. This option only works on linux
showExperimentalOptions	false	Whether to show experimental options that are still undergoing testing in the user interface. Turning this on does not enable experimental behavior on its own. It does enable user interface options that can be used to opt in to experimental features.

Section [Proxy]

Variable	Default	Meaning
host	127.0.0.1	The address of the proxy server.
port	8080	The port were the proxy is listening.
type	2	* 0 for System Proxy * 1 for SOCKS5 Proxy * 2 for No Proxy * 3 for HTTP(S) Proxy

Environment Variables

Introduction

The behavior of the client can also be controlled using environment variables.

The value of the environment variables override the values in the configuration file.

Most environment variables only exist for debugging or testing. They are not officially supported and may change from version to version. If you end up relying on a setting only available through an environment variable, please create a bug report.

Available Environment Variables

Setting	Default	Description
OWNCLOUD_CHUNK_SIZE	10000000 (or 10 MB)	Specifies the initial chunk size of uploaded files in bytes. The client will dynamically adjust this size within the maximum and minimum bounds (see below). To disable chunking completely, set OWNCLOUD_CHUNK_SIZE=0 .
OWNCLOUD_MAX_CHUNK_SIZE	100000000 (or 100 MB)	Specifies the maximum chunk size of uploaded files in bytes.
OWNCLOUD_MIN_CHUNK_SIZE	1000000 (or 1 MB)	Specifies the minimum chunk size of uploaded files in bytes.

Setting	Default	Description
OWNCLOUD_TARGET_CHUNK_UPLOAD_DURATION	60000	Target duration in milliseconds for chunk uploads. The client adjusts the chunk size until each chunk upload takes approximately this long. Set to 0 to disable dynamic chunk sizing.
OWNCLOUD_CHUNKING_NG	depend on server capability	Force-enable ("1") or force-disable ("0") the NG chunking algorithm.
OWNCLOUD_NO_TUS		Set to any value to disable uploads using the tus protocol
OWNCLOUD_TIMEOUT	300	The timeout for network connections in seconds.
OWNCLOUD_CRITICAL_FREE_SPACE_BYTES	50*1000*1000 bytes	The minimum disk space needed for operation. A fatal error is raised if less free space is available.
OWNCLOUD_FREE_SPACE_BYTES	250*1000*1000 bytes	Downloads that would reduce the free space below this value are skipped. More information available under the "Low Disk Space" section.
OWNCLOUD_MAX_PARALLEL	6	Maximum number of parallel jobs.
OWNCLOUD_BLACKLIST_TIME_MIN	25	Minimum timeout, in seconds, for blacklisted files.
OWNCLOUD_BLACKLIST_TIME_MAX	24*60*60 (or one day)	Maximum timeout, in seconds, for blacklisted files.
OWNCLOUD_HTTP2_ENABLED	depend on Qt version	Force-enable ("1") or force-disable ("0") HTTP2 support. Note that HTTP2 use also depends on whether the server supports it.
OWNCLOUD_MINIMAL_TRAY_MENU	unset	If set a minimal tray menu is used. Helpful if a platform’s tray has problematic behavior.
OWNCLOUD_TRAY_UPDATE_WHILE_VISIBLE	0	Set to "1" to allow the tray menu to be updated while it’s visible to the user.
OWNCLOUD_FORCE_TRAY_SHOW_HIDE	unset	Set to "1" to reestablish the tray icon every time the menu changes.
OWNCLOUD_FORCE_TRAY_FAKE_DOUBLE_CLICK	unset	Set to "1" if single tray clicks sometimes get recognized as double clicks.
OWNCLOUD_FORCE_TRAY_MANUAL_VISIBILITY	unset	Set to "1" if the tray menu is flickering while opened.
OWNCLOUD_FORCE_TRAY_NO_ABOUT_TO_SHOW	unset	Set to "1" if the tray menu sometimes contains stale entries.
OWNCLOUD_FULL_LOCAL_DISCOVERY_INTERVAL	3600000 (1 hour)	Maximum time in milliseconds that fast local discovery is allowed for after a full local discovery. Set to 0 to always require full local discovery. Set to -1 to never require full local discovery.

Setting	Default	Description
OWNCLOUD_SQLITE_JOURNAL_MODE	depends on filesystem	Set a specific sqlite journal mode.
OWNCLOUD_SQLITE_LOCKING_MODE	EXCLUSIVE	Set a specific sqlite locking mode.
OWNCLOUD_SQLITE_TEMP_STORE	unset	Set the given temp_store on the sqlite database.
OWNCLOUD_DISABLE_CHECKSUM_COMPUTATIONS	unset	Set to disable all file checksum computations.
OWNCLOUD_DISABLE_CHECKSUM_UPLOAD	unset	Set to disable computing checksums for uploaded files.
OWNCLOUD_CONTENT_CHECKSUM_TYPE	SHA1	Select the file checksumming algorithm. "Adler32", "MD5", "SHA1", "SHA256", "SHA3-256" are valid, but not all have server support.
OWNCLOUD_UPLOAD_CONFLICT_FILES	unset	Set to "1" to enable uploading conflict files to the server.
QT_LOGGING_RULES	unset	Set to "sync.httplogger=true" to enable verbose http logging. See also troubleshooting.adoc for more.

The Command Line Client

Introduction

The ownCloud Client packages contain a command line client, owncloudcmd, that can be used to synchronize ownCloud files to client machines.

owncloudcmd performs a single *sync run* and then exits the synchronization process. In this manner, owncloudcmd processes the differences between client and server directories and propagates the files to bring both repositories to the same state. Contrary to the GUI-based client, owncloudcmd does not repeat synchronizations on its own. It also does not monitor for file system changes.

