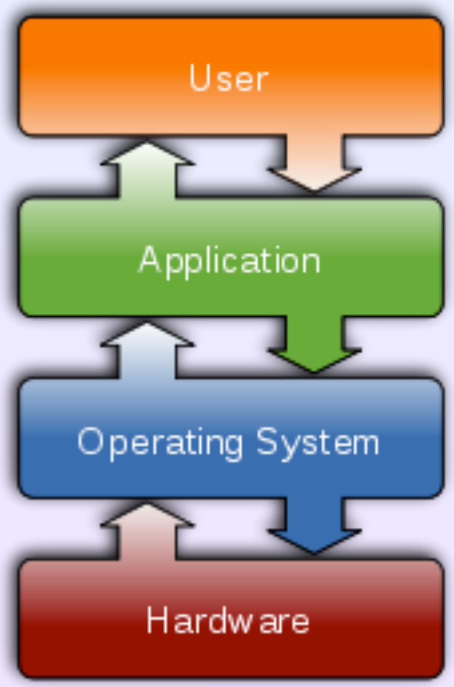
Operating Systems

Overview

Software to handle -

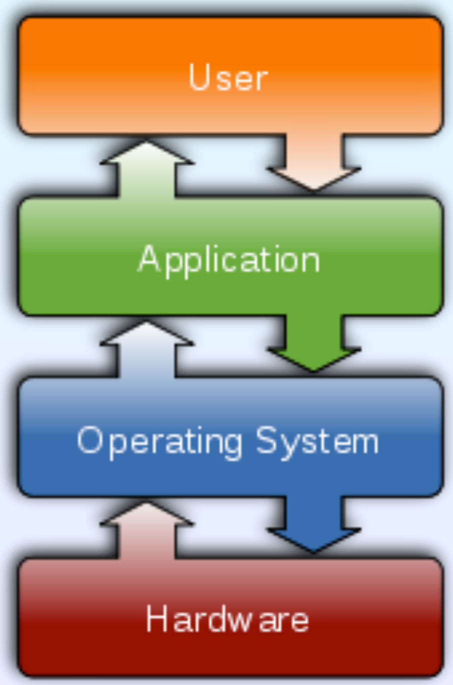
- activities
- sharing resources
- host for application programs - in particular - details of of hardware



Applications access operating system through
application programming interfaces = APIs

They may -

- request services
- pass parameters
- receive output



Users access through software interface

- command line interface
- graphical user interface = GUI -
  runs as API outside operating system

Unix and Unix-like Operating Systems

First developed 1969 at Bell Labs - today many versions/offshoots

Designed to be:

- portable - one machine to another
- multi-tasking
- multi-user - in time sharing mode

Characterized by

- unified method for treating devices as file
- large number of software tools that could be strung together
- master control program called the kernel - handling low level tasks shared by other programs
- a "shell" program that allows user interaction with Unix

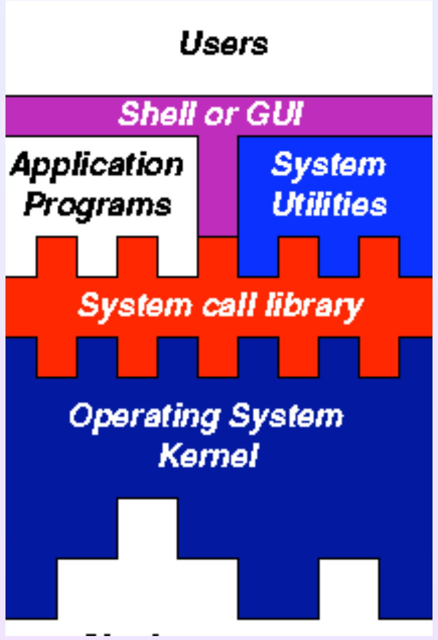
- The kernel is in direct control of underlying hardware and - provides low-level management functions dealing with

  - hardware interrupts
  - sharing the processor among multiple programs,
  - allocating memory
  - ....etc.

- Kernel interacts with higher-level programs through system calls - e.g.

  - creates a files
  - begin execution of programs, or
  - opens network connections to other computers)

- A shell - either text command line interface or a graphical user interface - provides direct user interaction with kernel



- 1973 Unix rewritten in C language - contrary accepted wisdom
  Result - Unix became more portable.

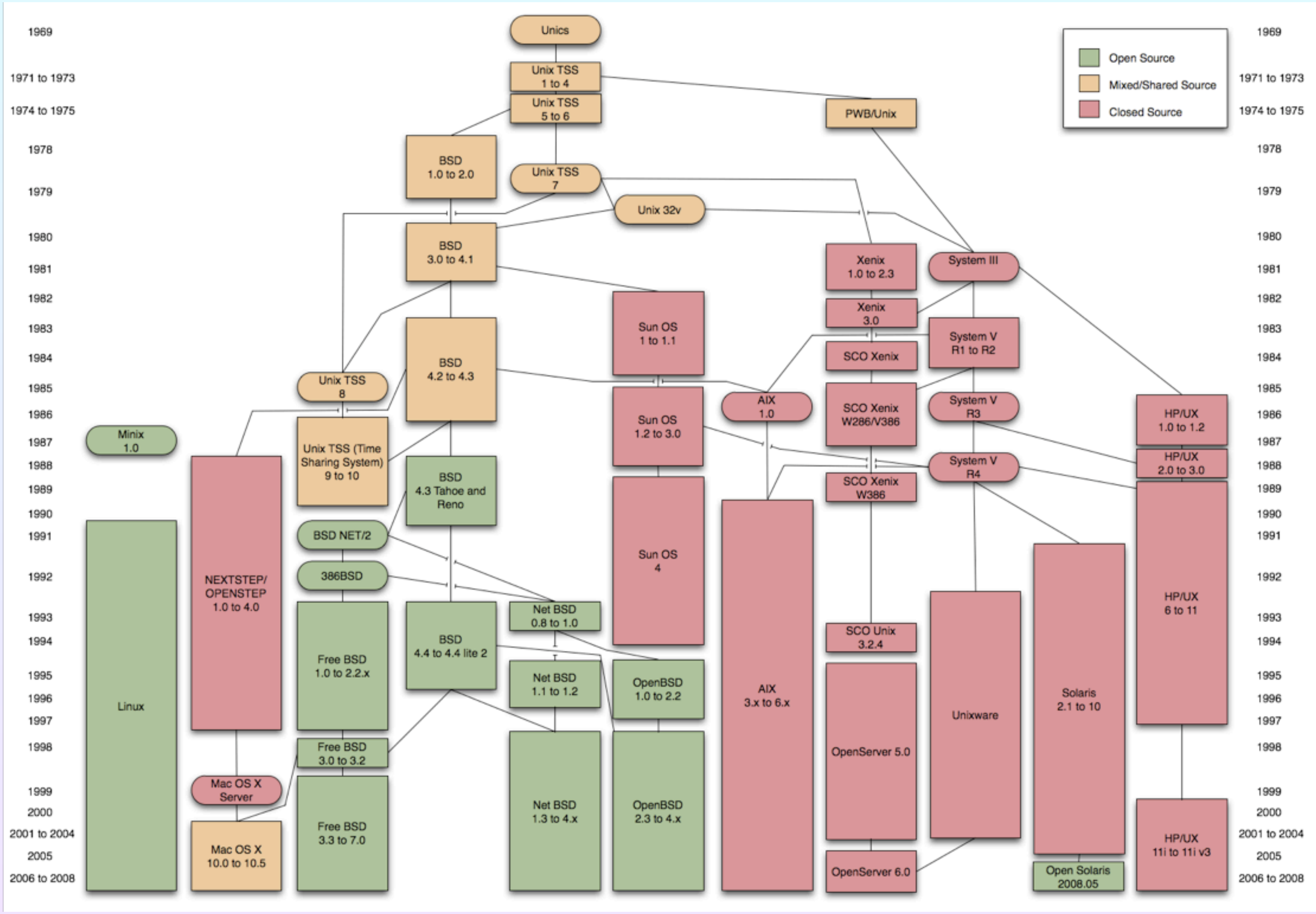
- AT&T made unix available for universities, commercial firms and government on license - System III later System V

- University of California began developing add-ons
  After licensing disputes became alternate version-
  BSD (Berkley Software Distribution) Unix - adding TCP/IP network code.

- Late 1980's version BSD became foundation of NeXTSTEP operating system for NeXT computers
  In turn became foundation of Apple's OS X operating system

