dnf Documentation

Release latest

Feb 20, 2020

Contents

Contents:

CHAPTER 1

DNF Use Cases

Contents

1.1 Introduction

Every feature present in DNF should be based on a reasonable use case. All the supported use cases are supposed to be enumerated in this document.

In case you use DNF to achieve a goal which is not documented here, either you found an error in the documentation or you misuse DNF. In either case we would appreciate if you share the case with us so we can help you to use DNF in the correct way or add the case to the list. You can only benefit from such a report because then you can be sure that the behavior that you expect will not change without prior notice in the *DNF Release Notes* and that the behavior will be covered by our test suite.

Important: Please consult every usage of DNF with our reference documentation to be sure what are you doing. The examples mentioned here are supposed to be as simple as possible and may ignore some minor corner cases.

Warning: The list is not complete yet - the use cases are being added incrementally these days.

1.2 General assumptions

The user in question must have the appropriate permissions.

1.3 Ensure that my system contains given mix of features (packages/files/providers/groups)

A system administrator has a list of features that has to be present in an operating system. The features must be provided by RPM packages in system repositories that must be accessible.

A feature may be for example a concrete version of a package (`hawkey-0.5.3-1.fc21.i686`), a pathname of a binary RPM file (`/var/lib/mock/fedora-21-i386/result/hawkey-0.5.3-2.20150116gitd002c90.fc21.i686.rpm`), an URL of a binary RPM file (`http://jenkins.cloud.fedoraproject.org/job/DNF/lastSuccessfulBuild/artifact/fedora-21-i386-build/hawkey-0.5.3-99.649.20150116gitd002c90233fc96893806836a258f14a50ee0cf47.fc21.i686.rpm`), a configuration file (`/etc/yum.repos.d/fedora-rawhide.repo`), a language interpreter (`ruby(runtime_executable)`), an extension (`python3-dnf`), a support for building modules for the current running kernel (`kernel-devel-uname-r = $(uname -r)`), an executable (`*/binaryname`) or a collection of packages specified by any available identifier (`kde-desktop`).

The most recent packages that provide the missing features and suit installation (that are not obsoleted and do not conflict with each other or with the other installed packages) are installed if the given feature is not present already. If any of the packages cannot be installed, the operation fails.

1.3.1 CLI

```
SPECS="hawkey-0.5.3-1.fc21.i686 @kde-desktop"  # Set the features here.

dnf install $SPECS
```

1.3.2 Plugins/CLI API

```
"""A plugin that ensures that given features are present."""


import dnf.cli
from dnf.i18n import _
from dnf.cli.option_parser import OptionParser

# The parent class allows registration to the CLI manager.
```

(continues on next page)

(continued from previous page)

```
class Command(dnf.cli.Command):

    """A command that ensures that given features are present."""

    # An alias is needed to invoke the command from command line.
    aliases = ['foo']  # <-- SET YOUR ALIAS HERE.

    def configure(self):
        """Setup the demands."""
        # Repositories are needed if we want to install anything.
        self.cli.demands.available_repos = True
        # A sack is required by marking methods and dependency resolving.
        self.cli.demands.sack_activation = True
        # Resolving performs a transaction that installs the packages.
        self.cli.demands.resolving = True
        # Based on the system, privileges are required to do an installation.
        self.cli.demands.root_user = True  # <-- SET YOUR FLAG HERE.

    @staticmethod
    def set_argparser(parser):
        """Parse command line arguments."""
        parser.add_argument('package', nargs='+', metavar=_('PACKAGE'),
                            action=OptionParser.ParseSpecGroupFileCallback,
                            help=_('Package to install'))

    def run(self):
        """Run the command."""
        # Feature marking methods set the user request.
        for ftr_spec in self.opts.pkg_specs:
            try:
                self.base.install(ftr_spec)
            except dnf.exceptions.MarkingError:
                raise dnf.exceptions.Error('feature(s) not found: ' + ftr_spec)
        # Package marking methods set the user request.
        for pkg in self.base.add_remote_rpms(self.opts.filenames, strict=False):
            try:
                self.base.package_install(pkg, strict=False)
            except dnf.exceptions.MarkingError as e:
                raise dnf.exceptions.Error(e)
        # Comps data reading initializes the base.comps attribute.
        if self.opts.grp_specs:
            self.base.read_comps(arch_filter=True)
        # Group marking methods set the user request.
        for grp_spec in self.opts.grp_specs:
            group = self.base.comps.group_by_pattern(grp_spec)
            if not group:
                raise dnf.exceptions.Error('group not found: ' + grp_spec)
            self.base.group_install(group.id, ['mandatory', 'default'])


# Every plugin must be a subclass of dnf.Plugin.
class Plugin(dnf.Plugin):

    """A plugin that registers our custom command."""

    # Every plugin must provide its name.
    name = 'foo'  # <-- SET YOUR NAME HERE.
```

(continues on next page)

(continued from previous page)

```
    # Every plugin must provide its own initialization function.
    def __init__(self, base, cli):
        """Initialize the plugin."""
        super(Plugin, self).__init__(base, cli)
        if cli:
            cli.register_command(Command)
```

If it makes a sense, the plugin can do the operation in appropriate hooks instead of registering a new command that needs to be called from the command line.

1.3.3 Extension API

```
"""An extension that ensures that given features are present."""


import sys

import dnf
import dnf.module
import dnf.rpm


if __name__ == '__main__':
    FTR_SPECS = {'acpi-1.7-10.fc29.x86_64'}  # <-- SET YOUR FEATURES HERE.
    RPM_SPECS = {'./acpi-1.7-10.fc29.x86_64.rpm'}  # <-- SET YOUR RPMS HERE.
    GRP_SPECS = {'kde-desktop'}  # <-- SET YOUR GROUPS HERE.
    MODULE_SPEC = {"nodejs:10/default"}  # <-- SET YOUR MODULES HERE.

    with dnf.Base() as base:
        # Substitutions are needed for correct interpretation of repo files.
        RELEASEVER = dnf.rpm.detect_releasever(base.conf.installroot)
        base.conf.substitutions['releasever'] = RELEASEVER
        # Repositories are needed if we want to install anything.
        base.read_all_repos()
        # A sack is required by marking methods and dependency resolving.
        base.fill_sack()
        # Feature marking methods set the user request.
        for ftr_spec in FTR_SPECS:
            try:
                base.install(ftr_spec)
            except dnf.exceptions.MarkingError:
                sys.exit('Feature(s) cannot be found: ' + ftr_spec)
        # Package marking methods set the user request.
        for pkg in base.add_remote_rpms(RPM_SPECS, strict=False):
            try:
                base.package_install(pkg, strict=False)
            except dnf.exceptions.MarkingError:
                sys.exit('RPM cannot be found: ' + pkg)
        # Comps data reading initializes the base.comps attribute.
        if GRP_SPECS:
            base.read_comps(arch_filter=True)
        # Group marking methods set the user request.
        if MODULE_SPEC:
            module_base = dnf.module.module_base.ModuleBase(base)
```

(continues on next page)

(continued from previous page)

```
        module_base.install(MODULE_SPEC, strict=False)
    for grp_spec in GRP_SPECS:
        group = base.comps.group_by_pattern(grp_spec)
        if not group:
            sys.exit('Group cannot be found: ' + grp_spec)
        base.group_install(group.id, ['mandatory', 'default'])
    # Resolving finds a transaction that allows the packages installation.
    try:
        base.resolve()
    except dnf.exceptions.DepsolveError:
        sys.exit('Dependencies cannot be resolved.')
    # The packages to be installed must be downloaded first.
    try:
        base.download_packages(base.transaction.install_set)
    except dnf.exceptions.DownloadError:
        sys.exit('Required package cannot be downloaded.')
    # The request can finally be fulfilled.
    base.do_transaction()
```

1.4 Get a list of available packages filtered by their relation to the system

A system user wants to obtain a list of available RPM packages for their consecutive automatic processing or for informative purpose only. The list of RPM packages is filtered by requested relation to the system or user provided <package-name-specs>. The obtained list of packages is based on available data supplied by accessible system repositories.

A relation to the system might be for example one of the following:

installed - packages already installed on the system

available - packages available in any accessible repository

extras - packages installed on the system not available in any known repository

obsoletes - installed packages that are obsoleted by packages in any accessible repository

recent - packages recently added into accessible repositories

upgrades - available packages upgrading some installed packages

1.4.1 CLI

```
dnf list *dnf*
dnf list installed *debuginfo
dnf list available gtk*devel
dnf list extras
dnf list obsoletes
dnf list recent
dnf list upgrades
```

1.4.2 Plugins/CLI API

```
"""A plugin that lists installed packages that are obsolted by any available␣
↪package"""

from dnf.i18n import _
import dnf
import dnf.cli


# If you only plan to create a new dnf subcommand in a plugin
# you can use @dnf.plugin.register_command decorator instead of creating
# a Plugin class which only registers the command
# (for full-fledged Plugin class see examples/install_plugin.py)
@dnf.plugin.register_command
class Command(dnf.cli.Command):

    """A command that lists packages installed on the system that are
       obsoleted by packages in any known repository."""

    # An alias is needed to invoke the command from command line.
    aliases = ['foo']  # <-- SET YOUR ALIAS HERE.

    @staticmethod
    def set_argparser(parser):
        parser.add_argument("package", nargs='*', metavar=_('PACKAGE'))

    def configure(self):
        """Setup the demands."""
        # Repositories serve as sources of information about packages.
        self.cli.demands.available_repos = True
        # A sack is needed for querying.
        self.cli.demands.sack_activation = True

    def run(self):
        """Run the command."""

        obs_tuples = []
        # A query matches all available packages
        q = self.base.sack.query()

        if not self.opts.package:
            # Narrow down query to match only installed packages
            inst = q.installed()
            # A dictionary containing list of obsoleted packages
            for new in q.filter(obsoletes=inst):
                obs_reldeps = new.obsoletes
                obsoleted = inst.filter(provides=obs_reldeps).run()
                obs_tuples.extend([(new, old) for old in obsoleted])
        else:
            for pkg_spec in self.opts.package:
                # A subject serves for parsing package format from user input
                subj = dnf.subject.Subject(pkg_spec)
                # A query restricted to installed packages matching given␣
```

```
↪subject
                inst = subj.get_best_query(self.base.sack).installed()
                for new in q.filter(obsoletes=inst):
                    obs_reldeps = new.obsoletes
                    obsoleted = inst.filter(provides=obs_reldeps).run()
                    obs_tuples.extend([(new, old) for old in obsoleted])

        if not obs_tuples:
            raise dnf.exceptions.Error('No matching Packages to list')

        for (new, old) in obs_tuples:
            print('%s.%s obsoletes %s.%s' %
                  (new.name, new.arch, old.name, old.arch))
```

1.4.3 Extension API

```
"""An extension that lists installed packages not available
   in any remote repository.
"""

import dnf


if __name__ == '__main__':

    with dnf.Base() as base:
        # Repositories serve as sources of information about packages.
        base.read_all_repos()
        # A sack is needed for querying.
        base.fill_sack()

        # A query matches all packages in sack
        q = base.sack.query()

        # Derived query matches only available packages
        q_avail = q.available()
        # Derived query matches only installed packages
        q_inst = q.installed()

        available = q_avail.run()
        for pkg in q_inst.run():
            if pkg not in available:
                print(str(pkg))
```

CHAPTER 2

DNF Command Reference

2.1 Synopsis

```
dnf [options] <command> [<args>...]
```

2.2 Description

'DNF'_ is the next upcoming major version of **'YUM'_**, a package manager for RPM-based Linux distributions. It roughly maintains CLI compatibility with YUM and defines a strict API for extensions and plugins.

Plugins can modify or extend features of DNF or provide additional CLI commands on top of those mentioned below. If you know the name of such a command (including commands mentioned below), you may find/install the package which provides it using the appropriate virtual provide in the form of `dnf-command(<alias>)`, where `<alias>` is the name of the command; e.g.''dnf install 'dnf-command(versionlock)'' installs a `versionlock` plugin. This approach also applies to specifying dependencies of packages that require a particular DNF command.

Return values:

- `0` : Operation was successful.
- `1` : An error occurred, which was handled by dnf.
- `3` : An unknown unhandled error occurred during operation.
- `100`: See *check-update*
- `200`: There was a problem with acquiring or releasing of locks.

Available commands:

- *alias*
- *autoremove*
- *check*
- *check-update*

- *clean*
- *deplist*
- *distro-sync*
- *downgrade*
- *group*
- *help*
- *history*
- *info*
- *install*
- *list*
- *makecache*
- *mark*
- *module*
- *provides*
- *reinstall*
- *remove*
- *repoinfo*
- *repolist*
- *repoquery*
- *repository-packages*
- *search*
- *shell*
- *swap*
- *updateinfo*
- *upgrade*
- *upgrade-minimal*
- *upgrade-to*

Additional information:

- *Options*
- *Specifying Packages*
- *Specifying Exact Versions of Packages*
- *Specifying Provides*
- *Specifying Groups*
- *Specifying Transactions*
- *Metadata Synchronization*
- *Configuration Files Replacement Policy*

- *Files*
- *See Also*

2.3 Options

-4 Resolve to IPv4 addresses only.

-6 Resolve to IPv6 addresses only.

--advisory=<advisory>, --advisories=<advisory> Include packages corresponding to the advisory ID, Eg. FEDORA-2201-123. Applicable for the install, repoquery, updateinfo and upgrade commands.

--allowerasing Allow erasing of installed packages to resolve dependencies. This option could be used as an alternative to the `yum swap` command where packages to remove are not explicitly defined.

--assumeno Automatically answer no for all questions.

-b, --best Try the best available package versions in transactions. Specifically during *dnf upgrade*, which by default skips over updates that can not be installed for dependency reasons, the switch forces DNF to only consider the latest packages. When running into packages with broken dependencies, DNF will fail giving a reason why the latest version can not be installed.

--bugfix Include packages that fix a bugfix issue. Applicable for the install, repoquery, updateinfo and upgrade commands.

--bz=<bugzilla>, --bzs=<bugzilla> Include packages that fix a Bugzilla ID, Eg. 123123. Applicable for the install, repoquery, updateinfo and upgrade commands.

-C, --cacheonly Run entirely from system cache, don't update the cache and use it even in case it is expired.

DNF uses a separate cache for each user under which it executes. The cache for the root user is called the system cache. This switch allows a regular user read-only access to the system cache, which usually is more fresh than the user's and thus he does not have to wait for metadata sync.

--color=<color> Control whether color is used in terminal output. Valid values are `always`, `never` and `auto` (default).

--comment=<comment> Add a comment to the transaction history.

-c <config file>, --config=<config file> Configuration file location.

--cve=<cves>, --cves=<cves> Include packages that fix a CVE (Common Vulnerabilities and Exposures) ID (http://cve.mitre.org/about/), Eg. CVE-2201-0123. Applicable for the install, repoquery, updateinfo, and upgrade commands.

-d <debug level>, --debuglevel=<debug level> Debugging output level. This is an integer value between 0 (no additional information strings) and 10 (shows all debugging information, even that not understandable to the user), default is 2. Deprecated, use `-v` instead.

--debugsolver Dump data aiding in dependency solver debugging into `./debugdata`.

`--disableexcludes=[all|main|<repoid>], --disableexcludepkgs=[all|main|<repoid>]`

Disable the configuration file excludes. Takes one of the following three options:

- `all`, disables all configuration file excludes
- `main`, disables excludes defined in the `[main]` section
- `repoid`, disables excludes defined for the given repository

--disable, --set-disabled Disable specified repositories (automatically saves). The option has to be used together with the `config-manager` command (dnf-plugins-core).

--disableplugin=<plugin names> Disable the listed plugins specified by names or globs.

--disablerepo=<repoid> Disable specific repositories by an id or a glob. This option is mutually exclusive with `--repo`.

--downloaddir=<path>, --destdir=<path> Redirect downloaded packages to provided directory. The option has to be used together with the *--downloadonly* command line option, with the `download` command (dnf-plugins-core) or with the `system-upgrade` command (dnf-plugins-extras).

--downloadonly Download the resolved package set without performing any rpm transaction (install/upgrade/erase).

-e <error level>, --errorlevel=<error level> Error output level. This is an integer value between 0 (no error output) and 10 (shows all error messages), default is 3. Deprecated, use `-v` instead.

--enable, --set-enabled Enable specified repositories (automatically saves). The option has to be used together with the `config-manager` command (dnf-plugins-core).

--enableplugin=<plugin names> Enable the listed plugins specified by names or globs.

--enablerepo=<repoid> Enable additional repositories by an id or a glob.

--enhancement Include enhancement relevant packages. Applicable for the install, repoquery, updateinfo and upgrade commands.

-x <package-file-spec>, --exclude=<package-file-spec> Exclude packages specified by `<package-file-spec>` from the operation.

--excludepkgs=<package-file-spec> Deprecated option. It was replaced by the *--exclude* option.

--forcearch=<arch> Force the use of an architecture. Any architecture can be specified. However, use of an architecture not supported natively by your CPU will require emulation of some kind. This is usually through QEMU. The behavior of `--forcearch` can be configured by using the *arch* and *ignorearch* configuration options with values `<arch>` and `True` respectively.

-h, --help, --help-cmd Show the help.

--installroot=<path> Specifies an alternative installroot, relative to where all packages will be installed. Think of this like doing `chroot <root> dnf`, except using `--installroot` allows dnf to work before the chroot is created. It requires absolute path.

- *cachedir*, *log files*, *releasever*, and *gpgkey* are taken from or stored in the installroot. *Gpgkeys* are imported into the installroot from a path relative to the host which can be specified in the repository section of configuration files.
- *configuration file* and *reposdir* are searched inside the installroot first. If they are not present, they are taken from the host system. Note: When a path is specified within a command line argument (`--config=<config file>` in case of *configuration file* and `--setopt=reposdir=<reposdir>` for *reposdir*) then this path is always relative to the host with no exceptions.
- *vars* are taken from the host system or installroot according to *reposdir* . When *reposdir* path is specified within a command line argument, vars are taken from the installroot. When *varsdir* paths are specified within a command line argument (`--setopt=varsdir=<reposdir>`) then those path are always relative to the host with no exceptions.
- The *pluginpath* and *pluginconfpath* are relative to the host.

  Note: You may also want to use the command-line option `--releasever=<release>` when creating the installroot, otherwise the *$releasever* value is taken from the rpmdb within the installroot (and thus it is empty at the time of creation and the transaction will fail). If `--releasever=/` is used, the releasever

will be detected from the host (`/`) system. The new installroot path at the time of creation does not contain the *repository*, *releasever* and *dnf.conf* files.

On a modular system you may also want to use the `--setopt=module_platform_id=<module_platform_name:str` command-line option when creating the installroot, otherwise the *module_platform_id* value will be taken from the `/etc/os-release` file within the installroot (and thus it will be empty at the time of creation, the modular dependency could be unsatisfied and modules content could be excluded).

Installroot examples:

`dnf --installroot=<installroot> --releasever=<release> install system-release`
Permanently sets the `releasever` of the system in the `<installroot>` directory to `<release>`.

`dnf --installroot=<installroot> --setopt=reposdir=<path> --config /path/dnf.conf upgra`
Upgrades packages inside the installroot from a repository described by `--setopt` using configuration from `/path/dnf.conf`.

`--newpackage` Include newpackage relevant packages. Applicable for the install, repoquery, updateinfo and upgrade commands.

`--noautoremove` Disable removal of dependencies that are no longer used. It sets *clean_requirements_on_remove* configuration option to `False`.

`--nobest` Set best option to `False`, so that transactions are not limited to best candidates only.

`--nodocs` Do not install documentation. Sets the rpm flag 'RPMTRANS_FLAG_NODOCS'.

`--nogpgcheck` Skip checking GPG signatures on packages (if RPM policy allows).

`--noplugins` Disable all plugins.

`--obsoletes` This option has an effect on an install/update, it enables dnf's obsoletes processing logic. For more information see the *obsoletes* option.

This option also displays capabilities that the package obsoletes when used together with the *repoquery* command.

Configuration Option: *obsoletes*

`-q, --quiet` In combination with a non-interactive command, shows just the relevant content. Suppresses messages notifying about the current state or actions of DNF.

`-R <minutes>, --randomwait=<minutes>` Maximum command wait time.

`--refresh` Set metadata as expired before running the command.

`--releasever=<release>` Configure DNF as if the distribution release was `<release>`. This can affect cache paths, values in configuration files and mirrorlist URLs.

`--repofrompath <repo>,<path/url>` Specify a repository to add to the repositories for this query. This option can be used multiple times.

- The repository label is specified by `<repo>`.
- The path or url to the repository is specified by `<path/url>`. It is the same path as a baseurl and can be also enriched by the *repo variables*.
- The configuration for the repository can be adjusted using *--setopt*=<repo>.<option>=<value>.
- If you want to view only packages from this repository, combine this with the `--repo=<repo>` or `--disablerepo="*"` switches.

--repo=<repoid>, --repoid=<repoid> Enable just specific repositories by an id or a glob. Can be used multiple times with accumulative effect. It is basically a shortcut for `--disablerepo="*" --enablerepo=<repoid>` and is mutually exclusive with the `--disablerepo` option.

--rpmverbosity=<name> RPM debug scriptlet output level. Sets the debug level to `<name>` for RPM scriptlets. For available levels, see the `rpmverbosity` configuration option.

--sec-severity=<severity>, --secseverity=<severity> Includes packages that provide a fix for an issue of the specified severity. Applicable for the install, repoquery, updateinfo and upgrade commands.

--security Includes packages that provide a fix for a security issue. Applicable for the upgrade command.

--setopt=<option>=<value> Override a configuration option from the configuration file. To override configuration options for repositories, use `repoid.option` for the `<option>`. Values for configuration options like `excludepkgs`, `includepkgs`, `installonlypkgs` and `tsflags` are appended to the original value, they do not override it. However, specifying an empty value (e.g. `--setopt=tsflags=`) will clear the option.

--skip-broken Resolve depsolve problems by removing packages that are causing problems from the transaction. It is an alias for the *strict* configuration option with value `False`. Additionally, with the *enable* and *disable* module subcommands it allows one to perform an action even in case of broken modular dependencies.

--showduplicates Show duplicate packages in repositories. Applicable for the list and search commands.

-v, --verbose Verbose operation, show debug messages.

--version Show DNF version and exit.

-y, --assumeyes Automatically answer yes for all questions.

List options are comma-separated. Command-line options override respective settings from configuration files.

2.4 Commands

For an explanation of `<package-spec>` and `<package-file-spec>` see *Specifying Packages*.

For an explanation of `<package-nevr-spec>` see *Specifying Exact Versions of Packages*.

For an explanation of `<provide-spec>` see *Specifying Provides*.

For an explanation of `<group-spec>` see *Specifying Groups*.

For an explanation of `<module-spec>` see *Specifying Modules*.

For an explanation of `<transaction-spec>` see *Specifying Transactions*.

2.4.1 Alias Command

Allows the user to define and manage a list of aliases (in the form `<name=value>`), which can be then used as dnf commands to abbreviate longer command sequences. For examples on using the alias command, see *Alias Examples*. For examples on the alias processing, see *Alias Processing Examples*.

To use an alias (name=value), the name must be placed as the first "command" (e.g. the first argument that is not an option). It is then replaced by its value and the resulting sequence is again searched for aliases. The alias processing stops when the first found command is not a name of any alias.

In case the processing would result in an infinite recursion, the original arguments are used instead.

Also, like in shell aliases, if the result starts with a `\`, the alias processing will stop.

All aliases are defined in configuration files in the /etc/dnf/aliases.d/ directory in the [aliases] section, and aliases created by the alias command are written to the USER.conf file. In case of conflicts, the USER.conf has the highest priority, and alphabetical ordering is used for the rest of the configuration files.

Optionally, there is the enabled option in the [main] section defaulting to True. This can be set for each file separately in the respective file, or globally for all aliases in the ALIASES.conf file.

dnf alias [options] [list] [<name>...]

List aliases with their final result. The [<alias>...] parameter further limits the result to only those aliases matching it.

dnf alias [options] add <name=value>...

Create new aliases.

dnf alias [options] delete <name>...

Delete aliases.

Alias Examples

dnf alias list Lists all defined aliases.

dnf alias add rm=remove Adds a new command alias called rm which works the same as the remove command.

dnf alias add upgrade="\upgrade --skip-broken --disableexcludes=all --obsoletes" Adds a new command alias called upgrade which works the same as the upgrade command, with additional options. Note that the original upgrade command is prefixed with a \ to prevent an infinite loop in alias processing.

Alias Processing Examples

If there are defined aliases in=install and FORCE="--skip-broken --disableexcludes=all":

- dnf FORCE in will be replaced with dnf --skip-broken --disableexcludes=all install
- dnf in FORCE will be replaced with dnf install FORCE (which will fail)

If there is defined alias in=install:

- dnf in will be replaced with dnf install
- dnf --repo updates in will be replaced with dnf --repo updates in (which will fail)

2.4.2 Auto Remove Command

dnf [options] autoremove

Removes all "leaf" packages from the system that were originally installed as dependencies of user-installed packages, but which are no longer required by any such package.

Packages listed in *installonlypkgs* are never automatically removed by this command.

dnf [options] autoremove <spec>...

This is an alias for the *Remove Command* command with clean_requirements_on_remove set to True. It removes the specified packages from the system along with any packages depending on the packages being removed. Each <spec> can be either a <package-spec>, which specifies a package directly,

or a `@<group-spec>`, which specifies an (environment) group which contains it. It also removes any dependencies that are no longer needed.

There are also a few specific autoremove commands `autoremove-n`, `autoremove-na` and `autoremove-nevra` that allow the specification of an exact argument in the NEVRA (name-epoch:version-release.architecture) format.

This command by default does not force a sync of expired metadata. See also *Metadata Synchronization*.

2.4.3 Check Command

`dnf [options] check [--dependencies] [--duplicates] [--obsoleted] [--provides]`

Checks the local packagedb and produces information on any problems it finds. You can limit the checks to be performed by using the `--dependencies`, `--duplicates`, `--obsoleted` and `--provides` options (the default is to check everything).

2.4.4 Check-Update Command

`dnf [options] check-update [--changelogs] [<package-file-spec>...]`

Non-interactively checks if updates of the specified packages are available. If no `<package-file-spec>` is given, checks whether any updates at all are available for your system. DNF exit code will be 100 when there are updates available and a list of the updates will be printed, 0 if not and 1 if an error occurs. If `--changelogs` option is specified, also changelog delta of packages about to be updated is printed.

Please note that having a specific newer version available for an installed package (and reported by `check-update`) does not imply that subsequent `dnf upgrade` will install it. The difference is that `dnf upgrade` has restrictions (like package dependencies being satisfied) to take into account.

The output is affected by the *autocheck_running_kernel* configuration option.

2.4.5 Clean Command

Performs cleanup of temporary files kept for repositories. This includes any such data left behind from disabled or removed repositories as well as for different distribution release versions.

`dnf clean dbcache` Removes cache files generated from the repository metadata. This forces DNF to regenerate the cache files the next time it is run.

`dnf clean expire-cache` Marks the repository metadata expired. DNF will re-validate the cache for each repository the next time it is used.

`dnf clean metadata` Removes repository metadata. Those are the files which DNF uses to determine the remote availability of packages. Using this option will make DNF download all the metadata the next time it is run.

`dnf clean packages` Removes any cached packages from the system.

`dnf clean all` Does all of the above.

2.4.6 Deplist command

`dnf [options] deplist [<select-options>] [<query-options>] [<package-spec>]`
Alias for *dnf repoquery –deplist*.

2.4.7 Distro-Sync command

dnf distro-sync [<package-spec>...] As necessary upgrades, downgrades or keeps selected installed packages to match the latest version available from any enabled repository. If no package is given, all installed packages are considered.

See also *Configuration Files Replacement Policy*.

2.4.8 Distribution-Synchronization command

dnf distribution-synchronization Deprecated alias for the *Distro-Sync command*.

2.4.9 Downgrade Command

dnf [options] downgrade <package-spec>... Downgrades the specified packages to the highest installable package of all known lower versions if possible. When version is given and is lower than version of installed package then it downgrades to target version.

2.4.10 Erase Command

dnf [options] erase <spec>... Deprecated alias for the *Remove Command*.

2.4.11 Group Command

Groups are virtual collections of packages. DNF keeps track of groups that the user selected ("marked") installed and can manipulate the comprising packages with simple commands.

dnf [options] group [summary] <group-spec> Display overview of how many groups are installed and available. With a spec, limit the output to the matching groups. `summary` is the default groups subcommand.

dnf [options] group info <group-spec> Display package lists of a group. Shows which packages are installed or available from a repository when `-v` is used.

dnf [options] group install [--with-optional] <group-spec>... Mark the specified group installed and install packages it contains. Also include *optional* packages of the group if `--with-optional` is specified. All *mandatory* and *Default* packages will be installed whenever possible. Conditional packages are installed if they meet their requirement. If the group is already (partially) installed, the command installs the missing packages from the group. Depending on the value of *obsoletes configuration option* group installation takes obsoletes into account.

dnf [options] group list <group-spec>... List all matching groups, either among installed or available groups. If nothing is specified, list all known groups. `--installed` and `--available` options narrow down the requested list. Records are ordered by the *display_order* tag defined in comps.xml file. Provides a list of all hidden groups by using option `--hidden`. Provides group IDs when the `-v` or `--ids` options are used.

dnf [options] group remove <group-spec>... Mark the group removed and remove those packages in the group from the system which do not belong to another installed group and were not installed explicitly by the user.

dnf [options] group upgrade <group-spec>... Upgrades the packages from the group and upgrades the group itself. The latter comprises of installing packages that were added to the group by the distribution and removing packages that got removed from the group as far as they were not installed explicitly by the user.

Groups can also be marked installed or removed without physically manipulating any packages:

dnf [options] group mark install <group-spec>... Mark the specified group installed. No packages will be installed by this command, but the group is then considered installed.

dnf [options] group mark remove <group-spec>... Mark the specified group removed. No packages will be removed by this command.

See also *Configuration Files Replacement Policy*.

