ITP20004 – Open-Source Software Labs

More Linux Commands

Charmgil Hong

charmgil@handong.edu

Spring, 2023

Handong Global University

Announcements

- Weekly schedule

Week	Mon		Week	Thur		
1	Course overview, motivation, administrivia		1	CPR: C Programming Reinforcement - Functions		
2	Computer organization and Linux environment (1)		2	CPR: C Programming Reinforcement - Strings		
3	Computer organization and Linux environment (2)		3	CPR: C Programming Reinforcement - User-defined types, and memory allocation		
4	Basic Linux commands + Writing code on Linux (vim)		4	Getting started with Linux / Hands-on Linux command-line tools		
5	More Linux commands		5	CPR: C Programming Reinforcement - Understanding compilation and build process		
6	Project management (1)	**Proj 1 출제**	6	Project management (2)		
7	-		7	Project: BASIC interpreter (2 periods)		Project 1
8	Midterm exam		8	Proj 1 due		
9	CPR: C Programming Reinforcement - Accessing files and		9	Debugging with GDB + Unit testing with gtest		
10	Code review GNU utilities		12	Writing an application in C		
11	Computer network basics		10	Linux network commands	**AWS 가입 - lightsail**	
12	Linux machine as a server + Web services		11	Service launching	**lab problem + AWS 가입해지**	
13	Project: Text-based Game		13	Github and open-source community		Project 2
14	Using Github		14	Socket programming		
15	Project: Multi-user game		15	Project: Multi-user game		Project 3
16	Final exam		16			

Announcements

- Tutor session for the 2nd post-lab report
  - 9-10:30pm, Tuesday, March 28
  - NTH220

  - All team members are supposed to stop by to report the outcomes
    - Be prepared to answer questions regarding the lab tasks
    - If you have a conflict in schedule, please get the agreement of your teammate & let the TA (tutor) knows
      - TA: 박민지 (mean0068@handong.ac.kr)
    - The tutor will see all teams in turn. Please expect to spend around 5 minutes with the tutor

Announcements

- There will be two re-shuffles of the teams
  - On Weeks 6 and 12
  - There will be a peer evaluation at the end of each cycle

Announcements

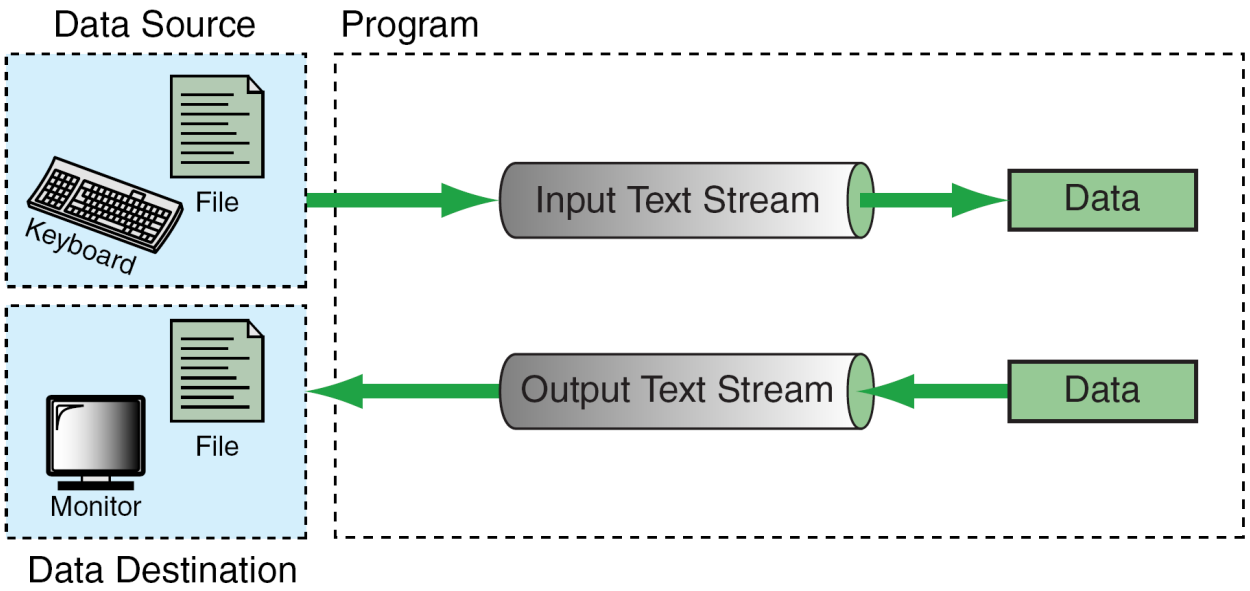
- For each lab
  - Before a lab, every student submits a pre-lab report (worksheet-type assignment) – **individual work**
  - After a lab, each team sees and reports to the TA with the results – **team work**

