# Unix

https://harvard-iacs.github.io/2019-CS207/lectures/lecture1/

David Sondak

Harvard University
Institute for Applied Computational Science

9/5/2019

# Last Time

- Course introduction
- Unix and Linux

# Today

- More on Unix / Linux
- Practice time

Again, some content adapted from Dr. Chris Simmons.

# Unix Commands

# Basic Commands

## UNIX / LINUX CHEAT SHEET



### FILE SYSTEM

- `ls` — list items in current directory
- `ls -l` — list items in current directory and show in long format to see perimissions, size, and modification date
- `ls -a` — list all items in current directory, including hidden files
- `ls -F` — list all items in current directory and show directories with a slash and executables with a star
- `ls dir` — list all items in directory dir
- `cd dir` — change directory to dir
- `cd ..` — go up one directory
- `cd /` — go to the root directory
- `cd ~` — go to to your home directory
- `cd -` — go to the last directory you were just in
- `pwd` — show present working directory
- `mkdir dir` — make directory dir
- `rm file` — remove file
- `rm -r dir` — remove directory dir recursively
- `cp file1 file2` — copy file1 to file2
- `cp -r dir1 dir2` — copy directory dir1 to dir2 recursively
- `mv file1 file2` — move (rename) file1 to file2
- `ln -s file link` — create symbolic link to file
- `touch file` — create or update file
- `cat file` — output the contents of file
- `less file` — view file with page navigation
- `head file` — output the first 10 lines of file
- `tail file` — output the last 10 lines of file
- `tail -f file` — output the contents of file as it grows, starting with the last 10 lines
- `vim file` — edit file
- `alias name 'command'` — create an alias for a command

### SYSTEM

- `shutdown` — shut down machine
- `reboot` — restart machine
- `date` — show the current date and time
- `whoami` — who you are logged in as
- `finger user` — display information about user
- `man command` — show the manual for command
- `df` — show disk usage
- `du` — show directory space usage
- `free` — show memory and swap usage
- `whereis app` — show possible locations of app
- `which app` — show which app will be run by default

### COMPRESSION

- `tar cf file.tar files` — create a tar named file.tar containing files
- `tar xf file.tar` — extract the files from file.tar
- `tar czf file.tar.gz files` — create a tar with Gzip compression
- `tar xzf file.tar.gz` — extract a tar using Gzip
- `gzip file` — compresses file and renames it to file.gz
- `gzip -d file.gz` — decompresses file.gz back to file

### PROCESS MANAGEMENT

- `ps` — display your currently active processes
- `top` — display all running processes
- `kill pid` — kill process id pid
- `kill -9 pid` — force kill process id pid

### SEARCHING

- `grep pattern files` — search for pattern in files
- `grep -r pattern dir` — search recursively for pattern in dir
- `grep -rn pattern dir` — search recursively for pattern in dir and show the line number found
- `grep -r pattern dir --include='*.ext` — search recursively for pattern in dir and only search in files with .ext extension
- `command | grep pattern` — search for pattern in the output of command
- `find file` — find all instances of file in real system
- `locate file` — find all instances of file using indexed database built from the updatedb command. Much faster than find
- `sed -i 's/day/night/g' file` — find all occurrences of day in a file and replace them with night - s means substitude and g means global - sed also supports regular expressions

### PERMISSIONS

- `ls -l` — list items in current directory and show permissions
- `chmod ugo file` — change permissions of file to ugo - u is the user's permissions, g is the group's permissions, and o is everyone else's permissions. The values of u, g, and o can be any number between 0 and 7.
- `7` — full permissions
- `6` — read and write only
- `5` — read and execute only
- `4` — read only
- `3` — write and execute only
- `2` — write only
- `1` — execute only
- `0` — no permissions
- `chmod 600 file` — you can read and write - good for files
- `chmod 700 file` — you can read, write, and execute - good for scripts
- `chmod 644 file` — you can read and write, and everyone else can only read - good for web pages
- `chmod 755 file` — you can read, write, and execute, and everyone else can read and execute - good for programs that you want to share

