# The Shell

David Morgan



# What is a shell?

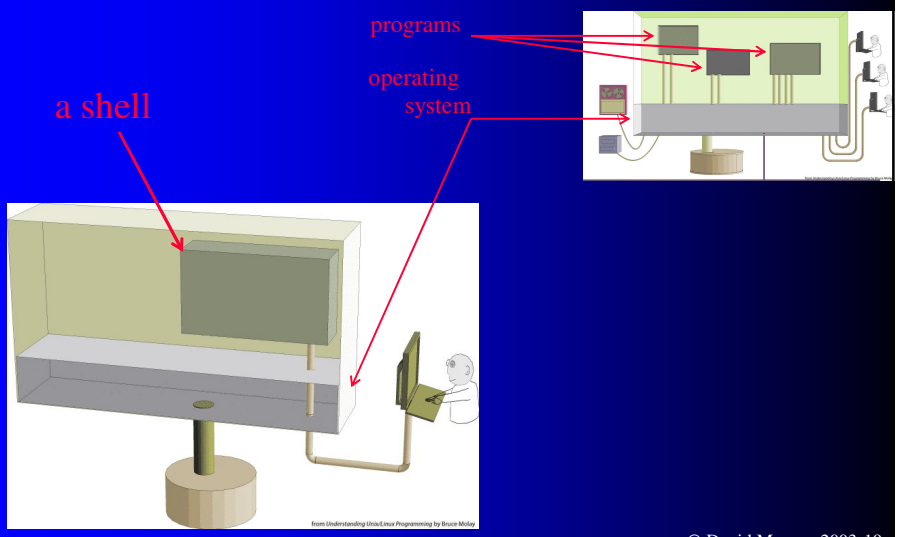
- a “command shell” is a program
  - defined by its function: it handles commands
  - /bin/sh, or maybe /bin/bash or /bin/csh or /bin/zsh (different ones)
- a “user shell” is also a program
  - but not defined by its function: it could do/be anything
    - *could* be a command shell
    - need not be
  - defined as the program that runs in response to a successful login
  - determined for each user in /etc/passwd (the list of defined user accounts)

# What is “the shell” not?

- *not* the operating system
  - merely one program that can run under the OS
- *not* necessary
  - machine doesn’t necessarily even run it

