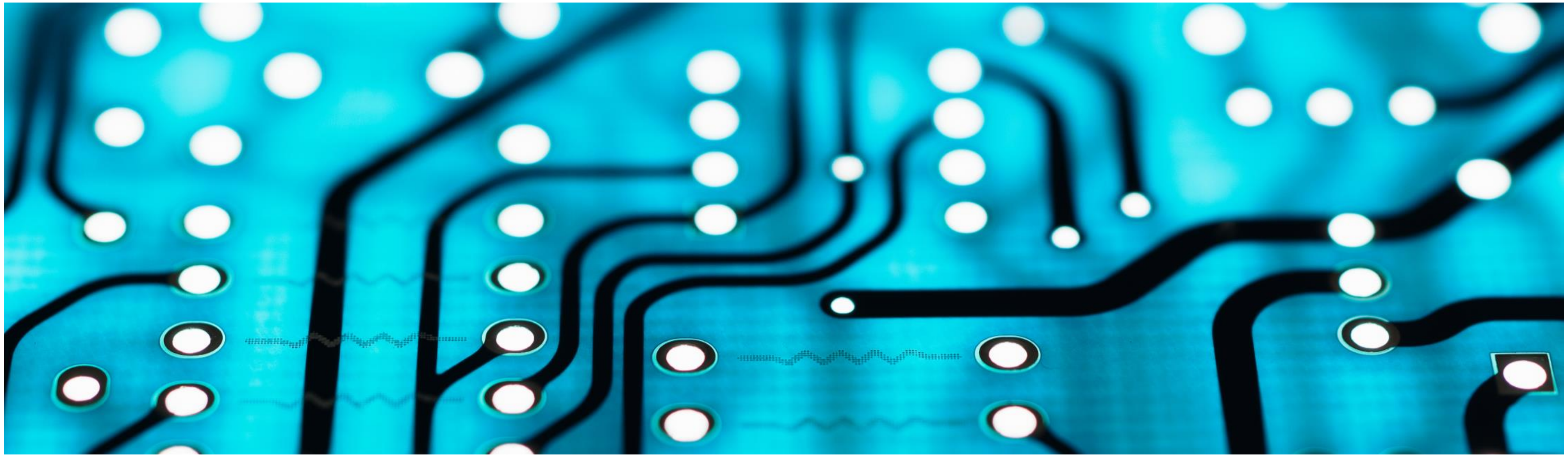


TCL BEGINNER'S GUIDE

A SMOOTH START PART FOUR

WHAT IS TCL ?

- Its pronounced (Tickle)

TCL (Tool Command Language) is a scripting language known for its simplicity and flexibility. It is commonly used for rapid prototyping, scripting, and embedded applications. This guide will cover the basics of TCL, including variables, input/output, control structures, and loops.

WHY DID IT BECAME INDUSTRY STANDARD ?

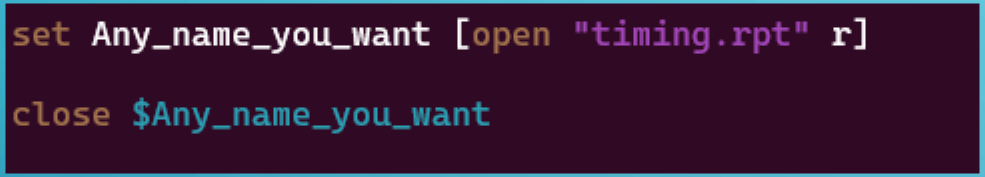
TCL has become the de facto standard embedded command language for Electronic Design Automation (EDA) applications. Whether you need to automate repetitive behavior, extend the functionality of an application, control multiple tools with a single script or create a custom GUI, TCL is your best choice.

FILES (WHY WE MUST DEAL WITH THEM) ?

- EDA tools are file-driven – TCL is used to read/write essential files like reports, constraints, logs, and netlists.
- Automates repetitive tasks – File manipulation allows scripting of flows like generating SDCs, launching runs, or extracting results.
- Enables report parsing and data extraction – Crucial for analyzing timing, power, and area data from large .rpt files.
- Supports debugging and validation – Helps in filtering logs, identifying errors, and comparing tool outputs efficiently.
- Facilitates tool integration – Bridges different stages of the flow by passing processed data between tools using files.