### NETWORKING

- `wget file` — download a file
- `curl file` — download a file
- `scp user@host:file dir` — secure copy a file from remote server to the dir directory on your machine
- `scp file user@host:dir` — secure copy a file from your machine to the dir directory on a remote server
- `scp -r user@host:dir dir` — secure copy the directory dir from remote server to the directory dir on your machine
- `ssh user@host` — connect to host as user
- `ssh -p port user@host` — connect to host on port as user
- `ssh-copy-id user@host` — add your key to host for user to enable a keyed or passwordless login
- `ping host` — ping host and output results
- `whois domain` — get information for domain
- `dig domain` — get DNS information for domain
- `dig -x host` — reverse lookup host
- `lsof -i tcp:1337` — list all processes running on port 1337

### SHORTCUTS

- `ctrl+a` — move cursor to beginning of line
- `ctrl+f` — move cursor to end of line
- `alt+f` — move cursor forward 1 word
- `alt+b` — move cursor backward 1 word

http://cheatsheetworld.com/programming/unix-linux-cheat-sheet/

# Absolutely Essential Commands

These commands should be at your fingertips at all times:

- `ls` — list items in current directory
- `ls -l` — list items in current directory and show in long format to see perimissions, size, and modification date
- `ls -a` — list all items in current directory, including hidden files
- `ls -F` — list all items in current directory and show directories with a slash and executables with a star
- `ls dir` — list all items in directory dir
- `cd dir` — change directory to dir
- `cd ..` — go up one directory
- `cd /` — go to the root directory
- `cd ~` — go to to your home directory
- `cd -` — go to the last directory you were just in
- `pwd` — show present working directory
- `mkdir dir` — make directory dir
- `rm file` — remove file
- `rm -r dir` — remove directory dir recursively
- `cp file1 file2` — copy file1 to file2
- `cp -r dir1 dir2` — copy directory dir1 to dir2 recursively
- `mv file1 file2` — move (rename) file1 to file2
- `ln -s file link` — create symbolic link to file
- `touch file` — create or update file
- `cat file` — output the contents of file
- `less file` — view file with page navigation
- `head file` — output the first 10 lines of file
- `tail file` — output the last 10 lines of file
- `tail -f file` — output the contents of file as it grows, starting with the last 10 lines
- `vim file` — edit file
- `alias name 'command'` — create an alias for a command

# The `ls` command

- The `ls` command displays the names of files
- Giving it the name of a directory will list all files in that directory
- `ls` commands:
  - `ls` — list files in current directory
  - `ls /` — list files in the root directory
  - `ls .` — list files in the current directory
  - `ls ..` — list files in the parent directory
  - `ls /usr` — list files in the `/usr` directory

# Command Line Options

- Modify output format of `ls` with *command line options*
- There are many options for the `ls` command, e.g.
  - `-l` — *long* format
  - `-a` — *all*; shows hidden files as well as regular files
  - `-F` — include special character to indicate file types

Note: Hidden files have names that start with `.`

```
-rw-r--r--  1 dsondak staff  1687 Jul  2 09:56 .gitignore
```

# `ls` Command Line Options

- How to use the command line options:
  - `ls -a`, `ls -l`, ...
- Two or more options can be used at the same time!
  - `ls -ltra`

# General `ls` Command Line

- The general form is
  - `ls [options] [names]`
  - Note: Options must come first
  - You can mix any options with any names
  - Example:

    `ls -al /usr/bin`
- The brackets around options and names means that something is optional
- You will see this kind of description often in the Unix commands documentation
- Some commands have required parameters
- You can also use variable argument lists
  - `ls /usr /etc`
  - `ls -l /usr/bin /tmp /etc`
  - This will display many files or directory names

# `man` and More Information

- **man pages** (manual pages) provide extensive documentation
- The Unix command to display a manual page is `man`
- Man pages are split into 8 numbered sections
  1. General commands
  2. System calls
  3. `C` library functions
  4. Special files (usually devices found in `/dev`
  5. File formats and convections
  6. Games
  7. Miscellaneous
  8. Sys admin commands and daemons
- You can request pages from specific sections, e.g.