2.4.12 Groups Command

dnf [options] groups Deprecated alias for the *Group Command*.

2.4.13 Help Command

dnf help [<command>] Displays the help text for all commands. If given a command name then only displays help for that particular command.

2.4.14 History Command

The history command allows the user to view what has happened in past transactions and act according to this information (assuming the `history_record` configuration option is set).

dnf history [list] [<spec>...] The default history action is listing information about given transactions in a table. Each `<spec>` can be either a `<transaction-spec>`, which specifies a transaction directly, or a `<transaction-spec>..<transaction-spec>`, which specifies a range of transactions, or a `<package-name-spec>`, which specifies a transaction by a package which it manipulated. When no transaction is specified, list all known transactions.

dnf history info [<spec>...] Describe the given transactions. The meaning of `<spec>` is the same as in the *History List Command*. When no transaction is specified, describe what happened during the latest transaction.

dnf history redo <transaction-spec>|<package-file-spec> Repeat the specified transaction. Uses the last transaction (with the highest ID) if more than one transaction for given <package-file-spec> is found. If it is not possible to redo some operations due to the current state of RPMDB, it will not redo the transaction.

dnf history rollback <transaction-spec>|<package-file-spec> Undo all transactions performed after the specified transaction. Uses the last transaction (with the highest ID) if more than one transaction for given <package-file-spec> is found. If it is not possible to undo some transactions due to the current state of RPMDB, it will not undo any transaction.

dnf history undo <transaction-spec>|<package-file-spec> Perform the opposite operation to all operations performed in the specified transaction. Uses the last transaction (with the highest ID) if more than one transaction for given <package-file-spec> is found. If it is not possible to undo some operations due to the current state of RPMDB, it will not undo the transaction.

dnf history userinstalled Show all installonly packages, packages installed outside of DNF and packages not installed as dependency. I.e. it lists packages that will stay on the system when *Auto Remove Command* or *Remove Command* along with *clean_requirements_on_remove* configuration option set to True is executed. Note the same results can be accomplished with `dnf repoquery --userinstalled`, and the repoquery command is more powerful in formatting of the output.

This command by default does not force a sync of expired metadata, except for the redo, rollback, and undo subcommands. See also *Metadata Synchronization* and *Configuration Files Replacement Policy*.

2.4.15 Info Command

dnf [options] info [<package-file-spec>...] Lists description and summary information about installed and available packages.

This command by default does not force a sync of expired metadata. See also *Metadata Synchronization*.

2.4.16 Install Command

dnf [options] install <spec>... Makes sure that the given packages and their dependencies are installed on the system. Each `<spec>` can be either a *<package-spec>*, or a @*<module-spec>*, or a @*<group-spec>*. See *Install Examples*. If a given package or provide cannot be (and is not already) installed, the exit code will be non-zero. If the `<spec>` matches both a @*<module-spec>* and a @*<group-spec>*, only the module is installed.

When *<package-spec>* to specify the exact version of the package is given, DNF will install the desired version, no matter which version of the package is already installed. The former version of the package will be removed in the case of non-installonly package.

There are also a few specific install commands `install-n`, `install-na` and `install-nevra` that allow the specification of an exact argument in the NEVRA format.

See also *Configuration Files Replacement Policy*.

Install Examples

dnf install tito Install the `tito` package (tito is the package name).

dnf install ~/Downloads/tito-0.6.2-1.fc22.noarch.rpm Install a local rpm file tito-0.6.2-1.fc22.noarch.rpm from the ~/Downloads/ directory.

dnf install tito-0.5.6-1.fc22 Install the package with a specific version. If the package is already installed it will automatically try to downgrade or upgrade to the specific version.

dnf --best install tito Install the latest available version of the package. If the package is already installed it will try to automatically upgrade to the latest version. If the latest version of the package cannot be installed, the installation will fail.

dnf install vim DNF will automatically recognize that vim is not a package name, but will look up and install a package that provides vim with all the required dependencies. Note: Package name match has precedence over package provides match.

dnf install https://kojipkgs.fedoraproject.org//packages/tito/0.6.0/1.fc22/noarch/tito-0.6 Install a package directly from a URL.

dnf install '@docker' Install all default profiles of module 'docker' and their RPMs. Module streams get enabled accordingly.

dnf install '@Web Server' Install the 'Web Server' environmental group.

dnf install /usr/bin/rpmsign Install a package that provides the /usr/bin/rpmsign file.

dnf -y install tito --setopt=install_weak_deps=False Install the `tito` package (tito is the package name) without weak deps. Weak deps are not required for core functionality of the package, but they enhance the original package (like extended documentation, plugins, additional functions, etc.).

dnf install --advisory=FEDORA-2018-b7b99fe852 * Install all packages that belong to the "FEDORA-2018-b7b99fe852" advisory.

2.4.17 List Command

Prints lists of packages depending on the packages' relation to the system. A package is `installed` if it is present in the RPMDB, and it is `available` if it is not installed but is present in a repository that DNF knows about. The list command can also limit the displayed packages according to specific criteria, e.g. to only those that update an installed package. The *exclude* option in the configuration file can influence the result, but if the *--disableexcludes* command line option is used, it ensures that all installed packages will be listed.

dnf [options] list [--all] [<package-file-spec>...] Lists all packages, present in the RPMDB, in a repository or both.

dnf [options] list --installed [<package-file-spec>...] Lists installed packages.

dnf [options] list --available [<package-file-spec>...] Lists available packages.

dnf [options] list --extras [<package-file-spec>...] Lists extras, that is packages installed on the system that are not available in any known repository.

dnf [options] list --obsoletes [<package-file-spec>...] List packages installed on the system that are obsoleted by packages in any known repository.

dnf [options] list --recent [<package-file-spec>...] List packages recently added into the repositories.

dnf [options] list --upgrades [<package-file-spec>...] List upgrades available for the installed packages.

dnf [options] list --autoremove List packages which will be removed by the `dnf autoremove` command.

This command by default does not force a sync of expired metadata. See also *Metadata Synchronization*.

2.4.18 Localinstall Command

dnf [options] localinstall <spec>... Deprecated alias for the *Install Command*.

2.4.19 Makecache Command

dnf [options] makecache Downloads and caches metadata for all known repos. Tries to avoid downloading whenever possible (e.g. when the local metadata hasn't expired yet or when the metadata timestamp hasn't changed).

dnf [options] makecache --timer Like plain `makecache`, but instructs DNF to be more resource-aware, meaning it will not do anything if running on battery power and will terminate immediately if it's too soon after the last successful `makecache` run (see *dnf.conf(5)*, *metadata_timer_sync*).

2.4.20 Mark Command

dnf mark install <package-spec>... Marks the specified packages as installed by user. This can be useful if any package was installed as a dependency and is desired to stay on the system when *Auto Remove Command* or *Remove Command* along with *clean_requirements_on_remove* configuration option set to `True` is executed.

dnf mark remove <package-spec>... Unmarks the specified packages as installed by user. Whenever you as a user don't need a specific package you can mark it for removal. The package stays installed on the system but will be removed when *Auto Remove Command* or *Remove Command* along with *clean_requirements_on_remove* configuration option set to `True` is executed. You should use this operation instead of *Remove Command* if you're not sure whether the package is a requirement of other user installed packages on the system.

dnf mark group <package-spec>... Marks the specified packages as installed by group. This can be useful if any package was installed as a dependency or a user and is desired to be protected and handled as a group member like during group remove.

2.4.21 Module Command

Modularity overview is available at *man page dnf.modularity(7)*. Module subcommands take *<module-spec>*... arguments that specify modules or profiles.

dnf [options] module install <module-spec>... Install module profiles, including their packages. In case no profile was provided, all default profiles get installed. Module streams get enabled accordingly.

This command cannot be used for switching module streams. It is recommended to remove all installed content from the module and reset the module using the *reset* command. After you reset the module, you can install the other stream.

dnf [options] module update <module-spec>... Update packages associated with an active module stream, optionally restricted to a profile. If the *profile_name* is provided, only the packages referenced by that profile will be updated.

dnf [options] module remove <module-spec>... Remove installed module profiles, including packages that were installed with the *dnf module install* command. Will not remove packages required by other installed module profiles or by other user-installed packages. In case no profile was provided, all installed profiles get removed.

dnf [options] module remove --all <module-spec>... Remove installed module profiles, including packages that were installed with the *dnf module install* command. With –all option it additionally removes all packages whose names are provided by specified modules. Packages required by other installed module profiles and packages whose names are also provided by any other module are not removed.

dnf [options] module enable <module-spec>... Enable a module stream and make the stream RPMs available in the package set.

Modular dependencies are resolved, dependencies checked and also recursively enabled. In case of modular dependency issue the operation will be rejected. To perform the action anyway please use *--skip-broken* option.

This command cannot be used for switching module streams. It is recommended to remove all installed content from the module, and reset the module using the *reset* command. After you reset the module, you can enable the other stream.

dnf [options] module disable <module-name>... Disable a module. All related module streams will become unavailable. Consequently, all installed profiles will be removed and the module RPMs will become unavailable in the package set. In case of modular dependency issue the operation will be rejected. To perform the action anyway please use *--skip-broken* option.

dnf [options] module reset <module-name>... Reset module state so it's no longer enabled or disabled. Consequently, all installed profiles will be removed and only RPMs from the default stream will be available in the package set.

dnf [options] module provides <package-name-spec>... Lists all modular packages matching `<package-name-spec>` from all modules (including disabled), along with the modules and streams they belong to.

dnf [options] module list [--all] [module_name...] Lists all module streams, their profiles and states (enabled, disabled, default).

dnf [options] module list --enabled [module_name...] Lists module streams that are enabled.

dnf [options] module list --disabled [module_name...] Lists module streams that are disabled.

dnf [options] module list --installed [module_name...] List module streams with installed profiles.

dnf [options] module info <module-spec>... Print detailed information about given module stream.

dnf [options] module info --profile <module-spec>... Print detailed information about given module profiles.

dnf [options] module repoquery <module-spec>... List all available packages belonging to selected modules.

dnf [options] module repoquery --available <module-spec>... List all available packages belonging to selected modules.

dnf [options] module repoquery --installed <module-spec>... List all installed packages with same name like packages belonging to selected modules.

2.4.22 Provides Command

dnf [options] provides <provide-spec> Finds the packages providing the given `<provide-spec>`. This is useful when one knows a filename and wants to find what package (installed or not) provides this file. The `<provide-spec>` is gradually looked for at following locations:

1. The `<provide-spec>` is matched with all file provides of any available package:

```
$ dnf provides /usr/bin/gzip
gzip-1.9-9.fc29.x86_64 : The GNU data compression program
Matched from:
Filename    : /usr/bin/gzip
```

2. Then all provides of all available packages are searched:

```
$ dnf provides "gzip(x86-64)"
gzip-1.9-9.fc29.x86_64 : The GNU data compression program
Matched from:
Provide     : gzip(x86-64) = 1.9-9.fc29
```

3. DNF assumes that the `<provide-spec>` is a system command, prepends it with `/usr/bin/`, `/usr/sbin/` prefixes (one at a time) and does the file provides search again. For legacy reasons (packages that didn't do UsrMove) also `/bin` and `/sbin` prefixes are being searched:

```
$ dnf provides zless
gzip-1.9-9.fc29.x86_64 : The GNU data compression program
Matched from:
Filename    : /usr/bin/zless
```

4. If this last step also fails, DNF returns "Error: No Matches found".

This command by default does not force a sync of expired metadata. See also *Metadata Synchronization*.

2.4.23 Reinstall Command

dnf [options] reinstall <package-spec>... Installs the specified packages, fails if some of the packages are either not installed or not available (i.e. there is no repository where to download the same RPM).

2.4.24 Remove Command

dnf [options] remove <package-spec>... Removes the specified packages from the system along with any packages depending on the packages being removed. Each <spec> can be either a <package-spec>, which specifies a package directly, or a @<group-spec>, which specifies an (environment) group which contains it. If clean_requirements_on_remove is enabled (the default), also removes any dependencies that are no longer needed.

dnf [options] remove --duplicates Removes older versions of duplicate packages. To ensure the integrity of the system it reinstalls the newest package. In some cases the command cannot resolve conflicts. In such cases the *dnf shell* command with remove --duplicates and upgrade dnf-shell sub-commands could help.

dnf [options] remove --oldinstallonly Removes old installonly packages, keeping only latest versions and version of running kernel.

There are also a few specific remove commands remove-n, remove-na and remove-nevra that allow the specification of an exact argument in the NEVRA format.

Remove Examples

dnf remove acpi tito Remove the acpi and tito packages.

dnf remove $(dnf repoquery --extras --exclude=tito,acpi) Remove packages not present in any repository, but don't remove the tito and acpi packages (they still might be removed if they depend on some of the removed packages).

Remove older versions of duplicated packages (an equivalent of yum's *package-cleanup –cleandups*):

```
dnf remove --duplicates
```

2.4.25 Repoinfo Command

An alias for the *repolist* command that provides more detailed information like dnf repolist -v.

2.4.26 Repolist Command

dnf [options] repolist [--enabled|--disabled|--all] Depending on the exact command lists enabled, disabled or all known repositories. Lists all enabled repositories by default. Provides more detailed information when -v option is used.

This command by default does not force a sync of expired metadata. See also *Metadata Synchronization*.

2.4.27 Repoquery Command

dnf [options] repoquery [<select-options>] [<query-options>] [<package-file-spec>] Searches available DNF repositories for selected packages and displays the requested information about them. It is an equivalent of rpm -q for remote repositories.

dnf [options] repoquery --querytags Provides the list of tags recognized by the *--queryformat* repoquery option.

There are also a few specific repoquery commands `repoquery-n`, `repoquery-na` and `repoquery-nevra` that allow the specification of an exact argument in the NEVRA format (does not affect arguments of options like –whatprovides <arg>, . . .).

Select Options

Together with `<package-file-spec>`, control what packages are displayed in the output. If `<package-file-spec>` is given, limits the resulting set of packages to those matching the specification. All packages are considered if no `<package-file-spec>` is specified.

<package-file-spec> Package specification in the NEVRA format (name[-[epoch:]version[-release]][.arch]), a package provide or a file provide. See *Specifying Packages*.

-a, --all Query all packages (for rpmquery compatibility, also a shorthand for repoquery '*' or repoquery without arguments).

--arch <arch>[,<arch>...], --archlist <arch>[,<arch>...] Limit the resulting set only to packages of selected architectures (default is all architectures). In some cases the result is affected by the basearch of the running system, therefore to run repoquery for an arch incompatible with your system use the `--forcearch=<arch>` option to change the basearch.

--duplicates Limit the resulting set to installed duplicate packages (i.e. more package versions for the same name and architecture). Installonly packages are excluded from this set.

--unneeded Limit the resulting set to leaves packages that were installed as dependencies so they are no longer needed. This switch lists packages that are going to be removed after executing the `dnf autoremove` command.

--available Limit the resulting set to available packages only (set by default).

--disable-modular-filtering Disables filtering of modular packages, so that packages of inactive module streams are included in the result.

--extras Limit the resulting set to packages that are not present in any of the available repositories.

-f <file>, --file <file> Limit the resulting set only to the package that owns `<file>`.

--installed Limit the resulting set to installed packages only. The *exclude* option in the configuration file might influence the result, but if the command line option *--disableexcludes* is used, it ensures that all installed packages will be listed.

--installonly Limit the resulting set to installed installonly packages.

--latest-limit <number> Limit the resulting set to <number> of latest packages for every package name and architecture. If <number> is negative, skip <number> of latest packages. For a negative <number> use the `--latest-limit=<number>` syntax.

--recent Limit the resulting set to packages that were recently edited.

--repo <repoid> Limit the resulting set only to packages from a repository identified by `<repoid>`. Can be used multiple times with accumulative effect.

--unsatisfied Report unsatisfied dependencies among installed packages (i.e. missing requires and and existing conflicts).

--upgrades Limit the resulting set to packages that provide an upgrade for some already installed package.

--userinstalled Limit the resulting set to packages installed by the user. The *exclude* option in the configuration file might influence the result, but if the command line option *--disableexcludes* is used, it ensures that all installed packages will be listed.

--whatdepends <capability>[,<capability>...] Limit the resulting set only to packages that require, enhance, recommend, suggest or supplement any of `<capabilities>`.

--whatconflicts <capability>[,<capability>...] Limit the resulting set only to packages that conflict with any of `<capabilities>`.

--whatenhances <capability>[,<capability>...] Limit the resulting set only to packages that enhance any of `<capabilities>`. Use *--whatdepends* if you want to list all depending packages.

--whatobsoletes <capability>[,<capability>...] Limit the resulting set only to packages that obsolete any of `<capabilities>`.

--whatprovides <capability>[,<capability>...] Limit the resulting set only to packages that provide any of `<capabilities>`.

--whatrecommends <capability>[,<capability>...] Limit the resulting set only to packages that recommend any of `<capabilities>`. Use *--whatdepends* if you want to list all depending packages.

--whatrequires <capability>[,<capability>...] Limit the resulting set only to packages that require any of `<capabilities>`. Use *--whatdepends* if you want to list all depending packages.

--whatsuggests <capability>[,<capability>...] Limit the resulting set only to packages that suggest any of `<capabilities>`. Use *--whatdepends* if you want to list all depending packages.

--whatsupplements <capability>[,<capability>...] Limit the resulting set only to packages that supplement any of `<capabilities>`. Use *--whatdepends* if you want to list all depending packages.

--alldeps This option is stackable with `--whatrequires` or *--whatdepends* only. Additionally it adds all packages requiring the package features to the result set (used as default).

--exactdeps This option is stackable with `--whatrequires` or *--whatdepends* only. Limit the resulting set only to packages that require `<capability>` specified by –whatrequires.

--srpm Operate on the corresponding source RPM.

Query Options

Set what information is displayed about each package.

The following are mutually exclusive, i.e. at most one can be specified. If no query option is given, matching packages are displayed in the standard NEVRA notation.

-i, --info Show detailed information about the package.

-l, --list Show the list of files in the package.

-s, --source Show the package source RPM name.

--changelogs Print the package changelogs.

--conflicts Display capabilities that the package conflicts with. Same as `--qf "%{conflicts}`.

--depends Display capabilities that the package depends on, enhances, recommends, suggests or supplements.

--enhances Display capabilities enhanced by the package. Same as `--qf "%{enhances}""`.

--location Show a location where the package could be downloaded from.

--obsoletes Display capabilities that the package obsoletes. Same as `--qf "%{obsoletes}"`.

--provides Display capabilities provided by the package. Same as `--qf "%{provides}"`.

--recommends Display capabilities recommended by the package. Same as `--qf "%{recommends}"`.

--requires Display capabilities that the package depends on. Same as `--qf "%{requires}"`.

--requires-pre Display capabilities that the package depends on for running a `%pre` script. Same as `--qf "%{requires-pre}"`.

--suggests Display capabilities suggested by the package. Same as `--qf "%{suggests}"`.

--supplements Display capabilities supplemented by the package. Same as `--qf "%{supplements}"`.

--tree Display a recursive tree of packages with capabilities specified by one of the following supplementary options: `--whatrequires`, `--requires`, `--conflicts`, `--enhances`, `--suggests`, `--provides`, `--supplements`, `--recommends`.

--deplist Produce a list of all direct dependencies and what packages provide those dependencies for the given packages. The result only shows the newest providers (which can be changed by using –verbose).

--nvr Show found packages in the name-version-release format. Same as `--qf "%{name}-%{version}-%{release}"`.

--nevra Show found packages in the name-epoch:version-release.architecture format. Same as `--qf "%{name}-%{epoch}:%{version}-%{release}.%{arch}"` (default).

--envra Show found packages in the epoch:name-version-release.architecture format. Same as `--qf "%{epoch}:%{name}-%{version}-%{release}.%{arch}"`

--qf <format>, --queryformat <format> Custom display format. `<format>` is the string to output for each matched package. Every occurrence of `%{<tag>}` within is replaced by the corresponding attribute of the package. The list of recognized tags can be displayed by running `dnf repoquery --querytags`.

--recursive Query packages recursively. Has to be used with `--whatrequires <REQ>` (optionally with `--alldeps`, but not with `--exactdeps`) or with `--requires <REQ> --resolve`.

--resolve resolve capabilities to originating package(s).

Examples

Display NEVRAs of all available packages matching `light*`:

```
dnf repoquery 'light*'
```

Display NEVRAs of all available packages matching name `light*` and architecture `noarch` (accepts only arguments in the “<name>.<arch>” format):

```
dnf repoquery-na 'light*.noarch'
```

Display requires of all lighttpd packages:

```
dnf repoquery --requires lighttpd
```

Display packages providing the requires of python packages:

```
dnf repoquery --requires python --resolve
```

Display source rpm of ligttpd package:

```
dnf repoquery --source lighttpd
```

Display package name that owns the given file:

```
dnf repoquery --file /etc/lighttpd/lighttpd.conf
```

Display name, architecture and the containing repository of all lighttpd packages:

```
dnf repoquery --queryformat '%{name}.%{arch} : %{reponame}' lighttpd
```

Display all available packages providing "webserver":

```
dnf repoquery --whatprovides webserver
```

Display all available packages providing "webserver" but only for "i686" architecture:

```
dnf repoquery --whatprovides webserver --arch i686
```

Display duplicate packages:

```
dnf repoquery --duplicates
```

Display source packages that require a <provide> for a build:

```
dnf repoquery --disablerepo="*" --enablerepo="*-source" --arch=src --whatrequires
↪<provide>
```

2.4.28 Repo-Pkgs Command

dnf [options] repo-pkgs Deprecated alias for the *Repository-Packages Command*.

2.4.29 Repository-Packages Command

The repository-packages command allows the user to run commands on top of all packages in the repository named `<repoid>`. However, any dependency resolution takes into account packages from all enabled repositories. The `<package-file-spec>` and `<package-spec>` specifications further limit the candidates to only those packages matching at least one of them.

The `info` subcommand lists description and summary information about packages depending on the packages' relation to the repository. The `list` subcommand just prints lists of those packages.

dnf [options] repository-packages <repoid> check-update [<package-file-spec>...]
Non-interactively checks if updates of the specified packages in the repository are available. DNF exit code will be 100 when there are updates available and a list of the updates will be printed.

dnf [options] repository-packages <repoid> info [--all] [<package-file-spec>...]
List all related packages.

dnf [options] repository-packages <repoid> info --installed [<package-file-spec>...]
List packages installed from the repository.

dnf [options] repository-packages <repoid> info --available [<package-file-spec>...]
List packages available in the repository but not currently installed on the system.

dnf [options] repository-packages <repoid> info --extras [<package-file-specs>...]
List packages installed from the repository that are not available in any repository.

dnf [options] repository-packages <repoid> info --obsoletes [<package-file-spec>...]
List packages in the repository that obsolete packages installed on the system.

dnf [options] repository-packages <repoid> info --recent [<package-file-spec>...]
List packages recently added into the repository.

dnf [options] repository-packages <repoid> info --upgrades [<package-file-spec>...]
List packages in the repository that upgrade packages installed on the system.

dnf [options] repository-packages <repoid> install [<package-spec>...] Install all packages in the repository.

dnf [options] repository-packages <repoid> list [--all] [<package-file-spec>...]
List all related packages.

dnf [options] repository-packages <repoid> list --installed [<package-file-spec>...]
List packages installed from the repository.

dnf [options] repository-packages <repoid> list --available [<package-file-spec>...]
List packages available in the repository but not currently installed on the system.

dnf [options] repository-packages <repoid> list --extras [<package-file-spec>...]
List packages installed from the repository that are not available in any repository.

dnf [options] repository-packages <repoid> list --obsoletes [<package-file-spec>...]
List packages in the repository that obsolete packages installed on the system.

dnf [options] repository-packages <repoid> list --recent [<package-file-spec>...]
List packages recently added into the repository.

dnf [options] repository-packages <repoid> list --upgrades [<package-file-spec>...]
List packages in the repository that upgrade packages installed on the system.

dnf [options] repository-packages <repoid> move-to [<package-spec>...] Reinstall all those packages that are available in the repository.

dnf [options] repository-packages <repoid> reinstall [<package-spec>...] Run the `reinstall-old` subcommand. If it fails, run the `move-to` subcommand.

dnf [options] repository-packages <repoid> reinstall-old [<package-spec>...]
Reinstall all those packages that were installed from the repository and simultaneously are available in the repository.

dnf [options] repository-packages <repoid> remove [<package-spec>...] Remove all packages installed from the repository along with any packages depending on the packages being removed. If `clean_requirements_on_remove` is enabled (the default) also removes any dependencies that are no longer needed.

dnf [options] repository-packages <repoid> remove-or-distro-sync [<package-spec>...]
Select all packages installed from the repository. Upgrade, downgrade or keep those of them that are available in another repository to match the latest version available there and remove the others along with any packages depending on the packages being removed. If `clean_requirements_on_remove` is enabled (the default) also removes any dependencies that are no longer needed.

dnf [options] repository-packages <repoid> remove-or-reinstall [<package-spec>...]
Select all packages installed from the repository. Reinstall those of them that are available in another repository and remove the others along with any packages depending on the packages being removed. If `clean_requirements_on_remove` is enabled (the default) also removes any dependencies that are no longer needed.

dnf [options] repository-packages <repoid> upgrade [<package-spec>...] Update all packages to the highest resolvable version available in the repository.

dnf [options] repository-packages <repoid> upgrade-to <package-nevr-specs>... Update packages to the specified versions that are available in the repository. Upgrade-to is a deprecated alias for the upgrade subcommand.

2.4.30 Search Command

dnf [options] search [--all] <keywords>... Search package metadata for keywords. Keywords are matched as case-insensitive substrings, globbing is supported. By default lists packages that match all requested keys (AND operation). Keys are searched in package names and summaries. If the "–all" option is used, lists packages that match at least one of the keys (an OR operation). In addition the keys are searched in the package descriptions and URLs. The result is sorted from the most relevant results to the least.

This command by default does not force a sync of expired metadata. See also *Metadata Synchronization*.

2.4.31 Shell Command

dnf [options] shell [filename] Open an interactive shell for conducting multiple commands during a single execution of DNF. These commands can be issued manually or passed to DNF from a file. The commands are much the same as the normal DNF command line options. There are a few additional commands documented below.

config [conf-option] [value]

- Set a configuration option to a requested value. If no value is given it prints the current value.

repo [list|enable|disable] [repo-id]

- list: list repositories and their status
- enable: enable repository
- disable: disable repository

transaction [list|reset|solve|run]

- list: resolve and list the content of the transaction
- reset: reset the transaction
- run: resolve and run the transaction

Note that all local packages must be used in the first shell transaction subcommand (e.g. *install /tmp/nodejs-1-1.x86_64.rpm /tmp/acpi-1-1.noarch.rpm*) otherwise an error will occur. Any *disable*, *enable*, and *reset* module operations (e.g. *module enable nodejs*) must also be performed before any other shell transaction subcommand is used.

2.4.32 Swap Command

`dnf [options] swap <remove-spec> <install-spec>`

Remove spec and install spec in one transaction. Each `<spec>` can be either a *<package-spec>*, which specifies a package directly, or a `@<group-spec>`, which specifies an (environment) group which contains it. Automatic conflict solving is provided in DNF by the –allowerasing option that provides the functionality of the swap command automatically.

2.4.33 Update Command

dnf [options] update Deprecated alias for the *Upgrade Command*.

2.4.34 Updateinfo Command

dnf [options] updateinfo [--summary|--list|--info] [<availability>] [<spec>...]
Display information about update advisories.

Depending on the output type, DNF displays just counts of advisory types (omitted or `--summary`), list of advisories (`--list`) or detailed information (`--info`). When the `-v` option is used with `--info`, the information is even more detailed.

`<availability>` specifies whether advisories about newer versions of installed packages (omitted or `--available`), advisories about equal and older versions of installed packages (`--installed`), advisories about newer versions of those installed packages for which a newer version is available (`--updates`) or advisories about any versions of installed packages (`--all`) are taken into account. Most of the time, `--available` and `--updates` displays the same output. The outputs differ only in the cases when an advisory refers to a newer version but there is no enabled repository which contains any newer version.

Note, that `--available` tooks only the latest installed versions of packages into account. In case of the kernel packages (when multiple version could be installed simultaneously) also packages of the currently running version of kernel are added.

To print only advisories referencing a CVE or a bugzilla use `--with-cve` or `--with-bz` options. When these switches are used also the output of the `--list` is altered - the ID of the CVE or the bugzilla is printed instead of the one of the advisory.

If given and if neither ID, type (`bugfix`, `enhancement`, `security`/`sec`) nor a package name of an advisory matches `<spec>`, the advisory is not taken into account. The matching is case-sensitive and in the case of advisory IDs and package names, globbing is supported.

Output of the `--summary` option is affected by the *autocheck_running_kernel* configuration option.

2.4.35 Update-Minimal Command

dnf [options] update-minimal Deprecated alias for the *Upgrade-Minimal Command*.

2.4.36 Upgrade Command

dnf [options] upgrade Updates each package to the latest version that is both available and resolvable.

dnf [options] upgrade <package-spec>... Updates each specified package to the latest available version. Updates dependencies as necessary.

dnf [options] upgrade <package-nevr-specs>... Upgrades packages to the specified versions.

dnf [options] upgrade @<spec>... Alias for the *dnf module update* command.

If the main `obsoletes` configure option is true or the `--obsoletes` flag is present, dnf will include package obsoletes in its calculations. For more information see *obsoletes*.

See also *Configuration Files Replacement Policy*.

2.4.37 Upgrade-Minimal Command

dnf [options] upgrade-minimal Updates each package to the latest available version that provides a bug-fix, enhancement or a fix for a security issue (security).

dnf [options] upgrade-minimal <package-spec>... Updates each specified package to the latest available version that provides a bugfix, enhancement or a fix for security issue (security). Updates dependencies as necessary.

2.4.38 Update-To Command

dnf [options] update-to <package-nevr-specs>... Deprecated alias for the *Upgrade Command*.

2.4.39 Upgrade-To Command

dnf [options] upgrade-to <package-nevr-specs>... Deprecated alias for the *Upgrade Command*.

