

Inary Handbook

Welcome to inary documentation.

This documentation contains information about inary package management for everyone, from end users to developers, even though you won't read it.

- [About Inary Software](#)
- [Installation of Inary](#)
 - [Installation under another distro](#)
 - [Installation on SulinOS](#)
 - [Upgrade inary](#)
 - [Package Post install Processes](#)
- [Inary Commands](#)
 - [inary blame](#)
 - [inary build](#)
 - [inary check](#)
 - [inary configure-pending](#)
 - [inary delete-cache](#)
 - [inary delta](#)
 - [inary emerge](#)
 - [inary emergeup](#)
 - [inary fetch](#)
 - [inary graph](#)
 - [inary info](#)
 - [inary install](#)
 - [inary help](#)
 - [inary history](#)
 - [inary index](#)
 - [inary rebuild-db](#)
 - [inary remove](#)
 - [inary remove-orphaned](#)
 - [inary search](#)
 - [inary search-file](#)
 - [inary upgrade](#)

- [Managing Repositories in INARY](#)
 - [Other Commands in INARY](#)
- [Frequently Asked Questions](#)
- [Glossary](#)

Bu belgeleme AGPL-3 ile lisanslanmıştır.

About Inary Software

PiSi fork: Copyright (C) 2005 - 2011, Tubitak / UEKAE Licensed
with GNU / General Public License version 2.

Inary fork and enhancements: Copyright (C) 2016 - 2018 Suleyman
POYRAZ (Zaryob) License has been upgraded to version 3 of the
GNU / General Public License.

Bifurcated from <https://github.com/Pardus-Linux/pisi>

Inary package management system started to work on the pisi fork on 21-12-2016 in order to repair the deficiencies and errors of the existing pisi package manager and to port it to python3 to catch up with the latest developments in Python Programming language; The first pisi fork (spam) was separated from the fork in terms of coding method and additional modules used and turned into a unique one. The name of the library has been changed from pisi to inary. The reason for this is to avoid conflicts with linux distributions that use similar infrastructure, such as [Solus](https://dev.sol.us/) [https://dev.sol.us/] and [PisiLinux](https://www.pisilinux.org) [https://www.pisilinux.org].

The software was called inary and was made available to the public through the [gitlab](https://gitlab.com/SulinOS) [https://gitlab.com/SulinOS] and [github](https://github.com/SulinOS) [https://github.com/SulinOS] when it provided the appropriate conditions for the end users.

What distinguishes it from other package management systems:

- Has dynamic file database. It is easy to follow up whether there is a change in the installed files.
- Compared to other package managers coded with Python, it is quite fast.
- Since all the installation script consists of python script. Since the other data of the package is stored in xml files, package building steps can be done without entering tons of code.

- Post-package and post-package operations (postinstall) do not cause any process confusion by separate software.

Other features:

- It is robust and fast because it works with a database embedded in python.
- Because it uses LZMA and XZ compression methods, it has smaller packages.
- Simple and high-level operations with the same determination
- It includes an API suitable for designing forend applications.
- Terminal interface is very understandable and user-friendly.

Installation of Inary

Installation under another distro

If you're using another distribution, inary can be used in two ways

- to completely switch to sulin: with inary it's easy to switch to sulin without downloading and reinstalling iso
- to install programs in your userspace: Like brew package manager

Use to switch to Sulin from another distro

```
sh ~# git clone https://github.com/SulinOS/inary.git
sh ~# cd inary
sh ~# python3 setup.py install
sh ~# cd ..
sh ~# git clone https://github.com/SulinOS/switch-sulin.git
sh ~# cd switch-sulin
sh ~# ./switch-sulin --interactive
sh ~# inary rdb
sh ~# inary cp
sh ~# shutdown -r now
```

Brew-Like using

```
sh ~$ mkdir ~/.inary
sh ~$ git clone https://github.com/SulinOS/inary.git
sh ~$ python3 setup.py build
sh ~$ python3 setup.py install --root=~/.inary
sh ~$ python3 scripts/inary-profile select brew-like
sh ~$ inary rdb
```

Installation on SulinOS

In the distribution of Sulin, Inary comes preloaded.

Note

you can install inary following steps to installing any other distro, but this could break the preloaded inary package.

Upgrade inary

Inary allows self-updating...

```
sh ~$ inary ur
sh ~$ inary up inary
```

...or you can also upgrade it from exist cloned repository

```
sh ~# cd inary
sh ~# git pull
sh ~# python3 setup.py install
```

Package Post install Processes

For inary package management system, Comar has been forked and developed and scom is a former inary dependency and contains pisi package configuration scripts.

Unless otherwise specified (scom script running can be ignored with using *-ignore-scom* parameter), scom scripts will be executed after package installation.

However, unlike in the past, existing inary allows packet processing without using scom. In this case, the configuration after the installation of the packages is left entirely to the users hand.