`man 3 printf` (shows manpage for `C` library function)

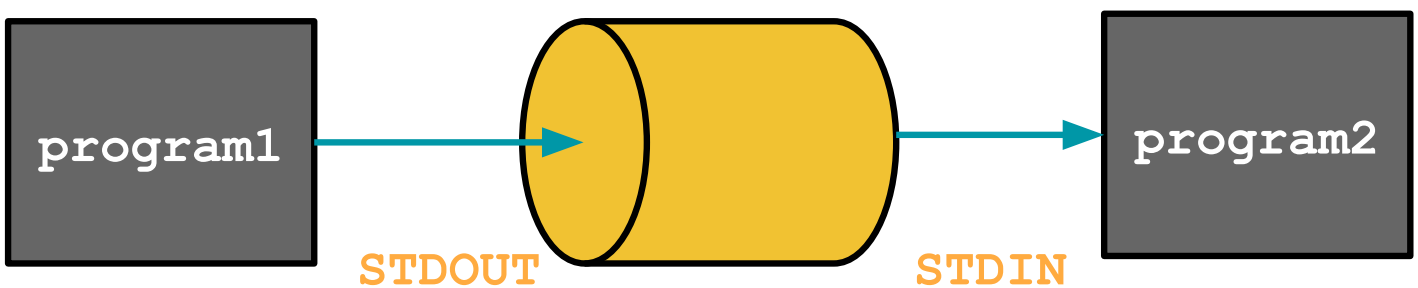
# Interacting with the Shell

# Running a Unix Program

- Type in the name of a program and some command line options
- The shell reads this line, finds the program, and runs it feeding it the options you specified
- The shell establishes 3 I/O streams:
  1. Standard input
  2. Standard output
  3. Standard error
- File descriptors associated with each stream:
  - 0 = STDIN
  - 1 = STDOUT
  - 2 = STDERR

# Unix Pipes

- A pipe is a holder for a stream of data
- A Unix **pipeline** is a set of processes chained by their standard streams
  - The output of each process (`stdout`) feeds directly as input (`stdin`) to the next one
- Very useful for using multiple Unix commands together to perform a task



# Building Commands

- More complicated commands can be built up by using one or more pipes
- The | character is used to *pipe* two commands together
- The shell does the rest for you!

```
dsondak:~/Teaching/Harvard/CS207/2019-CS207/content/lectures/lecture1
$ cat readings.md
Title: Lecture 1
Category: lectures
Date: 2019-09-03
Slug: lecture1
Author: David Sondak


# Lecture 0

* Unix and Linux
* `git` bash

### Introduction Slides
- [Lecture 1 Slides]({attach}presentation/lecture2.pdf)

### Lecture Notebook Slides
- [Lecture 1 Notebook]({filename}notebook/lecture2.ipynb)
```

```
dsondak:~/Teaching/Harvard/CS207/2019-CS207/content/lectures/lecture1
$ cat readings.md | wc
      17      37     302
```

- Note: `wc` prints the number of newlines, words, and bytes in a file.

# More Unix Commands: `find`

- `find` searches the filesystem for files whose name matches a specific pattern
- It can do much more than this and is one of the most useful commands in Unix
  - e.g. it can find files and then perform operations on them
- Example:

```
dsondak:~/Teaching/Harvard/CS207/2019-CS207/content/lectures/lecture1
$ ls
data  fig  notebook  notes  presentation  readings.md
dsondak:~/Teaching/Harvard/CS207/2019-CS207/content/lectures/lecture1
$ find . -name presentation -print
./presentation
```

# find

- `find` can also scan for certain file types:
  - Find directories with `find . -type d -print`
  - Find files with `find . -type f -print`
- The `exec` option can be used to make very powerful commands on files
  - `find . -type f -exec wc -l {} \;`
- What does this command do?

