Ceylan's HOWTO

HOW-TO

Contact: about (dash) howtos (at) esperide (dot) com

Creation date: Wednesday, November 17, 2021

Lastly updated: Sunday, December 19, 2021

Version: 0.0.1

Status: In progress

Dedication: Users of these HOWTOs

Abstract: The role of these `HOW-TOs` is, akin to a cookbook, to share a collection of (technical) recipes ("how-to do this task?) regarding various topics.

These elements are part of the Ceylan umbrella project.

The latest version of this documentation is to be found at the official Ceylan-HOWTOs website (`http://howtos.esperide.org`).

Table of Contents

The GNU/Linux Operating System **4**
- Overview . . . 4
- Software Update . . . 4
- Package Management . . . 6
  - Package-related Commands . . . 6
  - Interesting Packages . . . 7
- See Also . . . 7

Erlang **8**
- Overview . . . 8
- Let's Start with some Shameless Advertisement for Erlang and the BEAM VM . . . 8
- Installation . . . 9
- Ceylan's Language Use . . . 9
- Using the Shell . . . 10
- Language Implementation . . . 10
  - Message-Passing: Copying vs Sharing . . . 10
  - Just-in-Time Compilation . . . 10
  - Static Typing . . . 11
  - Intermediate Languages . . . 11
- About Security . . . 11
- Erlang Resources . . . 11

About 3D **13**
- Cross-Platform Game Engines . . . 13
  - Godot . . . 13
  - Unreal Engine . . . 13
  - Unity3D . . . 14
- 3D Data . . . 15
  - File Formats . . . 15
  - Samples . . . 16
- Modelling Software . . . 16
  - Blender . . . 16
  - Wings3D . . . 16
- Other Tools . . . 16
  - Draco . . . 16
- OpenGL Corner . . . 16
  - Hints . . . 16
  - Information . . . 16
- Operating System Support for 3D . . . 17
  - Testing . . . 17
  - Troubleshooting . . . 17
- Mini-Glossary . . . 17

Network Management **19**
Investigating Network Issues 19
Firewall Management 19
Configuration of a Gateway to the Internet 19
Firewall-related Troubleshooting 21
Network Troubleshooting 22
See Also 23

A Bit of Cybersecurity **24**
Pointers to various Security Topics 24
Securing thanks to OpenPGP 25
Purpose 25
Technical Solution 25
A Link With Decentralized Identifiers 32
Hints 32
See Also 33

About Build Tools **34**
Purpose of Build Tools 34
Choice 34
GNU make 34
See Also 35

Please React! **35**

Ending Word **35**

The GNU/Linux Operating System

Contact: about (dash) howtos (at) esperide (dot) com

Creation date: Sunday, December 19, 2021

Lastly updated: Sunday, December 19, 2021

Table of Contents

Overview **4**
Software Update **4**
Package Management **6**
Package-related Commands 6
Interesting Packages 7
See Also **7**

Overview

GNU/Linux is our operating system of choice, for many reasons: it is in free software, it is efficient, trustable, reliable and controllable, its mode of operation does not change much over time so any time invested on it is well spent.

Over the years we tried many distributions, including Ubuntu, Debian, Gentoo, Mint.

Our personal all-time favorite is clearly Arch Linux, because it leaves much control to its end user (not attempting to hide details that have to be mastered anyway), it is a "clean" one, driven by a skilled and knowledgeable community, and also because it is a rolling distribution: it updates constantly its packages *without needing to regularly upgrade the whole system*, which would jeopardise it in the same movement (global system updates rarely complete successfully and tend to be postponed because of the many problems they trigger; we found preferable to deal with issues incrementally on a live system - rather than on one that may fail to reboot properly).

It ends up with a very stable, hassle-free distribution, with cutting-edge packages and higher uptimes (several months without needing to reboot), which is desirable for server-like usages.

Software Update

The setup that we use is to perform **automatic nightly updates**. For that we use our update-distro.sh script, run through root's crontab as:

```
$ crontab -l
# Each day at 5:17 AM, update the distro:
17  05   *   *  * /usr/local/bin/update-distro.sh -q
```

As a result, all packages, libraries, executables, etc. are transparently updated, for the best.

However, for a proper management of modules[1], the kernel-related packages shall be special-cased; otherwise after the first kernel update no more modules

can be loaded (they will expect to link to that latest installed kernel version, not to the older one being running).

A first line of defense is to force the loading of the modules known to be of interest directly at boot-time, so that they can be for sure loaded and linked to the right kernel.

This may be done by populating `/etc/modules-load.d/` with as many files listing the modules to auto-load, like in:

```
::::::::::::::
/etc/modules-load.d/for-3g-keys.conf
::::::::::::::
option
usb_wwan

::::::::::::::
/etc/modules-load.d/for-all-usb-keys.conf
::::::::::::::
# To be able to mount all kinds of USB keys:
vfat
uas
dm_crypt

::::::::::::::
/etc/modules-load.d/for-mobile-file-transfer.conf
::::::::::::::
# To be able to transfer files between this hosts and mobile phones by MTP:
nls_utf8
isofs
sr_mod
cdrom
# Maybe also: agpgart ahci wdat_wdt wmi_bmof xts

::::::::::::::
/etc/modules-load.d/for-tty-serial-on-usb.conf
::::::::::::::
# To be able to connect tty-like interfaces through a USB port:
ftdi_sio
usbserial

::::::::::::::
/etc/modules-load.d/for-usb-tethering.conf
::::::::::::::
# To enable an Internet access thanks to a smartphone via USB:
usbnet
# Implies:
#rndis_host
#cdc_ether
```

[1] We tried to rely on DKMS for that, but had still issues with some graphic-related modules, so we preferred managing updates by ourselves.

```
::::::::::::::
/etc/modules-load.d/for-vlan-support.conf
::::::::::::::
# To be able to manage VLANs:
8021q
```

This is not sufficient though (ex: one cannot anticipate all modules needed after a while); disabling the *automatic* updates of kernels is also key to reduce issues; this can be done by specifying in `/etc/pacman.conf`:

```
IgnorePkg = linux linux-headers
```

LTS (*Long Term Support*) kernels are intentionally *not* listed here, as we prefer having them regularly updated in order to minimise the risk that the base and LTS kernels belong to too close versions (as then a problem in terms of hardware support is more likely to arise at the same time with both).

At least users of NVidia graphic cards may also list there their drivers, as apparently an hardware acceleration supported at boot may be lost after some time, presumably because of an update of its drivers (knowing that the update of the kernel itself was already disabled in that case) - so, if appropriate, better be safe than sorry:

```
IgnorePkg = linux linux-headers nvidia nvidia-utils
```

See also our section about operating system support for 3D.

Updating all packages but kernel-related ones is fine, but of course the latters shall still be also updated appropriately. The best moment for that is just prior to rebooting (knowing that your Linux box never crashes, isn't it?), so for that we use (as root) our shutdown-local-host.sh script, like in:

```
$ shutdown-local-host.sh --reboot
```

The kernel packages, and possibly driver-related ones, will then only be properly updated before the host is rebooted.

