Ceylan's HOWTO

HOW-TO

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

Creation date: Wednesday, November 17, 2021

Lastly updated: Wednesday, March 19, 2025

Version: 0.0.2

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`).

> **Note**
>
> This PDF document includes cross-references between HOWTOs, yet these links make sense only in the context of its HTML counterpart.

Table of Contents

Using the GNU/Linux Operating System

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

Creation date: Sunday, December 19, 2021

Lastly updated: Wednesday, March 19, 2025

Table of Contents

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[3], 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
```

[3] 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 (e.g. 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

Configuration

One may enable the multilib repository, which is useful to run 32-bit software on 64-bit hardware. This is useful for example if needing `wine`, knowing that its build from the AUR may fail.

To enable multilib, uncomment in `/etc/pacman.conf`:

```
[multilib]
Include = /etc/pacman.d/mirrorlist
```

then upgrade your system with `pacman -Syu`.

Package-related Commands

- to get information about a package (installed or not): `pacman -Si MY_PACKAGE`
- to list all packages:
  - installed (explicitly or not): `pacman -Q`
  - explicitly installed and not required as dependencies: `pacman -Qet`
- to determine which package installed a specified file:
  - on Arch: `pkgfile SOME_FILE`; `pkgfile` itself must have been installed beforehand, with `pacman -S pkgfile`, and be updated, with `pkgfile --update` (still as root)
  - on Debian: `apt-file search SOME_FILE` after a similar initial install thanks to: `sudo apt-get install apt-file && sudo apt-file update`
  - for many distros, one may rely on the command-not-found website
- to determine which package owns (would install) specified file(s): `pacman -Qo FILES`

See this page for many more Arch-related commands.

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 (e.g. with `inotifywait`) from scripts
- `jq`: for command-line JSON processing (e.g. `jq . myfile.json` to display it properly on a terminal)
- `yq`: for command-line YAML processing (e.g. `yq . myfile.yaml` to display it properly on a terminal), or use our validate-yaml.sh script
- `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

Systemd-related Hints

Systemd is the current reference system and service manager for GNU/Linux. It is in charge of configuring, launching, monitoring, controlling, etc. all the software services running on a given host.

Systemd Commands

- **listing** the units:
  - managed by systemd: `systemctl list-units [PATTERN]`
  - installed (as files): `systemctl list-unit-files [PATTERN]` or `tree /etc/systemd/system`
- **getting runtime status information** about units: `systemctl status [PATTERN]`
- **controlling** units: `systemctl start|stop|restart [PATTERN]`
- **reloading**:
  - a service-specific configuration of a unit: `systemctl reload [PATTERN]` (e.g. requesting Apache to reload its own httpd.conf file)
  - the systemd configuration file of a unit: `systemctl daemon-reload [PATTERN]` (e.g. reloading the `apache.service` systemd unit file)
- **enabling/disabling** for good units (while not starting/stopping them): `systemctl enable|disable [PATTERN]`

Systemd Journal

In order to query the contents of the systemd journal (as written by `systemd-journald.service`), the `journalctl` command may be used.

To consult the journal:

- from:
  - the oldest entry collected: `journalctl`
  - last boot: `journalctl -b`
  - yesterday: `journalctl --since yesterday`
  - a given duration: `journalctl --since "10 minutes ago"`
  - one or two time bounds: `journalctl --since "2023-02-10 21:00:00" --until "2023-02-10 22:00:00"`
  - the most recent journal entries, listing them to current end, and:
    - stopping there: `journalctl -e`
    - continuously printing the new entries as they are added: `journalctl -f` (or `--follow`)
- for a given unit `my_unit` (see Systemd Commands for selection): `journalctl -u my_unit`
- showing `COUNT` lines: add `-n COUNT` (default: 10)

So for example `journalctl -n 200 -fu my_unit` may be convenient to have recent history together with the next entries to come.

Process-related Post-Mortem Investigations

Sometimes a UNIX process crashes and, typically if one developed it, one wants to investigate the issue, based on a core dump produced by the operating system.

This Arch Linux article will give all relevant details.

In short, `coredumpctl list` will list all known core dumps from oldest to most recent, such as in:

```
$ coredumpctl list
TIME                            PID   UID  GID SIG     COREFILE EXE                SIZE
[...]
Tue 2021-12-21 20:53:02 CET 73873 1007 988 SIGSEGV present  [...]/bin/beam.smp 14.6M
```

The last core dump produced may be studied directly, thanks to `coredumpctl debug`, relying on `gdb` to fetch much lower-level information:

```
$ coredumpctl debug
           PID: 73873 (beam.smp)
           UID: 1007 (xxx)
           GID: 988 (users)
        Signal: 11 (SEGV)
     Timestamp: Tue 2021-12-21 20:53:01 CET (38min ago)
  Command Line: /home/xxx/Software/Erlang/Erlang-24.2/lib/erlang/erts-12.2/bin/beam.smp
    Executable: /home/xxx/Software/Erlang/Erlang-24.2/lib/erlang/erts-12.2/bin/beam.smp
 Control Group: /user.slice/user-1007.slice/session-2.scope
          Unit: session-2.scope
         Slice: user-1007.slice
       Session: 2
     Owner UID: 1007 (xxx)
       Boot ID: f8abe9473f7e4fea8ba24944e35ce7d9
    Machine ID: c9413a71e7b4498f831e2df7a08e5f33
      Hostname: xxx
       Storage: /var/lib/systemd/coredump/core.beam\x2esmp.1007.f8abe9473f7e4fea8ba2494
     Disk Size: 14.6M
       Message: Process 73873 (beam.smp) of user 1007 dumped core.

                Found module /home/xxx/Software/Erlang/Erlang-24.2/lib/erlang/erts-12.2
                Found module /home/xxx/Software/Erlang/Erlang-24.2/lib/erlang/lib/wx-2
[...]
    Stack trace of thread 74039:
    #0  0x00007f6e5461a74b __memmove_avx_unaligned_erms (libc.so.6 + 0x16374b)
    #1  0x00007f6d8a204428 n/a (iris_dri.so + 0xd12428)
    #2  0x00007f6d89733207 n/a (iris_dri.so + 0x241207)
    #3  0x00007f6d89733c97 n/a (iris_dri.so + 0x241c97)
    #4  0x00007f6d898d8b0d n/a (iris_dri.so + 0x3e6b0d)
    #5  0x00007f6d898d8bf2 n/a (iris_dri.so + 0x3e6bf2)
    #6  0x00007f6d8b2f241c n/a (/home/xxx/Software/Erlang/Erlang-24.2/lib/erlang/lib/wx
[New LWP 74039]
[New LWP 73873]
[...]
Core was generated by `/home/xxx/Software/Erlang/Erlang-24.2/lib/erlang/erts-12.2/bin/
```

```
Program terminated with signal SIGSEGV, Segmentation fault.
#0  0x00007f6e5461a74b in __memmove_avx_unaligned_erms () from /usr/lib/libc.so.6
[Current thread is 1 (Thread 0x7f6d900aa640 (LWP 74039))]
```

Then:

```
(gdb) bt
[...]
#0  0x00007f6e5461a74b in __memmove_avx_unaligned_erms () at /usr/lib/libc.so.6
#1  0x00007f6d8a204428 in  () at /usr/lib/dri/iris_dri.so
#2  0x00007f6d89733207 in  () at /usr/lib/dri/iris_dri.so
#3  0x00007f6d89733c97 in  () at /usr/lib/dri/iris_dri.so
#4  0x00007f6d898d8b0d in  () at /usr/lib/dri/iris_dri.so
#5  0x00007f6d898d8bf2 in  () at /usr/lib/dri/iris_dri.so
#6  0x00007f6d8b2f241c in ecb_glTexImage2D(ErlNifEnv*, ErlNifPid*, ERL_NIF_TERM*) (env
[...]
#29 0x00007f6d92967188 in wxe_main_loop(void*) (_unused=<optimized out>) at wxe_main.c
```

(this example was an Erlang wx/OpenGL-oriented crash)

From there, standard gdb-fu shall be sufficient to give much insight. Once done, use `q` to quit.

Preparing Adequate USB Keys

Objective

The goal here is, once having purchased a basic yet robust (e.g. with a proper lid) USB key (preferably from a good brand, bought from a reliable seller in order to avoid counterfeits), to prepare it efficiently for everyday use.

Often capacity matters a bit, speed not so much, and the best value for money is met for the mid-range keys of the time.

Conventions, conventions

The name of our key (`KEY_NAME`) will be `Kn`, where `n` is a key counter (e.g. `KEY_NAME=K7`).

We will create here two (GPT) partitions on such a key:

- a large, encrypted ("private"), Linux-friendly partition, as the main storage space of interest; based on EXT4 with dm-crypt and LUKS2, ciphered based on a 256-bit AES algorithm
- a smaller, plain/unencrypted ("public"), Windows-friendly partition, for convenience (when you have files to transfer yet you do not remember the passphrase of the previous partition)

The name of a partition (`P_NAME`) is a disk label, often limited to 11 characters. We choose it as prefixed with the name of the key, followed by either `pub` (for public, unencrypted) or `encr` (for encrypted), then with the partition number. For example, `K3-pub-1` or `K11-encr-4`.

The name of a filesystem (`FS_NAME`) of a partition is constrained as well, we specify it as: `KEY_NAME-(pub|encr)-PHYSICAL_SIZE[-PARTITION_NUMBER]`, the

partition number being useful to distinguish between any otherwise identical partitions of a given key. For instance `K7-pub-2GB` and `K7-encr-14GB-4`.

Any statically-defined mount point shall bear the same name as the associated filesystem: `MOUNTPOINT=/mnt/${FS_NAME}`. For example `MOUNTPOINT="/mnt/K7-pub-2GB"`.

Targeting the Right Device

Each key shall be registered in one's repository, and one shall be careful not to format one's local hard disk instead of a key.

To identify for sure such a key, run `lsblk --fs` just before plugging it in, and just after, so that the difference can be easily spotted.

The device name and size should be checked.

For an increased security, environment variables will be associated here to such a key, for example with: `export KEY_DEV=/dev/sdz`.

Its characteristics can be recorded in one's repository:

```
$ fdisk -l ${KEY_DEV}
$ parted ${KEY_DEV} print
```

Creating the Partitions

As root:

```
$ export KEY_NAME="K7"
$ fdisk -l
$ export KEY_DEV=/dev/sdz
$ fdisk ${KEY_DEV}
```

Then:

- print the partition table : (p)
- delete if necessary any previously existing partition(s): (d) for each of them
- create a GPT disklabel: (g) rather than a MBR one (i.e. not "dos" (o))
- create each partition (first the encrypted one(s), to favor their use):
  - creation thanks to (n)
  - size (e.g."+55G")
  - type (e.g. "Linux filesystem", i.e. 20, or "Microsoft basic data", i.e. 11)
  - partition name (with GPT: switch to expert mode (x), then (n), then a name like `K7-encr-part`); back to the main menu (r)
- check: (p), (F), (v)
- write: (w)

So that the kernel updates its partition table, it may be necessary to unplug and plug again the key.

All information (obtained in expert mode) regarding the new partitions may be stored in one's repository.

Erasing a Target Partition

If feeling paranoid about the previous content and having quite a lot of time ahead, a low-level erasure of a partition can be performed.

For example:

```
$ export PUB_DEV_NUM=1
$ export PUB_DEV="${KEY_DEV}${PUB_DEV_NUM}"; echo "PUB_DEV: ${PUB_DEV}"

$ fdisk -l ${PUB_DEV}
$ parted ${PUB_DEV} print

# Remove the echo after serious verification:
$ echo dd bs=256K if=/dev/urandom of=${PUB_DEV}
# (wait for *very* long)
dd: error writing '/dev/sdz1': No space left on device
# (still blocks for very long, despite any CTRL-C; just wait)
16385+0 records in
16384+0 records out
2147483648 bytes (2.1 GB, 2.0 GiB) copied, 241.353 s, 8.9 MB/s
```

Creating Plain, Unencrypted Partitions

Some devices (e.g. printers) may be confused should there be multiple partitions, or some with non-FAT or encrypted filesystems. This may be a reason to create a single, overall FAT partition.

Formatting a Plain Partition As FAT32 :

```
$ export PART_NUM=2
$ export PART_SIZE="2GB"

$ export FS_NAME="${KEY_NAME}-pub-${PART_NUM}-${PART_SIZE}"
# Or if a single partition is of that type:
$ export FS_NAME="${KEY_NAME}-pub-${PART_SIZE}"

$ echo ${FS_NAME}
$ export PUB_DEV="${KEY_DEV}${PART_NUM}"
# Remove the echo after serious verification:
$ echo mkdosfs -F 32 -n ${FS_NAME} ${PUB_DEV}
mkfs.fat 4.2 (2021-01-31)
mkfs.fat: Warning: lowercase labels might not work properly on some systems
```

Finalising and Testing a Plain Partition We take this opportunity to, after the previous section, create its own mount point (typically to be referenced in `/etc/fstab`):

```
$ export MOUNT_POINT=/mnt/${FS_NAME}; mkdir ${MOUNT_POINT} && mount ${PUB_DEV} ${MOUNT

# Should be not needed:
```

```
$ chown -R YOUR_USER:YOUR_GROUP ${MOUNT_POINT}

$ touch ${MOUNT_POINT}/WELCOME_TO_${KEY_NAME}_PUBLIC_${PART_NUM}_${PART_SIZE}_SPACE &&

# Or if a single partition is of that type:
$ touch ${MOUNT_POINT}/WELCOME_TO_${KEY_NAME}_PUBLIC_${PART_SIZE}_SPACE && ls -l ${MOU
```

If really wanting to register extraneous information:

```
$ mount | grep ${MOUNT_POINT}
/dev/sdb2 on /mnt/K5-pub-2GB type vfat (rw,relatime,fmask=0002,dmask=0002,allow_utime=

$ df ${MOUNT_POINT}
Filesystem     1K-blocks  Used Available Use% Mounted on
/dev/sdb2        2774720     4   2774716   1% /mnt/K5-pub-2GB

$ blkid ${PUB_DEV}
/dev/sdb2: LABEL_FATBOOT="K5-pub-2GB" LABEL="K5-pub-2GB" UUID="8C2C-1849" BLOCK_SIZE="

$ lsblk --fs ${PUB_DEV}
NAME FSTYPE FSVER LABEL      UUID                                 FSAVAIL FSUSE% MOUNT
sdb2 vfat   FAT32 K5-pub-2GB 8C2C-1849                              2.6G     0% /mnt/

$ parted ${PUB_DEV} print
Model: Unknown (unknown)
Disk /dev/sdb2: 2847MB
Sector size (logical/physical): 512B/512B
Partition Table: loop
Disk Flags:

Number  Start  End     Size    File system  Flags
1      0.00B  2847MB  2847MB  fat32

$ umount ${MOUNT_POINT}
```

Creating Encrypted Partitions

After Partitioning Such storage space is of course to be partitioned first like the plain ones (see Creating the Partitions), and can similarly be erased at a low level first (see Erasing a Target Partition).

For example we may end up with:

```
$ export ENCR_DEV_NUM=1

$ export ENCR_DEV="${KEY_DEV}${ENCR_DEV_NUM}"; echo "ENCR_DEV: ${ENCR_DEV}"
ENCR_DEV=/dev/sdb1

$ fdisk -l ${ENCR_DEV}
Disk /dev/sdb1: 12 GiB, 12884901888 bytes, 25165824 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
```

```
I/O size (minimum/optimal): 512 bytes / 512 bytes

$ parted ${ENCR_DEV} print

$ export PART_SIZE="12GB"
```

Creating an encrypted LUKS container We used to favor LUKS1 over LUKS2 for a better compatibility with ancient Linuces, yet it is no longer relevant, LUKS is widespread now.

In order to create such a container, there are many options to choose from; here are the ones that we prefer:

```
# Remove the echo after serious verification (enter YES then the main, daily
# passphrase to unlock that container):
#
$ echo cryptsetup --hash sha512 -c aes-xts-plain --key-size 512 luksFormat ${ENCR_DEV}
WARNING!
========
This will overwrite data on /dev/sdb1 irrevocably.

Are you sure? (Type 'yes' in capital letters): YES
Enter passphrase for /dev/sdb1:
Verify passphrase:
```

We strongly advise to add a second, "rescue" passphrase (longer, more difficult than the previous daily one, and potentially common to at least some keys), as a last chance:

```
# Remove the echo after serious verification (enter the previous passphrase, then
# the rescue one):
#
$ echo cryptsetup luksAddKey ${ENCR_DEV}
Enter any existing passphrase:
Enter new passphrase for key slot:
Verify passphrase:
```

Let's introduce then more variables:

```
$ export ENCR_DESIGNATOR="${KEY_NAME}-encr-${ENCR_DEV_NUM}-${PART_SIZE}"
# - OR -
$ export ENCR_DESIGNATOR="${KEY_NAME}-encr-${PART_SIZE}"

$ echo "ENCR_DESIGNATOR=${ENCR_DESIGNATOR}"
ENCR_DESIGNATOR=K5-encr-12GB
```

Now we unlock the LUKS container so that we can create an EXT4 partition in it.

A `-fs` suffix would not be very relevant (this is the name that will be used by to automounter):

```
$ export ENCR_FS_NAME="${ENCR_DESIGNATOR}"

# Remove the echo after serious verification:
$ echo cryptsetup config ${ENCR_DEV} --label ${ENCR_FS_NAME}
cryptsetup config /dev/sdb1 --label K5-encr-12GB
$ cryptsetup config ${ENCR_DEV} --label ${ENCR_FS_NAME}

$ export DM_NAME="my-${ENCR_DESIGNATOR}"

# Enter either of the two aforementioned passphrases:
$ cryptsetup luksOpen ${ENCR_DEV} ${DM_NAME}
Enter passphrase for /dev/sdb1:
```

Creating the In-Container EXT4 Filesystem Deactivating journaling (with the `-O ^has_journal` option) could increase a bit the lifespan of the key, but would weaken its filesystem, whereas USB keys may be (unfortunately) unplugged while being still mounted; so we prefer keeping the journaling:

```
# Remove the echo after serious verification:
$ echo mkfs.ext4 /dev/mapper/${DM_NAME} -L ${ENCR_FS_NAME} -E discard
mkfs.ext4 /dev/mapper/my-K5-encr-12GB -L K5-encr-12GB -E discard

$ mkfs.ext4 /dev/mapper/${DM_NAME} -L ${ENCR_FS_NAME} -E discard
mke2fs 1.47.0 (5-Feb-2023)
Creating filesystem with 3141632 4k blocks and 786432 inodes
Filesystem UUID: 92c44f73-8614-44f9-a03c-cfd718aecb8e
Superblock backups stored on blocks:
   32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208

Allocating group tables: done
Writing inode tables: done
Creating journal (16384 blocks): done
Writing superblocks and filesystem accounting information: done

# Should be needed (otherwise can be mounted yet not written by one's normal user):
$ chown -R YOUR_USER:YOUR_GROUP ${MOUNT_POINT}
```

So no more need here for `tune2fs -O ^has_journal /dev/mapper/${DM_NAME}`. One may still check the resulting settings with `tune2fs -l /dev/mapper/${DM_NAME}`.

Declaring keys in `/etc/fstab` and in `/etc/crypttab` is not necessary; moreover some care would be needed in order to ensure that the automounter would not freeze at the next reboot, expecting such a key to be available.

Finalising and Testing an Encrypted Partition Here also one could take this opportunity to, after the previous section, create a dedicated mount point (see just above for a warning about referencing it through `/etc/fstab` and/or `/etc/crypttab`).

We consider that the previous LUKS container is still opened (otherwise: `cryptsetup luksOpen ${ENCR_DEV} ${DM_NAME}`).

Then:

```
$ export MOUNT_POINT=/mnt/${ENCR_DESIGNATOR}; echo "MOUNT_POINT = $MOUNT_POINT"
MOUNT_POINT = /mnt/K5-encr-12GB

$ mkdir ${MOUNT_POINT}

# For example DM_NAME=/dev/mapper/my-K5-encr-12GB:
$ mount /dev/mapper/${DM_NAME} ${MOUNT_POINT}

$ touch ${MOUNT_POINT}/WELCOME_TO_${KEY_NAME}_ENCRYPTED_${PART_NUM}_${PART_SIZE}_SPACE
 or
$ touch ${MOUNT_POINT}/WELCOME_TO_${KEY_NAME}_ENCRYPTED_${PART_SIZE}_SPACE && ls -l ${
total 16
drwx------ 2 root root 16384 Apr 22 12:41 lost+found
-rw-rw-r-- 1 root root     0 Apr 22 16:14 WELCOME_TO_K5_ENCRYPTED_12GB_SPACE

# Record some information if wanted:
$ mount | grep ${MOUNT_POINT}
/dev/mapper/my-K5-encr-12GB on /mnt/K5-encr-12GB type ext4 (rw,relatime)

$ blkid ${ENCR_DEV}
/dev/sdb1: UUID="f00def37-529a-4400-aa8c-c7a36589c152" LABEL="K5-encr-12GB" TYPE="cryp

$ lsblk --fs ${ENCR_DEV}
NAME              FSTYPE      FSVER LABEL        UUID
sdb1              crypto_LUKS 2     K5-encr-12GB f00def37-529a-4400-aa8c-c7a36589c152
my-K5-encr-12GB ext4        1.0   K5-encr-12GB 92c45f73-8614-44f9-a03c-cfd718aecb8e

$ parted ${ENCR_DEV} print
Error: /dev/sdb1: unrecognised disk label
Model: Unknown (unknown)
Disk /dev/sdb1: 12.9GB
Sector size (logical/physical): 512B/512B
Partition Table: unknown
Disk Flags:

$ df ${MOUNT_POINT}
Filesystem                   1K-blocks  Used Available Use% Mounted on
/dev/mapper/my-K5-encr-12GB  12262536  2072  11615756   1% /mnt/K5-encr-12GB
```

Finally:

```
$ umount ${MOUNT_POINT}
$ cryptsetup close ${DM_NAME}
```

That's it! You should end up with a basic, inexpensive USB key - yet offering at least two satisfying partitions, one public (unencrypted), one adequately encrypted.

GRUB Ate My Distro Again: Fixing the Bootloader

So you tried recklessly to add some kernel parameters - presumably to prevent your laptop from freezing regularly for no apparent reason - and GRUB managed once again to replace a bootloader that used to work flawlessly with one that just cycles to the BIOS?

Here are a few guidelines (on Arch Linux, with UEFI and GPT - rather than any MBR) that could be useful, knowing that considerably more complete information can be found in this page.

Prerequisites

At least the following packages will be needed (anyway they are likely to be already available, either from a real install or from a rescue medium): `grub`, `efibootmgr` and `os-prober`.

You will also need a bootable (most probably removable) install/rescue medium, typically an USB stick whose content can be erased. Either you have access to another computer (Linux or not), or you have a multiboot and, after a smart journey in the BIOS menus, you will be able to nevertheless launch a Windows instance (generally GRUB just concentrates on wreaking havoc on unfortunate Linuces, rather than doing the same to Windows installations).

If having only a Windows instance at hand, one may just follow these guidelines, which mostly boil down to downloading and installing the (free software) Win32 Disk Imager tool, grabbing also a relevant ISO (e.g. `archlinux-2023.04.01-x86_64.iso` at the time of this writing) and writing in on said USB stick.

Running a Live Rescue Arch

Once having a proper rescue medium, ensure that it is inserted, and reboot your GRUB-affected host; possibly the BIOS configuration will have to be updated so that the host can boot on that medium.

Once done, you should end up with a root shell - albeit running from a live, rescue Arch Linux, whereas the one that you want to fix is of course the one of your host.

Preparing the chroot

First enforce any essential setting, such as `loadkeys fr` if you have a French keyboard. Then locate the `/boot` partition of the host. It may either be directly in the partition corresponding to the root `/` tree or, as per our conventions, in a separate one (so that all other partitions can be appropriately encrypted). For that, `fdisk -l` (and possibly `lsblk` as well) shall help finding out such a partition; for instance:

```
$ fdisk -l
Device             Start        End    Sectors   Size Type
/dev/nvme0n1p1      2048    1226751    1224704   598M EFI System
/dev/nvme0n1p2   1226752    1259519      32768    16M Microsoft reserved
/dev/nvme0n1p3   1259520  313274682  312015163 148.8G Microsoft basic data
/dev/nvme0n1p4 313276416  314570751    1294336   632M Windows recovery environment
/dev/nvme0n1p5 314572800  838860799  524288000   250G Microsoft basic data
```

```
/dev/nvme0n1p6  838860800  855638015   16777216     8G Linux swap
/dev/nvme0n1p7  855638016 1056964607  201326592    96G Linux filesystem
/dev/nvme0n1p8 1056964608 4000797326 2943832719   1.4T Linux filesystem
```

Here, one can guess that `nvme0n1p1` is `/boot` (because of the EFI type, and the size), that `nvme0n1p7` is `/` and `nvme0n1p8` is `/home`.

Managing the chroot

A relevant chroot shall be performed, once all appropriate trees have been mounted; we do not need `/home` (on the contrary, it is safer if it remains locked and not even mounted), but we need the right `/boot` (the one on disk and that contains the previous bootloader elements) to be mounted, instead of the empty one that would be inherited by a mere mounting of the `/` root. So:

```
$ cd /mnt

# Intermediary directory usually useless but clearer:
$ mkdir my_chroot_tree

# Mount the whole system tree, "/":
$ mount /dev/nvme0n1p7 my_chroot_tree

# Take care of our separate "/boot" as well:
$ ls my_chroot_tree/boot
# (empty)

$ mount /dev/nvme0n1p1 my_chroot_tree/boot
$ ls my_chroot_tree/boot
. EFI initramfs-linux-fallback.img initramfs-linux-lts-fallback.img
intel-ucode.img  vmlinuz-linux .. grub initramfs-linux.img
initramfs-linux-lts.img 'System Volume Information' vmlinuz-linux-lts

# Switch to the actual host Arch system:
$ arch-chroot my_chroot_tree
```

(Re)Installing GRUB

As we understand it, installing GRUB (i.e. copying its relevant file elements in a proper location in `/boot`) and configuring it can be done in any order; we prefer anyway installing it first, with:

```
# Note that the 'EFI' directory is not directly specified here, only
# the /boot mount point:
$ grub-install --target=x86_64-efi --efi-directory=/boot --bootloader-id=MY_GRUB
```

Configuring GRUB

> **Note**
> Configuring GRUB means generating a settings file whose syntax may be more recent than the one supported by any already-installed GRUB - which would thus be bound to fail at boot. So any configuration update of GRUB shall be done together with the update of its installation.

Now is the right time to update one's `/etc/default/grub`, notably to ensure that `GRUB_DISABLE_OS_PROBER=false` is indeed defined, and is not commented out.

If `os-prober` is especially useful to auto-detect any Windows installation, we could see that it was not working from a chroot (and thus it has to be done later, directly from the final Arch host).

The last step is to generate the corresponding GRUB configuration file:

```
$ grub-mkconfig -o /boot/grub/grub.cfg
```

Finally

Rebooting and selecting a right entry menu (note that LTS kernels will be sorted first and thus be selected by default) should result in a restored boot proceeding as normal.

Moreover, from now, `os-prober` should be able to pick up any other system (especially Windows installations) and declare them appropriately: this should be just a matter of running `grub-mkconfig -o /boot/grub/grub.cfg` again (and hope for the best).

Other Filesystem-related Issues

Word of wisdom: perform backups. Regularly. Safely stored, sheltered from disasters like burglars, floods, cats, fires, etc.

Target is busy

Umounting may fail with `umount: /var/foobar: target is busy`. Finding the culprit can be done with:

```
$ lsof /var/foobar
COMMAND    PID USER   FD   TYPE DEVICE SIZE/OFF     NODE NAME
bash    275648 root  cwd    DIR  254,1    20480 45236687 /var/foobar/www/Foo
```

Failed I/O Operations

Some operations (even a `touch` done by `root`) may fail, reporting `Read-only file system`; `systemd-fsck` may also report `Structure needs cleaning`.

Check with `mount` that a given filesystem/partition is (still) in the expected state:

```
# For example:
/dev/mapper/foobar on /var/foobar type ext4 (rw,relatime,data=ordered)
```

```
# may have become:
/dev/mapper/foobar on /var/foobar type ext4 (ro,relatime,errors=continue,data=ordered)
```

Best option here is to unmount, make a sector-level backup of the disk, and then check it. Probably that the disk already failed and you will have to resort to revert to a former backup.

Checking a Disk

This can be done with `e2fsck` once the filesystem is unmounted, like in: `e2fsck -cDp /dev/mapper/foobar`, with:

- `-c`: use `badblocks` to do a read-only scan of the device
- `-D`: optimize directories in the filesystem
- `-p`: automatically repair ("preen") the file system

If having errors like:

```
e2fsck: Input/output error while trying to open /dev/mapper/foobar‘‘
```

or:

```
The superblock could not be read or does not describe a valid
ext2/ext3/ext4 filesystem [...]
```

then, unless you are trying to check an encrypted partition (e.g. not having cryptsetup-opened a LUKS one) or a superblock copy is enough, prospects are grim.

Try for example `fdisk` / `gdisk` / `cgdisk`, `parted`, `smartctl --all -T verypermissive`, `badblocks -o ~/scan-result.txt -sv` (for which errors are counted as NumberOfReadErrors / NumberOfWriteErrors / NumberOfCorruptionErrors) - yet your disk is maybe already unrecoverable.

Quick Topics

Relying on Stable and User-Friendly Names for Network Interfaces

Tired of having your network interfaces be named `enp0s31f6`, or `wlp4s0` then `wlan0`, to mix them up afterwards and to have to update their naming in various places (e.g. for `netctl`, `iptables`)?

Time is to rename them once for all; and we prefer doing so with thanks to a udev rule.

To change the name of a given network interface (e.g. from `enp0s31f6` to `lanfoobar`, `fiber1` or, here, `eth0`), its MAC address must be obtained first:

```
$ ip link show enp0s31f6
2: enp0s31f6: <BROADCAST,MULTICAST> mtu 1500 qdisc fq_codel state DOWN mode DEFAULT gr
link/ether 54:e1:cd:f5:a4:f8 brd ff:ff:ff:ff:ff:ff
```

Then a corresponding entry (the MAC address must be in a lowercase) may be added (possibly among others) in a file named, for example, `/etc/udev/rules.d/10-network.rules`:

```
SUBSYSTEM=="net", ACTION=="add", ATTR{address}=="54:e1:cd:f5:a4:f8", NAME="eth0"
```

Then, provided that the target interface is down[4] , the new rule can be triggered for a test, with:

```
$ udevadm trigger --verbose --subsystem-match=net --action=add
```

Use `ip link` to check that the new names are indeed applied.

One should then hunt down under the `/etc/systemd` tree (especially in `/etc/systemd/system/multi-user.target.wants/`) all files whose name and/or content include the former interface name, in order to update them with the new name and try, after a `systemctl daemon-reload`, to (re)start that interface.

Installing Wine

Install it, once enabling multilib has been done, with: `pacman -S wine`.

When run, this may lead `wine-mono` to be auto-installed.

The pseudo-Windows filesystem is then located mostly in `~/.wine/drive_c`.

Overcoming Invalid PGP Signatures in Pacman Packages

This happens regularly, especially when updating a symptom being: `File /var/cache/pacman/pkg/xxx.pkg is corrupted (invalid or corrupted package (PGP signature))`.

Solution: `pacman -S archlinux-keyring` is at least often enough. Otherwise `pacman-key --populate archlinux`, or changing mirror might help.

Protecting Files and Directories

Beyond the basic `chown` / `chmod` commands (e.g. `chmod 400` to have it read-only for its owner only), using `chattr` as well shall be considered for the most sensitive filesystem elements (like the ones related to `~/.ssh`).

Notably, so that a file becomes immutable (even for root), one may use `chattr +i MY_FILE`.

Use `chattr -i MY_FILE` to clear that immutability, and `lsattr` to check it.

Converting Data Formats

- to convert multimedia files, `ffmpeg` is very convenient; for example to convert an AIF file into a WAV one: `ffmpeg -i sound.aif sound.wav`
- to read `*.pck` resource file, the Dragon UnPACKer (Windows) open source tool can be used

See also Ceylan-Hull's multimedia-related section.

[4] Actually it does not even seem necessary.

Adding a Locale

On some hosts, issues in terms of a lacking locales may be reported, like in the following:

```
bash: warning: setlocale: LC_ALL: cannot change locale (en_US.UTF-8)
```

This can be fixed by uncommenting the corresponding locale (`en_US.UTF-8` here) in `/etc/locale.gen`, and then regenerating the system locales by running (as root) `locale-gen`:

```
$ locale-gen
Generating locales (this might take a while)...
 en_US.UTF-8... done
 fr_FR.UTF-8... done
 fr_FR.ISO-8859-15@euro... done
Generation complete.
```

Performing Searches and Replacements

Searches: to be made with `grep` To search for a **set** of patterns, separate them by an (escaped) pipe; for example: `grep 'hello\|goodbye\|farewell' Foobar.java`.

Replacements: to be made with `sed`

- **multiple operations** (typically replacements) can be made in a row by combining them with `";"`; for example to uppercase only a set of letters: `echo "aabbccddee" | sed 's|b|B|g;s|d|D|g'` yields `aaBBccDDee`
- to perform **in-place** replacements in a file: `sed -i 's|MY_TAG|142|g' my_file.txt` (if a suffix is specified, backups may be made)

Common to `grep` and `sed` With grep and sed, `[[:space:]]` and `\s` are the same, they will both match any whitespace character spaces, tabs, etc.

Applying a XFCE4 configuration to a different user

The objective is to duplicate the configuration of the `user_a` desktop to the `user_b` one, for example if `user_b` is used to test an untrusted application.

A key point is to ensure that at least `user_b` has no ongoing graphical session (otherwise any new configuration may be overwritten with an older one at logout).

Then, typically as root:

```
$ rm -rf /home/user_b/.config/{xfce4,Thunar} /home/user_b/.local/share/xfce4
$ cp -rf /home/user_a/.config/{xfce4,Thunar} /home/user_b/.config
$ cp -rf /home/user_a/.local/share/xfce4 /home/user_b/.local/share
$ chmod -R /home/user_b/{.config,.local} user_b:users
```

Then a key point is to fully reset XFCE4. Unlogging and/or `killall -HUP xfdesktop` may not be sufficient. As a last resort, rebooting shall work.

Selecting a per-user wallpaper may be of help.

Solving PulseAudio Issues

As your current user:

