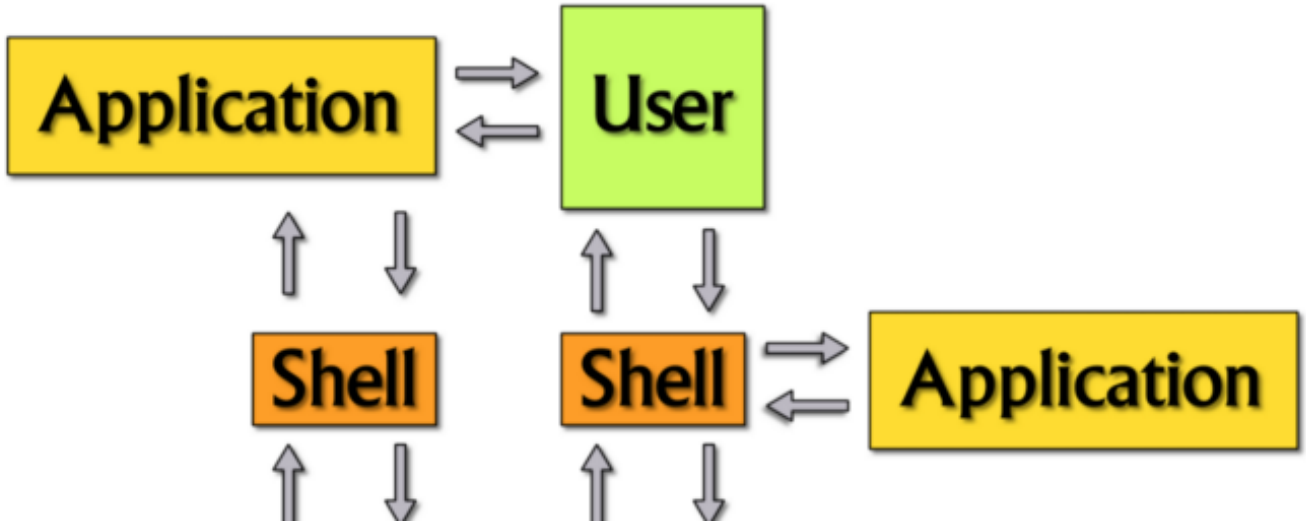
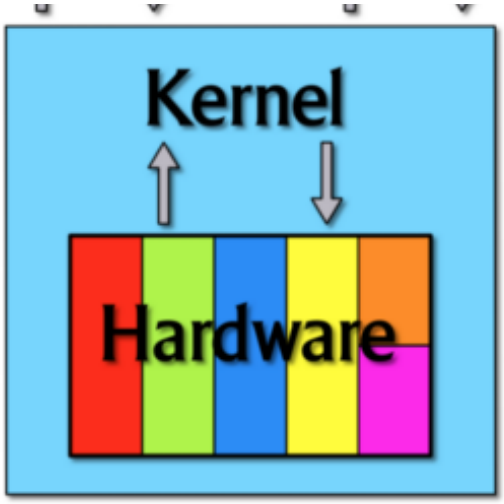
Jimmy Thong Follow
Dec 13, 2016 · 4 min read

Source

What (really) happens when you type ls -l in the shell

To understand what (really) happens when you type ***ls -l*** in the shell, you have to understand what the shell is, in relation to the *kernel*.





Source

The *kernel* is the central, most fundamental part of a computer operating system. It's a program that controls all other programs on the computer, talking to the hardware and software, highly involved in resource management.

The *shell* is an application, one of the two major ways of controlling a computer (GUI the other). It is the way a user *talks* to the *kernel,* by typing commands into the command line (why it's known as the *command line interpreter)*.

```
vagrant@vagrant-ubuntu-trusty-64:~$ ls -l
total 36
-rwxrwxr-x  1 vagrant vagrant 8583 Dec  9 22:56 a.out
drwxrwxr-x  5 vagrant vagrant 4096 Dec  1 20:31 Betty
drwxrwxr-x  7 vagrant vagrant 4096 Dec  4 18:12 c_is_fun
drwxrwxr-x 23 vagrant vagrant 4096 Dec  7 18:15 holbertonschool-low_level_programming
drwxrwxr-x  2 vagrant vagrant 4096 Dec 12 18:08 refinery
drwxrwxr-x  2 vagrant vagrant 4096 Nov 11 03:07 settings
drwxrwxr-x  3 vagrant vagrant 4096 Dec 12 20:33 src
vagrant@vagrant-ubuntu-trusty-64:~$
```

On the superficial level, typing ***ls -l*** displays all the files and directories in the current working directory, along with respective permissions, owners, and created date and time.

