Bash variables and functions

WE'LL COVER THE FOLLOWING

A variable is a temporary store for a piece of information. There are two actions we may perform for variables: first - setting a value for a variable, second - reading the value for a variable. To read the variable we place its name preceded by a `$` sign. To learn more, let's create a shell script that will backup the `home` directory into a zipped (`tar.gz`) file.

```
#!/bin/bash

 filename=homedirbackup_$(date +%Y%m%d).tar.gz
 tar -czf $filename /$HOME
```

Here the variable `filename` first accepts a file name that's being constucted off three strings `homedirbackup_`, current date, and `tar.gz` and then we use the variable to zip up our home dir. Note that when the variable gets called, it has a `$` infront of it. The `date` function with the given options will return today's date and built-in OS variable $HOME will give us the path to user's home directory.

Bash variable types:

In normal situations, we don't need to declare a variable type to use it. However, bash lets us declare integer, read only, array, associative array, and export type variables. For example, we can declare an integer (number) using `declare -i` we can read only a variable using `declare -r,` we use this when we want to assign a value to a variable that should not be allowed to change. Further more, Bash allows us to declare arrays with `-a` option or associative arrays with `-A` option (details given later in this chapter).

Bash vairables can be local or global:

The bash global variable's value do not change by the out of function activities, also note that "local" is bash reserved word.

Bash functions

We have already used a function above (called `bashfunction()`), as in other programming language, you can use bash functions to group pieces of code in a logical way or practice the divine art of recursion. It can also take arguments, however, Bash functions don't allow us to return a value, rather they allow us to set a return status.

```
#!/bin/bash

printfunction () {
   echo Hello $1
   return 0
   }

printfunction Hello

printfunction Big data!
```

Notice that the function takes only one arg, therefore Data! didn't print!