Agenda

- Streams, pipes, & redirection
- Regular expression in *vim*

Streams

- Streams: Inputs to and outputs from programs
  - Data is read and wrote through stream
  - A stream can be associated with terminal, file, and other data sources or destinations
    - Usually comes from the keyboard; goes to the screen

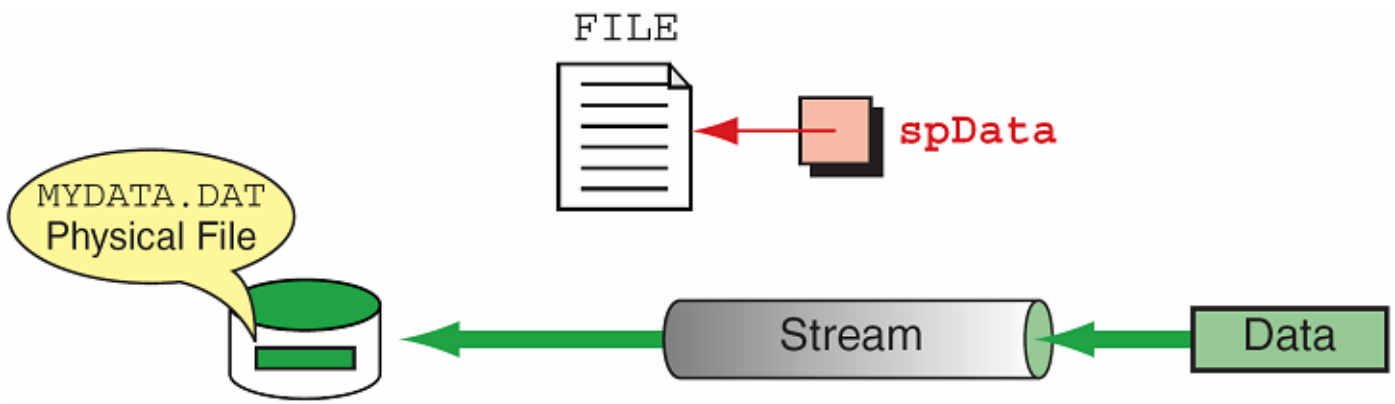


Streams

- File open: prepares a file for processing
  - Syntax: `FILE* fopen("filename", "mode");`
    - Filename: name of physical file
    - Mode: string to indicate how the file will be used
    - Return value: pointer to a stream (FILE*)
      - If it fails to open a file, return NULL.

