CENG-111 Introduction to Computer Engineering Concepts

Laboratory Manual

Prepared by: İlker Altıntaş, Hasan Ulusoy & Selçuk Şenkul

15 November 1996

Contents

1 Introduction to UNIX

1.1 What is UNIX

UNIX is a *multiuser* , *multiprocessing* operating system. Unlike some operating systems like DOS, more than one user can use the system at the same time. Also each user may have more than one process running at the same time. A process is a program that is currently running.

UNIX was written in AT&T Bell Laboratories at the end of 1960's. First version of it was written by Ken Thompson and did single processing. Later, Ken Thompson and Dennis Ritchie wrote it again using the C programming language. Except the core kernel, assembly language was not used.

Since UNIX was a trademark of AT&T, many major computer companies produced their own version of UNIX. Although the codes are different, most of the UNIX-clones are similar to each other. Also most known clones support the POSIX standard. This standard defines the ways a programmer or user should communicate with the system. Most popular free UNIX-clone is called LINUX. LINUX is being developed by thousands of people all over the world and is the fastest developing system. If you have a PC and a hard disk at home, you are advised to setup LINUX to your system.

Usually a UNIX system has a single CPU. So with one CPU, how can it possibly run more than one process for each user at the same time? Actually, the processes are not executed in CPU at the same time. The system gives every process a portion of time, in which the process is executed in the CPU. Each process is executed for small periods of time, but since the CPU is very fast users feel that their process is running continuously. Sharing of CPU and other resources among processes is called *timesharing*. The users can feel this sharing only if the system is overloaded (too much process running at the same time).

When a process has its turn for the CPU, it becomes a running process. While one process is running, other processes stay in memory (RAM). Operating system tries to share all the resources (hard disk, network, memory, etc.) among the processes fairly. UNIX can be divided into two parts. The low level part which deals with the hardware (disk transfer, network I/O, memory sharing) is called the *kernel*. Users communicate with the kernel using some interpreters called *shells*. Most known shell types are *sh* (bourne shell), *ksh* (korn shell), *bash* (bourne again shell), *csh* (C shell), *tcsh* (turbo C shell).

1.2 How to use the system

Login process: UNIX systems run a process called *login*, to let the users get into the system. You should be aware that you may not be the only user working in that system. In order to be fair, do not misuse the system resources. Also when you leave the system (logout), remember that other people may continue to use that system. If you were sitting on the console (physical machine the system is working on, not a PC), do not shut down the system. UNIX machines are designed to be open 24 hours a day and 7 days a week. If you thing that your console is not working, call your system administrator.

When you decide to go, do not! close the screen and go away. You have to see the login screen to make sure that you have logged out. If you do not log out from the system, the next person sitting on the same machine can make everything to and from your account (you are responsible for the damage someone does from your account.

1.2.1 Using the workstations (b-203)

Logging in

1. Check whether the screen is closed or open. If there is light on the button (lower right corner of the screen), move the mouse a little and goto step 3.

2. If the screen is closed (no light in button), press the button (a light should appear on the button) and move the mouse a little.

3. The login screen appears. Write your user-name, press enter key and make sure that cursor is in front of password line, write your password and press enter. (your password should not be seen when you type it)

4. If a message like "Login incorrect" appears on the screen (probably you have written either user-name or password wrong), goto step 3. If successful, you will get a window (xterm) to the workstation you are working at.

Logging out

1. To log out from the system, the default way is to use the mouse menu. Press the leftmost button of the mouse in an empty place (not inside a window) and hold it. Move the mouse to the "logout" line (bottom line) of the appearing menu, and release it there.
2. If this was successful, you will see a screen with Solaris logo, then the login screen again. If nothing happens, try first step again.
3. When you log out, press the screen button to close the screen.

1.2.2 PC Laboratory (B-Z01)

Logging in

1. Find an empty PC. If it is open and s: drive is present (the prompt begins with S:), goto step 3. Else put an nfs diskette to the driver, and open the PC. (If you don't have an nfs diskette, ask for it from other users. Once you are in drive S, you can make your own diskette. But an empty diskette to drive a, write "cd \makenfs" and run "hdnfs50")
2. Wait patiently while the system boots up. There will be several messages on your screen. Whenever you are asked to press any key, press the space key (longest key at bottom of the keyboard). Finally you should end up in a prompt like S:\user\1labXX>. If you can't see the prompt, ask the people around you for help.
3. Write "telnet saygun" and press enter. You should get a screen asking you the login (similar to the one below).

```
Trying 144.122.171.226 ...
Connected to saygun.
Escape character is '^]'.

SunOS UNIX (saygun)

login:
```

Once you enter you user-id and press enter, it will ask you for password (your password should not appear on the screen). If both inputs are correct, you will login to saygun.

```
login: selcuk
Password:
Last login: Fri Oct 11 11:00:40 from tere.lcsl.metu.edu.tr
SunOS Release 4.1.4 (GENERIC+DB) \#1: Sat Jul 29 14:16:03 EET DST
You have mail
<saygun:~>
```

Logging out

1. write "exit" while you are in saygun. When you hear a beep sound, press alt-x keys together.

