PointCab Renamer - Deployment-Anleitung

Version 4.2.1 | Datum: 16. Januar 2026

Build-Status (Testergebnisse 2026-01-16)

Build-Methode	Status	Hinweise
`build_windows.bat`	✅ Funktioniert	Empfohlen auf Windows
`build_linux.sh`	✅ Getestet	Funktioniert auf Ubuntu 20.04+
`build_windows_wine.sh`	⚠️ Experimentell	Fehlschläge auf Headless-Servern möglich
`build_windows_on_linux.sh`	⚠️ Docker	Nicht in Docker-in-Docker möglich

Übersicht

Diese Anleitung beschreibt, wie Sie aus dem Python-Quellcode ausführbare Dateien für Windows (.exe) und Ubuntu (Binary) erstellen.

Voraussetzungen

Benötigte Software

Komponente	Windows	Ubuntu
Python	3.8+	3.8+
PyInstaller	5.0+	5.0+
tkinter	(in Python enthalten)	`python3-tk`

Installation der Voraussetzungen

Windows

1. **Python installieren:**
   - Laden Sie Python von https://www.python.org/downloads/ herunter
   - Bei der Installation: ☑ “Add Python to PATH” aktivieren

2. **PyInstaller installieren:**
   ```
   cmd
   pip install pyinstaller
   ```

Ubuntu

1. **Python und tkinter installieren:**
   ```
   bash
   sudo apt update
   sudo apt install python3 python3-pip python3-tk
   ```

2. **PyInstaller installieren:**
   ```
   bash
   pip3 install pyinstaller
   ```

Windows-Build (.exe)

Automatisch (empfohlen)

1. Öffnen Sie eine Eingabeaufforderung (cmd)
2. Navigieren Sie zum Projektordner:
   ```
   cmd
   cd C:\Pfad\zum\pointcab_renamer
   ```
3. Führen Sie das Build-Skript aus:
   ```
   cmd
   build_windows.bat
   ```
4. Die fertige `.exe` finden Sie im Ordner `dist\pointcab_renamer\`

Manuell

1. Öffnen Sie eine Eingabeaufforderung
2. Navigieren Sie zum Quellcode-Ordner
3. Führen Sie PyInstaller aus:
   ```
   cmd
   pyinstaller --onefile --windowed --name "PointCab_Renamer" ^<br>     --add-data "cluster_cleanup.txt;." ^<br>     pointcab_renamer.py
   ```
4. Die `.exe` befindet sich in `dist\PointCab_Renamer.exe`

PyInstaller-Optionen erklärt

Option	Beschreibung
`--onefile`	Alles in eine einzige .exe packen
`--windowed`	Kein Konsolenfenster anzeigen
`--name`	Name der Ausgabedatei
`--add-data`	Zusätzliche Dateien einbinden
`--icon`	(Optional) Icon-Datei (.ico)

Bekannte Probleme unter Windows

Problem: Antivirus blockiert die .exe

Lösung: Die erstellte .exe als Ausnahme hinzufügen oder signieren.

Problem: “DLL nicht gefunden”

Lösung: Visual C++ Redistributable installieren.

Ubuntu-Build (Binary)

Automatisch (empfohlen)

1. Öffnen Sie ein Terminal
2. Navigieren Sie zum Projektordner:
   ```bash
   cd /pfad/zum/pointcab_renamer
   ```
3. Machen Sie das Build-Skript ausführbar und führen Sie es aus:
   ```bash
   chmod +x build_linux.sh
   ./build_linux.sh
   ```
4. Das fertige Binary finden Sie im Ordner `dist/`

Manuell

1. Öffnen Sie ein Terminal
2. Navigieren Sie zum Quellcode-Ordner
3. Führen Sie PyInstaller aus:
   ```bash
   pyinstaller --onefile --name "pointcab_renamer" \
     --add-data "cluster_cleanup.txt:." \
     pointcab_renamer.py
   ```
4. Das Binary befindet sich in `dist/pointcab_renamer`
5. Machen Sie es ausführbar:
   ```bash
   chmod +x dist/pointcab_renamer
   ```

Bekannte Probleme unter Ubuntu

Problem: “No display name and no $DISPLAY environment variable”

Lösung: Das Binary muss in einer grafischen Umgebung gestartet werden, nicht über SSH.

