sh

- sh is a command interpreter. There are many shells. The common ones:

  /bin/sh Bourne shell – the original AT&T Bell Labs UNIX shell

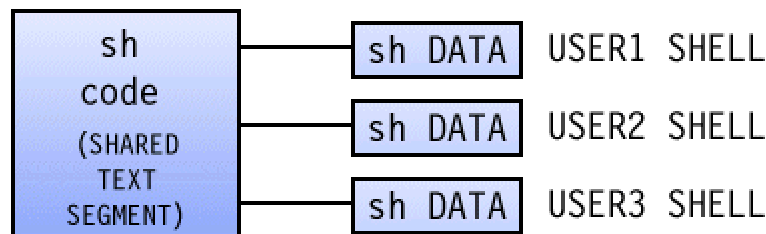
  /bin/csh C shell – the original BSD shell

  /bin/ksh Korn shell – a rewrite/update of the Bourne shell, also from Bell Labs

  /bin/tcsh "an enhanced, but completely compatible version of the Berkeley UNIX C shell" (see http://www.tcsh.org/)

  /bin/bash the GNU Bourne Again Shell – the default Linux shell
  "Bash is an sh-compatible shell that incorporates useful features from the Korn shell (ksh) and C shell (csh). It is intended to conform to the IEEE POSIX P1003.2/ISO 9945.2 Shell and Tools standard. It offers functional improvements over sh for both programming and interactive use. In addition, most sh scripts can be run by Bash without modification." (see http://www.gnu.org/software/bash/)

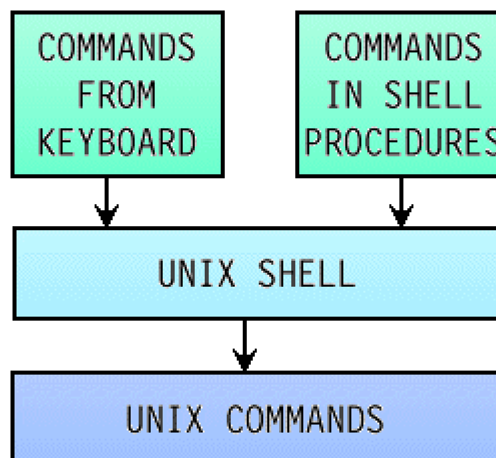
- Each shell supports a powerful command language.
- Each invocation of the *sh, csh, ksh, tcsh, or bash* program is called a **shell**.



- Most operating systems allow you to invoke commands one at a time from a terminal.

- Because the shell gives the user a high-level language interface to the operating system, the user can:
  - ➔ Combine commands to form new commands.
  - ➔ Pass positional parameters to a command.
  - ➔ Easily add or rename commands.
  - ➔ Execute commands within loops.
  - ➔ Execute commands conditionally.
  - ➔ Send commands into the background.

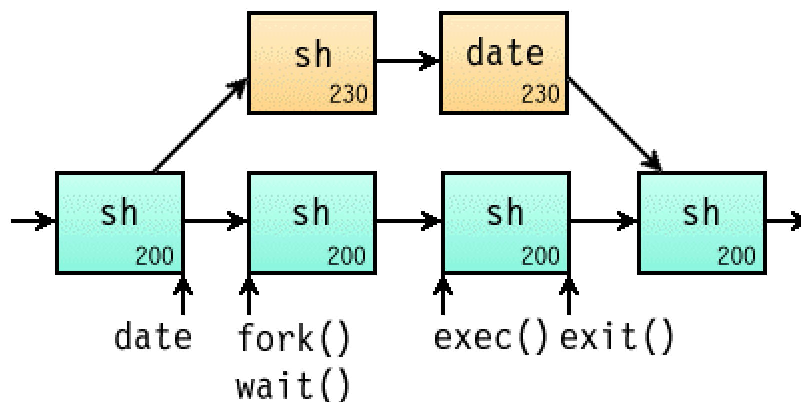
The Shell Command Interface



- Commands are typed at the user's terminal or read from a file.

Processes

- A **process** is an executing program.
- The kernel maintains a list of all processes.
- Most processes are waiting for I/O, or for some system function to complete.
- Processes compete for CPU time.
- Processes may spawn new **child** processes.
- The *shell* is a process.
- Each person logged into the operating system is assigned a unique shell process (you can check with **ps -a** or **ps -e**).
- When you type a command:



→ Your shell makes a copy of itself through the *fork(2)* system call (PID 230).

→ Your shell puts itself to sleep (via the *wait(2)* system call) until its *child process* (PID 230) calls *exit(2)*.

→ The child process calls *exec(date)* to overwrite itself with the code for *date*.

→ When the *date* command is finished, it calls *exit()*.

→ The child process (230) dies, and your shell wakes up again, and displays the prompt.

