Shmencode

Caml-Shcaml by Example

Alec Heller Jesse A. Tov

College of Computer and Information Science
Northeastern University

ML Workshop
21 September 2008

Shell programming terrifies me. There is something about writing a simple shell script that is just much, much more unpleasant than writing a simple C program, or a simple COMMON LISP program, or a simple Mips assembler program.

—Olin Shivers, "A Scheme Shell"

A Confession

Sometimes I like Perl.

Perl? How Could You?

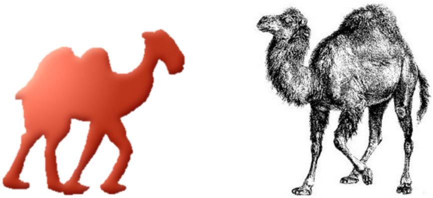
Perl gets things done.

- Easy access to system facilities
- Better abstractions than shell

OCaml?

What about OCaml?

- Better abstractions than Perl
- Dealing with Unix is a pain



Introducing Shcaml

What about OCaml? With Shcaml:

- Better abstractions than Perl
- Dealing with Unix is somewhat easier

Related Work

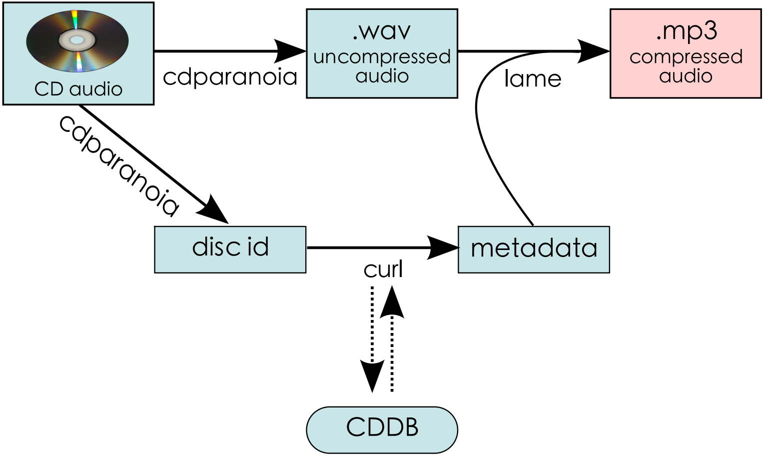
- Other work combining functional programming and the shell:
  - Scsh (Shivers 1994)
  - Cash (Verlyck 2002)
- Other work adding fancy metadata to shell pipelines:
  - Microsoft's Power Shell (Snover 2002)

Our Task

I would like to convert my CD collection to MP3.

Our Task

I would like to convert my CD collection to MP3.



Requirements

Two additional requirements:

- Parallelize ripping and encoding
- Have this working before lunch

Requirements

Two additional requirements:

- Parallelize ripping and encoding
- Have this working before lunch

Extracting Track Data

The program `cdparanoia` can print out track sizes and offsets.

```
# command "cdparanoia -Q 2>&1";;
```

Extracting Track Data

The program `cdparanoia` can print out track sizes and offsets.

```
# command "cdparanoia -Q 2>&1";;
- : ('_a elem -> text) fitting = <abstr>
#
```

Extracting Track Data

The program `cdparanoia` can print out track sizes and offsets.

```
# run (command "cdparanoia -Q 2>&1");;
```

Extracting Track Data

The program `cdparanoia` can print out track sizes and offsets.

```
# run (command "cdparanoia -Q 2>&1");;
cdparanoia III release 9.8 (March 23, 2001)

track_num = 1 start sector 0 msf: 0,2,0
track_num = 2 start sector 17868 msf: 4,0,18
track_num = 3 start sector 32216 msf: 7,11,41

Table of contents (audio tracks only):
track        length               begin        copy pre ch
===========================================================
  1.     17868 [03:58.18]        0 [00:00.00]    no   no  2
  2.     14348 [03:11.23]    17868 [03:58.18]    no   no  2
  3.     13799 [03:03.74]    32216 [07:09.41]    no   no  2
TOTAL    46015 [10:18.15]    (audio only)
- : Shcaml.Proc.status = Shcaml.Proc.WEXITED 0
#
```

