Irmin layers/GC and testing

Problem: At the moment, layers/GC is not part of Irmin main branch. This poses some issues for testing. In particular, trying to test against an existing system state (e.g. using Irmin "snapshots", or directly against the Irmin data-dir containing a `store.pack` file) is not possible because layers/GC uses a slightly different format for the `store.pack` data.

Description of layers/GC format: It is worth briefly describing the file format that Irmin layers/GC uses. The *existing* format of a Tezos context (without layers) might contain the following:

```
drwxr-xr-x 2 tom tom 4.0K 2022-02-16 16:02:28.000000000 +0000 index
-r--r--r-- 1 tom tom   67 2022-02-16 15:52:03.000000000 +0000 store.branches
-r--r--r-- 1 tom tom 6.5M 2022-02-16 15:52:03.000000000 +0000 store.dict
-r--r--r-- 1 tom tom  54G 2022-02-16 16:02:25.000000000 +0000 store.pack
```

(This is from a recent Hangzhou data directory.)

The main data is stored in the `store.pack` file. In layers, this is replaced as follows: `store.pack` file becomes a very small text file which contains the following contents:

```
((subdir_name layers.0))
```

This means that the data for the store.pack is now contained in a subdirectory `layers.0`. The context directory for layers/GC thus looks as follows:

```
drwxr-xr-x  2 tom tom 4.0K 2022-02-16 16:02:28.000000000 +0000 index
drwxrwx---  4 tom tom 4.0K 2022-02-23 13:51:26.630546679 +0000 layers.0
-r--r--r--  1 tom tom   67 2022-02-16 15:52:03.000000000 +0000 store.branches
-r--r--r--  1 tom tom 6.5M 2022-02-16 15:52:03.000000000 +0000 store.dict
-rw-rw-r--  1 tom tom   25 2022-02-23 13:51:26.630546679 +0000 store.pack
```

The `store.pack` file is now a very small text file, and all the data has been moved to the `layers.0` subdirectory. In that directory we find the following:

```
drwxrwx--- 2 tom tom 4.0K 2022-02-23 13:52:54.344974670 +0000 sparse.1234
drwxrwx--- 2 tom tom 4.0K 2022-02-23 13:51:26.630546679 +0000 suffix.1234
-rw-rw---- 1 tom tom   24 2022-02-23 13:52:54.344974670 +0000 control
```

The `sparse.1234` subdirectory contains the contents of the sparse prefix of the file:

```
-rw-rw---- 1 tom tom    0 2022-02-23 13:51:26.630546679 +0000 data
-rw------- 1 tom tom    0 2022-02-23 13:52:54.344974670 +0000 map
```

As can be seen, this is trivial (both `data` and `map` are 0 bytes long) because the sparse prefix at this point doesn't contain any data. Instead, all the data is contained in the suffix directory:

```
-rw-rw---- 1 tom tom  54G 2022-02-23 13:52:52.233011069 +0000 suffix_data
-rw-rw-r-- 1 tom tom   20 2022-02-23 13:51:26.630546679 +0000 suffix_offset
```

The `suffix_data` file contains the actual data (which is almost what was originally in `store.pack`, except that the first 16 bytes of that original file -- which contain metadata rather than data -- are dropped and stored in the control file instead). The `suffix_offset` file contains the following:

```
((suffix_offset 0))
```

This simply says that the suffix data starts from address 0 in the original file (that is, the suffix covers the entirety of the original data).

Finally, we have the `layers.0/control` file, which contains 3 integers in "native endian" format. These are defined as follows (in `control.ml` in the layers source code):

```
let generation_field : field = 0
let version_field : field = 1
let last_synced_offset_field : field = 2
```

Here, `generation` is an integer that gets bumped when a GC occurs; `version` is "1 or 2" (an Irmin-internal notion of file version); `last_synced_offset` is the offset when `fsync` was last called (used only by RO processes to sync with the RW process). `generation_field` defaults to `1234`.

Converting an existing data-dir to layers/GC format: This is actually fairly straightforward. We simply copy the data from the original `store.pack` file to the `layers.0/suffix.1234/suffix_data` file (omitting the first 16 bytes). Then we create the `layers.0/sparse.1234` directory, and the `layers.0/suffix.1234/suffix_offset` file (contents as above). Finally, we construct the control file. This is the hardest since it requires us to write integers in "native endian" format.

Script to perform conversion: There is a script `bin-layers/convert_store_dot_pack.ml` that can be built as an `.exe` using dune. This script takes a path to an original `store.pack` file, and an output directory, and converts the `store.pack` to use the new layers/GC format.

Example usage: `convert_store_dot_pack.exe ./path/to/store.pack ./existing_empty_directory/`