Today: Configuration in relation to envieroment (from last time), Scripting

Heavyweight & Lightweight scripting (spectrum, on a philosophical level + historical trends)

`sh` is lightweight langugae -- small and not do very much -- sets up for the big boys to do the actual work) (few but strong -- heavy commands)

`python` is a heavyweight scripting language (lots of operations -- which are small)

to some extent, converging. See command 1 and 2 -- more heavyweight. In python, for example, we now have pytorch script ML, all computation in C u just set up.

Scripting languages are ussuly very forgiving with typographical error -- give you some sensical response, and says "eh... let's keep going"

`sh` For Configuration

Take for example the following command

```
sort <foo.txt | sed 's/a/b/' | grep boot
```

a very brief summery of a vrey complicated configuration

We configured 3 programs and how they should run. We canve them stdin, args, and stdout. Think about doing it with your programs, like python programs and such.

`grep`, and all commands we don't configure, have *defualts*. For example, `grep` by defualt outputs stderr and stout to the terminal.

Another configuration is the current *working directory*. Supposadly, configuration is as if not more important the the software itself. It is disrespected and underdone. *Configuration wizards* try to guess what you will want and configure. *Configuration GUIs* vs *Configuration CLIs* -- for the latter, ther configuration must be expressible as text. The GUIs have the visual advantage, but, its a pain to maintain a picture which can mess up/limit configs.

Executing a `sh` command

`grep abc def ghi` does: 1. finds the executable grep file in usr/bin by consulting the $PATH variable. 2. Puts the arguments on the main mem as a regular array for `grep.exe` 3. return

Aside: Exit Status

- `0`: Success
- up to 127!

grep only:

```
 1- not found
2- trouble-fail
```

Quoting (more problamatic than you would think!)

In `sh`
needed some way to have a space in a string (other than \SPC)

- Single quotes -- explicit, what you se is what you get
- Double quotes -- Can place somethign in there that *gets evaluated* (suggested: use these only for a reason, use the simpler tool, not only for ptrformence, but for orginization -- think about your thinking time, preformence is negligable) Example:

  ```
      "the exit status was $?" -- evaluetes $? and places value as string. Instead we can put there "What did you find: $(grep
  ```

Can nest as deep as you would like.

List of `sh` special charecters

1. "
2. '
3. \
4. `
5. |
6. * -- *filename expension* -- globbing, not regExp. `*` is the `a*.c`
7. ? -- *filename expension* -- just like `*` but `?` matches one charecter only
8. [-- *filename expension* -- a[b-z].c anything in the range -- matches 1
9. & Running in the background, example `emacs &` -- to terminate `fg` (foreground) and then `C-c`.
10. `=` assigments of strings `abc=def` `echo $abc` outputs `def`.
11. SPACE
12. NEWLINE
13. TAB
14. < stdin

15. > -- std out
16. ; seperate args
17. ! -- not POSIX stdardised -- *negates the command* -- negates the exit status.
18. ~

Nonspecial `sh` charecters

`+,-,@,%,^,_,/, \, ,., {,},]`

Tokens (for the `sh`)

(builtins) |,&,|| (logical OR),&& (logical AND),() (run in subshell),<>, <<>> -- `<<EOF` -- std in until you type EOF, ;; , ;

(words) nonspecial chars or strings jammed together with no whitespace. Makes for a single argument. Can concatanate string just by not putting a space between them.

(reserved words) Are not for exceuting a word from usr/bin. Instead they:

- if
- while
- that sorta thing that works with the *shell's control structue*

```
! grep lebron /etc/passwd && echo 'Lebron is missing'
```

will echo lebron is missing if lebron is not found -- think about it

execute command `{COMANDS}` as a block, i.e. inexsesible from outside, etc. `()` does subshell while `{}` executes as the parent shell. Think of to see the diffrence:

```
(cd /etc; grep eggert passwd) VS {cd /etc; grep eggert passwd}
```

the second option actually changes the envieroment --2nd a bit mroe efficient

To *run in parallal*:

```
(A & B) or with curly
```

if then else

```
 if COMMANDS
then COMMANDS
elif COMMANDS
then COMMANDS
else COMMANDS
fi
```

pattern matching

```
    case WORD in
    (pattern) COMMANDS
    (pattern_2) COMMANDS
    (*) COMMANDS
    esac
```

-- can nest if you would like to AND multiple cases.

for in

```
 for $i in WORD
do COMMANDS

done
```

Variable substuiton

- `${abc-def}` -- interpolate `abc` unless it does not exist then use `def`.
- `${abc:-def}` -- treat the empty string as it was not assigned to.

- `${abc?} exit because there is no resonable defult
- `${abc?:}` -- add empty as unset
- `${abc=def}` -- if abc's value is empty assign abc to it and then interpolate def
- `${abc%.c}` printout abc but strip off a `.c` in the end if it has one.

Special Vars

1. $? most recent exit status
2. \$\$ current shell's process id

```
 ps -ef | grep \$\$
matan    3548703 3548697  0 14:06 pts/18   00:00:00 -bash
matan    3556074 3548703  0 15:37 pts/18   00:00:00 ps -ef
matan    3556075 3548703  0 15:37 pts/18   00:00:00 grep --color=auto 3548703
```

3. `$!` last background process id.

help you control all the stuff that you configured

distinvtion between unset and default value

Random Stuff (mostly commands)

1. `expr 3 +4` super slow old syntax -- executes another program
2. `expr $((3 + 4)) -- all in the shell no overhead
3. `$?` exit status of the most recent command that you ran. (2 is err)
4. `kill PROCESS_ID`, polite, add a -9 to end the process.
5. `wait PROCESS_ID` for a process to finish, don't do anything else in the meanwhile.
6. Note **field spilitting** example:

```
 x=$(pwd)
cd /etc
...doStuff...
cd $x
```

Working directory may be made up of multiple words and that will mess us up

7. use `--` to say whatever comes after this is a filename. **never assume that whatever is running your server is your client**

Questions

1. Try to turn 6,7,8 into regExp. Test yourself!
2. what do 17, and 18 do ?
3. What do all the variable substution do.

My opinion on this lesson

This stuff should be learned through practice and stored in a neat manual -- not studied.