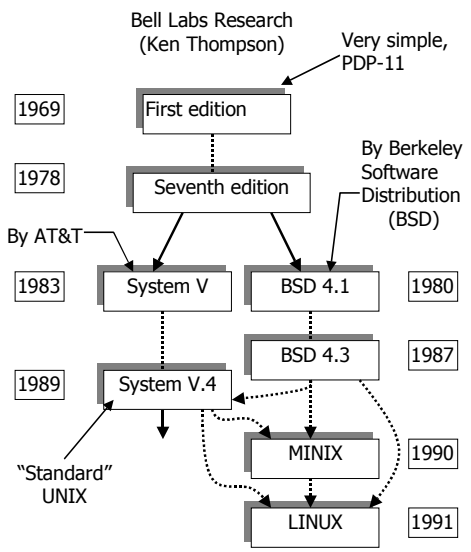
Lecture 1 - Data Structures

UNIX, C++ program in UNIX, system calls

1

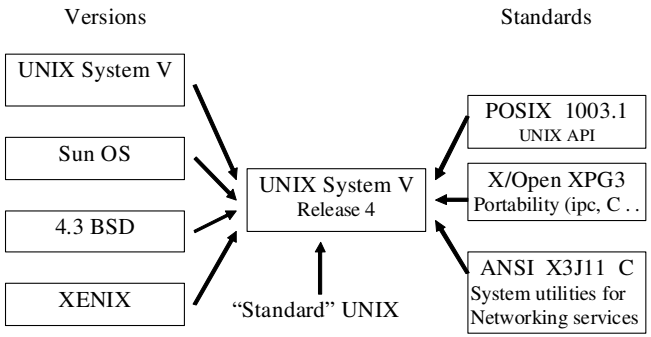
Major Steps in UNIX history



2

Review of the UNIX history and architecture.

- Convergence of **UNIX** versions and standards



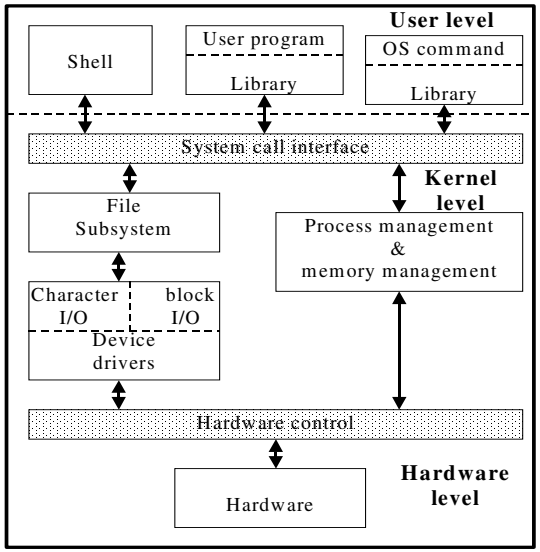
3

Some features of the UNIX System V Release 4

System V Release 3	•Remote File Sharing (RFS) Transport Layer Interface (TLI) STREAMS communication facility Inter Process Communication (IPC)
4.3 BSD	•TCP/IP Protocols Sockets •Fast File System
SUN OS	•Networked File System Remote Procedure Calls (RPC) •Memory Mapped Files
XENIX	•80386 Binary compatibility
New Features	•Virtual File System •Real Time •STREAMS enhancements

