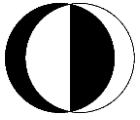




unix



MIDDLE EAST TECHNICAL UNIVERSITY  
C O M P U T E R C E N T E R



INTRODUCTION TO  
**UNIX**

VERSION 1.7

NOVEMBER 2005

**Introduction to UNIX** booklet has been prepared by User Support Group of METU-CC (Middle East Technical University - Computer Center), in order to help users for getting familiar with UNIX Operating System. Please send your comments to 'metucc@metu.edu.tr'.

**Prepared by:**

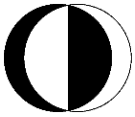
Toros GÖKKURT

**Edited and updated by:**

Cengiz ACARTÜRK

Feride ERDAL

Ömer GÖZÜ



Middle East Technical University  
Computer Center  
İnönü Bulvarı, 06531  
Ankara



Phone : +90 (312) 210 3301  
Fax : +90 (312) 210 1120  
E-mail : metucc@metu.edu.tr

## PREFACE

This document is intended to provide introductory information in order to help the new user of the computers that run a version of UNIX operating system. It should be noted that some of the examples in this document are only applicable for AIX, UNIX Operating System developed for IBM RISC computers and some are applicable for METU-CC central systems.

With this document, you will learn the basic concepts of UNIX so that you can become productive as soon as possible. Try reading a section at a time and experimenting on your account as you work your way through the document. There may be some sections for which you currently have no use; skip them. They will still be there when you need them.

### Things to remember

- UNIX operating system user interface (shell) is case-sensitive; i.e. a command in lower-case letters interpreted differently from one in capital letters. Always turn off the Caps Lock key on your terminal or microcomputer before you login and enter all commands in lower-case letters unless otherwise instructed.
- Be sure to end every command that you type with a carriage return (marked Return or Enter). The Enter key is shown as 'ENTER' in this document. The computer doesn't know you are trying to tell it to do something until it sees a carriage return, so if the computer seems to be ignoring you, check to see if you forgot to press the carriage return.

Welcome to UNIX World!

Computer Center  
Middle East Technical University  
November, 2005

**NOTE:** The fonts used in this booklet can be classified as follows:

commands to be entered to the system	: <b>Courier</b>
answers of the system	: Courier
mentioned words	: <b>bold</b>
variables	: <i>italic</i>
whole document	: Fujiyama2





## CONTENTS

<b>1. INTRODUCTION TO UNIX OPERATING SYSTEM</b> .....	<b>1</b>
1.1. What is an Operating System? .....	1
1.2. UNIX Operating System .....	1
<b>2. LOGIN, CHANGE YOUR PASSWORD AND LOGOUT</b> .....	<b>1</b>
2.1. Logging In .....	2
2.2. Changing Your Password .....	2
2.3. Logging Out .....	3
<b>3. UNIX COMMAND SYNTAX</b> .....	<b>3</b>
3.1. Command Line Structure .....	3
3.2. Commands with Options .....	4
<b>4. GETTING ON-LINE HELP</b> .....	<b>4</b>
4.1. Using 'man' .....	4
4.2. Understanding 'man' .....	4
<b>5. SHELLS</b> .....	<b>5</b>
5.1. Shell Features.....	6
5.1.1. Startup Files .....	6
5.1.2. Variables.....	6
5.1.3. Special Variables (Environment Variables) .....	7
5.1.4. Displaying Current Variables .....	7
5.2. Changing Your Environment.....	8
5.2.1. Making a Temporary Change .....	8
5.2.2. Making Permanent Changes .....	8
5.2.3. Manipulating the Command History .....	8
5.2.4. Editing a Previous Command Line .....	9
5.2.5. Some Operations on the Command History.....	9
5.2.6. Defining Command Aliases .....	10
5.2.7. Adding and Canceling a Command Alias.....	11
5.2.8. Setting the Prompt Variables .....	11
<b>6. THE FILE SYSTEM</b> .....	<b>12</b>
6.1. UNIX File and Directory Permissions.....	13
6.2. Setting File and Directory Permissions.....	14
6.3. The 'ls' Command .....	16
6.4. Wildcards and Special Characters.....	17
<b>7. CREATING AND MANAGING FILES</b> .....	<b>18</b>
7.1. File and Directory Names.....	18
7.2. Creating a File .....	18
7.3. Displaying the Contents of a File .....	18
7.4. Copying a File .....	19
7.5. Renaming and Moving a File.....	19
7.6. Sorting The Contents of a File .....	19
7.7. Searching for an Expression.....	20
7.8. Working with sed.....	20
7.9. Working with awk.....	21
7.10. Displaying the Types of the Files and Directories.....	22



7.11.	Comparing Files.....	23
7.12.	Deleting Repeated Lines .....	23
7.13.	Counting Words.....	24
7.14.	Removing a File .....	24
7.15.	Creating a Directory .....	24
7.16.	Putting Files Into a Directory .....	25
7.17.	Renaming a Directory .....	25
7.18.	Changing Your Current Directory.....	25
7.19.	Comparing Directories.....	26
7.20.	Removing a Directory .....	26
7.21.	Searching for a File .....	26
7.22.	Printing a File.....	27
7.23.	Managing File Transfer (FTP) .....	27
<b>8.</b>	<b>INPUT/OUTPUT REDIRECTION.....</b>	<b>28</b>
8.1.	Redirecting Standard Output .....	28
8.2.	Appending Output to a File.....	28
8.3.	Using Files For Standard Input .....	28
8.4.	Sending Output to Another Command.....	29
8.5.	Examples.....	29
<b>9.</b>	<b>EDITING FILES.....</b>	<b>29</b>
9.1.	vi Text Editor.....	29
9.2.	pico Text Editor .....	32
<b>10.</b>	<b>READING E-MAILS .....</b>	<b>34</b>
10.1.	Pine .....	34
10.2.	MUTT.....	36
<b>11.</b>	<b>COMPRESSING AND ARCHIVING FILES .....</b>	<b>38</b>
11.1.	Compress and Uncompress Commands .....	38
11.2.	Gzip and Gunzip Commands.....	39
11.3.	Bzip2 and Bunzip2 Commands.....	39
11.4.	Tar Command.....	40
<b>12.</b>	<b>COMPILING AND RUNNING C .....</b>	<b>41</b>
12.1.	Creating Source File .....	41
12.2.	Compiling Your Program .....	41
12.3.	Linking Your Program .....	42
12.4.	Running Your Program.....	42
12.5.	C Compilers.....	43
12.6.	Debugging Your Program .....	43
<b>13.</b>	<b>COMPILING AND RUNNING FORTRAN AND PASCAL.....</b>	<b>44</b>
13.1.	Libraries at METU-CC Central Services .....	45
13.2.	Software at METU-CC Central Services .....	46
<b>14.</b>	<b>JOB CONTROL .....</b>	<b>46</b>
14.1.	Background Jobs.....	46
14.2.	Job Control Commands.....	47
14.3.	Terminating Runaway Processes.....	47
<b>APPENDIX: SOME UTILITY COMMANDS .....</b>		<b>49</b>
<b>REFERENCES .....</b>		<b>50</b>

## 1. INTRODUCTION TO UNIX OPERATING SYSTEM

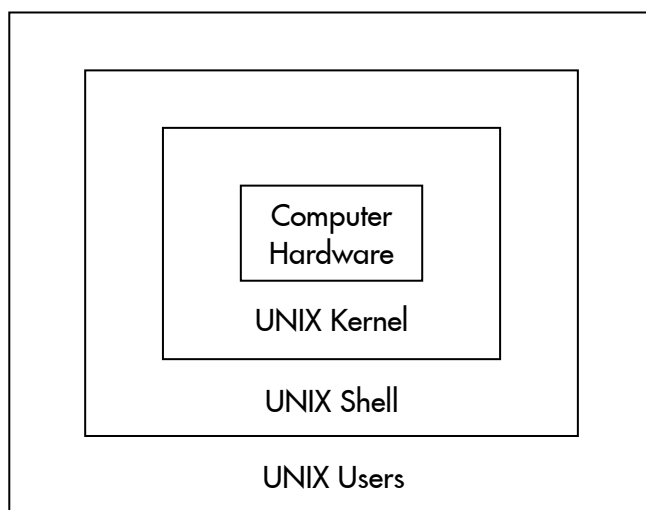
### 1.1. WHAT IS AN OPERATING SYSTEM?

