DejaVu Sans WenQuanYi Micro Hei

Aura Documentation

Release 1.3.0

Colin Woodbury

November 17, 2016

Contents

Aura is a package manager for Arch Linux. It's main purpose is as an "AUR helper", in that it automates the process of installating packages from the Arch User Repositories. It is, however, capable of much more.

CHAPTER 1

General

1.1 Aura 2 Design

1.1.1 Preface

This is a design document for version 2 of Aura. Note that specifications are written in present tense, as in, âĂIJAura does thisâĂİ even if at the time of writing those features arenâĂŹt implemented yet. This is to ensure that the document can act as a reference for AuraâĂŹs behaviour post-release.

1.1.2 Mission Statement

Aura is a cross-distribution package manager for GNU/Linux systems. It is based around a distribution-specific Hook system for custom build/install behaviour, while maintaining a custom interface across all distros. Aura itself provides:

- Dependency management.
- Package downloading.
- Package-state backups/restoration.

Aura's authors recognize that attempting to create universal standards can be problematic, but that is precisely why Aura exists. By having a unified interface over multiple packaging standards, users can transition between distributions more easily, and distribution developers can avoid reinventing the wheel by writing their own package management software.

1.1.3 Functionality

General

By default, Aura handles three types of packages:

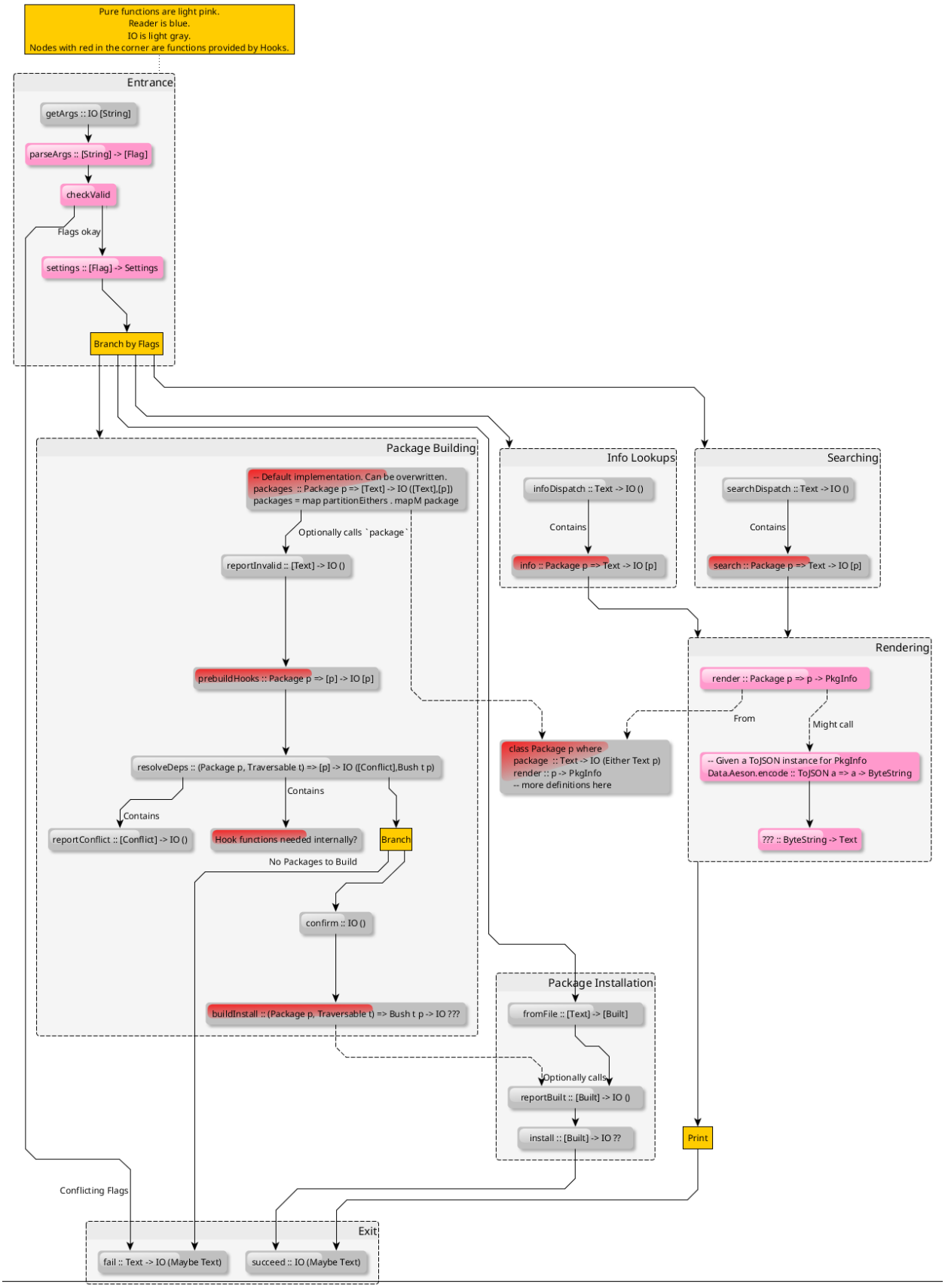
Repository Packages Prebuilt binaries available direct from the userâĂŹs Distribution.

Foreign Packages Packages that generally need to be compiled by the user. Their versioning/source locations may be managed by the Distribution is some way.

Local Packages Packages installed on the userâĂŹs system. Records of them and the files belonging to them are stored in a database, and package files themselves are stored in a cache (in `/var/cache/` or elsewhere).

A number of operations can be performed on these package types, as explained below.