```
$ /bin/rm -rf /tmp/pulse* ~/.pulse* ~/.config/pulse
$ pulseaudio -k ; pulseaudio --start
```

If not sufficient, check `/etc/pulse/default.pa` and run as a normal user `pulseaudio -vvv`, looking for errors (e.g. `pulseaudio -vvv 2>&1 | grep '^E:'`).

The playback hardware devices may be listed with `aplay -l`.

One may run `pacmd list-cards` to obtain the information (e.g. symbolic and profile names) to specify `set-card-profile` entries (useful to set proper defaults and switch off unwanted elements).

See also these Arch guidelines.

As last resort (rarely needed), as root, one may reinstall it: `pacman -Sy pulseaudio` and reboot.

Using Visual Studio Code

Visual Studio Code (a.k.a. *VS Code*; not to be confused with *Visual Studio* itself) is a popular free software (MIT licence) source-code multi-platform editor, developed by Microsoft.

It supports many languages and features through extensions.

Installing It can be installed on Arch thanks to `pacman -Sy code`.

If not installed thanks to one's distribution, the binaries of Visual Studio Code can be directly downloaded (through an archive named `code-stable-x64-XXX.tar.gz`) that may be extracted in, say, `~/Software/Vs-Code`.

Another option is to download and use VSCodium, a version of it without Microsoft branding/telemetry/licensing.

One may then put `~/Software/Vs-Code/VSCode-linux-x64/bin` in one's PATH so that VS Code can be run thanks to: `code`.

Updating Prior to running any VS Code, circumvent any proxy that is in the way.

Then launch VS Code, select in the *Help* menu first *Check for updates* then, if any is found, *Download Available Update* which, from GNU/Linux without relying on a package manager, should point to this download link.

Then the dowloaded file (e.g. `~/Downloads/code-stable-x64-1718139773.tar.gz`) shall be moved to `~/Software/VS-Code` and extracted (`tar xvf code-stable-x64-1718139773.tar.gz`) there, creating or updating a `VSCode-linux-x64` tree.

Executing One may run `code`, or rely on our run-vscode.sh script (notably to accommodate for any proxy).

Configuring

Fixing whitespaces In `File -> Preferences -> Settings`, preferably in the `User` tab (rather than only in the `Workspace` one), search and enable the `Files: Trim Trailing WhiteSpace` option (or select in the `Text Editor` category the `Files` subcategory and hunt that option down).

Ignoring files Some generated files (e.g. `*.beam` ones) should not be shown in the Explorer panel (usually on the left of the screen).

In the settings screen (`File > Preferences > Settings`), in Files, one may then add: `Exclude **/*.beam` to exclude them, regardless of the place in the filesystem tree.

General-purpose extensions They do not apply to specific language per se.

For makefiles, one may rely on the Makefile Tools Microsoft extension.

Fixing Problems Use `Ctrl-Shift-M` to display in the corresponding panel (generally on the bottom-right of the screen) the problems reported by the enabled extensions.

Supported Languages We recommend using the following extensions for:

- **Python**: the Microsoft Python extension, the Microsoft Python Debugger extension and the Microsoft Pylance extension; possibly also vscode-micromamba

Using E-mail Clients

Over time we were preferring Evolution to Thunderbird, notably regarding its far better OpenPGP support. Yet, at least for some IMAP servers (e.g. the ones of Mailo), errors (like: `Error copying messages: Error`) constantly happening (despite many attempts of configuration changes) and being reported ruined its use for us[5], so we switched back to Thunderbird.

- for Thunderbird:
  - the message filters can be copied as a whole from a computer to another, knowing that with IMAP servers the emails and their folder structure should be readily available from all clients; transferring filters is as simple as copying, whereas Thunderbird instances are closed, the relevant `msgFilterRules.dat` file (located typically in a generated directly like `~/.thunderbird/6r77gatw.default-release/ImapMail/YOUR_IMAP_SERV`
  - we recommend using the DKIM Verifier add-on, that we found much useful to better detect spoofing attempts; it can be configured to display colored DKIM statuses for the sender of each email, which is very convenient
- for Evolution: on Arch Linux, consider installing the `evolution-bogofilter` and/or `evolution-spamassassin` packages, otherwise spams (junk mails) do not seem to be managed properly (or at all)

[5] Moreover the fixed character-wrapping of Evolution is rather surprising and unfortunate.

See also Ceylan-Hull's email.sh and archive-emails.sh scripts.

Sandboxing an Application

Sometimes it is harder to trust an application, for example because it had to be installed as an AppImage.

Sandboxing reduces the risk of security breaches by restricting the running environment of untrusted applications: they can be prevented from accessing to filesystems, to the network, etc.

A well-known sandboxing tool is firejail. It can be installed on Arch Linux with `pacman -Sy firejail`. Then an untrusted executable `my_exec` can simply be run with: `firejail my_exec`.

Recording a Screencast

We rely on the impressive, feature-rich, free software SimpleScreenRecorder (`pacman -Sy simplescreenrecorder`).

This tool is run as `simplescreenrecorder` and offers many options and encoding choices; it is able to record OpenGL applications[6] and also the audio.

Note that, as notified by the tool, the MP4 container type will produce unreadable files if the recording is interrupted (then the MKV container shall be preferred).

See also this comparison of screencasting software.

Specifying iterations with Bash

With Bash, as an alternative to `$(seq N)`, variables may be iterated (e.g. to select filenames) based on:

- a list: `for i in images-{1,4,9}-of-cats.jpeg; do ...`
- a range: `ls images-{1..7}-of-cats.jpeg`, possibly with an increment: `ls images-{1..7..2}-of-cats.jpeg`

Shell Auto-completing the available `make` targets

If able to do so, just install a `bash-completion` package, available on most distributions (including Arch Linux).

Otherwise, one may add to one's shell profile / configuration (e.g. `~/.bashrc`):

```
complete -W "\`grep -oE '^[a-zA-Z0-9_.-]+:([^=]|$)' *akefile | sed 's/[^a-zA-Z0-9_.-]
```

Mini Shell Cheat Sheet

For the basic `/bin/sh` shell (Bash being a superset thereof).

[6] In our case it was sufficient to select `Record a fixed rectangle` and `Select window` (which happened to be the one of an OpenGL application); it seemed a better option than selecting `Record OpenGL`, as afterwards no recording could be done due to `Could not get the size of the OpenGL application.` (maybe this could be alleviated by entering adequate settings).

Switch-like Statement Example:

```
case "${extension}" in

     "erl"|"hrl")
         reformat_erlang_source_file "${target_file}"
         ;;

     "foo")
         reformat_foo "${target_file}"
         ;;


     *)
         do_something
         ;;

esac
```

Testing whether a Path is Absolute For almost all shells (including `/bin/sh`):

```
case $DIR in

   /*) echo "absolute path" ;;
   *)  echo "relative path" ;;

esac
```

Testing whether a Filesystem Element is a Regular File or a Symlink (as using symbolic links is often convenient)

Testing if a filesystem element is a symlink:

```
if [ -L "${elem}" ]; then
    # elem is a (possibly dead) symlink
    ...
fi
```

Testing also whether the element pointed to exists (as a file, directory, symlink in turn, etc.):

```
if [ -L "${elem}" ] && [ -e "${elem}" ]; then
    # elem is a non-dead symlink
    ...
fi
```

Testing whether a given file element is either a regular file or a non-dead symlink is as simple as:

```
if [ -f "${elem}" ]; then
    # elem is a file
    ...
fi
```

Mounting Manually crypttab-declared Partitions

Let's suppose such a partition (here a LUKS-encrypted one) is declared, in `/etc/crypttab`, as:

```
my-encrypted-nvme-storage UUID=a46879cf-xxx luks,timeout=10,fido2-device=auto
```

and then in `/etc/fstab`, as:

```
/dev/mapper/my-encrypted-nvme-storage /var/foobar ext4 rw,relatime,data=ordered,nofai
```

Such partition is expected to be mounted at boot time, but here was closed for any reason.

One can inquire about it thanks to:

```
$ SERVICE="systemd-cryptsetup@my\\x2dencrypted\\x2dnvme\\x2dstorage.service"
$ systemctl status "$SERVICE"
$ journalctl -xeu "$SERVICE"
$ cat "/run/systemd/generator/$SERVICE"
```

Mounting it back is "as simple" as executing: `systemctl start /var/foobar`.

See also this section for related details.

Other User Settings

See our preferences for Startpage; it can be set as homepage.

Shortcuts

Listing here the main keyboard shortcuts of interest for a few base tools.

For `less` Mix of `vi` / `more` conventions, in order to:

- **forward-search** for the N-th line containing the pattern (N defaults to 1): `/pattern`; use `?/pattern` to **backward-search**
- **repeat previous search**: `n`:
- **invoke an editor** on the current file being viewed: `v`
- **go to next file**: `:n`; for the **previous** one: `:p`
- **help**: `h`
- **quit**: `q`

For `mpv`

- **pause** / **unpause** the current playback: `<Space>`
- **decrease the volume**: `/`; **increase** it: `*`; **mute**: `m`
- **go backward** / **forward** in the current playback, by steps of:
  - 5 seconds: `Left` / `Right arrow` keys

  - 1 minute: `Down` / `Up arrow` keys

- **jump to the next playback**: `<Enter>` or `<Escape>`
- **stop** all playbacks: `<Ctrl-C>`
- **display temporarily durations**: `o` (*On-Screen Display*), i.e. elapsed, elapsed / total, etc.
- **toggle fullscreen**: `f`
- **double / halve the playback speed**: `{` / `}` (`Backspace` to reset it)
- **take a screenshot**: `s` (notably if using our v script, as the `-vf screenshot` command-line option must have been specified); then a `mpv-shot0001.jpg` file will be silently created in the directory whence mplayer was launched, next screenshot will be `mpv-shot0002.jpg`, etc.
- **quit**: `q`

For `mplayer`

- **pause/unpause** the current playback: `<space>`
- **decrease the volume**: `/`; to **increase** it: `*`
- **go backward/forward** in the current playback: `left` and `right arrow` keys
- **jump to next playback**: `<Enter>` or `<Escape>`
- **stop** all playbacks: `<CTRL-C>`
- **display durations**: `o` to toggle OSD (*On-Screen Display*), i.e. elapsed, elapsed / total, etc.
- **take a screenshot**: `s` (notably if using our v script, as the `-vf screenshot` command-line option must have been specified); then a `shot0001.png` file will be silently created in the directory whence mplayer was launched, next screenshot will be `shot0002.png`, etc.

See Also

One may refer to our other mini-HOWTO regarding:

- Network Management
- Cybersecurity

The Ceylan-Hull section system-related section might also be of interest.

Erlang

Organisation: Copyright (C) 2021-2025 Olivier Boudeville

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

Creation date: Saturday, November 20, 2021

Lastly updated: Wednesday, March 19, 2025

Table of Contents

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 (e.g. 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!

To check from the command-line the version of an Erlang install:

```
$ erl -eval '{ok, V} = file:read_file( filename:join([code:root_dir(), "releases", erl
24.2
```

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

- `l/1` (apparently undocumented), used as `l(my_module)`, to (re)load that module, purging any old version of it; convenient to reload also specifically-built BEAMs (e.g. based on parse transforms) that may have been compiled behind the scene
- `v/1`: `v(N)` returns, if:
  - `N > 0`: the result value of command of (absolute) number `N`
  - `N < 0`: the result value of any previous N-th command; for example `v(-1)` corresponds to the value of any previous command
- `rl/1` is *not* "reload (module)" (use `l/1` for that), it is "record list" (it prints selected record definitions)
- `c/1`, used as `c(my_module)`, to compile (if necessary) *and* (re)load that module, purging any old version of it
- `rr/{1,2,3}` (e.g. 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.

Distributed Mode of Operation

General Information

The goal is to have processes, running on multiple Erlang VMs instantiated on various hosts of a network, interact (of course, as always, by message-passing based on Erlang processes).

Let's suppose that we have two hosts, `foo.example.com` (possibly on the LAN) and `bar.other.info` (for example a gateway available on the Internet - hence a bit secured, and with many settings activated), each having the latest version of Erlang installed.

To test whether VMs on either side can communicate, one may launch on `foo.example.com` for example (killing EMPD and using a specific, rather random port for it, for safer/paranoid testing):

```
$ killall epmd

# Possibly safer to specify just '-name myfoo' rather directly the FQDN
# ('-name myfoo@foo.example.com'), as we can check the name resolution used
# by the VM (bogus values like '-name myfoo@ibm.com' would be accepted):
#
$ ERL_EPMD_PORT=4506 erl -name myfoo -setcookie abc
Erlang/OTP 27 [erts-15.2] [source] [64-bit] [smp:8:8] [ds:8:8:10] [async-threads:1] [

Eshell V15.2 (press Ctrl+G to abort, type help(). for help)
(myfoo@foo.example.com)1>
```

And on `bar.other.info`:

```
$ killall epmd
$ ERL_EPMD_PORT=4506 erl -name mybar@bar.other.info -setcookie abc
Erlang/OTP 27 [erts-15.2] [source] [64-bit] [smp:8:8] [ds:8:8:10] [async-threads:1] [

Eshell V15.2 (press Ctrl+G to abort, type help(). for help)
(mybar@bar.other.info)1>
```

Then the following calls may be of interest, to interlink these VMs and check that they agree on interacting.

On `foo.example.com`:

```
1> net_adm:ping('mybar@bar.other.info').
pong
```

And on `bar.other.info`:

```
1> net_adm:ping('myfoo@foo.example.com').
pong
```

Then either of the two nodes can perform mostly anything on the other host (of course depending on the respective permissions of the users that run them); and of course Erlang cookies are not a security feature, they are just an ad hoc way of preventing unwanted interactions between nodes.

A lot more often than expected, `pang` is received (instead of `pong`), meaning that unfortunately the connection could not be created.

Among the many possible pitfalls, there are (approximately sorted by decreasing level of probability, according to our experience):

- at least a **firewall** between the two hosts filtering packets, either exchanged by the Erlang VMs (using ephemeral ports for that, within a possibly user-specified range), or by the EPMD daemons; one may refer to our page about firewalls
- misconfigured (e.g. inconsistent) **DNS** services (e.g. one host unable to resolve the other, or resolving it as a different name that it knows itself); simply running `hostname -f`, and possibly `ping foo.example.com` from `bar`, and `ping bar.other.info` from `foo`, may unveil surprises)
- inconsistent EPMD **ports** ; beware of any lingering `ERL_EPMD_PORT` environment variable that may still apply (this is why, in the example, one was specified; otherwise the default one, 4369, is expected to be relied upon)
- at least one incorrect (misspelled) **target node**; moreover their names must be atoms (hence the use of single quotes, as they comprise dots) and the FQDN should match (e.g. from the point of view of the EPMD daemon, `bar` and `bar.other.info` might be considered different)
- mistmatching (Erlang) **cookies** (of course)
- mismatching **name addressing**: using short names (`-sname`) mixed with long names (`-name`)

- faulty EPMD **daemons** (e.g. one having missed that a node respawned; this seldom happens, under specific circumstances) - hence the prior killings in the example
- incompatible **versions** of Erlang (if in doubt, just use the latest stable one on either side); anyway such an issue should be reported on the console

With Our Conventions

To focus a bit on developments relying on our conventions (notably the Ceylan-Myriad ones) about firewall settings, one may first inspect the `/etc/iptables.settings-Gateway.sh` file to check the firewall settings that are currently listed (e.g. `grep 'myriad_default_epmd_port\|tcp_' iptables.settings-Gateway.sh`).

This may result for example in:

```
myriad_default_epmd_port=4502
enable_unfiltered_tcp_range="true"
tcp_unfiltered_low_port=60000
tcp_unfiltered_high_port=65000
```

(supposing they apply on both hosts)

One could ensure that these were indeed the ones applied (if ever the prior settings were changed yet with no firewall reloading afterwards), by checking the `/root/.last-gateway-firewall-activation` file.

Finally, one may run our `iptables-inspect.sh` script to request iptables to describe its current, actual state.

Then, to test the connectivity of corresponding VMs, first on `foo.example.com`:

```
$ killall epmd beam.smp
$ ERL_EPMD_PORT=4502 erl -name myfoo@foo.example.com -setcookie abc -kernel inet_dist
Erlang/OTP 27 [erts-15.2] [source] [64-bit] [smp:8:8] [ds:8:8:10] [async-threads:1] [

Eshell V15.2 (press Ctrl+G to abort, type help(). for help)
(myfoo@foo.example.com)1>
```

And on `bar.other.info`, this could be:

```
$ export ERL_EPMD_PORT=4502
$ erl -name mybar@bar.other.info -setcookie abc -kernel inet_dist_listen_min 60000 ine
Erlang/OTP 27 [erts-15.2] [source] [64-bit] [smp:8:8] [ds:8:8:10] [async-threads:1] [

Eshell V15.2 (press Ctrl+G to abort, type help(). for help)
(mybar@bar.other.info)1> net_adm:ping('myfoo@foo.example.com').
pong
```

At last it works!

About Security

- one should not encrypt messages directly with a key pair (e.g. 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*)

OTP Guidelines

The `.app` Specification

For an overall application named `Foobar`, we recommend defining in `conf/foobar.app.src` an application specification template that, once properly filled regarding the version of that application and the modules that it comprises (possibly automatically done thanks to the Ceylan-Myriad logic), will result in an actual application specification file, `foobar.app`.

Such a file is necessary in all cases, to generate an OTP application (otherwise with rebar3 nothing will be built), an OTP release (otherwise the application dependencies will not be reachable), and probably an hex package as well.

This specification content is to end up in various places:

- in `ebin/foobar.app`
- if using rebar3, in the OTP build tree (by default: `./_build/lib/foobar/ebin/foobar.app`)
- with `src/foobar.app.src` being a symbolic link pointing to `ebin/foobar.app` (probably at least for hex)

Starting OTP Applications

For an OTP *active* application of interest - that is one that provides an actual service, i.e. running processes, as opposed to a mere *library* application, which provides only code - such a specification defines, among other elements, which module will be used to start this application. We recommend to name this module according to the target application and to suffix it with `_app`, like in:

```
{application, foobar, [
    [...]
    {mod, {foobar_app, [hello]}},
    [...]
```

This implies that once a user code will call `application:start(foobar)`, then `foobar_app:start(_Type=normal, _Args=[hello])` will be called in turn.

This `start/2` function, together with its `stop/1` reciprocal, are the functions listed by the OTP (active) `application` behaviour; at least for clarity, it is better that `foobar_app.erl` comprises `-behaviour(application).`

Pre-Launch Functions

The previous OTP callbacks may be called by specific-purpose launching code; we tend to define an `exec/0` function for that: then, with the Myriad make system, executing on the command-line `make foobar_exec` results in `foobar_app:exec/0` to be called.

Having such a pre-launch function is useful when having to set specific information beforehand (see `application:set_env/{1,2}`) and/or when starting by oneself applications (e.g. see `otp_utils:start_applications/2`).

In any case this should result in `foobar_app:start/2` to be called at application startup, a function whose purpose is generally to spawn the root supervisor of this application.

Note that, alternatively (perhaps for some uncommon debugging needs), one may execute one's application (e.g. `foo`) by oneself, knowing that doing so requires starting beforehand the applications it depends on - be them Erlang-standard (e.g. `kernel, stdlib`) or user-provided (e.g. `bar, buz`); for that both their modules[7] and their `.app` file[8] must be found.

This can be done with:

```
$ erl -pa XXX/bar/ebin -pa YYY/buz/ebin -pa ZZZ/foo/ebin

Erlang/OTP 26 [erts-14.2] [source] [64-bit] [smp:8:8] [ds:8:8:10] [async-threads:1] [

Eshell V14.2 (press Ctrl+G to abort, type help(). for help)
1> application:ensure_all_started([kernel, stdlib, bar, buz, foo]).
```

Then the `foo` application shall be launched, and a shell be available to interact with the corresponding VM.

OTP Supervisors

The purpose of supervisors is to ease the development of fault-tolerant applications by building hierarchical process structures called *supervision trees*.

For that, supervisors are to monitor their children, that may be workers (typically implementing the `gen_{event,server,statem}` behaviour) and/or other supervisors (they can thus be nested).

We recommend to define a `foobar_sup:start_link/0` function (it is an user-level API, so any name and arity can be used). This `foobar_sup` module is meant to implement the `supervisor` behaviour (to be declared with

[7] If using Ceylan-Myriad, run, from the root of `foo`, `make copy-all-beams-to-ebins` to populate the `ebin` directories of all layers (knowing that by default each module is only to be found directly from its source/build directory, and thus such a copy is usually unnecessary).

[8] If using Ceylan-Myriad, run, from the root of `foo`, `make create-app-file`.

`-behaviour(supervisor).`), which in practice requires an `init/1` function to be defined.

So this results, in `foobar_sup`, in a code akin to:

```
-spec start_link() -> supervisor:startlink_ret().
start_link() ->
   % This will result in calling init/1 next:
   supervisor:start_link( _Registration={local, my_foobar_main_sup},
                          _Mod=?MODULE, _Args=[]).

-spec init(list()) -> {'ok', {supervisor:sup_flags(), [child_spec()]}}.
init(_Args=[]) ->
   [...]
   {ok, {SupSettings, ChildSpecs}}.
```

Declaring Worker Processes

Our `otp_utils` module may help a bit defining proper restart strategies and child specifications, i.e. the information regarding the workers that will be supervised, here, by this root supervisor.

For example it could be:

```
init(_Args=[]) ->
   [...]
   SupSettings = otp_utils:get_supervisor_settings(
                  _RestartStrategy=one_for_one, ExecTarget),
   % Always restarted in production:
   RestartSettings = otp_utils:get_restart_setting(ExecTarget),
   WorkerShutdownDuration =
       otp_utils:get_maximum_shutdown_duration(ExecTarget),
   % First child, the main Foobar worker process:
   MainWorkerChild = #{
       id => foobar_main_worker_id,
       start => {_Mod=foobar, _Fun=start_link,
                 _MainWorkerArgs=[A, B, C]},
       restart => RestartSettings,
       shutdown => WorkerShutdownDuration,
       type => worker,
       modules => [foobar] },
   ChildSpecs = [MainWorkerChild],
   {ok, {SupSettings, ChildSpecs}}.
```

Children are created synchronously and in the order of their specification[9].

So if `ChildSpecs=[A, B, C]`, then a child according to the A spec is first created, then, once it is over (either its `init/1` finished successfully, or it called `proc_lib:init_ack/{1,2}`[10] and then continued its own initialisation concurrently), a child according to the B spec is created, then once done a child according to the C spec.

[9] Yet some interleaving is possible thanks to `proc_lib:init_ack/1`.

[10] Typically: `proc_lib:init_ack(self())`.

Implementing Worker Processes

Such a worker, which can be any Erlang process (implementing an OTP behaviour, like `gen_server`, or not) will thus be spawned here through a call to the `foobar:start_link/3` function (another user-defined API) made by this supervisor. This is a mere *call* (an `apply/3`), not a spawn of a child process based on that function.

Therefore the called function is expected to *create* the worker process by itself, like, in the `foobar` module:

```
start_link(A, B ,C) ->
   [...]
   {ok, proc_lib:start_link(?MODULE, _Func=init,
       _Args=[U, V], _Timeout=infinity, SpawnOpts)}.
```

Here thus the spawned worker will start by executing `foobar:init/2`, a function not expected to return, often trapping EXIT signals (`process_flag(trap_exit, true)`), setting system flags and, once properly initialised, notifying its supervisor that it is up and running (e.g. `proc_lib:init_ack(_Return=self())`) before usually entering a tail-recursive loop.

Terminating Workers

Depending on the `shutdown` entry of its child specification, on application stop that worker may be terminated by different ways. We tend to prefer specifying a maximum shutdown duration: then the worker will be sent by its supervisor first a `shutdown` EXIT message, that this worker may handle, typically in its main loop:

```
receive
   [...]

   {'EXIT', _SupervisorPid, shutdown} ->
      % Just stop.
      [...]
```

If the worker fails to stop (on time, or at all) and properly terminate, it will then be brutally killed by its supervisor.

Supervisor Bridges

Non-OTP processes (e.g. WOOPER instances) can act as supervisors thanks to the supervisor_bridge module.

Such a process shall implement the `supervisor_bridge` behaviour, namely `init/1` and `terminate/2`. If the former function spawns a process, the latter shall ensure that this process terminates in a synchronous manner, otherwise race conditions may happen.

See traces_bridge_sup for an example thereof.

Extra Information

One may refer to;

- Ceylan-Myriad as an example of OTP *Library* Application
- Ceylan-WOOPER or Ceylan-Traces as examples of OTP *Active* Applications
- US-Web as an example of most of these supervisor principles

More Advanced Topics

Metaprogramming

Metaprogramming is to be done in Erlang through **parse transforms**, which are user-defined modules that transform an AST (for *Abstract Syntax Trees*, an Erlang term that represents actual code; see the Abstract Format for more details) into another AST that is fed afterwards to the compiler.

See also:

- this introduction to parse transforms
- Ceylan-Myriad's support for metaprogramming

Improper Lists

A proper list is created from the empty one (`[]`, also known as "nil") by appending (with the `|` operator, a.k.a. "cons") elements in turn; for example `[1,2]` is actually `[1 | [2 | []]]`.

However, instead of enriching a list from the empty one, one *can* start a list with any other term than `[]`, for example `my_atom`. Then, instead of `[2|[]]`, `[2|my_atom]` may be specified and will be indeed a list - albeit an improper one.

Many recursive functions expect proper lists, and will fail (typically with a function clause) if given an improper list to process (e.g. `lists:flatten/1`).

So, why not banning such construct? Why even standard modules like `digraph` rely on improper lists?

The reason is that improper lists are a way to reduce the memory footprint of some datastructures, by storing a value of interest instead of the empty list.

Indeed, as explained in this post, a (proper) list of 2 elements will consume:

- 1 list cell (2 words of memory) to store the first element and a pointer to second cell
- 1 list cell (2 more words) to store the second element and the empty list

For a total of 4 words of memory (so, on a 64-bit architecture, it is 32 bytes).

As for an improper list of 2 elements, only 1 list cell (2 words of memory) will be consumed to store the first element and then the second one.

Such a solution is even more compact than a pair (a 2-element tuple), which consumes 2+1 = 3 words. Accessing the elements of an improper list is also faster (one handle to be inspected vs also an header to be inspected).

Finally, for sizes expressed in bytes:

```
1> system_utils:get_size([2,my_atom]).
40

2> system_utils:get_size({2,my_atom}).
32

3> system_utils:get_size([2|my_atom]).
24
```

See also the 1, 2 pointers for more information.

Everyone shall decide on whether relying on improper lists is a trick, a hack or a technique to prohibit.

OpenCL

Open Computing Language is a standard interface designed to program many processing architectures, such as CPUs, GPUs, DSPs, FPGAs.

OpenCL is notably a way of using one's GPU to perform more general-purpose processing than typically the rendering operations allowed by GLSL (even compared to its compute shaders).

In Erlang, the cl binding is available for that.

A notable user thereof is Wing3D; one may refer to the `*.cl` files in this directory, but also to its optional build integration as a source of inspiration, and to wings_cl.erl.

Post-Mortem Investigations

Erlang programs may fail, and this may result in mere (Erlang-level) crashes (the VM detects an error, and reports information about it, possibly in the form of a crash dump) or (sometimes, quite infrequently though) in more brutal, lower-level core dumps (the VM crashes as a whole, like any faulty program run by the operating system); this last case happens typically when relying on faulty NIFs.

Monitoring To monitor a (live, Erlang) application, one may use:

- the (integrated) observer graphical tool (either locally, or remotely with `observer:start/1`)
- an headless (command-line) observer: observer_cli
- a monitoring and introspection application: system_monitor
- an observer web frontend

Erlang Crash Dumps If experiencing "only" an Erlang-level crash, a `erl_crash.dump` file is produced in the directory whence the executable (generally `erl`) was launched. The best way to study it is to use the `cdv` (refer to crashdump viewer) tool, available, from the Erlang installation, as `lib/erlang/cdv`[11].

Using this debug tool is as easy as:

[11] Hence, according to the Ceylan-Myriad conventions, in `~/Software/Erlang/Erlang-current-install/lib/erlang/cdv`.

```
$ cdv erl_crash.dump
```

Then, through the wx-based interface, a rather large number of Erlang-level information will be available (processes, ports, ETS tables, nodes, modules, memory, etc.) to better understand the context of this crash and hopefully diagnose its root cause.

Core Dumps In the worst cases, the VM will crash like any other OS-level process, and generic (non Erlang-specific) tools will have to be used. Do not expect to be pointed to line numbers in Erlang source files anymore!

Refer to our general section dedicated to core dumps for that.

Language Bindings

The two main approaches in order to integrate third-party code to Erlang are to:

- interact with it as if it was another Erlang node; we defined Ceylan-Seaplus for that purpose, to simplify the binding of C code; other libraries provide link to various languages
- directly link the current Erlang VM to this code, through NIF; for C, it can be done manually, or may be automatised thanks to nifty, which is an *Erlang NIF Wrapper Generator*; this can be especially useful for larger APIs (e.g. SDL); see also rusterl, which allows to write NIFs in safe Rust code

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 and also eqWAlizer exist, and are mostly complementary (see also 1 and 2).

More precisely:

- Dialyzer is a discrepancy analyzer that aims **to prove the presence of type-induced runtime crashes** (it may not be able to detect all type problems, yet "is never wrong"); Dialyzer does not use type specifications to guide the analysis: first it infers type information, and then only, if requested, it checks that information against the type specifications; so Dialyzer may operate with or without type specifications

- whereas tools like Gradualizer and eqWAlizer are more conventional type systems, based on the theory of gradual typing, that aim **to prove the absence of such crashes**; notably Gradualizer depends intimately on type specifications: by default, without them, no static typing happens

See EEP 61 for further typing-related information[12].

Also 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).

Beyond mere code, the messages exchanges between processes could also be typed and checked. Version upgrades could also benefit from it. Of course type-related errors are only a subset of the software errors.

Note that developments that rely on parse-transforms (almost all ours, directly or not) shall be verified based on their BEAM files (hence their actual, final output) rather than on their sources (as the checking would be done on code not transformed yet). See also the Type-checking Myriad section.

About Dialyzer

Installing Dialyzer Nothing is to be done, as Dialyzer is included in the standard Erlang distribution.

Using Dialyzer Our preferred options (beyond path specifications of course) are: `-Wextra_return -Wmissing_return -Wno_return -Werror_handling -Wno_improper_lists -Wno_unused -Wunderspecs`. See the `DIALYZER_OPTS` variable in Myriad's GNUmakevars.inc and the copiously commented options.

A problem is that typing errors tend to snowball: many false positives (functions that are correct but whose call is not because an error upward in the call-graph) may be reported (leading to the infamous `Function f/N has no local return`, which does not tell much).

[12] On a side note, the (newer) `dynamic()` type mentioned there is often used to mark "inherently dynamic code", like reading from ETS, message passing, deserialization and so on.

We recommend focusing on the first error reported for each module, and re-running the static analysis once supposedly fixed.

About Gradualizer

Installing Gradualizer We install Gradualizer that way:

```
$ cd ~/Software
$ mkdir gradualizer && cd gradualizer
$ git clone https://github.com/josefs/Gradualizer.git
$ cd Gradualizer
$ make escript
```

Then just ensure that the `~/Software/gradualizer/Gradualizer/bin` directory is in your PATH (e.g. set in your `.bashrc`).

Using Gradualizer Our preferred options (beyond path specifications of course) are: `--infer --fmt_location verbose --fancy --color always --stop_on_first_error`. See the `GRADUALIZER_OPTS` variable in Myriad's GNUmakevars.inc and the copiously commented options..

About eqWAlizer It is a tool developed in Rust.

Installing eqWAlizer We install eqWAlizer that way:

```
$ cd ~/Software
$ mkdir -p eqwalizer && cd eqwalizer
$ wget https://github.com/WhatsApp/eqwalizer/releases/download/vx.y.z/elp-linux.tar.gz
$ tar xvf elp-linux.tar.gz && mkdir -p bin && /bin/mv -f elp bin/
```

Then just ensure that the `~/Software/eqwalizer/bin` directory is in your PATH (e.g. set in your `.bashrc`).

Using eqWAlizer We use it out of a rebar3 context.

Settings are to be stored in a JSON file (e.g. `conf/foobar-for-eqwalizer.json`), to be designated thanks to the `--project` option.

See also the `EQWALIZER_OPTS` variable in Myriad's GNUmakevars.inc and its own myriad-for-eqwalizer.json project file.

(our first test was not successful, we will have to investigate more when time permits)

Software Robustness

Type correctness is essential, yet of course it does not guarantee that a program is correct and relevant. Other approaches, like the **checking** of other properties (notably concurrency, see Concuerror) can be very useful.

Beyond checking, **testing** is also an invaluable help for bug-fixing. Various tools may help, including QuickCheck.

Finally, not all errors can be anticipated, from network outages, hardware failures to human factor. An effective last line of defence is to rely on (this time

at runtime) **supervision mechanisms** in order to detect any kind of faults (bound to happen, whether expected or not), and overcome them. The OTP framework is an excellent example of such system, much useful to reach higher levels of robustness, including the well-known nine nines - that is an availability of 99.9999999%.

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 op-codes, x86-64 assembler generated by the JIT, etc.).

Short Hints

Dealing with Conditional Availability

Regarding modules, and from the command-line Depending on how Erlang was built in a given environment, some modules may or may not be available.

A way of determining availability and/or version of a module (e.g. `wx`, `cl`, `crypto`) from the command-line:

```
$ erl -noshell -eval 'erlang:display(code:which(wx))' -s erlang halt
"/home/bond/Software/Erlang/Erlang-24.2/lib/erlang/lib/wx-2.1.1/ebin/wx.beam"

$ erl -noshell -eval 'erlang:display(code:which(cl))' -s erlang halt
non_existing
```

A corresponding in-makefile test, taken from Wings3D:

```
# Check if OpenCL package is as external dependency:
CL_PATH = $(shell $(ERL) -noshell -eval 'erlang:display(code:which(cl))' -s erlang ha
ifneq (,$(findstring non_existing, $(CL_PATH)))
   # Add it if not found:
   DEPS=cl
endif
```

Regarding language features, from code Some features appeared in later Erlang versions, and may be conditionally enabled.

For example:

```
FullSpawnOpts = case erlang:system_info(version) >= "8.1" of

   true ->
       [{message_queue_data, off_heap}|BaseSpawnOpts];
```

```
        false ->
            BaseSpawnOpts

    end,
    [...]
```

Runtime Library Version

To know **which version of a library** (here, wxWidgets) a given Erlang install is using (if any), one may run an Erlang shell (`erl`), collect the PID of this (UNIX) process (e.g. `ps -edf | grep beam`), then trigger a use of that library (e.g. for `wx`, execute `wx:demo().`) in order to force its dynamic binding.

Then determine its name, for example thanks to `pmap ${BEAM_PID} | grep libwx`).

This may indicate that for example `libwx_gtk2u_core-3.0.so.0.5.0` is actually used.

Proper Naming

Variable Shorthands Usually we apply the following conventions:

- the *head* and *tail* of a list are designated as `H` and `T`, like in: `L = [H|T]`
- `Acc` means *accumulator*, in a tail-recursive function
- `K` designates a *key*, and `V` designates its associated *value*
- a list of elements is designated by a plural variable name, usually suffixed with `s`, like in: `Ints`, `Xs`, `Cars`

Function Pairs To better denote reciprocal operations, following namings for functions may be used:

- for services:
  - activation: `start` / `stop`
  - setup: `init` / `terminate`
- for instances:
  - life-cycle: `new` / `delete`
  - construction/destruction: `create` / `destruct` (e.g. avoid `destroy` there)

Formatting Erlang Code

Various tools are able to format Erlang code, see this page for a comparison thereof.

With `rebar3_format` For projects already relying on rebar, one may use rebar3_format (as a plugin[13]) that way:

```
$ cd ~/Software
$ git clone https://github.com/AdRoll/rebar3_format.git
$ cd rebar3_format
$ ERL_FLAGS="-enable-feature all" rebar3 format
```

Then `{project_plugins, [rebar3_format]}` shall be added to the project's `rebar.config`.

With `erlfmt` Another tool is erlfmt, which can be installed that way:

```
$ cd ~/Software
$ git clone https://github.com/WhatsApp/erlfmt
$ cd erlfmt
$ rebar3 as release escriptize
```

And then `~/Software/erlfmt/_build/release/bin` can be added to one's `PATH`.

Indeed `erlfmt` can be used as a rebar plugin or as a standalone escript - which we find useful, especially for projects whose build is not rebar-based.

Running it to reformat in-place source files is then as simple as:

```
$ erlfmt --write foo.hrl bar.erl
```

Language Features

Experimental features (such as `maybe` in Erlang 25) of the **compiler** (once Erlang has been built, they are *potentially available*) may have to be specifically *enabled* at runtime, like in `ERL_FLAGS="-enable-feature all" rebar3 as release escriptize`.

Disabling LCO

LCO means here *Last Call Optimisation*. This consists simply when, in a given module, a (typically exported) function `f` ends by calling a *local* function `g` (i.e. has for last expression a call like `g(...)`), in not pushing on the stack the call to `g`, but instead replacing directly the stackframe of `f` (which can be skipped here, as returning from `g` will mean directly returning from `f` as well) with a proper one for `g`.

This trick spares the use of one level of stack at each ending local call, which is key for recursive functions[14] (typically for infinitely-looping server processes): they remain then in constant stack space, whereas otherwise the number of their stackframes would grow indefinitely, at each recursive call, and explode in memory.

---

[13] Note that `rebar3_format` cannot be used as an escript (so no `ERL_FLAGS="-enable-feature all" rebar3 as release escriptize` shall be issued).

[14] When the last call of `f` branches to `f` itself, it is named TCO, for *Tail Call Optimisation* (which is thus a special - albeit essential - case of LCO).

So LCO is surely not an option for a functional language like Erlang, yet it comes with a drawback: if `f` ends with a last call to `g` that ends with a last call to `h` and a runtime error happens in them, none of these functions will appear in the resulting stacktraces: supposing all these functions use a library `foobar`, it will be as if the VM directly jumped from the entry point in the user code (typically a function calling `f`) to the failing point in a function of `foobar`; there will be no line number pointing to the expression of `f`, `g` or `h` that triggers the faulty behaviour, whereas this is probably the information we are mostly interested in - as these functions may make numerous calls to `foobar`'s function. This makes the debugging unnecessarily difficult.

Yet various workarounds exist (see this topic for more information) - just for debugging purposes - so that given "suspicious" functions (here `f`, `g` or `h`) are not LCO'ed:

- to have their returned values wrapped in a remote call to an identity function (we use `basic_utils:identity/1` for that)
- or to have them wrapped (at least their end, i.e. their last expression(s)) with `try ... catch E -> throw(E) end`
- or to return-trace these functions, as it temporarily disables LCO and allows to be very selective (one can limit this to a specific process or certain conditions, with match specs; refer to dbg for more details; note that the module of interest must be compiled with `debug_info` so that it can be traced)

The first workaround is probably the simplest, when operating on "suspicious" modules of interest (knowing that LCO *is* useful, and should still apply to most essential server-like processes).

One can nevertheless note that unfortunately LCO is not the only cause for the vanishing of calls in the stacktrace (even in the absence of any function inlining).

Within Ceylan-Myriad, if the (non-default) `myriad_disable_lco` compilation option is set (typically with the `-Dmyriad_disable_lco` command-line flag), this workaround is applied automatically on the modules on which the Myriad parse transform operates - i.e. all but bootstrapped modules (refer to the `lco_disabling_clause_transform_fun/2` function of the myriad_parse_transform module).

Using `run_erl/to_erl`

When using run_erl, lines like the following are output:

```
Write pipe '/tmp/launch-erl-3822033.w' found, waiting 2 seconds to ensure start-up is

 ************************************************************
 ** Node 'us_main' ready and running as a daemon.
 ** Use 'to_erl /tmp/launch-erl-3822033' to connect to that node.
 ** (then type CTRL-D to exit without killing the node)
 ************************************************************
 (authbind success reported)
```

```
EPMD names output (on default US-Main port ):
epmd: up and running on port 4507 with data:
name us_main at port 50002
name us_main_exec-xxx at port 60001
```

If connecting to that node with `to_erl /tmp/launch-erl-3822033` (hence from the local host), a direct access to an Erlang shell on that node is granted.

Remember that exiting the interpreter as usual (hitting CTRL-C twice) thus means killing that node; so prefer CTRL-D (once) instead!

Using wx

wx is now[15] the standard Erlang way of programming one's graphical user interface; it is a binding to the wxWidgets library.

Here are some very general wx-related information that may be of help when programming GUIs with this backend:

- in wxWidgets parlance, "window" designates any kind of widget (not only frame-like elements)
- if receiving errors about `{badarg,"This"}`, like in:

  ```
  {'_wxe_error_',710,{wxDC,setPen,2},{badarg,"This"}}
  ```

it is probably the sign that the user code attempted to perform an operation on an already-deallocated wx object; the corresponding life-cycle management might be error-prone, as some deallocations are implicit, others are explicit, and in a concurrent context race conditions easily happen

- if creating, from a wx-using process, another one, this one should set a relevant environment first (see `wx:set_env/1`) before using wx functions
- the way wx/wxWidgets manage event propagation is complex; here are some elements:
  - for each actual event happening, wx creates an instance of wxEvent (a direct, abstract child class of `wxObject`), which itself is specialised into many event classes that are more concrete
  - among them, they are so-called *command events*, which originate from a variety of simple controls and correspond to the wxCommandEvent mother class; by default only these command events are set to propagate, i.e. only them will be transmitted to the event handler of the parent widget if the current one does not process them by itself ("did not connect to them")

[15] `wx` replaced gs. To shelter already-developed graphical applications from any future change of backend, we developed MyriadGUI, an interface doing its best to hide the underlying graphical backend at hand: should the best option in Erlang change, that API would have to be ported to the newer backend, hopefully with little to (ideally) no change in the user applications.

  - by default, for a given type of event, when defining one's event handler (typically when connecting a process to the emitter of such events), this event will *not* be propagated anymore, possibly preventing the built-in wx event handlers to operate (e.g. to properly manage resizings); to restore their activation, `skip` (to be understood here as "propagate event" - however counter-intuitive it may seem) shall be set to `true` (either by passing a corresponding option when connecting, or by calling `wxEvent:skip/2` with `skip` set to `true` from one's event handler)[16]
  - when a process connects to the emitter of a given type of events (e.g. `close_window` for a given frame), this process is to receive corresponding messages and then perform any kind of operation; however these operations cannot be synchronous (they are non-blocking: the process does not send to anyone any notification that it finished handling that event), and thus, if `skip` is `true` (that is, if event propagation is enabled), any other associated event handler(s) will be triggered concurrently to the processing of these event messages; this may be a problem for example if a controller listens to the `close_window` event emitted by a main frame in order to perform a proper termination: the basic, built-in event handlers will then by default be triggered whereas the controller teardown may be still in progress, and this may result in errors (e.g. OpenGL ones, like `{{{badarg,"This"},{wxGLCanvas,swapBuffers,1}},...` because the built-in close handlers already deallocated the OpenGL context); a proper design is to ensure that `skip` remains set to false so that propagation of such events is disabled in these cases: then only the user code is in charge of closing the application, at its own pace[17]

- in terms of sizing, the dimensions of a parent widget prevail: its child widgets have to adapt (typically with sizers); if wanting instead that the size of a child dictates the one of its parent, the size of the client area of the parent should be set to the best size of its child, or its `fit/1` method shall be called

Extra information resources about `wx` (besides the documentation of its modules):

- the graphical class hierarchy of the wxWindow class, as this class corresponds to the concept of widget, which is very central
- graphical repositories of most widgets (screenshots thereof)

[16] MyriadGUI took a different convention: whether an event will propagate by default depends on its type, knowing that most of the types are to propagate. Yet the user can override these default behaviours, by specifying either the `trap_event` or the `propagate_event` subscription option, or by calling either the `trap_event/1` or the `propagate_event/1` function.

[17] This is why MyriadGUI applies per-event type defaults, thus possibly trapping events; in this case, if the built-in backend mechanisms would still be of use, they can be triggered by calling the `propagate_event/1` function from the user-defined handler, only once all its prerequisite operations have been performed (this is thus a way of restoring sequential operations).

- wxErlang: Getting started and Speeding up, by Arif Ishaq
- Doug Edmunds' wxerlang workups
- wxWidgets itself

Installing rebar3

One may use our install-rebar3.sh script for that (installed from sources, or prebuilt).

Using Emacs

Even when applying only the minimal Erlang configuration, a syntax-highlighting problem can often be noticed (at least with our code and Erlang 27.2 / Emacs 29.4), often starting after a docstring delimited with three double-quotes; then for example words in string literals are highlighted as if they were language keywords.

This can be fixed by selecting, in the `Erlang` (Emacs) menu, the `Syntax Highlighting` submenu, then any level (e.g. `Level 4`, corresponding to `erlang-font-lock-level-4`, which can be set with `(setq font-lock-maximum-decoration 4)`).

in *init.el*, a specific key (here, F11) may be assigned for that:

```
(global-set-key [f11] 'erlang-font-lock-level-4)
```

Problem: it would have to be entered again and again, even for a single buffer...

Contributing to Erlang/OTP

A simple procedure may be:

- first, check, especially in the case of a bug fixing, whether it corresponds to an already known issue
- read at least the Submitting Pull Requests section of the official Erlang/OTP contribution guide; for documentation-related contributions, refer to these guidelines
- from one's GitHub account, fork the Erlang/OTP official repository, `https://github.com/erlang/otp` (just click on `Fork`, at the top-right of this page), possibly as `erlang-otp` for clarity
- clone it; for example:

```
$ cd ~/Software/Erlang
$ git clone https://github.com/James-Bond/erlang-otp.git erlang-otp-github-my-fork
$ cd erlang-otp-github-my-fork
```

- (configure your clone appropriately in order to authenticate easily)
- let's suppose the branch of the upcoming pull request is to derive from the `maint` one, and be named `component/improve-something`:

```
$ git remote add upstream https://github.com/erlang/otp.git
$ git fetch upstream maint
$ git checkout maint
$ git checkout -b component/improve-something
```

- perform the changes, and commit them, with a message such as `Fix some element of something` (refer for that to the Writing good commit messages Erlang-official guidelines)
- push them: `git push origin HEAD`
- create a corresponding pull request (PR) by visiting the link proposed by the remote, for example `https://github.com/James-Bond/erlang-otp/pull/new/component/improv`
- in order to assess the corresponding authorisations of use/compliance, sign a suitable *Contributor License Agreement* (CLA), thanks to the CLAassistant that automatically participates to the PR conversation
- wait for the automated tests (continuous integration) to pass and a reviewer to accept or deny this PR, or to request changes that one's does one's best to apply

Finally the PR is successfully merged, accepted and closed (see this example) - and congratulations, you are an happy contributor to Erlang/OTP!

More in-depth readings on that topic:

- Everything you need to know to start contributing to Erlang today!, by Viacheslav Katsuba, especially to detail how changes should be performed and tested
- Writing OTP Modules, by Kenji Rikitake

Micro-Cheat Sheet

To avoid having to perform a lookup in the documentation:

- Erlang indices start at `1`, except the ones of the `array` module that are zero-based
- when they yield the same result, `=:=` is a bit more efficient than `==`
- for tuples of unknown number of elements:
  - **setting** an element is to be done with `setelement(positive_index(), tuple(), term()) -> tuple()`
  - **extracting** an element is to be done with `element(positive_index(), tuple()) -> term()`
  - **adding** an element at a given index (e.g. to append a record tag) is to be done with `erlang:insert_element(positive_index(), tuple(), term()) -> tuple()`
  - **removing** an element at a given index (e.g. to chop a record tag) is to be done with `erlang:delete_element(positive_index(), tuple()) -> tuple()`

(no need for the `erlang` module to be explicitly specified for the first two functions, as both are auto-imported)

- for `maps`: refer to map expressions, the maps module and the corresponding part of the efficiency guide; in short:
  - creating/updating maps is `EmptyMap = #{}`, `NewMFromScratch = #{K1 => V1, K2 => V2}` or `UpdatedM = M#{K1 => V1, K2 => V2}`
  - updating the value associated to an already-existing key `K` is `ModM = M#{K := V}`
  - matching is `#{K1 := V1, K2 := V2} = M` where the `K*` are guard expressions and the `V*` can be any pattern; matching an empty map is best done with `when Map == #{}`
  - obtaining the value `V` associated to a key `K` that:
    - is expected to exist: `V = maps:get(K, MyMap)` (generally could/should be directly matched instead)
    - may or may not exist: `case maps:find(K, MyMap) of {ok, V} -> ...; error -> ... end`
  - removing an entry: `ShrunkMap = maps:remove(Key,Map)`
- to detect in a guard whether a term is a map, there is no `is_map/1` guard available: `map_size/1` will fail the guard if its argument is not a map; a map includes its size in the header word, so `map_size/1` is O(1); furthermore, the JIT will inline calls to `map_size/1`
- a queue can be seen as being augmented (with `queue:in/2`) on the right ("rear" of the queue), and shrunk (with `queue:out/1`) on the left ("front" of the queue); for example, adding to an empty queue `a` then `b`, results, when converting the resulting queue in a list, in `[a,b]`; extracting an element of it returns `a`, and the remaining queue corresponds then to `[b]`
- a rule of thumb for naming in Erlang/OTP is that functions having `size` in their name are O(1), while functions having `length` in their name are O(N)
- when converting a float to a string, to set the precision (the number of digits after the decimal point) to `P`, use: `"~.Pf"`; for example, if `P=5`, we have: `"0.33333" = io_lib:format("~.5f", [1/3]).`
- the lesser-known `--` list subtraction operator returns a list that is a copy of the first argument where, for each element in the second argument, the first occurrence of this element (if any) is removed
- refer to the escape sequences, typically in order to specify characters like "space" (`$\s`); `$char` is the notation to designate the ASCII value or Unicode code-point of the character `char` (e.g. `$A`)
- see the always handy operator precedence and the various ways to handle exceptions with try/catch; note that to catch *all* exceptions - i.e. those of the `throw` class, but also of the `exit` and `error` classes, one may use: `try EXPR catch _:E -> ...` (rather than just: `try EXPR catch E -> ...`)

- in **guard expressions**:
  - because of side effects, no user-defined function is allowed in guards; refer to the two tables listing all the BIFs supported in guards expressions (test or non-test BIFs)
  - a guard will fail if it returns false *or if it throws an exception*
  - to express an "and" operator, use a comma, i.e. `","`, like in `f(A, B, C, D) when A =:= B, C =/= A + D ->`
  - to express an "or" operator, use a semi-colon, i.e. `";"`
  - `andalso` and `orelse` behave *there* as tests; they mostly convey the same meaning as `","` and `";"` respectively, except that they shortcut (they do not evaluate their second operand if the first one suffices), that they do not catch exceptions as they happen (i.e. if a first guard throws, any next guards will not be evaluated, and the whole guard sequence will fail) and that their code is a little less effective - but they can be nested inside guards (refer to this section for more details)
- to handle **binaries**:
  - a binary is a bit string whose number of bits is a multiple of 8 (it can be represented as whole bytes)
  - one may refer to this cheat sheet; see also an introduction to the bit syntax (including bit string comprehensions) and its full reference, the Constructing and Matching Binaries section and the binary module; see also our bin_utils module
  - as a rule of thumb:
    - even if it may seem a bit counter-intuitive, *integers* (the default matching type) are handled at the bit level (e.g. `A:16` means an integer on two bytes), whereas *binaries* are handled at the byte level (e.g. `B:16/binary` means 16 bytes) - indeed their bit-level counterparts are *bitstrings*
    - unlike lists, binaries are preferably built by appending on their right
- typically for non-critical code (e.g. tests or temporary, debugging code):
  - to avoid repeating a prefixing module, imports can be done, like in: `-import(lists, [map/2, foldl/3, foldr/3]).`
  - to suppress warnings/errors whenever elements are not used, specify for:
    - variables: `-compile(nowarn_unused_vars).`
    - functions (rather than silencing them with an `-export([f/1, g/3])`): `-compile([ {nowarn_unused_function, [{f,1}, {g,3}]} ]).`
    - types: they can be silenced with an `-export_type([t1/0, t2/1])`, which is better (more selective) than `-compile(nowarn_unused_type)` (unfortunately `-compile([ {nowarn_unused_type, [t1/0, t2/1]} ]).` is not supported)

- in boolean expressions **outside of guards**:
  - do not use `and` and `or`, which are strict boolean operators, not intended for control; indeed their precedence is low (parentheses around the conditions being then necessary, like in `(A =:= B) and (C =/= D)`) and, more importantly, they do not short-circuit, i.e. in `E = A and B`, B will be evaluated even if A is already known as false (so is E), and in `E = A or B`, B will be evaluated even if A is already known as true (so is E)
  - prefer `andalso` and `orelse`, which are *control* operators; they short-circuit, and their precedence is low enough to spare the need of extra parentheses[18]
  - so `case (A =:= B) and (C =/= D) of` shall become `case A =:= B andalso C =/= D of`
  - a common pattern is to use `andalso` in order *to evaluate a target expression iff a boolean expression is true*, to replace a longer expression like:

```
case BOOLEAN_EXPR of

    true ->
        DO_SOMETHING;

    false ->
        ok

end
```

with the equivalent (provided `BOOLEAN_EXPR` evaluates to either `true` or `false` - otherwise a `bad argument` exception will be thrown) yet shorter: `BOOLEAN_EXPR andalso DO_SOMETHING`; for example: `[...], OSName =:= linux andalso fix_for_linux(), [...]`

Similarly, `orelse` can be used *to evaluate a target expression iff a boolean expression is false*, to replace a longer expression like:

```
case BOOLEAN_EXPR of

    true ->
        ok;

    false ->
        DO_SOMETHING;

end
```

[18] Note that it is *not* the case of the `and` and `or` operators, whose precedence is higher than, notably, the comparison operators.

For example a clause defined as `f(I) when is_integer(I) and I >= 0 -> ...` will never be triggered as it is interpreted as `f(I) when (is_integer(I) and I) >= 0 -> ...`, and the `and` guard will always fail as I is an integer here, not a boolean. So such a clause should be defined as the (correct) `f(I) when is_integer(I) andalso I >= 0 -> ...` instead.

with: `BOOLEAN_EXPR orelse DO_SOMETHING`; for example: `[...], file_utils:exists("/etc/passwd") orelse throw(no_password_file), [...]`

In both `andalso / orelse` cases, the `DO_SOMETHING` branch may be a single expression, or a sequence thereof (i.e. a body), in which case a `begin/end` block may be of use, like in:

```
file_utils:exists("/etc/passwd") orelse
  begin
      trace_utils:notice("No /etc/password found."),
      throw(no_password_file)
  end
```

Similarly, taking into account the aforementioned precedences, `Count =:= ExpectedCount orelse throw({invalid_count, Count})` will perform the expected check.

Be wary of not having precedences wrong, lest bugs are introduced like the one in:

```
MaybeListenerPid =:= undefined orelse
   MaybeListenerPid ! {onDeserialisation, [self(), FinalUserData]}
```

(`orelse` having more priority than `!`, parentheses shall be added, otherwise, if having a PID, the message will actually be sent to any process that would be registered as `true`)

For the record, an alternative is `[f(...)  || BoolCondition]`, like in: `_ = [put('$ancestors', Ancestors) || Shell =/= {}]` (in `group.erl`).

Some of these elements have been adapted from the Wings3D coding guidelines.

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 (use this mirror in order to search through the past messages of this list)
- for those who are interested in parse transforms (the Erlang way of doing metaprogramming), the section about The Abstract Format is essential (despite not being well known)
- to better understand the inner workings of the VM: The Erlang Runtime System, a.k.a. "the BEAM book", by Erik Stenman
- BEAM Wisdoms, by Dmytro Lytovchenko

Rust

Organisation: Copyright (C) 2024-2025 Olivier Boudeville
Contact: about (dash) howtos (at) esperide (dot) com
Creation date: Tuesday, March 12, 2024
Lastly updated: Wednesday, March 19, 2025

Table of Contents

Overview

Rust is a multi-paradigm, general-purpose, efficient, safe language available as free software; see its official website for more details.

Documentation

One should read the official Rust book (*The Rust Programming Language*), which is clear and well written.

See also:

- Rust on Arch Linux

Installation

Even if done here in the context of Arch Linux, for the procedure that we recommend the actual distribution does not matter.

As we intend to develop (rather than just running or installing Rust software), we prefer relying on the Rustup toolchain manager, so that multiple toolchains, for multiple platforms and architectures, can be used.

Rather than using `pacman` to install its `rustup` package, we prefer going the more standard Rust way. One should follow the official Rust guidelines for that; here is our corresponding procedure:

```
$ curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs > install-rustup.sh
# Inspect install-rustup.sh before running it.
$ sh install-rustup.sh
```

From now the `rustc` compiler and the Cargo package manager are available on the system, from one's account (in the `~/.rustup` and `~/.cargo` trees respectively). All Rust-related commands are then be available from `~/.cargo/bin`.

One may prefer customising the installation and avoiding that one's shell configuration is automatically modified; then it is just a matter of editing typically one's `~/.bashrc` or related by oneself and adding there `export PATH="${HOME}/.cargo/bin:${PATH}"` (another option is to add `. ${HOME}/.cargo/env` instead).

Then updating the current shell accordingly (e.g. `. ~/.bashrc`) allows to check that for example `rustc` can now be executed as intended.

As Rust does not perform the linking by itself, a linker (typically provided by `gcc`) must be available.

Afterwards Rust may be updated thanks to `rustup update`.

Should some day Rust and rustup have to be uninstalled, run `rustup self uninstall`.

Examples

Most Basic One

In `~/hello.rs`:

```
fn main() {
   println!("Hello from Ceylan-HOWTOs!");
}
```

One may notice the use of a macro (`!` suffix), and that most lines of Rust code end with a semicolon (`;`).

To be compiled and linked with `rustc hello.rs` and run with `./hello`; yields as expected: `Hello from Ceylan-HOWTOs!`.

This executable occupies 3.7MB (!); more information:

```
$ file hello
hello: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, in
BuildID[sha1]=c77ef3f032c2f1961866e0889d54d6342a9e3554, for GNU/Linux 4.4.0, with debu
```

No need to devise one's GNU make automatic rules - cargo will do best.

Related Tools

Included with rustup

Cargo Cargo is the Rust integrated dependency manager and build tool.

In Rust, a package of code is referred to as a *crate*. So Cargo is the build system and package manager of choice to access Rust community's crate registry.

More precisely, Cargo allows installing crates and publishing ones, so that Rust projects can declare their various dependencies and so that their build is repeatable. See also the Cargo guide.

For its configuration, Cargo relies on a `Cargo.toml` manifest file, placed at the root of the tree of the package it applies to.

Its format is named TOML (*Tom's Obvious, Minimal Language*), and the various keys that it introduces are listed here.

Rustfmt Rustfmt is the source-formatting tool of choice: it formats Rust code according to the official style guidelines[19].

We found very useful that a stable, conventional formatting exists, and decided from the start to systematically use it. This allows a great standardization of sources, for easier readings and simplified merges.

Rustfmt should be readily available with `rustup`; otherwise execute `rustup component add rustfmt`.

External Ecosystem

rust-analyzer This is a Language Server Protocol implementation for Rust, typically used by IDEs.

There are multiple options to install it; rather than installing as an (OS) package, we prefer once again using `rustup`, with: `rustup component add rust-analyzer`.

Once installed, running `rust-analyzer --version` should switch from:

```
error: Unknown binary 'rust-analyzer' in official toolchain 'stable-x86_64-unknown-lin
```

to something like:

```
rust-analyzer 1.76.0 (07dca48 2024-02-04)
```

As rust-analyzer will need the Rust source code, the latter can be directly installed with: `rustup component add rust-src`.

Using Emacs with Rust The Emacs light-weight rust-mode can be used, yet rustic (which is based on `rust-mode`) provides additional features that we like.

A prerequisite thereof is rust-analyzer (and of course a recent Emacs).

Base Approach One may rely on straight.el to manage one's Emacs packages (as a replacement for the Emacs now built-in `package.el`), so, provided that the straight.el bases are respected, having `rustic` is just a matter of adding, typically in one's `~/.emacs.d/init.el`, a `(straight-use-package 'rustic)` line.

More Involved Approach We prefer installing and configuring `rustic` with `use-package rustic`, as done in Robert Krahn's page; see our complete init.el for that.

Then opening a Rust source file (e.g. `hello.rs`) should automatically exhibit syntax highlighting and the `Rustic` mode.

[19] e.g. indent with four spaces - not tabs; place the curly braces according to this convention, etc.

Notably, saving a Rust source file will automatically format it with Rustfmt, which is neat.

See also:

- the WikEmacs page
- the Robert Krahn's page about it

More Advanced Topics

Mode of Operation

Rust relies on LLVM to generate its code, in order to be platform and hardware-agnostic, and to achieve best performance.

Quick Facts

- Rust is well suited for **embedded development**, because it can target many platforms (thanks to LLVM) and can be as low-level as wanted; notably using `#![no_std]` allows to benefit from the *Rust Core Library* without relying on the *Rust Standard Library*, for smaller binary sizes and possibly improved performance
- thanks to the use of rustup and its support of toolchains, **cross-compiling** is rather easy; for instance, to install Rust using the stable channel for Windows, using the GNU Compiler, just execute `rustup toolchain install stable-x86_64-pc-windows-gnu`

Language Bindings

Short Hints

Optimizing for native CPU platform

The current target platform is given by `rustup toolchain list`; in our case it is `stable-x86_64-unknown-linux-gnu`.

Then, to request Cargo to always compile code optimized for the native CPU platform, one may add in `~/.cargo/config`:

```
[target.x86_64-unknown-linux-gnu]
rustflags = ["-C", "target-cpu=native"]
```

Note that the resulting binary is expected to depend on the precise local CPU and thus should not be distributed - lest it cannot be run on other computers.

Accelerating Builds

A shared compilation cache - specifically sccache may be used to speed up builds.

See this section for more details.

Micro-Cheat Sheet

- to get Rust (compiler) version: `rustc -V`; for cargo: `cargo --version`
- to create a binary (application) *my_project* package: `cargo new my_project`; Git files will be generated iff not already in a clone

Rust Resources

- the reference is the Rust official website

About 3D

Organisation: Copyright (C) 2021-2025 Olivier Boudeville

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

Creation date: Saturday, November 20, 2021

Lastly updated: Wednesday, March 19, 2025

Table of Contents

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

Cross-Platform Game Engines

The big three are Godot, Unreal Engine and Unity3D[20].

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.

Godot (version 3.4.1) will not be able to load FBX files that reference formats like PSD or TIF and/or of older versions (e.g. FBX 6.1). See for that our section regarding format conversions.

[20]Others could be considered, like Cocos Creator, an open-source engine using TypeScript and WebGL.

Installation On Arch Linux, one may simply use `pacman -Sy godot`.

Or just, for maximum control, one may instead directly download the GNU/Linux version from the Godot official website.

If planning to be able to develop in C# in addition to GDScript (refer to our Scripting Language section), prefer the .NET version (as opposed to the Standard one), i.e. the `".NET (x86_64)"` version - provided that the support for .NET is already secured, either based on the Microsoft .NET SDK or, possibly better, on the Mono SDK.

The installation procedure that we prefer can be done automatically thanks to our install-godot.sh script.

Use

Scripting Language Users of the Godot API may develop notably in GDScript (extension: `*.gd`) and/or in C# (extension: `*.cs`) and/or in C++.

See this comparison - knowing that languages can be mixed and matched.

We prefer using C# to GDScript or C++, as:

- C# is statically typed (GDScript is dynamically typed, with implicit casts)
- C# is widely-used / general-purpose (and Unity supports it as well), as opposed to GDScript
- C# is C++-inspired (yet offers a safer model, notably in terms of life-cycle management), whereas GDScript is Python-inspired[21]; so for example C# does not rely on indentation to define clauses

Development Hints

- all scripts are classes, which are by default anonymous
- before releasing one's game, it is certainly better to protect it thanks to some level of encryption (see this thread)

Important Paths A configuration tree lies in `.config/godot`, a cache tree in `~/.cache/godot`.

Logs Godot logs are stored per-project; e.g. `~/.local/share/godot/app_userdata/my-test-project/` past log files are kept once timestamped. They tend not to have interesting content.

[21] With additionally quite many differences. For example, with GDScript, variables are typed, like in `var my_msg : String = 'Hello!'`. So `my_msg = 7` is to result in a (*runtime*) error. Python lists are Godot arrays. `None` is `null`. A switch-like operator (`match`) exists, etc.

Assets The official Godot Asset Library, whose assets are at least mostly available through the rather permissive MIT licence, coexists with (probably too many) unofficial ones, like Godot A.L. (AGPLv3 license), Godot Asset Store, GodotAssets, etc.

For obvious reasons, many of the current open source assets are of a significantly lower quality than their non-Godot commercial counterparts. We believe than, until Godot-related assets progress (either as open-source or commercial ones), as soon as a game is a bit ambitious, relying on the asset stores of the other engines (see Engine-related Assets) and/or on asset providers is a better option.

Sources of Godot-related Information

- the official Godot Engine video channel
- the GDQuest channel, offering a large number of Godot tutorials covering many topics

Unreal Engine

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

The Unreal Engine 5 brings new features that may be of interest, including a fully dynamic global illumination and reflection system (Lumen, not requiring baked lightmaps anymore), a virtualized geometry system (Nanite, simplifying detailed geometries on the fly) and a quality 2D/3D asset library (the Quixel Megascans library, obtained from real-world scans).

Unreal does not offer a scripting language anymore, user developments have to be done in C++ (beyond *Blueprints Visual Scripting*).

Its licence is meant to induce costs only when making large-enough profits; more precisely, a *5% royalty is due only if you are distributing an off-the-shelf product that incorporates Unreal Engine code (such as a game) and the lifetime gross revenue from that product exceeds $1 million USD* (the first $1 million remaining royalty-exempt); in case of large success, it may be a costlier licence than Unity.

With an Unreal user account, the sources of the engine (in its latest stable version, 5) can be examined on Github (so it is open source - yet not free software).

See its official website.

Unreal Assets Purchased assets from the Unreal marketplace 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; see also this thread).

Note that parts of the content of assets will be Unreal-specific (`*.uasset`, `*.umap`, etc.), like scripts. Yet technically many can be adapted to other engines (see for example Exporting from Unreal Engine to Godot).

Unity3D

Unity is most probably still the most popular cross-platform game engine, despite recent controversies.

Regarding the licensing of the engine, various plans apply (warning: they may have changed since this writing), depending notably on whether one subscribes as an individual or a team, and on one's profile, revenue and funding; the general idea is not taking royalties, but **flat**, per seat yearly fees increasing with the organisation "size" (typically in the $400-$1800 range, per seat).

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 technically without too much effort/constraints (this may happen due to prefabs, specific animations, materials or shaders, conventions in use, etc.).

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-privileged 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 (e.g. GNU/Linux and Windows "Build Supports").

We rely here on the Unity version 2021.2.7f1.

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.

Running Unity Just execute `unityhub`, which requires signing up and activating a licence.

Troubleshooting The log files are stored in `~/.config/unity3d`:

- Unity Editor: `Editor.log` (the most interesting one)
- Unity Package Manager: `upm.log`
- Unity Licensing client: `Unity.Licensing.Client.log`

If the editor is stuck (e.g. when importing an asset), one may use as a last resort kill-unity3d.sh.

In term of persistent state, beyond the project trees themselves, there are:

- `~/.config/UnityHub/` and `~/.local/share/UnityHub/`
- `~/.config/unity3d/` and `~/.local/share/unity3d/`

(nothing in `~/.cache` apparently)

Unity Assets Once ordered through the Unity Asset Store, assets can be downloaded through the `Window -> Package Manager` menu, by replacing, in the top `Packages` drop-down, the `In Project` entry by the `My Assets` one. After having selected an asset, use the `Download` button at the bottom-right of the screen.

Then, to gain access to such downloaded assets, of course the simplest approach is to use the Unity editor; this is done by creating a project (e.g. `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 filesystem of the operating system with their actual name and content, for example in `MyProject/Assets/CorrespondingAssetProvider/AssetName`. Unfortunately we experienced reproducible freezes when importing some resources.

Yet such Unity packages, once downloaded (whether or not they have been imported in projects afterwards) are just files that are 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 (e.g. `167e85f3d750117459ff6199b79166fd`)[22]; such directory generally contains at least 3 files:

- `asset`: the resource itself, renamed to that unique checksum name, yet containing its exact original content (e.g. 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 (e.g. `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 (e.g. glTF 2.0); see blender import and blender convert for that.

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

3D Data

File Formats

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

glTF We prefer relying 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 for actual releases

See also the glTF 2.0 quick reference guide, the related section of Godot and this standard viewer of predefined glTF samples.

This (generic) online glTF viewer proved lightweight and convenient, notably because it displays errors, warnings and information regarding the glTF data that it decodes.

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

FBX, OBJ, etc. 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. FBX comes in two flavours: text-based (ASCII) or binary; see this retro-specification for more information.

In General Refer to blender import in order to handle the most common 3D file formats, and the next section about conversions.

The `file` command is able to report the version of at least some formats; for example:

```
# Means FBX 7.3:
$ file foobar.fbx
foobar.fbx: Kaydara FBX model, version 7300
```

Too often, some tool will not be able to load a file and will fail to properly report why. When suspecting that a binary file (e.g. a FBX one) references external content either missing or in an unsupported format (e.g. PSD or TIFF?), one may peek at their content without any dedicated tool, directly from a terminal, like in:

```
$ strings my_asset.fbx | sort | uniq | grep '\.'
```

This should list, among other elements, the paths that such a binary file is embedding.

Conversions

Due to the larger number of 3D file formats and the role of commercial software, interoperability regarding 3D content is poor and depends on many versions (of tools and formats).

Recommended Option: Relying on Blender Using blender import is the primary solution that we see: a content, once imported in Blender, can be exported in any of the supported formats.

Yet this operation may fail, for example on "older" FBX files, whose FBX version (e.g. 6.1) is not supported by Blender ("*Version 6100 unsupported, must be 7100 or later*") or by other tools such as Godot. See also the Media Formats supported by Blender.

Another option of interest is to use Godot's FBX2glTF command-line tool.

Workaround #1: Using Autodesk FBX Converter The simpler approach seems to download the (free) Autodesk FBX Converter and to use wine to run it on GNU/Linux. Just install then this converter with: `wine fbx20133_converter_win_x64.exe`.

A convenient alias (based on default settings, typically to be put in one's `~/.bashrc`) can then be defined to run it:

```
$ alias fbx-converter-ui="$HOME/.wine/drive_c/Program\ Files/Autodesk/FBX/FBX\ Convert
```

Conversions may take place from, for example, FBX 6.1 (also: 3DS, DAE, DXF, OBJ) to a FBX version in: 2006, 2009, 2010, 2011, 2013 (i.e. 7.3 - of course the most interesting one here), but also DXF, OBJ and Collada, with various settings (embedded media, binary/ASCII mode, etc.).

An even better option is to use directly the command-line tool `bin/FbxConverter.exe`, which the previous user interface actually executes. Use its `/?` option to get help, with interesting information.

For example, to update a file in a presumably older FBX into a 7.3 one (that Blender can import):

```
$ cd ~/.wine/drive_c/Program\ Files/Autodesk/FBX/FBX\ Converter/2013.3/bin
$ FbxConverter.exe My-legacy.FBX newer.fbx /v /sffFBX /dffFBX /e /f201300
```

We devised the update-fbx.sh script to automate such an in-place FBX update.

Unfortunately, at least on one FBX sample taken from a Unity package, if the mesh could be imported in Blender, textures and materials were not (having checked `Embed media` in the converter or not).

Workaround #2: Relying on Unity Here the principle is to import a content in Unity (the same could probably be done with Godot), and to export it from there.

Unity does not allow to export for example FBX natively, however a package for that is provided. It shall be installed first, once per project.

One shall select in the menu `Window -> Package Manager`, ensure that the entry `Packages:` points to `Unity Registry`, and search for `FBX Exporter`, then install it (bottom right button).

Afterwards, in the `GameObject` menu, an `Export to FBX` option will be available. Select the `Binary` export format (not `ASCII`) if wanting to be compliant with Blender.

Examples of 3D Content

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
- OBJ
- IFC; direct: Basic house (requires the BlenderBIM add-on for BIM support in Blender)

Engine-related Assets

Technically, and also legally, assets obtained in the context of any of these engines can be at least partially exported and adapted for re-use in other engines.

Textures may be exported as PNG, animations in the FBX (proprietary) file format, that can nevertheless be imported in Blender and converted to other file formats (e.g. glTF 2.0); see blender import and blender convert for that.

Scripts and alike (Nodes, Prefabs, Blueprints) are engine-specific, yet may be recreated or at least translated to some extent.

Asset Providers

Usually, for one's creation, much multimedia artwork has to be secured: typically graphical assets (e.g. 2D/3D geometries, animations, textures) and/or audio ones (e.g. music, sounds, speech syntheses, special effects).

Instead of creating such content by oneself (not enough time/interest/skill?), it may be more relevant to rely on specialised third-parties.

Hiring a professional or a freelance is then an option. This is of course relatively expensive, involves more efforts (to define requirements and review the results), longer, but it is supposed to provide exactly the artwork that one would like.

Another option is to rely on specialised third-party providers who **sell non-exclusive licences for the content that they offer**.

These providers can be either direct **content producers** (companies with staffs of modellers), or **asset aggregators** (marketplaces that federate the offers of many producers of any size) that are often created in link to a given multimedia engine. An interesting point is that assets purchased in these stores generally can be used in any technical context, hence are not meant to be bound to the corresponding engine.

Nowadays, much content is available, in terms of theme/setting (e.g. Medieval, Science-Fiction), of nature (e.g. characters, environments, vehicles), etc. and the overall quality/price ratio seems rather good.

The main advantages of these marketplaces is that:

- they favor the competition between content providers: the clients can easily compare assets and share their opinion about them
- they generalised simple, standard, unobtrusive licensing terms; e.g. royalty free, allowing content to be used as they are or in a modified form, not limited by types of usage, number of distributed copies, duration of use, number of countries addressed, etc.; the general rule is that much freedom is left to the asset purchasers provided that they use such assets for their own projects (rather than for example selling the artwork as they are)

The main content aggregators that we spotted are (roughly by decreasing order of interest, based on our limited experience):

- the Unity Asset Store, already discussed in the Unity Assets section; websites like this one allow to track the significant discounts that are regularly made on assets
- the UE Marketplace, i.e. the store associated to the Unreal Engine; in terms of licensing and uses (see also this section):
  - this article states that *When customers purchase Marketplace products, they get a non-exclusive, worldwide, perpetual license to download, use, copy, post, modify, promote, license, sell, publicly perform, publicly display, digitally perform, distribute, or transmit your product's content for personal, promotional, and/or commercial purposes. Distribution of products via the Marketplace is not a sale of the content but the granting of digital rights to the customer.*
  - this one states that *Any Marketplace products that have not been created by Epic Games can be used in other engines unless otherwise specified.*
  - this one states that *All products sold on the Marketplace are licensed to the customer (who may be either an individual or company) for the lifetime right to use the content in developing an unlimited number of products and in shipping those products. The customer is also licensed to make the content available to employees and contractors for the sole purpose of contributing to products controlled by the customer.*
- itch.io
- Turbosquid
- Free3D
- CGtrader
- ArtStation

- Sketchfab
- 3DRT
- Reallusion
- Arteria3D
- the GameDev Market (GDM)
- the Game Creator Store

Many asset providers organise interesting discount offers (at least -50% on a selection of assets, sometimes even more for limited quantities) for the Black Friday (hence end of November) or for Christmas (hence mid-December till the first days of January).

Modelling Software

Blender

Blender is a very powerful free software 3D toolset.

Blender (version 3.0.0) can import FBX files of version at least 7.1 ("7100"). See for that our section regarding format conversions.

We recommend the use of our Blender shell 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, advanced subdivision *modeler*[23], available for GNU/Linux, Windows and Mac OS X. Wings3D relies on OpenGL.

It can be installed on Arch Linux, from the AUR, as `wings3d`; one can also rely on our Wings3D shell scripts in order to install and/or execute it.

We prefer using the Blender-like camera navigation conventions, which can be set in Wings3D by selecting `Edit -> Preferences -> Camera -> Camera Mode` to `Blender`.

See also:

- its official website
- its development project
- its build instructions for UNIX-like systems

[23] As opposed to *renderer*; yet Wings3D integrates an OpenCL renderer as well, deriving from LuxCoreRender, an open-source Physically Based Renderer (it simulates the flow of light according to physical equations, thus producing realistic images of photographic quality).

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.

A `draco` AUR package exists, and results notably in creating the `/usr/lib/libdraco.so` shared library file.

Even once this package is installed, when Blender exports a mesh, a message like the following is displayed:

```
'/usr/bin/3.0/python/lib/python3.10/site-packages/libextern_draco.so' does
not exist, draco mesh compression not available, please add it or create
environment variable BLENDER_EXTERN_DRACO_LIBRARY_PATH pointing to the folder
```

Setting the environment prior to running Blender is necessary (and done by our `blender-*.sh` scripts):

```
$ export BLENDER_EXTERN_DRACO_LIBRARY_PATH=/usr/lib
```

but not sufficient, as the built library does not bear the expected name.

So, as root, one shall fix that once for all:

```
$ cd /usr/lib
$ ln -s libdraco.so libextern_draco.so
```

Then the log message will become:

```
'/usr/lib/libextern_draco.so' exists, draco mesh compression is available
```

The Compressonator

The Compressonator is an AMD tool (as a GUI, a command-line executable and a SDK) designed to compress textures (e.g. in DXT1, DXT3 or DXT5 formats; typically resulting in a `.dds` extension) and to generate mipmaps ahead of time, so that it does not have to be done at runtime.

F3D

f3d (installable from the AUR) is a fast and minimalist VTK-based 3D viewer.

Such a viewer is especially interesting to investigate whether a tool failed to properly export a content or whether it is the next tool that actually failed to properly import, and to gain another chance of accessing to relevant error messages.

Mikktspace

This tool (official website), created by Morten S. Mikkelsen, is a de facto (free) standard in terms of normal map baker: it generates *Tangent Space Normal Maps* (tangents), and helps ensuring consistency between 3D applications (such as Blender).

These fields of normals may be seen as an encoding - explaining why conventions like the ones enforced by this tool (which became an implementation standard) help performing a suitable, robust reciprocal decoding.

Mixamo

Mixamo is a website that allows to download and use for free a large number of rather high-quality 3D characters (about 110 of them; all being textured and rigged) and animations (about 2500 of them; full-body character animations, captured from professional motion actors), which can be arbitrarily mixed and matched.

This website allows also to rig one's (humanoid) character (see `Upload character`).

The licence attached to Mixamo is rather permissive; notably:

```
You can use both characters and animations royalty free for personal, commercial, and
   Incorporate characters into illustrations and graphic art.
   3D print characters.
   Create films.
   Create video games.
```

OpenGL Corner

Conventions

Refer to our Mini OpenGL Glossary for most of the terms used in these sections.

Code snippets will correspond to the OpenGL/GLU APIs as they are exposed in Erlang, in the gl and glu modules respectively.

These translate easily for instance in the vanilla C GL/GLU implementations. As an example, gl:ortho/6 (`6` designating here the arity of that function, i.e. the number of the arguments that it takes) corresponds to its C counterpart, glOrtho.

The reference pages for OpenGL (in version 4.x) can be browsed here.

Note that initially the information in this page related to older versions of OpenGL (1.1, 2.1, etc.; see history) that relied on a fixed pipeline (no shader support) - whereas, starting from OpenGL 3.0, many of the corresponding features were marked as deprecated, and actually removed as a whole in 3.1. However, thanks to the *compatibility context* (whose support is not mandatory - but that all major implementations of OpenGL provide), these features can still be used.

Yet nowadays relying on at least the OpenGL 3 *core* context (not using the *compatibility* context) would be preferable (source: this thread). Still better options would be to rely on OpenGL 4 Core or OpenGL ES 2+, or libraries on top of Vulkan, like wgpu. Specific libraries also exist for rendering for the web and for mobile, like WebGPU.

As of 2023, the current OpenGL version is 4.6; we will try to stick to the latest ones (4.x) only (e.g. skipping intermediate changes in 3.2); even though in this document reminiscences of older OpenGL versions remain, the current minimum that we target is the **Core Profile of OpenGL 3.3**, which is "modern OpenGL" and introduced most features that still apply; it will halt on error if any deprecated functionality is used.

For more general-purpose computations (as opposed to rendering operations) to be offset to a GPU/GPGU, one may rely on OpenCL instead.

The mentioned tests will be Ceylan-Myriad ones, typically located here.

Basics

- OpenGL is a **software interface to graphics hardware**, i.e. the *specification* of an API (of around 150 functions in its older version 1.1), developed and maintained by the Khronos Group

- a video card will run an *implementation* of that specification, generally developed by the manufacturer of that card; a good rule of thumb is to always update one's video card drivers to their **latest stable version**, as OpenGL implementations are constantly improved (bug-fixing) and updated (with regard to newer OpenGL versions)

- OpenGL concentrates on **hardware-independent 2D/3D rendering**; no commands for performing window-related tasks or obtaining user input are included; for example frame buffer configuration is done outside of OpenGL, in conjunction with the windowing system

- OpenGL offers only **low-level primitives** organised through a pipeline in which vertices are assembled into primitives, then to fragments, and finally to pixels in the frame buffer; as such, OpenGL is a building-block for higher-level engines (e.g. like Godot)

- OpenGL is a **procedural** (function-based, not object-oriented) **state machine** comprising a larger number of variables defined within a given OpenGL state (named an *OpenGL context*; comprising vertex coordinates, textures, frame buffer, etc.); said otherwise, all OpenGL state variables behave like "global" variables, more precisely they are actually relative to an OpenGL context that is often implicit; when a parameter is set, it applies and lasts as long as it is not modified (if still using the same OpenGL context); the effect of an OpenGL command may vary depending on whether certain modes are enabled (i.e. whether some state variables are set to a given value)

- so the **currently processed element** (e.g. a vertex) **inherits (implicitly) the current settings of the context** (e.g. color, normal, texture coordinate, etc.); this is the only reasonable mode of operation, knowing that a host of parameters apply whenever performing a rendering operation (specifying all these parameters would not be a realistic option); as a result, any specific parameter shall be set first (prior to triggering such an operation), and is to last afterwards (being "implicitly inherited"), until possibly being reassigned in some later point in time

- OpenGL respects a **client/server execution model**: an application (a specific client, running on a CPU) issues commands to a rendering server (on the same host or not - see GLX; generally the server can be seen as running on a local graphic card), that executes them **sequentially and in-order**; as such, most of the calls performed by user programs are **asynchronous**: they are triggered by the (client) program through OpenGL, and return almost immediately, whereas they have not been executed (by the server) 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; additional facilities like *Display Lists* allow to pipeline operations (as opposed to the default *immediate mode*), which are accumulated for processing at a later time, as fast as the graphic card can then process them
- state variables are mostly server-side, yet some of them are client-side; in both cases, they can be gathered in *attribute groups*, which can be pushed on, and popped from, their respective server or client attribute stacks
- OpenGL manages two types of data, handled by mostly different paths of its rendering pipeline, yet that are ultimately integrated in the framebuffer through fragment-yielding rasterization:
  - geometric data (vertices, lines, and polygons)
  - pixel data (pixels, images, and bitmaps)
- vertices and normals are transformed by the **model-view** and **projection** matrices (that can be each set and transformed on a stack of their own), before being used to produce an image in the frame buffer; as for texture coordinates, they are transformed by the **texture** matrix
- textures may reside in the main, general-purpose, client, **CPU-side memory** (large, but slow to access for the rendering) and/or in any auxiliary, dedicated, server-side **GPU memory** (more constrained, hence prioritized thanks to *texture objects*; and, rendering-wise, of significantly higher performance)
- OpenGL has to apply varied kinds of transformations - "linear" (e.g. rotation, scaling) or not (e.g. translation, perspective) - to geometries, for example in order to perform coordinate system changes or rendering; each of these transformations can be represented as a 4x4 homogeneous matrix, with floating-point (homogeneous) coordinates[24]; a series of transformations can then simply be represented as a single of such matrices, corresponding to the product (of course in a right order) of the involved transformation matrices

[24] So a 3D point is specified based on 4 coordinates: $P = \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$, with `w` being usually equal to `1.0` (otherwise the point can be normalised by dividing each of its coordinates by `w`, provided of course that `w` is not null - otherwise these coordinates do not specify a point but a direction).

- while this will not change *anything* regarding the actual OpenGL library and the computations that it performs, the conventions adopted by the OpenGL *documentation* regarding matrices are the following ones:
  - their in-memory representation is column-major order (even if it is unusual, at least in C; this corresponds to Fortran-like conventions), meaning that it enumerates their coordinates first per column rather than per row (and for them a vector is a *row* of coordinates), whereas tools following the row-major counterpart order, including Myriad, do the opposite (and for them vectors are *columns* of coordinates); more clearly, a matrix like $M = \begin{bmatrix} a11 & a12 & ... & a1n \\ a21 & a22 & ... & a2n \\ ... & ... & ... & ... \\ am1 & am2 & ... & amn \end{bmatrix}$
    - will be stored with row-major conventions (e.g. Myriad) as: `a11, a12, ... a1n, a21, a22, ... a2n, am1, am2, ... amn` (or, more precisely, as `[[a11, a12, ... a1n], [a21, a22, ... a2n], ..., [am1, am2, ... amn]]`)
    - whereas, with the conventions discussed, OpenGL will expect it to be stored in-memory in this order: `a11, a21, ..., am1, a21, a22, ..., am2, ..., a1n, a2n, ..., amn`, i.e. as the transpose of the previous matrix
  - these OpenGL storage conventions do not tell how matrices are to be multiplied (knowing of course that the matrix product is not commutative); if following the aforementioned OpenGL *documentation* conventions, one should consider that OpenGL relies on the usual multiplication order, that is **post-multiplication**, i.e. *multiplication on the right*; this means that, if applying on a given matrix M a transformation O (e.g. rotation, translation, scaling, etc.) represented by a matrix M_O, the resulting matrix will be $M' = M.M_O$ (and not $M' = M_O.M$); a series of operations O_1, then O_2, ..., then O_n will therefore correspond to a matrix $M' = M_{O1}.M_{O2}.[...].M_{On}$; applying a vector $\vec{V}$ to a matrix M will result in $\vec{V'} = M.\vec{V}$
  - so when an OpenGL program performs calls implementing first a rotation (r), then a scaling (s) and finally a translation (t):

    ```
    glRotatef(90, 0, 1, 0);
    glScalef(1.0, 1.1, 1.0);
    glTranslatef(5,10,5);
    ```

  the current matrix M ends up being multiplied (on the right) by $M' = M_r.M_s.M_t$; when applied to a vector $\vec{V}$, still multiplying on the right results in $\vec{V'} = M.\vec{V} = M.M_r.M_s.M_t.\vec{V'}$; so the input vector $\vec{V}$ is first translated, then the result is scaled, then rotated, then transformed by the

Thus summing (like vectors) two 4D points actually returns their mid-point (center of segment), as `w` will be normalised: $P1+P2 = \begin{pmatrix} x1 \\ y1 \\ z1 \\ 1.0 \end{pmatrix} + \begin{pmatrix} x2 \\ y2 \\ z2 \\ 1.0 \end{pmatrix} = \begin{pmatrix} x1+x2 \\ y1+y1 \\ z1+z2 \\ 2.0 \end{pmatrix} = \begin{pmatrix} (x1+x2)/2.0 \\ (y1+y2)/2.0 \\ (z1+z2)/2.0 \\ 1.0 \end{pmatrix}$

previous matrix M; as a result: **operations happen in the opposite order of their specification as calls**; said differently: one shall specify the calls corresponding to one's target series of transformations *backwards*

  - considering that the OpenGL storage is done in a surprising column-major order was actually a trick so that OpenGL could rely on the (modern, math-originating) vector-as-column convention while being still compliant with its GL ancestor - which relied on the (now unusual) vector-as-row convention and on *pre-multiplication* (where we would have $M' = M_O.M$); indeed, knowing that, when transposing matrices, $(A.B)^\top = B^\top.A^\top$, one may consider that OpenGL actually always operates on transpose elements, and thus that: (1) matrices are actually specified in row-order and (2) they are multiplied on the left (e.g. $M' = M_t.M_s.M_r.M$); note that switching convention does not affect at all the computations, and that the same operations are always performed in reverse call order

- OpenGL can operate on three mutually exclusive **modes**:
  - *rendering*: is the default, most common mode, discussed here
  - *feedback*: allows to capture the primitives generated by the vertex processing, i.e. to establish the primitives that would be displayed after the transformation and clipping steps; often used in order to resubmit this data multiple times
  - *selection*: determines which primitives would be drawn into some region of a window (like in *feedback* mode), yet based on stacks of only user-specified "names" (so that the actual data of the corresponding primitives is not returned, just their name identifier); a special case of selection is *picking*, allowing to determine what are the primitives rendered at a given point of the viewport (typically the onscreen position of the mouse cursor, to enable corresponding interactions)

Steps for OpenGL Rendering The usual analogy to describe them is the process of **producing a photography**:

1. a set of elements (3D objects) can be placed (in terms of position and orientation) as wanted in order to compose one's scene of interest (**modelling transformations**, based on *world* coordinates)
2. the photographer may similarly place as wanted at least one camera (**viewing transformations**, based on *camera* coordinates)
3. the settings of the camera can be adjusted, for example regarding its lens / zoom factor (**projection transformations**, based on *window* coordinates)
4. the snapshots that it takes can be further adapted before being printed, for example in terms of scaling (**viewport transformations**, based on *screen* coordinates)

One can see that the first two steps are reciprocal; for example, moving all objects in a direction or moving the camera in the opposite direction is basically the same operation. These two operations, being the two sides of the same coin, can thus be managed by a single matrix, the *model-view* one.

Finally, as mentioned in the section about storage conventions, in OpenGL, operations are to be defined in reverse order. If naming M_s the matrix implementing a given step S, the previous process would be implemented by an overall matrix, based on the previous bullet numbers: $M = M_4.M_3.M_2.M_1$, so that applying a vector $\vec{V}$ to M results in $\vec{V'} = M.\vec{V} = M_4.M_3.M_2.M_1.\vec{V} = M_4.(M_3.(M_2.(M_1.\vec{V})))$.

Transformations In this context - except notably the projections - most transformations are invertible, and a composition of invertible transformations, in any combination and sequence, is itself invertible.

As mentioned, they can all be expressed as 4x4 homogeneous matrices, and their composition translates into the (orderly) product of their matrices.

Coordinate system transitions are discussed further in this document, in the 3D coordinate systems section.

Translations / Rotations / Scalings / Shearings

- the inverse of a **translation** of a vector $\vec{T}$ is a translation of vector $-\vec{T}$, thus: $(Mt_{\vec{T})^{-1}=Mt_{-\vec{T}}}$
- the inverse of a **rotation** of an angle θ along a vector $\vec{U}$ is a rotation of an angle $-\theta$ along the same vector, thus: $(Mr_(\vec{u},\theta))^{-1} = Mr_(\vec{u},-\theta)$
- the inverse of a (uniform) **scaling** of a (non-null) factor f is a scaling of factor $1/f$, thus: $(Ms_f)^{-1} = Ms_{1/f}$; the same applies for each factor when performing a shear mapping

Reflections Symmetries with respect to an axis correspond to a scaling factor of -1 along this axis, and 1 along the other axes.

Affine Transformations An affine transformation designates all geometric transformations that preserve lines and parallelism (but not necessarily distances and angles).

They are compositions of a linear transformation and a translation of their argument.

For them $f(\lambda.x + y) = \lambda.f(x) + f(y)$.

Projections In OpenGL, the projection matrix (`GL_PROJECTION`) transforms eye coordinates to clip coordinates (not viewport coordinates).

A projection defines 6 clipping planes (and at least 6 additional ones can be defined).

A 3D plane is defined by including a given (3D) point and comprising all vectors orthogonal to a given vector; it can be defined thanks to 4 coordinates

(e.g. `(a, b, c, d)`); and a given point $P = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$ will belong to such a plane iff $a.x + b.y + c.z + d = 0$.

Two kinds of projections are considered below: orthographic and perspective; for extra information, refer to this OpenGL Projection Matrix page.

Orthographic Projections

Their viewing volume is a parallelepiped, precisely a rectangular cuboid.

With such projections, parallel lines remain parallel; see `gl:ortho/6` and `glu:ortho2D/4`.

Perspective Projections

Their viewing volume is a truncated pyramid.

They are defined based on a field of view and an aspect ratio; see `gl:frustum/6` and `glu:perspective/4`.

Viewport Transformations As for the viewport, it is generally defined with `gl:viewport/4` so that its size corresponds to the widget in which the rendering is to take place.

To avoid distortion, its aspect ratio must be the same as the one of the projection transformation.

Camera The default model-view matrix is an identity; the camera (or eye) is located at the origin, points down the negative Z-axis, and has an up-vector of `(0, 1, 0)`.

With Z-up conventions (like in Myriad ones), this corresponds to a camera pointing downward (see Coordinate Systems In 3D to picture it).

Calling `glu:lookAt/9` allows to set arbitrarily one's camera position and orientation.

In order to switch from (OpenGL) Y-up conventions to Z-up ones, another option is to rotate the initial (identity) model-view matrix along the X axis of an angle of $-\pi/2$, or to (post-)multiply the model-view matrix with:

$$M_{camera} = P_{zup \to yup} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

For example, if we want that this camera sees, in the (Z-up) MyriadGUI coordinate system, a point `P` at coordinates $P_{zup} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$ (thus a point in its Y axis), then the coordinates of the same point P this time in the base OpenGL (Y-up) coordinate system must be $P_{yup} = P_{zup \to yup}.P_{zup} = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}$; refer to the Computing Transition Matrices section for more information.

OpenGL Hints

- a frequent pattern is, for some type of OpenGL element (let's name it `Foo`; it could designate for example `Texture`, `Buffer` or `VertexArray`) is to call first (here in C) `glGenObjects(1, &fooId);` (note the plural), then `glBindObject(GL_SOME_TARGET, fooId);`
  - it must be understood that `glGenObjects` is the actual creator of (at least) one new (blank) instance of `Foo`, whose address is kept by OpenGL behind the scenes; the user program will be able to access this instance only once bound thanks to an additional level of indirection, its (GL) identifier (`fooId` here); "integer pointers/identifiers" are thus used
  - as for `glBindObject`, its role is to register the `Foo` pointer corresponding to the specified identifier `fooId` in the C-like struct that corresponds to the current context (i.e. the current state of OpenGL), in the field designated here by `GL_SOME_TARGET`, like in: `current_gl_context->gl_some_target = foo_pointer_for(fooId);` this operation is thus mostly a (quick) assignment
  - once bound, this `Foo` instance can be accessed implicitly (through the current context) by calls such as `glSetFooOption(GL_SOME_TARGET, GL_OPTION_FOO_WIDTH, 800);` (where neither its identifier nor any pointer for it is specified); once done, this instance can be unbound with `glBindObject(GL_SOME_TARGET, 0);`; rebinding that identifier later will restore the corresponding options; as a result, several instances can be created, corresponding to as many sets of predefined options, and when a given one shall apply, it just has to be bound
- in OpenGL:
  - 3D coordinates are processed iff they are *Normalized Device Coordinates* (see NDC), for all 3 dimensions
  - the **alpha** color coordinate encodes **opacity** (as usual); thus 1.0 means fully opaque, whereas 0.0 means fully transparent

Mini OpenGL Glossary Terms that are more or less specific to OpenGL:

- `Accumulation buffer`: a buffer that may be used for scene antialiasing; the scene is rendered several times, each time jittered less than one pixel, and the images are accumulated and then averaged
- `Alpha Test`: to reject fragments based on their alpha coordinate; useful to reduce the number of fragments rendered through transparent surfaces
- `Context`: a rendering context corresponds to the OpenGL state and the connection between OpenGL and the system; in order to perform a rendering, a suitable context must be current (i.e. bound, active for the OpenGL commands); it is possible to have multiple rendering contexts share buffer data and textures, which is specially useful when the application uses multiple threads for updating data into the memory of the graphics card

- `DDS`: a file format suitable for texture compression that can be directly read by the GPU
- `Display list`: a series of OpenGL commands, identified by an integer, to be stored, server-side, for subsequent execution; it is defined so that it can be sent and processed more efficiently, and probably multiple times, by the graphic card (compared to doing the same in immediate mode)
- `EBO`: a (GLSL) *Element Buffer Object*, a buffer storing the index of each vertex that OpenGL shall draw (rather than the vertex itself), relatively to a corresponding VBO; defining faces based on indices rather than on vertices allows to avoid vertex duplication (as by design a vertex is common to multiple faces; it should be best specified only once, and referenced as many times as needed); more information
- (pixel) `fragment`: two-dimensional description of elements (point, line segment, or polygon) produced by the rasterization step, before being stored as pixels in the frame buffer; also defined as: "a point and its associated information"; a fragment translates to a pixel after a process involving in turn: texture mapping, fog effect, antialiasing, tests (scissor, alpha, stencil, depth), blending, dithering, and logical operations on fragments (and, or, xor, not, etc.)
- `Evaluator`: the part of the pipeline to perform polynomial mapping (basis functions) and transform higher-level primitives (such as NURBS) into actual ones (vertices, normals, texture coordinates and colors)
- `Frame buffer`: the "server-side" pixel buffer, filled, after rasterization took place, by combinations (notably blending) of the selected fragments; it is actually made of a set of logical buffers of bitplanes: the color (itself comprising multiple buffers), depth (for hidden-surface removal), accumulation, and stencil buffers
- `GL`: *Graphics Library* (also a shorthand for *OpenGL*, which is an open implementation thereof)
- `GLU`: *OpenGL Utility Library*, a standard part of every OpenGL implementation, providing auxiliary features (e.g. image scaling, automatic mipmapping, setting up matrices for specific viewing orientations and projections, performing polygon tessellation, rendering surfaces, supporting quadrics routines that create spheres, cylinders, cones, etc.); see this page for more information
- `GLUT`, *OpenGL Utility Toolkit*, a system-independent window toolkit hiding the complexities of differing window system APIs and more complicated three-dimensional objects such as a sphere, a torus, and a teapot; its main interest was when learning OpenGL, it is less used nowadays
- `GLX`: the X extension of the OpenGL interface, i.e. a solution to integrate OpenGL to X servers; see this page for more information
- `GLSL`: OpenGL Shading Language, a C-like language with which the transformation and fragment shading stages of the pipeline can be programmed; introduced in OpenGL 2.0; see our GLSL section

- `NDC`: *Normalized Device Coordinate*; such a coordinate is, in OpenGL, in $[-1.0, 1.0]$, defining a cube (see this example, which does not represents the Z axis); only the points ultimately within this cube will be rendered, by being transformed to screen-space (viewport) coordinates and then fragments sent to the fragment shader; the conventions for texture coordinates (texels) are different

- `OpenCL`: Open Computing Language, a framework for writing programs that execute across heterogeneous platforms: central processing units (CPUs), graphics processing units (GPUs), digital signal processors (DSPs), field-programmable gate arrays (FPGAs) and other processors or hardware accelerators; in practice OpenCL defines programming languages, deriving from C and C++, for these devices, and APIs to control the platform and execute programs on the compute devices; OpenGL defines a standard interface for parallel computing using task-based and data-based parallelism; see also our Erlang-related section

- `OpenGL ES`: OpenGL for Embedded Systems is a subset of the OpenGL API, designed for embedded systems (like smartphones, tablet computers, video game consoles and PDAs)

- `Pixel`: *Picture Element*

- `Primitive`: points, lines, polygons, images and bitmaps

- (geometric) `Primitives`: they are (exactly) points, lines and polygons

- `Rasterization`: the process by which a primitive is converted to a two-dimensional image

- `Scissor Test`: an arbitrary screen-aligned rectangle outside of which fragments will be discarded; useful to clear or update only a part of the viewport

- `Shader`: a user-defined program providing the code for some programmable stages of the rendering pipeline; they can also be used in a slightly more limited form for general, on-GPU computation (source)

- `Stencil Test`: conditionally discards a fragment based on the outcome of a selected comparison between the value in the stencil buffer and a reference value; useful to perform non-rectangular clipping

- `Texel`: *Texture Element* ; it corresponds to a `(s,t)` pair of coordinates in `[0,1]` designating a point in a texture (see this example; NDCs span different ranges)

- `Vertex Array`: these in-memory client-side arrays may aggregate 6 types of data (vertex coordinates, RGBA colors, color indices, surface normals, texture coordinates, polygon edge flags), possibly interleaved; such arrays allow to reduce the number of calls to OpenGL functions, and also to share elements (e.g. vertices pertaining to multiple faces should preferably be defined only once); in a non-networked setting, the GPU just dereferences the corresponding pointers

- `Viewport`: the (rectangular) part (defined based on its lower left corner and its width and height, in pixels) within the current window in which OpenGL is to perform its rendering; so multiple viewports may be used in turn in order to offer multiple, composite views of the scene of interest in a given window; the ultimately processed 2D coordinates in OpenGL are both in `[-1.0, 1.0]` before they are finally mapped to the current viewport dimensions (e.g. abscissa in `[0,800]`, ordinate in `[0,600]`, in pixels)

- `Vulkan`: a low-overhead, cross-platform API, open standard for 3D graphics and computing; it is intended to offer higher performance and more balanced CPU and GPU usage than the OpenGL or Direct3D 11 APIs; it is lower-level than OpenGL, and not backwards compatible with it (source)

- `VAO`: a (GLSL) *Vertex Array Object* (OpenGL 4.x), able to store multiple VBOs (up to one for vertices, the others for per-vertex attributes); a VAO corresponds to an homogeneous chunk of data, sent from the CPU-space in order to be stored in the GPU-space; more information

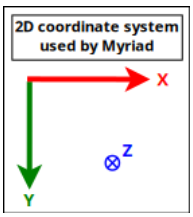
- `VBO`: a (GLSL) *Vertex Buffer Object*, a buffer storing a piece of information (vertex coordinates, or normal, or colors, or texture coordinates, etc.) for each element of a series of vertices; more information

Refer to the description of the pipeline for further details.

Coordinate Systems

Coordinate Systems In 2D A popular convention, for example detailed in this section of the (OpenGL) Red book, is to consider that the ordinates increase when going from the *bottom of the viewport to its top*; then for example the on-screen lower-left corner of the OpenGL canvas is `(0,0)`, and its upper-right corner is `(Width,Height)`.

As for us, we prefer the MyriadGUI 2D conventions, in which ordinates increase when going from the *top of the viewport to its bottom*, as depicted in the following figure:



Such a setting can be obtained thanks to (with Erlang conventions):

```
gl:matrixMode(?GL_PROJECTION),
gl:loadIdentity(),

% Like glu:ortho2D/4:
gl:ortho(_Left=0.0, _Right=float(CanvasWidth),
  _Bottom=float(CanvasHeight), _Top=0.0, _Near=-1.0, _Far=1.0)
```

In this case, the viewport can be addressed like a **usual (2D) framebuffer** (like provided by any classical 2D backend such as SDL) obeying the coordinate system just described: if the width of the OpenGL canvas is 800 pixels and its height is 600 pixels, then its top-left on-screen corner is `(0,0)` and its bottom-right one is `(799,599)`, and any pixel-level operation can be directly performed there "as usual". One may refer, in Myriad, to `gui_opengl_2D_test.erl` for a full example thereof, in which line-based letters are drawn to demonstrate these conventions.

Each time the OpenGL canvas is resized, this projection matrix has to be updated, with the same procedure, yet based on the new dimensions.

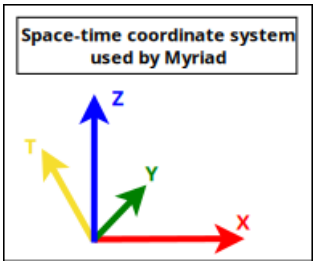
Another option - still with axes respecting the Myriad 2D conventions - is to operate this time based on **normalised, definition-independent coordinates** (see NDC), ranging in `[0.0, 1.0]`, like in:

```
gl:matrixMode(?GL_PROJECTION),
gl:loadIdentity(),

gl:ortho(_Left=0.0, _Right=1.0, _Bottom=1.0, _Top=0.0, _Near=-1.0, _Far=1.0)
```

Using "stable", device-independent floats - instead of integers directly accounting for pixels - is generally more convenient. For example a resizing of the viewport will then not require an update of the projection matrix. One may refer to `gui_opengl_minimal_test.erl` for a full example thereof.

Coordinate Systems In 3D We will rely here as well on the Myriad conventions, this time for 3D (not taking specifically time into account in this section, which anyway cannot be shown properly there):



These are thus Z-up conventions (the Z axis being vertical and designating altitudes), like modelling software such as Blender.

Note that perhaps the most popular convention is different, it is Y-up, for which X is horizontal, Y is up and Z is depth (hence Z-buffer) - this axis pointing then to the user.

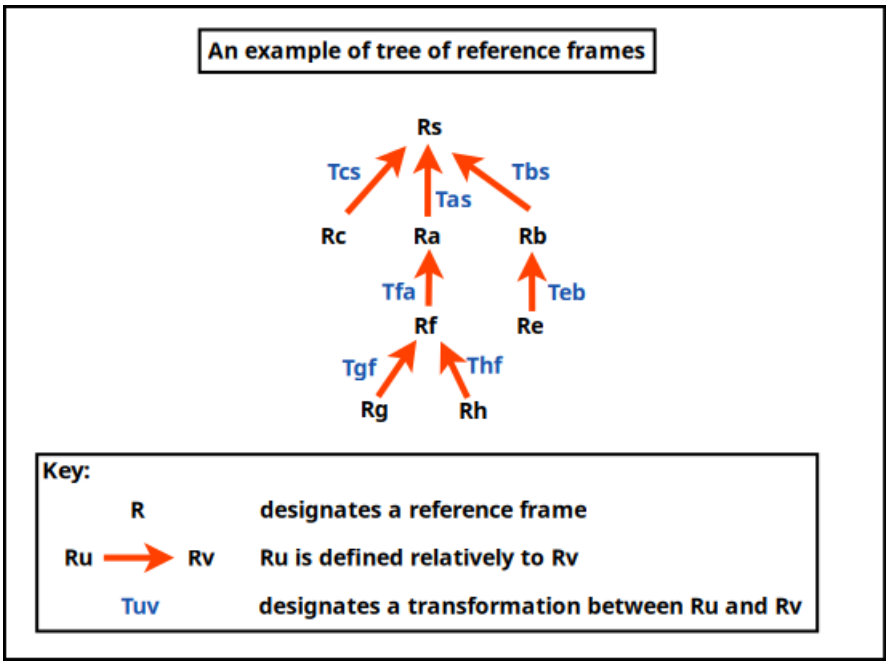
A Tree of Coordinate Systems In the general case, either in 2D or (more often of interest here) in 3D, a given scene (a model) is made of a set of elements (e.g. the model of a street may comprise a car, two bikes, a few people) that will have to be rendered from a given viewpoint (e.g. a window on the second floor of a given building) onto the (flat) user screen (with suitable clipping, perspective division and projection on the viewport). Let's start from the intended result and unwind the process.

The rendering objective requires to have ultimately one's scene transformed as a whole in **eye coordinates** (to obtain coordinates along the aforementioned 2D screen coordinate system, along the X and Y axes - the Z one serving to sort out depth, as per our 2D conventions).

For that, a prerequisite is to have the target scene correctly composed, with all its elements defined in the same, **scene-global**, space, in their respective position and orientation (then only the viewpoint, i.e. the virtual camera, can take into account the scene as a whole, to transform it ultimately to eye coordinates).

As each individual type of model (e.g. a bike model) is natively defined in an abstract, local coordinate system (an orthonormal basis) of its own, each actual model instance (e.g. the first bike, the second bike) has to be specifically placed in the coordinate system of the overall scene. This placement is either directly defined in that target space (e.g. bike A is at this absolute position and orientation in the scene global coordinate system) or relatively to a *series* of parent coordinate systems (e.g. this character rides bike B - and thus is defined relatively to it, knowing that for example the bike is placed relatively to the car, and that the car itself is placed relatively to the scene).

So in the general case, coordinate systems are nested (recursively defined relatively to their parent) and form a tree[25] whose root corresponds to the (possibly absolute) coordinate system of the overall scene, like in (`R` standing here for *reference frame*, a concept that we deem a bit more general than *coordinate system*):



[25]This is actually named a *scene graph* rather than a *scene tree*, as if we consider the leaves of that "tree" to contain actual geometries (e.g. of an abstract bike), as soon as a given geometry is instantiated more than once (e.g. if having 2 of such bikes in the scene), this geometry will have multiple parents and thus the corresponding scene will be a graph.

As for us, we consider *reference frame trees* (no geometry involved) - a given 3D object

A series of model transformations has thus to be operated in order to express all models in the scene reference frame:

```
(local reference frame of model Rh) -> (parent reference frame Rf) -> (parent referenc
```

For example the hand of a character may be defined in R_h, itself defined relatively to its associated forearm in R_f up to the overall reference frame R_a of that character, defined relatively to the reference frame of the whole scene, R_s. This reference frame may have no explicit parent defined, meaning implicitly that it is defined in the canonical, global, "world" reference frame.

Once the **model** is expressed as a whole in the scene-global reference frame, the next transformations have to be conducted : **view** and projection. The view transformation involves at least an extra reference frame, the one of the camera in charge of the rendering, which is R_c, possibly defined relatively to R_s (or any other reference frame).

So a geometry (e.g. a part of the hand, defined in R_f) has to be transformed upward in the reference frame tree in order to be expressed in the common, "global" scene reference frame R_s, before being transformed last in the camera one, R_c .

In practice, all these operations can be done thanks to the multiplication of homogeneous 4x4 matrices, each able to express any combination of rotations, scalings/reflections/shearings, translations[26], which thus include the transformation of one reference frame into another. Their product can be computed once, and then applying a vector (e.g. corresponding to a vertex) to the resulting matrix allows to perform in one go the full composition thereof, encoding all model-view transformations (and even the projection) as well.

Noting $P_{a \to b}$ the transition matrix transforming a vector $\vec{V_a}$ expressed in R_a into its representation $\vec{V_b}$ in R_b, we have:

$$\vec{V_b} = P_{a \to b}.\vec{V_a}$$

Thus, to express the geometry of said hand (natively defined in R_h) in camera space (hence in R_c), the following composition of reference frame changes[27] shall be applied:

$$P_{h \to c} = P_{s \to c}.P_{a \to s}.P_{f \to a}.P_{h \to f}.$$

So a whole series of transformations can be done by applying a single matrix - whose coordinates are determined now.

being possibly associated to (1) a reference frame and (2) a geometry (independently). This is as expressive, and most probably clearer.

[26] In practice the recommended order of the operations are: scaling, then rotation, then translation, otherwise they will become coupled and would interfere negatively (e.g. a translation vector would be scaled as well).

[27] Thus transformation matrices, knowing that the product of such matrices is in turn a transformation matrix.

Computing Transition Matrices For that, let's consider that an homogeneous 4x4 matrix is in the form of:

$$M = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

It can be interpreted as a matrix comprising two blocks of interest, R and $\vec{T}$:

$$M = P_{1\to 2} = \begin{bmatrix} R & \vec{T} \\ 0 & 1 \end{bmatrix}$$

with:

- R, which accounts for a 3D rotation submatrix:

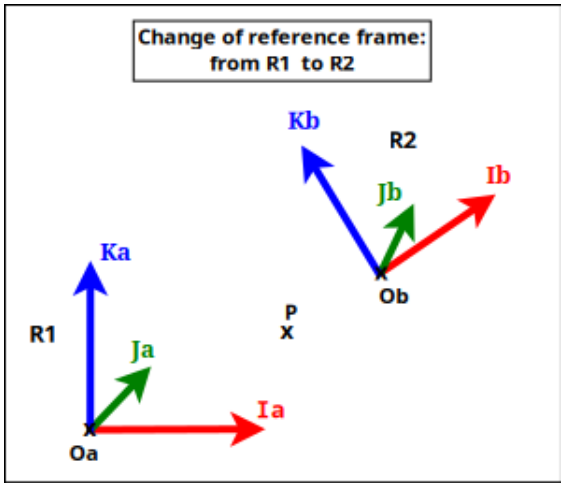
$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

- $\vec{T}$, which accounts for a 3D translation vector:

$$\vec{T} = \begin{bmatrix} t1 \\ t2 \\ t3 \end{bmatrix}$$

Applying a (4x4 homogeneous) point $P = \begin{Bmatrix} x \\ y \\ z \\ 1 \end{Bmatrix}$ to M yields $P' = M.P$ where P' corresponds to P once it has been (1) rotated by R and then (2) translated by $\vec{T}$ (the order matters).

Let's consider now:



- two coordinate systems (defined as orthonormal bases), R_1 and R_2; R_2 may for example be defined relatively to R_1; for a given point or vector U, U_1 will designate its coordinates in R_1, and U_2 its coordinates in R_2
- $P_{2\to1}$ the (homogeneous 4x4) transition matrix from R_2 to R_1, specified first by blocks then by coordinates as:

$$P_{2\to1} = \begin{bmatrix} R & \vec{T} \\ 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- any (4D) point P, whose coordinates are P_1 in R_1, and P_2 in R_2

The objective is to determine $P_{2\to1}$, i.e. R and $\vec{T}$.

By definition of a transition matrix, for any point P, we have: $P_1 = P_{2\to1}.P_2$ (1)

Let's study $P_{2\to1}$ by first choosing a point P equal to the origin of R_2 (shown as `Ob` in the figure).

By design, in homogeneous coordinates, $P_2 = Ob_2 = \begin{Bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{Bmatrix}$, and applying it on (1) gives us: $P_1 = Ob_1 = \begin{Bmatrix} t1 \\ t2 \\ t3 \\ 1 \end{Bmatrix}$.

So if $Ob_1 = \begin{Bmatrix} XOb_1 \\ YOb_1 \\ ZOb_1 \\ 1 \end{Bmatrix}$, we have: $\vec{T} = \vec{T_{2\to1}} = \begin{bmatrix} XOb_1 \\ YOb_1 \\ ZOb_1 \end{bmatrix}$.

Let's now determine the r_{xy} coordinates.

Let $R_{2\to1}$ be the (3x3) rotation matrix transforming any vector expressed in R_2 in its representation in R_1: for any (3D) vector $\vec{V}$, we have $\vec{V_1} = R_{2\to1}.\vec{V_2}$ (2)

(we are dealing with vectors, not points, hence the origins are not involved here).

By choosing $\vec{V}$ equal to the $\vec{Ib}$ (abscissa) axis of R_2 (shown as `Ib` in the figure), we have $\vec{Ib_1} = R_{2\to1}.\vec{Ib_2}$

Knowing that by design $\vec{Ib_2} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$, (2) gives us:

$$\vec{Ib_1} = \begin{bmatrix} r_{11} \\ r_{21} \\ r_{31} \end{bmatrix} = \begin{bmatrix} XIb_1 \\ YIb_1 \\ ZIb_1 \end{bmatrix}$$

So the first column of the R matrix is $\vec{Ib_1}$, i.e. the first axis of R_2 as expressed in R_1.

Using in the same way the two other axes of R_2 (shown as `Jb` and `Kb` in the figure), we see that:

$$R = R_{2\to1} = \begin{bmatrix} XIb_1 & XJb_1 & XKb_1 \\ YIb_1 & YJb_1 & YKb_1 \\ ZIb_1 & ZJb_1 & ZKb_1 \end{bmatrix}$$

So the transition matrix from R_2 to R_1 is:

$$P_{2\to1} = \begin{bmatrix} R_{2\to1} & \vec{T_{2\to1}} \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} XIb_1 & XJb_1 & XKb_1 & XOb_1 \\ YIb_1 & YJb_1 & YKb_1 & YOb_1 \\ ZIb_1 & ZJb_1 & ZKb_1 & ZOb_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where:

- $R_{2\to1}$ is the 3x3 rotation matrix converting vectors of R_2 in R_1, i.e. whose columns are the axes of R_2 expressed in R_1
- $\vec{T_{2\to1}} = Ob_1$ is the 3D vector of the coordinates of the origin of R_2 as expressed in R_1

This also corresponds to a matrix obtained by describing the R_2 coordinate system in R_1, by listing first the three (4D) vector axes of R_2 then its (4D) origin, i.e. $P_{2\to1} = \left[\vec{Ib_1} \quad \vec{Jb_1} \quad \vec{Kb_1} \quad Ob_1\right]$.

Often, transformations have to be used both ways, like in the case of a scene-to-camera transformation; as a consequence, transition matrices may have to be inversed, knowing that $(P_{2\to1})^{-1} = P_{1\to2}$ (since by definition $P_{2\to1}.P_{1\to2} = Id$).

An option to determine $P_{1\to2}$ from $P_{2\to1}$ could be to compute its inverse directly, as $P_{1\to2} = (P_{2\to1})^{-1}$, yet $P_{1\to2}$ may be determined in a simpler manner.

Indeed, for a given point P, whose representation is P_1 in R_1 and P_2 in R_2, we obtain $P_1 = P_{2\to1}.P_2$ by - through the way (4x4) matrices are multiplied - first applying a (3x3) rotation Rot_3 to P_2 and then a (3D) translation T_r: $P_1 = Rot_3.P_2 + T_r$ (in 3D; thus leaving out any fourth homogeneous coordinate); therefore $P_2 = (Rot_3)^{-1}.(P_1 - T_r)$. Knowing that the inverse of an orthogonal matrix is its transpose, and that rotation matrices are orthogonal, $(Rot_3)^{-1} = (Rot_3)^\top$, and thus $P_2 = (Rot_3)^\top.(P_1 - T_r) = (Rot_3)^\top.P_1 - (Rot_3)^\top.T_r$.

So if:

$$P_{2\to1} = \begin{bmatrix} R_{2\to1} & \vec{T_{2\to1}} \\ 0 & 1 \end{bmatrix}$$

then:

$$P_{1\to2} = \begin{bmatrix} (R_{2\to1})^\top & -(R_{2\to1})^\top.\vec{T_{2\to1}} \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} XIb_1 & YIb_1 & ZIb_1 & -(XIb_1.XOb_1 + YIb_1.YOb_1 + ZIb_1.Z \\ XJb_1 & YJb_1 & ZJb_1 & -(XJb_1.XOb_1 + YJb_1.YOb_1 + ZJb_1.Z \\ XKb_1 & YKb_1 & ZKb_1 & -(XKb_1.XOb_1 + YKb_1.YOb_1 + ZKb_1.Z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Note

Therefore, in a nutshell, the transition matrix from a coordinate system R_α to a coordinate system R_β is:

$$P_{\alpha\to\beta} = \begin{bmatrix} Rot_{\alpha\to\beta} & \vec{Tr_{\alpha\to\beta}} \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} XIb_\beta & XJb_\beta & XKb_\beta & XOb_\beta \\ YIb_\beta & YJb_\beta & YKb_\beta & YOb_\beta \\ ZIb_\beta & ZJb_\beta & ZKb_\beta & ZOb_\beta \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where:

- $Rot_{\alpha\to\beta}$ is the 3x3 rotation matrix converting vectors of R_α in R_β, i.e. whose columns are the axes of R_α expressed in R_β
- $\vec{Tr_{\alpha\to\beta}} = Ob_\beta$ is the 3D vector of the coordinates of the origin of R_α as expressed in R_β

This also corresponds to a matrix obtained by describing the R_α coordinate system in R_β, by listing first the three (4D) vector axes of R_α then its (4D) origin, i.e. $P_{\alpha\to\beta} = \begin{bmatrix} \vec{Ib_\beta} & \vec{Jb_\beta} & \vec{Kb_\beta} & Ob_\beta \end{bmatrix}$.

Its reciprocal (inverse transformation) is then:

$$P_{\beta\to\alpha} = \begin{bmatrix} (Rot_{\alpha\to\beta})^\top & -(Rot_{\alpha\to\beta})^\top . \vec{Tr_{\alpha\to\beta}} \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} XIb_\beta & YIb_\beta & ZIb_\beta & -(XIb_\beta.XOb_\beta + YIb_\beta.YO \\ XJb_\beta & YJb_\beta & ZJb_\beta & -(XJb_\beta.XOb_\beta + YJb_\beta.YO \\ XKb_\beta & YKb_\beta & ZKb_\beta & -(XKb_\beta.XOb_\beta + YKb_\beta.YO \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

As a result, from the definition of a tree of coordinate systems, we are able to compute the transition matrix transforming the representation of a vector expressed in any of them to its representation in any of the other coordinate systems.

A special case of interest is, for the sake of rendering, to transform, through that tree, a local coordinate system in which a geometry is defined into the one of the camera, defining where it is positioned and aimed[28]; in OpenGL parlance, this corresponds to the *model-view* matrix (for "modelling and viewing transformations") that we designate here as M_{mv} and which corresponds to $P_{local\to camera}$.

Taking into account the last rendering step, the *projection* (comprising clipping, projection division and viewport transformation), which can be implemented as well thanks to a 4x4 (non-invertible) matrix designated here as M_p, we see that a single combined overall matrix $M_o = M_p.M_{mv}$ is sufficient[29] to convey in one go all the transformations that shall be applied to a given geometry for its rendering.

[28] `gluLookAt` can define such a viewing transformation matrix, when given (1) the position of the camera, (2) a point at which it shall look, and (3) a vector specifying its up direction (i.e. where is the upward direction for the camera - as otherwise all directions orthogonal to its line of sight defined by (1) and (2) could be chosen).

[29] In practice, for more flexibility, in older (pre-shader) OpenGL the management of the viewport, of the projection and of the model-view transformations was done separately (for example, respectively, with: `glViewport`, `glMatrixMode(GL_MODELVIEW)` and `glMatrixMode(GL_PROJECTION)`; so, in compatibility mode, there is a matrix stack corresponding to `GL_MODELVIEW` and another one to `GL_PROJECTION`.

Main Matrices

These matrices account for the main processing steps of a rendering.

Three types of coordinate systems can be considered:

- **world** coordinate system: the absolute, overall coordinate system where 3D scenes are to be assembled
- **local** coordinate system: the coordinate system in which a given model is defined (generally placed at its origin)
- **camera** coordinate system: a coordinate system where a camera is at the origin, looking down on the negative Z axis

The **clip space** can also be considered; this is the post-projection space, where the view frustum is transformed into a cube, centered in the origin, and going from -1 to 1 in every axis.

The transformations between coordinate systems can be represented by 4×4 transition matrices:

- **model matrix** (M_M): to transform from local to world coordinate system
- **view matrix** (M_V): to transform from world to camera coordinate system
- **projection matrix** (M_P): to transform from camera coordinate system to clip space

Finally, two composite matrices are especially useful (note the aforementioned reverse multiplication order) and are typically passed through uniform variables in shaders:

- ModelView: $M_{MV} = M_V.M_M$
- ModelViewProj: $M_{MVP} = M_P.M_{MV} = M_P.M_V.M_M$

Shaders

They are covered in-depth in the Khronos wiki.

A Programmable Pipeline Shaders are the basic rendering building blocks of applications using modern OpenGL (e.g. 3.x/4.0 versions).

Such an application will indeed program its own shaders, instead of calling functions like `glBegin()/glEnd()` as it was done with OpenGL 1.x-2.x and its fixed-pipeline immediate mode.

This mode of operation, albeit more complex, offers more control and enables increased performances.

Parallelism in the Pipeline The key is to write programs that can be executed in a *Single Instruction, Multiple Data* (SIMD) setting, in order to take advantage of the vectorization typically supported by GPUs.

A goal is to avoid conditional branching based on values that may differ from a shader invocation to another (see this explanation).

If having to take into account two dynamically-uniform (i.e. non-statically predictable, yet having the same value for every shader invocation within that group) branches performing simple computations, it is likely that the compiler will generate code evaluating both expressions, until dropping the result of the one finally not happening.

Six Types of GLSL Shaders Shaders are written in the **GLSL** language, i.e. the OpenGL Shading Language.

They are portions of C-like code that can be inserted in the rendering pipeline implemented by the OpenGL driver of a GPU card. Six different kinds of shaders can be defined, depending on the processing step that they implement and on their purpose: vertex, tessellation for control or for evaluation, geometry, fragment or compute shaders.

Except this last type (compute shader), all types are mostly dedicated to *rendering*. If wanting to perform on one's GPU more general-purpose processing, OpenCL shall be preferred to GLSL.

Each shader is to receive data to process that is appropriate to its type; for example each vertex shader instance will receive a vertex (multiple of such instances will process each their own vertex in parallel) whereas each fragment shader will operate on data specific to a pixel.

So shader instances will vary in terms of role (e.g. in charge of the processing of a vertex or a fragment), data types (input and output ones) and multiplicities (number of instances). Indeed, if considering a triangle whose vertices are each pure green, red or blue, only 3 vertices will be processed by the vertex shaders, whereas all the numerous pixels of the triangle will be the result of the evaluation of as many fragment shaders, each input of which is computed by interpolating the attributes of said 3 vertices - which ultimately results in a smooth gradient over the whole triangle.

Runtime Build Shaders are compiled at (application) runtime[30] (to target exactly the actual hardware), then linked and attached to a separate program running on the GPU. This is fairly low-level, black-box direct programming, in sharp contrast with the reliance on APIs that used to be the norm with OpenGL 1.x.

Yet offline compilers exist as well, as well as debuggers (like the NVIDIA NsightShader Debugger).

Implementing a Shader A shader is quite similar to a C program, yet based on a specific, core language that enables the definition of relevant data types and functions.

Data types are usually based on elementary types (`float`, `double`, `bool`, `int` and `uint`), and composed in larger structures, like `{vec,mat}{2,3,4}`, `mat2x3`, arrays and structures, possibly `const`; see this page for further details.

Similarly, control flow statements and (non-recursive) functions can be defined; every shader must have a `main` function, and can define other auxiliary functions as well, in a way similar to a C program. Function parameters may

[30] So each shader is built each time the application is started, and the operation may fail (e.g. with `0(40) : error C1503: undefined variable "foobar"`).

have the `in` (which is the default), `out` or `inout` qualifiers specified. Additionally a function may return a result, thanks to `return`.

So, regarding output, for example a fragment shader must return the color that it computed: `out vec3 my_color;` declares this; and the shader code may be as simple as returning a constant color in all cases, like in:

```
#version 330 core
out vec3 my_color;

void main()
{
   // Same color returned for all fragments:
   my_color = vec3(0.05, 0.2, 0.67);
}
```

Communicating with Shaders Of course the application must have a way of supplying information to its shaders (the other way round does not really happen, except for compute shaders), and a given shader must be able to pass information to (only) any next shader in the pipeline.

Two options exist for shaders to have inputs and outputs, from/to the CPU and/or other shaders:

- basically each shader is fed with a stream of vertices[31] with associated data, named **vertex attributes**; these attributes are either *user-defined* or *built-in* (each type of shader having its own set of built-in input attributes)
- global, read-only data can also be defined, as **uniform** variables

These communication options are discussed more in-depth next.

Vertex Attributes Defining Attributes

A vertex attribute, whether user-defined or built-in, may store any kind of data - notably positions, texture coordinates and normals.

Either a given attribute is a single, *standalone* one (then a unique value will be read and will apply to all vertices), or is *per-vertex*, in which case it is read from a buffer, each element of it being bound accordingly when its associated vertex is processed by the shader. Such arrays are either used in-order, or according to any indices defined (then themselves defined thanks to an array as well)[32].

Said differently, for each attribute used by a shader, either a single value or an array thereof must be specified.

Referencing Attributes

[31] Then the user-defined *primitives*, applied later in the pipeline, will allow OpenGL to interpret such a series of vertices in terms of a sequence of triangles, or points, or lines, etc.

[32] This is the preferred method, as it prevents vertex duplication, and allows to process each of them once: there is a vertex cache that stores the outputs of the last processed vertices, so that if a vertex is mentioned multiple times (e.g. being included in a triangle fan or strip), the corresponding output may be directly re-used (provided it is still in the cache) instead of having to be computed again.

In order that attributes can be defined in a place (program or shader) and referenced later by at least one shader, they must be matched:

- by (attribute) *name*: then they must bear exactly the same name in the main program and in the shader(s) using them, knowing that any name beginning with `gl_` is reserved
- or by (attribute) *location* (i.e. a positive integer): their common location is specified, and a per-shader variable name is associated - which is more flexible
- or by *block*, like for the uniform variables, discussed below

In the two last cases, the layout of variables must match on either side (for example in the main program and a given shader); for instance, with `"layout(location = 0) in vec3 input_vertex;"` in its code, a vertex shader will expect a (single) vector of 3 (floating-point) coordinates (`vec3`) to be found at index 0 (`location = 0`) as input (`in`); the application will need to specify a corresponding *Vertex Buffer Object* (VBO) for that.

So that they can be fetched for a given vertex, attributes have to be appropriately located in buffers. For each attribute, either the developer defines, prior to linking, a specific location (as an index starting at zero) with `glBindAttribLocation`, or he lets OpenGL choose it, and queries it afterwards, with `glGetAttribLocation`; refer to this page for further details.

If a given program is linked with two shaders, a vertex one and a fragment one, the former one will probably have to pass its outputs as inputs of the latter one; this requires as many variables defined on either side, with relevant out/in specifications, and a matching name and type; for example the vertex shader may declare `out vec3 my_color;` whereas the fragment shader will declare `in vec3 my_color;`.

Providing Attribute Data : VAO and VBO

Vertex data is provided thanks to a (single current) Vertex Array Object (VAO).

A VAO references (rather than storing directly) the format of the vertex data, as well as the buffers (VBOs, see below) holding that data.

A vertex attribute is identified by a number (in `[0;GL_MAX_VERTEX_ATTRIBS-1]`), and by default is accessed as a single value (as opposed to as an array).

A Vertex Buffer Object (VBO) is a data array, typically referenced by a VAO. A VBO defines its internal structure and where the corresponding data can be found.

So, in practice, each homogeneous chunk of data (vertices, normals, colors) to be sent from the CPU-space to the GPU-space (hence from the client side to the server one) is stored in an array corresponding to a *Vertex Buffer Object* (VBO), itself referenced by a *Vertex Array Object* (VAO). A VAO may gather vertex data and colour data in separate VBOs, and store them on the graphics card for any later use (as opposed to streaming vertices through to the graphics card when they become needed). A VAO is only meant to hold one (VBO) array of vertices, each other VBO being used then for per-vertex attributes.

Uniform Variables are Read-Only and Global Instead of relying on attributes, an alternate way of passing information, provided that it may change relatively infrequently, is to use *uniform variables*, which behave, for a shader and in the course of a draw call, as read-only, constant global variables for all vertices (hence their *uniform* naming). Any shader can access every uniform variable (they are global), as long as it declares that variable, and these variables hold as long as they are not reset or updated.

Examples of uniform variables could be the position of a light, transformation matrices, fog settings, variables such as gravity and speed, etc.

Uniform variables may be defined individually, or be grouped in named blocks, for a more effective data setup (to share uniform variables between programs) and transfer from the application to the shader (setting multiple values at once).

Uniform variables are declared at the program-level (as opposed to a per-vertex level) thanks to:

- the `uniform` qualifier on the shader-side, like in `uniform mat4 MyMatrix;`
- a `glGetUniformLocation` call on the application-side, to create a location associated to a variable name (e.g. `my_matrix`) in a shader, and to associate it to a given value, like, in C:

```
mat4 some_matrix = [...];

GLuint location = glGetUniformLocation(programId, "my_matrix");

if (location >= 0)
{
   // Defining a single matrix (1), not to transpose (GL_FALSE):
   glUniformMatrix4fv(location, 1, GL_FALSE, &some_matrix[0][0]);
   [...]
```

Individual variables may be used as uniform, as well as arrays and structs.

From the point of view of a shader, these named input variables may be initialised when declared, but then are read-only; otherwise the application may choose to set them.

Built-in Variables are Defined by Each Shader Type Finally, depending on the type of a shader, some predefined, built-in variables ("intrinsic attributes") may be set; they are specified here.

For example, for a vertex shader, following output variables are predefined:

- `vec4 gl_Position` corresponding to the clip-space (homogeneous) position of the output vertex
- `float gl_PointSize`
- `float gl_ClipDistance[]`

Using Multiple Shaders of the Same Type One may want to use different shaders of the same type (e.g. to have a choice in terms of fragment shaders) in the same scene (e.g. one fragment shader dealing with solid colors, another one with textures).

At any time, up to one shader of a given type may be bound (active), but any number of shader objects (i.e. shaders loaded into memory and compiled) can be defined and available.

One approach is to switch shaders (with `glUseProgram`) between draw calls: set a shader, draw a model, set another shader, draw another model. However switching shaders incurs some overhead, so a better course of action may be to group models/materials according to the shader they are to rely on, and to iterate on these shaders, bound one after the other, each only once per frame.

Other approaches are to render to textures, to rely on a Framebuffer Object with Renderbuffers Objects attached, or to use deferred shading or GLSL subroutines.

A last approach, perhaps the simplest and overall best, is to define a "macro-shader" per shader type (e.g. a macro-fragment shader) that regroups the code of each of the shaders of interest, and that may apply one or multiple of these effects based on conditions (variables); no switching will be needed then.

Refer to this thread for more details.

Troubleshooting Shaders Once a shader builds correctly, it may misbehave at runtime.

One may refer to the "Debugging shader output" section of this page.

Examples of Shaders See the ones of Wings3D (in GLSL "1.2" apparently, presumably for maximum backward compatibility; note that some elements, with the `*.cl` extension, are OpenCL ones), or these ones.

Managing Spatial Transformations Modern OpenGL (and GLU) implementations basically dropped the direct matrix support (the so-called immediate mode does not exist anymore, except in a compatibility context). So no more calls to `glTranslate`, `glRotate`, `glLoadIdentity` or `gluPerspective` shall be done; now the application has to compute such matrices (for model, view, texture, normal, projection, etc.) *by itself* (on the CPU), and the result thereof can be send on the GPU, as input to its GLSL shaders (typically thanks to uniform variables).

For that, applications may use dedicated, separate libraries, such as, in C/C++, GLM, i.e. OpenGL Mathematics[33] (Myriad's linear support aims to provide, in Erlang, a relevant subset of these operations - albeit admittedly in a simplified form).

The matrices that correspond to the transformations to be applied are then typically shared with the shaders thanks to uniform variables.

This is especially the case for the vertex shader, in charge of transforming coordinates expressed in a local coordinate system into screen coordinates.

[33]GLM is a free software header-only, template-based C++ library. See its manual and for example its implementation of 4x4 float-based matrices (and their corresponding type definition). Note that, as GLM is targeted at OpenGL, by default it adopts column-major internal conventions, leading to a somehow unfamiliar mode of operation.

More precisely, the modeling (object-space to absolute, world-space), viewing (world-space to camera-space) and projection (camera-space to clip-space) transformations are applied in the vertex shader, whereas the final perspective division and the viewport transformation are applied in the fixed-function stage after the vertex shader.

So a vertex shader is usually given two 4x4 homogeneous, uniform matrices:

- a modelview matrix, combining modeling and viewing, to transform object-space to camera-space in one go
- a projection matrix

More Advanced Topics

Shadows Determining the shadow of an arbitrary object on an arbitrary plane (representing typically the ground - or other objects) from an arbitrary light source (possibly at infinity) corresponds to performing a specific **projection**. For that, a relevant 4x4 (based on homogeneous coordinates) matrix (singular, i.e. non-invertible matrix) can be defined.

This matrix can be multiplied with the top of the model-view matrix stack, before drawing the object of interest in the shadow color (a shade of black generally).

Refer to this page for more information.

Reference GLSL Compiler As always, different compilers (corresponding to different brands of graphical cards) will not implement exactly the same way the (OpenGL GLSL, here) specification; and a shader may work correctly on one type of card and not on another.

Testing shader code with the OpenGL / OpenGL ES Reference Compiler, a.k.a. `glslang` (installed on Arch Linux thanks to `pacman -Sy glslang`, to be run as `glslangValidator`) may report interesting information.

See the *OpenGL GLSL reference compiler* section of this page for more information.

Sources of Information

The reference pages for the various versions of OpenGL are available on the Khronos official OpenGL Registry.

Two very well-written books, strongly recommended, that are still relevant for 3D graphics despite their old age (circa 1996; for OpenGL 1.1) are:

- *The Official Guide to Learning OpenGL*: the OpenGL Red book
- *The OpenGL Reference Manual*: the OpenGL Blue book

More modern tutorials (applying to OpenGL 3.3 and later) are:

- Opengl-tutorial
- Learn OpenGL
- regarding GLSL shaders: lighthouse3d

Other elements of interest:

- the OpenGL 3.3 Specification (Core Profile) (the 425-page reference PDF)
- FAQ for OpenGL and GLUT
- the (archived) OpenGL FAQ and Troubleshooting Guide, containing much valuable information, including regarding transformations
- About OpenGL Performance
- in French: Introduction à OpenGL et GLUT, by Nicolas Roussel
- any textbook on linear algebra

Operating System Support for 3D

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

The very first step is to **update's one's video drivers to their latest, official stable version** according to your OS/distribution of choice (even if it implies using closed-source binaries...) and to check that they are in use (probably rebooting is then needed; note that updating one's kernel may also make the hardware acceleration be lost until the next reboot).

Testing

First, one may check whether such acceleration is already available by running, from the command-line and as the current, non-privileged user, 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, sacrifice a chicken, reboot, etc.

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

In our case, relevant installations could be done with `pacman -Sy nvidia nvidia-utils` but required 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.

Minor Topics

Camera Navigation Conventions

Multiple tools introduced conventions in order to navigate, with mouse and keyboard, in a 3D world.

We prefer the way Blender manages the observer viewpoint (current camera), as described here; notably, supposing a three-button mouse with a scrollwheel, basic navigation will be **based on the middle button**:

- `orbit the view around the currently selected object` (or `Tumble`) by **holding the middle button down** and moving the mouse
- `pan` (moving the view up, down, left and right) by holding down **Shift and the middle button**, and moving the mouse
- `zoom in/out` with the **scrollwheel**; a variation of it, `Dolly`, can be obtained by holding down **Ctrl and the middle button**, and moving the mouse

3D-Related 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. that may be retained so that the hand is ultimately correctly placed on the chair (more information)
- **Material**: controls the optical properties of an object, i.e. how a 3D object appears on the screen, that is: the color of each point of the object (generally thanks to multiple texture maps, like diffusion, normal, specular, glow, etc.) and how reflective or dull its surface appears; designates, with OpenGL, a set of coefficients that define **how the lighting model interacts with the surface**; in particular, ambient, diffuse, and specular coefficients for each color component (R,G,B) are defined and applied to a surface and effectively multiplied by the amount of light of each kind/color that strikes the surface; a final emissivity coefficient is then added to each color component so that objects can also be light emitters
- **NURBS**: *Non-Uniform Rational B-Spline*, a mathematical model using basis splines (B-splines) that is commonly used in computer graphics for representing **curves and surfaces**, whose shape is determined by control points (more information)
- **PBR**: *Physically-Based Rendering* designates approaches to render images in a way that **models the flow of light in the real world**, for example thanks to photogrammetry; many PBR pipelines aim to achieve photorealism; in practice they often rely on the **micro-facet theory**, with

specific materials (generally based on texture maps) and shaders (is also called PBS, for *Physically-Based Shading*); PBR is slowly becoming the standard for all materials

- **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 that 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*, collectively forming the skeleton or *rig*) in order to animate that mesh (more information)

- **Textures**: bitmaps (images) used to **skin 3D objects**, by defining the color of each point on the surface of the object in terms of texture coordinates; besides such 2D textures, 1D, 3D and 4D ones exist

- **Texture Atlas**: a texture (an image) containing a **set of separate, elementary graphic elements**, meant to be extracted based on 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, which implements workflows across a range of platforms, from mobile to high-end consoles and PC (in practice URP is the low-end counterpart of HDRP)

See also the Wikipedia's glossary of computer graphics.

Online Interactive Multimedia

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

Creation date: Sunday, January 16, 2022

Lastly updated: Wednesday, March 19, 2025

Table of Contents

Overview

Here, "online interactive multimedia" could be seen as an euphemism for networked video games, yet the topic may be a bit larger, including cases where some kind of online, persistent, multi-user virtual world has to be simulated, for example MMORPGs or any sorts of metaverse.

As for the topic of graphical 2D/3D rendering, it is specifically addressed in this section.

Networking Subsystem

Various architectures can be considered for networked applications, from a totally decentralised peer-to-peer one to a strict client/server one, possibly based on an authoritative server (which is to perform most of the world evaluation by itself, rather than delegating a part of this processing to clients).

Notably when an application is intensively interactive (e.g. a real-time strategy game, as opposed as a turn-based one), compensating for the lag and jitter induced by the network is a difficult technical challenge.

Dedicated solutions exist for that, either released as free software or commercially, and they are all built from the same standards.

Standards

Network Protocols

IP In terms of low-level network carriers, ultimately all traffic will be conveyed of course by the IP protocol (on the Internet or even in a local network) - but no one directly forges IP packets.

A little higher in the OSI model, communications will certainly be handled by the TCP and/or UDP protocols (which are both implemented on top of IP).

TCP The TCP protocol offers strong guarantees to the application: instead of thinking in terms of a stream of packets being sent (some possibly being lost or corrupted in the process), TCP provides *connections*, i.e. reliable bidirectional streams of bytes between two networked peers.

Yet this higher-level service comes at a price: latency. Indeed, under the hood, TCP has to detect communication issues and overcome them, typically by handling network congestion and requesting the re-emission of IP packets, which have thus to be waited for and delay the whole communication.

Many algorithms have been fine-tuned to maximise the resulting bandwidth and minimise the latency, yet of course guaranteeing a perfect communication remains demanding.

UDP The UDP protocol will be preferred whenever having data to be sent primarily with a low latency. As UDP packets can be lost or received in a order different from the emission one (data corruption is not a real problem, thanks to IP-level and UDP-level checksums), it is generally dedicated to transient, fast-paced exchanges, where the loss of a packet can be just ignored, the next ones making up for any lacking information with fresher data.

So UDP offers weaker guarantees, which is bound to increase the complexity of the application.

Howeover, depending on the application needs, better guarantees can be implemented on top of UDP, dealing with integrity, order and reliability. Of course the closer to the guarantees of TCP requirements are, the higher the cost of a UDP-based solution will be. Of course, if an application requires the properties provided by TCP, just use TCP rather than trying to recode a poor man's version of it over UDP.

Other Protocols & Facilities They include the WebSockets (on top of TCP) and WebRTC (*Web Real-Time Communication*).

A host of protocols are associated to WebRTC: SCTP (*Stream Control Transmission Protocol*, typically for a data channel), RTP (*Real-time Transport Protocol*) and SRTP (*Secure RTP*, typically for a media channel).

Middleware Their role is to marshall/demarshall application data so that it can be sent over the wire, through the aforementioned protocols: the information to sent through the network shall be transformed in a series of bytes that the other end will be able to interpret, according to a data format (that is generally cross-platform); this is a special case of serialisation/deserialisation.

A popular choice for that is Protocol Buffers (a.k.a. Protobuf).

Integrated Solutions

They provide a complete set of high-level services to be directly used by applications, implemented in libraries.

Free Software Solutions In this category, the main libraries that we spotted are:

- Mirror: open-source, moreover with a permissive licence

- Bevy engine: a data-driven game engine implemented in the Rust language
- DarkRift 2: an high performance, multithreaded and open source networking system for Unity

The Godot game engine also offers interesting network services. Godot has native support for Websockets, and libraries like Godobuf implement the decoding/encoding of Protobuf messages.

Commercial Solutions As for commercial offers, in order to build multiplayer games in Unity, one may take into account Unity3D Multiplayer Networking (Netcode) or Photon Fusion, and their Unreal counterpart.

Information Pointers

Much expert information on these topics can be read from the articles of Gaffer On Games.

As for Erlang-based servers, these posts are of interest: [1] and [2]. See also this Demonware presentation, which offers a rich Erlang-related operational feedback.

Application Architecture

The key topic here is *synchronisation*, i.e. how the game state is managed so that all players can seamlessly access it and enjoy fair interactions. The overall complexity increases drastically and multiplicatively on two dimensions (scalability and real-time), and theoretical results proved that upper bounds exist in that matter (see the CAP theorem and the FLP impossibility result).

For the core game (as opposed to backend services like authentication, chat, lobbies, match-making, etc.), a trade-off must be found between:

- a centralised architecture, where a logical server is authoritative, i.e. the sole controller of the truth
- a decentralised architecture, where the communication takes part mostly between peers, the logical server (if any) being merely a relay

Such a trade-off is game-specific, depending a lot on the intended reactiveness (consider a game of chess versus a frantic first-person shooter, where latency will be measured in terms of dozens of milliseconds).

The centralised architecture is simpler (e.g. a single, reliable true state exists; and by nature it better resists to cheating attempts), but it is more resource-demanding (bandwidth but also processing power) and depends a lot of the network-induced latency.

Some approaches like client-side prediction can hide a bit this problem; notably, when the same software runs on both sides (headless server and clients), the same logic can predictively result in the same evolution, which facilitates a lot any anticipations made by the clients.

As for decentralised architectures, they require some consensus to be reached between the peers involved, with a lack of trust to overcome. Not all peers have to be equal, for example each game session may elect a game leader (typically

the one enjoying the best overall connectivity with the other players): this host will be both the server and a player. One way or another, reliance onto the clients will be needed; short of being able to trust them, at least checking their reports and auditing them will be needed.

Network Management

Organisation: Copyright (C) 2021-2025 Olivier Boudeville

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

Creation date: Saturday, November 20, 2021

Lastly updated: Wednesday, March 19, 2025

Table of Contents

Investigating Network Issues

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

For example:

- to get all/most DNS records of a given domain, knowing that `any` requests (e.g. `dig +noall +answer +multiline TARGET_DOMAIN any`) may be not honored by some servers, we recommend using our list-dns-records.sh script
- to perform a reverse DNS lookup (thus translating a given IP address into a domain name), one may use: `drill -x TARGET_IP`.

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 (e.g. for masquerading, IPTV, different kinds of servers) as an example that we hope is secure enough[34].

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

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.

# 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 (e.g. 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; e.g.

```
[...]
$ 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` (then of course any previous rule may eclipse it)
- inserted either at the *beginning* of the selected chain with `-I`, or at its position N with `-I N`

Adding/removing rules "safely" from the command-line may be done more easily by rule specification rather than by rule position.