An operating system is a system software that acts as an interface between the user of a computer and the computer hardware. Its primary goal is to use the computer resources and hardware in an efficient manner and its secondary goal is to make the computer convenient to use.

An operating system is alternatively defined as a 'Control Program' that coordinates and controls all the activities of a computer. If it hadn't been for operating systems, the computer would have been a futile box of gizmas and gadgets, boards and chips.

### 1.2. UNIX OPERATING SYSTEM

UNIX is a complete multi-tasking and multi-user operating system. That is, many users may login (see section 2.1) to the UNIX system and each user may run multiple programs simultaneously. Besides, UNIX is a layered operating system.



As it's seen in the figure, the **shell** is a program which reads and executes commands typed by the users. The shell acts as an interpreter between the user and the UNIX kernel. The kernel in turn, directs the hardware to handle the basic operation required by the tasks you initiate, such as reading and writing the data in disk files. In general, you never deal directly with the kernel but only with the shell (for more information about the shells, see section 5).

## 2. LOGIN, CHANGE YOUR PASSWORD AND LOGOUT

You can connect to a UNIX host from any of the following: an on-campus terminal, a microcomputer or a workstation connected to the METU campus computing network or with an attached modem; or a computer at another site using **ssh** or **telnet** (for campus-wide connections) protocols.

## 2.1. LOGGING IN

When you are connected to a UNIX host, your screen will display the name of the host and the prompt 'login:'. You must now identify yourself to the UNIX host with your username and password. This process is called **logging in**. For example, when you have connected to the UNIX host called 'beluga.cc.metu.edu.tr', you will see something like:

```
AIX Version 5
(C) Copyrights by IBM and by others 1982, 2004.
login:
```

At the login prompt, type your username and press ENTER. The example below shows the login for someone with the username **e100001**. The system will then prompt you for the password:

```
AIX Version 5
(C) Copyrights by IBM and by others 1982, 2004.
login:e100001
e100001's password:
```

Type your password and press ENTER. Your password does not appear on the screen as you type. If you mistype your username or password, system will respond with:

```
3004-007 You entered an invalid login name or password.
login:
```

If this happens retype your username and password. After the correct entry of user account and password is done, UNIX host will accept you as a logged in user.

You are now logged in to UNIX, after some system messages in which there may be some news and information about the system, you will be prompted by '\$' (according to the shell you are in) which indicates that the system is ready to accept your commands.

## 2.2. CHANGING YOUR PASSWORD

Immediately after you login for the first time you should change your password and you have to do this periodically thereafter to preserve the security of your account. You, as being a METU user, are expected to read the 'METU Information Technology Resources Use Policy' document at METU-CC web site (<http://computing-ethics.metu.edu.tr>) and observe them.

It is recommended that passwords must;

- be at least 6 characters long,
- be a mixture of upper-lower case letters and numerals,
- include some non-alphabetical characters,
- not be real words or variations of your name, not be easily guessable.

Remember that the UNIX host will only consider the first 8 characters of the passwords more than 8 characters long.

After login, you can change your password by typing the command;

### `passwd`

and pressing ENTER. When the system respond with;

```
Changing password for "e100001"
e100001's Old password:
```

type your current password and press ENTER. Then the system will prompt for your new password:

```
e100001's New password:
```

Type your new password and press ENTER. Since passwords do not appear on the screen, it is possible to mistype your password and not to be aware of it. To avoid this possibility, the system will prompt:

```
Enter the new password again:
```

Type your new password again and press ENTER. If you successfully complete the process, you will see the following system message;

```
NIS passwd changed on beluga
```

and return to the system prompt.

## 2.3. LOGGING OUT

Logging out ends your work session. You must finish each work session by logging out to protect your usercode against unauthorized use. To logout from UNIX, type;

```
exit
```

and press ENTER. Another commonly accepted way to logout is to hold down the 'Ctrl' key while pressing 'D' key (Ctrl+D).

## 3. UNIX COMMAND SYNTAX

### 3.1. COMMAND LINE STRUCTURE

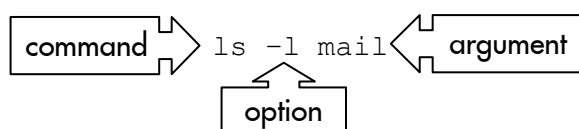
The command line consists of three parts: **command**, **options** and **arguments**. Options are preferences that define the way of how command will be executed. Arguments are the words or string of characters that commands act upon. Arguments include options and filenames. The general structure of commands includes these options or some combinations such as:

- A command by itself with no arguments
- A command with one or more options
- A command with one or more required arguments
- A command with one or more optional arguments

Sample Syntax:

- *Command*
- *Command -options argument*
- *Command -options argument1 argument2 ...*

An example may be defined as follows:



## 3.2. COMMANDS WITH OPTIONS

Many UNIX commands take **options**, which are qualifiers used to modify or expand the function and output of commands. An option tells the computer to execute a particular variation of the command. Options are usually designated with a hyphen (-) and a letter (e.g. `ls -a`). The correct syntax requires a space between the command and the hyphen. A command can have multiple options in upper and lower case, which can be combined with one hyphen (e.g. `ls -al`).

As you use options, you will see that they are command-specific. Each command has its own set of options. Thus, an option used with one command may have a different effect when used with another command.

## 4. GETTING ON-LINE HELP

While using UNIX, you can get on-line help by using `man` or `help` commands.

### 4.1. USING 'MAN'

`man` command displays information from printed UNIX reference manuals, called **manual pages**. This command displays a command's syntax plus a detailed description of the command and its options and arguments (if any). Type `man`, followed by a UNIX command you are interested in. The syntax of the command is:

```
man command
```

After running this command, you will see the man page entry for the specified command.

### 4.2. UNDERSTANDING 'MAN'

You can use `man` to learn more about the `man` command itself. After you type;

```
man man
```

and press ENTER, an information display appears on your screen. At the bottom of your screen, you will see a message like `---More---(7%)` which means you have viewed 7% of the file and 93% remains. At this point, you can do any of the following;

- step through the file a page at a time by pressing the space bar or 'f' key,
- step backwards through the file a page at a time by pressing 'b' key,
- scroll through the file a line at a time by pressing ENTER,
- search for a given keyword by pressing '/' key,
- get help by pressing 'h' key,
- quit viewing the reference page by pressing 'q' key.

