

CENG-111 Introduction to Computer Engineering Concepts

Laboratory Manual

Prepared by: İlker Altıntaş, Hasan Ulusoy & Selçuk Şenkul

October 2014

Updated by: M. Akif Akkuş & Hazal Moğultay

Contents

1	Introduction to UNIX	2
1.1	What is Unix and Linux	2
1.2	How to use the system	3
1.2.1	Using the Laboratories	4
1.3	Important Notes about department laboratories	4
1.4	Files and Directories	5
1.5	Permissions	7
1.6	Reading manual pages	8
1.7	Useful Unix commands	9
1.7.1	Directory commands	9
1.7.2	File commands	12
1.7.3	File compression&uncompression commands	12
1.7.4	Other commands	13
1.8	The shell	14
1.8.1	Command line	14

1.8.2	Standard input,output,error	14
1.8.3	Redirection	15
1.8.4	Pipe	17
1.9	Wildcards	17
1.10	Customizing Your Shell	18
2	Useful UNIX applications	19
2.1	Secure Shell (SSH)	19
2.2	SFTP (Secure File Transfer Protocol)	20
2.3	The nano text editor	21
2.4	The vi text editor	21
2.4.1	History	21
2.4.2	Modes of operation	22
2.4.3	Entering vi editor	23
2.4.4	Switching between modes	24
2.4.5	Command mode keys	24
2.4.6	Last line mode commands	25
2.5	E-Mail	26
2.5.1	Mozilla Thunderbird	26
2.5.2	Webmail	26
2.6	News Groups	27
2.6.1	Mozilla Thunderbird	27
2.6.2	COW News	27
2.7	Security: Protecting your account	28

1 Introduction to UNIX

1.1 What is Unix and Linux

Unix is a *multiuser, multiprocessing* operating system. 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. The GNU Project started in the 80's to produce a "*Free Software*" version of Unix; which can be used, modified and distributed without limitations. It was led by Richard Stallman, who was an Artificial Intelligence (AI) researcher at Massachusetts Institute of Technology(MIT).

At the beginning of 90's, GNU Operating System had compilers, editors, shells and other programs, but didn't have a complete operating system kernel. The Linux kernel filled this gap, and GNU/Linux system provided a completely "*free*" (as in freedom) version of Unix. GNU/Linux (or shortly *Linux*) is being developed and supported by thousands of people and companies, all over the world. You are advised to install Linux to your computer.

Unlike some operating systems like DOS, more than one user can use a Unix 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. 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 *bash* (bourne again shell), *tcsh* (turbo C shell) and *zsh*. Modern systems include *Graphical User Interfaces (GUIs)* to ease daily tasks. Linux systems come with a number of GUIs; *GNOME*, *KDE*, *xfce*, *icewm*, *enlightenment*, *fluxbox* are commonly used.

1.2 How to use the system

Login process: Department systems are protected by passwords, you use your user name and password to login. 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. Do not shut down the system, and do not remove any cable of the system. Unix machines are designed to be running 7/24. If you think that a computer is not working, contact the system administrators.

When you decide to leave the computer,

- **do not!** close the screen and go away.
- **do not!** shut down the computer.

Instead, choose the logout option from the menu. 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 anything to and from your account (you are responsible for the damage someone does from your account.)

1.2.1 Using the Laboratories

Logging in

1. Check whether the screen is closed or open. If there is light on the on/off button, 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, write your password and press enter.
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 enter the machine.
5. There will not be any console(command line, terminal emulator) on the screen initially. To open one, use the applications menu to open a terminal emulator.

Logging out

1. To log out from the system, use the menu to choose “Log out” option.
2. If this was successful, you will see the login screen again. If nothing happens, try first step again.
3. When you log out, do not close the machine!!..

1.3 Important Notes about department laboratories

- Department laboratory doors can only be opened with your student id card. If the card reader near the door does not recognize your card, you should visit B-Z05.
- If you decide to take a break while you are working with a machine in the laboratory, do not leave the laboratory for more than 15 minutes. Every student has the right to log out an empty machine after 15 minutes. If you will not be back in 15 minutes, log out from the machine. You can login to another machine if previous one is occupied when you come back to the laboratory.
- Do not lock the machine before getting out of the laboratory.
- All department laboratories are equipped with security cameras which are always active.

You should read the documentation about department labs and computing infrastructure, from departments web site at <http://www.ceng.metu.edu.tr/doc/services/index>. It is prepared and updated by the system administrators of our department and contains useful and the latest information on accessing department machines, department services, labs, news server and e-mail service, etc.

