DYALOG

Elsinore 2023

Revisiting ⎕SH and ⎕CMD

Peter Mikkelsen

Revisiting *monadic* □SH and □CMD

Or: revisiting the way we run shell commands

Monadic □SH and □CMD are old and might work just fine for you

- There are no plans of changing or removing them

But they are not without issues

Explore what can be done better

- Proposal for a new system function that fixes the issues

If all goes well, we are aiming for 20.0

What is ⍞SH used for

Used to run an external program or shell command

- Mount a network drive, run test scripts, git status…

Two main outcomes

- Side effects
- Output, collected as an APL array (text)

Example – files containing a word

Example – files containing a word

```
  □SH 'grep example *.c'
a.c:/* The following example shows
a.c:example_mode = 0;
b.c:if(example_mode)
```

Output is *printed* to the session

Looks like the result of □SH is a matrix

Fine for interactive use – slow output printed as it is produced

Example – files containing a word

```
   x←⎕SH 'grep example *.c'
   ⍴x
3
   ↑x
a.c:/* The following example shows
a.c:int example_mode = 0;
b.c:if(example_mode)
```

Example – files containing a word

```
      ⎕SH'grep axample *.c'    ⍝ No output
   x←⎕SH'grep axample *.c'
DOMAIN ERROR: Command interpreter returned
failure code 1
   x←⎕SH'grep axample *.c'
      ^
```

Exit code is checked, but only when the result is used

Example – content of files

```
   x←⎕SH'cat file1 file2'
DOMAIN ERROR: …
```

`file1` exists and is printed to standard output

`file2` doesn't exist

- An error message is produced on standard error
- Wouldn't it be nice to know about that

Limitations of the current □SH

What happens to error output?

How can I control input?

What about slow commands?

What about output before an error?

What if the output isn’t text at all?

What about environment variables?

What happens to error output?

Right now, only standard output is collected

- Standard error is very often useful. It isn't *only* used for errors.

In a normal shell, stdout and stderr both go to the window

- The interleaving is done by the operating system
- Hard to tell the two apart by default

Having the two as separate parts of the result might be useful

Limitations of the current □SH

What happens to error output?

How can I control input?

What about slow commands?

What about output before an error?

What if the output isn’t text at all?

What about environment variables?

How can I control input?

Some programs expect to be able to read data from standard input

- Usually, the lines of text typed at a terminal window

Unix tools often read on stdin and write to stdout

- Example: `echo hello | tr a-z A-Z`
- Would be nice if we could specify the input from the APL side

Other times, no input at all is needed

Limitations of the current □SH

What happens to error output?

How can I control input?

What about slow commands?

What about output before an error?

What if the output isn't text at all?

What about environment variables?

What about slow commands?

- ⎕SH can be hard to stop once it is running
  - Issues interrupting it from RIDE
  - If ⎕SH is interrupted, what happens to the program?
- It blocks the rest of the APL threads

```
⎕SH&'update-system' ⋄ ⎕←'zzz'
```

- Being a thread-switch point makes slow commands easier to deal with. ⎕TKILL to stop it.

Limitations of the current □SH

What happens to error output?

How can I control input?

What about slow commands?

What about output before an error?

What if the output isn’t text at all?

What about environment variables?

What about output before an error?

Exit codes are used to indicate success/failure

- By convention 0 means success
- Giving a `DOMAIN ERROR` prevents the user from having to check

However, `DOMAIN ERROR` might be a bad idea

- Some programs return non-zero even when they succeed
- All output so far is completely lost
- Figuring out what went wrong becomes a challenge

Limitations of the current □SH

What happens to error output?

How can I control input?

What about slow commands?

What about output before an error?

What if the output isn’t text at all?

What about environment variables?

What if the output isn’t text at all?

We are used to working with text in the shell

Some commands might output non-text data

```
cat hello.png
```

Letting the user choose which APL type to interpret it as could be helpful

Similarly, ⎕NREAD also doesn’t always read data as text

The same is of course true for input to the program

Limitations of the current □SH

What happens to error output?

How can I control input?

What about slow commands?

What about output before an error?

What if the output isn’t text at all?

What about environment variables?

What about environment variables?

Environment variables is one of the major ways of passing data to a child program

- Standard input and program arguments two other ways

The interpreter already has a set of variables

- They can be inherited or cleared

Custom environment variables for each invocation

Limitations of the current □SH

What happens to error output?

How can I control input?

What about slow commands?

What about output before an error?

What if the output isn’t text at all?

What about environment variables?