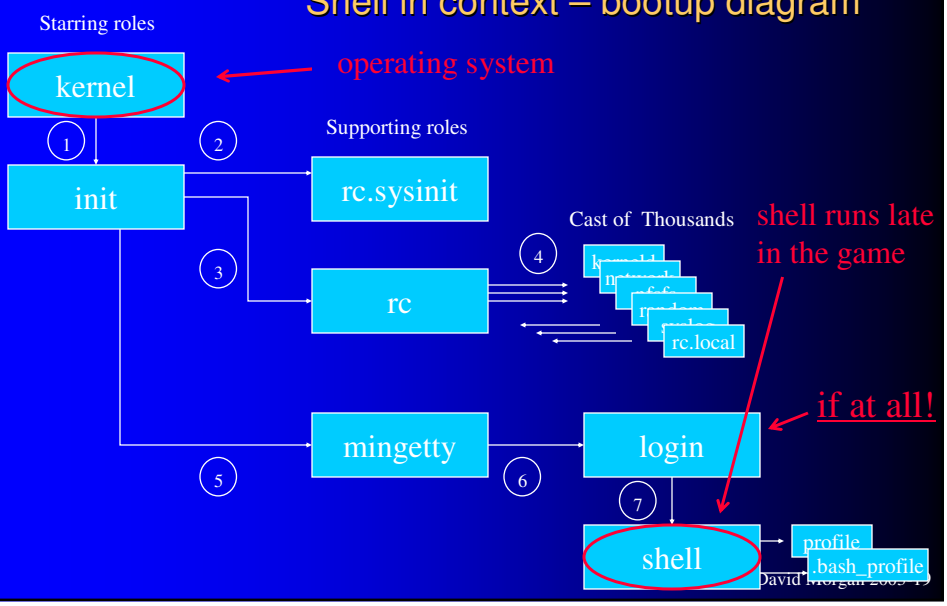


# *Not* the operating system



# *Not* necessary

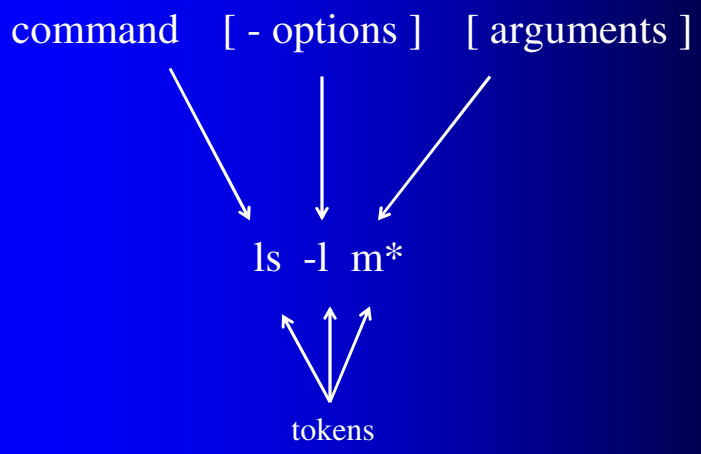
## Shell in context – bootup diagram




# What does the shell do?

- Command processing
  - parse
  - expand
  - execute
- I/O redirection
- Piping
- Environment control
- Background processing
- Shell scripts

## Command-line format






## Command processing

- Reads what you type at the prompt
- Parses it (command/options/args token analysis)
- Optionally expands (transforms) it
- Responds by
  - locating/executing program file, or
  - emitting error message

## Command-line expansion

- transformation of command line's args
- performed by shell on each argument
- per special characters in arg (if any)
- altered result gets executed
- several transformation types



## Types of transformation

- brace expansion
- tilde expansion
- parameter expansion
- variable expansion
- command substitution
- arithmetic expansion
- word splitting
- filename (pathname) expansion

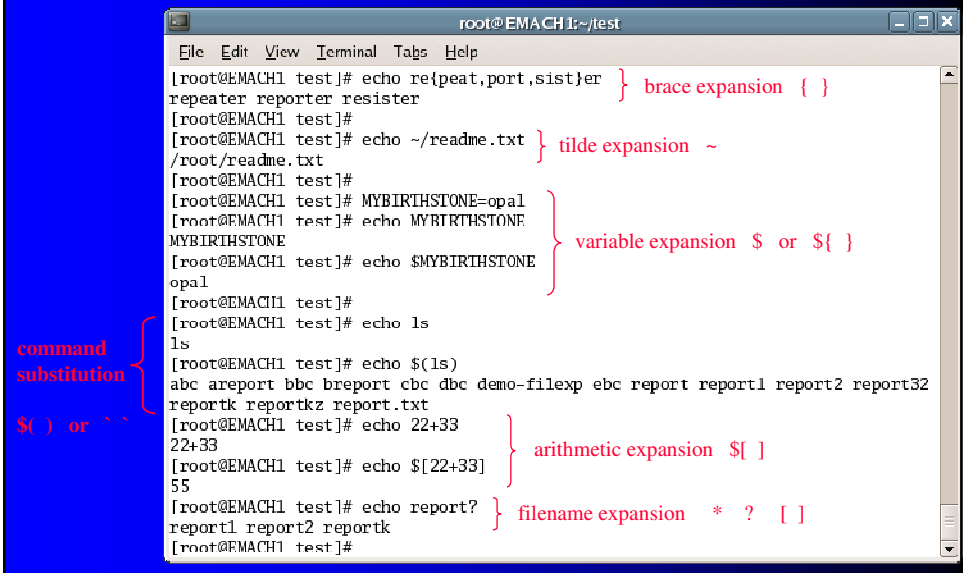
  – successively, in above order

# Transformation triggers

- caused by special characters (metacharacters)
- & ; | * ? ‘ “ ` [ ] ( ) $ < > { } ^ # / \ % ~
- shell has extraordinary responses to them
- ordinary characters are nouns, embedded special characters are adverbs– “how to” treat the surrounding ordinary characters



