Bash shell

by moshahmed@gmail.com NITK

Bash shell

Default terminal shell (command interpreter) on Linux is /bin/bash

Windows users can download c:/cygwin/bin/bash with cygwin.

History:

sh (ATT unix) → ksh → bash (current)

Obselete: zsh, csh, tcsh

Windows Cygwin Setup

- Google > Download Cygwin > setup*.exe
- Run cygwin setup.exe > next > next …
- Installs in c:/cygwin64 or c:/cygwin
- start > run > [cmd as admin]

> set PATH="c:\cygwin64\bin;%path%"

Save PATH in registry for all users

> setx PATH "c:\cygwin64\bin;%path%" –m

> bash

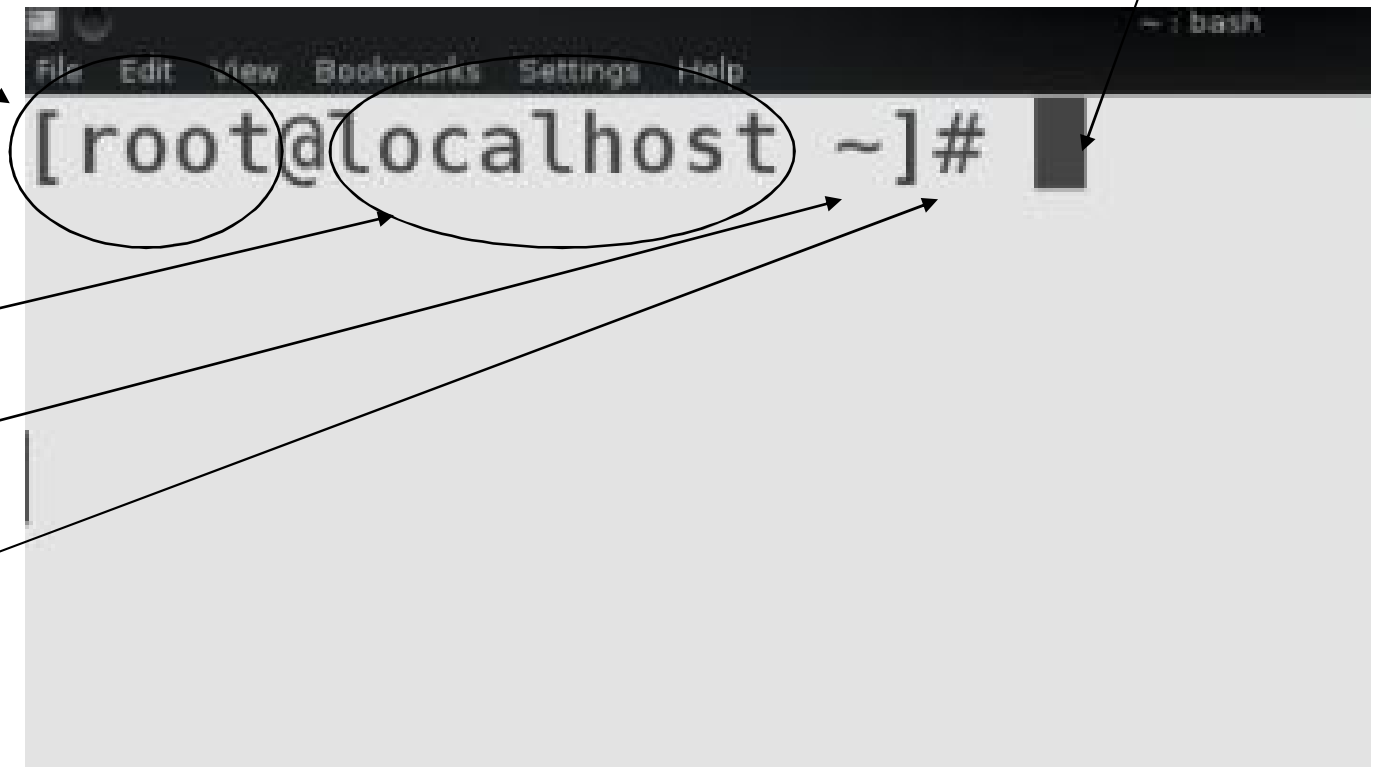
$

Start bash in a terminal

- start > terminal

Cursor, type input here

Username



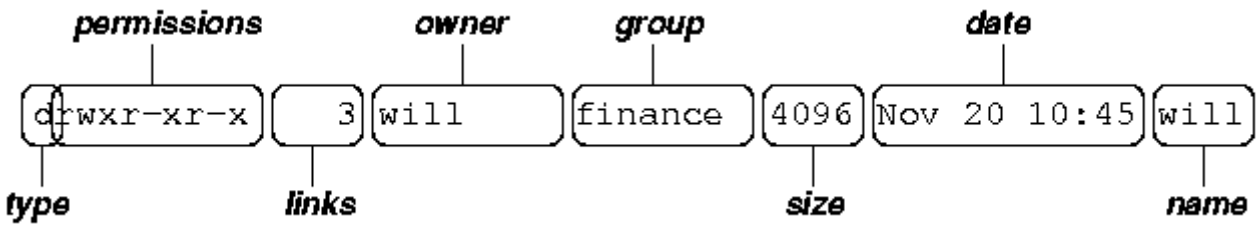
hostname

current directory
pwd, ~ is home dir

prompt means
superuser

ls – list directory and files

bash$ ls –l will .. –l option for long details



cat (concatenate, print file)

cat filename #.. prints the filename to stdout, which is the terminal screen, also called /dev/tty.

```
File  Edit  View  Bookmarks  Settings  Help
[root@localhost ~]# cat /etc/shells
/bin/sh
/bin/bash
/sbin/nologin
/bin/dash
[root@localhost ~]#
```