Program Flow



Common Behaviour

As can be gleamed from the program flow chart, the "capital letter" operators pertaining to packages share the same

- Installation: `aura -{S,F,L} <packages>`
- Searching: `aura -{S,F,L}s <regexp-like-pattern>`

Output sample:

```
$> aura -Ss nvidia
extra/nvidia 337.25-3 [installed]
    NVIDIA drivers for linux
extra/nvidia-304xx 304.121-5
    NVIDIA drivers for linux, 304xx legacy branch
extra/nvidia-304xx-libgl 304.121-2
    NVIDIA drivers libraries symlinks, 304xx legacy branch
```

Aura will fail silently when no pattern is given.

- Info Lookups: `aura -{S,F,L}i <packages>`

To the question, "What does it mean to install a local package?" consider the following the use cases:

```
$> aura -L foo-1.2-1.pkg.tar.gz  -- Installing a prebuilt package tarball.

$> aura -L bar
aura >=> Which version of `bar` do you want?
1. bar-1.1.1-1
2. bar-1.2.1-1
3. bar-1.2.3-1
>>  -- You choose which to install from your _local_ cache.
```

Local Package Removal

Local packages may be removed singularly, or in groups.

Usage:

- Just the packages named: `aura -R <packages>`
- Packages named and all deps (recursive): `aura -Rr <packages>`

Local Package Backups

The state of locally installed packages may be recorded and restored at a later date.

Usage:

- Store a snapshot of all installed packages: `aura -B`
  - This record is stored in `/var/cache/aura/states`.
  - Filenames are of the form: `YYYY.MM.MonthName.DD.HH.MM`.
  - The data itself is stored as JSON to ease use by other tools.
- Restore a snapshot: `aura -Br`

```
{ "date": "2014-04-09",
  "time": "20:00",
  "packages": [ { "pkgname": "alsa-lib",
                  "version": "1.0.27.2-1" },
                // more packages here
```

```
                ]
}
```

Other

Dependency Resolution

- AUR dependencies are no longer resolved through PKGBUILD bash parsing. The AUR 3.x API includes the necessary dependency information.
- **Resolution Successful**: Data in the form is yielded. These are groups of packages that may be built and installed simultaneously. That is, they are not interdependent in any way.
- **Version Conflicts**:
- Dependency resolution fails and the build does not continue.
- The user is shown the chart below so it is clear what dependencies from what packages are causing issues.
- All packages that had dependency issues are shown.
- Supplying the `--json` flag will output this data as JSON for capture by other programs.

```
+----------+--------+----------+---------+
| Dep Name | Parent | Status   | Version |
+==========+========+==========+=========+
| foo      | None   | Local    | 1.2.3   |
| foo      | bar    | Incoming | < 1.2.3 |
| foo      | baz    | Incoming | > 1.2.3 |
+----------+--------+----------+---------+
| curl     | git    | Local    | 7.36.0  |
| curl     | pacman | Incoming | 7.37.0  |
+----------+--------+----------+---------+
| lua      | vlc    | Incoming | 5.2.3   |
| lua      | conky  | Incoming | 5.2.2   |
+----------+--------+----------+---------+
```

```
// As JSON:
{ [ { "Name": "foo",
      "Local": { "Parent": "None",
                 "Version": "1.2.3" },
      "Incoming": [ { "Parent": "bar",
                      "Version": "< 1.2.3" },
                    { "Parent": "baz",
                      "Version": "> 1.2.3" }
                  ]
    },
    { "Name": "curl",
      "Local": { "Parent": "git"
                 "Version": "7.36.0" },
      "Incoming": [ { "Parent": "pacman",
                      "Version": "7.37.0" }
                  ]
    },
    { "Name": "lua",
      "Local": "None",
      "Incoming": [ { "Parent": "vlc",
                      "Version": "5.2.3" },
                      { "Parent": "conky",
```

```
                        "Version": "5.2.2" }
                    ]
    }
  ]
}
```

Dependency Information Output

- Information for all immediate dependencies for any given package can be output in human-readable format by default with `-{A,S}d`.
- Adding `--recursive` will yield all dependencies and *their* dependencies as well.
- Adding `--json` will output this information in JSON for use by other software that may sit on top of Aura.

Concurrent Package Building

- Package data is returned from dependency checking in the form `[[Package]]` (see *Dependency Resolution*). Each sublist of packages have no interdependencies, so they are built concurrent to each other and then installed as a block.

PkgInfo

Package searching and Info lookup algorithms work with `PkgInfo` data. It holds:

- Repository name
- Package name
- Version
- Description
- Architecture
- URL
- Licenses
- âĂIJProvidesâĂİ
- Dependencies
- âĂIJConflicts WithâĂİ
- Maintainer
- Optional fields (provided as `[(Text,Text)]`):
  - Download/Install sizes
  - Group
  - Votes
  - GPG information
  - etc.

Abnormal Termination

Users can halt Aura with `Ctrl-d`. The message `Stopping Aura...` is shown. All temporary files in use are cleared here.

Colour Output

All output to terminal (save JSON data) is output in colour where appropriate. The user can disable this with `--no-color{ur,r}`.

Usage Tips

The user is shown usage tips when waiting for dependencies to resolve, etc. A number of tips are Aura-centric, but distro-specific ones can be defined in *AuraConf*.

Plugins

Like XMonad, behaviour is built around hooks/plugins that are themselves written in Haskell. Each Linux distribution writes and provides to *AuraConf* functions that fill certain type/behaviour requirements as explained below.

AuraConf

AuraConf is AuraâĂŹs configuration file. Here, distributions and users can add Hooks to define custom behaviour for their native packaging system. The command `aura --recompile` rebuilds Aura with new Hooks. Also, the following paths can be defined in this file:

- Package cache.
- Aura log file.
- Default build directory.
- Mirror URLs for binary downloads.
- TODO: What else?

