# 14

# *Linux tutorial*

**William W.-Y. Hsu**
*Department of Computer Science and Engineering*
*Department of Environmental Biology and Fisheries Science*
*National Taiwan Ocean University*

## CONTENTS



\nobreak

## 14.1 Introduction

Linux has been around since the mid '90s, and has since reached a user-base that spans industries and continents. For those in the know, you understand that Linux is actually everywhere. It's in your phones, in your cars, in your refrigerators, your Roku devices. It runs most of the Internet, the supercomputers making scientific breakthroughs, and the world's stock exchanges. But before Linux became the platform to run desktops, servers, and embedded systems across the globe, it was (and still is) one of the most reliable, secure, and worry-free operating systems available. For those not in the know, worry not —here is all the information you need to get up to speed on the Linux platform.

Just like Windows XP, Windows 7, Windows 8, Windows 10 and Mac OS X, Linux is an operating system. An operating system is software that manages all of the hardware resources associated with your desktop or laptop. To put it simply –the operating system manages the communication between your software and your hardware. Without the operating system (often referred to as the "OS"), the software wouldn't function. The OS is comprised of a number of pieces:

- The Bootloader: The software that manages the boot process of your computer. For most users, this will simply be a splash screen that pops up and eventually goes away to boot into the operating system.

- The kernel: This is the one piece of the whole that is actually called "Linux". The kernel is the core of the system and manages the CPU, memory, and peripheral devices. The kernel is the "lowest" level of the OS.

- Daemons: These are background services (printing, sound, scheduling, etc) that either start up during boot, or after you log into the desktop.

- The Shell: You've probably heard mention of the Linux command line. This is the shell —a command process that allows you to control the computer via commands typed into a text interface. This is what, at one time, scared people away from Linux the most (assuming they had to learn a seemingly archaic command line structure to make Linux work). This is no longer the case. With modern desktop Linux, there is no need to ever touch the command line.

- Graphical Server: This is the sub-system that displays the graphics on your monitor. It is commonly referred to as the X server or just "X". Desktop Environment: This is the piece of the puzzle that the users actually interact with. There are many desktop environments to choose from (Unity, GNOME, Cinnamon, Enlightenment, KDE, XFCE, etc). Each desktop environment includes built-in applications (such as file managers, configuration tools, web browsers, games, etc).

- Applications: Desktop environments do not offer the full array of apps. Just like Windows and Mac, Linux offers thousands upon thousands of high-quality software titles that can be easily found and installed. Most modern Linux distributions (more on this in a moment) include App Store-like tools that centralize and simplify application installation. For example: Ubuntu Linux has the Ubuntu Software Center which allows you to quickly search among the thousands of apps and install them from one centralized location.

Material copied from `https://www.linux.com/learn/complete-beginners-guide-linux`

## 14.2 Terminals

A terminal is at the end of an electric wire, a shell is the home of a turtle, `tty` is a strange abbreviation and a console is a kind of cabinet. In unix/linux terminology, the short answer is that:

- terminal = `tty` = text input/output environment
- console = physical terminal
- shell = command line interpreter

A terminal or a console is a piece of hardware, using which a user can interact with a host. Basically a keyboard coupled with a text screen. Nowadays nearly all terminals and consoles represent "virtual" ones.

The file that represents a terminal is, traditionally, called a tty file. If you look under the "`/dev`" directory of a UNIX system, you'll find a lot of `tty` files connected to virtual consoles (e.g. `tty1` on linux), virtual terminals (e.g. `pts/0`) or physically connected hardware (e.g. ttyS0 is the physical serial terminal, if any, attached on first serial port of the host).

A console must be a piece of hardware physically connected to (or part of) the host. It has a special role in the system: it is the main point to access a system for maintenance and some special operation can be done only from a console (e.g. see single user mode). A terminal can be, and usually is, a remote piece of hardware.

A terminal emulator is a program that emulates a physical terminal (e.g. xterm, gnome-terminal, minicom, putty).

So when you look to a "text window" on your linux system (under X11) you are looking to: a terminal emulator, connected to a virtual terminal, identified by a `tty` file, inside which runs a shell.

We will be using the free terminal software:

- `putty` (`http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html`)
- `pietty` (`http://ntu.csie.org/~piaip/pietty/`)
- `Bitvise SSH Client` (`https://www.bitvise.com/ssh-client-download`)
- `xterm` (if you have desktop version of Linux installed)

Using **Bitvise SSH Client** as an example (see Figure 14.1, we fill in the server IP (or hostname) in **Host** field and your account in the **Username** field. Default port setting for secure connection (which is the only way to log into our systems) is port 22. By pressing **Login**, you will be directed to another window to our system. In addition, a SSH FTP window will be opened so you can transfer files between your desktop and our server cluster.

Currently, our course provides 1 physical Linux superserver as below:

- 140.121.197.14 linux.deepsea9.taipei

The primary central controller serve is `WatchTower.cse.ntou.edu.tw`, which hosts a web server and NIS information.

After pressing `Login` button, if it is the first time your machine is connection to the server, a prompt screen will appear as in Figure 14.2, generating the encryption keys to establish a secure connection. You can choose to save the key for future use by **Accept and Save** or accept the key for this single session only (**Accept for This Session**).

Following, a window will appear and ask you for your password (see Figure 14.3. This password is the one TA handed out to you prior to this class.