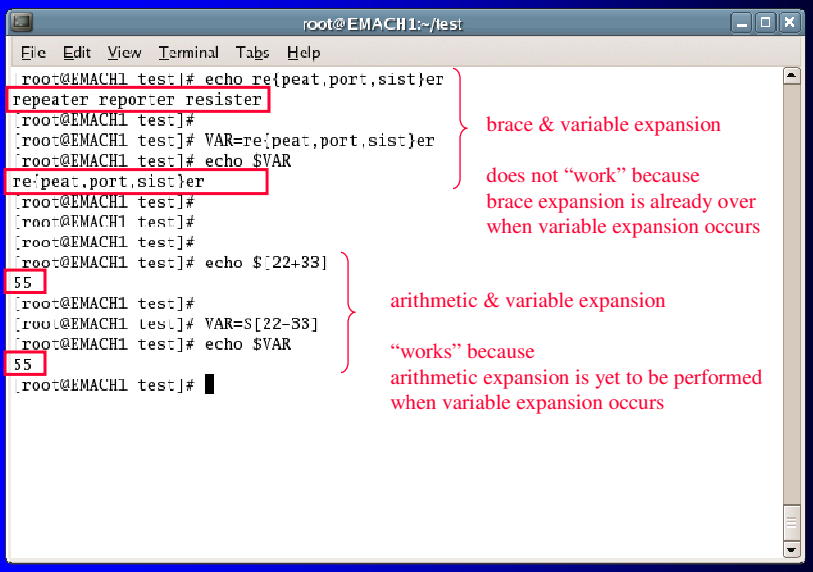
# Examples: various transformations



# Expansion order matters

different

same




brace & variable expansion

does not "work" because
brace expansion is already over
when variable expansion occurs

arithmetic & variable expansion

"works" because
arithmetic expansion is yet to be performed
when variable expansion occurs

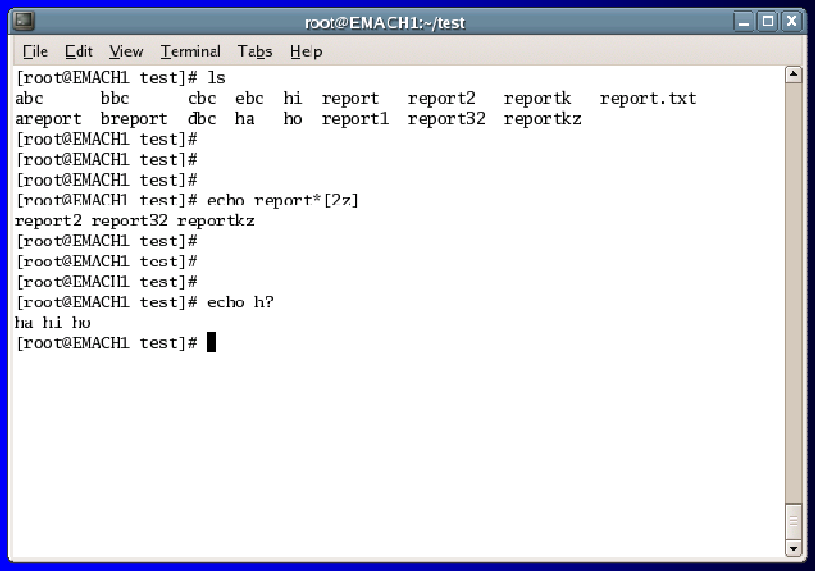


# Filename expansion metachars

- * any character(s) even none! echo *.*
- ? any single character cat lab?
- […] a single one of … ls lab[a-m]

