travisbhartwell / **nix-shell-shebang.md**

Last active 3 days ago

nix-shell and Shebang Lines

nix-shell-shebang.md

NOTE: a more up-to-date version of this can be found on my blog

nix-shell and Shebang Lines

A few days ago, version 1.9 of the Nix package manager was released. From the release notes:

> nix-shell can now be used as a #!-interpreter. This allows you to write scripts that dynamically fetch their own dependencies.

They followed with an example that used GHC's `runhaskell` to execute Haskell code using libraries that had been specified in the shebang line. Unfortunately, this specific example doesn't work, as it isn't sufficient information for Haskell to find the Network.HTTP library.

But this notwithstanding, it is still an interesting change that I have found useful. To use it, start your scripts with two lines similar to this:

```
#! /usr/bin/env nix-shell
#! nix-shell -i python3 -p python3 python34Packages.pygobject3 libnotify gobjectIntros
```

The `-i` parameter to nix-shell tells it which interpreter to use when executing the script. Often, it is from one of the dependencies, such as in the above example. The `-p` parameter gives one or more dependencies to be used. `After the above lines follows the script (shameless borrowed from the ArchWiki):

```
#! /usr/bin/env nix-shell
#! nix-shell -i python3 -p python3 python34Packages.pygobject3 libnotify gobjectIntros

from gi.repository import Notify
```

```
Notify.init("Hello world")
Hello=Notify.Notification.new("Hello world","This is an example notification.","dialog
Hello.show()
```

But its usefulness isn't limited to just writing scripts interpreted by one of the declared dependencies. I needed to write a wrapper for some NodeJS scripts I had installed in the node_modules directory of a project I am working on. I didn't want Node installed globally, so I did this:

```
#!/run/current-system/sw/bin/env nix-shell
#!nix-shell -i bash -p nodejs

readonly BIN_DIR="$(cd -P "$(dirname "${BASH_SOURCE[0]}")" && pwd)"
readonly CMD="$(basename ${BASH_SOURCE[0]/%-wrapper/})"

"${BIN_DIR}"/"${CMD}" "$@"
```

I simply made a symlink for each program in my `node_modules/.bin` directory to this file, with the name `program-wrapper`, for example, `tern-wrapper` to wrap tern. Notice my script doesn't directly call nodejs, though the underlying script it calls does.

One more example. I wrote the following script to render this document as I was writing it to check the way it looked in HTML:

```
#!/run/current-system/sw/bin/env nix-shell
#!nix-shell -i bash -p inotifyTools pandoc

readonly FILE="$*"

if [ $# -lt 1 ]; then
    echo "Usage:  ${0} MARKDOWN_FILE" 2>&1
    echo "" 2>&1
    echo "MARKDOWN_FILE must exist before launching." 2>&1
    exit 1
fi

if [ ! -e "${FILE}" ]; then
    echo "${FILE} doesn't yet exist, create it before launching!" 2>&1
    exit 1
fi

# Assume the extension is .md
readonly OUTPUT=$(basename "${FILE}" ".md").html

echo "Press Control-C to quit watching for changes on ${FILE}."
while true; do
    inotifywait -q -e modify "${FILE}" &&
        echo "Updating HTML for ${FILE}" &&
        pandoc -s -f markdown -t html -o "${OUTPUT}" "${FILE}"
done
```

One last example, where I couldn't use nix-shell in a shebang line. I was playing with Hakyll. After you use `haykll-init` to generate your project structure, all work is done by compiling your own code (in this case, a `site.hs` that has an accompanying cabal file. Since something similar to the example from the release notes didn't work, I tried the following, a variant of what I've used in `.nix` files.

```
#! /usr/bin/env nix-shell
#! nix-shell -i bash --pure  -p 'pkgs.haskellPackages.ghcWithPackages (pkgs: with pkgs

cabal run $@
```

But nix-shell didn't like this:

```
nafai@shedemei:~/Documents/blog/hakyll/technically
$ ./site-wrapper
error: syntax error, unexpected ')', at (string):1:66
```

So I had to just make this one a regular shell script:

```
#!/run/current-system/sw/bin/bash

nix-shell --pure \
          -p "pkgs.haskellPackages.ghcWithPackages (pkgs: with pkgs; [ hakyll cabal-in
          --run "cabal run $@"
```

Anyway, just in these last few days I've found interesting ways to use this new capability. I hope this gives some examples of how it may be used. I welcome any feedback from more experienced Nix users (or comments in general about my scripting, I'm a little out of practice).

I intend to move this content to a blog hosted on my own server once I figure out a static blog generator to use and all of that associated nonsense. Putting this here for now, I will update with a pointer to the final location.

You can find me at @travisbhartwell or as `Nafai` on `#nixos` on freenode.net. Most of my personal code can now be found on Gitlab, including my shell scripts and my current configurations for Nix OS, bash, X, i3, and Spacemacs.

3noch commented on Jul 20, 2018

Use `"` instead of `'` in your hakyll shebang and it should work.

siers commented on Mar 3, 2019

This works

```
#! /usr/bin/env nix-shell
#! nix-shell -p "haskellPackages.ghcWithPackages (pkgs: with pkgs; [lens])" -i runhaskell
main = print 1
```

srid commented on Oct 11, 2019 • edited

To launch the script with ghcid (so as to re-compile whenever source file changes):

```
#! /usr/bin/env nix-shell
#! nix-shell -p ghcid -p "haskellPackages.ghcWithPackages (pkgs: with pkgs; [req])" -i "ghcid -c
'ghci -Wall' -T main"

main :: IO ()
main = do
  print True
```

Note that this also enables standard ghc warnings.

piperswe commented on Nov 29, 2022

Does anyone happen to know if there's a way to do this with Flakes?