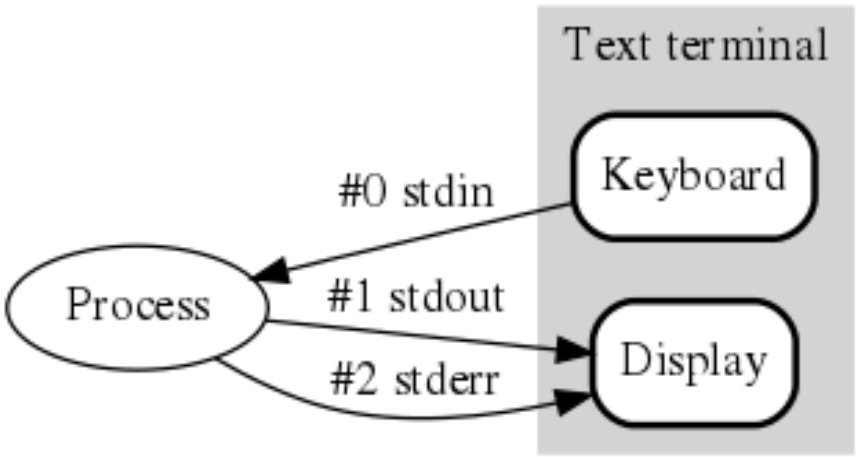
Lecture 02 - Standard Streams, grep, sed and find

CS 1XA3

Jan. 16, 2018

Standard Streams



There are three standard streams to be aware of

- stdin - standard input
- stdout - standard output
- stderr - error output

Standard Streams: Standard Output

Most bash commands output to stdout, but some particular functions for working with standard output are

- **echo**
  Prints argument to stdout
  Example:

```
echo "Hello World!"
```

- **cat**
  Outputs file to stdout
  Example:

```
cat file1.txt
```

Standard Streams: Standard Input

- **read**
  Reads user input and stores it in specified variable
  Example:

```
read name
echo $name
```

- Note: not very useful outside of a script

Standard Streams: IO Redirection StdOut

The following operators can be used to redirect stdout from any command to a file

- `>`
  Redirects stdout to create / **overwrite** a file
  Example:

```
ls -la > file1.txt # lists directory in file1.txt
> file2.txt  # Creates an empty file
```

- `>>`
  Redirects stdout to create / **append** a file
  Example:

```
ls -la >> file1.txt
```

Standard Streams: IO Redirection StdErr

The following operators can be used to redirect stderr from any command to a file

- 2 > and 2 >>
  Redirects stderr to create / **overwrite** a file and **append** a file respectively
  Example:

```
runhaskell Test.hs 2>> log.txt
```

- & >
  Redirects stdout and stderr to a file
  Example:

```
runhaskell Test.hs &> log.txt
```

Standard Streams: IO Redirection StdErr

Note: The previous example of combining stdout and stderr does not append the file (& >> is often not supported)

- 2 > &1
  Redirects stderr to stdout

- Use to **append** both stderr and stdout
  Example:

```
runhaskell Test.hs >> log.txt 2>&1
```

Standard Streams: Piping

The Pipe Operator | redirects stdout of one command into stdin of another command

Example:

```
cat file.txt | echo # useless pipe to echo

ps aux | grep ssh # finds all processes
                  # associated with ssh
```

Glob Patterns

- *(asterisk)
  Can represent any **sequence** of numbers and characters
- ?(question mark)
  Can represent any **single** character or number
- [](square brackets)
  Used to specify a range of strings
  Example:
  `sd[a,b,c] => sda sdb sdc`
- {}(curly braces)
  Term expansion. Like square brackets but can contain other *patterns*

Searching with grep

Syntax:

```
grep pattern input
```

- **grep** parses its input an returns lines that contain **pattern**
- **pattern** is any regular expression or just a string you're looking for
- you'll learn more about regular expressions later, they can get quite complicated, for now we'll just focus on wildcards or globbing patterns

Search and mutate with sed

Syntax:

```
sed -flag pattern input
```

- the only flag we'll concern ourselves with for now is **-i** for in-place
- to replace every instance of **old** to **new** in **file.txt**

  ```
  sed -i 's/old/new/g' file.txt
  ```
- to delete every **line** containing **new** in **file.txt**

  ```
  sed -i '/new/d' file.txt
  ```

Note: there are many, many more patterns you can use with **sed**, they also get quite complicated though. Often a quick google search will yield what you need though

Find Stuff with find

Syntax:

```
find startdir -name pattern -otherflag
```

From **startdir**, search all files and subdirectories for something matching **pattern**
Other flags include

- **type**
  Either **f** for file or **d** for directory Example:

  ```
  find . -name '*.tmp' -type f
  # search for all files with extension
  # tmp from the current directory
  ```

- **-exec**
  Allows you to execute a command on your findings

  ```
  find . -name '*.tmp' -type f -exec rm {} \;
  ```