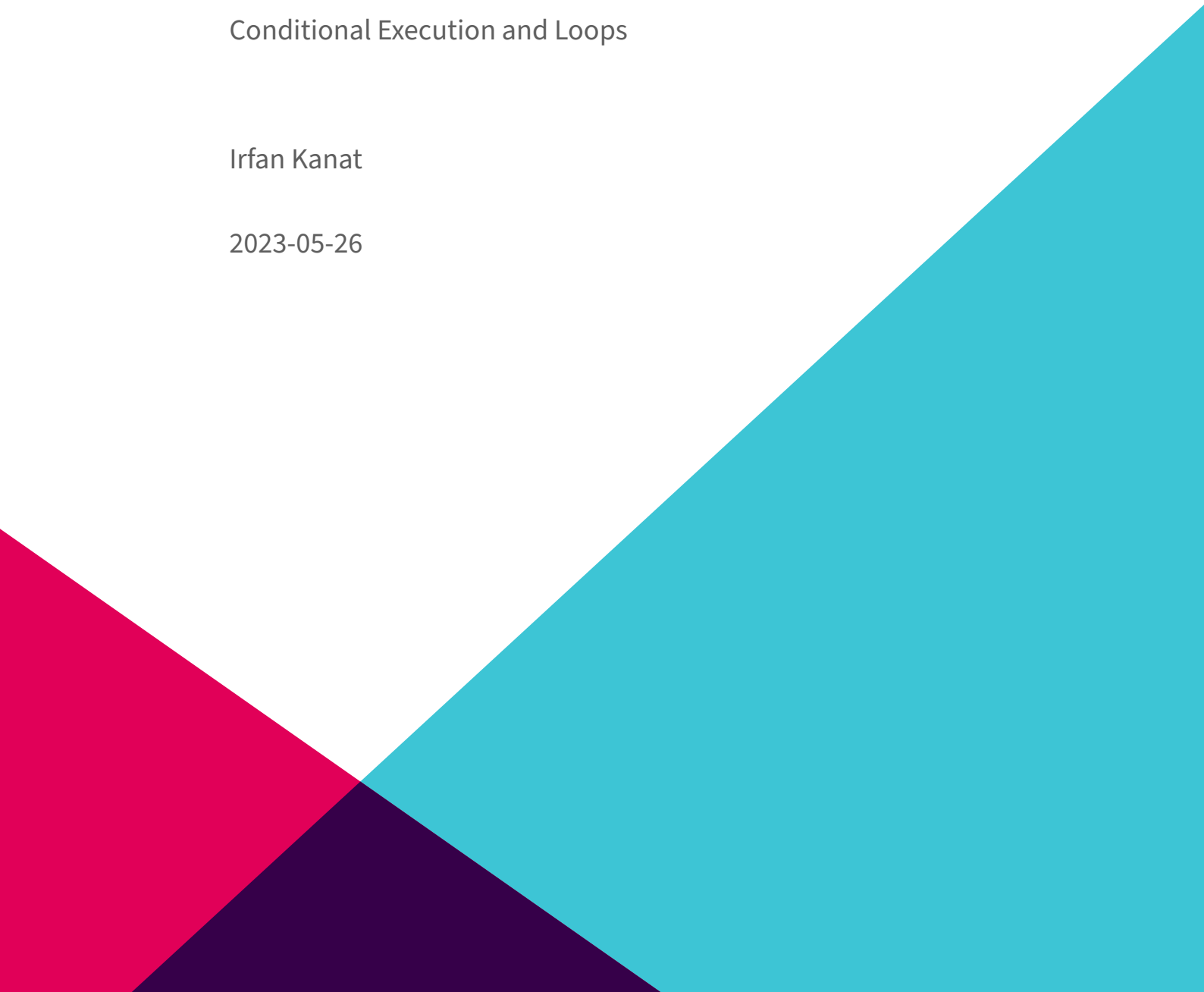

Linux Command Line

Conditional Execution and Loops

Irfan Kanat

2023-05-26



Conditional Execution

A common task in programming is to do something only when certain conditions are met. IF statements are designed for this purpose.

Option	Explanation
-gt	Greater than (numeric)
-lt	Less than (numeric)
-eq	Equals (numeric)
-s	File size greater than 0
-z	String length zero
-n	String length not zero
-t	File exists and is a regular file

Let's play around with the English proverb "If my aunt had a beard, she would be my uncle."

