CS35L – Fall 2018

Slide set:	2.1
Slide topics:	Shell scripting, regex, streams
Assignment:	2

Environment Variables

- Variables that can be accessed from any child process

Common ones:

- **HOME**: path to user's home directory
- **PATH**: list of directories to search in for command to execute

- Change value:

  export VARIABLE=…

Locale

A locale

- Set of parameters that define a user’s cultural preferences
  - Language
  - Country
  - Other area-specific things

`locale` command

prints information about the current locale environment to standard output

LC_* Environment Variables

`locale` gets its data from the LC_* environment variables

<u>Examples:</u>

- LC_TIME

  Date and time formats

- LC_NUMERIC

  Non-monetary numeric formats

- LC_COLLATE

  Order for comparing and sorting

The 'C' Locale

- The default locale
- An environment of “least surprise”
- Behaves like Unix systems before locales

Locale Settings Can Affect Program Behavior!!

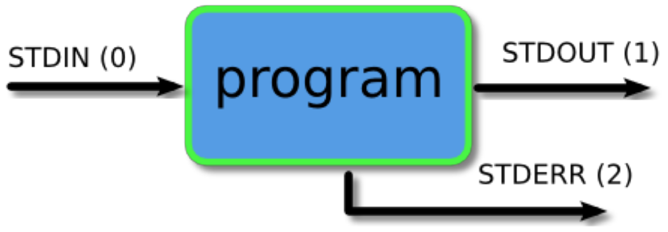
Default sort order for the `sort` command depends:

- LC_COLLATE=‘C’: sorting is in ASCII order
- LC_COLLATE=‘en_US’: sorting is case insensitive except when the two strings are otherwise equal and one has an uppercase letter earlier than the other.

Other locales have other sort orders!

Piping and Redirection

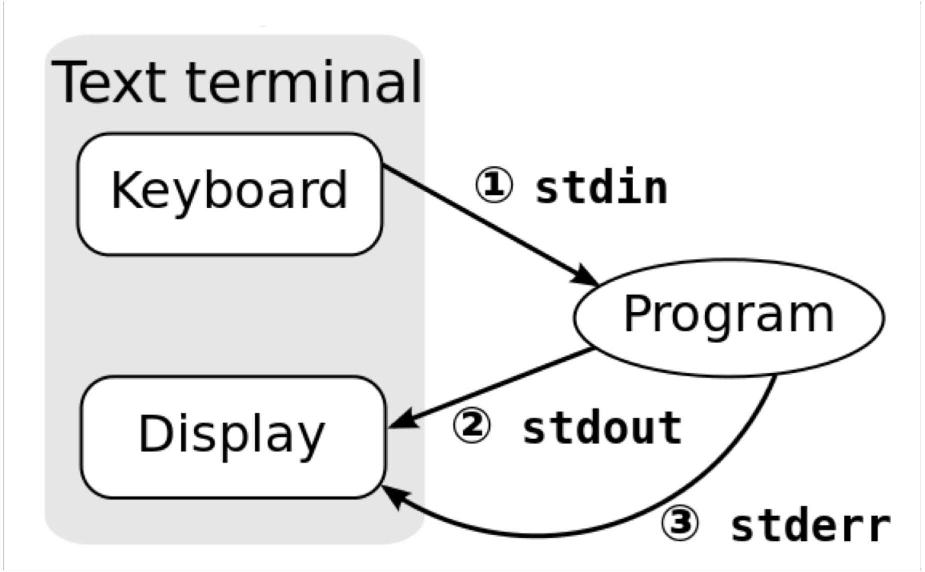
- Every program we run on the command line automatically has three data streams connected to it.
  - STDIN (0) - Standard input (data fed into the program)
  - STDOUT (1) - Standard output (data printed by the program, defaults to the terminal)
  - STDERR (2) - Standard error (for error messages, also defaults to the terminal)



Piping and redirection is the means by which we may connect these streams between programs and files to direct data in useful ways.

