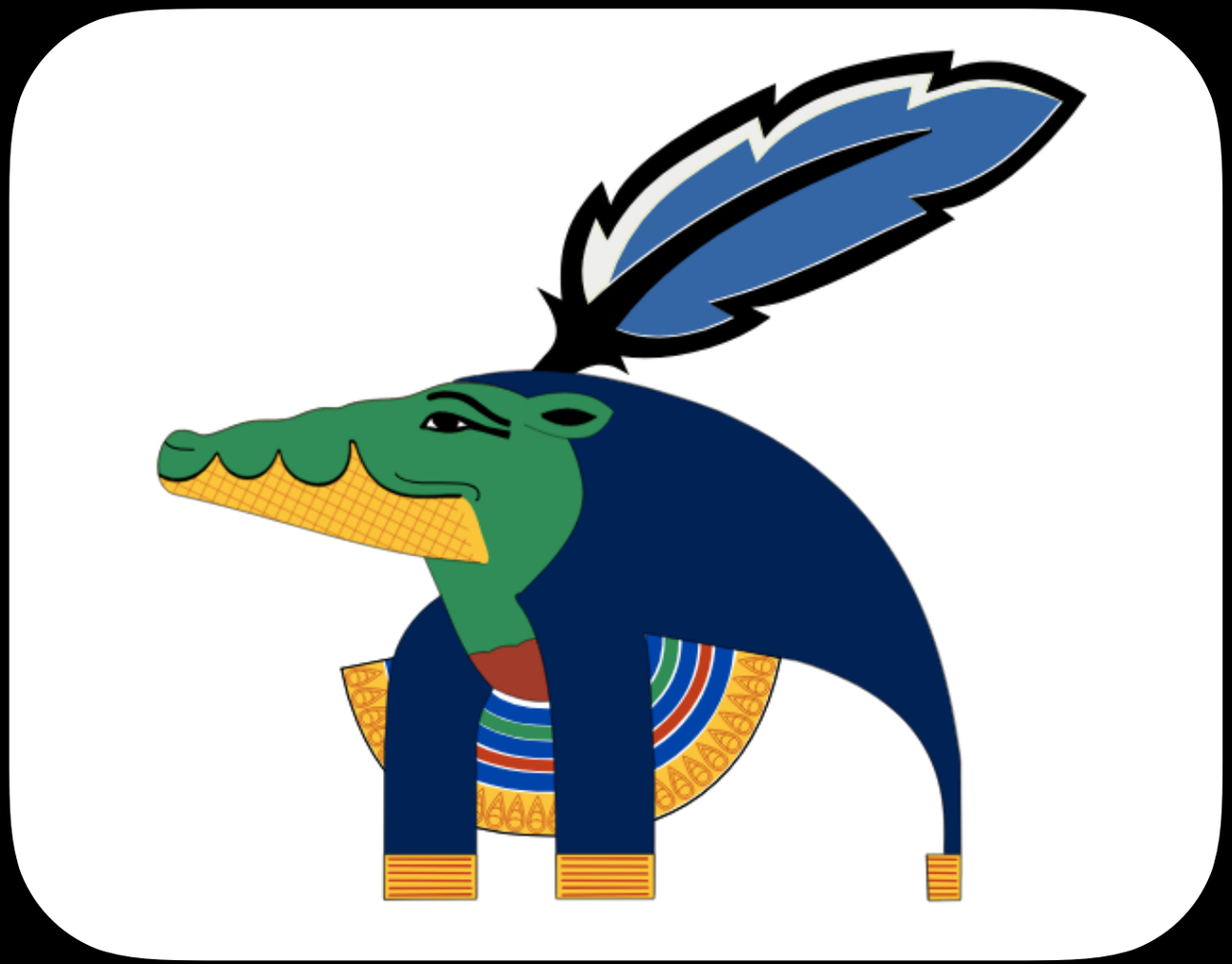
