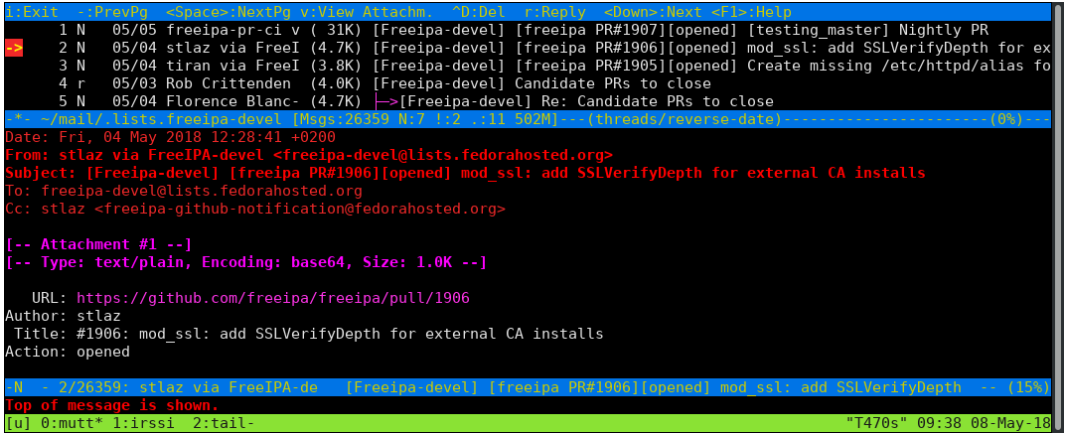
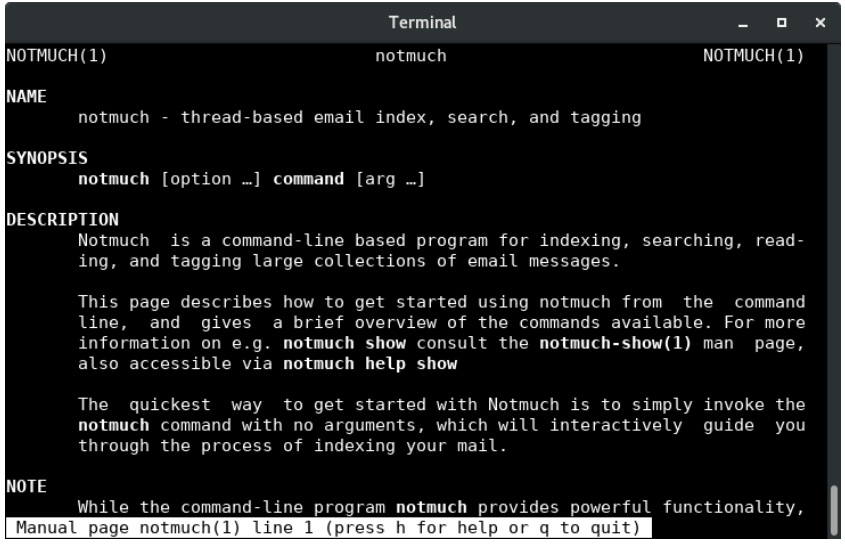
Taming the C Monster