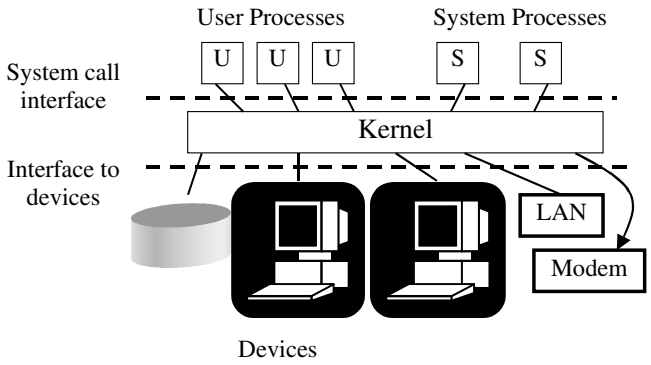
4

Basic modules of UNIX



5

UNIX and its basic services



6

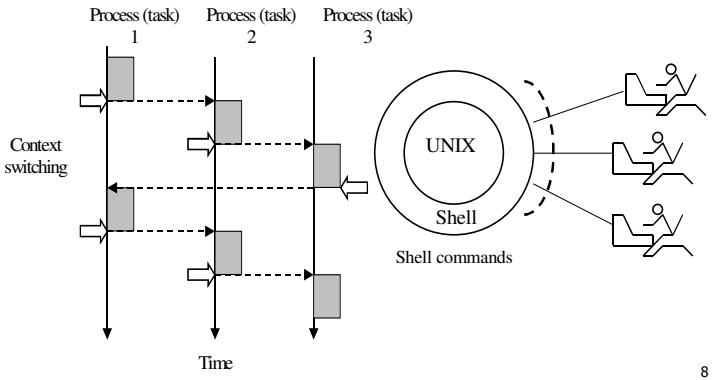
UNIX kernel

Basic services:

- System initialization ("bootstrap" facility).
- Process management (creation, control, and termination of process).
- Memory management (using a virtual memory).
- File system management.
- I/O management
- Communication facilities (inter-process communication, networks).
- Program interface (system calls).

7

- UNIX is a Multitasking Multiuser OS, with a Virtual Memory, Programmable Shell and Networking capability.

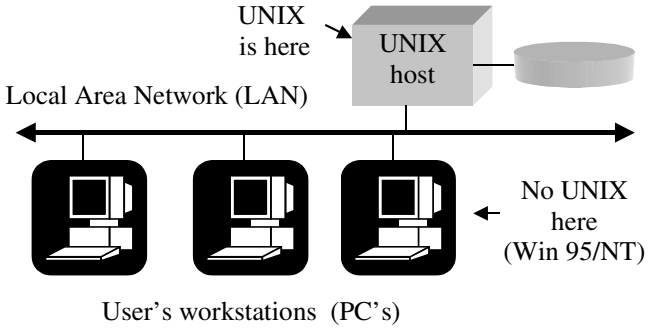


8

Networking capability of UNIX

- **Electronic mail (e-mail)**
- **File transfer (FTP) service**
- **World wide web (WWW) service**
- **Remote login (through Telnet)**
- **Archie –** find anonymous FTP files
- **Gopher –** Gopher space (Veronica, Jughead) – find information on Internet (WAIS – Wide Area Information Service)
- **Ping**
- **Finger –** 1) get info for UID 2) who is logged on a computer 3) services – coke, earthquake, forcast
- **Traceroute**
- **Etc.**

Multiuser capability of UNIX



A few UNIX commands

ls or ls -l	to list the current directory
cd <path>	to change a directory
rm <file>	to delete a file
mkdir <name>	to create a directory
rmdir <name>	to remove a directory
more <file>	to show a text file

A typical session in UNIX

Login: your identificator
Password: your password
%command 1
%command 2
% …
%logout

UNIX on-line help manual

Section	Contents
1	User commands (Shell commands)
2	OS services (system calls)
3	Library functions
4	Devices, networks, interfaces (special files)
5	System file formats
6	Demo programs, Games
7	Miscellaneous (ASCII, etc) information
8	System maintenance commands

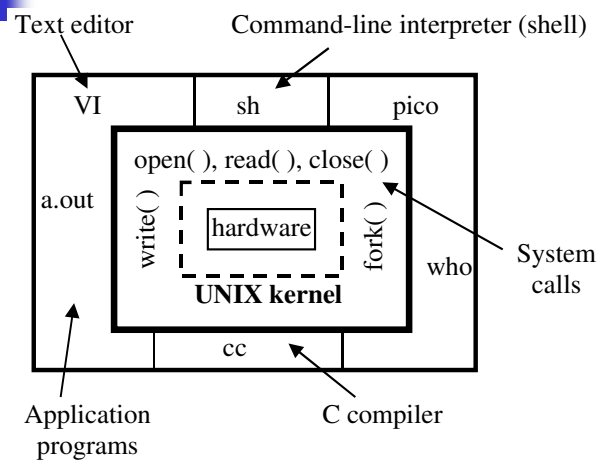
- Section organized in pages ;
- Each page – one command description (can be larger than one page)
- Each section has an introduction