2.5 Specifying Packages

Many commands take a `<package-spec>` parameter that selects a package for the operation. The `<package-spec>` argument is matched against package NEVRAs, provides and file provides.

`<package-file-spec>` is similar to `<package-spec>`, except provides matching is not performed. Therefore, `<package-file-spec>` is matched only against NEVRAs and file provides.

`<package-name-spec>` is matched against NEVRAs only.

2.5.1 Globs

Package specification supports the same glob pattern matching that shell does, in all three above mentioned packages it matches against (NEVRAs, provides and file provides).

The following patterns are supported:

***** Matches any number of characters.

? Matches any single character.

[] Matches any one of the enclosed characters. A pair of characters separated by a hyphen denotes a range expression; any character that falls between those two characters, inclusive, is matched. If the first character following the `[` is a `!` or a `^` then any character not enclosed is matched.

{} Matches any of the comma separated list of enclosed strings.

2.5.2 NEVRA Matching

When matching against NEVRAs, partial matching is supported. DNF tries to match the spec against the following list of NEVRA forms (in decreasing order of priority):

- `name-[epoch:]version-release.arch`
- `name.arch`
- `name`

- `name-[epoch:]version-release`
- `name-[epoch:]version`

Note that `name` can in general contain dashes (e.g. `package-with-dashes`).

The first form that matches any packages is used and the remaining forms are not tried. If none of the forms match any packages, an attempt is made to match the `<package-spec>` against full package NEVRAs. This is only relevant if globs are present in the `<package-spec>`.

`<package-spec>` matches NEVRAs the same way `<package-name-spec>` does, but in case matching NEVRAs fails, it attempts to match against provides and file provides of packages as well.

You can specify globs as part of any of the five NEVRA components. You can also specify a glob pattern to match over multiple NEVRA components (in other words, to match across the NEVRA separators). In that case, however, you need to write the spec to match against full package NEVRAs, as it is not possible to split such spec into NEVRA forms.

2.6 Specifying Exact Versions of Packages

Commands accepting the `<package-nevr-spec>` parameter need not only the name of the package, but also its version, release and optionally the architecture. Further, the version part can be preceded by an epoch when it is relevant (i.e. the epoch is non-zero).

2.7 Specifying Provides

`<provide-spec>` in command descriptions means the command operates on packages providing the given spec. This can either be an explicit provide, an implicit provide (i.e. name of the package) or a file provide. The selection is case-sensitive and globbing is supported.

2.8 Specifying Groups

`<group-spec>` allows one to select (environment) groups a particular operation should work on. It is a case insensitive string (supporting globbing characters) that is matched against a group's ID, canonical name and name translated into the current LC_MESSAGES locale (if possible).

2.9 Specifying Modules

`<module-spec>` allows one to select modules or profiles a particular operation should work on.

It is in the form of `NAME:STREAM:VERSION:CONTEXT:ARCH/PROFILE` and supported partial forms are the following:

- `NAME`
- `NAME:STREAM`
- `NAME:STREAM:VERSION`
- `NAME:STREAM:VERSION:CONTEXT`
- all above combinations with `::ARCH` (e.g. `NAME::ARCH`)

- `NAME:STREAM:VERSION:CONTEXT:ARCH`
- all above combinations with `/PROFILE` (e.g. `NAME/PROFILE`)

In case stream is not specified, the enabled or the default stream is used, in this order. In case profile is not specified, the system default profile or the 'default' profile is used.

2.10 Specifying Transactions

`<transaction-spec>` can be in one of several forms. If it is an integer, it specifies a transaction ID. Specifying `last` is the same as specifying the ID of the most recent transaction. The last form is `last-<offset>`, where `<offset>` is a positive integer. It specifies offset-th transaction preceding the most recent transaction.

2.11 Package Filtering

Package filtering filters packages out from the available package set, making them invisible to most of dnf commands. They cannot be used in a transaction. Packages can be filtered out by either Exclude Filtering or Modular Filtering.

2.11.1 Exclude Filtering

Exclude Filtering is a mechanism used by a user or by a DNF plugin to modify the set of available packages. Exclude Filtering can be modified by either *includepkgs* or *excludepkgs* configuration options in *configuration files*. The *--disableexcludes* command line option can be used to override excludes from configuration files. In addition to user-configured excludes, plugins can also extend the set of excluded packages. To disable excludes from a DNF plugin you can use the *--disableplugin* command line option.

To disable all excludes for e.g. the install command you can use the following combination of command line options:

```
dnf --disableexcludes=all --disableplugin="*" install bash
```

2.11.2 Modular Filtering

Please see *the modularity documentation* for details on how Modular Filtering works.

With modularity, only RPM packages from `active` module streams are included in the available package set. RPM packages from `inactive` module streams, as well as non-modular packages with the same name or provides as a package from an `active` module stream, are filtered out. Modular filtering is not applied to packages added from the command line, installed packages, or packages from repositories with `module_hotfixes=true` in their `.repo` file.

Disabling of modular filtering is not recommended, because it could cause the system to get into a broken state. To disable modular filtering for a particular repository, specify `module_hotfixes=true` in the `.repo` file or use `--setopt=<repo_id>.module_hotfixes=true`.

To discover the module which contains an excluded package use *dnf module provides*.

2.12 Metadata Synchronization

Correct operation of DNF depends on having access to up-to-date data from all enabled repositories but contacting remote mirrors on every operation considerably slows it down and costs bandwidth for both the client and the repository provider. The *metadata_expire* (see *dnf.conf(5)*) repository configuration option is used by DNF to determine

whether a particular local copy of repository data is due to be re-synced. It is crucial that the repository providers set the option well, namely to a value where it is guaranteed that if particular metadata was available in time `T` on the server, then all packages it references will still be available for download from the server in time `T + metadata_expire`.

To further reduce the bandwidth load, some of the commands where having up-to-date metadata is not critical (e.g. the `list` command) do not look at whether a repository is expired and whenever any version of it is locally available to the user's account, it will be used. For non-root use, see also the `--cacheonly` switch. Note that in all situations the user can force synchronization of all enabled repositories with the `--refresh` switch.

2.13 Configuration Files Replacement Policy

The updated packages could replace the old modified configuration files with the new ones or keep the older files. Neither of the files are actually replaced. To the conflicting ones RPM gives additional suffix to the origin name. Which file should maintain the true name after transaction is not controlled by package manager but is specified by each package itself, following packaging guideline.

2.14 Files

`Cache Files` /var/cache/dnf

`Main Configuration` /etc/dnf/dnf.conf

`Repository` /etc/yum.repos.d/

2.15 See Also

- *`dnf.conf(5)`*, *DNF Configuration Reference*
- *`dnf-PLUGIN(8)`* for documentation on DNF plugins.
- *`dnf.modularity(7)`*, *Modularity overview*.
- **'DNF'_** project homepage (https://github.com/rpm-software-management/dnf/)
- How to report a bug (https://github.com/rpm-software-management/dnf/wiki/Bug-Reporting)
- **'YUM'_** project homepage (http://yum.baseurl.org/)

CHAPTER 3

DNF Configuration Reference

3.1 Description

'DNF'_ by default uses the global configuration file at `/etc/dnf/dnf.conf` and all *.repo files found under `/etc/yum.repos.d`. The latter is typically used for repository configuration and takes precedence over global configuration.

The configuration file has INI format consisting of section declaration and `name=value` options below each on separate line. There are two types of sections in the configuration files: main and repository. Main section defines all global configuration options and should be only one.

The repository sections define the configuration for each (remote or local) repository. The section name of the repository in brackets serve as repo ID reference and should be unique across configuration files. The allowed characters of repo ID string are lower and upper case alphabetic letters, digits, -, _, `.` and `:`. The minimal repository configuration file should aside from repo ID consists of *baseurl*, *metalink* or *mirrorlist* option definition.

3.2 Distribution-Specific Configuration

Configuration options, namely *best* and *skip_if_unavailable*, can be set in the DNF configuration file by your distribution to override the DNF defaults.

3.3 [main] Options

`arch` *string*

The architecture used for installing packages. By default this is auto-detected. Often used together with *ignorearch* option.

`assumeno` *boolean*

If enabled dnf will assume `No` where it would normally prompt for confirmation from user input. Default is `False`.

assumeyes *boolean*

If enabled dnf will assume `Yes` where it would normally prompt for confirmation from user input (see also *defaultyes*). Default is `False`.

autocheck_running_kernel *boolean*

Automatic check whether there is installed newer kernel module with security update than currently running kernel. Default is `True`.

basearch *string*

The base architecture used for installing packages. By default this is auto-detected.

best *boolean*

`True` instructs the solver to either use a package with the highest available version or fail. On `False`, do not fail if the latest version cannot be installed and go with the lower version. The default is `True`. Note this option in particular *can be set in your configuration file by your distribution*.

cachedir *string*

Path to a directory used by various DNF subsystems for storing cache data. Has a reasonable root-writable default depending on the distribution. DNF needs to be able to create files and directories at this location.

cacheonly *boolean*

If set to `True` DNF will run entirely from system cache, will not update the cache and will use it even in case it is expired. Default is `False`.

check_config_file_age *boolean*

Specifies whether dnf should automatically expire metadata of repos, which are older than their corresponding configuration file (usually the dnf.conf file and the foo.repo file). Default is `True` (perform the check). Expire of metadata is also affected by metadata age. See also *metadata_expire*.

clean_requirements_on_remove *boolean*

Remove dependencies that are no longer used during `dnf remove`. A package only qualifies for removal via `clean_requirements_on_remove` if it was installed through DNF but not on explicit user request, i.e. it was pulled in as a dependency. The default is True. (*installonlypkgs* are never automatically removed.)

config_file_path *string*

Path to the default main configuration file. Default is `/etc/dnf/dnf.conf`.

debuglevel *integer*

Debug messages output level, in the range 0 to 10. The higher the number the more debug output is put to stdout. Default is 2.

debug_solver *boolean*

Controls whether the libsolv debug files should be created when solving the transaction. The debug files are created in the *./debugdata* directory. Default is `False`.

defaultyes *boolean*

If enabled the default answer to user confirmation prompts will be `Yes`. Not to be confused with *assumeyes* which will not prompt at all. Default is `False`.

diskspacecheck *boolean*

Controls wheather rpm shoud check available disk space during the transaction. Default is `True`.

errorlevel *integer*

Error messages output level, in the range 0 to 10. The higher the number the more error output is put to stderr. Default is 3. This is deprecated in DNF and overwritten by *--verbose* commandline option.

exit_on_lock *boolean*

Should the dnf client exit immediately when something else has the lock. Default is `False`.

gpgkey_dns_verification *boolean*

Should the dnf attempt to automatically verify GPG verification keys using the DNS system. This option requires libunbound to be installed on the client system. This system has two main features. The first one is to check if any of the already installed keys have been revoked. Automatic removal of the key is not yet available, so it is up to the user, to remove revoked keys from the system. The second feature is automatic verification of new keys when a repository is added to the system. In interactive mode, the result is written to the output as a suggestion to the user. In non-interactive mode (i.e. when -y is used), this system will automatically accept keys that are available in the DNS and are correctly signed using DNSSEC. It will also accept keys that do not exist in the DNS system and their NON-existence is cryptographically proven using DNSSEC. This is mainly to preserve backward compatibility. Default is `False`.

group_package_types *list*

List of the following: optional, default, mandatory. Tells dnf which type of packages in groups will be installed when 'groupinstall' is called. Default is: `default, mandatory`.

ignorearch *boolean*

If set to `True`, RPM will allow attempts to install packages incompatible with the CPU's architecture. Defaults to `False`. Often used together with *arch* option.

installonlypkgs *list*

List of provide names of packages that should only ever be installed, never upgraded. Kernels in particular fall into this category. These packages are never removed by `dnf autoremove` even if they were installed as dependencies (see *clean_requirements_on_remove* for auto removal details). This option append the list values to the default installonlypkgs list used by DNF. The number of kept package versions is regulated by *installonly_limit*.

installonly_limit *integer*

Number of *installonly packages* allowed to be installed concurrently. Defaults to 3. The minimal number of installonly packages is 2. Value 0 or 1 means unlimited number of installonly packages.

installroot *string*

The root of the filesystem for all packaging operations. It requires an absolute path. See also *–installroot commandline option*.

install_weak_deps *boolean*

When this option is set to True and a new package is about to be installed, all packages linked by weak dependency relation (Recommends or Supplements flags) with this package will be pulled into the transaction. Default is `True`.

keepcache *boolean*

Keeps downloaded packages in the cache when set to True. Even if it is set to False and packages have not been installed they will still persist until next successful transaction. The default is `False`.

logdir *string*

Directory where the log files will be stored. Default is `/var/log`.

log_rotate *integer*

Log files are rotated `log_rotate` times before being removed. If `log_rotate` is `0`, the rotation is not performed. Default is `4`.

log_size storage size

Log files are rotated when they grow bigger than log_size bytes. If log_size is 0, the rotation is not performed. The default is 1 MB. Valid units are 'k', 'M', 'G'.

The size applies for individual log files, not the sum of all log files. See also *log_rotate*.

metadata_timer_sync time in seconds

The minimal period between two consecutive `makecache timer` runs. The command will stop immediately if it's less than this time period since its last run. Does not affect simple `makecache` run. Use `0` to completely disable automatic metadata synchronizing. The default corresponds to three hours. The value is rounded to the next commenced hour.

module_platform_id *string*

Set this to $name:$stream to override PLATFORM_ID detected from `/etc/os-release`. It is necessary to perform a system upgrade and switch to a new platform.

multilib_policy *string*

Controls how multilib packages are treated during install operations. Can either be `"best"` (the default) for the depsolver to prefer packages which best match the system's architecture, or `"all"` to install all available packages with compatible architectures.

obsoletes *boolean*

This option only has affect during an install/update. It enables dnf's obsoletes processing logic, which means it makes dnf check whether any dependencies of given package are no longer required and removes them. Useful when doing distribution level upgrades. Default is 'true'.

Command-line option: *–obsoletes*

persistdir *string*

Directory where DNF stores its persistent data between runs. Default is `"/var/lib/dnf"`.

pluginconfpath *list*

List of directories that are searched for plugin configurations to load. All configuration files found in these directories, that are named same as a plugin, are parsed. The default path is `/etc/dnf/plugins`.

pluginpath *list*

List of directories that are searched for plugins to load. Plugins found in *any of the directories* in this configuration option are used. The default contains a Python version-specific path.

plugins *boolean*

Controls whether the plugins are enabled. Default is `True`.

protected_packages *list*

List of packages that DNF should never completely remove. They are protected via Obsoletes as well as user/plugin removals.

The default is: `dnf`, `glob:/etc/yum/protected.d/*.conf` and `glob:/etc/dnf/protected.d/*.conf`. So any packages which should be protected can do so by including a file in `/etc/dnf/protected.d` with their package name in it.

DNF will protect also the package corresponding to the running version of the kernel.

releasever *string*

Used for substitution of `$releasever` in the repository configuration. See also *repo variables*.

reposdir *list*

DNF searches for repository configuration files in the paths specified by `reposdir`. The behavior of `reposdir` could differ when it is used along with *--installroot* option.

rpmverbosity *string*

RPM debug scriptlet output level. One of: `critical`, `emergency`, `error`, `warn`, `info` or `debug`. Default is `info`.

strict *boolean*

If disabled, all unavailable packages or packages with broken dependencies given to DNF command will be skipped without raising the error causing the whole operation to fail. Currently works for install command only. The default is True.

tsflags *list*

List of strings adding extra flags for the RPM transaction.

tsflag value	RPM Transaction Flag
noscripts	RPMTRANS_FLAG_NOSCRIPTS
test	RPMTRANS_FLAG_TEST
notriggers	RPMTRANS_FLAG_NOTRIGGERS
nodocs	RPMTRANS_FLAG_NODOCS
justdb	RPMTRANS_FLAG_JUSTDB
nocontexts	RPMTRANS_FLAG_NOCONTEXTS
nocaps	RPMTRANS_FLAG_NOCAPS
nocrypto	RPMTRANS_FLAG_NOFILEDIGEST

The `nocrypto` option will also set the `_RPMVSF_NOSIGNATURES` and `_RPMVSF_NODIGESTS` VS flags. The `test` option provides a transaction check without performing the transaction. It includes downloading of packages, gpg keys check (including permanent import of additional keys if necessary), and rpm check to prevent file conflicts. The `nocaps` is supported with rpm-4.14 or later. When `nocaps` is used but rpm doesn't support it, DNF only reports it as an invalid tsflag.

upgrade_group_objects_upgrade *boolean*

Set this to False to disable the automatic running of `group upgrade` when running the `upgrade` command. Default is `True` (perform the operation).

varsdir *list*

List of directories where variables definition files are looked for. Defaults to `"/etc/dnf/vars", "/etc/yum/vars"`. See *variable files* in Configuration reference.

zchunk *boolean*

Enables or disables the use of repository metadata compressed using the zchunk format (if available). Default is `True`.

3.4 Repo Options

baseurl *list*

List of URLs for the repository. Defaults to `[]`.

cost *integer*

The relative cost of accessing this repository, defaulting to 1000. This value is compared when the priorities of two repositories are the same. The repository with *the lowest cost* is picked. It is useful to make the library prefer on-disk repositories to remote ones.

enabled *boolean*

Include this repository as a package source. The default is True.

gpgkey *list* of strings

URLs of a GPG key files that can be used for signing metadata and packages of this repository, empty by default. If a file can not be verified using the already imported keys, import of keys from this option is attempted and the keys are then used for verification.

metalink *string*

URL of a metalink for the repository. Defaults to `None`.

mirrorlist *string*

URL of a mirrorlist for the repository. Defaults to `None`.

module_hotfixes *boolean*

Set this to True to disable module RPM filtering and make all RPMs from the repository available. The default is False. This allows user to create a repository with cherry-picked hotfixes that are included in a package set on a modular system.

name *string*

A human-readable name of the repository. Defaults to the ID of the repository.

priority *integer*

The priority value of this repository, default is 99. If there is more than one candidate package for a particular operation, the one from a repo with *the lowest priority value* is picked, possibly despite being less convenient otherwise (e.g. by being a lower version).

type *string*

Type of repository metadata. Supported values are: `rpm-md`. Aliases for `rpm-md`: `rpm`, `repomd`, `rpmmd`, `yum`, `YUM`.

3.5 Repo Variables

Right side of every repo option can be enriched by the following variables:

`$arch`

Refers to the system's CPU architecture e.g, aarch64, i586, i686 and x86_64.

`$basearch`

Refers to the base architecture of the system. For example, i686 and i586 machines both have a base architecture of i386, and AMD64 and Intel64 machines have a base architecture of x86_64.

`$releasever`

Refers to the release version of operating system which DNF derives from information available in RPMDB.

In addition to these hard coded variables, user-defined ones can also be used. They can be defined either via *variable files*, or by using special environmental variables. The names of these variables must be prefixed with DNF_VAR_ and they can only consist of alphanumeric characters and underscores:

```
$ DNF_VAR_MY_VARIABLE=value
```

3.6 Options for both [main] and Repo

Some options can be applied in either the main section, per repository, or in a combination. The value provided in the main section is used for all repositories as the default value, which repositories can then override in their configuration.

bandwidth storage size

Total bandwidth available for downloading. Meaningful when used with the *throttle option*. Storage size is in bytes by default but can be specified with a unit of storage. Valid units are 'k', 'M', 'G'.

countme *boolean*

Determines whether a special flag should be added to a single, randomly chosen metalink/mirrorlist query each week. This allows the repository owner to estimate the number of systems consuming it, by counting such queries over a week's time, which is much more accurate than just counting unique IP addresses (which is subject to both overcounting and undercounting due to short DHCP leases and NAT, respectively).

The flag is a simple "countme=N" parameter appended to the metalink and mirrorlist URL, where N is an integer representing the "longevity" bucket this system belongs to. The following 4 buckets are defined, based on how many full weeks have passed since the beginning of the week when this system was installed: 1 = first week, 2 = first month (2-4 weeks), 3 = six months (5-24 weeks) and 4 = more than six months (> 24 weeks). This information is meant to help distinguish short-lived installs from long-term ones, and to gather other statistics about system lifecycle.

Default is False.

deltarpm *boolean*

When enabled, DNF will save bandwidth by downloading much smaller delta RPM files, rebuilding them to RPM locally. However, this is quite CPU and I/O intensive. Default is True.

deltarpm_percentage *integer*

When the relative size of delta vs pkg is larger than this, delta is not used. Default value is 75 (Deltas must be at least 25% smaller than the pkg). Use *0* to turn off delta rpm processing. Local repositories (with file:// baseurl) have delta rpms turned off by default.

enablegroups *boolean*

Determines whether DNF will allow the use of package groups for this repository. Default is True (package groups are allowed).

excludepkgs *list*

Exclude packages of this repository, specified by a name or a glob and separated by a comma, from all operations. Can be disabled using `--disableexcludes` command line switch. Defaults to `[]`.

fastestmirror *boolean*

If enabled a metric is used to find the fastest available mirror. This overrides the order provided by the mirrorlist/metalink file itself. This file is often dynamically generated by the server to provide the best download speeds and enabling fastestmirror overrides this. The default is False.

gpgcheck *boolean*

Whether to perform GPG signature check on packages found in this repository. The default is False.

This option can only be used to strengthen the active RPM security policy set with the `%_pkgverify_level` macro (see the `/usr/lib/rpm/macros` file for details). That means, if the macro is set to 'signature' or 'all' and this option is False, it will be overridden to True during DNF runtime, and a warning will be printed. To squelch the warning, make sure this option is True for every enabled repository, and also enable *localpkg_gpgcheck*.

includepkgs *list*

Include packages of this repository, specified by a name or a glob and separated by a comma, in all operations. Inverse of *excludepkgs*, DNF will exclude any package in the repository that doesn't match this list. This works in conjunction with `excludepkgs` and doesn't override it, so if you 'excludepkgs=*.i386' and 'includepkgs=python*' then only packages starting with python that do not have an i386 arch will be seen by DNF in this repo. Can be disabled using `--disableexcludes` command line switch. Defaults to `[]`.

ip_resolve IP address type

Determines how DNF resolves host names. Set this to '4'/'IPv4' or '6'/'IPv6' to resolve to IPv4 or IPv6 addresses only. By default, DNF resolves to either addresses.

localpkg_gpgcheck *boolean*

Whether to perform a GPG signature check on local packages (packages in a file, not in a repository). The default is False. This option is subject to the active RPM security policy (see *gpgcheck* for more details).

max_parallel_downloads *integer*

Maximum number of simultaneous package downloads. Defaults to 3.

metadata_expire time in seconds

The period after which the remote repository is checked for metadata update and in the positive case the local metadata cache is updated. The default corresponds to 48 hours. Set this to `-1` or `never` to make the repo never considered expired. Expire of metadata can bee also triggered by change of timestamp of configuration files (`dnf.conf`, `<repo>.repo`). See also *check_config_file_age*.

minrate storage size

This sets the low speed threshold in bytes per second. If the server is sending data at the same or slower speed than this value for at least *timeout option* seconds, DNF aborts the connection. The default is 1000. Valid units are 'k', 'M', 'G'.

password *string*

The password to use for connecting to a repository with basic HTTP authentication. Empty by default.

proxy *string*

URL of a proxy server to connect through. Set to an empty string to disable the proxy setting inherited from the main section and use direct connection instead. The expected format of this option is `<scheme>://<ip-or-hostname>[:port]`. (For backward compatibility, '_none_' can be used instead of the empty string.)

Note: The curl environment variables (such as `http_proxy`) are effective if this option is unset. See the `curl` man page for details.

proxy_username *string*

The username to use for connecting to the proxy server. Empty by default.

proxy_password *string*

The password to use for connecting to the proxy server. Empty by default.

proxy_auth_method *string*

The authentication method used by the proxy server. Valid values are

method	meaning
basic	HTTP Basic authentication
digest	HTTP Digest authentication
negotiate	HTTP Negotiate (SPNEGO) authentication
ntlm	HTTP NTLM authentication
digest_ie	HTTP Digest authentication with an IE flavor
ntlm_wb	NTLM delegating to winbind helper
none	None auth method
any	All suitable methods

Defaults to `any`

repo_gpgcheck *boolean*

Whether to perform GPG signature check on this repository's metadata. The default is False.

retries *integer*

Set the number of total retries for downloading packages. The number is accumulative, so e.g. for *retries=10*, dnf will fail after any package download fails for eleventh time. Setting this to *0* makes dnf try forever. Default is *10*.

skip_if_unavailable *boolean*

If enabled, DNF will continue running and disable the repository that couldn't be synchronized for any reason. This option doesn't affect skipping of unavailable packages after dependency resolution. To check inaccessibility of repository use it in combination with *refresh command line option*. The default is `False`. Note this option in particular *can be set in your configuration file by your distribution*.

sslcacert *string*

Path to the directory or file containing the certificate authorities to verify SSL certificates. Empty by default - uses system default.

sslverify *boolean*

When enabled, remote SSL certificates are verified. If the client can not be authenticated, connecting fails and the repository is not used any further. If `False`, SSL connections can be used, but certificates are not verified. Default is `True`.

sslclientcert *string*

Path to the SSL client certificate used to connect to remote sites. Empty by default.

sslclientkey *string*

Path to the SSL client key used to connect to remote sites. Empty by default.

throttle storage size

Limits the downloading speed. It might be an absolute value or a percentage, relative to the value of the *bandwidth option* option. `0` means no throttling (the default). The absolute value is in bytes by default but can be specified with a unit of storage. Valid units are 'k', 'M', 'G'.

timeout time in seconds

Number of seconds to wait for a connection before timing out. Used in combination with *minrate option* option. Defaults to 30 seconds.

username *string*

The username to use for connecting to repo with basic HTTP authentication. Empty by default.

user_agent *string*

The User-Agent string to include in HTTP requests sent by DNF. Defaults to

```
libdnf (NAME VERSION_ID; VARIANT_ID; OS.BASEARCH)
```

where NAME, VERSION_ID and VARIANT_ID are OS identifiers read from the *os-release(5)* file, and OS and BASEARCH are the canonical OS name and base architecture, respectively. Example:

```
libdnf (Fedora 31; server; Linux.x86_64)
```

3.7 Types of Options

boolean This is a data type with only two possible values.

One of following options can be used: 1, 0, True, False, yes, no

integer It is a whole number that can be written without a fractional component.

list It is an option that could represent one or more strings separated by space or comma characters.

string It is a sequence of symbols or digits without any whitespace character.

3.8 Files

Cache Files /var/cache/dnf

Main Configuration File /etc/dnf/dnf.conf

Repository /etc/yum.repos.d/

Variables Any properly named file in /etc/dnf/vars is turned into a variable named after the filename (or overrides any of the above variables but those set from commandline). Filenames may contain only alphanumeric characters and underscores and be in lowercase.

3.9 See Also

- *dnf(8)*, *DNF Command Reference*

CHAPTER 4

DNF Automatic

4.1 Synopsis

```
dnf-automatic [<config file>]
```

4.2 Description

Alternative CLI to `dnf upgrade` with specific facilities to make it suitable to be executed automatically and regularly from systemd timers, cron jobs and similar.

The operation of the tool is usually controlled by the configuration file or the function-specific timer units (see below). The command only accepts a single optional argument pointing to the config file, and some control arguments intended for use by the services that back the timer units. If no configuration file is passed from the command line, `/etc/dnf/automatic.conf` is used.

The tool synchronizes package metadata as needed and then checks for updates available for the given system and then either exits, downloads the packages or downloads and applies the packages. The outcome of the operation is then reported by a selected mechanism, for instance via the standard output, email or MOTD messages.

The systemd timer unit `dnf-automatic.timer` will behave as the configuration file specifies (see below) with regard to whether to download and apply updates. Some other timer units are provided which override the configuration file with some standard behaviours:

- dnf-automatic-notifyonly
- dnf-automatic-download
- dnf-automatic-install

Regardless of the configuration file settings, the first will only notify of available updates. The second will download, but not install them. The third will download and install them.

4.3 Run dnf-automatic

You can select one that most closely fits your needs, customize `/etc/dnf/automatic.conf` for any specific behaviors, and enable the timer unit.

For example: `systemctl enable --now dnf-automatic-notifyonly.timer`

4.4 Configuration File Format

The configuration file is separated into topical sections.

4.4.1 `[commands]` section

Setting the mode of operation of the program.

`apply_updates` boolean, default: False

> Whether packages comprising the available updates should be applied by `dnf-automatic.timer`, i.e. installed via RPM. Implies `download_updates`. Note that if this is set to `False`, downloaded packages will be left in the cache till the next successful DNF transaction. Note that the other timer units override this setting.

`download_updates` boolean, default: False

> Whether packages comprising the available updates should be downloaded by `dnf-automatic.timer`. Note that the other timer units override this setting.

`upgrade_type` either one of `default`, `security`, default: `default`

> What kind of upgrades to look at. `default` signals looking for all available updates, `security` only those with an issued security advisory.

`random_sleep` time in seconds, default: 0

> Maximal random delay before downloading. Note that, by default, the `systemd` timers also apply a random delay of up to 5 minutes.

4.4.2 `[emitters]` section

Choosing how the results should be reported.

`emit_via` list, default: `email, stdio, motd`

> List of emitters to report the results through. Available emitters are `stdio` to print the result to standard output, `command` to send the result to a custom command, `command_email` to send an email using a command, and `email` to send the report via email and `motd` sends the result to */etc/motd* file.

`system_name` string, default: hostname of the given system

> How the system is called in the reports.

4.4.3 `[command]` section

The command emitter configuration. Variables usable in format string arguments are `body` with the message body.

`command_format` format string, default: `cat`

> The shell command to execute.

stdin_format format string, default: `{body}`

The data to pass to the command on stdin.

4.4.4 [command_email] section

The command email emitter configuration. Variables usable in format string arguments are `body` with message body, `subject` with email subject, `email_from` with the "From:" address and `email_to` with a space-separated list of recipients.

command_format format string, default: `mail -s {subject} -r {email_from} {email_to}`

The shell command to execute.

stdin_format format string, default: `{body}`

The data to pass to the command on stdin.

email_from string, default: `root`

Message's "From:" address.

email_to list, default: `root`

List of recipients of the message.