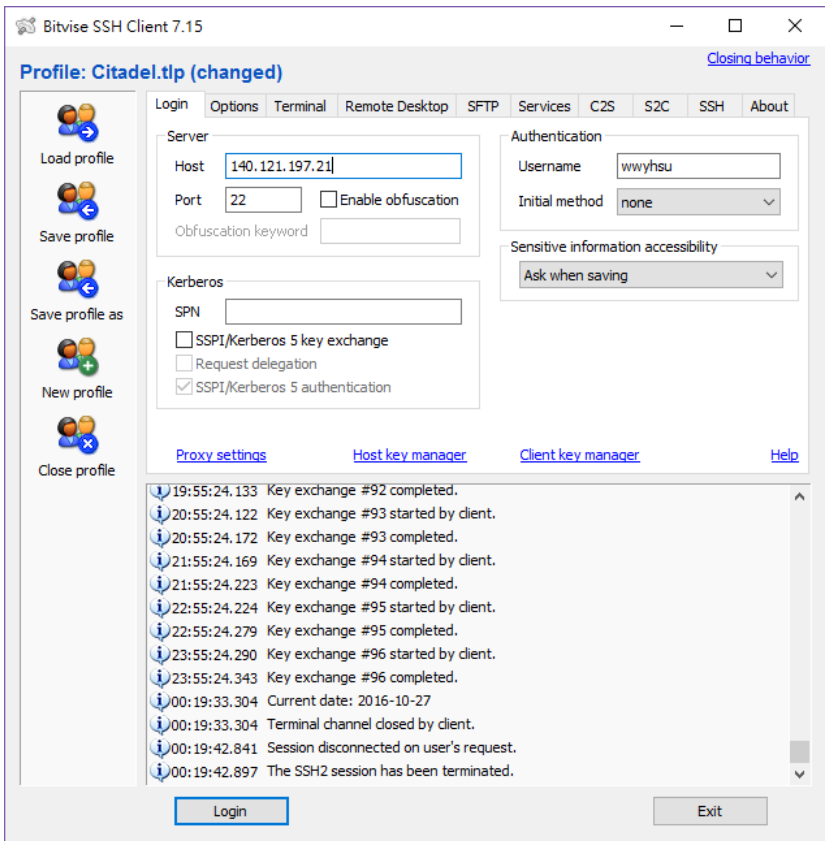


**FIGURE 14.1**
**Bitvise SSH client.**

If your password is correct, you will be given a terminal with a command prompt as shown in Figure 14.4 plus a secure FTP channel window which you can move files between your machine and the server.

**login** is used when signing onto a system. It can also be used to switch from one user to another at any time (most modern shells have support for this feature built into them, however). If an argument is not given, login prompts for the username.

If the user is not `root`, and if `/etc/nologin` exists, the contents of this file are printed to the screen, and the login is terminated. This is typically used to prevent logins when the system is being taken down.

If special access restrictions are specified for the user in `/etc/usertty`, these must be met, or the log in attempt will be denied and a `syslog` message will be generated.

If the user is `root`, then the login must be occurring on a `tty` listed in `/etc/securetty`. Failures will be logged with the `syslog` facility.

After these conditions have been checked, the password will be requested and checked (if a password is required for this username). Some systems have countermeasures for preventing password guessing. For example, ten attempts are allowed before login dies, but

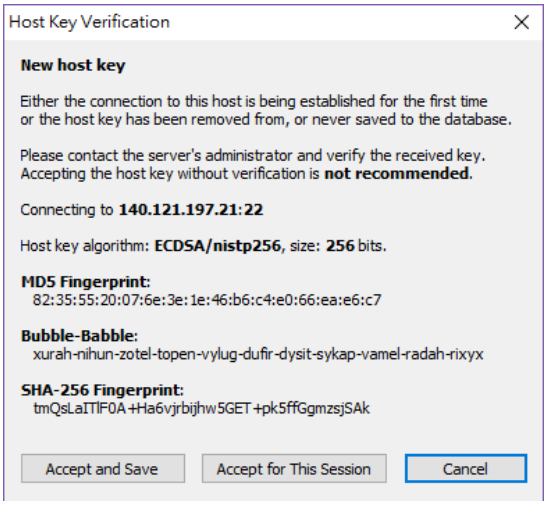


**FIGURE 14.2**
**Bitvise SSH key generation.**

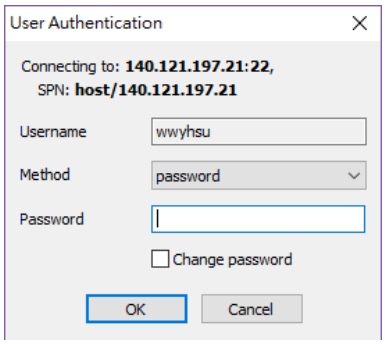


**FIGURE 14.3**
**Bitvise SSH login prompt.**

after the first three, the response starts to get very slow. Login failures are reported via the `syslog` facility. This facility is also used to report any successful `root` logins. Usually, modern system do not allow `root` login except for using console terminals.

If the file `.hushlogin` exists, then a "quiet" login is performed (this disables the checking of mail and the printing of the last login time and message of the day). Otherwise, if `/var/log/lastlog` exists, the last login time is printed (and the current login is recorded).

The user's shell is then started. If no shell is specified for the user in `/etc/passwd`, then

