IS this the right command to list all pacman packages currently installed in Msys in a file: pacman -Qqe > "/d/pacman\_list.txt"

Edit

Yes, that command is correct for listing explicitly installed packages in MSYS2. Here's what each part does:

* pacman -Qqe: Lists packages with minimal output (-Qq) that were explicitly installed (-e)
* > "/d/pacman\_list.txt": Redirects the output to a file on your D: drive

This will give you a clean list of package names without version numbers or descriptions, showing only the packages you explicitly installed (not their dependencies).

If you want more detailed information, you could also use:

* pacman -Qe (includes version numbers)
* pacman -Q (lists all packages including dependencies)

The command you have will work perfectly for creating a backup list of your explicitly installed packages.

Retry

D

pip list --format=freeze | cut -d'=' -f1 > "/d/python\_packages\_27072025.txt"

This command is okay for storing current python packages in a file without version. But installing them in the same order may result in some packages not finding dependencies (particularly initial list items) and fail. So How to install them on a new system taking this file as a base?

Edit

You're absolutely right about the dependency issue. When you install packages in the order they appear in your list, some packages might try to install before their dependencies are available, causing failures.

Here are several approaches to handle this:

**Method 1: Let pip handle dependencies automatically**

bash

*# Read each package and install individually, letting pip resolve dependencies*

while read package; do pip install "$package"; done < "/d/python\_packages\_27072025.txt"

**Method 2: Install all at once (recommended)**

bash

*# Install all packages in one command - pip will resolve dependencies*

pip install $(cat "/d/python\_packages\_27072025.txt" | tr '\n' ' ')

**Method 3: Use requirements.txt format instead**

For future exports, consider using the standard requirements.txt approach:

bash

*# Export with versions (better for reproducibility)*

pip freeze > "/d/requirements.txt"

*# Install from requirements.txt*

pip install -r "/d/requirements.txt"

**Method 4: Create a proper requirements file from your list**

bash

*# Convert your current list to a requirements.txt format*

cat "/d/python\_packages\_27072025.txt" | xargs -I {} echo "{}==\*" > "/d/temp\_requirements.txt"

pip install -r "/d/temp\_requirements.txt"

**Recommendation**: Method 2 is usually the most reliable for your current file format, as pip will automatically resolve and install dependencies in the correct order when given all packages at once.

Retry

D

I tried to install from mounted drive on linux like this: [admin@archlinux-nvme ~]$ pip install $(cat "/mnt/hdd/python\_packages\_27072025.txt" | tr '\n' ' ') error: externally-managed-environment × This environment is externally managed ╰─> To install Python packages system-wide, try 'pacman -S python-xyz', where xyz is the package you are trying to install.

If you wish to install a non-Arch-packaged Python package, create a virtual environment using 'python -m venv path/to/venv'. Then use path/to/venv/bin/python and path/to/venv/bin/pip.

If you wish to install a non-Arch packaged Python application, it may be easiest to use 'pipx install xyz', which will manage a virtual environment for you. Make sure you have python-pipx installed via pacman. note: If you believe this is a mistake, please contact your Python installation or OS distribution provider. You can override this, at the risk of breaking your Python installation or OS, by passing --break-system-packages. hint: See PEP 668 for the detailed specification.

Edit

This is a common issue on modern Linux distributions that implement PEP 668 to prevent breaking the system Python environment. Here are your options:

**Option 1: Use a virtual environment (recommended)**

bash

*# Create a virtual environment*

python -m venv ~/myenv

*# Activate it*

source ~/myenv/bin/activate

*# Now install your packages*

pip install $(cat "/mnt/hdd/python\_packages\_27072025.txt" | tr '\n' ' ')

*# When done, deactivate*

deactivate

**Option 2: Use pipx for applications**

If these are Python applications (not libraries), use pipx:

bash

*# Install pipx first*

sudo pacman -S python-pipx

*# Install each package with pipx*

while read package; do pipx install "$package"; done < "/mnt/hdd/python\_packages\_27072025.txt"

**Option 3: Use system packages when available**

Check if packages are available through pacman first:

bash

*# Search for Python packages in Arch repos*

pacman -Ss python-packagename

*# Install via pacman if available*

sudo pacman -S python-packagename

**Option 4: Override (not recommended)**

bash

*# This bypasses the protection but can break your system*

pip install --break-system-packages $(cat "/mnt/hdd/python\_packages\_27072025.txt" | tr '\n' ' ')

**Best practice**: Use Option 1 (virtual environment) for development work, or Option 2 (pipx) if these are standalone applications you want to run system-wide. This keeps your system Python clean and avoids conflicts.

Retry

D

That was archlinux where typically these errors occur when I am not in venv. I entered there using source ~/.venv/bin/activate and error vanished.

Edit

Exactly! That's the standard behavior on Arch Linux and other modern distributions that follow PEP 668. Once you activated your virtual environment with source ~/.venv/bin/activate, pip could install packages safely without affecting the system Python.

So your complete workflow would be:

bash

*# Activate your existing venv*

source ~/.venv/bin/activate

*# Install all packages from your list*

pip install $(cat "/mnt/hdd/python\_packages\_27072025.txt" | tr '\n' ' ')

*# When done, deactivate (optional)*

deactivate

This is actually the cleanest approach - it keeps your Python packages isolated in the virtual environment and prevents any conflicts with system packages. Arch Linux is particularly strict about this to maintain system stability.

If you don't have a virtual environment yet, you can create one with:

bash

python -m venv ~/.venv

source ~/.venv/bin/activate

Then proceed with your package installation as you did.

Retry

D

I think there still errors with packages like pywin32. How do I skip them? I think you can using your first option earlier or will that also not work. It should skip packages with errors and continue with rest in list:

# Read each package and install individually, letting pip resolve dependencies while read package; do pip install "$package"; done < "/d/python\_packages\_27072025.txt"

Edit

You're absolutely right! Packages like pywin32 are Windows-specific and will fail on Linux. The while read loop approach will handle this better, but you should add error handling to skip failed packages and continue with the rest.

Here are improved versions:

**Option 1: Continue on errors (basic)**

bash

while read package; do

pip install "$package" || true

done < "/mnt/hdd/python\_packages\_27072025.txt"

**Option 2: Continue with better error reporting**

bash

while read package; do

echo "Installing $package..."

if pip install "$package"; then

echo "✓ Successfully installed $package"

else

echo "✗ Failed to install $package (skipping)"

fi

done < "/mnt/hdd/python\_packages\_27072025.txt"

**Option 3: Log successes and failures**

bash

while read package; do

if pip install "$package" 2>/dev/null; then

echo "$package" >> successful\_installs.txt

else

echo "$package" >> failed\_installs.txt

echo "Failed: $package"

fi

done < "/mnt/hdd/python\_packages\_27072025.txt"

**Option 4: Pre-filter known problematic packages**

bash

*# Create a filtered list first*

grep -v -E "(pywin32|winsound|winreg)" "/mnt/hdd/python\_packages\_27072025.txt" > filtered\_packages.txt

*# Then install*

while read package; do pip install "$package" || true; done < filtered\_packages.txt

The while read approach is definitely better here because it installs packages one by one, so platform-specific failures won't stop the entire process.