Problem: “_tkinter not found”

Lösung: `sudo apt install python3-tk`

Cross-Compilation: Windows .exe unter Linux erstellen

Es gibt mehrere Möglichkeiten, eine Windows .exe unter Linux zu erstellen, ohne Windows zu installieren.

Methode 1: Docker (Empfohlen)

Die Docker-Methode ist die zuverlässigste und reproduzierbarste Option.

Voraussetzungen

1. **Docker installieren:**
   ```
   bash
   sudo apt update
   sudo apt install docker.io
   sudo systemctl start docker
   sudo systemctl enable docker
   ```

2. **Benutzer zur docker-Gruppe hinzufügen:**
   ```
   bash
   sudo usermod -aG docker $USER
   # Danach neu einloggen oder:
   newgrp docker
   ```

3. **Docker-Installation testen:**
   ```
   bash
   docker run hello-world
   ```

Verwendung

1. Navigieren Sie zum Projektordner:
   ```
   bash
   cd /pfad/zum/pointcab_renamer
   ```

2. Führen Sie das Build-Skript aus:
   ```
   bash
   ./build_windows_on_linux.sh
   ```

3. Die fertige `.exe` befindet sich in `dist/PointCab_Renamer.exe`

Was das Skript macht

1. Prüft Docker-Installation und -Status
2. Lädt das `cdrx/pyinstaller-windows` Docker-Image (beim ersten Mal)
3. Startet einen Container mit Windows-Umgebung
4. Führt PyInstaller im Container aus
5. Kopiert die .exe und Zusatzdateien nach `dist/`

Vorteile der Docker-Methode

- ✅ Zuverlässig und reproduzierbar
- ✅ Isolierte Build-Umgebung
- ✅ Keine manuelle Windows-Python-Installation
- ✅ Gleiche Ergebnisse wie auf echtem Windows
- ✅ Einfach in CI/CD-Pipelines integrierbar

Troubleshooting Docker

Problem: “Permission denied” beim Docker-Aufruf

Lösung:

```
sudo usermod -aG docker $USER
# Neu einloggen erforderlich!
```

Problem: Docker-Image-Download schlägt fehl

Lösung: Proxy-Einstellungen prüfen oder manuell herunterladen:

```
docker pull cdrx/pyinstaller-windows:python3
```

Problem: Container startet nicht

Lösung: Docker-Daemon prüfen:

```
sudo systemctl status docker
sudo systemctl restart docker
```

Methode 2: Wine (Fallback)

Die Wine-Methode ist weniger zuverlässig, kann aber ohne Docker verwendet werden.

Voraussetzungen

1. **Wine installieren:**
   `bash`
   `sudo dpkg --add-architecture i386`
   `sudo apt update`
   `sudo apt install wine64 wine32`

2. **Installation prüfen:**
   `bash`
   `wine --version`

Verwendung

1. Navigieren Sie zum Projektordner:
   `bash`
   `cd /pfad/zum/pointcab_renamer`

2. Führen Sie das Build-Skript aus:
   `bash`
   `./build_windows_wine.sh`

3. Das Skript installiert automatisch:
   - Windows-Python in Wine
   - PyInstaller

Einschränkungen der Wine-Methode

- ⚠️ Nicht alle Windows-Funktionen werden unterstützt
- ⚠️ Kann bei komplexen Abhängigkeiten fehlschlagen
- ⚠️ Langsamerer Build-Prozess
- ⚠️ Ergebnisse können von echter Windows-Build abweichen

Methode 3: GitHub Actions (Automatisiert)

Für regelmäßige Builds können Sie GitHub Actions verwenden.