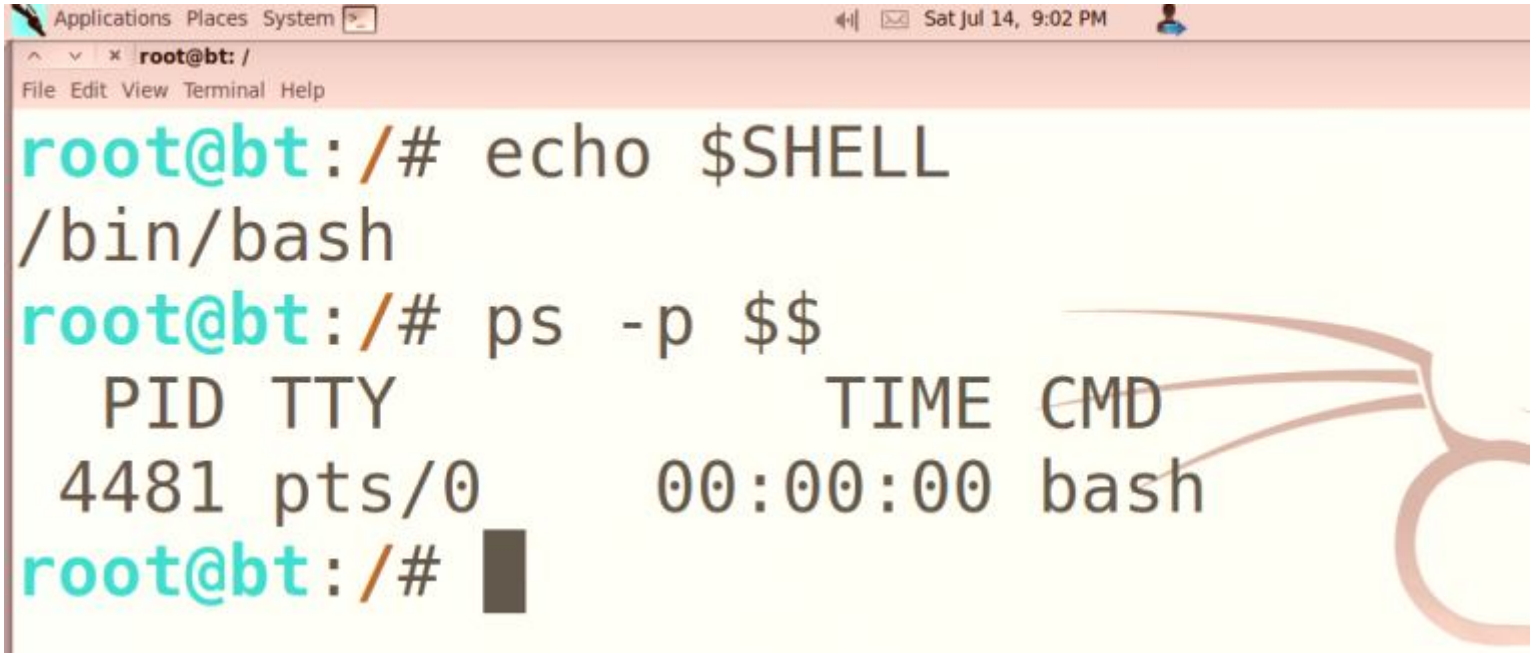
Environment

echo $HOME

ps -p $$ .. process info

$$ .. Is the pid (process