UNIX for MPE Admins

Presented by:
Fred Mallett
frederm@famece.com
FAME Computer Education
250 Beach Blvd
Laguna Vista, TX 78578
956-943-4040

Seminar Outline

Chapter 1

The Basics

Specific Objectives

After completion of this chapter the student will be able to:

Login and out.
Define common UNIX and Computer terms.
Use terminal emulators.
Issue basic UNIX commands in a UNIX shell.

Chapter Contents

Overview of UNIX

Terminology

Login

Your Initial Environment

Intro to Window systems (CDE)

Terminal emulators

Common Command Introduction

I/O redirection

Overview of UNIX

UNIX is an industry standard operating system due to portability of the operating system, of applications, of users.

UNIX Milestones:

1968-1971 Early development in New Jersey at Bell Labs

1973 UNIX rewritten in the C language. (Major cause of acceptance outside lab, first high level language OS?)

1977 Interactive Systems Corporation started reselling UNIX

1977 University of California at Berkely distributed their pascal interpreter (Later produced the BSD variant of UNIX, final release was 4.4BSD)

1977 - 1982 Several variants combined into what was known comercially as UNIX System III. Later several features were added to System III and it was dubbed System V

1983 AT&T announced official support for System V. Currently System V release 4 (System VR4)

Many UNIX clones developed along the way:
Don't require AT&T licenses
Have UNIX look and feel

1995 Posix command set:
All commands have a posix defined output, and identical actions for a set of options.

1996 CDE (Common Desktop Environment)
A single window system, with integrated desktop tools, such as mailtool, and calendar, to run on all vendor platforms.

1996? Linux
UNIX without the big corporation behind it.
Reliable operating system on an Intel server.
Back to the speed on UNIX, but now growing with features.

Terminology

Take notes on terms you are un-familiar with.

Workstation   Node   Host
File server   Application server   Data server
Multi-user   Mainframe

CPU   Physical Memory   Virtual Memory
Disk   Mount point   Drive letter

Program   Process   Job
Time Slice   Multi-processor   Threaded

File   Directory   Tree
Home directory   Current directory   Parent directory
Pathname   Root Directory   Root User

ASCII   Text   Binary
Cut   Copy   Paste   Move

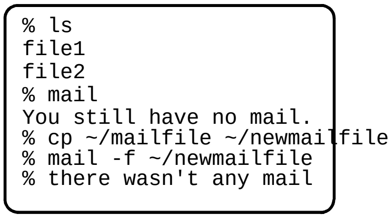
Shell   Command Interpreter

X   Xterm   X Terminal   Terminal window

Network File System (NFS)
Network Information System (NIS)

What is a Window system

Terminal based user interface:

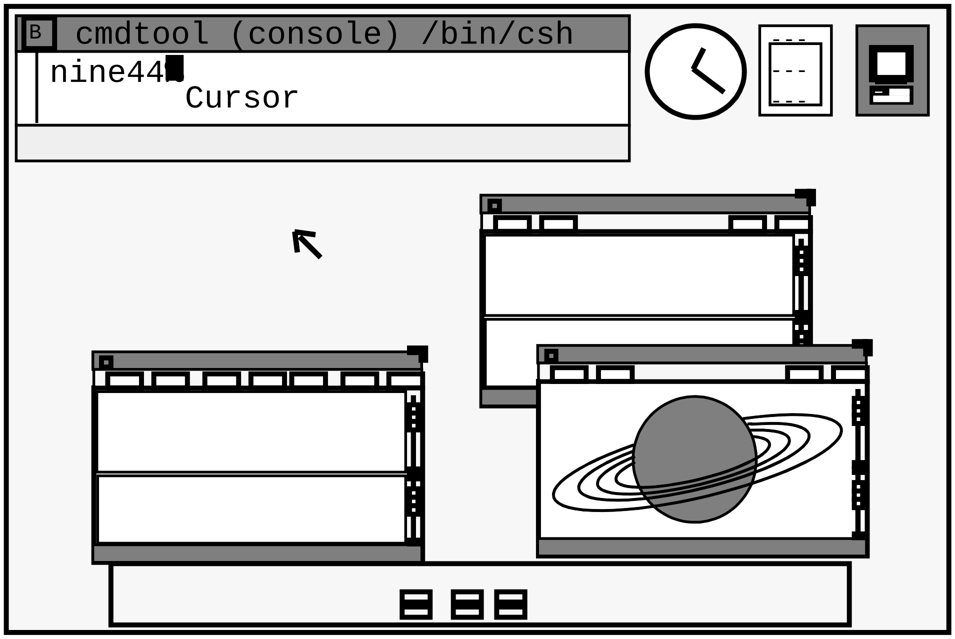


```
% ls
file1
file2
% mail
You still have no mail.
% cp ~/mailfile ~/newmailfile
% mail -f ~/newmailfile
% there wasn't any mail
```

Window System

The program that receives, and de-multiplexes user input
The program that displays system (program) output
The program that controls rectangles on the screen

Window based user interface:



Terminal Emulator

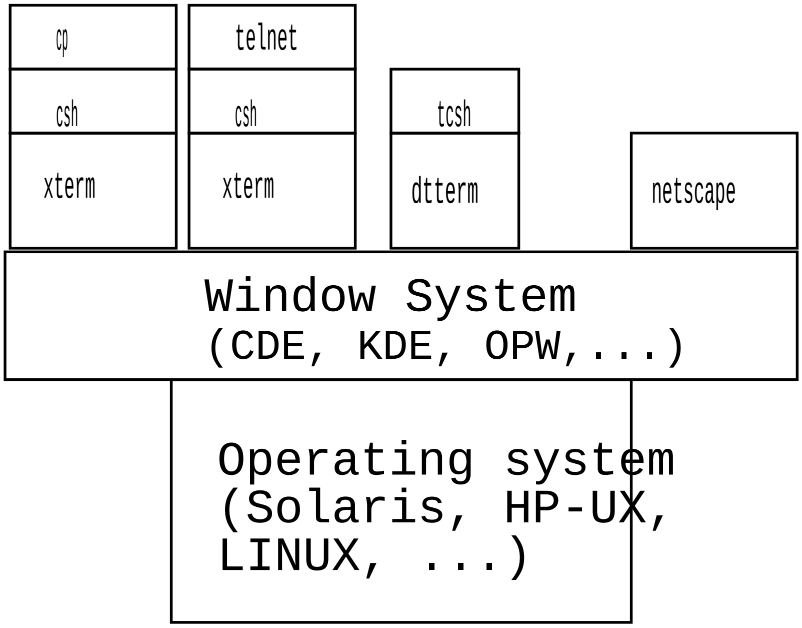
A program that allows a window on one computer to act as though it were a terminal connected to the computer. They can be run local or remotely displayed on another computer in the network.

What is a shell

Shell

A shell is a command interpretor. This means it accepts keyboard (standard) input, interprets what you enter as a command, then invokes that command (program).

Almost all programs are invoked through a shell, although the window system can also invoke a program when you click a menu item, or button on the screen.



Login

Login is required in UNIX to inform the system which user will be using the system. This is needed, as your user name is associated with a HOME directory that contains startup files that set up an environment specifically tailored to the user. Your account is also used to:

Determine access rights to existing files
Assign ownership to files you create

There are three types of Login methods on UNIX hosts:

Console or command line:

Found on hosts such as those using Openwindows as the graphical user environment. Windowless screen with text prompt for login. The mouse is inactive during the login, and a login startup file is used to start the window system.

Login:
Password:

Windows:

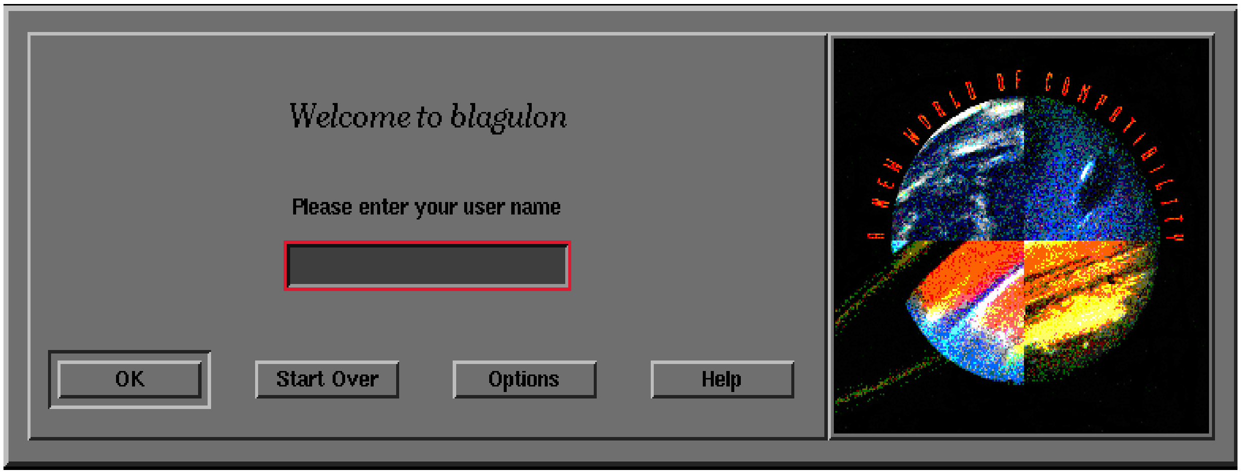
The graphical user environment manages the login session. In this type of login, the window system is already running, the mouse is active when the login screen is displayed.
Be sure not to have spaces before the login name.

Remote:

At a UNIX shell prompt, use the command *rlogin* or *telnet* to login to another host across the network, thus using that hosts CPU to execute commands and programs, with the output on your screen.
This is also used when accessing a UNIX program from a Microsoft Windows machine.

Logging in to a window system

Note: bitmap shown is not the Vendor specific version.



Welcome to blagulon

Please enter your user name

A graphic similar to above appears on the login screen, point to the text input box and click the left mouse button to select the box, then you can enter your login name. Pressing Return or clicking OK brings up a similar screen that asks for the password. There are also several options, similar to those listed below.

Restart server: Restarts the X display server

Command Line: Gives a console terminal based session

Session: Some systems give a choice of window systems, such as:
      Common Desktop Environment
      OpenWindows
      Same as last session

Failsafe Session: Starts a session with default startup files
       to provide one terminal, no window manager is
       running. Used to fix login problems.

Languages: Choose other languages for displaying text dialogs

Logout

To logout, you must save all data in applications.

Then issue the command that causes a logout to occur.

> This command stops all processes you have running, and might execute a special *logout* file to perform commands you assign.
>
> You can start processes in such a way that they ignore the logout command, so they will continue to process after logout.
>
> The login prompt re-appears for the next session.

Logout with the *EXIT* button in the lower center of the screen (on CDE), or from the right mouse button menu (in Openwindows), depending on the window system in use. From a terminal (or remote) session, issue the *exit* command, or exit the application.

On Openwindows, you must exit the window system with the Workspace menu *Exit* choice, then also issue the *exit* command at the console shell prompt (Unless *Openwindows* is started by the CDE login program shown on the previous page).

In a workstation session logout, when the login prompt re-appears, you are logged out, until it appears, you are still logged in.

Your Initial Environment

This is controlled by files, called startup files. There are three major types:

Shell startup files:

Live in your $HOME directory, filename starts with “.”

For the *csh* shell, there are two primary files:

.login *.cshrc* (or *.tcshrc* in the *tcsh*)

These files contain commands that you want executed at login time (*.login*), or at every shell startup (*.cshrc*).

Note that if you are using a window system, the window system startup file must either read the shell login file, or have any necessary environment settings.

Window system startup files and directories:

Live in your *$HOME* directory, name starts with “.”

Controls behavior of the window system at startup.

VUE:	*.vueprofile*
CDE:	*.dtprofile*
OPW:	*.openwin*

Application startup files and directories:

Typically live in your *$HOME* directory, often the name starts with “.”, and is named after the application (*.newsrc, .emacs*)

Controls behavior of the application at startup.

Choosing sessions in CDE

A session in CDE terms means the windows currently open, the applications running, and preference settings such as fonts, and colors.

What session do you get at login time?

The default action is to return to the last session, this is called ”current” session. All CDE aware applications running will be restarted.

You can also set the system to return to a pre-saved ”home” session.

