ubuntu4mercury

Ubuntu Docker image with fresh compilers for the Mercury language.

Pulling the image from Docker hub

```
sudo docker pull fabnicol/ubuntu4mercury:latest
```

The Docker Hub image will probably not be updated as frequently as the Release section of this repository. Please read further to build the image from scratch.

Building the Docker image

```
git clone --depth=1 https://github.com/fabnicol/ubuntu4mercury.git
cd ubuntu4mercury
sudo ./build.sh
```

... or download from the Release section of this repository.

For some time after the date of the ROTD mentioned in the generated Dockerfile, it is possible to try and build the compiler from git source at a given revision. For this, check official Mercury repository commits. For example, for the rotd of Jan. 9, 2022, the corresponding revision 06f81f1cf0d339a can be safely changed to 40ddd73098ae (Jan. 10, 2022). Then change the values of fields `rotd-date` (resp. `m-rev`) in the configuration file **.config** as indicated in the comments of this file.

Alternatively, you can use the script **build.sh** with arguments as follows:

```
# change the git source revision to be built:
sudo ./build.sh REVISION
# change the date of the ROTD to be built AND the git source revision:
sudo ./build.sh YYYY-MM-DD REVISION
```

Examples:

```
sudo ./build.sh 4183b8d62f
sudo ./build.sh 2022-01-10 4183b8d62f
```

Command line arguments always override field values in **.config**.
The date should be subsequent to 2022-01-09 and the revision to 06f81f1cf0d. Please note that the further from these references, the higher the risk of a build failure.
In the two-argument case, if the build succeeds, the git source revision indicated as the second argument has been built using the ROTD compiler dated as the first argument.

Git revisions can be anything legal: revision hashes or HEAD, HEAD, HEAD~n (n an integer), etc. However, if you do not use revision hashes, the Docker cache system may mistakenly uncache a prior call to `git clone` and result in an outdated image. To avoid this, either delete the prior **ubuntu:mercuryHEAD** image (resp. ubuntu:mercuryHEAD^, etc.) *and all its dependencies* or use a hash. In the former case, I usually clean up my Docker context to avoid any issue as follows:

```
sudo docker system prune --all --volumes --force
```

Contents

The docker image has two Mercury compilers. The default compiler is the ROTD indicated in field **rotd-date** of the configuration file **.config**, built with all grades.
The development compiler is built from git source, with only the most useful C grades (configured with –**disable-most-grades**). The default revision is specified by the field **m-rev** in **.config**.
Both compilers can build themselves and somewhat newer versions.

The image contains a reasonable set of development libraries and tools, including a build toolchain, gdb and vim. Emacs is built from development git source with tagging support for Mercury using `etags`. Details are given on how to use `etags` for Mercury in the man page (`man etags`).

Mercury compilers and emacs have been built within the Docker image in the course of its building process. A ROTD built under Gentoo from original sources is downloaded and used as a bootstrapping compiler, then is replaced by the ROTD rebuilt within the Docker container.

Invocation

[do this once and for all if you downloaded the compressed image from github]
```
# xz -d ubuntu4mercury.tar.xz && docker load -i ubuntu4mercury.tar
```

```
# docker run -it ubuntu:mercury [ or: fabnicol/ubuntu4mercury:latest if you pulled]
```
`ubuntu:mercury # mmc / mmake`: for ROTD versions.
`ubuntu:mercury # mmc-dev / mmake-dev`: for Mercury git source versions.
`ubuntu:mercury # emacs` : to launch Emacs from within the running container.

To launch mmc (ROTD) in container from your host computer, run:

```
# docker run -it [your image name] mmc [arguments]
```

Replace `mmc` (resp. `mmake`) with `mmc-dev` (resp. `mmake-dev`) for the git source development version.

To launch Emacs in the container from your host computer, run:

```
# docker run -it [your image name] /usr/local/emacs-DEV/bin/emacs
```

Emacs should run just as well (in non-GUI mode) as on your host and `mmc / mmake` will be in the PATH of the guest shell.