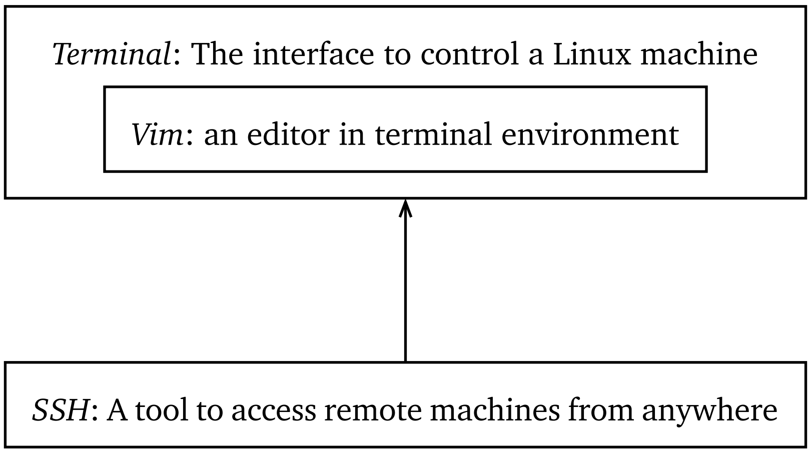
Terminal Environment

A terminal provides direct access to your operating system and is essential for efficient programming and system administration. In the following, we will introduce how to get a terminal, how to edit files with the terminal (Vim), and how to use the terminal to interact with remote machines (SSH).


Terminal: The interface to control a Linux machine
Vim: an editor in terminal environment
SSH: A tool to access remote machines from anywhere

Linux Operating System

Linux or macOS provides the most straightforward way to get a terminal. Linux is a free, open source operating system widely used world wide. It has became a standard for many clusters and supercomputers. Even Windows system has a good support to Linux, users can also access a Linux terminal through Windows Subsystem for Linux (WSL), which could be installed with a single command in the PowerShell:

```
wsl --install
```

Shell (or Terminal)

The shell interprets keyboard commands and passes them to the operating system. The default shell for most Linux distributions is Bash. You can also use more powerful shells like Zsh (often combined with oh-my-zsh), which offers features like spelling correction, advanced tab-completion, plugins and themes.

To get started with a terminal, the following shortcuts are essential:

- `man <command_name>`: Access command documentation
- `CTRL-C`: Interrupt a running program
- `CTRL-D`: Exit the shell
- `Up` and `Down`: Navigate through command history

In a terminal, you can:

- Navigate and manipulate files and directories:

  ```
  $  ls      # list directory contents
  Makefile      Manifest.toml Project.toml  README.md     book
  $  cd book  # change directory
  ```

```
$  pwd      # print name of current/working directory
/Users/liujinguo/Documents/SCFP/book
$  ls       # list directory contents
book.typ            chap2-linalg        chap4-ad            chap6-quantum
ebook.typ           shared
chap1-julia         chap3-optimization  chap5-tensornetwork dist
home.typ            templates
$  mkdir chap7-quantum  # make directories
$  rmdir chap7-quantum  # remove directories
$  rm -r dist           # remove directories and files recursively
```

- Read and write text files with the `echo` and `cat` commands.

```
$  echo "Hello, world!" > hello.txt  # write to a file
$  cat hello.txt                     # display the content of a file
Hello, world!
```

- Create an alias for a command.

```
$  alias ll="ls -l"   # create an alias for the `ls -l` command
$  ll                 # use the alias
total 312
-rw-r--r--@  1 liujinguo  staff    21K Feb  1 23:02 git.typ
drwxr-xr-x@ 10 liujinguo  staff   320B Feb 10 08:07 images
-rw-r--r--@  1 liujinguo  staff    25K Feb  4 19:24 julia-basic.typ
-rw-r--r--@  1 liujinguo  staff    12K Jan 19 23:13 julia-release.typ
```

- Monitor your system resources with the following commands.

```
$  lscpu   # display information about the CPU architecture
$  lsmem   # list the ranges of available memory with their online status
$  top     # display Linux processes
```