The `man` command generally has three different syntaxes:

```
man -k keyword ...  
man -f file ...  
man [ - ] [ section ] title ...
```

The brackets '[ ]', indicate that the enclosed parameter is optional. Using `man` with `-k` option followed by one or more keywords causes `man` to display one-line description of each manual entry whose on-line description contains text matching one or more of the specified keywords. e.g.

```
man -k disk
```

shows the commands whose on-line descriptions include the keyword 'disk'.

## 5. SHELLS

Whenever you login to a UNIX system you are placed in a program called the **shell**. You can see its default prompt at the bottom left of your screen. To get your work done, you enter the commands at this prompt and the shell acts as a command interpreter.

There are several different shells available for UNIX systems. Some of these are:

Shell Name	Shell Command Name	Default Prompt
Bourne Shell	sh	\$
C Shell	csh	<i>hostname%</i>
TC Shell	tcsh	<i>hostname%</i>
Korn Shell	ksh	\$
Bourn Again Shell	bash	\$

Default shell of central servers at METU-CC is Bash shell. However, on your system there may be other shells available. To find out which shells are available on your system enter the command below:

```
chsh
```

The shell that you enter whenever you login to the system is called your **login shell**. You can find out which shell is your login shell by entering the command:

```
echo $SHELL
```

If <code>echo \$SHELL</code> displays...	Then your shell is...
/bin/sh	Bourne Shell
/bin/csh	C Shell
/bin/ksh	Korn Shell
/bin/bash	Bash Shell

For the remainder of your login session if you do not want to use your login shell, you can switch to another shell available on your system. To do this, enter the 'shell command name' of the shell which you want to use at the system prompt. For example, the command `csh` switches you from your login shell to the C shell.

You do not change your login shell by switching to another shell. To change your login shell permanently use `chsh` command and enter the required information. Next time when you login, you will enter the new shell.

## 5.1. SHELL FEATURES

Shells provide us some features like; creating an environment that meets our needs, manipulating the command history, editing the command line, completing the command line, defining command aliases and writing shell scripts. Some shells provide more of these features than others. It can be figured as:

FEATURES	SHELLS				
	Bourne	C	TC	Korn	Bash
command history	No	Yes	Yes	Yes	Yes
command alias	No	Yes	Yes	Yes	Yes
shell scripts	Yes	Yes	Yes	Yes	Yes
filename completion	No	No	Yes	Yes	Yes
command line editing	No	No	Yes	Yes	Yes
job control	No	Yes	Yes	Yes	Yes

### 5.1.1. Startup Files

Each shell has a startup file, which is executed every time you enter the shell. Thus, if you want a certain command performed automatically every time you enter a shell, you simply add it to its startup file. Below there is a table showing some shells and their startup files.

Shell	Startup File
C	.cshrc and .login
TC	.tcshrc or .cshrc and .login
Bourne	.profile
Korn	.profile
Bash	.profile or .bash_profile

During the login process Bourne, Korn and Bash shells read a startup file, which is called `‘.profile’`. The C shell instead executes a file called `‘.login’`. In addition, whenever you start a new copy of the C shell the file `‘.cshrc’` is executed. And the file `‘.logout’` is executed when you logout.

These startup files are located in your **home** directory (the directory where you login). You can see your home directory by `pwd` command when you login UNIX server. You can change startup files, i.e. add your own commands and create your own environment. To create your own environment, you must have some knowledge about the variables.

### 5.1.2. Variables

Each shell variable has a name associated with it and into that variable you place a value, which is the information you want to store there. To place a value in a variable you can use the following command:

```
var_name=value          for Bourne, Korn and Bash shells
set var_name=value     for C and TC shells
```

Do not place spaces on either side of the equal sign, and the value can contain spaces only if enclosed in quotation marks (`“ ”`). Consider the following examples:



```
name=Belinda
city=" Winter Park "
```

To signify the value of the variable *var* use the notation below:

```
$var
```

For example, to display the value of the variable 'name', enter the following command:

```
echo $name
```

Then, the system will respond with:

```
Belinda.
```

### 5.1.3. Special Variables (Environment Variables)

Several variables have a special meaning to shell and they are automatically set whenever you login. Values of some of these variables are stored in names which collectively are called your **user environment**. Below are some examples of environment variable names and their descriptions.

**HOME** is the pathname of your home directory when you login and to which 'cd' command changes when invoked with no argument, e.g. **HOME=/home/e100001**.

**PATH** defines the directories searched when the shell is looking for the file to execute when processing a command. A colon (:) separates the name of each directory, e.g. **PATH=/bin:/usr/bin**. The searching priority is defined from left to right.

**MAILMSG** if set to an arrival message the shell will inform you of the arrival of a new mail, e.g. **MAILMSG='YOU HAVE A MAIL'**.

**PS1** contains the shell prompt string, e.g. **PS1="Yes, Master?"**, then the shell henceforth prompts with "Yes, Master?" rather than the usual prompt.

**PS2** is the continuation prompt when a command spans more than one line.

**TERM** contains a code that indicates the model of the terminal (a screen connected to a keyboard, with just enough of a computer to know how to display things on the screen) you are using. In METU-CC central services, the login shell assigns a default value to this variable. Thus, you see on the screen **TERM=(vt100)** whenever you login.

### 5.1.4. Displaying Current Variables

You can use the **echo** command to display the value of an environment variable, e.g.

```
echo $PATH
```

displays the value of the environment variable **PATH**. To see all the environment variables, the commands below are used.

COMMAND	TO DISPLAY
<b>set</b>	value of all environment variables for Bourne, Korn and Bash shells
<b>printenv</b> or <b>env</b>	value of all environment variables for C and TC shells
<i>command</i>   <b>more</b>	information one screenful at a time

## 5.2. CHANGING YOUR ENVIRONMENT

### 5.2.1. Making a Temporary Change

To make a temporary change on the value of an environment variable, enter the command:

```
var_name=value_of_variable          for Bourne and Korn shells
setenv var_name=value_of_variable  for C and TC shells
```

This value will remain until you logout from the system or exit from the shell you are working in. To make other programs that use the variable aware of the new value, enter the command:

```
export var_name          for Bourne and Korn shells
```

As an example, the below command changes the user's default editor to 'vi' and exports this value to other programs which make use of it.

```
EDITOR=vi; export EDITOR
```

### 5.2.2. Making Permanent Changes

To make lasting changes to the value of an environment variable, follow the steps below.

1. Use an editor (pico/vi, see section 9) to open your shell's startup file and add the line:

```
var_name=value_of_variable;export var_name          for Bourne, Korn and Bash
```

or shortly:

```
export var_name=value_of_variable
setenv var_name=value_of_variable  for C and TC shells
```

2. Save the file and leave the editor.
3. To add the new value to your working environment, enter the command:

```
. .profile          for Bourne, Korn and Bash shells
source .cshrc      for C and TC shells
```

### 5.2.3. Manipulating the Command History

C and Korn shells include a command history, whereas Bourne shell does not. By this feature, a list of the commands you have typed most recently is maintained, enabling you to repeat a command without retyping it or to repeat a command after making modifications. The syntax that you use for command history is shown below.