Package Typeclass Instances

Each Hook family (as described below) operates with one type of package. Any package type has to implement the *Package* typeclass. It takes the following shape:

```
class Package p where
  -- Converts a package name to its ADT form. Upon failure,
  -- yields its name wrapped in a `Left`.
  package :: Text -> IO (Either Text p)

  -- All Packages must be able to present their prime information
  -- in a standard way for Aura output functions.
  render :: p -> PkgInfo
```

Hooks ADT

Hooks are passed through Aura as an ADT of functions.

```
{-# LANGUAGE RankNTypes #-}

data Hooks p = Hooks { info   :: Package p => Text -> IO [p]
                     , search :: Package p => Text -> IO [p]
                     , -- more to come
                     }
```

Aesthetics

Size Information

Unless `-q` is passed to Aura, the following information is displayed before installation from the official repositories.

```
Total download size : xx MiB
Net upgrade size    : xx MiB
```

The units are displayed with binary prefixes, such as: B, KiB, MiB, GiB and TiB.

Localisation

Aura is available for use in multiple languages. Language can be set via environment variables or by using Aura flags that correspond to that language. Note that use of a flag will override whatever environment variable is set. Each language has an English name and its native equivalent (accents and other non-ascii characters are compatible). For example:

- `--croatian` and `--hrvatski`
- `--french` and `--franÃğais`

Version Information When Upgrading

Whenever a package needs an upgrade, unless `-q` is passed to Aura, then a detailed chart is produced, as described below.

The coloured part is denoted with `<colour></colour>` tags, enclosing the text to colourise such as <colour>text to colourise</colour>.

New Package Dependency Needed

```
âĞŠ New package needed:
repository/package        1.0-1            (required by xxx) (Net change: ±xx MiB)
```

New Package Release

```
âĞŠ New package release:
repository/package        1.0-1    -->    1.0-<green>2</green> (Net change: ±xx MiB)
```

New Package Version

```
âĞŠ New package version:
repository/package        1.0-1    -->    1.<green>2-1</green> (Net change: ±xx MiB)
```

Aura Versioning

- Aura uses Semantic Versioning, meaning itâĂŹs version numbers are of the form `MAJOR.MINOR.PATCH`.

Haskell Requirements

Strings

All Strings are represented as from `Data.Text`. This is available in the `text` package from Hackage. The following language pragma should be used where appropriate for String literals being converted to automatically:

```
{-# LANGUAGE OverloadedStrings #-}
```

JSON Data

All JSON input and output is handled through the `aeson` and `aeson-pretty` packages.

Parsing

All parsing is done with Parsec. Regular Expressions are no longer used anywhere in Aura.

Other Libraries

Information on other Hackage libraries used in Aura can be found here.

Package Requirements

Aura must be available in the following forms:

`haskell-aura` An AUR package pulled from Hackage, contains only the Aura âĂIJshellâĂİ layer. The user must install another package to get the Arch Linux Hooks, and then build the executable themselves.

`aura` Official Arch-flavoured Aura, built and configured in a cabal sandbox. `cabal-install` is the only Haskell related dependency.

`haskell-aura-git` Most recent version of Aura, as found on its source repository.

`aura-legacy` A static copy of Aura 1. Has Haskell dependencies.

1.1.4 Arch Linux Specifics

ABS Package Building/Installation

- There is no longer a `-M` option. All ABS package interaction is done through `-S`.

- Installs prebuilt binaries available from Arch servers by default.
- Build options:
- If the user specifies `--build`, the package will be built manually via the ABS.

AUR Package Building/Installation

- Builds manually by default, as there is no prebuilt alternative for the AUR (by design).

PKGBUILD/Additional Build-file Editing

- Support for `customizepkg` is dropped, as AUR 3.x provides dependency information via its API.
- Users can edit included `.install` files and the **behaviour** of PKGBUILDs with `--edit`. This is done after dependency checks have been made via the data from the AUR API. Users are urged *not* to edit dependencies at this point, as only `makepkg`, not Aura, will know about the changes.
- If you do want to build a package with different dependencies, consider whether there is value in creating your own forked package for the AUR (named `foo-legacy`, etc.). Others may benefit from your effort.
- If you are trying to fix a broken package, rather than circumventing the problem by building manually with `makepkg`, please contact the maintainer.

AUR Interaction

- AUR API calls are moved out of Aura and into a new Hackage package `aur` (exposing the `Linux.Arch.Aur.*` modules).
- It provides conversions to and from JSON data and Haskell data.
- This is preparation for future versions of Aura that allow use in other Linux distributions by swapping out sections of their back-end (with modules like `Linux.Debian.Repo` etc.)

1.1.5 Coding Standards

Record Syntax

When using record syntax for ADTs, function names should be suffixed with âĂIJOfâĂİ to reflect their noun-like nature:

```
data Package = Package { nameOf    :: String
                       , versionOf :: Version
                       , depsOf    :: [Package] }
                       deriving (Eq, Show)
```

1.2 Aura Changelog

1.2.1 1.3.5

- Aura now uses version 5 of the *aur* package, to fix a critial bug
- Updated Spanish and Polish

1.2.2 1.3.4

- Bash parser bug fix. Fixes some packages.

1.2.3 1.3.3

- Bash parser extended to be able to handle bash array expansions. This enables packages with more (Bash-wise) complex PKGBUILDs to build properly.

1.2.4 1.3.2.1

- *-Ai* and *-As* show popularity values.
- *aur4* is no longer referenced.
- *Yes/No* prompts are now localized.
- Aura can be built with *stack*.
- Updated German translation.