- Access many useful tools in a terminal, which include

```
$  vim <filename>    # Vi IMproved, a programmer's text editor
$  ssh <username>@<hostname>    # the OpenSSH remote login client
$  git <arguments>     # the stupid content tracker
$  tar <arguments>     # an archiving utility
```

Vim - a terminal text editor

To edit files in the terminal, you can use Vim - the default text editor in most Linux distributions. Vim has three primary modes, each tailored for specific tasks:

- **Normal Mode**: Navigate through the file and perform tasks like deleting lines or copying text. Enter by pressing `ESC`
- **Insert Mode**: Insert text as in conventional text editors. Enter by typing `i` in normal mode
- **Command Mode**: Input commands for tasks like saving files or searching. Enter by typing `:` in normal mode

Here are some essential Vim commands to get started:

```
i       # input
:w      # write
:q      # quit
:q!     # force quit without saving

u       # undo
CTRL-R  # redo
```

All commands must be executed in **normal mode** (press `ESC` to enter).

SSH - connect to remote machines

The Secure Shell (SSH) protocol enables secure remote command execution over unsecured networks. Using cryptography for authentication and encryption, SSH allows you to:

- Push code to remote git repositories
- Access and control remote machines

To connect to a remote system like a university cluster, you'll need:

1. The hostname or IP address
2. Your username
3. Authentication credentials

The basic connection syntax is:

```
ssh username@hostname
```

For example, to connect to a cluster with the username `user` and hostname `cluster.example.com`, you would use:

```
ssh user@cluster.example.com
```

Streamlining SSH Access

To avoid repeatedly typing hostnames and usernames, you can configure SSH using the `~/.ssh/config` file. This configuration file allows you to create aliases and set default parameters for your SSH connections.

Here's an example configuration:

```
Host amat5315
  HostName <hostname>
  User <username>
```

In this example, `amat5315` serves as an alias for the remote host. Once you've configured the `~/.ssh/config` file, you can connect to the remote machine simply by typing:

```
ssh amat5315
```

To avoid typing the password everytime you login, you can use the command

```
ssh-keygen
```

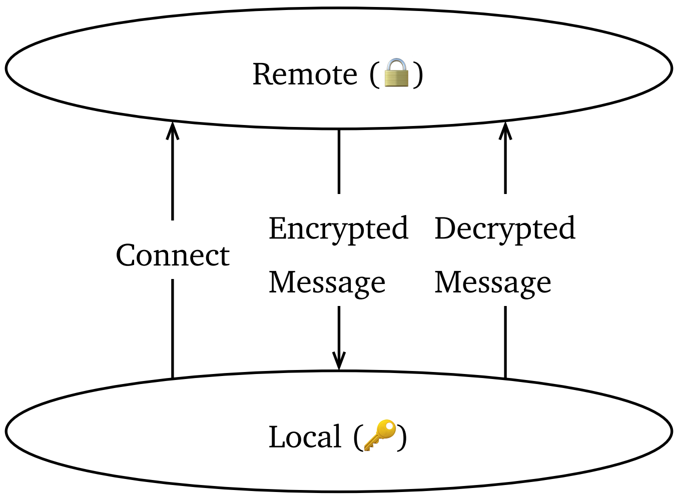
to generate a pair of public and private keys, which will be stored in the `~/.ssh` folder on the local machine. After setting up the keys, you can copy the public key to the remote machine by typing

```
ssh-copy-id amat5315
```

Try connecting to the remote machine again, and you will notice that entering the password is no longer necessary.

How does an SSH key pair work?

The SSH key pair consists of two asymmetric keys: a public key (or lock) and a private key. In the example above, the public key is uploaded to the remote machine, while the private key remains securely stored on your local machine. The public key can be shared freely, but the private key must remain confidential.



When connecting to a server, the server needs to verify your identity. It does this by checking if you possess the private key that matches the public key stored on the server. If you have the correct private key, access is granted.

The core principle of the SSH key pair is that the **public key can encrypt a message that only the private key can decrypt**. Think of the public key as a lock and the private key as the key to unlock it. This forms the basis of the SSH protocol. The server can send you a message encrypted with your public key, and only you can decrypt it with your private key. This ensures the server knows you have the private key without needing to send it.

Resources

- MIT Open Course: The Missing Semester of Your CS Education