E.g., `FILE* spData = fopen("MYFILE.DAT", "w");`

`FILE* spData = fopen("A:\\MYFILE.DAT", "w");`

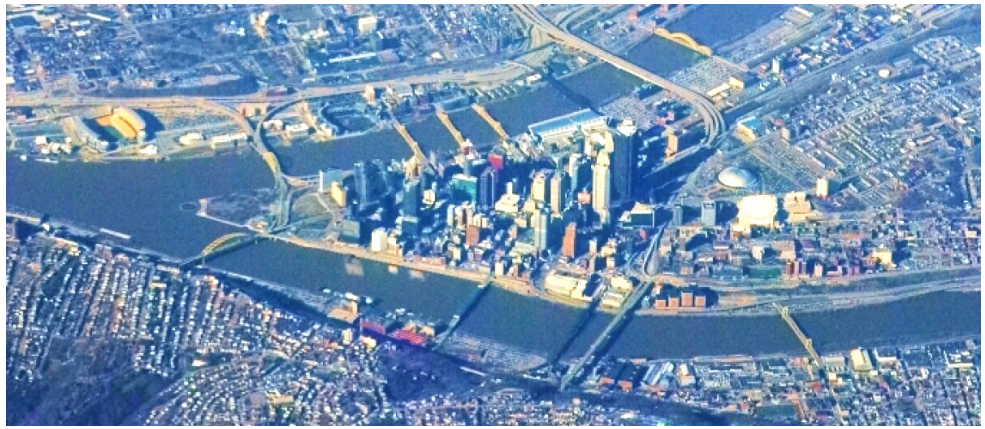


Streams

- Linux shells use three “standard” streams:
  - Standard input (`stdin`): usually the input from the keyboard
  - Standard output (`stdout`): displays the output from commands, usually to the terminal
  - Standard error (`stderr`): displays error output from commands
    - Usually sent to the same output as standard output
    - Can be redirected separately from `stdout`

Streams

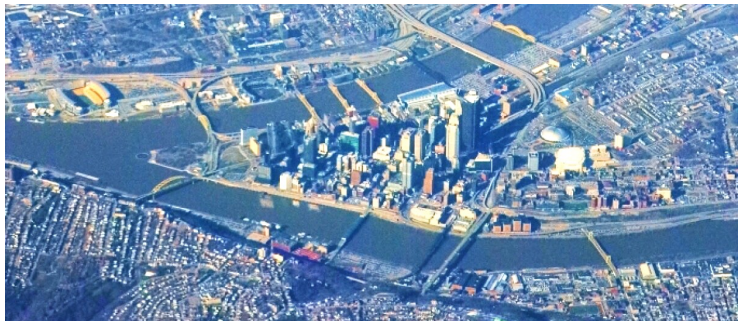
- Linux streams are like streams of water



* Image src: https://wikis.utexas.edu/display/CoreNGSTools/Linux+fundamentals;
https://www.wesa.fm/environment-energy/2013-02-04/pollutants-continue-to-hamper-wildlife-fishing-recreation-in-pittsburghs-three-rivers

Streams

- Linux streams are like streams of water
  - One can redirect streams
  - One can pipe to carry water from one place to another

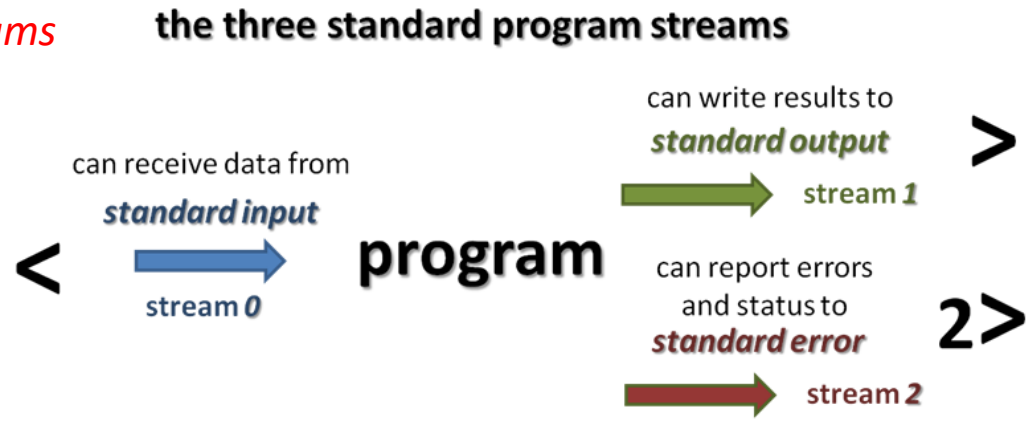


* Image src: https://wikis.utexas.edu/display/CoreNGSTools/Linux+fundamentals;
https://www.wesa.fm/environment-energy/2013-02-04/pollutants-continue-to-hamper-wildlife-fishing-recreation-in-pittsburghs-three-rivers

