npm-scope
Scoped packages

DESCRIPTION

All npm packages have a name. Some package names also have a scope. A scope follows the usual rules for package names (URL-safe characters, no leading dots or underscores). When used in package names, scopes are preceded by an `@` symbol and followed by a slash, e.g.

```
@somescope/somepackagename
```

Scopes are a way of grouping related packages together, and also affect a few things about the way npm treats the package.

Each npm user/organization has their own scope, and only you can add packages in your scope. This means you don't have to worry about someone taking your package name ahead of you. Thus it is also a good way to signal official packages for organizations.

Scoped packages can be published and installed as of `npm@2` and are supported by the primary npm registry. Unscoped packages can depend on scoped packages and vice versa. The npm client is backwards-compatible with unscoped registries, so it can be used to work with scoped and unscoped registries at the same time.

Installing scoped packages

Scoped packages are installed to a sub-folder of the regular installation folder, e.g. if your other packages are installed in `node_modules/packagename`, scoped modules will be installed in `node_modules/@myorg/packagename`. The scope folder (`@myorg`) is simply the name of the scope preceded by an `@` symbol, and can contain any number of scoped packages.

A scoped package is installed by referencing it by name, preceded by an `@` symbol, in `npm install` :

```
npm install @myorg/mypackage
```

Or in `package.json` :

```
"dependencies": {
  "@myorg/mypackage": "^1.3.0"
}
```

Note that if the `@` symbol is omitted, in either case, npm will instead attempt to install from GitHub; see `npm-install` .

Requiring scoped packages

Because scoped packages are installed into a scope folder, you have to include the name of the scope when requiring them in your code, e.g.

```
require('@myorg/mypackage')
```

There is nothing special about the way Node treats scope folders. This simply requires the `mypackage` module in the folder named `@myorg` .

Publishing scoped packages

Scoped packages can be published from the CLI as of `npm@2` and can be published to any registry that supports them, including the primary npm registry.

(As of 2015-04-19, and with npm 2.0 or better, the primary npm registry **does** support scoped packages.)

If you wish, you may associate a scope with a registry; see below.

Publishing public scoped packages to the primary npm registry

To publish a public scoped package, you must specify `--access public` with the initial publication. This will publish the package and set access to `public` as if you had run `npm access public` after publishing.

Publishing private scoped packages to the npm registry

To publish a private scoped package to the npm registry, you must have an npm Private Modules account.

You can then publish the module with `npm publish` or `npm publish --access restricted` , and it will be present in the npm registry, with restricted access. You can then change the access permissions, if desired, with `npm access` or on the npmjs.com website.

Associating a scope with a registry

Scopes can be associated with a separate registry. This allows you to seamlessly use a mix of packages from the primary npm registry and one or more private registries, such as npm Enterprise.

You can associate a scope with a registry at login, e.g.

```
npm login --registry=http://reg.example.com --scope=@myco
```

Scopes have a many-to-one relationship with registries: one registry can host multiple scopes, but a scope only ever points to one registry.

You can also associate a scope with a registry using `npm config`:

```
npm config set @myco:registry http://reg.example.com
```

Once a scope is associated with a registry, any `npm install` for a package with that scope will request packages from that registry instead. Any `npm publish` for a package name that contains the scope will be published to that registry instead.