macOS 10.15+ and zsh shell for scripting

Description

Apple announced changes to the default shell for macOS 10.15:

https://support.apple.com/HT208050

Information

For years now, Apple has appeared to avoid any tools that are covered by the GPL v3 Licence as well as remove any that were in use over time. Bash is one of these. In the early releases of 10.x Apple initially used tcsh shell as default, but soon moved to bash; so this is not the first time Apple has made a change of this kind.

In order to avoid newer versions of bash covered by GPL v3 licensing, it can be seen how old bash is on a macOS device:

```
$ bash -version
GNU bash, version 3.2.57(1)-release (x86_64-apple-darwin18)
Copyright (C) 2007 Free Software Foundation, Inc.
```

Run the same command on a modern Linux system, e.g. the FileWave Linux Server Appliance and you will see a much newer version:

```
$ bash -version
GNU bash, version 4.2.46(2)-release (x86_64-redhat-linux-gnu)
Copyright (C) 2011 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>

This is free software; you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
```

GPL v3 license has implications for Apple. zsh is licensed under MIT, which does not involve these same implications.

Considerations

Specifying the shell

The first line of a script should indicate which shell is used when a script runs.

For example, for bash this could typically be either of the following (this is known as the 'shebang'):

```
#!/bin/bash
```

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

By providing this, the script will run from a shell of the specified type. Any scripts created should have this set. If this is not set, then the script will run in the shell type that is currently set for that shell's session. Apple's changes will have an impact on any scripts not specified by default. The best practice is to always add the shebang at the beginning of any script to ensure expected behavior.

Bash vs zsh

Bash and zsh are very similar, but there are some differences that may interfere with scripts that were written for bash but are then run as zsh. Scripts should therefore be analyzed for behavior if the shell type of the script is changed.

Examples

Arrays

The first item of an array differs when being referenced:

- bash reference of the first item in an array: index 0
- zsh reference of the first item in an array: index 1

There is also a difference in deleting items from an array. The following produces the same output from the same original array, but note differences on the item being indexed and the method of removal:

bash arrays

```
#!/bin/bash


myarray=("one" "two" "three")
echo ${myarray[@]}
echo "First item in array: "${myarray[0]}

echo "Remove first item in array, item 0..."
unset myarray[0]
echo ${myarray[@]}

exit 0

# Script output:
one two three
First item in array: one
Remove first item in array, item 0...
two three
```

zsh arrays

```
#!/bin/zsh

myarray=("one" "two" "three")
echo ${myarray[@]}
echo "First item in array: "${myarray[1]}

echo "Remove first item in array, item 1..."
myarray[1]=()
echo ${myarray[@]}

exit 0

# Script output:
one two three
First item in array: one
Remove first item in array, item 1...
two three
```

Variable Expansion

Word splitting on variable expansion differs between bash and zsh. For example, bash will print the following, one line per word, whilst zsh will print the whole variable as one line. Note also, that bash, by default, will not expand aliases when the shell is not interactive, unlike zsh.

bash variable expansion

```
#!/bin/bash

# Expand aliases
shopt -s expand_aliases

alias printvar="printf '%s\n'"
myvar='one two'
printvar $myvar

exit 0

# Script output:
one
two
```

zsh variable expansion

```
#!/bin/zsh

alias printvar="printf '%s\n'"
myvar='one two'
printvar $myvar
```

```
exit 0

# Script output
one two
```

zsh does have the ability to set an option to change this behavior, but consider converting to an array instead.

Apple has chosen zsh over bash since the overlap on script compatibility is high. However, as seen there can be differences and so it is prudent to check all scripts for behavior. The above are just examples; other differences may be experienced and will require addressing appropriately.

Revision #2
★Created 14 July 2023 20:02:47 by Josh Levitsky
✎Updated 4 November 2024 13:43:14 by Josh Levitsky