%man 2 intro
%man 2 fork
%man 3 sin

BSD 4.3	System V.4
man	man
whatis	locate
apropos	usage
learn	starter
	glossary
	help

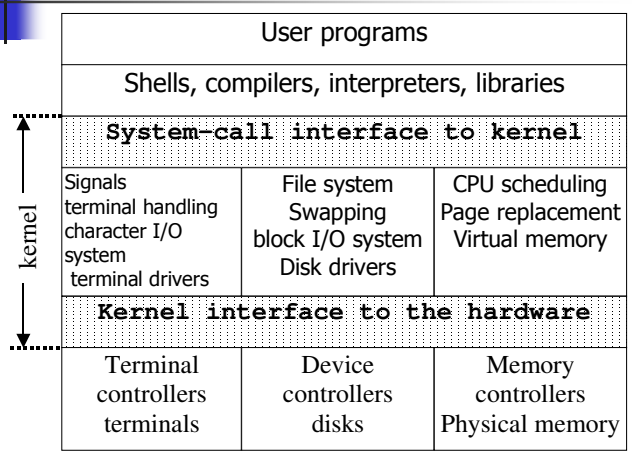
13

An abstract view of UNIX architecture



14

A more detailed (layer) structure of UNIX

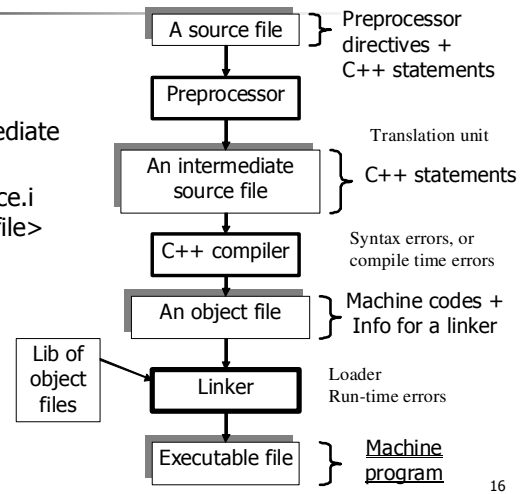


15

Steps in implementing a C++ program

To get an intermediate source file:

- g++ -o produce.i <source C++ file>



16

How to prepare a C program in UNIX

- **Case A**: Your program consists of one source file

1. After starting a text editor (for example PICO), create and save a source text file (for example myprog.cc).	%pico . . . . working with pico . . . .
2. Compile and link the program. The result is executable file (for example, myprog).	%g++ -o myprog myprog.cc
3. Start your program.	%myprog or %myprog param1 param2

Case B

Your program consists of a few separate source files (for example, module1.cc, module2.cc, module3.cc)

1. Create in a text editor, and save each source file.	
2. Compile separately each source file. The result is object files (module1.o, module2.o, module3.o)	%g++ -c module1.cc %g++ -c module2.cc %g++ -c module3.cc
3. Link the object files. The result is executable file.	%g++ -o myprog module1.o module2.o module3.o

Using the *make* utility

Your home directory → prog1.c prog2.c makefile

```
#possible makefile
CC=cc
CFLAGS = -g
LIBS = /lib/libm.a
all:  prog1  prog2
prog1 : prog1.o
	${CC} ${CFLAGS} prog1.o –o prog1
prog2 : prog2.o
	${CC} ${CFLAGS} prog2.o ${LIBS}\
	-o  prog2
clean:
	rm –f  *.o  *#core  prog1  prog2
```

- Using makefile:
- %make
- All necessary recompilations will be done automatically.

Two forms of the main function in a C++ program

```
int main( )
{
    body of the function
}
```

Without command line parameters
To start:
%progname

```
int main(int argc, char *argv[])
 {
    Body of the function
}
```

With one or more command line parameters
To start:
%progname par1 par2 ..

