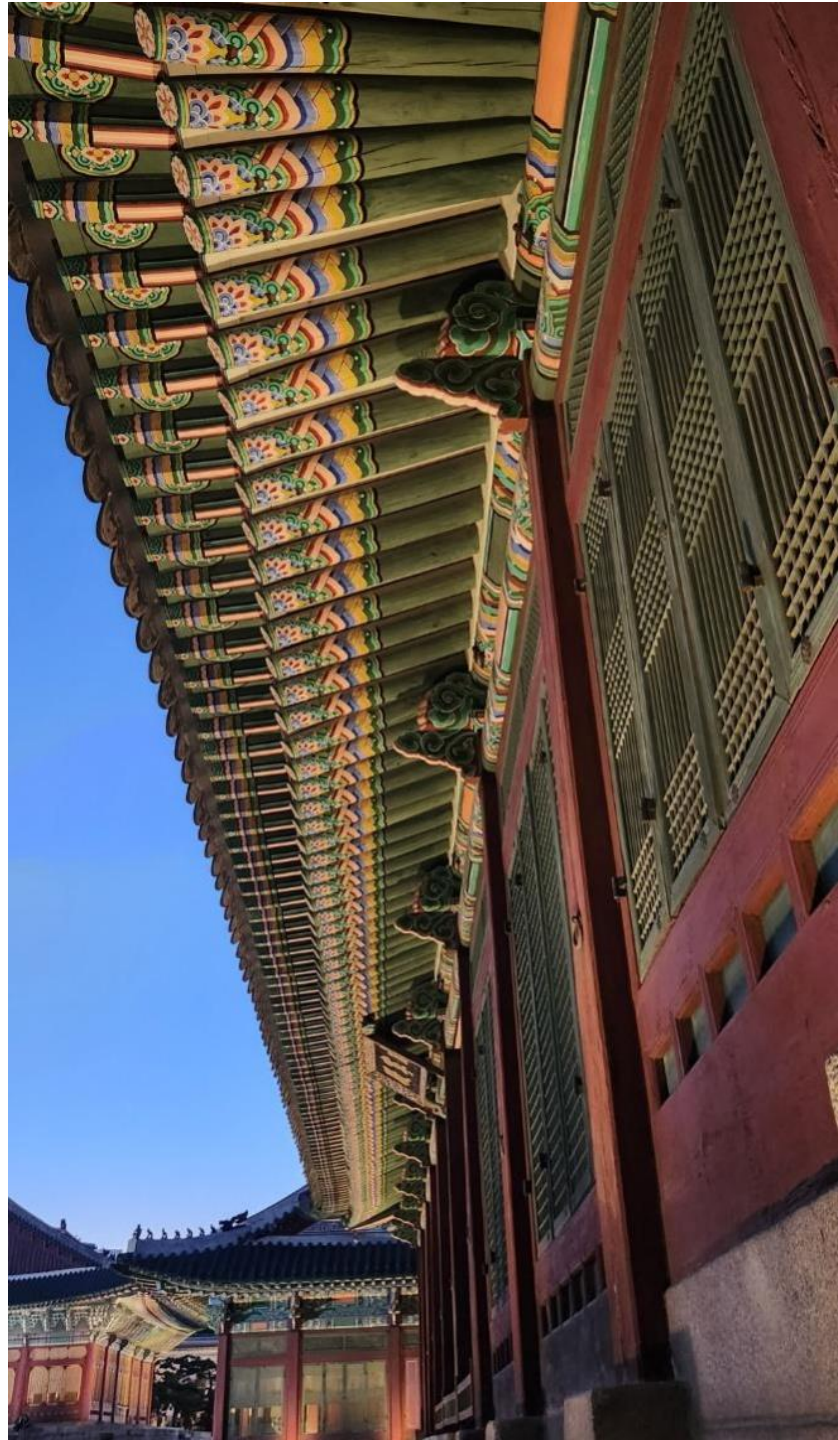
C Compiling with gcc

HGU



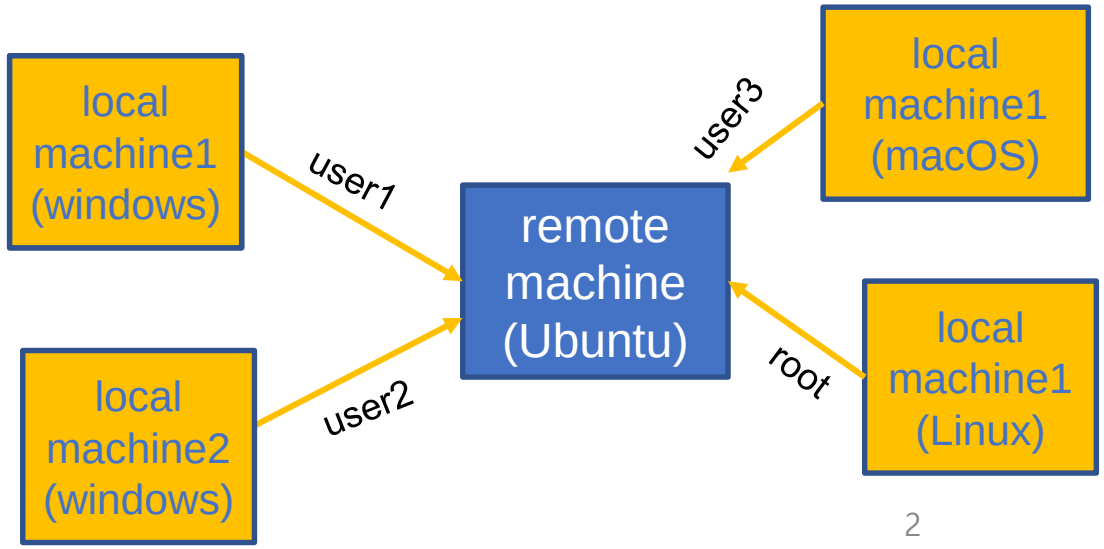
Lab-1: Remote Login to Our Server

- Local : WIndows or MacOS Machine (Laptop computer)
- Remote : Linux Server
- **ssh** command

  C:₩>**ssh <user_name>@<remote_machine_address>**
  - <user_name> : your student number
  - <remote_machine_address>: peace.handing.edu

ex) ssh yk@peace.handing.edu

```
C:\Users\yk\TinyChatEngine>ssh yk@peace.handong.edu
yk@peace.handong.edu's password:
Welcome to Ubuntu 16.04.7 LTS (GNU/Linux 4.15.0-142-gene
```