A new system function - ⎕SHELL

`(Exit Ids Content)←⎕SHELL cmd`

`Exit:` The exit code of the program

`Ids:` A vector of collected stream ids

`Contents:` A vector of content collected

One element for each stream

Standard output (stream 1) becomes: `(Ids⍳1)⊃Contents`

SHELL cmd

The command `cmd` can be specified in two ways

Character vector: evaluated using the system shell

- Easy to use shell features, such as pipelines

Nested character vector: `program arg1 arg2 …`

- No need to worry about the specific shell, and its argument quoting rules

`'rm "file name"'` vs `'rm' 'file name'`

Control via variant options

- `'ExitCheck' bool`
- `'InheritEnv' bool`
- `'WorkingDir' path`
- `'Window' winparam`

  Same as with CMD

- `'Shell' shellSpec`

  `'CMD.EXE' '/C'`

  `'/bin/sh' '-c'`
- `'Env' environment`
- `'Redirect' redirs`

  Details on the next 2 slides

Environment variables - `'Env'`

A n-by-2 matrix of extra environment variables

- Or a vector, which is treated as a 1 row matrix

Each row consists of two character-vectors: `('Name' 'Val')`

Variables are added only in the child process

- Not even temporarily in the parent process (the interpreter)
- If it is already set, the new value overwrites the old

Redirections - 'Redirect'

A n-by-3 matrix of redirections that should be setup

Or a vector, which is treated as a 1 row matrix

Each row is a redirection: `(Stream Mode X)`

Changes what the program ”sees” for `Stream`

`Mode` is either `'From'` or `'To'`

`X` is a source or a target, depending on `Mode`

Redirection targets (mode `'To'`)

`'Null'` – completely discard the output

`('File' tie)` – Send the output to some native file

`('Stream' n)` – Send the output to another stream

Useful to redirect standard error to standard output

`'Array'` – Collect the output as an array (vector of lines)

`('Array' type)` – Output as a vector of some type

Collect as boolean with `(N 'To' ('Array' 11))`

Redi

- As if the output was redirected to a file, and then read using ⎕NGET
- The encoding is guessed based on content
- Should it be possible to specify it?

```
('Array' 'UTF-8')
```

'Null

('Fil

('Str

Useful to redirect standard error to standard output

`'Array'` – Collect the output as an array (vector of lines)

`('Array' type)` – Output as a vector of some type

Collect as boolean with `(N 'To' ('Array' 11))`

Redirection sources (mode `'From'`)

`'Null'` – Provide no input at all

`('File' tie)` – Connect the file as input

`('Array' type data)` – Use the data array as input

`('Array' data)`

An alias for `('Array' (⎕DR data) data)`

To use a character vector X as stdin: `(0 'From' ('Array' X))`

Default redirections

- SHELL always sets up defaults for the three standard streams

0	From	Null
1	To	Array
2	To	Stream 1

Example – content of files

```
      x←⎕SH'cat file1 file2'
DOMAIN ERROR: …
```

Example from earlier

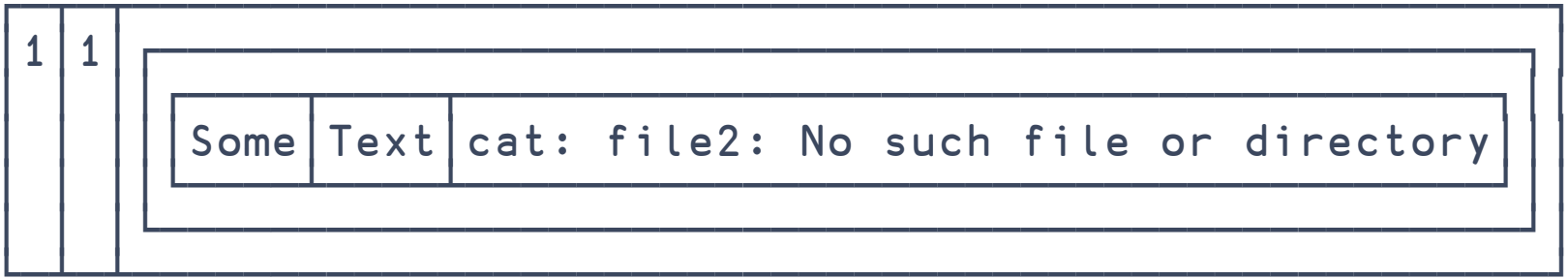
`file1` exists and is printed to standard output

`file2` doesn't exist

- An error message is produced on standard error
- Wouldn't it be nice to know about that

Example – content of files

```
SHELL'cat file1 file2'
```



Now we can see what is going on

Example – content of files

```
  s←⎕SHELL⍠'Redirect' (2 'To' 'Array')
  (exit ids contents)←s 'cat file1 file2'
  exit
1
  ↑¨contents[ids⍳1 2] ⍝ matrix stdout and stderr
```

Some Text	cat: file2: No such file or directory

Example – translate to uppercase

Note: I am not suggesting this is a good way to uppercase

```
   x←'hello dyalog 23'
   s←⎕SHELL⍠'Redirect' (0 'From' ('Array' x))
   ↑3 1⊃s'tr a-z A-Z'
HELLO DYALOG 23
```

Conclusion

There are many things that could be improved about □SH

- Impossible to extend it without introducing breaking changes
- The current monadic □SH/□CMD isn't going away

□SHELL allows for much finer control of input/output

- At the slight cost of a more complicated result value
- Hopefully will be a part of version 20.0

Thank you for listening ☺