

Offset	Topic
00:17	<ul style="list-style-type: none"> <li>• <b>Intro</b> <ul style="list-style-type: none"> <li>• Dragon*Con reminder</li> <li>• Interesting discussion on the 8/17 week in review</li> <li>• Questions for audio promo</li> </ul> </li> </ul>
04:00	<ul style="list-style-type: none"> <li>• <b>Word of the Week: dike</b> <ul style="list-style-type: none"> <li>• <a href="http://catb.org/jargon/html/D/dike.html">http://catb.org/jargon/html/D/dike.html</a></li> </ul> </li> </ul>
05:03	<ul style="list-style-type: none"> <li>• <b>Hacking 101: Errors and Exceptions</b> <ul style="list-style-type: none"> <li>• Regardless of what language you are learning, using, need to know how to indicate an error <ul style="list-style-type: none"> <li>• You may be able to rely on built in error handling, for common functions</li> <li>• Files, network, etc.</li> <li>• At some point, you are going to encounter a problem specific to your own code</li> <li>• Need to understand what is available to indicate errors</li> <li>• Errors are key to intentional programming</li> <li>• You want users, other hackers to be able to understand an error</li> <li>• See Inner Chapter on failures and errors</li> <li>• <a href="http://thecommandline.net/2007/01/10/the-command-line-2007-01-10-comment-line-360-252-7284/">http://thecommandline.net/2007/01/10/the-command-line-2007-01-10-comment-line-360-252-7284/</a></li> </ul> </li> <li>• In procedural, functional languages, how do you indicate a problem? <ul style="list-style-type: none"> <li>• Return a different value, usually one that is obviously not valid</li> <li>• Makes the caller check for valid returns, versus invalid returns</li> <li>• Relies on conventions, documentation</li> <li>• If there is no real return, this can work well</li> <li>• Error code or zero return for success</li> <li>• If there is a real return code, this can get awkward</li> <li>• Many simpler languages bolt together program exit with error</li> <li>• Makes the problem that of the user, not code</li> <li>• So consistent with Unix programs, there are conventions, zero for success, non-zero for failures</li> </ul> </li> <li>• Problems with error codes <ul style="list-style-type: none"> <li>• Forcing an error value to fit the return type if it is not a simple number <ul style="list-style-type: none"> <li>• Could use a wrapper type but is clunky</li> <li>• A union where one part is the valid return, the other the error</li> <li>• Forces a dereference every time, regardless</li> <li>• Makes your code less clear, need to wrap/unwrap or use facilities to do so</li> <li>• Silver lining is you can add more data to the error than just a simple value</li> </ul> </li> </ul> </li> </ul> </li> </ul>

## Offset

## Topic

- Unfortunately, that makes the code more complex, may not be worth it
- Could also use a global or a side affect on a struct argument
  - Makes the error non-local to your return
  - Not much different than checking, comparing return value
- Structured programming can help further
  - Already mentioned wrapper type
  - Using special structs for errors or error values
  - Standard C library does this, also has to use a global to prevent scoping problems
  - C does this for exiting out of the program itself
  - This can still yield errors if you reference the errors incorrectly
  - Does not help if you want to deal with the problem anywhere other than program exit
- Knowing, checking difference between valid return and error
  - Constants and enumerations can help
  - Fixed values against which you can compare
  - Still doesn't help describe or explain the error directly
  - Code can break if list of error values grows over time
- Multiple places in the code can encounter the same error condition
  - If error is tied to program exit, what if you want to try to recover?
  - A simple return code doesn't say anything about where the error occurred
  - Again, could use a struct instead of a simple value
    - Wasted when the caller doesn't care
    - Can add complexity, regardless, making building errors harder
- Exceptions
  - Meant to address many of the issues with error return values, other techniques
  - A first class object in OO programs
    - Can carry additional information, tied to specific sub-type
    - Creation is usually the same or similar to regular object
  - Exceptions get thrown, not returned
    - Use a special statement to raise or throw
    - Completes the current scope, just like a regular return
    - Often can execute a final block, allow unconditional code like resource clean up
  - Caller uses a different construction to catch them or let them bubble up
  - Distinct from getting and checking a return
  - A natural way to express normal completion separate from error completion
  - Allows errors to be dealt with, if possible, before program exit

- In most languages that support, aren't required to catch exceptions
- If you don't need to or cannot handle error, just let it keep going up the call stack
- Most exception implementations also capture information from the call stack
  - When the same error is reused, tells specifically where the error came from
  - Usually called a track, like a stack trace or trace back
- Error philosophies
  - Look before you leap
    - Check for any problems ahead of time
    - Are the inputs valid?
    - Is the state of the system correct?
    - Are the side effects, return value correct?
  - Formal version, to the extreme, design by contract
    - [http://en.wikipedia.org/wiki/Design\\_by\\_contract](http://en.wikipedia.org/wiki/Design_by_contract)
    - Gives names to all the particular aspects
    - Means the program fails early, consistently
    - If the developer understands the full contract, understands how to diagnose, repair very specifically
  - Can you really anticipate all of the problems ahead of time?
    - When failure is expensive, makes sense
    - Like rolling back a transaction
    - For a library, helps communicate expectations
  - Ask forgiveness rather than seek permission
    - With a rich error mechanism like exceptions, just let errors happen
    - Relies on complete, clear descriptions
    - Assumes most errors really cannot be recovered from
    - Makes sense when cost of error is cheap
    - Is the reason why many, if not most, exceptions are unchecked, do not need to be caught
  - As much as DbC appeals, I increasingly lean towards ask forgiveness
  - Intentional programming helps make up the difference
  - Be clear, explain well why the error happened, even provide guidance on how to correct

- Contact me
  - Email to [feedback@thecommandline.net](mailto:feedback@thecommandline.net)
  - Web site at <http://thecommandline.net/>
  - IM to [command.line@skype](skype:command.line@skype)
  - Listener comment line is 240-949-2638

## Offset

## Topic

- del.icio.us tag is "for:cmdln"
- <http://twitter.com/cmdln>
- I'd like to thank libsyn.com for AAC hosting and Wouter de Bie for MP3 hosting
- These notes and the show audio and music are covered by a Creative Commons license
  - <http://creativecommons.org/licenses/by-nc-sa/3.0/us/>
  - Attribution, non-commercial, share alike