To invoke owncloudcmd, you must provide the local and the remote repository URL using the following command:

```
owncloudcmd [OPTIONS...] sourcedir owncloudurl.
```

sourcedir is the local directory and owncloudurl is the server URL. Other command line switches supported by owncloudcmd include the following:

Switch	Description
--user, -u [user]	Use user as the login name.
--password, -p [password]	Use password as the password.
-n	Use netrc (5) for login.
--non-interactive	Do not prompt for questions.
--silent, --s	Inhibits verbose log output.

Switch	Description
--trust	Trust any SSL certificate, including invalid ones.
--httpproxy http://[user@pass:]<server>:<port>	Uses server as HTTP proxy.
--davpath [path]	Overrides the WebDAV Path with path
--exclude [file]	Exclude list file.
--unsyncedfolders [file]	File containing the list of un-synced remote folders (selective sync)
--max-sync-retries [n]	Retries maximum n times (defaults to 3)
-h	Sync hidden files,do not ignore them.

Credential Handling

owncloudcmd requires the user to specify the username and password using the standard URL pattern, for example:

```
$ owncloudcmd /home/user/my_sync_folder
https://carla:secret@server/owncloud/remote.php/webdav/
```

To synchronize the ownCloud directory Music to the local directory media/music, through a proxy listening on port 8080, and on a gateway machine using IP address 192.168.178.1, the command line would be:

```
$ owncloudcmd --httpproxy http://192.168.178.1:8080 \
        $HOME/media/music \
        https://server/owncloud/remote.php/webdav/Music.
```

owncloudcmd will prompt for the user name and password, unless they have been specified on the command line or -n has been passed.

Exclude List

owncloudcmd requires access to an exclude list file. It must either be installed along with owncloudcmd and thus be available in a system location, be placed next to the binary as sync-exclude.lst or be explicitly specified with the --exclude switch.

Low Disk Space

Introduction

When disk space is low, the ownCloud Client will be unable to synchronize all files. This section describes its behavior in a low disk space situation as well as the adjustable environment variables that influence it.

Cases

Issue	Adjustable Environment Variable
Synchronization of a folder aborts entirely if the remaining disk space falls below 50 MB.	OWNCLOUD_CRITICAL_FREE_SPACE_BYTES
Downloads that would reduce the free disk space below 250 MB will be skipped or aborted. The download will be retried regularly and other synchronization is unaffected.	OWNCLOUD_FREE_SPACE_BYTES

Appendices

In this section, you find supporting information.

Appendix Building the Client

Introduction

This section explains how to build the ownCloud Client from source for all major platforms. You should read this section if you want to develop for the desktop client. Build instructions are subject to change as development proceeds.

Please check the version for which you want to build.

These instructions are updated to work with the latest version of the ownCloud Client.

Getting the Source Code

The generic build instructions pull the latest code directly from GitHub, and work on Linux, Mac OS X, and Windows.

Linux

For the published desktop clients we link against QT5 dependencies from our own repositories so that we can have the same versions on all distributions. This chapter shows you how to build the client yourself with this setup. If you want to use the QT5 dependencies from your system, see the next chapter.

You may wish to use source packages for your Linux distribution, as these give you the exact sources from which the binary packages are built. These are hosted on the ownCloud repository from OBS. Go to the Index of repositories to see all the Linux client repositories.

To get the .deb source packages, add the source repository for your Debian or Ubuntu version, as in the following example for Debian 9:

```
# Run as root
echo 'deb
http://download.opensuse.org/repositories/isv:/ownCloud:/desktop/D
ebian_9.0/ /' >> /etc/apt/sources.list.d/owncloud-client.list
echo 'deb-src
http://download.opensuse.org/repositories/isv:/ownCloud:/desktop/D
ebian_9.0/ /' >> /etc/apt/sources.list.d/owncloud-client.list
```

The above registers the source repository of the released client. There is also .../desktop:/testing/... and e.g. .../desktop:/daily:/2.7/... for beta versions or daily snapshots.

Install the dependencies using the following commands for your specific Linux distribution. Make sure the repositories for source packages are enabled. These are:

Distribution	Installation Instructions
Debian/Ubuntu	apt update; apt build-dep owncloud-client

Distribution	Installation Instructions
openSUSE/SLES	zypper ref; zypper si -d owncloud-client
Fedora/CentOS/RHEL	yum install yum-utils; yum-builddep owncloud-client

Follow the generic build instructions, starting with step 2.

Linux with System Dependencies

Build sources from a GitHub checkout with dependencies provided by your Linux distribution. While this allows more freedom for development, it does not exactly represent what we ship as packages. See above for how to recreate packages from source.

To get the source dependencies on Debian and Ubuntu, run the following command:

```
sudo apt install qtdeclarative5-dev libinotifytools-dev \
  qt5keychain-dev python3-sphinx \
  libsqlite3-dev
```

Follow the generic build instructions, starting with step 1.

macOS

In addition to needing Xcode (along with the command line tools), developing in the macOS environment requires extra dependencies. You can install these dependencies through MacPorts or Homebrew. These dependencies are required only on the build machine, because non-standard libs are deployed in the app bundle.