1.2.5 1.3.1.0

- Aura builds against GHC 7.10.
- Updated German and Russian translations.

1.2.6 1.3.0.4

- Must use *–builduser* when building as root.
- Bug fix regarding *–needed*.
- Updated Portuguese translation.

1.2.7 1.3.0.3

- Pacman flags *–ignore* and *–ignoregroup* now work.
- Bug fixes.

1.2.8 1.3.0.2

- (Bug fix) If a user tries to install a package in *IgnorePkg*, they will now be prompted.
- Man page updated.
- Dependencies updated.

1.2.9 1.3.0.1

- (Bug fix) Tarballs are now downloaded from a URL provided by the RPC.

1.2.10 1.3.0.0

- Last major version of Aura 1! We have entered the design phase for Aura 2, the implementation of which will transform Aura into a multi-distro package management platform.
- Aura 1 itself has entered "legacy" mode. The only releases to be made on Aura 1 after this will be of *1.3.0.x*. You'll likely never see *1.3.1.x*.
- Befitting a major release, we have:
  - New AUR interaction layer via the *aur* package. This fixes nasty "AUR lookup failed" errors.
  - *http-conduit* dropped for *wreq*, which is much easier to use.
  - Better version number parsing/comparison on installation/upgrading.
  - Package state backups have had their format changed. This BREAKS _all_ previously saved states. Please delete your old ones!
  - Implemented extended *–needed* functionality for the AUR side of Aura. AUR packages won't build if they're already installed.
  - Indonesian translations!
  - Other updated translations.

1.2.11 1.2.3.4

- zsh completions completely redone (thanks to Sauyon Lee!) Having *aur-git* installed will let you auto-complete on AUR packages.

1.2.12 1.2.3.3

- *-As –{head,tail}* can now be passed numbers to truncate the results to any number you want. The default is 10.
- Updated Russian translation.

1.2.13 1.2.3.2

- **Expanded Bash completions:**

  Aura Only

  - Expanded completion for all options and search sub-options
  - Package completion for -M/–abssync
  - Completion for orphans using self-generated list

  Pacman

  - Include completion for all pacman options
  - Directory or file completion for some common options
- Use *–dryrun* with *-A* and *-M* install options to test everything up until actual building would occur (dependency checks, etc.)

1.2.14 1.2.3.1

- Network.HTTP.Conduit errors are now caught properly and don't crash aura.
- *customizepkg* usage corrected.
- zsh completions slightly expanded.

1.2.15 1.2.3.0

- Moved to *Network.HTTP.Conduit* from *Network.Curl* This fixes the AUR connection issues. Binary size has increased by quite a bit.

1.2.16 1.2.2.1

- *-Ai* now shows dependencies.

1.2.17 1.2.2.0

- Happy New Year!
- makepkg's *–ignorearch* flag is now visible to Aura. This allows users to build AUR packages on ARM devices without worrying about architecture restrictions in PKGBUILDs.
- Use *–head* and *–tail* to truncate *-As* results.
- *-B* now uses local time.
- Bug fixes and translation updates

1.2.18 1.2.1.3

- *-As* results now sort by vote. Use *–abc* to sort alphabetically.
- "[installed]" will now be shown in *-As* results if you have it.
- Fixed Bash parsing bug involving \ in arrays
- Fixed broken *-C*
- Updated Italian translation
- Updated French translation

1.2.19 1.2.1.2

- Happy Canadian Thanksgiving
- Bug fixes

1.2.20 1.2.1.1

- Norwegian translation added!
- Dependency checks slightly faster
- *–hotedit* and *–custom* can now be used together
- Bug fixes

1.2.21 1.2.1.0

- New *builduser* option
- *Prelude.head* bug fixed
- Dependency checking is faster
- New *-k* output
- *–absdeps* works properly now
- Other bug fixes

1.2.22 1.2.0.2

- Bug fixes and spelling corrections.

1.2.23 1.2.0.1

- Fixes dependency build order bug.

1.2.24 1.2.0.0

- New operator *-M* for building ABS packages. Has its own family of options.
- Pre-built binary package available (x86_64 only)
- Updates to Aura are now prioritized like pacman updates.
- Dependency checking is now faster.
- Use *-Ccc* to clean the cache of only packages not saved in any package record.
- *-Ai* now shows Maintainer name.
- Extensive API changes.

1.2.25 1.1.6.2

- New option *–no-pp*. Disables use of powerpill, even if you have it.
- This is a light release, as major work is being done on version 1.2 on another development branch.

1.2.26 1.1.6.1

- Compatable with pacman 4.1
- All pacman-color support removed
- *-As* output slightly altered to match pacman.
- Bug fixes.

1.2.27 1.1.6.0

- New option *–build* for specifying AUR package build path.
- Vote number now shown in *-As* output.
- Fixed Repo/AUR name collision bug.
- API Change: Argument order for functions in *Aura/Languages* changed.

1.2.28 1.1.5.0

- *customizepkg* now usable with Aura. Activate with the *–custom* option.
- API Change: Aura/Pkgbuilds now a set of libraries as Aura/Pkgbuild/*

1.2.29 1.1.4.3

- Fixed flaw in *-Br*.
- Fixed repititious *-Ad* output.
- API Change: Aura/AurConnection renamed to Aura/AUR
- API Change: function names in Aura/Languages now have better names.

1.2.30 1.1.4.2

- Haskell deps have been moved back to *makedepends*.
- haskell-http removed as dependency.
- API Change: function naming conventions in *Aura/Languages.hs* has been changed. The localisation guide was also updated to reflect this.