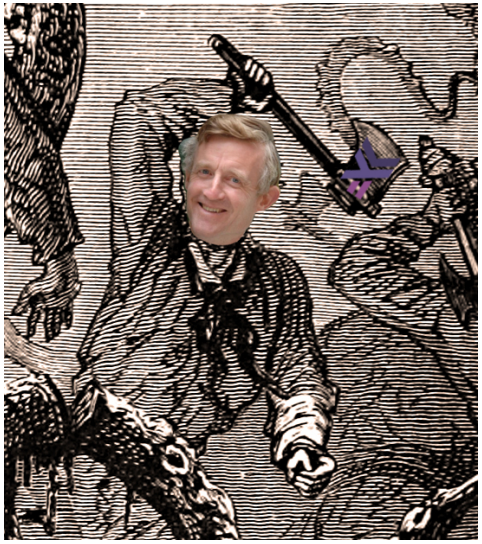
Haskell FFI Techniques

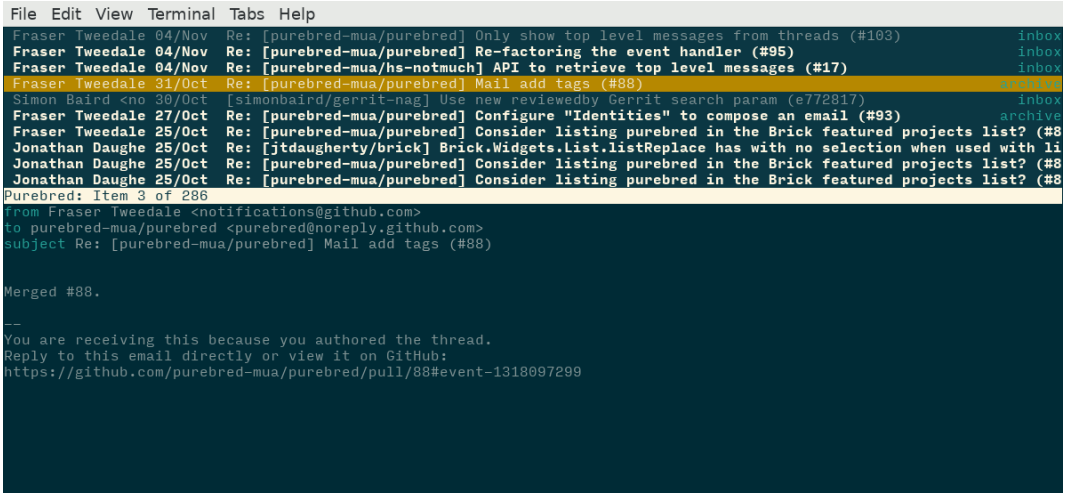
Fraser Tweedale
`@hackuador`

May 22, 2018


i:Exit -:PrevPg <Space>:NextPg v:View Attachm. ^D:Del r:Reply <Down>:Next <Fl>:Help
1 N 05/05 freeipa-pr-ci v (31K) [Freeipa-devel] [freeipa PR#1907][opened] [testing_master] Nightly PR
-> 2 N 05/04 stlaz via FreeI (4.7K) [Freeipa-devel] [freeipa PR#1906][opened] mod_ssl: add SSLVerifyDepth for ex
3 N 05/04 tiran via FreeI (3.8K) [Freeipa-devel] [freeipa PR#1905][opened] Create missing /etc/httpd/alias fo
4 r 05/03 Rob Crittenden (4.0K) [Freeipa-devel] Candidate PRs to close
5 N 05/04 Florence Blanc- (4.7K) [->[Freeipa-devel] Re: Candidate PRs to close
-*- ~/mail/.lists.freeipa-devel [Msgs:26359 N:7 !:2 .:11 502M]---(threads/reverse-date)----------(0%)---
Date: Fri, 04 May 2018 12:28:41 +0200
From: stlaz via FreeIPA-devel <freeipa-devel@lists.fedorahosted.org>
Subject: [Freeipa-devel] [freeipa PR#1906][opened] mod_ssl: add SSLVerifyDepth for external CA installs
To: freeipa-devel@lists.fedorahosted.org
Cc: stlaz <freeipa-github-notification@fedorahosted.org>
[-- Attachment #1 --]
[-- Type: text/plain, Encoding: base64, Size: 1.0K --]
URL: https://github.com/freeipa/freeipa/pull/1906
Author: stlaz
Title: #1906: mod_ssl: add SSLVerifyDepth for external CA installs
Action: opened
-N - 2/26359: stlaz via FreeIPA-de [Freeipa-devel] [freeipa PR#1906][opened] mod_ssl: add SSLVerifyDepth -- (15%)
Top of message is shown.
[u] 0:mutt* 1:irssi 2:tail-
"T470s" 09:38 08-May-18


Terminal
NOTMUCH(1) notmuch NOTMUCH(1)
NAME
notmuch - thread-based email index, search, and tagging
SYNOPSIS
notmuch [option …] command [arg …]
DESCRIPTION
Notmuch is a command-line based program for indexing, searching, reading, and tagging large collections of email messages.
This page describes how to get started using notmuch from the command line, and gives a brief overview of the commands available. For more information on e.g. notmuch show consult the notmuch-show(1) man page, also accessible via notmuch help show
The quickest way to get started with Notmuch is to simply invoke the notmuch command with no arguments, which will interactively guide you through the process of indexing your mail.
NOTE
While the command-line program notmuch provides powerful functionality,
Manual page notmuch(1) line 1 (press h for help or q to quit)