You should also read the Computer & Networking Ethics document on that site, thoroughly. You should have signed this document in the Laboratory, but even if you didn't, you are bound by its terms.

<http://www.ceng.metu.edu.tr/doc/services/ethics>

1.4 Files and Directories

What does **file** mean? The description below is taken from the Webster dictionary.

File: *noun*, Date:1525

a device(as a folder, case, or cabinet) by means of which papers are kept in order

You can think of the files in a computer as books where information is kept. Similar to books, files can be very small or very large depending on the contents. But unlike regular books, size of the computer files can be adjusted. You can make an existing file smaller by removing data from it. Similarly to make a file larger you can add data. The files can be used to store any combination and amount of text, images, sound, video, numbers, etc. You have to store every data to a file if it is to be preserved. Another description of file from the free on-line dictionary of computing (09 feb 02):

file : An element of data storage in a file system... the prototypical file has these characteristics:

- It is a single sequence of bytes.
- It has a finite length.
- It is stored in a non-volatile storage medium.
- It exists (nominally) in a directory.
- It has a name that it can be referred to by in file operations, possibly in combination with its path.

Now imagine a library with many books of different size and content. If all the books in the library were just lying on the floor with no grouping at all, you can imagine how hard it would be to find the book you are looking for. To overcome this difficulty, libraries use the book shelves to put books of similar content to the same shelf. The shelves are then labeled for you to make the search easier. In the computer world things are very similar. The name we use for a book shelf is **directory**. Directory (shelf) is used to group files

(books) and other directories (smaller shelves) in it while a file may contain only data in it.

The special character ‘/’ represents the root directory in every Unix system. This root directory holds every other directory and file in it. You can think the root directory as the library itself. Everything must be inside the library. The directories inside a directory are called it’s *subdirectories*. The directory structure grows like a tree which is upside down.

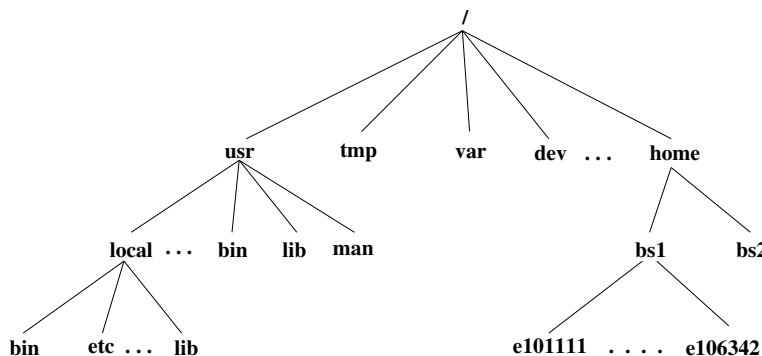


Figure 1: Directory Structure

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. You can think of the *home* directory as a book shelf in the library which belongs to you. How you organize that shelf and what files you put to it is up to you.

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 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 your 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” as “/home/bs1/e101111” or “e101111” or simply “~”. 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
.cpp	C++ source file
.h	header file
.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.

1.5 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	w	x	r	w	x	r	w	x	
									r:read permission
									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.

An easy way to tell the system that you want every new file in some certain mode (permission) is to set a variable named “umask”. For example “umask -S 022” makes each new directory’s permission to “readable and enterable”, and each new file’s permission to

“readable and executable” . The three octal numbers are the subtraction of wanted mode from 777. Calculating the wanted modes can be found in section 1.7.1 on page 9.

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.

1.6 Reading manual pages

During your education in this department, you will need to learn many different commands to effectively use the Unix systems. Trying to memorize all the commands and how they are used will be a waste of your time. The commands you use most frequently will stay in your mind, but most of them will be forgotten. To remember how a command works, or to learn a new command, the only thing you need will be the manual pages present in the Unix system you use.

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** [-v] [-duck={wild,green_head,flying}] [file...]

hebe is the name of the command. there are three optional parameters. First parameter is **v** (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)

The manual page will give an explanation about the use of this command and the use of the arguments.

1.7 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.

