tclpysh User Guide

Version 2025.06

DashThru

Copyright Notice

Contents

1

1 tclpysh Introduction

tclpysh is a standalone hybrid Tcl+Python interpreter developed by DashThru Technology, based on the DashThru FusionShell interpreter core. It supports Tcl mode, Python mode, and a Tcl+Python hybrid mode. This design preserves the traditional scripting style of Tcl while providing users with the flexibility of an open Python platform.

Traditional Tcl interpreters such as tclsh have become insufficient for modern scripting needs. In contrast, tclpysh remains compatible with tclsh while offering enhanced functionality such as native Python support.

For detailed information about the DashThru FusionShell interpreter core, please refer to the FusionShell User Guide.

1-1 tclpysh Feature Overview

tclpysh provides the following three usage modes. For detailed descriptions, please refer to Chapter 2: How to Use tclpysh.

- **Launching with Direct Execution** – Start an interactive command-line session or execute a script using the tclpysh executable.
- **Launching as a Script Interpreter** – Specify the tclpysh interpreter in the script header using a #! shebang.
- **Importing into Third-Party Tools** – Import tclpysh into third-party tools with a Python mode.

1-2 tclpysh Version Compatibility

The tclpysh interpreter is compatible with different versions of Python. Users should select the appropriate version based on the requirements of their Python scripts. When importing into third-party tools, please ensure that the Python version embedded in the tool matches that of tclpysh.

Currently, tclpysh supports Python 3.6 and above. The supported Python version can be identified from the name of the installation package. For example, the following package name indicates support for Python 3.9:

```
tclpysh_py39_v2025.06-Alpha.tar.xz
```

1-3 tclpysh Installation and Deployment

- **Recommended Operating Systems: RHEL/CentOS 6.5–7.9**
  Experimental Support: RHEL/Rocky/Alma 8.x

You can check the OS version in RHEL or CentOS by inspecting the system-release file:

```
% cat /etc/system-release
CentOS Linux release 7.9.2009 (Core)
```

- **Installing tclpysh**

Simply extract the compressed package to your desired installation path:

```
% tar xJvf tclpysh_py39_v2025.06-Alpha.tar.xz
```

- **Setting System Environment Variables**

You can add environment variable settings to your shell initialization file (e.g., `.cshrc`). In the example below, replace `<installation_path>` with the actual path where you installed tclpysh. Once set, the `tclpysh` executable can be used directly from the command line.

The environment variable `DASHTHRU_LICENSE_SERVER` is used to configure the license service. For instructions on obtaining and starting the license service, please refer to the DashLM User Guide.

`<port>` refers to the license server port (default is 28000), and `<hostname>` is the machine name or IP address hosting the DashLM license service.

```
% setenv DASHTHRU_LICENSE_SERVER
<port>@<hostname>:<port>@<hostname>:…
% set path = ($path <installation_path>/tclpysh_py39_v2025.06/bin)
```

Example using `csh`:

```
% setenv DASHTHRU_LICENSE_SERVER 28000@lic_server1:28000@lic_server2
% set path = ($path /edatool/tclpysh_py39_v2025.06/bin)
```

- **Launching tclpysh**

```
% which tclpysh
/edatool/tclpysh_py39_v2025.06/bin/tclpysh
% tclpysh
tcl>
```

2

2 How to Use tclpysh

To accommodate different user scenarios and script deployment needs, tclpysh provides three flexible modes of usage: Direct Execution, Script Interpreter, and Third-Party Integration.

The Direct Execution mode is ideal for interactive debugging and routine manual execution. It functions similarly to the traditional tclsh and allows users to quickly enter the development environment.

By adding a shebang (#!) with the tclpysh path at the top of a script, the Script Interpreter mode allows the script to be run independently. This is useful for integrating scripts into automated flows.

The Import Mode supports embedding tclpysh into third-party tools with a Python mode. This provides users with the flexibility to integrate hybrid Tcl/Python scripting directly into external applications.

