pacote

Fetches package manifests and tarballs from the npm registry.

USAGE

```
const pacote = require('pacote')

// get a package manifest
pacote.manifest('foo@1.x').then(manifest => console.log('got it', manifest))

// extract a package into a folder
pacote.extract('github:npm/cli', 'some/path', options)
  .then(({from, resolved, integrity}) => {
    console.log('extracted!', from, resolved, integrity)
  })

pacote.tarball('https://server.com/package.tgz').then(data => {
  console.log('got ' + data.length + ' bytes of tarball data')
})
```

`pacote` works with any kind of package specifier that npm can install. If you can pass it to the npm CLI, you can pass it to pacote. (In fact, that's exactly what the npm CLI does.)

Anything that you can do with one kind of package, you can do with another.

Data that isn't relevant (like a packument for a tarball) will be simulated.

`prepare` scripts will be run when generating tarballs from `git` and `directory` locations, to simulate what *would* be published to the registry, so that you get a working package instead of just raw source code that might need to be transpiled.

CLI

This module exports a command line interface that can do most of what is described below. Run `pacote -h` to learn more.

```
Pacote - The JavaScript Package Handler, v10.1.1

Usage:

  pacote resolve <spec>
    Resolve a specifier and output the fully resolved target
    Returns integrity and from if '--long' flag is set.

  pacote manifest <spec>
    Fetch a manifest and print to stdout
```

```
  pacote packument <spec>
    Fetch a full packument and print to stdout

  pacote tarball <spec> [<filename>]
    Fetch a package tarball and save to <filename>
    If <filename> is missing or '-', the tarball will be streamed to stdout.

  pacote extract <spec> <folder>
    Extract a package to the destination folder.

Configuration values all match the names of configs passed to npm, or
options passed to Pacote.  Additional flags for this executable:

  --long     Print an object from 'resolve', including integrity and spec.
  --json     Print result objects as JSON rather than node's default.
             (This is the default if stdout is not a TTY.)
  --help -h  Print this helpful text.

For example '--cache=/path/to/folder' will use that folder as the cache.
```

API

The `spec` refers to any kind of package specifier that npm can install. If you can pass it to the npm CLI, you can pass it to pacote. (In fact, that's exactly what the npm CLI does.)

See below for valid `opts` values.

- `pacote.resolve(spec, opts)` Resolve a specifier like `foo@latest` or `github:user/project` all the way to a tarball url, tarball file, or git repo with commit hash.
- `pacote.extract(spec, dest, opts)` Extract a package's tarball into a destination folder. Returns a promise that resolves to the `{from,resolved,integrity}` of the extracted package.
- `pacote.manifest(spec, opts)` Fetch (or simulate) a package's manifest (basically, the `package.json` file, plus a bit of metadata). See below for more on manifests and packuments. Returns a Promise that resolves to the manifest object.
- `pacote.packument(spec, opts)` Fetch (or simulate) a package's packument (basically, the top-level package document listing all the manifests that the registry returns). See below for more on manifests and packuments. Returns a Promise that resolves to the packument object.
- `pacote.tarball(spec, opts)` Get a package tarball data as a buffer in memory. Returns a Promise that resolves to the tarball data Buffer, with `from`, `resolved`, and `integrity` fields attached.

- `pacote.tarball.file(spec, dest, opts)` Save a package tarball data to a file on disk. Returns a Promise that resolves to `{from,integrity,resolved}` of the fetched tarball.
- `pacote.tarball.stream(spec, streamHandler, opts)` Fetch a tarball and make the stream available to the `streamHandler` function.

  This is mostly an internal function, but it is exposed because it does provide some functionality that may be difficult to achieve otherwise.

  The `streamHandler` function MUST return a Promise that resolves when the stream (and all associated work) is ended, or rejects if the stream has an error.

  The `streamHandler` function MAY be called multiple times, as Pacote retries requests in some scenarios, such as cache corruption or retriable network failures.

Options

Options are passed to `npm-registry-fetch` and `cacache`, so in addition to these, anything for those modules can be given to pacote as well.

Options object is cloned, and mutated along the way to add integrity, resolved, and other properties, as they are determined.