id) of this shell.

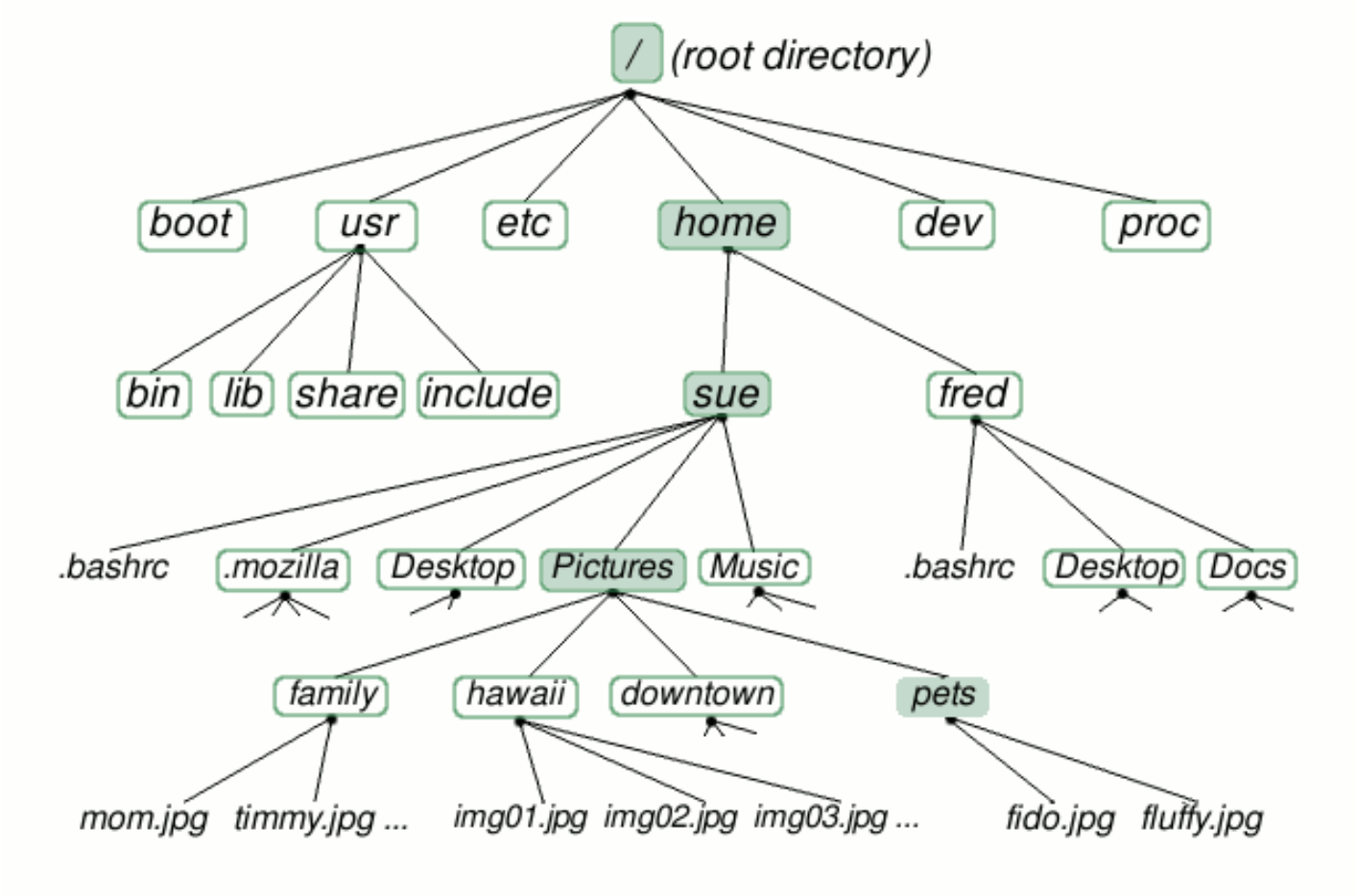
TTY .. is the controlling terminal of this bash



