- Página Imutável
- Informações
- Anexos
- Mais Ações:

- Ubuntu Wiki
- Entrar
- Help

Debug Symbol Packages

Debugging Central This page is part of the debugging series — pages with debugging details for a variety of Ubuntu packages.

Conteúdo

This document describes how to set up the debugging symbol packages (*-dbg.deb and *-.dbgsym.ddeb). You might need to do this when you are performing tasks like a Backtrace or using Valgrind.

Debug Symbol Packages

If you want to debug a crash in a project you are developing yourself or from a third-party package, or need the debug symbols for particular libraries very often, it is helpful to install those permanently into your system.

Debugging a program or analyzing stack traces can be very hard because most of the information that can be extracted from them will be in the form of offsets into the binary machine code of the executable. The extra information that allows mapping a machine instruction in a complex module to a function and line number in a corresponding C file is stored in the debug symbol packages.

The maintainers of a project usually place the `*-dbg.deb` packages alongside their binary package in the same repository. Installing them will just involve finding the appropriate package name based on the version of that is installed. However, not all maintainers do this and they may not be up to date. During compilation on the ubuntu servers debug packages are also created automatically for each binary package. These are named `*-dbgsym.ddeb` and are stored in a separate archive. This is because these packages are used relatively rarely compared to the main repositories and need not be mirrored as much. (the spec for this) Either one of these can be used, but for a given package only one can be used at a time.

Getting -dbgsym.ddeb packages

In case the `-dbg` package is not available for the version of the package you are using, you might have to use the `-dbgsym` packages. When a large number of dependancies need to have their symbols retrieved it might become necessary to enable the `-dbgsym` repositories anyway.

1. Create an `/etc/apt/sources.list.d/ddebs.list` by running the following line at a terminal:

   Esconder número das linhas

   ```
   echo "deb http://ddebs.ubuntu.com $(lsb_release -cs) main restricted universe multiverse
   deb http://ddebs.ubuntu.com $(lsb_release -cs)-updates main restricted universe multiverse
   deb http://ddebs.ubuntu.com $(lsb_release -cs)-proposed main restricted universe multiverse" | \
   sudo tee -a /etc/apt/sources.list.d/ddebs.list
   ```

   The last three repositories may not be needed on a testing (alpha, beta daily build etc) releases.

   You can also add these repositories in your software sources from the Ubuntu software center or from synaptic. (see this article especially the section on adding other repositories) You will need to add lines like

   ```
   deb http://ddebs.ubuntu.com trusty main restricted universe multiverse
   ```

2. Import the debug symbol archive signing key from the Ubuntu server. On Ubuntu 18.04 LTS and newer:

   Esconder número das linhas

   ```
   sudo apt install ubuntu-dbgsym-keyring
   ```

On earlier releases of Ubuntu use:

Esconder número das linhas

```
1 sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys
F2EDC64DC5AEE1F6B9C621F0C8CAB6595FDFF622
```

Note: The GPG key expired on 2021-03-21 and may need updating by either upgrading the ubuntu-dbgsym-keyring package or re-running the apt-key command. Please see Bug #1920640 for workaround details if that does not work.

3. Then run the following to update your package list or click the *Reload* button if you used the Synaptic Package Manager.

Esconder número das linhas

```
1 sudo apt-get update
```

Manual install of debug packages

If the package in question is the *xserver-xorg-core* package you can try

```
sudo apt-get install xserver-xorg-core-dbg
```

or

```
sudo apt-get install xserver-xorg-core-dbgsym
```

Automatic Resolution / Installation of necessary packages

The above procedure will install the debug symbol package for a single package only. Chances are that the binary uses shared libraries in other packages and debug symbols for them might be required in order to obtain a readable stack trace or perform other debugging tasks.

You can use the 'find-dbgsym-packages' command from the 'debian-goodies' package to find debug symbols for a core file, running PID or binary path.

For a binary path it only finds debug symbols for the actual binary itself, and not any dynamically linked library dependencies or other libraries loaded at runtime. For that functionality to work you need to use either a core file or running PID which is the preferred method.

This tool will find both "-dbg" and "-dbgsym" style packages. However it only finds debug symbols for apt repositories that are currently enabled and updated, so you need to ensure that you enable at least the "ddebs.ubuntu.com" archive as described elsewhere on this page and for a Launchpad PPA or the Ubuntu Cloud Archive you need to add another source line with the component changed from "main" to "main/debug".

```
apt install debian-goodies
find-dbgsym-packages [core_path|running_pid|binary_path]
```

If the above tool is not working for you, there is an older shell script that attempts to do this by parsing the gdb output instead however often does not work correctly.

You can download the following shell script (list-symbols-packages-v2.1.sh) to resolve all the dependencies. Attaching a debugger to an already running process may require elevated privileges even if you own the process. The following invocation will print out the list of files to install.

```
sudo bash ./list-symbols-packages-v2.sh -p $(pidof yelp)
```

To automatically install them apt can be called with the input from the script. The script is invoked with `-t` for a terse output without the descriptions of the packages and error messages are suppressed:

```
sudo bash ./list-symbols-packages-v2.sh -t -p $(pidof yelp) 2>/dev/null | xargs -d $'\n' sudo apt-get install
```

Version 2 of this script makes it compatible with the newer GDB (which no longer loads all libraries at startup by default). -v2 currently only works if you run it against a currently-executing binary (i.e., with '-p $(pidof <whatever>'). The older version of the script is also available.

CategoryBugSquad CategoryDebugging

Debug Symbol Packages (editada pela última vez em 2021-03-21 08:41:32 por lathiat @ localhost[127.0.0.1]:lathiat)