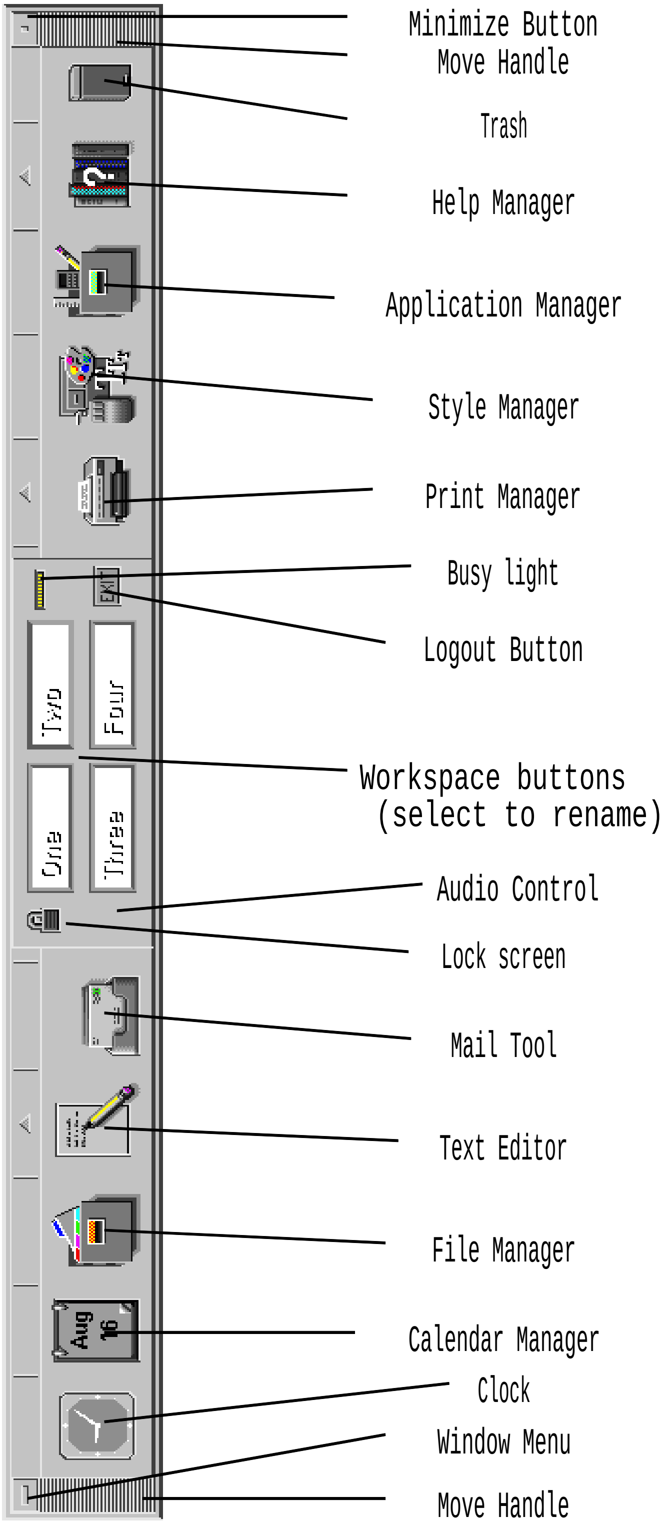
To set a home session, you use the following sequence:

1) Login and set all windows as you would like them
   to appear when you login
2) Click on the style manager button (art palette)
3) Click on the startup button
4) Click on ”Set home session”, then confirm
5) Click on ”Return to home session” or
   ”Ask at logout”
6) Choose if you want to be prompted for confirmation of
   logout intentions

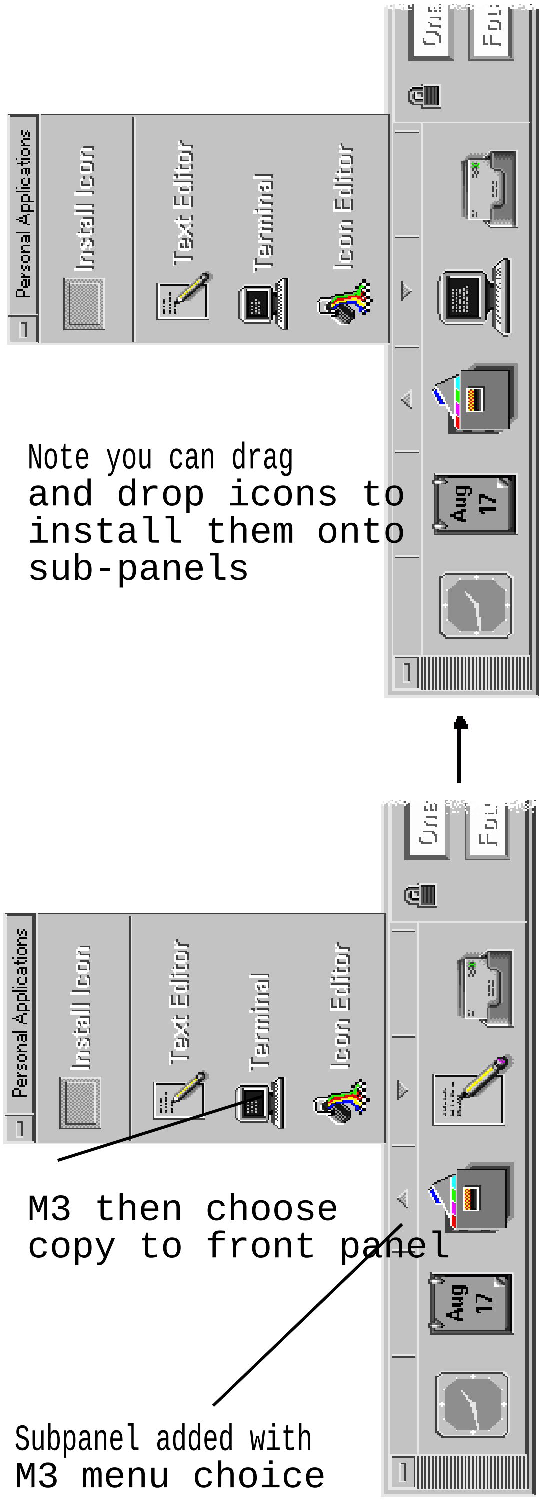
The next login will follow the choices you have made.

Note: This is only for when logging into the primary display of a worstation, or an xterminal type host. This has no effect when using telnet, rlogin, or sitting at a Win9x or WinNT host and starting a single window session (like an application or xterm).

CDE Front Panel



CDE Front Panel



Note you can drag and drop icons to install them onto sub-panels

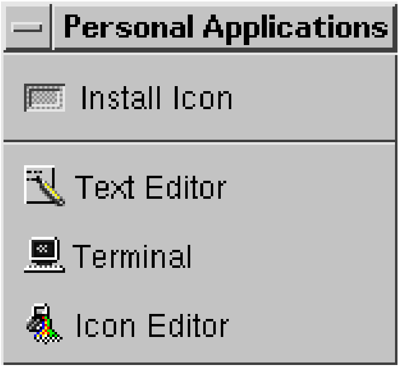
M3 then choose copy to front panel

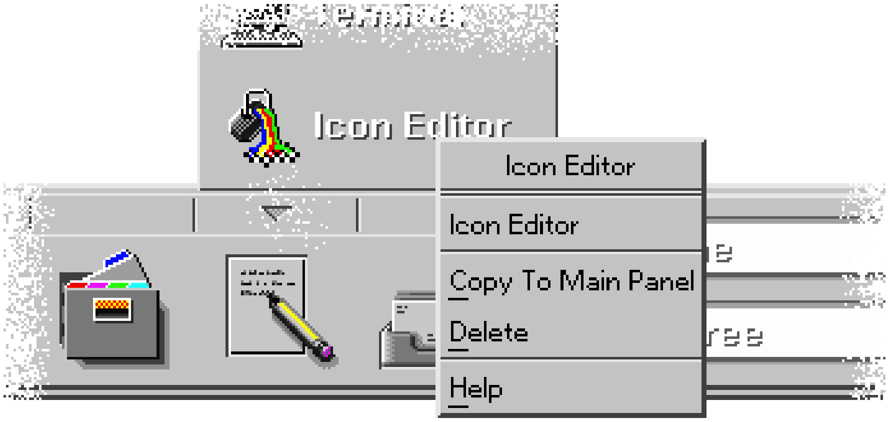
Subpanel added with M3 menu choice

CDE Front Panel User Customizations

Changes made to the front panel through menu items (add subpanel), replaced icons (move to front panel), and drag and drop (install icon) are easily removed.

Use the Application Managers Desktop tools application called “Restore Front Panel”. Changes made by editing files (Advanced customization) will not be affected. Note that “Restore Front Panel” does not work completely, as it does not delete the *.dt/sessions/dtwmfp.session* file , or remove the workspace resource changes. (It does remove *$HOME/.dt/types/fp_dynamic)*





Types of terminal emulators

Console

Only one should be running at a time. Acts as any terminal emulator tool, but displays system console messages

There should be exactly one running

If closed by accident, restart using the *-C* option of the shell command used to start that emulator:

```
% dtterm -C -ls  -name console &
% xterm -C -ls  -name console &
% cmdtool -C -ls  -name console &
```

dtterm

A graphical window to run a UNIX shell in. Allows scrolling of history of commands and output, as well as text editing functions. Text editor based tty command line interpreter

Has limited text editing facilities.

<M3> will post a cascading menu of options

Found on all systems that support CDE (or KDE on LINUX)

xterm

A graphical window to run a UNIX shell in. Emulates one of two terminals: Dec or Tektronix.

Found on almost all UNIX systems.

Use <ctrl>M3, <ctrl>M2, <ctrl>M1 to post menus in an xterm to control operation, such as scrollbars, and font size.

commandtool

Specific to Sun Openwindows.

hpterm

Specific to HP-UX (running either CDE or HPVue).

dtterm

dtterm acts as a combination of the OpenWindows Commandtool and Xterm as far as capabilities go.

There are three text editing capabilities (mostly used to re-issue commands):

Selecting text: Drag the M1 key to highlight text

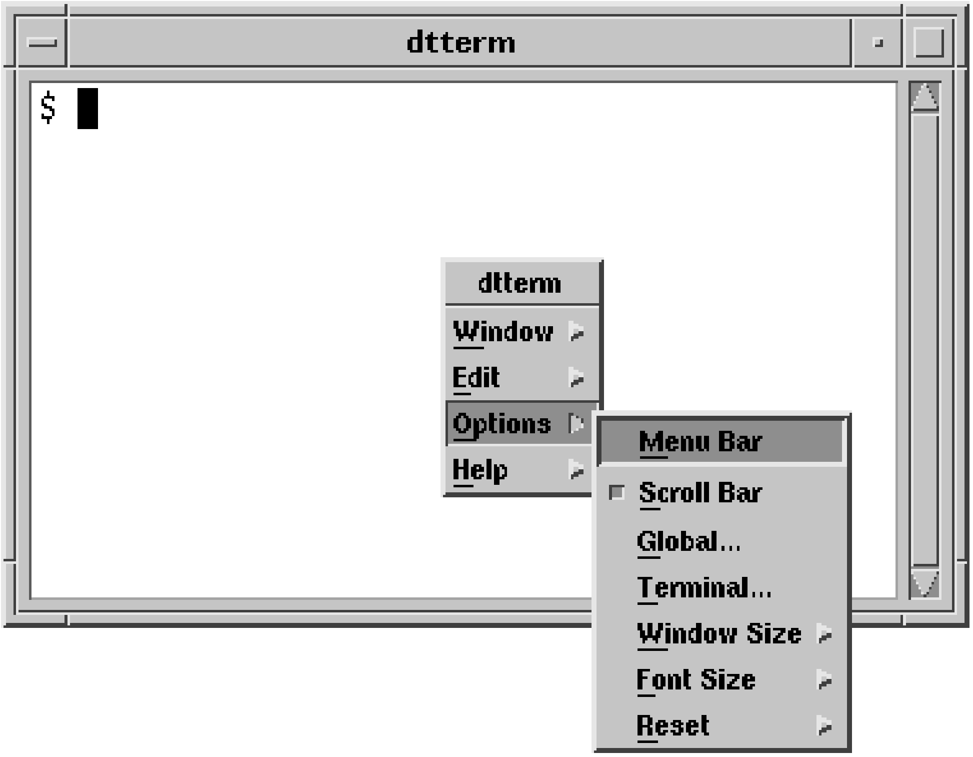
Use <esc> to de-select text

Copying text: Select the text to be copied, then use the

M2 button to paste the text at the current input

location. This can be in another window

<M3> posts the menu seen below



From menu choices you can change fonts, enable scroll bars, window size by column and row, create another dtterm, perform tty resets and clear, plus others.

dtpad

The CDE window system includes a simple text editor program. Other window systems on UNIX have something similar (*textedit* on Openwindows).

It has all the basic editing functions: Cut, Copy, Paste, Replace, Search, Spell check and fix, plus some simple formatting capabilities such as left/right/center justification, and word wrap.

Text selection notes: double click selects a word, triple = line

M1 highlight, M2 ”copy/paste” of dtterm works here also

M1 highlight, <Alt>M2 underscore does a copy-overwrite

Has the ability to do search and replace

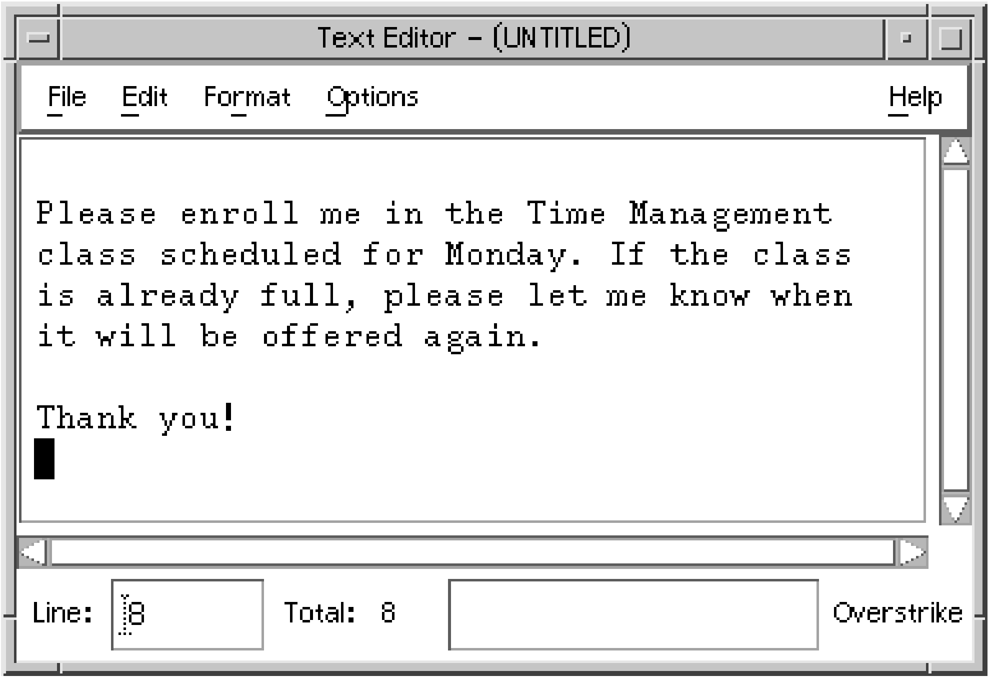
Can use a null replace string in replace all as a delete all