2. When you see the prompt (S:\user\1labXX>), either leave the PC open (if laboratory is very crowded) or close it.

1.2.3 İsmail Abi Laboratory (B-101)

This laboratory has some number of PC's which are used only by deparment students. Also the maintenance of these machines are done by the students. So there is no specific way of using them. You have to ask to students working there for more information. Remember that any machine in that laboratory can be used by you. So, do not hessitate to ask for an account from the linux machines in that laboratory.

1.3 Files and Directories

A directory may contain several files and directories. A directory can be thought as a book shelf. Directory (shelf) is used to group files (books) and other directories (smaller shelfs) in it. A file (book) may contain only data in it. You can not put another file or directory inside a file.

The special character '/' represents the root directory in every UNIX system. This root directory holds every other directory and file in it. The directories inside a directory are called *subdirectories*. The directory structure grows like a tree which is upside down.

Root directory is at the highest level and each subdirectory is one level below its parent.

When you login, the system lets you start in a predefined directory called *home directory*. This home directory can not be changed by users. The '~' character represents the home directory.

The links from one directory to some other is called the *path* of that directory. In the figure above, the path of e101111 directory is "/home/bs1/e101111" (each directory should be separated

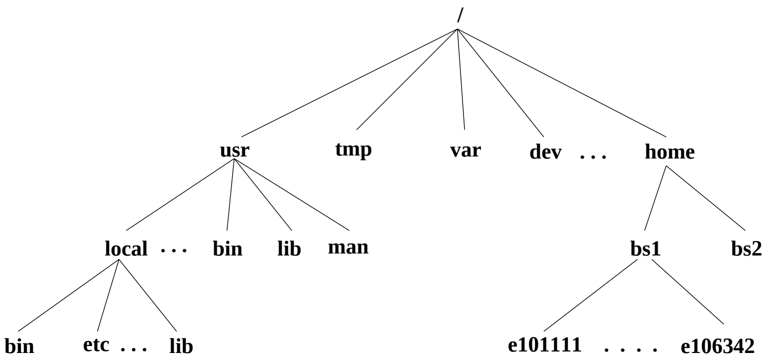


Figure 1: Directory Structure

by '/' character). Path can be represented in three ways. You can start from the root directory (absolute path), from the current directory (relative path) or from the home directory (relative path to home). Users should choose the most suitable one for the current situation. For example if current directory is /home/bs1, a user may write the path to e101111 by "cd /home/bs1/e101111" or "cd e101111" or simply "cd ". The relative path is useful for going to a sub directory of the current directory while absolute path is useful when you know the exact path of the directory, which is not below current directory.

Each directory name can be up to 256 characters provided that it does not include '/' character. It is useful to give meaningful names to directories. The standard directories in each UNIX system are the same. 'bin' directory contains the executable programs of the system. 'var' contains system related directories. 'etc' contains system related configuration files. 'lib' contains library files.

File names have the same properties with directory names. Although the uses are different, directory is also a special type of file in UNIX. Unlike directories, files may not contain subfiles or subdirectories. A path to a file is the same of a path to a directory. Although extensions with a dot means nothing to the system, some applications use them. Frequently used extensions in a UNIX system are

.txt	text file
.c	C source file
.cc	C++ source file
.h	header file
.z .gz	compressed file
.gif .jpg	image files

The amount of data in a file is called its size. Usually size is expressed in terms of bytes. Directories also have size, but their size depends on the number of files and subdirectories inside them.

1.4 Permissions

Since there are more than one user in a UNIX system, a method is provided for security. Every file and directory has a permission byte. The figure below shows which bits correspond to which permissions in a typical "ls -l" output.

```
 owner     group     other         r:read permission
r  w  x   r  w  x   r  w  x        w:write permission
                                   x:execute permission
```

First triple shows the permissions of the owner. Only owner or root can change a file's or directory's permissions. Second triple indicates the permissions of the users that are members of the same group with owner. Every user has at least one group in UNIX. So that some limited permissions can be given to members of that group, but not other users. Last triple indicates the rights of all other users.

For each triple, three bits indicate the corresponding permissions. r stands for read permission. If there is a read permission, if it is a file, it can be read, if it is a directory, it can be listed. w stands for write permission. If there is write permission for a file, it can be modified. If there is a write permission for a directory, then new files or directories can be created and old files or directories can be deleted. 'x' stands for the execute permission. If there is execute permission for a file, it can be executed if it is executable. If there is execute permission for a directory then it can be entered.