Let us look at the syntax below: Statement between square brackets is evaluated and only if it returns true, would the statements between then and fi are executed.

The below statement won't be executed because \$relative is not equal to "uncle". The statement within the square brackets evaluates as false.

```
relative="aunt"

if [ $relative == "uncle" ]
then
    echo "My $relative has a beard."
fi
```

The statement below will be executed.

```
relative="aunt"

if [ $relative == "aunt" ]
then
    echo "My auntie is my $relative."
fi
```

You can combine multiple conditions with logic operators and (&&) and or (||).

The following statement will be executed. While the relative is not an uncle, she still does have a beard. The or statement requires at least one condition to be TRUE.

```
relative="aunt"
beard="TRUE"

if [ $relative == "uncle" ] || [ $beard == "TRUE" ]
then
    echo "My $relative has a beard."
fi
```

The following statement however will never see the light of a display.

```
relative="aunt"
beard="TRUE"

if [ $relative == "uncle" ] && [ $beard == "TRUE" ]
then
    echo "My $relative has a beard."
fi
```

Let's work out an example that would make us use many of the topics we covered so far.

You want to print a message only **if** the home folder has more than 100MB of data.

We need a way to know the disk usage of the home folder.

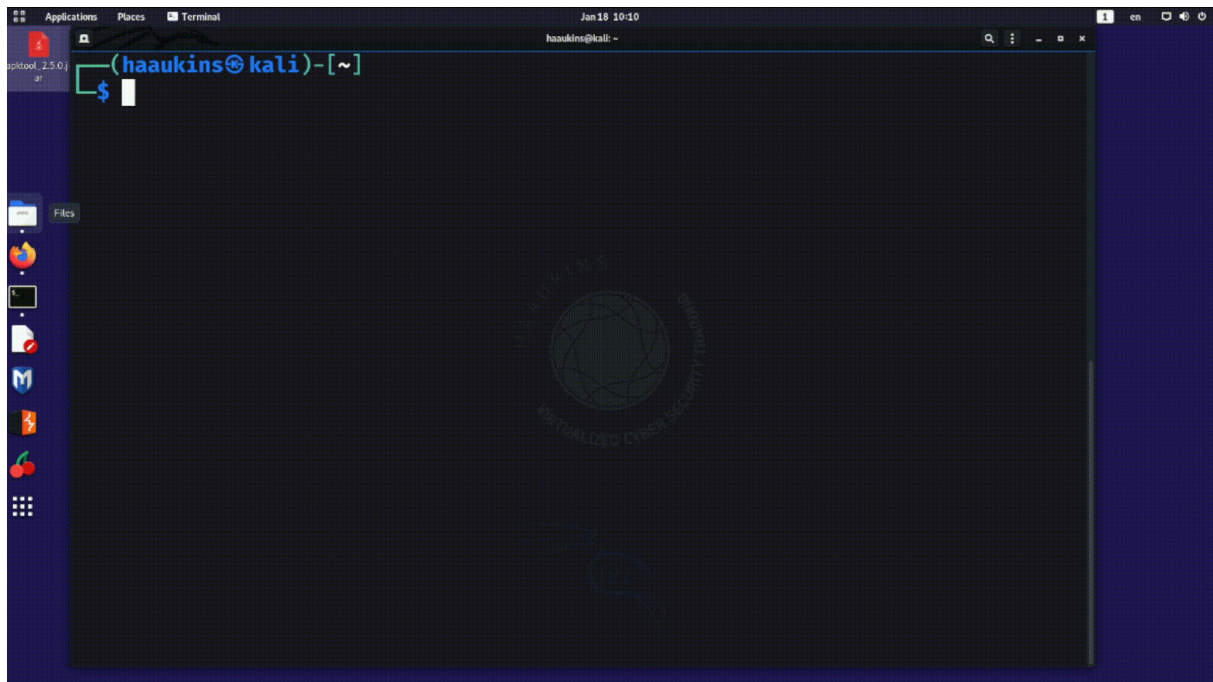
```
du -s ~
```

Would give us the usage but is not numeric. It comes with some additional output. So we pipe it to awk and only get the first column produced.

```
du -s ~ | awk '{print $1}'
```

Now we need to use it in an if statement. We want the entire disk usage statement to be evaluated and use its output so we use \$(). We then add that whole thing into the if statement.

```
if [ $(du -s ~ | awk '{print $1}') -gt 1000000 ]
then
    echo 'That is a lot of juicy data!'
fi
```



Of course you can use ; and turn it into a oneliner.

```
if [ $(du -s ~ | awk '{print $1}') -gt 1000000 ]; then echo 'That is a lot of ju
```

You can imagine what else can be placed between then and fi to make this exercise more exciting.