Overstrike mode is controlled by menu or insert char key

Has a menu based select all function

Allows “emacs” editing key strokes

Shows line numbers when you select status bar mode

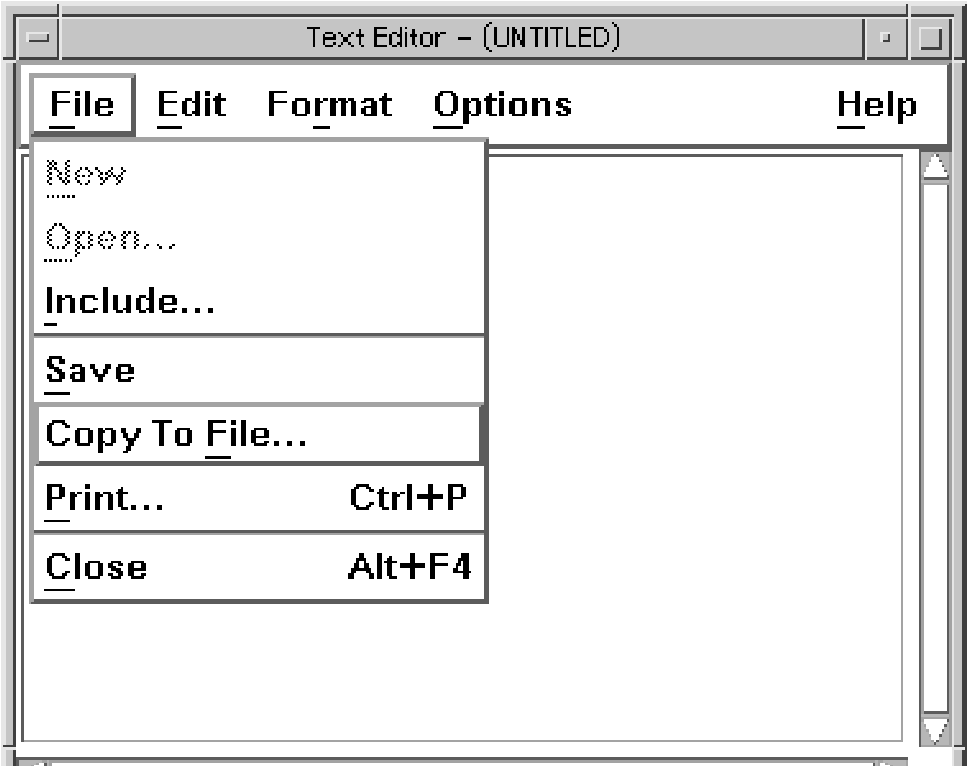


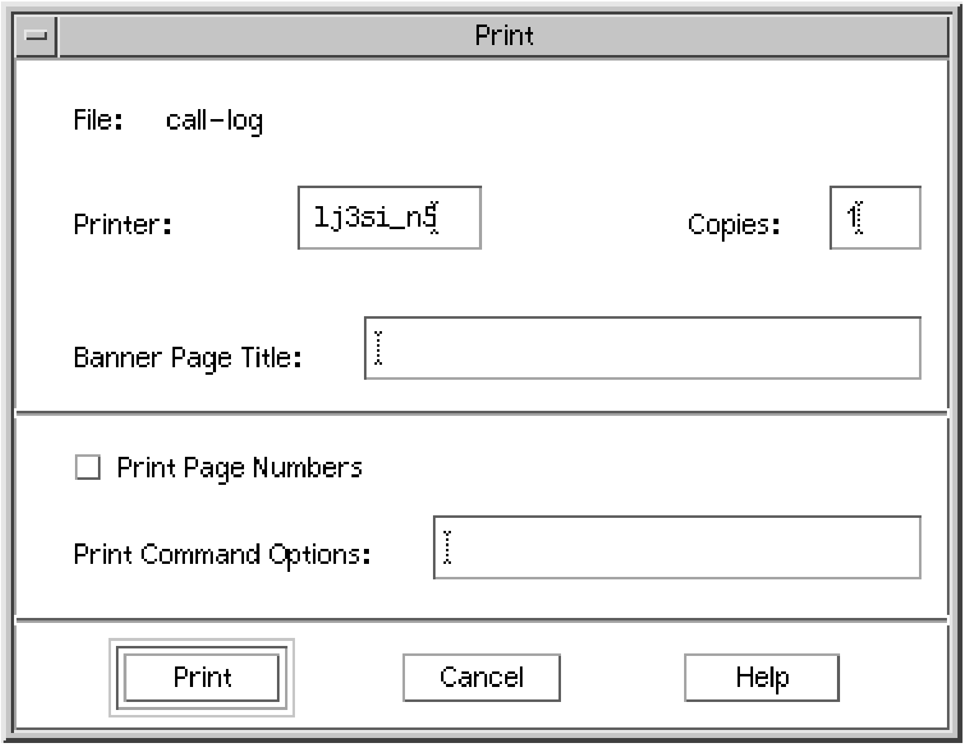
Please enroll me in the Time Management
class scheduled for Monday. If the class
is already full, please let me know when
it will be offered again.

Thank you!

Line: 8 Total: 8

dtpad





Print

File: call-log

Printer: lj3si_n5

Copies: 1

Banner Page Title:

□ Print Page Numbers

Print Command Options:

Remote application launching

X window system applications (such as terminal emulators) have the ability to re-direct its windows to appear on another display in the network.

The other host (xterminal, PC with X windows, or another workstation) will handle sending all keyboard input and application output to the correct places.

This is typicaly done when you need to execute a program on another host, and have it displayed on the host you are sitting at.

The reasons for this include:

Dis-similar hardware platforms
Need access to a powerful server
You are using an X terminal
Need to test a client on a different OS release
You are using a PC as an X terminal

The steps needed to accomplish this are:

1) The X server running on the display you are sitting at must allow access to the remote server that will run the client. This is done with *xhost* on UNIX boxes, and X terminals, it is either not needed, or done with menu choices on PC’s.

2) The X client must be “told” where to display, usually with the *DISPLAY* variable, but the *-display* command line option works for some clients.

3) Start the client program in the background.

Example sitting at the host *swift*, and starting an application over on the host *puxy* to display where you are sitting:

```
% xhost +puxy
% telnet  puxy
...
% set  DISPLAY  swift:0
% ileaf6  &
% exit
```

Running clients remotely

The error message:

```
% xterm -display swift:0  &
Xlib: connection to ”swift:0.0” refused by server
Xlib: Client is not authorized to connect to Server
Error: Can’t open display: swift:0
```

Usually means that the host being displayed to (*swift* in example above) is not allowing remote display from the host you are exectuing the client on. (run *xhost*)

The error message:

```
% xterm -display swifty:0
Error: Can’t open display: swifty:0
```

Usually means that the host is invalid, or cannot be reached. If the field after *display:* in the error message is empty, as seen below:

```
Error: Can’t open display:
```

The problem is usually that the *DISPLAY* variable is not set, or you did not use the *-display* option to the command.

UNIX Command Philosophy

By default, UNIX commands are designed to do what you say, and never check to be sure you meant it.

- Some commands have a ’make sure I meant it option’ that prompts before doing anything harmful. By default, all ’safety’ options are turned off.

The power of UNIX comes from the combining of components

- To allow for combining of commands, many UNIX commands
  follow these rules:
  - Use standard input/output
  - Print no extra verbage
  - Perform one function only

Some UNIX commands will take input from either a file, or standard input (which could be the keyboard, or another program).

- These commands typically filter or modify text.

Some UNIX commands write all results to standard output (this allows you to send that output to another program (that takes standard input), or to a file, or just to view the results.

- These commands typically filter or modify text.

Some UNIX commands take no standard input, they must be given a filename ot act on. Other take no filenames, and must be supplied text on standard input.

Some UNIX commands write no output, unless it is an error message.

- These commands are typically those that by their nature have some lasting effect.

Common Command List

Most UNIX commands can be categorized as follows:

Directory commands:

Listing	*ls*
Where am I	*pwd*
Change directory	*cd*
Create directory	*mkdir*
Delete directory	*rmdir*

File manipulation commands:

moving, renaming	*mv*
copying	*cp*
show size	*wc*, *ls -l*
compare	*cmp, diff, sdiff*

File filtering commands:

Show contents of file	*cat*
Show file with pagination	*more*
Show matching lines	*grep*
Sort file	*sort*
Merge files	*sort -m*
Print file	*lp, lpr*
Show selected fields	*cut*

Programming related commands:

Compile a C program	*cc*
Syntax check a C program	*lint*
Maintain groups of programs	*make*
Show printable strings in a binary	*strings*

Command grouping (Pipe) example:

Show selected fields on selected lines:

grep word file| cut -f2 or *cut -f2 | grep word*

Basic Command syntax

Format

```
        % command [-options] [arguments]
```

A flag argument invokes optional capabilities of the command, it is always preceded by a -. Usually called an option.

Note, the following symbols are often used when describing command syntax (these meaning are only for describing syntax):

[n]	*n is* Optional
n|m	Use either *n* or *m*
n...	You can use many *n*’s

Arguments are data passed to the command, commonly a pathname, expression, or special string

Always separate arguments with spaces or tabs:

```
      % ls -l      ~ted
      % man -k  directory
```

Multiple options may usually be strung together:

```
      % ls -a  -s        ~ted
      % ls -as     ~ted
```

Unless the options requires a sub argument:

```
      % grep -vl -f mycommands  textfile
```

In some commands even this is allowed:

```
      % command -vlf mycommands  textfile
```

where *mycommands* was a sub argument to the *f* option

I/O redirection

If a command generates text output, you can redirect that output using this format:

```
command   re-direction-symbol   target
```

The two most common symbols are:

```
>     Sends output of the command to a file
|     Sends output of the command to another command
```

Of course, if you send the output to another command, that command must accept text input.

Some reasonable examples:

```
% cat newlist  phone   > phone.new
% grep  miller  phone.new  > millers
% ls  | wc -w
% grep  NV  area-codes.txt > nevada
% grep  P_INPUT   result.1   result.2  |  grep 82  > test-inputs
```

Some bad examples:

```
% cp  phone.new  > phone.test
% grep miller  phone.new | mv  phone.test
% grep test  outputlist | ls
```

Command Examples

```
% cd
% pwd
/disc/users/fred
% cd TI
% ls
all_ti_courses_desc.txt.1.13.95  ti_ksh    explr-inet
% ls -l
total 38
-rw-r--r--   1 fredm    training   16720 Jan 13  1995 all_ti_cours
drwxr-xr-x   4 fredm    training    1024 Apr  9 09:08 explr-inet
drwxr-xr-x   2 fredm    training    1024 Apr  9 09:08 ti_ksh
% ls -F
all_ti_courses_desc.txt.1.13.95   ti_ksh/   explr-inet/
% cat > new_file
Hello,
 This text goes into the file
until I hit a <ctrl>d
bye
% ls -s
total 40
   2 new_file
  34 all_ti_courses_desc.txt.1.13.95
   2 ti_ksh
   2 explr-inet
% grep goes new_file
 This text goes into the file
% grep goes new_file | cut -d″ ″  -f2,3
This text
% wc  new*
4 13 63 new_file
% pwd
/disc/users/fred/TI
% cd ..
% pwd
/disc/users/fred
% mkdir test
% cd test
% ls
% cd ..
% rmdir test
% rmdir TI
rmdir: TI: Directory not empty
```

Chapter 2

Accessing Files

Specific Objectives

After completion of this chapter the student will be able to:

Navigate the file system tree structure with both UNIX commands, and the File Manager program of the chosen Window System.

Chapter Contents

Methods of file access in UNIX

Naming tree

File Access with the File Manager

File Access with Commands

File and Directory access commands

Home directory

Methods of file access in UNIX

There are two primary methods:

- Using UNIX commands.
  - Harder to learn
  - Tremendous Power and flexibility
    - (All commands, all options available)
  - Keyboard access

- Using Window System File Managers
  - Easy to learn
  - Less power, little flexibility
    - (Many commands, few options available)
  - Mouse access

Naming tree

Hierarchical structure that organizes every object

Used as a map to locate objects in the system

The root level is the upper-most level of a node

An upper level directory is a directory that lives
(is rooted at) the root level of a node

Pathnames

A pathname is a description of the route to a file or directory on some disk in the network

Two Types:

Absolute:

Describes one specific OBJECT no matter where it is used from in the network file system. Starts at the root level.

Relative:

Describes an OBJECT using a path relative to the current
location in the tree structure. The object it describes could
differ depending upon where it is used from.