Inary Commands

- [inary blame](#)
 - [Using](#)
 - [Options](#)
 - [Example Runtime](#)
- [inary build](#)
 - [Using](#)
 - [Options](#)
 - [Example Runtime Output](#)
- [inary check](#)
 - [Using](#)
 - [Options](#)
 - [Example Runtime Output](#)
- [inary configure-pending](#)
 - [Using](#)
 - [Example Runtime Output](#)
- [inary delete-cache](#)
 - [Using](#)
 - [Example Runtime Output](#)
- [inary delta](#)
 - [Using](#)
 - [Options](#)
 - [Example Runtime Output](#)
- [inary emerge](#)
 - [Using](#)
 - [Options](#)
 - [Example Runtime Output](#)
- [inary emergeup](#)
 - [Using](#)
 - [Options](#)
 - [Example Runtime Output](#)
- [inary fetch](#)
 - [Using](#)
 - [Options](#)

- [Example Runtime](#)
- [inary_graph](#)
 - [Output](#)
 - [Example Runtime Output](#)
- [inary_info](#)
 - [Using](#)
 - [Options](#)
 - [Example Runtime](#)
- [inary_install](#)
 - [Using](#)
 - [Options](#)
 - [Example Runtime](#)
- [inary_help](#)
 - [Using](#)
 - [Output](#)
- [inary_history](#)
 - [Using](#)
 - [Options](#)
 - [Hints](#)
 - [Example Runtime Output](#)
- [inary_index](#)
 - [Using](#)
 - [Options](#)
 - [Example Runtime Output](#)
- [inary_rebuild-db](#)
 - [Using](#)
 - [Options](#)
 - [Example Runtime Output](#)
- [inary_remove](#)
 - [Using](#)
 - [Options](#)
 - [Example Runtime](#)
- [inary_remove-orphaned](#)
 - [Using](#)
 - [Options](#)
 - [Example Runtime](#)
- [inary_search](#)

- [Using](#)
 - [Options](#)
 - [Example Runtime](#)
- [inary search-file](#)
 - [Using](#)
 - [Options](#)
 - [Example Runtime](#)
- [inary upgrade](#)
 - [Using](#)
 - [Options](#)
 - [Example Runtime](#)
- [Managing Repositories in INARY](#)
 - [Adding Repository.](#)
 - [Options](#)
 - [Using](#)
 - [Hints](#)
 - [Removing Repository.](#)
 - [Using](#)
 - [Update Repository.](#)
 - [Options](#)
 - [Using](#)
 - [Hints](#)
 - [Changing Activity of a Repository.](#)
 - [Enabling Repository.](#)
 - [Disabling Repository.](#)
- [Other Commands in INARY](#)
 - [inary list-available](#)
 - [Using](#)
 - [Options](#)
 - [inary list-sources](#)
 - [Using](#)
 - [Options](#)
 - [inary list-newest](#)
 - [Using](#)
 - [Options](#)
 - [inary list-upgrades](#)
 - [Using](#)

- [Options](#)
- [inary list-installed](#)
 - [Using](#)
 - [Options](#)
- [inary list-repo](#)
 - [Using](#)
- [inary list-orphaned](#)
 - [Using](#)
 - [Options](#)
- [inary list-components](#)
 - [Using](#)
 - [Options](#)

inary blame

inary blame gives information of packager name and email from [installdb](#) or package file.

blame operation gives information under history tag rather than giving detailed information

Using

```
sh ~$ inary blame <package-name or package-file>
sh ~$ inary bl <package-name or package-file>
```

Options

blame options:

<code>-r, --release</code>	Blame for the given release
<code>-a, --all</code>	Blame for the given release

Example Runtime

```
$ inary blame expat
Name: expat, version: 2.2.6, release: 1
Package Maintainer: Süleyman POYRAZ <zaryob.dev@gmail.com>
Release Updater: Süleyman POYRAZ <zaryob.dev@gmail.com>
Update Date: 2018-12-23
```

First release

inary build

inary build command is used to build binary packages from source files. Building operation can be made with giving [pspec.xml](#) path or source name whether any source repository have been added.

Using

Build command can be used two ways.

```
sh ~$ inary build <pspec-file>
sh ~$ inary bi <package-file>
```

and also used with giving source name (unlike emerge, only the package creations happens)

```
sh ~$ inary build <source-package-name>
sh ~$ inary bi <source-package-name>
```

Options

build options:

<code>-q, --quiet</code>	Run inary build operation without printing extra debug information.
<code>--ignore-dependency</code>	Do not take dependency information into account.
<code>-O, --output-dir</code>	Output directory for produced packages.
<code>--ignore-action-errors</code>	Bypass errors from ActionsAPI.
<code>--ignore-safety</code>	Bypass safety switch.
<code>--ignore-check</code>	Bypass testing step.

<code>--create-static</code>	Create a static package with ar files.
<code>-F, --package-format</code>	Create the binary package using the given format. Use ‘-F help’ to see a list of supported formats.
<code>--use-quilt</code>	Use quilt patch management system instead of GNU patch.
<code>--ignore-sandbox</code>	Do not constrain build process inside the build folder.

The inary package manager allows you to do the building step by step (each step is specified in the [actions.py](#) script as a function.) If an error occurs in any of these steps, the construction can be resumed from the step left. These step specifications is also given to build command as parameters.

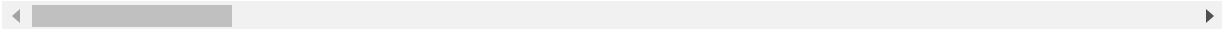
Build Steps:

<code>--fetch</code>	Break build after fetching the source archive.
<code>--unpack</code>	Break build after unpacking the source archive, checking sha1sum and applying patches.
<code>--setup</code>	Break build after running configure step.
<code>--build</code>	Break build after running compile step.
<code>--check</code>	Break build after running check step.
<code>--install</code>	Break build after running install step.
<code>--package</code>	Create INARY package.