# The Famous grep

- grep extracts lines from a file that match a given string or pattern

```
dsondak:~/Teaching/Harvard/CS207/2019-CS207/content/lectures/lecture1
$ grep -r "grep" presentation/
Binary file presentation//.lecture2.tex.swp matches
presentation//lecture2.tex:  \begin{frame}{The Famous \texttt{grep}}
presentation//lecture2.tex:      \item \texttt{grep} extracts lines from a file that match a given string or patter \\[0.5em]
presentation//lecture2.tex:      \item \texttt{grep} can also use a regular expression for the pattern search
presentation//lecture2.tex:      \includegraphics[width=0.9\textwidth]{grep_example.png}
```

- grep can also use a regular expression for the pattern search

# Regular Expressions

- `grep` isn't the only Unix command that supports regular expressions
  - `sed`
  - `awk`
  - `perl`
- General search pattern characters
  - Any character
  - "." matches any character except a newline
  - "*" matches zero or more occurrences of the single preceeding character
  - "+" matches one or more of the proceeding character
  - "?" matches zero or one of the proceeding character
- More special characters
  - "()" are used to quantify a sequence of characters
  - "|" functions as an OR operator
  - "{}" are used to indicate ranges in the number of occurrences

# More on Regular Expressions

- To match a special character, you should use the backslash "\"
  - e.g. to match a period do "\."
  - `a\.b` matches `a.b`
- A *character class* (a.k.a. character set) can be used to match *only one* out of several characters
- Place the characters you want to match between square brackets, `[ ]`
- A hyphen can be used to specify a range of characters
- A caret, ∧, after the opening square bracket will negate the class
  - The result is that the character class will match any character that is **not** in the character class
- Examples:
  - `[abc]` matches a single `a`, `b`, or `c`
  - `[0-9]` matches a single digit between `0` and `9`
  - `[∧A-Za-z]` matches a single character as long as it's not a letter

# Regular Expressions Continued

- Some shorthand character classes are available for convenience,
  - `\d` a digit, e.g. `[0-9]`
  - `\D` a non-digit, e.g. `[^0-9]`
  - `\w` a word character, matches letters and digits
  - `\W` a non-word character
  - `\s` a whitespace character
  - `\S` a non-whitespace character
- Some shorthand classes are available for matching boundaries,
  - `^` the beginning of a line
  - `$` the end of a line
  - `\b` a word boundary
  - `\B` a non-word boundary
- Some references:
  - RegexOne
  - Mastering Regular Expressions

# Regular Expression Examples and Practice

You are given a text file called `dogs.txt` that contains names, ages, and breeds of dogs. Use `grep` and regular expressions to accomplish the following:

1. Find all dogs named either *Sally* or *Joey*.
   - **Hint:** In addition to a regular expression, you may also find the `-E` option for `grep` useful
2. Find all dogs named *Joey*.
   - **Note:** There are two dogs named Joey, but one of them has been entered in all lowercase!
   - **Note:** The extended regex `grep` option (`-E`) is not needed here
3. Find all dogs that are 6 months old.
   - **Hint:** You may assume that dogs that are 6 months old have been entered as 0.5.

# File Attributes

Every file has a specific list of attributes:

- Access times
  - when the file was created
  - when the file was last changed
  - when the file was last read
- Size
- Owners
  - user (remember UID)
  - group (remember GID)
- Permissions

For example, time attributes access with `ls`,

- `ls -l` shows when the file was last changed
- `ls -lc` shows when the file was created
- `ls -lu` shows when the file was last accessed

# File Permissions

- Each file has a set of permissions that control who can access the file
- There are three different types of permissions:
  - read, abbreviated `r`
  - write, abbreviated `w`
  - execute, abbreviated `x`
- In Unix, there are permission levels associated with three types of people that might access a file:
  - owner (you)
  - group (a group of other users that you set up)
  - world (anyone else browsing around on the file system)

# File Permissions Display Format

```
-  rwx  rwx  rwx
   Owner Group Others
```

- The first entry specifies the type of file:
  - "-" is a plain file
  - "d" is a directory
  - "c" is a character device
  - "b" is a block device
  - "l" is a symbolic link

- Meaning for Files:
  - `r` - allowed to read
  - `w` - allowed to write
  - `x` - allowed to execute

- Meaning for Directories:
  - `r` - allowed to see the names of files
  - `w` - allowed to add and remove files
  - `x` - allowed to enter the directory

# Changing File Permissions

- The `chmod` command changes the permissions associated with a file or directory
- Basic syntax: `chmod <mode> <file>`
- The `<mode>` can be specified in two ways
  - Symbolic representation
  - Octal number
- It's up to you which method you use
- Multiple symbolic operations can be given, separated by commas