Example:

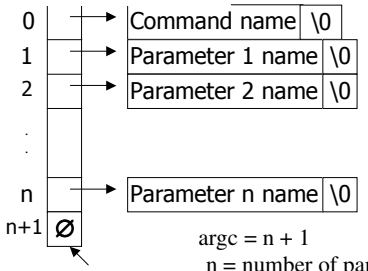
%time ← no parameter

%mkdir dirname ← **with parameter**

Understanding arguments of *main()* function

```
int main (int argc, char *argv[] )
    /* argc - number of command words */
    / * argv - array of pointers to words  */
{   . . . . . .
}
```

*argv [ ] structure



argc = n + 1
n = number of parameters

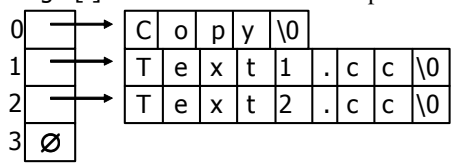
Using **argv** in the program:

argv[0] – points to the program name
argv[1] – points to the first argument
***argv** – points to the program name
***argv[2]** – points to the 2nd character in the program name

Example 1:

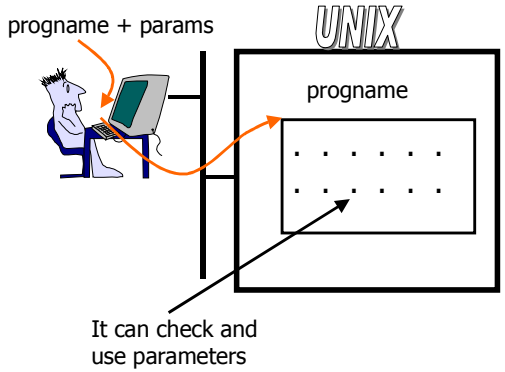
copy text1.cc text2.cc
We have:
argc = 3

*argv [ ] structure Includes pathname



argv[0] – points to "<path>/copy\0"
argv[1] – points to "text1.cc\0"
argv[2] – points to "text2.cc\0"
argv[3] – is NULL pointer

More about using command line parameters in a C++ program



Example 2 :

```
/*  source  myprog.cc  ,  executable myprog    */
#include  <stdio.h>
int main (int argc, char *argv[] )
{  if  ( argc != 2 )
    { printf( "Usage : %s parameter\n", argv[0] ) ;
      exit ( 1 ) ;
    }
    printf("Starting program %s \n", argv[0] ) ;
    printf("with %d parameter(s)\n", argc-1 ) ;
    printf("First parameter is %s\n", argv[1] ) ;
    exit ( 0 ) ;
}
```


Command line:

%myprog /* Usage: - wrong (no parameters) */

Output:

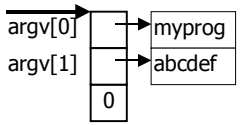
Usage: myprog parameter

Command line:

%myprog abcdef /* ← correct */

Output:

Starting program myprog
With 1 parameter(s)
First parameter is abcdef
%



Possible modifications of the program:

Instead of argv[0] → *argv
Instead of argv[1] → *++argv

Example 3 :

```
/* Source myprog.cc  ,  integer parameter  */
#include <stdio.h>
int main (int argc, char *argv[] )
{  int  p ;

    if  ( argc != 2 )
     { printf( "Usage : %s parameter\n", argv[0] ) ;
       exit ( 1 ) ;
     }
    printf("Starting program %s \n", argv[0] ) ;
    printf("with %d parameter(s)\n", argc-1 ) ;
                                          !!!
    p = atoi(argv[1]) ;
    printf("First parameter is %d\n", p ) ;
    exit ( 0 ) ;
}
```

Command line:
%myprog 12
Output:
. . . . . .
. . . . . .
First parameter is 12

An example of testing:

```
#include <stdio.h>
int main(int argc, char *argv[])
{  for  ( ; *argv ; ++argv )
    printf("%s\n", *argv ) ;
}
```