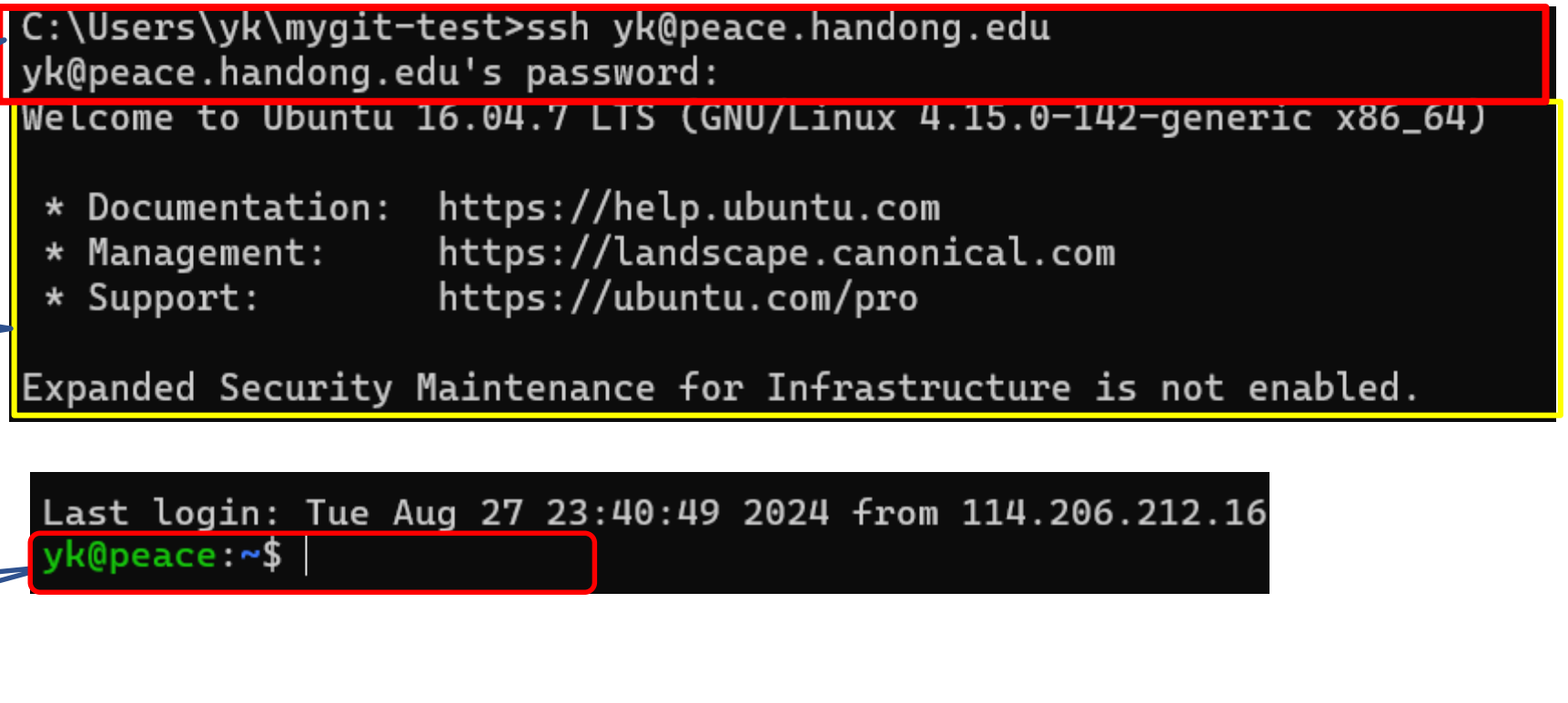
Connecting Linux Server in Windows Command Line

- Each Student Password should be changed at the first login

Windows command (ssh)

Login Welcome Message from Linux System

prompt($) from Linux shell



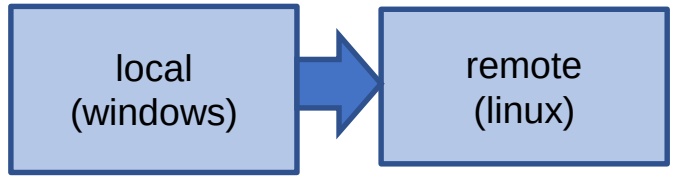
- *Your login id and password will be sent individually by personal email.*

Lab-2: How to transfer files to/from remote host

- local에서 원격 host로 파일 보내기(scp)
  - **scp** *<source-file> <user_id>@<remot-host>:<target-file>*

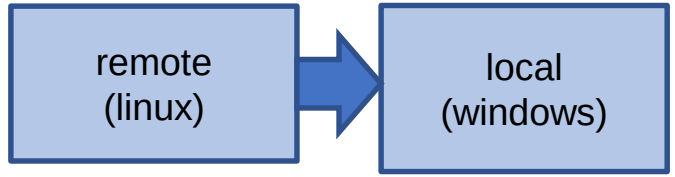
```
C:\Users\yk\Pictures>scp khj.jpg yk@peace.handong.edu:image.jpg
yk@peace.handong.edu's password:
khj.jpg

C:\Users\yk\Pictures>
```



- 원격 host에서 local로 파일 가져오기(scp)
  - **scp** *<user_id>@<remot-host>:<source_file> <target-file>*