Example Runtime Output

```
sh ~$ inary build
Building source package: "expat"
expat-2.2.6.tar.bz2 [cached]
>>> Unpacking archive(s)...
-> (/var/inary/expat-2.2.6-1/work) unpacked.
```

```
>>> Setting up source...
GNU Config Update Finished.
GNU Config Update Finished.
[Running Command]: ./configure --prefix=/usr
>>> Build source...
[Running Command]: make -j5
>>> Installing...
[Running Command]: make DESTDIR=/var/inariy/expat-2.2.6-1/install
[Running Command]: install -m 0644 "doc/expat.png" /var/inariy/ex
[Running Command]: install -m 0644 "doc/valid-xhtml10.png" /var/
[Running Command]: install -m 0644 "doc/reference.html" /var/inariy/ex
[Running Command]: install -m 0644 "doc/style.css" /var/inariy/ex
[Running Command]: install -m 0644 "Changes" /var/inariy/expat-2.
[Running Command]: install -m 0644 "README.md" /var/inariy/expat-
Removing special "libtool", file: "/var/inariy/expat-2.2.6-1/inst
Building package: "expat"
Creating "expat-2.2.6-1-s19-x86_64.inariy"...
Building package: "expat-devel"
Creating "expat-devel-2.2.6-1-s19-x86_64.inariy"...
Building package: "expat-docs"
Creating "expat-docs-2.2.6-1-s19-x86_64.inariy"...
Building package: "expat-pages"
Creating "expat-pages-2.2.6-1-s19-x86_64.inariy"...
Keeping build directory
*** 0 error, 1 warning
```



inary check

inary check command performs a detailed analysis of the packages installed in the system. If there is content that is broken or deleted, or a modified config file, it allows us to see.

Using

check command analyze only the named package if the argument is given...

```
sh ~$ inary check <package-name>
```

...but analyzes all system if no argument is given.

```
sh ~$ inary check
```

Options

check options:

<code>-c, --component</code>	Check installed packages under given component.
<code>--config</code>	Checks only changed config files of the packages.

Example Runtime Output

```
sh ~$ inary check
Checking integrity of "acl"          OK
Checking integrity of "bash"        OK
Checking integrity of "unzip"       OK
Checking integrity of "tar"         OK

sh ~$ inary check acl
Checking integrity of "acl"          OK
```

inary configure-pending

inary configure-pending is used to complete the post-install scripts of packages installed on the system, but did not process the scripts after installation

Note

This operation needs privileges and can be allowed by only super user.

Using

configure-pending operation configures only the named package if the argument is given...

```
sh ~# inary configure-pending <package-name>
sh ~# inary cp <package-name>
```

...but executes post-install scripts for all pending packages if no argument is given

```
sh ~# inary configure-pending
sh ~# inary cp
```

Example Runtime Output

```
sh ~# inary configure-pending acl bash unzip gcc binutils tar
Configuring "acl".
Configuring "acl" package.
Configured "acl".
Configuring "bash".
Configuring "bash" package.
Configured "bash".
Configuring "unzip".
Configuring "unzip" package.
Configured "unzip".
```



```
Configuring "tar".  
Configuring "tar" package.  
Configured "tar".
```

inary delete-cache

inary delete-cache allows to clean inary's temporary files package and source code archives. It does not take any other argument, or parameter.

Note

This operation needs privileges and can be allowed by only super user.

Using

```
sh ~# inary dc
sh ~# inary delete-cache
```

Example Runtime Output

```
sh ~# inary delete-cache
Cleaning package cache "/var/cache/inary/packages"...
Cleaning source archive cache "/var/cache/inary/archives"...
Cleaning temporary directory "/var/inary"...
Removing cache file "/var/cache/inary/packagedb.cache"...
Removing cache file "/var/cache/inary/sourcedb.cache"...
Removing cache file "/var/cache/inary/componentdb.cache"...
Removing cache file "/var/cache/inary/groupdb.cache"...
```

inary delta

The *inary delta* command is used to create delta packages from two different release of packages. It creates a delta package of only the changed files.

This allows us to minimize updates and reduce the data network less regularly. allows us to use.

Note

delta packets are recommended only for packages created from the same version source.

Using

```
sh ~$ inary delta <older-release-of-package> <newer-release-of-p
```



Options

delta options:

- t, --newest-package Use arg as the new package and treat other arguments as old packages.
- O, --output-dir Output directory for produced packages.
- F, --package-format Create the binary package using the given format. Use '-F help' to see a list of supported formats.

Example Runtime Output

```
sh ~$ inary delta expat-2.2.6-1-s19-x86_64.inary expat-2.2.6-2-s  
Creating delta package: "expat-1-2-s19-x86_64.delta.inary"...
```



inary emerge

inary emerge command is used to build a source package from the source repository and to install the system.

Note

This operation needs privileges and can be allowed by only super user.