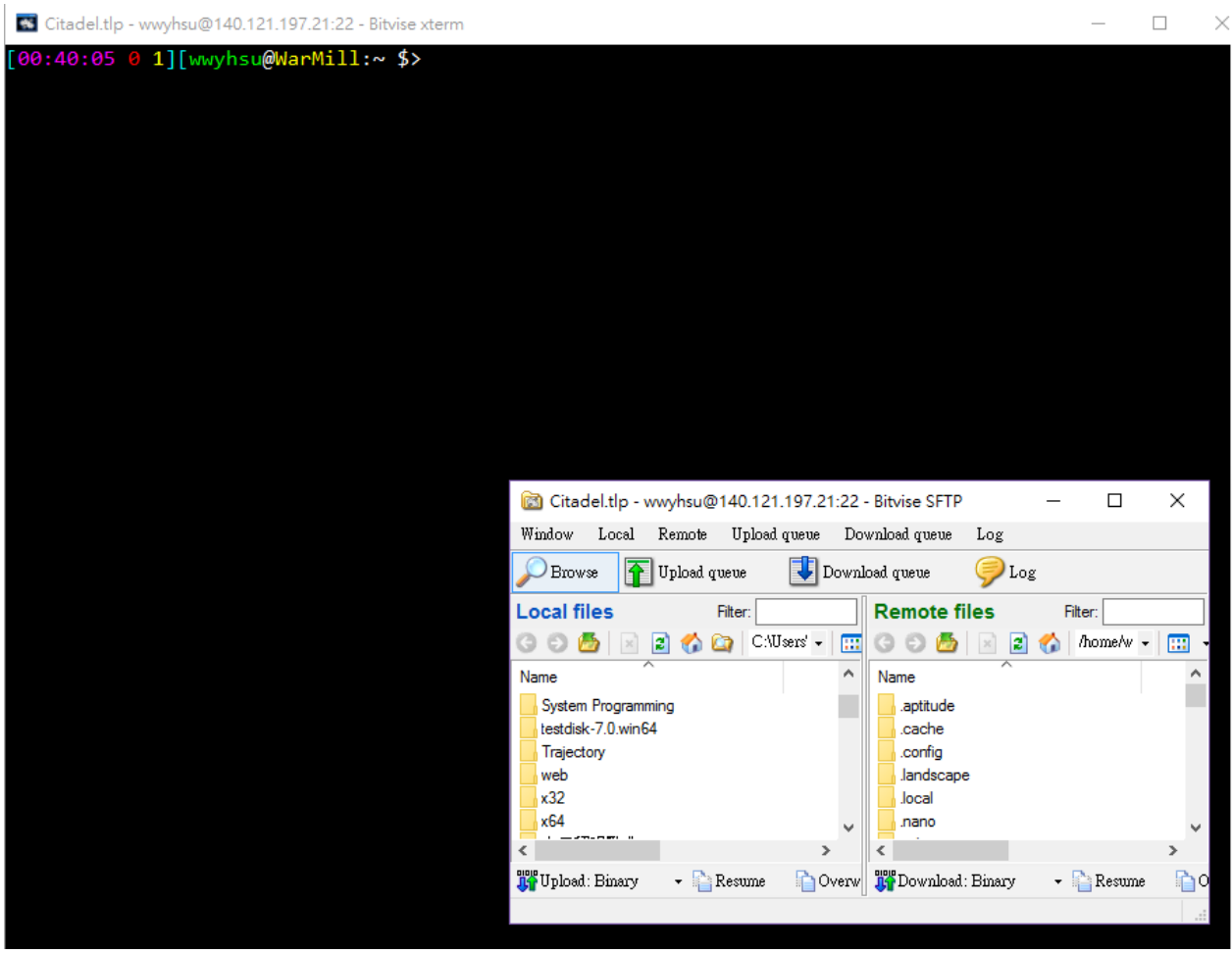


**FIGURE 14.4**
**Bitvise SSH key terminal and SFTP window.**

`/bin/sh` is used. If there is no directory specified in `/etc/passwd`, then `/` is used (the home directory is checked for the `.hushlogin` file described above).

To exit the system, simply type the command `logout`. Remember to do so in public systems because if you do not logout, other people may use your account to access your data.

## 14.3 Shells

In Unix, the shell is a program that interprets commands and acts as an intermediary between the user and the inner workings of the operating system. Providing a command-line interface (i.e., the shell prompt or command prompt), the shell is analogous to DOS and serves a purpose similar to graphical interfaces like Windows, Mac OS X, and the X Window System.

On most Unix systems, there are several shells available. For the average user, they offer similar functionality, but each has different syntax and capabilities. Most shells fall within one of two classes: those descended from the Bourne shell (i.e., sh), which first appeared in Version 7 Unix, and those from the C Shell (i.e., `csh,tcsh`), which made its debut in BSD. Nearly every Unix system has these two shells installed, but may also have several others, including:

| Shell | Class | Description |
|---|---|---|
| Korn shell (ksh) | sh | An extension of the Bourne shell with several features adapted from the C shell; the POSIX shell is based on the Korn shell |
| TC shell (tcsh) | csh | A revision of the C shell with substantially expanded capabilities; the default shell in modern BSD implementations, including Mac OS X/Darwin |
| Bourne-again shell (bash) | sh | An extension of the Bourne shell, but with unique features; part of the GNU project and the default shell for Linux |

You can check the file in `/etc/shells` to see what shells are available in your system. If you want to change a login shell, you can issue the following command `chsh` (**ch**ange **sh**ell):

```
wwyhsu@WatchTower:$ chsh
Password: (Input your personal password if prompted)
Changing the login shell for wwyhsu
Enter the new value, or press ENTER for the default
Login Shell [/bin/bash]: /bin/tcsh
```

The above command is for single machine layout. For our mini computer center, our machines are linked with NIS and NFS, you will have to use the command `ypchsh` to change your shell:

```
wwyhsu@WatchTower:$ ypchsh
Changing NIS account information for wwyhsu on WatchTower.
Please enter password: (Input your personal password if prompted)

Changing login shell for wwyhsu on WatchTower.
To accept the default, simply press return. To use the
system's default shell, type the word ``none''.
Login shell [/bin/bash]: /bin/tcsh

The login shell has been changed on WatchTower.
```

## 14.4 Changing User Passwords

Both Linux and UNIX use the `passwd` command to change user password. The `passwd` is used to update a user's authentication token (password) stored in **shadow file** (`/etc/shadow`).

The `passwd` changes passwords for user and group accounts. A normal user may only change the password for his/her own account, the super user (or `root`) may change the password for any account. The administrator of a group may change the password for the

group. `passwd` also changes account information, such as the full name of the user, user' s login shell, or password expiry date and interval.

### 14.4.1 Set or Change User Password

Type `passwd` command as follows to change your own password:

```
wwyhsu@WatchTower:$ passwd
Changing password for wwyhsu.
(current) UNIX password: (Your original password)
Enter new UNIX password: (Your new password)
Retype new UNIX password:
passwd: password updated successfully
```

In our mini computer center, you will have to use `yppasswd` to change your password:
Type `yppasswd` command as follows to change your own password:

```
wwyhsu@WatchTower:$ yppasswd
Changing NIS account information for wwyhsu on WatchTower.
Please enter old password: (Your original password)
Changing NIS password for wwyhsu on WatchTower.
Please enter new password: (Your new password)
Please retype new password:

The NIS password has been changed on WatchTower.
```

Remember, do not forget your password. In this case, only the system administrator or `root` can help you change your password (not recover).

## 14.5 File Permissions

The Linux operating system differs from other computing environments in that it is not only a multitasking system but it is also a multi-user system as well. In order to make this practical, a method had to be devised to protect the users from each other. After all, you could not allow the actions of one user to crash the computer, nor could you allow one user to interfere with the files belonging to another user.

Linux uses the same permissions scheme as Unix. Each file and directory on your system is assigned access rights for the owner of the file, the members of a group of related users, and everybody else. Rights can be assigned to *read* a file, to *write* a file, and to *execute* a file (i.e., run the file as a program). Example vertical listing of a directory:
Type `ls -l` command as follows to show directory listing:

```
wwyhsu@WatchTower:$ ls -l
drwxrwxr-x 2 wwyhsu wwyhsu 4096 Sep 17 03:48 www
```

```
<Permission> <Links> <Owner(User)> <Owner(Group)> <Filesize> <Creation Date>
<Filename>
```

The first field represents the file permissions. The second represent links in this file. In case it is a directory, it represents the number of files in this directory. The third and fourth field represents the owner of this file (user, group). The fifth field is the file size, the six field is the creation date of the file and the last field is the filename. File permission in Linux is separated into three components, user, group, and everyone.

```
- rwx rw- --- File permissions in groups of three.
1 2   3   4
```

The field *1* represent the file type. It can be

- **-**: normal file.
- **l**: soft links.
- **d**: a directory.
- **c**: a character device (i.e., keyboard, terminals).
- **b**: a block device (i.e., hard drives).

