## The Parser

The Parser is the software component that reads the command line such as “ls -al” and puts it into a data structure called ***Command Table*** that will store the commands that will be executed.

## The Executor

The executor will take the command table generated by the parser and for every SimpleCommand in the array it will create a new process. It will also if necessary create pipes to communicate the output of one process to the input of the next one. Additionally, it will redirect the standard input, standard output, and standard error if there are any redirections.

The figure below shows a command line “A | B | C | D”. If there is a redirection such as “**< *infile***” detected by the parser, the input of the first SimpleCommand A is redirected from ***infile***. If there is an output redirection such as “**> *outfile***”, it redirects the output of the last SimpleCommand (D) to ***outfile***.

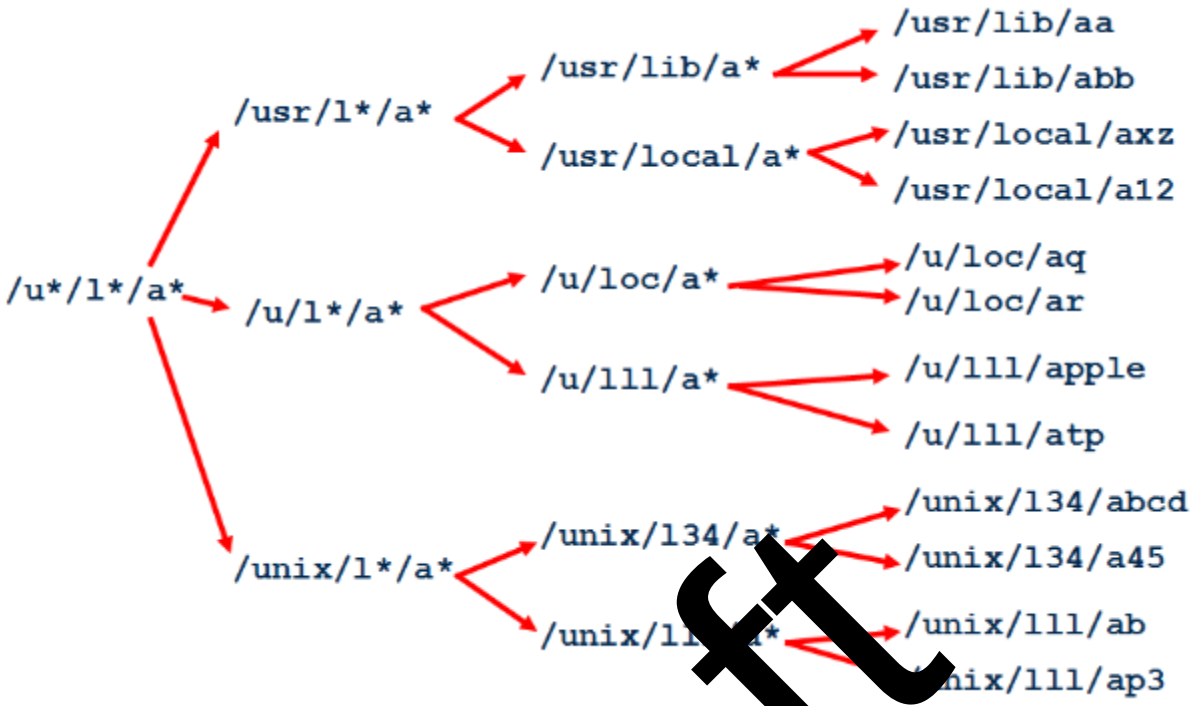

If there is a redirection to errfile such as “**>& errfile**” the stderr of all SimpleCommand processes will be redirected to **errfile**.

## Shell Subsystems

Other subsystems that complete your shell are:

- Environment Variables: Expressions of the form ${VAR} are expanded with the corresponding environment variable. Also the shell should be able to set, expand and print environment vars.
- Wildcards: Arguments of the form a*a are expanded to all the files that match them in the local directory and in multiple directories .
- Subshells: Arguments between `` (backticks) are executed and the output is sent as input to the shell.

We highly recommend that you implement your own shell following the steps in this chapter. Implementing your own shell will give you a very good understanding of how the shell interpreter applications and the operating system interact. Also, it will be a good project to show during your job interview to future employers.






You may implement the wildcard Strategy in the following way.

Write a function expandWildcard(prefix, suffix) where

prefix- The path that has been expanded already. It should not have wildcards.

suffix – The remaining part of the path that has not been expanded yet. It may have wildcards.

The prefix will be inserted as an argument when the suffix is empty

expandWildcard(prefix, suffix) is initially invoked with an empty prefix and the wildcard in suffix. expandWildcard will be called recursively for the elements that match in the path.