# Symbolic Representation

- Symbolic representation has the following form,
  - `[ugoa] [+-=] [rwxX]`
- `u=user`, `g=group`, `o=other`, `a=all`
- `+` — add permission, `-` — remove permission, `=` — set permission
- `r=read`, `w=write`, `x=execute`
- `X` — Sets to execute only if the file is a directory or already has execute permission
  - Very useful when using recursively

# Symbolic Representation Examples

```
dsondak:~/Teaching/Harvard/CS207/2019-CS207/content/lectures/lecture1
$ ls -al notes/
total 0
drwxr-xr-x 4 dsondak staff 128 Sep  3 18:37 .
drwxr-xr-x 8 dsondak staff 256 Sep  3 17:46 ..
-rw-r--r-- 1 dsondak staff   0 Sep  3 17:46 README.md
-rw-r--r-- 1 dsondak staff   0 Sep  3 18:37 foo
dsondak:~/Teaching/Harvard/CS207/2019-CS207/content/lectures/lecture1
$ chmod g=rw notes/foo
dsondak:~/Teaching/Harvard/CS207/2019-CS207/content/lectures/lecture1
$ ls -al notes
total 0
drwxr-xr-x 4 dsondak staff 128 Sep  3 18:37 .
drwxr-xr-x 8 dsondak staff 256 Sep  3 17:46 ..
-rw-r--r-- 1 dsondak staff   0 Sep  3 17:46 README.md
-rw-rw-r-- 1 dsondak staff   0 Sep  3 18:37 foo
dsondak:~/Teaching/Harvard/CS207/2019-CS207/content/lectures/lecture1
$ chmod u-w,g+x,o=x notes/foo
dsondak:~/Teaching/Harvard/CS207/2019-CS207/content/lectures/lecture1
$ ls -al notes
total 0
drwxr-xr-x 4 dsondak staff 128 Sep  3 18:37 .
drwxr-xr-x 8 dsondak staff 256 Sep  3 17:46 ..
-rw-r--r-- 1 dsondak staff   0 Sep  3 17:46 README.md
-r--rwx--x 1 dsondak staff   0 Sep  3 18:37 foo
dsondak:~/Teaching/Harvard/CS207/2019-CS207/content/lectures/lecture1
$
```

# Octal Representation

- Octal mode uses a single-argument string which describes the permissions for a file (3 digits)
- Each digit is a code for each of the three permission levels
- Permissions are set according to the following numbers:
  - `read=4`, `write=2`, `execute=1`
- Sum the individual permissions to get the desired combination

- 0 = no permission at all
- 1 = execute only
- 2 = write only
- 3 = write and execute (1+2)
- 4 = read only
- 5 = read and execute (4+1)
- 6 = read and write (4+2)
- 7 = read, write, and execute (4+2+1)

# Octal Representation Examples

```
dsondak:~/Teaching/Harvard/CS207/2019-CS207/content/lectures/lecture1
$ ls -al notes/
total 0
drwxr-xr-x 5 dsondak staff 160 Sep  3 18:47 .
drwxr-xr-x 8 dsondak staff 256 Sep  3 18:40 ..
-rw-r--r-- 1 dsondak staff   0 Sep  3 17:46 README.md
-rw-r--r-- 1 dsondak staff   0 Sep  3 18:47 bar
-r--rwx--x 1 dsondak staff   0 Sep  3 18:37 foo
dsondak:~/Teaching/Harvard/CS207/2019-CS207/content/lectures/lecture1
$ chmod 660 notes/bar
dsondak:~/Teaching/Harvard/CS207/2019-CS207/content/lectures/lecture1
$ ls -al notes
total 0
drwxr-xr-x 5 dsondak staff 160 Sep  3 18:47 .
drwxr-xr-x 8 dsondak staff 256 Sep  3 18:40 ..
-rw-r--r-- 1 dsondak staff   0 Sep  3 17:46 README.md
-rw-rw---- 1 dsondak staff   0 Sep  3 18:47 bar
-r--rwx--x 1 dsondak staff   0 Sep  3 18:37 foo
dsondak:~/Teaching/Harvard/CS207/2019-CS207/content/lectures/lecture1
$ chmod 417 notes/bar
dsondak:~/Teaching/Harvard/CS207/2019-CS207/content/lectures/lecture1
$ ls -al notes
total 0
drwxr-xr-x 5 dsondak staff 160 Sep  3 18:47 .
drwxr-xr-x 8 dsondak staff 256 Sep  3 18:40 ..
-rw-r--r-- 1 dsondak staff   0 Sep  3 17:46 README.md
-r----xrwx 1 dsondak staff   0 Sep  3 18:47 bar
-r--rwx--x 1 dsondak staff   0 Sep  3 18:37 foo
dsondak:~/Teaching/Harvard/CS207/2019-CS207/content/lectures/lecture1
```