On the deeper level, this is what happens when you type "ls -l" and "enter" in the shell:

First and foremost, the shell prints the prompt, prompting the user to enter a command. The shell reads the command ***ls -l*** from the getline() function's STDIN, parsing the command line into arguments that it is passing to the program it is executing.

The shell checks if ***ls*** is an alias. If it is, the alias replaces ***ls*** with its value.

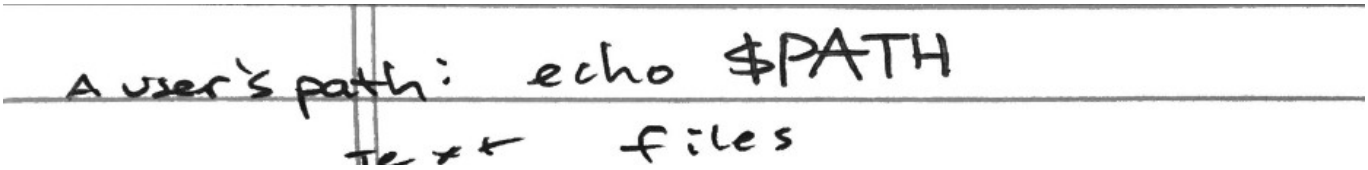
If ***ls*** isn't an alias, the shell checks if the word of a command is a built-in.

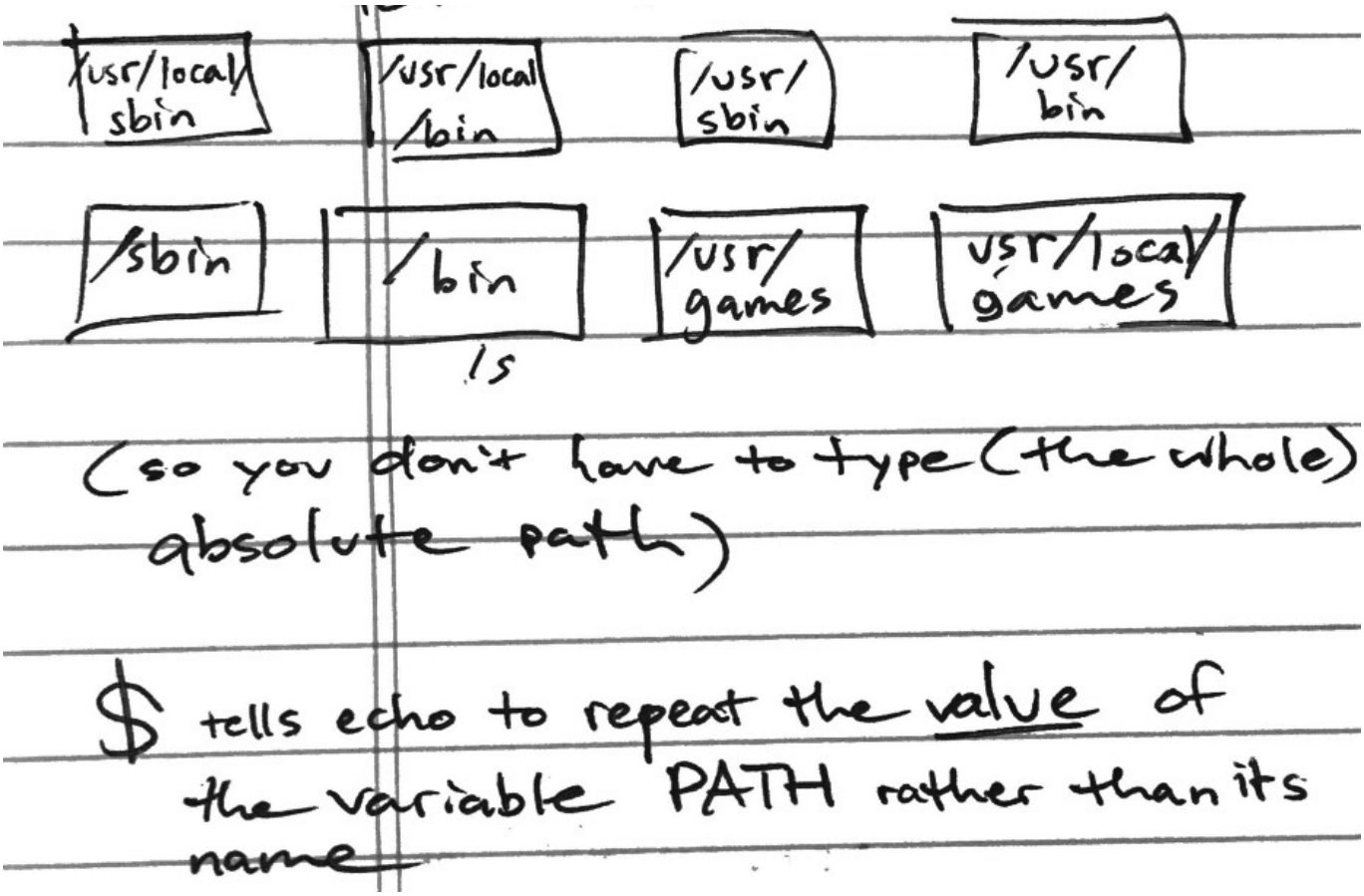
The environment is copied and passed to the new process