1.2.31 1.1.4.1

- Support for the $LANG environment variable.
- Aura will now pause before post-build installation if the package database lock exists. This means you can run multiple instances of Aura and avoid crashes.

1.2.32 1.1.4.0

- Serbian translation added. Thank you, Filip Brcic!
- Fixed bug that was breaking *aura -Ss*.

1.2.33 1.1.3.0

- Changed *–save* and *–restore* to *-B* and *-Br*. *–save* is now just an alias for *-B*, but *–restore* must be used with *-B*.
- New option *-Bc* for removing old unneeded package states.
- *-Br* output is now sorted better and makes more sense.
- Bash Parser can now properly parse *if* blocks, meaning packages that have conditional dependencies based on architecutre will now build properly.
- API Change: *Aura.General* is now *Aura.Core*
- Dep Change: *haskell-url* no longer needed.

1.2.34 1.1.2.1

- Added message to *–save*.

1.2.35 1.1.2.0

- Bash parser completely rewritten.
- Bug fixes (thanks to the new parser)

1.2.36 1.1.1.0

- New option *–devel*. Rebuilds all devel packages installed.
- Italian translation added! Thank you Bob Valantin!
- Support for *powerpill* added. It will be used if installed, unless the PACMAN variable is specifically set to something different.
- Aura can now handle PKGBUILDs that produce multiple .pkg.tar files.
- Bug fixes

1.2.37 1.1.0.0

- New *–save* and *–restore* options.
- New option *-Ak* for showing PKGBUILD diffs when upgrading.
- New option *–aurignore* for ignoring AUR packages.
- Aura now reads *color.conf*.
- Massive breaking API changes everywhere.
- Aura now runs on the Aura Monad.

- Code is quite cleaner now.

1.2.38 1.0.8.1

- Bash completions added.
- zsh completions added.
- Changed *–conf* to *–viewconf*
- Fixed bug involving "symlink" Haskell error.

1.2.39 1.0.8.0

- Moved certain general functions to *Aura.Utils*
- Moved *-L*, *-O*, *-A* functions out of *aura.hs*.
- *–hotedit* functionality altered (fix).
- The license message is now more badass.

1.2.40 1.0.7.0

- New libraries: Aura.Time, Aura.State
- Moved *-C* functionality to *Aura.C*
- New secret option you don't get to find out about until 1.1
- Fixed manually alignment stupidity with *-Li*.
- Bug fixes

1.2.41 1.0.6.0

- New libraries: ColourDiff, Data.Algorithm.Diff, Aura.Pkgbuilds
- Aura.AuraLib split into Aura.General, Aura.Build, Aura.Dependencies
- New secret option you don't get to find out about until 1.1

1.2.42 1.0.5.0

- Fixed bug where packages with + in their name couldn't be searched or built.
- *-As* now allows multi-word searches, as it always should have.
- *pacman-color* integration is more complete. Still does not read the color.conf directly.

1.2.43 1.0.4.0

- Added French translation. Thanks to Ma Jiehong!
- Added Russian translation. Thanks to Kyrylo Silin!
- Fixed bug where packages with dots in their name wouldn't build.

1.2.44 1.0.3.2

- Moved haskell dependencies out of *makedepends* field and into *depends* field in PKGBUILD. Makedepends can usually be ignored after building, but haskell packages are a pain to rebuild and reregister at every build. It's more realistic to just keep them installed. This is what other haskell packages in the AUR do as well.
- Fixed pacman-color issues.

1.2.45 1.0.3.1

- Added *–auradebug* option.

1.2.46 1.0.3.0

- Compatibility with AUR 2.0 added.
- Portuguese translation added. Thanks to Henry "Ingvij" Kupty!
- **Support for *pacman-color* added. Run sudo with *-E* a la:** sudo -E aura -Ayu
- Fixed backslash parsing bug in *Bash*.

1.2.47 1.0.2.2

- Fixed parsing bug in *Bash*. If one package fell victim, a whole *-Au* session would fail.

1.2.48 1.0.2.1

- Added License info to source files.
- Fixed virtual package recognition bug.
- Altered version conflict error message.
- Fixed bug in Bash parser that would occasionally break parsing.

1.2.49 1.0.2.0

- Bug fixes.
- Extended the Bash parser. PKGBUILDs that had bash variables in their dependency arrays will now be parsed correctly.

1.2.50 1.0.1.0

- German translation (use with –german). Thanks to Lukas Niederbremer!
- Spanish translation (use with –spanish) Thanks to Alejandro GÃşmez!
- Replaced regex-posix with regex-pcre.
- *-As* now highlights properly.
- Moved a number of modules to *Aura/*

1.2.51 1.0.0.0

- Fixed *-V* message in terminals other than urxvt.
- New *haskell-ansi-terminal* library to do this.

1.2.52 0.10.0.0

- Internet access moved to Network.Curl library.
- *Bash.hs* library created to help with PKGBUILD parsing. Can currently handle string expansions a la:

```
"this-is-{awesome,neat}" => ["this-is-awesome","this-is-neat"]
```

1.2.53 0.9.2.3

- Dependency determining speed up.
- Added AUR URL to *-Ai*.

1.2.54 0.9.3.2

- Swedish translation. Thanks to Fredrik Haikarainen!

1.2.55 0.9.2.0

- *-Ai* and *-As*!

1.2.56 0.9.1.0

- *-Au* is about 20 times faster.

1.2.57 0.9.?.?

- Polish translation. Thanks to Chris "Kwpolska" Warrick!
- Croatian translation. Thanks to Denis Kasak!

1.2.58 0.9.0.0

- New *-O* operation for dealing with orphan packages.
- A man page!

