

Good to know

- [Google Drive: Vollständiges Audit aller Freigaben](#)
- [RustDesk: Automatisierte Geräte-Bereinigung \(Device Cleaner\)](#)

Google Drive: Vollständiges Audit aller Freigaben

1. Übersicht & Funktion

Dieses Tool dient der Sicherheitsüberprüfung des Google Drive Accounts. Es erstellt eine **vollständige Inventur** aller Dateien, die nicht privat sind.

Das Skript erkennt:

- **Öffentliche Links:** Dateien, die für jeden im Internet sichtbar sind.
- **Personen-Zugriff:** Dateien, die gezielt mit externen E-Mail-Adressen geteilt wurden.
- **Ordner-Struktur:** Die Hierarchie bleibt erhalten (welche Datei liegt in welchem Ordner).

Das Ergebnis wird automatisch in eine neue Google Tabelle exportiert.

2. Anleitung für Einsteiger (Schritt-für-Schritt)


Auch ohne Programmierkenntnisse können Sie dieses Tool in wenigen Minuten einrichten.

Schritt A: Das Skript erstellen


1. Öffnen Sie Ihr Google Drive.
2. Klicken Sie oben links auf + **Neu** > **Mehr** > **Google Apps Script**.
3. Es öffnet sich ein neuer Tab mit einem Code-Editor.
4. Geben Sie dem Projekt oben links einen Namen (z. B. `Sicherheits-Check`).

Schritt B: Die "Drive API" aktivieren (WICHTIG!)


Damit das Skript schnell tausende Dateien scannen darf, muss ein spezieller Dienst aktiviert werden.

1. Schauen Sie im Editor auf die linke Leiste.
2. Klicken Sie neben dem Punkt „**Dienste**“ (Services) auf das  Zeichen.
3. Scrollen Sie in der Liste zu **Drive API** und wählen Sie es aus.
4. **Achtung:** Stellen Sie sicher, dass im Dropdown-Menü unten die **Version v3** ausgewählt ist.
5. Klicken Sie auf **Hinzufügen**.
 - *Unter "Dienste" erscheint nun "Drive".*

Schritt C: Code einfügen

1. Löschen Sie im Editor den kurzen Code, der dort bereits steht (`function myFunction...`).
2. Kopieren Sie den Code aus dem untenstehenden Abschnitt "Das Skript".
3. Fügen Sie ihn in den Editor ein.
4. Klicken Sie oben auf das Disketten-Symbol  (**Speichern**).

Schritt D: Ausführen & Genehmigen

1. Stellen Sie sicher, dass oben in der Leiste die Funktion `generateFullSharingReport` ausgewählt ist.
2. Klicken Sie auf  **Ausführen**.
3. **Nur beim ersten Mal:** Google fragt nach Berechtigungen.
 - Klicken Sie auf **Berechtigungen überprüfen**.
 - Wählen Sie Ihr Google-Konto.
 - Es erscheint oft eine Warnung ("Google hasn't verified this app"). Das ist normal, da Sie das Skript selbst erstellt haben.
 - Klicken Sie links unten auf **Erweitert** (Advanced).
 - Klicken Sie ganz unten auf den Link „**Zu ... (Name des Skripts) gehen (unsicher)**“.
 - Klicken Sie auf **Zulassen** (Allow).

Schritt E: Ergebnis prüfen

Das Skript läuft nun (unten sehen Sie ein gelbes Protokoll). Sobald dort „**Fertig!**“ steht, finden Sie direkt daneben den Link zur Google Tabelle mit Ihren Ergebnissen.