1.7.1 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.

```
e1234567@inek7:~$ ls -l
total 4
lrwxrwxrwx   1 e1234567          4 Oct 14 17:55 dnm -> hebe
-rw-r--r--   1 e1234567        24 Oct 14 17:43 hebe
-rw-r--r--   2 e1234567        69 Oct 14 17:43 orde
-rw-r--r--   2 e1234567        69 Oct 14 17:43 orde
drwxr-xr-x   2 e1234567       512 Oct 14 17:44 subdi
e1234567@inek7:~$
```

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	
1	1	1	1	0	1	1	0	1	r:read permission
									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.

```
e1234567@inek7:~$ ls -ld tmp
drwxr-xr-x  2 e1234567          512 Oct 14 17:26 tmp
e1234567@inek7:~$
```

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 our 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.

```
e1234567@inek7:~$ ls -l ceng111
-rwx--x--x  1 e1234567          716 Oct  6 11:01 ceng111*
e1234567@inek7:~$ chmod g+rw ceng111
e1234567@inek7:~$ ls -l ceng111
-rwxrwx--x  1 e1234567          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 /home/bs01/e1234567/tmp/subdir/second/dnm, and user directory is /home/bs01/e1234567. All three commands below will end up in the same directory although different representations are used.

```
e1234567@inek7:~/tmp/subdir/second/dnm$ cd /home/bs01/e1234567/tmp
e1234567@inek7:~/tmp/subdir/second/dnm$ cd ../../..
```

```
e1234567@inek7:~/tmp/subdir/second/dnm$ cd ~/tmp
```

Note that there exists a ‘~’ character instead of /home/bs01/e1234567 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.

1.7.2 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.

1.7.3 File compression&uncompression commands

- **tar** [*-cf*] *file.tar files*: Create a tar named file.tar containing files. Files should be under the current directory. File.tar will also be created under the same directory. All files to be compressed should be named consecutively and be separated by a space.
- **tar** [*-xf*] *file.tar*: Extracts the files from file.tar to the current directory.

- **tar** [-*czf*] *file.tar.gz files*: Creates a tar named *file.tar.gz* with Gzip compression. Files should be under the current directory. *File.tar* will also be created under the same directory. All files to be compressed should be named consecutively and be separated by a space.
- **tar** [-*xzf*] *file.tar.gz*: Extracts the files from *file.tar.gz* to the current directory.

1.7.4 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.
- **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*]: *finger* gives the list of the users on the system. If *user* is specified, only that user is searched and listed.
- **talk** *user*: *talk* establishes a communication between two users, and lets them talk to each other by writing. The other party must respond to the talk request also, by running *talk* himself. Then both users see a split screen, where they can write in their area and read the corresponding party's response in the other.
- **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. For example the command "**find** ~ -iname '*hede*" searches for all filenames containing "hede" in your home directory.
- **ps** [*option*]: *ps* prints information about processes.
- **kill** [-*sig*] *process-id*: *kill* sends the signal *sig* to the specified processes.
- **wc** [-*b/w/l*]: counts the number of **bytes/words/lines**.

1.8 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.

1.8.1 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/bin** and **/tmp** are arguments of the command.

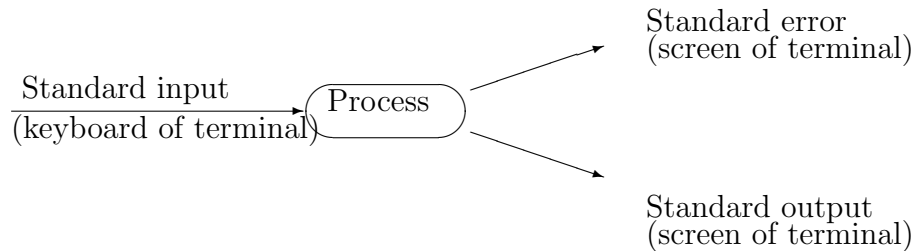
```
e1234567@inek7:~$ ls -lF /usr/bin /tmp
```

1.8.2 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.

```
e1234567@inek7:~$ 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.

1.8.3 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.

```
e1234567@inek7:~$ 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).

```
e1234567@inek7:~$ cat < test
```

Note that the command above is the same as “cat test” command. The command itself does not read the content of the “test” 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)

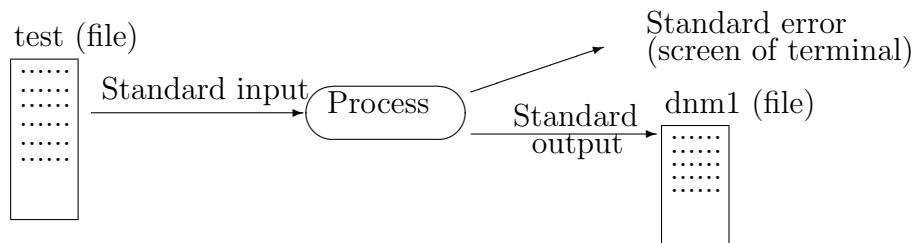
```
e1234567@inek7:~$ ls -l /usr >> ls.output_file
```

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 “test” file as standard input, “dnm1” file as the standard output.

