

**PHYS-4007/5007: Computational Physics**  
**Course Lecture Notes**  
**Section I**

Dr. Donald G. Luttermoser  
East Tennessee State University

**Version 5.0**

## **Abstract**

These class notes are designed for use of the instructor and students of the course **PHYS-4007/5007: Computational Physics** taught by Dr. Donald Luttermoser at East Tennessee State University.

# I. Choosing Hardware and an Operating System

## A. Introduction.

1. With the widespread availability of fast computers at affordable prices, it has become important to understand the capabilities and details of numerical modeling.
2. In physics and astronomy, many talk about four different classes of computer hardware. From most to least expensive they are:
  - a) **Super Computers:** A mainframe with many 64-bit CPU chips — **vector processors**. Cray used to be the biggest maker of super computers, but they have now gone out of business due to the speed and low-cost of the PCs. There are still a few Crays in operation around the country. Since the early 21st century, small “cluster” machines with multi-CPU chips have taken the place of “supercomputers” due to their much lower cost.
  - b) **Mainframes:** These were larger computers which allowed for multi-user use from remote locations. These machines all had multi-user operating system running the architecture. Mainframes in the 1960s and 1970s typically had 32-bit CPU chips and machines in the 1970s and 1980s contained 64-bit CPU chips. The old VAXes (various models, 750, 780, 8600, etc.) and the IBM 360 are two examples of mainframes. These machines are now virtually extinct due to the advent of workstations and fast PCs.
  - c) **Workstations:** A machine that runs a *multi-user* operating system like Unix and its variants. They are essentially

scaled down versions of the older mainframe computers. They typically have a 64-bit CPU processor (or series of processors), though some older machines (pre-1995) only have 32-bit chips in them. On these machines, numerous users can operate the machine remotely (and at the console) at the *same time*. Examples of such machines are the Sun workstations, AlphaStations (originally made by the now defunct Digital Equipment Company), and Silicon Graphics workstations. Note that the PC world has been calling machines running various flavors of Microsoft Windows “workstations,” but are not from the definition above, since they are not designed for simultaneous, real-time, multi-users use.

- d) Personal Computers (PCs):** Typically a 32-bit (though 64-bit processors are now becoming more common) CPU processor (or series of processors) running an *operating system* (OS) of either some version of Microsoft Windows, Linux (with X-Windows), or Mac OS (on the Macintoshes). These machines are designed for *single-user* mode (*i.e.*, one at a time).
- 3.** The *size* of the CPU can be as important as the *clock speed* of the CPU.
- a)** Early PCs only had only an 8-bit CPU chip. They evolved to 16-bit, then 32-bit, and now 64-bit.
  - b)** There are two reasons why the size of the CPU is important.

- i) The maximum and minimum size of a number is limited by the *chip size*.
  - ii) The amount of precision possible for a number is also limited by the size (though some compilers are sophisticated enough to increase precision for a given chip beyond what the chip can normally provide).
- 4. Computers work with numbers (and any character for that matter) in **binary format**: Two **bits** (= Binary digIT),  $0 \equiv \textit{off}$  and  $1 \equiv \textit{on}$ .
  - a) There are  $2^N$  integers that can be represented with  $N$  bits.
  - b) The sign of the integer is handled with the first bit ( $0 \equiv$  positive number), which leaves  $N - 1$  bits to represent the value of the integer.
  - c) Therefore,  $N$ -bit integers will be in the range (absolute value-wise) of  $[0 : 2^{N-1}]$ .
    - i) Hence an 8-bit machine can handle integers in the range  $[-128 : 127]$ .
    - ii) A 16-bit machine can handle integers in the range  $[-32,768 : 32,767]$ .
    - iii) A 32-bit machine can handle integers in the range  $[-2,147,483,648 : 2,147,483,647]$ .
    - iv) Finally, a 64-bit machine can handle integers in the range  $[-9.223372... \times 10^{18} : 9.223372... \times 10^{18}]$ . Note I am using ‘real’ notation here due to the

large size of the number, on a computer, such a number would have no decimal point. Also note that in computer programming languages, numbers in scientific notation use the “E” (“D” for double precision – see below) notation:  $[-9.223372\text{E}18 : 9.223372\text{E}18]$ .

- v) Calculating (or expressing) integers that are smaller (if negative) or larger (if positive) than the given chip size would result in an *underflow* (negative) or *overflow* of the computer register (and typically crash one’s program)  $\implies$  the bigger the chip, the better it is for numerical work.
  
- d) Since binary strings of numbers are not easy for people to work with, a **compiler** is used to translate the binary numbers of the machine to either *octal* (base 8, instead of base 2), *decimal* (base 10, what we normally are used to), or *hexadecimal* (base 16 numbers).
  
- e) English letters and punctuation marks are processed on many computers as an 8-bit number called **ASCII** (American Standard Code for Information Interchange) format (note that the first “sign” bit is not used in ASCII protocol). Each character on the keyboard (including their capital representations) has an ASCII-value associated with it ranging from 0 to 127 (*e.g.*, ‘0’ (zero) has an ASCII value of 48, ‘A’ (capital-a) = 65, and ‘a’ = 97). Many of the ASCII code numbers between 0 and 127 are non-printable control characters. I will often refer to “text” (\*.txt) files (in the Microsoft world), that is, those files that can be printed or viewed with an editor, as ‘ASCII’ files (as they are called in the Unix/Linux world).