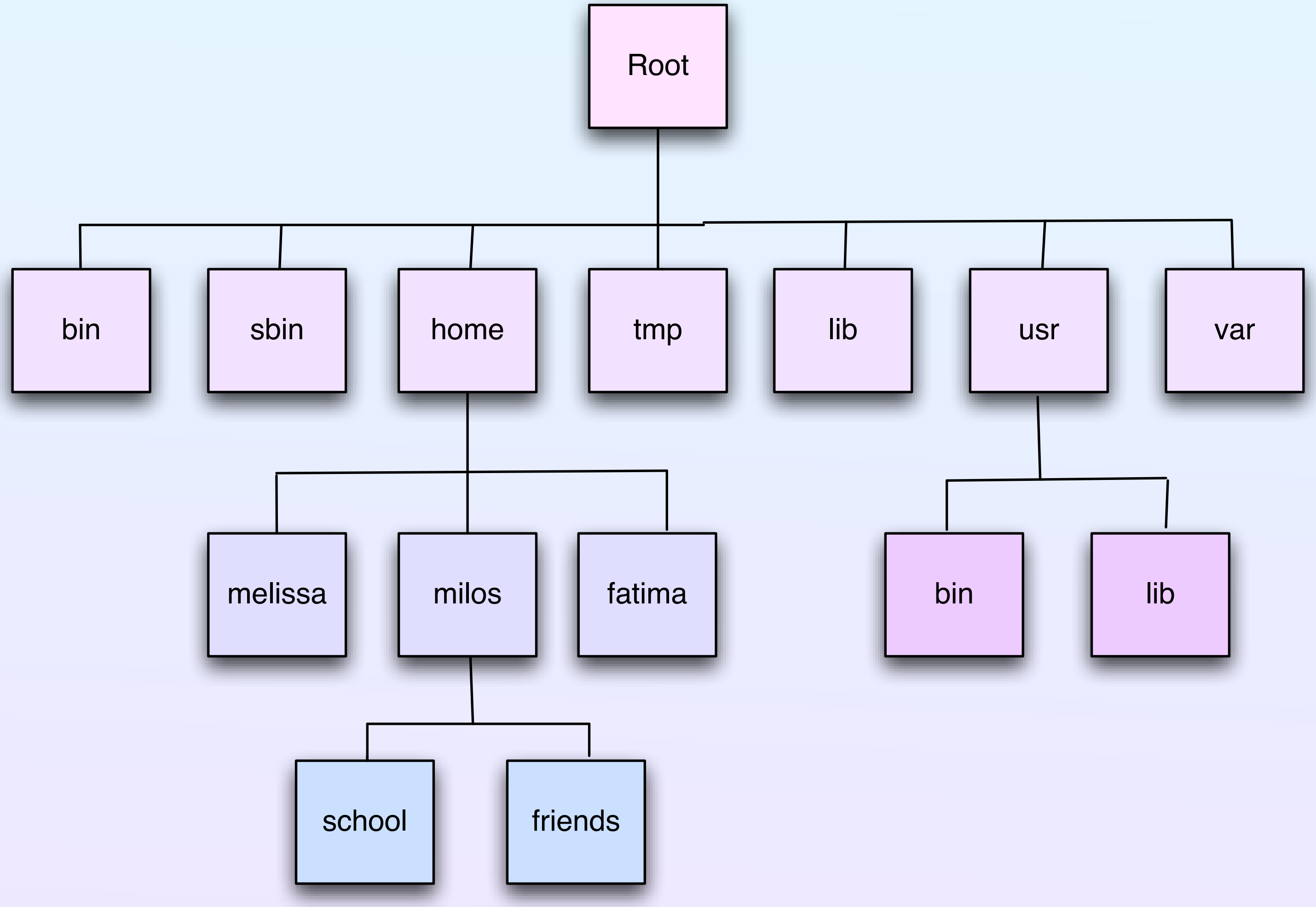
- All proprietary - licenses cost money
  Until Linux - free implementation - deemed Unix-like.
  Because of reliability today used on many servers

Unix versions



Unix File System

Directories form hierarchical tree structure -
top level denoted root - symbolically by forward slash /



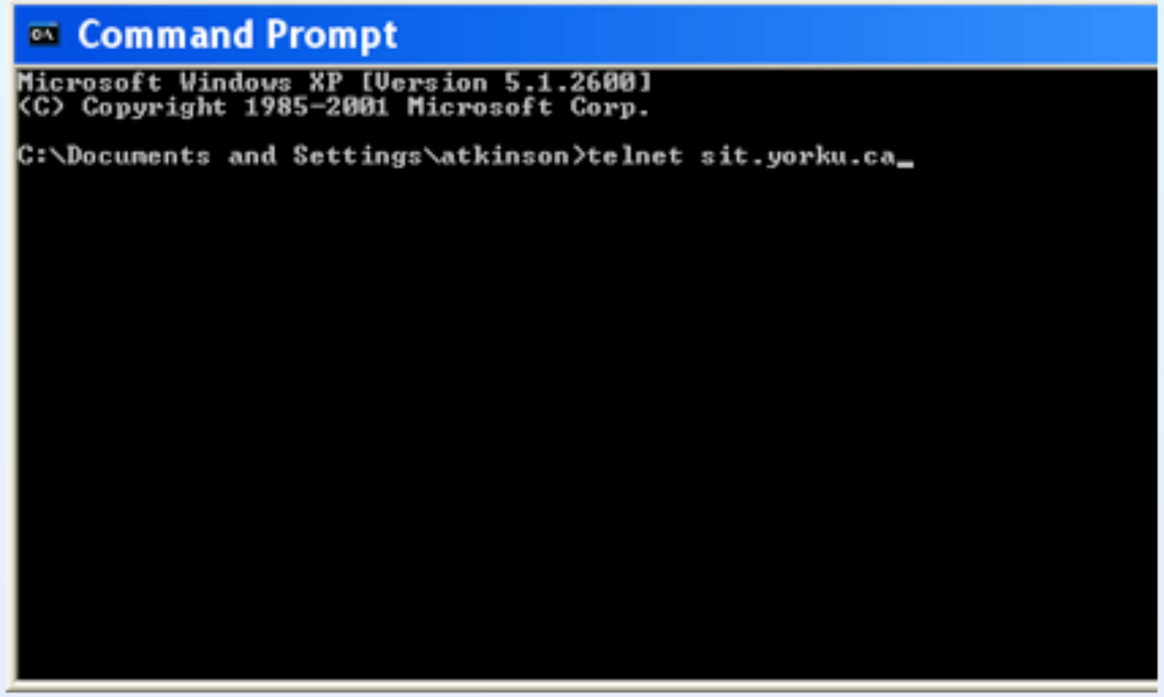
abbreviated directory tree

Typical Linux directories

directory	typical contents
/	root directory
/bin	essential low-level system utilities
/boot	startup files and kernel vmlinux
/usr/bin	higher-level system utilities & application programs
/sbin	superuser system utilities - admin tasks
/lib	program libraries - for low level system utilities
/usr/lib	program libraries for higher-level user programs
/proc	contains “virtual” files that are altered according to running processes
/tmp	temporary space for system - cleaned on reboot
/home	user home directories
/etc	important configuration files
/dev	references to peripheral hardware - files with special properties
/var	variable files - temporary files created by users

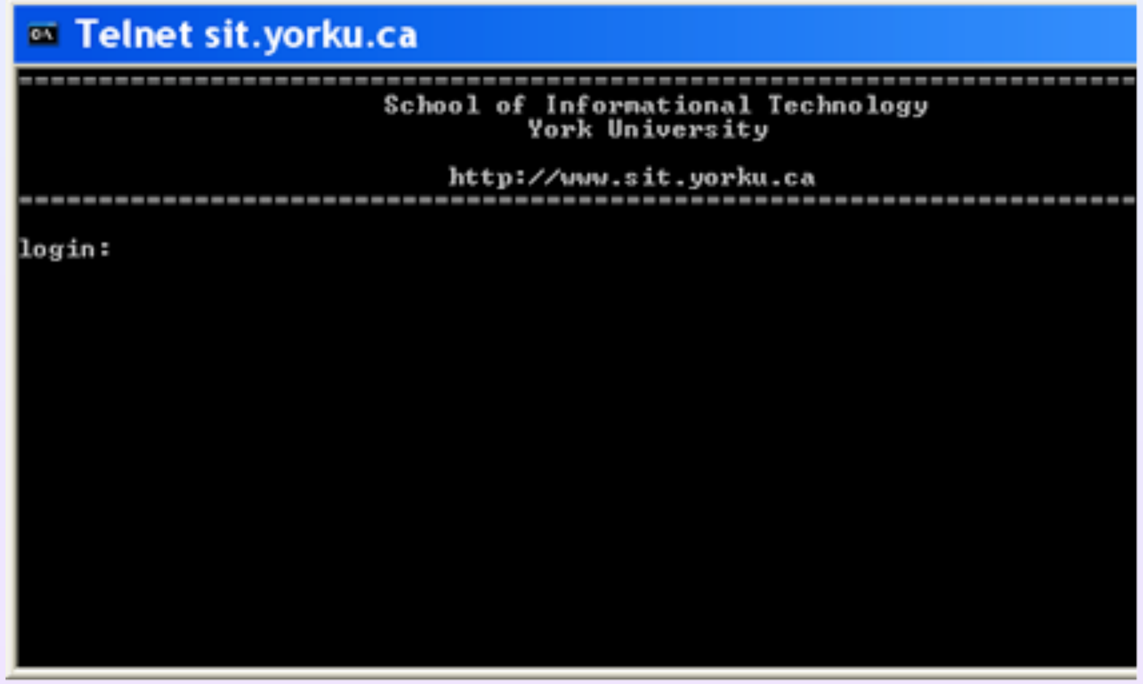
Using Unix - our version - Linux 5.2.1

For Windows users - to logon to the ITEC Unix computer, open Command Prompt window located under accessories