Unix filenames

- / is the root
- /dev/ .. are the devices, like keyboard, tty, harddisks.
- /bin .. are the programs
- /home/john .. user directory
- /etc .. system configuration (like registry)
- /usr .. user applications
- Symlinks, one link can point to another

Unix file system



Common terminal keys

- Control-Z .. suspend command
  - fg .. restart command in foreground
  - bg .. send command into background
- Control-C .. interrupt current command
- Control-\ .. Kill current command
- Control-D .. EOF to logout
- Control-S .. Stop screen output
- Control-Q .. continue screen output.

Readline (Command line editing) in bash

- C-a .. beginning of line
- C-e .. end of line
- C-r .. search history
- Up-arrow .. previous history command
- Down-arrow .. next history commands
- C-k .. delete to end of line

See google, same as Emacs editor keys, can remap keys in ~/.inputrc

Common Unix commands

- ls files .. list file or directory
- cat files .. print files to stdout
- man xyz .. show manual help page for xyz
- cp source target .. copy source to target
- mv source target .. move
- rm source .. remove source
- cd /usr/local .. change directory to
- pwd .. show present working dir
- grep regexp file .. search regexp in files
- more files .. show files page by page.

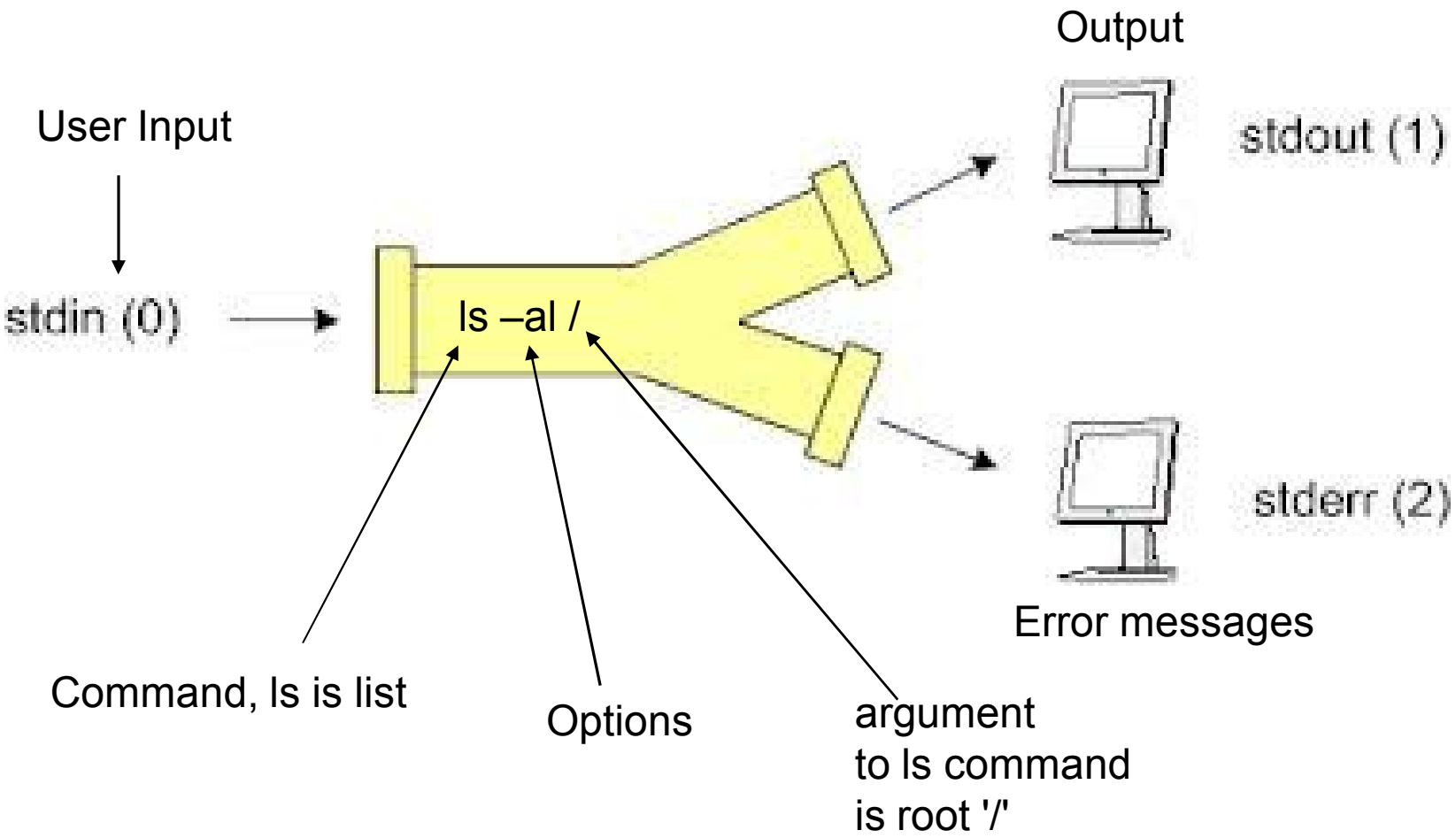
More Unix commands

- ps … show processes
- who … show who is logged on
- date .. show date
- cal .. show calendar
- stty .. terminal settings
- chmod .. change dir/file permissions
- vim files .. vi improved editor
- emacs files .. emacs editor

Network commands

- ping host .. check network connection to host
- tracert host .. trace route to host
- nslookup .. DNS name lookup
- mail .. read or send email
- ftp .. file transfer
- wget urls .. download urls
- telnet host .. login to host
- ssh host .. secure shell login to host
- finger user@host .. find out about user on host

Process and its IO



Saving output to a file

Count number of lines in /etc/shells and save it to x

```
$ wc –l /etc/shells  > x
$ cat x
   16 /etc/shells      .. number of lines in file
```

Save errors to a file (stderror is fd2):

```
$ gcc –Wall bigfact.c 2> errors.txt
$ more  errors.txt  ... show the file page by page
```