If you do not want anyone to write to your files, close the write permission of other and group. If everyone is going to be allowed to read and enter your home, then read and execute permissions should be open. An easy was to tell the system that you want every new file in some certain mode (permission) is to set a variable named "umask". For our case it has to be "umask -S 022". The three octal numbers are the subtraction of wanted mode from 777. Calculating the wanted modes can be found in section 1.6.1 on page 7.

Besides protecting the files and directories, UNIX system also protect every processes memory area from others. If this was not done (like DOS), one process could damage another by writing to the memory location directly.

1.5 Reading manual pages

Whenever you forget the syntax of a command, look around the manual pages of that command (if you forgot how **"ls"** works, write **"man ls"**).
Manual pages are grouped into sections. Each section contains the manual pages of the commands related to that section. Some examples of sections are:

- Section 3 - contains c function call manuals.
- Section 2 - contains system call manuals.
- Section 1 - contains user commands manuals.

If no section is specified with man command, it searches from smallest to biggest section until it finds a match. There may be two manual pages for the same name in different sections (time system call and time user command). If you can't get the correct page, try it with the section number.
Notations that are frequently used in this manual and man pages are listed below.

- The words written **bold** (dark color) are to be written exactly. They are the actual commands.
- [ ] Means that the things between brackets are optional. The command will work even if you do not write the parts in brackets.

- The words that are underlined are the explanations of arguments. Do not write those words as they are.

- You should choose one of the words from the set in { } (curly braces).

- **(X|Y)** means that either X or Y can be written. The '|' character stands of OR'ing.

  Example: **hebe** [-d] [–duck={wild,green_head,flying}] [file...]
  hebe is the name of the command. there are three optional parameters. First parameter is **d** (should be written with single dash). Second parameter is **duck** and has it's local parameter too. You should write one of the three parameters after **duck** parameter. (ex: –duck=wild). Third parameter tells you that, you can write one or more file names at the tail of the command. (The three dots lets you write multiple of that parameter)

1.6 Useful UNIX commands

The syntax of the commands given here may change from system to system. In order to see the exact syntax of a command on a specific system, use the manual pages. The working principles will be the same, but options may change.

1.6.1 Directory commands

- **ls** *[option] [(filename | directory)...]*: For each file or directory argument, ls lists the contents of the specified one. If no file are specified, contents of current working directory are listed. A typical ls output will list the files and directories only by their names. Most used option 'l' will list all files and directories in a single line with permissions, number of links, owner, size, last update date and name.

```
<mendelson:~/tmp> ls -l
total 4
lrwxrwxrwx   1 selcuk          4 Oct 14 17:55 dnm -> hebe
-rw-r--r--   1 selcuk         24 Oct 14 17:43 hebe
-rw-r--r--   2 selcuk         69 Oct 14 17:43 ordek
-rw-r--r--   2 selcuk         69 Oct 14 17:43 ordek
drwxr-xr-x   2 selcuk        512 Oct 14 17:44 subdir
<mendelson:~/tmp>
```

  In the example above, there are three files, one directory and one symbolic link. The leftmost bit indicates what kind of a file it is. Possible kinds are listed below.

d	Directory
l	Soft link (Not a real file. Used as a shortcut)
c	Character-type special file (Device files)
b	Block-type special file (Device files)
p	FIFO file (pipe)
s	Socket file (Used for communication between processes)

- **chmod** *mode (filename | directory)*: The mode of each named file or directory is changed according to mode, which may be absolute or symbolic.

– Absolute mode can be calculated simply by grouping the permissions to three octal numbers (modulo 8). If a permission is given, the corresponding bit is one and if it is not given, corresponding bit is zero. Note that this command can be used for both files and directories.

Example: You want to give read, write ,execute permissions to yourself, and read, execute permissions to group and others for the directory tmp. The corresponding permission bits are marked as one, and others are marked as zero. Starting from left, group three bits and change them to octal numbers by sequence. In our example, leftmost number is 7, middle number is 5 and rightmost number is 5.

```
  owner     group     other
 r  w  x   r  w  x   r  w  x         r:read permission
 1  1  1   1  0  1   1  0  1         w:write permission
 \     /   \     /   \     /         x:execute permission
   -|-       -|-       -|-
    7         5         5
```

Once you calculate the number you should give the command "chmod 755 tmp". If you ls the directory, the output will look as follows.

```
mendelson:~> ls -ld tmp
drwxr-xr-x   2 selcuk        512 Oct 14 17:26 tmp
mendelson:~>
```

If you ignore the leftmost letter 'd', remaining nine characters are the corresponding permission bits. If there is a '-' for a bit, this means that the corresponding user does not have that permission. In out example, group and other does not have the second permission (write permission).