The tested and preferred way to develop in this environment is through the use of HomeBrew. The ownCloud team has its own repository containing non-standard recipes. To set up your build environment for development using HomeBrew:

1. Install Xcode.
2. Install Xcode command line tools using

   ```
   xcode-select --install
   ```

3. Install Homebrew using

   ```
   /bin/bash -c "$(curl -fsSL
   https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
   ```

4. Add the ownCloud repository using the command

   ```
   brew tap owncloud/owncloud
   ```

5. Install a Qt5 version, ideally from from 5.10.1, using the command

```
brew install qt5
```

6. Install any missing dependencies, using the command:

```
brew install $(brew deps owncloud-client)
```

7. Install qtkeychain by running

```
git clone https://github.com/frankosterfeld/qtkeychain.git
```

Make sure you make the same install prefix as later while building the client e.g.

```
-DCMAKE_INSTALL_PREFIX=/Path/to/client/../install
```

8. For compilation of the client, follow the generic build instructions.
9. Install the Packages package creation tool.
10. In the build directory, run

```
admin/osx/create_mac.sh <CMAKE_INSTALL_DIR> <build dir> <installer sign identity>
```

If you have a developer signing certificate, you can specify its Common Name as a third parameter (use quotes) to have the package signed automatically.

Contrary to earlier versions, version 1.7 and later are packaged as a pkg installer. Do not call make package at any time when compiling for OS X, as this will build a disk image, which will not work correctly.

Windows Development Build with KDE Craft

If you want to test some changes, you can build the ownCloud Client natively on Windows using KDE Craft. You can also use it to build unsupported and unoptimized installers.

Install KDE Craft

To install KDE Craft, Python 2.7 or Python 3.6+, and PowerShell 5.0+ must be installed. You can find the full installation guide in the KDE Community Wiki.

If you want to use Microsoft Visual Studio, naturally, that must be installed as well.

When the dependencies are installed, install KDE Craft using the following lines in PowerShell:

```
Set-ExecutionPolicy -Scope CurrentUser RemoteSigned
iex ((new-object net.webclient).DownloadString
('https://raw.githubusercontent.com/KDE/craft/master/setup/install_craft.ps1'))
```

The first command allows running scripts from remote sources. The second command starts installing KDE Craft. You are asked where you want to put the main folder, called CraftRoot, which will contain all source, build, and install folders. Please chose a disk with sufficient free space.

Last but not least, you need to chose the compiler you want to use. The official builds only supports Microsoft Visual Studio 2019. However, if you're feeling adventurous, you can also try to use Mingw-w64. In contrast to Visual Studio, which you need to install in advance, KDE Craft can install Mingw-w64 for you.

Unless you need 32bit builds, you should stick to the default of x64 builds.

Setup KDE Craft

After you install KDE Craft, there are two steps left before the ownCloud Client can be compiled. These are:

1. Launch the KDE Craft Environment
2. Build the Client

Launch the KDE Craft Environment

To launch the KDE Craft environment, you need to run the following command in PowerShell. This provides you with a shell with all the environment variables set that you need to work with KDE Craft.

```
C:\CraftRoot\craft\craftenv.ps1
```

This needs to be done every time you want to work with Craft.

We're assuming that you installed KDE Craft in the default path of C:\CraftRoot. If you have installed it somewhere else, please adjust the path as necessary.

Setup the ownCloud repository

The last step before we can begin, is adding the ownCloud repository. It provides you with additional dependencies and tools, which are not available from the standard KDE repository.

```
craft --add-blueprint-repository https://github.com/owncloud/craft-blueprints-
owncloud.git
```

You only need to do this once.

Build The Client

Finally we can build the client with the following command:

```
craft owncloud-client
```

This installs all required dependencies and builds the ownCloud Client from the master git branch. If you want to build a different branch, first install all dependencies and then clone the source code from git, like this:

```
craft --install-deps owncloud-client
craft --fetch owncloud-client
```

You can find the git checkout in C:\CraftRoot\downloads\git\owncloud\owncloud-client. There you can use the usual git commands to switch branches and remotes, e.g., to build the 2.8 stable branch you can use craft with --set version parameter:

```
git checkout 2.8
craft --set version=2.8 owncloud-client
```

Afterwards you can build the client like this:

```
craft --configure --make --install
craft owncloud-client
```

Run the Client

Neither craft owncloud-client nor craft --configure --make --install make the ownCloud Client available in your PATH, they only install to the so-called image directory. This is so KDE Craft knows which files belong to which package. In order to run the client, you first need to merge the image directory to the regular KDE Craft root (C:\CraftRoot). Afterwards, you can run owncloud.exe from your shell.

```
craft --qmerge owncloud-client
owncloud.exe
```

Package the Client (Unsupported)

Although this is not officially supported, it is, generally, possible to build an installer with:

```
craft nsis
craft --package owncloud-client
```

Now you should have a file called: owncloud-client-master-${COMMIT_HASH}-windows-${COMPILER}.exe in C:\CraftRoot\tmp.

This is not supported, optimised, nor regularly tested! Fully supported Windows installers are currently only provided by ownBrander.

Generic Build Instructions

To build the most up-to-date version of the client:

1. Clone the latest versions of the client from Git as follows:

```
git clone git://github.com/owncloud/client.git
cd client
# master this default, but you can also check out a tag like v2.5.4
git checkout master
git submodule init
git submodule update
```

2. Create the build directory:

```
mkdir client-build
cd client-build
```

3. Configure the client build:

```
cmake -DCMAKE_PREFIX_PATH=/opt/ownCloud/qt-5.12.4
-DCMAKE_INSTALL_PREFIX=/Users/path/to/client/../install/ ..
```

   For Linux builds (using QT5 libraries via build-dep) a typical setting is

```
-DCMAKE_PREFIX_PATH=/opt/ownCloud/qt-5.12.4/
```

   However, the version number may vary. For Linux builds using system dependencies -DCMAKE_PREFIX_PATH is not needed. You must use absolute paths for the include and library directories.