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\atkinson>telnet sit.yorku.ca_
```

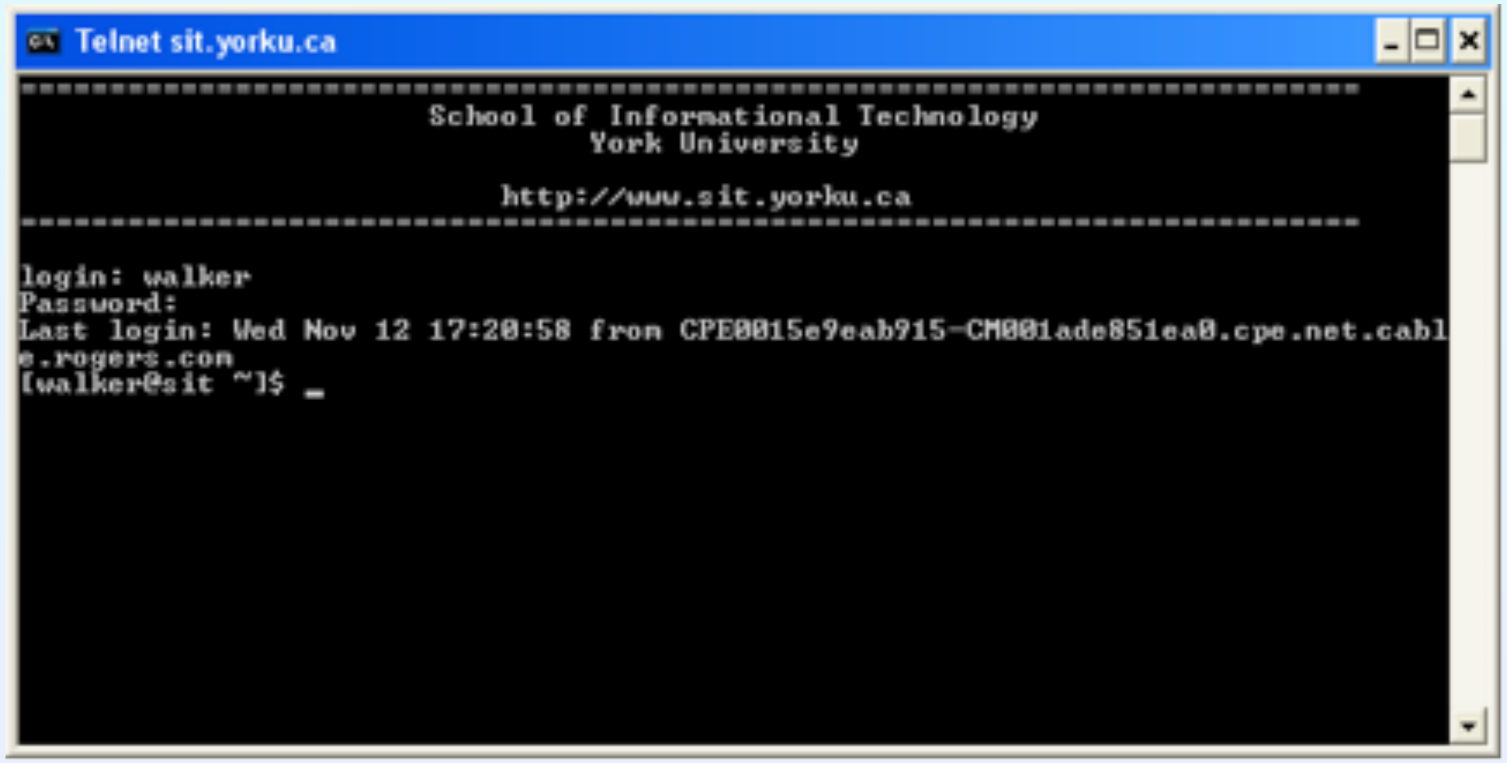
type: *telnet sit.yorku.ca*
or: *ssh sit.yorku.ca*
getting



```
================================================================================
                     School of Informational Technology
                              York University

                          http://www.sit.yorku.ca
================================================================================
login:
```

Next - enter your login name and password

You will have a window similar to -

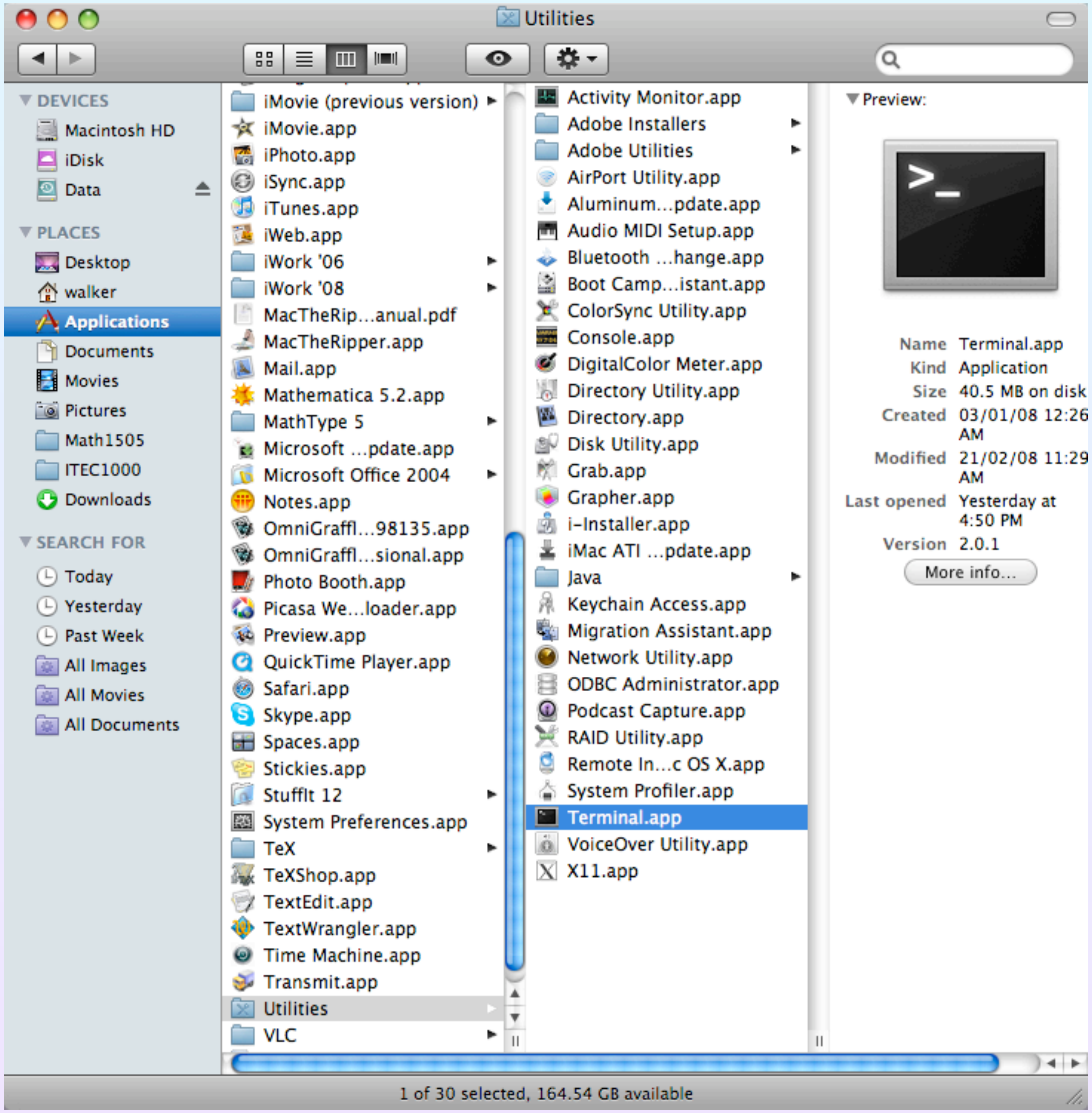


```
===========================================================================
                       School of Informational Technology
                                York University

                           http://www.sit.yorku.ca
===========================================================================

login: walker
Password:
Last login: Wed Nov 12 17:20:58 from CPE0015e9eab915-CM001ade851ea0.cpe.net.cabl
e.rogers.com
[walker@sit ~]$ _
```

Notice the $ prompt - a symbol for the shell interface

For Mac OS X users

Open Terminal.app - proceed as before



Name Terminal.app
Kind Application
Size 40.5 MB on disk
Created 03/01/08 12:26 AM
Modified 21/02/08 11:29 AM
Last opened Yesterday at 4:50 PM
Version 2.0.1

1 of 30 selected, 164.54 GB available

Unix Commands - the simple guide

There are 100's of commands - for most users a small number do the job
play with them

command	function
whoami	prints login name of current user
passwd	for changing password
logout	ends session
pwd	indicates path from root to current directory
clear	clears the screen
cd	change directory - give path - cd .. (with the two dots) gives next higher directory
man xxx	prints the manual page for command with name xxx

To get complete description of command with name xxx type:

man xxx as in - man cd

Permissions

Unix files and directories have 3 permission levels for access for each of the functions - reading (r) , writing(w) or executing (x)

user level -

- as user you have permission for all the files in your home directory or below for reading, writing, or executing, if the file is executable
- the super user or administrator owns the root and hence all files on the system

group level -

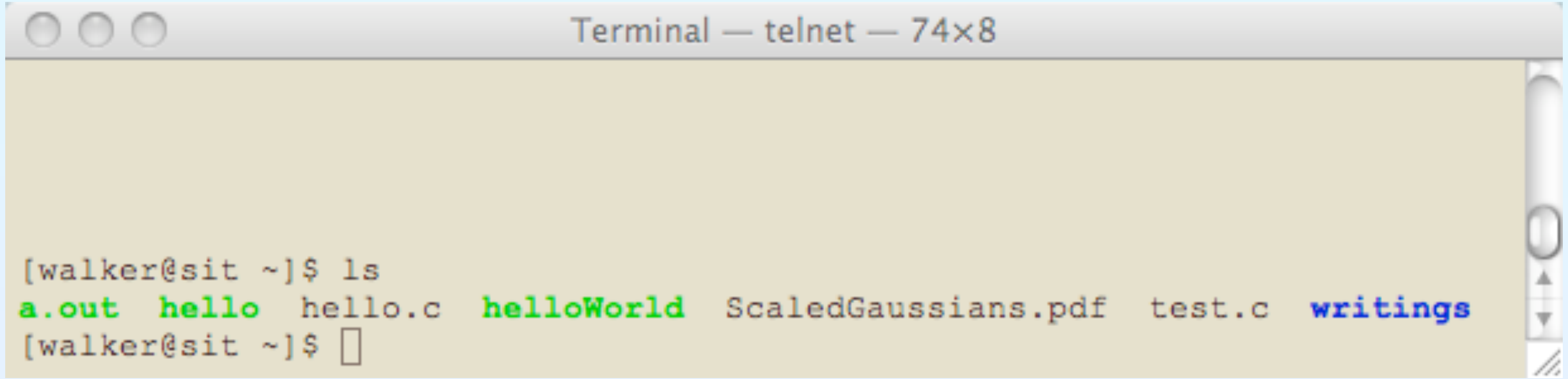
- groups of users can be defined - a group member has access to all files or directories tagged for the group - for reading, writing or executing

other level

- files tagged as “other” can be accessed for reading writing or executing by anyone on the system

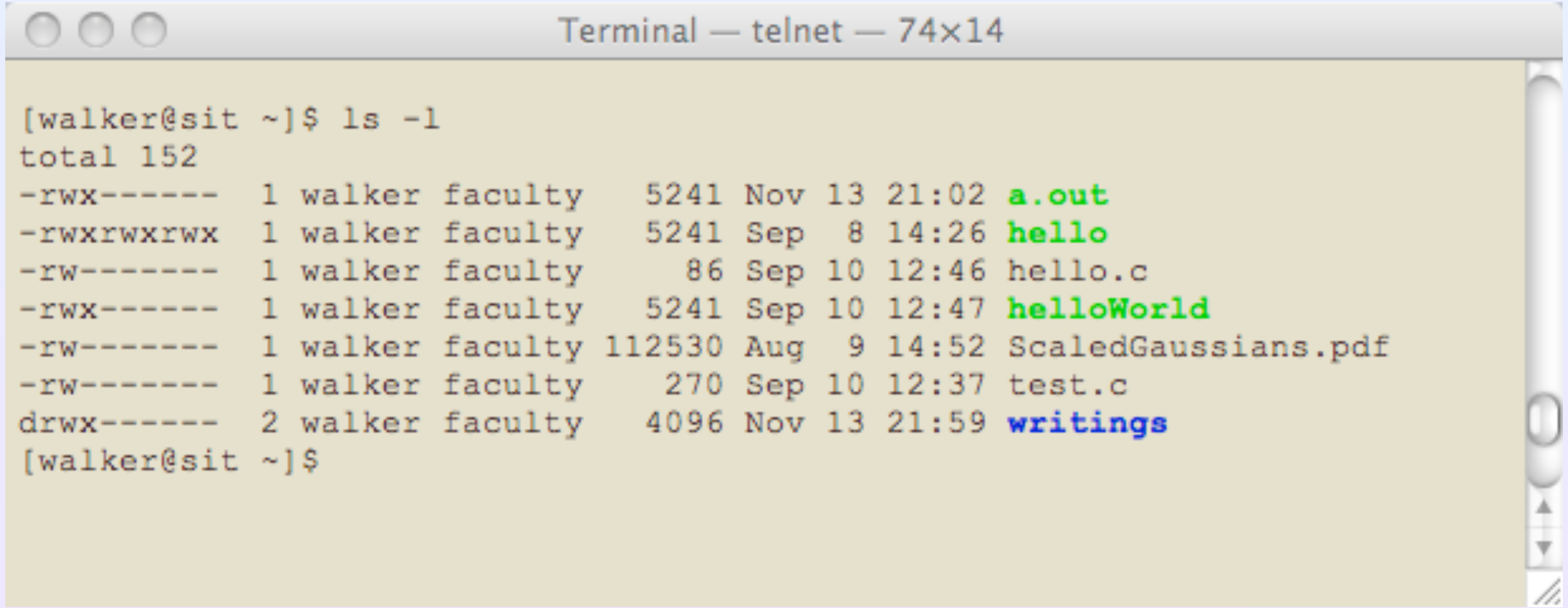
The list command - ls - has a number of options

with no options it is a simple list of names - as below