Reading input from a file

```
$ wc –l <  /etc/shells
   16 lines
```

Redirect input and output

```
$ wc –l < /etc/shells > x
```

Saving outputs

Save output, redirect stdout to a file.

```
$ wc /etc/shells > /tmp/y
$ cat /tmp/y
  16  16 186 /etc/shells
```

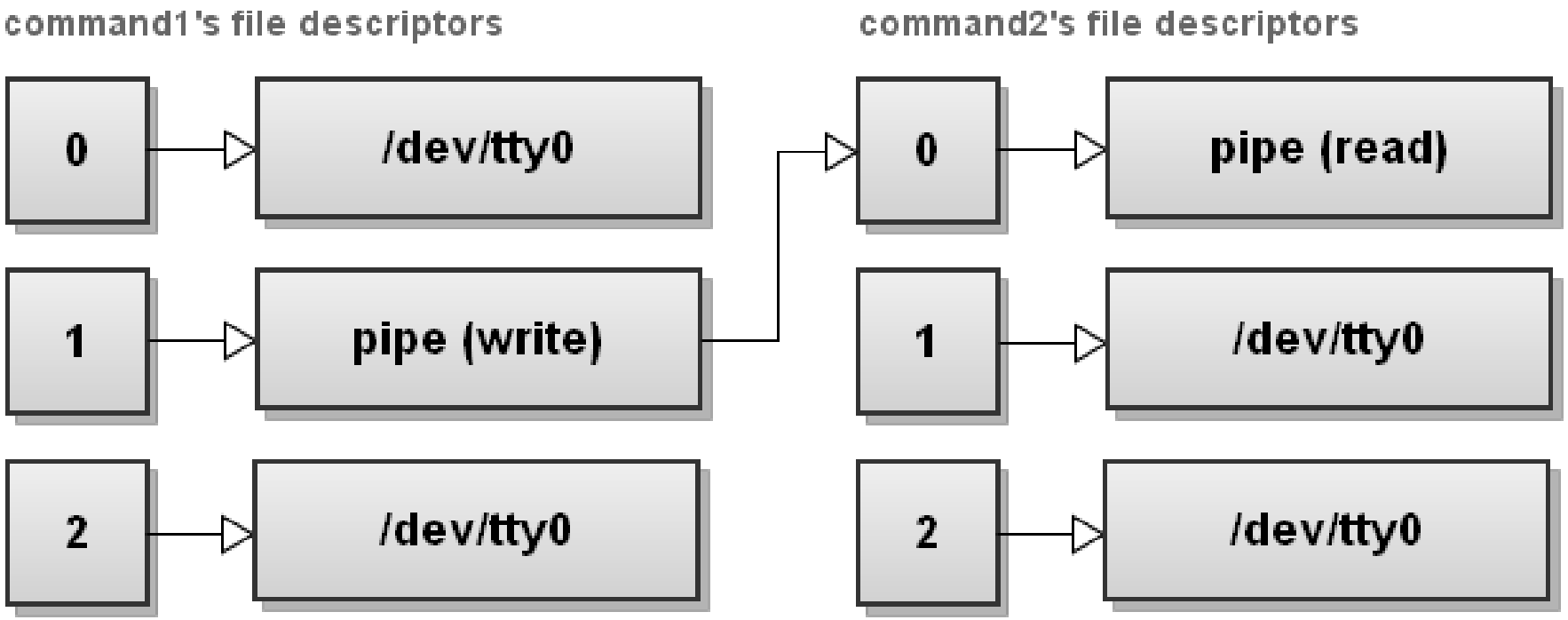
(means 16 lines,16 words, 186 chars in /etc/shells)

