The Shell (1)

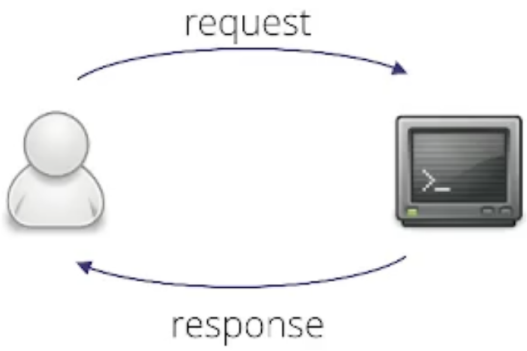
Terms for referring to the shell:

- **shell**
- **terminal**
- **console**
- **command line**

Below are the names of software providers privding a shell on different systems:

- (command) prompt
- xterm
- rxvt
- konsole
- (gnome)-terminal
- putty (windows)

At its core the shell is a user interface, the user types a request and then the shell provides a response to the request.



The Prompt

- **the indicator from the shell that says it is ready to receive the user's next request**
- usually a single character

for example:

- **$ -** you are in a normal POSIX shell (sh)
- **# - you are in a root shell**
  - this indicates you need to be careful as the root shell can apply **system-wide** changes fairly easily
- **% - you are probably in the C shell**
  - many programming languages have their own unique shells
- **> - this indicates we are on a continuation line e.g. inside a string**
  - to exit this CTRL C or close double quotes to end the string

If you see an usualy prompt you are likely inside an interpreter of some programming langauge, or youre working with an interactive peice of software

Useful Shell Tricks

- TAB: auto-completes a command or filename
  - as long as it is unabmiguous e.g. javaassignments and javaprojects youd need to get past java and type the first letter of the second part
- **DOUBLE TAB:** show a list of possible completions
- **UP/DOWN:** Scroll through history
- **^R text:** search history for commands

Builtins

which

- **$ which ls** - tells you where the file containing the ls program resides on the machine
  - /bin/ls
- $ which cd - built-in command that is part of the shell - so doesn't have a location on computer:

```
tom@Toms-MacBook-Pro ~ % which cd
cd: shell built-in command
```

Options & Conventions:

```
$ ls       // shows files in the directory

$ ls -l    // shows more info about the size of the file and
date it was
                        last modified

$ls -a     // we see hidden files, these are identified with
their file name
```

```
                          starting with a dot (.) e.g. .secretf
ile
```

```
tom@Toms-MacBook-Pro ~ % ls -a
.            .viminfo        MakeFile
..           .vscode         Movies
.CFUserTextEncoding .zprofile       Music
.DS_Store        .zsh_history        Pictures
.RData           .zsh_sessions       Public
.Rhistory        .zshrc          c_projects
.Trash           .zshrc.save     c_projects.c
.bash_history        Add_ASM         hello_world
.bash_profile        Applications        hello_world.c
.config          Counter.circ        labTestPart1.c
.gitconfig       Desktop         main1.c
.lesshst         Documents       prime_numbers
.local           Downloads       prime_numbers.c
.m2          IdeaProjects        tf-stuff
.oracle_jre_usage    Library         toprow.c
.ssh             Logisim         zsh.save
tom@Toms-MacBook-Pro ~ %


// ls -l below:
tom@Toms-MacBook-Pro Desktop % ls -l
total 13544
drwxr-xr-x@ 17 tom  staff      544 20 Feb 13:13 Computer Science
drwxr-xr-x   6 tom  staff      192 19 Feb 23:55 JavaProjects
drwxr-xr-x   2 tom  staff       64 30 Jan 12:06 Leet Code Exerci
drwxr-xr-x   8 tom  staff      256  5 Sep 09:25 Overtime Sheets
drwxr-xr-x  18 tom  staff      576 13 Feb 11:21 Personal
drwxr-xr-x@  5 tom  staff      160 11 Dec 11:14 Personal Project
drwxr-xr-x  11 tom  staff      352 26 Jan 20:48 Programming
drwxr-xr-x@ 11 tom  staff      352  8 Feb 23:51 Psychology (BSc)
```

```
drwxr-xr-x@  3 tom  staff       96 17 Oct 02:02 RStudio.app
drwxr-xr-x@  3 tom  staff       96 11 Oct 00:12 Visual Studio Co
drwxr-xr-x  29 tom  staff      928  4 Jan 16:54 assembly_practis
drwxr-xr-x   3 tom  staff       96 31 Jan 13:39 git_tutorial
-rw-r--r--@  1 tom  staff  6933898  3 Oct 14:03 logisim-generic-
drwxr-xr-x@  9 tom  staff      288  4 Jan 16:54 nand2tetris
drwxr-xr-x   7 tom  staff      224 17 Feb 17:51 software_tools
```

```
tom@Toms-MacBook-Pro ~ % ls
Add_ASM     Downloads   Movies      c_projects.c    prime_number
Applications    IdeaProjects    Music       hello_world prime_nu
Counter.circ    Library     Pictures    hello_world.c   tf-stuff
Desktop     Logisim     Public      labTestPart1.c  toprow.c
Documents   MakeFile    c_projects  main1.c     zsh.save
tom@Toms-MacBook-Pro ~ %
```

help

```
ls --help
```

- provides lots of help text related to ls
- explains what can be passed to the ls program, and what these options do

man pages

```
man [SECTION] COMMAND

man COMMAND // for info about specific commands

e.g. man ls // info about ls
```