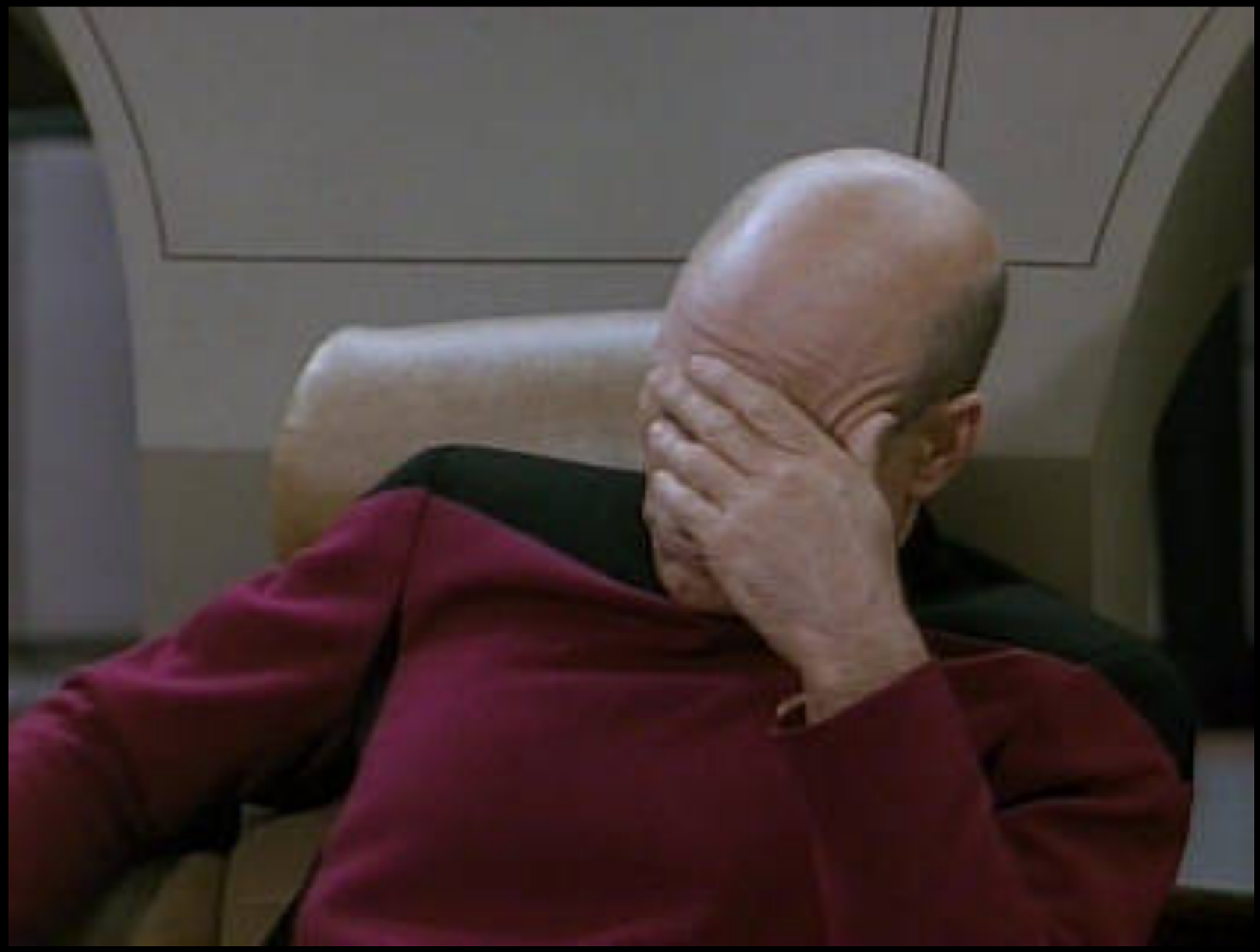
# The Sobyk Binary Distribution