OPEN A FILE FOR A CERTAIN PURPOSE

The basic way to open a file is :



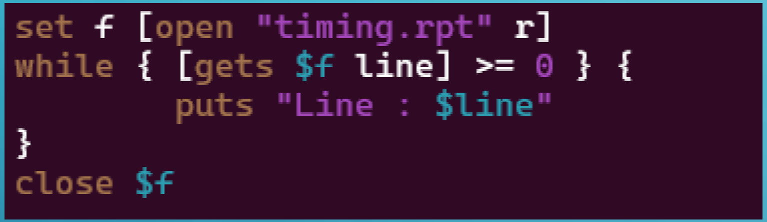
```
set Any_name_you_want [open "timing.rpt" r]

close $Any_name_you_want
```

While “timing.rpt” is the name of the file I wish to open

The letter (r) here is to represent the mode which is reading, w for writing and many other modes you can find them in the appendix of the slides.

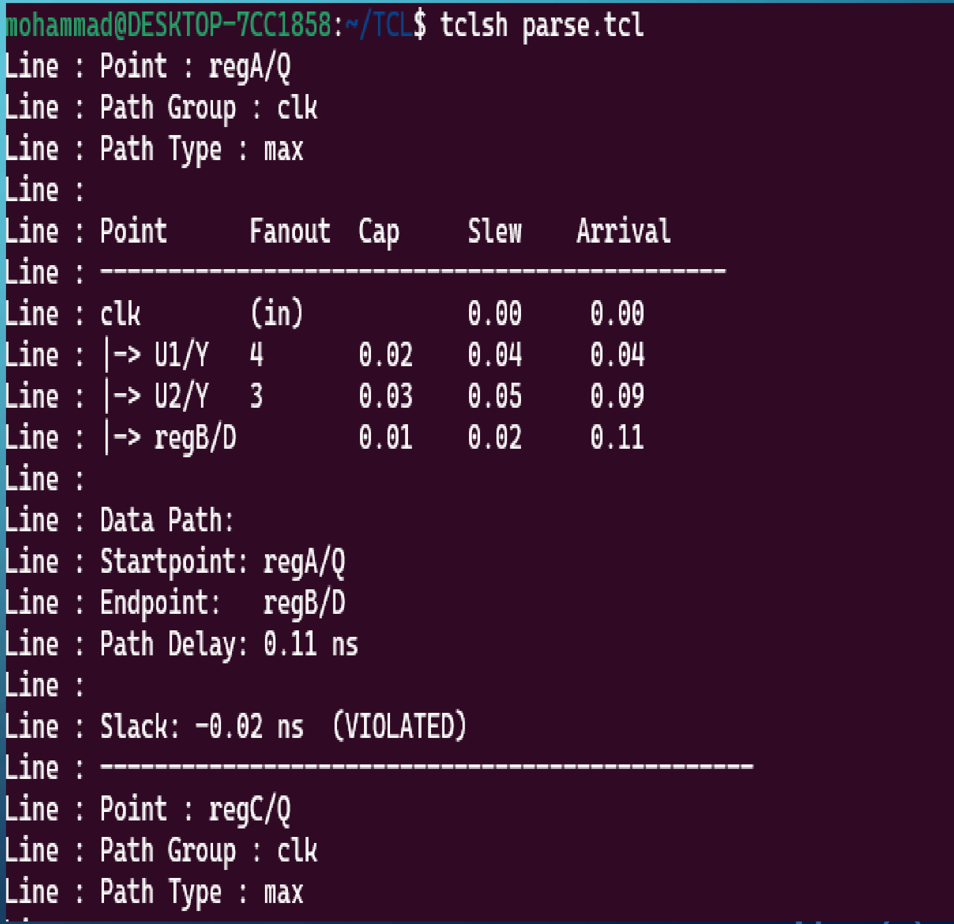
READING FROM A FILE

The most common use is reading line by line and this is something you will now when you try by your self.



```
set f [open "timing.rpt" r]
while { [gets $f line] >= 0 } {
        puts "Line : $line"
}
close $f
```

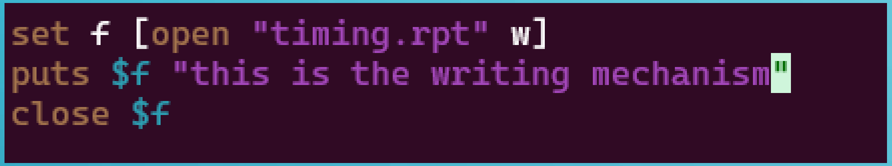
We changed the file name to (f) for simplicity only, the (gets) is used to return the number of lines in this file and when comparing the number with zero that means the file is not empty and the (puts) will print the whole content line by line, look at the right to see how