- section 1 = shell commands
- section 2 = system calls
- section 3 = C library
- **therefore “man 1 printf” and “man 3 printf” are different**

Shell Expansion

- It's a process where the shell automatically expands or translates certain characters or sequences of characters in commands into a different set of characters or a list of items before the command is executed. This feature simplifies many types of operations, making it more convenient to work with files, directories, and other command arguments. There are several types of shell expansions:

1. **Brace Expansion**: Generates arbitrary strings. For example, `file{1,2,3}.txt` expands to `file1.txt file2.txt file3.txt`.
2. **Tilde Expansion**: Expands `~` to the home directory of the current user or the specified user. For example, `cd ~` changes the directory to the current user's home directory.
3. **Parameter and Variable Expansion**: Expands variables to their values. If you have `VAR="Hello"`, then `echo $VAR` expands to `echo Hello`.
4. **Command Substitution**: Allows the output of a command to replace the command itself. For example, `echo $(date)` will print the current date and time.
5. **Arithmetic Expansion**: Allows for arithmetic operations to be performed and the result to be returned. For example, `echo $((2+3))` will output `5`.
6. **Wildcard Expansion (Globbing)**: Uses patterns to match filenames. For example, `.txt` matches all files in the current directory that have a `.txt` extension.
7. **Quote Removal**: After all the above expansions, the shell removes unquoted instances of certain characters (such as backslashes `\`, quotes `"`, and apostrophes `'`) that are not needed.

These expansions make it easier to work with multiple files, directories, and data within the shell, automating repetitive tasks and allowing for more dynamic and flexible command constructions.

- the shell can interpret certain characters in commands and turn them into arguments and then pass these arguments to programs

for example a common example:

```
cat * // concatenates all files in the current scope
```

Types of Shell Expansion

- * → AKA wildcard or asterix, this takes all file names in the current scope
  - used to refer to all files in the current directory

```
ls a*. // shows all files containing a

ls *.txt // shows all the files ending in .txt
```

Another common thing is to allow certain characters int he filename to vary. Say you have a file called shell.txt and a file called shill.txt and want to capture both of these:

```
ls sh?ll.txt // the '?' could either be e or i
```

- also helpful for numbers, say you want images 0 through 9

```
ls image[0-9].jpg  // lists image001.jpg --> image009.jpg
```

- or can check for charctars:
- say if there is also a program called shall.txt as well as shill and shell

```
ls sh[ai].txt

OUTPUT:

shall.txt  shill.txt



ls sh[ea].txt

OUTPUT:

shell.txt  shall.txt
```

Variable name expansion

- the shell also expands variables that are either set by yourself or the system that refer to various things

```
echo $PWD     // PWD refers to current working directory
```

Shell quoting

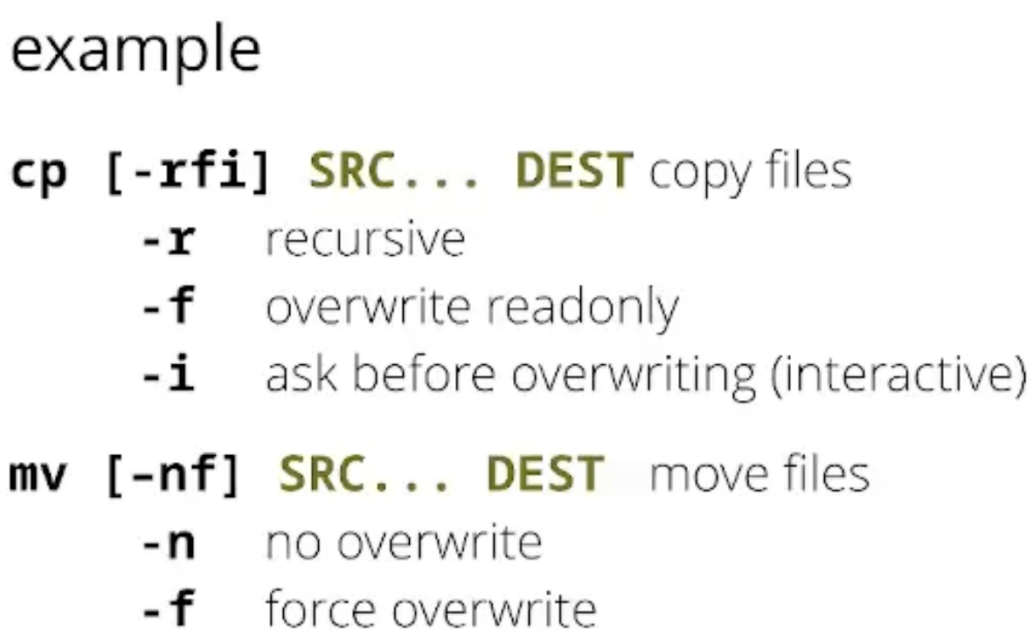
- shell also does more to handle text input
  - important when you want to pass strings to programs without the text being interpretted by the shell
- **double quotes turn off pattern-matching**

    - retains variable interpolation
        - e.g. will still interpret the variable
    - and interprets backslashes
- **single quotes turn off all interpretation**
    - everything would be treated as a string which would be passed to the program

You can also specifically exclude certain characters using the backslash

```
\* \? \[ \$ // says do not treat these as a pattern to be interp
```

globbing example



Finding files

- find program finds files

```
find DIR [EXPRESSION]   // this recursively searches the directo
                                                  // i.e. searches

e.g.
find . -name "a*"
```

- the above example searches for files where you can only remember part of the file name
    - file searching not limited to name, check man pages