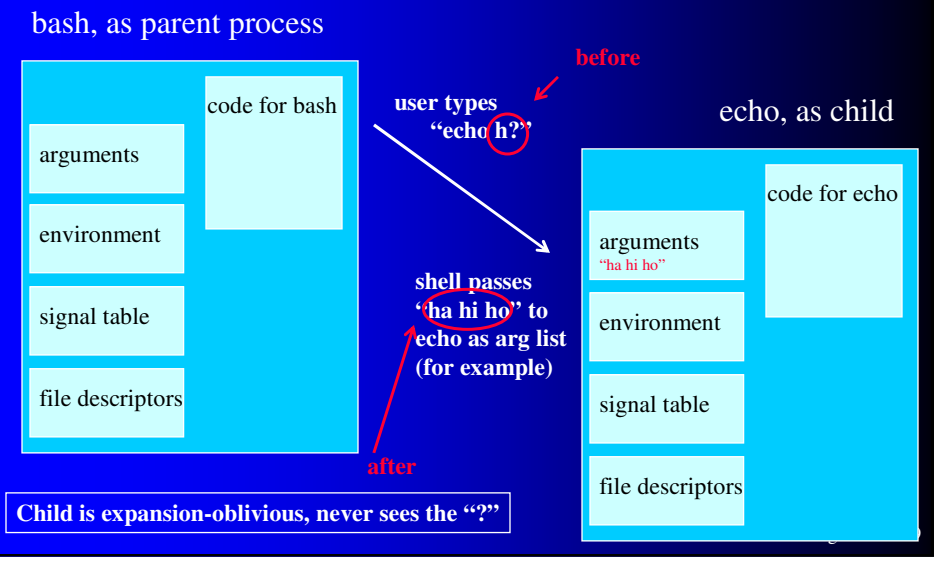
see man 7 glob

# Examples: filename expansion






# shell vs launched (child) process: command-line before vs after

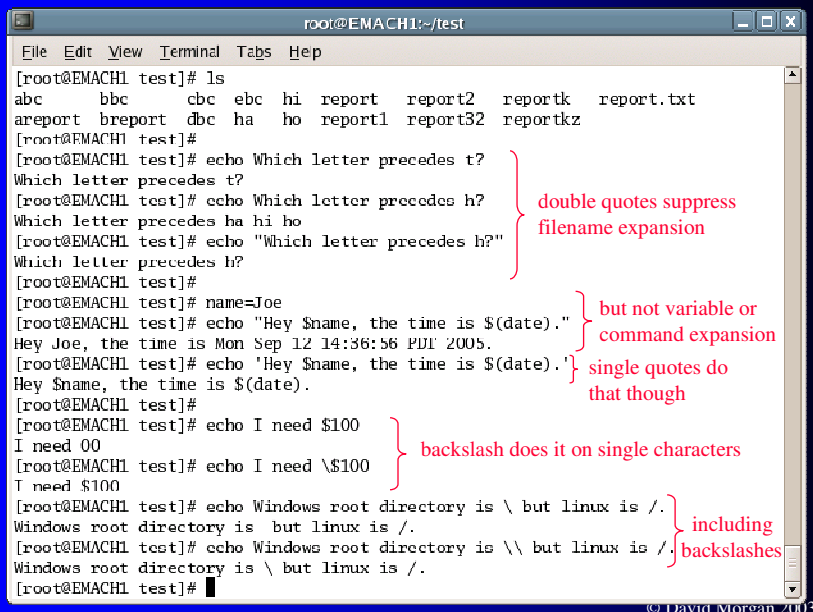


# Suppressing expansion

- prevents metacharacters' special interpretation
- single quotes
  - defeat all enclosed metacharacters
- backslash
  - defeats the metacharacter that follows it
- double quotes
  - defeat enclosed metacharacters except for variable ( $ ) and command ( $(  ) ) expansion