```
[walker@sit ~]$ ls
a.out  hello  hello.c  helloWorld  ScaledGaussians.pdf  test.c  writings
[walker@sit ~]$ 
```

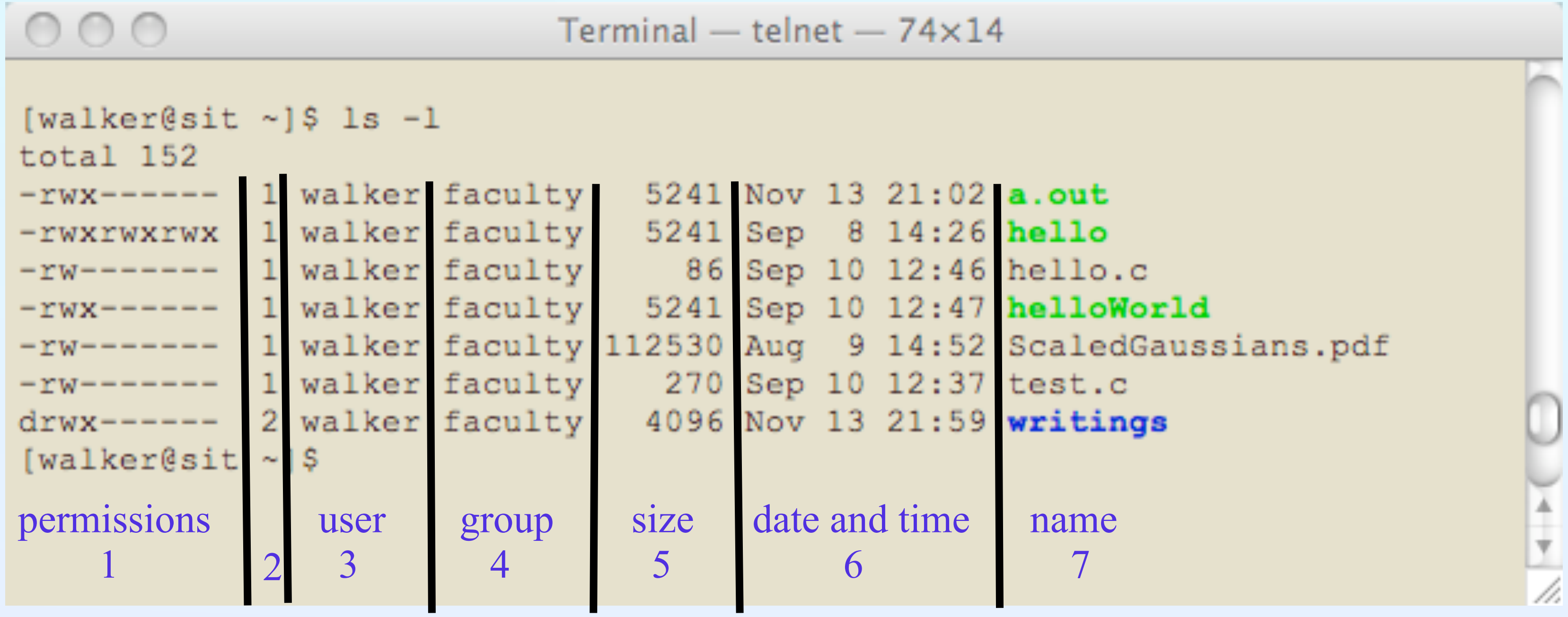
with the -l option it shows a detailed list - as below



```
[walker@sit ~]$ ls -l
total 152
-rwx------  1 walker faculty   5241 Nov 13 21:02 a.out
-rwxrwxrwx  1 walker faculty   5241 Sep  8 14:26 hello
-rw-------  1 walker faculty     86 Sep 10 12:46 hello.c
-rwx------  1 walker faculty   5241 Sep 10 12:47 helloWorld
-rw-------  1 walker faculty 112530 Aug  9 14:52 ScaledGaussians.pdf
-rw-------  1 walker faculty    270 Sep 10 12:37 test.c
drwx------  2 walker faculty   4096 Nov 13 21:59 writings
[walker@sit ~]$
```

Use man ls for a complete list of options

Directory fields