Using

emerge operation takes an argument which is source package name.

```
sh ~# inary emerge <package-name>
sh ~# inary em <package-name>
```

Options

emerge options:

- | | |
|---|---|
| <code>-c, --component</code> | Emerge available packages under given component |
| <code>--ignore-file-conflicts</code> | Ignore file conflicts. |
| <code>--ignore-package-conflicts</code> | Ignore package conflicts. |
| <code>--ignore-scom</code> | Bypass scom configuration agent. |

Example Runtime Output

```
sh ~$ inary emerge expat
Building source package: "expat"
expat-2.2.6.tar.bz2 [cached]
```

```
>>> Unpacking archive(s)...
-> (/var/inariy/expat-2.2.6-1/work) unpacked.
>>> Setting up source...
GNU Config Update Finished.
GNU Config Update Finished.
[Running Command]: ./configure --prefix=/usr
>>> Build source...
[Running Command]: make -j5
>>> Installing...
[Running Command]: make DESTDIR=/var/inariy/expat-2.2.6-1/install
[Running Command]: install -m 0644 "doc/expat.png" /var/inariy/ex
[Running Command]: install -m 0644 "doc/valid-xhtml10.png" /var/
[Running Command]: install -m 0644 "doc/reference.html" /var/inariy/ex
[Running Command]: install -m 0644 "doc/style.css" /var/inariy/ex
[Running Command]: install -m 0644 "Changes" /var/inariy/expat-2.
[Running Command]: install -m 0644 "README.md" /var/inariy/expat-
Removing special "libtool", file: "/var/inariy/expat-2.2.6-1/inst
Building package: "expat"
Creating "expat-2.2.6-1-s19-x86_64.inariy"...
Building package: "expat-devel"
Creating "expat-devel-2.2.6-1-s19-x86_64.inariy"...
Building package: "expat-docs"
Creating "expat-docs-2.2.6-1-s19-x86_64.inariy"...
Building package: "expat-pages"
Creating "expat-pages-2.2.6-1-s19-x86_64.inariy"...
Keeping build directory
Installing package "expat" version 2.2.6 release 1
Extracting files of "expat" package.
Adding files of "expat" to database...
Installing "expat".
```



inary emergeup

inary emergeup command analyzes the upgraded source packages and it is used to build a source package from the source repository and to install the system. So it supplies to upgrade packages with building in your system.

Note

This operation needs privileges and can be allowed by only super user.

Using

emergeup operation builds an upgrades one specific package if the argument is given...

```
sh ~# inary emergeup <package-name>
sh ~# inary emup <package-name>
```

...but upgrades all waiting source packages if no argument is given.

```
sh ~# inary emergeup
sh ~# inary emup
```

Options

emerge options:

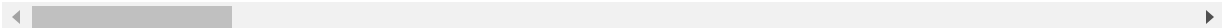
- c, --component Emerge available packages under given component
- ignore-file-conflicts Ignore file conflicts.
- ignore-package-conflicts Ignore package conflicts.

--ignore-scom

Bypass scom configuration agent.

Example Runtime Output

```
sh ~$ inary emergeup expat
Building source package: "expat"
expat-2.2.6.tar.bz2 [cached]
>>> Unpacking archive(s)...
  -> (/var/inary/expat-2.2.6-1/work) unpacked.
>>> Setting up source...
GNU Config Update Finished.
GNU Config Update Finished.
[Running Command]: ./configure --prefix=/usr
>>> Build source...
[Running Command]: make -j5
>>> Installing...
[Running Command]: make DESTDIR=/var/inary/expat-2.2.6-1/install
[Running Command]: install -m 0644 "doc/expat.png" /var/inary/ex
[Running Command]: install -m 0644 "doc/valid-xhtml10.png" /var/
[Running Command]: install -m 0644 "doc/reference.html" /var/in
[Running Command]: install -m 0644 "doc/style.css" /var/inary/ex
[Running Command]: install -m 0644 "Changes" /var/inary/expat-2.
[Running Command]: install -m 0644 "README.md" /var/inary/expat-
Removing special "libtool", file: "/var/inary/expat-2.2.6-1/inst
Building package: "expat"
Creating "expat-2.2.6-2-s19-x86_64.inary"...
Building package: "expat-devel"
Creating "expat-devel-2.2.6-2-s19-x86_64.inary"...
Building package: "expat-docs"
Creating "expat-docs-2.2.6-2-s19-x86_64.inary"...
Building package: "expat-pages"
Creating "expat-pages-2.2.6-2-s19-x86_64.inary"...
Keeping build directory
Installing package "expat" version 2.2.6 release 2
Extracting files of "expat" package.
Adding files of "expat" to database...
Installing "expat".
```



inary fetch

inary fetch is used to download packages from the repository. It fetches binary packages to working directory

Using

```
sh ~$ inary fetch <package-name>
sh ~$ inary fc <package-name>
```

Options

fetch options:

<code>-o, --output-dir</code>	Output directory for the fetched packages
<code>--runtime-deps</code>	Download with runtime dependencies.

Example Runtime

```
sh ~$ inary fc expat
"expat" package found in "core" repository.
expat-2.2.7-2-s19-x86_64.inary (194.0 KB)
```

inary graph

inary graph command draws the dependency graphic between the repositories attached to the system and creates an dot file.

Output

graph options:

<code>-r, --repository</code>	Specify a particular repository.
<code>-i, --installed</code>	Graph of installed packages
<code>--ignore-installed</code>	Do not show installed packages.
<code>-R, --reverse</code>	Draw reverse dependency graph.
<code>-o, --output</code>	Dot output file

Example Runtime Output

```
sh ~$ inary graph
Plotting a graph of relations among all repository packages.
```

inary info

inary info shows detailed information including provides, dependencies, component, etc. Unlike blame command, it allows us to see all the components written in [pspec.xml](#)

Using

```
sh ~$ inary info <package-name>
```

Options

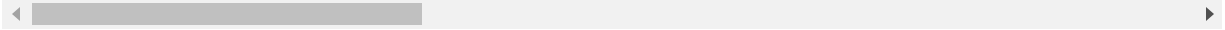
info options:

<code>-f, --files</code>	Show a list of package files.
<code>-c, --component</code>	Info about the given component.
<code>-F, --files-path</code>	Show only paths.
<code>-s, --short</code>	Do not show details.
<code>--xml</code>	Output in xml format.