4.4.5 [email] section

The email emitter configuration.

email_from string, default: `root`

Message's "From:" address.

email_to list, default: `root`

List of recipients of the message.

email_host string, default: `localhost`

Hostname of the SMTP server used to send the message.

4.4.6 [base] section

Can be used to override settings from DNF's main configuration file. See *DNF Configuration Reference*.

CHAPTER 5

DNF API Reference

Contents

5.1 Introduction

The provided Python API to DNF is supposed to mainly allow writing the following two categories of programs:

1. *plugins* to DNF which extend functionality of the system's DNF installation.
2. extension applications that embed DNF (by importing its Python modules) to perform specific package management tasks.

Please refer to the *DNF Use Cases* where you can find examples of API usage.

Note: The API consists of exactly those elements described in this document, items not documented here can change release to release. Opening a **'bugzilla'_** if certain needed functionality is not exposed is the right thing to do.

5.2 Versioning

DNF follows the Semantic Versioning as defined at http://semver.org/.

This basically means that if your piece of software depends on e.g. DNF 1.1, the requirement can be specified as `1.1 <= dnf < 2`. In other words, you can be sure that your software will be API-compatible with any later release of DNF until the next major version is issued. The same applies for the CLI compatibility.

Incompatible API changes are subject to our deprecation policy. Deprecated API items (classes, methods, etc.) are designated as such in the *DNF Release Notes*. The first release where support for such items can be dropped entirely must have, relative to the deprecating release, a higher major version number. DNF will log a warning when a deprecated item is used.

5.3 Contents

API Documentation Contents

5.3.1 Common Provisions of the DNF API

Logging

DNF uses the standard Python logging module to do its logging. Three standard loggers are provided:

- `dnf`, used by the core and CLI components of DNF. Messages logged via this logger can end up written to the stdout (console) the DNF process is attached too. For this reason messages logged on the `INFO` level or above should be marked for localization (if the extension uses it).
- `dnf.plugin` should be used by plugins for debugging and similar messages that are generally not written to the standard output streams but logged into the DNF logfile.
- `dnf.rpm` is a logger used by RPM transaction callbacks. Plugins and extensions should not manipulate this logger.

Extensions and plugins can add or remove logging handlers of these loggers at their own discretion.

5.3.2 `Base`—The centerpiece of DNF

class dnf.**Base**

Instances of *dnf.Base* are the central point of functionality supplied by DNF. An application will typically create a single instance of this class which it will keep for the runtime needed to accomplish its packaging tasks. Plugins are managed by DNF and get a reference to *dnf.Base* object when they run.

Base instances are stateful objects holding references to various data sources and data sinks. To properly finalize and close off any handles the object may hold, client code should either call *Base.close()* when it has finished operations with the instance, or use the instance as a context manager. After the object has left the context, or its *Base.close()* has been called explicitly, it must not be used. *Base.close()* will delete all downloaded packages upon successful transaction.

comps

Is `None` by default. Explicit load via *read_comps()* initializes this attribute to a *dnf.comps.Comps* instance.

conf

An instance of *dnf.conf.Conf*, concentrates all the different configuration options. *__init__()* initializes this to usable defaults.

goal

An instance of `dnf.goal.Goal` that this *Base* object is using.

repos
A *dnf.repodict.RepoDict* instance, this member object contains all the repositories available.

sack
The *Sack* that this *Base* object is using. It needs to be explicitly initialized by *fill_sack()*.

transaction
A resolved transaction object, a `dnf.transaction.Transaction` instance, or `None` if no transaction has been prepared yet.

__init__()
Init an instance with a reasonable default configuration. The constructor takes no arguments.

add_remote_rpms(*path_list*, *strict=True*, *progress=None*)
This function must be called before anything is added to the *goal*. Adds RPM files in path_list to the *sack* and return the list of respective *dnf.package.Package* instances. Downloads the RPMs to a temporary file for each path if it is a remote URL. Raises `IOError` if there are *IO* problems with files and *strict=True*. Raises *dnf.exceptions.Error* if the *goal* is not empty. *progress*, if given, should be a *DownloadProgress* instance which can be used to monitor the progress of the download.

close()
Close all external handles the object holds. This is called automatically via context manager mechanism if the instance is handled using the `with` statement.

init_plugins([*disabled_glob=None*, *cli=None*])
Initialize plugins. If you want to disable some plugins pass the list of their name patterns to *disabled_glob*. When run from interactive script then also pass your *dnf.cli.Cli* instance.

pre_configure_plugins()
Configure plugins by running their pre_configure() method. It makes possible to change variables before repo files and rpmDB are loaded. It also makes possible to create internal repositories that will be affected by `--disablerepo` and `--enablerepo`.

configure_plugins()
Configure plugins by running their configure() method.

fill_sack([*load_system_repo=True*, *load_available_repos=True*])
Setup the package sack. If *load_system_repo* is `True`, load information about packages in the local RPMDB into the sack. Else no package is considered installed during dependency solving. If *load_available_repos* is `True`, load information about packages from the available repositories into the sack.

This operation will call *load()* for repos as necessary and can take a long time. Adding repositories or changing repositories' configuration does not affect the information within the sack until *fill_sack()* has been called.

Before this method is invoked, the client application should setup any explicit configuration relevant to the operation. This will often be at least `conf.cachedir` and the substitutions used in repository URLs. See *Conf.substitutions*.

Throws *IOError* exception in case cached metadata could not be opened.

Example:

```
#!/usr/bin/python3
import dnf

base = dnf.Base()
conf = base.conf
conf.cachedir = '/tmp/my_cache_dir'
```

(continues on next page)

(continued from previous page)

```
conf.substitutions['releasever'] = '30'
conf.substitutions['basearch'] = 'x86_64'

base.repos.add_new_repo('my-repo', conf,
    baseurl=["http://download.fedoraproject.org/pub/fedora/linux/releases/
↪$releasever/Everything/$basearch/os/"])
base.fill_sack()

print("Enabled repositories:")
for repo in base.repos.iter_enabled():
    print("id: {}".format(repo.id))
    print("baseurl: {}".format(repo.baseurl))
```

do_transaction([*display*])
Perform the resolved transaction. Use the optional *display* object(s) to report the progress. *display* can be either an instance of a subclass of *dnf.callback.TransactionProgress* or a sequence of such instances. Raise *dnf.exceptions.Error* or dnf.exceptions.TransactionCheckError.

download_packages(*pkglist*, *progress=None*, *callback_total=None*)
Download packages in *pkglist* from remote repositories. Packages from local repositories or from the command line are not downloaded. *progress*, if given, should be a *DownloadProgress* and can be used by the caller to monitor the progress of the download. *callback_total* is a function accepting two parameters: total size of the downloaded content in bytes and time when the download process started, in seconds since the epoch. Raises *DownloadError* if some packages failed to download.

group_install(*group_id*, *pkg_types*, *exclude=None*, *strict=True*)
Mark group with corresponding *group_id* installed and mark the packages in the group for installation. Return the number of packages that the operation has marked for installation. *pkg_types* is a sequence of strings determining the kinds of packages to be installed, where the respective groups can be selected by including `"mandatory"`, `"default"` or `"optional"` in it. If *exclude* is given, it has to be an iterable of package name glob patterns: *group_install()* will then not mark the respective packages for installation whenever possible. Parameter *strict* is a boolean indicating whether group packages that exist but are non-installable due to e.g. dependency issues should be skipped (False) or cause transaction to fail to resolve (True).

group_remove(*group_id*)
Mark group with corresponding *group_id* not installed. All the packages marked as belonging to this group will be marked for removal. Return the number of packages marked for removal in this call.

group_upgrade(*group_id*)
Upgrade group with corresponding *group_id*. If there has been packages added to the group's comps information since installing on the system, they will be marked for installation. Similarly, removed packages get marked for removal. The remaining packages in the group are marked for an upgrade. The operation respects the package types from the original installation of the group.

environment_install(*env_id*, *types*, *exclude=None*, *strict=True*, *exclude_groups=None*)
Similar to *group_install()* but operates on environmental groups. *exclude_groups* is an iterable of group IDs that will not be marked as installed.

environment_remove(*env_id*)
Similar to *group_remove()* but operates on environmental groups.

environment_upgrade(*env_id*)
Similar to *group_upgrade()* but operates on environmental groups.

read_all_repos()
Read repository configuration from the main configuration file specified by `dnf.conf.Conf.`

`config_file_path` and any `.repo` files under `dnf.conf.Conf.reposdir`. All the repositories found this way are added to *repos*.

read_comps (*arch_filter=False*)
Read comps data from all the enabled repositories and initialize the *comps* object. If *arch_filter* is set to `True`, the result is limited to system basearch.

reset (***kwargs*)
Reset the state of different *Base* attributes. Selecting attributes to reset is controlled by passing the method keyword arguments set to `True`. When called with no arguments the method has no effect.

argument passed	effect
goal=True	drop all the current *packaging requests*
repos=True	drop the current repositories (see *repos*). This won't affect the package data already loaded into the *sack*.
sack=True	drop the current sack (see *sack*)

resolve (*allow_erasing=False*)
Resolve the marked requirements and store the resulting `dnf.transaction.Transaction` into *transaction*. Raise *dnf.exceptions.DepsolveError* on a depsolving error. Return `True` if the resolved transaction is non-empty.

Enabling *allow_erasing* lets the solver remove other packages while looking to fulfill the current packaging requests. For instance, this is used to allow the solver to remove dependants of a package being removed.

The exact operation of the solver further depends on the `dnf.conf.Conf.best` setting.

update_cache (*timer=False*)
Downloads and caches in binary format metadata for all known repos. Tries to avoid downloading whenever possible (e.g. when the local metadata hasn't expired yet or when the metadata timestamp hasn't changed).

If 'timer' equals 'True', DNF becomes more resource-aware, meaning DNF will not do anything if running on battery power and will terminate immediately if it's too soon after the last successful update_cache operation.

When the method is used after *fill_sack()*, information about packages will not be updated.

The *Base* class provides a number of methods to make packaging requests that can later be resolved and turned into a transaction. The *pkg_spec* argument some of them take must be a package specification recognized by *dnf.subject.Subject*. If these methods fail to find suitable packages for the operation they raise a *MarkingError*. Note that successful completion of these methods does not necessarily imply that the desired transaction can be carried out (e.g. for dependency reasons).

downgrade (*pkg_spec*)
Mark packages matching *pkg_spec* for downgrade.

install (*pkg_spec*, *reponame=None*, *strict=True*, *forms=None*)
Mark packages matching *pkg_spec* for installation. *reponame* can be a name of a repository or a list of repository names. If given, the selection of available packages is limited to packages from these repositories. If strict is set to False, the installation ignores packages with dependency solving problems. Parameter *forms* has the same meaning as in *dnf.subject.Subject.get_best_query()*.

package_downgrade (*pkg*, *strict=False*)
If *pkg* is a *dnf.package.Package* in an available repository, mark the matching installed package for downgrade to *pkg*. If strict=False it ignores problems with dep-solving.

package_install(*pkg*, *strict=True*)
Mark *pkg* (a *dnf.package.Package* instance) for installation. Ignores package that is already installed. *strict* has the same meaning as in *install()*.

package_upgrade(*pkg*)
If *pkg* is a *dnf.package.Package* in an available repository, mark the matching installed package for upgrade to *pkg*.

autoremove()
Removes all 'leaf' packages from the system that were originally installed as dependencies of user-installed packages but which are no longer required by any such package.

remove(*pkg_spec*, *reponame=None*, *forms=None*)
Mark packages matching *pkg_spec* for removal. *reponame* and *forms* have the same meaning as in *install()*.

upgrade(*pkg_spec*, *reponame=None*)
Mark packages matching *pkg_spec* for upgrade. *reponame* has the same meaning as in *install()*.

upgrade_all(*reponame=None*)
Mark all installed packages for an upgrade. *reponame* has the same meaning as in *install()*.

urlopen(url, repo=None, mode='w+b', **kwargs):
Open the specified absolute *url* and return a file object which respects proxy setting even for non-repo downloads

install_specs(*install*, *exclude=None*, *reponame=None*, *strict=True*, *forms=None*)
Provides unified way to mark packages, groups or modules for installation. The *install* and *exclude* arguments have to be iterables containing specifications of packages (e.g. 'dnf') or groups/modules (e.g. '@core'). Specifications from the *exclude* list will not be marked for installation. The *reponame*, *strict* and *forms* parameters have the same meaning as in *install()*. In case of errors the method raises *dnf.exceptions.MarkingErrors*.

Example to install two groups and a package:

```
#!/usr/bin/python3
import dnf
import dnf.cli.progress

base = dnf.Base()
base.read_all_repos()
base.fill_sack()

base.install_specs(['acpi', '@Web Server', '@core'])
print("Resolving transaction...",)
base.resolve()
print("Downloading packages...")
progress = dnf.cli.progress.MultiFileProgressMeter()
base.download_packages(base.transaction.install_set, progress)
print("Installing...")
base.do_transaction()
```

5.3.3 Exceptions

exception dnf.exceptions.**Error**
Base class for all DNF Errors.

exception dnf.exceptions.**CompsError**
Used for errors of comps groups like trying to work with group which is not available.

exception dnf.exceptions.**DeprecationWarning**
Used to emit deprecation warnings using Python's `warnings.warning()` function.

exception dnf.exceptions.**DepsolveError**
Error during transaction dependency resolving.

exception dnf.exceptions.**DownloadError**
Error during downloading packages from the repositories.

exception dnf.exceptions.**MarkingError**
Error when DNF was unable to find a match for given package / group / module specification.

exception dnf.exceptions.**MarkingErrors**
Categorized errors during processing of the request. The available error categories are `no_match_pkg_specs` for missing packages, `error_pkg_specs` for broken packages, `no_match_group_specs` for missing groups or modules, `error_group_specs` for broken groups or modules and `module_depsolv_errors` for modular dependency problems.

exception dnf.exceptions.**RepoError**
Error when loading repositories.

5.3.4 Configuration

Configurable settings of the *dnf.Base* object are stored into a *dnf.conf.Conf* instance. The various options are described here.

class dnf.conf.**Conf**

This object has attributes corresponding to all configuration options from both *"[main] Options"* and *"Options for both [main] and Repo"* sections. For example setting a proxy to access all repositories:

```
import dnf

base = dnf.Base()
conf = base.conf
conf.proxy = "http://the.proxy.url:3128"
conf.proxy_username = "username"
conf.proxy_password = "secret"
base.read_all_repos()
base.fill_sack()
```

get_reposdir
Returns the value of the first valid reposdir or if unavailable the value of created reposdir (string)

substitutions
A mapping of substitutions used in repositories' remote URL configuration. The commonly used ones are:

key	meaning	default
arch	architecture of the machine	autodetected
basearch	the architecture family of the current "arch"	autodetected
releasever	release name of the system distribution	`None`

dnf.rpm.detect_releasever() can be used to detect the `releasever` value.

Following example shows recommended method how to override autodetected architectures:

```
import dnf
import hawkey

arch = hawkey.detect_arch()
base = dnf.Base()
base.conf.substitutions['arch'] = arch
base.conf.substitutions['basearch'] = dnf.rpm.basearch(arch)
base.fill_sack()
...
```

exclude_pkgs (*pkgs*)
Exclude all packages in the *pkgs* list from all operations.

prepend_installroot (*option*)
Prefix config option named *option* with `installroot`.

read (*filename=None*)
Read configuration options from the `main` section in *filename*. Option values not present there are left at their current values. If *filename* is `None`, `config_file_path` is used. Conversely, the configuration path used to load the configuration file that was used is stored into `config_file_path` before the function returns.

dump()
Print configuration values, including inherited values.

write_raw_configfile (*filename*, *section_id*, *substitutions*, *modify*)
Update or create config file. Where *filename* represents name of config file (.conf or .repo); *section_id* represents id of modified section (e.g. main, fedora, updates); *substitutions* represents an instance of base.conf.substitutions; *modify* represents dict of modified options.

5.3.5 Repository Configuration

class dnf.repodict.**RepoDict**
Dictionary mapping repository IDs to the respective *dnf.repo.Repo* objects. Derived from the standard `dict`.

add (*repo*)
Add a *Repo* to the repodict.

add_new_repo (*repoid*, *conf*, *baseurl=()*, ***kwargs*)
Initialize new *Repo* object and add it to the repodict. It requires `repoid` (string), and *dnf.conf.Conf* object. Optionally it can be specified baseurl (list), and additionally key/value pairs from *kwargs* to set additional attribute of the *Repo* object. Variables in provided values (`baseurl` or `kwargs`) will be automatically substituted using conf.substitutions (like `$releasever`, ...). It returns the *Repo* object.

all()
Return a list of all contained repositories.

See the note at *get_matching()* for special semantics of the returned object.

enable_debug_repos()
Enable debug repos corresponding to already enabled binary repos.

enable_source_repos()
Enable source repos corresponding to already enabled binary repos.

get_matching (*key*)
Return a list of repositories which ID matches (possibly globbed) *key* or an empty list if no matching repository is found.

The returned list acts as a composite, transparently forwarding all method calls on itself to the contained repositories. The following thus disables all matching repos:

```
#!/usr/bin/python3
import dnf

base = dnf.Base()
base.read_all_repos()
base.fill_sack()

repos = base.repos.get_matching('*-debuginfo')
repos.disable()
```

iter_enabled()
Return an iterator over all enabled repos from the dict.

dnf.repo.**repo_id_invalid**(*repo_id*)
Return index of the first invalid character in the *repo_id* or `None` if all characters are valid. This function is used to validate the section names in `.repo` files.

class dnf.repo.**Metadata**
Represents the metadata files.

fresh
Boolean. `True` if the metadata was loaded from the origin, `False` if it was loaded from the cache.

class dnf.repo.**Repo**
Repository object used for metadata download. To configure it properly one has to give it either `metalink`, `mirrorlist` or `baseurl` parameter. This object has attributes corresponding to all configuration options from both *"Repo Options"* and *"Options for both [main] and Repo"* sections.

Important: Some *Repo* attributes have non-native Python types. Duck typing works (objects have identical behavior), but `isinstance()` and `type()` doesn't work as expected because of different types. For example *excludepkgs* and *includepkgs* return a `VectorString`, which is s SWIG wrapper on top of underlying libdnf C++ code.

---

id
ID of this repo. This attribute is read-only.

metadata
If *load()* has been called and succeeded, this contains the relevant *Metadata* instance.

pkgdir
Directory where packages of a remote repo will be downloaded to. By default it is derived from *cachedir* in *__init__()* but can be overridden by assigning to this attribute.

repofile
The path to configuration file of the class.

__init__(*name=None*, *parent_conf=None*)
Init repository with ID *name* and the *parent_conf* which an instance of *dnf.conf.Conf* holding main dnf configuration. Repository ID must be a string that can contain ASCII letters, digits, and -_.: characters.

add_metadata_type_to_download(*metadata_type*)
Ask for additional repository metadata type to download. Given *metadata_type* is appended to the default metadata set when repository is downloaded.

disable()
Disable the repository. Repositories are enabled by default.

dump()
Print repository configuration, including inherited values.

enable()
Enable the repository (the default).

get_http_headers()
Return user defined http headers. Return tuple of strings.

get_metadata_content(*metadata_type*)
Return contents of the repository's metadata file of the given metadata type. Contents of compressed files are returned uncompressed.

get_metadata_path(*metadata_type*)
Return path to the file with downloaded repository metadata of given type.

load()
Load the metadata of this repository. Will try to use local cache if possible and initiate and finish download if not. Returns `True` if fresh metadata has been downloaded and `False` if cache was used. Raises *dnf.exceptions.RepoError* if the repo metadata could not be obtained.

set_http_headers(*headers*)
Set new user headers and rewrite existing ones. *headers* must be an instance of tuple of strings or list of strings.

set_progress_bar(*progress*)
Set the download progress reporting object for this repo during *load()*. *progress* must be an instance of *dnf.callback.DownloadProgress*.

5.3.6 Sack

class dnf.sack.**Sack**
The package sack. Contains metadata information about all known packages, installed and available.

query()
Return a *Query* for querying packages contained in this sack.

dnf.sack.**rpmdb_sack**(*base*)
Returns a new instance of sack containing only installed packages (@System repo). Useful to get list of the installed RPMs after transaction.

5.3.7 Queries and Subjects

class dnf.query.**Query**
Facilitates lookup of packages in a *Sack* based on given criteria. Query actually does not consult the information in the `Sack` until it is evaluated. The evaluation happens either explicitly using *run()* or by iterating the query, for example:

```
#!/usr/bin/python3
import dnf

base = dnf.Base()
base.fill_sack()

q = base.sack.query()
i = q.installed()
i = i.filter(name='dnf')
```

(continues on next page)

(continued from previous page)

```
packages = list(i)  # i only gets evaluated here
print("Installed dnf package:")
for pkg in packages:
    print(pkg, pkg.reponame)
```

or:

```
#!/usr/bin/python3
import dnf

base = dnf.Base()
base.read_all_repos()
base.fill_sack()

q = base.sack.query()
a = q.available()
a = a.filter(name='dnf')

print("Available dnf packages:")
for pkg in a:  # a only gets evaluated here
    print('{} in repo {}'.format(pkg, pkg.reponame))
```

Notice that none of the filtering methods mutates the state of the *Query* but produces a new object instead.

available()
: Returns a new query limiting the original query to the packages available from the repositories.

difference(*other*)
: Returns a new query that contains only those results of original query that are not in the results of the `other` query.

downgrades()
: Returns a new query that limits the result only to packages that can be downgrade candidates to other packages in the current set. Downgrade candidate has the same name, lower EVR and the architecture of the original and the downgrade candidate are suitable for a downgrade. Specifically, the filtering does not take any steps to establish that the downgrade candidate can actually be installed.

duplicated()
: Returns a new query that limits the result only to installed packages of same name and different version. Optional argument exclude accepts a list of package names that will be excluded from result.

extras()
: Returns a new query that limits the result to installed packages that are not present in any repo

filter(***kwargs*)
: Returns a new query limiting the original query to the key/value pairs from *kwargs*. Multiple *kwargs* can be passed, the filter then works by applying all of them together (logical AND). Values inside of list or query are cumulative (logical OR).

  Allowed keys are:

key	value type	value meaning
arch	string	match against packages' architecture
downgrades	boolean	see *downgrades()*. Defaults to `False`.
empty	boolean	`True` limits to empty result set. Defaults to `False`.
epoch	integer	match against packages' epoch.
file	string	match against packages' files
latest	integer	limit to all packages of number of versions
latest_per_arch	integer	see *latest()*.
name	string	match against packages' names
release	string	match against packages' releases
reponame	string	match against packages repositories' names
version	string	match against packages' versions
pkg	Query	match against packages in query
pkg*	list	match against hawkey.Packages in list
provides	string	match against packages' provides
provides*	Hawkey.Reldep	match against packages' provides
<DEP>	string	match against packages' <DEP>
<DEP>*	Hawkey.Reldep	match a reldep against packages' <DEP>
<DEP>*	Query	match the result of a query against packages' <DEP>
<DEP>*	list(Package)	match the list of hawkey.Packages against packages' <DEP>
sourcerpm	string	match against packages' source rpm
upgrades	boolean	see *upgrades()*. Defaults to `False`.

`<DEP>` can be any of: requires, conflicts, obsoletes, enhances, recomments, suggests, supplements

* The key can also accept a list of values with specified type.

The key name can be supplemented with a relation-specifying suffix, separated by __:

key suffix	value type	semantics
eq	any	exact match; This is the default if no suffix is specified.
glob	string	shell-style wildcard match
gt	integer	the actual value is greater than specified
gte	integer	the actual value is greater than or equal to specified
lt	integer	the actual value is less than specified
lte	integer	the actual value is less than or equal to specified
neq	any	does not equal
substr	string	the specified value is contained in the actual value

For example, the following creates a query that matches all packages containing the string "club" in its name:

```
q = base.sack.query().filter(name__substr="club")
```

Note that using packages or queries for dependency filtering performs a more advanced resolution than using a string or a reldep. When a package list or a query is used, rich dependencies are resolved in a more precise way than what is possible when a string or a reldep is used.

filterm(***kwargs*)
Similar to *dnf.query.Query.filter()* but it modifies the query in place.

installed()
Returns a new query that limits the result to the installed packages only.

intersection(*other*)
Returns a new query where the result contains only packages that are found in both original and `other` queries.

latest(*limit=1*)
Returns a new query that limits the result to `limit` highest version of packages per package name and per architecture. In case the limit is negative number, it excludes the number of latest versions according to limit.

run()
Evaluate the query. Returns a list of matching *dnf.package.Package* instances.

union(*other*)
Returns a new query where the results of the `other` query are added to the results of the original query.

upgrades()
Returns a new query that limits the result only to packages that can be upgrade candidates to at least one package in the current set. Upgrade candidate has the same name, higher EVR and the architectures of the original and the upgrade candidate package are suitable for an upgrade. Specifically, the filtering does not take any steps to establish that the upgrade candidate can actually be installed.

class dnf.subject.**Subject**
As *explained on the DNF man page*, users of the CLI are able to select packages for an operation in different formats, leaving seemingly arbitrary parts out of the spec and even using globbing characters. This class implements a common approach to parsing such input and produce a *Query* listing all packages matching the input or a *Selector* selecting a single package that best matches the input given a transaction operation.

__init__(*pkg_spec*, *ignore_case=False*)
Initialize the *Subject* with *pkg_spec* input string with following *semantic*. If *ignore_case* is `True` ignore the case of characters in *pkg_spec*.

get_best_query(*sack*, *with_nevra=True*, *with_provides=True*, *with_filenames=True*, *forms=None*)
Returns a `Query` yielding packages matching the given input. The result of the returned query can be an empty set if no package matches. *sack* is the *Sack* that the returned query will search. *with_nevra* enable search by nevra, *with_provides* indicates whether besides package names also packages' provides are searched for a match, and *with_filenames* indicates whether besides package provides also packages' file provides are searched for a match. *forms* is a list of pattern forms from **'hawkey'_**. Leaving the parameter to `None` results in using a reasonable default list of forms.

get_best_selector(*sack*, *forms=None*, *obsoletes=True*, *reponame=None*, *reports=False*)
Returns a *Selector* that will select a single best-matching package when used in a transaction operation. *sack* and *forms* have the same meaning as in *get_best_query()*. If `obsoletes`, selector will also contain packages that obsoletes requested packages (default is True). If `reponame`, the selection of available packages is limited to packages from that repo (default is None). Attribute `reports` is deprecated and not used any more. Will be removed on 2018-01-01.

get_nevra_possibilities(*self*, *forms=None*)
Returns generator for every possible nevra. Each possible nevra is represented by NEVRA class (libdnf) that has attributes name, epoch, version, release, arch. *forms* have the same meaning as in *get_best_query()*.

Example how to use it when it is known that string could be full NEVRA or NEVR:

```
#!/usr/bin/python3
import dnf
import hawkey

nevra_string = "dnf-0:4.2.2-2.fc30.noarch"
subject = dnf.subject.Subject(nevra_string)
```

(continues on next page)

(continued from previous page)

```
possible_nevra = subject.get_nevra_possibilities(
    forms=[hawkey.FORM_NEVRA, hawkey.FORM_NEVR])

for i,nevra in enumerate(possible_nevra):
    print("Possibility {} for \"{}\":".format(i+1, nevra_string))
    print("name: {}".format(nevra.name))
    print("epoch: {}".format(nevra.epoch))
    print("version: {}".format(nevra.version))
    print("release: {}".format(nevra.release))
    print("architecture: {}".format(nevra.arch))
    print()
```

5.3.8 Selector

class dnf.selector.**Selector**

Specify a target of a transaction operation.

set()

Set content of Selector similarly like *dnf.query.Query.filter()*

matches()

Returns packages that represents the content of Selector

5.3.9 Package

class dnf.package.**Package**

Represents a unit of software management, typically corresponds to an RPM file.

arch

Architecture of the package (string).

baseurl

Baseurl of the package (string).

buildtime

Seconds since the epoch when the package was built (integer).

chksum

Tuple with package checksum and checksum type or `None`. The checksum is returned only for packages from repository. The checksum is not returned for installed package or packages from commandline repository. The checksum represents @pkgid value which links primary metadata with other repository metadata files.

conflicts

Packages that the package conflicts with (list of Hawkey.Reldep).

debug_name

The name of the debug-info package (string).

description

The description of the package (string).

downloadsize

The size of rpm package in bytes (integer).

epoch

Epoch of the package (integer).

enhances
Packages that the package enhances (list of Hawkey.Reldep).

evr
EVR (epoch:version-revision) of the package (string).

files
Files the package provides (list of strings).

group
Group of the package (string).

hdr_chksum
Tuple with package header checksum and checksum type or `None`. The checksum is returned only for installed packages.

hdr_end
Header end index for the package. Returns 0 for not known (integer).

changelogs
Changelogs for the package (list of dictionaries with "timestamp", "author" and "text" keys).

installed
Returns `True` if the package is installed (boolean).

installtime
Seconds since the epoch when the package was installed (integer).

installsize
Space in bytes the package takes on the system after installation (integer).

license
License of the package (string).

medianr
Media number for the package (integer).

name
The name of the package (string).

obsoletes
Packages that are obsoleted by the package (list of Hawkey.Reldep).

provides
Package's provides (list of Hawkey.Reldep).

recommends
Packages that are recommended by the package (list of Hawkey.Reldep).

release
Release of the package (string).

reponame
Id of repository the package was installed from (string).

requires
Package's requirements (list of Hawkey.Reldep).

requires_pre
Package's install-time requirements (list of Hawkey.Reldep).

rpmdbid
The rpmdb ID for the package (integer).

source_debug_name
The name of the source debug-info package (string).

source_name
The name of the source package (string).

sourcerpm
Full name of the SRPM used to build this package (string).

suggests
Packages that are suggested by the package (list of Hawkey.Reldep).

summary
Summary for the package (string).

supplements
Packages that the package supplements (list of Hawkey.Reldep).

url
URL for the package (string).

version
Version of the package (string).

remote_location (*schemes=('http', 'ftp', 'file', 'https')*)
The location from where the package can be downloaded from (string). If the information is unavailable it returns `None`. `schemes` limits result to list of protocols.

