Contents

1 Database Design

1.1 Notes

1.1.1 On-Disk

Notes are named by their path relative to the notebook directory (*i.e.* a note located at `$ADRUS_DIR/foo/bar/baz` will have the name `/foo/bar/baz`).

Notes are plain-text files that start with the word `adrus` and a list of attributes, terminated by a new line:

```
adrus attr1 attr2=value...
...
```

Attributes are a key with an (optional) value attached to them; see `db_attr_t` for more info.

1.1.2 `db_note_t`

- `name` – path relative to notebook
- `ctime` – time of creation
- `mtime` – time of modification
- `attrs` – a list of `db_attr_t` that a note has attached to them

1.1.3 `db_attr_t`

- `key` – name of the attribute
- `val` – the value attached to the attribute; can be any of the following types:
  - integer
  - string
  - date/time

1.2 Lookup Information

Internally two lookup tables are maintained:
1. a map from note name -> `db_note_t` – to access note attributes
2. a map from attr key -> list of `db_note_t *`– to list the notes that have a specific attribute.

These lookup tables are hash maps containing buckets of `db_note_ent_t` and `db_attr_ent_t` respectively.

1.2.1 `db_note_ent_t`

- `hash` – hash of the `db_note_t`'s name
- `note` – `db_note_t`

1.2.2 `db_attr_ent_t`

- `hash` – hash of the attribute's name
- `name` – name of the attribute
- `notes` – a list of pointers of `db_note_ts` associated with the attribute.
- `type` – a description of the type of an attribute; used to enforce all attributes of notes with a given key have the same type

2 Attribute Syntax

Attributes are pieces of data attached to notes, stored in the note's header.

- `+attr` / `-attr` – add/remove an attribute
- `attr=value` – assign a value to an attribute

3 Queries

3.1 Attribute Matching Syntax

Queries are a list of whitespace-delimited predicates used to filter notes in the database based on their attributes.

Queries take 2 forms:

- `+attr` / `-attr` – checks for the existence / non-existence of an attribute on a note
- `attr op value` – compares the value of an attribute on a note, using the following operators:
  - `=` / `/=` – check if `attr` equals / doesn't equal a `value`
  - `<` / `<=` / `>` / `>=` – compare ordering of an attribute against a `value` (alphanumeric ordering if a string)
  - `~` – check if attribute fuzzy matches against a `value`

3.2 Pattern Syntax

Patterns, similar to POSIX globs, are used to match note names against a given pattern in a query.

3.2.1 Syntax

Patterns take the form of a path – starting with a `/` – with asterisks being used to perform wildcard matches:

- `x*y` – matches a component of a path that starts with `x` and ends with `y`
- `**` – matches any number of parent directories of a path

Everything except for the wildcard character is treated literaly.

Patterns that end with a `/` have an implicit `**` at the end; *i.e.* they will match all notes who reside somewhere within the given parent folder.

Given a notebook containing the following notes:

```
/lang/semitic/arabic.txt
/lang/semitic/vocab
/lang/semitic/hebrew.txt
/lang/germanic/english
/comp/data-structures.txt
/comp/vocab
```

The following patterns should give the responses below:

1. `/lang/semitic/arabic.txt`
   - `/lang/semitic/arabic.txt`
2. `/doesnt/exist`
   - *N/A*
3. `/**/vocab`
   - `/lang/semitic/vocab`
   - `/comp/vocab`
4. `/lang/semitic/*.txt`
   - `/lang/semitic/arabic.txt`
   - `/lang/semitic/hebrew.txt`
5. `/**/*.txt`
   - `/lang/semitic/arabic.txt`
   - `/lang/semitic/hebrew.txt`
   - `/comp/data-structures.txt`

3.2.2 Tokens (regex)

1. `/` – path separator
2. `[^/\*]+` – literal
3. `\*` – any literal sequence
4. `\*\*` – any number of directory components

4 Interface

The primary interface to Adrus will be a command-line interface. All operations in the program should be invoked via textual commands.

4.1 Querying

```
adrus
adrus [CONDITION]...
```

Adrus invoked with an (optional) set of query conditions will match notes in the database against those conditions. Notes are unfiltered by default, without any conditions passed in.

e.g.

```
adrus
adrus +cool -bad importance<10 foo=bar
```

4.2 Opening a note

```
adrus [PATH]
```

Adrus invoked with a path as it's only argument will open a note in your text editor (determined via `$EDITOR`). If that note does not already exist, then a file will be created containing an empty Adrus header, along with all of it's parent directories.

e.g.

```
adrus /lang/slavic/russian.txt
```

4.3 Mutating a note

```
adrus [PATH] [ATTRIBUTE]...
```

Adrus invoked with a pattern and a list of attributes will modify the attributes of any notes who's name matches the pattern on-disk. Attempting to mutate a note that does not exist will result in an error.

e.g.

```
adrus /lang/semitic/arabic.txt +cool -semitic dir=rtl
```

4.4 Working with notes

```
adrus ls [PATTERN] [CONDITION]...
adrus rm [PATTERN] [CONDITION]...
adrus mv [PATH] [PATH]
```

These commands are roughly analogous to their POSIX equivalents. `ls` and `rm` will list and delete, respectively, all notes matching against a given pattern and an optional list of conditions.

`mv` will re-name a note given by the first path, to the second path. It will error if there is a conflict.

e.g.

```
adrus ls /**/*.txt +language -cool
adrus rm /**/*.txt +language -cool
adrus mv /some/note /new/name.txt
```

4.5 Configuration

Adrus is configured using environment variables:

- `ADRUS_DIR` – path to adrus notebook.
- `LOG_FILTER` – filter for log messages (from highest to lowest priority):

  - `error`
  - `warn`
  - `info` *(default)*
  - `debug`
  - `trace`
- `EDITOR` – the editor to be used when opening notes

4.6 Output

Adrus outputs logging to `stderr`, and usable output to `stdout`. Usable output is meant to be simple to parse and work with programatically.