```
[walker@sit ~]$ ls -l
total 152
-rwx------  1 walker faculty   5241 Nov 13 21:02 a.out
-rwxrwxrwx  1 walker faculty   5241 Sep  8 14:26 hello
-rw-------  1 walker faculty     86 Sep 10 12:46 hello.c
-rwx------  1 walker faculty   5241 Sep 10 12:47 helloWorld
-rw-------  1 walker faculty 112530 Aug  9 14:52 ScaledGaussians.pdf
-rw-------  1 walker faculty    270 Sep 10 12:37 test.c
drwx------  2 walker faculty   4096 Nov 13 21:59 writings
[walker@sit ~]$
```

- column 1 with 10 positions: where "-" (hyphen) denotes blank

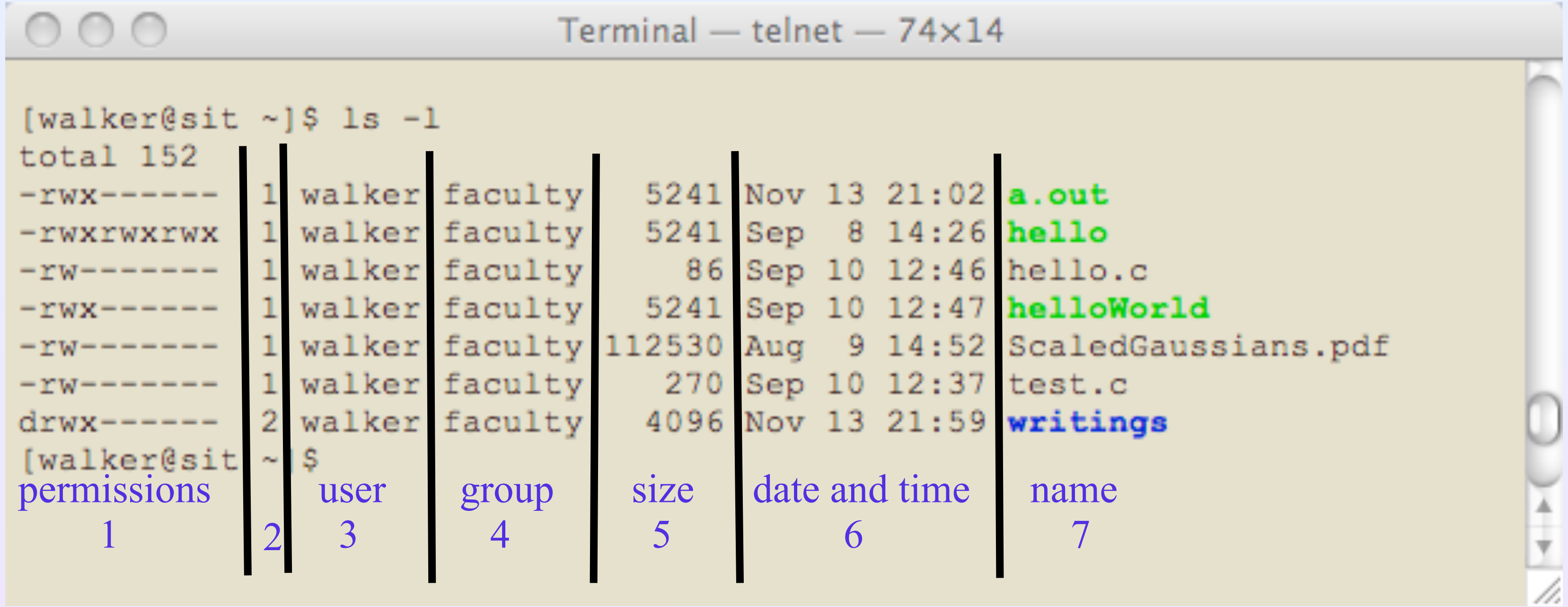
  - far left entry denotes file if blank or directory if "d"
    next 9 positions divided into 3 sets of 3
  - 1st set of 3 indicates if user has read (r), write (w), or execute (x) privilege
  - 2nd set indicates if group members have r, w, or x privilege
  - 3rd set indicates if anyone (other) logged on has r, w, or x privilege

- column 2 entry denotes links to file or directory - technical stuff
- column 3 entry - user name
- column 4 entry - group name
- column 7 entry - file or directory name

Interpretations:

row 1: the file a.out permits access to read, write, execute for only the user walker who belongs to the group faculty.
The file is 5,241 bytes long

row 2: the file hello has read, write, execute permission for walker, anyone in the group faculty or any other - i.e. file may be accessed by anyone logged in, whether faculty or the user walker



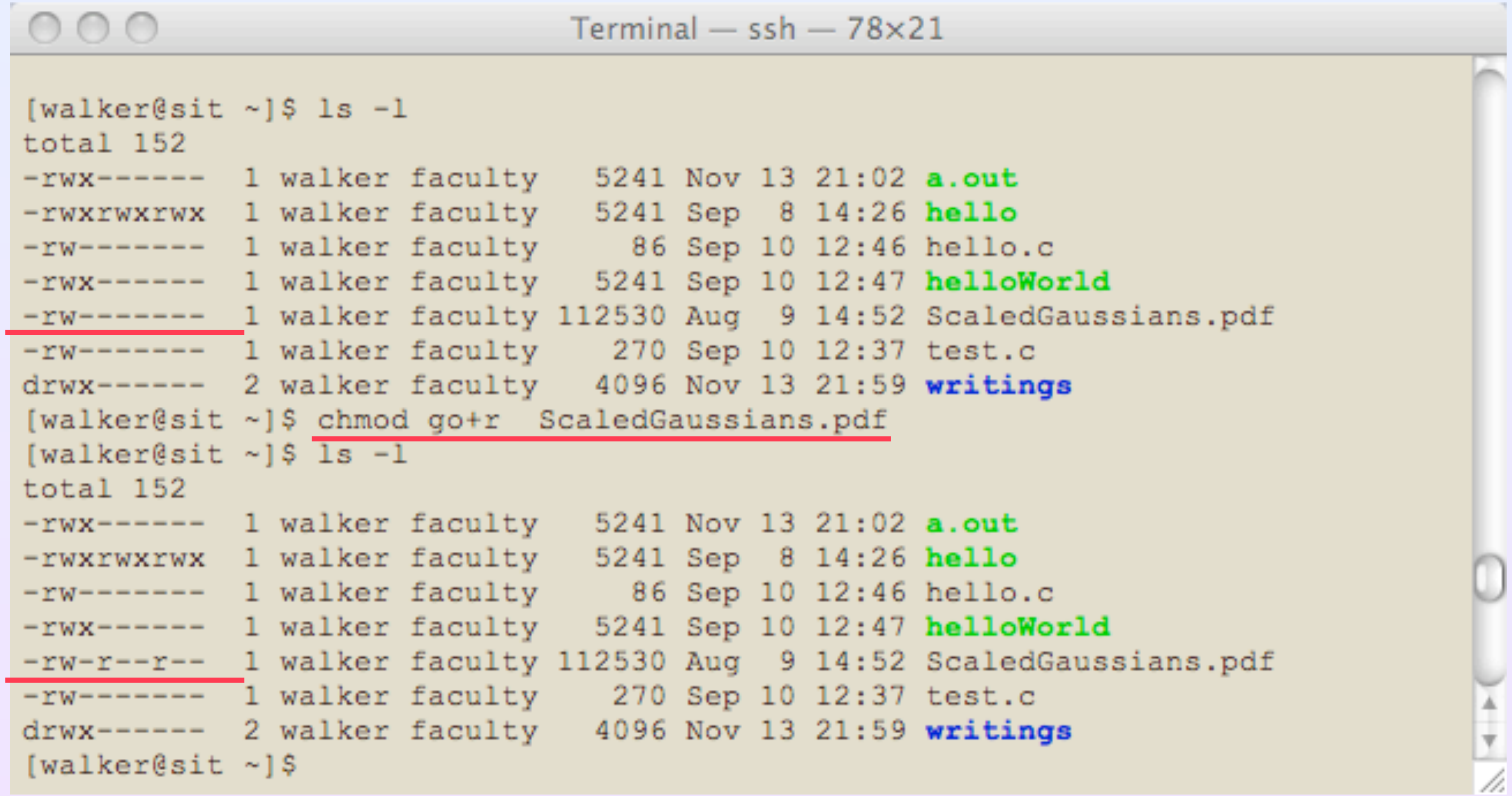
```
[walker@sit ~]$ ls -l
total 152
-rwx------  1 walker faculty   5241 Nov 13 21:02 a.out
-rwxrwxrwx  1 walker faculty   5241 Sep  8 14:26 hello
-rw-------  1 walker faculty     86 Sep 10 12:46 hello.c
-rwx------  1 walker faculty   5241 Sep 10 12:47 helloWorld
-rw-------  1 walker faculty 112530 Aug  9 14:52 ScaledGaussians.pdf
-rw-------  1 walker faculty    270 Sep 10 12:37 test.c
drwx------  2 walker faculty   4096 Nov 13 21:59 writings
[walker@sit ~]$
```

Changing permissions - chmod command

Only the owner of a file or the super-user is permitted to change permissions of a file.

Using symbols ``u", ``g", and ``o" to specify user, group, and other permissions can be changed as in examples - check man page for more

chmod	g+rw	filename	adds to group read and write privileges
chmod	o-rwx	filename	removes from other read, write, execute privileges



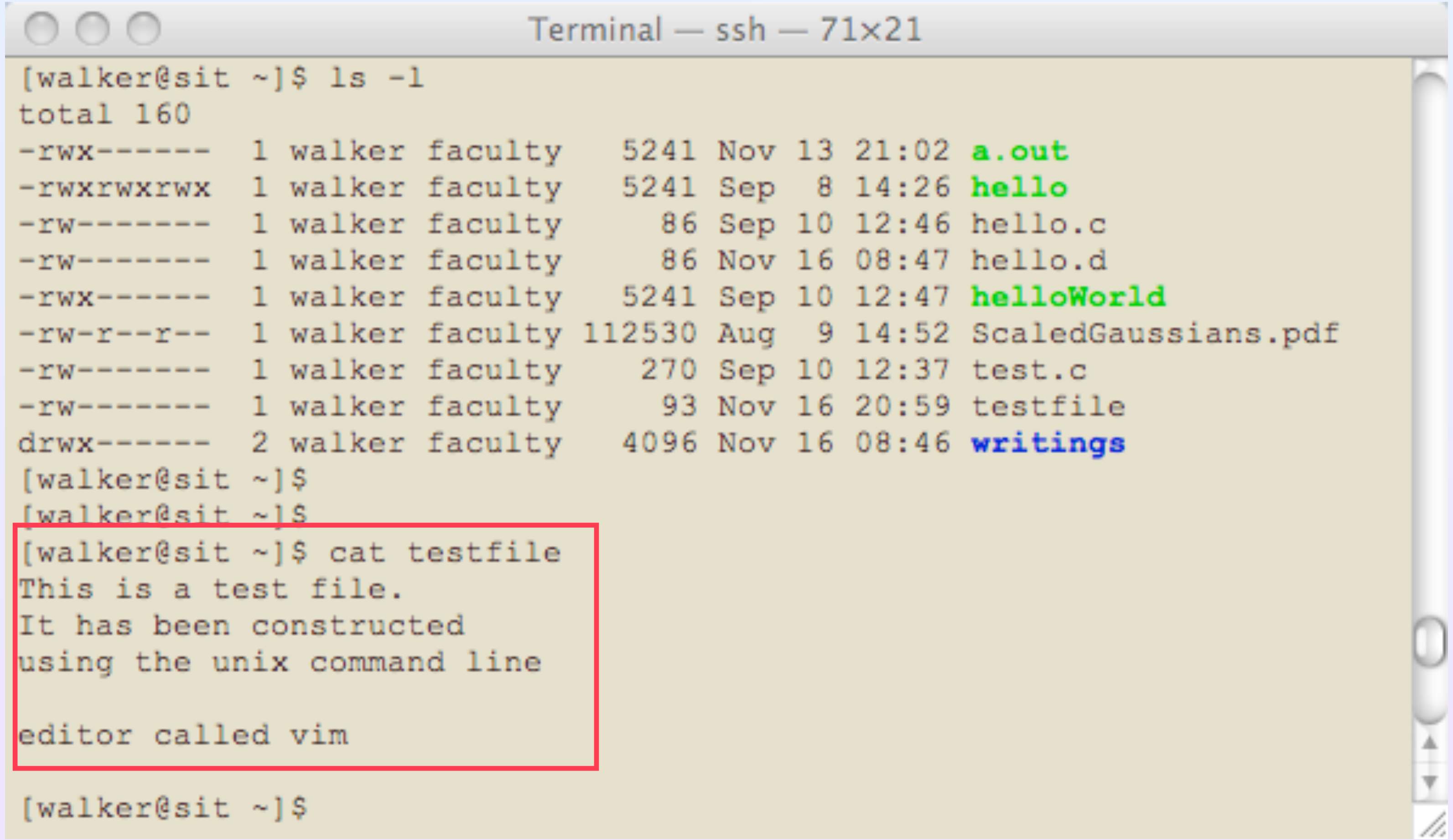
```
[walker@sit ~]$ ls -l
total 152
-rwx------  1 walker faculty   5241 Nov 13 21:02 a.out
-rwxrwxrwx  1 walker faculty   5241 Sep  8 14:26 hello
-rw-------  1 walker faculty     86 Sep 10 12:46 hello.c
-rwx------  1 walker faculty   5241 Sep 10 12:47 helloWorld
-rw-------  1 walker faculty 112530 Aug  9 14:52 ScaledGaussians.pdf
-rw-------  1 walker faculty    270 Sep 10 12:37 test.c
drwx------  2 walker faculty   4096 Nov 13 21:59 writings
[walker@sit ~]$ chmod go+r  ScaledGaussians.pdf
[walker@sit ~]$ ls -l
total 152
-rwx------  1 walker faculty   5241 Nov 13 21:02 a.out
-rwxrwxrwx  1 walker faculty   5241 Sep  8 14:26 hello
-rw-------  1 walker faculty     86 Sep 10 12:46 hello.c
-rwx------  1 walker faculty   5241 Sep 10 12:47 helloWorld
-rw-r--r--  1 walker faculty 112530 Aug  9 14:52 ScaledGaussians.pdf
-rw-------  1 walker faculty    270 Sep 10 12:37 test.c
drwx------  2 walker faculty   4096 Nov 13 21:59 writings
[walker@sit ~]$
```

Other Unix commands

The print to screen function

cat   file-xxx

prints the contents of file-xxx to the screen -
example below prints the contents of file named *testfile*