5.3.10 Transaction

class dnf.db.group.**RPMTransaction**
Instances of this class describe a resolved transaction set. The transaction object can be iterated for the contained `items`.

The packaging requests from the contained items are later passed to the core package manager (RPM) as they are without further dependency resolving. If the set is not fit for an actual transaction (e.g. introduces conflicts, has inconsistent dependencies) RPM then by default refuses to proceed.

install_set
Read-only property which contains set of *Packages* to be installed.

remove_set
Read-only property which contains set of *Packages* to be removed.

5.3.11 Comps, or the Distribution Compose Metadata

class dnf.comps.**Comps**
An object of this class can merge comps information from arbitrary repositories. It typically is instantiated from *dnf.Base* and covers all the available repositories.

The `*_by_pattern` methods all take a *pattern* and an optional *case_sensitive* parameter. The pattern is matched against names and IDs of objects in the domain (groups, categories, environments), the globbing characters in *pattern* retain their usual expanding meaning. If *case_sensitive* is `True`, matching is done in a case-sensitive manner.

categories
List of all contained *dnf.comps.Category* objects.

environments
List of all contained *dnf.comps.Environment* objects ordered by *display_order* tag defined in comps.xml file.

groups
List of all contained *dnf.comps.Group* objects ordered by *display_order* tag defined in comps.xml file.

category_by_pattern(*pattern*, *case_sensitive=False*)
Returns a *dnf.comps.Category* object matching *pattern*, or `None`.

categories_by_pattern(*pattern*, *case_sensitive=False*)
Return an iterable of *dnf.comps.Category* objects matching *pattern*.

categories_iter()
Return iterator over all contained *dnf.comps.Category* objects.

environment_by_pattern(*pattern*, *case_sensitive=False*)
Return a *dnf.comps.Environment* object matching *pattern*, or `None`.

environments_by_pattern(*pattern*, *case_sensitive=False*)
Return an iterable of *dnf.comps.Environment* objects matching *pattern* ordered by *display_order* tag defined in comps.xml file.

environments_iter
Return iterator over all contained *dnf.comps.Environment* objects in order they appear in comps.xml file.

group_by_pattern(*pattern*, *case_sensitive=False*)
Return a *dnf.comps.Group* object matching *pattern*, or `None`.

groups_by_pattern(*pattern*, *case_sensitive=False*)
Return an iterable of *dnf.comps.Group* objects matching *pattern* ordered by *display_order* tag defined in comps.xml file.

groups_iter
Return iterator over all contained *dnf.comps.Group* objects in order they appear in comps.xml file.

class dnf.comps.**Package**
Represents comps package data.

Note: Should not be confused with *dnf.package.Package* which represents a package contained in a *Sack*. There is no guarantee whether the comps package has a corresponding real sack package, i.e. there can be no package of given name in the sack, one such package, or more than one. For this reason two separate types are introduced.

---

name
Name of the package.

option_type
The type of inclusion of this particular package in its group. Must be one of the *inclusion types*.

class dnf.comps.**Category**

id
Unique identifier of the category.

name
Name of the category.

ui_name
The name of the category translated to the language given by the current locale.

ui_description
The description of the category translated to the language given by the current locale.

class dnf.comps.**Environment**
Has the same set of attributes as *dnf.comps.Category*.

class dnf.comps.**Group**
Has the same set of attributes as *dnf.comps.Category*.

packages_iter()
Return iterator over all *packages* belonging in this group.

Following types of inclusions of objects in their parent objects are defined:

dnf.comps.**CONDITIONAL**

dnf.comps.**DEFAULT**

dnf.comps.**MANDATORY**

dnf.comps.**OPTIONAL**

5.3.12 Plugin Interface

DNF plugin can be any Python class fulfilling the following criteria:

1. it derives from *dnf.Plugin*,
2. it is made available in a Python module stored in one of the Conf.pluginpath,
3. provides its own *name* and *__init__()*.

When DNF CLI runs it loads the plugins found in the paths during the CLI's initialization.

class dnf.**Plugin**
The base class all DNF plugins must derive from.

name
The plugin must set this class variable to a string identifying the plugin. The string can only contain alphanumeric characters and underscores.

static read_config(*conf*)
Read plugin's configuration into a ConfigParser compatible instance. *conf* is a *Conf* instance used to look up the plugin configuration directory.

__init__(*base*, *cli*)
The plugin must override this. Called immediately after all the plugins are loaded. *base* is an instance of *dnf.Base*. *cli* is an instance of *dnf.cli.Cli* but can also be None in case DNF is running without a CLI (e.g. from an extension).

pre_config()
This hook is called before configuring the repos.

config()
This hook is called immediately after the CLI/extension is finished configuring DNF. The plugin can use this to tweak the global configuration or the repository configuration.

resolved()
This hook is called immediately after the CLI has finished resolving a transaction. The plugin can use this to inspect the resolved but not yet executed Base.transaction.

sack()
This hook is called immediately after *Base.sack* is initialized with data from all the enabled repos.

pre_transaction()
This hook is called just before transaction execution. This means after a successful transaction test. RPMDB is locked during that time.

transaction()
This hook is called immediately after a successful transaction. Plugins that were removed or obsoleted by the transaction will not run the transaction hook.

register_command(*command_class*)
A class decorator for automatic command registration.

Example of a plugin that provides a hello-world dnf command (the file must be placed in one of the *pluginpath* directories:

```
import dnf

@dnf.plugin.register_command
class HelloWorldCommand(dnf.cli.Command):
    aliases = ('hello-world',)
    summary = 'The example command'

    def run(self):
        print('Hello world!')
```

To run the command:

```
$ dnf hello-world
Hello world!
```

You may want to see the comparison with yum plugin hook API.

5.3.13 Progress Reporting with Callbacks

class dnf.callback.**Payload**
Represents one item (file) from the download batch.

__str__()
Provide concise, human-readable representation of this Payload.

download_size
Total size of this Payload when transferred (e.g. over network).

class dnf.callback.**DownloadProgress**
Base class providing callbacks to receive information about an ongoing download.

start(*total_files*, *total_size*, *total_drpms=0*)
Report start of a download batch. *total_files* is the total number of payloads in the batch. *total_size* is the total number of bytes to be downloaded. *total_drpms* is the total number of drpms payloads in the batch.

progress(*payload*, *done*)
Report ongoing progress on the given *payload*. *done* is the number of bytes already downloaded from *payload*.

end(*payload*, *status*, *msg*)
Report finished download of a *payload*, *Payload* instance. *status* is a constant with the following meaning:

status value	meaning
STATUS_OK	Download finished successfully.
STATUS_DRPM	DRPM rebuilt successfully.
STATUS_ALREADY_EXISTS	Download skipped because the local file already exists.
STATUS_MIRROR	Download failed on the current mirror, will try to use next mirror in the list.
STATUS_FAILED	Download failed because of another error.

msg is an optional string error message further explaining the *status*.

class dnf.callback.**TransactionProgress**

Base class providing callbacks to receive information about an ongoing transaction.

error(*message*)

Report an error that occurred during the transaction. *message* is a string which describes the error.

progress(*package*, *action*, *ti_done*, *ti_total*, *ts_done*, *ts_total*)

Report ongoing progress on the given transaction item. *package* is the dnf.package.Package being processed and *action* is a constant with the following meaning:

action value	meaning	Appearance*
PKG_CLEANUP	*package* cleanup is being performed.	3
PKG_DOWNGRADE	*package* is being installed as a downgrade.	2
PKG_DOWNGRADED	installed *package* is being downgraded.	2
PKG_INSTALL	*package* is being installed.	2
PKG_OBSOLETE	*package* is obsoleting another package.	2
PKG_OBSOLETED	installed *package* is being obsoleted.	2
PKG_REINSTALL	*package* is installed as a reinstall.	2
PKG_REINSTALLED	installed *package* is being reinstalled.	2
PKG_REMOVE	*package* is being removed.	2
PKG_UPGRADE	*package* is installed as an upgrade.	2
PKG_UPGRADED	installed *package* is being upgraded.	2
PKG_VERIFY	*package* is being verified.	5
PKG_SCRIPTLET	*package* scriptlet is being performed.	Anytime
TRANS_PREPARATION	*transaction* is being prepared.	1
TRANS_POST	The post-trans phase started. In this case, all the other arguments are None.	4

*This is order in which state of transaction which callback action can appear. Only PKG_SCRIPTLET can appear anytime during transaction even before transaction starts.

ti_done is the number of processed bytes of the transaction item, *ti_total* is the total number of bytes of the transaction item, *ts_done* is the number of actions processed in the whole transaction and *ts_total* is the total number of actions in the whole transaction.

5.3.14 RPM Interface

dnf.rpm.**detect_releasever**(*installroot*)

Return the release name of the distribution of the tree rooted at *installroot*. The function uses information from RPMDB found under the tree.

Returns None if the information can not be determined (perhaps because the tree has no RPMDB).

dnf.rpm.**basearch**(*arch*)
Return base architecture of the processor based on *arch* type given. E.g. when *arch* i686 is given then the returned value will be i386.

5.3.15 Command Line Interface Hooks

dnf.cli is a part of DNF that contains code handling the command line tasks for DNF, like for instance `dnf install emacs`, and outputs the results to the terminal. It is usually of no interest for DNF extension applications, but some parts of it described here can be used by the *Plugin Interface* to hook up custom commands.

When packaging your custom command, we recommend you to define a virtual provide in the form of `Provides: dnf-command(<alias>)` in the spec file. See *the virtual provides usage* for the details.

exception dnf.cli.**CliError**
Signals a CLI-specific problem (reading configuration, parsing user input, etc.). Derives from *dnf.exceptions.Error*.

class dnf.cli.demand.**DemandSheet**
Instances are used to track requests of commands and plugins about how CLI should set up/handle other parts of CLI processing that are not under the command's/plugin's direct control. The boolean attributes of the sheet can not be reset once explicitly set, doing so raises an `AttributeError`.

allow_erasing
If `True`, the dependency solver is allowed to look for solutions that include removing other packages while looking to fulfill the current packaging requests. Defaults to `False`. Also see *dnf.Base.resolve()*.

available_repos
If `True`, during sack creation (*sack_activation*), download and load into the sack the available repositories. Defaults to `False`.

resolving
If `True`, at a place where the CLI would otherwise successfully exit, resolve the transaction for any outstanding packaging requests before exiting. Defaults to `False`.

root_user
`True` informs the CLI that the command can only succeed if the process's effective user id is `0`, i.e. root. Defaults to `False`.

sack_activation
If `True`, demand that the CLI sets up the *Sack* before the command's *run()* method is executed. Defaults to `False`.

Depending on other demands and the user's configuration, this might or might not correctly trigger metadata download for the available repositories.

load_system_repo
If `True`, DNF will load information about installed packages from the local RPMDB into the sack during *dnf.Base.fill_sack()*. Defaults to `True`.

cacheonly
When `True`, DNF will run entirely from the system cache (equivalent of `-C` command line option). Defaults to `False`.

fresh_metadata
`False` means that (even expired) cached repository metadata will be used. When `True`, the expired repository metadata caches are synchronized with server. Defaults to `True`.

freshest_metadata
If `True`, metadata caches for all enabled repositories are forcibly expired before the sack is activated. Defaults to `False`.

changelogs
If `True`, also the repository metadata containing changelogs for packages will be downloaded. Defaults to `False`.

success_exit_status
The return status of the DNF command on success. Defaults to `0`.

transaction_display
An additional instance of a subclass of *dnf.callback.TransactionProgress* used to report information about an ongoing transaction. Defaults to `None`.

class dnf.cli.**Command**
Base class of every DNF command.

aliases
Sequence of strings naming the command from the command line. Must be a class variable. The list has to contain at least one string, the first string in the list is considered the canonical name. A command name can be contain only letters and dashes providing the name doesn't start with a dash.

base
The *dnf.Base* instance to use with this command.

cli
The *dnf.cli.Cli* instance to use with this command.

summary
One line summary for the command as displayed by the CLI help.

__init__(*cli*)
Command constructor which can be overridden. The constructor is called during CLI configure phase when one of the command's aliases is parsed from *dnf* commandline. *cli* is an instance of *dnf.cli.Cli*.

pre_configure()
Perform any pre-configuration on the command itself and on the CLI. Typically, the command implements this call to set up releasever or enable/disable repository. This method is called before configuration of repos.

configure()
Perform any configuration on the command itself and on the CLI. Typically, the command implements this call to set up any *demands*, tweak the global configuration or the repository configuration. This method is called immediately after the CLI/extension is finished configuring DNF.

run()
Run the command. This method is invoked by the CLI when this command is executed. Should raise *dnf.exceptions.Error* with a proper message if the command fails. Otherwise should return `None`. Custom commands typically override this method and put their main work code here.

class dnf.cli.**Cli**
Manages the CLI, including reading configuration, parsing the command line and running commands.

demands
An instance of *DemandSheet*, exposed to allow custom commands and plugins influence how the CLI will operate.

register_command(command_cls):
Register new command. *command_cls* is a subclass of *Command*.

redirect_logger(self, stdout=None, stderr=None):
Change minimal logger level for terminal output to stdout and stderr according to specific command requirements. For stdout and stderr use logging.INFO, logging.WARNING, etc.

5.3.16 Modularity Interface

class dnf.module.module_base.**ModuleBase**

Basic class for handling modules.

dnf.module.module_base.**__init__**(*base*)
Initialize *dnf.module.module_base.ModuleBase* object. *base* is an instance of the *dnf.Base* class.

dnf.module.module_base.**enable**(*module_specs*)
Mark module streams matching the *module_specs* list and also all required modular dependencies for enabling. For specs that do not specify the stream, the default stream is used. In case that the module has only one stream available, this stream is used regardles of whether it is the default or not. Note that only one stream of any given module can be enabled on a system. The method raises *dnf.exceptions.MarkingErrors* in case of errors.

Example:

```
#!/usr/bin/python3
import dnf

base = dnf.Base()
base.read_all_repos()
base.fill_sack()

module_base = dnf.module.module_base.ModuleBase(base)
module_base.enable(['nodejs:11'])

base.do_transaction()
```

dnf.module.module_base.**disable**(*module_specs*)
Mark modules matching the *module_specs* list for disabling. Only the name part of the module specification is relevant. Stream, version, context, arch and profile parts are ignored (if given). All streams of the module will be disabled and all installed profiles will be removed. Packages previously installed from these modules will remain installed on the system. The method raises *dnf.exceptions.MarkingErrors* in case of errors.

Example:

```
#!/usr/bin/python3
import dnf

base = dnf.Base()
base.read_all_repos()
base.fill_sack()

module_base = dnf.module.module_base.ModuleBase(base)
module_base.disable(['nodejs'])

base.do_transaction()
```

dnf.module.module_base.**reset**(*module_specs*)
Mark module for resetting so that it will no longer be enabled or disabled. All installed pro-

files of streams that have been reset will be removed. The method raises *dnf.exceptions.MarkingErrors* in case of errors.

dnf.module.module_base.**install**(*module_specs*, *strict=True*)
Mark module profiles matching *module_specs* for installation and enable all required streams. If the stream or profile part of specification is not specified, the defaults are chosen. All packages of installed profiles are also marked for installation. If *strict* is set to `False`, the installation skips modules with dependency solving problems. The method raises *dnf.exceptions.MarkingErrors* in case of errors.

Example:

```
#!/usr/bin/python3
import dnf

base = dnf.Base()
base.read_all_repos()
base.fill_sack()

module_base = dnf.module.module_base.ModuleBase(base)
module_base.install(['nodejs:11/minimal'])

base.resolve()
base.download_packages(base.transaction.install_set)
base.do_transaction()
```

dnf.module.module_base.**remove**(*module_specs*)
Mark module profiles matching *module_spec* for removal. All packages installed from removed profiles (unless they are required by other profiles or user-installed packages) are also marked for removal.

dnf.module.module_base.**upgrade**(*module_specs*)
Mark packages of module streams (or profiles) matching *module_spec* for upgrade.

dnf.module.module_base.**get_modules**(*module_spec*)
Get information about modules matching *module_spec*. Returns tuple (module_packages, nsvcap), where *nsvcap* is a hawkey.NSVCAP object parsed from *module_spec* and *module_packages* is a tuple of *libdnf.module.ModulePackage* objects matching this *nsvcap*.

Example:

```
#!/usr/bin/python3
import dnf

base = dnf.Base()
base.read_all_repos()
base.fill_sack()

module_base = dnf.module.module_base.ModuleBase(base)
module_packages, nsvcap = module_base.get_modules('nodejs:11/minimal')

print("Parsed NSVCAP:")
print("name:", nsvcap.name)
print("stream:", nsvcap.stream)
print("version:", nsvcap.version)
print("context:", nsvcap.context)
print("arch:", nsvcap.arch)
print("profile:", nsvcap.profile)
```

(continues on next page)

(continued from previous page)

```
print("Matching modules:")
for mpkg in module_packages:
    print(mpkg.getFullIdentifier())
```

class libdnf.module.**ModulePackage**

This class represents a record identified by NSVCA from the repository modular metadata. See also https://github.com/fedora-modularity/libmodulemd/blob/master/spec.v2.yaml.

dnf.module.module_base.**getName**()
Return the name of the module.

dnf.module.module_base.**getStream**()
Return the stream of the module.

dnf.module.module_base.**getVersion**()
Return the version of the module as a string.

dnf.module.module_base.**getVersionNum**()
Return the version of the module as a number.

dnf.module.module_base.**getContext**()
Return the context of the module.

dnf.module.module_base.**getArch**()
Return the architecture of the module.

dnf.module.module_base.**getNameStream**()
Return string in the form of 'name:stream' for the module.

dnf.module.module_base.**getNameStreamVersion**()
Return string in the form of 'name:stream:version' for the module.

dnf.module.module_base.**getFullIdentifier**()
Return string in the form of 'name:stream:version:context:architecture' for the module.

dnf.module.module_base.**getProfiles**(*name=None*)
Return tuple of *libdnf.module.ModuleProfile* instancies representing each of the individual profiles of the module. If the *name* is given, only profiles matching the *name* pattern are returned.

dnf.module.module_base.**getSummary**()
Return the summary of the module.

dnf.module.module_base.**getDescription**()
Return the description of the module.

dnf.module.module_base.**getRepoID**()
Return the identifier of source repository of the module.

dnf.module.module_base.**getArtifacts**()
Return tuple of the artifacts of the module.

dnf.module.module_base.**getModuleDependencies**()
Return tuple of *libdnf.module.ModuleDependencies* objects representing modular dependencies of the module.

dnf.module.module_base.**getYaml**()
Return repomd yaml representing the module.

class libdnf.module.**ModuleProfile**

getName()
Return the name of the profile.

getDescription()
Return the description of the profile.

getContent()
Return tuple of package names to be installed with this profile.

class libdnf.module.**ModuleDependencies**

getRequires()
Return tuple of MapStringVectorString objects. These objects behave like standard python dictionaries and represent individual dependencies of the given module. Keys are names of required modules, values are tuples of required streams specifications.

class libdnf.module.**ModulePackageContainer**
This class is under development and should be considered unstable at the moment.

Indices:

- genindex

CHAPTER 6

DNF User's FAQ

Contents

6.1 General Questions

6.1.1 What does DNF stand for?

Dandified YUM.

6.1.2 Can I have DNF and YUM installed side by side?

Yes, you can. And this setup is tested by many.

There is one restriction: DNF and YUM keep additional data about each installed package and every performed transaction. This data is currently not shared between the two managers so if the admin installs half of the packages with DNF and the other half with YUM then each program can not benefit from the information held by the other one. The practical bottom line is that commands like `autoremove` can not take a completely informed decision and thus have to "play it safe" and remove only a subset of dependencies they would be able to otherwise. Similar situation exists with groups.

To transfer transaction additional data from yum to DNF, run:

```
dnf install python-dnf-plugins-extras-migrate && dnf-2 migrate
```

6.1.3 Is there a compatibility layer for YUM?

For the CLI, yes. Just install `dnf-yum` which supplies our own `/usr/bin/yum`. Note two things: all the *differences* between the two package managers still apply and this does not provide "yum" in terms of package dependencies (it conflicts with the YUM package though).

6.1.4 What to do with packages that DNF refuses to remove because their `%pre` or `%preun` scripts are failing?

If this happens, it is a packaging error and consider reporting the failure to the package's maintainer.

You can usually remove such package with `rpm`:

```
rpm -e <package-version> --noscripts
```

6.1.5 Why are `dnf check-update` packages not marked for upgrade in the following `dnf upgrade`

Sometimes one can see that a newer version of a package is available in the repos:

```
$ dnf check-update
libocsync0.x86_64 0.91.4-2.1              devel_repo
owncloud-client.x86_64 1.5.0-18.1         devel_repo
```

Yet the immediately following `dnf upgrade` does not offer them for upgrade:

```
$ dnf upgrade
Resolving dependencies
--> Starting dependency resolution
```

(continues on next page)

(continued from previous page)

```
--> Finished dependency resolution
Dependencies resolved.
Nothing to do.
```

It might seem odd but in fact this can happen quite easily: what the first command does is only check whether there are some available packages with the same name as an installed package but with a higher version. Those are considered upgrade candidates by `check-update`, but no actual dependency resolving takes place there. That only happens during `dnf upgrade` and if the resolving procedure then discovers that some of the packages do not have their dependencies ready yet, then they are not offered in the upgrade. To see the precise reason why it was not possible to do the upgrade in this case, use:

```
$ dnf upgrade --best
```

In DNF version 1.1 and above, you can see the skipped packages in the special transaction summary section. In order to pull these packages into transaction one has to remove conflicting packages, to do that execute:

```
$ dnf upgrade --best --allowerasing
```

6.1.6 Why do I get different results with `dnf upgrade` vs `yum update`?

We get this reported as a bug quite often, but it usually is not. One reason to see this is that DNF does not list update candidates as it explores them. More frequently however the reporter means actual difference in the proposed transaction. This is most often because the metadata the two packagers are working with were taken at a different time (DNF has a notoriously looser schedule on metadata updates to save time and bandwidth), and sometimes also because the depsolvers inside are designed to take a different course of action when encountering some specific update scenario.

The bottom line is: unless a real update problem occurs (i.e. DNF refuses to update a package that YUM updates) with the same set of metadata, this is not an issue.

6.1.7 Is it possible to force DNF to get the latest metadata on `dnf upgrade`?

Yes, clear the cache first:

```
$ dnf clean metadata
$ dnf upgrade
```

or by one command line simply put:

```
$ dnf upgrade --refresh
```

An alternative is to shorten the default expiry time of repos, for that edit `/etc/dnf/dnf.conf` and set:

```
metadata_expire=0
```

Of course, some repos might use a custom `metadata_expire` value, you'll currently have to change these manually too.

If you're the kind of the user who always wants the freshest metadata possible, you'll probably want to *disable the automatic MD updates*.

6.1.8 How do I disable automatic metadata synchronization service?

Several ways to do that. The DNF way is to add the following to `/etc/dnf/dnf.conf`:

```
metadata_timer_sync=0
```

6.1.9 Shouldn't DNF exit soon from certain commands if it is not run under root?

No, there can be systems and scenarios that allow other users than root to successfully perform `dnf install` and similar and it would be impractical to stop these from functioning by the UID check. Alternatively, the practice of checking filesystem permissions instead of the effective UID could lead to false positives since there is plenty of time between DNF startup and the possible transaction start when permissions can be changed by a different process.

If the time loss incurred by repeated runs of DNF is unacceptable for you, consider using the noroot plugin.

6.2 Using DNF in Fedora

6.2.1 For my stable Fedora release, can I install the rawhide packages for testing purposes?

Yes, in two steps: first install the necessary `.repo` files:

```
dnf install fedora-repos-rawhide
```

Then, when you want to include the packages from the rawhide repo, execute a DNF command with Rawhide enabled:

```
dnf --enablerepo=rawhide upgrade rpm
```

Note: Installing rawhide packages onto a stable Fedora release system is generally discouraged as it leads to less tested combinations of installed packages. Please consider this step carefully.

CHAPTER 7

Modularity

Modularity is new way of building, organizing and delivering packages. For more details see: https://docs.pagure.org/modularity/

7.1 Definitions

modulemd Every repository can contain `modules` metadata with modulemd documents. These documents hold metadata about modules such as `Name`, `Stream` or list of packages.

(non-modular) package Package that doesn't belong to a module.

modular package Package that belongs to a module. It is listed in modulemd under the `artifacts` section. Modular packages can be also identified by having `%{modularitylabel}` RPM header set.

(module) stream Stream is a collection of packages, a virtual repository. It is identified with `Name` and `Stream` from modulemd separated with colon, for example "postgresql:9.6".

Module streams can be `active` or `inactive`. `active` means the RPM packages from this stream are included in the set of available packages. Packages from `inactive` streams are filtered out. Streams are `active` either if marked as `default` or if they are explicitly `enabled` by a user action. Streams that satisfy dependencies of `default` or `enabled` streams are also considered `active`. Only one stream of a particular module can be `active` at a given point in time.

7.2 Package filtering

Without modules, packages with the highest version are used by default.

Module streams can distribute packages with lower versions than available in the repositories available to the operating system. To make such packages available for installs and upgrades, the non-modular packages are filtered out when they match by name with modular packages from any existing stream.

7.3 Hotfix repositories

In special cases, a user wants to cherry-pick individual packages provided outside module streams and make them available on along with packages from the active streams. Under normal circumstances, such packages are filtered out. To make the system use packages from a repository regardless of their modularity, specify `module_hotfixes=true` in the .repo file. This protects the repository from package filtering.

Please note the hotfix packages do not override module packages, they only become part of available package set. It's the package `Epoch`, `Version` and `Release` what determines if the package is the latest.

7.4 Fail-safe mechanisms

7.4.1 Repositories with module metadata are unavailable

When a repository with module metadata is unavailable, package filtering must keep working. Non-modular RPMs must remain unavailable and must never get on the system.

This happens when:

- user disables a repository via `--disablerepo` or uses `--repoid`
- user removes a .repo file from disk
- repository is not available and has `skip_if_unavailable=true`

DNF keeps copies of the latest modulemd for every active stream and uses them if there's no modulemd available for the stream. This keeps package filtering working correctly.

The copies are made any time a transaction is resolved and started. That includes RPM transactions as well as any `dnf module <enable|disable|reset>` operations.

When the fail-safe data is used, dnf show such modules as part of @modulefailsafe repository.

7.4.2 Orphaned modular packages

All packages that are built as a part of a module have `%{modularitylabel}` RPM header set. If such package becomes part of RPM transaction and cannot be associated with any available modulemd, DNF prevents from getting it on the system (package is available, but cannot be installed, upgraded, etc.)