The field *2,3,4* represents the permission. In each groups of three the first position represents *read* permission, the second position represents *write* permission, and the third position represents *execute* permission. A **-** represents the corresponding permission in not allowed. For example,

```
-rwxr-xr-- 1 wwyhsu wwyhsu 8728 Oct  3 19:07 series
drwxrwxr-x 2 wwyhsu wwyhsu 4096 Sep 17 03:48 www
```

represents that the file *series* can be read, write, and executed by the owner *wwyhsu*, and can be read and executed but not written by the group *wwyhsu*, and can be read but not write and executed by anyone else on this system. The file *www* is a directory. For a directory, it can only be accessed if the execute **x** permission is on. For a permission `d--x------`, the owner of this directory can go into the directory, but can not list the content of the directory.

Changing the permission of files can be done by many different arguments with the command **chmod**. The **chmod** command is used to change the permissions of a file or directory. To use it, you specify the desired permission settings and the file or files that you wish to modify. It is easy to think of the permission settings as a series of bits (which is how the computer thinks about them). Here's how it works:

```
rwx rwx rwx = 111 111 111
rw- rw- rw- = 110 110 110
rwx --- --- = 111 000 000
and so on...
rwx = 111 in binary = 7
rw- = 110 in binary = 6
r-x = 101 in binary = 5
r-- = 100 in binary = 4
```

Now, if you represent each of the three sets of permissions (owner, group, and other) as a single digit, you have a pretty convenient way of expressing the possible permissions settings. For example, if we wanted to set `series` to have read, write and execute permission for the owner, read and execute for the group, and only execute for anyone else on the system, we would:

```
wwyhsu@WatchTower:$ chmod 751 series
wwyhsu@WatchTower:$ ls -l series
-rwxr-x--x 1 wwyhsu wwyhsu 8728 Oct 3 19:07 series
```

Here is a table of numbers that covers all the common settings. The ones beginning with **7**

are used with programs (since they enable execution) and the rest are for other kinds of files.

| Value | Meaning |
|---|---|
| 777 | `(rwxrwxrwx)` No restrictions on permissions. Anybody may do anything. Generally not a desirable setting. |
| 755 | `(rwxr-xr-x)` The file's owner may read, write, and execute the file. All others may read and execute the file. This setting is common for programs that are used by all users. |
| 700 | `(rwx------)` The file's owner may read, write, and execute the file. Nobody else has any rights. This setting is useful for programs that only the owner may use and must be kept private from others. |
| 666 | `(rw-rw-rw-)` All users may read and write the file. |
| 644 | `(rw-r--r--)` The owner may read and write a file, while all others may only read the file. A common setting for data files that everybody may read, but only the owner may change. |
| 600 | `(rw-------)` The owner may read and write a file. All others have no rights. A common setting for data files that the owner wants to keep private. |

The **chmod** command can also be used to control the access permissions for directories. In most ways, the permissions scheme for directories works the same way as they do with files. However, the execution permission is used in a different way. It provides control for access to file listing and other things. Here are some useful settings for directories:

| Value | Meaning |
|---|---|
| 777 | `(rwxrwxrwx)` No restrictions on permissions. Anybody may list files, create new files in the directory and delete files in the directory. Generally not a good setting. |
| 755 | `(rwxr-xr-x)` The directory owner has full access. All others may list the directory, but cannot create files nor delete them. This setting is common for directories that you wish to share with other users. |
| 700 | `(rwx------)` The directory owner has full access. Nobody else has any rights. This setting is useful for directories that only the owner may use and must be kept private from others. |

The other method for changing permissions is to use the *letter method*.

- Use a + or - (plus or minus sign) to add or remove permissions for a file respectively.
- Use an equals sign =, to specify new permissions and remove the old ones for the particular type of user(s).
- You can use **chmod** letter where the letters are: **a** (all), **u** (user), **g** (group) and **o** (everyone).

For example:

```
wwyhsu@WatchTower:$ chmod u+rw somefile
```

This would give the user read and write permission.

```
wwyhsu@WatchTower:$ chmod o-rwx somefile
```

This will remove read/write/execute permissions from other users (doesn't include users within your group).

```
wwyhsu@WatchTower:$ chmod a+r somefile
```

This will give everyone read permission for the file.

```
wwyhsu@WatchTower:$ chmod a=rx somefile
```

This would give everyone execute and read permission to the file, if anyone had write permission it would be removed.

Other permissions include the **sticky bit**, which makes the file un-deletable, and the **setuid** permission, which allows others to execute a program as the permission of the owner (which is a security hazard in used without care).

## 14.6 Elevating Permissions

It is often useful to become the superuser to perform important system administration tasks, but as you have been warned (and not just by me!), you should not stay logged on as the superuser. In most distributions, there is a program that can give you temporary access to the superuser's privileges. This program is called **su** (short for substitute user) and can be used in those cases when you need to be the superuser for a small number of tasks. To become the superuser, simply type the su command. You will be prompted for the superuser's password:

```
wwyhsu@WatchTower:$ su
Password: (Enter password here)
root@WatchTower:/home/wwyhsu#
```

In some distributions, most notably **Ubuntu**, an alternate method is used. Rather than using **su**, these systems employ the **sudo** command instead. With **sudo**, one or more users are granted superuser privileges on an as needed basis. To execute a command as the superuser, the desired command is simply preceeded with the **sudo** command. After the command is entered, the user is prompted for the user's password rather than the superuser's. The following example elevates your permission to the superuser and executes the `ls -l /boot` command with this new permission:

```
wwyhsu@WatchTower:$ sudo ls -l /boot
[sudo] password for wwyhsu: (Enter your password here)
total 91382
-rw-r--r-- 1 root root 1241960 Aug 12 03:58 abi-4.4.0-36-generic
-rw-r--r-- 1 root root 1242262 Sep  7 02:52 abi-4.4.0-38-generic
```

However, you may not do this on our clusters because we **will not** distribute superuser password. You can only do that on your own machine.

## 14.7 Listing Files

When you first login, your current working directory is your home directory. Your home directory has the same name as your user-name and it is where your personal files and subdirectories are saved. The **ls** command lists the contents of your current working directory. To find out what is in your home directory, type:

```
wwyhsu@WatchTower:$ ls
aaa       empty.c         helloworld.s       motd            qrSquare
aaa       empty.s         helloworldStatic   omp             qrSquare.c
answer    face.c          helloworldStatic2  ompExample.cc   recursiveCall
```

There may be no files visible in your home directory, in which case, the UNIX prompt will be returned. Alternatively, there may already be some files inserted by the System Administrator when your account was created. **ls** does not, in fact, cause all the files in your home directory to be listed, but only those ones whose name does not begin with a dot **'.'**. Files beginning with a dot **'.'** are known as hidden files and usually contain important program configuration information. They are hidden because you should not change them unless you are very familiar with Linux.