Package Management

Package-related Commands

- to get information about a package (installed or not): `pacman -Si MY_PACKAGE`
- to list all packages explicitly installed and not required as dependencies: `pacman -Qet`
- to determine which package installed a specified file: `pkgfile SOME_FILE`

See this page for many more examples.

Interesting Packages

They might be lesser known:

- `cpulimit`: the way of limiting CPU usage of a given process, for example to avoid overheat (`nice` just defines respective process priorities)
- `inotify-tools`: to be able to monitor filesystem events (ex: with `inotifywait`) from scripts
- `jq`: for command-line JSON processing (ex: `jq . myfile.json` to display it properly on a terminal)
- `mathjax`: to generate LaTeX-like images for the web
- `most`: a replacement for more
- `pdftk`: to transform PDF files
- `pkgfile`: to retrieve file information about packages

See Also

Our other mini-HOWTO regarding:

- Network Management
- Cybersecurity

Erlang

Contact: about (dash) howtos (at) esperide (dot) com

Creation date: Saturday, November 20, 2021

Lastly updated: Friday, December 17, 2021

Table of Contents

Overview **8**
Let's Start with some Shameless Advertisement for Erlang and the BEAM VM **8**
Installation **9**
Ceylan's Language Use **9**
Using the Shell **10**
Language Implementation **10**
Message-Passing: Copying vs Sharing 10
Just-in-Time Compilation 10
Static Typing 11
Intermediate Languages 11
About Security **11**
Erlang Resources **11**

Overview

Erlang is a concurrent, functional programming language available as free software; see its official website for more details.

Erlang is dynamically typed, and is executed by the BEAM virtual machine. This VM (*Virtual Machine*) operates on bytecodes and can perform Just-In-Time compilation. It powers also other related languages, such as Elixir and LFE.

Let's Start with some Shameless Advertisement for Erlang and the BEAM VM

Taken from this presentation:

Hint

What makes Elixir StackOverflow's #4 most-loved language?
What makes Erlang and Elixir StackOverflow's #3 and #4 best-paid languages?
How did WhatsApp scale to billions of users with just dozens of Erlang engineers?
What's so special about Erlang that it powers CouchDB and RabbitMQ?
Why are multi-billion-dollar corporations like Bet365 and Klarna built on Erlang?
Why do PepsiCo, Cars.com, Change.org, Boston's MBTA, and Discord all rely on Elixir?
Why was Elixir chosen to power a bank?
Why does Cisco ship 2 million Erlang devices each year? Why is Erlang used to control 90% of Internet traffic?

Installation

Erlang can be installed thanks to the various options listed in these guidelines.

Building Erlang from the sources of its latest stable version is certainly the best approach; for more control we prefer relying on our custom procedure.

For a development activity, we recommend also specifying the following options to our `conf/install-erlang.sh` script:

- `--doc-install`, so that the reference documentation can be accessed locally (in `~/Software/Erlang/Erlang-current-documentation/`); creating a bookmark pointing to the module index, located in `doc/man_index.html`, would most probably be useful
- `--generate-plt` in order to generate a PLT file allowing the static type checking that applies to this installation (may be a bit long and processing-intensive, yet it is to be done once per built Erlang version)

Run `./install-erlang.sh --help` for more information.

Once installed, ensure that `~/Software/Erlang/Erlang-current-install/bin/` is in your PATH (ex: by enriching your `~/.bashrc` accordingly), so that you can run `erl` (the Erlang interpreter) from any location, resulting a prompt like:

```
$ erl
Erlang/OTP 24 [erts-12.1.5] [source] [64-bit] [smp:8:8] [ds:8:8:10] [async-threads:1]

Eshell V12.1.5  (abort with ^G)
1>
```

Then enter `CTRL-C` twice in order to come back to the (UNIX) shell.

Congratulations, you have a functional Erlang now!

Ceylan's Language Use

Ceylan users shall note that most of our related developments (namely Myriad, WOOPER, Traces, LEEC, Seaplus, Mobile, US-Common, US-Web and US-

Main) depart significantly from the general conventions observed by most Erlang applications:

- notably because of their reliance on parse transforms, by default they rely on our own build system based on GNU make (rather than on rebar3)
- they tend not to rely on OTP abstractions such as `gen_server`, as WOOPER offers OOP (*Object-Oriented Programming*) ones that we prefer

Using the Shell

If it is as simple to run `erl`, we prefer, with Ceylan settings, running `make shell` in order to benefit from a well-initialized VM (notably with the full code path of the current layer and the ones below).

Refer then to the shell commands, notably for:

- `f/1`, used as `f(X).` in order to *forget* a variable `X`, i.e. to remove the binding of this variable and be able to (re)assign it afterwards
- `rr/{1,2,3}` (ex: used as `rr(Path).`) to *read records* and have them available on the shell; for example, to be able to use the records defined by xmerl:

```
1> rr(code:lib_dir(xmerl) ++ "/include/xmerl.hrl").
```

See also the JCL mode (for *Job Control Language*) to connect and interact with other Erlang nodes.

Language Implementation

Message-Passing: Copying vs Sharing

Knowing that, in functional languages such as Erlang, terms ("variables") are immutable, why could not they be shared between local processes when sent through messages, instead of being copied in the heap of each of them, as it is actually the case with the Erlang VM?

The reason lies in the fact that, beyond the constness of these terms, their life-cycle has also to be managed. If they are copied, each process can very easily perform its (concurrent, autonomous) garbage collections. On the contrary, if terms were shared, then reference counting would be needed to deallocate them properly (neither too soon nor never at all), which, in a concurrent context, is bound to require locks.

So a trade-off between memory (due to data duplication) and processing (due to lock contention) has to be found and at least for most terms (excepted larger binaries), the sweet spot consists in sacrificing a bit of memory in favour of a lesser CPU load. Solutions like persistent_term may address situations where more specific needs arise.

Just-in-Time Compilation

This long-awaited feature, named *BeamAsm* and whose rationale and history have been detailed in these articles, has been introduced in Erlang 24 and shall transparently lead to increased performances for most applications.

Static Typing

Static type checking can be performed on Erlang code; the usual course of action is to use Dialyzer - albeit other solutions like Gradualizer exist.

A few statically-typed languages can operate on top of the Erlang VM, even if none has reached yet the popularity of Erlang or Elixir (that are dynamically-typed).

In addition to the increased type safety that statically-typed languages permit (possibly applying to sequential code but also to inter-process messages), it is unsure whether such extra static awareness may also lead to better performances (especially now that the standard compiler supports JIT).

Intermediate Languages

To better discover the inner workings of the Erlang compilation, one may look at the eplaypen online demo (whose project is here) and/or at the Compiler Explorer (which supports the Erlang language among others).