Save output and error messages of gcc, send stdout to file x, and also redirect stderr/2 to stdout/1.

```
$ gcc -Wall bigfac.c  >  x     2>&1
```

pipe, and io redirection

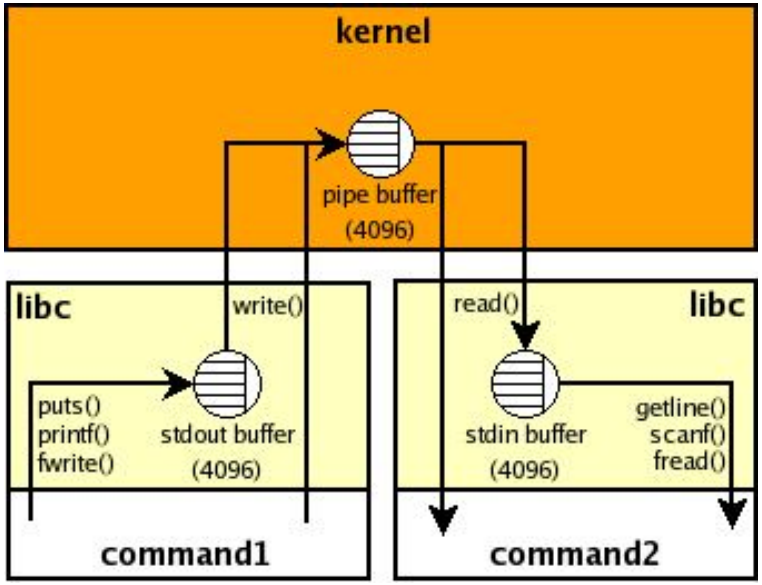
$ command1 | command2



Piping '|'

Pipe output of first cmd
to next cmd, example

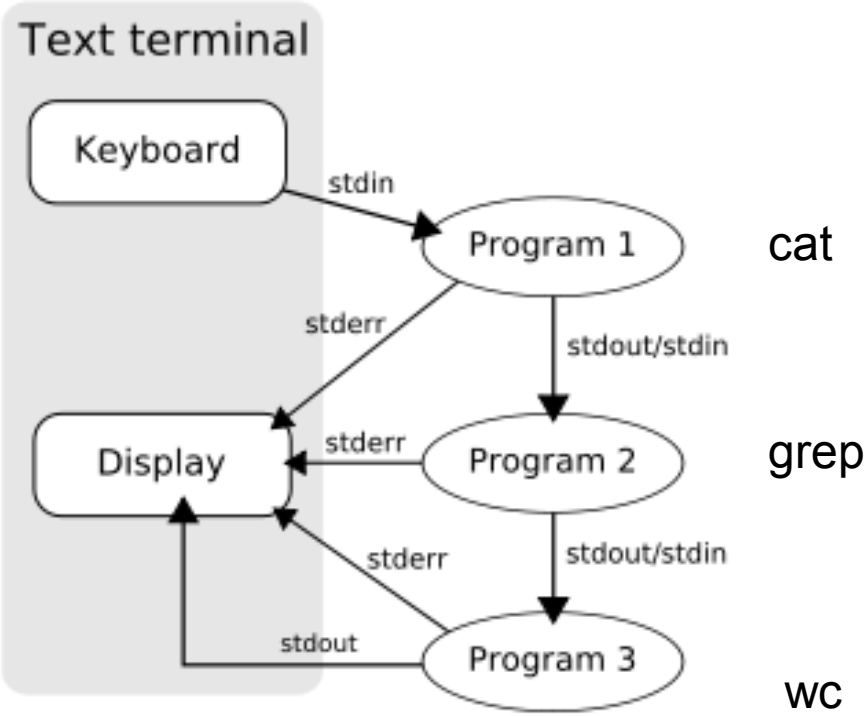
```
$ cat /etc/shells | wc
  16 16 186
```



Pipe example with 3 commands

Example: Count number of lines in file containing the string 'sh' or 'SH' ..

```
$ cat /etc/shells |
  grep -i sh |
  wc -l
2
```