CHAPTER 8

DNF Release Notes

8.1 4.2.18 Release Notes

- [doc] Remove note about user-agent whitelist
- Do a substitution of variables in repo_id (RhBug:1748841)
- Respect order of config files in aliases.d (RhBug:1680489)
- Unify downgrade exit codes with upgrade (RhBug:1759847)
- Improve help for ‘dnf module’ command (RhBug:1758447)
- Add shell restriction for local packages (RhBug:1773483)
- Fix detection of the latest module (RhBug:1781769)
- Document the retries config option only works for packages (RhBug:1783041)
- Sort packages in transaction output by nevra (RhBug:1773436)
- Honor repo priority with check-update (RhBug:1769466)
- Strip ‘’ from aliases when processing (RhBug:1680482)
- Print the whole alias definition in case of infinite recursion (RhBug:1680488)
- Add support of commandline packages by repoquery (RhBug:1784148)
- Running with tsflags=test doesn’t update log files
- Restore functionality of remove –oldinstallonly
- Allow disabling individual aliases config files (RhBug:1680566)

Bugs fixed in 4.2.18:

- **:rhbug:‘1773483‘**
- **:rhbug:‘1758447‘**
- **:rhbug:‘1748841‘**

- **:rhbug:‘1679008‘**
- **:rhbug:‘1680482‘**
- **:rhbug:‘1680566‘**
- **:rhbug:‘1784148‘**
- **:rhbug:‘1680488‘**
- **:rhbug:‘1759847‘**
- **:rhbug:‘1773436‘**
- **:rhbug:‘1783041‘**
- **:rhbug:‘1680489‘**
- **:rhbug:‘1781769‘**

8.2 4.2.17 Release Notes

- Enable versionlock for check-update command (RhBug:1750620)
- Add error message when no active modules matched (RhBug:1696204)
- Log mirror failures as warning when repo load fails (RhBug:1713627)
- dnf-automatic: Change all systemd timers to a fixed time of day (RhBug:1754609)
- DNF can use config from the remote location (RhBug:1721091)
- [doc] update reference to plugin documentation (RhBug:1706386)
- [yum compatibility] Report all packages in repoinfo
- [doc] Add definition of active/inactive module stream
- repoquery: Add a switch to disable modular excludes
- Report more informative messages when no match for argument (RhBug:1709563)
- [doc] Add description of excludes in dnf
- Report more descriptive message when removed package is excluded
- Add module repoquery command
- Fix assumptions about ARMv8 and the way the rpm features work (RhBug:1691430)
- Add Requires information into module info commands
- Enhance inheritance of transaction reasons (RhBug:1672618,1769788)

Bugs fixed in 4.2.17:

- **:rhbug:‘1696204‘**
- **:rhbug:‘1709563‘**
- **:rhbug:‘1721091‘**
- **:rhbug:‘1769788‘**
- **:rhbug:‘1706386‘**
- **:rhbug:‘1750620‘**

- **:rhbug:‘1713627‘**
- **:rhbug:‘1672618‘**
- **:rhbug:‘1754609‘**
- **:rhbug:‘1691430‘**

8.3 4.2.16 Release Notes

- Make DNF compatible with FIPS mode (RhBug:1762032)
- Return always alphabetically sorted modular profiles
- Revert “Fix messages for starting and failing scriptlets”

8.4 4.2.15 Release Notes

- Fix downloading local packages into destdir (RhBug:1727137)
- Report skipped packages with identical nevra only once (RhBug:1643109)
- Restore functionality of dnf remove –duplicates (RhBug:1674296)
- Improve API documentation
- Document NEVRA parsing in the man page
- Do not wrap output when no terminal (RhBug:1577889)
- Allow to ship alternative dnf.conf (RhBug:1752249)
- Don’t check if repo is expired if it doesn’t have loaded metadata (RhBug:1745170)
- Remove duplicate entries from “dnf search” output (RhBug:1742926)
- Set default value of repo name attribute to repo id (RhBug:1669711)
- Allow searching in disabled modules using “dnf module provides” (RhBug:1629667)
- Group install takes obsoletes into account (RhBug:1761137)
- Improve handling of vars
- Do not load metadata for repolist commands (RhBug:1697472,1713055,1728894)
- Fix messages for starting and failing scriptlets (RhBug:1724779)
- Don’t show older install-only pkgs updates in updateinfo (RhBug:1649383,1728004)
- Add –ids option to the group command (RhBug:1706382)
- Add –with_cve and –with_bz options to the updateinfo command (RhBug:1750528)

Bugs fixed in 4.2.15:

- **:rhbug:‘1738837‘**
- **:rhbug:‘1674296‘**
- **:rhbug:‘1577889‘**
- **:rhbug:‘1669711‘**
- **:rhbug:‘1643109‘**

- **:rhbug:‘1649383‘**
- **:rhbug:‘1666236‘**
- **:rhbug:‘1728894‘**
- **:rhbug:‘1727137‘**
- **:rhbug:‘1689645‘**
- **:rhbug:‘1742926‘**
- **:rhbug:‘1761137‘**
- **:rhbug:‘1706382‘**
- **:rhbug:‘1761518‘**
- **:rhbug:‘1752249‘**
- **:rhbug:‘1760937‘**
- **:rhbug:‘1713055‘**
- **:rhbug:‘1724779‘**
- **:rhbug:‘1745170‘**
- **:rhbug:‘1750528‘**

8.5 4.2.11 Release Notes

- Improve modularity documentation (RhBug:1730162,1730162,1730807,1734081)
- Fix detection whether system is running on battery (used by metadata caching timer) (RhBug:1498680)
- New repoquery queryformat: %{reason}
- Print rpm errors during test transaction (RhBug:1730348)
- Fix: –setopt and repo with dots
- Fix incorrectly marked profile and stream after failed rpm transaction check (RhBug:1719679)
- Show transaction errors inside dnf shell (RhBug:1743644)
- Don’t reinstall modified packages with the same NEVRA (RhBug:1644241)
- dnf-automatic now respects versionlock excludes (RhBug:1746562)

Bugs fixed in 4.2.11:

- **:rhbug:‘1498680‘**
- **:rhbug:‘1730348‘**
- **:rhbug:‘1719679‘**
- **:rhbug:‘1601741‘**
- **:rhbug:‘1665636‘**
- **:rhbug:‘1739457‘**
- **:rhbug:‘1715807‘**
- **:rhbug:‘1734081‘**

- **:rhbug:‘1739773‘**
- **:rhbug:‘1730807‘**
- **:rhbug:‘1728252‘**
- **:rhbug:‘1746562‘**
- **:rhbug:‘1730162‘**
- **:rhbug:‘1743644‘**
- **:rhbug:‘1737201‘**
- **:rhbug:‘1689645‘**
- **:rhbug:‘1741381‘**

8.6 4.2.9 Release Notes

- Prevent printing empty Error Summary (RhBug: 1690414)
- [doc] Add user_agent and countme options

8.7 4.2.8 Release Notes

- Enhance synchronization of rpm transaction to swdb
- Accept multiple specs in repoquery options (RhBug:1667898)
- Prevent switching modules in all cases (RhBug:1706215)
- [history] Don’t store failed transactions as succeeded
- [history] Do not require root for informative commands
- [dnssec] Fix UnicodeWarning when using new rpm (RhBug:1699650)
- Print rpm error messages during transaction (RhBug:1677199)
- Report missing default profile as an error (RhBug:1669527)
- Apply excludes before modular excludes (RhBug:1709453)
- Improve help for command line arguments (RhBug:1659328)
- [doc] Describe a behavior when plugin is removed (RhBug:1700741)
- Add new modular API method ModuleBase.get_modules
- Mark features used by ansible, anaconda and subscription-manager as an API

Bugs fixed in 4.2.8:

- **:rhbug:‘1630113‘**
- **:rhbug:‘1653736‘**
- **:rhbug:‘1669527‘**
- **:rhbug:‘1661814‘**
- **:rhbug:‘1667898‘**
- **:rhbug:‘1673075‘**

- **:rhbug:‘1677199‘**
- **:rhbug:‘1699650‘**
- **:rhbug:‘1700741‘**
- **:rhbug:‘1706215‘**
- **:rhbug:‘1709453‘**

8.8 4.2.7 Release Notes

- Set default to skip_if_unavailable=false (RhBug:1679509)
- Fix package reinstalls during yum module remove (RhBug:1700529)
- Fail when “-c” option is given nonexistent file (RhBug:1512457)
- Reuse empty lock file instead of stopping dnf (RhBug:1581824)
- Propagate comps ‘default’ value correctly (RhBug:1674562)
- Better search of provides in /(s)bin/ (RhBug:1657993)
- Add detection for armv7hcnl (RhBug:1691430)
- Fix group install/upgrade when group is not available (RhBug:1707624)
- Report not matching plugins when using –enableplugin/–disableplugin (RhBug:1673289) (RhBug:1467304)
- Add support of modular FailSafe (RhBug:1623128)
- Replace logrotate with build-in log rotation for dnf.log and dnf.rpm.log (RhBug:1702690)

Bugs fixed in 4.2.7:

- **:rhbug:‘1702690‘**
- **:rhbug:‘1672649‘**
- **:rhbug:‘1467304‘**
- **:rhbug:‘1673289‘**
- **:rhbug:‘1674562‘**
- **:rhbug:‘1581824‘**
- **:rhbug:‘1709783‘**
- **:rhbug:‘1512457‘**
- **:rhbug:‘1673913‘**

8.9 4.2.6 Release Notes

- librepo: Turn on debug logging only if debuglevel is greater than 2 (RhBug:1355764,1580022)
- Fix issues with terminal hangs when attempting bash completion (RhBug:1702854)
- Rename man page from dnf.automatic to dnf-automatic to match command name
- [provides] Enhanced detecting of file provides (RhBug:1702621)
- [provides] Sort the output packages alphabetically

Bugs fixed in 4.2.6:

- **:rhbug:‘1355764‘**
- **:rhbug:‘1580022‘**
- **:rhbug:‘1702621‘**
- **:rhbug:‘1702854‘**

8.10 4.2.5 Release Notes

- Fix multilib obsoletes (RhBug:1672947)
- Do not remove group package if other packages depend on it
- Remove duplicates from “dnf list” and “dnf info” outputs
- Installroot now requires absolute path
- Fix the installation of completion_helper.py
- Allow globs in setopt in repoid part
- Fix formatting of message about free space required
- [doc] Add info of relation update_cache with fill_sack (RhBug:1658694)
- Fix installation failure when duplicate RPMs are specified (RhBug:1687286)
- Add command abbreviations (RhBug:1634232)
- Allow plugins to terminate dnf (RhBug:1701807)

Bugs fixed in 4.2.5:

- **:rhbug:‘1701807‘**
- **:rhbug:‘1634232‘**
- **:rhbug:‘1687286‘**
- **:rhbug:‘1658694‘**
- **:rhbug:‘1672947‘**

8.11 4.2.2 Release Notes

- [conf] Use environment variables prefixed with `DNF_VAR_`
- Enhance documentation of –whatdepends option (RhBug:1687070)
- Allow adjustment of repo from –repofrompath (RhBug:1689591)
- Document cachedir option (RhBug:1691365)
- Retain order of headers in search results (RhBug:1613860)
- Solve traceback with the “dnf install @module” (RhBug:1688823)
- Build “yum” instead of “dnf-yum” on Fedora 31

Bugs fixed in 4.2.2:

- **:rhbug:‘1689591‘**

- **:rhbug:‘1687070‘**

8.12 4.2.1 Release Notes

- Do not allow direct module switch (RhBug:1669491)
- Use improved config parser that preserves order of data
- Fix `alias list` command (RhBug:1666325)
- Postpone yum conflict to F31
- Update documentation: implemented plugins; options; deprecated commands (RhBug:1670835,1673278)
- Support zchunk (“.zck”) compression
- Fix behavior of `--bz` option when specifying more values
- Follow RPM security policy for package verification
- Update modules regardless of installed profiles
- Add protection of yum package (RhBug:1639363)
- Fix `list --showduplicates` (RhBug:1655605)

Bugs fixed in 4.2.1:

- **:rhbug:‘1655605‘**
- **:rhbug:‘1669247‘**
- **:rhbug:‘1670835‘**
- **:rhbug:‘1673278‘**
- **:rhbug:‘1677640‘**
- **:rhbug:‘1597182‘**
- **:rhbug:‘1666325‘**
- **:rhbug:‘1678689‘**
- **:rhbug:‘1669491‘**

8.13 4.1.0 Release Notes

- Allow to enable modules that break default modules (RhBug:1648839)
- Enhance documentation - API examples
- Add best as default behavior (RhBug:1670776,1671683)
- Add –nobest option

Bugs fixed in 4.1.0:

- **:rhbug:‘1585509‘**
- **:rhbug:‘1672432‘**
- **:rhbug:‘1509393‘**
- **:rhbug:‘1667423‘**

- **:rhbug:‘1656726‘**
- **:rhbug:‘1671683‘**
- **:rhbug:‘1667426‘**

8.14 4.0.10 Release Notes

- Updated difference YUM vs. DNF for yum-updateonboot
- Added new command `dnf alias [options] [list|add|delete] [<name>...]` to allow the user to define and manage a list of aliases
- Enhanced documentation
- Unifying return codes for remove operations
- [transaction] Make transaction content available for commands
- Triggering transaction hooks if no transaction (RhBug:1650157)
- Add hotfix packages to install pool (RhBug:1654738)
- Report group operation in transaction table
- [sack] Change algorithm to calculate rpmdb_version

Bugs fixed in 4.0.10:

- **:rhbug:‘1654738‘**
- **:rhbug:‘1495482‘**

8.15 4.0.9 Release Notes

- Added *dnf.repo.Repo.get_http_headers()*
- Added *dnf.repo.Repo.set_http_headers()*
- Added *dnf.repo.Repo.add_metadata_type_to_download()*
- Added *dnf.repo.Repo.get_metadata_path()*
- Added *dnf.repo.Repo.get_metadata_content()*
- Added –changelogs option for check-update command
- [module] Add information about active modules
- Hide messages created only for logging
- Enhanced –setopt option
- [module] Fix dnf remove @<module>
- [transaction] Make transaction content available for plugins

Bugs fixed in 4.0.9:

- **:rhbug:‘1541832‘**
- **:rhbug:‘1642796‘**
- **:rhbug:‘1637148‘**

- **:rhbug:‘1639998‘**
- **:rhbug:‘1615164‘**
- **:rhbug:‘1636480‘**

8.16 4.0.4 Release Notes

- Add dnssec extension
- Set termforce to AUTO to automatically detect if stdout is terminal
- Repoquery command accepts –changelogs option (RhBug:1483458)
- Calculate sack version from all installed packages (RhBug:1624291)
- [module] Allow to enable module dependencies (RhBug:1622566)

Bugs fixed in 4.0.4:

- **:rhbug:‘1508649‘**
- **:rhbug:‘1590690‘**
- **:rhbug:‘1624291‘**
- **:rhbug:‘1631217‘**
- **:rhbug:‘1489308‘**
- **:rhbug:‘1625879‘**
- **:rhbug:‘1483458‘**
- **:rhbug:‘1497171‘**
- **:rhbug:‘1620242‘**

8.17 3.6.1 Release Notes

- [module] Improved module commands list, info
- [module] Reports error from module solver

Bugs fixed in 3.6.1:

- **:rhbug:‘1626011‘**
- **:rhbug:‘1631458‘**
- **:rhbug:‘1305340‘**
- **:rhbug:‘1305340‘**
- **:rhbug:‘1623866‘**
- **:rhbug:‘1600444‘**
- **:rhbug:‘1628056‘**

8.18 3.5.1 Release Notes

- [module] Fixed list and info subcommands

8.19 3.5.0 Release Notes

- New implementation of modularity

8.20 3.0.2 Release Notes

- Add limited compatibility with dnf-2.0 (constants)

8.21 3.0.1 Release Notes

- Support of MODULES - new DNF command *module*
- `dnf.conf.Conf.proxy_auth_method`
- New repoquery option *–depends* and *–whatdepends*
- Enhanced support of variables
- Enhanced documentation

Bugs fixed in 3.0.1:

- **:rhbug:‘1565599‘**
- **:rhbug:‘1508839‘**
- **:rhbug:‘1506486‘**
- **:rhbug:‘1506475‘**
- **:rhbug:‘1505577‘**
- **:rhbug:‘1505574‘**
- **:rhbug:‘1505573‘**
- **:rhbug:‘1480481‘**
- **:rhbug:‘1496732‘**
- **:rhbug:‘1497272‘**
- **:rhbug:‘1488100‘**
- **:rhbug:‘1488086‘**
- **:rhbug:‘1488112‘**
- **:rhbug:‘1488105‘**
- **:rhbug:‘1488089‘**
- **:rhbug:‘1488092‘**
- **:rhbug:‘1486839‘**

- **:rhbug:‘1486839‘**
- **:rhbug:‘1486827‘**
- **:rhbug:‘1486816‘**
- **:rhbug:‘1565647‘**
- **:rhbug:‘1583834‘**
- **:rhbug:‘1576921‘**
- **:rhbug:‘1270295‘**
- **:rhbug:‘1361698‘**
- **:rhbug:‘1369847‘**
- **:rhbug:‘1368651‘**
- **:rhbug:‘1563841‘**
- **:rhbug:‘1387622‘**
- **:rhbug:‘1575998‘**
- **:rhbug:‘1577854‘**
- **:rhbug:‘1387622‘**
- **:rhbug:‘1542416‘**
- **:rhbug:‘1542416‘**
- **:rhbug:‘1496153‘**
- **:rhbug:‘1568366‘**
- **:rhbug:‘1539803‘**
- **:rhbug:‘1552576‘**
- **:rhbug:‘1545075‘**
- **:rhbug:‘1544359‘**
- **:rhbug:‘1547672‘**
- **:rhbug:‘1537957‘**
- **:rhbug:‘1542920‘**
- **:rhbug:‘1507129‘**
- **:rhbug:‘1512956‘**
- **:rhbug:‘1512663‘**
- **:rhbug:‘1247083‘**
- **:rhbug:‘1247083‘**
- **:rhbug:‘1247083‘**
- **:rhbug:‘1519325‘**
- **:rhbug:‘1492036‘**
- **:rhbug:‘1391911‘**
- **:rhbug:‘1391911‘**

- **:rhbug:‘1479330‘**
- **:rhbug:‘1505185‘**
- **:rhbug:‘1305232‘**

8.22 2.7.5 Release Notes

- Improved performance for excludes and includes handling
- Fixed problem of handling checksums for local repositories
- Fix traceback when using dnf.Base.close()

Bugs fixed in 2.7.5:

- **:rhbug:‘1502106‘**
- **:rhbug:‘1500361‘**
- **:rhbug:‘1503575‘**

8.23 2.7.4 Release Notes

- Enhanced performance for excludes and includes handling
- Solved memory leaks at time of closing of dnf.Base()

Bugs fixed in 2.7.4:

- **:rhbug:‘1480979‘**
- **:rhbug:‘1461423‘**
- **:rhbug:‘1499564‘**
- **:rhbug:‘1499534‘**
- **:rhbug:‘1499623‘**

8.24 2.7.3 Release Notes

Bugs fixed in 2.7.3:

- **:rhbug:‘1472847‘**
- **:rhbug:‘1498426‘**
- **:rhbug:‘1427144‘**

8.25 2.7.2 Release Notes

API additions in 2.7.2:

- Added new option `--comment=<comment>` that adds a comment to transaction in history
- `dnf.Base.pre_configure_plugin()` configure plugins by running their pre_configure() method

- Added pre_configure() method for plugins and commands to configure dnf before repos are loaded

Bugs fixed in 2.7.2:

- **:rhbug:‘1421478‘**
- **:rhbug:‘1491560‘**
- **:rhbug:‘1465292‘**
- **:rhbug:‘1279001‘**
- **:rhbug:‘1212341‘**
- **:rhbug:‘1299482‘**
- **:rhbug:‘1192811‘**
- **:rhbug:‘1288845‘**
- **:rhbug:‘1237349‘**
- **:rhbug:‘1470050‘**
- **:rhbug:‘1347927‘**
- **:rhbug:‘1478115‘**
- **:rhbug:‘1461171‘**
- **:rhbug:‘1495116‘**
- **:rhbug:‘1448874‘**

8.26 2.6.3 Release Notes

API additions in 2.6.3:

- Added auto substitution for all variables used for repo creation by *dnf.repodict.RepoDict.add_new_repo()*
- Added description of `--downloaddir=<path>` dnf option

Bugs fixed in 2.6.3:

- **:rhbug:‘1476215‘**
- **:rhbug:‘1473964‘**
- **:rhbug:‘1359482‘**
- **:rhbug:‘1476834‘**
- **:rhbug:‘1244755‘**
- **:rhbug:‘1476748‘**
- **:rhbug:‘1476464‘**
- **:rhbug:‘1464192‘**
- **:rhbug:‘1463107‘**
- **:rhbug:‘1426196‘**
- **:rhbug:‘1457507‘**

8.27 2.6.2 Release Notes

API additions in 2.6.2:

- `dnf.conf.Conf.basearch`
- `dnf.conf.Conf.arch`
- `dnf.conf.Conf.ignorearch`
- Introduced new configuration option `autocheck_running_kernel`
- *`dnf.subject.Subject.get_best_selector()`* can use three additional key words: `obsoletes`, `reports`, and `reponame`.

From commandline it is possible to use new option `--noautoremove` to disable removal of dependencies that are no longer used.

Bugs fixed in 2.6.2:

- **:rhbug:‘1279001‘**
- **:rhbug:‘1397848‘**
- **:rhbug:‘1361424‘**
- **:rhbug:‘1387925‘**
- **:rhbug:‘1332099‘**
- **:rhbug:‘1470116‘**
- **:rhbug:‘1161950‘**
- **:rhbug:‘1320254‘**
- **:rhbug:‘1424723‘**
- **:rhbug:‘1462486‘**
- **:rhbug:‘1314405‘**
- **:rhbug:‘1457368‘**
- **:rhbug:‘1339280‘**
- **:rhbug:‘1138978‘**
- **:rhbug:‘1423472‘**
- **:rhbug:‘1427365‘**
- **:rhbug:‘1398871‘**
- **:rhbug:‘1432312‘**

8.28 2.5.1 Release Notes

API additions in 2.5.1:

- *`dnf.Plugin.pre_transaction()`* is a hook that is called just before transaction execution.
- *`dnf.subject.Subject.get_nevra_possibilities()`* returns generator for every possible nevra.

Bugs fixed in 2.5.1:

- **:rhbug:‘1456419‘**
- **:rhbug:‘1445021‘**
- **:rhbug:‘1400714‘**
- **:rhbug:‘1250702‘**
- **:rhbug:‘1381988‘**
- **:rhbug:‘1397848‘**
- **:rhbug:‘1321407‘**
- **:rhbug:‘1291867‘**
- **:rhbug:‘1372895‘**
- **:rhbug:‘1444751‘**

8.29 2.5.0 Release Notes

API additions in 2.5.0:

dnf.callback.DownloadProgress.start() can use one additional key word `total_drpms`.

Bugs fixed in 2.5.0:

- **:rhbug:‘1350546‘**
- **:rhbug:‘1449618‘**
- **:rhbug:‘1270451‘**
- **:rhbug:‘1254966‘**
- **:rhbug:‘1426787‘**
- **:rhbug:‘1293983‘**
- **:rhbug:‘1370062‘**
- **:rhbug:‘1293067‘**
- **:rhbug:‘1393814‘**
- **:rhbug:‘1398040‘**
- **:rhbug:‘1342157‘**
- **:rhbug:‘1379906‘**
- **:rhbug:‘1198975‘**

8.30 2.4.1 Release Notes

DNF command additions in 2.4.1:

- `dnf [options] repoquery --userinstalled` limit the resulting set only to packages installed by user.

Bugs fixed in 2.4.1:

- **:rhbug:‘1446756‘**

- **:rhbug:‘1446432‘**
- **:rhbug:‘1446641‘**
- **:rhbug:‘1278124‘**
- **:rhbug:‘1301868‘**

8.31 2.4.0 Release Notes

API additions in 2.4.0:

- *dnf.subject.Subject.get_best_query()* can use two additional key words: `with_nevra`, and `with_filenames`.
- Added description of `dnf.repo.Repo.cost`
- Added description of `dnf.repo.Repo.excludepkgs`
- Added description of `dnf.repo.Repo.includepkgs`

DNF command additions in 2.4.0:

- `--enableplugin=<plugin names>` *command line argument* enable the listed plugins specified by names or globs.
- `--releasever=<release>` *command line argument* now autodetect releasever in installroot from host if `/` value is used as `<release>`.

Bugs fixed in 2.4.0:

- **:rhbug:‘1302935‘**
- **:rhbug:‘1248684‘**
- **:rhbug:‘1441636‘**
- **:rhbug:‘1438438‘**
- **:rhbug:‘1256313‘**
- **:rhbug:‘1161950‘**
- **:rhbug:‘1421244‘**

8.32 2.3.0 Release Notes

API additions in 2.3.0:

- *dnf.package.Package.remote_location()* returns location from where the package can be downloaded from.

DNF command additions in 2.3.0:

- `dnf [options] repoquery --whatconflicts <capability>` limit the resulting set only to packages that conflict `<capability>`.
- `dnf [options] repoquery --whatobsoletes <capability>` limit the resulting set only to packages that obsolete `<capability>`.
- `dnf [options] repoquery --location` show a location where the package could be downloaded from.

- `dnf [options] repoquery --nvr` show found packages in format name-version-release.
- `dnf [options] repoquery --nevra` show found packages in format name-epoch:version-release.architecture (default).
- `dnf [options] repoquery --envra` show found packages in format epoch:name-version-release.architecture.
- `dnf [options] repoquery --recursive` query packages recursively. Can be used with `--whatrequires <REQ>` (optionally with –alldeps, but it has no effect with –exactdeps), or with `--requires <REQ> --resolve`.

Bugs fixed in 2.3.0:

- **:rhbug:‘1290137‘**
- **:rhbug:‘1349314‘**
- **:rhbug:‘1247122‘**
- **:rhbug:‘1298717‘**

8.33 2.2.0 Release Notes

API additions in 2.2.0:

- *`dnf.callback.TransactionProgress.progress()`* has new actions: TRANS_PREPARATION, TRANS_POST, and PKG_SCRIPTLET.

Bugs fixed in 2.2.0:

- **:rhbug:‘1411432‘**
- **:rhbug:‘1406130‘**
- **:rhbug:‘1411423‘**
- **:rhbug:‘1369212‘**

8.34 2.1.1 Release Notes

Bugs fixed in 2.1.1:

- **:rhbug:‘1417542‘**
- **:rhbug:‘1401446‘**
- **:rhbug:‘1416699‘**
- **:rhbug:‘1427132‘**
- **:rhbug:‘1397047‘**
- **:rhbug:‘1379628‘**
- **:rhbug:‘1424939‘**
- **:rhbug:‘1396992‘**
- **:rhbug:‘1412970‘**

8.35 2.1.0 Release Notes

API additions in 2.1.0:

- *dnf.Base.update_cache()* downloads and caches in binary format metadata for all known repos.

Bugs fixed in 2.1.0:

- **:rhbug:‘1421835‘**
- **:rhbug:‘1415711‘**
- **:rhbug:‘1417627‘**

8.36 2.0.1 Release Notes

API changes in 2.0.1:

- *dnf.Base.package_downgrade()* now accept keyword strict to ignore problems with dep-solving

API additions in 2.0.1:

- *dnf.Base.autoremove()* removes all ‘leaf’ packages from the system that were originally installed as dependencies
- `dnf.cli.Cli.redirect_logger()` changes minimal logger level for terminal output to stdout and stderr

DNF command additions in 2.0.1:

- `dnf [options] shell [filename]` opens an interactive shell for conducting multiple commands during a single execution of DNF
- `dnf [options] swap <remove-spec> <install-spec>` removes spec and install spec in one transaction

Bugs fixed in 2.0.1:

- **:rhbug:‘1409361‘**
- **:rhbug:‘1414512‘**
- **:rhbug:‘1238808‘**
- **:rhbug:‘1386085‘**
- **:rhbug:‘1286553‘**
- **:rhbug:‘1337731‘**
- **:rhbug:‘1336879‘**
- **:rhbug:‘1173349‘**
- **:rhbug:‘1329617‘**
- **:rhbug:‘1283255‘**
- **:rhbug:‘1369411‘**
- **:rhbug:‘1243393‘**
- **:rhbug:‘1243393‘**
- **:rhbug:‘1411349‘**

- **:rhbug:‘1345976‘**
- **:rhbug:‘1369212‘**
- **:rhbug:‘1349247‘**
- **:rhbug:‘1403930‘**
- **:rhbug:‘1403465‘**
- **:rhbug:‘1110780‘**
- **:rhbug:‘1405333‘**
- **:rhbug:‘1254879‘**

8.37 2.0.0 Release Notes

List of all incompatible changes can be found at: *dnf-1 vs dnf-2*

API changes in 2.0.0:

- *dnf.Base.add_remote_rpms()* now suppresses any error if `strict` equals to `False`.
- *dnf.Base.read_comps()* now limits results to system basearch if `arch_filter` equals to `True`.
- `dnf.cli.Cli.configure()` now doesn't take any additional arguments.
- `dnf.cli.Cli.run()` now doesn't take any additional arguments.
- *dnf.Plugin.read_config()* now doesn't take any name of config file.
- `dnf.Repo.__init__()` now takes *parent_conf* argument which is an instance of *dnf.conf.Conf* holding main dnf configuration instead of *cachedir* path.
- `exclude` and `include` configuration options change to `excludepkgs` and `includepkgs`.

API additions in 2.0.0:

- *dnf.Base.init_plugins()* initializes plugins. It is possible to disable some plugins by passing the list of their name patterns to `disabled_glob`.
- *dnf.Base.configure_plugins()* configures plugins by running their `configure()` method.
- `dnf.Base.urlopen()` opens the specified absolute `url` and returns a file object which respects proxy setting even for non-repo downloads
- Introduced new configuration options: `check_config_file_age`, `clean_requirements_on_remove`, `deltarpm_percentage`, `exit_on_lock`, `get_reposdir`, `group_package_types`, `installonlypkgs`, `keepcache`, `protected_packages`, `retries`, `type`, and `upgrade_group_objects_upgrade`. For detailed description see: *DNF API*.
- Introduced new configuration methods: `dump()` and `write_raw_configfile()`. For detailed description see: *DNF API*.
- Introduced *dnf.package.Package* attributes `debug_name`, `downloadsize`, `source_debug_name` and `source_name`. For detailed description see: *DNF Package API*.
- *dnf.query.Query.extras()* returns a new query that limits the result to installed packages that are not present in any repo.
- `dnf.repo.Repo.enable_debug_repos()` enables debug repos corresponding to already enabled binary repos.

- `dnf.repo.Repo.enable_source_repos()` enables source repos corresponding to already enabled binary repos.
- *`dnf.repo.Repo.dump()`* prints repository configuration, including inherited values.
- *`dnf.query.Query.filter()`* now accepts optional argument *pkg*.

DNF command changes in 2.0.0:

- `dnf [options] group install [with-optional] <group-spec>...` changes to `dnf [options] group install [--with-optional] <group-spec>...`.
- `dnf [options] list command [<package-name-specs>...]` changes to *dnf [options] list –command [<package-name-specs>...]*'.
- `dnf [options] makecache timer` changes to `dnf [options] makecache --timer`.
- `dnf [options] repolist [enabled|disabled|all]` changes to `dnf [options] repolist [--enabled|--disabled|--all]`.
- `dnf [options] repository-packages <repoid> info command [<package-name-spec>...]` changes to `dnf [options] repository-packages <repoid> info --command [<package-name-spec>...]`.
- `dnf repoquery --duplicated` changes to `dnf repoquery --duplicates`.
- `dnf [options] search [all] <keywords>...` changes to `dnf [options] search [--all] <keywords>...`.
- `dnf [options] updateinfo [<availability>] [<spec>...]` changes to `dnf [options] updateinfo [--summary|--list|--info] [<availability>] [<spec>...]`.
- `--disablerepo` *command line argument* is mutually exclusive with `--repo`.
- `--enablerepo` *command line argument* now appends repositories.
- `--installroot` *command line argument*. For detailed description see: *DNF command API*.
- `--releasever` *command line argument* now doesn't detect release number from running system.
- `--repofrompath` *command line argument* can now be combined with `--repo` instead of `--enablerepo`.
- Alternative of yum's `deplist` changes from `dnf repoquery --requires` to `dnf repoquery --deplist`.
- New systemd units *dnf-automatic-notifyonly*, *dnf-automatic-download*, *dnf-automatic-download* were added for a better customizability of *dnf-automatic*.

DNF command additions in 2.0.0:

- `dnf [options] remove --duplicates` removes older version of duplicated packages.
- `dnf [options] remove --oldinstallonly``removes old installonly packages keeping only ``installonly_limit` latest versions.
- `dnf [options] repoquery [<select-options>] [<query-options>] [<pkg-spec>]` searches the available DNF repositories for selected packages and displays the requested information about them. It is an equivalent of `rpm -q` for remote repositories.
- `dnf [options] repoquery --querytags` provides list of recognized tags by repoquery option *--queryformat*.
- `--repo` *command line argument* enables just specific repositories by an id or a glob. Can be used multiple times with accumulative effect. It is basically shortcut for `--disablerepo="*" --enablerepo=<repoid>` and is mutually exclusive with `--disablerepo` option.