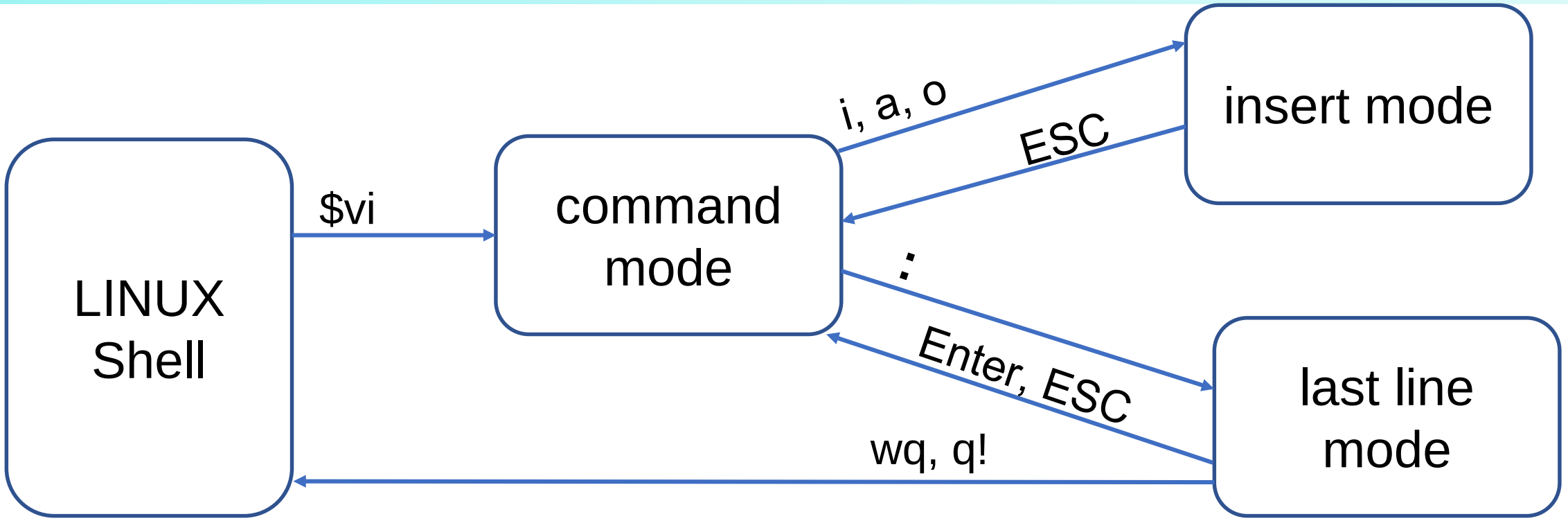
```
C:\Users\yk\Pictures>scp yk@peace.handong.edu:image.jpg k.jpg
yk@peace.handong.edu's password:
image.jpg
```



Text editor in Linux

- ed : line editor
- vi : visual editor
- vim : modified vi (allows arrow key)

vi Screen Editor mode



- command mode : navigate, delete, copy, paste, and do a number of other tasks—except entering text.
- insert mode : entering text (contents)
- last line mode : reading/writing file to filesystem or setting (configuration) editor