1.2.59 0.8.0.0

- Help message now supports multiple languages.
- Broke "no overlapping options" convention.
- *-Cz* is now *-Cb*.
- New option *-Ad*. Lists _all_ dependencies of an AUR package. This is to aid pre-building research. This option shows information you can't get from looking at PKGBUILDS!

1.2.60 0.7.3.0

- New option *–conf*. Lets you quickly view your pacman.conf.

1.2.61 0.7.2.3

- *–log* is now *-L*.
- New option *-Ls*. Search the log file via a regex.
- New option *-Li*. Reports information on a given package that has had any appearance in the log file.

1.2.62 0.7.0.0

- *–hotedit* option added.
- *Shell* library added.

1.2.63 0.6.0.0

- Aura passes proper exit codes to the shell upon completion.

1.3 Aura Development Roadmap

1.3.1 1.2.0.0

- The Great Abstraction
- New *-M* option for building packages from the ABS.
- This will make *A* and *M* run on the same backend.
- Depedency checks will be unrestricted and beautiful.

1.3.2 1.3.0.0

- Milestone for "legacy" Aura. Last stable version as a pacman-reliant AUR helper.

1.3.3 2.0.0.0

- Aura becomes a core for multi-distro package management.
- It exposes a Hook interface for writing distro-specific install behaviour.
- These Hooks may or may not bolt directly to a preexisting package manager, as Aura 1 did.
- The preferred method is to write Haskell bindings to the libraries those managers use, for more direct control.

CHAPTER 2

Guides

2.1 Language Localisation Guide

Welcome!

ãĄŞãĆŞãĄńãĄąãĄŕïijĄ

If you're reading this then it's likely that you want to help localise Aura into another language. Arch users everywhere can benefit from your contribution, and its a great opportunity to contribute to Open Source.

2.1.1 What You Need

1. The aura source code. Get it at: https://github.com/aurapm/aura
2. An editor. Whichever one you like.

   Vim users, run the following easter-egg command to unlock a better version:

```
:! perl -e "system('hpdfv' =~ tr/a-z/x-za-w/r)"
```

   Emacs users can achieve a similar enhanced version with:

```
M-! perl -e "system('ylp' =~ tr/a-z/x-za-w/r)"
```

3. **`git`**. As Aura is hosted on github, cloning, making changes and adding pull requests will be easiest if you have a working git/github setup.
4. Minimal Haskell knowledge. You'll see just how much below.
5. A brain (hopefully yours) with a language in it. As for me, no hablo espaÃśol, Đŕ Đ¡Đţ ĐijĐ¿ĐşÑČ ĐşĐ¿ĐšĐ¿ÑĂĐÿÑĆÑŇ Đ£Đ¿-ÑĂÑČÑĄÑĄĐžĐÿ, nor do I ÙŁØłÙČÙĎÙĚ ØğÙĎØźØśØÍÙŁØľ, so that's where you come in.

2.1.2 Getting Started

Step One - Tell Haskell About the New Language

All strings that contain messages for the user are stored in a single source file: `src/Aura/Languages.hs`. Let's take a look at the top:

```
data Language = English
              | Japanese
                deriving (Eq,Enum,Show)
```

This is where we define output languages for Aura. For the purpose of demonstration, we'll use `French` as the language we're adding. Add a new language by adding a new value to the Language data type. Like this:

```
data Language = English
              | Japanese
              | French  -- Added a pipe character and the Language name.
                deriving (Eq,Enum,Show)
```

Step Two - Adding your Language's Locale Code

See the function `langFromEnv`. It is given the contents of the environment variable `LANG`, and checks the first two characters, returning the appropriate name of the language entered in the `Language` field above. English is the default.

```
langFromEnv :: String -> Language
langFromEnv = \case
    "ja" -> Japanese
    _    -> English
```

For French, we would add a new field above `English`.

```
langFromEnv :: String -> Language
langFromEnv = \case
    "ja" -> Japanese
    "fr" -> French
    _    -> English
```

Don't know your locale code? You can find them all in `/usr/share/i18n/locales`.

Step Three - Translation

This is the real work. Let's take a look at a simple message. The user has passed some bogus/conflicting flags to Aura. What to tell them?

```
-----------------
-- aura functions
-----------------
executeOpts_1 :: Language -> String
executeOpts_1 = \case
    Japanese -> "ç§Żçż¿ãĄůãĄęãĄĎãĆŃãĆłãČůãĆůãČğãČşãĄĆãĆłãĂĆ"
    _        -> "Conflicting flags given!"
```

All functions in Aura code that output messages to the user get that message with a dispatch. That is, they call a function with the current language they're using, and that function returns the appropriate message.

Notice the handy label in the comment there. This tells *where* in the Aura code the calling function is located. If you ever need more context as to what kind of message you're writing, checking the code directly will be quickest. The format is:

```
SomeLanguage -> "The message."
```

This naming is nothing more than a convention. So let's go ahead and add the French message:

```
----------------
-- aura functions
----------------
executeOpts_1 :: Language -> String
executeOpts_1 = \case
    Japanese -> "ç$Żçż¿ãĄůãĄęãĄĎãĆŃãĆłãČůãĆůãČğãČşãĄĆãĆŁãĂĆ"
    French   -> "Arguments contradictoires!"
    _        -> "Conflicting flags given!"
```

Sometimes you'll get functions with extra variables to put in the message:

```
------------------------
-- Aura/Build functions
------------------------
buildPackages_1 :: String -> Language -> String
buildPackages_1 (bt -> p) = \case
    Japanese   -> p ++ "ãĆŠä¡IJæĹŘäÿ■ãČżãČżãČż"
    _          -> "Building " ++ p ++ "..."
```

What the heck is `p`? Well it's probably a package name. To double check, just check out the function that calls this message dispatch. We know it's in `src/Aura/Build.hs`, and the function is called `buildPackages`. Once you know what's going on, go ahead and add the translation:

```
------------------------
-- Aura/Build functions
------------------------
buildPackages_1 :: String -> Language -> String
buildPackages_1 (bt -> p) = \case
    Japanese   -> p ++ "ãĆŠä¡IJæĹŘäÿ■ãČżãČżãČż"
    French     -> "Construction de " ++ p ++ "âĂę"
    _          -> "Building " ++ p ++ "..."
```

Obviously the syntax among languages is different, and so where you insert the variables you've been given into the sentence depends on your language.

