Erlang/OTP BEAM runtime and shell

START HERE

`erl`
Start the shell

`erl -pa <dir>`
Add to code path

`erlc <file>.erl`
Compile a module

`q().`
Quit the shell

`c(<mod>).`
Compile and load in shell

`erl -noshell -eval "<expr>" -s init stop`
Run an expression

`escript <file>`
Run a script

`erl -sname <name>`
Named node

Node one running VM (erl)
Application started as a unit
Supervisor restarts its children
Process lightweight, own mailbox

CONCEPTS

Every Erlang shell expression ends with a dot. Variables start upper case and bind once; use f(X). or f(). to forget them.

Nodes only talk if they share a cookie and the same name style: -sname and -name nodes cannot connect to each other.

START THE VM

`erl`
Start an interactive shell

`erl -noshell`
Run without a shell

`erl -noinput`
Never read stdin (implies -noshell)

`erl -eval "<expr>"`
Evaluate an expression at start

`erl -s <mod> <fun> <args>`
Call a function, args as atoms

`erl -run <mod> <fun> <args>`
Call a function, args as strings

`erl -pa <dir>`
Prepend a directory to the code path

START THE VM (cont.)

`erl -pz <dir>`
Append a directory to the code path

`erl -config <file>`
Load application config (sys.config)

`erl -args_file <file>`
Read arguments from a file

`erl -detached`
Run in the background, no console

`erl -version`
Print the emulator version

COMPILE

`erlc <file>.erl`
Compile to a .beam file

`erlc -o <dir> <file>.erl`
Put output in a directory

`erlc -I <dir> <file>.erl`
Add an include directory

`erlc -D<name>=<value> <file>.erl`
Define a macro

`erlc -Werror <file>.erl`
Treat warnings as errors

`erlc +debug_info <file>.erl`
Pass a compiler option

COMPILE (cont.)

Any +term argument to erlc is handed to the compiler as an option, for example +debug_info.

IN THE SHELL

`c(<mod>).`
Compile and load a module

`l(<mod>).`
Load or reload a module

`q().`
Quit the node

`b().`
Show variable bindings

`f().`
Forget all bindings

`f(<Var>).`
Forget one binding

`v(-1).`
Reuse the last result

`e(-1).`
Repeat the last command

`h().`
Show the history

`rl().`
List record definitions

`rr("<file>.hrl").`
Read records from a header

IN THE SHELL (cont.)

`help().`
List shell functions

`observer:start().`
Open the GUI observer

`Ctrl+G`
Job control menu

`Ctrl+C`
Break menu (a to abort)

`i / k / j / q`
At →: interrupt, kill, list jobs, quit

Ctrl+G then q quits cleanly; Ctrl+C then a aborts the VM at once. In scripts prefer init:stop() or halt().

DISTRIBUTION

`erl -sname <name>`
Node with a short host name

`erl -name <name>@<host>`
Node with a full host name

`erl -setcookie <cookie>`
Set the shared cookie

`erl -remsh <node>`
Shell connected to a remote node

`node().`
Name of this node

DISTRIBUTION (cont.)

`nodes().`
List connected nodes

`net_adm:ping(<node>).`
Connect; pong means reachable

`rpc:call(<node>, <mod>, <fun>, [<args>]).`
Call a function on another node

The default cookie lives in ~/.erlang.cookie. Keep it private: whoever has it can run code on the node.

ESCRIPT

`escript <file>`
Run a script

`#!/usr/bin/env escript`
First line: makes the file executable

`main(Args) →`
Entry point, Args is a list of strings

`halt(<code>)`
Exit with a status code

An uncaught exception in main/1 exits the script with status 127; a normal return gives 0.

REBAR3

`rebar3 new app <name>`
New application skeleton

`rebar3 compile`
Fetch deps and compile

`rebar3 shell`
Shell with project on the path

`rebar3 eunit`
Run EUnit tests

`rebar3 ct`
Run Common Test suites

rebar3 is the usual build tool for Erlang projects; see its own sheet for the full command list.

RECIPES

Run an expression and exit

```
erl -noshell -eval
'io:format("~p~n",
[lists:sum([1,2,3])])' -s init
stop
```

Compile and run a module

```
erlc hello.erl
erl -noshell -pa . -eval
'hello:world()' -s init stop
```

Script with arguments

```
#!/usr/bin/env escript
%%! -noshell
main([Name]) → io:format("hi
~s~n", [Name]);
main(_) → halt(1).
```