Begin with a symbol

This determines where to start the search

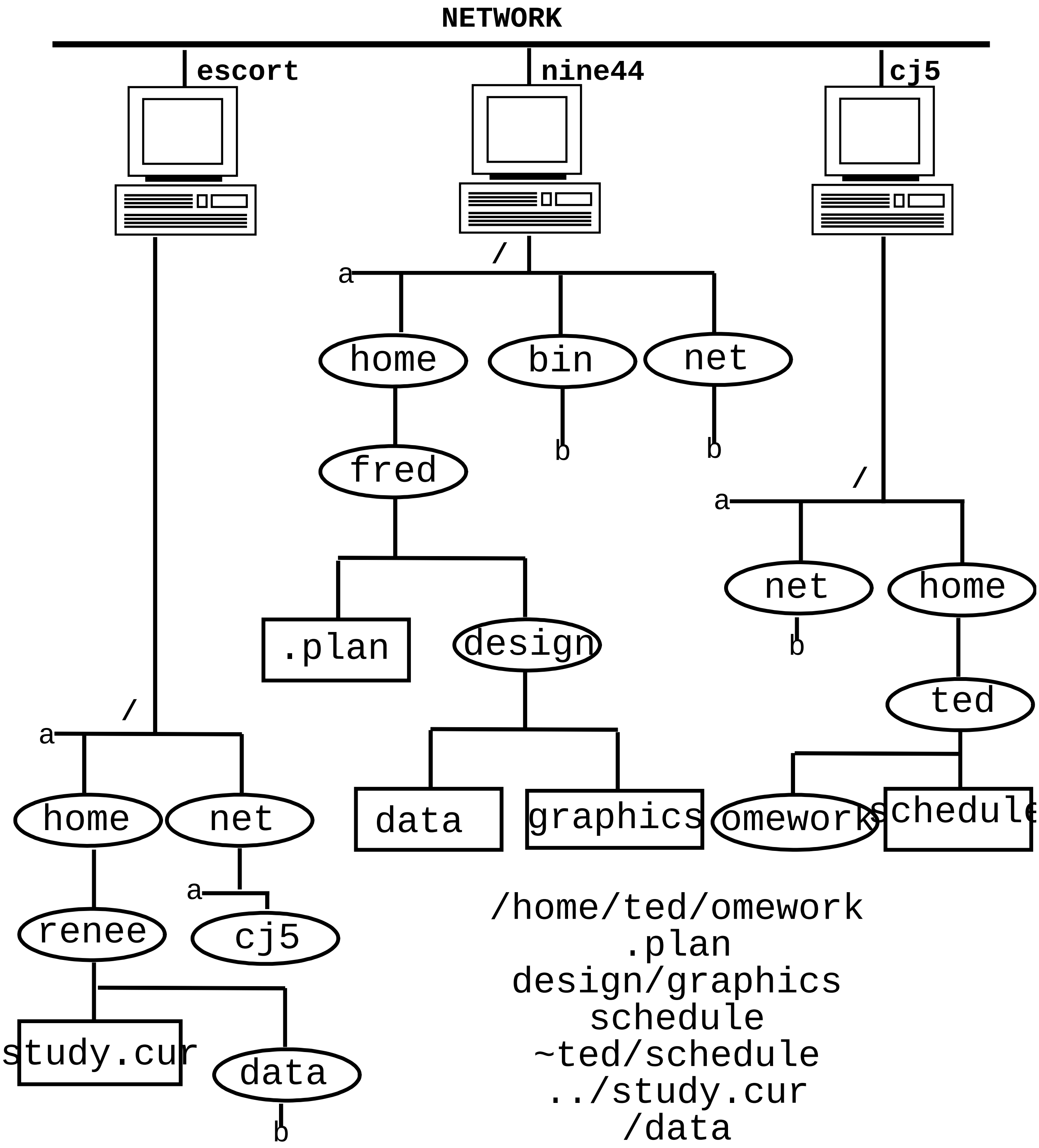
End with an object name you want to act on (the leaf)

Contains Object names separated by **/**

Objects can be files, or directories

/home/ted/omework

Pathname Examples



/home/ted/omework
.plan
design/graphics
schedule
~ted/schedule
../study.cur
/data

Which directory are each of these pathnames valid from (if any)?

Pathnames starting symbols

If a pathname starts with:

/net/host This is used in an NFS network to access files
on another host. Some systems are configured to use
many other ”mount” points to access directories on
another host. Commonly */user* or */home* might have
entries for every users home directory, no matter which
system the directory actually resides on. Other examples
of NFS use are:
/designlib/some-design
/home/bgates

/ This tells the system to start the search at the
root level and the first object in the pathname is
an entry directory (such as home or usr)

. This tells the system to start the search in your
current working directory

Use: *cat* **.***/filename* *cp* */ct_lib/ctr/ctr2* **.**

.. This tells the system to start the search in the
parent of your current directory

Use: *cd* **..** *cd* **../../***design* *ls* **..**

~ This tells the system to start the search in this
processes home directory

Use: *cd* ~/*data* *mv* */x/y/z/q* ~

~*name* This tell the system to start the search in
user *name*'s home directory: *~dshaw*

General File manager actions

CDE uses *dtfile,* Openwindows also has a filemanager.

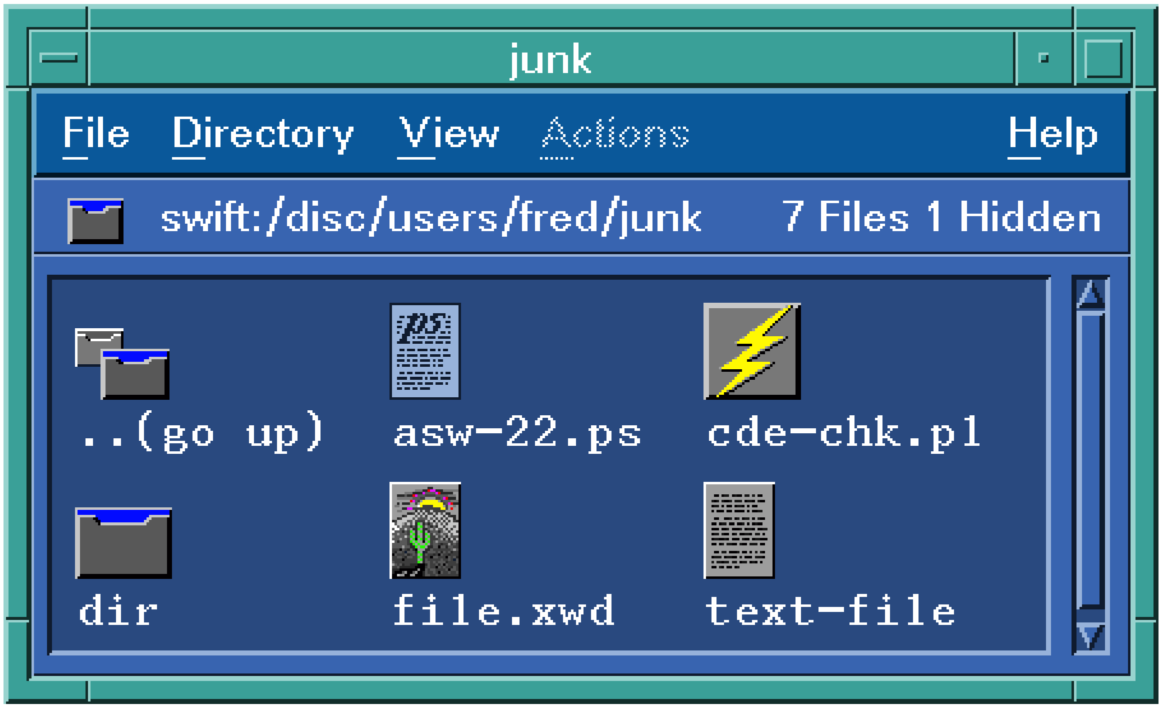
These tools provide graphical display and manipulations of the UNIX file system. They can copy, move, find, open, link, remote copy, close, delete to trash, un-delete, and print files. Files can be directories, data, or executables. You can set and save preferences for which files are displayed, or hidden.

Methods used in filemanager:

```
  Selection of objects:
        Click left mouse on icon
        Drag left mouse to create a drag box
        Select additional items by <ctrl>drag or
  click M1
        Un-select from a group with <ctrl>drag or
  click M1
        De-select everything with the escape key
  Move and copy via Drag and drop:
        Typically a drag is a move, and <ctrl>drag
  is copy
        and in some tools, <shift>drag is a
  symbolic link.
        You can drag a file (or selected group)
  to:
        File manager window in a different
  directory
        Trashcan   dtpad or textedit
        Print Icon Desktop    Mail
        And other action Icons
  Double-Click, or M3 pop-up menus:
        DC does default action from pop-up menu,
  typically
        this default is whatever is apropos for
  the file type:
              text file-> open in editor
              directory -> CD to it
              executable -> execute it
```

```
           picture -> display it
           html -> start a browser
           etc....
File rename can be performed by selecting name,
then type
```

File manager



Sample file manager tool above, sample menu choices below:

File

New...
Rename...
Copy
Properties...
Find...
Put on Desktop
Annotate...
Review Annotation
Delete to Trash
Show Trash...
Close

Directory

New...
Reread
Fast change to
Change to...
Home
Up
Remote Systems
Terminal

View

New
Cleanup
Select all
Unselect all
Show Hidden File
Modify Filter li
Set Preferences.
Save Settings

Actions

Open in place
Open new view

EDIT
VIEW
PRINT

EXECUTE
EDIT

File Access with the File Manager

Note that most of the below actions can be done from menu choices also, here we list the mouse methods.

To rename a file/directory:

Select the name, not the icon, wait 2 seconds, a text box will appear. Be sure not to leave spaces in the name.

To copy a file:

The directory you want to copy to must be visable, either open in another file manager view, or as a directory icon. Press <ctrl>, then use a select drag of the file, and drop it on the directory target. To copy multiple files, hold <ctrl> and click select, or drag with select to get a “rubber band”,

To move a file:

Do as listed in copy above, but do not hold <ctrl> while dragging the file.

To delete a file:

Drag the file to the trashcan, and drop it (periodially empty trash).

To print a file:

Drag the file to the printer Icon.

To read/edit a file:

Select the file, then use the menu and choose edit or read. If the file is an ASCII file, and does not have execute rights, it will appear as a sheet of paper, in this case you can double-click select on it.

To execute a program:

Double click M1 on it if it appears as an executable.

File Access with Commands

All file and directory management commands can be issued from any terminal emulator type, and any shell type (*sh, csh, ksh, tcsh, bash,* etc...)

All commands will accept any pathname type that is valid for the current directory the shell is in.

To issue any of these commands, you must know the syntax of the command, the pathname of the object you want to act on, and the target pathname if it is a copy/move/rename type operation.

It helps to know what options are available to control the operation, and what happens by default.

Remember, file commands are usually destructive, unless you specify a ’safety’ type option.

File Access with Commands

Note that most of the below actions can be done from menu choices also, here we list the mouse methods.

To rename a file/directory:

Use the *mv* command

To copy a file:

Use the *cp* command

To move a file:

Use the *mv* command with a directory in the target pathname

To delete a file:

Use the *rm* command

To print a file:

Use the *lp* or *lpr* command

To read/edit a file:

Use one of many commands (cat, more, vi, emacs, nedit,...)

To execute a program:

Type in the name or pathname of the executable.

File and Directory access commands

cd [pathname]

```
cd /user/fred    Sets current directory to
arg[1]
cd           Sets current directory to $home
cd  dirfile  Searches for dirfile in $cdpath
             if found, sets current directory in
{c|k}sh
```

pwd Prints current directory

```
% pwd
/user/fred/desktop
```

mkdir [-p] [-m mode] pathname

```
     -m  Specify creation rights   ( default mode 777)
     -p  Create entire path specified by pathname
     Older bsd versions don’t take any options
```

```
mkdir name
mkdir  /home/luigi/dat.1
mkdir  x  x/y  x/y/z  or  mkdir -p x/y/z
```

rmdir [-p] name Removes empty directories

```
      -p    Removes this, and all parent
directories that
            become empty as a result
```

```
% mkdir -p x/y/z
% rmdir -p x/y/z
rmdir: x/y/z: Whole path removed.
```

File and Directory access commands

ls   List Directory

```
ls [ -abcCdfFgilLmnopqrStux1 ] name ...
ls
ls  /home
% ls  -la  /home/fred
drwxrwxrwx       2 root      bin  6144 Sep 27
13:12      .
drwxr-xr-x       2 root      bin  1024 Feb 14
1989  ..
-rwxr-xr-- 1 fredm trng       33 Feb 27 1989
      .dtprofile
-rwxr-xr--       1 fredm trng       22 Jun  1
09:25      .kshrc
-rwx------ 1 fredm trng       12 Sep 24 1989       .pro
drwx--x--x 4 fredm trng     1024 Feb 28 1989       bin
drwx------ 2 fredm trng     1024 Feb 14 1989       data
drwxr--r--       11fredm trng     6144 Sep 27
12:14      desktop
-rwxrwxrwx       5 fredm trng      427 Feb 24
1989  ed_svcs
```

Some commonly used options:

a   List all files   A   All but not .
and ..

C   Multicolumn   d   Directory, not
contents

l   Long format   1   force output
1 per line

F   dir=/   slink=@   executable=*   regular
file =

```
% ls -FC
$desktop* Xstuff/   admin_alias*   adus
bin/      compilers@     data/                des
functions/     hp_driver/     hp_prof/        inf
junk/          mail/          mailrc*         req
```

```
scripts/  sys_admin_sig*
```

File and Directory access commands

touch Creates files, but also updates time modified, or time used, if file exists

touch [-amc] [-r ref_file | -t time] file

-c Do not create if file does not exist
-a Only update time used (accessed)
-m Only update time modified (default is -am)
-r Set times to same as ref_file
-t Set times to specified time

touch program.c Updates the last time modified or creates
cat > file.txt Allows you to type text into a new file until a <*ctrl*>*d* is entered.

rm Remove, or unlink files

rm [-fiRr] file ...

