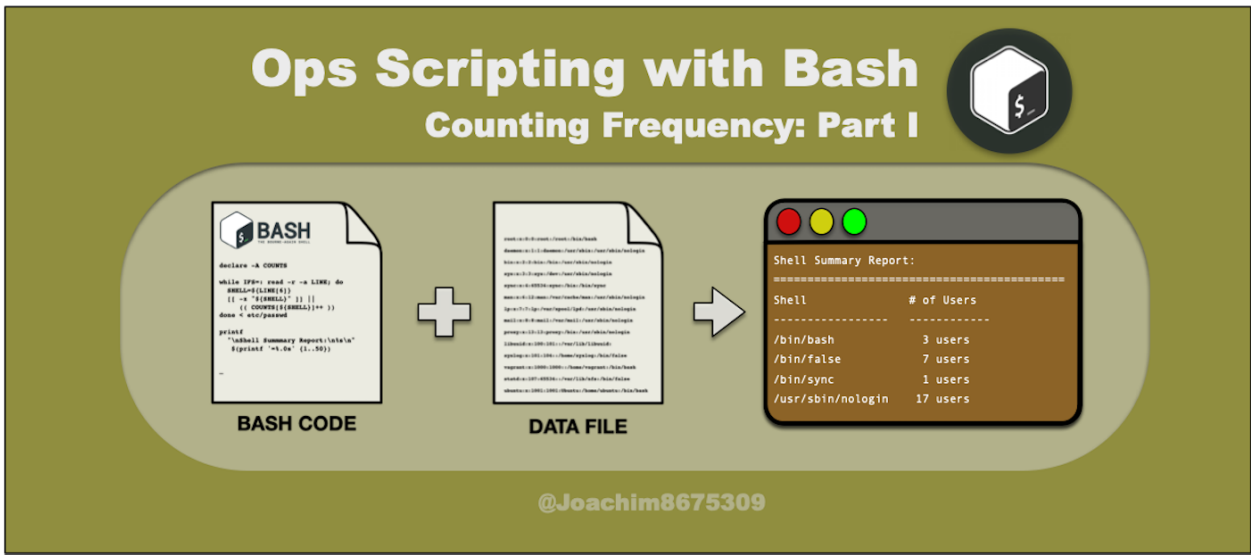


Ops Scripting w. BASH: Frequency

Tracking Frequency in BASH (Bourne Again Shell): Part I

Joaquin Menchaca Follow

May 15 · 3 min read

Operations oriented roles, almost always require skills in automation and scripting with shell programming. These days with the ubiquity of Linux and GNU command line tools, **GNU Bash** has become ubiquitous.

Bash combined with command line tools, is regulated for small automation chores, data structures are supports strings, integers, and arrays. However, starting with **Bash 4**, you can use more advanced

data structures like **associative arrays** (hashes, maps, or dictionaries) to create data structures that are used for algorithms like tracking frequency.

This problem and later solutions, show how to use a **Bash** *associative array* to track frequency, and along the way, show some cool tricks that you can use with **Bash**.

But first you need to make sure **Bash 4** or higher is available on your system…

Getting Bash 4+

If you have a recent Linux distro, such as **Debian**, **Ubuntu**, **CentOS**, a current version of Bash 5 is already installed.

macOS

Apple's macOS (aka Mac OS X) comes a very old 15-year old version of **Bash**. With **Homebrew** package manager, you can get a recent version of **Bash**:

```
brew install bash
sudo bash -c 'echo /usr/local/bin/bash >> /etc/shells'
chsh -s /usr/local/bin/bash
```

Windows

On Windows 10, we can use **MSYS2** (Minimal System 2) that comes with a recent version of **Bash**. You can install **MSYS2** with the package manager **Chocolatey** from either a **command shell** (`cmd.exe`) or **PowerShell** (`powershell.exe`) in Administrative mode:

```
choco install msys2
```

The Problem

The goal of this exercise is to print a summary of shell usage on a system. For this exercise, we'll do it in two parts:

1. Build a data structure containing the shell counts, called `COUNTS` from a supplied local password file `./passwd`.
2. Given the shell counts data structure `COUNTS`, produce a report.