To list all files in your home directory including those whose names begin with a dot, type:

```
wwyhsu@WatchTower:$ ls -a
.            bitOp.c~     helloworldC       .joe_state      primeSeive.c
..           bitOp.py      helloworldC2      .landscape      primeSeive.c~
```

As you can see, **ls -a** lists files that are normally hidden.

A common way to use **ls** is to use the long format `ls -l`, which shows more information of the file, including permissions, owner, size, and creation date:

```
wwyhsu@WatchTower:$ ls -l
total 2276
-rw-rw-r-- 1 wwyhsu wwyhsu 48861 Sep 21 18:50 aaa
-rw-rw-r-- 1 wwyhsu wwyhsu 48860 Sep 21 18:45 aaa
-rw-rw-r-- 1 wwyhsu wwyhsu   542 Oct 25 15:59 answer
-rwxrwxr-x 1 wwyhsu wwyhsu  8760 Oct 25 16:31 a.out
...
```

Other common forms of using **ls** are:

| | |
|---|---|
| `ls -r` | List directory contents in reverse order. |
| `ls -lR` | List directory contents and recursively list subdirectory contents. |
| `ls -h` | List directory and display size in human readable form. |

Since users tend to mistype `ls` into `sl`, funny Linux commands have been installed into system as a sense of humor. Try typing `sl` and see what happens.

## 14.8 Directories

We will now make a subdirectory in your home directory to hold the files you will be creating and using in the course of this tutorial. To make a subdirectory called unixstuff in your current working directory type:

```
wwyhsu@WatchTower:$ mkdir testDir
wwyhsu@WatchTower:$ ls -l
total 4
drwxrwxr-x 2 wwyhsu wwyhsu 4096 Oct 30 02:54 testDir ...
```

The command **cd** directory means change the current working directory to 'directory'. The current working directory may be thought of as the directory you are in, i.e. your current position in the file-system tree. The assisting command **pwd** prints your current working directory. To change to the directory you have just made, type:

```
wwyhsu@WatchTower:$ pwd
/home/wwyhsu/
wwyhsu@WatchTower:$ cd testDir
wwyhsu@WatchTower:$ pwd
/home/wwyhsu/testDir
wwyhsu@WatchTower:$ ls -la
total 8
drwxrwxr-x 2 wwyhsu wwyhsu 4096 Oct 30 02:54 .
drwxrwxr-x 3 wwyhsu wwyhsu 4096 Oct 30 02:54 ..
```

As you can see, in the `testDir` directory (and in all other directories), there are two special directories called **'.'** and **'..'**. In Linux, **'.'** means the current directory, so typing **cd .** means stay where you are. This may not seem very useful at first, but using **'.'** as the name of the current directory will save a lot of typing. The parent directory is represented by **'..'**, so typing **cd ..** will take you one directory up the hierarchy (back to your home directory). Note that typing cd with **no argument** always returns you to your **home** directory. This is very useful if you are lost in the file system.

Home directories can also be referred to by the tilde ~ character. It can be used to specify paths starting at your home directory. So typing **ls ~** will take you back to your

home directory. For the previous example, the directory created can be reached by typing **ls ∼/testDir**.

The `tree` command shows the current directory structure (and file locations). It will list all files, subdirectories, and symbolic links within the directory.

```
wwyhsu@WatchTower:7mud $ tree

.
+--acpi
|   +-- events
|   |   +-- powerbtn
|   +-- powerbtn.sh
+-- adduser.conf
+-- alternatives
|   +-- awk -> /usr/bin/gawk
|   +-- awk.1.gz -> /usr/share/man/man1/gawk.1.gz
|   +-- builtins.7.gz -> /usr/share/man/man7/bash-builtins.7.gz
|   +-- c++ -> /usr/bin/g++
|   +-- c89 -> /usr/bin/c89-gcc
|   +-- c89.1.gz -> /usr/share/man/man1/c89-gcc.1.gz
|   +-- c99 -> /usr/bin/c99-gcc
...
...
+-- ypserv.securenets
+-- zsh_command_not_found

216 directories, 1853 files
```

## 14.9 Files

**cp file1 file2** is the command which makes a copy of **file1** in the current working directory and calls it **file2**. First, cd to your `testDir` directory. Then at the UNIX prompt, type, **cp /proc/version .** Remember, in Linux, the dot means the current directory. The above command means copy the file science.txt to the current directory, keeping the name the same. Use **ls** to check if your file is successfully copied.

To move a file from one place to another, use the **mv** command. **mv file1 file2** moves (or renames) **file1** to **file2**. This has the effect of moving rather than copying the file, so you end up with only one file rather than two. It can also be used to rename a file, by moving the file to the same directory, but giving it a different name.

Some other manipulation commands are listed in the following table:

| | |
|---|---|
| `mkdir Prog` | Creates a directory called `Prog`. |
| `rmdir Prog` | Removes a directory `Prog`. The directory must be empty. |
| `mv <source> <dest>` | Moves a source file to the destination. If the destination is a directory, it moves the file into that directory, otherwise, it renames the `<source>` file to `<dest>`. |
| `rename <source> <dest>` | Renames the `<source>` file to `<dest>`. |
| `cp <source> <dest>` | Copies a source file to the destination. |
| `rm <file>` | Deletes the `<file>`. |
| `rm -rf <dir>` | Recursively deletes the directory `<dir>` and everything contained in it. |
| `clear` | This will clear all text and leave you with a prompt. |
| `cat` | Display the contents of a file on the screen, i.e., `cat /proc/version`. |
| `less` | Writes the contents of a file onto the screen a page at a time, i.e., `less /proc/cpuinfo`. Press the `space-bar` if you want to see another page, and type `q` if you want to quit reading. |
| `head` | Writes the first 10 lines of a file to the screen, i.e., `head /proc/cpuinfo`. |
| `head -5` | Writes the first 5 lines of a file to the screen. |
| `tail` | Writes the last 10 lines of a file to the screen, i.e., `tail /proc/cpuinfo`. |
| `tail -5` | Writes the last 5 lines of a file to the screen. |
| `tail -f` | Follow the changes of a file and writes the result on the screen. |

`less` command can be used to search for phrases in a file. For example, to search through `/proc/cpuinfo` for the word 'Intel', type:
`wwyhsu@WatchTower:$` **`less /proc/cpuinfo`**
then, still in less, type a forward slash **'/'** followed by the word to search `/Intel`
As you can see, less finds and highlights the keyword. Type **'n'** to search for the next occurrence of the word.

