build.clj

```
(ns build
  (:require [clojure.string :as string]
            [clojure.tools.build.api :as b]
            [deps-deploy.deps-deploy :as deploy]))

(def lib 'art/pedestaldocker)
(def main-cls (string/join "." (filter some? [(namespace lib) (name lib)
"core"])))
(def version (format "0.0.1-SNAPSHOT"))
(def target-dir "target")
(def class-dir (str target-dir "/" "classes"))
(def uber-file (format "%s/%s-standalone.jar" target-dir (name lib)))
(def basis (b/create-basis {:project "deps.edn"}))

(defn clean
  "Delete the build target directory"
  [_]
  (println (str "Cleaning " target-dir))
  (b/delete {:path target-dir}))

(defn prep [_]
  (println "Writing Pom...")
  (b/write-pom {:class-dir class-dir
                :lib lib
                :version version
                :basis basis
                :src-dirs ["src/clj"]})
  (b/copy-dir {:src-dirs ["src/clj" "resources" "env/prod/clj"]
               :target-dir class-dir}))

(defn uber [_]
  (println "Compiling Clojure...")
  (b/compile-clj {:basis basis
                  :src-dirs ["src/clj" "env/prod/clj"]
                  :class-dir class-dir})
  (println "Making uberjar...")
  (b/uber {:class-dir class-dir
           :uber-file uber-file
           :main main-cls
           :basis basis}))

(defn all [_]
  (do (clean nil) (prep nil) (uber nil)))
```

실행

```
clj -T:build all
```

Polylith build.clj

```
(ns build
  "The build script for the example of the poly documentation.
   Targets:
   * uberjar :project PROJECT
     - creates an uberjar for the given project
   For help, run:
     clojure -A:deps -T:build help/doc
   Create uberjar for command-line:
     clojure -T:build uberjar :project command-line"
  (:require [clojure.java.io :as io]
            [clojure.tools.build.api :as b]
            [clojure.tools.deps.alpha :as t]
            [clojure.tools.deps.alpha.util.dir :refer [with-dir]]
            [org.corfield.build :as bb]))

(defn- get-project-aliases []
  (let [edn-fn (juxt :root-edn :project-edn)]
    (-> (t/find-edn-maps)
        (edn-fn)
        (t/merge-edns)
        :aliases)))

(defn- ensure-project-root
  "Given a task name and a project name, ensure the project
   exists and seems valid, and return the absolute path to it."
  [task project]
  (let [project-root (str (System/getProperty "user.dir") "/projects/"
project)]
    (when-not (and project
                   (.exists (io/file project-root))
                   (.exists (io/file (str project-root "/deps.edn"))))
      (throw (ex-info (str task " task requires a valid :project option")
{:project project})))
    project-root))

(defn uberjar
  "Builds an uberjar for the specified project.
   Options:
   * :project - required, the name of the project to build,
   * :uber-file - optional, the path of the JAR file to build,
     relative to the project folder; can also be specified in
```

```
    the :uberjar alias in the project's deps.edn file; will
    default to target/PROJECT.jar if not specified.
  Returns:
  * the input opts with :class-dir, :compile-opts, :main, and :uber-file
    computed.
  The project's deps.edn file must contain an :uberjar alias
  which must contain at least :main, specifying the main ns
  (to compile and to invoke)."
 [{:keys [project uber-file] :as opts}]
 (let [project-root (ensure-project-root "uberjar" project)
       aliases      (with-dir (io/file project-root) (get-project-aliases))
       main         (-> aliases :uberjar :main)]
   (when-not main
     (throw (ex-info (str "the " project " project's deps.edn file does not
specify the :main namespace in its :uberjar alias")
                     {:aliases aliases})))
   (binding [b/*project-root* project-root]
     (let [class-dir "target/classes"
           uber-file (or uber-file
                         (-> aliases :uberjar :uber-file)
                         (str "target/" project ".jar"))
           opts      (merge opts
                            {:class-dir    class-dir
                             :compile-opts {:direct-linking true}
                             :main         main
                             :uber-file    uber-file})]
       (b/delete {:path class-dir})
       (bb/uber opts)
       (b/delete {:path class-dir})
       (println "Uberjar is built.")
       opts))))
```

관련 문서

Plugin Backlinks: 아무 것도 없습니다.