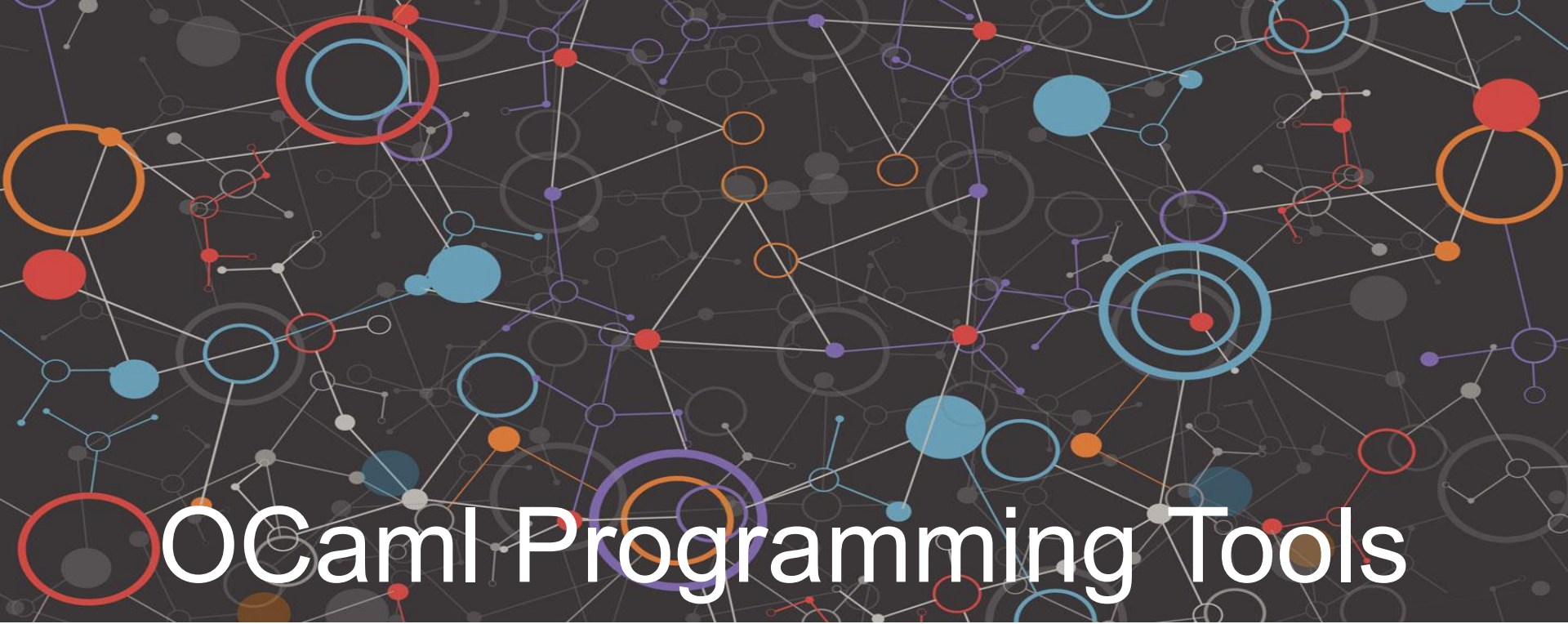


OCaml Programming Tools

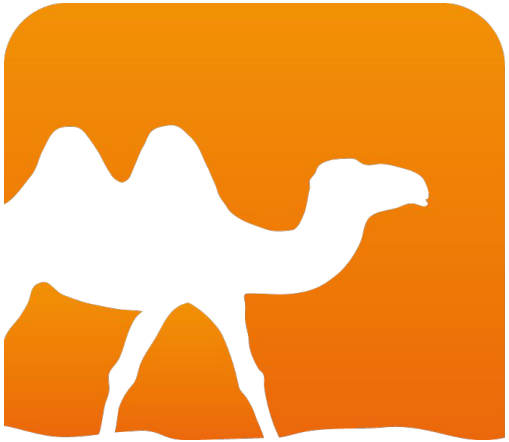
Letterio Galletta

Today's lecture: tools for OCaml development

- tryocaml
- ocaml
- opam
- ocamlc
- ocamlopt
- ocamlrun
- ocamlmktop
- ocamlbuild
- ocamlcp/ocamloptp
- ocamlprof
- ocamldoc