`grep` is one of many standard UNIX utilities. It searches files for specified words or patterns. First clear the screen, then type `wwyhsu@WatchTower:$` **`grep Intel /proc/cpuinfo`**
As you can see, `grep` has printed out each line containing the word 'Intel'. The `grep` command is case sensitive; it distinguishes between Intel and intel. To ignore upper/lower case distinctions, use the `-i` option, i.e. type:
`wwyhsu@WatchTower:$` **`grep -i intel /proc/cpuinfo`**
To search for a phrase or pattern, you must enclose it in single quotes (the apostrophe symbol). For example to search for 'fpu vme', type
`wwyhsu@WatchTower:$` **`grep -i 'fpu vme' /proc/cpuinfo`**
Some of the other options of grep are:

- `-v` display those lines that do NOT match.
- `-n` precede each matching line with the line number.
- `-c` print only the total count of matched lines.

Don't forget, you can use more than one option at a time. For example, the number of lines without the words Intel or intel is
`wwyhsu@WatchTower:$` **`grep -ivc intel /proc/cpuinfo`**

A handy little utility is the `wc` command, short for word count. To do a word count on `/proc/cpuinfo`, type:
`wwyhsu@WatchTower:$` **`wc -w /proc/cpuinfo`**
To find out how many lines the file has, type:
`wwyhsu@WatchTower:$` **`wc -l /proc/cpuinfo`**

*Can you use `wc` and `grep` together to find home many CPUs does your machine have?*

## 14.10 Redirection

Most processes initiated by Linux commands write to the *standard output* (that is, they write to the terminal screen), and many take their input from the *standard input* (that is, they read it from the keyboard). There is also the *standard error*, where processes write their error messages, by default, to the terminal screen. We have already seen one use of the `cat` command to write the contents of a file to the screen. Now type `cat` without specifying a file to read

`wwyhsu@WatchTower:$` **cat**

Then type a few words on the keyboard and press the `Enter` key. Finally hold the `Ctrl` key down and press `d` (written as `^D` for short) to end the input. If you run the cat command without specifying a file to read, it reads the standard input (the keyboard), and on receiving the '*end of file*' (`^D`), copies it to the standard output (the screen). In Linux, we can redirect both the input and the output of commands.

We use the > symbol to redirect the output of a command. For example, to create a file called list1 containing a list of fruit, type:

`wwyhsu@WatchTower:$` **cat > list1**
**pear**
**banana**
**apple**
`^D` *(this means press **Ctrl** and **d** to stop.)*

What happens is the cat command reads the standard input (the keyboard) and the > redirects the output, which normally goes to the screen, into a file called list1 To read the contents of the file, type:

`wwyhsu@WatchTower:$` **cat list1**

*Using the above method, create another file called `list2` containing the following fruit: `orange, plum, mango, grapefruit`. Read the contents of `list2`.*

The form >> appends standard output to a file. So to add more items to the file `list1`, type:

`wwyhsu@WatchTower:$` **cat >> list1**
**peach**
**grape**
**orange**
`^D` *(this means press **Ctrl** and **d** to stop.)*

You should now have two files. One contains six fruit, the other contains four fruit. We will now use the `cat` command to join (concatenate) `list1` and `list2` into a new file called `biglist`. Type `wwyhsu@WatchTower:$` **cat list1 list2 > biglist**
What this is doing is reading the contents of `list1` and `list2` in turn, then output the text to the file `biglist`. To read the contents of the new file, type:

`wwyhsu@WatchTower:$` **cat biglist**

The command **sort** alphabetically or numerically sorts a list. Type

`wwyhsu@WatchTower:$` **sort**
**dog**
**cat**
**bird**
**ape**
`^D` *(this means press **Ctrl** and **d** to stop.)*

The output will be

```
wwyhsu@WatchTower:$ sort
ape
bird
cat
dog
```

We use the < symbol to redirect the input of a command. Using < you can redirect the input to come from a file rather than the keyboard. For example, to `sort` the list of fruit, type:
`wwyhsu@WatchTower:$` **sort < biglist**
and the sorted list will be output to the screen. Redirection of both direction can be used together: to output the sorted list to a file, type:
`wwyhsu@WatchTower:$` **sort < biglist > slist**

Linux can combine the standard output of one command to a standard input of a command using the pipe |. For example, we can list the content of the file `/proc/meminfo` and send it to the `sort` command and have all lines sorted by:
`wwyhsu@WatchTower:$` **cat /proc/meminfo | sort**
The pipe operator can also be used consecutively, for example, we can dump the content of `/proc/meminfo` and find lines containing the word 'Pages' and count the number of lines by:
`wwyhsu@WatchTower:$` **cat /proc/meminfo | grep Pages | sort**

## 14.11 WildCards

The character ***** is called a wildcard, and will match against none or more character(s) in a file (or directory) name. For example, in your directory, type
`wwyhsu@WatchTower:$` **ls list***
This will list all files in the current directory starting with `list****`. Try typing
`wwyhsu@WatchTower:$` **ls *list**
This will list all files in the current directory ending with `****list`.
The character **?** will match exactly one character. So `?ouse` will match files like **h**ouse and **m**ouse, but not grouse. Try typing
`wwyhsu@WatchTower:$` **list ?list**

## 14.12 Archiving

The Linux `tar` stands for tape archive, which is used by large number of Linux/Unix system administrators to deal with tape drives backup. The `tar` command used to rip a collection of files and directories into highly compressed archive file commonly called *tarball* or `tar`, `gzip` and `bzip` in Linux. Although Linux provides `zip` and `unzip` which is identical to `WinZIP`, it is more common for people to use `tar` and `gzip`, `bzip`, `bzip2`, or `pbzip2` (parallel version of `bzip2`) altogether to create compressed archive files and that can be moved easily from one disk to another disk or machine to machine.