   On Mac OS X, you need to specify -DCMAKE_INSTALL_PREFIX=target, where target is a private location, i.e. in parallel to your build dir by specifying ../install.

   qtkeychain must be compiled with the same prefix e.g.,

```
-DCMAKE_INSTALL_PREFIX=/Users/path/to/client/../install/
```

4. Call

```
make
```

   The ownCloud binary will appear in the bin directory.

5. (Optional) Call make install to install the client to the /usr/local/bin directory (or as per CMAKE_INSTALL_PREFIX).
   The following are known CMake parameters:

- QTKEYCHAIN_LIBRARY=/path/to/qtkeychain.dylib -DQTKEYCHAIN_INCLUDE_DIR=/path/to/qtkeychain/ Used for stored credentials. When compiling with Qt5, the library is called qt5keychain.dylib. You need to compile QtKeychain with the same Qt version. If you install QtKeychain into the CMAKE_PREFIX_PATH then you don't need to specify the path manually.
- WITH_DOC=TRUE: Creates doc and man pages through running make; also adds install statements, providing the ability to install using make install.
- CMAKE_PREFIX_PATH=/path/to/Qt5.12.4/5.12.4/yourarch/lib/cmake/: Builds using that Qt version.
- CMAKE_INSTALL_PREFIX=path: Set an install prefix. This is mandatory on Mac OS.

6. **Optional:** Run a client that was installed in a custom CMAKE_INSTALL_PREFIX may not pick up the correct libraries automatically. You can use LD_LIBRARY_PATH to help finding the libraries like this:

```
LD_LIBRARY_PATH=/opt/ownCloud/qt-5.12.4/lib/x86_64-linux-
gnu/:/Users/path/to/client/../install/lib/x86_64-linux-gnu/
/Users/path/to/client/../install/bin/owncloud
```

Compiling via ownBrander

If you don't want to go through the trouble of doing all the compile work manually, you can use ownBrander to create installer images for all platforms.

Appendix History and Architecture

Introduction

ownCloud provides desktop sync clients to synchronize the contents of local directories from computers, tablets, and handheld devices to the ownCloud server.

Synchronization is accomplished using csync, a bidirectional file synchronizing tool that provides both a command line client as well as a library. A special module for csync was written to synchronize with the ownCloud built-in WebDAV server.

The ownCloud Client software is written in C++ using the Qt Framework As a result, the ownCloud Client runs on Linux, Windows, and MacOS.

The Synchronization Process

The process of synchronization keeps files in two separate repositories the same. When synchronized:

- If a file is added to one repository it is copied to the other synchronized repository.
- When a file is changed in one repository, the change is propagated to any other synchronized repository.
- If a file is deleted in one repository, it is deleted in any other.

It is important to note that the ownCloud synchronization process does not use a typical client/server system where the server is always master. This is a major difference between the ownCloud synchronization process and other systems like a file backup, where only changes to files or folders and the addition of new files are propagated, but these files and folders are never deleted unless explicitly deleted in the backup.

During synchronization, the ownCloud Client checks both repositories for changes frequently. This process is referred to as a *sync run*. In between sync runs, the local repository is monitored by a file system monitoring process that starts a sync run immediately if something was edited, added, or removed.

Synchronization by Time versus ETag

Until the release of ownCloud 4.5 and ownCloud Client 1.1, the ownCloud synchronization process employed a single file property — the file modification time — to decide which file was newer and needed to be synchronized to the other repository.

The *modification timestamp* is part of the files metadata. It is available on every relevant filesystem and is the typical indicator for a file change. Modification timestamps do not require special action to create, and have a general meaning. One design goal of csync is to not require a special server component. This design goal is why csync was chosen as the backend component.

To compare the modification times of two files from different systems, csync must operate on the same base. Before ownCloud Client version. 1.1.0, csync required both device repositories to run on the exact same time. This requirement was achieved through the use of enterprise standard NTP time synchronization on all machines.

Because this timing strategy is rather fragile without the use of NTP, ownCloud 4.5 introduced a unique number (for each file?) that changes whenever the file changes. Although this number is a unique value, it is not a hash of the file. Instead, it is a randomly chosen number, that is transmitted in the Etag field. Because the file number changes if the file changes, its use is guaranteed to determine if one of the files has changed and, thereby, launching a synchronization process.

ownCloud Client release 1.1 and later requires file ID capabilities on the ownCloud server. Servers that run with release earlier than 4.5.0 do not support using the file ID functionality.

Before the 1.3.0 release of the Desktop Client, the synchronization process might create false conflict files if time deviates. Original and changed files conflict only in their timestamp, but not in their content. This behavior was changed to employ a binary check if files differ.

Like files, directories also hold a unique ID that changes whenever one of the contained files or directories is modified. Because this is a recursive process, it significantly reduces the effort required for a synchronization cycle, because the client only analyzes directories with a modified ID.

The following table outlines the different synchronization methods used, depending on server/client combination:

Table 1. Compatibility Table

Server Version	Client Version	Sync Methods
4.0.x or earlier	1.0.5 or earlier	Time Stamp
4.0.x or earlier	1.1 or later	n/a (incompatible)
4.5 or later	1.0.5 or earlier	Time Stamp
4.5 or later	1.1 or later	File ID, Time Stamp

We strongly recommend using ownCloud Server release 4.5 or later when using ownCloud Client 1.1 or later. Using an incompatible time stamp-based synchronization

mechanism can lead to data loss in rare cases, especially when multiple clients are involved and one utilizes a non-synchronized NTP time.

Comparison and Conflict Cases

As mentioned above, during a *sync run* the client must first detect if one of the two repositories have changed files. On the local repository, the client traverses the file tree and compares the modification time of each file with an expected value stored in its database. If the value is not the same, the client determines that the file has been modified in the local repository.