http://www.etoyoc.com/yoda/papers/tcl2019.Sobyk_Slides.pdf



**“Not this again, Hypnotoad”**

*–Everyone in this room*

You broke my 10 year old production code in a point release.

Namespace deletion on 8.6 was fundamentally flawed and needed to be refactored

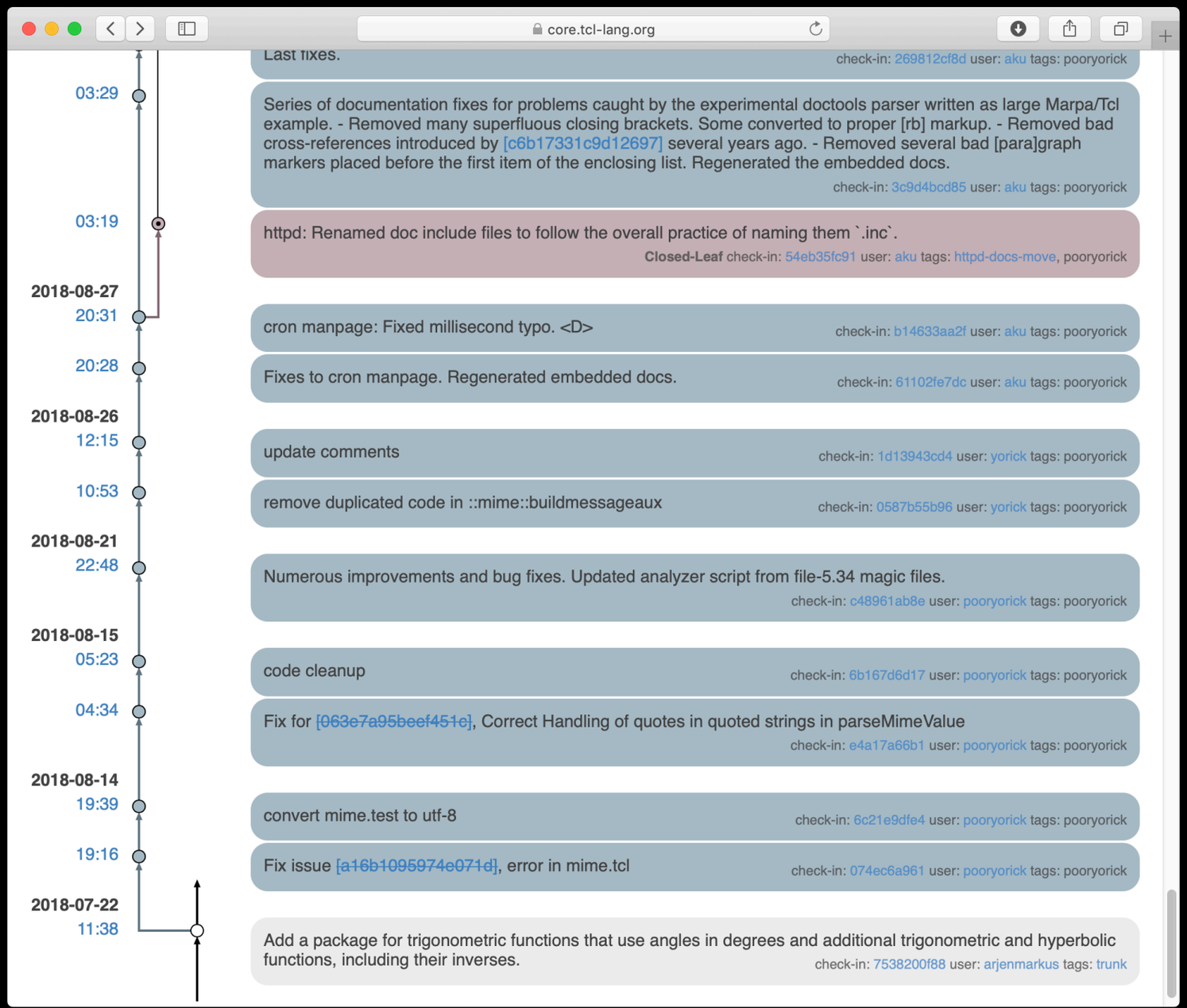
```
#    the <Destroy> we are seeing is intended for us.
  ###
  method Hull_Destroy {} {
    ###
    # Destroy our Tk representation
    ###
    my variable tkalias
    if {[info exists tkalias]} {
      set alias $tkalias
    } else {
      set alias {}
    }
    if {$alias ne {}} {
      my Hull_Unbind $alias
    }
    catch {my action destroy}

    # Destroy an alias we may have created
    if { $alias ne {} && [winfo exists $alias] } {
      catch {rename [namespace current]::tkwidget {}}
    } else {
      set hull [my organ hull]
      if { $hull ne "."} {
        catch {::destroy $hull}
      }
    }

    ###
    # Clean up children
    ###
    foreach subobj [info command [self]/*] {
      catch {$subobj destroy}
    }
    foreach subobj [info command [self].*] {
      if {[winfo exists $subobj]} continue
      catch {$subobj destroy}
    }



  }
```

```
#    the <Destroy> we are seeing is intended for us.
  ###
  method Hull_Destroy {} {
    ###
    # Destroy our Tk representation
    ###
    my variable tkalias
    set tkWidget {}
    if {[info exists tkalias]} {
      set tkWidget $tkalias
    }
    if {$tkWidget eq {}} {
      set tkWidget [my widget hull]
    }
    if {$tkWidget eq {}} {
      set tkWidget [my organ hull]
    }
    if {$tkWidget ne {}} {
      my Hull_Unbind $tkWidget









    }

    ###
    # Clean up children
    ###
    foreach subobj [info command [self]/*] {
      catch {$subobj destroy}
    }
    foreach subobj [info command [self].*] {
      if {[winfo exists $subobj]} continue
      catch {$subobj destroy}
    }

    catch {my action destroy}

    # Destroy an alias we may have created
    if { $tkWidget ni {. {}} && [winfo exists $tkWidget] } {
      ::destroy $tkWidget
    }
  }
```