With these three usage modes, users can choose the most appropriate method based on their specific application requirements. This chapter provides detailed explanations of the following:

- Launching with Direct Execution
- Launching as a Script Interpreter
- Importing into Third-Party Tools

2-1 Launching with Direct Execution

- **Launching the Interactive Command Line**

You can launch the interactive shell by directly executing the `tclpysh` binary located in the installation directory. A full example is shown below:

```
%  tclpysh
tcl> set a 0
0
tcl> pymode
py> a
'0'
py> b='1'
py> tclmode()
tcl> return $a
0
tcl> return $b
1
tcl> exit
Info: thank you for using FusionShell! (err:0, crit:0, warn:0, info:1)
 CPU-Shell : 0.25 second(s)     Mem-Shell-Res: 18MB
 CPU-Core  : 0.0 second(s)      Mem-Core-Res : 2MB
 Wall      : 127.98 second(s)   Mem-Core-Virt: 13MB
```

- **Launching with a Script File**

You can also use `tclpysh` to execute a script file directly by providing the script as an argument. A full example is provided below:

mix_tcl_py.scr

```
set a 0
pymode
print(f'python: a = {a}')
b='1'
tclmode()
puts "tcl: a = $a"
puts "tcl: b = $b"
exit
```

```
%  tclpysh mix_tcl_py.scr
python: a = 0
tcl: a = 0
tcl: b = 1
Info: thank you for using FusionShell! (err:0, crit:0, warn:0,
info:1)
 CPU-Shell : 0.21 second(s)  Mem-Shell-Res: 18MB
 CPU-Core  : 0.0 second(s)   Mem-Core-Res : 2MB
 Wall      : 0.01 second(s)  Mem-Core-Virt: 13MB
```

2-2 Launching as a Script Interpreter

When executing a script file directly, you can specify tclpysh as the script interpreter by adding a `#!` shebang line at the top of the file. Replace `<installation_path>` with the actual path to the `tclpysh` binary in the shebang example below. This allows the script to be executed independently.

```
#!<installation_path>/tclpysh_py39_v2025.06/bin/tclpysh
```

A full example is provided below:

mix_tcl_py.scr

```
#!/edatool/tclpysh_py39_v2025.06/bin/tclpysh

set a 0
pymode
print(f'python: a = {a}')
b='1'
tclmode()
puts "tcl: a = $a"
puts "tcl: b = $b"
exit
```

```
%  mix_tcl_py.scr
python: a = 0
tcl: a = 0
tcl: b = 1
Info: thank you for using FusionShell! (err:0, crit:0, warn:0,
info:1)
 CPU-Shell : 0.21 second(s)  Mem-Shell-Res: 18MB
 CPU-Core  : 0.0 second(s)   Mem-Core-Res : 2MB
 Wall      : 0.01 second(s)  Mem-Core-Virt: 13MB
```

2-3 Importing into Third-Party Tools

tclpysh can also be imported into third-party tools that support Python mode. Before importing, you need to enter the Python environment of the third-party tool and use `sys.version` to confirm that its Python version matches the version of tclpysh. An import example is shown below. Please replace `<installation_path>` with your actual installation path:

```
third_party_py> import sys
third_party_py> sys.version
Python 3.6.8 (default, Oct 13 2020, 16:18:22)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-44)]
third_party_py> sys.path = [
'<installation_path>/tclpysh_py36_v2025.06/lib'] + sys.path
third_party_py> import tclpysh
```

A full example of importing and using tclpysh in the third-party tool:

```
third_party_py> import sys
third_party_py> sys.path = ['/edatool/tclpysh_py36_v2025.06/lib'] +
sys.path
third_party_py> import tclpysh
tcl> set a 0
0
tcl> pymode
py> a
'0'
py> b='1'
py> tclmode()
tcl> return $a
0
tcl> return $b
1
tcl> exit
Info: thank you for using FusionShell! (err:0, crit:0, warn:0, info:1)
 CPU-Shell : 0.25 second(s)     Mem-Shell-Res: 18MB
 CPU-Core  : 0.0 second(s)      Mem-Core-Res : 2MB
 Wall      : 127.98 second(s)   Mem-Core-Virt: 13MB
```