Streams

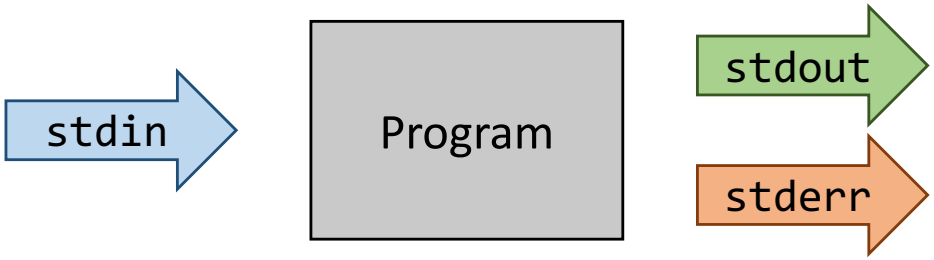
- Linux streams are like streams of water
  - One can redirect streams
  - One can pipe to carry water from one place to another
    - One can use UNIX pipes to carry data from one program to another
    - Streams can be used to pass data into programs and to get data out of them

We should have some ways of connecting programs like a garden hose — screw in another segment when it becomes necessary to massage data in another way. This is the way of I/O also.
— Douglas McIlroy



Streams

- In Linux (POSIX OS's), programs have the `stdin`, `stdout`, and `stderr` streams attached to them by default



Sidenote: POSIX

- POSIX: **P**ortable **O**perating **S**ystem **I**nterface (IEEE)
  - A family of standards, specified by the IEEE
  - To clarify and make uniform the application programming interfaces (API) and ancillary issues, such as command line shell utilities among Unix-like operating systems
    - C API
    - CLI (command-like interface) utilities
    - Shell language
    - Environment variables (`HOME`, `PATH`, …)
    - Program exit status
    - Regular expression
    - Directory structure
    - Filenames
    - Command line utility API conventions

Sidenote: POSIX

- POSIX: **P**ortable **O**perating **S**ystem **I**nterface (IEEE)
  - If you write your programs to rely on POSIX standards, you can easily port them among a large family of Unix derivatives
    - Apple OS X
    - IBM AIX
    - HP HP-UX
    - Oracle Solaris

    - Most Linux distros are very compliant,
      but not certified because they do not want to pay the compliance check

Streams

- Example: `fortune` and `cowsay`
  - `fortune` tells you some pieces of wisdom

```
$ fortune
Don't worry.  Life's too long.
-- Vincent Sardi, Jr.
```