Example Runtime

```
sh ~$ ianry info expat
"expat" package is not installed.
Package found in "trykl" repository:
Name           : expat, version: 2.2.7, release: 2
Summary        : XML parsing libraries
Description     : This is expat, the C library for parsing >
                  to starting the parse. These handlers are
                  of structures for which you may register h
Licenses        : as-is
Component       : system.base
Provides        :
Dependencies    :
Distribution    : Sulin, Dist. Release: 2019
Architecture    : x86_64, Installed Size: 753.35 KB, Package
```

Reverse Dependencies: mesa wayland perl-XML-Parser fontconfig dk
"expat" package is not found in source repositories.



inary install

inary install command is used to installing binary packages from inary file or repository.

Note

This operation needs privileges and can be allowed by only super user.

Using

```
sh ~# inary install <package-name>
sh ~# inary it <package-name>
```

Options

install options:

<code>--ignore-dependency</code>	Do not take dependency information into account.
<code>--ignore-safety</code>	Bypass safety switch.
<code>--ignore-scom</code>	Bypass scom configuration agent.
<code>-n, --dry-run</code>	Do not perform any action, just show what would be done.
<code>--reinstall</code>	Reinstall already installed packages.
<code>--ignore-check</code>	Skip distribution release and architecture check.
<code>--ignore-file-conflicts</code>	Ignore file conflicts.
<code>--ignore-package-conflicts</code>	

	Ignore package conflicts.
<code>-c, --component</code>	Install component's and recursive components' packages.
<code>-r, --repository</code>	Name of the component's repository.
<code>-f, --fetch-only</code>	Fetch upgrades but do not install.
<code>-x, --exclude</code>	When installing packages, ignore packages and components whose basenames match pattern.
<code>--exclude-from</code>	When installing packages, ignore packages and components whose basenames match any pattern contained in file.
<code>-s, --store-lib-info</code>	Store previous libraries info when package is updating to newer version.

Example Runtime

```
sh ~# inary it gsm
Checking dependencies for install...
Total size of package(s): 0.00 B / 93.87 KB
After this operation, 389.92 KB space will be used.
Downloading 1 / 1
Package "gsm" found in repository "core"
Installing 1 / 1
Installing "gsm", version 1.0.13, release 1
Extracting the files of "gsm"
Adding files of "gsm" package to database...
Installed "gsm"
```

inary help

inary help comand gives information about all inary commands and options.

Using

```
sh ~$ inary help
sh ~$ inary ?
```

Output

```
sh ~$ inary help
usage: inary [options] <command> [arguments]
```

where <command> is one of:

```
    add-repo (ar) - Add a repository
      blame (bl) - Information about the package owner a
      build (bi) - Build INARY packages
        check - Verify installation
  config-manager (cm) - Inary Config file manager.
configure-pending (cp) - Configure pending packages
  delete-cache (dc) - Delete cache files
      delta (dt) - Creates delta packages
  disable-repo (dr) - Disable repository
      emerge (em) - Build and install INARY source packag
  emergeup (emup) - Build and upgrade INARY source packag
  enable-repo (er) - Enable repository
      fetch (fc) - Fetch a package
      graph - Graph package relations
      help (?) - Prints help for given commands
  history (hs) - History of inary operations
      index (ix) - Index INARY files in a given director
      info - Display package information
      install (it) - Install INARY packages
  list-available (la) - List available packages in the reposi
  list-components (lc) - List available components
  list-installed (li) - Print the list of all installed packa
  list-newest (ln) - List newest packages in the repositor
```

```

list-orphaned (lo) - List orphaned packages
list-pending (lp) - List pending packages
  list-repo (lr) - List repositories
list-sources (ls) - List available sources
list-upgrades (lu) - List packages to be upgraded
  rebuild-db (rdb) - Rebuild Databases
    remove (rm) - Remove INARY packages
remove-orphaned (ro) - Remove orphaned packages
  remove-repo (rr) - Remove repositories
    search (sr) - Search packages
  search-file (sf) - Search for a file
  update-repo (ur) - Update repository databases
    upgrade (up) - Upgrade INARY packages

```

Use "inary help <command>" for help on a specific command.

Options:

```

--version           : show program's version number ar
-h [--help]         : show this help message and exit

```

general options:

```

-D [--destdir] arg   : Change the system root for INARY
-y [--yes-all]       : Assume yes in all yes/no queries
-u [--username] arg  :
-p [--password] arg  :
-L [--bandwidth-limit] arg : Keep bandwidth usage under speci
-v [--verbose]        : Detailed output
-d [--debug]          : Show debugging information.
-N [--no-color]       : Suppresses all coloring of INARY

```


inary history

inary history command allows us to list the commands made in the past, undo changes made after a certain time (takeback), and create a system restore point (snapshot). Basically it is inary's transactional history manager.

Using

```
sh ~$ inary hs
sh ~$ inary history
```

Options

history options:

<code>-l, --last</code>	Output only the last 'n' operations.
<code>-s, --snapshot</code>	Take snapshot of the current system.
<code>-t, --takeback</code>	Takeback to the state after the given operation finished.

Hints

history operation is not used to display only historical operations. This allows us to roll back operations or create a restore point at a time whenever user want.

Note

snapshot and takeback operations needs privileges and can be allowed by only super user.