Also, I enjoy backticks. As a convention I wrap up all package names in these messages in backticks, using the `bt` function as seen in the examples. This also colours them cyan.

Step Four - Command-line Flag

We choose output languages in Aura by using flags on the command line. Japanese, for example, uses the **`--japanese`** flag. We'll have to make a flag for the new language you're adding too.

This step is not actually necessary for you to do... so long as the translations are done I can take care of the rest of the code editing. But for the interested, edit `src/Aura/Flags.hs`:

```
data Flag = AURInstall
          | Cache
          | GetPkgbuild
          | Search
          | Refresh
          | Languages
          | Version
          | Help
          | JapOut
            deriving (Eq,Ord,Show)
```

You could add French like this:

```
data Flag = AURInstall
          | Cache
          | GetPkgbuild
          | Search
          | Refresh
          | Languages
          | Version
          | Help
          | JapOut
          | FrenchOut  -- Here.
            deriving (Eq,Ord,Show)
```

Then we need to add it to the options to be checked for, edit `Aura/Flags.hs`:

```
languageOptions :: [OptDescr Flag]
languageOptions = map simpleMakeOption
                  [ ( [], ["japanese","æŮěæIJňèłđ"], JapOut ) ]
```

...would thus become:

```
languageOptions :: [OptDescr Flag]
languageOptions = map simpleMakeOption
                  [ ( [], ["japanese","æŮěæIJňèłđ"],  JapOut     )
                  , ( [], ["french", "franÃğais"], FrenchOut ) ]
```

Notice how each language has two long options. Please feel free to add your language's *real* name in its native characters.

Last step in the flag making:

```
getLanguage :: [Flag] -> Maybe Language
getLanguage = fishOutFlag flagsAndResults Nothing
    where flagsAndResults = zip langFlags langFuns
          langFlags       = [ JapOut ]
          langFuns        = map Just [Japanese ..]
```

This function extracts your language selection from the rest of the options. Let's add French.

```
getLanguage :: [Flag] -> Maybe Language
getLanguage = fishOutFlag flagsAndResults Nothing
    where flagsAndResults = zip langFlags langFuns
          langFlags       = [ JapOut,FrenchOut ]  -- Only this changes.
          langFuns        = map Just [Japanese ..]
```

Where `FrenchOut` is the value you added to `Flags` above.

Step Five - Pull Request

With the translations complete, you'll need to tell us about it on github. Once your changes are looked over, we'll release a new version of Aura with your language included as soon as possible. Provided you followed the above instructions, this shouldn't take long. Furthermore, chances are we won't be able to proofread the translation itself, as we probably don't speak your language. You could hide your doomsday take-over plans in the code and no one would know.

Step Six - You've Helped Others who Speak your Language

You've done a great thing by increasing Aura's usability. Your name will be included in both Aura's README and in its **-V** version message. Thanks a lot for your hard work!

2.2 Hacking Aura

Hi. You're either reading this because you want to make Aura better, or because you want to study some Haskell. Great!

2.2.1 For Haskell Study

Aura code has examples of:

- Monad Transformers => `src/Aura/Monad/Aura.hs`
- Parsec => Bash/Parser.hs - Applicative Functors => `src/Bash/Parser.hs`
- Regular Expressions => `src/Aura/Utils.hs`
- CLI flag handling => `src/Aura/Flags.hs`
- Shell escape codes => `src/Aura/Colour/Text.hs` or `src/Shell.hs`

2.2.2 For Aura Hacking

The `main` function is housed in `src/aura.hs`. All function dispatches occur here. General libraries also housed in the root folder:

- `src/Bash/` (A custom Bash script parser and simplifier)
- `src/Data/Algorithm/Diff` (A classic diff algorithm)
- `src/Network/HTTP` (A copy of key functions from Network.HTTP)
- `src/Internet` (For https requests)
- `src/Shell` (Shell access in the IO Monad)
- `src/Utilities` (Random helper functions)

Aura specific libraries are housed in `Aura/`. The main areas are:

- `src/Aura/` (General Aura-specific libraries)
- `src/Aura/Commands/` (Functions that back the main capital letter Aura operations)
- `src/Aura/Monad/` (Everything to do with the Aura Monad)
- `src/Aura/Packages/` (Backends for handling various package types)
- `src/Aura/Pkgbuild/` (Functions for handling PKGBUILDs)
- `src/Aura/Settings/` (`Settings` for the ReaderT portion of the Aura Monad)

2.2.3 The Aura Monad

Many functions in the Aura code are within the Aura Monad. The Aura Monad is a stack of Monad Transformers, but is in essence a glorified IO Monad.

```
{-# LANGUAGE GeneralizedNewtypeDeriving #-}

import Control.Monad.Reader
import Control.Monad.Error
```

```
import Aura.Settings.Base (Settings)

---

newtype Aura a = A { runA :: ErrorT AuraError (ReaderT Settings IO) a }
  deriving (Monad, MonadError AuraError, MonadReader Settings, MonadIO, Functor)

runAura :: Aura a -> Settings -> IO (Either AuraError a)
runAura a = runReaderT $ runErrorT (runA a)

data AuraError = M String deriving (Eq,Show)

instance Error AuraError where
    noMsg  = strMsg "No error message given."
    strMsg = M
```

The Aura Monad is an `ErrorT` at the top, meaning its binding (>>=) behaviour is the same as an `Error` Monad. This allows failure to halt actions partway through.