-f Do not prompt for confirmation
-i Prompt for confirmation
-r or *R* Recursively remove subdirectories, and files

You may remove a file that exists in a directory you have write rights on. The rights and ownership of the file does not matter.

rm x y ~/data/z /tmp/my_file Delete these files
*rm -i *test* Delete files whose names end with *test*
rm -r this_dir Delete entire structure below *this_dir*

File and Directory access commands

mv   Move files

```
mv [-i|f]  file [file...] directory|file
     -f    Do not prompt for confirmation
     -i    Prompt for confirmation
mv file1 file2
mv file2 file3  dir3/subdir
mv dir3 dir4
```

cp   Copy files

```
cp [-fip]  file1   file2
cp -r|R [ -fip ] dirfile1 dirfile2
cp [-fip] file1  [file2...]  dirfile
     -r|R  Recursive
     -f    Do not prompt for confirmation
     -i    Prompt for confirmation
     -p    Preserve modification times, owner, modes

cp file1 file2
cp file2 file3  dir3
cp  -r  dir3   dir4
```

The *mv* and *cp* commands follow these rules when deciding to overwrite, or create the target object:

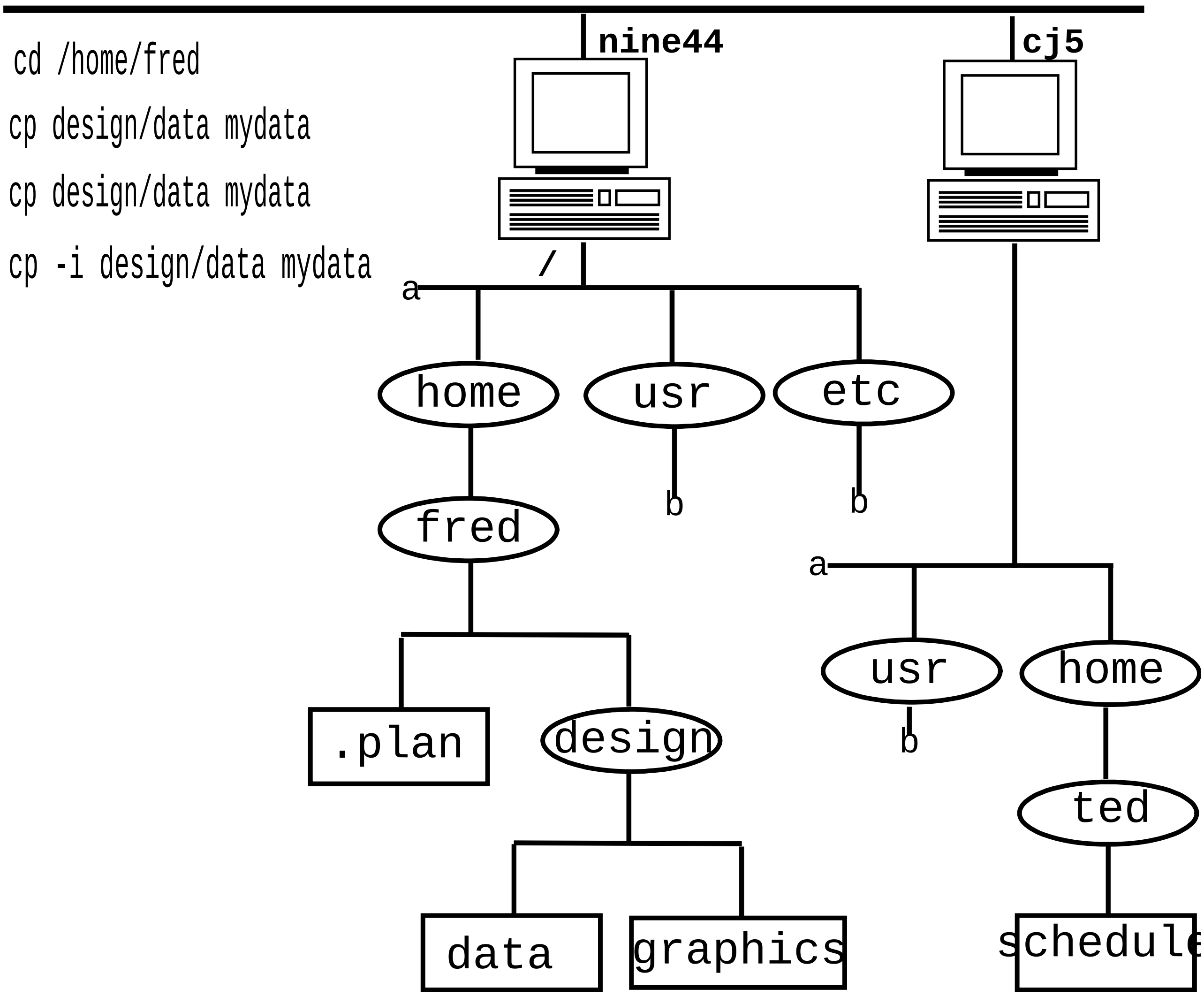
1) If the target name supplied does not exist, create it.

2) If the target name exists and it is a file:
   - a) overwrite, if the source is a file
   - b) error if the source is a directory

3) If the target name exists and it is a directory:
   - a) put the source in the target directory with its original name, overwriting if the name exists in that directory

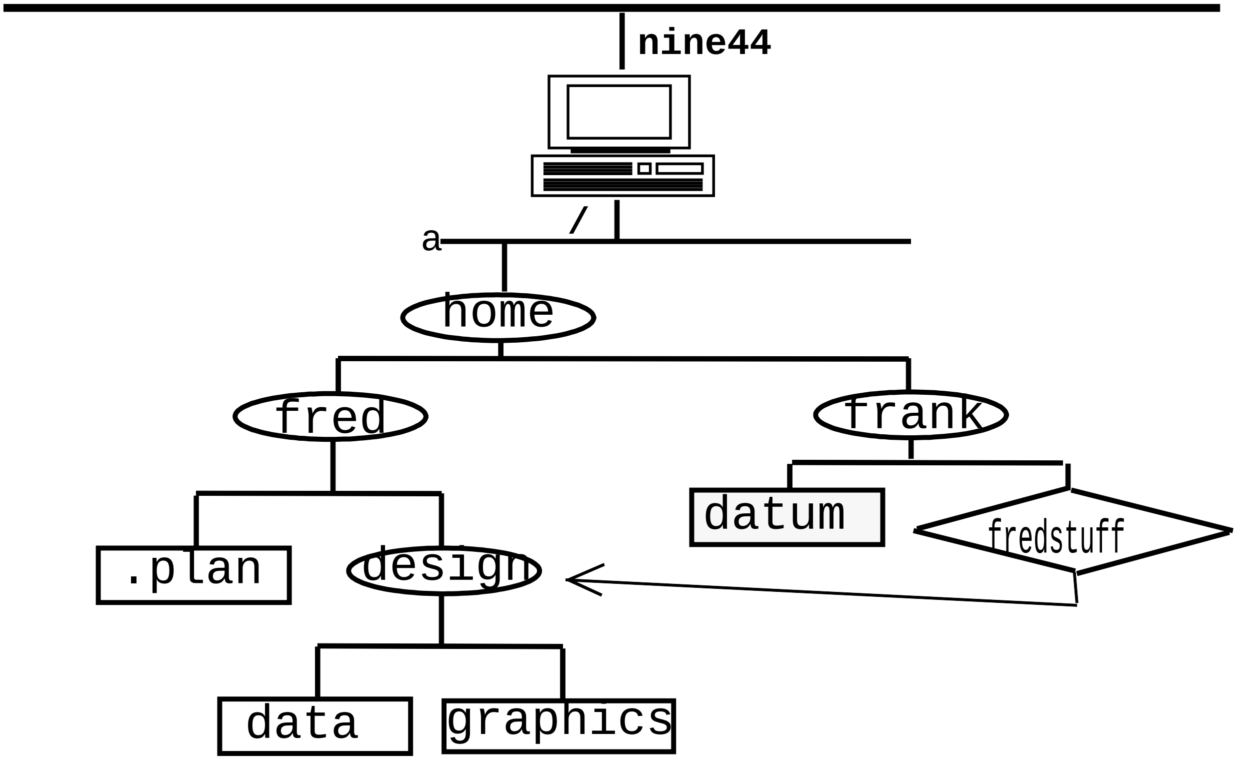
cp pathname examples

```
1) cd /home/fred
2) cp design/data mydata
3) cp design/data mydata
4) cp -i design/data mydata
```



```
5) cp  .plan  design

6) rcp design/data  cj5:/tmp/data
   cp design/data  ~ted


7) cp -r /home/fred/design  /home/ted/f-des
```

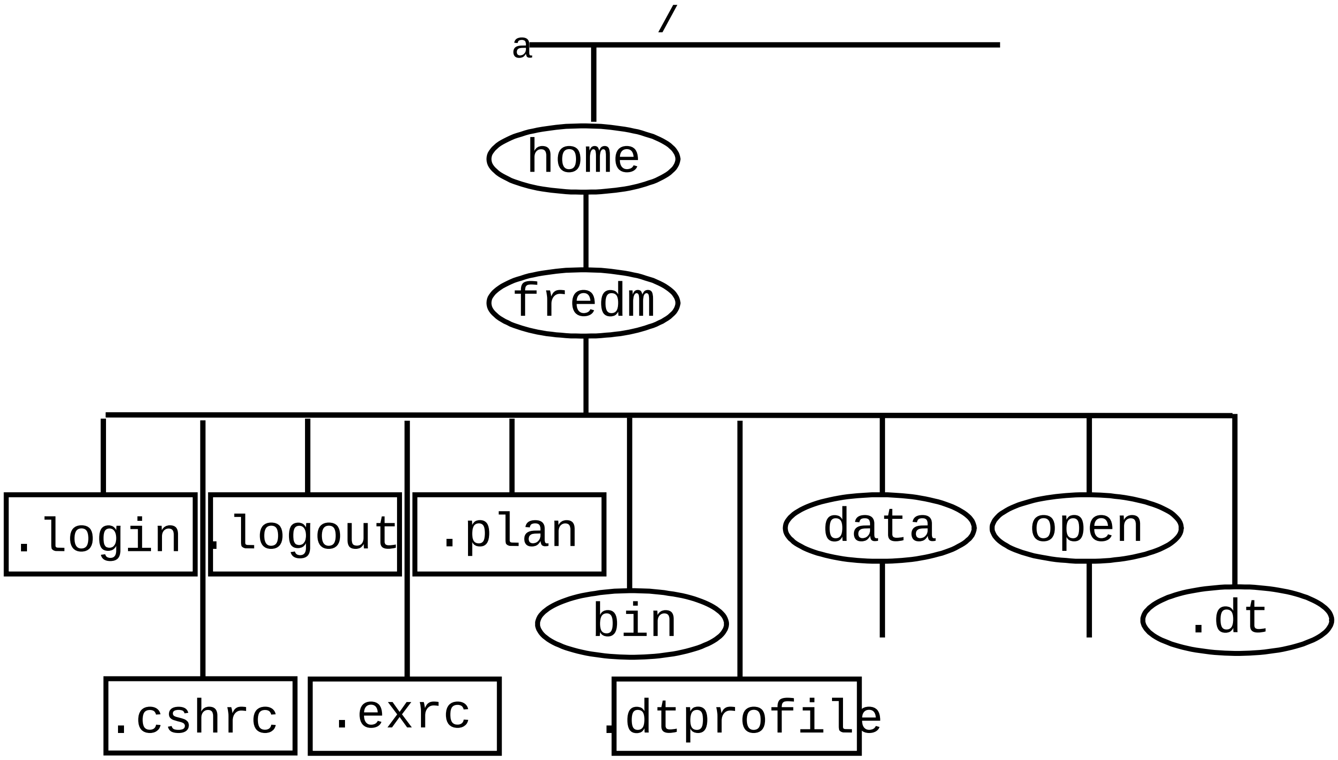
soft Links



Soft (or symbolic) links are objects that “point” to another pathname. They are handy for providing short names for long pathnames.

```
% ln -s /home/fred/design  /home/frank/fredstuff
% ls /home/frank/fredstuff
data  design
% cat /home/frank/fredstuff/data
the contents of the file data
%ls -l /home/frank
-rwxr-xr-- 1 fredm 2189 Sep 7:10:15 data
lrwxr-xr-x 1 fredm    4 Sep 6 08:20 fredstuff-
>/home/fred/design
```

Sample Home directory tree



Chapter 3

Shell Usage

Specific Objectives

After completion of this chapter the student will be able to:

List the types of shells available under most UNIX systems
List the features of each shell
Describe the function of I/O redirection
Describe a shell script
List the function of environment variables
Describe how filename expansion takes place
Edit .cshrc files
Use csh history and alias features

Chapter Contents

Common UNIX shells

Bourne shell *//bin/sh* or */usr/bin/sh*

Released in 1979 with V7 UNIX
Written by Stephen Bourne
Simplest of shells
Found on all UNIX systems
"the common denominator"
Commonly used for shell scripts

C shell */bin/csh* or */usr/bin/csh*

Released with 4BSD ~1980
Written by a group of Berkely and IIASA people
Found on Berkely systems, and
AT&T "with Berkely extensions"
Commonly used for it's interactive features:
history, job control, aliases, filename completion
Programming language resembles C ??

Korn shell */bin/ksh* or */usr/bin/ksh*

Written by David Korn
Most powerful of shells
Found on most UNIX systems
Upwardly compatible with the Bourne shell
Contains interactive features from the C shell
More efficient than other shells
Chosen as the POSIX shell

tcsh shell */bin/tcsh* or */usr/local/bin/tcsh*

The T comes from TOPS-20, whose features were emulated
All csh features, with many add-ons:
Command line editing, command completion, enhanced prompt, OS & language support, directory stack manipulation

What does a shell do?