- `cache` Where to store cache entries and temp files. Passed to `cacache`. Defaults to the same cache directory that npm will use by default, based on platform and environment.
- `where` Base folder for resolving relative `file:` dependencies.
- `resolved` Shortcut for looking up resolved values. Should be specified if known.
- `integrity` Expected integrity of fetched package tarball. If specified, tarballs with mismatched integrity values will raise an `EINTEGRITY` error.
- `umask` Permission mode mask for extracted files and directories. Defaults to `0o22`. See "Extracted File Modes" below.
- `fmode` Minimum permission mode for extracted files. Defaults to `0o666`. See "Extracted File Modes" below.
- `dmode` Minimum permission mode for extracted directories. Defaults to `0o777`. See "Extracted File Modes" below.
- `preferOnline` Prefer to revalidate cache entries, even when it would not be strictly necessary. Default `false`.
- `before` When picking a manifest from a packument, only consider packages published before the specified date. Default `null`.
- `defaultTag` The default `dist-tag` to use when choosing a manifest from a packument. Defaults to `latest`.
- `registry` The npm registry to use by default. Defaults to `https://registry.npmjs.org/`.
- `fullMetadata` Fetch the full metadata from the registry for packuments, including information not strictly required for installation (author, descrip-

tion, etc.) Defaults to `true` when `before` is set, since the version publish time is part of the extended packument metadata.
- `fullReadJson` Use the slower `read-package-json` package insted of `read-package-json-fast` in order to include extra fields like "readme" in the manifest. Defaults to `false`.
- `packumentCache` For registry packuments only, you may provide a `Map` object which will be used to cache packument requests between pacote calls. This allows you to easily avoid hitting the registry multiple times (even just to validate the cache) for a given packument, since it is unlikely to change in the span of a single command.
- `silent` A boolean that determines whether the banner is displayed when calling `@npmcli/run-script`.

Advanced API

Each different type of fetcher is exposed for more advanced usage such as using helper methods from this classes:

- `DirFetcher`
- `FileFetcher`
- `GitFetcher`
- `RegistryFetcher`
- `RemoteFetcher`

Extracted File Modes

Files are extracted with a mode matching the following formula:

```
( (tarball entry mode value) | (minimum mode option) ) ~ (umask)
```

This is in order to prevent unreadable files or unlistable directories from cluttering a project's `node_modules` folder, even if the package tarball specifies that the file should be inaccessible.

It also prevents files from being group- or world-writable without explicit opt-in by the user, because all file and directory modes are masked against the `umask` value.

So, a file which is `0o771` in the tarball, using the default `fmode` of `0o666` and `umask` of `0o22`, will result in a file mode of `0o755`:

```
(0o771 | 0o666) => 0o777
(0o777 ~ 0o22) => 0o755
```

In almost every case, the defaults are appropriate. To respect exactly what is in the package tarball (even if this makes an unusable system), set both `dmode` and `fmode` options to `0`. Otherwise, the `umask` config should be used in most cases where file mode modifications are required, and this functions more or less the same as the `umask` value in most Unix systems.

Extracted File Ownership

When running as `root` on Unix systems, all extracted files and folders will have their owning `uid` and `gid` values set to match the ownership of the containing folder.

This prevents `root`-owned files showing up in a project's `node_modules` folder when a user runs `sudo npm install`.

Manifests

A `manifest` is similar to a `package.json` file. However, it has a few pieces of extra metadata, and sometimes lacks metadata that is inessential to package installation.

In addition to the common `package.json` fields, manifests include:

- `manifest._resolved` The tarball url or file path where the package artifact can be found.
- `manifest._from` A normalized form of the spec passed in as an argument.
- `manifest._integrity` The integrity value for the package artifact.
- `manifest.dist` Registry manifests (those included in a packument) have a `dist` object. Only `tarball` is required, though at least one of `shasum` or `integrity` is almost always present.
  - `tarball` The url to the associated package artifact. (Copied by Pacote to `manifest._resolved`.)
  - `integrity` The integrity SRI string for the artifact. This may not be present for older packages on the npm registry. (Copied by Pacote to `manifest._integrity`.)
  - `shasum` Legacy integrity value. Hexadecimal-encoded sha1 hash. (Converted to an SRI string and copied by Pacote to `manifest._integrity` when `dist.integrity` is not present.)
  - `fileCount` Number of files in the tarball.
  - `unpackedSize` Size on disk of the package when unpacked.
  - `npm-signature` A signature of the package by the `npmregistry` Keybase account. (Obviously only present for packages published to `https://registry.npmjs.org`.)

Packuments

A packument is the top-level package document that lists the set of manifests for available versions for a package.

When a packument is fetched with `accept: application/vnd.npm.install-v1+json` in the HTTP headers, only the most minimum necessary metadata is re-

turned. Additional metadata is returned when fetched with only `accept: application/json`.

For Pacote's purposes, the following fields are relevant:

- `versions` An object where each key is a version, and each value is the manifest for that version.
- `dist-tags` An object mapping dist-tags to version numbers. This is how `foo@latest` gets turned into `foo@1.2.3`.
- `time` In the full packument, an object mapping version numbers to publication times, for the `opts.before` functionality.