```
e1234567@inek7:~$ cat < test > dnm1
```

The contents of “test” 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, “test” and “dnm1” files will have the same content. This is a hacky way of copying files.



The configuration after “cat < test > dnm1” command

1.8.4 Pipe

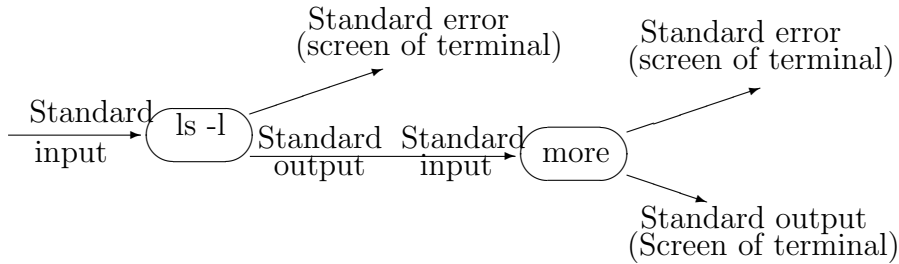
The `|` represents a pipe between two commands. There may be more than one command running at the same time, and one program can require the output of another. For example, if `more` command did not have a file name argument, you would have to use the standard input. To see the output of `ls` command 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.

```
e1234567@inek7:~$ ls -l > tmp_file
e1234567@inek7:~$ more < tmp_file
e1234567@inek7:~$ 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.

```
e1234567@inek7:~$ 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

1.9 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*`

```
e1234567@inek7:~/ceng111$ ls -l
-rw-r--r--  1 e1234567      0 0ct  6 11:14 fin
-rw-r--r--  1 e1234567      0 0ct  6 11:14 fina
```

```

-rw-r--r--  1 e1234567          0 Oct  6 11:14 final
-rw-r--r--  1 e1234567          0 Oct  6 11:14 finb
-rw-r--r--  1 e1234567          0 Oct  6 11:14 grab
e1234567@inek7:~/ceng111$ ls -l fin?
-rw-r--r--  1 e1234567          0 Oct  6 11:14 fina
-rw-r--r--  1 e1234567          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*`

```

e1234567@inek7:~/ceng111$ ls -l fin*
-rw-r--r--  1 e1234567          0 Oct  6 11:14 fin
-rw-r--r--  1 e1234567          0 Oct  6 11:14 fina
-rw-r--r--  1 e1234567          0 Oct  6 11:14 final
-rw-r--r--  1 e1234567          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`.

1.10 Customizing Your Shell

You can customize your shell (which is *bash* on department machines) by writing commands in your `~/.bashrc` file. The commands found in this file will be executed when you open a new shell.

You can run any command in the `.bashrc` file but most useful ones are:

- **alias**: With the `alias` command you can give a name to a –possibly– long command line. For example this line in the `.bashrc` file will create an alias named **myls** which you can run from the command line as if it is a real command.

```
alias myls='ls --almost-all -F --group-directories-first'
```

- **export**: You can set variables using `export` command. Variables are used by some programs or the shell itself. The following commands set the default editor as `vim`, and increase the amount of command line history, which is accessible by pressing the up arrow key.

```
export EDITOR=vim
export HISTSIZE=2500
```

2 Useful UNIX applications

2.1 Secure Shell (SSH)

As we mentioned before, Unix workstations are multiuser systems. Since there is only one console for each workstation, then how can other people use that machine? There are several ways of doing this. **Secure Shell (SSH)** is one of those ways. If you want to use a workstation but the console is occupied, then you can login to any machine, open a terminal and issue the `ssh` command to connect to the remote machine. For example, to connect to the machine `divan`, you give the command “`ssh divan`”. While writing your password, there will be no output on the screen. If your password is correct, `divan` accepts you and the terminal will look like:

```
e1234567@inek7:~$ ssh divan
e1234567@divan.ceng.metu.edu.tr's password:
Linux divan 2.6.26-2-xen-amd64 #1 SMP Thu Sep 16 16:32:15 UTC 2010 x86_64
```

The programs included with the Debian GNU/Linux system are free software; the exact distribution terms for each program are described in the individual files in `/usr/share/doc/*/copyright`.