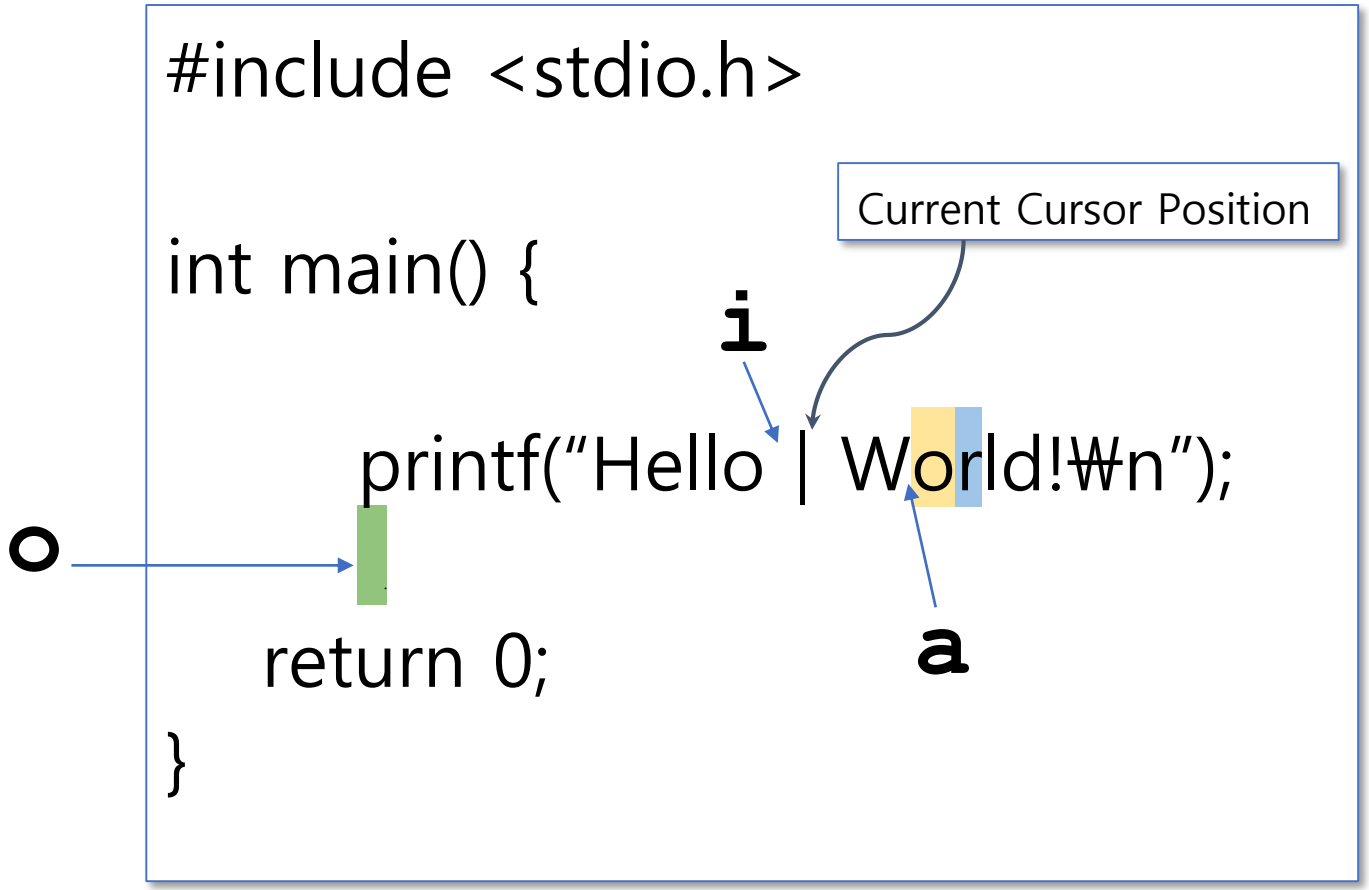
Writing C Source using vi editor

- Starting Vim in Shell
  - Open a file: `$ vi filename.c`
  - Vim starts in Normal mode as default - edit text or enter commands

- Saving and Exiting in Command Mode
  - Save the file `:w`
  - Exit without saving `:q!`
  - Save and exit `:wq`

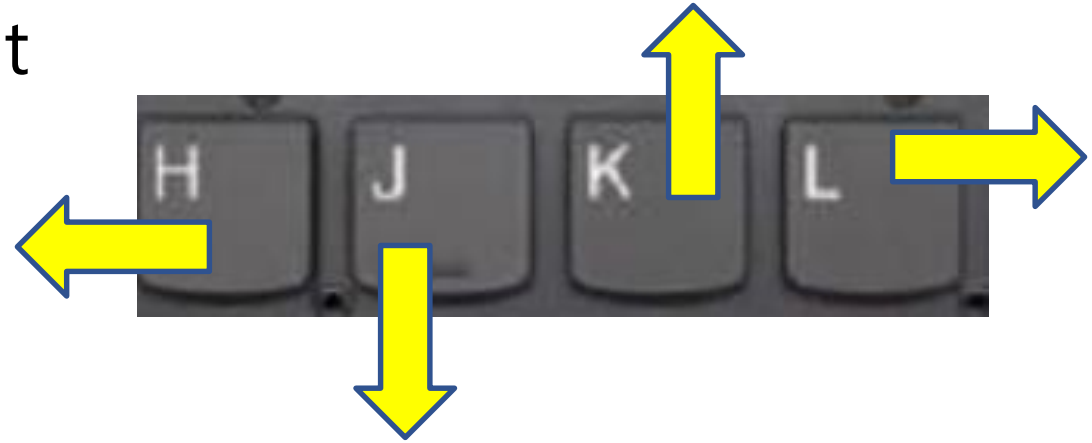
Insert Modes of vi

- Insert mode - Write the hello.c prints out “Hello World!”