References

- OCaml Manual
- OCaml Standard Tools Cheat Sheets



OCaml

- OCaml – Official page
  - Compilers, toplevel, standard library, docs, tutorial, examples, etc.
- OPAM - OCaml Package Manager
  - Support different version of compiler
  - Make easy installation of library and tools
- Try OCaml
  - A online OCaml interpreter + a nice tutorial

Installation: http://ocaml.org/docs/install.html

Ubuntu (Versions 18.04 and newer)

```
$ add-apt-repository ppa:avsm/ppa

$ apt update

$ apt install opam

$ apt-get install opam

$ opam init    # Initialize ~/.opam and install the compiler
```

See the website for other Linux distro

Installation: http://ocaml.org/docs/install.html

MacOS X using Homebrew (Fink and Macports are supported too)

```
$ brew install gpatch

$ brew install opam

$ opam init    # Initialize ~/.opam and install the compiler

$ port install opam  # for macports
```

For Windows and other systems see the web page

Toplevel system (`ocaml`)

In the command line

```
ocaml           # interactive mode

ocaml foo.ml    # script mode
```

The toplevel doesn’t perform line editing, use `ledit` or `rlwrap`

Toplevel directives

- `#quit;;` Exit the toplevel
- `#help;;` Prints all directives
- `#cd "dir";;` Change current directory
- `#load "file-name"` Load a bytecode file
- `#use "file-name"` Read, compile and run a source file
- `#trace function-name;;` Start tracing the given function
- `#untrace function-name;;` Stop tracing the given function

See the Reference manual for all available directives and options

OPAM: Package manager

`opam init` # at the first use only to initialize `.opam` directory (*)

`opam switch create 4.10.0` # opam switch create 4.10.0

`opam search query` # search a package

`opam show package` # show information about a package

`opam install package` # download, compile and install a package (and deps)

`opam remove package` # remove a package

See https://opam.ocaml.org/doc/Usage.html for further info

(*) `eval `opam config env``

Interactive Toplevel

```
tmaeda — ocamlrun — 72×17
ocamlrun
macair:~ tmaeda$ utop
┌──────────────────────────────────────────────────────────────────────┐
│ Welcome to utop version 1.2.1 (using OCaml version 4.00.1+annot)! │
└──────────────────────────────────────────────────────────────────────┘
Cannot find file toplevel/topfind.
File ".ocamlinit", line 6, characters 0-20:
Error: Unbound module Toploop

Type #utop_help for help about using utop.

─( 09:00:00 )─< command 0 >──────────────────────────────{ counter: 0 }─
utop # List.fold_
┌──────────┬──────────┬──────────┬──────────┬──────────────────────────┐
│fold_left│fold_left2│fold_right│fold_right2│                          │
└──────────┴──────────┴──────────┴──────────┴──────────────────────────┘
```

Advantages

- Immediate feedbacks
- Rapid prototyping
- Interactive programming

Disadvantages

- No standalone applications
- No optimized native code

The rest of the lecture: OCaml compilers + others tools

Compilers

`ocamlc` generates bytecode(*)

`ocamlopt` generates native optimized code

(*) bytecode interpreter `ocamlrun`

Compilers

Bytecode

`ocamlc[.opt]`

- `*.cmo` bytecode object
- `*.cmi` interface object
- `*.cma` bytecode library

Native-code

`ocamlopt[.opt]`

- `*.cmx` & `*.o` asm object
- `*.cmi` interface object
- `*.cmxa` & `*.a` native library

