PJ's Bash Tips #6 (ebook excerpt) - Generating Fibonacci Big Numbers

Author: Paulo Jerônimo
Last update / commit - 2023-03-23 13:55:46 -0300 / 850e4df

1. Question to motivate you to write the script `1.sh`

Would you know how to calculate which is the index and the value of the first Fibonacci number that has 55 digits?

Here is a command line we can use to solve this question:

```
./1.sh 500 | tee fib-seq.txt | awk 'length == 59' | head -1
```

We'll explore details about the command above here. For now, you can copy it from here. Let's focus on its output. It produces the following:

```
261:15714085184275463781678466585241861481334453009875507 86
```

So, the first Fibonacci number with 55 digits has the index 261 in the sequence (0 1 1 2 3 5 ...) by knowing that index starts from 0.

Let's go ahead and understand this command line.

2. Generating the script 1.sh

2.1. Differences between it and the last script in the previous chapter

Starting from the last script we implement in the chapter 4 (Bash Tips #4), let's first see the differences we have to code in order to change it to produce Fibonacci big numbers.

We see the differences by typing the following commands:

```
sh=1.sh; batcat -p $sh.diff
```

The content of the `diff` that we'll implement in this chapter is here:

```
1  --- ../4-memoization/n.sh   2023-03-17 09:59:06.804714223 -0300
2  +++ 1.sh    2023-03-19 07:43:30.807765846 -0300
3  @@ -1,5 +1,6 @@
4   #!/usr/bin/env bash
5   fs=(0 1)
6  +export BC_LINE_LENGTH=0
7
8   f() {
9     case "$1" in
10 @@ -9,9 +10,9 @@
11        ;;
12      *)
13        ! [ "${fs[$1]+x}" ] || { echo ${fs[$1]}; return; }
14 -      fs+=($(( `f $(($1-1))` + `f $(($1-2))` )))
15 +      fs+=($(echo "`f $(($1-1))` + `f $(($1-2))`" | bc))
16        echo ${fs[-1]}
17    esac
18  }
19
20 -for n in `seq 0 ${1:-15}`; do echo -n $n:; f $n; done
21 +for n in `seq 0 ${1:-15}`; do printf '%03d:' $n; f $n; done
```

So, these changes are very simple: only one line was added and two were changed.

In line 15, we saw that the only change made was instead of putting the calculation result directly into the array, we first generated a unique string containing two numbers concatenated with a plus sign (+) between them. Then, we send this string to the bc command.

The bc command, among many other complex calculations, can sum up big numbers. So, by using it inside a Bash script, we can eliminate the sum overflow when the result is more significant than 2^63-1.

Big Fibonacci numbers can be generated from index 93 and above. But, if we try to calculate `f(93)` without using `bc`, we have a sum overflow, and the result will be the number `-6246583658587674878`, which is entirely wrong, right?

The `bc` command can use an environment variable called `BC_LINE_LENGTH`. It is declared in line 6 and is used to instruct `bc` to not break a produced result according to a line length. By default, `bc` will do this. But, if we set the value 0 to this variable, no line break will occurs if the number is too big.

The last change was made on line 21. In this line, we used `printf` instead of `echo` to put leading zeros before the index, assuming we'll generate numbers from `000` to `999` only.

2.2. Generating the script

To generate the script `1.sh`, let's use the patch command. It receives two arguments. The first is the original file we want to change. In this case, it is the last file we develop in chapter 4. The second argument is the diff file we saw.

```
patch ../4-memoization/n.sh $sh.diff -o $sh && chmod +x $sh
```

After applying the patch, this was the script generated:

```
#!/usr/bin/env bash
fs=(0 1)
export BC_LINE_LENGTH=0

f() {
  case "$1" in
    ''|*[!0-9]*)
      echo \"$1\" is not a valid number!
      return 1
      ;;
    *)
      ! [ "${fs[$1]+x}" ] || { echo ${fs[$1]}; return; }
      fs+=($(echo "`f $(($1-1))` + `f $(($1-2))`" | bc))
      echo ${fs[-1]}
  esac
}

for n in `seq 0 ${1:-15}`; do printf '%03d:' $n; f $n; done
```

3. Running the script `1.sh`

To generate a `fib-seq.txt` file with the first 1000 Fibonacci numbers, we type this command:

```
f=fib-seq.txt; ./1.sh 999 | tee $f && echo "$f was generated with $(wc -l < $f) lines"
```

The command above will print the output generated by the script in the console while, at the same time, saving this output to the file `fiq-seq.txt`.

Each line of this file has the format `nnn:XXXXXXXXX`. So, to check if a Fibonacci number has the length X in one line of this file, we need to add 4 to it (the length of `nnn:`). For this reason, in the first command that we saw in this chapter, to verify if the Fibonacci number has 55 digits we compare the length of the line with 59 (inside the `awk` command). And since we would like to print only the first line with this length, we also use the `head -1` command.

3.1. Playing around with the generated file (`file-seq.txt`)

Let's answer another curiosity: how many Fibonacci numbers exists containing from 55 to 89 digits? Well, using the `awk` and `wc` commands we can quickly get an answer to that by typing this command:

```
awk -v h=4 -v min=55 -v max=89 \
'{ if (length >= min+h && length <= max+h) print }' fib-seq.txt | wc -l
```

The answer is 167. *We'll double check this witn an alternative command later.*

By reading the code above, here is what we have: first, we define three variables that will be used by `awk`. The `h` (from "h"eader) was set with the value 4. `min` and `max` represent the number of digits. Since we want to check how many numbers we have between min and max, we need an if condition inside `awk` to check if length is between the boundary. Lastly, we use `wc -l` to count the number of lines returned.