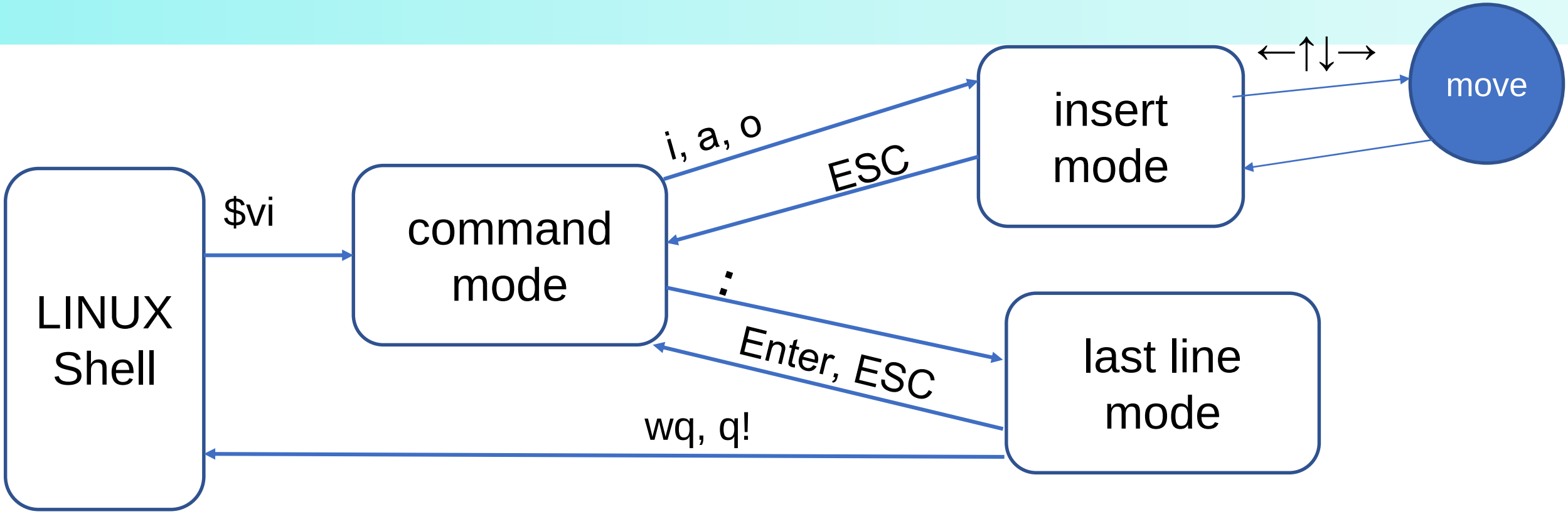


Nevigating in Command mode

- one character movement
  - h : left
  - j: down
  - k: up
  - l: right
- page movement
  - ^F : forward (next page)
  - ^B : backward (previous page)