On the local side, the modification time is a good attribute to use for detecting changes, because the value does not depend on time shifts and such.

For the remote (that is, ownCloud server) repository, the client compares the ETag of each file with its expected value. Again, the expected ETag value is queried from the client database. If the ETag is the same, the file has not changed and no synchronization occurs.

In the event a file has changed on both the local and the remote repository since the last sync run, it can not easily be decided which version of the file is the one that should be used. However, changes to any side will not be lost. Instead, a *conflict case* is created. The client resolves this conflict by renaming the local file, appending a conflict label and timestamp, and saving the remote file under the original file name.

Example: Assume there is a conflict in message.txt because its contents have changed both locally and remotely since the last sync run. The local file with the local changes will be renamed to message. (conflicted copy 2016-01-01 153110).txt and the remote file will be downloaded and saved as message.txt.

Conflict files are always created on the client and never on the server.

Checksum Algorithm Negotiation

In ownCloud 10.0 we implemented a checksum feature which checks the file integrity on upload and download by computing a checksum after the file transfer finishes. The client queries the server capabilities after login to decide which checksum algorithm to use. Currently, SHA1 is hard-coded in the official server release and can't be changed by the end-user. Note that the server additionally also supports **MD5** and **Adler-32**, but the desktop client will always use the checksum algorithm announced in the capabilities:

```
GET http://localhost:8000/ocs/v1.php/cloud/capabilities?format=json
```

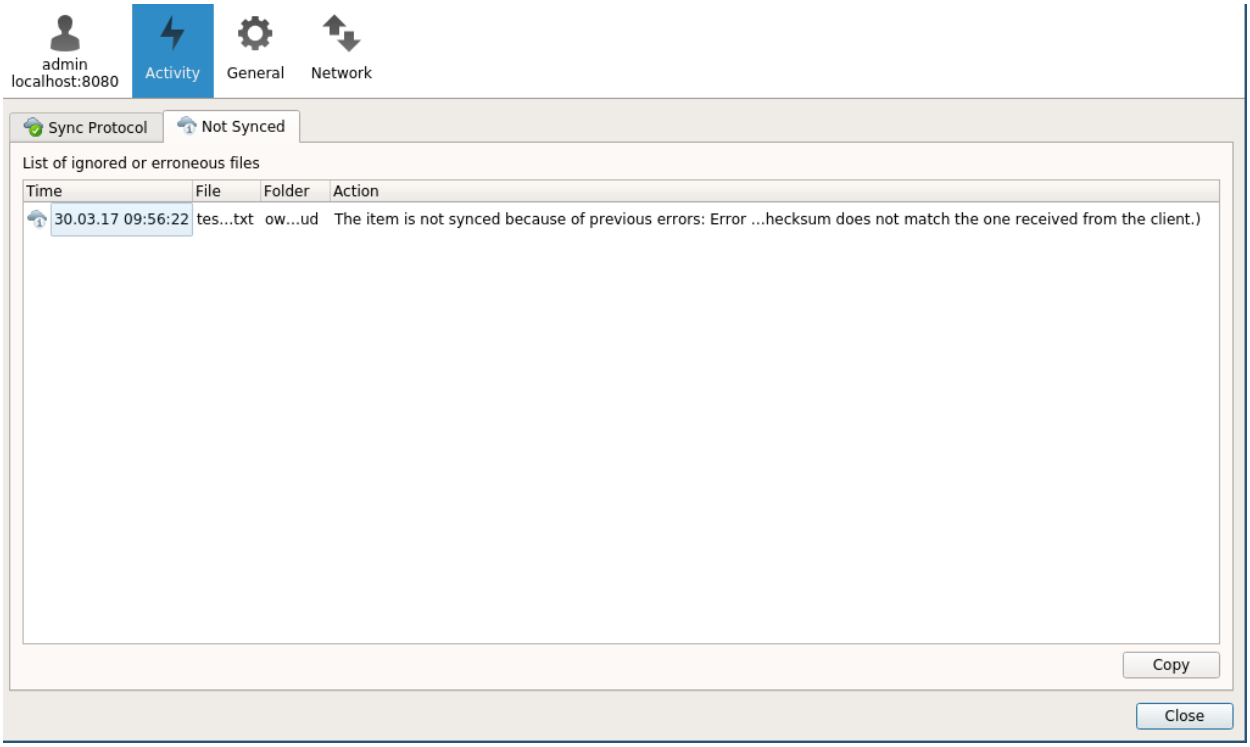

```
{
  "ocs":{
    "meta":{
      "status":"ok",
      "statuscode":100,
      "message":"OK",
      "totalitems":"",
      "itemsperpage":""
    },
    "data":{
```

```
    "version":{
      "major":10,
      "minor":0,
      "micro":0,
      "string":"10.0.0 beta",
      "edition":"Community"
    },
    "capabilities":{
      "core":{
        "pollinterval":60,
        "webdav-root":"remote.php/webdav"
      },
      "dav":{
        "chunking":"1.0"
      },
      "files_sharing":{
        "api_enabled":true,
        "public":{
          "enabled":true,
          "password":{
            "enforced":false
          },
          "expire_date":{
            "enabled":false
          },
          "send_mail":false,
          "upload":true
        },
        "user":{
          "send_mail":false
        },
        "resharing":true,
        "group_sharing":true,
        "federation":{
          "outgoing":true,
          "incoming":true
        }
      },
      "checksums":{
        "supportedTypes":[
          "SHA1"
        ],
        "preferredUploadType":"SHA1"
      },
      "files":{
        "bigfilechunking":true,
        "blacklisted_files":[
          ".htaccess"
        ],
        "undelete":true,
```

```
            "versioning":true
          }
        }
      }
    }
  }
```

