

The Alire Package Manager - Exercises

Alejandro R. Mosteo

Jun 14th, 2022

Professor with Centro Universitario de la Defensa, Zaragoza, Spain

 Twitter: @mosteobotic

 GitHub: mosteo

Getting started

Visit the following repository to get the materials:

- `https://github.com/mosteo/2022-AEiC-alire-tutorial`

Installation

Installation

- From binaries
 - <https://github.com/alire-project/alire/releases>
 - Scroll down to **Assets** section and expand
 - Grab latest stable release (1.2) for your platform
 - For Windows, the installer is recommended
 - **Ubuntu** recommended for minimal friction

Installation from sources

Useful to experiment with the latest features in the master branch

1. `git clone --recurse-submodules
https://github.com/alire-project/alire.git`
2. `cd alire`
3. On macOS only:
 - `export OS=macOS`
4. `gprbuild -j0 -P alr_env`
5. Executable generated at `bin/alr`
6. Run e.g. `alr version`

alr version output

```
$ alr version
```

APPLICATION

```
alr version:          1.2.0
libalire version:     1.2.0
compilation date:     2022-06-03 17:32:24
compiler version:     Community 2021 (20210519-103)
```

CONFIGURATION

```
config folder:        /home/jano/.config/alire
...
```

Toolchain selection (interactive)

```
$ alr toolchain --select
```

1. gnat_native=11.2.4
2. None
3. gnat_external=9.4.0 [Detected at /usr/bin/gnat]
4. gnat_arm_elf=11.2.4
5. gnat_avr_elf=11.2.4
6. gnat_riscv64_elf=11.2.4
- a. (See more choices...)

Enter your choice index (first is default):

>

Toolchain selection (headless)

Latest compiler and builder

```
$ alr toolchain --select gnat_native gprbuild
```

Latest within a major version

```
$ alr toolchain --select gnat_native^11 gprbuild
```

^^^

Precise version

```
$ alr toolchain --select gnat_native=10.3.2 gprbuild
```

^^^^^^

Toolchain selection (distro compiler)

- Choosing 'None' will use gprbuild blindly
 - You must ensure there is one gprbuild/gnat in \$PATH
- Why choose a gnat_external?
 - The selected compiler takes precedence, as long as it is compatible with dependencies.
 - This may prevent a large download in some cases

Toolchain inspection

```
$ alr toolchain
```

CRATE	VERSION	STATUS	NOTES
gprbuild	22.0.1	Default	
gprbuild	2019.0.0	Available	Detected at /usr/bin/gprbuild
gnat_arm_elf	11.2.3	Available	
gnat_native	10.3.2	Available	
gnat_native	11.2.4	Default	
gnat_external	9.4.0	Available	Detected at /usr/bin/gnat

- Available compilers take precedence over uninstalled ones
 - May avoid a large download

Testing the (ecosystem) waters

Retrieve and run a release

```
$ alr get hangman
$ ls
hangman_1.0.0_be628ad5
$ cd hangman*
$ alr run --list
Crate hangman builds these executables:
    hangmain (not found)
$ alr run
```


 Other executable crates to experiment with:

eagle_lander, lace_gel_full_demo, mathpaqs,
play_2048, rsfile, septum, shoot_n_loot, zipada

Look up by name

- Look up crates by name/description:
 - `alr search --crates [<substring>]`
- look up releases by name/description:
 - `alr search <substring> [--full]`
 - `alr search --list [--full]`

Without `--full`, only the last release of a crate will be returned.

 Experiment with these commands to locate some crate related to a topic of your interest: `toml`, `yaml`, `py`, `math`, ...

Look up by tag/other

Looking into names/descriptions sometimes is not enough:

```
$ alr search --property <substring>
```

📁 Identify releases related to a person

📁 Locate crates with tags: embedded, game, spark, terminal, ...