– using *chmod* with symbolic permissions is easier in some situations. The symbolic permissions have the following syntax:
[who] *operator* *[permissions]*

  * who zero or more of the characters u, g, o and a specifying whose permissions are to be changed or assigned:

    u user's permissions g group's permissions o others' permissions a all permissions (user, group, and other)

    If who is omitted, the default is a.

  * operator either +, - or = stating how the permissions will be changed.

    + Add permissions to the current. - Remove permissions from the current = assign permissions absolutely. (This has the same effect with absolute mode parameter explained in the above item.)

  * permission any compatible combination of the following letters:

    r read permission w write permission x execute permission l mandatory locking s user or group set-ID t sticky bit u,g,o indicate that permission is to be taken from the current user, group or other mode respectively.

    For example, to add read and write permissions for group of the file example.ceng111, you have to give the following command "chmod g+rw example.ceng111". The resulting mode bits of the file will be as follows.

```
<mendelson:~> ls -l ceng111
-rwx--x--x   1 selcuk        716 Oct  6 11:01 ceng111*
<mendelson:~> chmod g+rw ceng111
<mendelson:~> ls -l ceng111
-rwxrwx--x   1 selcuk        716 Oct  6 11:01 ceng111*
```

As you can see from the example, the permissions of owner and others did not change. Also the existing permissions of group (x) was not deleted. The read and write permissions were added on.

- **cd** [directory]: cd changes the current directory to *directory*. Directory can be represented in three ways. Either as a *absolute path* or *relative path* or *relative path to user directory*. If the directory is not specified, then you will end up in your home directory.
  Example: Current directory is /home1/selcuk/tmp/subdir/second/dnm, and user directory is /home1/selcuk. All three commands below will end up in the same directory although different representations are used.

  ```
  <mendelson:~/tmp/subdir/second/dnm> cd /home1/selcuk/tmp
  <mendelson:~/tmp/subdir/second/dnm> cd ../../..
  <mendelson:~/tmp/subdir/second/dnm> cd ~/tmp
  ```

  Note that there exists a '~' character instead of /home1/selcuk in the prompt.
- **mkdir** directory: mkdir creates specified directories with the mode determined by the *umask* setting. The *dirname* can be given as one of the three path styles.
- **rmdir** directory: rmdir removes the specified directory if it is empty. (Does not contain files or subdirectories in it.)
- **pwd** : pwd prints the path of the working (current) directory starting from the root directory.

1.6.2 File commands

- **cp** *source-file... (target-file |u target-directory)*: source-file is copied onto target-file or source-file(s) are copied into the target-directory. After the cp command two copies of the same file exists.
- **mv** *source-file... (target-file | target-directory)*: mv moves (changes the name of) source-file to target-file or moves the source-file(s) into the target-directory. Note that after mv command, there is not a second copy of the file(s).
- **rm** *[option] filename...*: rm removes the entries for one or more files from a directory.
- **cat** *[-u] [filename...]*: cat reads each file in sequence and writes it on the standard output. The standard output is usually the screen you are working at. Cat allows you to see the contents of a file on the screen. If the file is longer than one page, the first pages will scroll up and last page will stay on screen.
- **more** *[filename...]*: more reads each file in sequence and displays them on standard output, page by page. You should be prompted to press a key after each page is displayed.
- **less** *[filename...]*: less works like more, but has some additional features. You can go up, down, to a specific line number with less.
- **grep** *[option] expression [filename]*: grep searches the input files (standard input default) for lines matching a pattern.

1.6.3 Other commands

- **man** [section] title: man formats a specified set of manual pages. If section is requested, man looks in that section of the manual for the given title. If no section is specified, all man pages from smallest section to the biggest are searched for a match.
- **passwd** *[user]*: This command changes a password associated with the user name (your own name by default). The owner must prove knowledge of the old password. If the systems are using NIS (you can login to many machines with the same password) then you should use **yppasswd** instead.
- **login** *[user]*: This command is used when a user initially signs on, or it may be used at any time to change from one user to another.
- **who** *[am i]*: who lists the login name, terminal name and login time for each logged in user. Only the users that are using the same system with you are listed (they are logged in to that machine).
- **finger** *[option] [user[@hostname]]*: finger gives the list of the users on the system (if there is NIS like our department, all the users in all the machines). If user is specified, only that user is searched and listed. If the hostname is specified, then the users at that system are listed.
- **talk** *user[@hostname]*: talk establishes a communication between two users, and lets them talk to each other by writing. Both users see each others screen. If the users are at different systems, then they should specify the hostname in the command. (user and @hostname must be written as a whole. No space. Ex: talk selcuk@bach.ceng.metu.edu.tr).
- **write** *user [terminal]*: write copies lines from your terminal to that of another user. Write does not establish a connection between users like talk, it only send the message to the users screen.
- **find** *pathname-list expression*: find recursively descends the hierarchy for each pathname in the pathname-list seeking files that match a boolean expression.
- **ps** *[option]*: ps prints information about processes.
- **kill** *[-sig] process-id*: kill sends the signal sig to the specified processes.
- **lpr** *[option] [filename]*: lpr causes the named files to be queued for printing.

