Go-filecoin code layout do-over

@icorderi, September 2019
Issue: #2148
Status: receiving feedback

Background

The go-filecoin codebase has evolved without clear direction since early prototyping days. The result is a mass of top-level packages that provide little guidance to the structure of the components or their relationships. This makes the code difficult for newcomers to navigate and provides scant help enforcing or even recognizing inter-dependency relationships.

The Go community has evolved somewhat-standard code structure ideas, described in https://github.com/golang-standards/project-layout. While not necessary or enforced, these ideas provide some clear guidance, and following them will help our codebase become more navigable to all.

Separate to package layout, go-filecoin also suffers from poorly defined logical architecture. Addressing that is outside the scope of this document, but the layout ideas should support and complement architectural improvements.

Goals

- A package structure that is navigable to newcomers, especially Go programmers
- A package structure that supports high level architectural concerns, especially:
    - the distinction between library and application code
    - the distinction between exported components and internal details
    - the distinction between core, differentiating components from general utility code

Non-goals

- Perfectionist ideas that stands in the way of incremental improvement
    - Don't let lack of clear logical architecture block improvements to package structure, even if we have to change it a bit later

Design

This section describes a proposed high-level layout, in line with the golang-standards reference. Detailed decisions about which component belongs where are to be left for implementation.

Top-level

./cmd/	the main binary entry points: go-filecoin
./pkg/	public library code (things others might import and use)
./internal/	private code (import at your own risk)
./internal/app/go-filecoin/	application-specific implementation code
./internal/pkg/	library implementation code
./vendor/	git submodules and vendored dependencies
./scripts/	installation and other scripts (we might have none)
./test	test data, external testing
./tools	supporting tools
./build	build scripts

The `/pkg` directory

These are public packages that we are exporting because they are useful to import, link against, or otherwise integrate into other software systems.

Nothing in this directory should depend on anything in another directory. Many of these are generic and candidates for moving into a separate repository at some point.

Immediate examples:

- ./pkg/rleplus
- ./pkg/cborutil
- ./pkg/crypto
- ./pkg/address
- ./pkg/clock
- ./pkg/metrics

In the future, we might move the whole “Filecoin protocol library” component here as an exported item, but for now we’ll keep it internal.

The `/internal/pkg` directory

These are private packages that could be used to build different binaries, but that are not exported because they are unstable or essentially internal details. As a rough heuristic, things that are described in the Filecoin protocol spec live here.

Nothing in this directory should depend on anything in the /internal/app directory.

Immediate examples:

- ./internal/pkg/vm
- ./internal/pkg/actor
- ./internal/pkg/consensus

```
./internal/pkg/types
```

The `/internal/app` directory

These are private application implementations. This is generally code that is not constrained by the spec, but could vary in other Filecoin node implementations.

Note that many of our packages currently have a poor separation between the “library” (i.e. spec-compliant) and “application” (operational) code. E.g. block production is mostly operational, but includes some necessary protocol features. Similar for most of what are now called “protocols”.

Some examples:

```
## Supporting code
./internal/app/go-filecoin/config
./internal/app/go-filecoin/net
./internal/app/go-filecoin/repo
./internal/app/go-filecoin/util
./internal/app/go-filecoin/wallet

## Worker/process implementations
./internal/app/go-filecoin/chain
./internal/app/go-filecoin/heartbeat
./internal/app/go-filecoin/mining
./internal/app/go-filecoin/protocol/*
./internal/app/go-filecoin/sectorbuilder

## Glue and APIs
./internal/app/go-filecoin/node
./internal/app/go-filecoin/plumbing
./internal/app/go-filecoin/porcelain
```

Alternatives

One alternative is for package layout to closely mirror application architecture (or desired architecture). Unfortunately that depends on us having clearly articulated that architecture in order for us to make improvements here. The work of de-coupling and breaking dependencies is a lot harder than mere re-arrangement, so waiting for that could postpone layout improvements indefinitely.