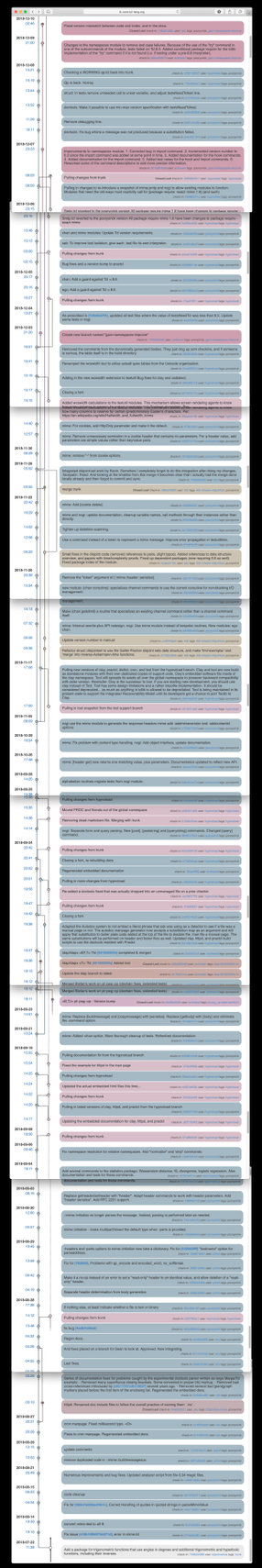
I had to back out 6 months of changes in fossil because you checked breakage straight into trunk

The MIME package was fundamentally flawed and needed to be refactored from first principles to be right.

# I had to back out a little bit of history on Tcllib



# Just a bit...

# What was all that?

- Each teal commit was originally to the trunk of Tcllib
- Tests passed perfectly fine in the modules the particular developer was working on
- They outright broke other modules
- Those changes polluted any merge with trunk made after that date
- By the end, developers like me couldn’t even run some tests because modules started require Tcl 8.6.9

# And why couldn’t I test with 8.6.9?

```
#    the <Destroy> we are seeing is intended for us.
  ###
  method Hull_Destroy {} {
    ###
    # Destroy our Tk representation
    ###
    my variable tkalias
    if {[info exists tkalias]} {
      set alias $tkalias
    } else {
      set alias {}
    }
    if {$alias ne {}} {
      my Hull_Unbind $alias
    }
    catch {my action destroy}

    # Destroy an alias we may have created
    if { $alias ne {} && [winfo exists $alias] } {
      catch {rename [namespace current]::tkwidget {}}
    } else {
      set hull [my organ hull]
      if { $hull ne "."} {
        catch {::destroy $hull}
      }
    }

    ###
    # Clean up children
    ###
    foreach subobj [info command [self]/*] {
      catch {$subobj destroy}
    }
    foreach subobj [info command [self].*] {
      if {[winfo exists $subobj]} continue
      catch {$subobj destroy}
    }



  }
```

```
#    the <Destroy> we are seeing is intended for us.
  ###
  method Hull_Destroy {} {
    ###
    # Destroy our Tk representation
    ###
    my variable tkalias
    set tkWidget {}
    if {[info exists tkalias]} {
      set tkWidget $tkalias
    }
    if {$tkWidget eq {}} {
      set tkWidget [my widget hull]
    }
    if {$tkWidget eq {}} {
      set tkWidget [my organ hull]
    }
    if {$tkWidget ne {}} {
      my Hull_Unbind $tkWidget









    }

    ###
    # Clean up children
    ###
    foreach subobj [info command [self]/*] {
      catch {$subobj destroy}
    }
    foreach subobj [info command [self].*] {
      if {[winfo exists $subobj]} continue
      catch {$subobj destroy}
    }

    catch {my action destroy}

    # Destroy an alias we may have created
    if { $tkWidget ni {. {}} && [winfo exists $tkWidget] } {
      ::destroy $tkWidget
    }
  }
```

# Core Development

- Core development is driven by TIPs
- TIPs are driven by developer interest
- Developers should not be lashed into making broken code work