Check out most popular tags at

<https://alire.ada.dev/tags/>

Showing details before retrieving

```
$ alr show <crate>[version subset]
```

 Try it:

- `alr show hello`
- `alr show gnat_native^10`

Details about dependencies:

- `alr show hello --solve | --tree | --graph`

 Try these with a crate with more dependencies:

`lace_gel_full_demo, septum, shoot_n_loot`

Showing details after retrieving

From within a local crate folder, you can simply issue:

- `alr show`
- `alr show --solve | --tree | --graph`
- `alr with --solve | --tree | --graph | --versions`

Useful to display your work-in-progress manifest

👉 Try it inside the own Alire source tree.

Starting crates

Starting from scratch

At the parent location:

- `$ alr init --lib <crate name>`
- `$ alr init --bin <crate name>`

Naming rules:

- `$ alr help identifiers`

Starting from preexisting code

Inside the root of your existing code:

- `$alr init --lib --in-place --no-skel <crate name>`

Generated disk structure

```
$ alr init --bin demo
✓ demo initialized successfully.

$ tree demo
demo
├── alire.toml      # Manifest
├── demo.gpr        # Project file
├── share
│   └── demo        # Place exported resources here
└── src
    └── demo.adb     # Main subprogram
```

Generated disk structure (after build)

```
$ cd demo && alr build
Build finished successfully in 0.27 seconds.

$ tree ../demo
demo
├── alire
│   ├── alire.lock           # Dependency solution
│   └── config.toml         # Local configuration
├── bin
│   └── demo                 # Built executable
├── config
│   ├── demo_config.ads     # Generated metadata & config
│   ├── demo_config.gpr     # Generated project file
│   └── demo_config.h       # C equivalent to Ada spec
└── obj
    └── development         # Intermediate build artifacts
        ├── demo.ali
        └── demo.o
```

Working with dependencies

The goal

- Create a new binary crate (or reuse demo)
- Add the 'libhello' dependency
- Call its `Libhello.Hello_World` procedure from your main subprogram
- Run it

Adding a dependency via alr

✍ Inside your crate:

```
$ alr with libhello
```

✍ Verify new code is being built:

```
$ alr build
```

Compile

```
[Ada]          demo.adb
```

```
[Ada]          libhello.adb
```


Changes caused by adding a dependency

- ✎ Inspect the `alire.toml` manifest contents
- ✎ Inspect the `config/demo_config.gpr` project file
- ✎ Edit the main subprogram to obtain this output, using the library instead of `Ada.Text_IO`:

```
$ alr run  
Hello, world!
```

Adding a dependency manually

✎ Try to remove the `[[depends-on]]` section from `alire.toml` and build again

✎ Re-add the dependency manually, by editing `alire.toml`:

```
[[depends-on]]  
libhello = "*" # This means any version whatsoever
```

✎ Observe the feedback from `alr` when the manifest is edited by hand

Conditional expressions in the manifest

Alire supports dynamic expressions in some properties:

property = value becomes

property. 'case(var)'.var_case = value

Available variables: distribution, host-arch, os, word-size

Valid values: alire-platforms.ads¹

¹<https://github.com/alire-project/alire/blob/master/src/alire/alire-platforms.ads>


Conditional expression examples

```
available = true      # Implicit in all manifests unless...

[available.'case(os)']
linux = true
'...' = false

[[depends-on]]
[depends-on.'case(os)'.linux]
libhello = "*"

```

 Edit your manifest so the dependency on libhello is only for your operating system.

Verifying a conditional expression

```
$ alr show
demo=0.1.0-dev: Shiny new project
Origin: path /tmp/demo
Properties:
  Author: Alejandro R. Mosteo
  ...
Dependencies (direct):
  case OS is
    when Linux => libhello*
```

Build profiles

Basics of build profiles

<https://github.com/alire-project/alire/blob/master/doc/catalog-format-spec.md>

- Predefined build profiles:
 - Development
 - Validation
 - Release
- Each defines a set of predefined switches
- You can totally override in your main project file
 - There are ways to avoid needing to do so

Building with default profiles

- Dependencies by default use release mode
- For the root crate:
 - `alr build [--release|--validation|--development]`
 - Defaults to development
- Query last build mode with:
 - `alr config last_build_profile`
 - Contents of `config/<crate>_config.ads`

