UNIX

- **UNIX** is an operating system (OS) for multi-user systems.

- **UNIX** was originally develop at Bell Labs in 1969 by Thompson and Ritchie.

- **UNIX** is an open system. (No one company owns it.)

- **UNIX** is portable - it can be installed on different types of machines.

- **Main Components of the UNIX OS:**
  - **Kernel:**
    Master control program of the computer. In charge of managing resources and multitasking.
  - **File System:**
    Data are stored in files, which are organized in directories.
  - **Shell:**
    Accepts user commands and passes them on to the kernel.
  - **Utilities:**
    These are the commands that UNIX understands.
    %date
    %who
    %whoami
    %mail
    %pine (see chapter 20)
    %man

- In summary, you type a **utility** into the UNIX **shell**, which gets passed to the **kernel** for processing.

The UNIX File System (chapter 6)

- The File System is used to organize data.

- In UNIX, anything from which data can be taken from or sent to, is a file; for example,
  - an ordinary file on a disk (text or binary)
  - printer
  - keyboard

- A **directory** in UNIX corresponds to a **folder** in Windows.

- When you are logged into your UNIX account, you are always in some directory. This is called the **current directory** (%pwd)

- Your **home directory** is the default directory you are in when you log in.

- Directories has a hierarchial structure represented by a **Directory Tree**.

- A **pathname** is the address of a file or directory. It is the path of the file (or directory) in the directory tree.

- An **absolute pathname** is the path from the root to the specified file (or directory).
  Example: **/users3/ziegler/myfile**
  Example: **/users3/ziegler/pgm1/mypgm.cpp**
  Example: **/users3/ziegler/public_html/index.html**
  (Note: the first / represents the root)

- A **relative pathname** is the path is the path in the directory tree beginning from the current directory
  Example: **myfile**
  Example: **pgm1/mypgm.cpp**
  Example: **public_html/index.html**

- In UNIX, **file names are case sensitive.** Do not use spaces or special characters in your file names (. and _ are allowed)

Shell Utilities for File Management

%pwd	//print working (current) directory
%cd *pathname*	//change current directory
%cd	//change to home directory
%ls	//lists files in current directory
%ls -l	//detailed list
%ls -la	//detailed list including hidden files
%cat *files*	//print files to stdout
%more *files*	//file perusal filter (see manual)
%less *files*	//file perusal filter (see manual)
%lpr *files*	//print files
%rm *files*	//remove (delete) a file
%mkdir *pathname*	//create a directory
%rmdir *pathname*	//remove an empty directory
%cp *file1 file2*	//copy file1 to file2
%mv *file1 file2*	//move (or rename) file1 to file2
%chmod *perm pathname*	//change (permissions) mode
%ps	//print process status and PIDs

- **Creating Files:**
  - **copy or move another file:**
    cp *file1 file2* //copy file1 to file2
    mv *file1 file2* //move (or rename) file1 to file2

  - **redirect standard output (funnel output to a file):**
    ls > myfile // > erases file if it exists
    ls >> myfile // >> appends to end of file if it exists

  - **text editor:**
    emacs
    vi
    pico
    nedit //GUI editor

  - **computer program:** (see chapter 12)
    e.g., out of a C++ program

- **Permissions:**
  The permissions of each file and directory can be viewed (using ls -l) as a sequence of 10 bits.
  drwxrwxrwx (owner|group|public)

  d - directory
  r - read permission
  w - write permission
  x - execute permission

- **Changing Permissions:** (use the **chmod** command)
  chmod 711 myfile
  chmod 600 myfile //only the owner has access rights
  chmod 644 myfile
  chmod a=rx myfile //read and execute permission for all
  chmod a+r myfile //add read permission to all

Pipes

- You can send the output of one utility to the input of another utility using a pipe|
  Example:
  ```
  who | sort
  cat myfile | more
  ```

- Note:
  Redirection > sends to a file.
  Piping | sends to another command.

The grep Command

- grep will search for a word inside a file or a directory
  Example:

grep users myfile	//searches file myfile for words containing “users”
grep m myfile	//searches file myfile for words containing an “m”
grep users m*	//search all files that start with m for words containing “users”

Processes (chapter 11)

- Since UNIX is a multitasking OS, a user can run several processes at once.

- You can run each process in a different window.

- You can run processes in the background:

```
%mozilla&       //firefox will run in the background
                //the shell prompt will return to the window
%xterm&         //a background terminal window
%nedit&         //nedit will run in the background
```

- %ps
  prints the list of currently running processes and their PID

- jobs
  lists jobs current running

- kill *PID* (or kill %*jobnumber*)
  terminates process PID

- CNTL-C
  terminates a foreground process

- CNTL-Z
  suspends a foreground process

- fg *PID* (or fg %*jobnumber*)
  brings process PID to the foreground

- bg *PID* (or bg %*jobnumber*)
  sends process PID to the background

C++ Programming under UNIX

- **Compiling:**

```
g++ -c mypgm.cpp    //this creates an object file mypgm.o
```

- **Linking:**

```
g++ mypgm.o         //this creates an executable file a.out
```

- **Running the Program:**

```
a.out               //this runs the executable file if current
                    //directory is in the search path
./a.out             //runs the executable if the current
                    //directory is not in the search path
```

- **Compiling and Linking in One Step:**

```
g++ mypgm.cpp       //this creates an executable file a.out
```

- **Renaming the Executatble File:**

```
mv a.out mypgm      //renames the file
mypgm               //runs the program
```

- **Compiling and Linking - setting the name of the executable file:**

```
g++ -o mypgm mypgm.cpp      //this creates an
                            //executable file  mypgm
```

Multifile Projects under UNIX

The make Utility

- The **make** utility looks for its instructions in a file name **makefile**.

- **Example - Simple makefile:**

```
#makefile for Bank Accounts program      //comment

bank: BankAccounts.cpp                   //dependency line
   g++ -o bank BankAccounts.cpp          //action line
```

- **Example - Multifile makefile:**

```
#makefile for Contestant Database program      //comment

quiz: ContestantDatabase.o Contestant.o Name.o Job_info.o Personal_info.o
   g++ -o quiz ContestantDatabase.o Contestant.o Name.o Job_info.o Personal_info.o

ContestantDatabase.o: ContestantDatabase.cpp    //dependency line
   g++ -c ContestantDatabase.cpp                //action line

Contestant.o: Contestant.cpp                    //dependency line
   g++ -c Contestant.cpp                        //action line

Name.o: Name.cpp                                //dependency line
   g++ -c Name.cpp                              //action line

Job_info.o: Job_info.cpp                        //dependency line
   g++ -c Job_info.cpp                          //action line

Personal_info.o: Personal_info.cpp              //dependency line
   g++ -c Personal_info.cpp                     //action line

clean:                          //clean: does not depend on any file
   rm ./*.o                     //action for "make clean" command
```