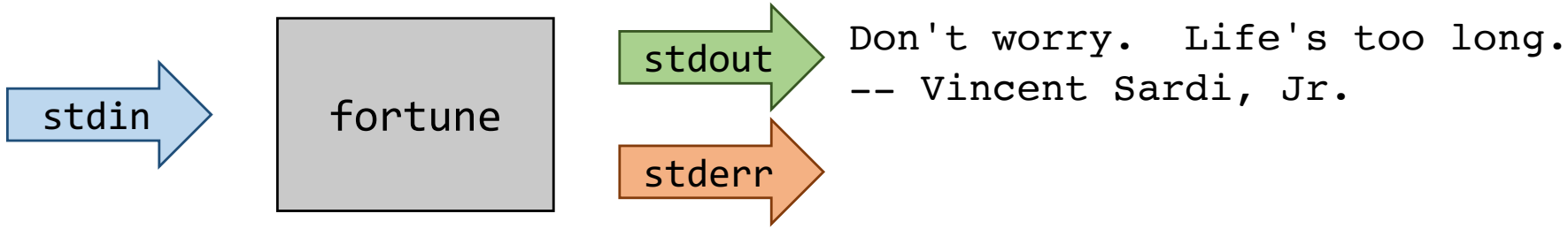
  - `cowsay` takes a string and shows a cow saying it

```
$ cowsay "OSSL rules!”

 _____________
< OSSL rules! >
 -------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

Streams

- When `fortune` runs, it does not produce any errors and does not get any external input; it just wrote its output to `stdout`

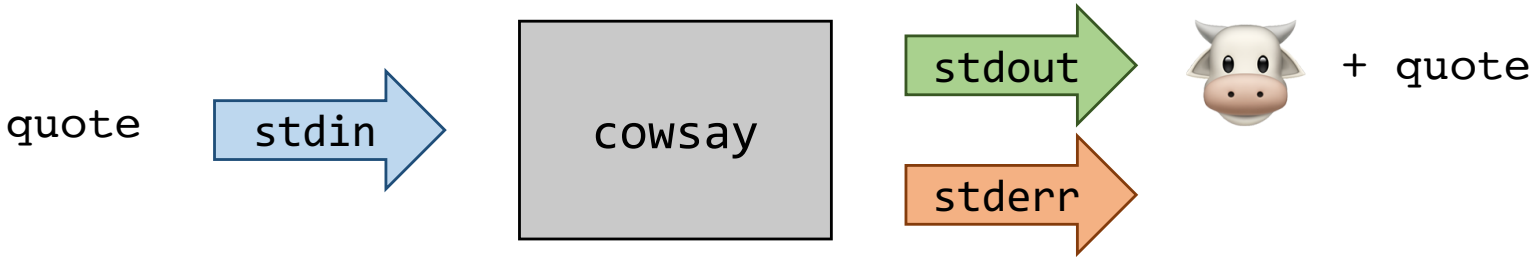
```
$ fortune
Don't worry.  Life's too long.
-- Vincent Sardi, Jr.
```



Streams

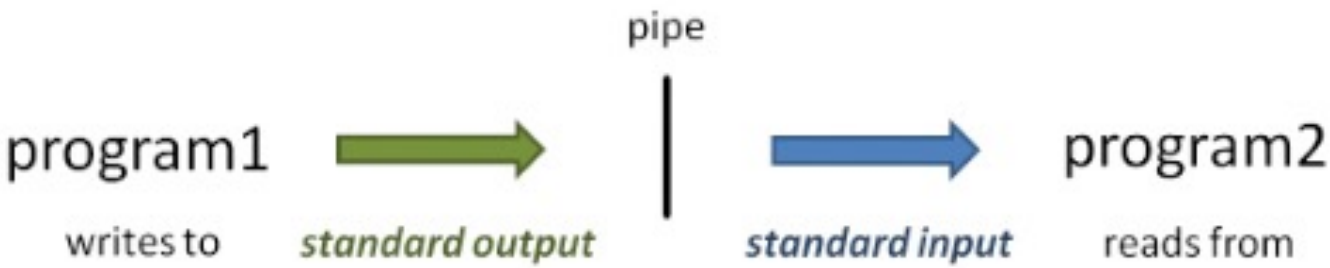
- When `fortune` runs, it does not produce any errors and does not get any external input; it just wrote its output to `stdout`

```
$ cowsay "OSSL rules!”
 _______________
< OSSL rules! >
 ---------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```



Streams & Pipes

- Let us feed `cowsay` through the `stdin` stream using a pipe (`|`)

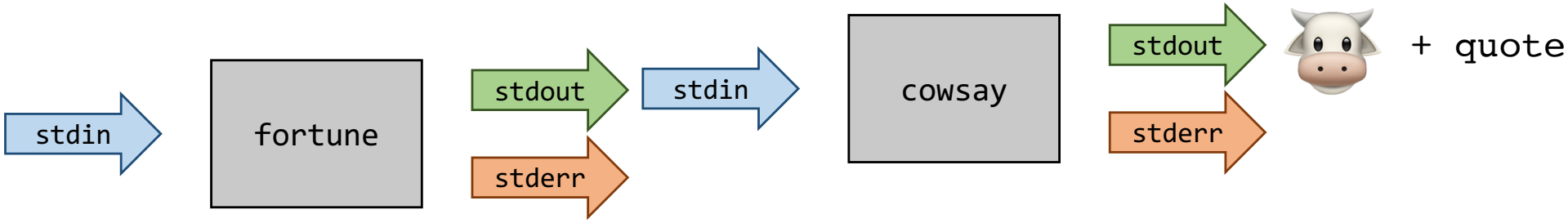


* Image src: https://wikis.utexas.edu/display/CoreNGSTools/Linux+fundamentals

Streams & Pipes

- Let us feed `cowsay` through the `stdin` stream using a pipe (`|`)

```
$ fortune | cowsay
 _____________________________________________
/ Ships are safe in harbor, but they were \
\ never meant to stay there.              /
 ---------------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```



Streams & Pipes

- cat, sed, and a pipe (|)

