Examples: for-loop, if-else-condition

KIM bwHPC group

2022-10-10

Sect. 0: Bash variables

You can set variables in bash and then later retrieve their values by addressing them via **$**. These are only temporarily set, i.e. set only in this bash environment.

In the following code snippets: everything after # is just a comment (and will also be ignored when executing the full line in a bash shell).

```
echo $HOME #preset variable: Your home directory
echo "Value of testvar:" $testvar #not assigned yet...
testvar=TEST1 #Assign TEST1 to a new variable testvar
echo $testvar
```

```
## /home/fabfreund
## Value of testvar:
## TEST1
```

ADVANCED: Variables defined as described above are just present in this specific bash instance/shell. For instance, they are not present within a bash script run within this shell. If you want to have it present in all child shells (still temporarily), use **export** (so the assignment above would change to **export testvar=TEST1**). Such variables are called *environment variables*, the bash command **printenv** shows all of them (temporary and permanent).

Sect. 1: For-loops

If you want to run the same command(s) across a set of values (a *list*) in bash, you can use a for-loop. Find two examples below to see the basic syntax. The running variable of the loop is set as a bash variable.

```
for it1 in 1 2 3 4 5 #You name the running variable and provide the list
do
echo $it1          #Adress loop variable via $
done

echo "Last it1 value: " $it1 #Last value from the loop
```

```
## 1
## 2
## 3
## 4
## 5
## Last it1 value:  5
```

```
for word1 in this is an example
do
echo $word1
done
```

```
## this
## is
## an
## example
```

Exercise 1: Wha happens here?

```
echo this is new > test1.txt #In one line
echo or is it? >> test1.txt #2nd line
for var1 in $(cat test1.txt)
do
echo $var1
done
```

Exercise 2: Use a for-loop to echo all file names in your current directory

Sect. 2: Bash conditions (if ... else ...)

You may want to only execute a command if a condition is met or execute different commands depending on a condition. For this, you can use the if-else syntax in bash. The condition is within square brackets [...] (with spaces left and right).

In this introduction, we only touch how to compare two integer numbers or two text strings

You can have a single if-condition, so nothing happens if the condition is not met:

```
it1=-1
echo $it1
if [ $it1 -gt 0 ] #is integer it1 > 0?
then
echo "it1 is positive"
fi
```

```
## -1
```

You can have an if-else condition, where one command runs if the condition is met and another one if it is not met.

```
it1=-1
echo $it1
if [ $it1 -lt 10 ] #is integer it1 is < 10
then
echo "it1 is smaller than 10"
else
echo "it1 is at least 10"
fi
```

```
## -1
## it1 is smaller than 10
```

You can also nest one if condition in another one

```
it1=-1
doit=NO
if [ $doit != "NO" ]
then
  if [ $it1 < 0 ]
  then
    echo "it1 is negative"
  else
```

```
    echo "it1 is non-negative"
  fi
else
 echo "I was told to do nothing!!!"
fi
```

```
## I was told to do nothing!!!
```

Exercise: Set variables it1 and doit to different values to generate all different outputs of the three examples.

Here is a non-exhaustive list of possible conditions to check, including the ones used above

- [**int1 -gt int2**] first integer int1 is bigger than second integer int2
- [**int1 -lt int2**] first integer int1 is smaller than second integer int2
- [**int1 -eq int2**] first integer int1 is equal to second integer int2
- [**int1 -ne int2**] first integer int1 is equal to second integer int2
- [**str1 == str2**] text strings str1 and str2 are identical
- [**str1 != str2**] text strings str1 and str2 are not identical

Advanced: [**...**] runs the bash command test on the condition within the square brackets. See **man test** for more conditions to check, e.g. for file comparisons (so argument is a file name),