# Examples: expansion suppression

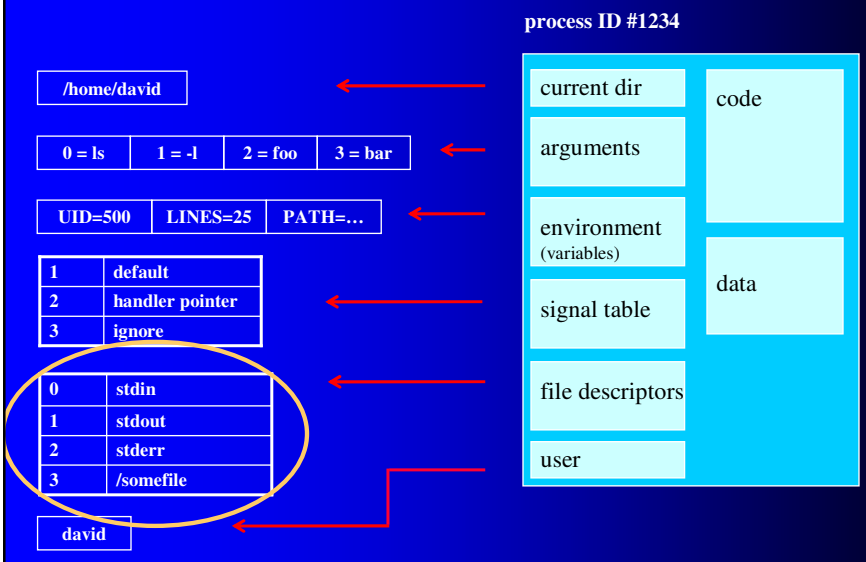


# I/O redirection

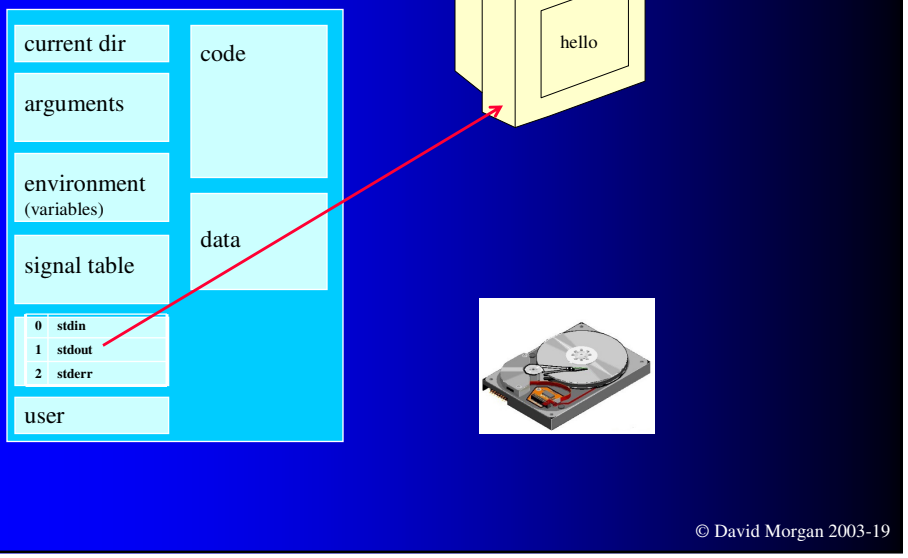
- cat lab? >> assignments
- mail dmorgan1 < prepared-message
- ls –z  2>  errorlog.txt