Overriding defaults via the manifest

```
[build-profiles]
"*"          = "development"  # Applies to non-explicit crates
libhello    = "release"
demo        = "validation"    # The root crate may appear here
```

 Try it and verify via `config/demo_config.gpr`

Adjusting switches of a profile

Alire has collections of switches grouped by topic.
E.g.:

- Optimization
 - Performance
 - Size
 - Debug

Then, *for the own crate*, you can:

```
[build-switches]  
release.optimization = "performance"  
"*.optimization      = "debug"
```

Switches categories

<https://alire.ada.dev/docs/#release-information>

- Ada_Version
- Compile_Checks
- Contracts
- Debug_Info
- Optimization
- Runtime_Checks
- Style_Checks


Category overrides

```
[build-switches]  
release.optimization = ["-O2", "-gnatn"]
```

Environment

Display and set the environment

- `$ alr printenv`
 - `--unix` (default)
 - `--powershell`
 - `--wincmd`

 Use `alr help printenv` for ways to apply the output

Executing within the environment

- `$ alr edit`
 - Defaults to gnatstudio
 - `alr config --set editor.cmd <editor> ${GPR_FILE}`
- `$ alr exec -- <command>`

Publishing

Preliminaries

- Have a github user
- Fork the community index repository:
 - <https://github.com/alire-project/alire-index>
- It is also possible to publish a tar/zip file hosted anywhere
 - No need to have fork of the index
 - GitHub account still needed

Publishing a git commit

Inside your up-to-date local repository:

```
$ alr publish [--skip-build]
```

Works only for trusted hostings:

```
$ alr publish --trusted-sites
```

Publishing a tarball

Source file already on-line:

```
$ alr publish https://url/of/file.zip
```

Create file on the go:

```
$ alr publish --tar
```

Example of on-line manifest (commit)

`https://github.com/alire-project/alire-index/blob/
stable-1.2/index/aa/aaa/aaa-0.2.5.toml`

```
[origin]
commit = "521a8669cf8dbd0eeb71d22d2634421265d52f62"
url = "git+https://github.com/mosteo/aaa.git"
```

Example of on-line manifest (tarball)

```
https://github.com/alire-project/alire-index/blob/  
stable-1.2/index/ap/apdf/apdf-5.0.3.toml
```

```
[origin]  
url =  
    "https://sourceforge.net/projects/apdf/files/apdf_005_r3.zip"  
hashes =  
    [ "sha512:db27598986b1744b024803348350e48b9fe14a14b4..." ]
```

Example of on-line manifest (monorepo commit)

```
https://github.com/alire-project/alire-index/blob/  
stable-1.2/index/js/json/json-5.0.2.toml
```

```
[origin]  
url = "git+https://github.com/onox/json-ada.git"  
commit = "d429d7af880ab9ed38d58ac08c1c9a16e7697752"  
subdir = "json"
```

Pinning dependencies

Inspecting pins

Pins are useful to substitute a dependency with a work-in-progress version, or with an unpublished commit.

List pins with `$ alr pin`

 Pins in alr v1.1:

1. `git clone https://github.com/alire-project/alire`
`--branch release/1.1`
2. `cd alire`
3. `alr pin`


Pinning dependencies

- Pin to a local folder
 - alr with `--use /path/to/folder`
- Pin to a remote repository
 - alr with `https://github.com/mosteo/wordlist`

alr with will add dependency + pin

alr pin only works when the dependency already exists in the manifest

Pins for testing/demo subcrates

 Create a library crate and a binary demo subcrate

1. `alr init --lib mylib`
2. `cd mylib`
3. `alr init --bin mylibdemo`
4. `cd mylibdemo`
5. `alr with --use ..`
6. `alr edit`

Structure of lib + demo subcrate

```
$ tree mylib
```

```
mylib
├── alire.toml
├── mylibdemo
│   ├── alire.toml
│   ├── mylibdemo.gpr
│   └── src
│       └── mylibdemo.adb
├── mylib.gpr
├── src
│   └── mylib.ads
```

WORDLE example

Implementing WORDLE in an afternoon

Wordle is a popular game of guessing a word of 5 letters in six attempts or less.