5. Numbers and character strings are stored on computers in **words**, where the *word length* is often expressed in **bytes**:

$$1 \text{ byte (1 B)} \equiv 8 \text{ bits (8 b)}. \quad (\text{I-1})$$

- a) Conventionally, storage size is measured in bytes (or kilobytes [KB], megabytes [MB], and gigabytes [GB]).
- b) However, here “kilo” does not mean 1000, instead it is equal to

$$1 \text{ KB (1 K)} = 2^{10} \text{ bytes} = 1024 \text{ bytes}. \quad (\text{I-2})$$

- c) In the past, many machines measure their memory size in units of 1/2-kilobytes called **blocks**, or more precisely

$$512 \text{ B} = 2^9 \text{ bytes} = 512 \text{ bytes}. \quad (\text{I-3})$$

- d) As mentioned above for ASCII format, 1 byte is the amount of memory required to store a single character. This adds up to a typical typed page requiring  $\sim 3$  KB.

6. **Data types** on computers: There are two basic types of data that are operated on and stored by computers  $\implies$  **integers** and **real numbers**. Depending on the programming language one is using, there are various types of “integers” and “reals.”

- a) **Integers** are stored as

$$I = (-1)^s \times \left( \sum_{n=1}^{N-1} \alpha_n 2^{n-1} \right), \quad (\text{I-4})$$

where ‘ $s$ ’ is the sign bit ( $0 \equiv$  positive,  $1 \equiv$  negative) and  $\alpha_n$  takes on either a ‘1’ value (the bit is set) or ‘0’ value (the bit is not set). For example, on an 8-bit machine ( $N = 8$ ), we can write  $-57$  as

$$\begin{aligned} & (\text{sign bit set to } 1) \times \left[ (1 \times 2^0) + (0 \times 2^1) + (0 \times 2^2) \right. \\ & \quad \left. + (1 \times 2^3) + (1 \times 2^4) + (1 \times 2^5) + (0 \times 2^6) \right] \\ & = (-) \ 1 + 0 + 0 + 8 + 16 + 32 + 0 = -57, \end{aligned}$$

or in binary format (where the bits are listed in the opposite-order of the summation above), that is, the least significant digit ( $2^0$ ) is recorded on the far right side of the binary number (just as it is in decimal notation):

$$-57 = 1\ 011\ 1001,$$

where the first bit is the sign bit (1 = negative).

- b) Integers can come in a variety of flavors in various programming languages:
- i) **Logical:** 1-bit word, values = [.FALSE. (=0) : .TRUE. (=1)], maximum value = 1.
  - ii) **Short Integer:** 8-bit or one-byte word length, range = [-128 : 127], maximum value =  $2^7 - 1$ , number of digits = 3.
  - iii) **Character:** Like a short integer typically containing the ASCII code of the character, however, the programmer in a higher-level language (like **Fortran** or **IDL**) uses the **string** notation (characters surrounded by single- and/or double-quotation marks) and the compiler changes these characters to the short integer ASCII code, then to binary, to which the machine then operates.
  - iv) **Integer:** 16-bit or two-byte word length, range = [-32,768 : 32,767], maximum value =  $2^{15} - 1$ , number of digits = 5.
  - v) **Long Integer** (sometimes just called ‘Long’ or ‘Double-Precision Integer’): 32-bit or four-byte word length, range = [-2,147,483,648 : 2,147,483,647],

maximum value =  $2^{31} - 1$ , number of digits = 10.

**vi) Double-Long Integer** (sometimes just called ‘Double Long’ or ‘Quad-Precision Integer’): 64-bit or eight-byte word length, range =  $[-9.22\text{E}18 : 9.22\text{E}18]$ , maximum value =  $2^{63} - 1$ , number of digits = 20. Note that these types of integers are only found on 64-bit architectures, and even then, only found in some programming languages (*e.g.*, Intel Fortran which is based on the old DEC Fortran compilers).

- c) Unlike integers, **real** numbers are indicated with a decimal points located in the number.
- d) The computer can handle real numbers in two different ways: *Fixed-point* (not to be confused with ‘fixed’ or ‘integer’ data types) and *floating-point* notation.
  - i) In fixed-point real notation, the number  $x$  is represented as

$$x_{\text{fix}} = \text{sign} \times (\alpha_n 2^n + \alpha_{n-1} 2^{n-1} + \dots + \alpha_0 2^0 + \dots + \alpha_{-m} 2^{-m}), \quad (\text{I-5})$$

(note that the textbook uses  $I_{\text{fix}}$  for this type of number).

- ii) The first bit is used to store the sign of the number and the remaining  $N - 1$  bits are used to store the  $\alpha_i$  values such that  $n + m = N - 2$ , where  $N$  is the total number of bits used to represent a number. The particular values for  $N$ ,  $m$ , and  $n$  are machine dependent. For instance, the number 9.875 could

be represented in real fixed-point binary as

$$\begin{aligned} 0 \ 1001111 &\implies (\text{sign bit}) \times [(1 \times 2^3) + (0 \times 2^2) \\ &\quad + (0 \times 2^1) + (1 \times 2^0) + (1 \times 2^{-1}) + (1 \times 2^{-2}) \\ &\quad + (1 \times 2^{-3})] = 8 + 0 + 0 + 1 + 0.5 + 0.25 + 0.125 \\ &= 9.875 \end{aligned}$$

on an 8-bit machine.