It is not UNIX, just a UNIX command
Windows programs, and the shell are the most common interfaces to the operating system for non-programmers
Interactive parser of commands

```
forever {
            prompt(”enter what you want to do:”);
            read (command);
            check syntax (command);
            expand metacharacters (command);
            execute (command);
}
```

The shells distributed with UNIX provide a powerful programming language built-in, as the shell was designed by programmers, for programmers

Common Shell Features

All shells have the following basic features:

I/O redirection
Scripting
Filename expansion (wildcards)
Piping
Variables

The *csh*, *tcsh*, and *ksh* shells also have these features:

History
Aliases
Job control

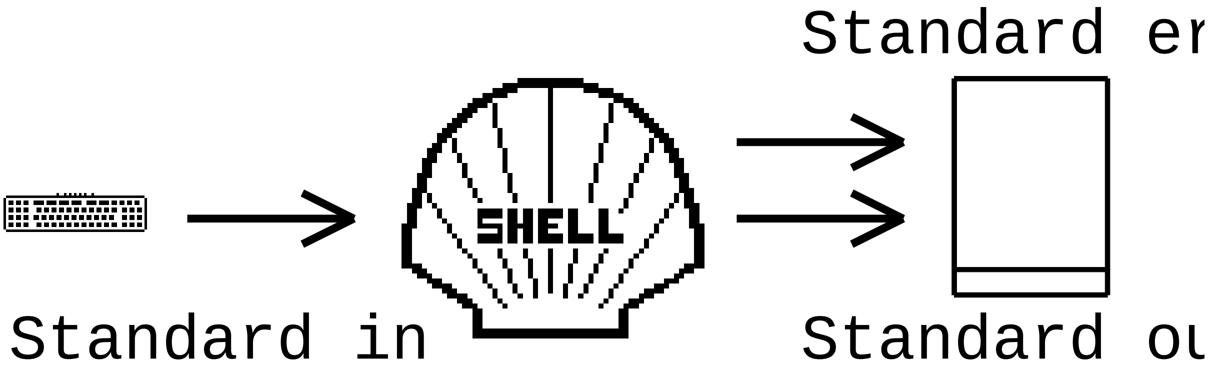
The *tcsh* and *ksh* shell also have:

Command line editing

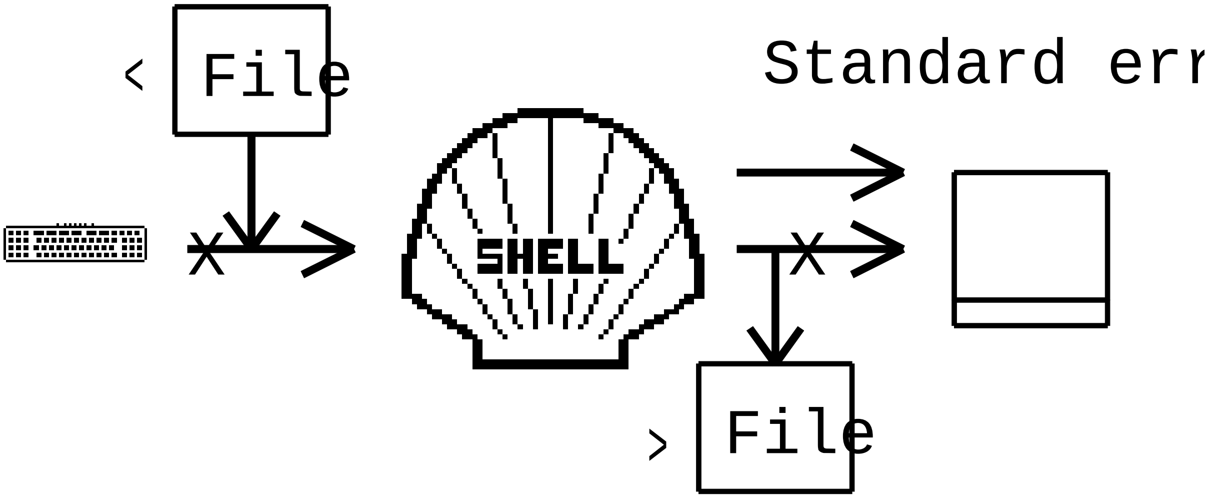
This class will concentrate on the most common shell: *csh*
(With some notes about the tcsh)

I/O Redirection

As mentiones before, all shells have methods of accepting and
sending data, these are called streams



These streams can be redirected to files, or other programs



This I/O redirection is provided by the shell not the command:

```
>    redirects output:      % program  > ~/data/test12
<    redirects input        % program  <  proj2
>> appends output           % program  >> ~/data/test12
```

Note that redirection of input is not used very commonly, since most UNIX commands accept filename arguments to read from.

CSH I/O redirection notes

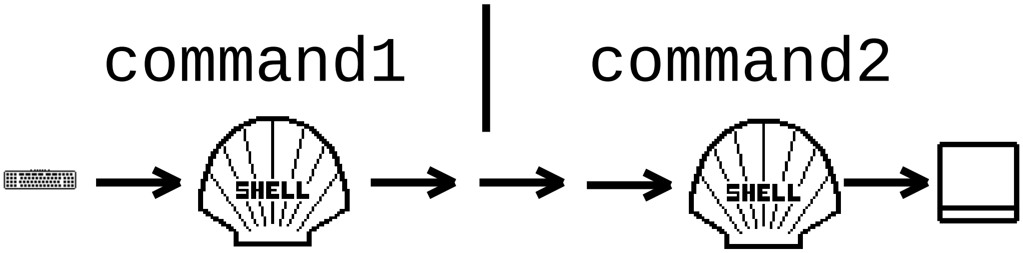
The following metacharacters redirect

```
< name
redirect input

> name           >> name
redirect output  append output

>! name          >>! name
override noclobber     override noclobber

>& name          >>& name
error and std output   error and std output

>&! name         >>&! name

|                |&
pipe             Send error out through a pipe
```

```
% ls
slideset.ps
% set noclobber
% date > slideset.ps
slideset.ps: File exists.
% date >! slideset.ps
% date >> nofilebythisname
nofilebythisname: No such file or directory
```

Piping

Piping is name name used when you redirect input from, or output to another command



Many commands can be piped together

catenate a file | pattern match | filter text | print

```
cat file1 file2 | grep  ERROR  | cut -c7-15 | lp -dqms3
```

Shell Scripting

Text files with execute rights that contain shell commands

Use standard streams

This makes them easily extensible

Can be sequential commands:

```
/home/ted/bin/gohome
```

```
echo "went home at:" >> ~/data/timel
date >> ~/data/timelog
echo "time has been logged, bye."
```

Can be programs using flow control:

sh	**ksh**	**csh & tcsh**
for	*for*	*foreach*
case	*case*	*switch*
if	*if*	*if*
while	*while*	*while*
function()	*function*	*goto*
	select	*onintr*
	until	*repeat*

Environment Variables

Process-wide variables that the shell, or some other program recognizes.
Often used by programs to pass data.
Created with the *setenv* command:

```
% setenv  MGC_HOME   /opt/mgc
```

Usually edited into login startup files, these variables are often needed by an application.

Filename expansion

Done by the shell, not shell commands

Creates the list of pathnames that match a pattern

```
      ls   designs/*/test[2-8]ver??
```

Symbol	Matches
?	any single character
*	any text string includi the null string Note /
[...]	any single character from set
[c1-c2]	any character lexically between c1 and c2
~	your home directory
~username	username's home directo (csh, ksh only)
{pat,pat,pat...}	Use each pat in order

Note that the *tcsh* also supports the following:

[^abc]   any character here except a or b or c

^pattern files that do not match pattern

Note that you must explicitly use . for hidden file wildcards.

Note that matches are returned in alphabetical order.

Note that the { , } construct returns patterns in the order listed.

Note that these two commands are rather different:

```
rm test*  # Remove all names starting with test
rm test *      # NOTE!!! Don’t ever do this!!
```

The *tcsh* has a variable you can set to prevent this from happening by accident:

```
  set rmstar
```

Key meanings in a shell

^ means <ctrl> in list below:

Key sequence	Function
<return>	Return
<enter>	Depends on keyboard
^M	Return
^D	EOF
^Z	Stop process (usually followed by an *fg* or *bg*)
^C	Interrupt Process
^\	Quit Process (core dump)
^U	Delete line of input
^W	Delete word (some systems)
<backspace>	Delete previous character
^S	Stop output
^Q	Resume output

Removing a Shell Process

Stop the shell process any of the listed ways (the terminal emulator will automatically exit when the shell program terminates):

Using the *kill* command

<CTRL>D (end of file)

Using the *exit* command

Using the *logout* command

To stop the job currently running, but not the shell itself:

Use the pre-defined key <CTRL> C

Or the *kill* command on the appropriate process id

CSH History

Saves previous commands AS entered for:

Re-invocation
Recalling portions of command lines(words)
Correcting typo errors

How many are stored is determined by *history* variable:

```
% set history = 20
```

default is 1

List is stored in the each shell separate from other shells

List is displayed by the *history* command

```
% history [-r #]
2 set history = 20
3 ld /usr/apollo/bin
4 ls /usr/apollo/bin
5 repeat 3 lcnode -c
```

To directly re-invoke a history command:

redo last command:	!!
relative numbered	!-1
absolute number	!5
command that starts with r	!r
command that contains cn	!?cn?

CSH History

Think of history as a text substitution command:

```
% cat employee.list | grep toledo
bill toledo, podunk iowa
tom jerry, toledo ohio
% !! > employee.toledo.list
```

You can also extract words from a command

```
Last command currently:
  cat employee.list | grep toledo >
employee.toledo.list
    0            1          2    3       4
5              6
% cat !!:6 is the same as typing “cat
employee.toledo.list”
% cat !!:$ same as above
```

Other symbols

words 1 to end	:*
words 4 to next to last	:4-
words 2,3,4,5	:2-5
words 1,2,3,4	:-4
Last word	:$
Command	:0

A quick method for fixing typos in one word

```
% cat employe.list | grep toledo >
employee.toledo.list
cat: cannot open employe.list: No such file or
directory
% nyenyee
cat employee.list | grep toledo >
employee.toledo.list
```

Quick method for swapping a word

```
%  !!:gs/toledo/eureka/
cat employee.list | grep eureka >
employee.eureka.list
%  !128:s/text/newtext/
```

You can also just reprint a command using the *:p* modifier

CSH Alias

Assigns an execution name to a command, or list of commands

To assign an alias

```
% alias h history
% alias mv mv -i
% mv x  y
y exists. Overwrite?  n

% alias cp cp -i
% alias  ll  ls -l
```

To display aliases

```
% alias
h     history
mv    mv -i
cp    cp -i
ll    ls -l
```

To remove an alias from the list

```
% unalias  alias_name
```

If an alias contains shell special characters, use quotes:

```
% alias   llk  ’ls | grep ’ #this will list names that look like..
% llk  test
alltest   testresults  test41   test5   xtestx

% alias  cd   ’cd \!*;ls’   #this cd’s to, then lists a directory
```

Escape mechanism

The following characters can be used to escape the meaning of metacharacters

\   escapes the next character

```
% cp   somefile  \
newname
%
```

’chars’   escapes characters in quotes except \ !

```
% echo  ’$money?’
$money?
% set money=100
%  echo ’$’$money
$100
%
```

″chars″   escapes characters in quotes except \ ‘ $!

```
$ echo  ″$money?″
100?
% echo ″help!!″
echo ″helpecho ″$money?″ ″
echo: No match.
%
```

Command Search Path

Per process

Note that the order of command lookup includes:
1) aliases
2) built-in commands
3) commands located via order in the search *path*

The *path* variable determines directories the shell will search to locate commands (the *cdpath* variable locates directories for *cd)*

If no "/" in first argument to the shell, it is a command

/bin/ls Is a pathname to the exact command to execute

ls Is a command name located by lookup order above

To change the path:

```
% set path = ( $path /some/new/dir/tosearch  . )
% set path = ( ~/bin /usr/bin /bin /usr/local/bin )
```

Note that a leading period would be a security problem.

To determine which command will be located and executed:

```
% which  command_name
```

The *cd* command uses the *cdpath* variable

```
% set cdpath = ( .  ~ /net/server/design_samples )
```

A leading period for CDPATH is correct.

Command Line Editing

The *tcsh* has command line editing capabilities.

If you want to use the *tcsh* only, you need to change your login shell to the full pathname of where the *tcsh* is installed on your system (if it is installed at all). It might be best to ask your system administrator to do this.

If you just want to test the *tcsh*, or use it occasionally, issue this command at the *csh* prompt:

```
exec  tcsh
```

To enable command line editing, you use the *bindkey* command (though it might already be enabled, test it first).

There are two methods that CLE can work in:

You use *vi* editing commands to recall and edit previous lines:

```
bindkey -v
```

You use *emacs* editing commands

```
bindkey -e
```

If you are not familiar with either editor, use the emacs method, since it allows you to perform MS-DOS doskey like editing:

Use the up or down arrow keys to display a command
Use the left/right arrow keys to move around on the line
Use the backspace/delete keys, and typing to change the line
Use the return key to execute the command

.login file

Used to set shell variables at login time on some systems.

You must own the file
You must have execute rights (*chmod u+x*)
Executed whenever a login shell is started (first /bin/csh on console (non window) type logins)

Used to:
set environment variables (on SunOS, and for telnets)
start window system (on SunOS).

Not usually used in window system logins, unless a window system startup file is configured to read it.

Sample:

```
echo .login running
cat /etc/motd
set path=( ~/bin /usr/ucb /usr/openwin/bin /bin\
           /usr/vue/bin /usr/bin  /usr/local/bin
. )
if ( ‘uname‘ != HP-UX) then
  setenv MGC_HOME /mgc  #all other setenv
comands
  echo “openwindows? (y/n): \c”
  if ( $< == “y” ) then
   /usr/openwin/bin/openwin
  endif
endif
echo .login complete
```

.cshrc file

This file is executed whenever you create a shell, on some systems */etc/cshrc* is run first. (*csh -f* suppresses execution)

Used to:

set shell conditions
set aliases
set local variables (and often environment variables)

Sample:

```
echo .cshrc running
set prompt='[\!]% '
set path=( ~/bin /usr/ucb /usr/openwin/bin /bin\
           /usr/vue/bin /usr/bin  /usr/local/bin
. )
set history=20
set noclobber
set filec
stty erase ^h
alias h history
alias mv mv -i
alias cp cp -i
set cdpath=( . ~ /design
/net/moby_dick/archives)
```

The format to set a local variable is:

```
      set   variable_name  =   value
or    set   variable_name=value
```

There is also a *~/.logout* file that is executed whenever you exit a “login” shell.

