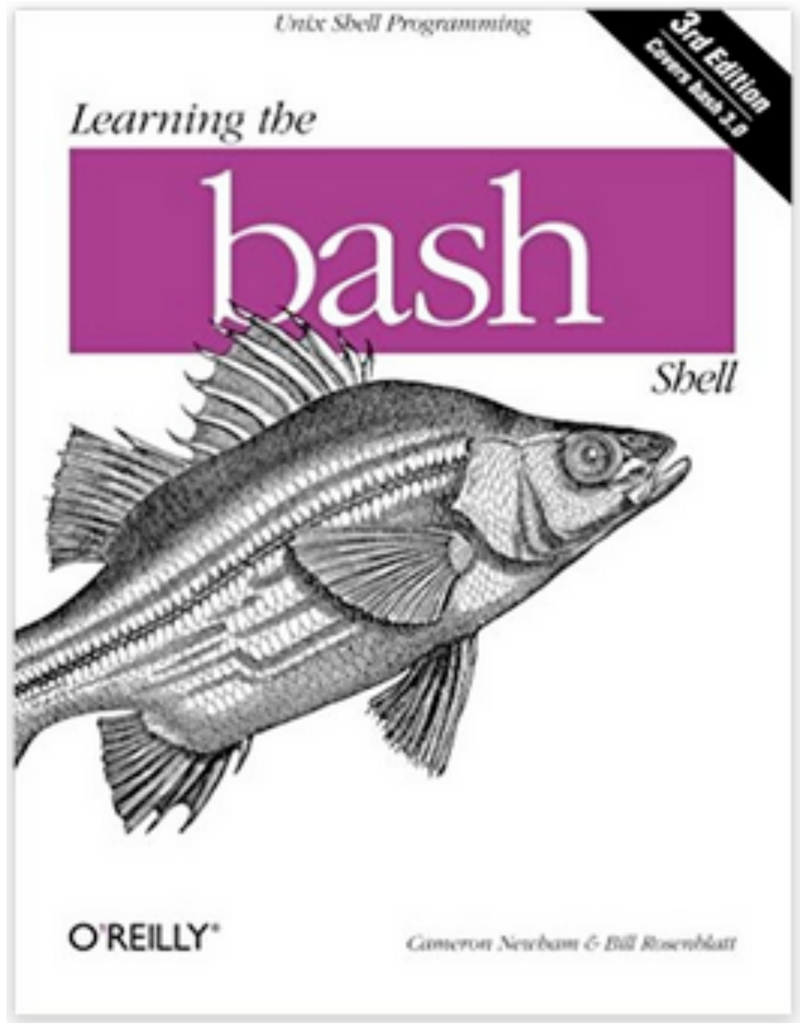
Welcome to the bash script debugger (`bashed`)

How and Why?

- Debugging bash scripts is painful (`set -x` and `echo`)
- There are no good tools:
  - Intellij plugin didn't work when I tried it
  - gnu bash debugger is clunky and only works with bash 4 (MacOS has bash 3 which is difficult to change)

- `bashed` started as a hackday project
- Debugger outline in the back of this →
- 99.9% of the code left

  "as an exercise for the reader"
- Supplied 0.1% of code was poor
- Probably retained only a few lines



Compatibility

- `bashed` was written on macos, where bash 3 is the norm
- Should work with bash 4 and above
- Limited testing on Linux... seems to work

Installation

- Installation is manual, but straightforward
- One script plus two dozen functions in a subdirectory
- Copy these to your favourite bin directory (keep script and functions together)
- On startup, it uses its own path to find the function directory

```
$ mv bashed bashed-functions ~/my-install-directory/bashed
$ cd ~/my-bin-directory
$ ln -s ~/my-install-directory/bashed/bashed bashed
$ bashed ~/my-install-directory/bashed/example.sh
Bash debugger version 0.1.  Type 'h' or '?' for help.
Debugging /Users/tony.bamford/bashed/example.sh (with bash 3) and no args

Stopped at main:3: echo "Simple script for debugging purposes"
bashed>
```

What does it do?