Both of them allow to read the intermediate representations involved when compiling Erlang code (BEAM stage, erl_scan, preprocessed sources, abstract code, Core Erlang, Static Single Assignment form, BEAM VM assembler opcodes, x86-64 assembler generated by the JIT, etc.).

About Security

- one should not encrypt messages directly with a key pair (ex: with RSA, only messages up to around 200 bytes long can be encrypted): one should encrypt only a *symmetric key* (generated by a cryptographically-safe random algorithm) that is then used to encrypt one's message(s); ensure an Encrypt-Then-Authenticate scheme to prevent padding oracle attacks (and a secure-compare algorithm for the *Message Authentication Code* verification to prevent timing attacks); using libsodium should make mistakes using the standard crypto primitives less error-prone; see the enacl Erlang binding for that; for more information, refer to the corresponding thread
- relevant sources of information:
  - books:
    - `Cryptography Engineering: Design Principles and Practical Applications`
    - `Practical Cryptography`
  - the Security Working Group of the EEF (*Erlang Ecosystem Foundation*)

Erlang Resources

- the reference is the Erlang official website
- for teaching purpose, we would dearly recommend Learn You Some Erlang for Great Good!; many other high-quality Erlang books exist as well; one may also check the Erlang track on Exercism

- in addition to the module index mentioned in the Erlang Installation section, using the online search and/or Erldocs may also be convenient
- the Erlang community is known to be pleasant and welcoming to newcomers; one may visit the Erlang forums, which complement the erlang-questions mailing list
- for those who are interested in parse transforms (the Erlang way of doing metaprogramming), the section about The Abstract Format is esential (despite not being well known)
- to better understand the inner working of the VM: The Erlang Runtime System, a.k.a. "the BEAM book", by Erik Stenman

About 3D

Organisation: Copyright (C) 2021-2021 Olivier Boudeville

Contact: about (dash) howtos (at) esperide (dot) com

Creation date: Saturday, November 20, 2021

Lastly updated: Sunday, December 19, 2021

Table of Contents

Cross-Platform Game Engines **13**
Godot 13
Unreal Engine 13
Unity3D 14
3D Data **15**
File Formats 15
Samples 16
Modelling Software **16**
Blender 16
Wings3D 16
Other Tools **16**
Draco 16
OpenGL Corner **16**
Hints 16
Information 16
Operating System Support for 3D **17**
Testing 17
Troubleshooting 17
Mini-Glossary **17**

As usual, these information pertain to a GNU/Linux perspective.

Cross-Platform Game Engines

The big three are Godot, Unreal Engine and Unity3D.

Godot

Godot is our personal favorite engine, notably because it is free software (released under the very permissive MIT license).

See its official website and its asset library.

Installation On Arch Linux: `pacman -Sy godot`.

Unreal Engine

Another contender is the Unreal Engine, a C++ game engine developed by Epic Games; we have not used it yet.

Its licence is meant to induce costs only when making large-enough profits.

See its official website and its marketplace.

Assets Purchased assets may be used in one's own shipped products (source) and apparently at least usually no restrictive terms apply.

Assets not created by Epic Games can be used in other engines unless otherwise specified (source).

Unity3D

Unity is most probably the cross-platform game engine that is the most popular.

Regarding the licensing of the engine, various plans apply, depending notably on whether one subscribes as an individual or a team, and on one's profile, revenue and funding.

See its official website and its asset store.

Unity may be installed at least in order to access its asset store, knowing that apparently an asset purchased in this store may be used with any game engine of choice. Indeed, for the standard licence, it is stipulated in the EULA legal terms that:

Licensor grants to the END-USER a non-exclusive, worldwide, and perpetual license to the Asset to integrate Assets only as incorporated and embedded components of electronic games and interactive media and distribute such electronic game and interactive media.

So, in legal terms, an asset could be bought in the Unity Asset Store and used in Godot, for example - provided that its content can be used there without too much technical effort/constraint.

Installation Unity shall now be obtained thanks to the Unity Hub.

On Arch Linux it is available through the AUR, as an AppImage; one may thus use: `yay -Sy unityhub`.

Then, when running (as a non-priviledged user) `unityhub`, a Unity account will be needed, then a licence, then a Unity release will have to be added in order to have it downloaded and installed for good, covering the selected target platforms (ex: Linux and Windows "Build Supports").

Additional information: Unity3D on Arch.

Configuration Configuring Unity so that its interface (mouse, keyboard bindings) behave like, for example, the one of Blender, is not natively supported.

Assets They can be downloaded through the `Window -> Package Manager -> Packages:My Assets` menu option.

To access the assets provided by such packages, of course the simplest approach is to use the Unity editor; this is done by creating a project (ex: `MyProject`), selecting the aforementioned menu option (just above), then clicking on `Import` and selecting the relevant content that will end up in clear form in your project, i.e. in the UNIX filesystem with their actual name and content, for example in `MyProject/Assets/CorrespondingAssetProvider/AssetName`.

Yet such Unity packages, once downloaded (whether or not they have been imported in projects afterwards) are files stored typically in the `~/.local/share/unity3d/Asset Store-5.x` directory and whose extension is `.unitypackage`.

Such files are actually `.tar.gz` archives, and thus their content can be listed thanks to:

```
$ tar tvzf Foobar.unitypackage
```

Inside such archives, each individual package resource is located in a directory whose name is probably akin to the checksum of this resource (ex: `167e85f3d750117459ff6199b79166fd`)[2]; such directory generally contains at least 3 files:

- `asset`: the resource itself, renamed to that unique checksum name, yet containing its exact original content (ex: the one of a Targa image)
- `asset.meta`: the metadata about that asset (file format, identifier, timestamp, type-specific settings, etc.), as an ASCII, YAML-like, text
- `pathname`: the path of that asset in the package "virtual" tree (ex: `Assets/Foo/Textures/baz.tga`)

When applicable, a `preview.png` file may also exist.

Some types of content are Unity-specific and thus may not transpose (at least directly) to another game engine. This is the case for example for materials or prefabs (whose file format is relatively simple, based on YAML 1.1).

Tools like AssetStudio (probably Windows-only) strive to automate most of the process of exploring, extracting and exporting Unity assets.

Meshes are typically in the FBX (proprietary) file format, that can nevertheless be imported in Blender and converted to other file formats (ex: gltF 2.0); see blender import and blender convert for that.

3D Data

File Formats

They are designed to store 3D content (scenes, nodes, vertices, normals, meshes, textures, materials, animations, skins, cameras, lights, etc.).

We prefer to rely on the open, well-specified, modern glTF 2.0 format in order to perform import/export operations.

It comes in two forms:

- either as `*.gltf` when JSON-based, possibly embedding the actual data (vertices, normals, textures, etc.) as ASCII base64-encoded content, or referencing external files
- or as `*.glb` when binary; this is the most compact form, and the one that we recommend especially

See also the glTF 2.0 quick reference guide and the related section of Godot.