Predefined and Environment variables

The following variables may be set in your *.cshrc* file, see the *csh* man page for more details (man is covered in the next chapter).

path Set to list of directories to look for comands

cdpath Set to list of directories to look for directory names supplied to the cd command.

prompt Set to the sting you want as a prompt. A “\!” in the string means put the history number in.

echo When set (*set echo)* the command is printed after all parsing, before execution.

filec Means allow the <escape> key to finish a filename after typing enough letters to define it uniquely. (*tcsh* uses <tab> key) If the system beeps, you must type more characters, or press ^d for a list of matches.

history Set to the number of commands you want in the history list.

ignoreeof When set, do not exit on ^d.

mail Set to the list of files to watch for mail.

noclobber No not overwrite files on output redirect.

```
noglob              Do not allow filename
wildcards.
```

.tcshrc file

This file is executed whenever you create a *tcsh* shell, if it does not exist, the *.cshrc* file will be executed instead.

Used to do all the same things that *.cshrc* does

This makes it hard to run the *tcsh* unless you do one of these:

Maintain two startup file copies
Have your *.tcshrc* file also read the *.cshrc* file
Put everything in the *.cshrc* file, but protect the *tcsh* special
things from causing errors in the *csh*

Sample *.cshrc* file that also sets up for *tcsh*:

```
echo .cshrc running
set prompt='[\!]% '
set path=( ~/bin /usr/ucb /usr/openwin/bin /bin\
           /usr/vue/bin /usr/bin  /usr/local/bin
. )
if ( $?tcsh ) then
     bindkey -e  # enable command line editing
endif
```

Sample *.tcshrc* file that also sets up for *csh*:

```
echo .tcshrc running
bindkey -e  # enable command line editing
source  .cshrc
```

Chapter 4

Common UNIX Commands

Specific Objectives

After completion of this chapter the student will be able to:

Use online help (man pages)

Issue UNIX commands that manipulate text

Create text filter pipelines.

Chapter Contents

Commands

UNIX commands

Designed to do one thing well, they are also designed to be piped together.

This means that they are not big on verbose output, some commands have key words to give output.

UNIX commands are written to be correctly used, and typically do not warn you of impending destruction.

UNIX commands on different systems live in many places:

```
/bin        /usr/bin   /usr/ucb
/usr/new    /usr/X11/bin
/usr/lib/X11/bin
/usr/bin/X11     /usr/contrib
/usr/5bin
/usr/bin/posix              /usr/xpg4/bin
```

Often different from SysV to Bsd, and vendor to vendor

This is why the shell variable *path* must be set properly to locate commands. This is done in the *.cshrc*, *.login*, or window system startup file.

```
      set path = ( ~/bin  /usr/openwin/bin
/bin    \
            /usr/dt/bin /usr/bin  /usr/local/bin
.  )
```

The best way to add a directory to your search path is to include previous definitions, and just append, or prefix the new directory:

```
set path = ( $path  /new/directory/bin   )
set path = ( /new/directory/bin  $path )
```

The drawback to this method is that it is possible for the same location to appear multiple times (A bad thing).

UNIX On-line manuals

The *man* command has two formats:

*% **man** [section] title*   # To get a man page
or
*% **man** -k keyword*   #To get a list of man page names that
                              match *keyword* (Must be enabled)

Section contents	Section number
Commands and programs	1
System Calls	2
Libraries	3
Devices	7
File formats	4
Games	6
Miscellaneous	5
System Maintenance	1M

Most systems *man* commands will:

Search the sections for the title supplied in the order above.
Send the output to a screen pager, such as *more.*
Use a *manpath* variable to determine where to look for manpages.
Man leaves formatting in, such as bold text, this can cause problems when trying to do a search. The workaround is to use the following pipeline:
     man title | *col -b* | *more*

Examples:

```
man   ls
man -k list
man passwd
man 4 passwd
man 1M  mount
set manpath ( /usr/openwin/man  $manpath )
```

Sample man page

```
NAME
     command_name(1) - one line description of function
SYNOPSIS
     How to create a valid command line.
     Symbols used in man pages include:
      [ ]    Optional
      ...    Allows multiple of preceeding
      |      exclusive list (or)
      Beware of order of arguments, sometimes
      any order is allowed, sometimes not

DESCRIPTION
      Verbose description of the commands
      functionality. Lots of do’s and don’t listed here.

OPTIONS
      Each option is listed here, along with what it does

EXAMPLES
      Not all command man pages have examples

BUGS (called warnings on some systems)
     Must read information. Examples:
     The output device is assumed to be 80 columns wide.
     Gives un-predictable results for files over 6400
lines.
FILES
     Files used by or related to this command. Examples:
     /etc/passwd    To get user ID’s for ls -l.
     /etc/group     To get group ID’s for ls -g.
SEE ALSO
    List of other related commands. Example:
    diff(1), 3diff(1), comm(1)
```

Man page for cut command

cut(1)   cut(1)

NAME

cut - cut out (extract) selected fields of each line of a file

SYNOPSIS

```
cut -c list [file ...]
cut -b list [-n] [file ...]
cut -f list [-d char] [-s] [file ...]
```

DESCRIPTION cut cuts out (extracts) columns from a table or fields from each line in a file; in data base parlance, it implements the projection of a relation. Fields as specified by list can be fixed length (defined in terms of character or byte position in a line when using the -c or -b option), or the length can vary from line to line and be marked with a field delimiter character such as the tab character (when using the -f option). cut can be used as a filter; if no files are given, the standard input is used.

When processing single-byte character sets, the -c and -b options are equivalent and produce identical results. When processing multi-byte character sets, when the -b and -n options are used together, their combined behavior is very similar, but not identical to the -c option.

OPTIONS

Options are interpreted as follows:

list A comma-separated list of integer byte (-b option), character (-c option), or field (-f option) numbers, in increasing order, with optional - to indicate ranges.

For example:

1,4,7	Positions 1, 4, and 7.
1-3,8	Positions 1 through 3 and 8.
-5,10	Positions 1 through 5 and 10.
3-	Position 3 through last position.

-b list Cut based on a list of bytes. Each selected byte is output unless the -n option is also specified.

-c list Cut based on character positions specified by list (-c 1-72 extracts the first 72 characters of each line).

-f list Where list is a list of fields assumed to be separated in the file by a delimiter character (see -d); for example, -f 1,7 copies the first and seventh field only. Lines with no field delimiters will be passed through intact (useful for table subheadings), unless -s is specified.

-d char The character following -d is the field delimiter (-f option only). Default is tab. Space or other characters with special meaning to the shell must be quoted. Adjacent field delimiters delimit null fields.

-n Do not split characters. If the high end of a range within a list is not the last byte of a character, that character is not included in the output. However, if the low end of a range within a list is not the first byte of a character, the entire character is included in the output.”

-s Suppresses lines with no delimiter characters when using -f option.

Unless -s is specified, lines with no delimiters appear in the output without alteration.

Man page for cut command (cont)

Hints
Use grep to extract text from a file based on text pattern recognition (using regular expressions). Use paste to merge files line-by-line in columnar format. To rearrange columns in a table in a different sequence, use cut and paste. See grep(1) and paste(1) for more information.

EXAMPLES

Password file mapping of user ID to user names:

```
cut -d : -f 1,5 /etc/passwd
```

Set environment variable name to current login name:

```
name=‘who am i | cut -f 1 -d ”  ”‘
```

Convert file source containing lines of arbitrary length into two files where file1 contains the first 500 bytes (unless the 500th byte is within a multi-byte character), and file2 contains the remainder of each line:

```
cut -b 1-500 -n source > file1
cut -b 500- -n source > file2
```

DIAGNOSTICS

line too long — Line length must not exceed LINE_MAX characters or fields, including the new-line character (see limits(5).

bad list for b/c/f option — Missing -b, -c, or -f option or incorrectly specified list.
No error occurs if a line has fewer fields than the list calls for.

no fields — List is empty.

WARNINGS

cut does not expand tabs. Pipe text through expand(1) if tab expansion is required.

Backspace characters are treated the same as any other character.
To eliminate backspace characters before processing by cut, use the fold or col command (see fold(1) and col(1)).

Other UNIX manual commands

whereis *[-bms] [cmd or file]*

Options:
b binaries m manual section s sources

Used to locate a command, if it exists.

which *[command]*

Gives the full path to the file that would be executed as found in the command search "path". Looks for aliases in the users .cshrc file. Used to determine if a command is being executed directly, or if it has been aliased to some set of options.

Windows man page readers

Many window systems supply a man page reader that is often easier to use than the UNIX man command

Usually accessible from a menu (Openwindows and DecWindows) or a front panel button (HPVUE, and CDE). Sometimes you must issue a command to start the reader, such as the generic X windows command: *xman &*

Can scroll back through text, often a search function is available.

Basic UNIX Protection Model

Uses three rights

For three classes of users

user group others

Displayed with the ls -l[g] command

```
-rwxrwxrwx  1 ted  design   1024 Dec 13 11:21 test_file
```

Object Type

-	Plain file
l	Symbolic link
d	Directory
c	Character special file
b	Block special file
p	Named pipe (fifo) file

Rights

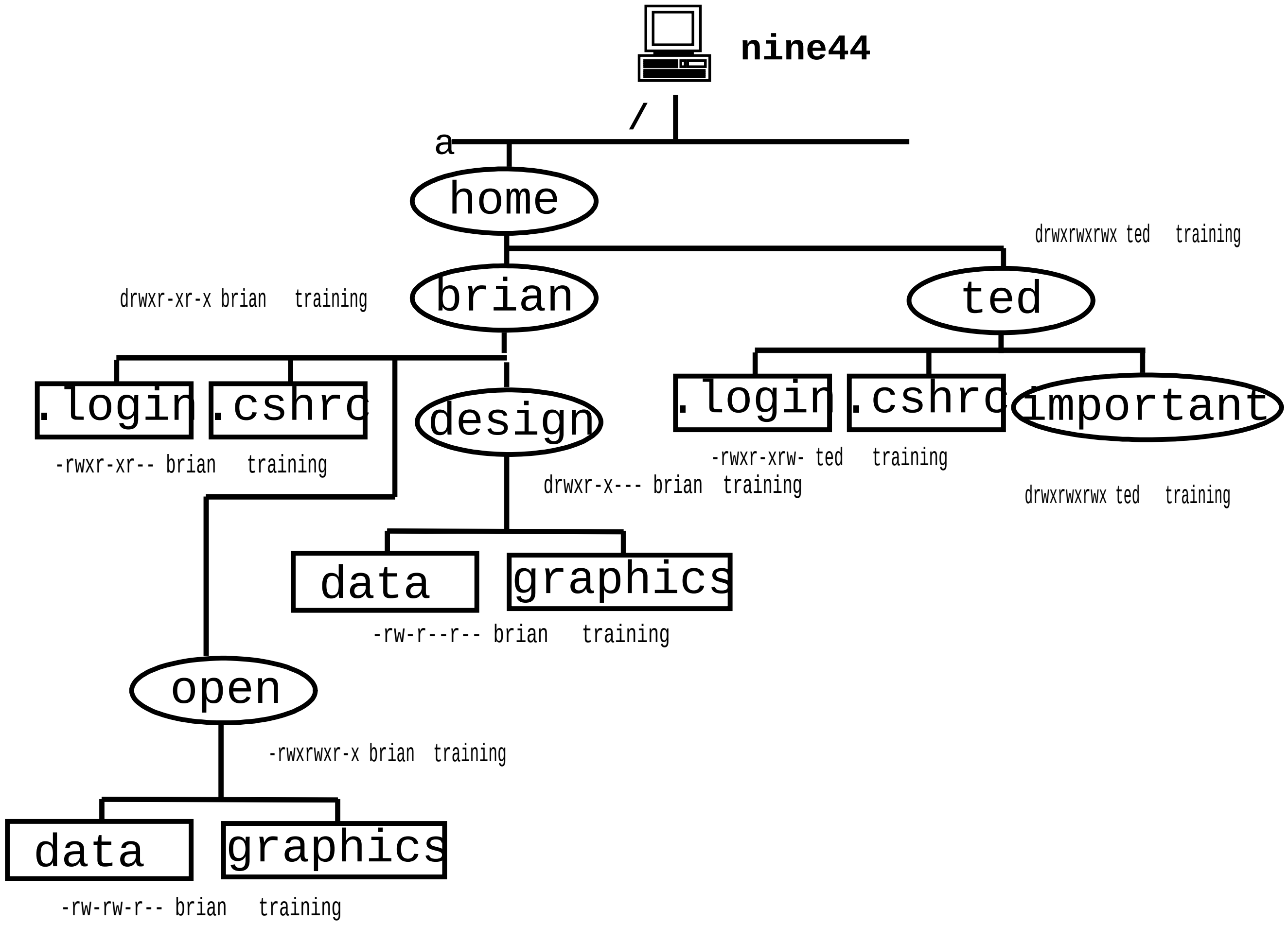
For files

r	permission to read file
w	permission to write file
x	permission to execute file

For directories

r	read the file names in directory
w	create and delete files in directory
x	can make the directory your working directory, search through it ,and copy files from it.
-	Permission at this location not given

Home Directory Permissions example



Will these commands work?

```
% login brian
% cd /home/ted
% vi  .login
% rm .login
% cp ~/myjunk  important
% cat important/*
% rm -r important
```

Will these commands work?

```
% login ted
% cd /home/brian
% vi  .login
% rm .login
% cp ~/myjunk  data
% cat design/data
% cd design
% ls design
```

Defining Rights

chmod [-fR] *mode* *filename ...*

-f force -R Recursively

Absolute method

Rights can be expressed in octal

	User			Group			Others		
	r	w	x	r	w	x	r	w	x
	4	2	1	4	2	1	4	2	1

chmod 764 test_file

Symbolic method

u user (owner) - remove right
g group + add right
o others = absolute
permission
a all

Examples:

```
chmod      u=rwx,g=rx,o=r  test_file
                  7     5     4
chmod      o-w          test_file
chmod      g+rx            test_file
chmod      g+x,o+rw     test_file
```

Note: Assigning a right of +X means give everyone
execute rights if anyone has x rights

Protection command list

Protection control commands:

Standard UNIX protection commands:

/bin/chmod	Change mode of existing object
<umask>	Change umask value which removes listed rights from objects being created
/etc/chown	Change the owner id (sys5 run only by root bsd run by owner or root posix run by owner or root)
/bin/chgrp	Changes group owner id

```
% umask 000
% mkdir q ; touch m
% ls -ld q m
-rw-rw-rw- 1  fredm  training     0 Jun  9 02:48 m
drwxrwxrwx 2  fredm  training    24 Jun  9 02:48 q
% rmdir q ; rm m ; mkdir q ; touch m

% umask 027
% ls -ld q m
-rw-r-----  1 fredm  training     0 Jun  9 02:49 m
drwxr-x---  2 fredm  training    24 Jun  9 02:49 q
```

Text file commands

cat   Concatenate file to standard out

```
cat [-u]    [-|file ...]
     -u   Unbuffered output
     -    read from stdin, can be used in file
list
cat file1        # prints file to screen
cat f1 - f2 > f3 # puts f1, stdin, then f2 into
file f3
cat > file2      #  same as:     cat - > file2
                 # creates file2, contents are
typed
                 # in until a <ctrl>d
cat f1 f2|grep x # Sends f1 and f2 to grep
```

more   Browse files page at a time based on window size

```
more [-ceisu][-n number][-p cmd][-t tagstring]
[file]

When in more:
     <cr>       displays one more line
     q          to quit
     <space>    displays the next screen of
text
     /text      Searches for next occurance of
text
     h or ?          for help while in more
     <c>b       back up a screen (in a file,
does not
                always work in piped data)
     =          display line number
```

In some versions of more there are many other commands for editing, searching, jumping lines forward and back, etc...

