INSTALL

Todd Wintermute

2025-05-23

Contents

1 Prerequisites

Download and install the following before running this program.

- *Python*
  - This program in written in Python. Therefore, Python is required to run the program
  - Python is usually already installed on Linux
- *tmux*
  - Used to split the screen per ssh session

2 Install

2.1 Linux

Install the prerequisites using the instructions on the respective websites.

The program is installed using `python` and the `pip` module. There are no Python dependencies to this program. All code utilizes only the Python standard library.

If you do not wish to install this package to your system, a Python zipapp is included (`.pyz`). If you system has Python installed, the Pyzip zipapp can simply be downloaded and ran. It is located in the `releases` folder or on the github releases page and is named `mssh-menu.pyz`.

Two installation files are included in the `releases` folder or on the github releases page. One is a Python "wheel" file (`.whl`) and the other one is a compressed `tar` file (`.tar.gz`). Either file can install the program, although it is recommended to install the `.whl`. The following are insturctions to install the `.whl`.

Open a `terminal` window.

Install with the following command:

```
python3 -m pip install <mssh-menu-{version}-py3-none-{platform}.whl>
```

Python `pip` installs the program, the `man` page, and shell completions.

2.1.1 Externally Managed Base Environments

On newer distributions of Debian, Ubuntu, and more PEP 668 introduces challenges to the user to install Python packages, even if they do not modify the system packages.

If you are on one of these systems (you most likely are), you will receive a message like the following:

```
error: externally-managed-environment

× This environment is externally managed
╰─> To install Python packages system-wide, try apt install
    python3-xyz, where xyz is the package you are trying to
    install.

    If you wish to install a non-Debian-packaged Python package,
    create a virtual environment using python3 -m venv path/to/venv.
    Then use path/to/venv/bin/python and path/to/venv/bin/pip. Make
    sure you have python3-full installed.

    If you wish to install a non-Debian packaged Python application,
    it may be easiest to use pipx install xyz, which will manage a
    virtual environment for you. Make sure you have pipx installed.

    See /usr/share/doc/python3.12/README.venv for more information.

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.
```

There are a few ways to work around this. Because there is not a `.deb` or `.rpm` created for this project you cannot install this program using your system package manager. That leaves creating a virtual environment, using `pipx`, or installing an `altinstall` of python.

2.1.1.1 VENV Install

A virtual environment can be created for installing this program. The downside to this method is the user must activate the virtual environment before being able to access the program. Once the virtual environment is activated the program and the man page are available to the user. However, I notice the shell completions are not automatically working as the virtual environment does not set the proper paths for shell-completions. More on this later.

To install using a virtual environment, first install `python3-venv`.

```
sudo apt update
sudo apt install python3-venv
```

Then create the virtual environment like this:

```
python3 -m venv $HOME/.venv/python3
```

Activate the `venv`:

```
source $HOME/.venv/python3/bin/activate
```

Then install this program using `pip`.

```
pip install <mssh-menu-{version}-py3-none-{platform}.whl>
```

In my observations, the `venv`'s `activate` script does not set the paths needed for shell-completions. You can run the following commands at the command line for your respective shell to set the correct path and remove any existing completions for this program.

`bash`

```
complete -r mssh-menu
XDG_DATA_DIRS="$VIRTUAL_ENV/share:$XDG_DATA_DIRS"
```

`zsh`

```
```

```
FPATH="$VIRTUAL_ENV/share/zsh/site-functions:$FPATH"
compinit
```

`fish`

```
set -gx fish_complete_path $VIRTUAL_ENV/share/fish/vendor_completions.d
$fish_complete_path
```

I will leave it to you to decide if and how to modify the python venv activate script to avoid typing these commands every time.

2.1.1.2 PIPX Install

PIPX has the benefit of installing the program into a virtual environment without forcing the user to run a command to enter the virtual environment. PIPX handles creating the virtual environment for the program and running the program in the virtual environment transparently. PIPX installs the program and the man pages. The shell-completions are copied over, but will not work as the location is not on the path. More on this later.