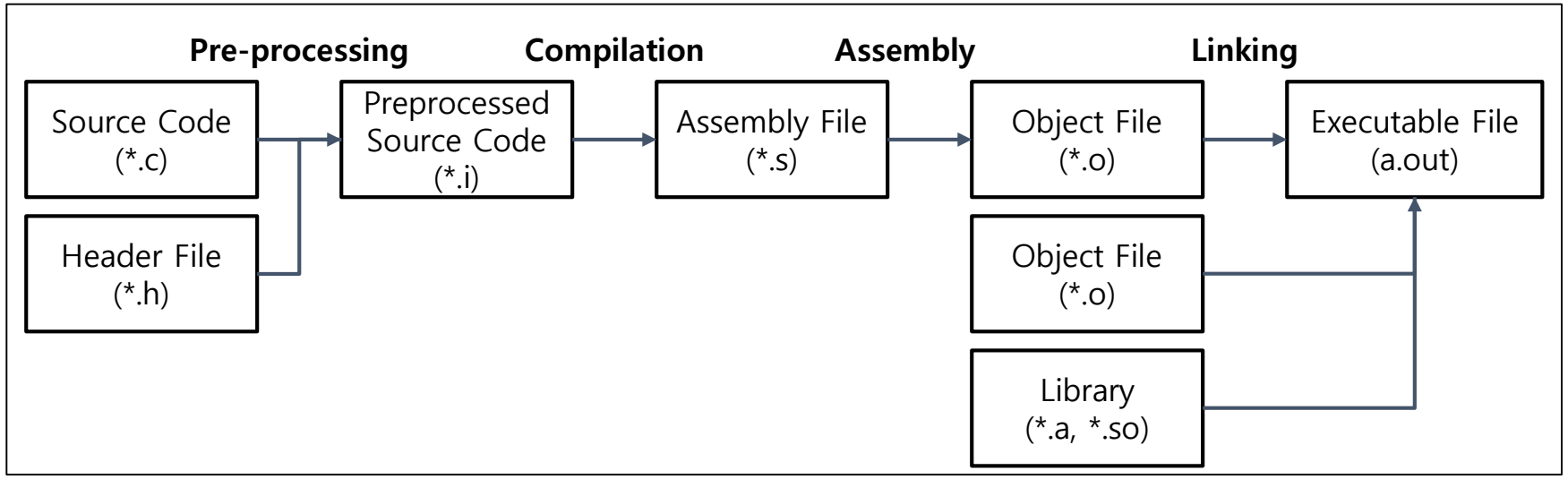
vim



- Arrow keys (up,down,left,right) in insert mode for navigation
  - ←: ESC + h + i
  - ↓: ESC + j + i
  - ↑: ESC + k + i
  - →: ESC + l + i

Compiling C Source

- Compilation Steps



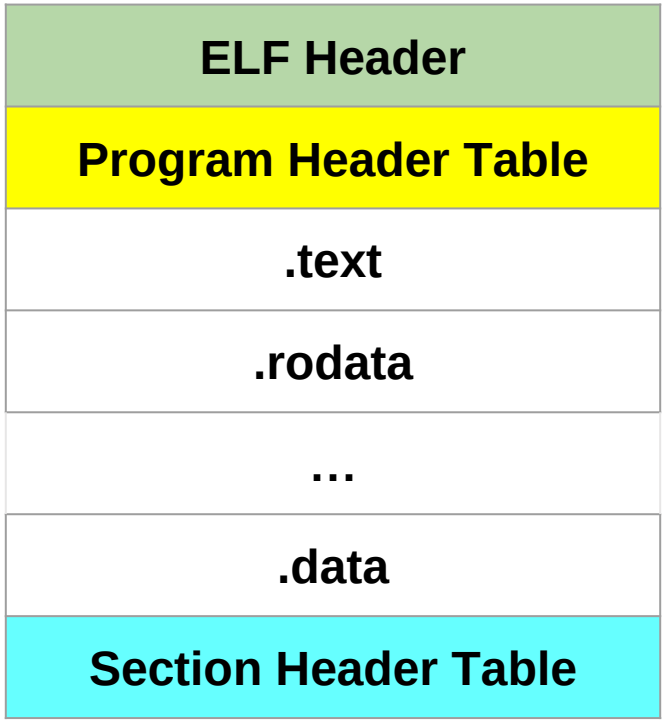
1) **Preprocessing** directives such as **`#define`** and **`#include`**
2) **Compilation** translates the source code to assembly code (.s)
3) **Assembly** converts the assembly code into relocatable binary object code (.o)
4) **Linking** creates a executable file from relocatable binaries (.o) and libraries (.a or .so)

Linux Executable and Linkable Format (ELF)

- There are 3 main ELF object files
  - **Relocatable File** is an object file generated from source code (.o files)
  - **Shared Object File** containing code and data that can be used by multiple programs (.so files)
  - **Executable File** is generated by linking together Relocatable Files and Shared Object Files
  - Core Dumps