Erstellen Sie `.github/workflows/build.yml`:

```yaml
name: Build Windows Executable

on:
  push:
    tags:
      - 'v*'
  workflow_dispatch:

jobs:
  build-windows:
    runs-on: windows-latest
    steps:
      - uses: actions/checkout@v3

      - name: Set up Python
        uses: actions/setup-python@v4
        with:
          python-version: '3.10'

      - name: Install dependencies
        run: pip install pyinstaller

      - name: Build executable
        run: |
          pyinstaller --onefile --windowed --name "PointCab_Renamer" `
            --add-data "cluster_cleanup.txt;." `
            --add-data "BENUTZERHANDBUCH.md;." `
            pointcab_renamer.py

      - name: Upload artifact
        uses: actions/upload-artifact@v3
        with:
          name: PointCab_Renamer_Windows
          path: |
            dist/PointCab_Renamer.exe
            cluster_cleanup.txt
            BENUTZERHANDBUCH.md
```

Vergleich der Cross-Compilation-Methoden

Methode	Zuverlässigkeit	Geschwindigkeit	Aufwand
Docker	⭐⭐⭐⭐⭐	⭐⭐⭐	Niedrig
Wine	⭐⭐	⭐⭐	Mittel
GitHub Actions	⭐⭐⭐⭐⭐	⭐⭐⭐⭐	Niedrig
Echtes Windows	⭐⭐⭐⭐⭐	⭐⭐⭐⭐⭐	Hoch (VM)

Empfehlung: Verwenden Sie die Docker-Methode für lokale Builds und GitHub Actions für automatisierte Release-Builds.

Testen der Executables

Windows-Test

1. Kopieren Sie die `.exe` und `cluster_cleanup.txt` in einen Testordner
2. Doppelklicken Sie auf die `.exe`
3. Das Hauptmenü sollte erscheinen
4. Testen Sie alle drei Modi mit einem Testprojekt

Ubuntu-Test

1. Kopieren Sie das Binary und `cluster_cleanup.txt` in einen Testordner
2. Starten Sie das Programm:
   ```bash
   ./pointcab_renamer
   ```
3. Das Hauptmenü sollte erscheinen
4. Testen Sie alle drei Modi mit einem Testprojekt

Checkliste für Tests

- [ ] Programm startet ohne Fehler
- [ ] Hauptmenü wird angezeigt
- [ ] LSDX-Datei kann ausgewählt werden
- [ ] PointCloud-Ordner wird erkannt
- [ ] Vorschau wird korrekt angezeigt
- [ ] Umbenennung funktioniert
- [ ] LSDX wird aktualisiert
- [ ] Log-Datei wird erstellt
- [ ] Batch-Modus funktioniert
- [ ] Merger funktioniert

Distribution an Mitarbeiter

Bereitstellung

1. **Für Windows:**
   - Kopieren Sie diese Dateien in einen Ordner:
     - `PointCab_Renamer.exe`
     - `cluster_cleanup.txt`
     - `BENUTZERHANDBUCH.md` (oder als PDF)
     - Erstellen Sie ein ZIP-Archiv
     - Verteilen Sie über Netzlaufwerk oder E-Mail
2. **Für Ubuntu:**
   - Kopieren Sie diese Dateien in einen Ordner:
     - `pointcab_renamer`
     - `cluster_cleanup.txt`
     - `BENUTZERHANDBUCH.md`
     - Erstellen Sie ein tar.gz-Archiv:
       `bash`
       `tar -czvf pointcab_renamer_linux.tar.gz pointcab_renamer cluster_cleanup.txt BENUTZER-HANDBUCH.md`
     - Verteilen Sie über Netzlaufwerk

Empfohlene Ordnerstruktur für Mitarbeiter

```
PointCab_Renamer/
├── PointCab_Renamer.exe (oder pointcab_renamer für Linux)
├── cluster_cleanup.txt
├── BENUTZERHANDBUCH.md
└── logs/  (wird automatisch erstellt)
```

Updates verteilen

1. Erstellen Sie die neue Executable
2. Informieren Sie die Mitarbeiter über Änderungen (CHANGELOG)
3. Mitarbeiter ersetzen die alte .exe durch die neue
4. `cluster_cleanup.txt` kann beibehalten werden (falls angepasst)

Troubleshooting beim Build

“ModuleNotFoundError”

Lösung: Fehlende Module installieren:

```
pip install <modulname>
```

“Hidden import not found”

Lösung: Hidden imports hinzufügen:

```
pyinstaller --hidden-import=<modulname> ...
```

“Executable zu groß” (>100MB)

Lösung: UPX-Kompression aktivieren:

```
pip install upx
pyinstaller --onefile --upx-dir=/pfad/zu/upx ...
```

“tkinter funktioniert nicht”

Windows: tkinter ist normalerweise in Python enthalten. Reinstallieren Sie Python mit der “tcl/tk” Option.