Demo: bytecode compiler

- `ocamlc -c ex1.ml`
  - compile a source file
  - generate `ex1.cmo` & `ex1.cmi`
  - the option `-c` means “only compilation”
- `ocamlc ex1.ml -o ex1`
  - build a bytecode executable
  - generate `ex1.cmo` & `ex1.cmi` & `ex1`
  - the option `-o` specifies the executable name
- `./ex1` & `ocamlrun ex1` to run the executable

Demo: native-code compiler

- `ocamlopt -c ex1.ml`
  - compile a source file
  - generate `ex1.cmx` & `ex1.cmi` & `ex1.o`
  - the option `-c` means “compile only”
- `ocamlopt ex1.ml -o ex1`
  - build a native executable
  - generate `ex1.cmx` & `ex1.cmi` & `ex1.o` & `ex1`
  - the option `-o` specifies the executable name
- `./ex1` runs the program

Demo: use a library

- `ocamlc graphics.cma -o sierpinski sierpinski.ml`
  - build the bytecode executable `sierpinski`
  - `graphics.cma` bytecode library
  - `ocamlc` links together `graphics.cma` and `sierpinski.cmo`
- `ocamlopt graphics.cmxa -o sierpinski sierpinski.ml`
  - build a native executable
  - `graphics.cmxa` native-code library
  - `ocamlopt` links together our application and the library
- `#load "graphics.cma";;` in the OCaml interactive toplevel
- `ocamlmktop graphics.cma -o graphicstop`
  - a toplevel with code of a library preloaded at startup

Demo: create a library

- Generate a library (`*.cma`)
  - `ocamlc -c util.ml`
  - `ocamlc -a -o libutil.cma util.cmo`
- Compile your application
  - `ocamlc libutil.cma -o main main.ml`
- Native-code
  - `ocamlopt -c util.ml`
  - `ocamlopt -a -o libutil.cmxa util.cmx`
  - `ocamlopt libutil.cmxa main.ml -o main`

`ocamlbuild`: a generic build system

- Simplify the compilation of ocaml projects
  - determine the sequence of calls to the compiler
- in many cases automatically discover the various source files and dependencies of a project

Demo: simple use of `ocamlbuild`

- bytecode application
  - `ocamlbuild ex1.byte` bytecode application
  - `ocamlbuild -libs graphics sierpinsky.byte`
  - `ocamlbuild util.cma` bytecode library
- native-code application
  - `ocamlbuild ex1.native` native-code application
  - `ocamlbuild -libs graphics sierpinsky.native`
  - `ocamlbuild util.cmxa` native-code library

Other tools

- Profiling (`ocamlprof`)
  - `ocamlcp/ocamloptp ex1.ml -o ex1` compile the program
  - `./ex1` run
  - `ocamlprof ex1.ml` call `ocamlprof`
- Documentation (`ocamldoc`)
  - special comments in the code (see next slide)
  - `mkdir doc && ocamldoc -html -d doc ex1.ml`
  - other output format
    - -latex
    - -texi
    - -man

`ocamldoc`: special comments

```
(** This is a ocamldoc comment *)

(* This is not a ocamldoc comment *)
```

Note: Comments beginning with (and more than two * are ignored

`ocamldoc`: what inside a special comment?

Text formatting

`{n text}` section header level `n`

`{b text}` set `text` in bold

`{i text}` set `text` in italic

`{C text}` center `text`

`{{:url} text}` create a link

`[text]` set `text` in code style

Tags

`@author`

`@param id text`

`@return text`

`@raise Exception text`

See the reference manual for a complete list of text formatting option and tags

Others useful tools (not in this lecture)

- ocamldebug
- ocamllex, ocamlyacc and menhir
- camlp4, camlp5
- Merlin
- ocp-indent
- dune
- Ocamlformat
- utop
- ...