The below example command will create a tar archive file `www.tar` for a directory `/home/wwyhsu/www` in current working directory.

```
wwyhsu@WatchTower:$ tar -cvf www.tar www
www/
www/HW5/
www/HW5/match2-link.html
www/HW5/match1-link.html
www/HW5/match5.html
www/HW5/match8.html
www/HW5/forward.gif
www/HW5/match6-top.html
...
www/HW5/help-sim-en.html
www/HW5/match2.html
wwyhsu@WatchTower:$
```

To create a compressed `gzip` archive file we use the option as **z**. Notice that the dash before the parameter list is removed.

```
wwyhsu@WatchTower:$ tar cvfz www.tar www
www/
www/HW5/
www/HW5/match2-link.html
www/HW5/match1-link.html
www/HW5/match5.html
www/HW5/match8.html
www/HW5/forward.gif
www/HW5/match6-top.html
...
www/HW5/help-sim-en.html
www/HW5/match2.html
wwyhsu@WatchTower:$ ls www.tar*
www.tar www.tar.gz
```

The `bz2` feature compress and create archive file less than the size of the `gzip`. The `bz2` compression takes more time to compress and decompress files as compared to `gzip` which takes less time. To create highly compressed tar file we use option as **j**. There is a parallel version of `bzip2` which we can supply the parameter **-I pbzip2** to let the `tar` program use an alternative compressor.

```
wwyhsu@WatchTower:$ tar cvfj www.tar www
www/
www/HW5/
www/HW5/match2-link.html
www/HW5/match1-link.html
www/HW5/match5.html
www/HW5/match8.html
www/HW5/forward.gif
www/HW5/match6-top.html
...
www/HW5/help-sim-en.html
www/HW5/match2.html
wwyhsu@WatchTower:$ ls www.tar*
www.tar www.tar.bz2 www.tar.gz
```

To **untar** or extract a tar file, just issue following command using option **x** (extract).

It can be used with both `gzip` and `bzip` archives.

```
wwyhsu@WatchTower:$ tar -xvf www.tar.bz2
www/
www/HW5/
www/HW5/match2-link.html
www/HW5/match1-link.html
www/HW5/match5.html
www/HW5/match8.html
www/HW5/forward.gif
www/HW5/match6-top.html
...
www/HW5/help-sim-en.html
www/HW5/match2.html
```

`tar` was designed to support tape devices (sequential devices) and so you may add files or directories into an archive using the option **r**. However, you can not add files into a compressed `tar` archive such as `tar.gz` or `tar.bz2`.

Linux provides a command `split` to chop files into smaller chunks for storage or network transmission. Let us break the file created in the last section into 5K chunks.

```
wwyhsu@WatchTower:$ split -b 5K www.tar.bz2 "www.tar.bz2.part"
wwyhsu@WatchTower:$ ls www.tar.bz2.part*
-rw-rw-r-- 1 wwyhsu wwyhsu 5120 Nov 19 15:58 www.tar.bz2.partaa
-rw-rw-r-- 1 wwyhsu wwyhsu 3983 Nov 19 15:58 www.tar.bz2.partab
```

After successfully splitting `tar` files or any large file in Linux, you can join the files using the `cat` command. Employing `cat` is the most efficient and reliable method of performing a joining operation. To join back all the blocks or tar files, we issue the command below:

```
wwyhsu@WatchTower:$ cat www.tar.bz2.part* > www.tar.bz2.joined
wwyhsu@WatchTower:$ ls *.bz2 www.tar.bz2.joined*
-rw-rw-r-- 1 wwyhsu wwyhsu 9103 Nov 19 15:44 www.tar.bz2
-rw-rw-r-- 1 wwyhsu wwyhsu 9103 Nov 19 16:01 www.tar.bz2.joined
```

## 14.13 Manuals and Help

There are on-line manuals which gives information about most commands. The manual pages tell you which options a particular command can take, and how each option modifies the behaviour of the command. Type **man** command to read the manual page for a particular command. For example, to find out more about the `wc` (word count) command, type:

`wwyhsu@WatchTower:$` **man wc**

Alternatively,

`wwyhsu@WatchTower:$` **whatis wc**

gives a one-line description of the command, but omits any information about options etc. When you are not sure of the exact name of a command,

`wwyhsu@WatchTower:$` **apropos <keyword>**

will give you the commands with keyword in their manual page header. For example, try typing

`wwyhsu@WatchTower:$` **apropos copy**

## 14.14 Processes and Jobs

A process is an executing program identified by a unique *PID* (process identifier). To see information about your processes, with their associated PID and status, type
`wwyhsu@WatchTower:$ ps`
A process may be in the foreground, in the background, or be suspended. In general the shell does not return the Linux prompt until the current process has finished executing. Some processes take a long time to run and hold up the terminal. Backgrounding a long process has the effect that the Linux prompt is returned immediately, and other tasks can be carried out while the original process continues executing.

To background a process, type an **&** at the end of the command line. For example, the command sleep waits a given number of seconds before continuing. For example,

```
wwyhsu@WatchTower:$ sleep 60
This will wait 10 seconds before returning the command prompt. Until the command prompt is returned, you can do nothing except wait.
wwyhsu@WatchTower:$ sleep 60 &
[1] 6259
```

The **&** runs the job in the background and returns the prompt straight away, allowing you do run other programs while waiting for that one to finish. The first line in the above example is typed in by the user; the next line, indicating job number and PID, is returned by the machine. The user is be notified of a job number (numbered from 1) enclosed in square brackets, together with a PID and is notified when a background process is finished. Backgrounding is useful for jobs which will take a long time to complete.

Foreground process in the previous example, i.e., the one without &, can also be backgrounded.

```
wwyhsu@WatchTower:$ sleep 600
You can suspend the process running in the foreground by typing ^Z, i.e.hold down the Ctrl key and type z. To put it in the background, type
wwyhsu@WatchTower:$ bg
```

When a process is running, backgrounded or suspended, it will be entered onto a list along with a job number. To examine this list, type

```
wwyhsu@WatchTower:$ sleep 100 &
wwyhsu@WatchTower:$ sleep 200 &
wwyhsu@WatchTower:$ sleep 300
^Z
wwyhsu@WatchTower:$ jobs
[1]  Running      sleep 100 &
[2]- Running      sleep 200 &
[3]+ Stopped      sleep 300
```

To restart (foreground) a suspended processes, type
`wwyhsu@WatchTower:$ fg` < *jobnumber* >
For example, to restart `sleep 300`, type
`wwyhsu@WatchTower:$ fg %3`
Typing **fg** with no job number foregrounds the last suspended process.