Then, the shell looks for a program file called ***ls*** where all the executable files are in the system — in the shell's environment (an array of strings), specifically in the $PATH variable. The $PATH variable is a list of directories the shell searches every time a command is entered. $PATH, one of the environment variables, is parsed using the '=' as a delimiter. Once the $PATH is identified, all the directories in $PATH are tokenized, parsed further using ':' as a delimiter.

```
vagrant@vagrant-ubuntu-trusty-64:~/simple_shell$ vim _createTokenAndChildP.c
vagrant@vagrant-ubuntu-trusty-64:~/simple_shell$ printenv
XDG_SESSION_ID=130
TERM=xterm-256color
SHELL=/bin/bash
SSH_CLIENT=10.0.2.2 51030 22
SSH_TTY=/dev/pts/3
USER=vagrant
MAIL=/var/mail/vagrant
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games
PWD=/home/vagrant/simple_shell
LANG=en_US.UTF-8
SHLVL=1
HOME=/home/vagrant
LOGNAME=vagrant
SSH_CONNECTION=10.0.2.2 51030 10.0.2.15 22
LC_CTYPE=en_US.UTF-8
XDG_RUNTIME_DIR=/run/user/1000
_=/usr/bin/printenv
OLDPWD=/home/vagrant
vagrant@vagrant-ubuntu-trusty-64:~/simple_shell$
```

The ***ls*** binary executable file will be located [in one of the major subdirectories of the '**/usr**' directory] in the file '**/usr/bin/ls**' — '**/usr/bin**' contains most of the executable files (ie. ready-to-run programs).





One $PATH, many directories

(For performance's sake, ***ls*** could either be [a shell] built-in or hashed.)