```
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Thu Jul 15 14:46:37 2010 from tassadar.ceng.metu.edu.tr
e1234567@divan:~$
```

When you see the name of the remote machine at the prompt, you can understand that any command you give from that window will work on the remote machine. This is sufficient if your job does not contain the use of any graphical user interfaces (GUI). There are ways to use a remote machine including the GUI applications, but these are not needed for this course. You may learn them on your own if you like.

When you are done with the remote host, you have to close your ssh session. To close ssh, simply type **exit** and you should be disconnected from the remote host.

To connect to department machines from outside the department, you should first connect to *divan* or *login* machine. (*divan* is also known as *external*). After that you can connect to any inek machine. While connecting from outside the department, you should give your department username and the full name of the host:

```
ssh e1234567@divan.ceng.metu.edu.tr
```

```
ssh e1234567@inekXX
```

2.2 SFTP (Secure File Transfer Protocol)

FTP (File Transfer Protocol) is a protocol which allows a user on one computer to transfer files to and from another computer that can be reached over a network. “Secure FTP” is the version of it running over the encrypted connection of SSH. The Secure FTP program is called as **sftp**.

To make an sftp connection to a computer, you can give the address of the remote machine from command line; like

```
e1234567@inek7:~$ sftp e123456@beluga.cc.metu.edu.tr
```

After username and password verification, an ftp prompt appears to give commands. Most commands resemble Unix commands, like; ls, cd, mkdir etc. which are executed in the remote machine. To give commands in the local machine, you must use their “local” versions; lls, lcd, lpwd.

There are two commands for file transfer:

put *filename* : transfer file from your (local) computer to the remote computer

get *filename* : transfer file from the remote computer to your (local) computer.

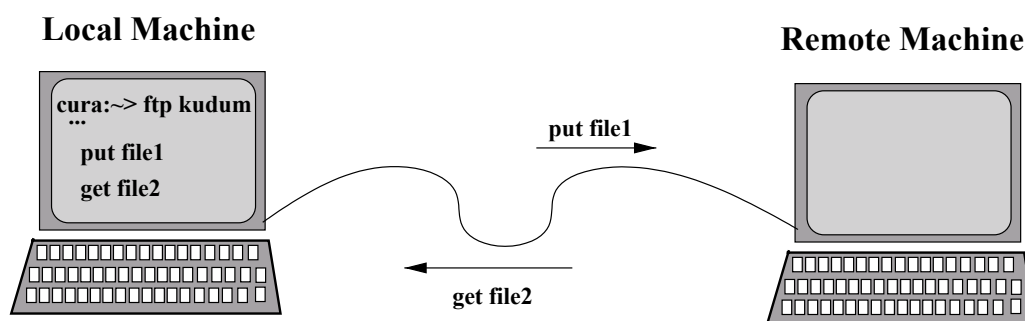


Figure 2: How ftp works

When you are done with the file transfer, give the command **quit** to close your connection.

A similar command for file transfer is **scp**. You can use scp command to transfer files between a remote server and your machine. You can also use scp for transferring a directory, including all files under it.

Here are some example usage of scp command. The first copies the “mytestfile” in current directory to beluga, under “mydirectory”

```
e1234567@inek7:~$ scp mytestfile e123456@beluga.cc.metu.edu.tr:mydirectory/
```

This second command copies the “test-directory” on beluga to the “dnm” directory under your home.