```
[walker@sit ~]$ ls -l
total 160
-rwx------  1 walker faculty   5241 Nov 13 21:02 a.out
-rwxrwxrwx  1 walker faculty   5241 Sep  8 14:26 hello
-rw-------  1 walker faculty     86 Sep 10 12:46 hello.c
-rw-------  1 walker faculty     86 Nov 16 08:47 hello.d
-rwx------  1 walker faculty   5241 Sep 10 12:47 helloWorld
-rw-r--r--  1 walker faculty 112530 Aug  9 14:52 ScaledGaussians.pdf
-rw-------  1 walker faculty    270 Sep 10 12:37 test.c
-rw-------  1 walker faculty     93 Nov 16 20:59 testfile
drwx------  2 walker faculty   4096 Nov 16 08:46 writings
[walker@sit ~]$
[walker@sit ~]$
[walker@sit ~]$ cat testfile
This is a test file.
It has been constructed
using the unix command line

editor called vim

[walker@sit ~]$
```

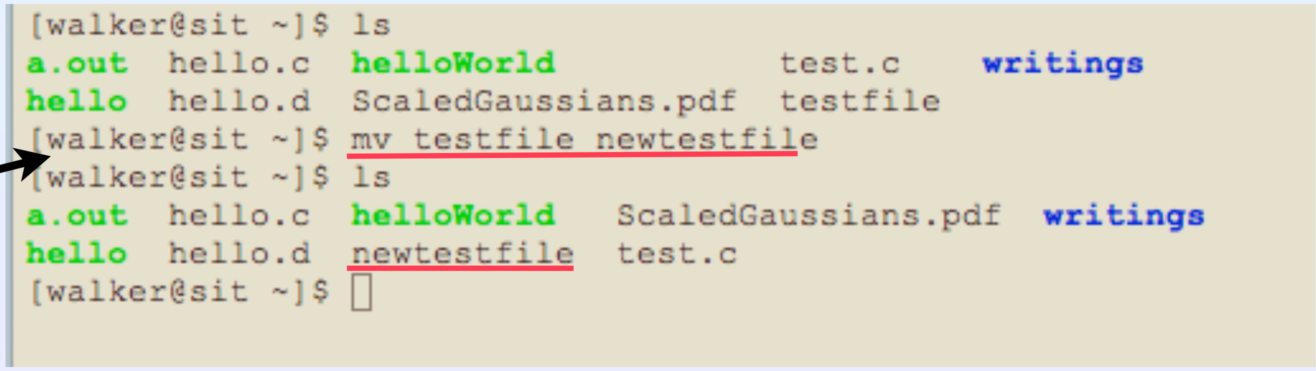
The move function mv

mv has two uses -

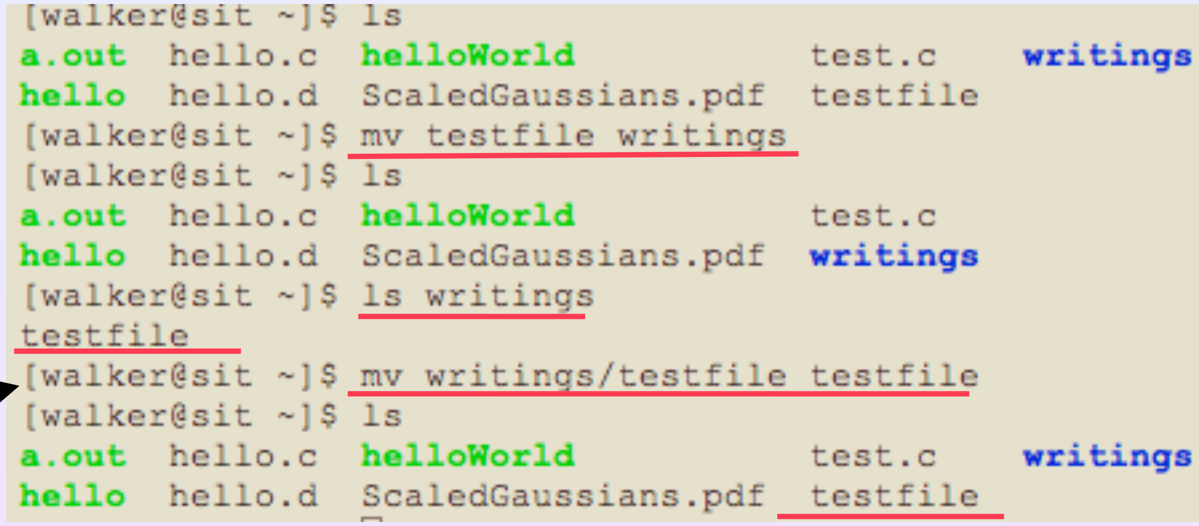
- to rename a file
- to move a file o a named directory

mv file-xxx file-yyy renames file-xxx as file-yyy

mv file-xxx directory-yyy moves file-xxx to directory-yyy



```
[walker@sit ~]$ ls
a.out  hello.c  helloWorld           test.c    writings
hello  hello.d  ScaledGaussians.pdf  testfile
[walker@sit ~]$ mv testfile newtestfile
[walker@sit ~]$ ls
a.out  hello.c  helloWorld   ScaledGaussians.pdf  writings
hello  hello.d  newtestfile  test.c
[walker@sit ~]$ 
```

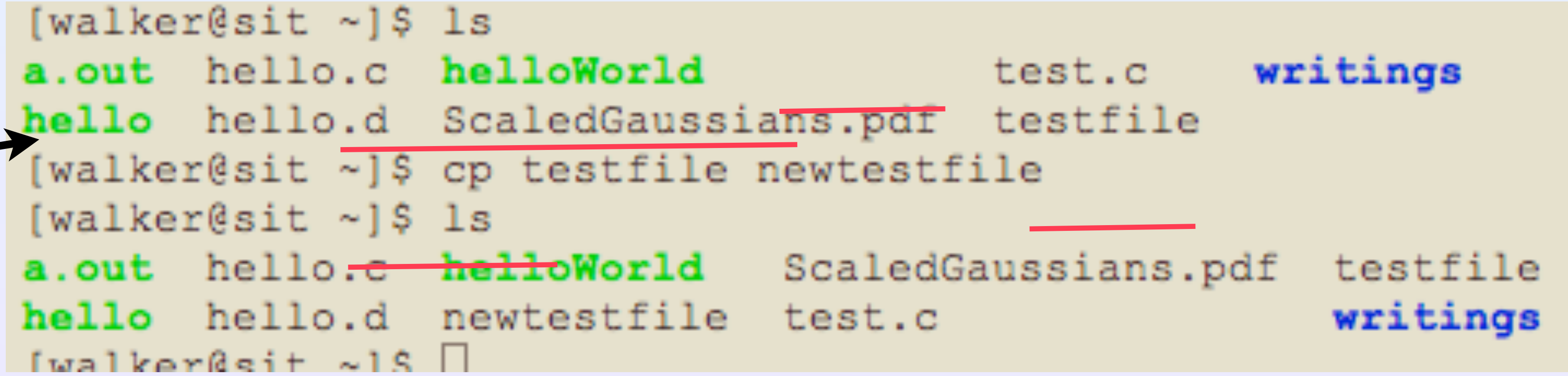


```
[walker@sit ~]$ ls
a.out  hello.c  helloWorld           test.c    writings
hello  hello.d  ScaledGaussians.pdf  testfile
[walker@sit ~]$ mv testfile writings
[walker@sit ~]$ ls
a.out  hello.c  helloWorld           test.c
hello  hello.d  ScaledGaussians.pdf  writings
[walker@sit ~]$ ls writings
testfile
[walker@sit ~]$ mv writings/testfile testfile
[walker@sit ~]$ ls
a.out  hello.c  helloWorld           test.c    writings
hello  hello.d  ScaledGaussians.pdf  testfile
```

The copy function cp

The copy function copies a named file to a new file with specified name

cp file-xxx file-yyy - file-xxx is copied to a new file with given name file-yyy