File Edit View Terminal Tabs Help
Fraser Tweedale 04/Nov Re: [purebred-mua/purebred] Only show top level messages from threads (#103) inbox
Fraser Tweedale 04/Nov Re: [purebred-mua/purebred] Re-factoring the event handler (#95) inbox
Fraser Tweedale 04/Nov Re: [purebred-mua/hs-notmuch] API to retrieve top level messages (#17) inbox
Fraser Tweedale 31/Oct Re: [purebred-mua/purebred] Mail add tags (#88) archive
Simon Baird <no 30/Oct [simonbaird/gerrit-nag] Use new reviewedby Gerrit search param (e772817) inbox
Fraser Tweedale 27/Oct Re: [purebred-mua/purebred] Configure "Identities" to compose an email (#93) archive
Fraser Tweedale 25/Oct Re: [purebred-mua/purebred] Consider listing purebred in the Brick featured projects list? (#8
Jonathan Daughe 25/Oct Re: [jtdaugherty/brick] Brick.Widgets.List.listReplace has with no selection when used with li
Jonathan Daughe 25/Oct Re: [purebred-mua/purebred] Consider listing purebred in the Brick featured projects list? (#8
Jonathan Daughe 25/Oct Re: [purebred-mua/purebred] Consider listing purebred in the Brick featured projects list? (#8
Purebred: Item 3 of 286
from Fraser Tweedale <notifications@github.com>
to purebred-mua/purebred <purebred@noreply.github.com>
subject Re: [purebred-mua/purebred] Mail add tags (#88)
Merged #88.
--
You are receiving this because you authored the thread.
Reply to this email directly or view it on GitHub:
https://github.com/purebred-mua/purebred/pull/88#event-1318097299

FFI basics

why FFI?

- want to do $THING in Haskell
- there exists a C library for $THING
- interoperability / bug-compatibility
- performance / timing-critical code

C FFI

```haskell
{-# LANGUAGE ForeignFunctionInterface #-}

import Foreign.C.Types

foreign import ccall "math.h sin"
  c_sin :: CDouble -> CDouble

main :: IO ()
main = print $ c_sin 1.0
```

hsc2hs

- file extension: `.hsc`
- part of GHC distribution
- good support for marshalling structs

c2hs

- file extension: `.chs`
- more features than `hsc2hs`
- automatic generation of `foreign import` declarations

```
library
  ...
  build-tools:
    c2hs >= 0.19.1
```

c2hs - example

```
...
  result <- {#call notmuch_database_open #} path 1 ptr
  ...
```

c2hs - example

```
...
  result <- notmuch_database_open path 1 ptr
  ...

foreign import ccall "Notmuch/Binding.chs.h notmuch_database_open"
  notmuch_database_open
    :: CString -> CInt -> Ptr (Ptr Database) -> IO CInt
```

Foreign.Ptr

```
data Ptr a

nullPtr :: Ptr a

plusPtr :: Ptr a -> Int -> Ptr b

castPtr :: Ptr a -> Ptr b
```

Foreign.ForeignPtr

```
data ForeignPtr a

type FinalizerPtr a = FunPtr (Ptr a -> IO ())

newForeignPtr :: FinalizerPtr a -> Ptr a -> IO (ForeignPtr a)

withForeignPtr :: ForeignPtr a -> (Ptr a -> IO b) -> IO b
```

Foreign.C.String

```
type CString = Ptr CChar

peekCString :: CString -> IO String

withCString :: String -> (CString -> IO a) -> IO a
```

Foreign.Storable

```haskell
class Storable a where
  peek :: Ptr a -> IO a
  ...

instance Storable (Ptr a)


-- Foreign.Marshal.Alloc
alloca :: Storable a => (Ptr a -> IO b) -> IO b
```

C constructions and idioms

enum types

```
typedef enum _notmuch_status {
  NOTMUCH_STATUS_SUCCESS = 0,
  NOTMUCH_STATUS_OUT_OF_MEMORY,
  NOTMUCH_STATUS_READ_ONLY_DATABASE,
  NOTMUCH_STATUS_UNBALANCED_FREEZE_THAW,
  ...
} notmuch_status_t
```

enum types

```
{#enum notmuch_status_t as Status {underscoreToCase} deriving (Eq) #}
```

enum types

```
data Status = StatusSuccess
            | StatusOutOfMemory
            | StatusReadOnlyDatabase
            | StatusUnbalancedFreezeThaw
            ...
  deriving (Eq)

instance Enum Status where
  ...
```

opaque pointer types

```
typedef struct _notmuch_database notmuch_database_t;
```

opaque pointer types

```
{#pointer *notmuch_database_t as DatabaseHandle foreign newtype #}
```