2 Useful UNIX applications

2.1 The vi editor

2.1.1 History

The vi editor was developed at U.C. Berkeley as a part of Berkeley UNIX. Before vi was developed, the standard UNIX system editor was ed. The ed editor was line-oriented, making it difficult to see the context of your editing. Then ex, a superset of ed came. It's advantage was a display editing facility. When you are working in ex, you can enter to this facility by typing **vi**. When this facility became frequently used, the developers made it possible to directly enter this display facility. Instead of calling ex first, just give the **vi** command in the prompt.

Even if you enter the vi directly, alternating between two modes (ex and vi) is still possible.

2.1.2 Modes of operation

There is two major modes in ex editor. These are

- ex mode
- vi mode

Since you will use the vi mode in this course, we will skip the command and input modes of ex. All you have to know is that you can alternate between ex and vi with the following commands.

- To go from vi mode to ex mode, give **Q** command.
- To go from ex mode to vi mode, give **vi** command.

There are three modes of operation in vi.

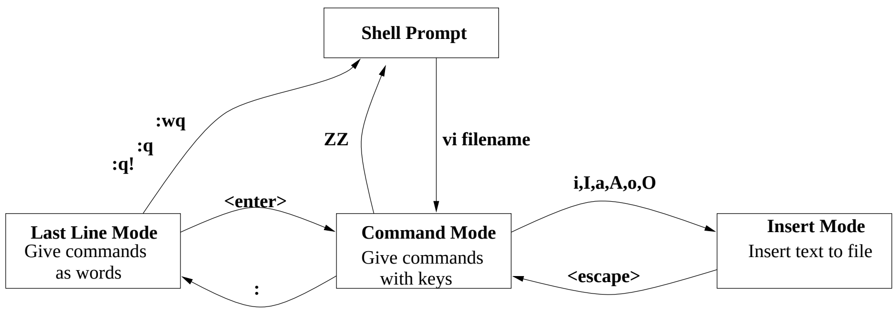
- Command mode
- Input mode
- Last line mode

While in command mode, vi accepts keystrokes as commands. Each key you press is recognized as a command. If you press the **Q** key in command mode, it will take you to ex editor. In input mode, vi accepts keystrokes as text, displaying the text as you enter it. All commands that start with colon '**:**' put vi in last line mode. The colon moves the cursor to the bottom line of the screen here you enter the rest of the command.

The main difference between the command mode and last line mode is that, each key you press can mean a different command in command mode. But the last line mode waits until you finish the command and press enter (like giving a command in prompt). Until enter is pressed the keys you press will appear in last line, and you can move back with backspace key. In command mode, a single key stroke has a meaning, and is executed at that instant. You do not press the enter key after finishing the command.

When you enter the vi, it opens in command mode. Several commands like insert and append puts you in insert mode. The **escape** key will always put you back to the command mode.

You can only write text or come one character back with backspace key in input mode. Do not use the arrow keys to move up, down, right or left when you are in insert mode. To go to a specific character on the text, first you have to enter command mode, then use the movement commands or cursor keys.



The commands in this manual are limited to practical use. To learn other commands and shortcuts, please refer to reference book for vi.

2.1.3 Entering vi editor

The simplest syntax of the vi command is as follows

vi [**-r** [**line number**]] [**file name...**]

If no file is specified, an empty screen will appear. If an existing file is specified, the contents of that file appears on the screen. The -r option lets you start from the last if no line number is given. If a line number is given, then you start from that line to the file. If more than one file is specified, you can edit them one by one.

2.1.4 Switching between modes

When you first enter vi, you are in command mode. To enter the last line mode press **:**. If successful, your cursor should appear in the bottom line of the screen. From last line mode, you return to command mode after pressing enter. Only way of going to insert mode is pressing **i,I,a,A,o** or **O** in the command mode. To return back to command mode from insert mode, press the **escape** key.

2.1.5 Command mode keys

The italic *n* stands for a positive integer in the commands below. If it is not specified, then default value is 1.