For example removing a given rule can be done:

- by first listing all rules thanks to `iptables -S`, to pick the one of interest
- by removing the prefix (typically `-A`) of the spec, and pasting the rest of that spec after `iptables -D` in order to remove the corresponding rule

Putting back this rule at the first position on its chain instead is just a matter of using `iptables -I` and the same shortened spec.

So, for example if wanting to silence an untrusted device (e.g. some "smart" box, a netcam of dubious origin), an approach is to:

- determine its MAC address and associate a static IP to it; for example, if using `dnsmasq`, in `/etc/dnsmasq.conf` one may define `dhcp-host=30:ef:50:36:54:68,10.0.17.203`, so that the IP `10.0.17.203` is always assigned to the device of MAC address `30:ef:50:36:54:68`
- define a firewall rule so that one's gateway never forwards outgoing packets from that (local) IP, for example with: `iptables -I FORWARD -s 10.0.17.203/32 -i my_lan_interface -j DROP`

See also:

- our filtered_local_hosts section of `iptables.rules-Gateway.sh`, the hosts being filtered out based on the ones specified in the `setting_file` configuration file
- 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
- to resolve a DNS hostname, i.e. to obtain its IP from its name: `dig +short my.host.in.domain.tld | tail -n1`
- to review *host-local* open ports and sockets, use:
  - `ss` (for *socket statistics*, replacement of `netstat`), e.g. to determine which program is running at TCP port #8080: `ss --inet --listening -np | grep :8080`
  - `netstat` could be for example `netstat -ltnp | grep :8080`
  - `lsof -i :8080`
  - `fuser 8080/tcp`, before looking up the returned PID
- for remote testing of open ports and sockets, use `nmap`
- 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 (e.g. 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 (e.g. 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 (e.g. 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 (e.g. 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

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

Creation date: Saturday, November 20, 2021

Lastly updated: Wednesday, March 19, 2025

Table of Contents

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 section about OpenPGP below

Authentication Using SSH

This is certainly the right approach to rely on nowadays (e.g. no more HTTPS with a `.netrc` file).

To generate a key pair, one may use `ssh-keygen` and prefer for example the ED25519 cipher; for example, if wanting to use a specific key filename:

```
$ KEY_PATH="~/.ssh/id_ed25519-${USER}-for-general-use-$(date '+%Y%m%d')"
$ ssh-keygen -t ed25519 -C "General, personal key notably for VM connections, generate
```

One may:

- specify a passphrase (which would then be requested at each use of the private key), or not
- not specify a file name, if the new key would not collide with any pre-existing one

Then the corresponding *public* key may be transferred to any target host, like in:

```
$ ssh-copy-id -i "${KEY_PATH}" foobar.org
```

(it would be then the unique time for this host that the user general password on the target host would be requested)

and/or declare it in any web interface (e.g. GitLab) of interest (associating to one's user one's public key).

In a Git clone, any specific key name must be specified, typically with:

```
$ git config core.sshCommand "ssh -o IdentitiesOnly=yes -i ${KEY_PATH} -F /dev/null"
```

Securing One's E-mail Service In General

Now that electronic exchanges are central to most communications, controlling one's e-mail services is of paramount importance.

It is a real pity that most individuals will not be able in practise to run their own mail server (Message transfer agent), short of being able to setup through their Internet provider a proper reverse DNS information (as any e-mail that such a home server would emit would likely be considered as spam/junk mail by the recipients).

So one will have to resort to third-party e-mail services ("MX plan"). We do not see the reliance on one's Internet provider e-mail solution as a good choice (even if being able to make use of such an address regardless of any active subscription to that provider), possibly not all registrars are also good email hosters, and surely we do not want to depend on any GAFAM-related service.

The last remaining solution is therefore to elect a dedicated provider of e-mail hosting, which most of the time incurs monthly fees, albeit rather low (e.g. 1 euro per month).

In that case we strongly recommend:

- choosing an e-mail hosting that supports for real SPF, DKIM and DMARC, and a catch-all in terms of incoming e-mail addresses (we see all these as hard requirements; an example of a corresponding provider being, certainly among others, Mailo - although we would have liked that it supported as well MTA-STS for an increased security, including for OpenPGP; refer to these information to better understand the interest of MTA-STS; DANE and DNSSEC would be also of interest)
- purchasing (actually leasing) one's domain name (like `foobar.org`), in order to obtain (hopefully life-time) e-mail address(es) (e.g. `john.doe@foobar.org`); this is another little expense, around 10-15 euros per year, yet it opens many other interesting opportunities (starting with the possibility of hosting one's webserver and other online services)

Wanting to test whether your current e-mail system is reliable? We recommend using the MECSA online testing tool, which is provided by the European Commission and allows to learn a lot about the level of security reached by one's email system.

Increasing Security 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.

Common solutions exist to ensure that a given mailserver is indeed in charge of a given domain name (SPF), that a given email originates from a given mailserver and has not been tampered with (DKIM) and that any non-conformance can be managed according to a policy recommended by the emitter (DMARC), but none is about the privacy of your messages.

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 A 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[35], so applying the default settings retained by gpg, or similar ones yet a bit stronger (e.g. 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`

[35] 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 reasonable gpg defaults, or use paranoid settings only for a fully-private primary key whence actual work keys are derived.

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 (e.g. for `Real name`, one may enter `James Bond`), one's email address of interest (e.g. `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 that such a file exists)

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>
```

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 multiple, 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):
  - 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[36].

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-sec
```

[36] 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).

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 (e.g. 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:

```
$ 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; e.g. 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 public 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); apparently uploading each public key separately (if multiple ones are associated to a given master key) shall be preferred so that they can be found by a look-up based on an electronic address.

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 match is returned is important in order 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 (except if DKIM is used).

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)[37] or a (shorter) fingerprint thereof (e.g. the full fingerprint of my key is `B8235ECE469EB77F`).

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 (e.g. 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 (e.g. binaries), so that the intended public can check the signature if wanted
- **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`

[37] Note the HTTPS certification.

- **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, unless special measures are taken
- **access to online services**, such as GitHub, GitLab, etc., typically to sign commits
- **encrypt** and/or **sign** emails

Updating Your Keys Keys are meant to expire, so that they are updated as technology progresses.

Typically tools (e-mail clients like Thunderbird) will notify the user whenever once of their registered key is out of date.

To check whether one's (secret) keys expired:

```
$ gpg --list-secret-keys --keyid-format LONG
[...]
sec   brainpoolP512r1/3D60ADA5E251A8B5 2021-11-26 [SCA] [expired: 2022-11-26]
      C3987680AD9B79FDC6B7D25C9D60ADA5E115A8B5
uid                 [ expired] James Bond <james.bond@mi6.org>
[...]
```

Below the method to update keys (primary or subkeys) is described; once done, the new versions can be published (see the Decentralised Sharing section) and then found by tools (typically e-mail clients like Thunderbird), which can be requested to search for them in key servers.

Update a Primary Key So here the primary key 3D60ADA5E251A8B5 expired and shall be renewed; let's extend it of two years, and proceed interactively, based on its KEYID:

```
$ gpg --edit-key 3D60ADA5E251A8B5
Secret key is available.

sec  brainpoolP512r1/3D60ADA5E251A8B5
    created: 2021-11-26  expired: 2022-11-26  usage: SCA
    trust: ultimate      validity: expired
[ expired] (1). James Bond <james.bond@mi6.org>

gpg> expire
Changing expiration time for the primary key.
Please specify how long the key should be valid.
        0 = key does not expire
     <n>  = key expires in n days
     <n>w = key expires in n weeks
     <n>m = key expires in n months
     <n>y = key expires in n years
Key is valid for? (0) 2y

Key expires at Wed 01 Jan 2025 12:40:27 PM CET
```

```
Is this correct? (y/N)

sec  brainpoolP512r1/3D60ADA5E251A8B5
    created: 2021-11-26  expires: 2025-01-01  usage: SCA
    trust: ultimate      validity: ultimate
[ultimate] (1). James Bond <james.bond@mi6.org>

 gpg> trust
 gpg> save
```

Update Subkeys Here let's suppose that we created 4 subkeys from a primary one: K1 dedicated to signing, K2 to encrypting, K3 to authenticating and K4 (typically to be declared to your e-mail client) to perform these three operations.

The lifespan of these subkeys has been chosen relatively small, 2 years, and intentionally smaller than the primary key they derive from.

Let's extend their lifespan of 2 more years, by selecting them all:

```
# Based on primary one:
$ gpg --edit-key 3D60ADA5E251A8B5
sec  brainpoolP512r1/54EC65AAE18A5162
   created: 2023-01-02  expires: 2029-01-21  usage: C
   trust: ultimate      validity: ultimate
ssb  rsa4096/K1**************
   created: 2023-01-02  expires: 2027-01-22  usage: S
ssb  rsa4096/K2**************
   created: 2023-01-02  expires: 2027-01-22  usage: E
ssb  rsa4096/K3**************
   created: 2023-01-02  expired: 2025-01-01  usage: A
ssb  rsa4096/K3**************
   created: 2023-01-02  expired: 2025-01-01  usage: SEA

# Select these subkeys (they are then listed with a star):
$ key K1**************
$ key K2**************
$ key K3**************
$ key K4**************
$ expire
2y
save
```

Deleting Your Keys Typically if having expired keys not intended to be renewed:

```
$ gpg --delete-secret-key KEYID
```

A Corresponding Cheat Sheet

Root Key Create a master key that will never leave one's network, thanks to `gpg --expert --full-generate-key`:

- with a strong algorithm (e.g. `Brainpool P-512`, by selecting `ECC (set your own capabilities)`)
- only able to certify (C) - not sign (S) or authenticate (A) - thus switch off the sign capability
- with a rather long lifespan (e.g. 6 years, hence `6y`)
- with a relevant comment (e.g. `this public key can be verified against its reference in https://mi6.org/james-bond.pub`), as subkeys will inherit it

Possibly add other identities (typically email addresses), with `gpg --edit-key`, `adduid`, then `uid 2`, `trust`, `uid 2` (to unselect), `primary` (for uid 1) and `save`.

Sign this new key with any past one:

```
$ gpg --default-key OLDKEYID --edit-key NEWKEYID
gpg> sign
gpg> save
```

Subkeys Define as many of them as needed (based on `gpg --expert --edit-key NEWKEYID`), possibly one for signing, one for encrypting and one for authenticating (apparently not automatically generated), each time with `addkey`; select a strong yet commonly-accepted algorithm (e.g. RSA 4096 bits), and a shorter lifespan (e.g. 2 years).

A simpler "triple" use (S/E/A) single key may be preferred, or additionally created.

Prepare Revocation Create a revocation certificate for your master key, specifying reason 1 (`Key has been compromised`) and clarify with a comment (`This revocation certificate has been generated at key creation.`):

```
$ gpg --output $(date '+%Y%m%d')-masterkey.gpg-revocation-certificate.key --gen-revoke
```

Stores possibly multiple offline copies of that certificate.

Finally Backup secret keys:

```
$ gpg -o $(date '+%Y%m%d')-full-key-backup-for-james.bond.gpg --export-secret-keys
```

Dispatch the various public keys generated (for signing, encrypting, authenticating):

```
$ gpg --send-keys KEYID
```

Transfer the `james-bond.pub` public key to webserver (e.g. `https://mi6.org/`), based on `gpg -o james-bond.pub --export PUBKEYID`.

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 them in 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
- for a proper OpenPGP support, we have had to change our e-mail client, from Thunderbird (problems importing by itself strong/recent key types, and non-terminating attempts of reading the local pgp key ring) to Evolution (worked directly as expected); yet Evolution came with other drawbacks (regarding agenda/calendar features), so we came back to Thunderbird with rather reasonable settings

Obtaining my Current Public Key

As the time of this writing (Tuesday, January 3, 2023), my daily key for signing / encrypting / authenticating is a RSA 4096-bit one designated as `3e090de4d08e42944d195a7bb8235ece469eb77f`.

It can be obtained by different means:

- downloading https://esperide.com/olivier-boudeville.pub
- searching in SKS key servers, e.g. keyserver.ubuntu.com
- searching in the openpgp.org key server

See Also

- a complete, well-written tutorial, in French: Bien démarrer avec GnuPG
- still in French: other interesting usage hints and GnuPG : Créer la paire de clé gpg parfaite
- GnuPG on Arch, for much additional information
- Network Management information

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 (e.g. SHA1), offer a solution bringing many interesting properties and opening new possibilities to distributed systems (e.g. for blockchains, a user account is often identified by the fingerprint of its associated public certificate).

About Build Tools

Organisation: Copyright (C) 2021-2025 Olivier Boudeville

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

Creation date: Saturday, November 20, 2021

Lastly updated: Wednesday, March 19, 2025

Table of Contents

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 (e.g. 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 (e.g. 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 (e.g. `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.

Version Control Systems: in Practice, now, Git

Organisation: Copyright (C) 2021-2025 Olivier Boudeville

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

Creation date: Saturday, November 20, 2021

Lastly updated: Wednesday, March 19, 2025

Table of Contents

Overview

No real software development shall happen without the use of a VCS - standing for *Version Control System* - of some sorts, notably in order to track the versions of the source files involved and to ease the collaborative work on them.

Many solutions have been defined for this purpose (CVS, Clearcase, SVN, Mercurial, etc.), but now a single tool is the de facto standard: Git, which is a distributed version control system available as free software; refer to its website for more details.

Git Usage

Beyond the documentation relative to its general use, projects have to adopt their own set of conventions - regarding the management of branches, commits, tags, etc - based on their preferences and context.

Recommended General Conventions

These are certainly very basic and possibly idiosyncratic, yet any VCS content (typically a release branch) should, in our opinion, meet a few criteria in terms of quality.

The ones to which we try to stick are:

- the path (including names) of files and directories should not contain diacritic, special character or spaces - only plain, basic, boring ASCII characters; as separator, prefer dashes (`-`) to underscores (`_`)
- the character case shall be uniform (e.g. directory names starting always or never with a capital letter)
- the language used shall be uniform (e.g. only proper English)
- a commit message shall describe synthetically the modifications operated on the corresponding new filesystem snapshot; such a message, preferably in English, should always start with a capital letter and end with a dot; e.g. "*Fixed the computation of angles.*" or "*Upgraded the Frobnicator to 2.1.*"
- permissions, both for directories and files, shall be uniform and properly set (e.g. no plain file shall be executable)
- the file formats shall be, as much as possible, homogeneous, notably for text files with regard to the line terminators; either only UNIX conventions (only `LF`; preferred), or only Windows ones (`CRLF`); use `dos2unix` whenever necessary, possibly automated through a Git hook
- abbreviations are convenient (e.g. `br` for branch, `co` for `checkout`, etc.); they can be defined in one's `~/.gitconfig`
- a repository should:
  - only include relevant, well-selected binary files (*if any*)
  - neither include generated files (at least no trivially-generated one), such as `*.o`, `*.so`, `*.class` nor useless/temporary ones (such as `.~lock.foo.bar#` or `commit-30860f7`, `buz.log`)

Base Know-How

- **specifying a target revision**:
  - a revision is generally the name of a commit object; it can be its SHA-1, or a symbolic name like `HEAD` or `master@{yesterday}`
  - the path that it designates may be either "absolute" (i.e. relative to the root of the clone, like `PREFIX:a/b/c/my-file.txt`) or relative to the current directory in the clone (provided they start with `./`, like `PREFIX:./c/my-file.txt` if being in the previous `b` directory)
  - suffixes may be added:
    - `~[n]` (e.g. `v1.5.1~3`) allows to designate the generation ancestor (parent, grand-parent, etc.) of a revision; for example: `git show HEAD~:a/b/c/my-file.txt` shows the first parent of the current commit of that file
    - `^[n]` (e.g. `HEAD^` or `HEAD^2`) allows to designate the n-th parent (at the same level, should a commit have multiple parents, like after a merge) of the current commit of that file
- **managing branches**:
  - **creating** branches is done thanks to the `checkout` command, often abbreviated as `co` here
  - to create a branch deriving from the current one (the current `HEAD`) and switching to it at the same time (performing its co, while inheriting any local changes): `git co -b my_new_branch`
  - to create a local branch corresponding to a remote one (let's suppose it is named `some_branch`), assuming that a remote server (e.g. `my_remote`, possibly `origin`) has already been declared (e.g. `git remote add my_remote URL`):
    - first step is to update the remote-tracking branches with `git fetch my_remote`, then to create the target local branch tracking that remote (upstream) one with: `git co -b some_branch my_remote/some_branch` (also switching to it here)
    - a shortcut is to use `git co --track my_remote/some_branch` instead
    - even shorter, if the name of the target local branch name does not exist yet and matches exactly matches a name on only one remote, `git co some_branch` will suffice
  - to **delete** a local branch (while another one is checked out): `git branch -d my_branch`; to force-delete it (typically if not fully merged), use `git branch -d my_branch`; to delete it from a specified remote as well: `git push origin --delete my_branch`
- **managing tags** (note that they are repository-level, for example not related to any current branch):
  - to list all (local) tags: `git tag`

  - to list all (annotated) tags, from the oldest one to the latest one: use Ceylan-Hull's `list-tags-by-date.sh` script
  - to have information about an already-existing tag: `git show my_tag`
  - to set a new annotated tag: `git tag -a foobar-version-2.4.0 -m "Release of the version 2.4.0 of Foobar."`; prefer naming tags differently from branches (e.g. `foobar-version-2.4.0` rather than `foobar-2.4.0`) to spare ambiguities to Git
  - a set tag must be specifically pushed on a remote, for example: `git push origin my_tag`; all tags can be pushed with `git push --tags` (the remote can be implied)
  - to delete a tag that was not pushed: `git tag --delete my_tag`

- **determining whether a file is in VCS**, knowing that due to `.gitignore` rules, `update-index --skip-worktree`, etc. it is not always obvious:

```
# Target file is tracked iff is listed by:
$ git ls-files | grep my_file

# Or, in order to trigger an error if this target file is not tracked:
git ls-files --error-unmatch my_file
```

- getting the **version of a file as it was at a given revision**:

```
# Replaces the current version of that file by the designated one:
$ git checkout COMMIT_ID path/to/the/target/file

# So, for example, in order to revert/replace the current version of a file by
# the one at which is was *prior* to a given commit:
#
$ git checkout COMMIT_ID~ path/to/the/target/file

# Outputs on the console the designated version:
$ git show COMMIT_ID:path/to/the/target/file

# Outputs on the console the diff between the designated
# version and the current one:
#
$ git diff COMMIT_ID:path/to/the/target/file
```

- listing the **files modified by a given commit**: `git show --name-only MY_COMMIT_ID` will print relevant extra information; if wanting only the list of modified files: `git diff-tree --no-commit-id --name-only -r --pretty COMMIT_ID`

- **merging a remote branch**, without having it / updating it *locally*: for example, to merge the remote `main` branch in the current `my_branch` branch, without updating the local counterpart of `main` (i.e. without having to run for example `git checkout main && git pull && git my_branch` first), just execute: `git merge origin/main`

Managing Branches

Creating branches allows to separate threads of work (while preserving their lineage) and progress concurrently. Yet often their content will have to converge ultimately; depending on the intent, two use cases can be considered, resulting in different Git uses.

Merge versus Rebase Here one may want:

- either to **integrate back a development branch** (e.g. `my-feature`) **in a shared, parent one** (e.g. `master`): then one shall prefer using `merge`, in order to keep separate histories and not affect the past one of the shared branch
- or to **resynchronise a development branch** (e.g. `my-feature`) **on the last version of a shared branch and continue these developments**: then one shall prefer `rebase`, so that the history of the development branch contains only its own changes (less noise, linear history)

In practice, in order to transfer the changes of a branch A in a branch B:

```
$ git co B

# Either first case (integrate development A in master B):
$ git merge A # or: git pull A

# Or second one (resynchronise development B on master A):
$ git rebase A # or: git pull -rebase A
```

How such a last `rebase` of branch A in branch B is done? The bifurcation point of B compared to A is moved from its initial position to the current head of A, on which all changes recorded in B are applied; the resulting history of B looks like if these changes had been directly performed from the version of A designated in this rebase, and thus B can be then directly fast-forwarded to its tip, which comprises both the changes synchronised from A and, then, the ones specifically introduced in B.

Then, to update the remote with these post-rebase commits, `git push --force-with-lease` shall be used[38].

More information: [1] or, in French: [2], [3], [4].

Directly Transferring Changes Sometimes, one may want to directly transfer the changes of a derivate branch B in a parent branch A. When one knows for sure that the versions in B shall be preferred in all cases to their counterparts in A (note that a classical merge is already fully able to manage fast-forwards), one may use:

```
$ git checkout A
$ git merge -X theirs B
```

[38] Rather than just performing just a push, having it fail, pulling, and ending up with duplicates of the changes. Should this happen, rewind these changes, for example with: `git reset --hard <full_hash_of_commit_to_reset_to>`.

No conflict should arise (source); note that this does *not* imply that the contents of the two branches match.

The same is possible with `rebase`; for example: `git rebase -X theirs B`. Note that `-X` a *strategy* option, whereas `-s` would be a *merge strategy* option. Using here `ours` rather than `theirs` :

- `-X ours` uses "our" version of a change *only when there is a conflict*
- whereas `-s ours` ignores the content of the other branch entirely (in all cases), and use "our" version instead; `-s theirs` does not exist

Another (brutal but sure) way of forcing the content of a branch B to be the same as the one of a branch A is, while B is checked-out, to execute: `git reset --hard A`. As mentioned previously, push shall be done then with `git push --force-with-lease`.

Sometimes, we know for sure that a given file must be transferred as it is on a given branch (`some_branch`), as a whole, to the current branch (`current_branch`). Cherry-picking is not exactly what is needed, as git cherry deals with commits, not actual contents. Here, what we presented with checkout may be more appropriate:

```
$ git checkout current_branch
# The current location on the branch matters:
$ git checkout some_branch path/to/the/target/file
Updated 1 path from 6f154364
```

Merging with no Auto-Commit Often a bit anxious to acknowledge an automatic merge with so little control on the corresponding changes?

One approach is, when being in a branch A, to execute `git merge --no-commit --no-ff B`: the merge of B in A will be done, but not committed, leaving the possibility to review - and possibly correct - it.

The staged changes can indeed then be inspected with `git diff --cached` (or our `difs` script) and, if finding a file whose merge is not satisfactory, just correct it (possibly `git restore` its version from A), and possibly further fix the merge, before committing it.

If there was at least one conflict, run `git merge --abort` to get rid of the 'MERGING' state.

Common Procedures

Overcoming auto-signed SSL certificate issues To avoid, typically in a company internal setting, errors like:

```
Cloning into 'XXX'...
fatal: unable to access 'https://foo.bar.org/XX/XXX/': SSL certificate problem: self s
```

the `http.sslVerify=false` option may be used, even if it weakens the overall security.

This is typically useful initially:

```
$ git -c http.sslVerify=false clone https://foo.bar.org/XX/XXX
```

In order that the next operations (e.g. future pushes) overcome too this problem for the current repository, use from within the current clone:

```
$ git config http.sslVerify false
```

Setting the right metadata for the next commits Doing so prevent from having to amend commits a posteriori.

If these information apply for all projects:

```
$ git config --global user.name "John Doe"
$ git config --global user.email john.doe@foobar.org
```

Otherwise shall be done at least on a per-project basis with:

```
$ git config user.name "John Doe"
$ git config user.email john.doe@foobar.org
```

Also `git config --global --edit` may be of use (beware to trigger a `vi` by accident...).

Performing operation on remotes with no systematic authentication Using a SSH key pair, hence with its public key declared on said remote, is a relevant approach, safer and more convenient than from example using a `~/.netrc` file, with or without a *Personal Access Token* (PAT) - as they tend to expire.

Refer to this section in order to set up a proper SSH configuration for that.

Updating One's Fork from its Upstream So you forked a repository (let's say it is in `https://github.com/some_project/some_repo.git`) and made progress - yet in the meantime the upstream repository may also have been updated, and you want to integrate these changes in yours.

First step is to ensure that this repository (designated here as `upstream` for convenience) is locally known:

```
$ git remote add upstream https://github.com/some_project/some_repo.git
```

Then, from a fully-committed clone of your fork (let's suppose we are using the `main` branch in all repositories):

```
$ git fetch upstream

# More appropriate than a merge:
$ git rebase upstream/main

# Repeatedly, as long as conflicts are found:
$ git rebase --continue

# Forced, as otherwise the current branch will deemed to be behind our remote:
# (hopefully your branch at origin is not protected by a hook; otherwise:
# 'git checkout -b some_branch', etc.)

$ git push -f origin main
```

Creating an empty branch Rather than creating it from a pre-existing branch and removing all inherited content, prefer:

```
$ git checkout --orphan my_new_branch
```

(typically useful for GitHub Pages branches; may then be followed by some adds and `git commit --allow-empty -m "Initial website."`)

Listing differences with prior versions of a file In order to list the differences of a given file with the previous commits (precisely: of a set of pathspecs), one may use our `dif-prev.sh` script, which by default reports the differences with the last committed version. With the `--all` option, it lists all differences, until the first addition of this file.

Performing a Merge with Emacs Our procedure is to rely on our configuration of Emacs, which configures the `smerge-mode` (which is automatically triggered in the case of files containing conflicts; see the `SMerge` menu) so that it relies on the `C-c v`[39] smerge command prefix (that we found more convenient).

Then the following main commands are useful:

- to move between conflicts:
  - go to *next* one: `smerge-command-prefix n` (for `smerge-next`), which thus corresponds to `C-c v n`
  - go to *previous* one: `smerge-command-prefix p` (for `smerge-previous`), which thus corresponds to `C-c v p`
- to select a version:
  - the one the cursor is currently on: `smerge-command-prefix RET` (for `smerge-keep-current`), which thus corresponds to `C-c v RET`
  - that was in our current merge-target branch: `smerge-command-prefix m` (for `smerge-keep-mine`), which thus corresponds to `C-c v m`
  - that is in the other branch, to merge in our current one: `smerge-command-prefix o` (for `smerge-keep-other`), which thus corresponds to `C-c v o`

Sometimes we would like to accept or refuse **all** changes of a file *as a whole*; apparently there is no way of doing so easily with `smerge-mode`, so to select a full file version we recommend executing, respectively, `git checkout --theirs -- MY_FILE` or `git checkout --ours -- MY_FILE`.

See also our more general Emacs section.

[39] Hence: press and hold the "Ctrl" key, hit the "c" key, then release all, then press the "v" key, and release. This is obtained thanks to `(setq smerge-command-prefix "\C-cv")`.

Using the Stash The stash allows to record the current state of the working directory and the index while going back to a clean working directory: the command saves the local modifications away, and reverts the working directory to match the HEAD commit.

The stash is convenient in order to switch branches without having to perform an arbitrary (meaningless) commit just for the sake of switching.

A basic use of the stash is the following:

- to store the currently modified files in the stash: `git stash` (equivalent of `git stash push`)
- to restore the (by default lastly) stored state (hence the reciprocal of the previous command), i.e. to apply it to the current files (with no review and no changes to add afterwards) and then drop that stashed entry: `git stash pop [STASH_INDEX]`
- to list the various sets of modifications stashed away: `git stash list`
- to show the changes recorded in the stash entry as a diff between the stashed contents and the commit back when the stash entry was first created:
  - with just one synthetic line per file changed: `git stash show [STASH_INDEX]`
  - with all actual changes: `git stash show -p [STASH_INDEX]`
- to drop a stash entry: `git stash drop [STASH_INDEX]`

Refer to the git stash documentation for more information.

Preventing the commit of a file in VCS that is often locally modified One should use this method:

```
$ git update-index --skip-worktree <file-list>
```

The opposite operation is:

```
$ git update-index --no-skip-worktree <file-list>
```

Listing the files managed in VCS from the current directory Use `git ls-files` to determine the files that are already managed in VCS, recursively from the current directory.

To list the untracked files (i.e. the files *not* in VCS), use `git ls-files --others`.

Reducing the size of a repository One may use our list-largest-vcs-blobs.sh script to detect any larger files that should not be in VCS (e.g. should a colleague have committed by mistake a third-party archive, or unexpected data such as CSV files).

Then install BFG Repo-Cleaner:

```
$ mkdir -p ~/Software/bfg-repo-cleaner/
$ cd $_
$ mv ~/bfg-1.14.0.jar .
$ ln -s bfg-1.14.0.jar bfg.jar
# For example in ~/.bashrc:
$ alias bfg="java -jar ~/Software/bfg-repo-cleaner/bfg.jar"
```

All developers should be asked to commit their sources (git add + push), to archive their clone (e.g. in a timestamped `.xz` file like `20220412-archive-clone-foobar.tar.xz`), and to wait until notified that they can create a new clone.

The repository may be then cleaned up (e.g. from large, unnecessary CSV files) in isolation, with:

```
$ git clone --mirror XXX/foobar.git
$ bfg --delete-files '*.csv' foobar.git
$ cd foobar
$ git reflog expire --expire=now --all && git gc --prune=now --aggressive
$ git push
```

Then all developers shall be requested to perform a new clone and to check the fetched content (e.g. with regard to the content of the last branch in which they committed).

Fixing LF vs CRLF End of Line Problems Use Git Attributes to specify proper files and paths attributes.

One may define a `.gitattributes` file for example with `*.js eol=lf`, `* text=auto`, or:

```
# No CRLF conversion for DOS/Windows batch files.
# They should be stored with the CRLF line terminators.
#
*.bat -crlf
```

Fixing a commit message If no push was done, it is as simple as replacing the former message by a new one, like in:

```
$ git commit --amend -m "This is a fixed commit message."
```

Removing a commit (rollback) The goal here is to withdraw (revert) a (presumably faulty) commit.

If it has not been pushed to remote, use `git reset HEAD~1`; otherwise (already pushed), use `git revert HEAD` (then a push can be made).

Restoring a branch to a past state Sometimes mistakes are made, committed and pushed - typically when messing up some merge.

Various operations can be of use to correct them:

- to consult a past state (e.g. based on a SHA1) and possibly create a dedicated branch out of it: use `checkout`

- to selectively remove the changes introduced by specific commits, by adding reverting commits (hence not losing history); use `revert`; such an operation must be validated thanks to a well-documented commit
- to come back to a past overall state (e.g. a SHA1), losing all changes done since then (hard delete, rewriting history): use `reset`, with the `--soft` option to keep intermediary changes as non-committed, or with the `--hard` option to remove all changes; no commit is involved here (newer commits being just forgotten); if satisfied with this new state, it may be validated thanks to `git push --force origin HEAD` - provided that the current branch is not protected

See also these exchanges.

Listing local or list the remote branches by their last modified date
For *local* ones:

```
$ git for-each-ref --sort='-committerdate:iso8601' --format=' %(committerdate:iso8601)

2024-05-24 18:02:52 +0200  refs/heads/17-xxx
2024-05-24 18:02:52 +0200  refs/heads/main
2024-05-24 14:51:15 +0200  refs/heads/36-yyy
2024-05-24 12:19:51 +0200  refs/heads/zzz
2024-04-19 18:03:37 +0200  refs/heads/aaa
```

For *remote* ones:

```
$ git for-each-ref --sort='-committerdate:iso8601' --format=' %(committerdate:iso8601)
```

See also:

- our list-lastly-updated-vcs-branches.sh script
- these exchanges.

Tools

On Most Platforms

At least on UNIX, the command-line Git client (`git`) is certainly the best tool. In difficult situations, graphical tools such as `gitk` may be of help.

Some distributions (e.g. Debian) do not come with a relevant Git autocompletion (for commands, branches, etc.) regarding one's shell of interest (like Bash). A solution is to download git-completion.bash, store it for example in `~/Software/Git/`, and source it automatically from one's `~/.bashrc`.

See also our Ceylan-Hull section about VCS-related scripts.

On Windows

Tools like TortoiseGit may foster a view on the usage of Git that is a bit particular, conflating concepts or introducing extra ones (e.g. a `sync` command). Apparently also at least some pulls did not reintroduce files just removed from the working directory.

More generally, cloning on a Windows host an UNIX-originating repository comprising symbolic links may induce oddities (e.g. a symlink named `S` pointing to `Foobar` resulting, on a Windows clone, in a file named `S` whose content is, literally, the text `"Foobar"`, instead of the expected content of the `Foobar` file).

Another option is to use Visual Studio Code (`vscode`), which supports natively Git (provided that the command-line version is already installed). One may select `View -> SCM` (or `Ctrl-Shift-G`) for that. Clicking on the "VCS" icon (three rings links by two curves; the third from the top) displays a contextual view offering various associated operations (here based on Git).

We finally preferred using MSYS2 + Git rather than Git Bash, named "Git for Windows"; hints to speed up these tools may apply.

Inner Workings

Git stores internally every version of every file separately (not as a diff with a parent version) as a blob (an opaque binary content) identified by its (SHA1) hash.

A commit is the identifier of a tree representing the filesystem of interest at a given moment (snapshot). This tree references the files through their SHA1, similarly to a Merkle tree.

A branch is thus nothing but a pointer on a given commit, and `HEAD` designates the current branch. Git stores natively only blobs, trees and commits.

The reported differences in the content of a file or a tree are thus only recreated (established dynamically) by Git commands, they are not natively tracked.

Translations

From English to French:

- `repository` -> `dépôt`
- `to checkout` -> `extraire`
- `to commit` -> `valider`
- `a commit` -> `une validation`
- `a tag` -> `une étiquette`
- `in VCS` -> `en GCL` (Gestion de Configuration Logicielle)
- `snapshot` -> `instantané` (de l'état du sous-système de fichiers géré en GCL)
- `merge` -> `fusion`
- `head` -> `tête`

- `fast-forward` -> `avancement direct`
- `fast-forwarded` -> `directement avancée`

Documentation

Many pointers exist, doing a great job in unveiling how Git is to be used.
In English, Pro GIT is surely a reference.
In French:

- introduction en français
- cours sur OpenClassrooms
- référence incontournable et conseillée : Pro GIT, notamment pour l'explication de ses rudiments puis de son fonctionnement interne, à commencer par ses objets

Documentation Generation

Contact: about (dash) howtos (at) esperide (dot) com
Creation date: Wednesday, January 12, 2022
Lastly updated: Wednesday, March 19, 2025

Table of Contents

Objective

The goal is to generate nice documentations of any kind (not necessarily technical), as static content - as opposed to wikis or content management systems (CMS).

We want to be able to generate, **from a single source**, at least two **documentation formats**:

- a set of **interlinked static web pages** (the most popular, flexible format)
- a single, **standalone PDF file** (convenient for offline reading, printing, etc.)

The document source shall be expressed in a simple, non-limiting, high-level syntax; in practice a rather standard, lightweight markup language.

All standard documentation elements shall be available (e.g. title, tables, images, links, references, tables of content, etc.) and be customisable.

The resulting documents shall be quickly and easily generated, with proper error report, and be beautiful and user-friendly (e.g. with well-configured LaTeX, with appropriate CSS, icons and features like banners, with proper rendering of equations).

Per-format overriding shall be possible (e.g. to define different image sizes depending on web output or PDF one).

The whole documentation process shall be powered only by free software solutions, easily automated (e.g. with Make) and suitable for version control (e.g. with Git).

For that we rely on two possible approaches, a lighter one and a more involved one, depending on the project at hand.

Our Preferred Lightweight Approach

Principle

We found this approach convenient for lighter projects, i.e. ones comprising a limited number of pages. This is the case of this page and more generally of the whole Ceylan-HOWTO website.

We chose to rely on the reStructuredText syntax and tools, also known as RST, a part of the Docutils project. Here we do not specifically rely on elements related to Python or the Sphinx toolchain, as our more heavy-duty approach does.

We augmented reStructuredText with:

- a set of **make-based defines and rules** (automatic or explicit) that were aggregated in Ceylan-Myriad (see notably GNUmakerules-docutils.inc and the generate-docutils.sh generation script); this mechanism is layer-friendly, in the sense that all layers defined (directly or not) on top of Myriad are able by default to re-use these elements and to customise them if needed
- a **template** on which we rely for most documents, featuring notably a standard table (to specify usual metadata such as organisation, contact information, abstract, versions, etc.), a table of contents, conventions in terms of title hierarchy and, for the HTML output, a banner (a fixed, non-scrolling panel offering shortcuts, in the top-right corner of the page)

- a simple **tag-based system** to have the actual document markup (`*.rst`) directly generated from a higher-level source one (`*.rst.template`); in practice, if defined, only the latter element is edited by the user, and tags (such as `*_VERSION_TAG`, `*_DATE_TAG`, etc.) are automatically filled-in appropriately

Of course this website, and many others that we created, rely on this approach; as an example, one may look at the sources of the current document.

Specific Topics

Rendering Mathematical Elements With the RST toolchain, the PDF output, thanks to LaTeX, offers built-in high-quality rendering of mathematical elements such as equations, matrices, etc.

By default, the HTML output does not benefit from LaTeX, and remains significantly less pleasing to the eye, and less readable.

So we complement it by MathJax, a neat open-source "*JavaScript display engine for mathematics that works in all browsers*".

It shall thus be installed once for all first.

Basic, Less-than-Satisfactory, Installation Approach For example, on Arch Linux, as `root`, it is sufficient to execute:

```
$ pacman -Sy mathjax
```

If not having root permissions, it can be installed directly in one's user account, for example:

```
$ cd /tmp && git clone https://github.com/mathjax/MathJax.git
$ mv MathJax/es5 ~/Software/MathJax
```

Then, to enable the use of MathJax for a given website based on Ceylan-Myriad, run from its root (often a `doc` directory):

```
$ make create-mathjax-symlink
```

(this target is defined in GNUmakerules-docutils.inc; it boils down to symlinking `/usr/share/mathjax`; see also the HOWTOs corresponding makefile to properly manage this dependency afterwards, notably when deploying web content)

Yet, depending on settings and conventions, updating MathJax in a web root may lead to permission errors; in that case the next approach shall be favored.

Better, Webroot-compliant Installation Approach Just install MathJax directly in your user account (e.g. in `~/Software/mathjax`), follow the guidelines in the Fixing Permissions in Third-Party Content to Integrate in a Web Root section, and add symlinks to the result in all documentation trees requiring MathJax.

For that, rather than installing MathJax by oneself (as we found its website rather unclear about how to install it when not in a Node.JS context) or

possibly taking inspiration from this PKGBUILD, the simplest way is to install it from one's package manager (e.g. `pacman -Sy mathjax`) and to copy the result in one's account: `cp -r /usr/share/mathjax ~/Software/`, before fixing permissions there. This local copy shall just be regularly updated.

If Needing to Generate Images from Formulas This is typically generating an image file file from a LaTeX formula to include in a presentation, an e-mail, any kind of post, etc.

LaTeX to SVG

tex2svg is the tool of choice here.

On Arch Linux, the `texlive-bin` (for *pdflatex*) and `pdf2svg` packages may have to be installed.

LaTeX to PNG

We do not think this is the best approach as the resulting bitmap file is likely to have issues in terms of rendering/aliasing.

This can be done thanks to tex2png, a simple yet effective Bash script.

At least the `texlive-fontsrecommended` Arch package shall be installed beforehand, so that the `lmodern.sty` file is available.

Example of use: `./tex2png -c "$\phi_n(\kappa) = \frac{1}{4\pi^2\kappa^2}$" -T -D 500 -o my-example.png`.

Usage

The list of TeX/LaTeX commands supported by MathJax may be of use.

Each LaTeX command may either be specified directly inline, in the text (with `:math:‘LATEX_CMD‘`) or in a block indented after a `.. math::` directive.

This allows to define inline mathematical elements, like $P = \begin{pmatrix} 10 \\ 45 \end{pmatrix}$ (obtained with `P = \\begin{pmatrix} 10 \\\\ 45\\end{pmatrix}`) or standalone ones, like:

$$M = \begin{bmatrix} a11 & a12 & ... & a1n \\ a21 & a22 & ... & a2n \\ ... & ... & ... & ... \\ am1 & am2 & ... & amn \end{bmatrix}$$

obtained thanks to:

```
M = \begin{bmatrix}
        a11 & a12 & ... & a1n \\
        a21 & a22 & ... & a2n \\
        ... & ... & ... & ... \\
        am1 & am2 & ... & amn \\
    \end{bmatrix}
```

For $\phi : R \rightarrow]0,1[$ (i.e. `\phi: \mathbb{R} \rightarrow ]0,1[`), we may have $P_e = \phi(m + \phi^{-1}(P_n))$ is `P_e = \phi(m+\phi^{-1}(P_n))`.

If $\phi(x) = e^x/(1+e^x)$ (translating to `\phi(x) = e^{x}/(1+e^{x})`), then:

$$P_e = \frac{P_n.e^m}{1 + P_n.(e^m - 1)}$$

(translating to `P_e = \frac{P_n.e^{m}}{1 + P_n.(e^{m}-1)}`)

A few other examples of resulting math-related outputs can be seen in this section.

See the next section for a proper use of MathJax in webservers.

Other LaTeX elements that may be convenient:

- multiplying: use `A \times B` for $A \times B$, and `A \cdot B` for $A \cdot B$
- Greek letters: use `\alpha` for α (lowercase), and `\Gamma` for Γ (uppercase)
- figures, like with:

```
.. figure:: static/xx.png
    :align: center

    Some relevant caption.
```

Title Hierarchy It must be consistent: a given type of subtitle must always be placed at the same level in the title hierarchy.

We rely on the markup conventions exposed in this demonstration file (created by David Goodger), whose source is here.

From the top-level title to the most nested ones:

- `=`, on top and below the title (document title)
- `-`, on top and below the title (document subtitle)
- `=`, below the title (H1)
- `-`, below the title (H2)
- `.` , below the title (H3)
- `_`, below the title (H4)
- `*`, below the title (H5)
- `:`, below the title (H6)
- `+`, below the title (H7)

Image Sizes Responsive images, i.e. images that automatically adjust to fit the size of the screen, can be used. They are then defined for example thanks to:

```
<img src="foobar.png" id="responsive-image-large"></img>
```

Various standard sizes have been defined, all prefixed with `responsive-image-`; from the biggest (95%) to the smallest (10%), as defined for example in myriad.css, they are: `full`, `large`, `intermediate`, `medium`, `reduced`, `small`, `tiny`, `xsmall`.

Multi-File Documents

Targeting a Standalone Document Although they tend to be less convenient to edit, longer documents may be split in a **set of RST source files** (the Myriad documentation is an example of it; the WOOPER documentation is an example of the opposite approach, based on a single source file).

Targeting Interlinked Modular Documents In some cases, at least for the HTML output, the need is not to produce a single, large, monolithic document, but a set of interlinked ones (the present HOWTO is an example thereof) that can be browsed as separate pages.

Then a convenient approach is to define different entry points for different output formats, like, for these HOWTOs, this one for the HTML output and this one for the PDF output.

Inner Links Defining any title (e.g. the `"Rendering Mathematical Elements"` one above) automatically introduces in turn a corresponding anchor, which, for the HTML output, can then be referenced from any page, for example as raw HTML (like MyPage.html#rendering-mathematical-elements, or directly from the current page as #rendering-mathematical-elements) or directly through RST in the document (e.g. specified as `‘Rendering Mathematical Elements‘_`, resulting in: Rendering Mathematical Elements).

Note the light transformation (spaces becoming dashes) of the specified name once a it is translated into a legit HTML anchor.

Extra local anchors (e.g. that could be named `"how to render equations"`) can also be specified anywhere in the document (e.g. just before the previously mentioned title, so that it can be designated with other words), thanks to:

```
.. _‘how to render equations‘:
```

It can then be referenced from the same page as #how-to-render-equations or from another one as MyPage.html#how-to-render-equations.

Note that titles and hypertext links introduce local links as well, so one's inner links may clash with them (resulting in `(ERROR/3) Duplicate target name [...]`); the best option is generally to phrase these inner links differently.

Citations & References

Choice of Conventions Engineering generally relies often on the IEEE or the APA citation style.

We dislike a bit IEEE, as its references are just numbers (e.g. `[1]` in `Lindberg and Lee [1]`), instead of more informative elements (like `[CIT2002]`), so we chose APA, which seems more in-line with RST conventions.

Within APA, we prefer parenthetical citations (e.g. `(Salas & D'Agostino, 2020)`) to narrative ones (e.g. `Salas and D'Agostino (2020)`).

Here are extra examples thereof:

- the in-text reference pointers shall be, for books, like `(Author, Year)`; for example `(Taylor, 2005)`
- each of these pointers designates an actual reference in the form:

Authors' Last name, First Initial. (Year). Book title: Subtitle. (Edition) [if other than the 1st]. Publisher

Unfortunately neither types of APA citations is supported (possibly because of the parentheses and/or the space, the text is not interpreted as a citation. So for example `[(Taylor, 2005)]_` / `.. [(Taylor, 2005)]` will not work.

So we finally retained conventions that are a bit different: **APA with no parentheses or space**.

Example Finally an actual example just follows: like explained in [Taylor2005], the conservation of momentum can help solving elegantly some problems.

Commenting To comment-out a block of text, just add `..` at the beginning of a line, then, from the next line, put that block, indented of at least one space; this must be a legit block (see Defining Blocks).

Defining Blocks All lines of a block shall start with the same whitespace. So, whenever a given block is not left-justified (meaning that at least one of its lines starts with a different offset), prefer having all lines of such a block be indented of (at least) 4 spaces (i.e. a tabulation).

Otherwise, if using a single space to indent, as soon as a line of the block is to start with 3 spaces, whitespace-cleanup operations will combine them with the first one to form a tabulation (4 spaces in a row), and all lines of the commented block will not start with the same whitespaces, which could result, from the point of view of RST tools, in an invalid block.

To indent on Emacs, one may select the region of interest and then hit `C-x TAB TAB TAB [...]` (or even just `TAB` once the block is selected).

Either a standard or a code indented block may be used.

A **standard block** is introduced by a non-indented text finishing with two colons (`::`), like in: `Here is what she said::`.

> **Note**
> Unfortunately this does not allow proper French syntax, for which a space is needed before the colons (e.g. typed as `Elle a dit ensuite ::`): adding such a space will result in no colon to be displayed.
> So we stick to writing an improper `Elle a dit ensuite::`, which is rendered as `Elle a dit ensuite:`, better than `Elle a dit ensuite` but worse than `Elle a dit ensuite :`.

Before a **code block** (e.g. introduced with `.. code:: erlang`), a single colon should be used, not two of them. For example:

```
This algorithm can be:

.. code:: erlang
   [...]
```

[Taylor2005] Taylor, J, (2005), Classical Mechanics, (2005), University Science Book, 98-100

Fixing `WARNING: Duplicate explicit target name: "some link"` This typically happens whenever defining a link with a given label more than once.

A solution is to use anonymous reference instead, i.e. *double* underscores (so: `__`) to define references, like in:

```
Here is ‘some link <http://example.org/xxx>‘__.
```

Using double underscores for links could be considered the norm.

Our Preferred More Heavy-Duty Approach

Principle

It applies to more ambitious projects, involving larger content with potentially many interlinked pages.

It is based on the Sphinx toolchain, and therefore shares many elements with our lightweight approach above, starting from the RST syntax.

It offers out of the box many useful mechanisms beyond the generation of a single-page website, from a smooth navigation between a set of pages based on foldable menus to a generated index and a local search engine. It moreover can be easily customised.

More types of outputs are readily supported: HTML, PDF, EPUB, man pages.

Installation

One may follow these guidelines. On Arch Linux, we install the python-sphinx package, thanks to: `pacman -Sy python-sphinx gnu-free-fonts texlive-binextra` (last package being needed now for `latexmk`, to generate PDFs).

Configuration

Running `sphinx-quickstart` is one's best route; in terms of choices:

- we strongly recommend separating the source and build directories
- the "Project name" shall not include "documentation", as this word will be added automatically wherever needed

Then `make html` should generate a base website (including makefiles) that can be browsed in the `build/html` directory (use `make clean` to force its erasure; run just `make` to list all targets of interest, including the `linkcheck` one, to check all external links for integrity).

The project settings can be edited in `source/conf.py`.

Theme Selection

The default (HTML) theme is the Alabaster one. Other themes may be preferred, whether they are Sphinx built-ins or external ones. Many can be customised.

As for us, we prefer mobile-compliant themes with a left column to navigate. This includes the popular Read the Docs theme, which will use here, but also the classic theme.

The documentation of the Read the Docs theme details everything needed.
For an installation thereof on Arch, the simplest is to run `pacman -Sy python-sphinx_rtd_theme`.

Then it is just a matter of editing `source/conf.py` so that `'sphinx_rtd_theme'` is listed in the `extensions` list, and that the `html_theme` is now set to `'sphinx_rtd_theme'`.

Running `make html` again should be sufficient to take this new theme into account. The generated result is quite satisfying.

Here is one customisation thereof (theme options) that we like:

```
#html_theme = 'alabaster'
html_theme = 'sphinx_rtd_theme'

html_theme_options = {
   #'analytics_id': 'G-XXXXXXXXXX',  #  Provided by Google in your dashboard
   #'analytics_anonymize_ip': True,
   'logo_only': False,
   'display_version': True,
   'prev_next_buttons_location': 'both',
   'style_external_links': True,
   'vcs_pageview_mode': '',
   #'style_nav_header_background': 'white',
   #'style_nav_header_background': '#2980B9',
   'style_nav_header_background': 'black',
   # Toc options
   'collapse_navigation': False,
   'sticky_navigation': True,
   'navigation_depth': 4,
   'includehidden': True,
   'titles_only': False
}

html_logo = '../foobar-title.png'
html_favicon =  '../foobar-icon.png'
html_last_updated_fmt = ''
html_copy_source = False
html_show_sourcelink = False
html_show_sphinx = False

html_static_path = ['_static']

# So that inner cross-RST file references can be found:
default_role = 'any'
```

Adding Content

This is just a matter of adding `*.rst` files (each defining at least one title) in the `source` directory, and to reference them in a least one table of contents (e.g. in `source/index.rst`), like in:

```
.. toctree::
  :maxdepth: 2
  :caption: Contents:
```

```
./foobar.rst
./buz.rst
```

The result in a static website that can safely be transferred and served by any webserver of choice.

Inner Links with Sphinx

We recommend allowing referencing sections based on their title (each title becoming a possible link target). This is done by adding, among the `extensions`, the `'sphinx.ext.autosectionlabel'` one. Adding in turn, among the `suppress_warnings`, the `'autosectionlabel.*'` one is then useful to silence messages like `WARNING: duplicate label foo, other instance in bar.rst`.

Then, in our preferred approach for inner links, described here and there, such links - which are possibly defined and referenced in *different* RST files - are to be managed in a slightly different way than for Docutils's inner links; indeed in order to be able to reference:

- label sections (i.e. titles in the document):
  - first, as shown in the conf.py example above, the `default_role = 'any'` setting should be specified
  - then, each inner link target (e.g. a `Brief Answers` title) shall be referenced like with Docutils but *without* a trailing underscore, e.g. as `‘brief answers‘` (rather than as `‘brief answers‘_`); optionally such references may be prefixed with `:ref:`, like in: `:ref:‘brief answers‘`; if wanting to have a specific text displayed for a reference, just use for example `‘my text <brief answers>‘` (or `:ref:‘my text <brief answers>‘`)
- custom anchors:
  - one must be defined with: `.. _my anchor:` or, more classically, as `.. _‘my anchor‘:`
  - then, like for label sections, be referenced as: `‘this is the text of the link <my anchor>‘`; optionally such links may be prefixed with `:ref:`, like in: `:ref:‘this is the text of the link <my anchor>‘`; at least in our tests, using just `:ref:‘my anchor‘` (i.e. not indicating a specific text for the link) would not work (and would result in `WARNING: Failed to create a cross reference. A title or caption not found: 'my anchor'`)

The Problem of Nested Includes

We would have liked to organise a Sphinx document according to a filesystem tree, so that there is in each directory a RST file that comprises the content for that level and that just lists its direct local children as the RST files in its subdirectories, as relative files, like, in a `a/b/c/c.rst` file: `.. include:: d/d.rst`.

Strangely enough, it worked for 3-level nesting (`a`, `a/b` and `a/b/c`), but not for the next level: even though the RST files in `c` where included as `d/d.rst`, they were never found:

```
a/b/c/c.rst:4: CRITICAL: Problems with "include" directive path:
InputError: [Errno 2] No such file or directory: 'a/b/d/d.rst'.
```

(we can see that `c` is lacking; no way of adding it once; and trying there to specify `.. include:: c/d/d.rst` results in `a/b/c/c/d/d.rst` not being found...)

The only (unsatisfactory) solution we found it is specify paths that are "absolute" (i.e. relative to the root of the tree), for example as `.. include:: /a/b/c/d/d.rst`.

Miscellaneous

These hints apply more generically than only with a RST toolchain.

Validating / Checking HTML Content

In addition to the verification of the messages reported when the document is built, some tools allow to perform some checks on a generated document.

Notably an online HTML page, or set of pages, can be verified by third-party tools like this one, to detect dead links.

Fixing Permissions in Third-Party Content to Integrate in a Web Root

The objective is to ensure that a filesystem tree can be transferred as a whole without permission errors to a given server more than once, whereas the destination user and group (typically specialised and restricted on a server) differ from the source ones.

It is indeed often necessary to fix permissions in a third-party tree before it is transferred to a server (e.g. MathJax being copied from a client host through `scp` in a web root on a given server); otherwise the next transfers will stumble on the initial, inadequate group rights, typically preventing them to be overwritten by a process belonging to a different user yet being in the same group, like `700` instead of `770` - resulting in `Permission denied` errors.

For that we recommend executing our fix-www-metadata.sh script in the *source* root prior to transfer it to a webroot of choice. This typically applies to MathJax (see the Rendering Mathematical Elements section), which therefore should not be installed in the system tree thanks to a package manager (as our script will alter its permissions) but in the user account (e.g. as `~/Software/mathjax`).

So typically this script shall be symlinked in each third-party root of interest, and be executed there at each update thereof.

However doing so does not solve all issues: the file entries created/updated by `scp` on the server will be owned by the user on the server implied by the `scp` command, for example `stallone:users` - not the desired `web-srv-user:web-srv-group`. Of course the `fix-www-metadata.sh` script can be run (by root) on the server to correct that. Yet the next update of this webroot will fail again with `Permission`

`denied` errors, as the groups are not expected to match anymore (we cannot overwrite with our client-side `users` group a remote directory whose permission is `770` that is owned by group `web-srv-group`).

A solution is to ensure that the source content bears already the target group. As `scp` relies on user/group IDs, not on names (e.g. on a numerical GID like `1001`, not a name like `users` or `web-srv-group`), the simplest solution is to determine the actual GID of the target group on the server (e.g. running, as root, `grep web-srv-group /etc/group` may tell us that the GID of `web-srv-group` is `1002` there) and to create *on the client* a group with the same GID (if ever possible - that is if there is not already another group happening to have be set to that GID) and to apply it to the source content to transfer, like in:

```
# We are on the "client", source host, as root:
# (web-srv-group-of-target-server clearer than web-srv-group)
$ groupadd --gid 1002 web-srv-group-of-target-server
$ usermod -a -G web-srv-group-of-target-server stallone
$ chgrp -R web-srv-group-of-target-server /home/stallone/mathjax
```

Then, afterwards, when the `stallone` user performs his `scp` repeatedly to transfer updated versions of his `mathjax` directory from the client to the server, he should be able to perform a flawless update of its files and directories.

Pointing to a Specific Moment in a Linked Video

It is as simple as designating, in an HTML link, the targeted second by suffixing the URL video filename with `#t=DURATION_IN_SECONDS`, like in `some-video.mp4#t=1473`[40].

Conversion between Markup Formats

Pandoc is the tool of choice for such operations, as it often yields good results.

For example, in order to convert a page written in Mediawiki syntax, whose source content has been pasted in a `old-content-in-mediawiki.txt` file, into one that be specified in a GitLab wiki (hence in GFM markup, for *GitLab Flavored Markdown*) from a converted content, to be written in a `converted-content.gfm` file, one may use:

```
$ pandoc old-content-in-mediawiki.txt --from=mediawiki --to=gfm --standalone -o conver

# Or, for older versions of pandoc not supporting a gfm writer:
$ pandoc old-content-in-mediawiki.txt --from=mediawiki --to=markdown_github --standalo
```

Then the content in `converted-content.gfm` file can be pasted in the target GitLab wiki page.

Another example is the conversion of a GitLab wiki page into a RST document (e.g. then for a PDF generation):

[40] With `mplayer`, use the `o` hotkey to display elapsed durations.

```
$ pandoc my-gitlab-wiki-extract.gfm --from=gfm --to=rst --standalone -o my-converted-d

# Or, for older versions of pandoc not supporting a gfm reader:
$ pandoc my-gitlab-wiki-extract.gfm --from=markdown_github --to=rst --standalone -o my
```

Finally, if really needing to generate a Word document, an example may be:

```
$ pandoc my-document.rst --from=rst --to=docx -o my-converted-document.docx
```

The lists of the input and output formats supported by Pandoc and of their corresponding command-line options is specified here.

These options are also returned by: `pandoc --list-input-formats` and `pandoc --list-output-formats` (or, for older versions of pandoc, thanks to `pandoc --help`).

An input file may not be encoded in UTF-8, which can result in:

```
pandoc: Cannot decode byte '\xe9': Data.Text.Internal.Encoding.Fusion.streamUtf8: Inva
```

In this case, the actual encoding shall be determined, for example with:

```
$ file input.html
input.html: HTML document, ISO-8859 text
```

Then the encoding may be changed before calling pandoc, for example like:

```
$ iconv -f ISO-8859-1 -t utf-8 input.html | pandoc --from=html --to=markdown_github --
```

Transformation of PDF files

For that, one may use the `pdftk` tool, possibly with the `convert` one, which comes from ImageMagick (typically available thanks to a `imagemagick` package):

- to **split all pages of a PDF** in as many individual files (named `pg_0001.pdf`, `pg_0002.pdf`, etc.): `pdftk document.pdf burst`
- to **extract a range of pages from a PDF**: `pdftk original-document.pdf cat 276-313 output my-extract.pdf`
- to **convert a PDF file** (typically a single page) into a **PNG** one (typically in order to edit the PNG with The Gimp afterwards): `convert pg_000x.pdf pg_000x.png`
- to convert (possibly back) a **PNG file to a PDF** one: `convert pg_000x-modified.png pg_000x-modified.pdf`
- to **concatenate PDFs**: `pdftk 1.pdf 2.pdf 3.pdf cat output 123.pdf`

PDF documents may contain images/scans (possibly of texts) and/or actual, raw texts. If a PDF is a scan, OCR (Optical character recognition) can be used in order to convert the embedded scans into their actual text. Such a transformation can be done online, and we found PDF24 very useful for that. From such services, usually a PDF (thus including text instead of images) is generated. To obtain a text version thereof respecting its layout (typically to preserve the indentation of a scanned program), one may use: `pdftotext -layout my-OCRed-document.pdf` in order to enjoy a proper `my-OCRed-document.txt`.

Image Transformations

One may rely on:

- GIMP (*GNU Image Manipulation Program*; corresponding, on Arch, to the `gimp` package), for bitmap (e.g. PNG) graphics
- Inkscape, for vector-based (e.g. SVG) graphics
- or on command-line ImageMagick (on Arch Linux, install the `imagemagick` package, which provides notably the `convert` and `display` executables)

Inverting an Image To invert/negate an image (swap colors with their complementary ones, while preserving alpha coordinates):

```
$ convert source.png -channel RGB -negate target.png
```

See also the Myriad's automatic rules, which generate `X-negated.png` from `X.png` thanks to: `make X-negated.png`.

Rendering a Vector Image at a given Scale/Size Let's suppose that a SVG file is available (for example obtained thanks to our latex-to-image.sh script).

Going for a PNG of a width of 1000 pixels while selecting a high-enough DPI, preserving the aspect ratio and keeping the background transparent:

```
$ convert -density 1200 -size 1000 -background none latex-formula.svg target.png
```

Setting a background to a solid color (e.g. `white`) may allow, when adding a border, to have its color applied only on that border (rather than on the full background).

Adding a Border to an Image For example to add a 10-pixel wide / 5-pixel tall red border to an image:

```
$ magick source.png -bordercolor red -border 10x5 target.png
```

A more classical 2-pixel thick black border:

```
$ magick source.png -bordercolor black -border 2 target.png
```

Compositing/Blitting an Image onto Another Let's suppose we have an overall, larger image (e.g. `my-overall-plot.png`) onto which we want to composite / blit a smaller one (e.g. `my-formula.png`) at position (100,150) - in pixels, relatively to the top-left corner - with no specific scaling:

```
$ magick composite my-formula.png my-overall-plot.png -geometry +100+150 target.png
```

Positioning the inner image based on "gravity" (preset areas, based on various possible origins; see `magick -list gravity`) is often convenient; for example relatively to the bottom-right corner of the final image (knowing that positive axes are then, for "SouthEast", the opposite of the default ones - positive coordinate offsets have therefore to be specified in order to go towards the center):

```
$ magick composite my-formula.png my-overall-plot.png -gravity SouthEast -geometry +50
```

Or simply to have the inner image centered into the overall one:

```
$ magick composite my-formula.png my-overall-plot.png -gravity Center target.png
```

See also our affix-images.sh script.

Plot Generation

Refer to our data display section.

UML Diagrams

If SysML can also be of interest, we focus here on UML2 class diagrams (one of the 14 types of diagrams provided by UML2).

Quick UML Cheat Sheet

Multiplicities A **multiplicity** is a definition of *cardinality* (i.e. number of elements) of some collection of elements.

It can be set for attributes, operations, and associations in a class diagram, and for associations in a use case diagram. The multiplicity is an indication of how many objects may participate in the given relationship.

It is defined as an inclusive interval based on non-negative integers, with `*` denoting an unlimited upper bound (not, for example, `n`).

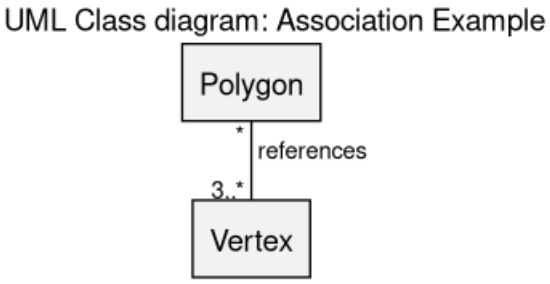
Most common multiplicities are:

- no instance or one instance: `0..1`
- any number of instances, including zero: `*` (shorthand for `0..*`)
- exactly k instances: `k` (so, if k=5, `5`)
- at least M instances: `M..*` (`2..*`)
- at least M instances, but no more than N (hence bounds included): `M..N` (e.g. `3..5`)

For associations, the default multiplicity is automatically is `0..1`, while new attributes and operations have a default multiplicity of `1`.

Association An **association** is a relation between two classes (*binary association*) or more (*N-ary association*) that describes structural relationships between their instances.

For example a polygon may be defined from at least 3 vertices that it would reference, whereas a point may take part to any number of polygones (including none):



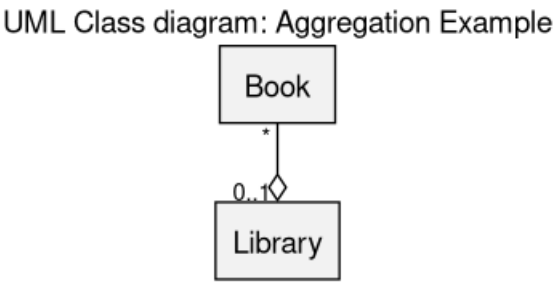
(see the sources of this diagram)

The **multiplicity** of an endpoint denotes the number of instances of the corresponding class that may take part to this association. For example, at least 3 points are needed to form a polygon, whereas any number of polygons can include a given point.

In UML the **direction** of the association is easily ambiguous (here we have to rely on external knwoledge to determine whether a polygon is composed of points, of if a point is composed of polygons). Adding a chevron (like > or <, e.g. `"references >"` ; ideally this should be a small solid triangle) to the text is not a good solution either, as the layout may place the respective endpoints in any relative position. Adding an arrow to the end of the line segment cannot be done either, as it would denote the *navigability* of the association instead.

Aggregation An **aggregation** is a specific association that denotes that an instance of a class (e.g. `Library`) is to loosely contain instances of another class (e.g. `Book`), in the sense that the lifecycle of the contained classes is not strongly dependent on the one of the container (e.g. books will still exist even if the library is dismantled).

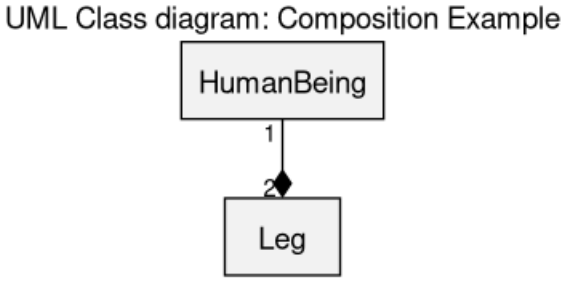
Here a library may contain any number of books (possibly none), and a given book belongs to at most one library.



(see the sources of this diagram)

Composition A **composition** is a specific association that denotes that an instance of a class (e.g. `HumanBeing`) is to own instances of another class (e.g. `Leg`), in the sense that the lifecycle of the contained classes fully depends on the one of the container (here, if a human being dies, his/her legs will not exist anymore either).

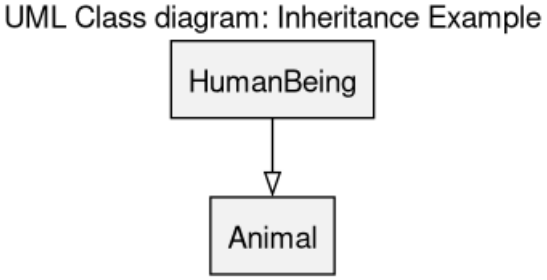
Here a human being has exactly 2 legs, and any given leg belongs to exactly one human being (therefore this model does not account for one-legged persons).



(see the sources of this diagram)

Inheritance An **inheritance** relationship is a specific association that denotes that a class (e.g. `HumanBeing`) is a specific case of a more general one (e.g. `Animal`), and thus that an instance of the first one is also an instance of the second one ("is-a" relationship).

Here a human being is a specific animal.



(see the sources of this diagram)

Tooling In a design phase, one may prefer lightweight tools like Graphviz, PlantUML or even Dia.

As long as the architecture of a framework is not stabilised, having one's tool determine by itself the layout of the rendering (rather than having to place manually one's graphical components) is surely preferable.

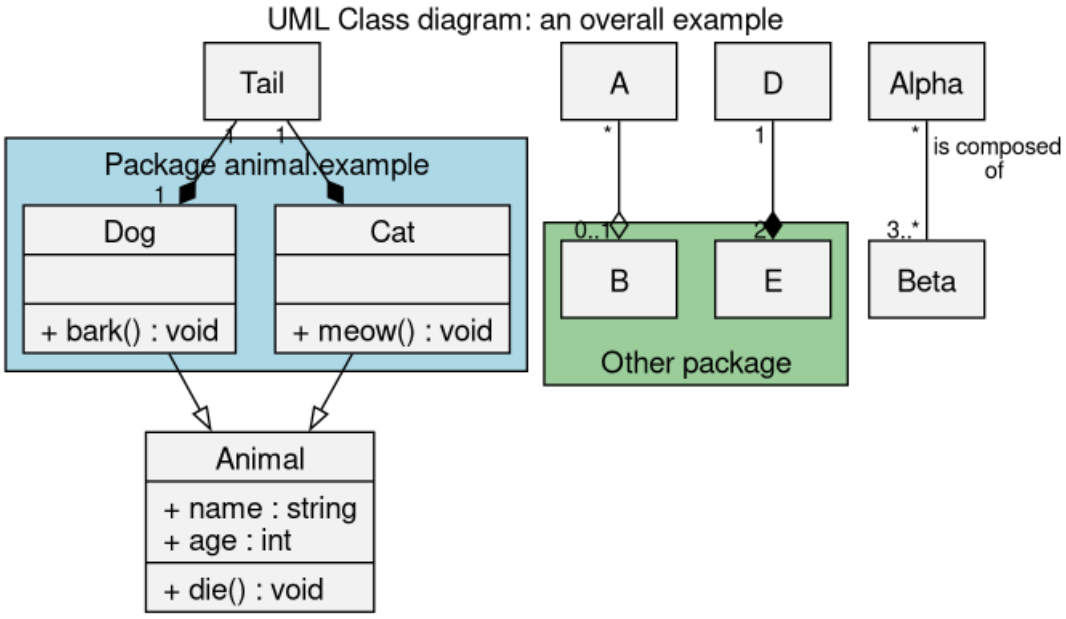
For that we use Graphviz, with our own build conventions.

For example, supposing this diagram example, i.e. a source file named `uml_class_diagram_example.graph`:

```
$ make uml_class_diagram_example.png
# or, to force a regeneration and a displaying of the result:
```

```
$ make clean uml_class_diagram_example.png VIEW_GRAPH=true
```

This example results in the following diagram:



(see the sources of this diagram)

Finding Usable Content

Assets can be found thanks to Creative Commons, which references, among others, OpenClipart, otherwise possibly publicdomainvectors.org.

The simplest approach is to rely on SVG files, to edit them on Inkscape (select any subset of interest of the image), and possibly to export a selection of them as PNG files of the desired size (in the `Export` pane, enable the *Export Selected only* option and, in the *Image Size* tab, set the absolute width and/or height wanted).

In order to find the icons needed to devise a GUI from open elements, refer to the tremendously useful iconify.design website.

Using Additional Fonts

One may use websites like Dafont in order to select, based on appearance and licence, a given TTF font.

At least on Arch, it is sufficient to copy the corresponding downloaded TTF file (as root) in `/usr/share/fonts/` so that tools like The Gimp support it right afterwards (e.g. no need to run `fc-cache` beforehand).

Data Management

Organisation: Copyright (C) 2022-2025 Olivier Boudeville

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

Creation date: Saturday, November 20, 2021

Lastly updated: Wednesday, March 19, 2025

Table of Contents

Overview

This section concentrates information about **data management**, including data **formats** and data **processing tools**.

General-Purpose Data Format

Such a format is typically useful to hold configuration information.

We prefer JSON to, for example, YAML, due to the Python-style indentation on which the latter relies in order to indicate nesting.

Language-Independent Data Formats

JSON A JSON document is in plain-text and may contain:

- basic types:
  - Number: `2` or `4.1`
  - String: `"I am a string"`
  - Boolean: `true` or `false`
  - `null`: to denote an empty value
- attribute-value pairs (e.g. `"firstName": "John"`)
- "arrays" (ordered lists), e.g. `"myNumbers": ["12", "7", "4"]`
- "objects" (collection of name-value pairs), e.g.