The Data

Here's the `passwd` file you will use for this exercise:

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologi
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
libuuid:x:100:101::/var/lib/libuuid:
syslog:x:101:104::/home/syslog:/bin/false
messagebus:x:102:106::/var/run/dbus:/bin/false
landscape:x:103:109::/var/lib/landscape:/bin/false
sshd:x:104:65534::/var/run/sshd:/usr/sbin/nologin
pollinate:x:105:1::/var/cache/pollinate:/bin/false
vagrant:x:1000:1000::/home/vagrant:/bin/bash
colord:x:106:112:colord colour management daemon,,,:/var/lib/colord:/bin/false
statd:x:107:65534::/var/lib/nfs:/bin/false
puppet:x:108:114:Puppet configuration management daemon,,,:/var/lib/puppet:/bin/
```

passwd

The Output

When generating a report, the output should look like this:

```
Shell Summary Report:
================================================
Shell               # of Users
-----------------   ------------
/bin/bash             3 users
/bin/false            7 users
/bin/sync             1 users
/usr/sbin/nologin    17 users
```

The Code

Here is sample code to get you started:

```
#!/usr/bin/env bash
declare -A COUNTS
# >> YOUR SOLUTION HERE <<

printf "\nShell Summmary Report:\n%s\n"  $(printf '=%.0s' {1..50})
printf "%-17s    %s\n" "Shell" "# of Users"
printf "%-17s   %s\n" $(printf '%.0s-' {1..17}) $(printf '%.0s-' {1..12})

for SHELL in "${!COUNTS[@]}"; do
  printf "%-17s   %3d users\n" ${SHELL} ${COUNTS[${SHELL}]}
done
```

Code Notes

These are some of the techniques used in the code above.

Formatted Output

We `printf` command (see **printf**), which behaves like the C-Language counterpart:

```
printf FORMATTED_STRING STRING1 STRING2 STRING3
printf "%s %s %s\n" STRING1 STRING2 STRING3
```

Repetition

You can use repetition to print out, a line for example, using this technique:

```
printf '=%.0s' {1..10}
```

This will generate ten equal symbols. This itself can be wrapped into a sub-shell `$()` or `` `` `` it is needs to be concatenated with another string.

Enumerating Keys

The keys and values of an associative array in **Bash** can be enumerated with this notation:

```
KEYS   = "${!ASSOC_ARRAY[@]}"
VALUES = "${ASSOC_ARRAY[@]}"
```

If you put these into a subshell, you then sort the keys.

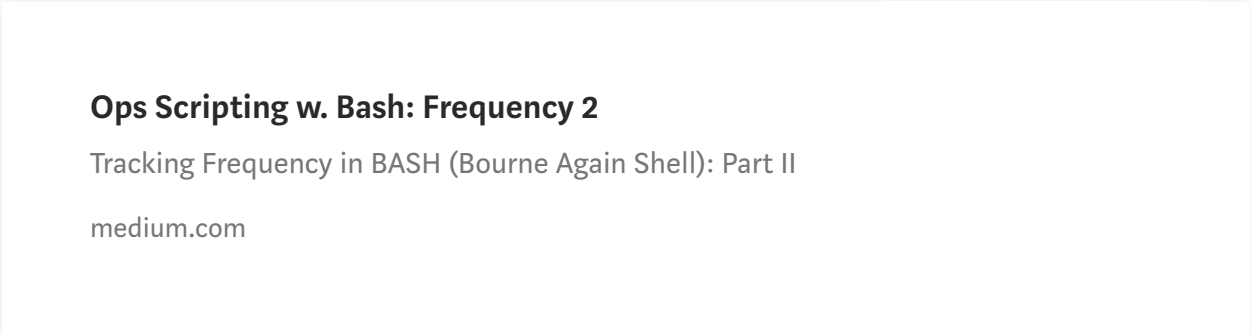
```
SORTED_KEYS = $(echo ${![ASSOC_ARRAY[@]} | sort)
```

Referencing a Value

You can extract a value given a key like this:

```
VALUE = ${ASSOC_ARRAY[$KEY]}
```

Next Article



The Conclusion

For now, I'll just present the problem, and **next article** I will present some solutions.

With the sample code, you can get the following takeaways for Bash

- Creating an Associative Array with `declare -A`
- Enumerating Keys and Values from an associative array
- Referencing (looking up) an item from the associative array
- Formatted output with `printf`
- Repetition using `printf "%.0s=" {1..10}` trick