- Moving around the text
  - *n***l** or **right arrow**: move the cursor *n* character right.
  - *n***h** or **left arrow**: move the cursor *n* character left.
  - *n***j** or **down arrow**: move the cursor *n* line down. (cursor will be at the same column if line is long enough. If new line is short, cursor goes to the end of that line.
  - *n***k** or **up arrow**: move the cursor *n* line up.
  - *n***w**: move the cursor to the start of *n* next word.
  - *n***b**: move the cursor to the start of *n* previous word.
  - *n***e**: move the cursor to the end of the *n* next word.
  - **$**: move the cursor to the last character of current line.
  - ^(caret): move the cursor to the first character of current line.
  - **%**: show the matching parenthesis of current parenthesis.
- Inserting text (once you press one of these keys, you go to insert mode.)
  - **a**: append after cursor.
  - **A**: append after end of current line.
  - **i**: insert before cursor.
  - **I**: insert before beginning of current line.
  - **o**: open new line below current line and insert to new line.
  - **O**: open new line above current line and insert to new line.
- Deleting text
  - *n***x**: delete *n* characters beginning from current character.

- n**X**: delete previous n characters.
- n**dw**: delete next n words beginning with current.
- n**db**: delete previous n words.
- n**dd**: delete n lines beginning with current line. Puts the deleted lines to buffer.
- **D**: delete from cursor to end of current line.

- Copying and placing text
  - n**yy** : Yank n lines to buffer.
  - **p** : Put yanked text before cursor; also puts last deleted text.

2.1.6 Last line mode commands

- edit, write and quit a file
  - **:w** filename : save file with name filename, but do not exit. Omitting filename saves file with current name.
  - **:w!** filename : save file with name filename overriding normal checking.
  - **:q** : Leave vi. If you made any changes, you will be prompted to save the changes first.
  - **:q!** : Leave vi without saving any changes since last write.
  - **:e** filename : Edit file named filename. If you made any changes in current file, you will be prompted to save them or use :e!.
  - **:e!** filename : Edit file named filename disregarding changes since last write. If no filename is specified, current file is re-edited.
- Inserting file
  - **:r** filename : Read the file named filename and insert it beginning from the current line.
- set commands
  - **:set showmatch** : Shows the matching left parenthesis when you type a right parenthesis.
  - **:set tabstop=**n : Set the tab length to n spaces. (Useful for indentation).

2.2 E-Mail : elm and mail

E-Mail stands for electronic-mail in network environments. In UNIX systems there are some programs to manage mailing facilities. By using these facilities you can send files (probably a text file that is your mail) to other users in the system (any body in the Internet) and read your mails that are send by others.

`Sending mail :`

First you write your mail using a text editor (like vi) and save it as file (assume the file name is 'mymail'). Next you must know the Internet address of the one that you want to send your mail. It is simply the username if you are in the same domain (assume the address is 'e10XXXX@ceng.metu.edu.tr'). Then

```
mail e10XXXX < mymail
or
elm e10XXXX < mymail
```

An other way is to run elm with a mail address.

```
elm e10XXXX
```

Then elm starts in send only mode. It starts an editor (vi), so you write your mail, save it and exit (from vi). Elm will ask for conformation on sending and tell you if the mail was sent or not.

`Reading mail :`

When you start a shell (for example when you login), it will warn you if you have mail in your mailbox. To read your mails simply run 'elm'. Use cursor keys and enter for reading your mails. You can delete the mails you read, save them if you wish or just leave them in your mailbox.

Further information about elm commands is available in elm by giving 'help' command (by pressing '?').

2.3 Tin : News Reader

When you enter tin, you see the groups that you have subscribed. You can read messages posted to these groups by using cursor keys and enter. You exit from articles, threads, groups and finally the program by pressing *'q'*. You can see other groups that you did not subscribed by *'yank in/out'* command. Then you can highlight the group by moving the cursor and *'subscribe'* to that group. Also you can *'unsubscribe'* a group that you do not want to read. When you enter a group, you can *'list thread'* of an article, that is the list of replies to an article. When you are in a group, you can *'post'* an article to the group. Similarly when reading an article or a thread, you can *'post a followup'* that is a reply to a message. You can also *'save'* an article or a thread that you wish to keep in your own directory (in regular the *'save'*d article is put under "News" directory under your home).

How to give these *'commands'* and other commands is listed in tin program by pressing *'h'* (help command) in group selection menu, group menu, thread menu and while reading an article. Also pressing *'H'* key toggles mini help menu for quick referencing simple commands.

Do not forget that we have a "test" group if you just want to try posting something to "tin".

2.4 The shell

The shell executes a program when you give it a command in response to its prompt. The output of the command is displayed on the standard output.

2.4.1 Command line

The line that contains the command, options and arguments, is called the *command line*

A typical command line should be like :

```
command [opt1] [opt2] ... [optM] [arg1] [arg2] ... [argN] <return>
```

As you can see, there should be at least one space between command, each option and each argument. When you forget the space, shell tries to interpret the whole word as a single command or a single argument.