3. Das Skript

[Source Code on GitHub](#)

```
/**
 * SKRIPT: Audit aller Google Drive Freigaben (Personen & Links)
 * VERSION: 4.0 (Full Audit & Email-List)
 * * ZWECK:
 * Listet JEDE Datei auf, die nicht rein privat ist.
 * Zeigt Freigabestatus und konkrete E-Mail-Adressen an.
 *
 * VORAUSSSETZUNG:
 * Dienst "Drive API" (Version 3) muss aktiviert sein.
 */

function generateFullSharingReport() {

    // --- 1. SETUP ---
    // Erstellt die Zieltabelle mit aktuellem Zeitstempel
    var timestamp = new Date().toLocaleString();
    var spreadsheet = SpreadsheetApp.create("Sicherheitsbericht: Alle Freigaben (" + timestamp +
    ")");
    var sheet = spreadsheet.getActiveSheet();

    Logger.log("Starte vollständigen Scan...");

    // --- 2. DATEN SAMMELN ---
    var allItemsMap = {};
    var rootItems = [];

    var pageToken;
    var count = 0;

    do {
        // API v3 Abfrage: Holt ID, Name, Link, Eltern und detaillierte Berechtigungen
        var result = Drive.Files.list({
```

```

    q: "'me' in owners and trashed = false",
    pageSize: 1000,
    pageToken: pageToken,
    fields: "nextPageToken, files(id, name, mimeType, webViewLink, parents,
permissions(type, role, emailAddress, allowFileDiscovery))"
  });

  var files = result.files;

  if (files && files.length > 0) {
    for (var i = 0; i < files.length; i++) {
      var file = files[i];

      // Berechtigungen analysieren
      var analysis = analyzePermissions(file.permissions);

      // Wenn die Datei NICHT privat ist (also irgendeine Freigabe hat), nehmen wir sie auf
      if (analysis.isShared) {
        file.children = [];
        file.analysis = analysis;
        allItemsMap[file.id] = file;
        count++;
      }
    }
  }
  pageToken = result.nextPageToken;
} while (pageToken);

Logger.log(count + " geteilte Elemente gefunden. Baue Struktur...");

// --- 3. HIERARCHIE-STRUKTUR BAUEN ---
// Ordnet Dateien ihren Eltern-Ordnern zu, um einen Baum zu erstellen
for (var id in allItemsMap) {
  var item = allItemsMap[id];
  var parentId = (item.parents && item.parents.length > 0) ? item.parents[0] : null;

  if (parentId && allItemsMap[parentId]) {
    allItemsMap[parentId].children.push(item);
  } else {

```

```

    rootItems.push(item);
  }
}

// --- 4. AUSGABE GENERIEREN (Rekursiv) ---
var outputRows = [];
outputRows.push(["Dateiname (Struktur)", "Typ", "Link", "Freigabe-Status", "Berechtigte
Personen"]);

function processNode(node, depth) {
  // Einrückung für Baumansicht berechnen
  var indent = "";
  for (var k=0; k<depth; k++) indent += "  ";
  var visualName = indent + (depth > 0 ? "└─ " : "") + node.name;

  var typeLabel = (node.mimeType === 'application/vnd.google-apps.folder') ? "Ordner" :
"Datei";

  var status = node.analysis.statusLabel;
  var users = node.analysis.userList;

  outputRows.push([visualName, typeLabel, node.webViewLink, status, users]);

  // Kinder sortieren (Ordner zuerst, dann alphabetisch)
  if (node.children.length > 0) {
    node.children.sort(function(a, b) {
      var aIsFolder = (a.mimeType === 'application/vnd.google-apps.folder');
      var bIsFolder = (b.mimeType === 'application/vnd.google-apps.folder');
      if (aIsFolder && !bIsFolder) return -1;
      if (!aIsFolder && bIsFolder) return 1;
      return a.name.localeCompare(b.name);
    });

    for (var j=0; j<node.children.length; j++) {
      processNode(node.children[j], depth + 1);
    }
  }
}

```

```

// Root-Elemente sortieren und Prozess starten
rootItems.sort(function(a, b) {
    var aIsFolder = (a.mimeType === 'application/vnd.google-apps.folder');
    var bIsFolder = (b.mimeType === 'application/vnd.google-apps.folder');
    if (aIsFolder && !bIsFolder) return -1;
    if (!aIsFolder && bIsFolder) return 1;
    return a.name.localeCompare(b.name);
});

for (var i=0; i<rootItems.length; i++) {
    processNode(rootItems[i], 0);
}

// --- 5. IN TABELLE SCHREIBEN ---
if (outputRows.length > 1) {
    sheet.getRange(1, 1, outputRows.length, 5).setValues(outputRows);
    sheet.getRange("A1:E1").setFontWeight("bold");
    sheet.setFrozenRows(1); // Fixiert die Kopfzeile
    sheet.setColumnWidth(1, 400); // Verbreitert die Namensspalte
    sheet.setColumnWidth(5, 300); // Verbreitert die E-Mail Spalte
}

Logger.log("Fertig! Bericht: " + spreadsheet.getUrl());
}

/**
 * HILFSFUNKTION: Analysiert Berechtigungen einer Datei
 * Priorisiert öffentliche Links über Personen-Freigaben
 */
function analyzePermissions(perms) {
    var isShared = false;
    var statusLabel = "Privat";
    var userEmails = [];

    if (perms) {
        for (var i = 0; i < perms.length; i++) {
            var p = perms[i];
            if (p.role === 'owner') continue; // Eigentümer ignorieren