# I/O descriptors in a unix process

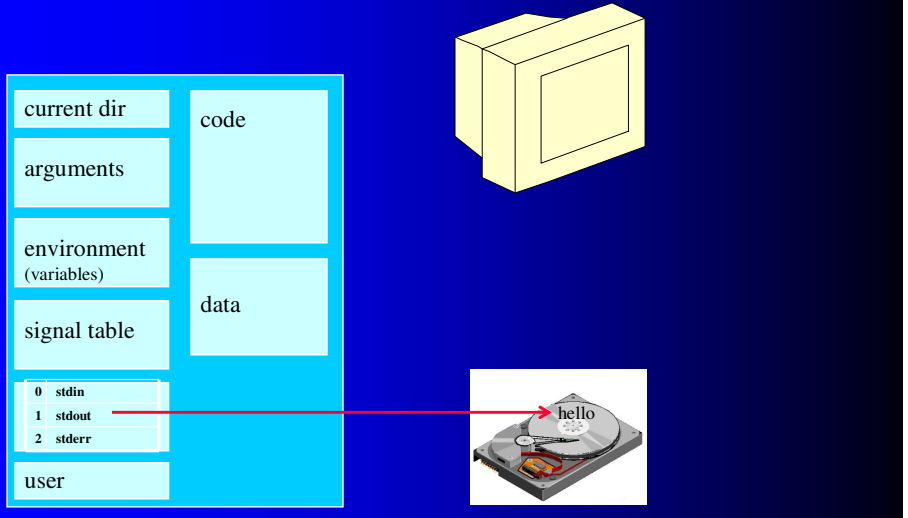


# echo “hello”






# echo “hello” > file



# myecho.c (an echo command do-alike)

```
[root@instructor ~]# > file
[root@instructor ~]# cat file
[root@instructor ~]#
[root@instructor ~]# cat myecho.c

main (int argc, char *argv[])
{
write(1,        argv[1],        strlen(argv[1]) );
write(1,        "\n",           1               );

}

[root@instructor ~]# ./myecho "roses are red"
roses are red
[root@instructor ~]# ./myecho "roses are red" > file
[root@instructor ~]#
[root@instructor ~]# cat file
roses are red
[root@instructor ~]#
```

empty file

a program that writes stuff to "1"

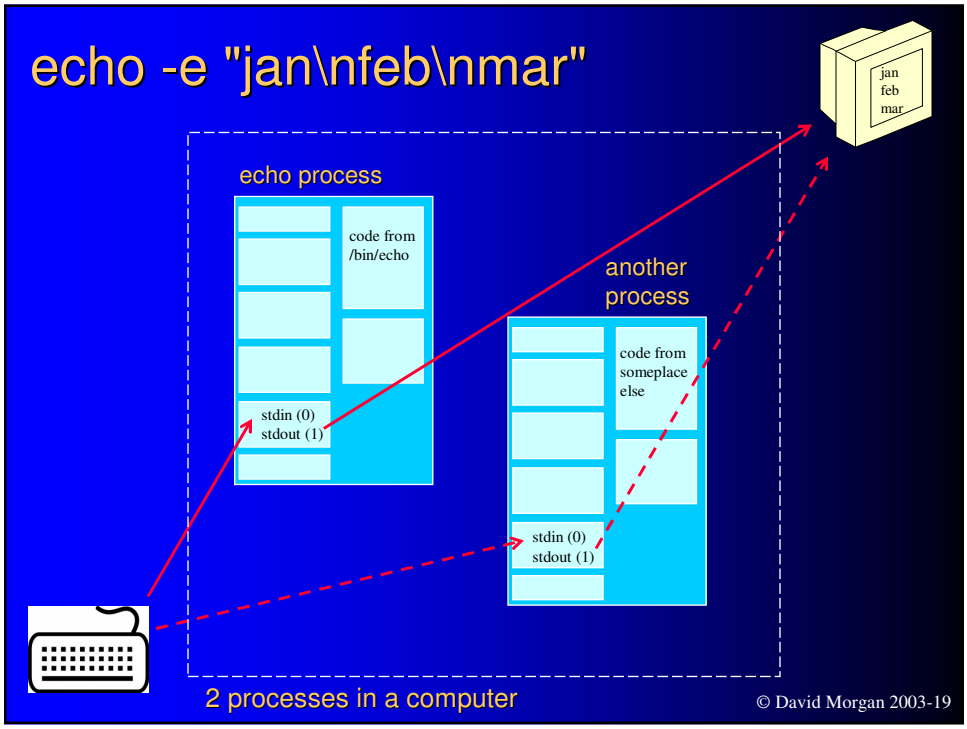
stuff program wrote is on screen; program does not write to "screen"