Running cmd in background

```
$ wc /etc/shells > /tmp/x  &
  [1] 2804
```

... process number of background job.

```
$ ls
  [1]+ Done
```

... later background job is done

Quoting arguments

```
$ echo *          .. '*' is globbed into matches
home etc usr
$ echo "*"        .. prevent globbing of *.
*
$ echo \*         .. backslash quotes next char
*
```

Quoting Variables

```
$ echo $HOME
  /home/john
$ echo 22${HOME}99  #  Paren around var
  22/home/john99
$ echo "$HOME"  # Double quoted is expanded
  /home/john
$ echo ‘$HOME’   # Single quoted is not expanded
  $HOME
$ echo \$HOME    # backslash protects next char
  $HOME
```

Inline functions

```
# Using $(command )
$ echo  "TIME1=$(perl -e ‘print time’)"
TIME1=1365864924

# Using backquotes `command`
$ echo  `TIME2=$(perl -e ‘print time’)`
TIME2=1365864925
```

Bash aliases

Make 'dir' same as 'ls –al' command

```
$ alias dir='ls –al'
$ alias date-is='date +%Y-%m-%d'
$ date-is
2013-04-13
```

Bash functions

```
$ function  print_dates ( ) {
   echo DATE_SECONDS=$(perl -e "print time")
   echo DATE_YESTERDAY=$(date --date="1 days ago" +%Y-%m-%d)
   echo DATE_TODAY=$(date --date="0 days ago" +%Y-%m-%d)
   echo DATE_TOMORROW=$(date --date="1 days" +%Y-%m-%d)
}
$ print_dates
DATE_SECONDS=1365864924
DATE_YESTERDAY=2013-04-12
DATE_TODAY=2013-04-13
DATE_TOMORROW=2013-04-14
```

bash scripting

```
$ cat script
  #!/bin/bash
  # my first comment in this file.
  echo "My first script, hello $USER"
$ chmod +x script
$ ./script
  My first script, hello john
$ bash –x –v script      #  .. To debug verbose
  My first script, hello john
```

Bash script commands, if then

```
$ cat myscript1.sh      # comment.
if [[ file1 –nt file2 ]] ;then
    echo "file1 is newer"
;elsif [[ 20 –gt 5 ]] ;then
    echo "20 is greater than 5"
;else
    true; # dummy stmt.
; fi
```

case stmt

```
$ cat myscript2.sh
case $# in
    0) echo You typed no arguments ;;
    1) echo You typed $1 ;;
    2) echo You typed $1 and $2 ;;
    *) echo You typed $* ;;
esac
```

for loop

```
$ for user in a b c ;do
      ls –l /home/$user
; done > list.txt
```

while loop

```
$ file=/tmp/x.log
$ while [[ ! –s $file ]] ; do
  echo waiting for $file to fill up
  sleep 1
  done
```

Perl power user

Fix spelling of 'thier' to 'their' in all c files

```
$ perl -p -i.bak -e 's/\bthier\b/their/g'  *.c
```

---

s/// is Substitute/search-regexp/replacement/

Options:

-p print .. print each line after substitute

-i.bak .. save original as file.bak

-e expr .. to execute perl expression on each line

Windows / Unix differences

	Dos/Windows	Unix
File separator	\	/
Root	C:\	/
Line ending (Fix dos2unix, unix2dos)	\r\n	\n
Shell	cmd.exe	bash
File case	dir == DIR	ls != LS
Syntax	Inconsistent	Good
variables	%USER%	$USER

Windows commands with same names as Unix commands:

- echo, find, date, mkdir, link, time (for if)

- On windows, use doskey for aliasing, e.g.
- @doskey find=c:\cygwin64\bin\find.exe $*

Other tools:

- ssh (secure shell)
- putty (ssh from windows to unix)
- tmux/ gnu screen (virtual terminal)
- find, grep,

Scripting languages:

- perl, python3, nodejs

References

- *Bash manual* (gnu bash version 5)
- *Unix Programming Environment*, by Kernighan and Pike.