For Loop in Bash

After conditional execution, most common task is iteration.

Basic for loop syntax is as follows:

```
list=(1 2 3)
```

```
for iterator in list
do
```

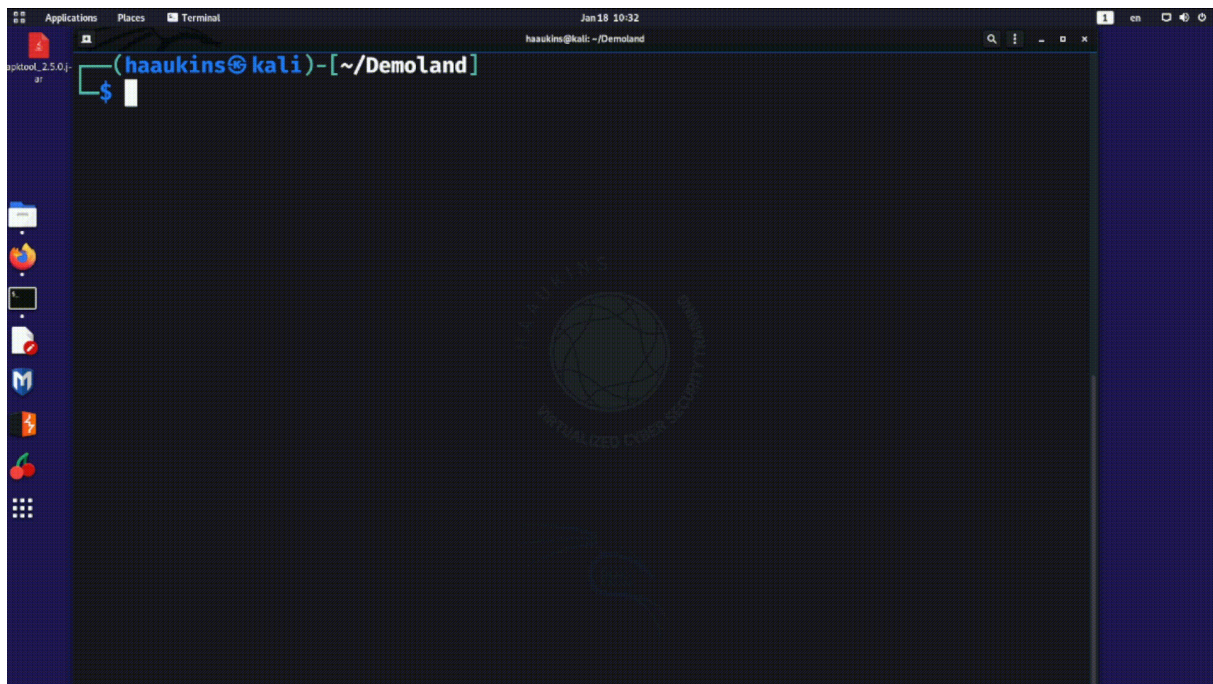
```
    echo "The number is \$iterator"
done
```

You set an iterator in a list after the word `for`, then for each instance in the list the statement between `do` and `done` gets executed.

You can also iterate over files in a folder.

```
for file in *
do
    echo $file
done
```

In the below video you can see us iterating over some very important datafiles.



While Loop in Bash

While is especially useful for files and streams. So long as there are lines, or input from the source it will iterate.

```
for i in {1..10}; do echo "This is line $i" >> file.txt ; done
```

```
cat file.txt | while read iterator
do
    echo $iterator
done
```

This is especially useful if you are reading data from a stream such as a socket.



This work is licensed under a [Creative Commons Attribution 4.0 International License](https://creativecommons.org/licenses/by/4.0/).