- iii) All fixed-point real numbers have the same absolute error of  $2^{-m-1}$  [the term left off the right-hand side of Eq. (I-5)].
- iv) The correspondingly disadvantage is that *small* numbers (those which the first string of  $\alpha$  values are zeros) have large *relative* errors.
- v) Relative errors tend to be more important than absolute errors (we will be covering errors in more detail in §V of these notes), fixed-point real numbers are used mainly in special applications (like business), but typically not used in science and engineering.
- e) In scientific work, the programming language compilers (like Fortran and IDL) use floating-point numbers for reals (as such, reals are often referred to as ‘floats’ in some languages).
  - i) In floating-point notation, the number  $x$  is stored as a sign, a mantissa, and an exponential field (*expfld* in Eq. I-6).
  - ii) The number is reconstituted as

$$x_{\text{float}} = (-1)^s \times \text{mantissa} \times 2^{(\text{expfld} - \text{bias})}, \quad (\text{I-6})$$

(note that the book uses a slightly different form for this equation). The mantissa contains the significant figures of the number,  $s$  is the sign bit (still the first bit of the binary number), and the actual exponent of the number has a *bias* added to it.

- Since we have a sign bit, the mantissa will always be positive.
- The bias guarantees that the number stored as the exponent field is always positive (of course the actual exponent can be negative).
- The use of the bias is rather indirect. For example, a single-precision 32-bit word may use 8-bits for the exponent in Eq. (I-6), and represent it as an integer. This 8-bit integer “exponent” has a range of [0:255]. Numbers with actual negative exponents are represented by a bias equal to 127, a fixed number for a given machine. Consequently, the exponent has the range [-127:128] even though the value stored in the exponent in Eq. (I-6) is a positive number.
- Of the remaining bits, one is use for the sign and the other 23 for the mantissa, where

$$\begin{aligned} \text{mantissa} = & (m_1 \times 2^{-1}) + (m_2 \times 2^{-2}) + \\ & \cdots + (m_{23} \times 2^{-23}), \end{aligned} \quad (\text{I-7})$$

with the  $m_i$  stored liked the  $\alpha_i$  in Eq. (I-5).

- As an example, the number 0.5 is stored as

0 0111 1111 1000 0000 0000 0000 0000 000

on a 32-bit machine, where the bias is  $0111\ 1111_2 = 127_{10}$ .

- iii) Typically, the largest possible floating-point number for a 32-bit machine is

0 1111 1111 1111 1111 1111 1111 1111 111 ,

which has the value 1 for all its bits (except the sign) and adds up to  $2^{128} = 3.4 \times 10^{38}$ . Whereas the smallest possible floating-point number for a 32-bit machine is

0 0000 0000 1000 0000 0000 0000 0000 000 ,

which has the value 0 for almost all the bits and adds up to  $2^{-128} = 2.9 \times 10^{-39}$ . As built in by the use of the bias, the smallest number possible to store is the inverse of the largest.

- f) Just as was the case for integers, reals come in a variety of “flavors” in various programming languages. For example:

i) **Single-Precision Real** (sometimes called *Floats* or *Real\*4* numbers): 32-bit = 4-byte word-size numbers, 6-7 decimal places of precision (1 part in  $2^{23}$ ), and a range =  $[1.17549435 \times 10^{-38} : 3.40282347 \times 10^{38}]$ .

ii) **Double-Precision Real** (sometimes called just *Double Precision* or *Real\*8*): 64-bit (2 32-bit words  $\rightarrow$  11 bits used for the exponent, 1 for the sign, and 52 bits for the mantissa) = 8-byte word-size numbers, about 16 decimal places of precision (1 part in  $2^{52}$ ), and a range =  $[2.2250738585072014 \times 10^{-308} : 1.7976931348623157 \times 10^{308}]$ .

iii) **Quad-Precision Real** (*Real\*16*): 128-bit (4 32-bit words  $\rightarrow$  15 bits used for the exponent, 1

for the sign, and 112 bits for the mantissa) = 16-byte word-size numbers, about 36 decimal places of precision (1 part in  $2^{112}$ ), and a range = [ $\sim 3.362 \times 10^{-4932} : \sim 1.189 \times 10^{4932}$ ]. Currently this precision is only available on 64-bit Sun workstations Fortran compilers.

- iv) **Complex Numbers:** Some programming languages (in particular, Fortran and IDL) have complex data types. A complex number has the form

$$z = x + iy , \quad (\text{I-8})$$

where  $i \equiv \sqrt{-1}$  and  $x$  corresponds to the real part of the number and  $y$  to the imaginary part. These numbers are stored as two-element arrays (real, imaginary) of real numbers.

## B. The Inner Workings of a Computer.

1. A computer does mathematics (which is the same thing as running programs, whether your program is designed to math problems or not) via **machine language**.
  - a) Machine language is built into the CPU of the machines — each CPU family of chips has its own specific machine language or *architecture*, where machine language manipulates numbers on the “bit” level. Coding is written in machine language which controls the CPU registers.
    - i) The old 32-bit Intel Pentium chips used one type of machine language, primarily based on the *CISC* (Complex Instruction Set Computer). Such a processor uses small code sizes carrying out instructions over multi-clock cycles. In this architecture, transistors on a chip are used for storing complex instructions.