# Text Editors and Shell Customization

# Text Editors

- For programming and changing of various text files, we need to make use of available Unix text editors
- The two most popular and available editors are vi and emacs
- You should familiarize yourself with at least one of the two
  - Editor Wars
- We will have very short introductions to each

# A Brief Text Editor History

- `ed` : line mode editor
- `ex` : extended version of ed
- `vi` : full screen version of ex
- `vim` : Vi IMproved
- `emacs` : another popular editor
- `ed/ex/vi` share lots of syntax, which also comes back in `sed/awk`: useful to know.

# `vi` Overview

- The big thing to remember about `vi` is that it has two different modes of operation:
  - Insert Mode
  - Command mode
- The insert mode puts anything typed on the keyboard into the current file
- The command mode allows the entry of commands to manipulate text
- Note that `vi` starts out in the command mode by default

# `vim` Quick Start Commands

- `vim <filename>`
- Press `i` to enable insert mode
- Type text (use arrow keys to move around)
- Press `Esc` to enable command mode
- Press `:w` (followed by `return`) to save the file
- Press `:q` (followed by `return`) to exit `vim`

# Useful vim Commands

- `:q!` - exit without saving the document. Very handy for beginners
- `:wq` - save and exit
- `/ <string>` - search within the document for text. `n` goes to next result
- `dd` - delete the current line
- `yy` - copy the current line
- `p` - paste the last cut/deleted line
- `:1` - goto first line in the file
- `:$` - goto last line in the file
- `$` - end of current line
- $\wedge$ - beginning of line
- `%` - show matching brace, bracket, parentheses

Here are some vim resources: https://vim.rtorr.com/, https://devhints.io/vim, https://vim-adventures.com/, `vimtutor`.

# Shell Customization

- Each shell supports some customization.
  - user prompt settings
  - environment variable settings
  - aliases
- The customization takes place in startup files which are read by the shell when it starts up
  - Global files are read first - these are provided by the system administrators (e.g. `/etc/profile`)
  - Local files are then read in the user's `HOME` directory to allow for additional customization

# Shell Startup Files

Useful information can be found at the `bash man` page:
https://linux.die.net/man/1/bash

- `~/.bash_profile`
  - Conventionally executed at login shells
  - Conventially only run once: at login
  - MacOS executes it for *every* new window
- `~/.bashrc`
  - Conventionally executed for each new window
  - Can contain similar information as the `.bash_profile`
- `~/.bash_login`
  - Relic of a bygone time; rarely (if ever) modify
- `~/.profile`
  - Executed after looking for `.bash_profile` and `.bashrc`; generally don't modify
- `~/.bash_logout`
  - Executed when the shell exits

Decent reference on the difference between `.bash_profile` and `.bashrc`:
Apple Stack Exchange, Scripting OS X