ACTION	C,TC and Bash	Korn
List recent commands	History	history
Rerun command number 176	!176	r 176
Rerun last command starting with v	!v	r v
Access last word of the preceding command	!\$	\$_
Run the previous command	!!	r



For example, to use the last word in a preceding command as an argument in the vi command, look at the following commands. In the vi command '!\$' represents '/usr/spool/news/rec/humor'.

```
cat /usr/spool/news/rec/humor
vi !$
```

#### 5.2.4. Editing a Previous Command Line

In C shell, you can edit any previous command from the history list to create a new command. Suppose that you run the history command and choose the command numbered 23.

```
history
:
23 lpr part[1-4].ps
:
```

To change [1-4] to [5-8] use the command:

```
!23:s/[1-4]/[5-8]/
```

This command will now run the command: `lpr part[5-8].ps`

In TC and Bash shells, by pressing the arrow keys showing up (↑) and down (↓) on your keyboard, you can fetch the previous and next commands and edit these commands.

In Korn shell, the method for command line editing depends upon the value to which the environment variable EDITOR is set. This may be 'pico' or 'vi' (see section 9). To use the editing commands provided by pico editor you would add the following line to your '.profile' file.

```
EDITOR=pico
```

Command line editing can also be turned on by using a shell command. For example the command;

```
set -o vi
```

can be given on the command line or included in your '.profile' file. Whichever editor is set, command line editing is in the style of that editor. That is, you can edit your command on the command line, as if you are editing it in the editor which you set.

In vi editing mode, first you have to press ESC key once, then pressing k key fetches the previous command and pressing j key fetches the next command.

#### 5.2.5. Some Operations on the Command History

You can keep a history of the fifty most recent commands that you have used. To do this use the command:

```
set history=50                for C and TC shells
HISTSIZE=50;export HISTSIZE  for Korn and Bash shells
```

To save your command history between one login session and another, to reuse the commands from your previous login session, you need your shell's start-up files. For example the command;

```
set savehist=25                written in '.login' or '.cshrc' file
```

saves the previous 25 commands you have used to the file `'.history'` in your home directory whenever you logout. When you login next time, the contents of the file `'.history'` are placed in the history list for C and TC shells. And the command;

```
HISTSIZE=50;export HISTSIZE
```

written in `'profile'` file

saves your command history in the file `'.sh_history'` in your home directory. When you next login the contents of `'.sh_history'` are placed in the history list for the Korn shell. History file for Bash is `'.bash_history'`.

### 5.2.6. Defining Command Aliases

An alias enables you to introduce abbreviations for commands that you use often. You type the abbreviation and the shell replaces it with the original command.

To create an alias, use the command:

```
alias name_of_alias definition_of_alias
```

for C and TC shells

```
alias name_of_alias=definition_of_alias
```

for Korn and Bash

Some examples of defining command aliases are given below.

1. To create the alias `'del'` for the command `'rm -i'`, which prompts you for the confirmation that you want to remove a file before it does so, enter the command:

```
alias del 'rm -i'
```

Next time when you issue the command, the system will respond with:

```
del memo.txt
rm: remove memo.txt?
```

Make this alias a permanent feature of your environment by adding it to your shell start up file.

2. To create a command alias that consists of several commands, use the command:

```
alias what 'ps -aux | grep $USER | less'
```

This creates the alias `'what'`. Entering the command `what` will now run the command;

```
ps -aux | grep $USER | less
```

which consists of three commands linked together with pipes.

3. To refer to another command alias within an alias, use the commands:

```
alias h history
alias rev 'h | tail -10'
```

The first command assigns the alias `h` to the `history` command. The next command assigns the alias `rev` to the command `'h | tail -10'`. This takes the output from the alias `h` (the `history` command) and pipes it through the `tail` command to list the ten most recent commands in the command history.

4. To pass command arguments to the alias, use the command:

```
alias print 'lpr \\!^ -P ps5'
```

To print a file to the printer named `'ps5'` the user enters a command such as:



```
print memo.txt
```

The notation `!\^` causes the first argument to the command `alias print` to be inserted in the command at this point. The command that is carried out is:

```
lpr memo.txt -P ps5
```

Notice that; the `!` character is preceded by a `\\` to prevent it being interpreted by the shell as a history command.

### 5.2.7. Adding and Canceling a Command Alias

To make a command alias a permanent part of your working environment, first edit the shell startup file. Add the line defining the alias, save the changes and leave the editor. Enter the below command, to add the new value to your working environment.

```
. .profile          for Bourne and Korn shells
source .cshrc       for C and TC shells
```

To cancel a command alias during your current login session, use the command:

```
unalias alias_name
```

For example the command;

```
unalias print
```

cancels the command `alias print` for your current login session. You will have to remove alias definition from your shell start-up file, if you want to cancel the alias permanently.

### 5.2.8. Setting the Prompt Variables

For the Korn shell, you may set the prompt variable by following the steps below.

1. Use your editor to open your `.profile` file and add one of the lines:

```
PS1='${PWD##*/} $'    to display your current working directory in the prompt
PS1='${USER} $'      to display your username in the prompt
PS1='${HOST} $'      to display the name of the system in the prompt
```

2. Save the file and leave the editor.
3. Lastly, to add the new value to your working environment, enter the command:

```
. .profile
```

As for the TC shell, you may set the prompt variable as explained in the following steps.

1. Use your editor to open your `.cshrc` and add one of the lines:

```
set prompt="% . %"    to display your current working directory in the prompt
set prompt="%${user} %" to display your username in the prompt
set prompt="%m %"     to display the name of the system in the prompt
set prompt="%t" [ "%c" ]%" to display the time and the current working directory in the prompt
```

2. Save the file and leave the editor.

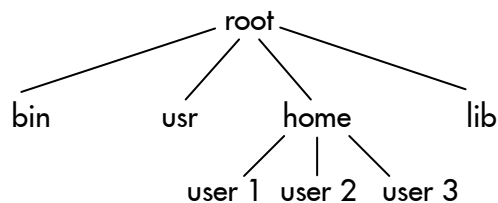
3. Lastly, to add new values to your working environment enter the command:

```
source .cshrc
```

Creating your own environment, which you are accustomed to will make your work go as well as you want.

## 6. THE FILE SYSTEM

Data, document, program source and such information are kept in a special structure called **file**. Files are classified and kept hierarchically in another structure called **directory**. These are the fundamental concepts for any computer user. UNIX uses a hierarchical (or tree-structured) directory system to store files. A sample diagram of a portion of a file tree is shown below, with the **root** directory at the top of the tree.



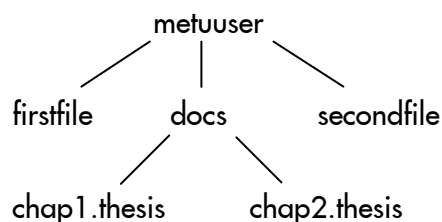
In the figure above, some of the directories can be explained as:

Directory Name	Directory Contents
/bin	UNIX utilities, such as ls, cat, cp, date etc.
/usr	additional utilities
/home	user directories
/lib	libraries for installed software

You will be assigned a directory with your username as the name when you are given an account. When you complete login process, you will be attached at a directory called as your **HOME directory**. You can create other directories and files below your HOME directory as you need them. For example, if your username is 'metuser' and assigned a HOME directory with the name /home/metuser, then you can create a directory named 'docs' and put all the chapters of your thesis in that directory. Then the full path of a file called 'chap1.thesis' would be:

```
/home/metuser/docs/chap1.thesis
```

Within your HOME directory (/home/metuser), there can be several files and subdirectories which also contain files:



It is possible that more than one person could have a file called 'chap1.thesis'. How does UNIX distinguish between files with the same name? The full name of each file includes a 'path' through the directory system to that file. The full name of two different 'chap1.thesis' files might be:

```
/home/metuser/docs/chap1.thesis
/home/testuser/thesis/chap1.thesis
```

The names of two 'chap1.thesis' files shown above are **absolute pathnames**. An absolute pathname starts with a '/' to represent the root directory, and then traces the path through various subdirectories to the file.