- ii) Meanwhile the 64-bit chip “workstation-class” machines used their own machine language based on the *RISC* (Reduced Instruction Set Computer) architecture. Such a processor uses large code sizes carrying out instructions over a single clock cycle. This architecture spends more transistors on memory registers than the CISC chips.
  - iii) Newer chip architectures have been developed in the recent past especially for *cluster* machines with multiple CPUs, such as the *massive parallel processing* architecture.
- b) Operating systems are designed to communicate with the machine language of the chip. As such, a discussion of computer hardware needs to include a discussion of operating systems. However, first we need to define a few terms about the inner workings of a computer. (Refer to Chapter 13 in your textbook.)
- i) **CPU:** The **central processing unit** is the fastest part of the computer. The CPU consists of a number a very high-speed memory units called *registers*, containing the *instructions* sent to the hardware to do things like fetch, store, and operate on data.
  - ii) **FPU:** The **floating point** (or arithmetic) **unit** is a piece of hardware designed for the quick operation of floating-point arithmetic. On machines in the past, these chips were called *math coprocessors*.

- iii) **Cache:** A small, very fast bit of memory (sometimes called the *high-speed buffer*) that holds instructions, addresses, and data in their passage between the very fast CPU registers and the slower main RAM memory. The main memory is also called *dynamic* RAM (DRAM), while the cache is *static* RAM (SRAM).
  
- iv) **Cache and data lines:** The data transferred to and from the cache or CPU are grouped into *cache lines* or *data lines*. The time it takes to bring data from memory into cache is called *latency*.
  
- v) **RAM: Random access memory** or central memory is in the middle memory hierarchy. This is where your program resides while it is being processed. The contents of RAM is lost upon completion of the jobs (or the turning off of the machine). The RAM of your computer is analogous to the memory centers of your brain.
  
- vi) **ROM: Read only memory** contains data that is hard-coded on the chip (typically, non-erasable). Information on this chip tells the machine its identity, senses the devices hooked to the motherboard (called *peripherals*), and tells the machine where to find the boot software. The ROM is analogous to your DNA.
  
- vii) **Pages:** Central memory is organized into **pages**, which are blocks of memory of fixed length. The operating system labels and organizes its memory

pages much like we do with the pages of a book  $\implies$  they are numbered and kept track of in a *table of contents*.

- viii) **Hard disk:** Finally, at the bottom of the memory hierarchy is the permanent storage on magnetic disks or optical devices. They are slow, but can store large amounts of data. The coding of the operating system, compilers, and any other documentation, whether in ASCII format or binary format is stored here.
  
- ix) **Backup devices:** Since hard disks work very hard, they are the first thing that is likely to go bad on a computer. Once the *head* crashes on a disk drive, the data stored there is typically lost forever. As such, it is good practice to store the contents of your hard drive (*i.e.*, hard disk) on a more permanent and safe medium: **USB Flash Drives**, **CD-ROMs**, and writable **DVDs**.
  
- x) **Virtual memory:** Virtual memory permits your program to use more pages of memory than will physically fit into RAM at one time. Pages not currently in use are stored in slower memory (typically hard disks in a region called **swap space**) and brought into fast memory only when needed. The amount of memory space needed for a program to run is limited by the machine's RAM plus the swap space size. In **Unix**, the amount of swap space available and total can be found with the "swap" (on some machines called "swapon") command.

2. The speed of the CPU is determined by the clock chip that is attached to it. The faster the clock (measured in MHz or GHz), the faster your CPU.
3. CPU clock speed doesn't tell the whole story in computer speed. The speed of the cache and the speed of the *bus* that talks to the hard disk (and other peripherals) are also important in “turn-around-time” in running a program.

### C. A Brief History of Operating Systems.

1. A simple definition of a operating system is the suite of programs that make the hardware usable. The operating system manages the CPU, disks, and I/O devices.
2. Manipulation of the operating system was typically one of the hardest aspects of learning to use computers which is why the “GUI (Graphic User Interface) mentality” has taken hold of the modern computers — the user no long needs to talk to the operating system, the GUI does it for you! (Which is actually a “pain-in-the-butt” if you need to actually talk to the operating system!)
3. We now follow the history of operating systems and programming languages that are important in the scientific community, starting in the 1940s:
  - a) Early computers had no operating system (*e.g.*, the IBM 604).
  - b) The computer was told what to do with a low-level set of commands  $\implies$  **assembly language**.

- c) The IBM 604 could undertake 60 program steps before using punch cards as a backing store.
  - d) Often, the user had to manipulate toggle switches to input the code for the mainframe.
4. The 1950s saw the advent and rapid changes in the capability of operating systems.
- a) Jobs were *batched* so that the time between jobs was minimized.
  - b) The user was now distanced from the mainframe and entered a program via a card reader.
  - c) This era saw the rapid development in higher-level programming languages such as **Algol 60** and the first version of **Fortran**.
5. The 1960s saw the advent of multiprogramming and the concept of *time-sharing*.
- a) *Multi-user* operating systems were developed which allowed users to actually log into the computer simultaneously.
  - b) Since scientists were the primary users of mainframes, an official programming language for the sciences was declared: **Fortran IV** (which was also called **Fortran 66**).
6. The 1970s saw numerous multi-user operating systems come on line. Two of the most successful were **Unix** and **VMS** (Virtual Memory System).
- a) **Unix** developed by **Bell labs** in 1970 and further modified by the **University of California at Berkeley**.