```

```

isShared = true;

// Priorisierung des Status-Labels (Warnung bei öffentlichen Links)
if (p.type === 'anyone') {
    statusLabel = p.allowFileDiscovery ? "△ Öffentlich im Web" : "△ Jeder mit Link";
} else if (p.type === 'domain' && !statusLabel.includes("△")) {
    statusLabel = "□□ Domain / Firma";
} else if ((p.type === 'user' || p.type === 'group')) {
    var email = p.emailAddress || "Gruppe/Unbekannt";
    userEmails.push(email);
}
}
}

// Fallback: Wenn Status noch "Privat" scheint, aber User drin sind
if (statusLabel === "Privat" && userEmails.length > 0) {
    statusLabel = "□□ Beschränkt (Personen)";
}

return {
    isShared: isShared,
    statusLabel: statusLabel,
    userList: userEmails.join(", ")
};
}

```

4. Ergebnisse interpretieren

Das Skript nutzt eine Sicherheits-Logik bei der Anzeige: **Es wird immer der "gefährlichste" (offenste) Status in der Spalte "Freigabe-Status" priorisiert.**

Hier sind Beispiele, wie typische Szenarien in der Tabelle dargestellt werden:

Szenario A: Der Misch-Fall (Link + Person)

Situation: Ein Ordner ist per Link für jeden erreichbar ("Jeder mit Link"), aber zusätzlich wurde die Person `kollege@extern.de` explizit eingeladen.

- **Anzeige:** Der Status warnt vor dem öffentlichen Link (da dies das größere Sicherheitsrisiko ist). Die E-Mail-Adresse wird trotzdem rechts gelistet, damit Sie wissen, wer Zugriff behält, falls Sie den Link entfernen.

| Dateiname | Typ | Freigabe-Status | Berechtigte Personen |
|---------------|--------|------------------|----------------------|
| Projekt Alpha | Ordner | ⚠ Jeder mit Link | kollege@extern.de |

Szenario B: Nur bestimmte Personen

Situation: Die Datei ist privat (kein öffentlicher Link), wurde aber gezielt mit Kollegen geteilt.

- **Anzeige:** Der Status zeigt "Beschränkt", da nur explizit genannte Personen Zugriff haben.

| Dateiname | Typ | Freigabe-Status | Berechtigte Personen |
|------------------|-------|-------------------------|--|
| Gehaltsliste.pdf | Datei | 🔒 Beschränkt (Personen) | chef@firma.de, buchhaltung@firma.de |

Szenario C: Komplette Öffentlich

Situation: Die Datei ist im Web veröffentlicht und kann über Suchmaschinen (Google Suche) gefunden werden, ohne dass man den Link kennt.

| Dateiname | Typ | Freigabe-Status | Berechtigte Personen |
|---------------|-------|---------------------|----------------------|
| Webseite-Logo | Datei | ⚠ Öffentlich im Web | |

Szenario D: Firmenweiter Zugriff (Domain)

Situation: (Nur bei Google Workspace Business-Konten) Die Datei ist für jeden *innerhalb* der eigenen Organisation auffindbar/aufrufbar, aber nicht für Externe.

| Dateiname | Typ | Freigabe-Status | Berechtigte Personen |
|---------------|-------|------------------|----------------------|
| Urlaubsantrag | Datei | 🌐 Domain / Firma | |

RustDesk: Automatisierte Geräte-Bereinigung (Device Cleaner)

Dieses Python-Skript dient der automatisierten Wartung des RustDesk Server Pro. Es verhindert, dass das **Gerätelimit der Lizenz** durch nicht mehr genutzte oder einmalige Support-Sitzungen blockiert wird.