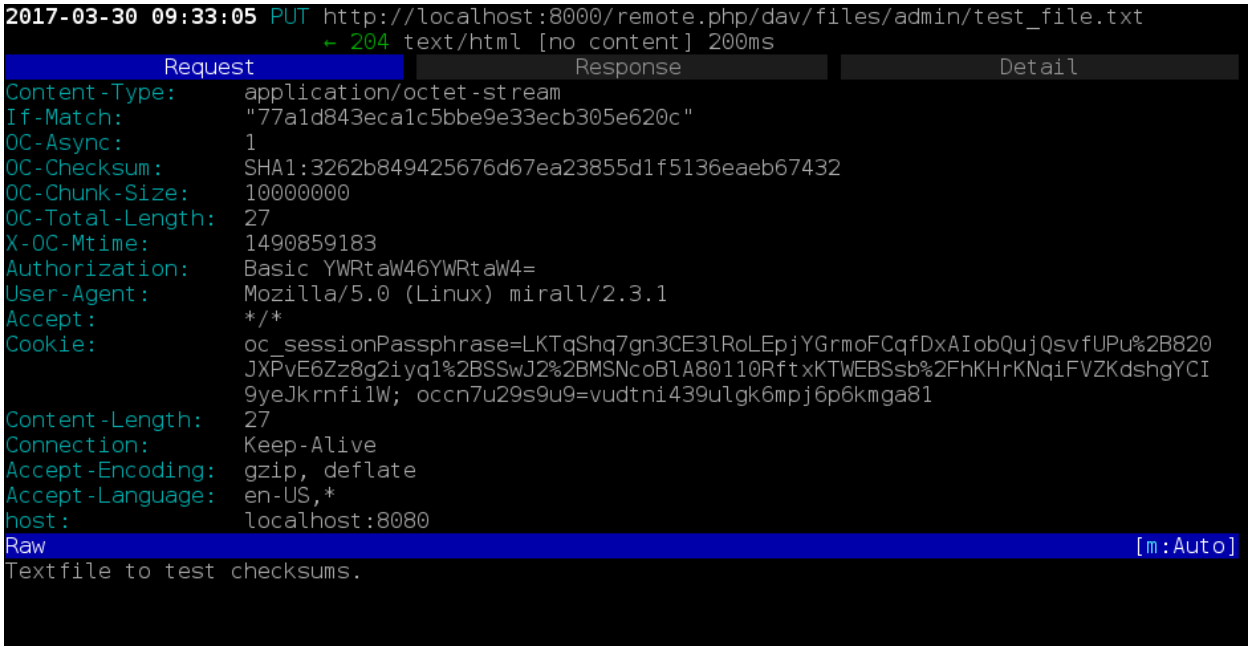
Upload

A checksum is calculated with the previously negotiated algorithm by the client and sent along with the file in an HTTP Header: OC-Checksum: [algorithm]:[checksum].



During file upload, the server computes SHA1, MD5, and Adler-32 checksums and compares one of them to the checksum supplied by the client.

On mismatch, the server returns HTTP Status code 400 (Bad Request) thus signaling the client that the upload failed. The server then discards the upload, and the client blacklists the file:




```
<?xml version='1.0' encoding='utf-8'?>
<d:error xmlns:d="DAV:" xmlns:s="http://sabredav.org/ns">
  <s:exception>Sabre\DAV\Exception\BadRequest</s:exception>
  <s:message>The computed checksum does not match the one received from the
client.</s:message>
</d:error>
```

The client retries the upload using exponential back-off. On success, (matching checksum) the computed checksums are stored by the server in oc_filecache alongside the file.

Chunked Upload

Mostly same as above. The checksum of the full file is sent with every chunk of the file. But the server only compares the checksum after receiving the checksum sent with the last chunk.

Download

The server sends the checksum in an HTTP header with the file. (same format as above) If no checksum is found in oc_filecache (freshly mounted external storage) it is computed and stored in oc_filecache on the first download. The checksum is then provided on all subsequent downloads but not on the first.

Ignored Files

The ownCloud Client supports the ability to exclude or ignore certain files from the synchronization process. Some system wide file patterns that are used to exclude or ignore files are included with the client by default and the ownCloud Client provides the ability to add custom patterns.

By default, the ownCloud Client ignores the following files:

- Files matched by one of the patterns defined in the Ignored Files Editor.
- Files starting with .*sync*.db*, .sync_*.db*, .csync_journal.db*, .owncloudsync.log*, as these files are reserved for journalling.
- Files with a name longer than 254 characters.

- The file Desktop.ini in the root of a synced folder.
- Files matching the pattern _conflict- unless conflict file uploading is enabled.
- Files matching the pattern (conflicted copy unless conflict file uploading is enabled.
- Windows only: Files containing characters that do not work on typical Windows filesystems (\, /, :, ?, *, ", >, <, |).
- Windows only: Files with a trailing space or dot.
- Windows only: Filenames that are reserved on Windows.

If a pattern selected using a checkbox in the Ignored Files Editor, or if a line in the exclude file starts with the character] directly followed by the file pattern, files matching the pattern are considered.

fleeting meta data.

These files are ignored and *removed* by the client if found in the synchronized folder. This is suitable for meta files created by some applications that have no sustainable meaning.

If a pattern ends with the forward slash (/) character, only directories are matched. The pattern is only applied for directory components of filenames selected using the checkbox.

To match filenames against the exclude patterns, the UNIX standard C library function fnmatch is used. This process checks the filename against the specified pattern using standard shell wildcard pattern matching. For more information, please refer to the pattern matching documentation.