Standard Streams

- Every program has these 3 streams to interact with the world
  - stdin (0): contains data going into a program
  - stdout (1): where a program writes its output data
  - stderr (2): where a program writes its error msgs



Redirection and Pipelines

- *program < file* redirects *file to programs's stdin*:
  ```
  cat <file
  ```
- *program > file* redirects *program*'s stdout to *file2*:
  ```
  cat <file >file2
  ```
- *program 2> file* redirects *program*'s stderr to *file2*:
  ```
  cat <file 2>file2
  ```
- *program >> file* **appends** program's stdout to *file*
- *program1 | program2* assigns stdout of *program1* as the stdin of *program2; text 'flows' through the pipeline*
  ```
  cat <file | sort >file2
  ```

Pipe

- It lets you feed the output from the program on the left as input to the program on the right.
- Example –
  - **ls | head -3**

    ```
    barry.txt
    bob
    example.png
    ```
  - **ls | head -3 | tail -1**

    ```
    example.png
    ```

`sort`, `comm`, and `tr`

`sort`: sorts **lines** of **text** files

- Usage: sort [OPTION]…[FILE]…
- Sort order depends on locale
- C locale: ASCII sorting

`comm`: compare two **sorted** files **line by line**

- Usage: comm [OPTION]…FILE1 FILE2
- Comparison depends on locale

`tr`: translate **or** delete characters

- Usage: tr [OPTION]…SET1 [SET2]
- Ex: echo "12345" | tr "12" "ab"

Shell Scripting

The Shell and OS

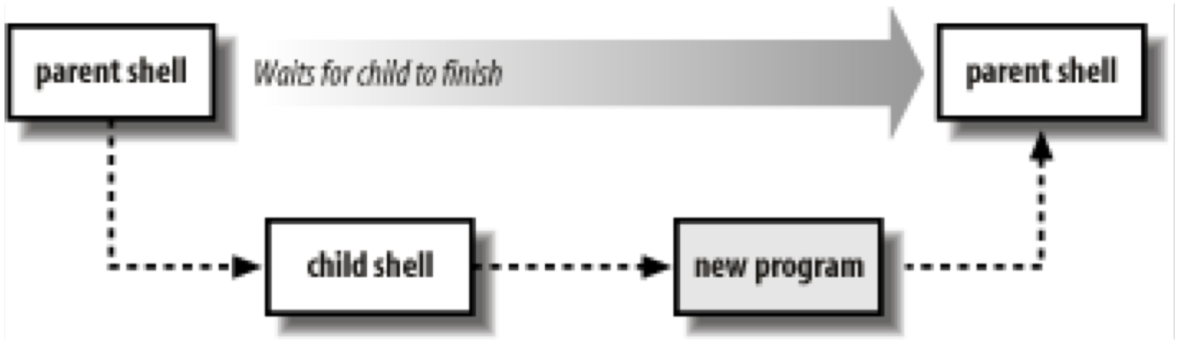
- The shell is a user interface to the OS
- Accepts commands as text, interprets them, uses OS API to carry out what the user wants – open files, start programs...

- Common shells
  - bash, sh, csh, ksh

Scripts: First Line

- A shell script file is just a file with shell commands
- When shell script is executed a new child “shell” process is spawned to run it
- The first line is used to state which child “shell” to use

```
#! /bin/sh
#! /bin/bash
```



Example

- A lab directory for each lab

Before each lab:

- Remove old directory called “lab”
- Create new directory called “lab”
- Create 3 files in “lab”
  - lab.log
  - lab.txt
  - hw.txt

```
rm –rf lab

mkdir lab

touch lab/lab.log
touch lab/lab.txt
touch lab/hw.txt
```

Execute shell scripts

```
$ touch script.sh
$ ./script.sh
-bash: ./script.sh: Permission denied
$ ls –al
-rw-r--r--   1 user1 csgrad      0 Apr  6 11:19 script.sh
$ chmod +x script.sh
$ ./script.sh
```

Simple Execution Tracing

- Shell prints out each command as it is executed

- Execution tracing within a script:

  `set -x`: to turn it on

  `set +x`: to turn it off

Output Using `echo` or `printf`

`echo` writes arguments to stdout, can't output escape characters (without –e)

```
$ echo “Hello\nworld”
Hello\nworld
$ echo –e “Hello\nworld”
Hello
world
```

`printf` can output data with complex formatting, just like C printf()

```
$ printf "%.3e\n" 46553132.14562253

4.655e+07
```