Using OPAM

This tutorial covers most of the OPAM features you will want to use. For a quick overview of most common OPAM commands, the Basic Usage tutorial is a good reading. There is also a dedicated guide explaining how to use OPAM efficiently in a development workflow.

Updating packages

When adding a new repository, as it is the case when running `opam init`, OPAM will store its state into `~/.opam/repo/<name>`, including the list of all available packages, their versions, etc. To be sure this state reflects the state of your remote repositories, just run:

```
opam update
```

This command will update the locally-saved states of your repositories to make sure you have access to the last updates from them.

Examining packages

You are now ready to install packages. But first you probably want to see what packages are available and get some info about those packages.

```
opam list
```

This will display as many lines as there are packages available, and each line displays the name of the package, its version if it is installed, and a short description. For the moment, you don't have any installed package, except the so-called *base* packages that aren't real OPAM packages, but modules that may or may not be distributed with you OCaml installation. The *base* packages are just an indication of whether the compiler comes with these modules or not.

```
opam search foo
```

This will display something similar to `opam list` except for it is only going to display available packages whose name or description match the string *foo*.

```
opam info opam
```

This will display information about the package *opam*. This information includes the installed version if the package is installed, all available versions that can be installed, and the full description of the package.

Installing packages

We are now ready to install some packages. Suppose you want to install the package *lwt*:

```
opam install lwt
```

If the package to be installed has no dependencies or if all its dependencies are already installed, then OPAM will install it without further ado. Otherwise, it will print a summary of the actions that are going to be performed, and you will be asked if it should go ahead or not.

A package can also have *optional dependencies*. These are dependencies that the package can make use of, but that are not mandatory. They will not be installed with the package by default, but *opam install* will take advantage of them if they are already installed while installing a package that optionally depends on them. For example, if you install *react* before installing *lwt*, *opam install lwt* will configure *lwt* to use *react*, but just installing *lwt* will not install *react*.

OPAM is even able to *track* optional dependencies. This means that while installing a new package, OPAM will check if any already installed package optionally depends on the package to be installed, and will recompile such packages and all their forward dependencies. For example, doing *opam install react* after an *opam install lwt* will have the effect to recompile *lwt* (with *react* support) as well as all the packages that depend on *lwt*.

Upgrading packages

After running `opam update`, it is possible that some packages that you installed got updated upstream, and it is now possible to upgrade them on your system. Just type:

```
opam upgrade
```

to upgrade your packages. The dependency solver will be called to make sure the upgrade is possible, that is, that *most* packages can get upgraded. OPAM will select the the best upgrade scenario and display a summary of what will be done during the upgrade. You will be asked if it should go ahead or not. This is similar to what happen when you upgrade your packages in most operating systems.

Using a different compiler

OPAM has the ability to install and use different OCaml compilers. This functionality is useful if you need to use different compilers on the same computer, and will make it very easy to switch between different compiler versions.

This functionality is driven by the `opam switch` command. Using `opam switch --help` will give you the full documentation. What follows is a short primer for the most useful features.

- `opam switch list` will display a list of the available compilers. The first section is a list of installed compilers on this computer. It contains at least *system*, which is not a compiler installed by opam but the compiler that was used to compile opam in the first place. A "*" symbol will be displayed before the current selected compiler.
- `opam switch 4.00.0` will make opam to switch to OCaml 4.00.0. If opam did not install it already, it will do so now. The first *opam switch* therefore takes the time it needs for your system to compile OCaml 4.00.0.
- `opam switch remove <version>` will just delete an opam-installed compiler from your system, thus freeing some disk space.

After switching to another compiler, opam will ask you to update your environment by running `eval ‘opam config env‘`. Indeed, compiler switching rely on environment variables so that your shell can find the libraries and binaries corresponding to the compiler you selected. Please don't forget to run this command!

These are the basic features of *opam switch*. There is two useful additions to them, which we will present now:

Ability to make a copy of a given compiler under an alias

It is useful if you want to use two instances of the same compiler. As *ocamlfind* allows only one version of an OCaml package to be installed, you can use this as a workaround to install multiple versions of the same package. Or more generally, if you need to hack packages and are afraid to break your clean opam installation, you can just use this feature as well. The sytax is

```
opam switch install <alias> --alias-of <version>
```

For example do `opam switch -install foo -alias-of 4.00.0` will make a copy of OCaml 4.00.0 under the name *foo*. Note that the state of compiler *4.00.0* will not be replicated in *foo*. *foo* is brand new with no packages installed.

Ability to *export* a compiler universe into another one

This means "replicate the state (installed packages, etc.) from one compiler to another one". It is useful if you want to have the ability to fork a given compiler. The syntax is

```
opam switch 4.00.0
opam switch export -f universe_for_4.00.0
opam switch 4.00.1
opam switch import -f universe_for_4.00.0
```

This will install all packages installed in the compiler `<version>` into the currently selected compiler.

Version pinning

```
opam pin <package> </local/path>
```

This command will use the content of `</local/path>` to compile `<package>`. This means that the next time you will do `opam install <package>`, the compilation process will be using a mirror of `</local/path>` instead of downloading the archive. This also means that any modification to `</local/path>` will be picked up by `opam update`, and thus `opam upgrade` will recompile `<package>` (and its forward dependencies) if needed.

To unpin a package, simply run:

```
opam pin <package> none
```

You can also pin a package to a specific version: `opam pin <package> <version>`

Handling of repositories

OPAM supports using multiple repositories at the same time, and supports multiple repository backends as well. Currently supported backends are *HTTP*, *rsync*, and *git*.

- The *HTTP* backend is used when the repository is available via the HTTP protocol, typically because it resides on a public website. This backend is the equivalent of what most Linux distributions are using to manage their packages. This backend needs either the *curl* or *wget* program to be installed on your system to work. It is also the default backend used by opam when doing a `opam init`.
- The *rsync* backend uses the *rsync* program to fetch data from a repository. It can thus be used if the repository is accessible to the rsync program, that is either locally (on your computer's filesystem) or via *sftp*.

- The *git* backend uses *git* to fetch data from a repository. It will be used if the repository is stored as a git repository.

These three backends should be sufficient to access most repositories. Additional backends can be added without much effort because of the modularized interface, basically, adding a backend means just implementing a module matching the `REPOSITORY` signature.

From repositories, OPAM makes a global index of all available packages. This means that if two repositories export the same package, OPAM will download it from a random one (in practice, from the last added repository). You can change that by editing `~/.opam/repo/index` and moving the repository you want to use in the beginning of each package line you want to install from this repository.

Using multiple repositories covers several cases:

You made packages and you want to use them

in addition of the ones available in the default OPAM repository. In order to do that, put these packages in a private repository and then add this repository in order to be able to install these packages the same way you install public OPAM packages. For example, if your packages are stored in a git repository, do:

```
opam init # Use the default repository
opam remote add devel git://devel.git
```

OPAM will add the default OPAM repository when initializing, and you add your development repository afterwards under the name *devel*. The *git* backend will be used because the URL starts by `git://`. By default, OPAM manage to figure out automatically which backend to use. See `opam remote -help` for more information.

You want more control over the public repository

because for example, you want to hack the packages, or you want to add some packages that are not available in the public repository yourself.

In this case, you probably want to do

```
git clone git://github.com/OCamlPro/opam.git ~/myrepo
opam init default ~/myrepo
```

Afterwards, you can modify packages into `~/myrepo` and use them in opam after doing an `opam update`.