Each command has a *command name*. There has to be at least one command name in a command line. In the example below, command name is **ls**

An *option* is an argument that modifies the effect of a command. Options are specific to the command, and usually written with a '-' (minus) character in front of them. This minus character is used by the command to distinguish the options from arguments. Each command has its own options. In the example below, **-lF** is two options written together. They could have been written as **-l -F** too.

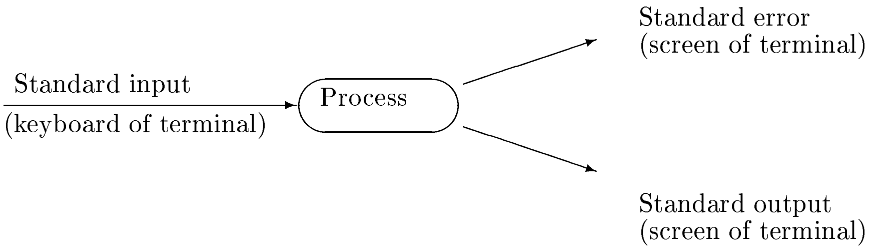
An *argument* is a filename, string of text, number or some other object that a command acts on. In our example **/usr/local/bin** and **/tmp** are arguments of the command.

```
<mendelson:~> ls -lF /usr/local/bin /tmp
```

2.4.2 Standard input,output,error

In UNIX systems, there are three special files that are predefined for each process. These special files are

- *Standard input* is the file that is used for getting data to the command. This file is defined as the keyboard by default. So when you press the keys, system gets the keys and send them to the command using the standard input. If standard input is changed to a regular file, then system gives the contents of that file to the command.
- *Standard output* is the file that is used for sending the normal output of the command execution. By default this file is defined as the screen you are working at. So you will see the list of the directory on the screen after "ls" command. If standard output is changed to a regular file, then the outputs of command will be written in that file. (you won't see them on the screen any more)
- *Standard error* is the file that is used for sending abnormal messages while executing a command. This file is also defined as the screen. Usually meaningful error messages are written to this file. If standard error is changed to a regular file, then the error messages are written in that file. (you won't see them on the screen any more)



The default configuration of input and output

Note that in UNIX everything can be used as a file. A directory, memory, network I/O and even the keyboard are defined as special types of file. Do not get confused with types of a file. When you see *file*, it means a regular file. Other special files have their own names.

Nearly every program in UNIX sends its output to the standard output, and errors to standard error. Most commands accept file names as arguments and use them as their input, but if no file name is specified, then standard input is used for input. In the example below, **cat** command will use its predefined standard input, since no file is specified.

```
<mendelson:~>cat
trial and error
trial and error
```

```
is it okay????
is it okay????
why is it repeating them two times?
why is it repeating them two times?
^D
```

As you can see, the command gets each line from the standard input(keyboard) and displays it on standard output (screen). When you press enter, the contents of that line is sent to the command, and it displays them on screen. To tell the command that the standard input is ended (you don't want to enter anything more from keyboard), you should press `^D` (Ctrl-d) keys together in a new line. Press the d key for only one instant, don't hold it down. When the command gets this character (`^D` end of file), it finishes the execution.

2.4.3 Redirection

If you want to put a commands output to a file, all you have to do is redirecting it. The '>' '<' '>>' characters can be used for redirection. Each character has a special meaning.

- > character is used to redirect the output of a command to a file. More formally, you are redirecting its standard output to a file. So when you give the command below, the result will not be displayed on the screen.

  ```
  <mendelson:~>ls -l /tmp > ls.output_file
  ```

  The system puts the data coming from standard output line to a file. Note that the file is created if it does not exist. If it exists, you loose the old content of that file.

- < character is used to redirect the input of a command from a file. More formally, you are redirecting the standard input from a file. In the command below, the inputs are read from the file, and written to standard output (screen).

  ```
  <mendelson:~>cat < tmp
  ```

  Note that the command above is the same as "cat tmp" command. The command itself does not read the content of the "tmp" file, but the system reads them and sends them to the command from standard input.

- >> is used to redirect the output of a command to the end of a file. If the file does not exist, then it is created. If a file exists, then outputs are written to the end of the file. (you do not loose the old data in that file)

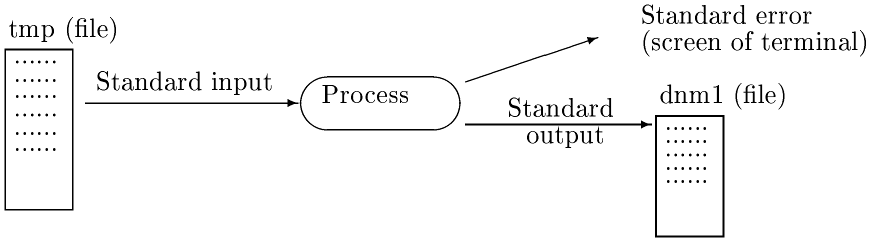
  ```
  <mendelson:~>ls -l /usr >> ls.output_file
  ```

  After the command above, there should be first the listing of /tmp from the previous command, and after that, the listing of /usr.