```
$ cat
Everything I write is repeated by cat
Everything I write is repeated by cat
```

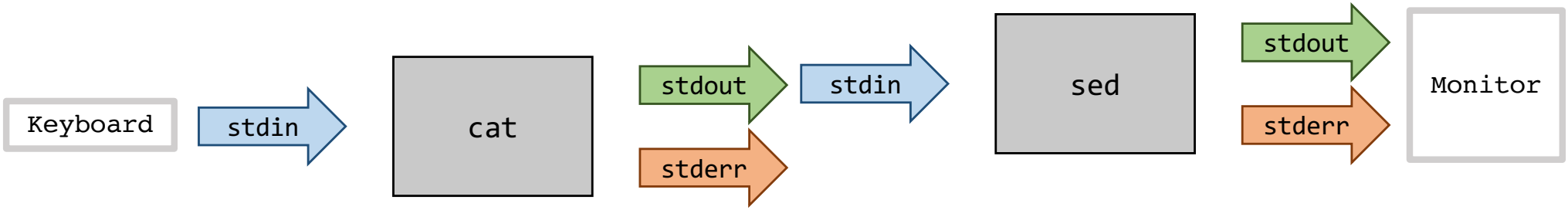
Streams & Pipes

- cat, sed, and a pipe (|)

```
$ cat
Everything I write is repeated by cat
Everything I write is repeated by cat
```

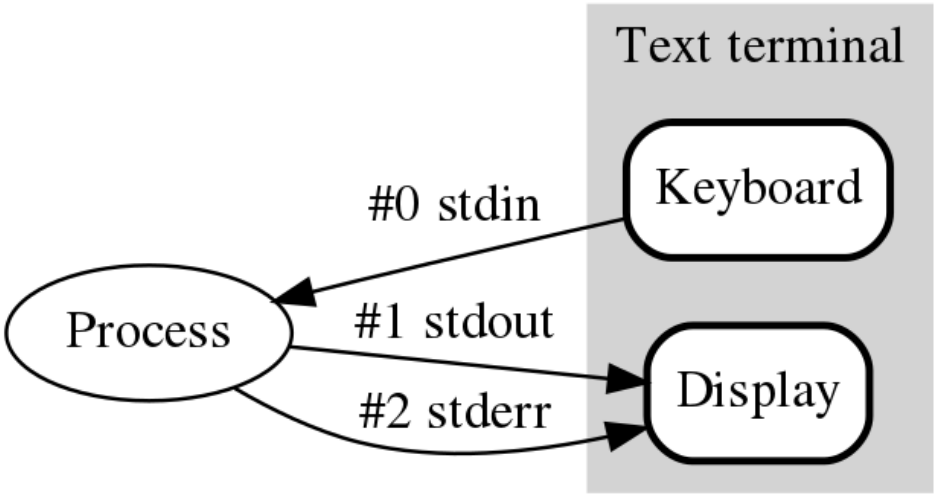
- sed - stream editor for filtering and transforming text