opaque pointer types

```haskell
newtype DatabaseHandle = DatabaseHandle (ForeignPtr DatabaseHandle)

withDatabaseHandle
  :: DatabaseHandle -> (Ptr DatabaseHandle -> IO b) -> IO b
withDatabaseHandle (DatabaseHandle fptr) =
  withForeignPtr fptr
```

double-pointer constructors

```
notmuch_status_t
notmuch_database_open (const char *path,
                       notmuch_database_mode_t mode,
                       notmuch_database_t **database);
```

double-pointer constructors

```
databaseOpen :: CString -> IO (Either Status DatabaseHandle)
databaseOpen path =
  alloca $ \ptr -> do
    result <- {#call notmuch_database_open #} path 1 ptr
    case toEnum (fromIntegral result) of
      StatusSuccess ->
        Right . DatabaseHandle <$> (peek ptr >>= newForeignPtr_)
      e ->
        pure (Left e)
```

iterator

```
notmuch_tags_t *
notmuch_message_get_tags (notmuch_message_t *message);

notmuch_bool_t
notmuch_tags_valid (notmuch_tags_t *tags);

const char *
notmuch_tags_get (notmuch_tags_t *tags);

void
notmuch_tags_move_to_next (notmuch_tags_t *tags);
```

iterator

```
tagsToList :: Tags -> IO [String]
tagsToList (Tags ptr) = go
  where
  go = test ptr >>= \valid -> case valid of
    0 -> pure []
    _ -> (:)
          <$> (get ptr >>= mk >>= \x -> next ptr $> x)
          <*> go

  test = {#call notmuch_tags_valid #}
  get = {#call notmuch_tags_get #}
  next = {#call notmuch_tags_move_to_next #}
  mk = peekCString
```

macros

```
void *talloc_steal(const void *new_ctx, const void *ptr);
```

macros

```
#if (__GNUC__ >= 3)
#define _TALLOC_TYPEOF(ptr) __typeof__(ptr)
#define talloc_steal(ctx, ptr) ({ \
  _TALLOC_TYPEOF(ptr) __talloc_steal_ret = (_TALLOC_TYPEOF(ptr)) \
    _talloc_steal_loc((ctx), (ptr), __location__); \
  __talloc_steal_ret; })
#else /* __GNUC__ >= 3 */
#define _TALLOC_TYPEOF(ptr) void *
#define talloc_steal(ctx, ptr) \
  (_TALLOC_TYPEOF(ptr)) _talloc_steal_loc((ctx), (ptr), __location__)
#endif /* __GNUC__ >= 3 */
void *_talloc_steal_loc(
  const void *new_ctx, const void *ptr, const char *location);
```

macros

Two options:

- bind to non-public API (e.g. `_talloc_steal_loc`)
- write *"c bits"*

external object lifecycles

```
notmuch_query_t *
notmuch_query_create (notmuch_database_t *database,
                      const char *query_string);

void
notmuch_query_destroy (notmuch_query_t *query);
```

external object lifecycles

```
query_create :: DatabaseHandle -> String -> IO (Query a)
query_create db s = withCString s $ \s' ->
  withDatabaseHandle db $ \db' ->
    {#call notmuch_query_create #} db' s'
      >>= fmap Query . newForeignPtr query_destroy

foreign import ccall "&notmuch_query_destroy"
  query_destroy :: FinalizerPtr Query
```

external object lifecycles

```
query_create :: DatabaseHandle -> String -> IO (Query a)
query_create db s = withCString s $ \s' ->
  withDatabaseHandle db $ \db' ->
    {#call notmuch_query_create #} db' s'
      >>= fmap Query . newForeignPtr query_destroy

foreign import ccall "&notmuch_query_destroy"
  query_destroy :: FunPtr (Ptr Query -> IO ())
```

external object lifecycles - beware

- hidden references in derived objects
- fancy allocators (e.g. *talloc)*

API safety

read-only mode

```
/* can return NOTMUCH_STATUS_READ_ONLY_DATABASE */
notmuch_status_t
notmuch_message_add_tag (notmuch_message_t *message, const char *tag);
```

read-only mode

```
{#enum notmuch_database_mode_t as DatabaseMode {underscoreToCase} #}
```

read-only mode

```
data DatabaseMode = DatabaseModeReadOnly
                  | DatabaseModeReadWrite

instance Enum DatabaseMode where
  ...
```