Example Runtime Output

```
sh ~$ inary history
Inary Transactional History:
Operation #1: repository update:
Date: 2019-10-06 14:10
    * trykl
```

inary index

inary index command creates an index file of repository. This command collects all informations in [pspec.xml](#) and [metadata.xml](#) from binary packages then accumulates this informations as an index file.

index operation creates index files from both of source packages and binary packages. But filtering is also possible.

Using

index operation creates index from one directory if the argument is given...

```
sh ~$ inary index <directory>
sh ~$ inary ix <directory>
```

...but indexes all tree on working directory if no argument is given.

```
sh ~$ inary index
sh ~$ inary ix
```

Options

index options

<code>-a, --absolute-urls</code>	Store absolute links for indexed files.
<code>-o, --output</code>	Index output file
<code>--compression-types</code>	Comma-separated compression types for index file
<code>--skip-sources</code>	Do not index INARY spec files.
<code>--skip-signing</code>	Do not sign index.

Example Runtime Output

```
sh ~# inary graph
```

```
sh ~$ inary ix
```

```
Building index of Inary files under "."
```

```
* Generating index tree...
```

- * Adding "components.xml" to index
- * Adding "distribution.xml" to index
- * Adding "groups.xml" to index

```
* Adding binary packages:
```

- * Adding packages from directory "a"... done.
- * Adding packages from directory "b"... done.
- * Adding packages from directory "c"... done.
- * Adding packages from directory "d"... done.
- * Adding packages from directory "e"... done.
- * Adding packages from directory "f"... done.
- * Adding packages from directory "g"... done.
- * Adding packages from directory "h"... done.
- * Adding packages from directory "i"... done.
- * Adding packages from directory "j"... done.
- * Adding packages from directory "k"... done.
- * Adding packages from directory "l"... done.
- * Adding packages from directory "liba"... done.
- * Adding packages from directory "libb"... done.
- * Adding packages from directory "libc"... done.
- * Adding packages from directory "libd"... done.
- * Adding packages from directory "libe"... done.
- * Adding packages from directory "libf"... done.
- * Adding packages from directory "libg"... done.
- * Adding packages from directory "libi"... done.
- * Adding packages from directory "libj"... done.
- * Adding packages from directory "libk"... done.
- * Adding packages from directory "libl"... done.
- * Adding packages from directory "libm"... done.
- * Adding packages from directory "libn"... done.
- * Adding packages from directory "libo"... done.
- * Adding packages from directory "libp"... done.
- * Adding packages from directory "libq"... done.
- * Adding packages from directory "libr"... done.
- * Adding packages from directory "libs"... done.
- * Adding packages from directory "libt"... done.
- * Adding packages from directory "libu"... done.

- * Adding packages from directory "libv"... done.
- * Adding packages from directory "libw"... done.
- * Adding packages from directory "libx"... done.
- * Adding packages from directory "liby"... done.
- * Adding packages from directory "m"... done.
- * Adding packages from directory "n"... done.
- * Adding packages from directory "o"... done.
- * Adding packages from directory "p"... done.
- * Adding packages from directory "q"... done.
- * Adding packages from directory "r"... done.
- * Adding packages from directory "s"... done.
- * Adding packages from directory "t"... done.
- * Adding packages from directory "u"... done.
- * Adding packages from directory "v"... done.
- * Adding packages from directory "w"... done.
- * Adding packages from directory "x"... done.
- * Adding packages from directory "y"... done.
- * Adding packages from directory "z"... done.
- * Writing index file.
- * Index file written.

inary rebuild-db

inary rebuild-db command is used to rebuilding all databases including [filesdb](#)

Note

This operation needs privileges and can be allowed by only super user.

Using

```
sh ~# inary rdb
sh ~# inary rebuild-db
```

Options

rebuild-db options:

<code>-f, --files</code>	Rebuild files database
--------------------------	------------------------

Example Runtime Output

```
sh ~# inary rebuild-db
Rebuild INARY databases? (yes/no): y
Creating files database...
Adding files of "expat" package to database...
Added files database...
```

inary remove

inary remove is the command to remove the package from the system.

Note

This operation needs privileges and can be allowed by only super user.

Using

```
sh ~$ inary remove <package-name>
sh ~$ inary rm <package-name>
```


Options

remove options:

<code>--ignore-dependency</code>	Do not take dependency information into account.
<code>--ignore-safety</code>	Bypass safety switch.
<code>--ignore-scom</code>	Bypass scom configuration agent.
<code>-n, --dry-run</code>	Do not perform any action, just show what would be done.
<code>--purge</code>	Removes everything including changed config files of the package.
<code>-c, --component</code>	Remove component's and recursive components' packages.

Example Runtime

```
sh ~$ inary rm mjpegtools
The following list of packages will be removed in the respective
gst-plugins-bad-dbginfo gst-plugins-bad-devel gst-plugins-bad m
Removing "gst-plugins-bad-dbginfo"
Removing files of "gst-plugins-bad-dbginfo" package from databas
Removing files of "gst-plugins-bad-dbginfo" package from system.
Removed "gst-plugins-bad-dbginfo"
Removing "gst-plugins-bad-devel"
Removing files of "gst-plugins-bad-devel" package from database.
Removing files of "gst-plugins-bad-devel" package from system...
Removed "gst-plugins-bad-devel"
Removing "gst-plugins-bad"
Removing files of "gst-plugins-bad" package from database...
Removing files of "gst-plugins-bad" package from system...
Removed "gst-plugins-bad"
Removing "mjpegtools-devel"
Removing files of "mjpegtools-devel" package from database...
Removing files of "mjpegtools-devel" package from system...
Removing "mjpegtools"
Removing files of "mjpegtools" package from database...
Removing files of "mjpegtools" package from system...
```

A horizontal scrollbar is located at the bottom of the terminal window, indicating that the output text is scrollable. It consists of a grey track with a darker grey slider in the middle, and small arrowheads at both ends.

inary remove-orphaned

inary remove-orphaned allows us to delete packages whose come as a dependency of any package then reverse dependencies have been deleted. That is, it automatically removes unneeded packages.