Another way to describe a file is by its **relative pathname**. Relative pathnames do not begin with a '/'. A relative pathname shows how to get to the file from the current working directory. If we are in the directory 'metuser', the relative pathname to our file 'chap1.thesis' is:

```
docs/chap1.thesis
```

## 6.1. UNIX FILE AND DIRECTORY PERMISSIONS

In order to see what permissions are set for all the files in a directory, use the `ls` command with `-al` option. Sample output from the command 'ls -al' is shown below, with column numbers at the top for reference with the explanation following.

1	2	3	4	5	6	7	8
-rw-r--r--	1	user1	usr	22	Aug 12	11:42	.history
-rwxr-xr-x	1	user1	usr	1024	Jun 9	16:21	News
drwxr-xr-x	4	user1	usr	1024	Apr 16	14:34	docs

**Column 1:** Ten characters giving file information; first character ('-' or 'd') is type of file; remaining nine are permissions.

The 'd' and '-' shown in the first column of above example, stands for a 'directory' and 'ordinary file' respectively.

UNIX divides the world of users into three classes;

- user (**u**) : owner (namely; you)
- group (**g**) : group to which owner belongs (namely; your group)
- others (**o**) : other users (namely; users other than you and your group)

The next nine characters show the file permissions given to user (user1) - (first three characters), the group to which user1 belongs (usr) - (next three), and the other users on the system (last three).

The permission characters are as follows:

Character	Permission
r	read permissions to view, print and copy
w	write permissions to change a file (or directory)
x	execute permission to run an executable file or 'working in' permission for a directory
-	for permission denied

If all permissions are given for a file, these entries would be:

**-rwxrwxrwx**

which gives the user, group and other read, write and execute permissions.

For an ordinary file or a directory, the other columns can be explained as the following:

**Column 2:** links; for a file, it is the number of links to that file (the value is 1 for an unlinked file); for a directory, it is the number of directories it is in (its parent and itself) plus the directories under it.

**Column 3:** owner of the file/directory.

**Column 4:** group to which owner belongs.

**Column 5:** number of bytes (characters) in the file or the number of characters reserved for a directory.

**Column 6:** last modified date.

**Column 7:** last modified time.

**Column 8:** file/directory name.

## 6.2. SETTING FILE AND DIRECTORY PERMISSIONS

As we had mentioned in the previous section, every file or directory has permissions to read, write and execute for three classes of users; owner, owner's group and others. Owner of a file or directory may change these permissions. This process is called 'Setting File Permissions'. Let us look at the following example:

file type	owner's permissions	group's permissions	others' permissions
-	r w x	r - x	- - x

Here, the owner may read the file, write to it (modify it) and execute the file. Execute access is appropriate if the file contains an executable program. Members of the group may read and execute the file, but not write to it. Other users may only execute the file, not read it or write to it.

You can set the permissions on files and directories by using the **chmod** command. The command syntax is as follows:

**chmod** *mode filename*

where:

*mode* is a string of characters stand for 'class', 'action' and 'permission' shown in the table below, *filename* is the name of the file/directory whose permissions are to be changed.

Changing file permissions can be summarized as follows:

Class	Action	Permission
u (user)	+	r (read)
g (group)	-	w (write)
o (others)		x (execute)



You can give permissions with a plus (+) sign and deny with a minus (-) sign. For example, you may want to remove read permission on a file named 'personal' from everyone except you. Use the command:

```
chmod go-r personal
```

As another example, you may want to make a command file called 'printit' readable and executable by everyone. You can use **a** instead of **ugo**, which stands for all the users, (i.e. **u**ser, **g**roup and **o**thers) to give the permission. Use the command:

```
chmod ugo+rx printit    or    chmod a+rx printit
```

Also, you might have encountered things like;

```
chmod 755 <filename>
```

Every mode has a corresponding code number (in binary mode) such that;

$$r = 2^2 = 4 \qquad w = 2^1 = 2 \qquad x = 2^0 = 1$$

Accordingly;

	Owner	Group	Others
r	4 or 0	4 or 0	4 or 0
w	2 or 0	2 or 0	2 or 0
x	1 or 0	1 or 0	1 or 0

For example:

1. `chmod 700 <filename>`

command means the owner has all permissions, group and others have no permission:

	Owner	Group	Others
r	4	0	0
w	2	0	0
x	1	0	0
SUM	7	0	0

2. `chmod 755 <filename>`

command means the owner has all permissions, group and others may read and execute the file.

	Owner	Group	Others
r	4	4	4
w	2	0	0
x	1	1	1
SUM	7	5	5



### 3. `chmod 640 <filename>`

command means the owner may read and write the file, group may only read the file and others have no permission:

	Owner	Group	Others
r	4	4	0
w	2	0	0
x	0	0	0
SUM	6	4	0

For changing permissions on specified directory and entire subdirectory and fields the following command can be used:

```
chmod -R <directoryname>
```

## 6.3. THE 'LS' COMMAND

To see a list of the contents of a directory, use the command `ls` (short for list). When you type;

```
ls
```

you will receive the list of your files, such as:

```
firstfile project1 project2
```

Some of the options for the `ls` command are as follows:

Option	Effect
<code>-a</code>	lists 'hidden' files (which begin with a period)
<code>-l</code>	lists in long format
<code>-d</code>	used with <code>-l</code> option to find the status of a directory, instead of listing contents
<code>-s</code>	gives the size of each file in kilobytes
<code>-t</code>	lists files sorted by time of most recent modification
<code>-r</code>	reserves the order of the sort to get reverse alphabetic
<code>-R</code>	recursively lists content of any subdirectories

For example, `-l` option lists directory contents in long format. If you type;

```
ls -l
```

you will get the list of your files:

```
total 4
-rwx----- 1 user users 6 Jul 4 14:22 firstfile
-rw----- 1 user users 512 Jul 3 11:56 secondfile
-rwxr--r-- 1 user users 4 Jul 8 08:30 thirdfile
drwxr--r-- 2 user users 1024 Aug 1 13:31 docs
```

You can use more than one option at a time,

```
ls -al
```

and get the list of your files, including the hidden ones (starting with a dot):



```
total 7
drwx-----  8  user  users  3072  Aug 6  11:18  .
drwxr-xr-x  40  root  sys   1024  Jul 5  12:20  ..
-rwxr--r--  1  user  users   500  Aug 7  15:45  .profile
-rwx-----  1  user  users    6  Jul 4  14:22  firstfile
-rw-----  1  user  users   512  Jul 3  11:56  secondfile
-rwxr--r--  1  user  users    4  Jul 8  08:30  thirdfile
drwxr--r--  2  user  users  1024  Aug 1  13:31  docs
```

If you enter the command `ls -R`, you will see the list of your files, including subdirectories:

```
firstfile  secondfile  thirdfile  /docs
docs:
chap1.thesis  chap2.thesis
```

## 6.4. WILDCARDS AND SPECIAL CHARACTERS

UNIX shells recognize some special characters. The most common ones are those associated with file names. When included in a file's name, these characters allow you to specify groups of files more easily. Here is a table of Wildcards and Special Characters.