Examples of Typed In Commands

Writing Shell Procedures

- First put some commands into a file:

```
$ cat > GREETINGS
echo HI THERE
echo HOW ARE YOU?
^D
$
```

- Make the file executable:

```
$ chmod +x GREETINGS
$
```

- You have just added a new command to UNIX:

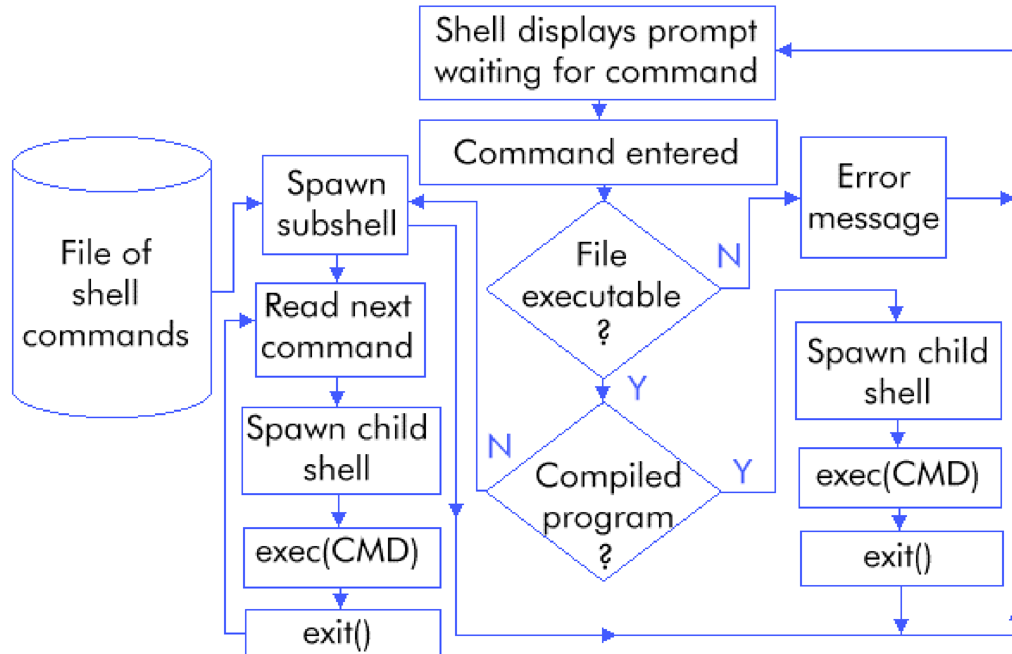
```
$ GREETINGS
HI THERE
HOW ARE YOU?
$
```

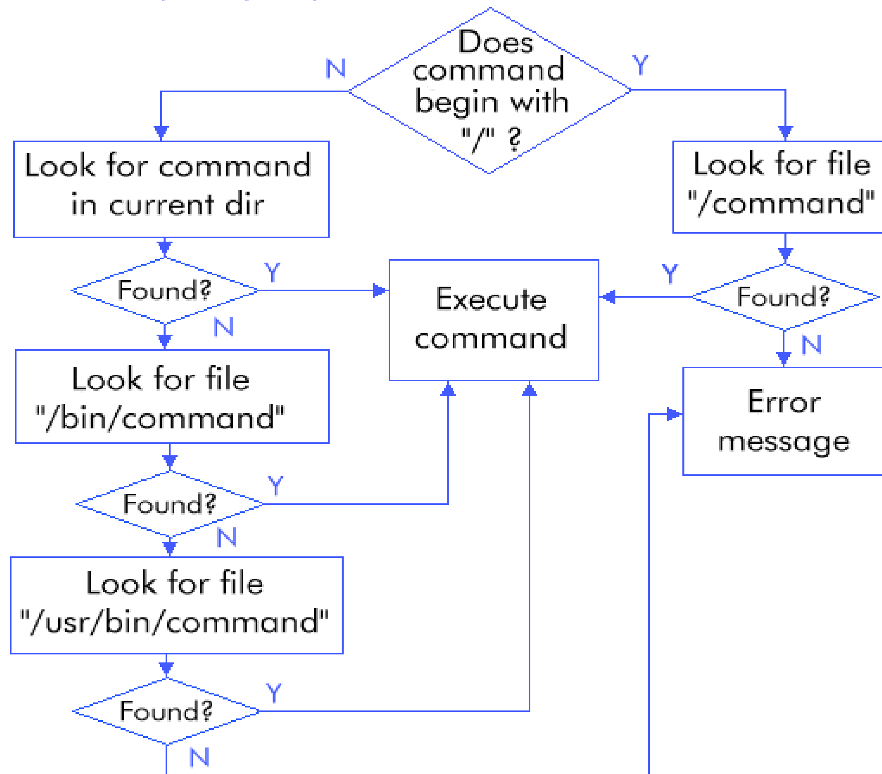
- You can invoke a non-executable procedure file:

```
$ sh GREETINGS
HI THERE
HOW ARE YOU?
$
```

Invoking Compiled Commands and Procedure Files

How the Shell Finds Commands





- Automatic path searching provides a convenient way to execute any accessible command.
- The particular sequence of directories searched is set by the **PATH** variable, and may be changed.
- If you have the current directory (.) in your path, you can execute a command or script from the current directory by simply typing the name of the command. Otherwise, you have to type ./*command*.
- It is easy to have a private version of a public command. Place it in your home directory, and add your home directory to your path *before the standard directories*.

For example,

```
$ echo $PATH
/bin:/usr/bin:/usr/local/bin
$ PATH=$HOME:$PATH
$ echo $PATH
/usr/user1:/bin:/usr/bin:/usr/local/bin
$
```