The second best choice we see is Collada (`*.dae` files), an XML-based counterpart (open as well, yet older and with less validating facilities) to glTF.

Often, assets can be found as FBX of OBJ files and thus may have to be converted (typically to glTF), which is never a riskless task.

Refer to blender import in order to handle the most common 3D file formats.

[2] Yet no checksum tool among `md5sum`, `sha1sum`, `sha256sum`, `sha512sum`, `shasum`, `sha224sum`, `sha384sum` seems to correspond; it must a be a different, possibly custom, checksum.

Samples

Here are a few samples of 3D content (useful for testing):

- glTF, notably glTF 2.0; direct: .gltf Buggy example, .glb Fish example (also: a simple cube)
- DAE; direct: Duck example (also: a simple cube)
- FBX; direct: Stylized character
- IFC; direct: Basic house (requires the BlenderBIM add-on for BIM support in Blender)

Modelling Software

Blender

Blender is a very powerful open-source 3D toolset.

We recommend the use our Blender scripts in order to:

- import conveniently various file formats in Blender, with `blender-import.sh`
- convert directly on the command-line various file formats (still thanks to a non-interactive Blender), with `blender-convert.sh`

Wings3D

Wings3D is a nice, Erlang-based, free software subdivision modeler.

Other Tools

Draco

Draco is an open-source library for compressing and decompressing 3D geometric meshes and point clouds.

It is intended to improve the storage and transmission of 3D graphics; it can be used with glTF, with Blender, with Compressonator, or separately.

OpenGL Corner

Hints

OpenGL allows the main program running on the CPU to communicate with typically a graphic card. As such most of the calls performed by user programs are asynchronous: through OpenGL they are triggered by the program and return almost immediately, whereas they have not been executed yet; they have just be queued. Indeed OpenGL implementations are almost always pipelined, so the rendering must be thought as primarily taking place in a background process.

Information

- FAQ for OpenGL and GLUT
- About OpenGL Performance

Operating System Support for 3D

Benefiting from a proper 2D/3D hardware acceleration on GNU/Linux is unfortunately not always straightforward, and sometimes brittle.

Testing

First, one may check whether such acceleration is already available by running, from the command-line, the `glxinfo` executable (to be obtained on Arch Linux thanks to the `mesa-utils` package), and hope to see, among the many displayed lines, `direct rendering: Yes`.

One may also run our display-opengl-information.sh script to report relevant information.

A final validation might be to run the `glxgears` executable (still obtained through the `mesa-utils` package), and to ensure that a window appears, showing three gears properly rotating.

Troubleshooting

If it is not the case (no direct rendering, or a GLX error being returned - typically involving any `X Error of failed request: BadValue` for a `X_GLXCreateNewContext`), one should investigate one's configuration (with `lspci | grep VGA`, `lsmod`, etc.), update one's video driver on par with the current kernel, reboot, sacrifice a chicken, etc.

If using a NVidia graphic card, consider reading this Arch Linux wiki page first.

In our case, installation could be done with `pacman -Sy nvidia nvidia-utils` but requested a reboot.

Despite package dependencies and a not-so-successful attempt of using DKMS in order to link kernel updates with graphic controller updates, too often a proper 3D support was lost, either from the boot or afterwards. Refer to our software update section for hints in order to secure the durable use of proper drivers.

Mini-Glossary

- **HDRP**: *High Definition Render Pipeline*, a high-fidelity scriptable render pipeline, made by Unity to target modern (Compute Shader compatible) platforms (so HDRP is the high-end counterpart of URP)
- **IK**: *Inverse Kinematics*, the **computation of intermediary joint parameters** so that the end of the kinematic chain is at a given position and orientation; typically, if one wants the hand of a character to grasp the top of a chair, IK is used in order to determine the parameters of the character's wrist, arm, elbow, etc. to retain so that the hand is ultimately correctly placed on the chair (more information)
- **PSD**: *Photoshop Document*, a proprietary format for graphics with layers, masks, etc. used by Adobe Photoshop (a commercial counterpart to Gimp, Krita, etc.) often used to store textures that may still be edited as templates by the user - provided they are using Photoshop as well;

however, at least to some extent, Gimp is able to edit PSD files and Krita too

- **Rigging** (or *Skeletal Animation*) consists in controlling the deformation of a mesh (a.k.a. a *skin*, the surface of a body) of an articulated object (typically a character) **based on a virtual inner armature** (a hierarchical set of interconnected parts, called *bones*, and collectively forming the skeleton or *rig*) in order to animate that mesh (more information)
- **Texture Atlas**: a texture containing a **set of separate, elementary graphic elements**, meant to be extracted thanks to texture coordinates, akin to a sprite sheet; doing so is useful to reduce the overhead that would be induced by the management of many smaller textures (more information)
- **URP**: *Universal Render Pipeline*, a prebuilt scriptable render pipeline, made by Unity that implements workflows across a range of platforms, from mobile to high-end consoles and PC (so URP is the low-end counterpart of HDRP)

Network Management

Contact: about (dash) howtos (at) esperide (dot) com

Creation date: Saturday, November 20, 2021

Lastly updated: Friday, December 17, 2021

Table of Contents

Investigating Network Issues **19**
Firewall Management **19**
Configuration of a Gateway to the Internet 19
Firewall-related Troubleshooting 21
Network Troubleshooting **22**
See Also **23**

Investigating Network Issues

Tools like `ping`, `traceroute`, `drill`, `arp`, etc. are invaluable.

Use ip-scan.sh to scans all IPs with any specified prefix, and ip-examine.sh to collect information about a given IP.

Use monitor-network.sh to investigate unstable connections.

Firewall Management

On GNU/Linux, some level of knowledge about `iptables` is useful, notably if exposing a computer to the Internet; note though that it is to be superseded by nftables.

One should read first the very clear Arch wiki section about iptables basic concepts.

A general rule that we retain, especially for an Internet gateway, is to drop all packets by default, and then only to accept the expected ones explicitly and carefully.

Configuration of a Gateway to the Internet

Our iptables.rules-Gateway.sh script sets up an iptables configuration with various services that can be enabled (ex: for masquerading, IPTV, different kinds of servers) as an example that we hope is secure enough[3].

This script expects a settings file to be available as `/etc/iptables.settings-Gateway.sh` (this file is meant to be sourced, not executed).