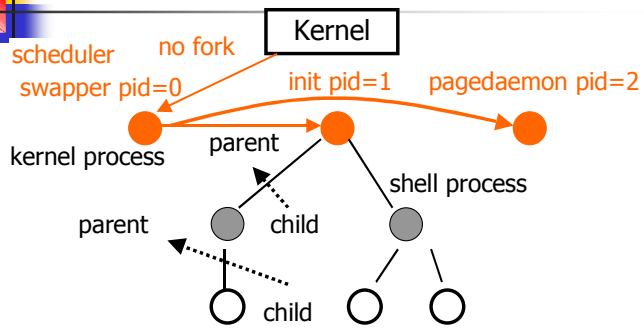
Command line:

%myprog this is a test

Output:

myprog
this
is
a
test

Process Hierarchy in UNIX

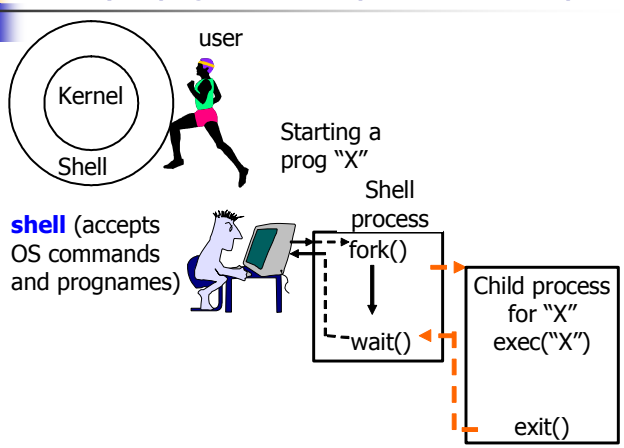


The parent/ child process relationship

- ***The parent process <u>init (1)</u> forks all UNIX processes (except 0 and 2).***

29

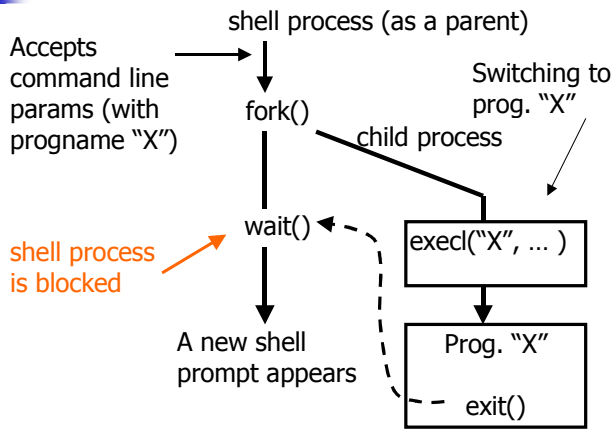
You start your program as a child process of the shell process



shell (accepts OS commands and prognames)

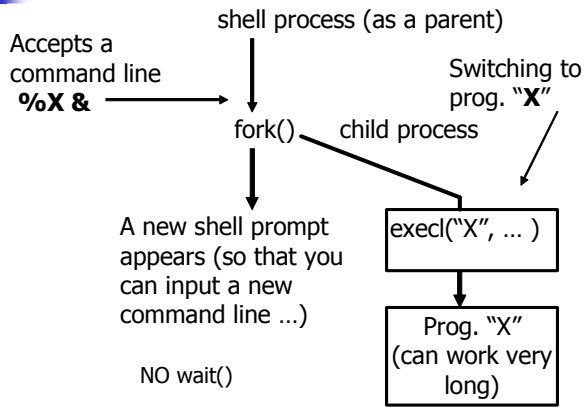
30

Shell process
(starting your program in *foreground*)



31

Shell process
(starting your program in *background*)



32

Shell

the software that is the interface between the user and
the operating system (the *interpreter* of the commands)
commands are *case-sensitive:* cd is not the same as CD

* Bourne Shell (sh), C Shell (csh), Turbo C Shell (tcsh), Korn Shell (ksh), Bourne Again Shell (bash)
* bash: the default for Linux

- Shell Commands Examples ($ == prompt character)

change your password	$ kpasswd
translate & link	$ g++ pgm8.cc -o pgm8
execute (run)	$ ./pgm8
(input redirection (cin))	$ ./pgm8 < pgm8.dat
(output redirection (cout))	$ ./pgm8 > pgm8.out **&**
(both)	$ ./pgm8 < pgm8.dat > pgm8.out