```
[walker@sit ~]$ ls
a.out   hello.c   helloWorld            test.c    writings
hello   hello.d   ScaledGaussians.pdf   testfile
[walker@sit ~]$ cp testfile newtestfile
[walker@sit ~]$ ls
a.out   hello.c   helloWorld    ScaledGaussians.pdf   testfile
hello   hello.d   newtestfile   test.c                writings
[walker@sit ~]$ □
```

More functions -

- remove a file function rm

  `rm file-xxx` removes the file named file-xxx

- make a directory function mkdir

  `mkdir directory-xxx` makes a new directory named directory-xxx

- remove a directory function rmdir

  `rmdir directory_xxx` removes the directory named directory-xxx

Command line editors

There are a number of ways to create a file in your unix directory

create a file on your PC and then transfer it

fine, provided not trying to write a program to run on the unix machine - PC text editing methods often add extraneous control characters.

use a unix command line editor

vi - a favorite with hard-core systems people - written 1976 for early BSD unix - big improvement over prior IBM line editors

vim - written 1991 - an extended version of vi - more friendly

pico - easier to learn - lacks versatility

At command line type one of the above - and create a text file.

DOS NT Windows

DOS short for disk operating system developed to provide first PCs with an operating system

- First version written quickly in 1980 for Seattle computer - called QDOS = quick and dirty operating system
- Microsoft bought rights to market for $25,000 - later full rights for another $50,000
- DOS became foundation of the PC - - until recently.
- Allowed Microsoft to become what it is.
- Single user - single function - simple compared with UNIX
- Allows user full access to all aspects of machine
- Interesting history

NT (New Technology) - first developed as partnership IBM/Microsoft

Idea in late 80's - create operating system for Intel 8086

high-level-language-based,

processor-independent,

multiprocessing,

multiuser operating system

features comparable to Unix

First called OS2 - later, after IBM and Microsoft partnership split, marketed by Microsoft as NT in 1993

NT supplanted DOS as the foundation of various Microsoft Windows editions

Windows NT 3.1, Windows NT 3.5, Windows NT 3.51, Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista and Windows Server 2008 are all part of the Windows NT family of computer operating systems .

10 every day DOS commands

cd	change directory
cls	clears screen
copy	copies one or more files to a location
del	deletes one or more named files
dir	displays list of files and subdirectories in a named directory
mkdir	makes a named directory
more	displays the content of a named file one screen at a time
move	moves files to destination and renames directories
rmdir	removes all files and directories in a named directory
rename	renames files or directories

Complete DOS information

FTP - file transfer protocol

Software for transferring files between computers

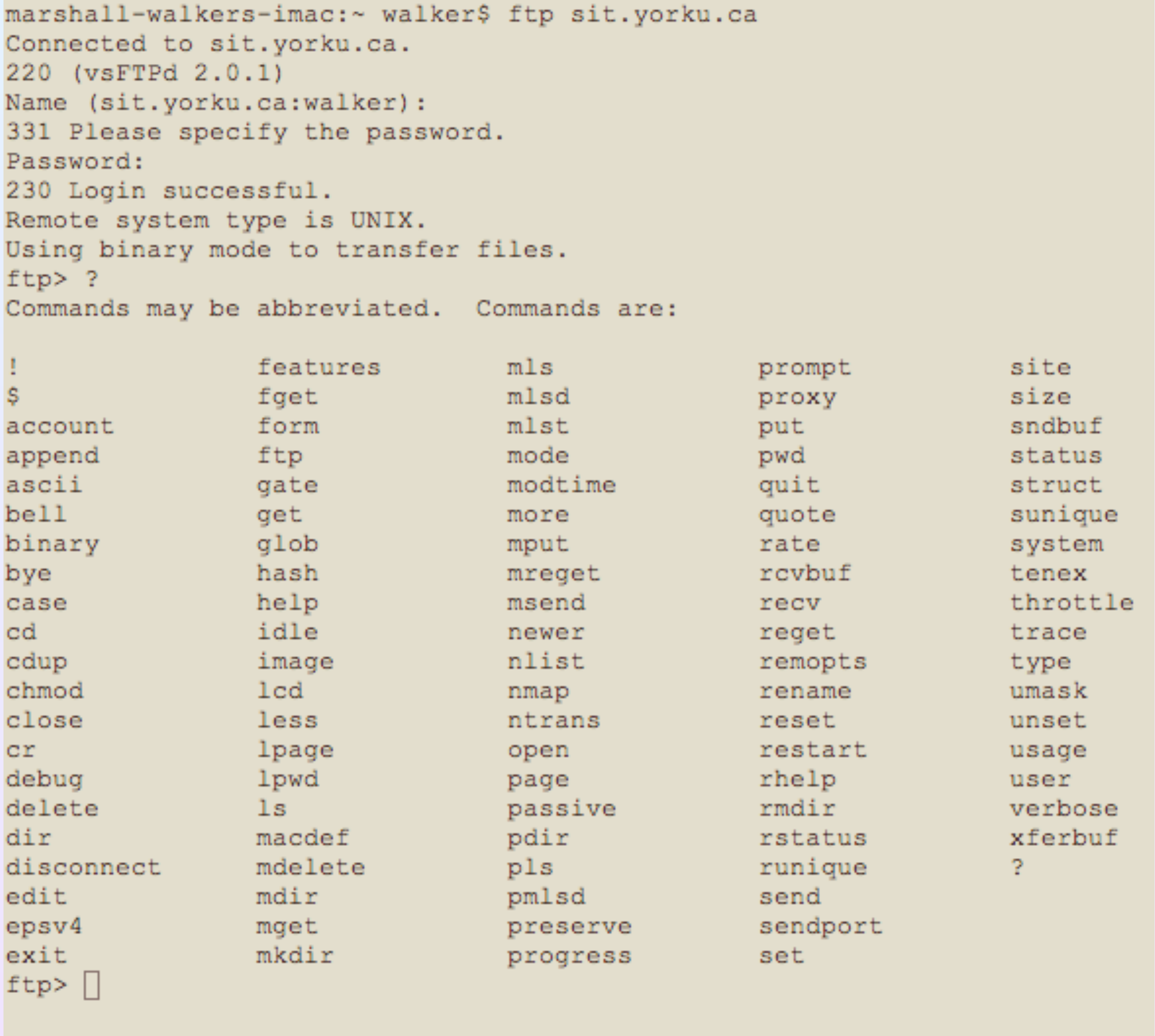
May be accessed through terminal window or graphical user interface

From terminal - type ftp sit.yorku.ca - enter password.

To see list of ftp commands - type ?

For full descriptions see

[ftp commands]()



```
marshall-walkers-imac:~ walker$ ftp sit.yorku.ca
Connected to sit.yorku.ca.
220 (vsFTPd 2.0.1)
Name (sit.yorku.ca:walker):
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ?
Commands may be abbreviated.  Commands are:

!               features        mls             prompt          site
$               fget            mlsd            proxy           size
account         form            mlst            put             sndbuf
append          ftp             mode            pwd             status
ascii           gate            modtime         quit            struct
bell            get             more            quote           sunique
binary          glob            mput            rate            system
bye             hash            mreget          rcvbuf          tenex
case            help            msend           recv            throttle
cd              idle            newer           reget           trace
cdup            image           nlist           remopts         type
chmod           lcd             nmap            rename          umask
close           less            ntrans          reset           unset
cr              lpage           open            restart         usage
debug           lpwd            page            rhelp           user
delete          ls              passive         rmdir           verbose
dir             macdef          pdir            rstatus         xferbuf
disconnect      mdelete         pls             runique         ?
edit            mdir            pmlsd           send
epsv4           mget            preserve        sendport
exit            mkdir           progress        set
ftp> ▯
```

Ftp commands offer ability to manipulate files and directories on remote machine.

Important commands - notice similarity to UNIX and DOS

put	transfers named file in current local directory to remote machine
get	transfers named file on remote machine to current local directory
mget/mput	allows transfer of multiple files
dir or ls	displays contents of remote directory
more	displays contents of named file in remote directory
delete	deletes named file in remote directory
mkdir/rmdir	makes/deletes a subdirectory in remote directory
exit	exits ftp and returns to local machine

Ftp - graphical interfaces

Make life easy - most very similar

For PC :

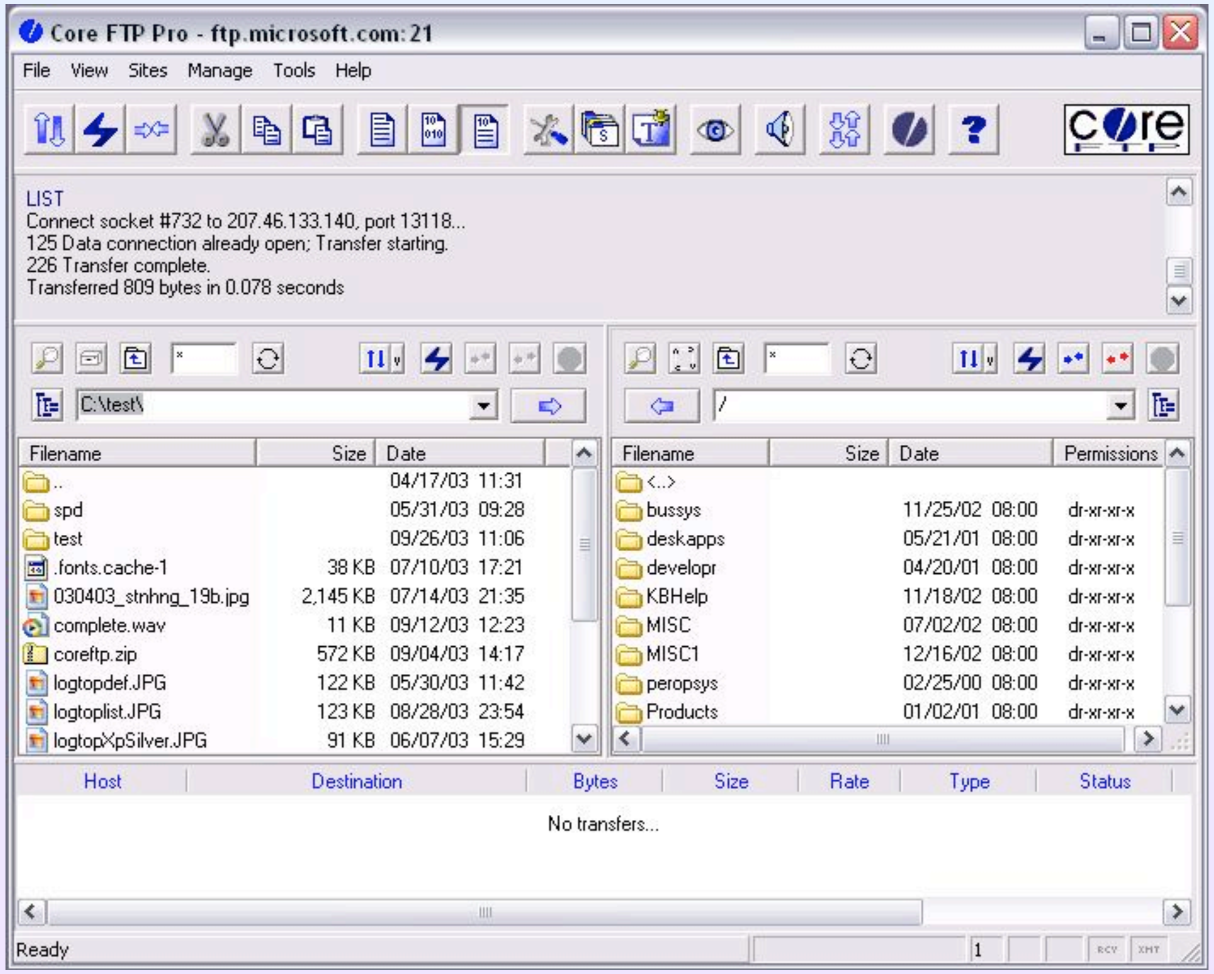
WS_FTP offers free download of home version

Core FTP offers free download of the LE 2.1 version

For Mac:

Fetch costs a few dollars

Transmit costs a few dollars



Core FTP Pro - ftp.microsoft.com:21
File View Sites Manage Tools Help
core FTP
C:\test\
/
Ready
1