In order to check if the last line of this result really has 89 digits, we can make a variant of the previous command in this way:

```
results=$(awk -v h=4 -v min=55 -v max=89 \
'{ if (length >= min+h && length <= max+h) print }' fib-seq.txt) &&
wc -l <<< "$results"
```

This way we can check the length of the last Fibonacci number in these results:

```
len=$(tail -1 <<< "$results" | tee last | cut -d: -f2 | tr -d '\n' | wc -c) &&
echo -e "The line \"$(cat last)\" has the last Fibonacci number with $len digits"
```

4. An alternative and more efficient script (2.sh)

This version does not use recursion. Nor the memoization technique. But I inserted it here to demonstrate a more efficient way of getting to the same file generated in the previous section.

It makes more heavy use of what is offered directly by bc while at the same time making less use of Bash. This version could be compared to using a stored procedure in a database to achieve better performance.

```
#!/usr/bin/env bash
max=${1:-15}
[[ $max =~ ^[0-9]{1,3}$ ]] || {
  echo "A valid argument is a number between 0 and 999!"
  exit 1
}
while IFS= read -r n fn; do
  printf "%03d:%s\n" $n $fn
done < <(BC_LINE_LENGTH=0 bc <<EOF
define fib(x) {
  if (x <= 1) return x
  a = 0; b = 1
  for (i = 1; i < x; i++) {
    c = a + b; a = b; b = c
  }
  return c
}
for (i = 0; i <= $max; i++) {
  print i, " ", fib(i), "\n"
}
quit
EOF
)
```

4.1. Comparing the performance between `1.sh` and `2.sh`

Let's run script `1.sh` to produce the first 1000 Fibonacci numbers. Running it through the `time` command we can calculate how long it takes to do its job:

```
$ time ./1.sh 999 > fib-seq.txt

real    0m2.326s
user    0m2.657s
sys 0m0.909s
```

Now, let's do the same with the script `2.sh`:

```
$ time ./2.sh 999 > fib-seq.2.txt

real    0m0.213s
user    0m0.226s
sys 0m0.016s
```

We can check if the generated files are equal in content by using the command `cmp` (or even by running `diff` as an alternative):

```
$ cmp -s fib-seq.txt fib-seq.2.txt && echo Files are equal
Files are equal
```

So, in terms of performance, script `2.sh` is much more faster to generate the same result!

5. Generating an alternative output for `fib-seq.txt`

Can we create a printer script (`p.sh`) content to see the contents of the file `fib-seq.txt` like in the following ouput?

```
digits=1
000:0
001:1
002:1
003:2
004:3
005:5
006:8

digits=2
007:13
008:21
009:34
010:55
011:89

digits=3
012:144
013:233
014:377
015:610
016:987

digits=4
017:1597
018:2584
019:4181
020:6765
```

To produce the ouput above, we want to type a command line like this:

```
./1.sh 20 | ./p.sh | tee fib-seq.p.txt
```

So this is one way to write the `p.sh` script:

```
#!/usr/bin/env bash
header=${header:-4}
digits=1
echo "digits=1"
while IFS= read -r line; do
  if ((digits != ${#line})); then
    if ((digits == 1)); then
      echo -e "$line"
    else
      echo -e "\ndigits=$((digits-header+1))\n$line"
    fi
    digits=${#line}
  else
    echo "$line"
  fi
done
```

By creating this output, we can use another alternative to know the Fibonacci numbers having a certain quantity of digits. Beyond using `awk` like we saw, with this output, we can now use a simple `grep` to print the Fibonacci numbers having this number of digits by typing the following:

```
$ grep 'digits=4' -A2 fib-seq.p.txt
digits=4
017:1597
018:2584
```

Also, another way to see the question we made in this section is by using `sed` like in this command:

```
./2.sh 999 | ./p.sh | sed -n '/^digits=55$/,/^digits=90$/p' |
sed -e '/^d/d' -e '/^$/d' | wc -l
```

The line above can be rewritten in many ways. Another alternative (very similar) is this:

```
./2.sh 999 | ./p.sh | sed -n '/^digits=55$/,/^digits=90$/p' | sed '/^d/d;/^$/d' |
wc -l
```

Shell is really amazing, isn't it? ☺

EARN YOUR COPY OF THIS EBOOK AND OTHER GIFTS (*GANHE SUA CÓPIA DESTE EBOOK E OUTROS BRINDES*)!

In a few days I'll publish details about how you can gain a free copy of this book and also other gifts like NFTs and discount coupons (*Em alguns dias irei publicar detalhes sobre como você poderá ganhar uma cópia gratuita deste ebook e também outros brindes como NFTs e cupons de desconto*).

For details, follow me (*Para detalhes, acompanhe-me*)!

Telegram → t.me/paulojeronimo_com
Instagram → instagram.com/paulojeronimo_com

Next chapter: "Generating Fibonacci numbers using Bash coproc"

Coming soon!

If you're impatient or itching to know more about this subject, try to create an excellent prompt for ChatGPT to explain it! Maybe this can surprise you! ☺.