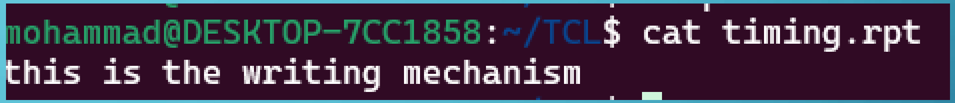
```
mohammad@DESKTOP-7CC1858:~/TCL$ tclsh parse.tcl
Line : Point : regA/Q
Line : Path Group : clk
Line : Path Type : max
Line :
Line : Point      Fanout  Cap     Slew    Arrival
Line : ---------------------------------------------
Line : clk        (in)            0.00     0.00
Line : |-> U1/Y   4       0.02    0.04     0.04
Line : |-> U2/Y   3       0.03    0.05     0.09
Line : |-> regB/D         0.01    0.02     0.11
Line :
Line : Data Path:
Line : Startpoint: regA/Q
Line : Endpoint:   regB/D
Line : Path Delay: 0.11 ns
Line :
Line : Slack: -0.02 ns  (VIOLATED)
Line : ----------------------------------------------
Line : Point : regC/Q
Line : Path Group : clk
Line : Path Type : max
```

WRITING TO A FILE

We have to ways here one is overwriting like this **(notice the w in the mode)**



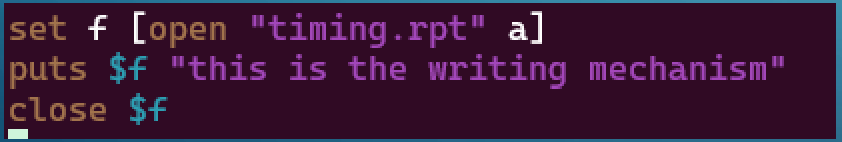
```
set f [open "timing.rpt" w]
puts $f "this is the writing mechanism"
close $f
```



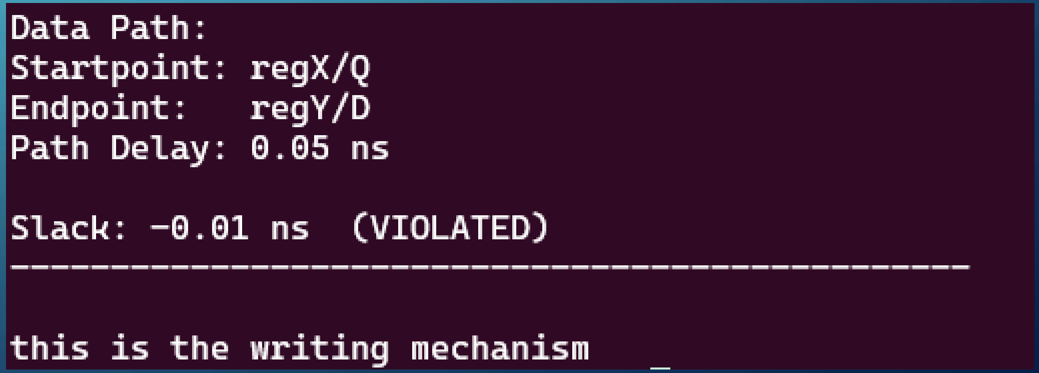
```
mohammad@DESKTOP-7CC1858:~/TCL$ cat timing.rpt
this is the writing mechanism
```

The picture on the right is the new timing report after editing

The second way is to append to a file like this **(notice the a in the mode)**



```
set f [open "timing.rpt" a]
puts $f "this is the writing mechanism"
close $f
```



```
Data Path:
Startpoint: regX/Q
Endpoint:   regY/D
Path Delay: 0.05 ns

Slack: -0.01 ns  (VIOLATED)
------------------------------------------------


this is the writing mechanism
```