```
{

  "address": {
     "streetAddress": "21 2nd Street",
     "city": "New York"
  }

}
```

The order in arrays is expected to be preserved, but not the one of the elements in an object.

Defining an element (e.g. an attribute-value pair) more than once is allowed, and the last instance thereof will be the one kept.

For instance:

```
{
 "tcp_port": 8084,
 "tcp_port": 8085,
 [...]
}
```

Here, once the document is parsed, `tcp_port` will be considered equal to `8085`.

Pretty-Printing On GNU/Linux, one may rely on `jq`, a command-line JSON processor.

For instance: `jq . my_document.json`.

Validating One may consider that a given document is a legit JSON one iff `jq type` reports a non-empty output.

Example:

```
$ jq type my_document.json
"object"
```

Example Regarding syntax, a typical JSON document is:

```
{
  "firstName": "John",
  "__comment": "This is a comment!",
  "lastName": "Smith",
  "isAlive": true,
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
```

```
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [],
  "spouse": null
}
```

Specifying comments With JSON, there is, on purpose, no built-in way to add comments.

The sole solution/workaround is to add comments as specific fields, although they will end up as data like the other fields.

We recommend to mark them specifically (e.g. as `__comment`) so that they should not interfere with the "real" data. As an example, see the second key of the previous JSON document.

YAML YAML is a data serialization language for all programming languages.

We prefer the `.yaml` extension to the `.yml` one.

No tabulation should be used for indentation, only spaces, and preferably a fixed amount of them; we used to prefer 4, now 2, since it allows to properly align the items listed with a dash (e.g. `"- I am an item"`).

With Emacs, the Yaml Mode may be of help.

Erlang-Friendly Data Format: ETF

Such a format is typically useful to hold configuration information in an Erlang context.

We recommend the use of ETF (the *Erlang Term Format*), that we find particularly useful and even more suitable than JSON (entry order preserved, comments supported, etc.).

Data-related Processing Tools

Whenever needing to perform **numerical operations** on data, we recommend the use of Scilab or GNU Octave, which are the two major open-source alternatives to MATLAB.

As such, the three of them support mostly the same syntax (even if Scilab puts less emphasis on syntactic compatibility with MATLAB than Octave does).

From now on, the specific tool being used among the two MATLAB alternatives will not be specifically mentioned (mentioning "the tool" instead). We tried both alternatives and had more issues (build/installation, proper display) with Scilab, so we mostly used Octave, but did not find it very user-friendly.

> **Note**
> A personal consideration: for basic uses, relying on a generic-purpose programming language remains often vastly more convenient.

Common Hints About Scilab and Octave

Syntax Putting a semicolon at the end of statement prevents the console from printing the corresponding value (answer, variable assignment, etc.).

Value Ranges Ranges can be expressed thanks to:

- `MIN:STEP_SIZE:MAX`, like in: `x = -10:2:10` resulting in 11 points
- or `linspace(MIN, MAX, STEP_COUNT)`, like in: `x = linspace(-10, 10, 11)` to produce the same list as before

Array-based Functions Many functions are defined so that they can be called also with *arrays* of parameters, instead of just standalone values. For that, a dot/period-based syntax has been introduced so that each element of an input vector is applied to the function in turn. A key understanding is that such a dot applies not to variables but to operators; for example `2*x.^2+1` shall not be read as `2*(x.)^2+1` but as `2*x(.^)2+1`.

As an example, a ϕ function can be defined[41] as $\phi(x) = e^x/(1+e^x)$:

```
function retval = phi (x)
 retval = exp(x)./(1+exp(x));
endfunction
```

Then this function can be called either with a standalone value:

```
phi(1)
ans = 0.7311
```

or with an array thereof (a vector):

```
phi([1,2])
ans =
  0.7311   0.8808
```

Outputs Beware to the lower-precision of textual outputs, which may be misleading (e.g. use `format long` with Octave to request 15 significant figures).

Script Files Rather than being directly interpreted (e.g. from a pasted text), a series of statements can be gathered in a script file (a bit different from a function file[42]) that can be loaded and executed afterwards.

[41] Note that, for basic operation like additions, the `.+` operator has been deprecated in favor of just `+`.

[42] A so-called "function file" is one that starts with a function definition and that must only be called from a "script file".

Their conventional extension in `m` (e.g. `foobar.m`), due to the MATLAB legacy.

Avoid dashes in the script filenames, as it may be interpreted as `minus`; prefer underscores (e.g. `phi_exp.m` rather than `phi-exp.m`).

An example of script, named `phi_exp.m`:

```
# An initial comment prevents Octave from thinking that this
# is a function file:
1;

# The function name can be freely chosen, it does not have to
# correspond to the script filename:
#
function retval = phi_exp (x)
   retval = exp(x) ./ (1+exp(x))
endfunction

xs = -10:10
ys = phi_exp(xs)

plot(xs,ys)

title ("A function phi to map values V to a probability-suitable ]0,1[ interval")
ylabel ("\phi(V)")
xlabel ("V")
grid on
```

Provided that such a script is located in a well-known directory of the tool (typically its working directory), it can be executed simply by entering its filename without extension; for example:

```
octave:1> phi_exp
warning: function 'phi_exp' defined within script file 'xxx/phi_exp.m'
xs =
 -10   -9   -8   [...]
```

Then a script can be run from the command-line; for example:

```
$ octave --eval phi_exp.m --persist
# OR
$ octave phi_exp.m
```

Using Scilab

Scilab may be best obtained, on Arch Linux, from the AUR (e.g. `yay -Sy scilab`) by selecting the `scilab-bin` option (relying then on prebuilt binaries); it can then be run with: `scilab`.

We experienced quite a few issues in order to obtain Scilab by any means. A last-resort option is to rely on an AppImage instead; one may then sandbox it: first the Scilab AppImage shall be downloaded, for example as `~/Scilab-x86_64.AppImage`. Then:

```
$ mkdir -p ~/Software/scilab
$ cd ~/Software/scilab
$ mv ~/Scilab-x86_64.AppImage .
$ chmod +x Scilab-x86_64.AppImage
$ firejail --appimage ./Scilab-x86_64.AppImage &
```

Once installed that way, our run-scilab.sh can be used instead.

Note that the copy/paste behaviour is not consistent with the usual UNIX/X11 one, and that tabulation can be used for auto-completion.

Scilab does not seem to offer any symbolic support out of the box. See Using Maxima instead (knowing that Maxima may be integrated within Scilab).

Defining a Function Let's suppose we want to define $my\_func : x \to 2.x^2 + 1$.

For that, in Scilab's shell, enter:

```
--> function [y] = my_func(x)
  > y = 2*x^2+1
  > endfunction
```

Then:

```
--> my_func(5)
 ans  =
   51.
```

Plotting a Function Let's define the support of our function, here computed from 0 to 10 with 50 values: `my_xs = linspace(0, 10, 50)`. Then just execute `plot(my_xs, my_func)`.

We experienced rendering issues that prevented a proper display of plots.

Using Octave

Octave can be installed on Arch Linux with `pacman -Sy octave`; extra packages may be needed (e.g. `octave-quaternion`, available in the AUR).

The command-line version can be run as `octave`. Typing `quit` at the prompt allows to exit.

The GUI version can be launched with `octave --force-gui`.

Defining a Function As already seen, so that it can operate on single values or arrays, a $\phi(x) = e^x/(1 + e^x)$ function can be defined as:

```
function retval = phi (x)
  retval = exp(x)./(1+exp(x))
endfunction
```

Plotting a Function We consider first a function of a single variable.

Let `my_xs = 0:0.2:10` define[43] the support / display range of our function; then:

```
my_ys = my_func(my_xs)
plot(my_xs, my_ys)
```

A key point is to understand that, for all plots (`plot`, `mesh`, `surf`, etc.), the last element to be specified is *not* the function to which the previous elements are to be applied, but directly the final values to plot.

Extra display settings can be added afterwards:

```
title ("This is my title")
ylabel ("My ordinate label")
xlabel ("My abscissa label")
grid on
```

This results in:

The image can be saved either by using the `Save As` GUI menu and typically selecting PNG, or directly from the console/scripts thanks to the following command: `print("my_plot.png", "-dpng")`.

Using Maxima

Maxima is a free software (GPL) tool for the manipulation of symbolic and numerical expressions.

It can be installed on Arch with `pacman -Sy maxima`.

A graphical frontend exists, considerably more user-friendly, and useful for teaching the use of Maxima: `wxmaxima`; it is available on the Arch AUR, yet its build fails at the time of this writing; it has however an AppImage - just try to find a recent `wxmaxima-x86_64.AppImage` there.

One may then store this image in `~/Software/maxima/` and run it either directly or in a sandboxed environment: `firejail --appimage wxmaxima-x86_64.AppImage`; see also our run-maxima.sh script.

For example, taking into account that (refer to this introduction for more details):

- to assign a value to a variable, use the colon (`:` ; for example: `a : [1,2]`), not the equal sign (that is used for representing equations)
- each statement is to be ended, on:
  - the command-line: with a semi-colon (`;`) followed by `Enter`
  - the GUI: with just `Shift-Enter` (the semi-colon is auto-added; `Enter` is used here for multiline inputs)
- one shall enter `quit();` to exit
- a general matrix `m` may be defined that way:

[43] The range syntax is `Min:StepSize:Max`. As `Min` and `Max` are both included, `my_range = 0:0.2:1` would correspond therefore to a vector of 6 values: `[0, 0.2, 0.4, 0.6, 0.8, 1]`.

Alternatively `linspace` can be used in order to specify the number of points (rather than the step size); for example `linspace(0,5,9)` is a vector of 9 evenly-spaced points between 0 and 5 (both included: `[0, 0.625, 1.25, 1.875, 2.5, 3.125, 3.75, 4.375]`).

```
m: matrix(
 [m11,m12,m13,tx],
 [m21,m22,m23,ty],
 [m31,m32,m33,tz],
 [0,0,0,1]
);
```

- a diagonal matrix `s` may be defined that way:

```
s: matrix(
 [Sx,0,0,0],
 [0,Sy,0,0],
 [0,0,Sz,0],
 [0,0,0,1]
);
```

- a diagonal matrix `t` may be defined that way:

```
t: matrix(
 [1,0,0,trx],
 [0,1,0,try],
 [0,0,1,trz],
 [0,0,0,1]
);
```

- multiplication is represented by a dot : `m.s;`

then Maxima can be directly used in a terminal:

```
$ maxima
;;; Loading #P"/usr/lib/ecl-23.9.9/sb-bsd-sockets.fas"
;;; Loading #P"/usr/lib/ecl-23.9.9/sockets.fas"
Maxima 5.47.0 https://maxima.sourceforge.io
using Lisp ECL 23.9.9
Distributed under the GNU Public License. See the file COPYING.
Dedicated to the memory of William Schelter.
The function bug_report() provides bug reporting information.
[...]
(%i1) m: matrix(
 [m11,m12,m13,tx],
 [m21,m22,m23,ty],
 [m31,m32,m33,tz],
 [0,0,0,1]
);
                              [ m11  m12  m13  tx ]
                              [                   ]
                              [ m21  m22  m23  ty ]
(%o1)                         [                   ]
                              [ m31  m32  m33  tz ]
                              [                   ]
                              [  0    0    0   1  ]
```

```
(%i2) s: matrix(
 [Sx,0,0,0],
 [0,Sy,0,0],
 [0,0,Sz,0],
 [0,0,0,1]
);
                              [ Sx  0   0   0 ]
                              [               ]
                              [ 0   Sy  0   0 ]
(%o2)                         [               ]
                              [ 0   0   Sz  0 ]
                              [               ]
                              [ 0   0   0   1 ]
(%i3) m.s;
                       [ Sx m11  Sy m12  Sz m13  tx ]
                       [                            ]
                       [ Sx m21  Sy m22  Sz m23  ty ]
(%o3)                  [                            ]
                       [ Sx m31  Sy m32  Sz m33  tz ]
                       [                            ]
                       [   0       0       0     1  ]
```

Doing the same this time with wxmaxima:
A session can be saved in a file, using the `*.wxmx` extension.
See also Maxima's documentation.

Using LibreOffice

To obtain a list of all the different values in a selection The objective is to determine the set of the different (unique) values in a selection (typically a whole column).

Once the selection has been defined, select the `Data -> More Filters -> Standard Filter` menu item, set the `Field name` dropdown to `"-none -"` and, in the panel for `Options` settings, enable `No duplications`.

Then each resulting *selected* row while have a unique value in the selected column (the other rows will still exist but not be selected).

Often it might be convenient to sort this selection afterwards; the up/down arrow icons may be used, and keeping the `Current selection` is generally used (rather than extending it).

Data-related Displaying Tools

For that we rely mostly on gnuplot.

Our conventions are the following:

- rely on a recent version of gnuplot (e.g. 5.4)
- the image formats for the generated plots are (large-enough) `PNG` or `SVG` (better in spirit, yet usually of higher file size, and with a slightly different rendering)

- the extension of command files is `p` (e.g. `foobar.p`), the one for data files that they refer to is `dat` (e.g. `foobar.dat`); running gnuplot scripts is then as simple as executing `gnuplot foobar.p`

Besides generating images, gnuplot is able, thanks to interactive terminal types (like `qt`), to let the user navigate in the plots (e.g. move around, zoom on them).

As an example to be copied in a `hyperboloid.p` command file:

```
reset
set grid
set parametric
set view equal

splot [-pi:pi][-2.5:2.5] cos(u)*sinh(v), sin(u)*sinh(v), cosh(v), cos(u)*sinh(v), sin

set terminal qt persist
pause mouse close
```

Then running `gnuplot hyperboloid.p` results in an interactive viewer like:

that we can explore with the mouse and/or keyboard. Press the `h` key to list the available mouse/keyboard commands.

Based on our Erlang developments, we implemented the plot_utils module, which is a library (relying on gnuplot) to generate plots more conveniently.

Emacs

Organisation: Copyright (C) 2025-2025 Olivier Boudeville

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

Creation date: Sunday, March 16, 2025

Lastly updated: Wednesday, March 19, 2025

Table of Contents

Overview

Emacs is a family of free software text editors that are characterised by their extensibility and their ability to be customised at will.

Installation

Quite surprisingly, Emacs still changes a lot (notably in terms of function names), and (Elisp) scripts that work for an Emacs version may not work for the next one. So at least controlling one's version may be of use; run for that `emacs --version`. We consider using here Emacs 30.1 or more recent.

Of course the best option to install Emacs is to use a (OS-level) package manager, for example: `apt-get install emacs` - if the supported version is not too ancient.

Otherwise, to perform a manual installation of Emacs on one's user account, it must be first downloaded; one may thus fetch for example `emacs-30.1.tar.xz`.

Prerequisites may be needed; running - unfortunately as root - `apt-get build-dep emacs` or alike may be of use, or at least having packages like `libgtk-3-dev` and `librust-tree-sitter-dev` installed.

Then:

```
$ mkdir -p ~/Software/Emacs && cd ~/Software/Emacs
$ EMACS_VERSION=30.1
$ mv ~/Downloads/emacs-${EMACS_VERSION}.tar.xz .
$ tar xvJf emacs-${EMACS_VERSION}.tar.xz
$ cd emacs-${EMACS_VERSION}
```

```
# If not having these dependencies:
$ ./configure --with-xpm=ifavailable --with-gif=ifavailable \
--with-tiff=ifavailable --with-gnutls=ifavailable --prefix=${HOME}/Software/Emacs/emac
$ make install
$ cd .. && ln -s emacs-${EMACS_VERSION}-install emacs-current-install
```

Then one's shell environment shall be updated once for all with:

```
$ export PATH="${HOME}/Software/Emacs/emacs-current-install/bin:${PATH}"
```

Configuration

Our base Emacs configuration, init-myriad-base, is designed to be minimal enough not to depend on any third-party element (so it is not relying on a package manager).

It may be used as:

- a simple configuration of its own, for example on servers (hence just on the console, with no GUI), typically once copied as `~/.emacs.d/init.el`
- a fallback configuration, useful when a reliable, unaffected editor is needed to debug one's Emacs configuration (see Relying a safe, minimal, fallback Emacs for that) - a base configuration expanded (i.e. included) by more involved / specialised configurations (e.g. see our main, most complete Emacs configuration for daily usage, init-myriad-fully-integrated)

In addition to these base and complete configurations, we defined modular, specialised configuration files for a few topics.

First, the standalone ones, i.e. the specialisations that do not depend on any specific (third-party) package, are:

- init-myriad-rst-base for the support of ReStructuredText
- init-myriad-erlang-base, for the basic support of Erlang, following the official guidelines (with no extra tooling involved)
- init-myriad-c-cpp-base, for a basic support of C/C++
- init-myriad-python-base, for a basic support of Python

As for the configurations involving packages, we have:

- init-myriad-package-management, to setup a proper management of Emacs packages (currently based on elpaca)
- init-myriad-erlang-advanced, to provide additional tooling on top of the prior `init-myriad-erlang-base` (currently not deemed satisfactory enough, at least for our use cases)

Finally, an optional (loaded iff found available) init-myriad-local (that link pointing to an example version) configuration allows to define per-computer settings (e.g. to define relevant initial window sizes for that host).

All these configuration files are included by the aforementioned init-myriad-fully-integrated one.

For the sake of testing, an instance of Emacs relying on a given configuration file (typically `init.el`) can be best run with: `emacs --init-directory=$SOME_DIR`, where `SOME_DIR` contains this configuration file (possibly as a symbolic link).

Hints

Emacs Parlance

With Emacs:

- a `frame` is actually a regular GUI window
- a `window` is a subdivision of a `frame` (hence to be understood as a *window pane*)
- a `buffer` is the content shown in a `window`
- (the) `point` is the (current) cursor position
- the `mark` is the beginning of the current selection (typically set by `C-SPC` at the current point)
- ELPA stands for *Emacs Lisp Package Archive* (see more package-related explanations):
  - GNU ELPA is ~450 packages, considered part of Emacs
  - NonGNU ELPA contains third-party packages whose copyright has not been assigned to the Free Software Foundation
  - MELPA for *Milkypostman's Emacs Lisp Package Archive*; ~6000 packages that can be freely added

Key Bindings

Here `C` corresponds to the "Ctrl" key, `M` the "Meta" one (i.e. "Alt", generally), `S` to the `Shift` one, and `-` is just a separator between a modifier (like Ctrl, Meta, etc.) and an actual key to press *while the modifier is still held down*. `RET` means the Enter key, `SPC` the spacebar, `M-L_ARROW` / `M-R_ARROW` the left / right arrow keys.

For example `C-x C--` means: press and hold the "Ctrl" key, and hit the "x" key, then release all, then press and hold the "Ctrl" key, and hit the "-" key, and release all.

At least based on our (Emacs and French keyboard) **base** settings (i.e. the ones in `init-base.el`), the following key bindings trigger the corresponding actions:

- `C-a` / `C-e`: move cursor to leftmost / rightmost position
- `C-k`: kill all characters on the right from cursor (and thus put them in the kill ring)

- `C-s PATTERN RET` / `C-r PATTERN RET`: perform direct / reverse case-insensitive search of the specified pattern; then each `C-s` / `C-r` will jump direct / reverse to any next occurrence; `RET` will stop the search
- `C-d SEARCH_PATTERN RET REPLACE_PATTERN RET`: perform replacements, bound to `replace-string` (default binding to delete char at cursor - `delete-char` - not useful enough, as a dedicated key, the Delete one, exists for that)
- `C-l LINE_NUMBER`: go to the specified line
- `C-x C-s`: save current buffer
- `C-x C-k`: kill current buffer
- `C-x C-c`: exit Emacs
- `C-g`: cancel command/selection
- `C-z`: undo last action (or `C-x u`); to redo, rather than typing `C-g C-/` (which would involve a shift, on French keyboards), just perform a non-editing command (such as `C-SPC`) and then the next `C-z` commands will go in the opposite direction in the undo chain (they will redo)
- `C-SPC`: set the mark (begin selection)
- `C-w`: cut selection in the kill ring (selection is thus removed from the current buffer)
- `M-w`: copy selection in the kill ring (selection thus remains in the current buffer)
- `C-y`: yank (paste from kill ring, replacing any current selection)
- `M-y`: replace text just yanked with an earlier batch of killed text
- `C-p`: compile, make
- `C-x TAB`: indent rigidly, moving the whole selection left/right with `M-L_ARROW` / `M-R_ARROW` (see also `kill-rectangle`)
- `M-<` / : `M-S-<`: go to beginning / end of the current buffer
- `C-x 2`: split window horizontally
- `C-x 3`: split window vertically
- `M-R_ARROW` / : `M-L_ARROW`: go to next / previous buffer
- `M-q`: fill current paragraph (reformat / wordwrap it), i.e. applies `fill-paragraph`
- `M-m`: jump to first non-whitespace in the current line
- `M-x` will allow to enter / select commands from the minibuffer (see our useful commands section below)
- `C-o` or `F8`: perform whitespace cleanup (now built-in)

Frequent Actions

- to tune the font size:
  - temporarily: use `C-x C-+` to increase, `C-x C--` to decrease (or press Shift then click the first mouse button to select a relevant option; or, even better, use `C-mousewheel`)
  - permanently: one may add in one's Emacs configuration file for example: `(set-face-attribute 'default nil :height 100)`
- to insert special characters (e.g. tab or newline) as raw characters in commands (e.g. in I-search or replace-string): use for example `M-x quoted-insert <tab>`, which is often bound (yet not in our conventions) to `C-q`; note that pasting a tab with the mouse in the minibuffer works as well
- to insert in the current buffer the output of a shell command: `C-u then M-!`: enter that shell command, whose output will be pasted at the current cursor position
- to prefix all lines of the selected region: `C-x` then `r` then `t`, then type the prefix then type Enter (useful for example to indent a series of lines)
- to sort alphabetically the selected region: `M-x sort-lines`
- to go to the matching delimiter (parenthesis, bracket, end of word, etc.): to go forward, use `C-M-f`, and to go backward use `C-M-b`
- to re-select a region (e.g. to perform multiple substitutions in a row on the same region): `C-x C-x`
- to abort the current entry in the minibuffer: `C-g`
- to perform replacements based on regular expressions: `M-x replace-regexp RET regexp RET newstring RET`, with this REGEXP syntax (more information); for example, to remove, in each line, all characters from the first comma: `M-x replace-regexp RET ,.* RET , RET`
- to open a file that is very large and/or difficult to parse/display: `M-x find-file-literally`

Other **useful commands** to trigger, possibly explicitly with `M-x`:

- `replace-string SEARCH_PATTERN REPLACEMENT`; add `M-c` to set the case-sensitive flag, i.e. to search for the exact string (even if it is lowercase - otherwise uppercase versions thereof will match); the `M-%` default shortcut requires a shift on French keyboards
- `query-replace SEARCH_PATTERN REPLACEMENT`
- `kill-rectangle` (operates on a previous selection)
- `indent-region` (`C-M-\` difficult on French keyboards)

See also our Performing a Merge with Emacs section.

Package Managers

There are at least:

- package.el
- straight.el
- Elpaca: *Async Emacs Package Manager* (current state of the art, and the one on which we currently rely)

`use-package` is not a package manager, it is an extensible configuration macro that is specialised by most of the package managers so that it can be used with any.

Refer to this page for a comparison of package managers.

Elisp Hints

Emacs Lisp is a Lisp dialect made for Emacs.

See the GNU Emacs Lisp Reference Manual.

An Emacs init file contains a series of Lisp expressions, each of them consisting of a function name followed by arguments (expressions), all surrounded by parentheses.

For example: `(setq fill-column 60)` calls the function `setq` (*set quoted*) to set the variable `fill-column` to `60`; as `setq` affects only the current buffer's local value, in an initialisation file `setq-default` is generally preferred.

- boolean values: `t` stands for "true", and `nil` for "false"
- to display a message: `(message "Hello world!")`
- a leading single-quote makes a symbol a constant - otherwise it would be treated as a variable name; for example: `(setq-default major-mode 'text-mode)`

Elisp examples may be the following conditional setting:

```
(if (boundp 'coding-category-utf-8)
   (set-coding-priority '(coding-category-utf-8)))
```

Or the next function definition and key binding:

```
(defun my-split-window-func ()
 (interactive)
 (split-window-below)
 (set-window-buffer (next-window) (other-buffer)))

(global-set-key (kbd "C-x 2") #'my-split-window-func)
```

If needing to include a configuration file in another:

```
(load-file "~/elisp/foo.el")
```

To trigger multiple calls in a single expression, use:

```
(progn do-this
       do-that)

(defun func (arg1 arg2)
   "Always document your functions."
   <function body>)

(defvar var-name <the value>
   "Always document your variables.")
```

To concatenate filesystem elements: `(file-name-concat "/tmp" "foo")` results in `"/tmp/foo"`.

Related Elisp information sources:

- Emergency Elisp
- Learn X in Y minutes Where X=Emacs Lisp
- Emacs Lisp Guide

Troubleshooting

Handling Errors

Sometimes problems arise due to older packages, for example when a new version of Emacs is used. This may be solved by removing the cache of the package manager (e.g. `~/.emacs.d/straight`, `~/.emacs.d/elpaca`), relaunching Emacs and waiting for its state (e.g. all related clones) to be downloaded/built again.

To investigate a problem, one may run Emacs with: `emacs --debug-init`.

Relying a safe, minimal, fallback Emacs

As many software ecosystems, the Emacs one tends to change/break frequently, and then one's `init.el` cannot be edited anymore with a functional Emacs - and the whole situation quickly degenerates in a mess.

One way of overcoming these issues is to have multiple versions of Emacs' configuration, including a very basic one that is never expected to break - just for the purpose of having at all times a base Emacs to fix the others; this is one of the purposes of our rather minimal init-myriad-base configuration, to be used that way:

```
$ DIR="${HOME}/.emacs.d/myriad-fallback"
$ mkdir $DIR && cd $DIR
$ ln -s $CEYLAN_MYRIAD/conf/init-myriad-base.el init.el
```

Then an always-available Emacs may be run with: `emacs --init-directory=$DIR`, which can made easily available thanks to, in one's shell configuration: `alias esafe='emacs --init-directory=${HOME}/.emacs.d/myriad-fallback'` (`esafe` standing for *editor safely available*).

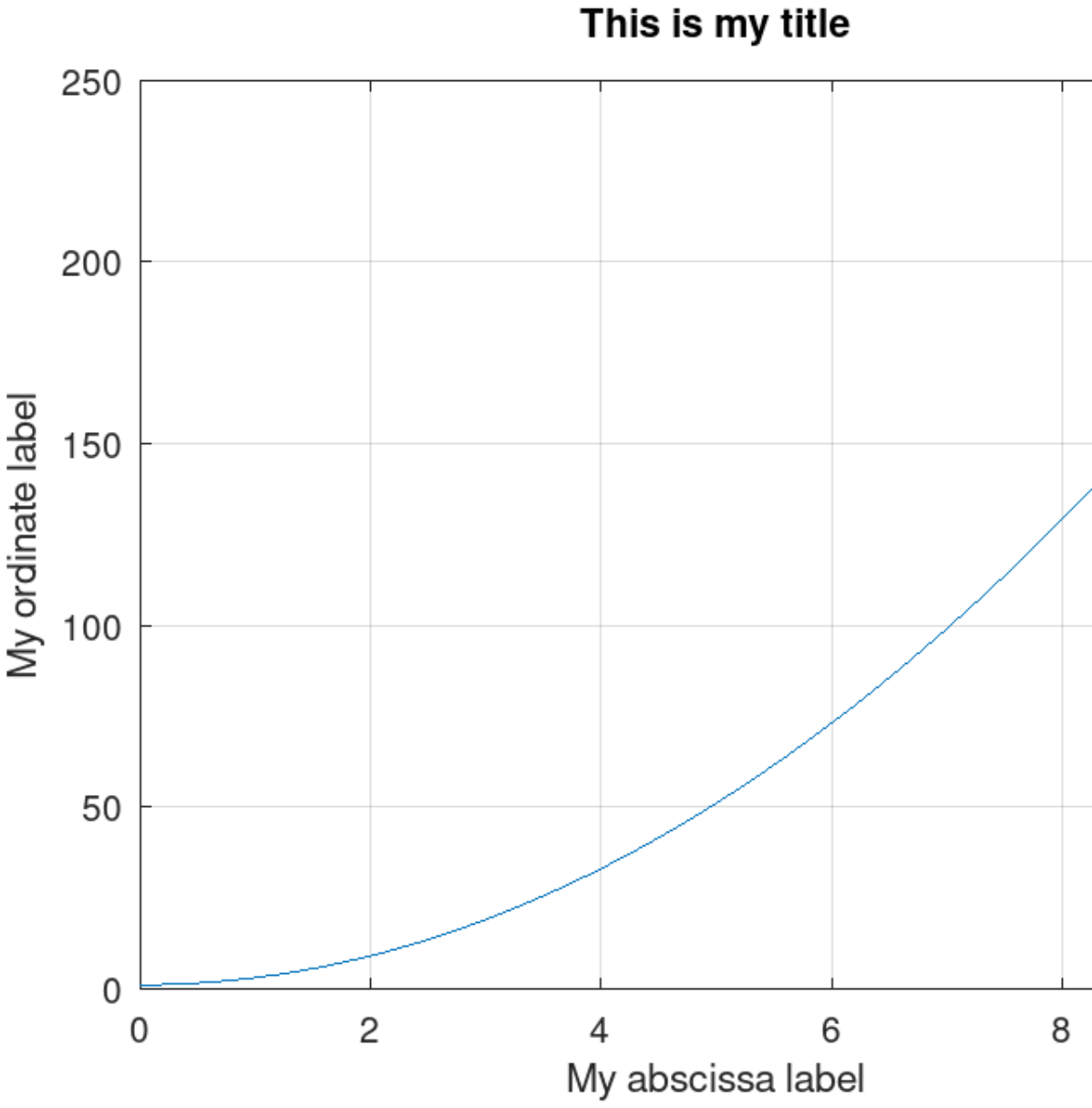
Alternatively, such a feature can better be implemented thanks to a shell function (to rely as much as possible on a safe emacs, otherwise on gedit, otherwise on nano); see our .bashrc.basics shell file for that.

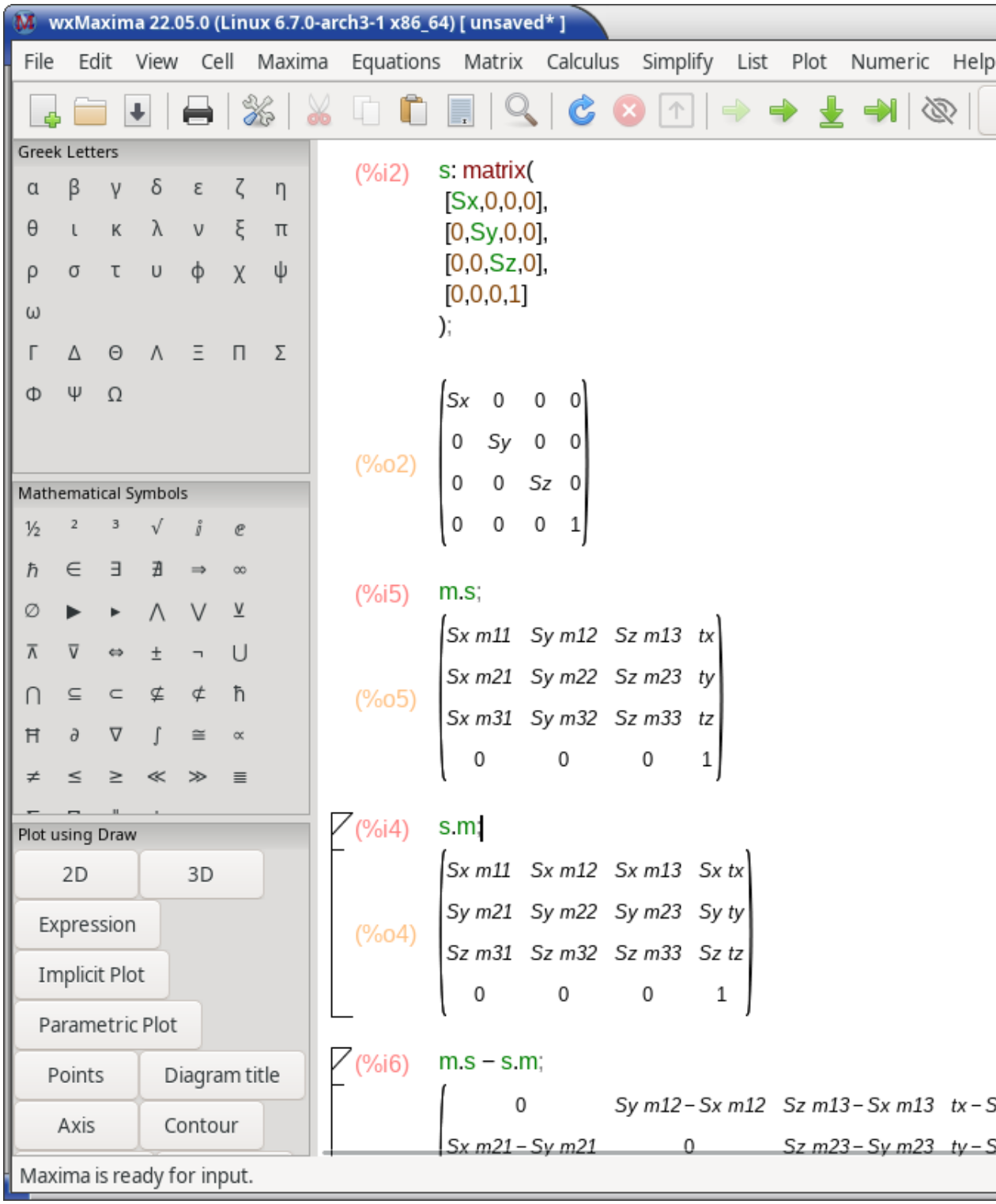
Please React!

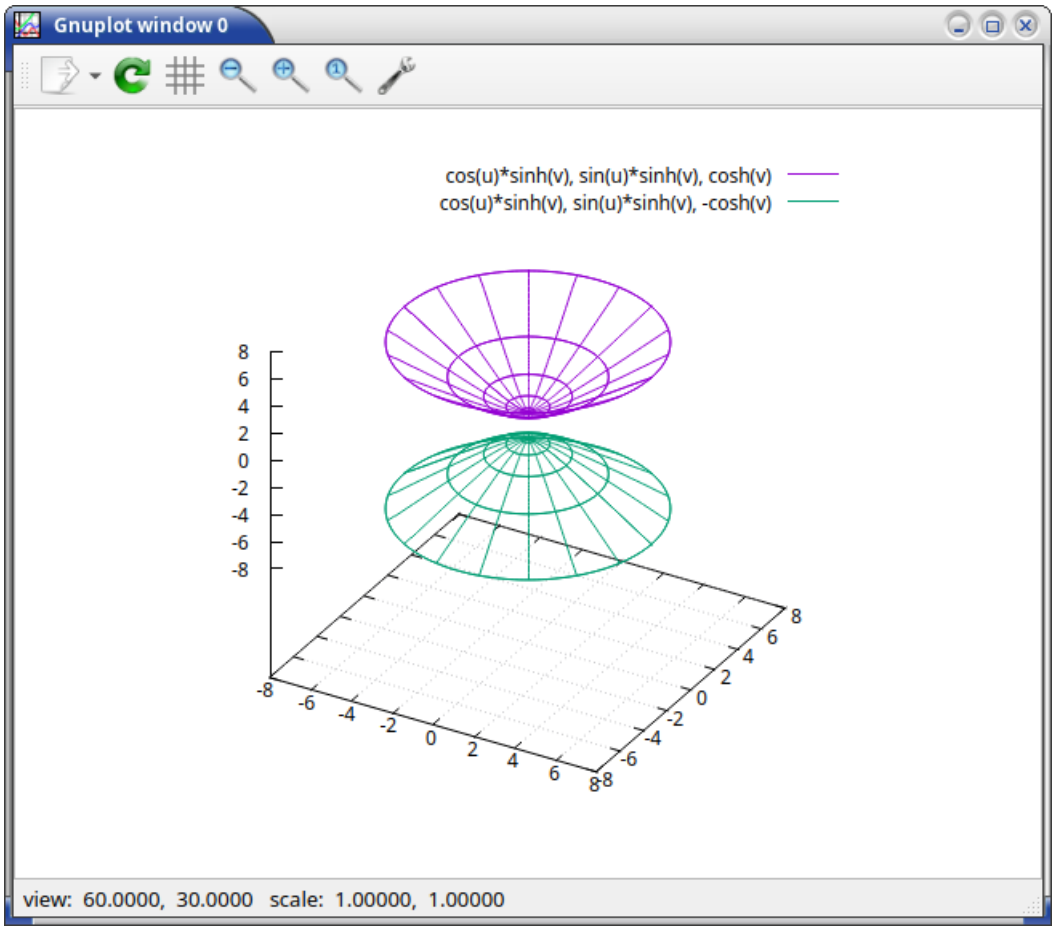
If you have information more detailed or more recent than those presented in this document, if you noticed errors, neglects or points insufficiently discussed, drop us a line! (for that, use the contact address at the top of this document).

Ending Word

Hoping that these Ceylan-HOWTOs may be of help!


This is my title
My ordinate label
My abscissa label
250
200
150
100
50
0
0
2
4
6
8


wxMaxima 22.05.0 (Linux 6.7.0-arch3-1 x86_64) [unsaved*]
File Edit View Cell Maxima Equations Matrix Calculus Simplify List Plot Numeric Help
Greek Letters
Mathematical Symbols
Plot using Draw
2D
3D
Expression
Implicit Plot
Parametric Plot
Points
Diagram title
Axis
Contour
(%i2) s: matrix([Sx,0,0,0], [0,Sy,0,0], [0,0,Sz,0], [0,0,0,1]);
(%o2)
(%i5) m.s;
(%o5)
(%i4) s.m;
(%o4)
(%i6) m.s − s.m;
Maxima is ready for input.


Gnuplot window 0
cos(u)*sinh(v), sin(u)*sinh(v), cosh(v)
cos(u)*sinh(v), sin(u)*sinh(v), -cosh(v)
8
6
4
2
0
-2
-4
-6
-8
-8
-6
-4
-2
0
2
4
6
8
view: 60.0000, 30.0000 scale: 1.00000, 1.00000

HOW-TO