It is a `ReaderT` and has a MonadReader instance, meaning we can obtain the local runtime settings by using the `ask` function anywhere we wish in a function within the Aura Monad.

It is `IO` at its base and has a MonadIO instance, meaning we can perform IO actions with `liftIO` in any function in the Aura Monad. To extract it's inner value, we use the helper function `runAura`.

Why the Aura Monad?

The Aura Monad is convenient for two reasons:

1. The local runtime settings are referenced heavily throughout the Aura code. Passing a `Settings` parameter around explicitely makes for long function signatures. Furthmore, being accessed from an internal Reader Monad also means its access is *read-only*. This way, the run-time settings could never be altered unknowingly.
2. Being an `ErrorT`, it can fail. These failures can also be caught elegantly, demanding no need for try/catch blocks a la imperitive languages. Example:

```
    foo :: Whatever -> Aura Whatever
    foo w = risky w >>= more >>= evenMore >>= most
```

Here, if `risky` fails, `more`, `evenMore`, and `most` will never execute. Anything binding `foo` at a higher level would also fail accordingly if not caught.

In essence, if you've ever programmed in a language with error handling and an idea of constant global variables, you've programmed in the Aura Monad.

Notes on Aura Monad Style

Access to `Settings` is frequently needed, thus calls to `ask` are plentiful. When writing a function in the Aura Monad with `do` notation and calling `ask`, please do so in the following way:

```
foo :: Whatever -> Aura Whatever
foo w = ask >>= \ss -> do
  ...  -- Rest of the function.
```

If you only need one function out of `Settings`, you can use `asks`, which directly applies a function to the result of `ask`:

```
-- For example, if I only need the cache path from Settings...
foo :: Whatever -> Aura Whatever
foo w = asks cachePathOf >>= \path -> do
  ...  -- Rest of the function.
```

The idea is to keep interaction with `ask` to the first line, before `do`.

2.2.4 String Dispatching

No Strings meant for user-viewed output are hardcoded. All current translations of all Strings are kept in `Aura/Languages.hs`. Messages are fetched by helper functions after being passed the current runtime `Language` stored in `Settings`. This leads to:

1. More advanced String manipulation, regardless of spoken language.
2. More convenient translation work.
3. (Unfortunately) larger executable size.

See the Language Localisation Guide for more information.

2.3 Automatic Package Record Backups with Cron

When upgrading packages, sometimes things just go wrong. I tend to keep on top of my upgrading and my `.pacnew` files, but even so I'll get a system-crippling update maybe twice a year - just frequent enough to keep me on my toes.

`aura -B` comes to the rescue at times like these. Options prefixed with the **`-B`** operator manage the saving and restoring of package records. **`-B`** offers a light-weight way of backing up your system without doing hard copies of all the files in your package cache.

By setting up cron jobs, we can automate the process of saving these records and clearing out old ones. To set up cron jobs, we need to edit our **`crontab`**. A **`crontab`** is a listing of timings and programs to run. It's a schedule for our cron jobs. Since **`aura -B`** is typically ran with **`sudo`**, we'll need to edit root's crontab. You can do this with **`sudo crontab -e`**. This will open root's default editor.

See **`man 5 crontab`** for more details into the editing of crontabs. For now, simply copying the following two entries can get you started:

```
# Save a package record at 8pm on Wednesday and Sunday every week.
0 20 * * wed,sun /usr/bin/aura -B

# Reduce the package record number to 5 once a month at 8:01pm.
1 20 1 * * /usr/bin/aura -Bc 5 --noconfirm
```

Since both of these entries would touch the same files, we set them one minute apart to prevent any problems. With these in place, if any upgrade problems occur, we know our most recent save was at most only three days ago.

Save and exit, and you're all set. Happy package managing!

P.S. I also use the following job to keep the size of my package cache down:

```
# Clean out the package cache once a month in the same way.
0 20 1 * * /usr/bin/aura -Cc 5 --noconfirm
```

CHAPTER 3

Announcements

3.1 Aura 1.1 Release

For you, Arch users, the fruits of my winter holiday: Aura 1.1

Aura is a package manager for Arch Linux with full support for installing and upgrading AUR packages. With version 1.1 it has many new features to make managing your system easier.

3.1.1 New with Version 1.1

- `--save` option. Stores a record of all installed packages.
- `--restore` option. Restores a state stored with `--save`. Good for reversing system breakage.
- `-k` suboption of `-A`. Shows PKGBUILD diffs when installing / upgrading.
- `--aurignore` option. Ignores given AUR packages when installing / upgrading.
- Bash and zsh completions now available.
- Aura now reads pacman-color's /etc/pacman.d/color.conf if available. This affects the colours in `-Ai` and `-As`.
- Aura now stores the most recent PKGBUILD when installing a package. This is so `-Ak` has something to diff with when upgrading next.
- Much cleaner code.

3.1.2 On the Horizon

- Fish completions.
- Reworked PKGBUILD parser.
- Haskell binding to libalpm.

A big thanks to all of Aura's users! I couldn't have got this far without your support.

Aura Github Page Aura Bitbucket Page Aura AUR Page Aura Wiki Page