This is a system of building and caching dependencies necessary for building Bitcoin. There are several features that make it different from most similar systems:

It is designed to be builder and host agnostic

In theory, binaries for any target OS/architecture can be created, from a builder running any OS/architecture. In practice, build-side tools must be specified when the defaults don't fit, and packages must be amended to work on new hosts. For now, a build architecture of x86_64 is assumed, either on Linux or macOS.

No reliance on timestamps

File presence is used to determine what needs to be built. This makes the results distributable and easily digestable by automated builders.

Each build only has its specified dependencies available at build-time.

For each build, the sysroot is wiped and the (recursive) dependencies are installed. This makes each build deterministic, since there will never be any unknown files available to cause side-effects.

Each package is cached and only rebuilt as needed.

Before building, a unique build-id is generated for each package. This id consists of a hash of all files used to build the package (Makefiles, packages, etc), and as well as a hash of the same data for each recursive dependency. If any portion of a package's build recipe changes, it will be rebuilt as well as any other package that depends on it. If any of the main makefiles (Makefile, funcs.mk, etc) are changed, all packages will be rebuilt. After building, the results are cached into a tarball that can be re-used and distributed.

Package build results are (relatively) deterministic.

Each package is configured and patched so that it will yield the same build-results with each consequent build, within a reasonable set of constraints. Some things like timestamp insertion are unavoidable, and are beyond the scope of this system. Additionally, the toolchain itself must be capable of deterministic results. When revisions are properly bumped, a cached build should represent an exact single payload.

Sources are fetched and verified automatically

Each package must define its source location and checksum. The build will fail if the fetched source does not match. Sources may be pre-seeded and/or cached as desired.

Self-cleaning

Build and staging dirs are wiped after use, and any previous version of a cached result is removed following a successful build. Automated builders should be able to build each revision and store the results with no further intervention.