# Tcl Users

- Many are not even aware they are running Tcl
- Wrap the workings of production software around semi-supported extensions
- Expect to be able to write code once, ensure that it works, and not have to re-write it
- Care less about “fundamental flaws” and more by the cost of maintaining software

# Sean’s Stupid Idea

- Let’s make an environment where Tcl Developers and Tcl Users can both agree should always work
- Let’s make that tool something that can run in any environment
- Let’s make the process of full-up testing something that is repeatable

# Sobyk

- Sobyk doesn’t stand for anything.
- The name was picked out of a fantasy name generator tool because it spoke to me

- Is a distinct enough pattern that application devs who cargo-cult can simply globally search for references to “Sobyk” and replace it with their own product’s name

# Sobek

Egyptian God

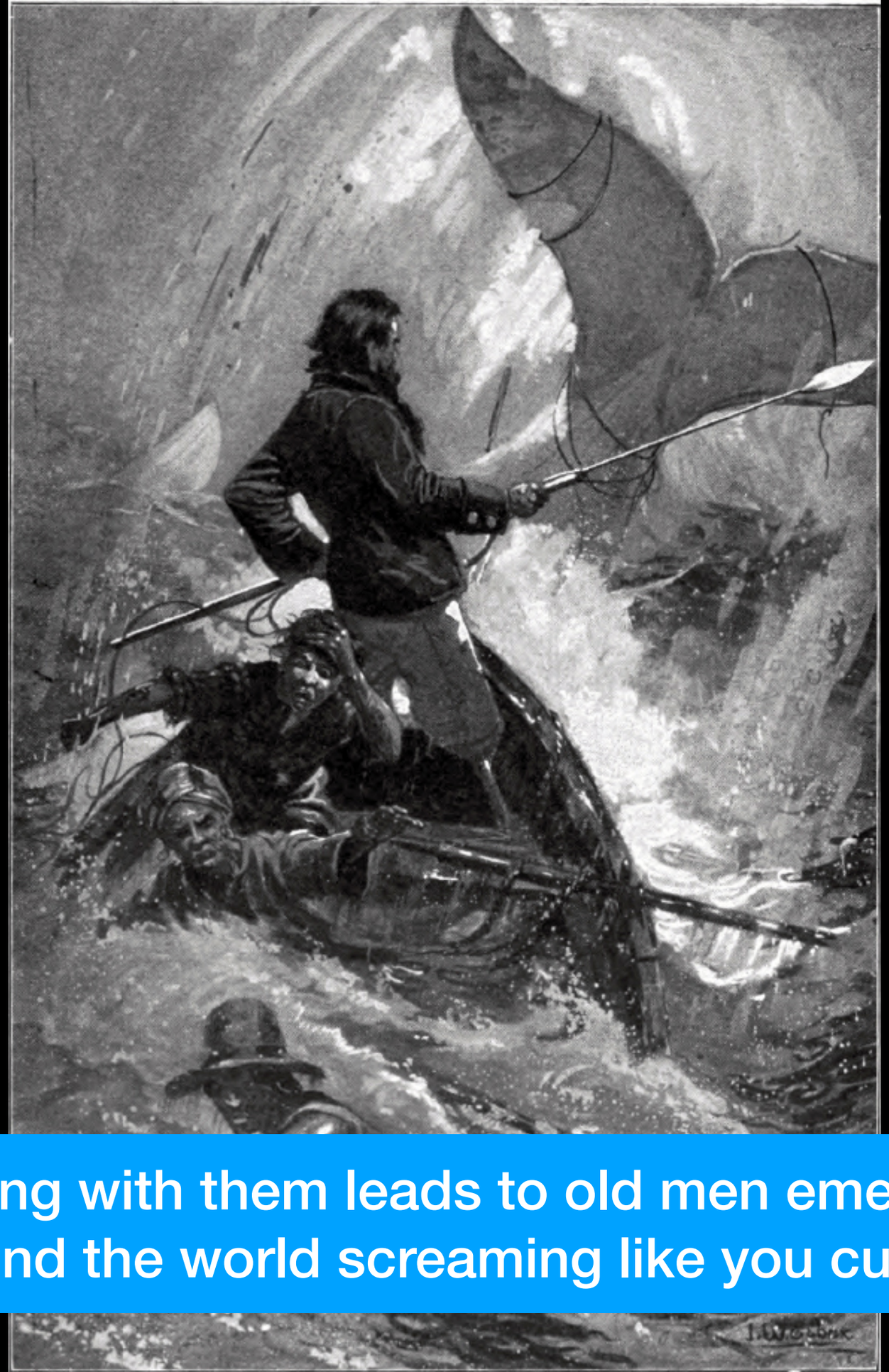
(see: https://en.wikipedia.org/wiki/Sobek)

