Using OPAM

This document starts with a quick introduction, then covers most commonly-used OPAM features.

If you are a developper and want to get a project packaged or change an existing package, see the step-by-step packaging guide (Packaging.html)

The full documentation is available inline, using

```
opam --help
opam <command> --help
```

This document is intended as a quicker overview, use the above to dig into the details.

Basics

```
# ** Get started **
opam init            # Initialise ~/.opam using an already installed OCaml
opam init --compiler 4.02.1
                     # Initialise with a freshly compiled OCaml 4.02.1

# ** Lookup **
opam list -a         # List all available packages
opam search QUERY    # List packages with QUERY in their name or description
opam show PACKAGE    # Display information about PACKAGE

# ** Install **
opam install PACKAGE # Download, build and install the latest version of PACKAGE
                     # and all its dependencies

# ** Upgrade **
opam update          # Update the packages database
opam upgrade         # Bring everything to the latest version possible

# ** More **
opam CMD --help      # Command-specific manpage
```

You may prefer to browse packages online (https://opam.ocaml.org/packages). If you find a package there but not on your computer, either it has been recently added and you should simply run `opam update`, or it's not available on your system or OCaml version -- `opam install PACKAGE` will give you the reason.

Details on commands

opam init

OPAM needs to initialise its internal state in a `~/.opam` directory to work. This command can also take care of installing a version of OCaml there if needed, using the `--comp VERSION` option.

To operate as expected, some variables need to be set in your environment. You will be prompted to update your configuration, and given instructions on how to proceed manually if you decline.

opam update

This command synchronises OPAM's database with the package repositories. The lists of available packages and their details are stored into `~/.opam/repo/<name>`. Remember to run this regularly if you want to keep up-to-date, or if you are having trouble with a package.

Looking up packages

There are three useful commands for that:

- `opam list` List installed packages, or packages matching a pattern
- `opam search` Search in package descriptions
- `opam show` Print details on a given package.

opam install

This command installs packages along with all their dependencies. You can specify one or several packages, along with version constraints. E.g:

```
opam install lwt
opam install ocp-indent ocp-index.1.0.2
opam install "ocamlfind>=1.4.0"
```

opam upgrade

Will attempt to upgrade the installed packages to their newest versions. You should run it after `opam update`, and may use `opam pin` to prevent specific packages from being upgraded.

opam switch

This command enables the user to have several installations on disk, each with their own prefix, set of installed packages, and OCaml version. Use cases include having to work or test with different OCaml versions, keeping separate development environments for specific projects, etc.

Use `opam switch <version>` to *switch* to a different OCaml version, or `opam switch <name> --alias-of <version>` to name the new *switch* as you like. Don't forget to run the advertised `eval $(opam config env)` to update your PATH accordingly.

Creating a new switch requires re-compiling OCaml, unless you make it an alias of the "system" switch, relying on the global OCaml installation.

There are a bunch of specific or experimental OCaml compiler definitions on the official repository, list them all with `opan switch list --all`.

opam pin

This command allows to pin a package to a specific version, but in fact, as you know if you've read the Packaging guide (Packaging.html), it can do much more.

The syntax is

```
opam pin add <package name> <target>
```

Where `<target>` may be a version, but also a local path, an http address or even a git, mercurial or darcs URL. The package will be kept up-to-date with its origin on `opam update` and when explicitely mentionned in a command, so that you can simply run `opam upgrade <package name>` to re-compile it from its upstream. If the upstream includes OPAM metadata, that will be used as well.

```
opam pin add camlpdf 1.7                                  # version pin
opam pin add camlpdf ~/src/camlpdf                        # path
opam pin add opam-lib https://github.com/ocaml/opam.git#1.2   # specific branch or commit
opam pin add opam-lib --dev-repo                          # upstream repository
```

This can be used in conjunction with `opam source` to start and hack an existing package before you know it:

```
opam source <package> --dev-repo --pin
cd <package>; hack hack hack;
opam upgrade <package>
```

opam repo

OPAM is configured by default to use the community's software repository at opam.ocaml.org (https://opam.ocaml.org), but this can easily be changed at `opam init` time or later.

`opam repo add <name> <address>` will make OPAM use the definitions of any package versions defined at `<address>`, falling back to the previously defined repositories for those which aren't defined. The `<address>` may point to an http, local or version-controlled repository.

Defining your own repository, either locally or online, is quite easy: you can start off by cloning the official repository (https://github.com/ocaml/opam-repository) if you intend it as a replacement, or just create a new directory with `packages` and `compilers` sub-directories. See the packaging guide (Packaging.html) if you need help on the package format.

If your repository is going to be served over HTTP, you should generate an index using the `opam-admin` tool.