```
LIST
Connect socket #732 to 207.46.133.140, port 13118...
125 Data connection already open; Transfer starting.
226 Transfer complete.
Transferred 809 bytes in 0.078 seconds
```

Filename	Size	Date
..		04/17/03 11:31
spd		05/31/03 09:28
test		09/26/03 11:06
.fonts.cache-1	38 KB	07/10/03 17:21
030403_stnhng_19b.jpg	2,145 KB	07/14/03 21:35
complete.wav	11 KB	09/12/03 12:23
coreftp.zip	572 KB	09/04/03 14:17
logtopdef.JPG	122 KB	05/30/03 11:42
logtoplist.JPG	123 KB	08/28/03 23:54
logtopXpSilver.JPG	91 KB	06/07/03 15:29

Filename	Size	Date	Permissions
<..>			
bussys		11/25/02 08:00	dr-xr-xr-x
deskapps		05/21/01 08:00	dr-xr-xr-x
developr		04/20/01 08:00	dr-xr-xr-x
KBHelp		11/18/02 08:00	dr-xr-xr-x
MISC		07/02/02 08:00	dr-xr-xr-x
MISC1		12/16/02 08:00	dr-xr-xr-x
peropsys		02/25/00 08:00	dr-xr-xr-x
Products		01/02/01 08:00	dr-xr-xr-x

Host	Destination	Bytes	Size	Rate	Type	Status

No transfers...

Computer Languages

Text

- Machine code: zeros and ones representing instructions, data , memory locations

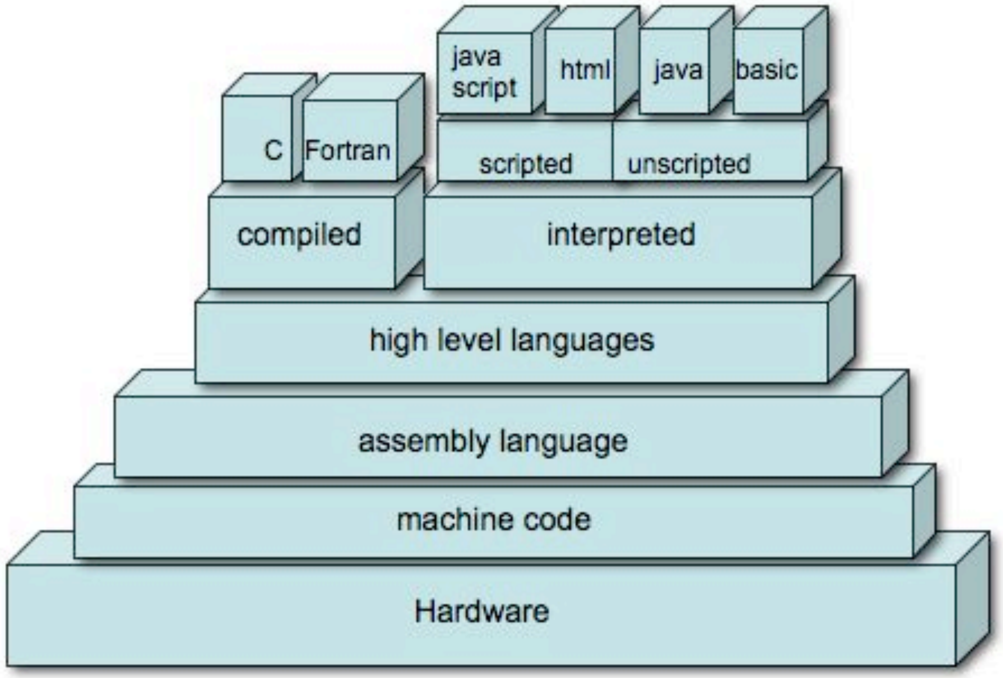
- Assembly code: as with Little Man Computer - one to one translation to machine code

- High level languages: english-like syntax

  - Languages whose programs are compiled - i.e entire content translated to machine code via a compiler program - prominent examples C and Fortran

  - Languages whose instructions are interpreted “on the fly” where sequentially each instruction is interpreted in machine code and executed

- Two types of interpreted languages -
  - scripted : job control languages, shells, application specific languages, web browsers, text processing - see scripted
  - non-scripted: general application programming languages examples: Java, Basic



Complete history

Compiled vs. Interpreted Languages:

Using visual basic as example of interpreted language see following web text

By default, applications created in Visual Basic are compiled as interpreted or p-code executables. At run time, the instructions in the executables are translated or interpreted by a run-time dynamic-link library (DLL). The Professional and Enterprise editions of Visual Basic include the option to compile a native code .exe. In many cases, compiling to native code can provide substantial gains in speed over the interpreted versions of the same application; however, this is not always the case. The following are some general guidelines regarding native-code compilation.

- Code that does a lot of primitive operations on hard-typed, nonstring variables will yield a maximum ratio of generated native code to displaced p-code operations. Complex financial calculations or fractal generation, therefore, would benefit from native code.
- Computationally intensive programs, or programs that shuffle a lot of bits and bytes around within local data structures, will gain very visibly with native code.
- For many programs, especially those doing a lot of Windows API calls, COM method calls, and string manipulations, native code will not be much faster than p-code.
- Applications that consist primarily of functions from the Visual Basic for Applications run-time library are not going to see much if any advantage from native code, because the code in the Visual Basic for Applications run-time library is already highly optimized.
- Code that involves a lot of subroutine calls relative to inline procedures is also unlikely to appear much faster with native code. This is because all the work of setting up stack frames, initializing variables, and cleaning up on exit takes the same time with both the p-code engine and generated native code.

Note that any calls to objects, DLLs or Visual Basic for Applications run-time functions will negate the performance benefits of native code. This is because relatively little time is spent executing code — the majority of time (usually around 90–95%) is spent inside forms, data objects, Windows .dlls, or the Visual Basic for Applications run time, including intrinsic string and variant handling.

In real-world tests, client applications typically spent about 5% of their total execution time executing the p-code. Hence, if native code was instantaneous, using native code for these programs would provide at most a 5% performance improvement.

What native code does is to enable programmers to write snippets of code or computationally intensive algorithms in Basic that were never possible before because of performance issues. Enabling these "snippets" to run much faster can also improve the responsiveness of certain portions of an application, which improves the perceived performance of the overall application.

Compiled Languages

Examples:

- Fortran (FORmula TRANslator) language
  - one of first - 1954-1958
  - ever popular in scientific settings
  - history of Fortran
- C language
  - developed 1972 for construction of Unix
  - powerful, flexible, fast execution, few constraints on programmers
  - modularity - sections of code can be stored for re-use in future programs
  - history of C

    -in particular see Development of the C language by Dennis Ritchie

Evolution of computing practice

In the beginning with machine code

Data and instructions encoded in long, featureless strings of 1s and 0s. Up to t programmer to keep track of where everything stored in the memory.

With assembly language

Raw binary codes replaced by symbols such as load, store, add, sub. The symbols translated into binary by a program called assembler, which also calculated addresses. Still the programmer had to keep in mind all the minutiae.

With high level language

In '70's programmers freed of thinking in terms of addresses and registers. Now had tools allowing serious trouble. Spaghetti code. Huge headache to business

Hence development of structured programing - eliminate "goto" statement allowing logic to jump around - top down design built around self contained modules.

In ’80’s as complexity still increased - object oriented programming

lets you group operations and data into modular units called *objects*

lets you combine objects into structured networks to form a complete program

in object-oriented programming objects and object interactions form basic elements of design

Examples: Java, C^{++}, $C^{\#}$/.NET, Python, Ruby

For discussion on future programming methods see

post object oriented paradigm