- **Exiting tclpysh**

Please note that using `exit` in tclpysh will terminate the entire process. To exit only the tclpysh shell and return to the third-party Python environment, use Tcl's `quit` command or Python's `quit()` function:

```
third_party_py> import sys
third_party_py> sys.path = ['/edatool/tclpysh_py36_v2025.06/lib'] +
sys.path
third_party_py> import tclpysh
tcl> quit
third_party_py>
```

- **Manually Launching the tclpysh Shell**

By default, importing tclpysh automatically starts the command-line interface. To prevent this, set the `_autorun` variable to `False` before `import`. After importing, you can manually launch the shell using `tclpysh.run()` as needed:

```
third_party_py> import sys
third_party_py> sys.path = ['/edatool/tclpysh_py36_v2025.06/lib'] +
sys.path
third_party_py> _autorun = False
third_party_py> import tclpysh
third_party_py>
third_party_py> tclpysh.run()
tcl>
```

- **Manually Closing tclpysh**

Normally, exiting the host application will close tclpysh. If that does not happen, you can manually call `tclpysh.close()`. A success message confirms the shutdown:

```
third_party_py> import sys
third_party_py> sys.path = ['/edatool/tclpysh_py36_v2025.06/lib'] +
sys.path
third_party_py> import tclpysh
tcl> quit
third_party_py> tclpysh.close()
Info: thank you for using FusionShell! (err:0, crit:0, warn:0, info:1)
 CPU-Shell : 0.25 second(s)     Mem-Shell-Res: 18MB
 CPU-Core  : 0.0 second(s)      Mem-Core-Res : 2MB
 Wall      : 127.98 second(s)   Mem-Core-Virt: 13MB
third_party_py>
```

- **Customizing the Prompt Prefix**

By default, after importing tclpysh into a third-party tool, the command-line prompts for Tcl and Python modes are `tcl>` and `py>`, respectively. If you wish to add a prefix to these prompts, you can set the `_prompt` variable before importing tclpysh.

For example, if you set `_prompt` to `'your_platform'`, the command prompts will appear as `your_platform-tcl>` and `your_platform-py>`:

```
third_party_py> import sys
third_party_py> sys.path = ['/edatool/tclpysh_py36_v2025.06/lib'] +
sys.path
third_party_py> _prompt = 'your_platform'
third_party_py> import tclpysh
your_platform-tcl> pymode
your_platform-py>
```

- **Disabling Variable Sharing**

By default, tclpysh follows the FusionShell feature of shared variable scope across Tcl mode, Python mode, and the Python environment of the third-party tool. As shown in the example below, a variable `abc` set in Tcl mode is accessible in both tclpysh's Python mode and the host Python environment:

```
third_party_py> import sys
third_party_py> sys.path = ['/edatool/tclpysh_py36_v2025.06/lib'] +
sys.path
third_party_py> import tclpysh
tcl> set abc 123
123
tcl> pymode
py> abc
'123'
py> quit()
third_party_py> abc
'123'
```

If you want to disable variable sharing between tclpysh and the third-party tool, and ensure environment isolation, you can define a `_globals` variable before importing. This variable must be a `dict` object.

After this setting, any variables created within tclpysh will be stored in `_globals` and will no longer be directly accessible in the third-party tool's global namespace. You can access them only via `_globals['var_name']` as shown in the following example:

```
third_party_py> import sys
third_party_py> sys.path = ['/edatool/tclpysh_py36_v2025.06/lib'] +
sys.path
third_party_py> _globals = {}
third_party_py> import tclpysh
tcl> set abc 123
123
tcl> quit
third_party_py> _globals['abc']
'123'
third_party_py> abc
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'abc' is not defined
```