Shell Commands

list the directory	$ ls // l is "ell"
long listing	$ ll
print a file	$ lpr pgm8.cc

display a file	$ more pgm8.cc
	space bar => next screen;
	Enter key => next line
	q => quit

catenate files	$ cat pgm1.cc pgm2.cc > pgmall.cc
copy a file	$ cp pgm1.cc pgm1.cc.save
delete a file	$ rm pgm1.cc
rename a file	$ mv test.cc main.cc
make a subdirectory	$ mkdir pgm9
change directory	$ cd pgm9
move up to parent	$ cd ..
goto home directory	$ cd

Login

In the *login* box:
enter *username*, then enter *password*

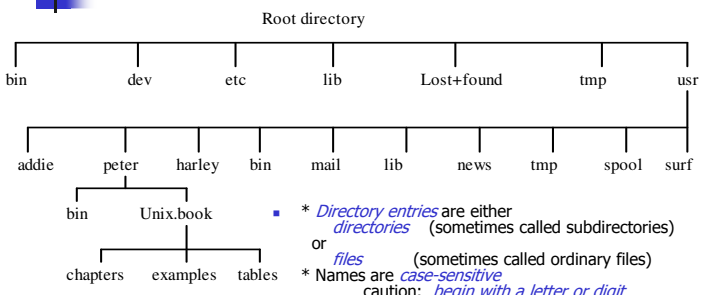
Linux Desktop

Left Click on the "Red Hat" (Start) to get Menus/programs
(can right click on desktop background to get other menus)

Click on Logout (at bottom of list) to logout!
Click on Terminal to get a terminal window
Click on Text Editor to get a text editor

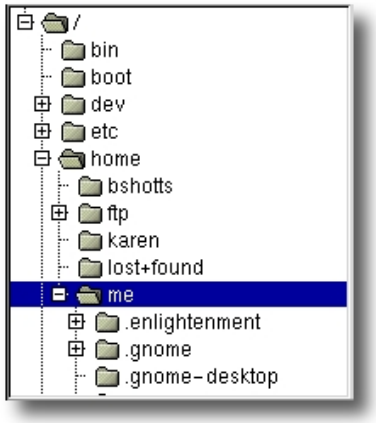
To work in a Unix (Linux, Solaris, AIX, etc.) environment, it will be a career plus if you learn to use ***"vi"*** as your text editor. In addition, if you access the Linux system remotely, you cannot use an interactive text editor; you have to use ***vi*** or ***vim*** or something similar.

File System and Directory Structure



- * *Directory entries* are either
  directories (sometimes called subdirectories)
  or
  files (sometimes called ordinary files)
  * Names are *case-sensitive*
  caution: *begin with a letter or digit*
  e.g. R25.87.ap.dat
  * Filenames can have a required extension:
  e.g. main.*cc*
  * *Current directory*
  the directory you are currently working in
  also known as the working directory
  * *Home directory*
  the directory you are assigned to when you log in
  also known as the login directory

- ***bin (binary)*** - contains *executable* programs or text files with shell scripts
- ***dev (device)*** - contains all the *special files*
- ***etc (et cetera)*** - contains the *programs*, shell scripts, and data files used for *system administration*
- ***lib (library)*** - holds collections of *standard programs and tools* that are available for general use. *Installation* programs often copy new software in this directory.
- ***lost+found*** - normally is empty
- ***tmp (temporary)*** - it is *cleared* automatically every time you *restart* Unix.
- ***usr (user)*** - directory
- ***mail*** - contains mail that has been *received but not yet read*. Each userid has its own file to act as a *personal mailbox* .
- ***spool*** - holds text files that are waiting to be printed (simultaneous peripheral operations off-line).
- ***peter, harley*** - home directories for userid's. Each user is free to create any files and subdirectories he wants within his own area of the tree.