- b) Unix is written in the C programming language, is capable on running on a variety of different architectures.
  - c) VMS was released in 1978 by the Digital Equipment Corporation (DEC) to run on their VAX mainframe computers.
  - d) Whereas Unix commands are often cryptic, VMS commands are based on English words, and hence was considered much more “user-friendly” than Unix. As such, it was typically the preferred OS by scientists during this decade, whereas, Unix was favored by the computer “scientists.”
  - e) The 1970s also saw a new Fortran standard come on line  $\implies$  Fortran 77. This version of Fortran was much more versatile and powerful than the earlier version of this programming language.
  - f) Finally, the 1970s saw the beginnings of the PC market open up through the release of the Apple IIe and IBM PC (running DOS) computers.
7. The 1980s saw an explosion of computers in science through the release of the *microcomputer* (which lasted only a few years) and *workstation*. Both of these types of machines were scaled down versions of mainframes, with the workstations being designed to fit on a desk.
- a) The most popular workstations at that time was the VAX-station, running VMS, and the Sun workstation, running Sun OS.
  - b) During this decade, Unix’s popularity grew leaps and bounds, whereas, VMS declined.

- c) Later this decade, DEC came out with their version of Unix to run on the VAX chip called *Ultrix*. Machines that had *Ultrix* installed on it were called DECstations.
  - d) Apple came out with a new 32-bit chip that they placed in a new machine called the Macintosh.
8. In the 1990s, there was a PC explosion which left the workstations in the dust.
- a) Microsoft began to dominate this market with their various flavors of the Windows OS (note that Windows originally was a GUI front end that sat on top of DOS). 16-bit PCs were replaced by 32-bit PCs during this decade. PCs started becoming very fast when the Intel Pentium chip was developed.
  - b) DEC came out with a new 64-bit chip (the VAX chip was 32-bit) called the *Alpha* which gave rise to the *Alpha-Stations*. This chip required new versions of DEC's OSs: VMS → OpenVMS, and *Ultrix* → OSF/1 (which later was renamed Digital Unix in 1998, then Tru64 Unix in 2000 when Compaq bought out DEC).
  - c) Sun then came out with the Sparc-station (32-bit CPU) and a new OS called Solaris to run on it.
  - d) This decade saw the last of the mainframes, as the workstation and PC explosion made this type of computing obsolete. Because of this, and the fact that their *Alpha-Stations* were not as popular as their *VAXstations*, DEC went “belly-up” at the end of this decade.

- e) Finally, this decade saw the emergence of a “free” version of Unix called Linux which was designed to run on PC-class machines.
9. In the 2000s, the bulk of computing is done by PCs. In the physical science community, most of the number crunching is still performed on workstations, but many scientists are now switching to PCs due their fast speeds and low costs.
- a) Sun came out with a new 64-bit chip and workstation called the Ultra stations, followed by the multi-processor Blade computers.
  - b) For workstations, the biggest sellers were Suns, Silicon Graphics, and IBM Workstations. By the end of this decade, these workstations gradually became less popular mainly due to their high cost.
  - c) In 2002, Hewlett Packard bought Compaq and announced that they were dropping the AlphaStation from their line meaning the death of both Tru64 Unix and OpenVMS. This is a real pity since Tru64 Unix was the most secure version of Unix on the market. (Note that OpenVMS was also a very secure operating system.)
  - d) In 2003, Apple came out with its 64-bit, dual processor, Power Mac G5, running Mac OS X which is a Unix-based operating system.
  - e) In 2004, the Athlon 64 chip by Advanced Micro Devices (AMD) and the Intel Xeon 64-bit chip have appeared in PCs. These machines run either the 64-bit version of Microsoft Windows or the 64-bit version of Linux.

- f) Intel continuously comes up with faster and faster chips. Their latest CPUs are called **Core** processors.

#### D. The Unix Operating System.

1. Since most of you are well aware of the workings on the various flavors of Microsoft's OS (*i.e.*, Windows), at this point, I will highlight the use of the Unix operating system since you will have to do some work on the Department's Linux computers.
  - a) Unix/Linux is the operating system of choice for *number-crunching* 64-bit workstations and PCs in the scientific community (physics and astronomy in particular) and in technical corporations.
  - b) There are various *flavors* of Unix that exist on the market, each design for use by specific computer platforms.
    - i) Solaris is the flavor of Unix that exist on Sun Microsystems' platforms (*e.g.*, Sparc Stations and Ultra Station). You may also run across an earlier Sun operating system called the SunOS. This operating system is no longer supported by Sun.
    - ii) Irix is the version of Unix used on Silicon Graphics' workstations.
    - iii) HP/UX is the Unix flavor on Hewlett Packard's RISC-based workstations. With its purchase of Compaq/DEC, HP/UX has incorporated many of the security features contained in Tru64 Unix.
    - iv) Digital Equipment Corporation's (DEC) version of Unix was called Tru64 Unix (previously known as Digital Unix, and previous to that, OSF/1) before

they went out of business. Even though HP no longer sells AlphaStations, they do still sell Tru64 Unix for those people with still functioning AlphaStations.

- v) There are at least two platform-independent versions of Unix, SCO (Santa Cruz Operating system) Unix and BSD/OS (Berkeley Systems Development Operating System) Unix.
- vi) Mac OS X of Apple is based on the BSD version of Unix.
- vii) Finally, the Linux version of Unix is designed to run on a variety of different platforms from Intel microprocessors to Sun/Ultra and Alpha chip architectures. Linux is one of the few versions of Unix that can be downloaded for free.