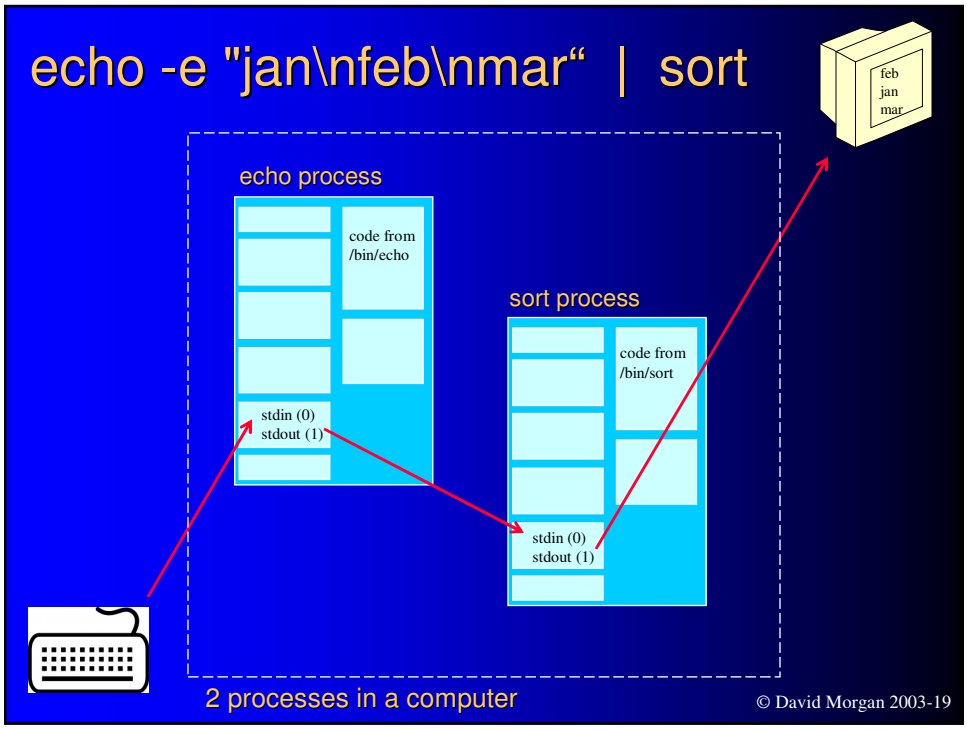
stuff program wrote is in file; program does not write to "file"



# Piping

- who | sort +4
- ls –l | grep ^d


echo -e "jan\nfeb\nmar"
jan
feb
mar
echo process
code from
/bin/echo
stdin (0)
stdout (1)
another
process
code from
someplace
else
stdin (0)
stdout (1)
2 processes in a computer
© David Morgan 2003-19


echo -e "jan\nfeb\nmar“ | sort
feb
jan
mar
echo process
code from
/bin/echo
stdin (0)
stdout (1)
sort process
code from
/bin/sort
stdin (0)
stdout (1)
2 processes in a computer
© David Morgan 2003-19

## Environment control

- variables
  - environment vars (get passed to child processes)
  - local vars (don't get passed)
- setting variables

  declare DAY=Monday , or

  DAY=Monday
- displaying variables
  - set or declare will display all variables
  - env or printenv will display environment vars only
  - echo $DAY to display value of single var



## Shell scripts

- Files containing a list of commands
- Permissions set to executable
- May be run as programs

# Information

- $man bash  (man page for bash)
- Unix Shells by Example, Ellie Quigley
- A Practical Guide to Red Hat Linux, Mark Sobell, Chapter 9, section “Command Line Expansion”