Linux ELF Layout

Contains information about the file's **segmentations** which defines how to create a process image for this program to run (necessary in **execution**)



Contains information about **Program Header Table** and **Section Header Table**

Contains information about the file's **sections** for linking and relocation (necessary in **linking**)

Libraries in Linux

- C library consists of two parts
  - **Application Programming Interface (API)** defined in header text files (.h)
  - **Implementation** usually provided as precompiled binary files (.a, .so)

- Two types of Library
  - **Static Library** is an archive(single binary file) of object files (**.a** files)
  - **Dynamic Library** is loaded during the runtime **(.so** files)
    - Shared Library uses the object file built with **PIC (Position Independent Code)**

PIC

- **Position Independent Code (PIC)** is a type of machine code that executes correctly regardless of its absolute address.
- The code can be relocated to any address without needing changes.
- Dynamic Libraries designed to support PIC have been referred to as shared libraries
- As 64-bit systems' compilers strongly prefer PIC, the distinction between dynamic libraries and shared libraries has faded and the terms are now almost synonymous

Static Library

- Is a collection of object files called archive(.a)
- Linker extracts needed object files from archive and attaches them to program
- When linker encounters an archive in command line, it searches the already passed objects to see if there is a reference to objects in this archive or not

Shared Library

- Is also a collection of objects
- When it is linked into another program, the program does not contain the whole object, but just references to the shared library
- Is not a collection of object files, but a single big object file which is a combination of object files
- Shared Libraries are Position Independent Codes, because the function in a .so, may be loaded at different addresses in different programs

Shared Library

- The linker just includes the name of the “so” in executable file
- The OS is responsible to find the specified “so” file
- By default, system searches only “/lib” and “/usr/lib”
- Programmer can indicate another path by setting the LD_LIBRARY_PATH environment variable or referring the directory path in GCC command line through -L option

Compiling with Libraries

- How to Build Static Library

GCC Command Example

```
$ gcc -c foo.c -o foo.o
$ ar rcs libfoo.a foo.o      # Build static library (libfoo.a)
                             # ar rcs is used to create, update the archive from the specified files
$ gcc main.c -L. -lfoo       # -lfoo is a alias of libfoo.a
                             # -L specify a directory where the linker should look for libraries
```

- How to Build Dynamic Library

GCC Command Example

```
$ gcc -fPIC -o foo.o -c foo.c      # -fPIC is needed to build object file as PIC
$ gcc -shared -o libfoo.so foo.o   # -shared builds shared library (libfoo.so)
$ gcc myprog.c -L/lib -lfoo        # -lfoo is a alias of libfoo.so
                                   # -L specify a directory where the linker should look for libraries
```

Compiling with Libraries

- How to Build with Libraries

GCC Command Example

```
$ gcc main.c -I/include -L/lib -lfoo
# -lfoo is a alias of libfoo.s
# -L specify a directory where the linker should look for libraries
# -I specify a directory where the linker should look for header files
```

- Why does the command need `-I` and `-L` Options?
  - Compiler searches the header files at current directory and system standard directories (such as '/usr/include')
  - Linker searches the libraries in the same sequence ('./', '/usr/lib')
  - `-I` and `-L` option tells to the compiler and linker to add specific directory to the list of searching directories

Compiling with Libraries

- How to Build with Libraries

GCC Command Example

```
$ gcc main.c -I/include -L/lib -lfoo
# -lfoo is a alias of libfoo.s
# -L specify a directory where the linker should look for libraries
# -I specify a directory where the linker should look for header files
```

- How does the system known `-lfoo` is static or shared library?
- Priority of shared library(.so) is higher than static library(.a) unless explicitly specified

  (-static option in GCC command line)

Reference

- https://www.vim.org/
- https://diveintosystems.org/book/C2-C_depth/advanced_libraries.html
- https://diveintosystems.org/book/C2-C_depth/advanced_writing_libraries.html
- https://www.slideshare.net/slideshow/advanced-c-programming-in-linux/85064690#8