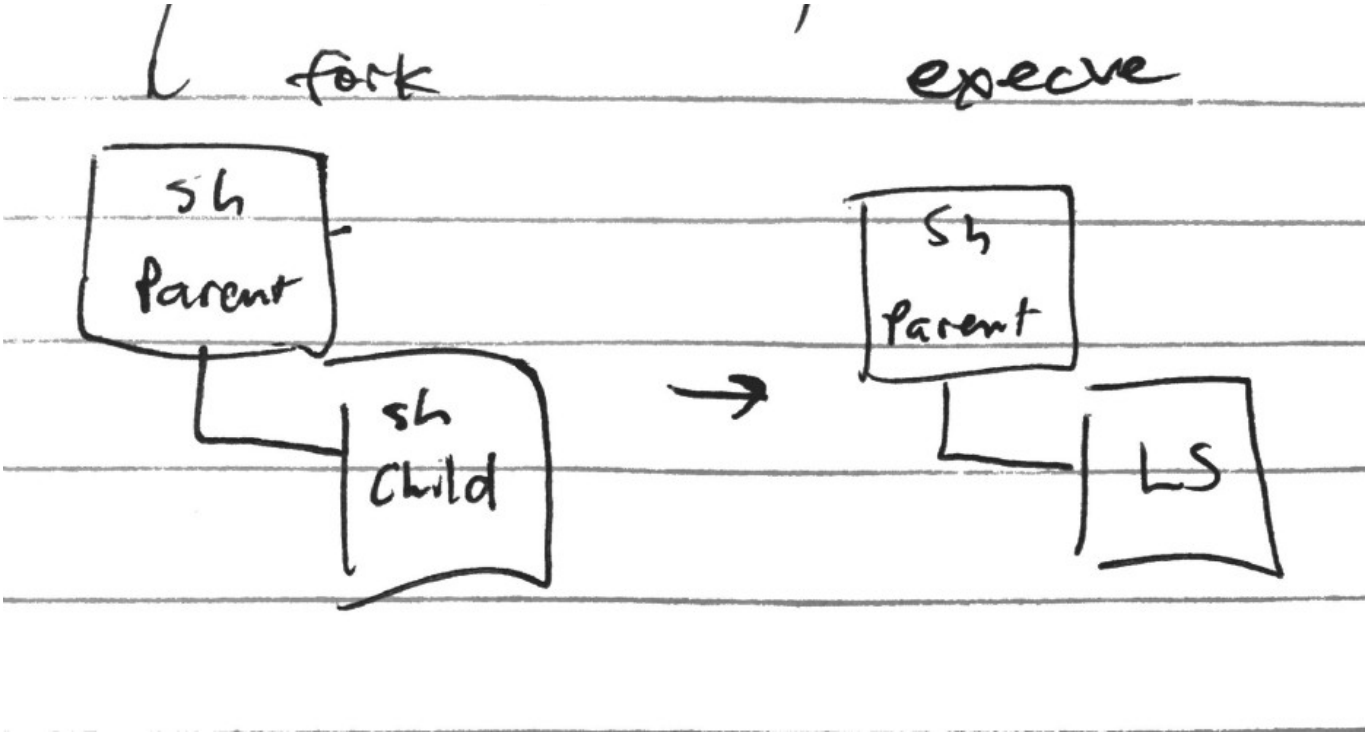
To execute ***ls,*** three system calls are made:

fork / execve / wait

(System calls are calls to the Kernel code to do *something*.)

To create new processes (to run commands) in Unix, the system call ***fork()*** is made to duplicate the (Shell) parent process, creating a child process of the (Shell) parent process.

The system call ***execve()*** is made and does three things: the operating system (OS) stops the duplicated process (of the parent), loads up the new program (in this case: ***ls***), and starts (the new program ***ls***). ***execve()*** replaces defining parts of the current process' memory stack with the new stuff loaded from the ***ls*** executable file.



Sh = Shell

The input to execve (for ***ls -l***) will be:

```
execve("/bin/ls", {"ls" or "/bin/ls", "-l", NULL}, NULL);
```

Say you have your shell, in the beginning you will have to use `/bin/ls` and not just ls, so you have:

```
$ /bin/ls -l
```

Tokenize this line into stuff = {"/bin/ls", "-l", NULL}

Then do execve(stuff[0], stuff, NULL).

Meanwhile,

the parent process continues to do other things, keeping track of its children as well, using the system call ***wait()***.

If the executable binary (***ls***) file does not exist, an error will be printed.

Lastly,

After ***ls -l*** is executed, the shell executes shutdown commands, frees up memory, exits, and re-prompts the user for input.

. . .

References:

http://www.makeuseof.com/tag/linux-kernel-explanation-laymans-terms/

https://brennan.io/2015/01/16/write-a-shell-in-c/

http://www.linfo.org/usr_bin.html

Special thanks to: my project partner WalLee2 & https://twitter.com/1million40

Terminal Programming Shell Kernel