```
e1234567@inek7:~$ scp -r e123456@beluga.cc.metu.edu.tr:test-directory ~/dnm/
```

2.3 The nano text editor

The nano editor is the easiest and the most powerless editor in the unix environment. To start the editor simply type **nano** on the command line. An empty screen will appear. You can simply press the keys on the keyboard to write something. Use the arrow keys to move up, down, right or left on the screen.

There are two lines at the bottom which explain the available command at that time. **^X Exit** means that if you press **x** key while pressing the control **Ctrl** key, you will execute the command exit. If you did not save the new text you have written, the editor will ask you if you would like to save it or not. Press **Y** key and enter a file name when you are prompted.

To edit an existing file, you can start the nano editor by writing **nano filename** on the command line. Nano will open the file named *filename* and you can start editing it. If nano is already open, you can open another file by giving the command **^R Read File**. write the name of your file when asked. If you do not remember the exact name use **^T To files** command to choose your file from the appearing list.

2.4 The vi text editor

2.4.1 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 could enter to this facility by typing **vi**. When this facility became frequently used, the developers made it possible to directly enter this display facility and the vi editor was born. Even if you enter the vi directly, alternating between two modes (ex and vi) is still possible. Vi has many clones, and **vim** is a very powerful vi-clone with tons of features. It is installed on department machines and actually **vim** is run even if you give the **vi** command. In

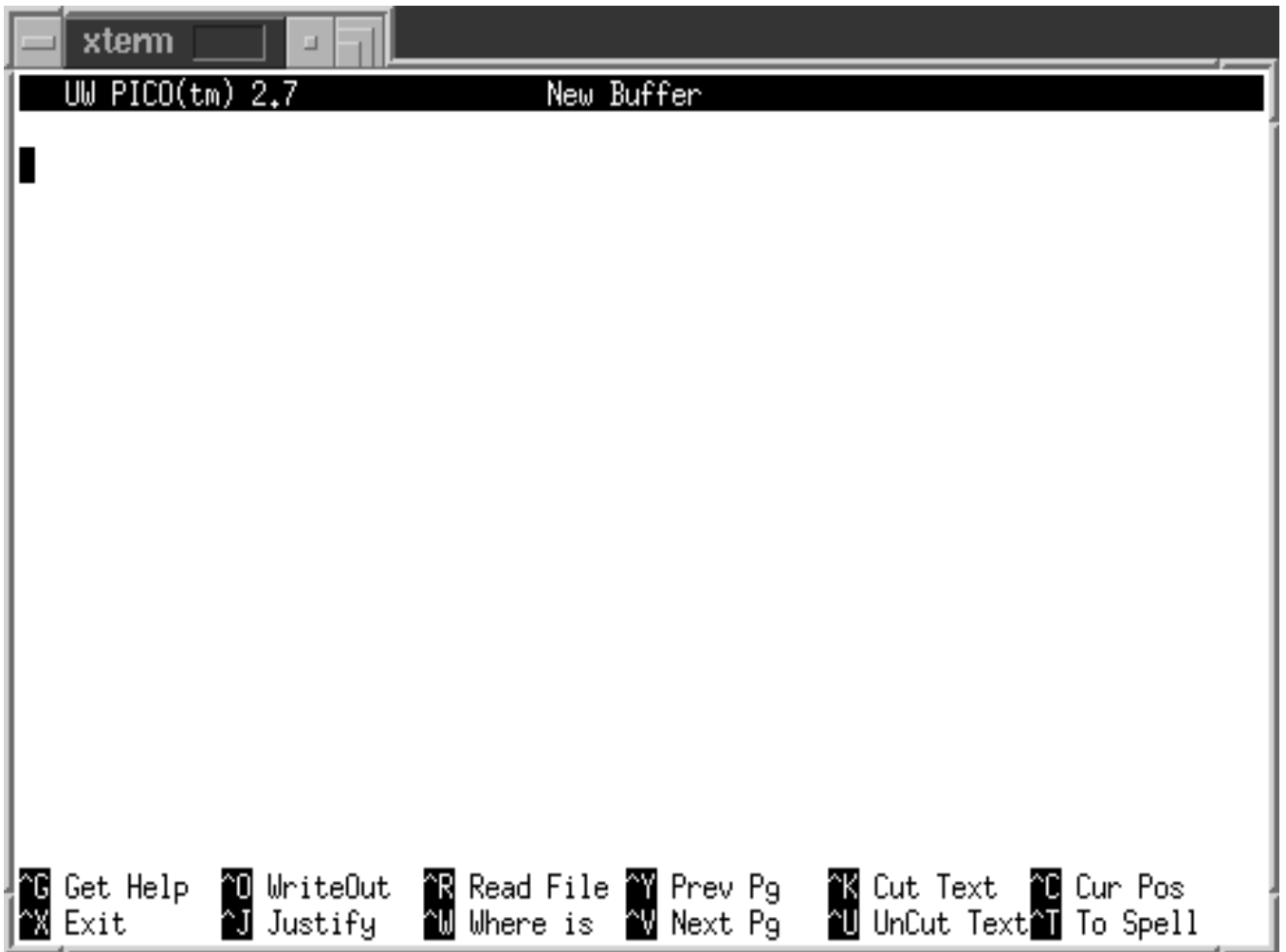


Figure 3: Typical nano screen

this manual vi and vim are used interchangeably, but on computer systems, running the command as **vim** would enable more features of the vim editor.

2.4.2 Modes of operation

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 any one of the **i,I,a,A,o,O** keys in command mode, it will take you to the input mode.

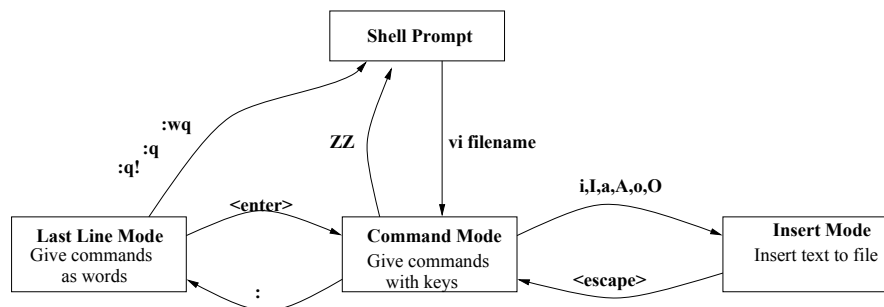
In input mode, vi accepts keystrokes as text, displaying the text as you enter it.

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. A single key stroke has a meaning, and is executed at that instant. You do not press the enter key after finishing the command. 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.

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 delete text with backspace or delete keys in input mode.



The commands in this manual are limited to practical use. To learn other commands and shortcuts, please refer to reference book for vi on the Internet, the **:help** last line command, or the **vimtutor** command.

2.4.3 Entering vi editor

The simplest syntax of the vi command is as follows

vi [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. If more than one file is specified, you can edit them one by one.

2.4.4 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.

2.4.5 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
 - `nl` or **right arrow**: move the cursor *n* character right.
 - `nh` or **left arrow**: move the cursor *n* character left.