The command below uses the "tmp" file as standard input, "dnm1" file as the standard output.

```
<mendelson:~>cat < tmp > dnm1
```

The contents of "tmp" file is fed to **cat** and outputs are written to "dnm1" file. If any errors occur, then the error messages would be written to standard error, so they will appear on the screen. Since cat writes the input to output line by line, after this command, "tmp" and "dnm1" files will have the same content. This is a way of copying files.



The configuration after "cat < tmp > dnm1" command

2.4.4 Pipe

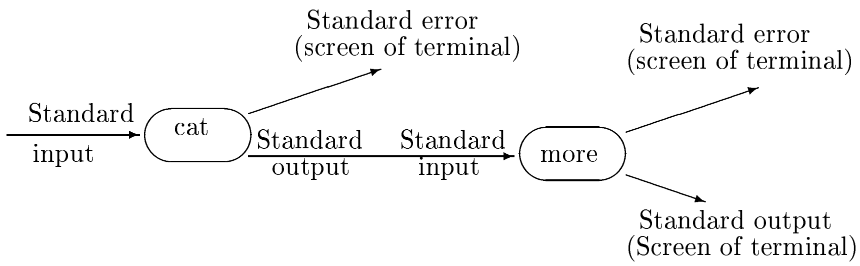
The | represents a pipe between two commands. There may be more than one command running at the same time, and one program can require the output of another. For example, if **more** command did not have a file name argument, you would have to use the standard input. To see the output of **ls** command page by page you had to first put the output of ls to a file, then redirect that file as the standard input of more and delete that file.

```
<mendelson:~>ls -l > tmp_file
<mendelson:~>more < tmp_file
<mendelson:~>rm tmp_file
```

If you want to send the standard output of a process to the standard input of another, then you should put | character between them. The command below has the same effect with 3 commands above.

```
<mendelson:~>ls -l | more
```

When you put | character between two command, the system gets the output from left process and gives them to the right process. In our example outputs of "ls" are taken and fed to "more". The picture below shows the connection between two commands.



The configuration after "ls -l | more" command

2.5 Wildcards

In a command line, in order to write many file names that have a common property, wildcards can be used. There is a huge set of wildcards which are used to specify regular expression. These will be learned by heavy usage of the system. We will show the most frequently used wildcards.

- **?** stands for one character. For example if you want to list all the files that start with *fin* and has one additional character in the name, give the following command: ls -l fin*
  .

```
<mendelson:~/ceng111> ls -l
-rw-r--r--   1 selcuk          0 Oct  6 11:14 fin
-rw-r--r--   1 selcuk          0 Oct  6 11:14 fina
-rw-r--r--   1 selcuk          0 Oct  6 11:14 final
-rw-r--r--   1 selcuk          0 Oct  6 11:14 finb
-rw-r--r--   1 selcuk          0 Oct  6 11:14 grab
<mendelson:~/ceng111> ls -l fin?
-rw-r--r--   1 selcuk          0 Oct  6 11:14 fina
-rw-r--r--   1 selcuk          0 Oct  6 11:14 finb
```

Note that only the files that start with fin and has one more letter are displayed.

- * Stands for zero or more letters. In the above example to list all files that start with *fin*, give the following command: ls -l fin*
  .

```
<mendelson:~/ceng111> ls -l fin*
-rw-r--r--   1 selcuk          0 Oct  6 11:14 fin
-rw-r--r--   1 selcuk          0 Oct  6 11:14 fina
-rw-r--r--   1 selcuk          0 Oct  6 11:14 final
-rw-r--r--   1 selcuk          0 Oct  6 11:14 finb
```

The wildcards are expanded by the shell, not the command itself. This means that the ls command we gave in the following command is executed after the wildcards are expanded. The resulting command and arguments passed to *ls* is as follows: ls -l fin fina final finb .

3 Scheme

/subsectionUsing scheme
To enter the mit-scheme interpreter, first login to the systems in our department, then give the command '**scheme**'.

After a while (it takes time to load the program from network), your screen will clear and you will get a prompt like below.

```
Scheme saved on Sunday November 21, 1993 at 9:15:23 PM
  Release 7.3.1
  Microcode 11.146
  Runtime 14.166

1 ]=>
```

The number in the prompt shows the level of the interpreter. You should work in the first level. To return back to first level from higher levels, press *Ctrl-g* key. Higher levels will be used later for debugging. For further information on control keys, press *Ctrl-c* key and *?* key.

When you want to load a file (let the interpreter interpret the lines in that file) write the command **(load "filename")** in the interpreter.

To exit from the scheme interpreter, use the *Ctrl-d* key.