Running shell commands from Node.js

I run a lot of commands from Node.js and Gulp scripts, the most common is to spawn a new shell to run a command or an application.

Take the following snippet of a Bash script

```
#! /usr/bin/env bash

# Variable holding name of source image.
SOURCE_IMAGE='STSCI-H-p2022a-f-4398x3982'

'STSCI-H-p2022a-f-4398x3982'# Variables holding names of encoders's binar
IMAGE_MAGICKbp'
HEIC_ENCODER='heif-enc'
# DS'
HEIC_ENCODER='HEIC_ENCODER='heif-enc'
# DSSIM Binary
DSSIM_BINARY
if hash ${IMAGE_MAGICK} 2>/dev/null; then
  echo encoding to PNG
  ${IMAGE_MAGICK} ${SOURCE_IMAGE}.tif -quality 80 ${SOURCE_IMAGE}.png
  echo encoding to JPG
  ${IMAGE_MAGICK} ${SOURCE_IMAGE}.tif -quality 80 ${SOURCE_IMAGE}.jpg
else
  echo cannot convert to PNG or JPG
fi

if hash ${WEBP_ENCODER} 2>/de${IMAGE_MAGICK}echo encoding to lossy WebP
  ${WEBP_ENCODER} -q 80 \
  ${SOURCE_IMAGE}.tif \
  -o ${SOURCE_IMAGE}.webp
else
  echo cannot convert to WEBP
fi
```

```
if hash ${HEIC_ENCODER} 2>/dev/null; then
  echo encoding to lossy HEIC
  ${HEIC_ENCODER} --quality 80 \
  ${SOURCE_IMAGE}.png
else
  echo could not encode to HEIC
fi
```

The script creates variables to hold the name of the image we want to convert and the names of the encoders we want to use. The encoders must already be installed and on the PATH.

For each image format, the script will check if the encoder is available. If it is the script will use it to encode the image. If the encoder is not available thhe script will print an error message and continue on to the next encoder or exit if it's on the last one.

The Bash shell is powerful but it's not available everywhere out of the box. It is available by default on Linux and macOS, but not on Windows. To get Bash on Windows, you have a few options:

- Install Cygwin
- Install MinGW
- Install the Windows Subsytem for Linux (WSL)

Another posibility is to run a full script in node using tools like zx. The tools is more complicated to use and does a better job or emulating a shell than ShellJS.

It requires you to save the files with an `.mjs` extension to take advantage of top level await syntax.

ZX uses the "shebang" as the first line of the script, like shells, to find the location of the ZX executable.

It also makes Node packages available to the script. The available packages are:

The chalk package : console.log(chalk.blue('Hello world!'))

fs package : The fs-extra package. : let content = await fs.readFile('./package.json')

The globby package. : let packages = await globby(['package.json', 'packages/*/package.json']) : let pictures = globby.globbySync('content/*.(jpg|png)') : Also, globby available via the glob shortcut: : await `$svgo ${await glob('*.svg')}`

The os package. : `await $cd ${os.homedir()} && mkdir example`

path package : The path package.

: `await $mkdir ${path.join(basedir, 'output')}`

minimist package : The minimist package. : Available as global const argv

Examples

The following examples present some basic capabilities available to ZX scripts.

```
#!/usr/bin/env zx

// Changes the color of the text displayed to screen
console.log(chalk.blue.bold('Hello, world!'));

'Hello, world!'// Takes input from the keyboard and prints it in a string
console.log(chalk.green(`I like ${beer} the best`));

// 1. Fetches a'What kind of beer is best? '`I like ${beer} the best`// 1
// 2. Grabts the text from the file
// 3. Prints the text to screen
let doc = await fetch('https://caraya.github.io/gulp-starter-2021/a-css-c
let html = await doc.text(); // 2
console.log(html); // 3
```