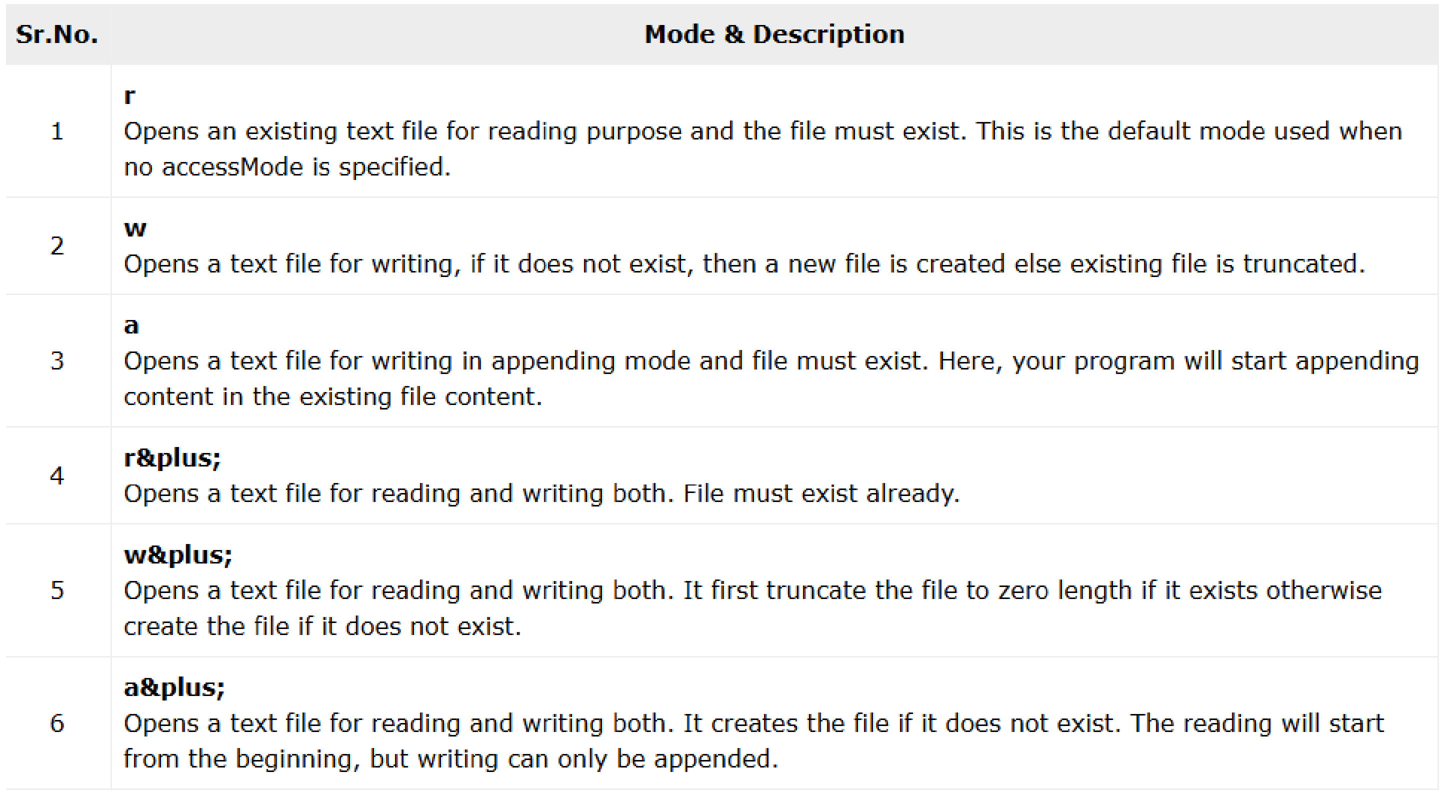
CHECKPOINT

- Now you can open, read and write on a file as well as parsing the whole contents of a file line by line
  but how can you extract the needed information from a timing report for example?

  Using regular expressions (**regex**) in TCL will help you to extract the needed information from a file without needing to read the whole content and this will be our next topic

APPENDIX

- Modes of file I/O in TCL



Sr.No.	Mode & Description
1	**r** Opens an existing text file for reading purpose and the file must exist. This is the default mode used when no accessMode is specified.
2	**w** Opens a text file for writing, if it does not exist, then a new file is created else existing file is truncated.
3	**a** Opens a text file for writing in appending mode and file must exist. Here, your program will start appending content in the existing file content.
4	**r&plus;** Opens a text file for reading and writing both. File must exist already.
5	**w&plus;** Opens a text file for reading and writing both. It first truncate the file to zero length if it exists otherwise create the file if it does not exist.
6	**a&plus;** Opens a text file for reading and writing both. It creates the file if it does not exist. The reading will start from the beginning, but writing can only be appended.

REFERENCES

- Tutorial point
- One Compiler
- TCL Manual

If you need help with more resources please feel free to contact me, or lets chat on Calendly :

Lets Chat