The path that is checked is the relative path under the sync root directory.

Pattern and File Match Examples:.

Pattern	File Matches
~$*	~$foo, ~$example.doc
fl?p	flip, flap
moo/	map/moo/, moo/

The Sync Journal

The client stores the ETag number in a per-directory database, called the *journal*. This database is a hidden file contained in the directory to be synchronized.

If the journal database is removed, the ownCloud Client CSync backend rebuilds the database by comparing the files and their modification times. This process ensures that both server and client are synchronized using the appropriate NTP time before restarting the client following a database removal.

Custom WebDAV Properties

In the communication between client and server a couple of custom WebDAV properties were introduced. They are either needed for sync functionality or help have a positive effect on synchronization performance.

This chapter describes additional XML elements which the server returns in response to a successful PROPFIND request on a file or directory. The elements are returned in the namespace oc.

Server Side Permissions

The XML element <oc:permissions> represents the permission- and sharing state of the item. It is a list of characters, and each of the chars has a meaning as outlined in the table below:

Code	Resource	Description
S	File or Folder	is shared.
R	File or Folder	can share (includes re-share)
M	File or Folder	is mounted (like on Dropbox, Samba, etc.)
W	File	can write file.
C	Folder	can create file in folder.
K	Folder	can create folder (mkdir)
D	File or Folder	can delete file or folder.
N	File or Folder	can rename file or folder.
V	File or Folder	can move file or folder.

Example:

```
<oc:permissions>RDNVCK</oc:permissions>
```

File- or Directory Size

The XML element <oc:size> represents the file- or directory size in bytes. For directories, the size of the whole file tree underneath the directory is accumulated.

Example:

```
<oc:size>2429176697</oc:size>
```

FileID

The XML element <oc:id> represents the so called file ID. It is a non volatile string id that stays constant as long as the file exists. It is not changed if the file changes or is renamed or moved.

Example:

```
<oc:id>00000020oc5cfy6qqizm</oc:id>
```

Appendix Troubleshooting

Introduction

The following two general issues can result in failed synchronization:

- The server setup is incorrect.

- The client contains a bug.

When reporting bugs, it is helpful if you first determine what part of the system is causing the issue.

Identifying Basic Functionality Problems

Performing a general ownCloud Server test

The first step in troubleshooting synchronization issues is to verify that you can log on to the ownCloud web application. To verify connectivity to the ownCloud server try logging in via your Web browser. If you are not prompted for your username and password, or if a red warning box appears on the page, your server setup requires modification. Please verify that your server installation is working correctly.

Ensure the WebDAV API is working

If all desktop clients fail to connect to the ownCloud Server, but access using the Web interface functions properly, the problem is often a misconfiguration of the WebDAV API. The ownCloud Client uses the built-in WebDAV access of the server content. Verify that you can log on to ownCloud's WebDAV server. To verify connectivity with the ownCloud WebDAV server, open a browser window and enter the address to the ownCloud WebDAV server. For example, if your ownCloud instance is installed at https://yourserver.com/owncloud, your WebDAV server address is https://yourserver.com/owncloud/remote.php/webdav. If you are prompted for your username and password but, after providing the correct credentials, authentication fails, please ensure that your authentication backend is configured properly.

Use a WebDAV command line tool to test

A more sophisticated test method for troubleshooting synchronization issues is to use a WebDAV command line client and log into the ownCloud WebDAV server. One such command line client — called cadaver — is available for Linux distributions. You can use this application to further verify that the WebDAV server is running properly using PROPFIND calls. As an example, after installing the cadaver app, you can issue the propget command to obtain various properties pertaining to the current directory and also verify WebDAV server connection.

CSync Unknown Error

If you see this error message stop your client, delete the ._sync_xxxxxxx.db file, and then restart your client. There is a hidden ._sync_xxxxxxx.db file inside the folder of every account configured on your client.

Please note that this will also erase some of your settings about which files to download.

See https://github.com/owncloud/client/issues/5226 for more discussion of this issue.

Isolating Other Issues

Other issues can affect synchronization of your ownCloud files:

- If you find that the results of the synchronizations are unreliable, please ensure that the folder to which you are synchronizing is not shared with other synchronization applications.
- Synchronizing the same directory with ownCloud and other synchronization software such as Unison, rsync, Microsoft Windows Offline Folders, or other cloud services such as Dropbox or Microsoft SkyDrive is not supported and should not be attempted. In the worst case, it is possible that synchronizing folders or files using

ownCloud and other synchronization software or services can result in data loss.

- If you find that only specific files are not synchronized, the synchronization protocol might be having an effect. Some files are automatically ignored because they are system files, other files might be ignored because their filename contains characters that are not supported on certain file systems. For more detailed information see the Ignored Files section.
- If you are operating your own server, and use the local storage backend (the default), make sure that ownCloud has exclusive access to the directory.

The data directory on the server is exclusive to ownCloud and must not be modified manually.

- If you are using a different file backend on the server, you can try to exclude a bug in the backend by reverting to the built-in backend.
- If you are experiencing slow upload/download speed or similar performance issues be aware that those could be caused by on-access virus scanning solutions, either on the server (like the files_antivirus app) or the client.

Log Files

Effectively debugging software requires as much relevant information as can be obtained. To assist the ownCloud support personnel, please try to provide as many relevant logs as possible. Log output can help with tracking down problems and, if you report a bug, log output can help to resolve an issue more quickly.

The client log file is often the most helpful log to provide.