Hintergrund & Problemstellung

Bei der Nutzung von RustDesk Server Pro (insbesondere mit Standard-Lizenzen) tritt häufig ein Problem auf, das auch in der **GitHub Discussion #182** thematisiert wird:

“ Das Problem: "Quick Support" Lizenz-Verbrauch

Jedes Gerät, das sich mit dem Server verbindet, verbraucht einen Lizenz-Slot. Dies gilt auch für **Quick-Support-Kunden**, denen nur einmalig geholfen wird (z.B. via "Nur Ausführen" des Clients ohne Installation).

- Diese Geräte verbleiben oft als "ungruppierte" Einträge im Server.
- Die Geräteliste füllt sich schnell mit **inaktiven Einträgen**.
- Das **Lizenzlimit wird erreicht**, und neue (legitime) Geräte können nicht mehr registriert werden.
- Der Admin muss diese veralteten Datensätze normalerweise mühsam manuell löschen.

Referenz: [GitHub Discussion #182: Delete devices via API](#)

Die Lösung

Dieses Skript automatisiert den Bereinigungsprozess. Es identifiziert Geräte, die **keiner Gruppe zugewiesen** sind oder seit **X Tagen offline** sind, und entfernt diese aus der Lizenzverwaltung.

Da die RustDesk API ein direktes Löschen nicht erlaubt, führt das Skript den notwendigen zweistufigen Prozess automatisch durch:

1. **Disable:** Das Gerät wird deaktiviert.
2. **Delete:** Das Gerät wird endgültig gelöscht und der Lizenz-Slot wird frei.

Funktionen

- **Lizenz-Optimierung:** Hält Slots für aktive Geräte frei.
- **Filterung:** Zielt spezifisch auf "Einmal-Kunden" (ohne Gruppe) oder Altgeräte (Offline > X Tage).
- **Safety First:** Standardmäßig ist ein `DRY_RUN` (Simulationsmodus) aktiv.
- **Logging:** Protokolliert Aktionen in `rustdesk_cleanup.log`.

Installation & Vorbereitung

Das Skript benötigt Python 3 und die `requests` Bibliothek.

1. Abhängigkeiten installieren (Debian/Ubuntu):

```
sudo apt update && sudo apt install python3 python3-pip  
pip3 install requests
```

2. Skript ablegen:

Erstelle eine Datei, z.B. unter `/opt/scripts/rustdesk_cleaner.py` und füge den Python-Code (siehe Abschnitt unten) ein.

3. Konfiguration anpassen:

Öffne das Skript und passe den oberen Konfigurationsbereich an deine Umgebung an:

```
API_URL = "https://your-rustdesk-server.com"  
API_TOKEN = "your-bearer-token-here"
```



Hinweis: Den API Token erhältst du im RustDesk Web-Interface unter Einstellungen -> API.

Nutzung

Manueller Testlauf (Dry Run)

Standardmäßig führt das Skript keine Änderungen durch (Simulation). Um zu sehen, welche Geräte gelöscht würden:

```
python3 rustdesk_cleaner.py delete
```

Manuelle Ausführung (Ernstfall)

Um die Geräte wirklich zu löschen, muss das Flag `--no_dry_run` gesetzt werden. Dies ist der Befehl, um z.B. alle Geräte ohne Gruppe endgültig zu entfernen:

```
python3 rustdesk_cleaner.py delete --no_dry_run
```

Weitere Parameter

Das Skript akzeptiert diverse Argumente zur Steuerung:

| Parameter | Beschreibung |
|-------------------------------|--|
| <code>--offline_days X</code> | Löscht Geräte, die länger als X Tage offline sind. |
| <code>--no_group</code> | (Standard) Löscht Geräte, die keiner Gruppe zugewiesen sind (ideal für Quick Support). |
| <code>--only_disable</code> | Deaktiviert die Geräte nur, löscht sie aber nicht. |
| <code>--yes</code> | Bestätigt die Sicherheitsabfrage automatisch (für Skripte notwendig). |

Beispiel: Alte Geräte löschen Löschen von veralteten Datensätzen, die länger als 90 Tage offline waren:

```
python3 rustdesk_cleaner.py delete --offline_days 90 --no_dry_run
```

Automatisierung (Cronjob)

Um das Skript regelmäßig (z.B. jede Nacht um 03:00 Uhr) laufen zu lassen, kann ein Cronjob eingerichtet werden.

