

?all-versions(PATTERN)

Selects packages where all versions match PATTERN. When matching versions instead, same as PATTERN.

?any-version(PATTERN)

Selects any version where the pattern matches on the version.

For example, while ?and(?version(1),?version(2)) matches a package which has one version containing 1 and one version containing 2,

?any-version(?and(?version(1),?version(2))) restricts the ?and to act on the same version.

?narrow(PATTERN...)

Selects any version matching all PATTERNS, short for ?any-version(?and(PATTERN...)).

PACKAGE PATTERNS

These patterns select specific packages.

?architecture(WILDCARD), ~rWILDCARD

Selects packages matching the specified architecture, which may contain wildcards using any.

?automatic, ~M

Selects packages that were installed automatically.

?broken, ~b

Selects packages that have broken dependencies.

?config-files, ~c

Selects packages that are not fully installed, but have solely residual configuration files left.

?essential, ~E

Selects packages that have Essential: yes set in their control file.

?exact-name(NAME)

Selects packages with the exact specified name.

?garbage, ~g

Selects packages that can be removed automatically.

?installed, ~i

Selects packages that are currently installed.

?name(REGEX), ~nREGEX

Selects packages where the name matches the given regular expression.

?obsolete, ~o

Selects packages that no longer exist in repositories.

?upgradable, ~U

Selects packages that can be upgraded (have a newer candidate).

?virtual, ~v

Selects all virtual packages; that is packages without a version. These exist when they are referenced somewhere in the archive, for example because something depends on that name.

VERSION PATTERNS

These patterns select specific versions of a package.

?archive(REGEX), ~AREGEX

Selects versions that come from the archive that matches the specified regular expression. Archive, here, means the values after a= in apt-cache policy.

?codename(REGEX)

Selects versions that come from the codename that matches the specified regular expression. Codename, here, means the values after n= in apt-cache policy.

?origin(REGEX), ~OREGEX

Selects versions that come from the origin that matches the specified regular expression. Origin, here, means the values after o= in apt-cache policy.

?section(REGEX), ~sREGEX

Selects versions where the section matches the specified regular expression.

?source-package(REGEX), ~eREGEX

Selects versions where the source package name matches the specified regular expression.

?source-version(REGEX)

Selects versions where the source package version matches the specified regular expression.

?version(REGEX), ~VREGEX

Selects versions where the version string matches the specified regular expression.

?priority(NAME), ~pNAME

Selects versions where the Priority string equals the given name.

PACKAGE RELATIONSHIP PATTERNS

These patterns match specific package versions that depend/conflict with some other

packages.

?depends(PATTERN), ~DPATTERN, ?pre-depends(PATTERN), ~DPre-Depends:PATTERN,
?suggests(PATTERN), ~DSuggests:PATTERN, ?conflicts(PATTERN), ~DConflicts:PATTERN,
?replaces(PATTERN), ~DReplaces:PATTERN, ?obsoletes(PATTERN), ~DObsoletes:PATTERN,
?breaks(PATTERN), ~DBreaks:PATTERN, ?enhances(PATTERN), ~DEnhances:PATTERN

Selects versions depending/pre-depending/suggesting/conflicting/etc on/with/ packages
matching PATTERN.

?reverse-depType(PATTERN), ~RDepType:PATTERN

Opposite of ?depends and friends - selects all packages that have reverse-dependencies
(versions) matching PATTERN.

depType is one of the dependency types such as depends, so that we don't have to
repeat the entire list from the first paragraph here.

EXAMPLES

```
apt remove ?garbage
```

Remove all packages that are automatically installed and no longer needed - same as
apt autoremove

```
apt purge ?config-files
```

Purge all packages that only have configuration files left

```
apt list '~i !~M (~slibs|~sperl|~spython)'
```

List all manually-installed packages in sections matching libs, perl, or python.

MIGRATING FROM APTITUDE

Patterns in apt are heavily inspired by patterns in aptitude, but with some tweaks:

? Syntax is uniform: If there is an opening parenthesis after a term, it is always
assumed to be the beginning of an argument list.

In aptitude, a syntactic form "?foo(bar)" could mean "?and(?foo,bar)" if foo does not
take an argument. In APT, this will cause an error.

? Not all patterns are supported.

? Some additional patterns are available, for example, for finding gstreamer codecs.

? Escaping terms with ~ is not supported.

? A trailing comma is allowed in argument lists

? ?narrow accepts infinite arguments

? foo cannot be used as a shorthand for ?name(foo), as this can cause typos to go

unnoticed: Consider ?and(...,~poptional): this requires the package to have required

priority, but if you do not type the ~, it would require the package name to contain optional.

? Grouping patterns with (...) or writing ?or(A,B) as A|B are not supported. We do not believe that the use of | is that common, and the grouping is not necessary without it.

? Dependency types for ~D and related operators need to be specified in the canonical case.

SEE ALSO

apt-get(8), apt(8)

BUGS

APT bug page[1]. If you wish to report a bug in APT, please see /usr/share/doc/debian/bug-reporting.txt or the reportbug(1) command.

AUTHOR

APT was written by the APT team <apt@packages.debian.org>.

AUTHORS

Jason Gunthorpe

APT team

NOTES

1. APT bug page

<http://bugs.debian.org/src:apt>

APT 2.4.12

26 December 2021

APT-PATTERNS(7)