 - `nj` 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.
 - `nk` or **up arrow**: move the cursor *n* line up.
 - `nw`: move the cursor to the start of *n* next word.
 - `nb`: move the cursor to the start of *n* previous word.
 - `ne`: 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
 - ***nx***: delete *n* characters beginning from current character.
 - ***nX***: delete previous *n* characters.
 - ***ndw***: delete next *n* words beginning with current.
 - ***ndb***: delete previous *n* words.
 - ***ndd***: 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
 - ***nyy*** : Yank *n* lines to buffer.
 - ***p*** : Put yanked text before cursor; also puts last deleted text.
- Search and replace text
 - ***/w***: Search forward for *w*
 - ***?w***: Search backward for *w*
 - ***:%s/old/new*** replace next occurrence of *old* with *new*

2.4.6 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).

2.5 E-Mail

E-Mail stands for electronic-mail in network environments. By using e-mail client programs, and utilizing the e-mail servers; you can exchange electronic mail with other people.

You should follow your department e-mail regularly, since it can be used by your instructors, assistants or system administrators for contacting you. Your department e-mail is also for your personal use, as long as you obey the rules in Computer & Networking Ethics document.

2.5.1 Mozilla Thunderbird

Mozilla Thunderbird is a popular GUI program for e-mail access. It is installed on department laboratories you can set up your department e-mail and newsgroup accounts to Thunderbird by following the instructions given in the web pages:

<http://www.ceng.metu.edu.tr/doc/services/email/thunderbird>

<http://www.ceng.metu.edu.tr/doc/services/news/thunderbird>

You can also download and install Thunderbird on your personal computer, by visiting <http://getthunderbird.com/>

2.5.2 Webmail

You can use a webmail to read & write e-mail in a web browser. Department's webmail service is available at:

<https://webmail.ceng.metu.edu.tr/>

Using a desktop program (like Thunderbird) designed for e-mail is recommended, as they are more resourceful and customizable. *Mutt* is a command line e-mail client, which can be run in a terminal screen or over an ssh connection (please refer its man pages for further information).

2.6 News Groups

The department's news group server is central to courses and announcements. And it contains many news groups for discussing on a variety of topics. Please be considerate during your communication on news groups, just as you would do when doing face-to-face communication.

The following is the list of discussion groups you should subscribe and follow regularly.

- metu.ceng.course.111
- metu.ceng.announce.official
- metu.ceng.announce.admin
- metu.ceng.news

All announcements related to the course will be made through the ceng111 group. You should check this group for announcements and ask your questions about the course or laboratory to this newsgroup. The instructor and the assistants will be reading the group as well.

It is advisable to subscribe to the following groups.

- metu.ceng.student.freshman
- metu.ceng.student.courses
- metu.ceng.python
- metu.ceng.unix

2.6.1 Mozilla Thunderbird

Mozilla Thunderbird is used for news group access, in addition to e-mail access. You can set up Thunderbird for accessing department news groups by following the instructions in the web page:

<http://www.ceng.metu.edu.tr/doc/services/news/thunderbird>

2.6.2 COW News

The Department's COW System includes a web based news group reader, which allows reading news groups and writing new posts to news groups. But as it lacks lots of features found on GUI programs, you are **strongly recommended** to use Thunderbird.

Tin is a command line news client, which can be run in a terminal screen or over an ssh connection (please refer its man pages for further information).

2.7 Security: Protecting your account

There are some concerns about account and data security. In this section we will briefly discuss some security issues and their solutions.

First of all, your password is your key to your account. Guard it like you would guard your house key. Do not share your password with others, do not write it down on random papers. (A “secure” paper in your wallet can be considered OK, as long as you keep your wallet secure).

Consider the following:

- **do not** choose dictionary words, names, dates as your password
- **do not** use the same password for all your accounts
- **do** choose a password consisting of uppercase and lowercase letters, numbers and special characters.
- **do** change your password from time to time (say, at the beginning of each term)
- **do** choose a unique password for your department account, your METU account, and all your other e-mail accounts. You *may* use same password for accounts of lower security, but keeping your important passwords unique will help protecting these lower security accounts.
- **do** use mnemonics which will help you remember your password without writing it down. For example, the sentence “**our teacher breaks 34 windows yearly**” is a mnemonic for the password **oTb.34wY**

The computers are connected to each other by means of cables. The total of the computers, these cables and switch devices are called the network. The network usually is not a direct cable between your local machine and the remote machine you connect to. there can be many types of cables and many other machines connected to the same cables. So any information you send from these cables can be read by any machine connected to the same cable as seen in figure.

The most valuable information one can get from such a cable is your password. Once a person gets your password, they can do many things using your account and you will be responsible for them. To prevent this from happening you must use secure protocols and programs, ssh and secure ftp are such programs. These programs don’t send your data to the remote host as plain text. Instead they apply an encoding(encryption) algorithm that results in a –seemingly– meaningless data and send this over the line. The remote host gets this data and applies a decoding(decryption) algorithm to find out what was written.

The protocol used for browsing web pages, HTTP, has a secure version HTTPS. When accessing web sites that require logging in(including department pages), always confirm that you are on a secure connection by checking that the URL begins with “*https://*”

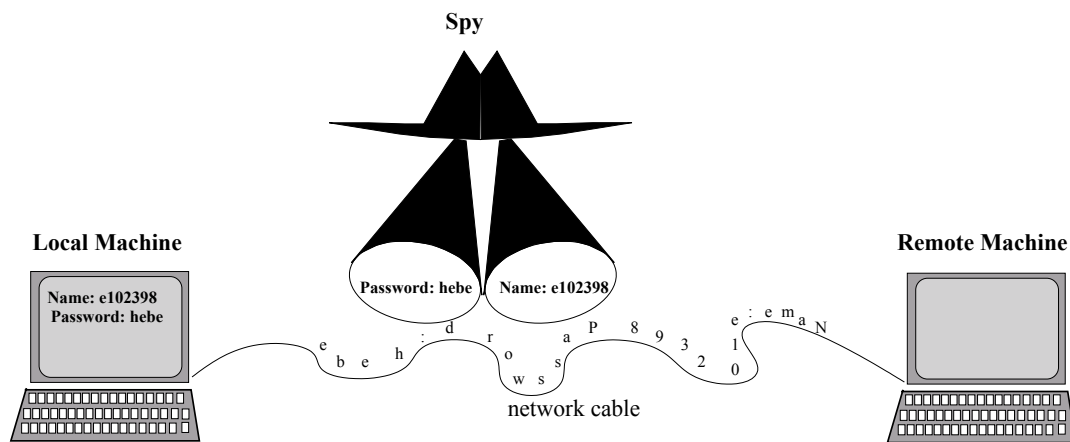


Figure 4: One possible way to find a password

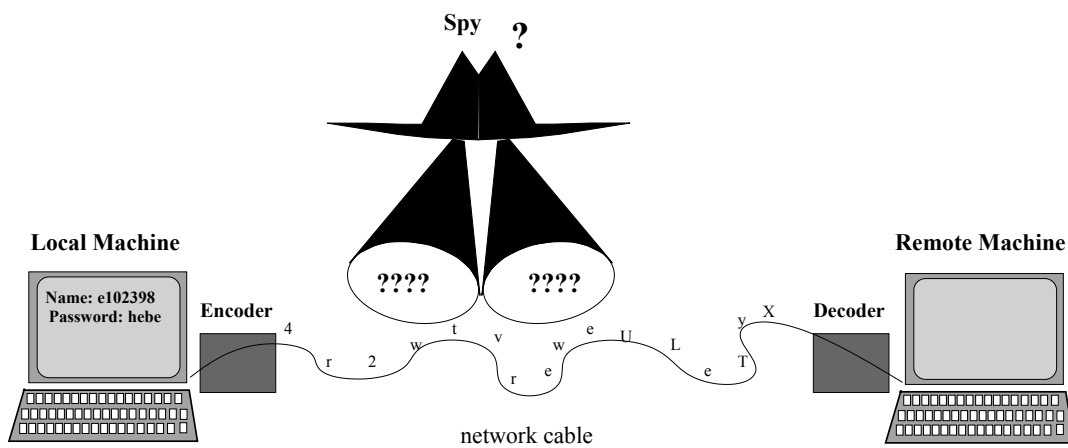


Figure 5: Your information are safe