An example thereof:

```
# Local firewall settings.
#
# Meant to be sourced by the iptables.rules-Gateway.sh script.
```

[3]Please email us if you found otherwise! Refer to the top of this document for that.

```
# Where firewall-related outputs will be written:
log_file=/root/.last-gateway-firewall-activation

# Local (LAN) interface, the one we trust:
#lan_if=eth1
lan_if=enp2s0

# Internet (WAN) interface, the one we distrust:

# For PPP ADSL connections:
#net_if=ppp0

# For direct connection to a set-top (telecom) box from your provider:
#net_if=eth0
net_if=enp4s0

ban_file="/etc/ban-rules.iptables"

# As the IPs banned through the ban file above are quite minimal:
use_ban_rules="true"
#use_ban_rules="false"

# IP of a test client (to avoid too many logs, selecting only related events):
#test_client_ip="xxx"

# Enabled input TCP port range for traffic from LAN to gateway:
enable_unfiltered_tcp_range="true"

# TCP unfiltered window (ex: for passive FTP and BEAM port ranges):
tcp_unfiltered_low_port=50000
tcp_unfiltered_high_port=55000

# Tells whether IPTV (TV on the Internet thanks to a box) should be allowed:
enable_iptv=false

# Tells whether a SMTP server can be used:
enable_smtp=false

# Typically a set-top box from one's ISP (defined as a possibly log match
# criteria):

# Classical example:
telecom_box="192.168.0.254"

# DHT subsection, for P2P exchanges:
# More infos: https://github.com/rakshasa/rtorrent/wiki/Using-DHT

dht_udp_port=7881

#use_dht="true"
```

```
use_dht="false"

# One may use a non-standard port:
#ssh_port=22
ssh_port=22320

smtp_port=25

# SMTPS is obsolete:
smtp_secure_port=465

# STARTTLS over SMTP is the proper way of securing SMTP:
msa_port=587

pop3_port=110

# POP3S:
pop3_secure_port=995

imap_port=143
imap_secure_port=993
```

A script to configure iptables is best integrated to systemd, see the iptables.rules-Gateway.service file for that (typically to be placed in `/etc/systemd/system`). Then one may test with:

```
$ systemctl start iptables.rules-Gateway.service
```

and enable it for good with:

```
$ systemctl enable iptables.rules-Gateway.service
```

Note that often these scripts are setup remotely, while being connected thanks to SSH from another host. Care must be taken in order not to lock oneself out of the target server, notably when updating rules (this happens quite easily). We advise to prefer the `restart` option of our iptables script in order to reduce the risk of "bricking" one's server.

Firewall-related Troubleshooting

Use iptables-inspect.sh to list the currently-used firewall rules for the chains of the main tables. Like `iptables -nL --line-numbers`, it displays the number of each rule of a given chain, which allows to add/remove rules more easily, like in:

```
# Deletes the first rule of the FORWARD chain (of the 'filter' table):
# (note that all the next rules will bear a decremented number afterwards!)
$ iptables -D FORWARD 1
```

Setting environment variables (either through files such as `/etc/iptables.settings-Gateway.sh` or directly in the shell) is less error-prone; ex:

```
[...]
$ lan_if=enp2s0
$ net_if=enp4s0
$ iptables -I FORWARD -i ${lan_if} -o ${net_if} -d ${telecom_box} -j LOG
$ journalctl -kf --grep="IN=.*OUT=.*" | grep -v "SRC=${telecom_box}"
```

To further match packets, one may specify log prefixes, like in:

```
$ iptables -A INPUT -i lan.foobar -j LOG --log-prefix "[VLAN INP FOO]"
```

Note that the `LOG` target does not intercept a packet, which thus continues to flow in the next rule(s). so log targets are better defined as first rules (and thus could be inserted lastly).

As a reminder, for a given table (`filter` by default), rules may be:

- appended *at the end* of the selected chain with `-A`
- inserted either at the *beginning* of the selected chain with `-I`, or at its position N with `-I N`

See also the iptables section in the Arch wiki.

Network Troubleshooting

A few pieces of advice/information:

- be familiar with `ip link`, `ip addr` and `ip route` (generally used in that order), and `tcpdump` for the worst cases
- nowadays, many devices change their MAC address regularly, like smartphones do
- one may rely on `netctl`, and create as many profiles as found useful
- regularly inspect network-related messages (ex: with `journalctl -kf`) to detect anomalies such as `IPv4: martian source 192.168.0.49`
- interfaces may be associated to any number of IP addresses, this may create surprises
- when a network does not work properly, always consider that this device may be faulty, that cables may malfunction, and that power supplies may be culprits
- having smart switches may help a lot, to better control one's network (ex: disabling ports, checking statuses, isolating sections, etc.)
- beware to DHCP server(s) being left unnoticed; various devices may use them to get a random address and become difficult to spot
- netmasks shall not be neglected, for example in routes:

```
$ ip route add 192.168.0.0/16 dev enp4s0 scope link
$ ip route
default via 192.168.0.254 dev enp4s0 proto dhcp src 192.168.0.1 metric 1002
10.0.0.0/8 dev enp2s0 proto kernel scope link src 10.0.0.1
192.168.0.0/16 dev enp4s0 scope link
```

Here for example, in `192.168.0.0/16`, `16` corresponds to the length of *the network prefix*; the next 16 bits are left to designate hosts, whose addresses therefore range in `192.168.[0..254].[1..254]`. So `192.168.0.0/16` includes the `192.168.27.0/24` network, whereas `192.168.0.0/24` would not.

- go for VLAN only when having reached a first level of correct operation; note that some devices (ex: non-manageable switches) are not able to handle VLAN-tagged packets and may reject or overwrite this information
- in some cases, hard reboots / returns to factory settings will fix inexplicable situations; updating to latest firmware may help too (network appliances *do* have bugs as well!)
- secure spare parts (if possible all cables, fibers, devices, power supply, etc. shall exist at least in two copies, tested just after purchased): when the one in operation will fail, the outage will be quickly solved by switching element; the troubleshooting will be easier as well: replace the whole set of equipment, check that everything works again, and try to progress by dichotomy (change half of the elements, and check whether everything remains functional)
- purchase only equipment of quality, and treat it gently (ex: use an Uninterruptible Power Supply providing good-quality current)
- take notes about the operations that are performed, the detected issues and the current configuration, and put the whole in VCS
- check temperature, ventilation and prevent dust accumulation
- consider monitoring temperatures, fans, availability, performances

See Also

- Ceylan-Hull's section about scripts for network management and for firewall configuration
- A bit of Cybersecurity

A Bit of Cybersecurity

Organisation: Copyright (C) 2021-2021 Olivier Boudeville

Contact: about (dash) howtos (at) esperide (dot) com

Creation date: Saturday, November 20, 2021

Lastly updated: Saturday, December 18, 2021

Table of Contents

Pointers to various Security Topics **24**
Securing thanks to OpenPGP **25**
Purpose 25
Technical Solution 25
Obtaining One's Keys 25
Where are the keys, and how to backup them? . . . 27
How Can Public Keys be Shared? 29
What can be done with these keys? 31
A Link With Decentralized Identifiers **32**
Hints **32**
See Also **33**

Pointers to various Security Topics

A goal here is to favor cryptographic privacy and authentication for data communication.