Note

This operation needs privileges and can be allowed by only super user.

Using

```
sh ~$ inary remove-orphaned
sh ~$ inary ro
```


Options

remove-orphaned options:

<code>--ignore-dependency</code>	Do not take dependency information into account.
<code>--ignore-safety</code>	Bypass safety switch.
<code>--ignore-scom</code>	Bypass scom configuration agent.
<code>-n, --dry-run</code>	Do not perform any action, just show what would be done.
<code>-x, --exclude</code>	When removing orphaned, ignore packages and components whose basenames match pattern.

Example Runtime

```
sh ~$ inary remove-orphaned
The following list of packages will be removed in the respective
libnut libdc1394
After this operation, 11.21 MB space will be freed.
Removing "libnut"
Removing files of "libnut" package from database...
Removing files of "libnut" package from system...
Removed "libnut"
Removing "libdc1394"
Removing files of "libdc1394" package from database...
Removing files of "libdc1394" package from system...
Removed "libdc1394"
```



inary search

inary search command allows to search from package names and summary.

Using

```
sh ~$ inary search <package-name>
sh ~$ inary sr <package-name>
```


Options

search options:

<code>-l, --language</code>	Summary and description language.
<code>-r, --repository</code>	Name of the source or package repository.
<code>-i, --installdb</code>	Search in installdb.
<code>-s, --sourcedb</code>	Search in sourcedb.
<code>-c, --case-sensitive</code>	Case sensitive search.
<code>--name</code>	Search in the package name.
<code>--summary</code>	Search in the package summary.
<code>--description</code>	Search in the package description.

Example Runtime

```
sh ~$ inary search expa
expat          - XML parsing libraries
expat-32bit    - 32-bit shared libraries for expat
expat-devel    - Development files for expat
expat-docs     - Documentation for expat package.
expat-pages    - ManPages for expat package.
perl-XML-Parser - A Perl extension interface to James Clark's XM
```



inary search-file

inary search-file command is used to search files from [installdb](#). It allows us to determine which package contains a given filename. It allows regex inside of it. So, just one keyword is enough to searching, in order to allows regex script.

Using

```
sh ~$ inary search-file <file-name>
sh ~$ inary sf <file-name>
```

Options

search-file options:

-l, --long	Show in long format.
-q, --quiet	Show only package name.

Example Runtime

```
sh ~$ inary search-file expat.h
Searching for "expat.h"
Package "expat-devel" has file "/usr/include/expat.h"
Package "python3-devel" has file "/usr/include/python3.7m/pyexpat.h"
Package "python-devel" has file "/usr/include/python2.7/pyexpat.h"
```



inary upgrade

inary upgrade command allows to upgrade new release of installed packages.

Using

upgrade operation upgrades specific one of package if the argument is given...

```
sh ~$ inary upgrade <package-name>
sh ~$ inary up <package-name>
```

...but upgrades all new releases if no argument is given.

```
sh ~$ inary upgrade
sh ~$ inary up
```

Options

upgrade options:

<code>--ignore-dependency</code>	Do not take dependency information into account.
<code>--ignore-safety</code>	Bypass safety switch.
<code>--ignore-scom</code>	Bypass scom configuration agent.
<code>-n, --dry-run</code>	Do not perform any action, just show what would be done.
<code>--security-only</code>	Security related package upgrades only.
<code>-b, --bypass-update-repo</code>	Do not update repositories.
<code>--ignore-file-conflicts</code>	

Ignore file conflicts.

`--ignore-package-conflicts`

Ignore package conflicts.

`-c, --component`

Upgrade component's and recursive components' packages.

`-r, --repository`

Name of the to be upgraded packages' repository.

`-f, --fetch-only`

Fetch upgrades but do not install.

`-x, --exclude`

When upgrading system, ignore packages and components whose basenames match pattern.

`--exclude-from`

When upgrading system, ignore packages and components whose basenames match any pattern contained in file.

`-s, --compare-sha1sum`

Compare sha1sum repo and installed packages.

Example Runtime

```
sh ~# inary up
Updating repositories.
Updating package repository: "core"
No signature found for "/repo/SulinRepository/inary-index.xml"
Package database updated.
Checking dependencies for install...
The following packages will be upgraded:
libsoup libsoup-devel
Total size of package(s): 0.00 B / 410.70 KB
After this operation, -4160.00 B space will be freed.
Downloading 1 / 2
Package "libsoup" found in repository "core"
Downloading 2 / 2
Package "libsoup-devel" found in repository "core"
Installing 1 / 2
Installing "libsoup", version 2.68.1, release 2
Upgrading to new distribution release.
```

Extracting the files of "libsoup"
Removing files of "libsoup" package from database...
Adding files of "libsoup" package to database...
Upgraded "libsoup"
Installing 2 / 2
Installing "libsoup-devel", version 2.68.1, release 2
Upgrading to new distribution release.
Extracting the files of "libsoup-devel"
Removing files of "libsoup-devel" package from database...
Adding files of "libsoup-devel" package to database...
Upgraded "libsoup-devel"

Managing Repositories in INARY

inary package manager can download binary packages from repositories on local and remote servers. It allows adding and removing package repositories with a few commands.