Extracting Track Data

The program `cdparanoia` can print out track sizes and offsets.

```
# run (command "cdparanoia -Q 2>&1");;
cdparanoia III release 9.8 (March 23, 2001)

track_num = 1 start sector 0 msf: 0,2,0
track_num = 2 start sector 17868 msf: 4,0,18
track_num = 3 start sector 32216 msf: 7,11,41

Table of contents (audio tracks only):
track        length               begin        copy pre ch
===========================================================
  1.     17868 [03:58.18]        0 [00:00.00]    no   no  2
  2.     14348 [03:11.23]    17868 [03:58.18]    no   no  2
  3.     13799 [03:03.74]    32216 [07:09.41]    no   no  2
TOTAL    46015 [10:18.15]    (audio only)
- : Shcaml.Proc.status = Shcaml.Proc.WEXITED 0
#
```

Extracting Track Data

The program `cdparanoia` can print out track sizes and offsets.

```
# run begin
    command "cdparanoia -Q 2>&1"
    -| grep_string (starts_with " "))
  end;;
```

Extracting Track Data

The program `cdparanoia` can print out track sizes and offsets.

```
# run begin
    command "cdparanoia -Q 2>&1"
    -| grep_string (starts_with " "))
  end;;
  1.     17868 [03:58.18]        0 [00:00.00]    no    no  2
  2.     14348 [03:11.23]    17868 [03:58.18]    no    no  2
  3.     13799 [03:03.74]    32216 [07:09.41]    no    no  2
- : Shcaml.Proc.status = Shcaml.Proc.WEXITED 0
#
```

Interlude: What’s the Deal with Fittings?

Fittings are meant to evoke shell pipelines:

```
cdparanoia -Q 2>&1 \
  | grep '^ '
```

Interlude: What’s the Deal with Fittings?

Fittings are meant to evoke shell pipelines:

```
cdparanoia -Q 2>&1 \
  | grep '^ '
```

```
command "cdparanoia -Q 2>&1"
 -| grep_string (starts_with " ")
```

Interlude: What’s the Deal with Fittings?

Fittings are meant to evoke shell pipelines:

```
cdparanoia -Q 2>&1 \
  | grep '^ '
```

```
command "cdparanoia -Q 2>&1"
 -| grep_string (starts_with " ")
```

But:

- Fittings have types
- Fittings carry “hidden” metadata
- Fittings are first-class

Fittings Have Types

An `(`$\alpha \rightarrow \beta$`)fitting` is a pipeline component that consumes αs and produces βs.

Fittings Have Types

An $(\alpha \rightarrow \beta)$ `fitting` is a pipeline component that consumes αs and produces βs.

We compose them with the pipe:

```
val (-|) : (α → β) fitting
                → (β → γ) fitting
                        → (α → γ) fitting
```

Fittings Have Types

An ($\alpha \rightarrow \beta$) `fitting` is a pipeline component that consumes αs and produces βs.

We compose them with the pipe:

```
val (-|) : (α → β) fitting
              → (β → γ) fitting
                    → (α → γ) fitting
```

	are made out of	and transmit
Shell pipelines	Unix processes	untyped bytes.
Shcaml pipelines	Shcaml fittings	OCaml values.

Fittings Carry Metadata

```
val CdParanoia.fitting
  : unit →
      (<Line| delim: absent;  .. as α > →
       <Line| delim: present; .. as α >) fitting
```

`CdParanoia.fitting ()` is a fitting adaptor.

Fittings Carry Metadata

```
val CdParanoia.fitting
  : unit →
      (<Line| delim: absent;  .. as α > →
       <Line| delim: present; .. as α >) fitting
```

`CdParanoia.fitting ()` is a fitting adaptor.

- It does not change the “main” field of record
- It splits records into fields, which are then accessible by name:

  ```
  val Line.Delim.get_int
    : string → <Line| delim: present; .. > → int
  ```

Fittings Are First-Class

Evaluating a fitting does not “run” the fitting.

For that, we need fitting runners:

```
val run       : (text → 'o elem) fitting → Proc.status
```

Fittings Are First-Class

Evaluating a fitting does not “run” the fitting.