## 2. The History of Unix.

- a) Brian Kernighan and Dennis Ritchie both work at Bell Labs and were involved in the development of the Unix operating system and the C language.
- b) In 1968, Ken Thompson of Bell Labs wrote the original version of Unix in PDP-7 (a DEC mainframe which preceded the VAX) assembly language.
- c) In 1969 a language called TMG was ported to Unix — the compiler was written in assembly.
- d) Using this, Thompson created B, a pared down version of BCPL. B was essentially C without types.

- e) Dennis Ritchie added character, integer, and floating point types to B, anticipating floating point operations in new versions of the PDP. This was the beginning of the C programming language.
  - f) In 1973, the Unix microkernel was rewritten in C.
  - g) The K&R White book was first published in 1978. Brian Kernighan wrote the body of book, and Ritchie wrote the technical appendices.
  - h) The ANSI C standard was released in 1983.
- 3.** Note that the Unix operating system is set up in a hierarchal system as shown in Figure 2.1 of your textbook.
- a) The user and programmer “talks” to the *shell* (*e.g.*, the Bourne shell [sh], the Korn shell [ksh], the C shell [csh]). Note that these shells are not designed well for user interface at the keyboard (up/down/side arrows won’t recall previous commands or edit them). As such, other shells have been developed that allow keyboard editing: csh → tcsh, sh → bash.
  - b) The shell talks to the Unix utilities (*i.e.*, commands like ‘cp’ and ‘ls’).
  - c) The utilities talk to the kernel (*i.e.*, the actual operating system).
  - d) The kernel talks to the hardware.

## E. The Structure of **Unix**.

1. Unix consist of two main parts
  - a) The **kernel** controls:
    - i) Computer memory.
    - ii) Resource allocation.
    - iii) Peripheral systems.
    - iv) Filestore organization.
    - v) File security and access.
  - b) The **shell** provides:
    - i) User interface or command processor.
    - ii) Utility programs.
  - c) Examples of *shell* are the **Bourne** shell (**sh**), **C** shell (**csh**), **Bourne Again** shell (**bash**), **Korne** shell (**ksh**), and the **tcsh** (**tcsh**).
2. Why use **Unix**?
  - a) **Unix** is portable.
  - b) **Unix** is a Multi-user Operating System.
  - c) **Unix** is a Multi-tasking Operating System.
  - d) Long running programs can be sent to **batch** which will run a process in background mode freeing up the terminal for coincidence interactive use.

- e) Users have the ability to write scripts to control the running of complicated processes.
- f) Unix supports the X-Window protocol. X-Windows was initially created by computer scientists from MIT to allow a windowing environment for multi-user operating systems such as Unix and VMS.

### 3. Getting Started in Unix.

- a) Unix is designed to be user-interactive — that is the user can ‘talk’ to the operating system directly through *terminal* windows. **Much of the work you will do on the Linux machines will involve use of such a terminal window.**
- b) Users and passwords. The following commands are useful for checking on current users and changing your password:
  - `passwd` sets users password
  - `who` allows you to see who else is logged on, showing the users on the system, their terminal ID numbers and when they logged in to the system
  - `whoami` tells you your own username and when you logged in
- c) Getting Help:
  - `man` Displays the online manual page for a command
- d) Other useful utilities are available through ‘pull-down’ menus on the X-Window ‘toolbar’ (*e.g.*, Email, web browser, etc.).

### 4. Unix File System.

- a) The Unix file structure is a hierarchy with the root directory ‘/’ at the top (see Figure I-1).

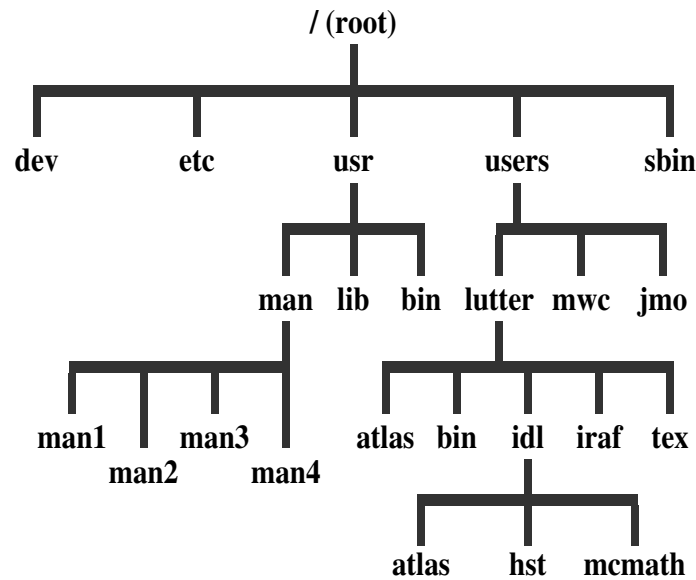


Figure I-1: Directory Tree Structure of a Unix File System

- b) When you log in to a Unix system you are assigned to a **login** (*home*) directory. For example: */users/lutter*.
- c) The directory where you are working is your *current* directory. The *pathname* of the file gives you the exact location of the file. The *full path* is the absolute pathname relative to the root directory. For example:  
*/users/lutter/idl/mcmath*  
 is the directory where I might keep my observed spectra taken with the McMath Telescope.
- d) The *relative path* is the pathname relative to your current directory. For example: From the login directory */users/lutter* the relative path is *idl/mcmath*.