To install using `pipx` first install `pipx` on your system with the commands:

```
sudo apt update
sudo apt install pipx
pipx ensurepath
```

Then install this package with this command:

```
pipx install <mssh-menu-{version}-py3-none-{platform}.whl>
```

One way to make shell-completions work is to add the pipx path to this programs's share folder and remove any existing completions for this program:

`bash`

```
complete -r mssh-menu
XDG_DATA_DIRS="$HOME/.local/share/pipx/venvs/mssh-menu/share:$XDG_DATA_DIRS"
```

`zsh`

```
FPATH="$HOME/.local/share/pipx/venvs/mssh-menu/share/zsh/site-functions:
$FPATH"
compinit
```

`fish`

```
set -gx fish_complete_path $HOME/.local/share/pipx/venvs/mssh-menu/share/
fish/vendor_completions.d $fish_complete_path
```

I will leave it to you to decide if and how to modify your system to avoid typing these commands every time.

Another way to make shell-completions work with pipx is to run the `mssh-menu --completion <shell>` option. For details on this method see the section below Shell Completions

2.1.1.3 Python Altinstall

Python altinstall installs an alternate version of Python on your system by compiling the python source and then performing an `altinstall`. Use this method if you want a user version of Python on your system that isn't externally managed by your Linux distribution's package manager, and you don't care to create a virtual environment to run this program. You can use the altinstall of Python for anything Python related. The altinstall method will install the program, man pages, and shell-completions.

You should reference Building Python (Python docs) for instructions.

Here I will summarize the steps for an example system.

Install the prerequisites:

```
sudo apt update
sudo apt install build-essential gdb lcov pkg-config \
      libbz2-dev libffi-dev libgdbm-dev libgdbm-compat-dev liblzma-dev \
      libncurses5-dev libreadline6-dev libsqlite3-dev libssl-dev \
      lzma lzma-dev tk-dev uuid-dev zlib1g-dev
```

Download the Python source from Python.

Extract the archive and cd into the created directory.

```
tar -xvf Python-MAJOR.MINOR.PATCH.tar.xz
cd Python-MAJOR.MINOR.PATCH
```

Then use these steps to build python.

```
./configure --enable-optimizations
make
make test
sudo make altinstall
```

You should now have an alternate installation of Python on your system.

Install this package by replacing `python3` with your alternate installation version of Python (e.g. `python3.13`).

Here is an example how to do that.

```
python3.13 -m pip install --upgrade pip
python3.13 -m pip install <mssh-menu-{version}-py3-none-{platform}.whl>
```

3 Uninstall

3.1 Linux

Open a `terminal` window.

Use python pip to uninstall the program with the following command:

```
python3 -m pip uninstall mssh-menu
```

3.1.1 Externally managed base environments

For externally managed base environments, use one of the following.

3.1.1.1 VENVs

For virtual environments, first activate the virtual environment.

Then run the command:

```
pip uninstall mssh-menu
```

3.1.1.2 PIPX

For pipx installs, uninstall with:

```
pipx uninstall mssh-menu
```

3.1.1.3 Python altinstall

For python `altinstalls`, substitute python3 with your version of python (e.g. python3.13)

Here is an example:

```
python3.13 -m pip uninstall mssh-menu
```

4 Extra Info

4.1 History file

The selections are saved in a history file. The location of the history file is:

- `$HOME/.local/share/mssh-menu/mssh-menu.json`

You can delete this file to clear your history.

This folder is not automatically removed on program uninstallation. You can manually delete it without issues.

4.2 Shell Completions

This program includes shell completions which should be installed by default to your local user's share directory (e.g. `$HOME/.local/share`).

- `bash`
  - `$HOME/.local/share/bash-completion/completions/mssh-menu`
- `zsh`
  - `$HOME/.local/share/zsh/site-functions/_mssh-menu`
- `fish`
  - `$HOME/.local/share/fish/vendor_completions.d/mssh-menu.fish`

If shell completions are not working, this program includes an option to provide the shell completion to stdout.

Run the command:

```
mssh-menu --completion <shell>
```

Where `<shell>` is the name of your shell.

Using shell redirection you can place these commands in a file where your shell looks for shell completions. This part you will have to research for your shell.

4.3 TMUX defaults

Here is my default `.tmux.conf` file for reference.

`mssh-menu` will automatically take care of setting the `base-index` and `pane-base-index` to 1.

`mssh-menu` will not configure the `default-terminal`, `history-limit`, or `mouse` options. If you want those options, create a `.tmux.conf` file in your user's home directory, and add the contents below to that file.

```
# Improve colors
set -g default-terminal screen-256color

# Set scrollback buffer to 10000
set -g history-limit 10000

# Enable mouse mode (tmux 2.1 and above)
set -g mouse on
```

```
# Set first windows and pane to index of 1 (instead of zero)
set -g base-index 1
set -g pane-base-index 1
```