# Lecture Exercise

Update your `.bash_profile`

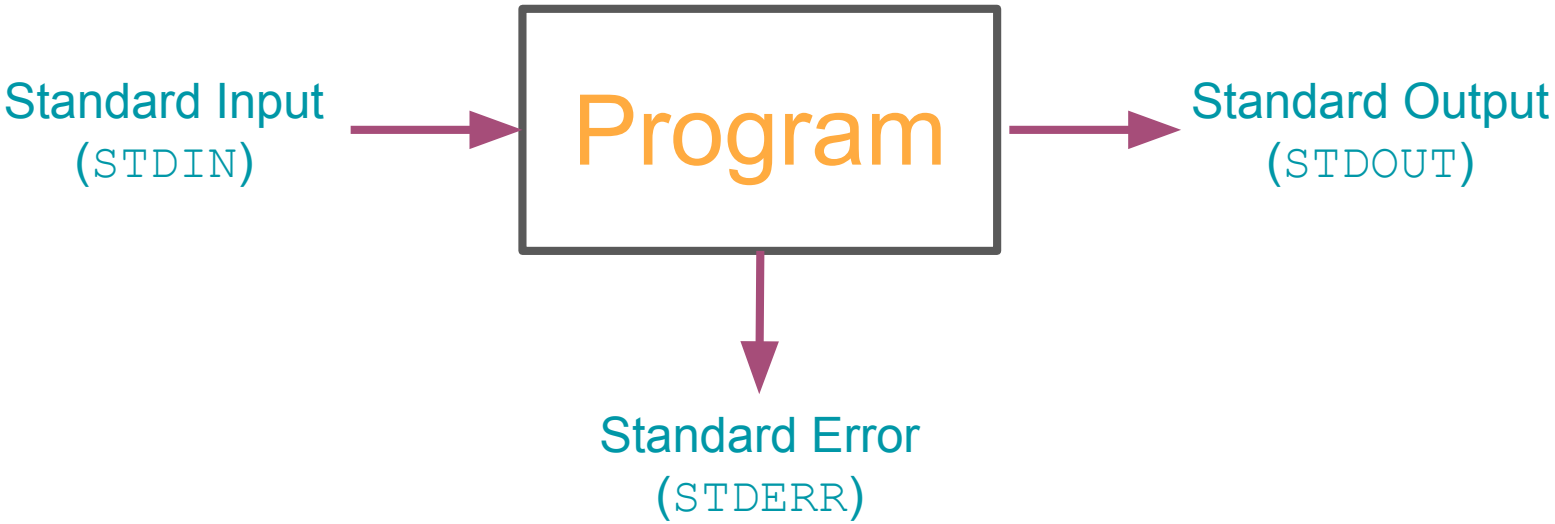
Exercise goals:

- Familiarize with a text editor (like `vim`)
- Create an alias for `ls` (e.g. `ll`) [see https://www.tecmint.com/create-alias-in-linux/]
- Change command line prompt format (see https://www.cyberciti.biz/tips/howto-linux-unix-bash-shell-setup-prompt.html)

Note to Windows users: Modify Bash Profile in Windows

Note: The Dracula Theme is pretty fun.

# I/O




- File descripters are associated with each stream,
  - `0=STDIN`, `1=STDOUT`, `2=STDERR`
- When a shell runs a program for you,
  - Standard input is the keyboard
  - Standard output is your screen
  - Standard error is your screen
- To end the input, press `Ctrl-D` on a line; this ends the input stream

# Shell Stream Redirection

- The shell can attach things other than the keyboard to standard input or output
  - e.g. a file or a pipe
- To tell the shell to store the output of your program in a file, use >,
  - `ls > ls_out`
- To tell the shell to get standard input from a file, use <,
  - `sort < nums`
- You can combine both forms together,
  - `sort < nums > sortednums`

# Modes of Output Redirection

- There are two modes of output redirection,
  - `>` — create mode
  - `>>` — append mode
- `ls > foo` creates a new file `foo`, possibly deleting any existing file named `foo` while `ls >> foo` appends the output to `foo`
- `>` only applies to `stdout` (not `stderr`)
- To redirect `stderr` to a file, you must specify the request directly
  - `2>` redirects `stderr` (e.g. `ls foo 2> err`)
  - `&>` redirects `stdout` and `stderr` (e.g. `ls foo &> /dev/null`)
  - `ls foo > out 2> err` redirects `stdout` to `out` and `stderr` to `err`

# Wildcards

- The shell treats some characters as special
- These special characters make it easy to specify filenames
- `*` matches anything
- Giving the shell `*` by itself removes `*` and replaces it with all the filenames in the current directory
- `echo` prints out whatever you give it (e.g. `echo hi` prints out `hi`)
- `echo *` prints out the entire working directory!
- `ls *.txt` lists all files that end with `.txt`