More precisely:

- for **data storage** (be it a USB key or a SSD disk), it may translate to partition encryption, typically with LUKS2 and cryptsetup
- **individual files** may be encrypted/decrypted with the help of appropriate scripts; see also Ceylan-Myriad's support for additional basic, old-school ciphering
- for the **management of credentials** such as passwords, some Ceylan-Hull scripts may be of help, including for the generation of proper passwords or for the locking of screens
- regarding **network**, each host may be protected by a relevant firewall configuration, opened ports may be checked, etc.; see also our section for firewall management
- for **webservers**, it relates to use the HTTPS protocol with proper X.509 security certificates for TLS-secured exchanges, possibly thanks to Ceylan-LEEC
- for **emails**, see the next section about OpenPGP

Securing thanks to OpenPGP

Purpose

Albeit such a securing scheme may apply to at least most of the digital exchanges, in practice it is mainly used in the context of email security.

In the general case, sending an email will end up having its content stored at least on:

- your disk
- a disk of one of the servers of your Internet provider
- a disk of a server of the provider of the recipient
- the recipient's host

Possibly with intermediate organisations between the endpoint ones, possibly stored on several locations per organisation - possibly times the number of specified recipients.

Moreover many countries require by law that emails are stored by Internet providers durably (often at least for one year) - not to mention the large-scale data harvesting that many countries perform, officially or not, with their own measures, on their own territory or on the one of others.

That's a rather large number of copies for one's private correspondence - to the point that emails sent in clear text could be mostly considered as public. Not to mention that they could also be altered in the process, at some point(s) in the chain.

Encrypting and signing are solutions to restore some privacy and safety - yours, but also the ones of the persons with whom you happen to correspond.

Technical Solution

It is currently best done thanks to the OpenPGP open standard for encrypting, signing and decrypting data and communications.

GnuPG (*GNU Privacy Guard*) is a complete and free implementation of it (we suppose here that at least its 2.2.* version is used).

The corresponding command-line executable, `gpg`, can be installed on Arch Linux with: `pacman -Sy gnupg`.

Obtaining One's Keys The first step is to generate locally one's key pair, knowing that each public key is bound to a username or an e-mail address (which is our preference; having one's domain name allows to create any number of them).

A nice feature of this cryptographic scheme is that one may issue any number of keys in full autonomy and with neither consequences nor cost. So as many key pairs as notions of "unrelated identities" may be freely created.

Several settings can be chosen when generating a key, and logically the strongest keys are preferred. Yet uncommon/too recent generation algorithms and/or higher key lengths may not be supported by the various tools[4], so applying the default settings retained by gpg, or similar ones yet a bit stronger

(ex: at the time of this writing, November 2021, RSA 4096 bits rather than 3072 bits) is probably the way to go (it can already be deemed safe, and will be widely supported); so the generation may be best triggered simply thanks to:

```
# For current defaults:
$ gpg --gen-key

# Or, for more control:
$ gpg --full-gen-key
```

If preferring rather paranoid settings, presumably for an extra security/durability, one can select ECC (for Elliptic-curve cryptography), with the `Sign`, `Certify` and `Authenticate` capabilities enabled (even if authentification is not used by many common protocols), and opt for the `Brainpool P-512` curve through:

```
$ gpg --full-gen-key --expert
```

In all cases, one may enter `1y` to set the initial validity duration of the generated key to one year, and already plan in one's agenda, a dozen days before the end of its validity, its renewal.

Then one may enter one's selected identity (ex: for `Real name`, one may enter `James Bond`), one's email address of interest (ex: `james.bond@mi6.org`) and possibly:

- either no specific comment (they are not normalised anyway)
- or one pointing to an authoritative source against which the public key may be verified (such as: "This public key can be verified against its reference in https://mi6.org/james-bond.pub" - provided of course such a file is to exist)

The requested passphrase only consists on a last-resort protection of the generated private key (that you should *never* transmit to *anyone*), in order to avoid that anyone accessing this file on your computer becomes directly able to fully impersonate this identity.

The operation generates a public/private key pair, and also an associated emergency revocation certificate, so that you can invalidate it at any time and for any reason:

```
gpg: key 9A60ADA4E151B8B5 marked as ultimately trusted
gpg: directory '/home/james/.gnupg/openpgp-revocs.d' created
gpg: revocation certificate stored as '/home/james/.gnupg/openpgp-revocs.d/C3987680AD9
public and secret key created and signed.

pub   brainpoolP512r1 2021-11-26 [SCA] [expires: 2022-11-26]
      C3987680AD9B79FDC6B7D25C9D60ADA5E115A8B5
uid   James Bond <james.bond@mi6.org>
```

[4] With "cutting-edge" settings, some tools (like Thunderbird) on your side and/or the email clients of your recipients may be unable to make use of the resulting keys, and may fail to report clearly that they actually do not support this algorithm or its parametrisation. So one may consider sticking to the reasonable gpg defaults.

Here `C3987680AD9B79FDC6B7D25C9D60ADA5E115A8B5` is the full fingerprint of the public key; it could be shortened to its 8, if not 4, last characters (long/short ID), yet it would expose to the forging of intentionally-colliding keys, so one should only designate a key based on its full fingerprint, and forget unsafe abbreviations.

The public key can be freely shared, whereas the private one and the revocation certificate must be equally well protected (preferably in different places).

The only well-known threats to these keys are either a flaw (intentional loophole or accidental weakness) in the cryptographic algorithms on which they rely, or the advent of major research progresses such as quantum computing. Yet it still remains possible for one to "upgrade" one's key with newer algorithms (a new key superseding an older one that is to be revoked afterwards), so as always it will be a never-ending struggle between the spear and the shield, i.e. attack and defense.

As signing and encrypting correspond to different use cases, having different keys for each may make sense. But instead of generating two unrelated keys, one shall create:

- first an infrequently-used, very-well protected (hence less accessible), signing-only "master" (primary) key of longer validity (one's actual identity)
- then at least two subkeys (deriving from the previous one, yet autonomous) may be of use:
  - one for everyday encrypting; a proper subkey has already been automatically created and used by gnupg
  - an extra one for everyday signing: such a subkey may be created with a sufficient lifespan so that past signatures can be durably verified

These "derived" subkeys are meant to change more frequently, to be able to be revoked independently, and thus are safer to expose in less secure systems.

Use `gpg --edit-key` and `addkey` in order to add a subkey to a key, and refer to this section to export the subkey.

See also these very relevant Debian guidelines for further information about subkey management.

Where are the keys, and how to backup them? The full gpg state is stored by default in its `~/.gnupg/` tree.

One may notably notice in it:

- the private keys, whose extension is `.key` and whose security is of course of paramount importance
- the revocation certificates, whose extension is `.rev`, in order to revoke one's corresponding key pair (as important as the related private key)
- certificate revocation lists, to consider that the corresponding certificates are valid yet shall *not* be trusted
- the sets of keys ("rings") containing the public keys that have been transmitted to you, gathered according to the level of trust that you dedicated to them

The public keys are usually given a `.pub` extension[5].