The Character	What it Means
*	refers to any string of characters
?	refers to any single character
~	refers to home directory
[ ]	indicates any single characters included in the brackets

The following examples illustrate the use of these special characters.

```
ls           Lists all files in the working directory
Makefile      globals.h      queues.c       sim.h
disk.c        main.c         queues.h       test1
disk.h        main.h         schedule.c     test2
drum.c        paging.c       schedule.h     testcases
drum.h        paging.h       sim.c          usernotes

ls *.c       Lists all files ending with '.c'.
disk.c        drum.c         main.c         queues.c
schedule.c    sim.c

ls s*.c     Lists all files beginning with 's' and ending with '.c'.
schedule.c    sim.c

ls test?    Lists all files beginning with 'test', ending in any single character.
test1        test2
```



```
ls *. [ch]    Lists all the files, ending in '.c' or '.h'.
disk.c       globals.h   paging.h     schedule.h
disk.h       main.c       queues.c     sim.c
drum.c      main.h       queues.h     sim.h
drum.h      paging.c    schedule.c

ls ~jones    Lists all files in the home directory for 'jones'.
Work        mbox        bin
```

## 7. CREATING AND MANAGING FILES

The processes which can be done on files and directories are described in this section.

### 7.1. FILE AND DIRECTORY NAMES

File and directory names may be up to 256 characters long. Names may consist of almost any character, such as:

```
chapter-one  chapter,two
```

UNIX is case-sensitive. Therefore, each of the following is considered as a unique file:

```
Myfile      myfile      MyFile      MYFILE
```

### 7.2. CREATING A FILE

To create a short file to practice with, type the following at the prompt;

```
cat > firstfile
This is just a test.
```

complete the file by entering **Ctrl+D** on a line by itself. This is the simplest way to create a file, but one can also use editors called 'vi' or 'pico' (see section 9) to create a file.

You can also append new lines to an existing file , using the command:

```
cat >> firstfile
This is the second line.
```

And complete the file again by entering **Ctrl+D**.

### 7.3. DISPLAYING THE CONTENTS OF A FILE

For this purpose the **cat** command is used. The command syntax is as follows:

```
cat file_name | more
```

Here, | **more** is used to go through the file page by page.

To display the contents of the file 'document.txt' type:

```
cat document.txt
```

The **cat** command is also used to join two or more files into a single file. The syntax is then:



```
cat file1 file2 > file3
```

## 7.4. COPYING A FILE

Use the command **cp** to copy a file. The syntax of the command is:

```
cp filename1 filename2
```

For example, to create an exact copy of the file called 'firstfile', type;

```
cp firstfile secondfile
```

and press ENTER.

## 7.5. RENAMING AND MOVING A FILE

The command **mv** is used to change the name of a file or to move a file into a different directory. The syntax of the command is:

```
mv filename1 filename2
```

When you execute;

```
mv thirdfile file3
```

the result of this command is that there is no longer a file called 'thirdfile', but a new file called 'file3' containing what was previously in 'thirdfile'. If 'file3' exists previously, its content is replaced by 'thirdfile'.

## 7.6. SORTING THE CONTENTS OF A FILE

You can sort the contents of a file with the command **sort**. Syntax of this command is;

```
sort options +position1 +position2... -o outfile filename
```

Some options for 'sort' command are as follows:

Option	Effect
-o	defines the output file, if not used output is shown on the screen
-d	makes only alphabetical sorting
-n	makes arithmetical sorting
-r	reverses order of the specified sorting

Positions can be thought as index of columns of data in the file. (0) is for the first character before any breaks. Suppose that we have a file named *file1* containing the following data:

```
mars
earth
venus
jupiter
```

To sort this file in respect to alphabetical order and put the result in a file named outfile, use the following command;

```
sort -d +0 -o outfile file1
```



Now, the contents of the file `outfile` is as follows;

```
earth
jupiter
mars
venus
```

## 7.7. SEARCHING FOR AN EXPRESSION

For this purpose `grep` command is used. Usage of this command can be summarized as follows:

```
grep expression filename
```

Here, it is used to find the expression in a specified file.

This command can be used to find out in which files the expression exists:

```
grep -l expression *
or grep -l expression <path_to_directory>/*
```

Suppose we have two files named *firstfile* and *secondfile* with the following statements in them:

```
This is just a test.
It is for learning UNIX.
```

With the command;

```
grep test firstfile
```

the system will display the line containing 'test' in the file. So, system respond to your command is as follows:

```
This is just a test.
```

If you use the following command;

```
grep -l test *
```

the system will display the list of files containing the word 'test'; in our case;

```
firstfile
secondfile
```

## 7.8. WORKING WITH SED

Sed program (Stream Editor) is used for making changes on the text documents. For example for changing all the 'bootle' word(s) to the word 'glass' '`s/bottle/glass/g`' command is used, where 's' means substitute; 'g' means global.

For example:

```
cat test.txt
I drank a bottle of milk instead of a glass of milk.
```

```
sed 's/bottle/glass/g' test.txt
I drank a glass of milk instead of a glass of milk.
```



The word 'bottle' in the test.txt is substituted by the word 'glass'. Sed command does not make any change on the original file. Sed commands can also be called from another file by *-f* such that:

```
cat sed.kmt
s/bottle/glass/g

sed -f sed.kmt test.txt
I drank a glass of milk instead of a glass of milk.
```

For only displaying the lines where the word 'bottle' is used, the following command is used:

```
sed -n '/bottle/p' test.txt
```

where '*-n*' means, give no output except the search criteria (here search criteria is bottle) and '*p*' means print the search criteria.

More information can be learned by the `man sed` command.

## 7.9. WORKING WITH AWK

Awk is a programming language developed by Alfred V. Aho, Peter J. Weinberger and Brian W. Kernighan. Awk is a pattern matching program with two inputs; data file and command file. The data file contains text and default character group separator is space, but other separators may be defined on awk starting command line. Awk is *typeless*, which means, that you can put anything, number or character string into its variables.

An awk program is composed of the following parts:

```
BEGIN {
# Starting command section
}
{
# Main section
}
END{
# Termination command section
}
```

The commands each are usually on one line. There are three execution types of commands:

1. **starting commands**, first word BEGIN, which are executed only once for each input file at the beginning of the file;
2. **pattern matching commands**, each of which is executed once for each line in the data file;
3. **ending commands**, first word END, which are executed only once for each input file when end of file has been reached.

Pattern matching commands are executed in order from up to down like reading the program. Lines are read from data file one by one. The dollar character is used to address the data and fields,

\$0	entire record
\$1...\$9	first data ... ninth data

A simple awk program example:

```
awk 'BEGIN {print "Hello World"}'  
Hello World
```

For displaying the contents of a file with an awk program:

```
awk print '{print $0}' test1.txt  
My name is John  
My name is Mary
```

```
awk print '{print $4}' test1.txt  
John  
Mary
```

```
awk print '{print $0}' test2.txt  
1.2.3.4  
5.6.7.8.  
9.10.11.12
```

```
awk -F '.' '{print $2}' test2.txt  
2  
6  
10
```

where `-F` defines `.` as a field separator.