- Sits in a continuous loop running the target script, until you quit
- Doesn't even stop after the target has completed - enters "post mortem debugging" mode (can examine the final values of variables)
- Single step/multi step through the target script
- Set breakpoints (by line number)
- Examine variables

What else?

- Debugs functions too, showing a stack trace
- Monitor calls to chosen programs/scripts on your PATH
- Post mortem mode allows command line arguments to be modified for next run
- Run commands in sub-shells

What else?

- Handles input/output redirection in the target script
- Command editing and command history available
- Works in "raw" mode, i.e. no need to press "enter" after everything
- Doesn’t alter your script (makes a temporary copy, alters that, then removes it)

What won't it do?

- `bashed` can't cope with:
  - *trap* which it uses heavily itself, so it issues a warning and continues
  - scripts that aren't newline terminated (issues a warning and quits)

- Doesn't cope well with:
  - debugging recursive scripts
  - debugging monitored scripts (it can do it, but it's all *really* confusing)
  - keyboard interrupts

Difficulties

- `bashed` copes with `autoload` (just because I tend to do this often)

- Looks for an executable called `expand` which is expected to list said function for inclusion in temporary script

What’s its future?

- Currently does pretty much everything it is capable of doing, although “drop frame” would be nice
- Would like echo variables to use <TAB> completion (haven’t figured out how to do this, it may not be possible)
- Sort out legal issues (will O'Reilly try to claim copyright?)
- Add to Github
- Feedback or fixes for versions of Unix/Linux/etc. would be appreciated

How does it work?

- Takes script to be debugged and copies/embellishes a new version in $HOME/.bashed.*script*.*pid*
- For example:

```
$ cat example
#!/bin/bash

echo "hello from the example script"

i=0
while (( i < 10 )); do
    echo "$i"
    (( i = i + 1 ))
done
```

How does it work?

```
$ cat ~/.bashed.example.79116
#!/bin/bash

######### START BASH DEBUGGER PREAMBLE #################
__dbname__=bashed; __target__=example; __debug_out__=~/.bashed.out.example.79116;
__initial_argv__=("$@"); __pager__=less
while read; do
    __lines__[++__i__]="$REPLY"
done < $__target__
for file in ~/bashed/bashed-functions/*; do
    source $file
done
unset file
__initialise__ /~/.bashed.out.example.79116
shopt -s extdebug
command trap '__step_trap__ $LINENO "$@"' DEBUG
command trap '__post_mortem_debugging__ $__debug_out__ "$@"' 0 1 2 3
(( __offset__ = $LINENO + 1 ))
#########  END BASH DEBUGGER PREAMBLE  #################
#!/bin/bash

echo "hello from the example script"

i=0
while (( i < 5 )); do
    echo "$i"
    (( i = i + 1 ))
done
```

Things to note

- `bashed` variables and functions start and end with double underscores to avoid conflict with the script being debugged

- Unfortunately, if you do need to debug the debugger, this can make things difficult to read

- Most of the work is done by `bash` itself

```
command trap '__step_trap__ $LINENO "$@"' DEBUG

command trap '__post_mortem_debugging__ $__debug_out__ "$@"' 0 1 2 3
```

How it looks

```
$ bashed example.sh
Bash debugger version 0.1.  Type 'h' or '?' for help.
Debugging example (with bash 3) and no args

Stopped at main:3: echo "hello from the example script"
bashed> l
1       #!/bin/bash
2
3  >    echo "hello from the example script"
4
5       i=0
6       while (( i < 5 )); do
7           echo "$i"
8           (( i = i + 1 ))
9       done
bashed>
```

And finally…

- You may hate bash scripting, but:

“… any language capable of debugging itself is always worth a second look”

(sorry, I forgot where I read this)