Note that the commands available change if you are reading a stream instead of file.

Many systems ship a program called *less* that does more than *more.* Sometimes it is shipped in place of *more*, and named *more.*

Text file commands

head   Copy the first part of a file to stdout

head [-n *count*] [file]
prints to screen first ”count” lines of a file
default is 15 or 10 (system vendor dependent)

tail Copy the last part of a file to stdout

tail [-f][-c *number*|-n *number*] [file]

```
     -f   Do not terminate when end of file is reached
          (wait for future writes, then show them)
     -c   count in bytes
     -n   count in lines
     +c or +n starts count from beginning of file
```

Note that the older version of these commands was to use the syntax shown below. The newer version will require a -*n* or +*n* before the number of lines.

```
% ls -l | head -1
total 144854
% ls -l | tail +2 | head -3
-rw-r--r-- 1 fredm training   132 Jun  5 23:56
#noName#
-rw-r----- 1 fredm training 78983 May 28 06:45
TCSHMAN
drwxr-xr-x 3 fredm training  1024 May 28 08:42
archive

% ps -u fredm | grep xterm | sort -r -n |
head -1
```

Text file commands

grep   Search a file for a pattern

```
grep [-E|-F] [-c|-l|-q][-insvx]
     [pattern|-e pattern...|-f pattern_in_file]
[file...]
     -E   Use extended regular expressions
     -F   Use no regular expressions
     -c   Only report count of lines containing
pattern
     -l   Only report names of files containing
pattern
     -q   Quiet
     -h   Suppress filename printing when
multiple files
           are being searched
     -i   Ignore case
     -n   Precede lines with line number
     -s   Suppress some types of errors
     -v   Select lines NOT containing pattern
     -e   Used to specify multiple patterns
     -f   Used to have grep look in file for
patterns
```

```
% grep ted /etc/passwd
ted:duuk89g0u3zr:25:30:TedBombo:/net/cj5/user/te
d:/bin/sh

% head -100 big_file | grep ”customer
complaint”
% grep -in  -e  “^Error”   recise1
% grep -vc  “NOTE:”  datafile

% cat f1 f2 f3 f4 | grep -i -e error -e
warning
Error: 123
Warning: line 12 truncated
Warning: line 11 truncated
% grep -e error -e warning  f1 f2 f3 f4
f2:Error: 123
```

```
f2:Warning: line 12 truncated
f4:Warning: line 11 truncated
```

Meta-characters In Basic Regular Expressions

.   (This is a dot/period)

Matches any single character (except ″newline″)

[...]

Matches any ONE of the characters represented in the brackets (a character class). Most meta-characters lose their special meaning inside the brackets.

*

Matches zero or more occurrences of the character(s) in the input stream that is(are) represented by the regular expression that PRECEDES it. Note that this is a different from the meaning of an ″*″ when interpreted by a shell, where it directly represents ″one or more characters″.

^

As the first character of a regular expression, matches the beginning of the line

$

As the last character of a regular expression, matches the end of the line.

\

Removes any special meaning of the character that follows it.

“^[0-9][0-9]$″* A line that has only digits on it.

*“^[ ]*Test”*   A line that begins with Test, but might have
some spaces before it.

“END$”   A line that ends with END.

“^END”   A line that begins with END.

″cell 2[1-4]″   A line that has cell 21 thru cell 24 on it.

Other Text file commands

cut - Cuts selected fields of each line in a file

patch - Merge corresponding lines of multiple files

pr - Used to pretty up text (prepare files), many switches to determine what is pretty (can merge files horizontally)

```
      Example:
      cat filename | pr -t -3  (3 columns, no header)
      pr -m  f1  f2     (Prints files side by side)
```

sort   Sorts a file in alphanumeric order, or numeric order (-n)

      Extremely powerful, use the man page in lab for details

sed   Stream editor, performs line by line edits

```
      cat filename | sed ″s/:/:   /″
```

split - Splits a large file into many smaller ones

wc - Counts words, lines and characters in a file

fold - filter for folding lines to specified width

compress, zcat, uncompress - compress files and entire directory trees, end output name with .Z

```
  % compress big_file
  % ls
  big_file.Z
  % zcat big_file
  ..
```

diff - Shows the lines that are different in two files

3diff - Shows the lines that are different in 3 files

bigdiff - Used for very large files

cmp   - Fails if the two files are different

comm - Shows the lines that two files have in common

dircmp - Compares tree structures

Find command

NAME

find - find files

SYNOPSIS

find pathname-list   expression options

Another way to look at the synopsis might be:
find where-to-look   what-for   what-to-do

find recursively descends the pathname-list directory hierarchies for each file that meets the expressions rules

EXAMPLES

```
find      .  -name lost.file  -print

find      .  -name “ *test* ”  -ls

find .  -name  ” *.bak ”  -exec cp {} ~/save \;

find     /archive  -name ” *.bak ”  -ok rm {} \;

find /net/nine44/home/fred -name *.bak -ok cat {} \;

find . /database -user fredm -type f -atime +180 -ls

find ~  -name  ” sd*.cor”  -print
```

Process commands

ps

ps [-options]
Some useful options (see man pages for more):

ps -u <username>   *ps -ef*

```
% ps -u dshaw
   PID TTY      TIME COMMAND
   253 ?          0:05 vuewm
   229 ?          0:00 vuesession
   259 ?          0:00 hpterm
   273 ttyp2    0:00 ksh
   257 ttyp1    0:00 softmsgsrv
   261 ?         0:00 hpterm
   272 ?         0:07 maker4X.exe
   274 ttyp3    0:00 ksh
   315 ?         0:02 iview-xm
   277 ?         0:00 fm_misd
   301 ttyp2   1:16 ileaf6
   281 ?         0:00 fm_flb
   325 ttyp2    0:00 ps
% ps -f
     UID   PID  PPID  C    STIME TTY      TIME
COMMAND
   fredm  1541  1538  0    Jul 25    ttyp2
0:00 ksh
   fredm  2518  1541  6   12:01:13 ttyp2    0:00
ps -f
```

Process commands

kill - send a signal to a process

kill -s signo Newer syntax

kill [-signo] PID *-signo* refers to the type of signal

kill -l To list signal names

```
 1) HUP               12) SYS           23) CHLD
 2) INT                   13) PIPE      24) TTIN
 3) QUIT              14) ALRM   25) TTOU
 4) ILL               15) TERM   26) TINT
 5) TRAP              16) USR1          27) XCPU
 6) IOT               17) USR2   28) XFSZ
 7) EMT               18) CLD    29) VTALRM
 8) FPE               19) bad trap      30) PROF
 9) KILL              20) STOP   31) URG
10) BUS               21) TSTP
11) SEGV              22) CONT
```

kill -3 1541 *kill -s 3 1541*

If the signal number is omitted, most versions send a -15 :

kill 301

To stop a process, use the default signal in most cases.

renice - Set scheduling priority of a running process

nice - Run a process at a low priority

Printing

You can print from applications, or with shell commands. If using shell commands, which of the following is operational depends upon your network configuration.

lp (Sysv)

```
% lp   data_file
% lp   -dapple   /home/ted/data_file1 data_file
```

Associated commands:

```
/usr/bin/lp      Submit jobs

/usr/bin/cancel  Cancel jobs

/usr/bin/lpstat       Show status of printer
and jobs
```

```
% lpstat
ljlocal-3313 fredm priority 0 Jun 7 13:17
on ljlocal
        ch5_vi.pcl  797884 bytes
ljlocal-3314 fredm priority 0 Jun 7 13:18
        ch5_vi.pcl  797884 bytes
% cancel ljlocal-3314
request ”ljlocal-3314” cancelled
% lpstat -d
QMS27
%
```

lpr (Bsd)

```
% lpr   data_file
% lpr  -Papple /home/ted/data_file1 data_file
```

Associated commands:

```
     lprm       Remove a request
     lpq        List jobs
```

Tar

Tape file ARchiver

Tar is currently used just as often to bundle many files into a single disk file, as it is to write files to a tape.

Tar performs no compression, it simply ’gathers’ listed files or trees into a single image.

A similar command, named *pax*, is the POSIX standard for an archive command, but *tar* is more commonly used. Most modern *tar* commands write the same format of archive file as *pax*.

The command line syntax for *tar is:*

```
tar key [arg...] [file | -C directory] ...
```

Tar man pages use the word *key* where most commands use the word options. You can put a hyphen before the *key*s on most versions:

```
tar cf class.tar  labs  ch*.doc  notes
tar -tf class.tar
```

Almost all invocations of the *tar* command require the *f* key, which tells *tar* where to write the bundle , or where the bundle exists that we want to read. The *f* key requires an argument to follow the key list. In the example above, we were writing and reading from the file named *class.tar* in the current directory.

Other uses for the *f* key is to write or read a tape in a drive. In that case, the argument would be a pathname to a device file for that drive. There are often several device files for the same drive, specifying different options that the drive is capable of.

```
Sample Names for device files:
/dev/rmt/0m  Usual name for first tape drive
             with rewind after write, or read
/dev/rmt/0mn No rewind, same device as above
```

Tar

Writing archive keys:

c Create new archive at beginning of file
u Update files to archive, only if not there, or updates since version in archive
r Add to end of existing archive

Reading archive keys:

t List names of files in archive file
x Extract the named file/files from the archive file. If a directory, it restores the tree.
p Attempt to preserve original ownership and protections
m Use extraction time as modification time
C Used in file list to change directories

General keys:

f Archive file pathname argument required
v Be verbose about the listing
V Be more verbose (list object type)
h Follow symbolic links when encountered
w Display action for each file and prompt for confirmation

There are other keys, these are the most common.

The file list to be archived should use relative pathnames, since you cannot change the path when extracting (you can change the restored path with *pax*, which is one of its best features).

Tar

```
% tar -cf vi.tar /disc/users/fred/class-
stuff/VI-class
% tar tf vi.tar
/disc/users/fred/class-stuff/VI-class/
/disc/users/fred/class-stuff/VI-
class/suess.txt
/disc/users/fred/class-stuff/VI-class/lab2
/disc/users/fred/class-stuff/VI-class/lab1
/disc/users/fred/class-stuff/VI-
class/sd7080.cor
/disc/users/fred/class-stuff/VI-class/lab3
/disc/users/fred/class-stuff/VI-class/lab4
% rm vi.tar
% cd ~/class-stuff
% tar -cf vi.tar VI-class
% tar -tf vi.tar
VI-class/
VI-class/suess.txt
VI-class/lab2
VI-class/lab1
VI-class/sd7080.cor
VI-class/lab3
VI-class/lab4
% tar tvf vi.tar
rwxr-xr-x  25/30      0 Mar 27 04:56 1997 VI-
class/
rw-r--r--  25/30   1373 Mar 31 08:30 1996 VI-
class/suess.txt
rw-r--r--  25/30   6482 Mar 31 14:26 1996 VI-
class/lab2
rw-r--r--  25/30   5287 Mar 31 14:26 1996 VI-
class/lab1
rw-r--r--  25/30    860 Mar 31 09:13 1996 VI-
class/sd7080.cor
rw-r--r--  25/30   3878 Mar 31 19:54 1996 VI-
class/lab3
```

```
rw-r--r--  25/30   4121 Apr  1 08:30 1996 VI-
class/lab4
```

Tar

```
% tar cf tw.tar -C ~/class-stuff VI-class -C  ~
UNIX_use
% tar tf tw.tar
VI-class/
VI-class/suess.txt
VI-class/lab2
VI-class/lab1
VI-class/sd7080.cor
VI-class/lab3
VI-class/lab4
UNIX_use/
UNIX_use/x y
UNIX_use/text/
UNIX_use/text/countrycodes
UNIX_use/text/weights+meas
UNIX_use/text/suess.txt
UNIX_use/text/att-access
UNIX_use/vilab
UNIX_use/foils/
UNIX_use/foils/clark-y.cor
UNIX_use/foils/sd7084.cor
UNIX_use/foils/sd7080.cor
UNIX_use/foils/sd7090.cor
UNIX_use/foils/fx63137.cor
UNIX_use/foils/7060-8-6.dxf
UNIX_use/.cshrc
UNIX_use/.login
```

The following two lines does what the first line above did:

```
% cd ~/class-stuff ; tar cf  ../tw.tar  VI-
class
% cd ~ ;   tar rf tw.tar  UNIX_use
```

Here is an example of writing everything from the current directory to a tape:

```
% tar cf /dev/rmt/0m  *
```