Also the number of lines in a file can be displayed by an awk program by using the `NR` variable such that:

```
awk 'BEGIN{print "Starting to read file"} {print $0} END {print  
"Number of lines readed " NR}' test2.txt  
Starting to read file  
1.2.3.4  
5.6.7.8.  
9.10.11.12  
Number of lines readed 3
```

Some other variables are:

<b>NF</b>	number of fields in the input line
<b>NR</b>	number of records (lines)
<b>FILENAME</b>	name of the input file
<b>FS</b>	input field separator
<b>RS</b>	input record (line) separator (default is newline)

## 7.10. DISPLAYING THE TYPES OF THE FILES AND DIRECTORIES

The `file` command is used to see the types of the files and directories. The command can be used as follows:

```
file *
```

output would be like this:



```
deneme:          directory
denemelink:      symbolic link to deneme2/.
myapplet.class: data or International Language text
myapplet.html:  ascii text
```

## 7.11. COMPARING FILES

You can compare files with respect to their contents as follows:

```
diff file1 file2
```

This command shows different lines in two files.

Suppose you have a file named *first1* with the following contents:

```
I am a METU student.
I can use UNIX machines.
```

and another file named *first2* with the contents as:

```
I am a METU student.
I can not use UNIX machines.
```

The command;

```
diff first1 first2
```

will make the system respond to you displaying different lines in the files, as:

```
2c2
< I can use UNIX machines.
---
> I can not use UNIX machines.
```

## 7.12. DELETING REPEATED LINES

The **uniq** command deletes repeated lines in a file and the input file must be a text file.

For deleting repeated lines in a file named *test1* and save it to a file named *test2*, enter:

```
uniq test1 test2
```

If the *test1* file contains the following lines:

```
ali
ali
ayse
fatma
fatma
mehmet
mehmet
filiz
```

then the *test2* file will contain the following lines after you run the **uniq**



```
ali
ayse
fatma
mehmet
filiz
```

### 7.13. COUNTING WORDS

You may want to count the number of lines, characters or words in a file. For this purpose you can use the command `wc` with some options as follows:

```
wc options filename
```

Option	Effect
<code>-l</code>	counts the number of lines in a file
<code>-w</code>	counts the number of words in a file
<code>-c</code>	counts the number of characters in a file

Suppose we have again a file named *first1* with the same contents on the example above at section 7.11. The following command;

```
wc -l first1
```

will make the system respond to you as;

```
2 first1
```

### 7.14. REMOVING A FILE

The `rm` command is used to remove a file. Please be careful that, you can not get back the file you removed. The syntax of the command is:

```
rm filename1 filename2 ...
```

For example, the command `'rm file3'` removes `'file3'`. You may remove more than one file at a time by giving a list of files to be removed. e.g.

```
rm firstfile secondfile
```

If you wish to be prompted when removing files, you may use the `-i` option with `rm` command. Using the `-i` option with the previous example would produce:

```
rm -i firstfile secondfile
firstfile : ?
```

Type `'y'` or `'yes'` to remove the file, type `'n'` or `'no'` to leave it. The next file in the list will then be displayed:

```
secondfile : ?
```

Type `'y'` or `'yes'` to remove the file, type `'n'` or `'no'` to leave it.

### 7.15. CREATING A DIRECTORY

Creating directories permits you to organize your files. The syntax of the command is:



```
mkdir directory_name
```

Command below, creates a directory called 'project1', where you might store files related to that particular project.

```
mkdir project1
```

Note that the new directory is created under the current directory.

## 7.16. PUTTING FILES INTO A DIRECTORY

The **mv** and **cp** commands can be used to put files into a directory. For example;

```
mv bibliography project1
```

will move the file 'bibliography' into the directory 'project1'. The file 'bibliography' does not exist any more in the current directory. As another example;

```
cp chapter1 project1
```

will put a copy of the file 'chapter1' in to the directory 'project1'. The original copy of the file 'chapter1' still exists in the current directory.

## 7.17. RENAMING A DIRECTORY

The **mv** command is also used to rename and to move directories. For example, after executing the command;

```
mv project1 project2
```

the directory called 'project1' will be given the new name of 'project2' as long as a directory called 'project2' did not previously exist. If 'project2' already exists before **mv** command is issued, then 'project1' directory will be put into the directory 'project2'.

## 7.18. CHANGING YOUR CURRENT DIRECTORY

Using the **cd** (change directory) command, you can change your current working directory. The syntax of the command is:

```
cd directory_name
```

For example the command;

```
cd projects
```

moves you into 'projects' directory. To verify that you have in fact changed your current working directory, use **pwd** (print working directory) command, which displays your current directory. For example, when you execute the command;

```
pwd
```

you will get a response like:

```
/home/users/projects
```

To move to parent directory type:

```
cd ..
```



You will be in the directory:

```
/home/users
```

If you get lost among directories, do not panic; to return to your home directory (i.e. the directory that you were in at the beginning of your session) enter the `cd` command with no arguments. Also `cd -` brings you to the previous directory.

You can also go through directories by specifying their path. For example, suppose that you have a directory called *dir1* under your home directory and under *dir1* you have a directory called *dir2*. You can go to *dir2* directly by typing:

```
cd dir1/dir2
```

## 7.19. COMPARING DIRECTORIES

You can compare two directories with respect to their contents with the following command;

```
dircmp -d dirname1 dirname2
```

With this command you can see files or directories that have different or same names.

## 7.20. REMOVING A DIRECTORY

Multiple empty directories may also be removed by using a single `rmdir` command. After executing the `ls` command, taking the list and deciding which directories will be removed, you can remove them by `rmdir` command. The syntax of the command is:

```
rmdir directory_name1 directory_name2 ...
```

To remove directories named, 'testdir1' and 'testdir2', type the command:

```
rmdir testdir1 testdir2
```

To remove all the directories and the files in a directory recursively you may use the below command syntax:

```
rm -r directory_name1 directory_name2 ...
```

## 7.21. SEARCHING FOR A FILE

The `find` command is used to search a file. For searching a file from the root directory, the command syntax is as follows:

```
find / -name file_name -print
```

To search a file from a current directory, the command syntax is as below:

```
find . -name file_name -print
```

As an example, to search a file named 'document.txt' from the current directory type:

```
find . -name document.txt -print
```

As a result you will see directories containing the searched file or nothing if there is no such a file.



## 7.22. PRINTING A FILE

One can print a text file by using **enscript -d** command . The command syntax is:

```
enscript -d printer_name file_name
```

One can print a postscript file by using the **lpr -P** command . The command syntax is:

```
lpr -P printer_name file_name
```

As an example you can print 'myfile.ps' on the printer named as 'center' as follows:

```
lpr -P center myfile.ps
```

To display a report about the printer status and the order of your print job in the queue, type:

```
lpq -P printer_name
```

To cancel a print job, enter **lpjobrm** command with *job\_id* number for your job, where *job\_id* is obtained from output of **lpq -P** command. Command syntax is:

```
lpjobrm job_id
```

At METU, you can print a postscript file on both sides of the paper by adding the number 2 next to the names of the printers:

```
lpr -P printer_name2 file_name
```

As an example you can print 'myfile.ps' on both sides of the paper, on the printer named as 'center' as follows:

```
lpr -P center2 myfile.ps
```

If you don't want to print out cover page, the following command is used:

```
lpr -h -P center2 myfile.ps
```

It is important that the file be in PostScript format (.ps). Windows generated printer files (.prn) are same as PostScript files and thus should be treated in the same way.