- New commands have been introduced: `check` and `upgrade-minimal`.
- New security options introduced: `bugfix`, `enhancement`, `newpackage`, `security`, `advisory`, `bzs`, `cves`, `sec-severity` and `secseverity`.

Bugs fixed in 2.0.0:

- **:rhbug:‘1229730‘**
- **:rhbug:‘1375277‘**
- **:rhbug:‘1384289‘**
- **:rhbug:‘1398272‘**
- **:rhbug:‘1382224‘**
- **:rhbug:‘1177785‘**
- **:rhbug:‘1272109‘**
- **:rhbug:‘1234930‘**
- **:rhbug:‘1341086‘**
- **:rhbug:‘1382247‘**
- **:rhbug:‘1381216‘**
- **:rhbug:‘1381432‘**
- **:rhbug:‘1096506‘**
- **:rhbug:‘1332830‘**
- **:rhbug:‘1348766‘**
- **:rhbug:‘1337731‘**
- **:rhbug:‘1333591‘**
- **:rhbug:‘1314961‘**
- **:rhbug:‘1372307‘**
- **:rhbug:‘1373108‘**
- **:rhbug:‘1148627‘**
- **:rhbug:‘1267298‘**
- **:rhbug:‘1373591‘**
- **:rhbug:‘1230355‘**
- **:rhbug:‘1366793‘**
- **:rhbug:‘1369411‘**
- **:rhbug:‘1366793‘**
- **:rhbug:‘1369459‘**
- **:rhbug:‘1306096‘**
- **:rhbug:‘1368832‘**
- **:rhbug:‘1366793‘**
- **:rhbug:‘1359016‘**

- :rhbug:‘1365593‘
- :rhbug:‘1297087‘
- :rhbug:‘1227053‘
- :rhbug:‘1356926‘
- :rhbug:‘1055910‘
- :rhbug:‘1219867‘
- :rhbug:‘1226677‘
- :rhbug:‘1350604‘
- :rhbug:‘1253120‘
- :rhbug:‘1158548‘
- :rhbug:‘1262878‘
- :rhbug:‘1318852‘
- :rhbug:‘1327438‘
- :rhbug:‘1343880‘
- :rhbug:‘1338921‘
- :rhbug:‘1284349‘
- :rhbug:‘1338921‘
- :rhbug:‘1284349‘
- :rhbug:‘1306096‘
- :rhbug:‘1218071‘
- :rhbug:‘1193823‘
- :rhbug:‘1246211‘
- :rhbug:‘1193851‘
- :rhbug:‘1158548‘
- :rhbug:‘1215208‘
- :rhbug:‘1212693‘
- :rhbug:‘1212341‘
- :rhbug:‘1306591‘
- :rhbug:‘1227001‘
- :rhbug:‘1163028‘
- :rhbug:‘1279185‘
- :rhbug:‘1289067‘
- :rhbug:‘1328674‘
- :rhbug:‘1380580‘
- :rhbug:‘1327999‘
- :rhbug:‘1400081‘

- **:rhbug:‘1293782‘**
- **:rhbug:‘1386078‘**
- **:rhbug:‘1358245‘**
- **:rhbug:‘1243393‘**
- **:rhbug:‘1339739‘**

8.38 1.1.10 Release Notes

Fixed unicode handling and fixing other bugs.

Bugs fixed in 1.1.10:

- **:rhbug:‘1257965‘**
- **:rhbug:‘1352130‘**
- **:rhbug:‘1343764‘**
- **:rhbug:‘1308994‘**
- **:rhbug:‘1230183‘**
- **:rhbug:‘1295090‘**
- **:rhbug:‘1325869‘**
- **:rhbug:‘1338046‘**
- **:rhbug:‘1214768‘**
- **:rhbug:‘1338504‘**
- **:rhbug:‘1338564‘**

8.39 1.1.9 Release Notes

From this release if you use any non-API methods warning will be printed and bugfixes.

Bugs fixed in 1.1.9:

- **:rhbug:‘1324086‘**
- **:rhbug:‘1332012‘**
- **:rhbug:‘1292892‘**
- **:rhbug:‘1328674‘**
- **:rhbug:‘1286556‘**
- **:rhbug:‘1245121‘**

8.40 1.1.8 Release Notes

Improvements in documentation, bugfixes, translation updates.

Bugs fixed in 1.1.8:

- **:rhbug:‘1309408‘**
- **:rhbug:‘1209649‘**
- **:rhbug:‘1272977‘**
- **:rhbug:‘1322226‘**
- **:rhbug:‘1315349‘**
- **:rhbug:‘1214562‘**
- **:rhbug:‘1313215‘**
- **:rhbug:‘1306057‘**
- **:rhbug:‘1289164‘**

8.41 1.1.7 Release Notes

Added *dnf.rpm.basearch()* method, intended for the detection of CPU base architecture.

The *group list* command was enriched with `installed` and `available` switches.

Documented a standard way of overriding autodetected architectures in *DNF API*.

Bugs fixed in 1.1.7:

- **:rhbug:‘1286477‘**
- **:rhbug:‘1305356‘**
- **:rhbug:‘1258503‘**
- **:rhbug:‘1283432‘**
- **:rhbug:‘1268818‘**
- **:rhbug:‘1306304‘**
- **:rhbug:‘1302934‘**
- **:rhbug:‘1303149‘**
- **:rhbug:‘1302217‘**

8.42 1.1.6 Release Notes

Added support of socks5 proxy.

Bugs fixed in 1.1.6:

- **:rhbug:‘1291895‘**
- **:rhbug:‘1256587‘**
- **:rhbug:‘1287221‘**
- **:rhbug:‘1277360‘**
- **:rhbug:‘1294241‘**
- **:rhbug:‘1289166‘**
- **:rhbug:‘1294355‘**

- **:rhbug:‘1226322‘**
- **:rhbug:‘1275878‘**
- **:rhbug:‘1239274‘**

8.43 1.1.5 Release Notes

Improved the start-up time of bash completion.

Reviewed documentation.

Bugs fixed in 1.1.5:

- **:rhbug:‘1286619‘**
- **:rhbug:‘1229046‘**
- **:rhbug:‘1282250‘**
- **:rhbug:‘1265391‘**
- **:rhbug:‘1283017‘**
- **:rhbug:‘1278592‘**
- **:rhbug:‘1260421‘**
- **:rhbug:‘1278382‘**
- **:rhbug:‘1230820‘**
- **:rhbug:‘1280240‘**

8.44 1.1.4 Release Notes

API additions in 1.1.4:

- newly added `dnf.Query.duplicated()`
- extended `dnf.Query.latest()`

Bugs fixed in 1.1.4:

- **:rhbug:‘1278031‘**
- **:rhbug:‘1264032‘**
- **:rhbug:‘1209056‘**
- **:rhbug:‘1274946‘**

8.45 1.1.3 Release Notes

Now *dnf.Base.group_install()* is able to exclude mandatory packages of the group from transaction.

8.46 1.1.2 Release Notes

Implemented *–downloadonly* command line option.

Bugs fixed in 1.1.2:

- **:rhbug:‘1262082‘**
- **:rhbug:‘1250038‘**
- **:rhbug:‘1048433‘**
- **:rhbug:‘1259650‘**
- **:rhbug:‘1260198‘**
- **:rhbug:‘1259657‘**
- **:rhbug:‘1254982‘**
- **:rhbug:‘1261766‘**
- **:rhbug:‘1234491‘**
- **:rhbug:‘1256531‘**
- **:rhbug:‘1254687‘**
- **:rhbug:‘1261656‘**
- **:rhbug:‘1258364‘**

8.47 1.1.1 Release Notes

Implemented `dnf mark` *command*.

Bugs fixed in 1.1.1:

- **:rhbug:‘1249319‘**
- **:rhbug:‘1234763‘**
- **:rhbug:‘1242946‘**
- **:rhbug:‘1225225‘**
- **:rhbug:‘1254687‘**
- **:rhbug:‘1247766‘**
- **:rhbug:‘1125925‘**
- **:rhbug:‘1210289‘**

8.48 1.1.0 Release Notes

API additions in 1.1.0:

`dnf.Base.do_transaction()` now accepts multiple displays.

Introduced `install_weak_deps` *configuration* option.

Implemented `strict` *configuration* option.

API deprecations in 1.1.0:

- `dnf.callback.LoggingTransactionDisplay` is deprecated now. It was considered part of API despite the fact that it has never been documented. Use *`dnf.callback.TransactionProgress`* instead.

Bugs fixed in 1.1.0

- **:rhbug:‘1210445‘**
- **:rhbug:‘1218401‘**
- **:rhbug:‘1227952‘**
- **:rhbug:‘1197456‘**
- **:rhbug:‘1236310‘**
- **:rhbug:‘1219638‘**
- **:rhbug:‘1207981‘**
- **:rhbug:‘1208918‘**
- **:rhbug:‘1221635‘**
- **:rhbug:‘1236306‘**
- **:rhbug:‘1234639‘**
- **:rhbug:‘1244486‘**
- **:rhbug:‘1224248‘**
- **:rhbug:‘1243501‘**
- **:rhbug:‘1225237‘**

8.49 1.0.2 Release Notes

When a transaction is not successfully finished, DNF preserves downloaded packages until the next successful transaction even if `keepcache` option is set to `False`.

Maximum number of simultaneous package downloads can be adjusted by newly added `max_parallel_downloads` *configuration* option.

`--repofrompath` *command line argument* was introduced for temporary configuration of repositories.

API additions in 1.0.2:

Newly added package attributes: *`dnf.package.Package.obsoletes`*, *`dnf.package.Package.provides`* and *`dnf.package.Package.requires`*.

`dnf.package.Query.filter`’s keys `requires` and `provides` now accepts list of `Hawkey.Reldep` type.

Bugs fixed in 1.0.2:

- **:rhbug:‘1148630‘**
- **:rhbug:‘1176351‘**
- **:rhbug:‘1210445‘**
- **:rhbug:‘1173107‘**
- **:rhbug:‘1219199‘**
- **:rhbug:‘1220040‘**

- **:rhbug:‘1230975‘**
- **:rhbug:‘1232815‘**
- **:rhbug:‘1113384‘**
- **:rhbug:‘1133979‘**
- **:rhbug:‘1238958‘**
- **:rhbug:‘1238252‘**
- **:rhbug:‘1212320‘**

8.50 1.0.1 Release Notes

DNF follows the Semantic Versioning as defined at http://semver.org/.

Documented SSL *configuration* and *repository* options.

Added virtual provides allowing installation of DNF commands by their name in the form of `dnf install dnf-command(name)`.

dnf-automatic now by default waits random interval between 0 and 300 seconds before any network communication is performed.

Bugs fixed in 1.0.1:

- **:rhbug:‘1214968‘**
- **:rhbug:‘1222694‘**
- **:rhbug:‘1225246‘**
- **:rhbug:‘1213985‘**
- **:rhbug:‘1225277‘**
- **:rhbug:‘1223932‘**
- **:rhbug:‘1223614‘**
- **:rhbug:‘1203661‘**
- **:rhbug:‘1187741‘**

8.51 1.0.0 Release Notes

Improved documentation of YUM to DNF transition in *Changes in DNF CLI compared to YUM*.

Auto remove command does not remove *installonly* packages.

Downgrade command downgrades to specified package version if that is lower than currently installed one.

DNF now uses *dnf.repo.Repo.id* as a default value for `dnf.repo.Repo.name`.

Added support of repositories which use basic HTTP authentication.

API additions in 1.0.0:

configuration options *username* and *password* (HTTP authentication)

`dnf.repo.Repo.username` and `dnf.repo.Repo.password` (HTTP authentication)

Bugs fixed in 1.0.0:

- **:rhbug:‘1215560‘**
- **:rhbug:‘1199648‘**
- **:rhbug:‘1208773‘**
- **:rhbug:‘1208018‘**
- **:rhbug:‘1207861‘**
- **:rhbug:‘1201445‘**
- **:rhbug:‘1210275‘**
- **:rhbug:‘1191275‘**
- **:rhbug:‘1207965‘**
- **:rhbug:‘1215289‘**

8.52 0.6.5 Release Notes

Python 3 version of DNF is now default in Fedora 23 and later.

yum-dnf package does not conflict with yum package.

dnf erase was deprecated in favor of *dnf remove*.

Extended documentation of handling non-existent packages and YUM to DNF transition in *Changes in DNF CLI compared to YUM*.

API additions in 0.6.5:

Newly added *pluginconfpath* option in *configuration*.

Exposed *skip_if_unavailable* attribute from *Repository Configuration*.

Documented *IOError* exception of method *fill_sack* from `dnf.Base`.

Bugs fixed in 0.6.5:

- **:rhbug:‘1203151‘**
- **:rhbug:‘1187579‘**
- **:rhbug:‘1185977‘**
- **:rhbug:‘1195240‘**
- **:rhbug:‘1193914‘**
- **:rhbug:‘1195385‘**
- **:rhbug:‘1160806‘**
- **:rhbug:‘1186710‘**
- **:rhbug:‘1207726‘**
- **:rhbug:‘1157233‘**
- **:rhbug:‘1190671‘**
- **:rhbug:‘1191579‘**
- **:rhbug:‘1195325‘**

- **:rhbug:‘1154202‘**
- **:rhbug:‘1189083‘**
- **:rhbug:‘1193915‘**
- **:rhbug:‘1195661‘**
- **:rhbug:‘1190458‘**
- **:rhbug:‘1194685‘**
- **:rhbug:‘1160950‘**

8.53 0.6.4 Release Notes

Added example code snippets into *DNF Use Cases*.

Shows ordered groups/environments by *display_order* tag from *cli* and *Comps, or the Distribution Compose Metadata* DNF API.

In commands the environment group is specified the same as *group*.

skip_if_unavailable configuration option affects the metadata only.

added *enablegroups*, *minrate* and *timeout* *configuration options*

API additions in 0.6.4:

Documented *install_set* and *remove_set attributes* from *Transaction*.

Exposed *downloadsize*, *files*, *installsize* attributes from *Package*.

Bugs fixed in 0.6.4:

- **:rhbug:‘1155877‘**
- **:rhbug:‘1175466‘**
- **:rhbug:‘1175466‘**
- **:rhbug:‘1186461‘**
- **:rhbug:‘1170156‘**
- **:rhbug:‘1184943‘**
- **:rhbug:‘1177002‘**
- **:rhbug:‘1169165‘**
- **:rhbug:‘1167982‘**
- **:rhbug:‘1157233‘**
- **:rhbug:‘1138096‘**
- **:rhbug:‘1181189‘**
- **:rhbug:‘1181397‘**
- **:rhbug:‘1175434‘**
- **:rhbug:‘1162887‘**
- **:rhbug:‘1156084‘**
- **:rhbug:‘1175098‘**

- **:rhbug:‘1174136‘**
- **:rhbug:‘1055910‘**
- **:rhbug:‘1155918‘**
- **:rhbug:‘1119030‘**
- **:rhbug:‘1177394‘**
- **:rhbug:‘1154476‘**

8.54 0.6.3 Release Notes

Deltarpm configuration option is set on by default.

API additions in 0.6.3:

- dnf-automatic adds *motd emitter* as an alternative output

Bugs fixed in 0.6.3:

- **:rhbug:‘1153543‘**
- **:rhbug:‘1151231‘**
- **:rhbug:‘1163063‘**
- **:rhbug:‘1151854‘**
- **:rhbug:‘1151740‘**
- **:rhbug:‘1110780‘**
- **:rhbug:‘1149972‘**
- **:rhbug:‘1150474‘**
- **:rhbug:‘995537‘**
- **:rhbug:‘1149952‘**
- **:rhbug:‘1149350‘**
- **:rhbug:‘1170232‘**
- **:rhbug:‘1147523‘**
- **:rhbug:‘1148208‘**
- **:rhbug:‘1109927‘**

8.55 0.6.2 Release Notes

API additions in 0.6.2:

- Now `dnf.Base.package_install()` method ignores already installed packages
- *CliError* exception from `dnf.cli` documented
- *Autoerase*, *History*, *Info*, *List*, *Provides*, *Repolist* commands do not force a sync of expired *metadata*
- *Install* command does installation only

Bugs fixed in 0.6.2:

- **:rhbug:‘909856‘**
- **:rhbug:‘1134893‘**
- **:rhbug:‘1138700‘**
- **:rhbug:‘1070902‘**
- **:rhbug:‘1124316‘**
- **:rhbug:‘1136584‘**
- **:rhbug:‘1135861‘**
- **:rhbug:‘1136223‘**
- **:rhbug:‘1122617‘**
- **:rhbug:‘1133830‘**
- **:rhbug:‘1121184‘**

8.56 0.6.1 Release Notes

New release adds *upgrade-type command* to *dnf-automatic* for choosing specific advisory type updates.

Implemented missing *history redo command* for repeating transactions.

Supports *gpgkey* repo config, *repo_gpgcheck* and *gpgcheck* [main] and Repo configs.

Distributing new package *dnf-yum* that provides */usr/bin/yum* as a symlink to */usr/bin/dnf*.

API additions in 0.6.1:

- *exclude*, the third parameter of `dnf.Base.group_install()` now also accepts glob patterns of package names.

Bugs fixed in 0.6.1:

- **:rhbug:‘1132335‘**
- **:rhbug:‘1071854‘**
- **:rhbug:‘1131969‘**
- **:rhbug:‘908764‘**
- **:rhbug:‘1130878‘**
- **:rhbug:‘1130432‘**
- **:rhbug:‘1118236‘**
- **:rhbug:‘1109915‘**

8.57 0.6.0 Release Notes

0.6.0 marks a new minor version of DNF and the first release to support advisories listing with the *udpateinfo command*.

Support for the *include configuration directive* has been added. Its functionality reflects YUM’s `includepkgs` but it has been renamed to make it consistent with the `exclude` setting.

Group operations now produce a list of proposed marking changes to group objects and the user is given a chance to accept or reject them just like with an ordinary package transaction.

Bugs fixed in 0.6.0:

- **:rhbug:‘850912‘**
- **:rhbug:‘1055910‘**
- **:rhbug:‘1116666‘**
- **:rhbug:‘1118272‘**
- **:rhbug:‘1127206‘**

8.58 0.5.5 Release Notes

The full proxy configuration, API extensions and several bugfixes are provided in this release.

API changes in 0.5.5:

- *cachedir*, the second parameter of *dnf.repo.Repo.__init__()* is not optional (the method has always been this way but the documentation was not matching)

API additions in 0.5.5:

- extended description and an example provided for *dnf.Base.fill_sack()*
- `dnf.conf.Conf.proxy`
- `dnf.conf.Conf.proxy_username`
- `dnf.conf.Conf.proxy_password`
- `dnf.repo.Repo.proxy`
- `dnf.repo.Repo.proxy_username`
- `dnf.repo.Repo.proxy_password`

Bugs fixed in 0.5.5:

- **:rhbug:‘1100946‘**
- **:rhbug:‘1117789‘**
- **:rhbug:‘1120583‘**
- **:rhbug:‘1121280‘**
- **:rhbug:‘1122900‘**
- **:rhbug:‘1123688‘**

8.59 0.5.4 Release Notes

Several encodings bugs were fixed in this release, along with some packaging issues and updates to *DNF Configuration Reference*.

Repository *priority* configuration setting has been added, providing similar functionality to YUM Utils’ Priorities plugin.

Bugs fixed in 0.5.4:

- **:rhbug:‘1048973‘**
- **:rhbug:‘1108908‘**
- **:rhbug:‘1116544‘**
- **:rhbug:‘1116839‘**
- **:rhbug:‘1116845‘**
- **:rhbug:‘1117102‘**
- **:rhbug:‘1117293‘**
- **:rhbug:‘1117678‘**
- **:rhbug:‘1118178‘**
- **:rhbug:‘1118796‘**
- **:rhbug:‘1119032‘**

8.60 0.5.3 Release Notes

A set of bugfixes related to i18n and Unicode handling. There is a `-4/-6` switch and a corresponding *ip_resolve* configuration option (both known from YUM) to force DNS resolving of hosts to IPv4 or IPv6 addresses.

0.5.3 comes with several extensions and clarifications in the API: notably `Transaction` is introspectible now, *`Query.filter`* is more useful with new types of arguments and we've hopefully shed more light on how a client is expected to setup the configuration *`substitutions`*.

Finally, plugin authors can now use a new *`resolved()`* hook.

API changes in 0.5.3:

- extended description given for *`dnf.Base.fill_sack()`*
- `dnf.Base.select_group()` has been dropped as announced in *0.4.18 Release Notes*

API additions in 0.5.3:

- *`dnf.conf.Conf.substitutions`*
- *`dnf.package.Package.arch`*
- *`dnf.package.Package.buildtime`*
- *`dnf.package.Package.epoch`*
- *`dnf.package.Package.installtime`*
- *`dnf.package.Package.name`*
- *`dnf.package.Package.release`*
- *`dnf.package.Package.sourcerpm`*
- *`dnf.package.Package.version`*
- *`dnf.Plugin.resolved()`*
- *`dnf.query.Query.filter()`* accepts suffixes for its argument keys now which change the filter semantics.
- *`dnf.rpm`*
- `dnf.transaction.TransactionItem`

- `dnf.transaction.Transaction` is iterable now.

Bugs fixed in 0.5.3:

- **:rhbug:‘1047049‘**
- **:rhbug:‘1067156‘**
- **:rhbug:‘1093420‘**
- **:rhbug:‘1104757‘**
- **:rhbug:‘1105009‘**
- **:rhbug:‘1110800‘**
- **:rhbug:‘1111569‘**
- **:rhbug:‘1111997‘**
- **:rhbug:‘1112669‘**
- **:rhbug:‘1112704‘**

8.61 0.5.2 Release Notes

This release brings autoremove command that removes any package that was originally installed as a dependency (e.g. had not been specified as an explicit argument to the install command) and is no longer needed.

Enforced verification of SSL connections can now be disabled with the *sslverify setting*.

We have been plagued with many crashes related to Unicode and encodings since the 0.5.0 release. These have been cleared out now.

There’s more: improvement in startup time, extended globbing semantics for input arguments and better search relevance sorting.

Bugs fixed in 0.5.2:

- **:rhbug:‘963345‘**
- **:rhbug:‘1073457‘**
- **:rhbug:‘1076045‘**
- **:rhbug:‘1083679‘**
- **:rhbug:‘1092006‘**
- **:rhbug:‘1092777‘**
- **:rhbug:‘1093888‘**
- **:rhbug:‘1094594‘**
- **:rhbug:‘1095580‘**
- **:rhbug:‘1095861‘**
- **:rhbug:‘1096506‘**

8.62 0.5.1 Release Notes

Bugfix release with several internal cleanups. One outstanding change for CLI users is that DNF is a lot less verbose now during the dependency resolving phase.

Bugs fixed in 0.5.1:

- **:rhbug:‘1065882‘**
- **:rhbug:‘1081753‘**
- **:rhbug:‘1089864‘**

8.63 0.5.0 Release Notes

The biggest improvement in 0.5.0 is complete support for groups and environments, including internal database of installed groups independent of the actual packages (concept known as groups-as-objects from YUM). Upgrading groups is supported now with `group upgrade` too.

To force refreshing of metadata before an operation (even if the data is not expired yet), the refresh option has been added.

Internally, the CLI went through several changes to allow for better API accessibility like granular requesting of root permissions.

API has got many more extensions, focusing on better manipulation with comps and packages. There are new entries in *Changes in DNF CLI compared to YUM* and *DNF User's FAQ* too.

Several resource leaks (file descriptors, noncollectable Python objects) were found and fixed.

API changes in 0.5.0:

- it is now recommended that either *dnf.Base.close()* is used, or that *dnf.Base* instances are treated as a context manager.

API extensions in 0.5.0:

- *dnf.Base.add_remote_rpms()*
- *dnf.Base.close()*
- *dnf.Base.group_upgrade()*
- *dnf.Base.resolve()* optionally accepts *allow_erasing* arguments now.
- *dnf.Base.package_downgrade()*
- *dnf.Base.package_install()*
- *dnf.Base.package_upgrade()*
- dnf.cli.demand.DemandSheet
- *dnf.cli.Command.base*
- *dnf.cli.Command.cli*
- *dnf.cli.Command.summary*
- dnf.cli.Command.usage
- *dnf.cli.Command.configure()*
- *dnf.cli.Cli.demands*

- *dnf.comps.Package*
- *dnf.comps.Group.packages_iter()*
- *dnf.comps.MANDATORY* etc.

Bugs fixed in 0.5.0:

- **:rhbug:‘1029022‘**
- **:rhbug:‘1051869‘**
- **:rhbug:‘1061780‘**
- **:rhbug:‘1062884‘**
- **:rhbug:‘1062889‘**
- **:rhbug:‘1063666‘**
- **:rhbug:‘1064211‘**
- **:rhbug:‘1064226‘**
- **:rhbug:‘1073859‘**
- **:rhbug:‘1076884‘**
- **:rhbug:‘1079519‘**
- **:rhbug:‘1079932‘**
- **:rhbug:‘1080331‘**
- **:rhbug:‘1080489‘**
- **:rhbug:‘1082230‘**
- **:rhbug:‘1083432‘**
- **:rhbug:‘1083767‘**
- **:rhbug:‘1084139‘**
- **:rhbug:‘1084553‘**
- **:rhbug:‘1088166‘**

8.64 0.4.19 Release Notes

Arriving one week after 0.4.18, the 0.4.19 mainly provides a fix to a traceback in group operations under non-root users.

DNF starts to ship separate translation files (.mo) starting with this release.

Bugs fixed in 0.4.19:

- **:rhbug:‘1077173‘**
- **:rhbug:‘1078832‘**
- **:rhbug:‘1079621‘**

8.65 0.4.18 Release Notes

Support for `dnf distro-sync <spec>` finally arrives in this version.

DNF has moved to handling groups as objects, tagged installed/uninstalled independently from the actual installed packages. This has been in YUM as the `group_command=objects` setting and the default in recent Fedora releases. There are API extensions related to this change as well as two new CLI commands: `group mark install` and `group mark remove`.

API items deprecated in 0.4.8 and 0.4.9 have been dropped in 0.4.18, in accordance with our deprecating-label.

API changes in 0.4.18:

- `dnf.queries` has been dropped as announced in *0.4.8 Release Notes*
- `dnf.exceptions.PackageNotFoundError` has been dropped from API as announced in *0.4.9 Release Notes*
- *`dnf.Base.install()`* no longer has to return the number of marked packages as announced in *0.4.9 Release Notes*

API deprecations in 0.4.18:

- `dnf.Base.select_group()` is deprecated now. Please use *`group_install()`* instead.

API additions in 0.4.18:

- *`dnf.Base.group_install()`*
- *`dnf.Base.group_remove()`*

Bugs fixed in 0.4.18:

- **:rhbug:‘963710‘**
- **:rhbug:‘1067136‘**
- **:rhbug:‘1071212‘**
- **:rhbug:‘1071501‘**

8.66 0.4.17 Release Notes

This release fixes many bugs in the downloads/DRPM CLI area. A bug got fixed preventing a regular user from running read-only operations using `--cacheonly`. Another fix ensures that `metadata_expire=never` setting is respected. Lastly, the release provides three requested API calls in the repo management area.

API additions in 0.4.17:

- *`dnf.repodict.RepoDict.all()`*
- *`dnf.repodict.RepoDict.get_matching()`*
- *`dnf.repo.Repo.set_progress_bar()`*

Bugs fixed in 0.4.17:

- **:rhbug:‘1059704‘**
- **:rhbug:‘1058224‘**
- **:rhbug:‘1069538‘**
- **:rhbug:‘1070598‘**

- **:rhbug:‘1070710‘**
- **:rhbug:‘1071323‘**
- **:rhbug:‘1071455‘**
- **:rhbug:‘1071501‘**
- **:rhbug:‘1071518‘**
- **:rhbug:‘1071677‘**

8.67 0.4.16 Release Notes

The refactorings from 0.4.15 are introducing breakage causing the background `dnf makecache` runs traceback. This release fixes that.

Bugs fixed in 0.4.16:

- **:rhbug:‘1069996‘**

8.68 0.4.15 Release Notes

Massive refactoring of the downloads handling to provide better API for reporting download progress and fixed bugs are the main things brought in 0.4.15.

API additions in 0.4.15:

- *dnf.exceptions.DownloadError*
- *dnf.Base.download_packages()* now takes the optional *progress* parameter and can raise *DownloadError*.
- *dnf.callback.Payload*
- *dnf.callback.DownloadProgress*
- *dnf.query.Query.filter()* now also recognizes `provides` as a filter name.

Bugs fixed in 0.4.15:

- **:rhbug:‘1048788‘**
- **:rhbug:‘1065728‘**
- **:rhbug:‘1065879‘**
- **:rhbug:‘1065959‘**
- **:rhbug:‘1066743‘**

8.69 0.4.14 Release Notes

This quickly follows 0.4.13 to address the issue of crashes when DNF output is piped into another program.

API additions in 0.4.14:

- *Repo.pkgdir*

Bugs fixed in 0.4.14:

- **:rhbug:‘1062390‘**
- **:rhbug:‘1062847‘**
- **:rhbug:‘1063022‘**
- **:rhbug:‘1064148‘**

8.70 0.4.13 Release Notes

0.4.13 finally ships support for delta RPMS. Enabling this can save some bandwidth (and use some CPU time) when downloading packages for updates.

Support for bash completion is also included in this version. It is recommended to use the `generate_completion_cache` plugin to have the completion work fast. This plugin will be also shipped with `dnf-plugins-core-0.0.3`.

The *keepcache* config option has been readded.