# Moby Applications

- Projects that ship one binary that is the entire working environment
- Need to maintain the ability to re-run processes with older releases to compare output
- Are end-user applications which need to work on Mac and Windows, simultaneously
- Documents and simulations developed for these can involve man-years of effort and multi-year support contracts

# Moby Applications

- Major Efforts
- Old code
- Binary Packages
- Year’s long Commitments



And messing with them leads to old men emerging from halfway around the world screaming like you cut their leg off

# Moby Apps

| Product | Developer | Current State |
|---|---|---|
| The Integrated Recoverability Model | T&E Solutions, LLC | Actively Developed |
| BRL Cad | Army Research Labs | Actively Maintained, Stuck in Tcl 8.5 |
| SOAR (Tcl Implementation) | University of Michigan | Continues to exist despite the best intentions of management |
| Clay Game Engine | (Me) | Hypothetical Product |

# Wait… What

- The “Clay Game Engine” is a thought experiment
- The project models all of the problems of a MOBY application
- Project is engineered from the start to include regression tests for high-level integration
- Development work in CGE will act as a path to bring other MOBY Applications out of the darkness

# Why a Game?

*In ev'ry job that must be done*
*There is an element of fun*
*You find the fun and snap!*
*The job's a game*

# Just for Fun

- Developers are human beings
- We normally require payment for drudge work
- Testing is not fun. Supporting Mac and Windows is even less fun.
- To retain developer interest we need to make the end result something that is fun

# Game Engine Requirements

- Development effort for a title should be roughly that of RenPy

- Rather than target SDL, games would utilize either Tk or HTML and SVG in a captive browser

- The Game Engine also happens to mirror the simulation nature of many MOBY applications

# HELP

# WANTED

Though it’s more

like...


AFRICA
POLITICAL
SIR GEORGE HEAD

# Why Can’t I use Undroid / KBS / KitCreator

- Undroid / KBS / KitCreator make generic Tcl environments

- Each imposes shims on packages that conflict with the shims that MOBY applications already have on those same packages

- Many “simplify” the build process by disabling options on packages that MOBY application users depend on

# Bits That I've Worked Out So Far

- Build fulfillment works with ZipFS based kits using Tcl 8.6
- Build system is based on Practcl, can build the Integrated Recoverability Model for both Mac and Windows
- User-Developed code in both Tcl and C is "easy" to integrate

# Features (that work)

- Mac and Windows applications can be built directly from source

- The source is checked out of fossil or github, using version tags that are known to play well together

- The Tcl-Based build system supports falling back to TEA, Autoconf, and CMAKE where existing build system is not worth the hassle of rewriting

- Mirrors on http://fossil.etoyoc.com/fossil contain branches of each major project that include shims to support the Practcl build system

# What Needs to be Done

- Windows is currently cross compiled in MinGW. Future builds need to use MSVC
- Code Signing
- Big component will be external project and code provenance tracking. Not even started yet.
- The impact of radical changes to the core and Tcllib on older applications needs to be taken more seriously by developers


COMING
ATTRACTIONS

# Code Provenance

- A website devoted to the history of Tcl and Tcl packages
- Focus is on the developers themselves as well as the oddball “beyond the implementation” history of various modules and components
- Allows people to feel like they have made a mark, even if their implementations are later rewritten or replaced
- Content is suggested by the community, but curated by designated historians

# Code Provenance

- Will start with GUTTER, and also include machine readable hints for how projects interact, and which versions of packages belong in which profiles

# Sobyk Technology Profiles

- Sobyk profiles represent a “technology freeze” that dominates the development cycle of many projects
- Once a selection of packages is selected, tested, and certified, they do not change
- The requirements for each profile will be spelled out, and changes that do not serve a specific requirement will have to wait for another development cycle

# Planned Technology Profiles

- Tcl 8.5.X (BRL Cad)
- Tcl 8.6.8 (IRM Version 4)
- Tcl 8.6.X (Technology Evaluator for the Core Team)
- Tcl 8.7.X (Technology Evaluator for the Core Team)

# Wait… 8.6.8

- IRM Version 4 has been Certified by the US Navy for simulation work

- I’m stuck in 8.6.8 because that was the core that ran without require a massive rewrite

- Updating the technology after this point requires undergoing another certification process