read-only mode

```haskell
{-# LANGUAGE DataKinds #-}

newtype Database (a :: DatabaseMode) = Database DatabaseHandle

withDatabase :: Database a -> (Ptr DatabaseHandle -> IO b) -> IO b
withDatabase (Database dbh) = withDatabaseHandle dbh

data Message (a :: DatabaseMode) = Message MessageHandle
```

read-only mode

```
type RW = 'DatabaseModeReadWrite  -- convenient alias

messageAddTag :: Message RW -> Tag -> IO ()
messageAddTag msg tag = void $ withMessage msg $
  tagUseAsCString tag . {#call notmuch_message_add_tag #}
```

locking

```
/* can return NOTMUCH_STATUS_READ_ONLY_DATABASE */
notmuch_status_t
notmuch_message_freeze (notmuch_message_t *message);

/* can return NOTMUCH_STATUS_READ_ONLY_DATABASE
   or NOTMUCH_STATUS_UNBALANCED_FREEZE_THAW      */
notmuch_status_t
notmuch_message_thaw (notmuch_message_t *message);
```

locking

```
{-# LANGUAGE DataKinds #-}
{-# LANGUAGE TypeFamilies #-}
{-# LANGUAGE TypeOperators #-}

import GHC.TypeLits

data Message (n :: Nat) (a :: DatabaseMode) = Message MessageHandle

messageAddTag :: Message n RW -> Tag -> IO ()
messageAddTag msg tag = void $ withMessage msg $
  tagUseAsCString tag . {#call notmuch_message_add_tag #}
```

locking

```
messageFreeze :: Message n RW -> IO (Message (n + 1) RW)
messageFreeze msg =
  withMessage msg {#call notmuch_message_freeze #} $> coerce msg


messageThaw :: (1 <= n) => Message n RW -> IO (Message (n - 1) RW)
message_thaw msg =
  withMessage msg {#call notmuch_message_thaw #} $> coerce msg
```

Performance

unsafe

```
{#call        notmuch_messages_valid #}


foreign import ccall        "notmuch.h notmuch_messages_valid"
  notmuch_messages_valid :: Messages -> IO CInt
```

unsafe

```
{#call unsafe notmuch_messages_valid #}
```

```
foreign import ccall unsafe "notmuch.h notmuch_messages_valid"
  notmuch_messages_valid :: Messages -> IO CInt
```

unsafe

Before:

```
total time  =        6.53 secs   (6530 ticks @ 1000 us, 1 processor)
total alloc = 260,249,536 bytes  (excludes profiling overheads)
```

After:

```
total time  =        3.73 secs   (3728 ticks @ 1000 us, 1 processor)
total alloc = 260,249,536 bytes  (excludes profiling overheads)
```

lazy iteration

```
messagesToList :: Messages -> IO [Message n a]
messagesToList (Messages ptr) = go
  where
  go = test ptr >>= \valid -> case valid of
    0 -> pure []
    _ -> (:)
          <$> (get ptr >>= mk >>= \x -> next ptr $> x)
          <*> go
```

lazy iteration

```
import System.IO.Unsafe (unsafeInterleaveIO)

messagesToList :: Messages -> IO [Message n a]
messagesToList (Messages ptr) = go
  where
  go = test ptr >>= \valid -> case valid of
    0 -> pure []
    _ -> (:)
          <$> (get ptr >>= mk >>= \x -> next ptr $> x)
          <*> unsafeInterleaveIO go
```

lazy iteration (search $*$, take 10, count tags)

Before:

```
total time  =        1.79 secs   (1795 ticks @ 1000 us, 1 processor)
total alloc =  59,500,568 bytes  (excludes profiling overheads)
```

After:

```
total time  =        0.07 secs   (68 ticks @ 1000 us, 1 processor)
total alloc =      79,960 bytes  (excludes profiling overheads)
```

lazy iteration (search *, count tags)

Before:

```
68,431,240 bytes maximum residency (9 sample(s))

total time  =        8.37 secs   (8370 ticks @ 1000 us, 1 processor)
total alloc = 218,627,008 bytes  (excludes profiling overheads)
```

After:

```
40,965,384 bytes maximum residency (8 sample(s))

total time  =        7.59 secs   (7586 ticks @ 1000 us, 1 processor)
total alloc = 257,666,440 bytes  (excludes profiling overheads)
```

things that weren't covered

- `foreign export` (callbacks)
- (un)marshalling C structs
- other FFIs (JVM, JavaScript, ...)
- other performance tips (avoiding unnecessary copies, interning, ...)

Questions?

https://speakerdeck.com/frasertweedale

@hackuador

purebred-mua/hs-notmuch