Ubuntu: Installieren Sie python3-tk:

```
sudo apt install python3-tk
```

Versionskontrolle

Bei jeder neuen Version:

1. Version im Quellcode aktualisieren (`VERSION = "4.2"` etc.)
2. CHANGELOG.md aktualisieren
3. Neue Builds für Windows und Ubuntu erstellen
4. Builds testen
5. Im Git-Repository taggen:
   `bash`
   `git tag -a v4.2 -m "Version 4.2"`
   `git push origin v4.2`

Troubleshooting: Docker-Probleme

Docker ist nicht installiert

Ubuntu/Debian:

```
sudo apt update
sudo apt install docker.io
sudo systemctl start docker
sudo systemctl enable docker
sudo usermod -aG docker $USER
# Dann neu einloggen oder: newgrp docker
```

Fedora/RHEL:

```
sudo dnf install docker
sudo systemctl start docker
```

Docker-Daemon startet nicht

Symptom: `Cannot connect to the Docker daemon`

Lösungen:

1. **Service starten:**
   `bash`
   `sudo systemctl start docker`
2. **Status prüfen:**
   `bash`
   `sudo systemctl status docker`
3. **Logs prüfen:**
   `bash`
   `sudo journalctl -u docker.service`

Berechtigung verweigert

Symptom: `permission denied while trying to connect to the Docker daemon`

Lösung: Benutzer zur Docker-Gruppe hinzufügen:

```
sudo usermod -aG docker $USER
# Danach neu einloggen
```

Oder mit sudo ausführen:

```
sudo ./build_windows_on_linux.sh
```

Docker in Container-Umgebung (Docker-in-Docker)

Problem: Docker kann nicht in unprivilegierten Containern laufen.

Lösungen:

1. **Wine-Alternative verwenden:**
   `bash`
   `./build_windows_wine.sh`
2. **Auf Host-System bauen**
3. **GitHub Actions nutzen** (siehe `.github/workflows/`)
4. **Container mit `--privileged` starten** (nicht empfohlen für Produktion)

WSL2 unter Windows

Problem: Docker-Befehle schlagen in WSL2 fehl.

Lösung:
1. Docker Desktop für Windows installieren

2. In Docker Desktop: Settings → Resources → WSL Integration aktivieren
3. WSL-Distribution auswählen

Docker-Image Download schlägt fehl

Symptom: `Error pulling image` oder Timeout

Lösungen:

1. **Internetverbindung prüfen**
2. **Proxy konfigurieren:**
   ```bash
   export HTTP_PROXY=http://proxy:port
   export HTTPS_PROXY=http://proxy:port
   ```
3. **Manueller Download:**
   ```bash
   sudo docker pull cdrx/pyinstaller-windows:python3
   ```

Vergleich der Build-Methoden

Methode	Plattform	Vorteile	Nachteile
`build_windows.bat`	Windows	Nativ, zuverlässig	Braucht Windows
`build_windows_on_linux.sh`	Linux + Docker	Cross-compilation	Docker erforderlich
`build_windows_wine.sh`	Linux + Wine	Kein Docker nötig	Weniger zuverlässig
GitHub Actions	Cloud	Automatisiert	Braucht GitHub-Repo

Empfehlung: Für zuverlässige Windows-Builds verwenden Sie:
1. **Native Windows** (build_windows.bat) - Am zuverlässigsten
2. **Docker auf Linux** (build_windows_on_linux.sh) - Gut für CI/CD
3. **GitHub Actions** - Automatisiert bei jedem Push/Release