- ***pwd*** - The directory you are standing in is called the *working directory*

```
[me@linuxbox me]$ pwd
/home/me
[me@linuxbox me]$ ls
Desktop        Xrootenv.0     linuxcmd
GNUstep        bin            nedit.rpm
GUILG00.GZ     hitni123.jpg   nsmail
```

- ***cd*** - To change your working directory
  - The *"."* symbol refers to the *working directory* and
  - The *".."* symbol refers to the working directory's *parent directory*.
- with an *absolute pathname:*

```
[me@linuxbox bin]$ cd /usr/X11R6
[me@linuxbox X11R6]$ cd /usr/X11R6/bin
```

- *relative pathname:*

```
[me@linuxbox bin]$ cd ..
[me@linuxbox X11R6]$ cd ./bin
```

- ***find*** find files

  e.g. find ~ -name list.cc -print

  e.g. find . -name list.cc -print

  -print: display full path name to file

rmdir remove a directory if it is *empty*

rm -r *remove* a directory and its contents *recursively*

i.e. remove all files and subdirectories

cat *catenate* files; send *result to standard output*

send the file *main.cc* to the monitor

```
cat main.cc
```

send files d1.dat, d2.dat and d3.dat to the monitor

```
cat d1.dat d2.dat d3.dat
```

send files d1.dat, d2.dat and d3.dat to d.all

```
cat d1.dat d2.dat d3.dat >d.all
```

(ls > ls.txt)

Wildcards in File Names

- ***?*** Match *any single character* in the name of a file

```
ls memo?
memo5    memo9    memos

ls may?rep
may.rep    may4rep    may_rep
```

******* Match *zero or more characters* in the name of a file

```
ls memo*
memo   memo.txt   memo5   memo9   memos

ls *.o
pgm2.o    stackADT.o    utility.o

ls stack*
stack  stackADT.cc  stackADT.h  stackADT.o

rm *.o

cp ~georgiev/public/Programs/pgm9/*.cc   ./
cp ~georgiev/public/Programs/pgm9/*      ./
```

41

Invisible Files

- A file whose name *begins* with a *dot (period)* is called an invisible file because it is not displayed with the "ls" or "ls -l" command. Such a file is only listed when the *"-a"* option is used *(a == all)*
  e.g. ***ls -a*** or ***ls -la***

```
[howeg89@csf6 howeg89]$  ls  -l
total 85
drwxrwxr-x   4 howeg89  howeg89    2048 Nov  9 13:18 271
-rwxrwx--x   1 howeg89  howeg89   59356 Aug 10 16:53 Doom

[howeg89@csf6 howeg89]$  ls  -alG
total 150
drwx------   24 howeg89    2048 Nov 18 15:35 .
drwxrwxrwx    2 root       6144 Oct 31 14:52 ..
-rw-rw-r--    1 howeg89      85 Sep 12 08:19
.bashrc.local
drwxrwxr-x    4 howeg89    2048 Nov  9 13:18 271
-rwxrwx--x    1 howeg89   59356 Aug 10 16:53 Doom
```

42

Shell Scripts

A *"file"* that contains *"shell commands"* to be executed by the "shell"; The commands are executed just as if you had typed them at the command line
It's a script just like an actor follows a script

Comment Lines
start the line with "#" *(# must be the 1st character)*

Includes *commands* such as *if, foreach, while, switch*
The shell has *"variables"* that can be used in a script
You can use *command line arguments*; they are referenced as *$1, $2,* etc.
To *execute the script*, the file must have *execute permission*
so, you have to do *"chmod u+x Explode"*
You can *select a shell* other than the default(bash)

```
$ cat doCompile
g++ -c main.cc
g++ -c stack.cc
g++ main.o stack.o  -o reverse
```

43

Mount a floppy

$ usermount (or: usermount &)
At this point a *window* labeled *"User Mount Tool"* will appear
Select */mnt/floppy/* and then click the *"Mount"* button
Click the *"Close"* button
The *floppy* is now *part of the Unix file system*
The floppy is now the *"directory" /mnt/floppy/*

Display the contents of the floppy:
```
$ ls  /mnt/floppy/
```

Copy a file from the floppy to your current directory:
```
$ cp  /mnt/floppy/main.cc  .
```

Copy a file from the Linux system back to your floppy:
```
$ cp  pgm8.a  /mnt/floppy/
```

44

More Information

- You can find more info at:

http://www.linuxcommand.org/index.php

45