Bugs fixed in 0.4.13:

- **:rhbug:‘909468‘**
- **:rhbug:‘1030440‘**
- **:rhbug:‘1046244‘**
- **:rhbug:‘1055051‘**
- **:rhbug:‘1056400‘**

8.71 0.4.12 Release Notes

This release disables fastestmirror by default as we received many complains about it. There are also several bugfixes, most importantly an issue has been fixed that caused packages installed by Anaconda be removed together with a depending package. It is now possible to use `bandwidth` and `throttle` config values too.

Bugs fixed in 0.4.12:

- **:rhbug:‘1045737‘**
- **:rhbug:‘1048468‘**
- **:rhbug:‘1048488‘**
- **:rhbug:‘1049025‘**
- **:rhbug:‘1051554‘**

8.72 0.4.11 Release Notes

This is mostly a bugfix release following quickly after 0.4.10, with many updates to documentation.

API additions in 0.4.11:

- *Plugin.read_config()*
- *repo.Metadata*

- *repo.Repo.metadata*

API changes in 0.4.11:

- `Conf.pluginpath` is no longer hard coded but depends on the major Python version.

Bugs fixed in 0.4.11:

- **:rhbug:'1048402'**
- **:rhbug:'1048572'**
- **:rhbug:'1048716'**
- **:rhbug:'1048719'**
- **:rhbug:'1048988'**

8.73 0.4.10 Release Notes

0.4.10 is a bugfix release that also adds some long-requested CLI features and extends the plugin support with two new plugin hooks. An important feature for plugin developers is going to be the possibility to register plugin's own CLI command, available from this version.

`dnf history` now recognizes `last` as a special argument, just like other history commands.

`dnf install` now accepts group specifications via the `@` character.

Support for the `--setopt` option has been readded from YUM.

API additions in 0.4.10:

- *Command Line Interface Hooks*
- *Plugin.name*
- *Plugin.__init__()* now specifies the second parameter as an instance of *.cli.Cli*
- *Plugin.sack()*
- *Plugin.transaction()*
- *repo.repo_id_invalid()*

API changes in 0.4.10:

- Plugin authors must specify *Plugin.name* when authoring a plugin.

Bugs fixed in 0.4.10:

- **:rhbug:'967264'**
- **:rhbug:'1018284'**
- **:rhbug:'1035164'**
- **:rhbug:'1036147'**
- **:rhbug:'1036211'**
- **:rhbug:'1038403'**
- **:rhbug:'1038937'**
- **:rhbug:'1040255'**
- **:rhbug:'1044502'**

- **:rhbug:‘1044981‘**
- **:rhbug:‘1044999‘**

8.74 0.4.9 Release Notes

Several YUM features are revived in this release. `dnf history rollback` now works again. The `history userinstalled` has been added, it displays a list of packages that the user manually selected for installation on an installed system and does not include those packages that got installed as dependencies.

We’re happy to announce that the API in 0.4.9 has been extended to finally support plugins. There is a limited set of plugin hooks now, we will carefully add new ones in the following releases. New marking operations have ben added to the API and also some configuration options.

An alternative to `yum shell` is provided now for its most common use case: replacing a non-leaf package with a conflicting package is achieved by using the `--allowerasing` switch now.

API additions in 0.4.9:

- *Plugin Interface*
- *Logging*
- *`Base.read_all_repos()`*
- *`Base.reset()`*
- *`Base.downgrade()`*
- *`Base.remove()`*
- *`Base.upgrade()`*
- *`Base.upgrade_all()`*
- `Conf.pluginpath`
- `Conf.reposdir`

API deprecations in 0.4.9:

- `PackageNotFoundError` is deprecated for public use. Please catch *`MarkingError`* instead.
- It is deprecated to use *`Base.install()`* return value for anything. The method either returns or raises an exception.

Bugs fixed in 0.4.9:

- **:rhbug:‘884615‘**
- **:rhbug:‘963137‘**
- **:rhbug:‘991038‘**
- **:rhbug:‘1032455‘**
- **:rhbug:‘1034607‘**
- **:rhbug:‘1036116‘**

8.75 0.4.8 Release Notes

There are mainly internal changes, new API functions and bugfixes in this release.

Python 3 is fully supported now, the Fedora builds include the Py3 variant. The DNF program still runs under Python 2.7 but the extension authors can now choose what Python they prefer to use.

This is the first version of DNF that deprecates some of its API. Clients using deprecated code will see a message emitted to stderr using the standard Python warnings module. You can filter out *dnf.exceptions.DeprecationWarning* to suppress them.

API additions in 0.4.8:

- *dnf.Base.sack*
- dnf.conf.Conf.cachedir
- dnf.conf.Conf.config_file_path
- dnf.conf.Conf.persistdir
- *dnf.conf.Conf.read()*
- *dnf.package.Package*
- *dnf.query.Query*
- *dnf.subject.Subject*
- *dnf.repo.Repo.__init__()*
- *dnf.sack.Sack*
- *dnf.selector.Selector*
- dnf.transaction.Transaction

API deprecations in 0.4.8:

- dnf.queries is deprecated now. If you need to create instances of *Subject*, import it from *dnf.subject*. To create *Query* instances it is recommended to use *sack.query()*.

Bugs fixed in 0.4.8:

- **:rhbug:‘1014563‘**
- **:rhbug:‘1029948‘**
- **:rhbug:‘1030998‘**
- **:rhbug:‘1030297‘**
- **:rhbug:‘1030980‘**

8.76 0.4.7 Release Notes

We start to publish the *DNF API Reference* with this release. It is largely incomprehensive at the moment, yet outlines the shape of the documentation and the process the project is going to use to maintain it.

The :ref:upgrade_requirements_on_install <upgrade_requirements_on_install_dropped> configuration option was dropped.

Bugs fixed in 0.4.7:

- **:rhbug:‘1019170‘**

- **:rhbug:‘1024776‘**
- **:rhbug:‘1025650‘**

8.77 0.4.6 Release Notes

0.4.6 brings two new major features. Firstly, it is the revival of `history undo`, so transactions can be reverted now. Secondly, DNF will now limit the number of installed kernels and *installonly* packages in general to the number specified by *installonly_limit* configuration option.

DNF now supports the `group summary` command and one-word group commands no longer cause tracebacks, e.g. `dnf grouplist`.

There are vast internal changes to `dnf.cli`, the subpackage that provides CLI to DNF. In particular, it is now better separated from the core.

The hawkey library used against DNF from with this versions uses a recent RPMDB loading optimization in libsolv that shortens DNF startup by seconds when the cached RPMDB is invalid.

We have also added further fixes to support Python 3 and enabled librepo's fastestmirror caching optimization to tighten the download times even more.

Bugs fixed in 0.4.6:

- **:rhbug:‘878348‘**
- **:rhbug:‘880524‘**
- **:rhbug:‘1019957‘**
- **:rhbug:‘1020101‘**
- **:rhbug:‘1020934‘**
- **:rhbug:‘1023486‘**

8.78 0.4.5 Release Notes

A serious bug causing tracebacks during package downloads made it into 0.4.4 and this release contains a fix for that. Also, a basic proxy support has been readded now.

Bugs fixed in 0.4.5:

- **:rhbug:‘1021087‘**

8.79 0.4.4 Release Notes

The initial support for Python 3 in DNF has been merged in this version. In practice one can not yet run the `dnf` command in Py3 but the unit tests already pass there. We expect to give Py3 and DNF heavy testing during the Fedora 21 development cycle and eventually switch to it as the default. The plan is to drop Python 2 support as soon as Anaconda is running in Python 3.

Minor adjustments to allow Anaconda support also happened during the last week, as well as a fix to a possibly severe bug that one is however not really likely to see with non-devel Fedora repos:

- **:rhbug:‘1017278‘**

8.80 0.4.3 Release Notes

This is an early release to get the latest DNF out with the latest librepo fixing the Too many open files bug.

In Fedora, the spec file has been updated to no longer depend on precise versions of the libraries so in the future they can be released independently.

This release sees the finished refactoring in error handling during basic operations and adds support for `group remove` and `group info` commands, i.e. the following two bugs:

- **:rhbug:‘1013764‘**
- **:rhbug:‘1013773‘**

8.81 0.4.2 Release Notes

DNF now downloads packages for the transaction in parallel with progress bars updated to effectively represent this. Since so many things in the downloading code were changing, we figured it was a good idea to finally drop urlgrabber dependency at the same time. Indeed, this is the first version that doesn't require urlgrabber for neither build nor run.

Similarly, since librepo started to support this, downloads in DNF now use the fastest mirrors available by default.

The option to *specify repositories' costs* has been readded.

Internally, DNF has seen first part of ongoing refactorings of the basic operations (install, update) as well as a couple of new API methods supporting development of extensions.

These bugzillas are fixed in 0.4.2:

- **:rhbug:‘909744‘**
- **:rhbug:‘984529‘**
- **:rhbug:‘967798‘**
- **:rhbug:‘995459‘**

8.82 0.4.1 Release Notes

The focus of this release was to support our efforts in implementing the DNF Payload for Anaconda, with changes on the API side of things (better logging, new `Base.reset()` method).

Support for some irrelevant config options has been dropped (`kernelpkgnames`, `exactarch`, `rpm_check_debug`). We also no longer detect metalinks in the `mirrorlist` option (see Fedora bug 948788).

DNF is on its way to drop the urlgrabber dependency and the first set of patches towards this goal is already in.

Expect the following bugs to go away with upgrade to 0.4.1:

- **:rhbug:‘998859‘**
- **:rhbug:‘1006366‘**
- **:rhbug:‘1008444‘**
- **:rhbug:‘1003220‘**

8.83 0.4.0 Release Notes

The new minor version brings many internal changes to the comps code, most comps parsing and processing is now delegated to libcomps by Jindřich Luža.

The `overwrite_groups` config option has been dropped in this version and DNF acts if it was 0, that is groups with the same name are merged together.

The currently supported groups commands (`group list` and `group install`) are documented on the manpage now.

The 0.4.0 version is the first one supported by the DNF Payload for Anaconda and many changes since 0.3.11 make that possible by cleaning up the API and making it more sane (cleanup of `yumvars` initialization API, unifying the RPM transaction callback objects hierarchy, slimming down `dnf.rpmUtils.arch`, improved logging).

Fixes for the following are contained in this version:

- **:rhbug:‘997403‘**
- **:rhbug:‘1002508‘**
- **:rhbug:‘1002798‘**

8.84 0.3.11 Release Notes

The default multilib policy configuration value is `best` now. This does not pose any change for the Fedora users because exactly the same default had been previously achieved by a setting in `/etc/dnf/dnf.conf` shipped with the Fedora package.

An important fix to the repo module speeds up package downloads again is present in this release. The full list of fixes is:

- **:rhbug:‘979042‘**
- **:rhbug:‘977753‘**
- **:rhbug:‘996138‘**
- **:rhbug:‘993916‘**

8.85 0.3.10 Release Notes

The only major change is that `skip_if_unavailable` is *enabled by default now*.

A minor release otherwise, mainly to get a new version of DNF out that uses a fresh librepo. The following issues are now a thing of the past:

- **:rhbug:‘977661‘**
- **:rhbug:‘984483‘**
- **:rhbug:‘986545‘**

8.86 0.3.9 Release Notes

This is a quick bugfix release dealing with reported bugs and tracebacks:

- **:rhbug:‘964584‘**
- **:rhbug:‘979942‘**
- **:rhbug:‘980227‘**
- **:rhbug:‘981310‘**

8.87 0.3.8 Release Notes

A new locking module has been integrated in this version, clients should see the message about DNF lock being taken less often.

Panu Matilainen has submitted many patches to this release to cleanup the RPM interfacing modules.

The following bugs are fixed in this release:

- **:rhbug:‘908491‘**
- **:rhbug:‘968159‘**
- **:rhbug:‘974427‘**
- **:rhbug:‘974866‘**
- **:rhbug:‘976652‘**
- **:rhbug:‘975858‘**

8.88 0.3.7 Release Notes

This is a bugfix release:

- **:rhbug:‘916662‘**
- **:rhbug:‘967732‘**

8.89 0.3.6 Release Notes

This is a bugfix release, including the following fixes:

- **:rhbug:‘966372‘**
- **:rhbug:‘965410‘**
- **:rhbug:‘963627‘**
- **:rhbug:‘965114‘**
- **:rhbug:‘964467‘**
- **:rhbug:‘963680‘**
- **:rhbug:‘963133‘**

8.90 0.3.5 Release Notes

Besides few fixed bugs this version should not present any differences for the user. On the inside, the transaction managing mechanisms have changed drastically, bringing code simplification, better maintainability and better testability.

In Fedora, there is a change in the spec file effectively preventing the makecache timer from running *immediately after installation*. The timer service is still enabled by default, but unless the user starts it manually with `systemctl start dnf-makecache.timer` it will not run until after the first reboot. This is in alignment with Fedora packaging best practices.

The following bugfixes are included in 0.3.5:

- **:rhbug:'958452'**
- **:rhbug:'959990'**
- **:rhbug:'961549'**
- **:rhbug:'962188'**

8.91 0.3.4 Release Notes

0.3.4 is the first DNF version since the fork from YUM that is able to manipulate the comps data. In practice, `dnf group install <group name>` works again. No other group commands are supported yet.

Support for `librepo-0.0.4` and related cleanups and extensions this new version allows are included (see the buglist below)

This version has also improved reporting of obsoleted packages in the CLI (the YUM-style "replacing <package-nevra>" appears in the textual transaction overview).

The following bugfixes are included in 0.3.4:

- **:rhbug:'887317'**
- **:rhbug:'914919'**
- **:rhbug:'922667'**

8.92 0.3.3 Release Notes

The improvements in 0.3.3 are only API changes to the logging. There is a new module `dnf.logging` that defines simplified logging structure compared to YUM, with fewer logging levels and simpler usage for the developers. The RPM transaction logs are no longer in `/var/log/dnf.transaction.log` but in `/var/log/dnf.rpm.log` by default.

The exception classes were simplified and moved to `dnf.exceptions`.

The following bugs are fixed in 0.3.3:

- **:rhbug:'950722'**
- **:rhbug:'903775'**

8.93 0.3.2 Release Notes

The major improvement in this version is in speeding up syncing of repositories using metalink by looking at the repomd.xml checksums. This effectively lets DNF cheaply refresh expired repositories in cases where the original has not changed: for instance the main Fedora repository is refreshed with one 30 kB HTTP download. This functionality is present in the current YUM but hasn't worked in DNF since 3.0.0.

Otherwise this is mainly a release fixing bugs and tracebacks. The following reported bugs are fixed:

- **:rhbug:'947258'**
- **:rhbug:'889202'**
- **:rhbug:'923384'**

8.94 0.3.1 Release Notes

0.3.1 brings mainly changes to the automatic metadata synchronization. In Fedora, `dnf makecache` is triggered via SystemD timers now and takes an optional `background` extra-argument to run in resource-considerate mode (no syncing when running on laptop battery, only actually performing the check at most once every three hours). Also, the IO and CPU priorities of the timer-triggered process are lowered now and shouldn't as noticeably impact the system's performance.

The administrator can also easily disable the automatic metadata updates by setting *metadata_timer_sync* to 0.

The default value of *metadata_expire* was increased from 6 hours to 48 hours. In Fedora, the repos usually set this explicitly so this change is not going to cause much impact.

The following reported issues are fixed in this release:

- **:rhbug:'916657'**
- **:rhbug:'921294'**
- **:rhbug:'922521'**
- **:rhbug:'926871'**
- **:rhbug:'878826'**
- **:rhbug:'922664'**
- **:rhbug:'892064'**
- **:rhbug:'919769'**

CHAPTER 9

Changes in DNF CLI compared to YUM

9.1 `--skip-broken`

For install command:

The `--skip-broken` option is an alias for `--setopt=strict=0`. Both options could be used with DNF to skip all unavailable packages or packages with broken dependencies given to DNF without raising an error causing the whole operation to fail. This behavior can be set as default in dnf.conf file. See *strict conf option*.

For upgrade command:

The semantics that were supposed to trigger in YUM with `--skip-broken` are now set for plain `dnf update` as a default. There is no need to use `--skip-broken` with the `dnf upgrade` command. To use only the latest versions of packages in transactions, there is the `--best` command line switch.

9.2 Update and Upgrade Commands are the Same

Invoking `dnf update` or `dnf upgrade`, in all their forms, has the same effect in DNF, with the latter being preferred. In YUM `yum upgrade` was exactly like `yum --obsoletes update`.

9.3 `clean_requirements_on_remove` on by default

The *clean_requirements_on_remove* switch is on by default in DNF. It can thus be confusing to compare the "remove" operation results between DNF and YUM as by default DNF is often going to remove more packages.

9.4 No `resolvedep` command

The YUM version of this command is maintained for legacy reasons only. The user can just use `dnf provides` to find out what package provides a particular file.

9.5 No `deplist` command

An alternative to the YUM `deplist` command to find out dependencies of a package is `dnf repoquery --deplist` using *repoquery command*.

Note: Alternatively there is a YUM compatibility support where `yum deplist` is alias for `dnf repoquery --deplist` command

9.6 Excludes and repo excludes apply to all operations

YUM only respects excludes during installs and upgrades. DNF extends this to all operations, among others erasing and listing. If you e.g. want to see a list of all installed `python-f*` packages but not any of the Flask packages, the following will work:

```
dnf -x '*flask*' list installed 'python-f*'
```

9.7 YUM's conf directive `includepkgs` is just `include`

`include` directive name of [main] and Repo configuration is a more logical and better named counterpart of `exclude` in DNF.

9.8 The `include` option has been removed

Inclusion of other configuration files in the main configuration file is no longer supported.

9.9 `dnf provides /bin/<file>` is not fully supported

After UsrMove there's no directory `/bin` on Fedora systems and no files get installed there, `/bin` is only a symlink created by the `filesystem` package to point to `/usr/bin`. Resolving the symlinks to their real path would only give the user a false sense that this works, while in fact provides requests using globs such as:

```
dnf provides /b*/<file>
```

will fail still (as they do in YUM now). To find what provides a particular binary, use the actual path for binaries on Fedora:

```
dnf provides /usr/bin/<file>
```

Also see related Fedora bugzillas 982947 and 982664.

9.10 `skip_if_unavailable` could be enabled by default

In some distributions DNF is shipped with `skip_if_unavailable=True` in the *DNF configuration file*. The reason for the change is that third-party repositories can often be unavailable. Without this setting in the relevant

repository configuration file YUM immediately stops on a repository synchronization error, confusing and bothering the user.

See the related Fedora bug 984483.

9.11 `overwrite_groups` dropped, comps functions acting as if always disabled

This config option has been dropped. When DNF sees several groups with the same group ID it merges the groups' contents together.

9.12 `mirrorlist_expire` dropped

To simplify things for the user, DNF uses `metadata_expire` for both expiring metadata and the mirrorlist file (which is a kind of metadata itself).

9.13 metalink not recognized in the `mirrorlist` repo option

The following part of `yum.conf(5)` no longer applies for the `mirrorlist` option:

> As a special hack if the mirrorlist URL contains the word "metalink" then the value of mirrorlist is copied to metalink (if metalink is not set).

The relevant repository configuration files have been fixed to respect this, see the related Fedora bug 948788.

9.14 `alwaysprompt` dropped

Unsupported to simplify the configuration.

9.15 `upgrade_requirements_on_install` dropped

Dropping this config option with blurry semantics simplifies the configuration. DNF behaves as if this was disabled. If the user wanted to upgrade everything to the latest version she'd simply use `dnf upgrade`.

9.16 `dnf history rollback` check dropped

Since DNF tolerates the use of other package managers, it is possible that not all changes to the RPMDB are stored in the history of transactions. Therefore, DNF does not fail if such a situation is encountered and thus the `force` option is not needed anymore.

9.17 Packages replacement without `yum swap`

Time after time one needs to remove an installed package and replace it with a different one, providing the same capabilities while other packages depending on these capabilities stay installed. Without (transiently) breaking consistency of the package database this can be done by performing the remove and the install in one transaction. The common way to set up such a transaction in DNF is to use `dnf shell` or use the `--allowerasing` switch.

E.g. say you want to replace `A` (providing `P`) with B (also providing `P`, conflicting with `A`) without deleting `C` (which requires `P`) in the process. Use:

```
dnf --allowerasing install B
```

This command is equal to `yum swap A B`.

DNF provides swap command but only `dnf swap A B` syntax is supported

9.18 Dependency processing details are not shown in the CLI

During its depsolving phase, YUM outputs lines similar to:

```
---> Package rubygem-rhc.noarch 0:1.16.9-1.fc19 will be an update
--> Processing Dependency: rubygem-net-ssh-multi >= 1.2.0 for package: rubygem-rhc-1.
↪16.9-1.fc19.noarch
```

DNF does not output information like this. The technical reason is that depsolver below DNF always considers all dependencies for update candidates and the output would be very long. Secondly, even in YUM this output gets confusing very quickly especially for large transactions and so does more harm than good.

See the related Fedora bug 1044999.

9.19 `dnf provides` complies with the YUM documentation of the command

When one executes:

```
yum provides sandbox
```

YUM applies extra heuristics to determine what the user meant by `sandbox`, for instance it sequentially prepends entries from the `PATH` environment variable to it to see if it matches a file provided by some package. This is an undocumented behavior that DNF does not emulate. Just typically use:

```
dnf provides /usr/bin/sandbox
```

or even:

```
dnf provides '*/sandbox'
```

to obtain similar results.

9.20 Bandwidth limiting

DNF supports the `throttle` and `bandwidth` options familiar from YUM. Contrary to YUM, when multiple downloads run simultaneously the total downloading speed is throttled. This was not possible in YUM since downloaders ran in different processes.

9.21 `installonlypkgs` config option

Compared to YUM, DNF appends list values from the `installonlypkgs` config option to DNF defaults, where YUM overwrites the defaults by option values.

9.22 The usage of Delta RPM files

The boolean `deltarpm` option controls whether delta RPM files are used. Compared to YUM, DNF does not support `deltarpm_percentage` and instead chooses some optimal value of DRPM/RPM ratio to decide whether using deltarpm makes sense in the given case.

9.23 Handling .srpm files and non-existent packages

DNF will terminate early with an error if a command is executed requesting an installing operation on a local `.srpm` file:

```
$ dnf install fdn-0.4.17-1.fc20.src.rpm tour-4-6.noarch.rpm
Error: Will not install a source rpm package (fdn-0.4.17-1.fc20.src).
```

The same applies for package specifications that do not match any available package.

YUM will only issue a warning in this case and continue installing the "tour" package. The rationale behind the result in DNF is that a program should terminate with an error if it can not fulfill the CLI command in its entirety.

9.24 Promoting package to install to a package that obsoletes it

DNF will not magically replace a request for installing package `X` to installing package `Y` if `Y` obsoletes `X`. YUM does this if its `obsoletes` config option is enabled but the behavior is not properly documented and can be harmful.

See the related Fedora bug 1096506 and guidelines for renaming and obsoleting packages in Fedora.

9.25 Behavior of `--installroot` option

DNF offers more predictable behavior of installroot. DNF handles the path differently from the `--config` command-line option, where this path is always related to the host system (YUM combines this path with installroot). Reposdir is also handled slightly differently, if one path of the reposdirs exists inside of installroot, then repos are strictly taken from installroot (YUM tests each path from reposdir separately and use installroot path if existed). See the detailed description for *--installroot* option.

9.26 Different prompt after transaction table

DNF doesn't provide download functionality after displaying transaction table. It only asks user whether to continue with transaction or not. If one wants to download packages, they can use the 'download' command.

9.27 List command shows all repo alternatives

DNF lists all packages from all repos, which means there can be duplicates package names (with different repo name). This is due to providing users possibility to choose preferred repo.

9.28 `yum-langpacks` subcommands have been removed

Translations became part of core DNF and it is no longer necessary to manage individual language packs.

Following sub-commands were removed:

- langavailable
- langinstall
- langremove
- langlist
- langinfo

CHAPTER 10

Changes in DNF plugins compared to YUM plugins

Original YUM tool	DNF command/option	Package
`yum check`	*dnf repoquery* `--unsatisfied`	`dnf`
`yum-langpacks`		`dnf`
`yum-plugin-aliases`	*dnf alias*	`dnf`
`yum-plugin-auto-update-debuginfo`	`--setopt` in `debuginfo-install.conf`	`dnf-plugins-core`
`yum-plugin-changelog`		`dnf-plugins-core`
`yum-plugin-copr`	dnf copr	`dnf-plugins-core`
`yum-plugin-fastestmirror`	`fastestmirror` option in dnf.conf	`dnf`
`yum-plugin-fs-snapshot`		`dnf-plugins-extras-snapper`
`yum-plugin-local`		`dnf-plugins-core`
`yum-plugin-merge-conf`		`dnf-plugins-extras-rpmconf`
`yum-plugin-priorities`	`priority` option in dnf.conf	`dnf`
`yum-plugin-remove-with-leaves`	*dnf autoremove*	`dnf`
`yum-plugin-show-leaves`		`dnf-plugins-core`
`yum-plugin-tmprepo`	`--repofrompath` option	`dnf`
`yum-plugin-tsflags`	`tsflags` option in dnf.conf	`dnf`
`yum-plugin-versionlock`		`python3-dnf-plugin-versionlock`
`yum-rhn-plugin`		`dnf-plugin-spacewalk`

Plugins that have not been ported yet:

`yum-plugin-filter-data`, `yum-plugin-keys`, `yum-plugin-list-data`, `yum-plugin-post-transaction-actions`, `yum-plugin-protectbase`, `yum-plugin-ps`, `yum-plugin-puppetverify`, `yum-plugin-refresh-updatesd`, `yum-plugin-rpm-warm-cache`, `yum-plugin-upgrade-helper`, `yum-plugin-verify`

Feel free to file an RFE for missing functionality if you need it.

CHAPTER 11

Changes in DNF plugins compared to YUM utilities

All ported YUM tools are now implemented as DNF plugins.

Original YUM tool	New DNF command	Package
`debuginfo-install`	dnf debuginfo-install	`dnf-plugins-core`
`find-repos-of-install`	dnf list installed	`dnf`
`needs-restarting`	dnf tracer	`dnf-plugins-extras-tracer`
`package-cleanup`	*dnf list*, *dnf repoquery*	`dnf`, `dnf-plugins-core`
`repoclosure`	dnf repoclosure	`dnf-plugins-extras-repoclosure`
`repodiff`	dnf repodiff	`dnf-plugins-core`
`repo-graph`	dnf repograph	`dnf-plugins-extras-repograph`
`repomanage`	dnf repomanage	`dnf-plugins-extras-repomanage`
`repoquery`	*dnf repoquery*	`dnf`
`reposync`	dnf reposync	`dnf-plugins-core`
`repotrack`	dnf download –resolve –alldeps	`dnf-plugins-core`
`yum-builddep`	dnf builddep	`dnf-plugins-core`
`yum-config-manager`	dnf config-manager	`dnf-plugins-core`
`yum-debug-dump`	dnf debug-dump	`dnf-plugins-extras-debug`
`yum-debug-restore`	dnf debug-restore	`dnf-plugins-extras-debug`
`yumdownloader`	dnf download	`dnf-plugins-core`

Detailed table for `package-cleanup` replacement:

`package-cleanup --dupes`	`dnf repoquery --duplicates`
`package-cleanup --leaves`	`dnf repoquery --unneeded`
`package-cleanup --orphans`	`dnf repoquery --extras`
`package-cleanup --problems`	`dnf repoquery --unsatisfied`
`package-cleanup --cleandupes`	`dnf remove --duplicates`
`package-cleanup --oldkernels`	`dnf remove --oldinstallonly`
`package-cleanup --oldkernels --keep=2`	`dnf remove $(dnf repoquery --installonly --latest-limit=-2)`

11.1 yum-updateonboot and yum-cron

DNF does not have a direct replacement of yum-updateonboot and yum-cron commands. However, the similar result can be achieved by `dnf automatic` command (see *DNF Automatic*).

You can either use the shortcut:

```
$ systemctl enable --now dnf-automatic-install.timer
```

Or set `apply_updates` option of `/etc/dnf/automatic.conf` to True and use generic timer unit:

```
$ systemctl enable --now dnf-automatic.timer
```

The timer in both cases is activated 1 hour after the system was booted up and then repetitively once every 24 hours. There is also a random delay on these timers set to 5 minutes. These values can be tweaked via `dnf-automatic*.timer` config files located in the `/usr/lib/systemd/system/` directory.

11.2 Utilities that have not been ported yet

`repo-rss`, `show-changed-rco`, `show-installed`, `verifytree`, `yum-groups-manager`

Take a look at the FAQ about YUM to DNF migration. Feel free to file an RFE for missing functionality if you need it.

CHAPTER 12

Changes in the DNF hook API compared to YUM

This table provides what alternative hooks are available in DNF compared to YUM.

Hook Number	YUM hook	DNF hook
1	config	init
2	postconfig	init
3	init	init
4	predownload	
5	postdownload	
6	prereposetup	
7	postreposetup	sack
8	exclude	resolved
9	preresolve	
10	postresolve	resolved but no re-resolve
11	pretrans	pre_transaction
12	postrans	transaction
13	close	transaction
14	clean	

Feel free to file an RFE for missing functionality if you need it.

CHAPTER 13

Changes in DNF-2 compared to DNF-1

13.1 CLI changes

13.1.1 Reintroduction of YUM's configuration options `includepkgs` and `excludepkgs`

Due to a better compatibility with YUM, configuration options `include` and `exclude` were replaced by the original options *includepkgs* and *excludepkgs*.

13.1.2 DNF group install `--with-optional` option

Installation of optional packages of group is changed from subcommand `with-optional` to option `--with-optional`.

13.2 Python API changes

13.2.1 All non-API methods and attributes are private

Warning: All non-API methods and attributes of *documented modules* are now private in order to accomplish more distinguishable API.

13.2.2 Following API methods accept different arguments

1. *dnf.Base.add_remote_rpms()*
2. *dnf.Base.group_install()*
3. *dnf.cli.Command.configure()*

4. *dnf.cli.Command.run()*
5. *dnf.Plugin.read_config()*

DNF Plugins and components

- DNF Plugins Core
- DNF Plugins Extras
- **'Hawkey'_**

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

Index

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W