Note

Repository operations needs privileges and can be allowed by only super user.

Adding Repository

Adding a repository in system making with *inary add-repo* command

Note

Repository url can be a file path.

Options

add-repo options:

<code>--ignore-check</code>	Ignore repository distribution check
<code>--no-fetch</code>	Does not fetch repository index and does not check distribution match
<code>--at</code>	Add repository at given position (0 is first)

Using

```
sh ~$ inary add-repo <repository-name> <repository-url>
sh ~$ inary ar <repository-name> <repository-url>
```


Hints

- GPG key check is performed during the repository addition. This check action can be by-passed, if not bypass, the added repository will be deactivated

Removing Repository

Removing a repository in system making with *inary remove-repo* command

Using

```
sh ~$ inary remove-repo <repository-name>
sh ~$ inary rr <repository-name>
```

Update Repository

Refreshing the repository informations can be made with *inary update-repo* command. It synchronizes repository information whether index's sha1sum has changed.

Options

update-repo options:

<code>--f, --force</code>	Update database in any case
---------------------------	-----------------------------

Using

update-repo operation refreshes given repositor(y/ies) if the argument is given...

```
sh ~$ inary update-repo <repository-name>
sh ~$ inary ur <repository-name>
```

...but refreshes all repositories if no argument is given.

```
sh ~$ inary update-repo
sh ~$ inary ur
```

Hints

- GPG key checking is also making when this process is happened. So, if you won't confirm to pass this check action your updated repository will be deactivated.
- Disable repositories will not updated unless otherwise specified.

Changing Activity of a Repository

There are two types of repository in the system: * Enable Repositories: added repositories are enabled by default. * Disable Repositories: if a GPG key error is occurred while adding/updating a repository, or, if the user requests this, the repositories are deactivated.

Enabling Repository

inary enable-repo command is used to activate a repository which has already deactivated.

```
sh ~$ inary enable-repo <repository-name>
sh ~$ inary er <repository-name>
```

Disabling Repository

inary disable-repo command is used to deactivate a repository.

```
sh ~$ inary disable-repo <repository-name>
sh ~$ inary dr <repository-name>
```

Other Commands in INARY

inary list-available

inary list-available commands allows to see installable packages in added repositories.

Using

```
sh ~# inary list-available
sh ~# inary la
```

Options

list-available:

<code>-l, --long</code>	Show in long format
<code>-c, --component</code>	List available packages under given component
<code>-U, --uninstalled</code>	Show uninstalled packages only

inary list-sources

inary list-sources commands allows to see source packages in added repositories.

Using

```
sh ~# inary list-sources
sh ~# inary ls
```

Options

list-sources options:

`-l, --long`

Show in long format

inary list-newest

inary list-newest commands allows to latest added/updated packages in added repositories.

Using

```
sh ~# inary list-newest
sh ~# inary ln
```

Options

list-newest options:

`-s, --since`

List new packages added to repository after this given date formatted as yyyy-mm-dd.

`-l, --last`

List new packages added to repository after last nth previous repository update.

inary list-upgrades

inary list-upgrades commands allows to upgrade-able packages in added repositories.

Using

```
sh ~# inary list-upgrades
sh ~# inary lu
```

Options

list-upgrades options:

`-l, --long`

Show in long format.

`-c, --component`

List upgradable packages under given component.

`-i, --install-info`

Show detailed install info.

inary list-installed

inary list-installed commands allows to see installed packages in system.

Using

```
sh ~# inary list-installed
sh ~# inary li
```

Options

list-installed options:

`-b, --with-build-host`

Only list the installed packages built by the given host.

`-l, --long`

Show in long format

`-c, --component`

List installed packages under given component.

`-i, --install-info`

Show detailed install info.

inary list-repo

inary list-repo commands allows to added repositories and their information in system.

Using

```
sh ~# inary list-repo
sh ~# inary lr
```

inary list-orphaned

inary list-orphaned commands allows to see orphaned packages in system.

Using

```
sh ~# inary list-orphaned
sh ~# inary lo
```

Options

list-orphaned options:

<code>-a, --all</code>	Show all packages without reverse dependencies.
<code>-x, --exclude</code>	Ignore packages and components whose basenames match pattern.

inary list-components

inary list-components commands allows to see components in repositories.

Using

```
sh ~# inary list-components
sh ~# inary lc
```

Options

list-components options:

<code>-l, --long</code>	Show in long format
<code>-r, --repository</code>	Name of the source or package repository

Frequently Asked Questions

Glossary

installdb

One of the database used in inary. In this database is created automatically with information of installed packages, their files, their dependency and provides

filesdb

One of database used in inary. In this database is created first run of inary and updates regularly. It contains files' paths, sha1sum and permissions of packages.

pspec.xml

This file is a part of source packages. It contains specific details about package.

actions.py

This file is instruction of building.

metadata.xml

This file generates automatically from [pspec.xml](#). It includes some details under Package tag in [pspec.xml](#).

files.xml

This file contains all details about files which included in package.

Index

[A](#) | [F](#) | [I](#) | [M](#) | [P](#)

A

[actions.py](#)

F

[files.xml](#)

[filesdb](#)

I

[installdb](#)

M

[metadata.xml](#)

P

[pspec.xml](#)