Even if a backup of one's key pair could be made by creating and encrypting an archive of this gpg filesystem tree, a far better solution is to use its integrated procedure, as the structure of its internal state may change from a version/platform of gpg to another. So the best course of action is to use the following command in order to generate a backup of a key pair in a standard, durable form:

```
$ gpg -o $(date '+%Y%m%d')-full-key-backup-for-james.bond-at-mi6.org.gpg --export-secr
```

This will produce a half-kilobyte file containing the full key pair, whose type is:

```
20211126-full-key-backup-for-james.bond-at-mi6.org.gpg: OpenPGP Secret Key Version 4,
```

Of course, so that it may be used in the future, this backup of (notably) the private key should *not* be encrypted with that same key.

Specifying in filenames the email address may be avoided, in the sense that rather than having multiple keys (ex: as many as email accounts), it is often more convenient to have a single key supporting multiple names/addresses (see the section about subkey below); so:

```
# If using fingerprints and potentially having multiple registered email
# accounts,just focusing on their common identity:
#
$ gpg -o $(date '+%Y%m%d')-full-key-backup-for-james.bond.gpg --export-secret-keys C39
```

A backup of the revocation certificate shall be done as well (knowing that by design it is not password-protected, and thus having access to this certificate is sufficient to be able to kill your key), preferably in a different location as the role of this certificate is to serve as an urgent safety measure should the private key be lost (non-emergency revocations should be performed thanks to the more adapted and informative `--generate-revocation` option instead).

For long-term auxiliary storage, such a backup can be printed (on paper), possibly thanks to Paperkey (installed on Arch with `pacman -Sy paperkey`). For example:

```
# To print directly:
gpg --export-secret-key my_key_fingerprint | paperkey | lpr

# To store first (less secure):
gpg --export-secret-key my_key_fingerprint | paperkey --output my_key_fingerprint.asc
```

Such exports are ASCII texts, but they can also take the perhaps more convenient (and maybe less secured if having to trust one's smartphone) form of a QR code:

[5] Other common extensions are `.gpg` (for encrypted content and also standard signatures), `.asc` (for clear-text signatures and other ASCII content), and `.sig` (for detached signatures).

```
$ gpg --export-secret-key my_key_fingerprint | paperkey --output-type raw | qrencode
```

Besides key pairs, following backups shall be done:

- the known public keys, thanks to: `gpg -o $(date '+%Y%m%d')-known-public-keys.gpg --export`
- the associated level of trust (level per public key): `gpg --export-ownertrust > $(date '+%Y%m%d')-openpgp-trust.txt`

How Can Public Keys be Shared? As mentioned, public keys can be freely shared without involving any specific risk, as in practice a private key cannot be derived from its public counterpart.

So basically any means of sharing them is legit, including the least secured ones. However the point is that their recipients must be sure that they obtained the right public certificate, and not one that has been tampered with.

Indeed, any man-in-the-middle M between peers A and B able to intercept the communication of A's public key could replace it by his. B would then have no means of detecting that it is actually relying on M's keys rather than on A's ones.

So, on top of the generation of key pairs, a safe mechanism to share public ones shall be carefully considered, to establish the authenticity of the binding between a public key and its owner. Such mechanisms exist in two forms, peer-to-peer ones, or centralised ones.

Decentralised Sharing The Web of trust is a decentralized trust model, which - like Internet federates a large number of computer networks - is to federate trust networks.

A user may have multiple key pairs, and each of the corresponding public keys may be known of various trust networks.

The trust conceded by identity A to identity B means that A endorses the association of the public key of B with the person or entity listed in its certificate.

The goal is to enable the emergence of some level of global trust from the trust that each given identity concedes to the various identities that it knows directly.

Trust is indeed to be spread, by extending it from peer to peer (or friend to friend) in an increasingly large network of trust, typically with trust levels that decrease with the number of peers that have to be traversed in the network before reaching a given identity: you may trust friends of your friends, albeit probably a bit less than your direct friends; networks of trust may reflect that increasing risk, typically based on mean shortest distance between endpoints.

In practice, if A expresses some level of trust to B, A will digitally sign (thus with its own private key) the public certificate of B, to assess its association with the identity it embeds. This is commonly done at key signing parties (a nice way of meeting likely-minded folks as well).

Various schemes for vetting (validating in practice the identity carried by B; ex: should we request B to show their identity card, to prove they control a given domain, or any other identity/ownership proof?) and voting (to decide

on the overall trust to be derived from a potentially conflicting set of peer-to-peer endorsements A1, A2, etc. about B) exist; one remains of course free to decide for oneself on which grounds one concedes trust, it is the beauty of a decentralised mode of operation.

In practice, the sharing of public certificates used to be done through SKS key servers; it is as simple as requesting gpg to send the pblic key that corresponds to the specified fingerprint (here its last 8 characters):

```
$ gpg --send-keys E115A8B5
gpg: sending key 9D60ADA5E115A8B5 to hkps://keyserver.ubuntu.com
```

Note that this sharing discloses the corresponding email address, and thus exposes it to spam.

As various issues threaten SKS-based solutions, public keys may also be sent to the Hagrid-based OpenGPG server, `keys.openpgp.org` (which is not replicated to peer servers, yet performs more verification of the issuer of registered certificates).

To do so, register first this server in your configuration:

```
$ echo "keyserver hkps://keys.openpgp.org" >> ~/.gnupg/dirmngr.conf

# Reload gpg daemon:
$ gpgconf --reload dirmngr

# Extract the public key of interest in a .pub file:
$ gpg -o $(date '+%Y%m%d')-james.bond-at-mi6.org.pub --export james.bond@mi6.org
```

This file shall be uploaded via this web page that will guide you through the verification process, i.e. sending an email to the electronic address embedded in the transmitted public key in order to check that it is legit (by waiting for you to visit the URL that it generated and specified in said email).

More generally, various keyservers are looked up by gpg and thus can be considered (with different configurations regarding federation, verification, ability to forget keys, etc.).

Afterwards anyone will be able to search for such key:

```
$ gpg --search-keys james.bond@mi6.org
gpg: data source: https://keys.openpgp.org:443
(1)     James Bond <james.bond@mi6.org>
          512 bit ECDSA key 9A60ADA4E151B8B5, created: 2021-11-26
```

Of course checking that only one matches is returned is important to detect spoofing attempts.

Specifying your OpenPGP fingerprint in your email footers offers little interest, as your recipients cannot be sure that such incoming emails have not been tampered with.

So ultimately one will have either to trust such a decentralised scheme, or to trust a central authority like discussed next.

Centralised Sharing A centralized trust model is based on a Public Key Infrastructure (PKI, usually based on the X.509 standard), which relies exclusively on a Certificate Authority (CA), or more often a hierarchy of such: a CA's certificate may itself be signed by a different CA, all the way up to a self-signed root certificate.

So a certificate chain has to be validated, knowing that tools like browsers, and operating systems alike, come with their own keystore already comprising root certificates, and regularly updating them.

These certificates are well protected, yet any compromising thereof may jeopardise their whole "subtree".