1. Crontab öffnen: `crontab -e`
2. Zeile hinzufügen:

```
# RustDesk Cleanup: Löscht Geräte ohne Gruppe täglich um 03:00 Uhr
0 3 * * * /usr/bin/python3 /opt/scripts/rustdesk_cleaner.py delete --no_dry_run --yes >>
/var/log/rustdesk_cron.log 2>&1
```

⚠ **Wichtig:** Das Flag `--yes` ist im Cronjob notwendig, um Sicherheitsabfragen bei der Massenlöschung automatisch zu bestätigen.

Quellcode (Python)

[Source Code on GitHub](#)

```
#!/usr/bin/env python3

"""
RustDesk Device Cleaner
=====

This script manages devices on a RustDesk API server. It can view,
disable, enable, or delete devices based on filters such as group
membership or offline duration.

Prerequisites & Installation:
-----

1. Install Python 3 (if not already present):
   - Linux (Ubuntu/Debian): sudo apt update && sudo apt install python3 python3-pip
   - Windows: Download from python.org

2. Install required 'requests' library:
```

```
pip install requests
```

How it works:

1. The script fetches all devices from the server via API.
2. It filters the list (e.g., 'No Group' or 'X days offline').
3. It performs the requested action (view, disable, enable, delete).
4. For 'delete' operations, a device MUST be DISABLED first (MANDATORY RustDesk API requirement), then it is DELETED.
5. All actions are logged to the file 'rustdesk_cleanup.log'.

How to create a Cronjob:

1. Open your crontab editor:

```
crontab -e
```

2. Add a line at the end of the file.

```
Format: [minute] [hour] [day] [month] [day_of_week] [command]
```

Example (Delete ungrouped devices every day at 01:00 AM):

```
0 1 * * * /usr/bin/python3 /absolute/path/to/rustdesk_cleaner.py delete
```

3. Save and exit. The script will now run automatically.

Important Override Parameters (CLI):

```
--offline_days X : Overrides config with X days.  
--no_group       : Forces filtering for ungrouped clients.  
--dry_run        : Forces simulation mode.  
--no_dry_run     : Forces execution even if DRY_RUN=True is set in the script.  
--yes            : Confirms actions immediately (required for cron).  
--only_disable   : Only disables clients without deleting them.  
--disable_before_delete: (Default: True) Ensures mandatory disabling before deletion.  
"""
```

```
import requests
```

```
import argparse
```

```
import logging
```

```
from datetime import datetime, timedelta
```

```
# --- CONFIGURATION ---
```

```
DRY_RUN = True          # True = Simulation only | False = Real deletion/disabling
```

```

AUTO_CONFIRM = True      # True = Automatically confirm multiple devices (required for cron)
API_URL = "https://your-rustdesk-server.com"
API_TOKEN = "your-bearer-token-here"

# Filter Options (Default values for automated runs)
DELETE_UNGROUPED = True # True = Only target devices without a group
OFFLINE_DAYS = None     # None = Disable filter | Integer (e.g. 3) = Only devices offline for
X days

# Process Options
# NOTE: RustDesk requires devices to be DISABLED before they can be DELETED.
DISABLE_BEFORE_DELETE = True # Must stay True for successful deletion in RustDesk
ONLY_DISABLE = False       # Set to True to only disable clients without deleting them
# -----

def view(
    url,
    token,
    id=None,
    device_name=None,
    user_name=None,
    group_name=None,
    device_group_name=None,
    offline_days=None,
    no_group=False,
):
    """
    Fetches and filters devices from the RustDesk server.

    Filters:
    - offline_days: Only includes devices offline for at least X days.
    - no_group: Only includes devices that are not assigned to any group.
    """
    headers = {"Authorization": f"Bearer {token}"}
    pageSize = 100
    params = {
        "id": id,
        "device_name": device_name,
        "user_name": user_name,
        "group_name": group_name,