1. `$ git clone https://github.com/mosteo/wordle_ada
--branch demo`
2. `$ cd wordle_ada`
3. `$ alr run`

Example run

```
Please enter your guess #6: beaks
575
576 ## Structure of lib + demo subcrate
577
578 \linespread{0.8}
579 *** bash
580 $ tree mylib
581
582 mylib
583 |— alire.toml
584 |— mylibdemo
585 |   |— alire.toml
586 |   |— mylibdemo.gpr
587 |   |— src
588 |       |— mylibdemo.adb
589 |— mylib.gpr
590 |— src
591 |   |— mylib.ads
592 ***
593
594
595 I'm sorry, you ran out of attempts.
596 The word was: DHAKS
```

Project structure

```
$ alr with --tree
```

```
wordle=0.1.0
├── aaa=0.2.6 (~0.2.6)
├── ansiada=1.0.0 (^1.0.0)
├── wordlelib=0.1.0 (~0.1.0)
│   └── aaa=0.2.6 (~0.2.5)
└── wordlist=0.1.2 (~0.1.2)
    ├── aaa=0.2.6 (~0.2.5)
    ├── ada_toml=0.3.0 (~0.3.0)
    ├── gnat=11.2.4 (gnat_native) (>=10 & <2000)
    └── resources=0.1.0 (~0.1.0)
```

Used crates

- wordle
 - user interaction
- aaa
 - string casing
 - word containers
- ansiada
 - terminal colors
- wordlelib
 - game logic
- wordlist
 - list of English words
- ada_toml
 - wordlist loading
- resources
 - locating wordlist.toml

Fix the WORDLE project yourself

- 📖 Follow the STEPs in the source code to complete the WORDLE project
- 📖 Each STEP indicates where to locate the next one to continue
- 📖 Start with
 - `git clone https://github.com/mosteo/wordle_ada --branch tutorial`
 - `cd wordle_ada`
 - Your first STEP is at the top of `alire.toml`
 - Each step usually includes several TODOs

File generation

Pre-build file generation

- In some restricted embedded environments, it is not feasible to use some ada dynamic constructs. This may difficult efficiently tailoring the code for a particular board.
- Alire allows to define variables that are generated into spec files before the build, and so can be used statically.


Variable types

<https://github.com/alire-project/alire/blob/master/doc/catalog-format-spec.md#using-crate-configuration>

```
[configuration.variables]
Enable_Logs = {type = "Boolean", default = false}
Buffer_Size = {type = "Integer",
               first = 128, last = 1024, default = 256}
```

Setting values

```
[configuration.values]  
crate_name.Enable_Logs = true
```

 Try adding a configuration variable to your crate.
Inspect the generated file in `config/demo_config.ads`

Indexes

Working with indexes

- Listing indexes:
 - `$ alr index`
- Updating indexes:
 - `$ alr index --update-all`
- Adding another index
 - `$ alr index --add <url> --name <name>`
 -  Try adding
`https://github.com/mosteo/aeic22_index`
- Removing an index:
 - `$ alr index --del <name>`
 -  Try removing the community index to see the contents of my index only

Resources

Exporting resources

Resources are files to be used at run-time: graphics, texts, databases, etc.

Place your resources at `./share/my_crate/`

```
$ tree demo
demo
├── alire.toml      # Manifest
├── demo.gpr        # Project file
├── share
│   └── demo        # Place exported resources here
└── src
    └── demo.adb     # Main subprogram
```

Using resources [from other crates]

Crate within the community index:

- `$ alr show resources`
- `$ alr with resources`

 Inspect `resources.ads`

Provides a path to a crate's resources:

- During development, inside the crate folder
- After `gprinstall`, relative to the binary

```
$ tree prefix
prefix
├── bin
└── share
    ├── demo
    └── another_crate
```

Configuration

Inspecting configuration

```
$ alr config
```


```
$ alr help config
```

Setting variables

```
$ alr config --set [--global] <key> <value>
```

This is the basis of the *aliases* mechanism:

```
$ alr help aliases
```

 Define the alr graph alias as suggested in the help output