Sharing Largely So a public certificate can be spread as widely as wanted, through key servers / PKIs, but also it should be shared through any reliable, authoritative reference of a given identity, like one's own webserver, emails, social accounts, etc.

This can be directly your public certificate (here is mine)[6] or a (shorter) fingerprint thereof (ex: the full fingerprint of my key is `DCA8E181DC3CEAF0EAE4033F9987EE77188E9BF4`).

Such public keys can be listed and then obtained respectively thanks to:

```
$ gpg --list-keys james.bond@mi6.org
pub   brainpoolP512r1 2021-11-26 [SCA] [expires: 2022-11-26]
      C3987680AD9B79FDC6B7D25C9D60ADA5E115A8B5
uid           [ultimate] James Bond <james.bond@mi6.org>

# For a binary version of the public key:
$ gpg -o james-bond.pub ---export C3987680AD9B79FDC6B7D25C9D60ADA5E115A8B5

# For an ASCII-based version (ex: suitable to register in GitHub):
$ gpg -o james-bond.pub.asc --armor --export C3987680AD9B79FDC6B7D25C9D60ADA5E115A8B5
```

What can be done with these keys? One may:

- **encrypt** a file: `gpg -r james.bond@mi6.org -e my_file_to_encrypt`; this generates a `my_file_to_encrypt.gpg` file
- **sign** a file, with three possibilities:
  - `--sign` / `-s` to generate a file containing both the input file (wrapped in an OpenPGP packet) and the signature
  - `--clear-sign` to generate a file containing both the input file (verbatim, expected to be a text file) and the signature
  - `--detach-sign` / `-b` to only generate a file containing said signature; so the input file will be needed in this mode to verify that signature; this possibility is useful when distributing content (ex: binaries), so that the intended public can check the signature if wanted

[6] Note the HTTPS protection and that it currently refers to `online.fr` rather than to `esperide.com`.

- **decrypt** and possibly in the same movement **check the signature** of a file: `gpg -d my_file_to_decrypt.gpg` (everything will be output to the standard stream)
- **verify** a signature: see the `--verify` option for the 3 types of signatures
- **verify** signed emails:
  - import the public key of the sender: `gpg --search-keys dr.no@foobar.org`
  - determine whether it is valid and, more importantly, deserving trust (is it the right public key?); if yes; sign it with `gpg --edit-key dr.no@foobar.org`
- **import** keys (yours or not) in your email client; if using a (recent) Thunderbird, no plugin is needed, but the local gpg rings will *not* be used by Thunderbird; refer to this documentation
- **encrypt** and/or **sign** emails

A Link With Decentralized Identifiers

The use of key pairs in the absence of a certificate authority directly relates to Decentralized Identifiers (DIDs), a class of universal solutions (not depending on any context/organisation, and able to be recognized by any) with which anyone can create one's (globally unique) identifiers that remain in one's full control: one freely issues them, they remain valid as long as their issuer wishes (as none but their creator itself can revoke them), and (for example unlike mere UUIDs) they can be cryptographically verified by anyone.

No external central authority applies to such identifiers, which cannot reveal personal information unless decided by their issuer and thus sole controller.

In practice, although other solutions could maybe be considered, it involves, like discussed in the previous sections, generating on one's own at least a public/private key pair, to store safely the private one and to share as widely as needed the public one. Then one can sign and/or encrypt one's messages with a pretty good hope that they will remain secure for a while; such a system enables partial disclosure (as one chooses what one encrypts or signs) in full control (as all operations are driven by the private key that the issuer is the only one to control).

These decentralised identifiers, together with the principle of addressing a digital content by its fingerprint (ex: SHA1), offer a solution bringing many interesting properties and opening new possibilities to distributed systems (ex: for blockchains, a user account is often identified by the fingerprint of its associated public certificate).

Hints

- whenever useful, add the `--armor` option to use ASCII output armor, suitable for copying and pasting content in text format
- if you have multiple email accounts, thanks to `--edit-key` you can add each one of themin the same key as an identity (name), using the `adduid` command; you can then set your favourite one as primary

- to always show full fingerprints of keys, add `with-fingerprint` to your configuration file (typically `~/.gnupg/dirmngr.conf`)
- these Debian guidelines describe a robust, well-defined process for key management that may apply to most developers

See Also

- a complete, well-written tutorial, in French: Bien démarrer avec GnuPG
- other interesting usage hints, still in French
- GnuPG on Arch, for much additional information
- Network Management information

About Build Tools

Organisation: Copyright (C) 2021-2021 Olivier Boudeville

Contact: about (dash) howtos (at) esperide (dot) com

Creation date: Saturday, November 20, 2021

Lastly updated: Friday, December 17, 2021

Table of Contents

Purpose of Build Tools **34**
Choice **34**
GNU make **34**
See Also **35**

Purpose of Build Tools

A build tool allows to automate all kinds of tasks, by **applying rules and tracking dependencies**: not only compiling, linking, etc. applications, but also checking them, generating their documentation, running and debugging them, etc.

Choice

Often build tools are tied to some programming languages (ex: Maven for Java, Rebar3 for Erlang, etc.).

Some tools are more generic by nature, like late GNU autotools, or Cmake, GNU make, etc.

For most uses, our personal preference goes to the latter. Notably all our Erlang-based developments, starting from Ceylan-Myriad, are based on GNU make.

GNU make

We recommend the reading of this essential source for reference purpose, notably the section about The Two Flavors of Variables.

Taking our Erlang developments as an example, their base, first layer, Ceylan-Myriad, relies on build facilities that are designed to be also reused and further adapted / specialised / parametrised in turn by all layers above in the stack (ex: Ceylan-WOOPER).

For that, Myriad defines three top-level makefiles:

- base build-related *variables* (settings) in GNUmakevars.inc, providing defaults that can be overridden by upper layers
- *automatic rules*, in GNUmakerules-automatic.inc, able to operate generically on patterns, typically based on file extensions
- *explicit rules*, in GNUmakerules-explicit.inc, for all specific named make targets (ex: `all`, `clean`)

Each layer references its specialisation of these three elements (and the ones of all layers below) in its own GNUmakesettings.inc file, which is the only element that each per-directory `GNUmakefile` file will have to include.

Such a system allows defining (build-time and runtime) settings and rules once for all, while remaining flexible and enabling individual makefiles to be minimalistic: beside said include, they just have to list which of their subdirectories the build should traverse (thanks to the `MODULES_DIRS` variable, see example).

See Also

asdf, an extendable version manager for various languages (Ruby, Node.js, Elixir, Erlang, etc.).

One may refer to the development section of Ceylan-Hull, or go back to the Ceylan-HOWTOs main page.

Please React!

If you have information more detailed or more recent than those presented in this document, if you noticed errors, neglects or points insufficiently discussed, drop us a line! (for that, use the contact address at the top of this document).

Ending Word

Hoping that these Ceylan-HOWTOs may be of help!

HOW-TO