```
$ cat | sed -E "s/write/type/"
Everything I write is repeated by cat
Everything I type is repeated by cat
```



Streams & Pipes

- By default, data goes in through `stdout` and `stderr`, and goes out in the other end: your monitor



* Image src: https://en.wikipedia.org/wiki/Standard_streams

Redirection

- Redirecting output streams
  - To take the standard output of a program and save it to a file, you use the **>** operator
    - A single **>** overwrites any existing target; a double **>>** appends to it
    - Since standard output is stream #1, this is the same as **1>**
  - To redirect the standard error of a program you must specify its stream number using **2>**
  - To redirect standard output and standard error to the same place, use the syntax **2>&1**

Redirection

- Examples

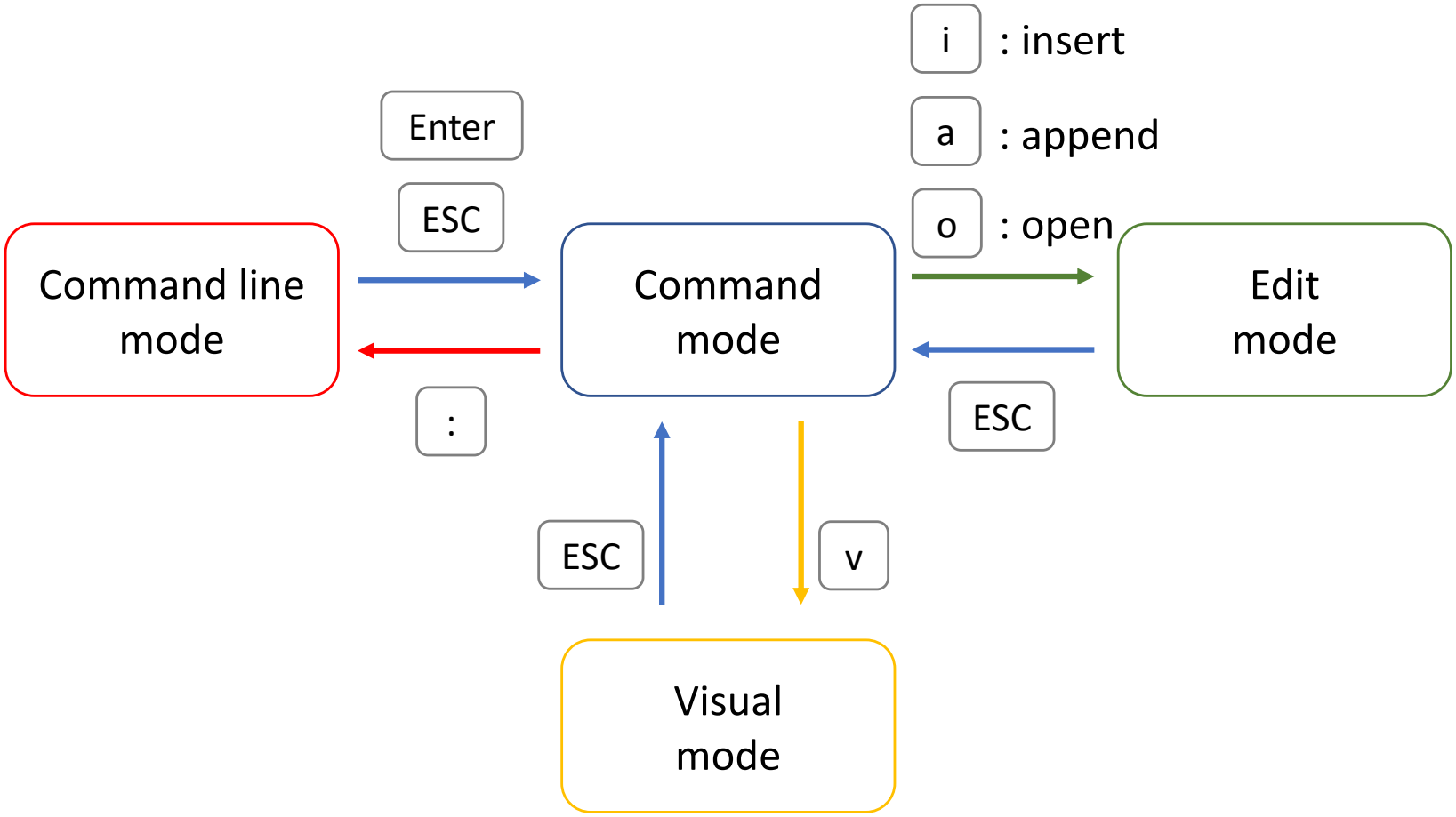
```
# redirect a long listing of your $HOME directory to a file
ls -la ~ > cmd.out
# look at the contents -- you'll see just files
cat cmd.out