```

```

    "device_group_name": device_group_name,
}

# Add wildcards to search parameters if not already present
params = {
    k: "%" + v + "%" if (v != "-" and "%" not in v) else v
    for k, v in params.items()
    if v is not None
}
params["pageSize"] = pageSize

devices = []
current = 0

# Paginated API requests
while True:
    current += 1
    params["current"] = current
    response = requests.get(f"{url}/api/devices", headers=headers, params=params)
    if response.status_code != 200:
        print(f"Error: HTTP {response.status_code} - {response.text}")
        exit(1)

    response_json = response.json()
    if "error" in response_json:
        print(f"Error: {response_json['error']}")
        exit(1)

    data = response_json.get("data", [])

    # Apply custom filters locally
    for device in data:
        # 1. Filter by offline duration
        if offline_days is not None:
            last_online_str = device.get("last_online")
            if not last_online_str:
                continue
            last_online = datetime.strptime(
                last_online_str.split(".")[0], "%Y-%m-%dT%H:%M:%S"
            )
            if (datetime.utcnow() - last_online).days < offline_days:

```

```

        continue

    # 2. Filter by group membership
    if no_group:
        # If a group name is present, the device is not 'ungrouped'
        if device.get("device_group_name"):
            continue

    devices.append(device)

total = response_json.get("total", 0)
# Check if we've reached the end of the list
if len(data) < pageSize or current * pageSize >= total:
    break

return devices

def check(response):
    if response.status_code != 200:
        print(f"Error: HTTP {response.status_code} - {response.text}")
        exit(1)

    try:
        response_json = response.json()
        if "error" in response_json:
            print(f"Error: {response_json['error']}")
            exit(1)
        return response_json
    except ValueError:
        return response.text or "Success"

def disable(url, token, guid, id):
    """Sends a request to disable a device by its GUID."""
    print("Disable", id)
    headers = {"Authorization": f"Bearer {token}"}
    response = requests.post(f"{url}/api/devices/{guid}/disable", headers=headers)
    return check(response)

```

```

def enable(url, token, guid, id):
    """Sends a request to enable a device by its GUID."""
    print("Enable", id)
    headers = {"Authorization": f"Bearer {token}"}
    response = requests.post(f"{url}/api/devices/{guid}/enable", headers=headers)
    return check(response)

def delete(url, token, guid, id):
    """Sends a request to delete a device by its GUID."""
    print("Delete", id)
    headers = {"Authorization": f"Bearer {token}"}
    response = requests.delete(f"{url}/api/devices/{guid}", headers=headers)
    return check(response)

def assign(url, token, guid, id, type, value):
    print("assign", id, type, value)
    valid_types = [
        "ab",
        "strategy_name",
        "user_name",
        "device_group_name",
        "note",
        "device_username",
        "device_name",
    ]
    if type not in valid_types:
        print(f"Invalid type, it must be one of: {'', '.join(valid_types)}")
        return
    data = {"type": type, "value": value}
    headers = {"Authorization": f"Bearer {token}"}
    response = requests.post(
        f"{url}/api/devices/{guid}/assign", headers=headers, json=data
    )
    return check(response)

def main():
    parser = argparse.ArgumentParser(description="Device manager")
    parser.add_argument(

```

```

        "command",
        choices=["view", "disable", "enable", "delete", "assign"],
        help="Command to execute",
    )
    parser.add_argument("--url", default=API_URL, help=f"URL of the API (Default from config:
{API_URL})")
    parser.add_argument(
        "--token", default=API_TOKEN, help="Bearer token for authentication (Default from
config: ***)"
    )
    parser.add_argument("--id", help="Device ID")
    parser.add_argument("--device_name", help="Device name")
    parser.add_argument("--user_name", help="User name")
    parser.add_argument("--group_name", help="User group name")
    parser.add_argument("--device_group_name", help="Device group name")
    parser.add_argument(
        "--assign_to",
        help="<type>=<value>, e.g. user_name=mike, strategy_name=test,
device_group_name=group1, note=note1, device_username=username1, device_name=name1, ab=ab1,
ab=ab1,tag1,alias1,password1,note1"
    )
    parser.add_argument(
        "--offline_days", type=int, default=OFFLINE_DAYS, help=f"Offline duration in days
(Default from config: {OFFLINE_DAYS})"
    )
    parser.add_argument(
        "--no_group", action="store_true", default=DELETE_UNGROUPED, help=f"Only target
devices with no device group (Default from config: {DELETE_UNGROUPED})"
    )
    parser.add_argument(
        "--dry_run", action="store_true", default=DRY_RUN, help=f"Simulate the operations
(Current default: {DRY_RUN})"
    )
    # Allow explicit disabling if the constant is True
    parser.add_argument(
        "--no_dry_run", action="store_false", dest="dry_run", help="Force execution even if
DRY_RUN is set to True in script"
    )
    parser.add_argument(
        "--log_file", default="rustdesk_cleanup.log", help="Path to the log file (default:
rustdesk_cleanup.log)"
    )