Obtaining the Client Log File

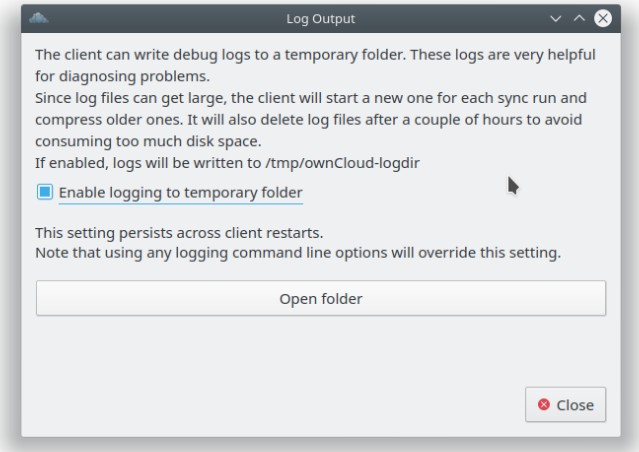
There are several ways to produce log files. The most commonly useful is enabling logging to a temporary directory, described first.

Client log files contain file and folder names, metadata, server URLs and other private information. Only upload them if you are comfortable sharing the information. Logs are often essential for tracking down a problem though, so please consider providing them to developers privately.

Logging to a Temporary Directory

1. Open the ownCloud Desktop Client.
2. Press **[F12]** or **[Ctrl-L]** or **[Cmd+L]** on your keyboard.

   The Log Output window opens.



3. Enable the **[Enable logging to temporary folder]** checkbox.
4. Later, to find the log files, click the **[Open folder]** button.
5. Select the logs for the time frame in which the issue occurred.

That the choice to enable logging will be persist across client restarts.

Saving Files Directly

The ownCloud client allows you to save log files directly to a custom file or directory. This is a useful option for easily reproducible problems, as well as for cases where you want logs to be saved to a different location.

To save log files to a file or a directory:

1. To save to a file, start the client using the --logfile <file> command, where <file> is the filename to which you want to save the file.
2. To save to a directory, start the client using the --logdir <dir> command, where <dir> is an existing directory.

When using the --logdir command, each sync run creates a new file. To limit the amount of data that accumulates over time, you can specify the --logexpire <hours> command. When combined with the --logdir command, the client automatically erases saved log data in the directory that is older than the specified number of hours.

Adding the --logdebug flag increases the verbosity of the generated log files.

As an example, to define a test where you keep log data for two days, you can issue the following command:

```
owncloud --logdir /tmp/owncloud_logs --logexpire 48
```

Logging in the Console

If the ownCloud client isn't able to start and immediately crashes the first two options are not available. Therefore it might need to be necessary to start the ownCloud client using the command line in order to be see the error message

On Linux and Mac simply open the terminal and run:

```
owncloud --logfile - --logflush
```

On Windows open a PowerShell and run the following command:

```
& 'C:\Program Files\ownCloud\owncloud.exe' --logfile - --logflush | Write-Host
```

Make sure to copy the whole command and adjust the path to your owncloud.exe, if you have chosen to install the client in a different path.

To further increase the verbosity of the output you can also combine these commands with the --logdebug argument.

Control Log Content

Thanks to the Qt framework, logging can be controlled at run-time through the QT_LOGGING_RULES environment variable.

Exclude log item categories

```
QT_LOGGING_RULES='gui.socketapi=false;sync.database*=false' \
 /PATH/TO/CLIENT \
 --logdebug --logfile <file>
```

Add HTTP logging entries

```
QT_LOGGING_RULES='sync.httplogger=true' \
 /PATH/TO/CLIENT \
 --logdebug --logfile <file>
```

Only show specific log item categories

```
QT_LOGGING_RULES='*=false;sync.httplogger=true' \
 /PATH/TO/CLIENT \
 --logdebug --logfile <file>
```

ownCloud Server Log File

The ownCloud server also maintains an ownCloud specific log file. This log file must be enabled through the ownCloud Administration page. On that page, you can adjust the log level. We recommend that when setting the log file level that you set it to a verbose level like Debug or Info.

You can view the server log file using the web interface or you can open it directly from the file system in the ownCloud server data directory.

Need more information on this. How is the log file accessed? Need to explore procedural steps in access and in saving this file, similar to how the log file is managed for the client. Perhaps it is detailed in the Admin Guide and a link should be provided from here. I will look into that when I begin heavily editing the Admin Guide.

Webserver Log Files

It can be helpful to view your webserver's error log file to isolate any ownCloud-related problems. For Apache on Linux, the error logs are typically located in the /var/log/apache2 directory. Some helpful files include the following:

- error_log — Maintains errors associated with PHP code.
- access_log — Typically records all requests handled by the server; very useful as a debugging tool because the log line contains information specific to each request and its result.

You can find more information about Apache logging at http://httpd.apache.org/docs/current/logs.html

Core Dumps

On macOS X and Linux systems, and in the unlikely event the client software crashes, the client is able to write a core dump file. Obtaining a core dump file can assist ownCloud Customer Support tremendously in the debugging process.

To enable the writing of core dump files, you must define the OWNCLOUD_CORE_DUMP environment variable on the system.

For example:

```
OWNCLOUD_CORE_DUMP=1 owncloud
```

This command starts the client with core dumping enabled and saves the files in the current working directory.

Core dump files can be fairly large. Before enabling core dumps on your system, ensure that you have enough disk space to accommodate these files. Also, due to their size, we strongly recommend that you properly compress any core dump files prior to sending them to ownCloud Customer Support.

Release Notes

Changelog for the Desktop Client

ownCloud provides a full changelog with a summary and details for each release of the Desktop Client. Click the following link to access it at GitHub.