# this command gives an error because the target does not exist
ls -la bad_directory

# redirect any errors from ls to a file
ls -la bad_directory 2> cmd.out
# look at the contents -- you'll see an error message
cat cmd.out

# now redirect both error and output streams to the same place
ls -la bad_directory $HOME > cmd.out
# look at the contents -- you'll see both an error message and files
cat cmd.out
```

Agenda

- Streams, pipes, & redirection
- Regular expression in *vim*

vim – Modal Editor

- Switching between modes



Search

- Search in *vim*
  - Hit the **Esc** and **/** one after the other
  - Then enter the word you want to search for
  - Hit the **Enter** key will take you to the occurrence of the search word after the cursor
  - To move between occurrences, you can press the **n** key and to move backwards you press the **N** key

  - The **?** command is very similar to the **/** command, but it searches the whole file backwards
    - In this case, the **n** key searches backwards, and the **N** key searches forwards

* Source: https://www.linuxfordevices.com/tutorials/linux/vim-search-and-replace

Search for the Word at the Cursor

- Search for the word under the cursor
  - Press the ***** key in normal mode, the cursor will be placed to the nearest occurrence of the word under the cursor
    - Press the ***** key again to search for the next occurrence
    - Use the **n** and the **N** key to cycle through the search results forward or backwards
  - The **#** is the same as ***** but it searches backwards
- You can search ignoring the case by issuing the command
  - `:set ic`

* Source: https://www.linuxfordevices.com/tutorials/linux/vim-search-and-replace

Replace (Substitute)

- **:*range* s[ubstitute]/*pattern*/*string*/cgiI**
  - For each line in ***the range*** replace a match of ***the pattern*** with ***the string*** where:

c	Confirm each substitution
g	Replace all occurrences in the line (without **g** - only first)
i	Ignore case for the pattern
I	Don't ignore case for the pattern

  - Examples
    - `:s/old_word/new_word`
    - `:%s/old_word/new_word`
    - `:%s/old_word/new_word/g`

* Source: http://vimregex.com/

Replace (Substitute)

- **:*range* s[ubstitute]/*pattern*/*string*/cgiI**
  - Some Vim commands can accept a line range in front of them
  - By specifying the line range, you restrict the command execution to this particular part of text only

Specifier	Description
number	an absolute line number
.	the current line
$	the last line in the file
%	the whole file. The same as **1,$**
't	position of mark "t"

  - `:%10,20s/old_word/new_word`

* Source: http://vimregex.com/

Replace (Substitute)

- **:*range* s[ubstitute]/*pattern*/*string*/cgiI**
  - Anchors
    - `pattern`
    - `\<pattern\>`
    - `^pattern`
    - `pattern$`
    - `^pattern$`

* Source: http://vimregex.com/

Replace (Substitute)

- **:*range* s[ubstitute]/*pattern*/*string*/cgiI**
  - "Escaped" characters (metacharacters)

#	Matching	#	Matching
.	any character except new line		
\s	whitespace character	**\S**	non-whitespace character
\d	digit	**\D**	non-digit
\x	hex digit	**\X**	non-hex digit
\o	octal digit	**\O**	non-octal digit
\h	head of word character (a,b,c...z,A,B,C...Z and _)	**\H**	non-head of word character
\p	printable character	**\P**	like **\p**, but excluding digits
\w	word character	**\W**	non-word character
\a	alphabetic character	**\A**	non-alphabetic character
\l	lowercase character	**\L**	non-lowercase character
\u	uppercase character	**\U**	non-uppercase character

* Source: http://vimregex.com/

Replace (Substitute)

- **:*range* s[ubstitute]/*pattern*/*string*/cgiI**
  - "Escaped" characters (metacharacters) examples
    - To match a date like 09/01/2000, `\d\d/\d\d/\d\d\d\d`
    - To match 6 letter word starting with a capital letter: `\u\w\w\w\w\w`

* Source: http://vimregex.com/