## 5. File Basics and Filenames.

### a) Conventions.

- i) Filenames can be any length, and almost any character is valid. Avoid the following:  $\sim$  \$ % & ( ) [ ] ' ' " ? \ ; < > + - | !

- ii) Unix is case sensitive.
  
- iii) There is no division into filename and file extension, although there are certain naming conventions. For example:
  - \*.c C language source code
  - \*.f Fortran 77 source code
  - \*.f90 Fortran 90 source code
  - \*.pro IDL source code
  - \*.tex  $\text{T}_{\text{E}}\text{X}$  or  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  source code
  
- iv) Wildcards can be used:
  - ? represents any one character
  - \* represents any number of characters, including none
  
- b) Directory Navigation:
  - `pwd` prints the working directory to the screen so that you know where you are
  - `cd` changes directory, takes you back to your home directory:
    - `cd /users/lutter/idl` (moves you to that directory)
    - `cd ~` (moves you to your login directory)
  
- c) Listing Files:
  - `ls` lists the (unhidden) files in the current directory
  
- d) Command switches, or options, are placed after the command and before any arguments.
  - `ls -a` lists files including hidden files  
(note that a *hidden* file begins with a `'.'`)
  - `ls -l` lists the contents of a directory in long form

## 6. File Management Utilities.

### a) Copying Files:

`cp` makes a copy of a file. The syntax of the command is:

`cp sourcefile targetfile`

`mv` moves or renames a file. The syntax of the command is:

`mv sourcefile targetfile`

i) `cp` makes a new copy of a file leaving the existing sourcefile unchanged.

ii) The new copy will overwrite a file of the same name in the target directory.

iii) If there is more than one sourcefile then the targetfile must be a directory.

iv) `mv` creates a new copy of a file without retaining the sourcefile.

v) `mv` is used to move files from one directory to another.

vi) `mv` is used to rename files.

### b) Removing Files:

`rm` deletes unwanted files. The syntax of the command is: `rm filename`

`rm *.old` deletes a group of files ending in `.old`

`rm -i *.old` Unix asks to confirm a deletion before deleting files ending in `.old`

- c) Creating and Removing Directories:
- `mkdir` creates new subdirectories. The syntax of the command is  
`mkdir newdirectory`
  - `rmdir` deletes directories. The syntax of the command is:  
`rmdir directory`
- i) `mkdir` can create a subdirectory of the current working directory.
- ii) `mkdir` can create directories within directories other than the current. For example:  
`mkdir /users/data/ newdirectory`  
(note that you must have “write” permission to create directories and files in directories “outside” your login directory tree).
- iii) To remove a directory: (1) you must be the owner; (2) the directory must be empty; (3) it must not be the current directory.

## 7. Other Unix Commands and Tools.

- a) Displaying Contents of Files:
- `cat` concatenates and displays the contents of a file. The syntax of the command is: `cat filename`  
If the file is more than 24 lines the display will scroll off the screen.
  - `more` displays output from file one screenful at a time.  
The syntax of command is: `more filename`
- b) Pipes and Redirection: The shell normally expects to receive input commands from **stdin** (the keyboard). Output is normally sent to **stdout** (the screen) and any error messages to **stderr** (the screen).

| pipe allows standard output from one command to be used as standard input for another command. For example:  
`cat filename | more`

The output of the `cat` command is *piped* through the `more` command.

> redirects stdout to a real file. For example:

`ls -l > filelist`

< redirects stdin to a real file. For example:

`newfile < filelist`

Redirection always sends output to a file, whereas pipe sends output to another command. There is no limit to the number of pipelines that can be set up.

c) Searching for Strings:

`grep` searches for text in either a single file or group of files. The syntax of the command is:

`grep "search string" filename`

`grep -i` ignores the case of the search string

`grep -l` lists only the names of files containing the search string

`wc` counts words and lines in files. The syntax of the command is: `wc "options" filenames`

`wc -c` displays only the number of characters

`wc -l` displays only the number of lines

`wc -w` displays only the number of words

i) `grep` searches one line at a time.

ii) `grep` looks for strings of text and does not limit itself to whole words.

iii) `grep` can be used with wildcards.

d) File Permissions:

i) The default setting for a new files is:

`-rw-----`

which means that only the user can read and write a file.

ii) The permissions column consists of ten characters. The first character denotes the type of file.

For example:

- ordinary file
- d directory
- l link

iii) The next three show access mode for the user (owner) of the file.

iv) The next three show the access for the group.

v) The last three show access for all others.

vi) The protection symbols mean:

- r **read:** allows user to read the contents of a file
- w **write:** allows user to modify a file
- x **execute:** allows user to execute or run a program file

e) User Access: The different classes are defined as:

- u user (owner) of file
- g group file belongs to
- o other users
- a all users

f) Changing Access Privileges: **chmod** changes the access mode of files you own, to restrict or allow access. The syntax of the command is:

**chmod** *“class(es)” “operation(s)” filename(s)*

*“operation”* is one of: + - = (to add, take away or set

permissions). For example:  
`chmod ug+w example1`

## F. Summary of Unix Commands

1. There are a large number of commands in the Unix operating system. Table I-2 lists some of the more common ones.
2. Note that commands that are listed with Unix flags (*i.e.*, letters after the initial command word) require a minus sign “-”. (Note that some flavors of Unix do not require the “-” qualifier.)
  - a) Most processes are run in *foreground* — this means that one does not get the Unix *prompt* back until the process is complete.
    - i) For typical Unix commands and processes, this is the way to go since these commands and processes take a fraction of a second to run.
    - ii) However, for long running processes (editing a file for instance), it is often wise to put a process in *background* mode. To do this, just include an *ampersand* sign at the end of a command (*e.g.*, `emacs myfile.txt &`).
    - iii) The job will then run in background mode and the Unix prompt will immediately come back to the screen awaiting further input.
  - b) You can check on background jobs by entering the `jobs` command.
    - i) To bring a background job back to foreground mode, either enter `fg` to bring the last entered background job to foreground or enter `fg PID`, where

Table I-1: Common Unix Commands

Unix Command	Description
cat <i>file</i>	concatenate and display <i>file</i> contents
cd <i>dirname</i>	Change to directory <i>dirname</i>
cd ..	Go back to the parent directory of the current directory
cd ~	Go back to <i>home</i> ( <i>i.e.</i> , login) directory
chmod <i>ijk file</i>	Change the read-write-execute protection of a file
cp <i>file1 file2</i>	Copy file <i>file1</i> to file <i>file2</i>
df	Give information about the mounted disks
emacs <i>file</i>	Edit file <i>file</i> with the emacs editor (the shortcut command puts the emacs editor into a pretty screen with big fonts)
grep <i>str file</i>	Search file(s) for pattern
head <i>file</i>	Displays beginning of <i>file</i> (default is ten lines)
logout	Log off of the terminal session
lpr <i>file</i>	Print the file <i>file</i> on the laser printer
ls	Give listing of a directory
ls -l	Give a long listing of a directory
ls -a	Give listing of a directory including hidden files
man <i>command</i>	Display information (man pages) of the Unix command <i>command</i>
mkdir <i>dirname</i>	make directory <i>dirname</i>
more <i>file</i>	Type the contents of file <i>file</i> to the screen one page at a time
mv <i>file1 file2</i>	Move or rename file <i>file1</i> to <i>file2</i>
passwd	Change your password to a new password
printenv <i>var</i>	Display the value of the environment variable <i>var</i>
ps	Display current process status
ps -af	just gives information on the user's processes).
pwd	Print working directory
rm <i>file</i>	Delete file <i>file</i>
rmdir <i>dirname</i>	remove directory <i>dirname</i> , if empty
source <i>file</i>	Run a Unix shell script called <i>file</i>
tail <i>file</i>	Displays end of <i>file</i> (default is ten lines)
who	Displays names and other information about users on system

Table I-1: (continued)

Unix Command	Description
<code>latex file</code>	Run the $\text{\LaTeX}$ file <i>file</i> through the $\text{\LaTeX}$ word processor
<code>dvipdf file</code>	Convert a dvi file <i>file</i> to a PDF file (one that can be printed on the laser printer)
<code>dvips file</code>	Convert a dvi file <i>file</i> to a postscript file (one that can be printed on the laser printer)
<code>xdvi file</code>	Preview a <i>dvi</i> file on an X-Windows terminal
<code>idl</code>	Start IDL in command line mode
<code>idldc</code>	Start IDL in GUI mode

*PID* is the process ID number which can be obtained with the `ps` command described above.

- ii) If you ever need to stop a job while it is in background, enter `kill -9 PID`. (**Be very careful with this command!**)
  - c) You can also *suspend* a foreground job by entering `<Ctrl>-z` (*i.e.*, pressing down on the *control* (Ctrl) key when you hit the *z* key — note that this doesn't work from inside an `emacs` process, `<Ctrl>-z` will save and exit the file you are editing).
3. Finally, we have only scratched the surface of **Unix** in this section of the notes. This, however, should be enough to enable you to work on the **Linux** machines.

## G. The Emacs Editor.

1. Emacs is a user-friendly editor that exists on most **Unix** workstations.
  - a) Although `emacs` is not shipped with **Unix** itself, it is freely available to the Internet community.

- b) It was written (and still evolving) by the *GNU Project*, a group of computer scientists who don't like the large amounts of money that vendors are charging for software — they write software and give it out for free!
- 2. Unix's standard editor is `vi` and it is very user-unfriendly — instead, use `emacs`!
- 3. To edit a file in the `emacs` editor, move to the subdirectory containing the file and enter `emacs filename`, where *filename* is the name of the file to be edited.
- 4. The Linux workstation has the most recent version of `emacs` on it which brings up a nice GUI widget. The buttons and pull-down menus are obvious in their operation, as such, we won't describe the keyboard commands that one could also use inside of `emacs`.

## H. Additional Information about *Linux*.

- 1. Most of you are not used to talking to the operating system by issuing commands at the system prompt in a terminal window (though you should practice getting used to it).
- 2. Many Unix operating systems have a front-end windowing system based on the X Window protocol called CDE (Common Desktop Environment). Most Linux operating systems (*e.g.*, Red Hat) use the `gnome` front-end which is also based on X-Windows.
- 3. As such, you can following the same protocol that you do on a Window's machine by *double clicking* on icons and through the use of pop-up menus from the “toolbar” located at the bottom of the main console screen.