It is sometimes necessary to kill a process (for example, when an executing program is

in an infinite loop). To kill a job running in the foreground, type `^C`. For example, run

```
wwyhsu@WatchTower:$ sleep 300 &
wwyhsu@WatchTower:$ jobs
[1]+ Running      sleep 300 &
Kill the job label 1.
wwyhsu@WatchTower:$ kill %1
wwyhsu@WatchTower:$
[1]+ Terminated      sleep 300
```

Alternatively, processes can be killed by finding their process numbers (PIDs) and using kill PID number. PIDs can be found by using the **ps** command.

```
wwyhsu@WatchTower:$ sleep 10000 &
wwyhsu@WatchTower:$ ps
PID TTY          TIME CMD  304 pts/0    00:00:00 sleep  305 pts/0    00:00:00 ps 28103 pts/0    00:00:00 bash
Kill the job label with PId 28103.
wwyhsu@WatchTower:$ kill 28103
wwyhsu@WatchTower:$ ps
PID TTY          TIME CMD
304 pts/0    00:00:00 sleep
305 pts/0    00:00:00 ps
28103 pts/0    00:00:00 bash
If a process refuses to be killed, uses the -9 option, i.e. type
wwyhsu@WatchTower:$ kill -9 28103
wwyhsu@WatchTower:$ ps
PID TTY          TIME CMD
304 pts/0    00:00:00 sleep
305 pts/0    00:00:00 ps
```

## 14.15 Quotas and Free Space

All accounts are allocated a certain amount of disk space on the file system for their personal files. In our configuration, this space is 1G. If you go over your quota, you are given 7 days to remove excess files. To check your current quota and how much of it you have used, type
`wwyhsu@WatchTower:$ quota -v`

```
Disk quotas for user b00357133 (uid 1142):
     Filesystem  blocks   quota   limit   grace   files   quota   limit   grace
/dev/mapper/WatchTower--vg-root
                     0  1048576 1048576               0       0       0
      /dev/sdb1      64  1048576 1048576              13       0       0
```

The **df** command reports on the space left on the file system. For example, to find out how much space is left on the fileserver, type
`wwyhsu@WatchTower:$ df`

```
Filesystem                     1K-blocks     Used   Available Use% Mounted on
udev                            13358992        0    13358992   0% /dev
tmpfs                            2675404   262984     2412420  10% /run
/dev/mapper/WatchTower--vg-root  75979864  5868456    66228812   9% /
tmpfs                           13377004        0    13377004   0% /dev/shm
```

```
tmpfs                                  5120        0        5120   0% /run/lock
tmpfs                              13377004        0    13377004   0% / sys/ fs/
cgroup
/dev/sda1                            240972   100290      128241  44% /boot
/dev/sdb1                       12781261848 35056768 12102043648   1% /home
cgmfs                                  100        0         100   0% /run/cgmanager/
fs
tmpfs                                2675404        0     2675404   0% /run/user/
2000
```

The du command outputs the number of kilobyes used by each subdirectory. Useful if you have gone over quota and you want to find out which directory has the most files. In your home-directory, type

```
wwyhsu@WatchTower:$ du

12      ./www
8       ./.local/share/mc
12      ./.local/share
16      ./.local
...
8       ./.cache/update-manager-core
8       ./.cache/mc
20      ./.cache
48      ./motd
8       ./.ssh
2460    .

wwyhsu@WatchTower:$ du -s

2460    .
```

The **-s** flag will display only a summary (total size) and the * means all files and directories.

The amount of memory available can be shown using the command **free**.

```
wwyhsu@WatchTower:$ free

              total        used        free      shared  buff/cache   available
Mem:       26754008      152924    24714412      271988     1886672    25942780
Swap:      27258876           0    27258876
```

## 14.16 The GNU C/C++ Compiler

The GNU Compiler Collection (GCC) is a compiler system produced by the GNU Project supporting various programming languages. GCC is a key component of the GNU toolchain. As well as being the official compiler of the unfinished GNU operating system, GCC has been adopted as the standard compiler by most other modern Unix-like computer operating systems, including Linux, and the BSD family. Some compilers provided by GNU includes:

| Command | Description |
|---|---|
| gcc | GNU C compiler |
| g++ | GNU C++ compiler |
| gfortran | GNU fortran compiler |
| gdc | GNU D programming language |
| gobjc | GNU objective-C |
| gobjc++ | GNU objective-C++ |

To compile your program in Linux, use the following command:

| Command | Description |
|---|---|
| `gcc` *`filename`* | Compile your file using `GCC` and produce the default output executable file `a.out` |
| `gcc -o` *`runfile filename`* | Compile your file using `GCC` and produce the output executable file `runfile` |
| `g++ -O2 -o` *`runfile filename`* | Compile your file using `G++` and produce the output executable file `runfile`. Also perform code optimization level 2. |
| `g++ -g -ggdb-O2 -o` *`runfile filename`* | Compile your file using `G++` and produce the output executable file `runfile`. Also perform code optimization level 2. In addition, generate debugging information in your code for use with `gdb` (GNU Debugger). |

## 14.17 Text Editors

Something seems to be missing in our system. We need a way to edit the files! Our `ubuntu` should have provide `vim` (visual editor improved), but it is relatively hard to pick up for beginners. We shall install `joe` editor. It can be done by the following command:

```
sudo apt-get install joe
```

To edit a file, you type:

```
joe <filename>
```

Basic command for the `joe` editor are listed as follows:

| | |
|---|---|
| `Ctrl+K, H` | Displays the help menu. Type `Ctrl+K, H` again to hide the help menu. |
| `Ctrl+K, F` | Searches the text for a specific content. |
| `Ctrl+K, L` | Goes to certain line number. |
| `Ctrl+K, S` | Saves the file. |
| `Ctrl+K, X` | Exits the editor. |
| `Ctrl+Y` | Deletes a line. |

## 14.18 Fun Utilities

Linux OS is to be made as small as possible, and thus many utility programs are created to enhance the system. For example, `cal` program displays a simple, formatted calender from the command line. `wwyhsu@WatchTower:$ cal`

```
   November 2017
Su Mo Tu We Th Fr Sa
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30
```

`ncal` display the calender in another format. `wwyhsu@WatchTower:$ ncal`

```
    November 2017
Su     5 12 19 26
```

```
Mo     6 13 20 27
Tu     7 14 21 28
We  1  8 15 22 29
Th  2  9 16 23 30
Fr  3 10 17 24
Sa  4 11 18 25
```

By giving parameters to these programs, a yearly calender can be shown.

## 14.19 Lab Questions

1. Create a directory named `www`.
2. Change the permission of the directory `www` to `drwxr-x--x`, and explain what does this mean.
3. From the previous exercises, upload your personal web page to the `www` directory. Use `http://linux.deepsea9.taipei/~youraccount` to view the page.
4. Copy the file `/proc/cpuinfo` to your directory, and rename the file to `CPU` (case sensitive).
5. Dump (show) the content of the file `CPU` and isolate (using Linux commands) the CPU model.
6. Count the disk usage of your `/www` directory.
7. Write a program to find the prime numbers between 1 and 10000 and compile it using **gcc** or **g++**.
8. Execute the file and redirect the output to the file `primes`.