```

```

)

parser.add_argument(
    "--yes", "-y", action="store_true", default=AUTO_CONFIRM, help=f"Confirm the operation
without prompting (Current default: {AUTO_CONFIRM})"
)
parser.add_argument(
    "--only_disable", action="store_true", default=ONLY_DISABLE, help=f"Only disable
clients, do not delete them (Current default: {ONLY_DISABLE})"
)
parser.add_argument(
    "--disable_before_delete", action="store_true", default=DISABLE_BEFORE_DELETE,
help=f"Ensure devices are disabled before deletion (Current default: {DISABLE_BEFORE_DELETE})"
)

args = parser.parse_args()

# Configure logging
logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s - %(levelname)s - %(message)s",
    handlers=[
        logging.FileHandler(args.log_file),
        logging.StreamHandler()
    ]
)

logger = logging.getLogger(__name__)

while args.url.endswith("/"): args.url = args.url[:-1]

devices = view(
    args.url,
    args.token,
    args.id,
    args.device_name,
    args.user_name,
    args.group_name,
    args.device_group_name,
    args.offline_days,
    args.no_group,

```

```

)

if args.command == "view":
    for device in devices:
        print(device)
elif args.command in ["disable", "enable", "delete", "assign"]:
    # Safety check for multiple devices
    if len(devices) > 1 and not args.yes and not args.dry_run:
        logger.warning(f"Found {len(devices)} devices. Operation '{args.command}' requires
--yes or -y flag for multiple devices without interaction.")
        print(f"Found {len(devices)} devices. Use --yes or -y to confirm this operation in
a script.")
        return

    if args.command == "disable":
        for device in devices:
            if args.dry_run:
                logger.info(f"[Dry Run] Would disable device: {device['id']} (GUID:
{device['guid']})")
            else:
                response = disable(args.url, args.token, device["guid"], device["id"])
                logger.info(f"Disabled device {device['id']}: {response}")
    elif args.command == "enable":
        for device in devices:
            if args.dry_run:
                logger.info(f"[Dry Run] Would enable device: {device['id']} (GUID:
{device['guid']})")
            else:
                response = enable(args.url, args.token, device["guid"], device["id"])
                logger.info(f"Enabled device {device['id']}: {response}")
    elif args.command == "delete":
        for device in devices:
            if args.dry_run:
                action = "disable" if args.only_disable else "disable and delete"
                logger.info(f"[Dry Run] Would {action} device: {device['id']} (GUID:
{device['guid']})")
            else:
                # MANDATORY RUSTDESK LOGIC: A client MUST be disabled before it can be
deleted.
                logger.info(f"Processing device {device['id']}. Disabling first (required
for deletion)...")

```

```

        disable_response = disable(args.url, args.token, device["guid"],
device["id"])
        logger.info(f"Disable response for {device['id']}: {disable_response}")

        if args.only_disable:
            logger.info(f"ONLY_DISABLE is active. Skipping deletion for
{device['id']}.")
        else:
            # Proceeding to final deletion
            logger.info(f"Proceeding to delete device {device['id']}...")
            delete_response = delete(args.url, args.token, device["guid"],
device["id"])
            logger.info(f>Delete response for {device['id']}: {delete_response}")
    elif args.command == "assign":
        if "=" not in args.assign_to:
            logger.error("Invalid assign_to format, it must be <type>=<value>")
            return
        type, value = args.assign_to.split("=", 1)
        for device in devices:
            if args.dry_run:
                logger.info(f"[Dry Run] Would assign {type}={value} to device:
{device['id']}")
            else:
                response = assign(
                    args.url, args.token, device["guid"], device["id"], type, value
                )
                logger.info(f"Assigned {type}={value} to {device['id']}: {response}")

if __name__ == "__main__":
    main()

```