For that, we need fitting runners:

```
val run      : (text → 'o elem) fitting → Proc.status
val run_bg   : (text → 'o elem) fitting → Proc.t
```

Fittings Are First-Class

Evaluating a fitting does not “run” the fitting.

For that, we need fitting runners:

```
val run      : (text → 'o elem) fitting → Proc.status
val run_bg   : (text → 'o elem) fitting → Proc.t
val run_list : (text → 'o) fitting → 'o list
```

Fittings Are First-Class

Evaluating a fitting does not “run” the fitting.

For that, we need fitting runners:

```
val run      : (text → 'o elem) fitting → Proc.status
val run_bg   : (text → 'o elem) fitting → Proc.t
val run_list : (text → 'o) fitting → 'o list
val run_out  : ?procref:(Proc.t option ref)
                 → (text → 'o elem) → out_channel
val run_in   : ?procref:(Proc.t option ref)
                 → (text → 'o elem) → in_channel
```

Fittings Are First-Class

Evaluating a fitting does not “run” the fitting.

For that, we need fitting runners:

```
val run      : (text → 'o elem) fitting → Proc.status
val run_bg   : (text → 'o elem) fitting → Proc.t
val run_list : (text → 'o) fitting → 'o list
val run_out  : ?procref:(Proc.t option ref)
                  → (text → 'o elem) → out_channel
val run_in   : ?procref:(Proc.t option ref)
                  → (text → 'o elem) → in_channel
```

Now back to work . . .

Getting the Disc Id

We can write a function that produces the track data as a list:

```
let get_track_data () = run_list begin
  command "cdparanoia -Q 2>&1"
  -| grep_string (starts_with " ")
  -| CdParanoia.fitting ()
  -| sed (fun line → (Line.Delim.get_int "length" line,
                       Line.Delim.get_int "begin" line))
end
```

Getting the Disc Id

We can write a function that produces the track data as a list:

```
let get_track_data () = run_list begin
  command "cdparanoia -Q 2>&1"
  -| grep_string (starts_with " ")
  -| CdParanoia.fitting ()
  -| sed (fun line → (Line.Delim.get_int "length" line,
                      Line.Delim.get_int "begin" line))
end
```

To get the disc id, we pass the track lengths and offsets to the hash function:

```
let get_discid () = CddbID.discid (get_track_data ())
```

Filling in the Gaps

How are `CdParanoia` and `CddbId` defined?

```
module CdParanoia = Delim.Make_names(struct
  let options = { Delimited.default_options with
                    Delimited.field_sep = ' ' }
  let names   = [ "track"; "length"; "length-msh";
                  "begin"; "begin-msh"; "copy";
                  "pre"; "ch" ]
end)
```

`CdParanoia` is an *adaptor* module; we provide a variety of adaptors for different file formats.

Filling in the Gaps

How are `CdParanoia` and `CddbId` defined?

```
module CddbID : sig
  val discid : (int * int) list → string
end = struct
  open Int32
  open List

  let ((+), (%), (/),       (<<<), (|||)) =
      (add, rem, div, shift_left, logor)

  let ten = of_int 10
  let fps = of_int 75

  let sum_digits =
    let rec loop acc n = if n = zero then acc else loop (acc + n % ten) (n / ten) in
      loop zero

  let discid track_list =
    let lengths = map (fun (x,_) → of_int x) track_list in
    let offsets = map (fun (_,y) → of_int y) track_list in
    let ntracks = of_int (length lengths) in
    let n  = fold_left (fun x y → x + sum_digits (y / fps + of_int 2)) zero offsets in
    let t  = fold_left (+) zero lengths / fps in
    let id = (n % of_int 0xff <<< 24) ||| (t <<< 8) ||| ntracks in
      sprintf "%08lx" id
end
```

Next Stop CDDB

Now we need to query CDDB with the disc id.

Function `cddb_request` takes the id and returns the URL for our query:

```
let cddb_request discid =
  "http://freedb.freedb.org/~cddb/cddb.cgi" ^
  "?cmd=cddb+read+rock+" ^ discid ^ "&hello=" ^
  backquote "whoami" ^ "+" ^ backquote "hostname" ^
  "+shmendcode+0.1b&proto=6"
```

Next Stop CDDB

Now we need to query CDDB with the disc id.

Function `cddb_request` takes the id and returns the URL for our query:

```
let cddb_request discid =
  "http://freedb.freedb.org/~cddb/cddb.cgi" ^
  "?cmd=cddb+read+rock+" ^ discid ^ "&hello=" ^
  backquote "whoami" ^ "+" ^ backquote "hostname" ^
  "+shmendcode+0.1b&proto=6"
```

Function `curl` constructs a fitting that retrieves a URL:

```
let curl url = program "curl" ["-s"; url]
```

Let's give it a try. . . .

CDDB Query Results

```
# run begin
    curl (cddb_request (get_discid ()))
  end;;
```

CDDB Query Results

```
# run begin
    curl (cddb_request (get_discid ()))
  end;;
210 rock e882a039 CD database entry follows (until terminating ‘.’)
# xmcd
#
# Track frame offsets:
#       150
#       81375
#
# Disc length: 2280 seconds
#
DISCID=e882a039
DTITLE=Miles Davis / In a Silent Way
DYEAR=1969
DGENRE=Jazz
TTITLE0=Shhh/Peaceful
TTITLE1=In a Silent Way/It’s About That Time
EXTD=
.
- : Shcaml.Proc.status = Shcaml.Proc.WEXITED 0
```

CDDB Query Results

```
# run begin
    curl (cddb_request (get_discid ()))
  end;;
210 rock e882a039 CD database entry follows (until terminating ‘.’)
# xmcd
#
# Track frame offsets:
#       150
#       81375
#
# Disc length: 2280 seconds
#
DISCID=e882a039
DTITLE=Miles Davis / In a Silent Way
DYEAR=1969
DGENRE=Jazz
TTITLE0=Shhh/Peaceful
TTITLE1=In a Silent Way/It’s About That Time
EXTD=
.
- : Shcaml.Proc.status = Shcaml.Proc.WEXITED 0
```

CDDB Query Results

```
# run begin
    curl (cddb_request (get_discid ()))
    -| Key_value.fitting ()
  end;;
```

CDDB Query Results

```
# run begin
    curl (cddb_request (get_discid ()))
    -| Key_value.fitting ()
  end;;
examples/shmencode.ml: shtream warning: Key_value.splitter: key_value
line has 1 fields, needs 2
DISCID=e882a039
DTITLE=Miles Davis / In a Silent Way
DYEAR=1969
DGENRE=Jazz
TTITLE0=Shhh/Peaceful
TTITLE1=In a Silent Way/It's About That Time
EXTD=
examples/shmencode.ml: shtream warning: Key_value.splitter: key_value
line has 1 fields, needs 2
- : Shcaml.Proc.status = Shcaml.Proc.WEXITED 0
```

CDDB Query Results

```
# run begin
    curl (cddb_request (get_discid ()))
    -| Key_value.fitting ~quiet:true ()
  end;;
DISCID=e882a039
DTITLE=Miles Davis / In a Silent Way
DYEAR=1969
DGENRE=Jazz
TTITLE0=Shhh/Peaceful
TTITLE1=In a Silent Way/It's About That Time
EXTD=
- : Shcaml.Proc.status = Shcaml.Proc.WEXITED 0
```

CDDB Query Results

```
# run begin
    curl (cddb_request (get_discid ()))
    -| Key_value.fitting ~quiet:true ()
    -| sed (Line.select Line.Key_value.value)
  end;;
```

CDDB Query Results

```
# run begin
    curl (cddb_request (get_discid ()))
    -| Key_value.fitting ~quiet:true ()
    -| sed (Line.select Line.Key_value.value)
  end;;
e882a039
Miles Davis / In a Silent Way
1969
Jazz
Shhh/Peaceful
In a Silent Way/It's About That Time

- : Shcaml.Proc.status = Shcaml.Proc.WEXITED 0
```

Parsing CDDB Results (1)

The `Key_value` adaptor gets us key-value pairs. We need:

- Whole album metadata: artist, title, year, genre

- Per-track metadata: track number and title

Parsing CDDB Results (1)

The `Key_value` adaptor gets us key-value pairs. We need:

- Whole album metadata: artist, title, year, genre

  A `string list` of command-line flags

- Per-track metadata: track number and title

Parsing CDDB Results (1)

The `Key_value` adaptor gets us key-value pairs. We need:

- Whole album metadata: artist, title, year, genre

  A `string list` of command-line flags

- Per-track metadata: track number and title

```
type track = {
  index: int;
  title: string;
  wav:   string;
  mp3:   string;
}
```

Parsing CDDB Results (1)

The `Key_value` adaptor gets us key-value pairs. We need:

- Whole album metadata: artist, title, year, genre

  A `string list` of command-line flags

- Per-track metadata: track number and title

```
type track = {
  index: int;
  title: string;
  wav:   string;
  mp3:   string;
}
```

We fold over the stream of key-value pairs to build these.

`let parse_cddb_line =` ⟨22 lines⟩

Parsing CDDB Results (2)

A function that queries CDDB and returns the parsed result:

```
let get_cddb discid =
  let (tracks, album_tags) =
    Shtream.fold_left parse_cddb_line ([], [])
      (run_source begin
          curl (cddb_request discid)
          -| Key_value.fitting ~quiet:true ()
       end) in
  (List.rev tracks, album_tags)
```

Parsing CDDB Results (2)

A function that queries CDDB and returns the parsed result:

```
let get_cddb discid =
  let (tracks, album_tags) =
    Shtream.fold_left parse_cddb_line ([], [])
      (run_source begin
         curl (cddb_request discid)
         -| Key_value.fitting ~quiet:true ()
       end) in
  (List.rev tracks, album_tags)
```

```
# get_cddb (get_discid ());;
```

Parsing CDDB Results (2)

A function that queries CDDB and returns the parsed result:

```
let get_cddb discid =
  let (tracks, album_tags) =
    Shtream.fold_left parse_cddb_line ([], [])
      (run_source begin
         curl (cddb_request discid)
         -| Key_value.fitting ~quiet:true ()
       end) in
  (List.rev tracks, album_tags)
```

```
# get_cddb (get_discid ());;
- : track list * string list =
([{index = 1; title = "Shhh/Peaceful"};
  {index = 2;
    title = "In a Silent Way/It's About That Time"}],
 ["--tg"; "Jazz"; "--ty"; "1969"; "--ta";
  "Miles Davis"; "--tl"; "In a Silent Way"])
```

Let ’Er Rip (and Encode)

How should we call the ripping and encoding programs?
We’ll make fittings:

Let ’Er Rip (and Encode)

How should we call the ripping and encoding programs? We’ll make fittings:

```
let rip track =
  program "cdparanoia"
    ["--"; string_of_int track.index; track.wav]
```

Let ’Er Rip (and Encode)

How should we call the ripping and encoding programs? We’ll make fittings:

```
let rip track =
  program "cdparanoia"
    ["--"; string_of_int track.index; track.wav]
    />/ [ 2 %>* ‘Null; 1 %>& 2 ]
```

Let 'Er Rip (and Encode)

How should we call the ripping and encoding programs? We'll make fittings:

```
let rip track =
  program "cdparanoia"
    ["--"; string_of_int track.index; track.wav]
    />/ [ 2 %>* `Null; 1 %>& 2 ]

let encode album_tags track =
  program "lame"
    (album_tags @
     ["--tn"; string_of_int track.index;
      "--tt"; track.title; "--quiet";
      track.wav; track.mp3])
```

Let ’Er Rip (and Encode)

How should we call the ripping and encoding programs? We’ll make fittings:

```
let rip track =
  program "cdparanoia"
    ["--"; string_of_int track.index; track.wav]
    />/ [ 2 %>* `Null; 1 %>& 2 ]

let encode album_tags track =
  program "lame"
    (album_tags @
     ["--tn"; string_of_int track.index;
      "--tt"; track.title; "--quiet";
      track.wav; track.mp3])
  &&^ program "rm" [track.wav]
```

Ripping, Then Encoding

At this point, we can rip a CD sequentially:

Ripping, Then Encoding

At this point, we can rip a CD sequentially:

1. Compute the disc id

```
let discid          = get_discid ()                   in
```

Ripping, Then Encoding

At this point, we can rip a CD sequentially:

1. Compute the disc id
2. Query CDDB and parse the response

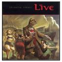
```
let discid          = get_discid ()                        in
let (tracks, album) = get_cddb discid                      in
```

Ripping, Then Encoding

At this point, we can rip a CD sequentially:

1. Compute the disc id
2. Query CDDB and parse the response
3. Rip each track

```
let discid          = get_discid ()                  in
let (tracks, album) = get_cddb discid                in
let rip_fittings    = List.map rip tracks            in
```



Ripping, Then Encoding

At this point, we can rip a CD sequentially:

1. Compute the disc id
2. Query CDDB and parse the response
3. Rip each track
4. Encode each track

```
let discid          = get_discid ()                      in
let (tracks, album) = get_cddb discid                    in
let rip_fittings    = List.map rip tracks                in
let encode_fittings = List.map (encode album) tracks in
```

Ripping, Then Encoding

At this point, we can rip a CD sequentially:

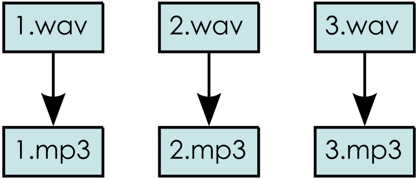
1. Compute the disc id
2. Query CDDB and parse the response
3. Rip each track
4. Encode each track

```
let discid           = get_discid ()                    in
let (tracks, album) = get_cddb discid                  in
let rip_fittings     = List.map rip tracks              in
let encode_fittings = List.map (encode album) tracks in
  run ~>>(rip_fittings @ encode_fittings)
```

We'd like our program to take advantage a multicore machine.

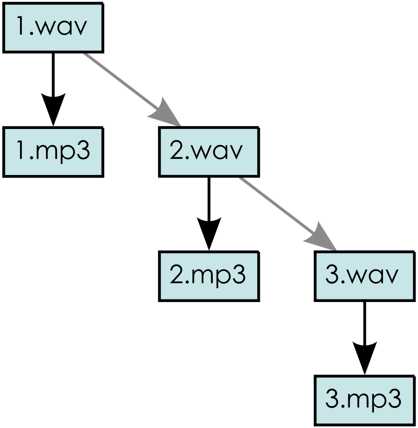
Parallelization Constraints

- We must rip each track before encoding it



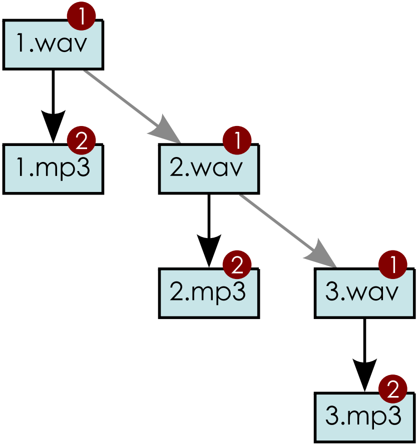
Parallelization Constraints

- We must rip each track before encoding it
- We can rip at most one track at once



Parallelization Constraints

- We must rip each track before encoding it
- We can rip at most one track at once
- Prefer ripping over encoding



Building the Dependency DAG

```
let build_dag (tracks, album) =
  let each (mp3s, prev) track =
    let wav = DepDAG.make ~prio:1 {|
                printf "Ripping  %s\n%!" track.wav;
                run_bg (rip track)
              |} prev in
    let mp3 = DepDAG.make ~prio:2 {|
                printf "Encoding %s\n%!" track.mp3;
                run_bg (encode album track)
              |} [wav] in
    (mp3::mp3s, [wav]) in
  let mp3s, _ = List.fold_left each ([], []) tracks in
    DepDAG.make_par mp3s
```

Putting It All Together

```
let main () =
  let opts     = Flags.go "-N <max-procs:int>" in
  let n        = opts#int ~default:2 "-N" in
  let discinfo = get_cddb (get_discid ()) in
    DepDAG.run ~n (build_dag discinfo)
```

Thank You

Contact us or try Shcaml:

- `tov@ccs.neu.edu`
- `http://www.ccs.neu.edu/~tov/shcaml/`