Detailed information about taking print out from METU-CC systems can be found at: <http://www.cc.metu.edu.tr> → Services → Laser Print-out Service

## 7.23. MANAGING FILE TRANSFER (FTP)

For downloading or uploading files one must first use the following command for connecting to the remote machine:

```
ftp remote_machine_name
```

After connecting to the remote machine, FTP operations can be done using the following commands on the command line.

```
binary file_name    To transfer the file in binary format.
```

```
get file_name        To transfer a file from the FTP site.
```



<code>put file_name</code>	To insert a file into the FTP site.
<code>mget file_name</code>	To transfer more than one files from the FTP site.
<code>mput file_name</code>	To insert more than one file into the FTP site.
<code>lcd file_name</code>	To allow you to view and change the directory on your computer.

## 8. INPUT/OUTPUT REDIRECTION

One of the features provided by UNIX operating system is the ability to redirect output from a command to a file or to another command and redirect input to a command from a file. The characters used for redirection are listed below.

Redirection Character	Used to
>	Write output from a command to a file
>>	Append output from a command to a file
<	Read input to a command from a file
	Send output from one command to another

Combining redirection characters with the files called standard input (the keyboard), standard output (the screen) and standard error (the screen), you will have the tools and symbols you need to make use of redirection.

### 8.1. REDIRECTING STANDARD OUTPUT

After running a command or a program, the output can be redirected to other files. If the file exists, its previous contents are lost. If the file does not exist, it is created. In its simplest form, the command syntax is as follows;

```
command > out_file
```

where *command* is the command whose output is redirected, and *out\_file* is the name of the file to which the command writes its standard output.

### 8.2. APPENDING OUTPUT TO A FILE

If file exists, new data is appended to the end of the file. If the file doesn't exist, it is created. The command syntax is;

```
command >> out_file
```

where *command* is the command whose output is redirected, and *out\_file* is the name of the file to which the command appends the standard output.

### 8.3. USING FILES FOR STANDARD INPUT

The shell lets you redirect the standard input of a process so that input is read from a file instead of keyboard. In simplest form the command syntax is as follows;

```
command < input_file
```



where *command* is the command whose input is redirected and *input\_file* is the name of the file from which the process reads standard input.

## 8.4. SENDING OUTPUT TO ANOTHER COMMAND

You can direct the result of a command, to be used by another command. That is, output of first command will be input for the second command. This process can be done by putting a pipe character ( | ) between the commands.

## 8.5. EXAMPLES

Study examples below to get some idea about Input/Output Redirection.

1. To send a letter called 'welcome.letter' to 'user1@virginia.edu' by e-mail:

```
mail user1@virginia.edu < welcome.letter
```

2. To send all the Fortran compiler error messages to a file called 'errors' rather than having them displayed on screen:

```
xlf myprog.f > errors
```

3. To send output from `ls` command to file 'ls.output' rather than to screen:

```
ls > ls.output
```

4. To join 'file1' and 'file2', then to send output to sort command and to put the sorted output into the file 'sorted.big.file':

```
cat file1 file2 | sort > sorted.big.file
```

5. To append the sorted output of the file 'file3' to the file 'sorted.big.file':

```
sort file3 >> sorted.big.file
```

## 9. EDITING FILES

There are many text editors on UNIX systems. `vi` and `pico` are two major text editors. `vi` is a standard UNIX editor, but `pico` is not. Usage of `pico` is more simpler than `vi`. On the other hand, `vi` has a wide usage in UNIX systems. At this point, it is better to learn using of both system editors.

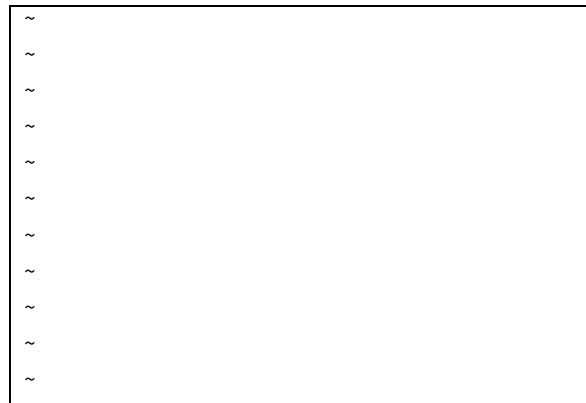
### 9.1. VI TEXT EDITOR

`vi` editor is available on almost all UNIX systems. `vi` can be used from any type of terminal because it does not depend on arrow and function keys, which might be undefined for UNIX system you are using. It uses the standard alphabetic keys for commands. `vi` is the abbreviation of 'visual editor'.

You may use `vi` by typing;

```
vi
```

and pressing ENTER. You will get an empty screen which is shown below.



To open an existing file or a new file, type;

```
vi filename
```

where *filename* is the name of a new file or an existing file. If *filename* exists, then it is opened. Otherwise you may create a new file, named as *filename*. For example, to create a new file type;

```
vi newname
```

where 'newname' is the name you wish to give the new file.

vi has two modes; **command mode** and **insert mode**. In command mode, the letters you type perform editing functions (commands to vi) while in insert mode they form the content of the file.

The mode you are in when you enter vi is the command mode. In order to begin entering text, you must change from command mode to insert mode. To do this, type letter i. You can also turn back to command mode by pressing ESC button.

While you are editing in vi, you can move the cursor to another position in insert mode. If you have finished typing text, go back to command mode by pressing ESC. The cursor is controlled with 4 keys; h, j, k, l. The functions of these keys are given in the following table.

Key	Cursor Movement
h	left one character
j	down one line
k	up one line
l	right one character

In **Command Mode**, the commands that you can use are listed below.

**Deleting Characters (x,X):** To delete one character, move cursor until it is on the letter to be deleted, then press x on keyboard. The character under the cursor disappears. To remove 4 characters (the one under the cursor and the next 3) type 4x. To delete the characters before the cursor, type X (uppercase x).

**Deleting Words (dw):** To delete a word, move the cursor to the first letter of the word, and type dw. The command deletes the word and the space following it. To delete 3 words, type 3dw.



**Deleting Lines (dd):** To delete a whole line, type **dd**. The cursor does not have to be at the beginning of the line. **dd** deletes the entire line containing the cursor, and places the cursor at the start of the next line. **2dd** deletes 2 lines.

**Moving Lines (dd and p):** To move a whole line, type **dd** first. Then, take your cursor to the place where you want to get the line and type **p**. To cut and paste 5 lines type **5dd** first.

**Copying Lines (yy and p):** To copy a whole line, type **yy** first. Then, take your cursor to the place where you want to copy the line and type **p**. To copy 6 lines type **6yy** first.

**Replacing Character (r):** To replace one character with another move the cursor to the character to be replaced. Type **r**, then the replacement character. Also, you may type **R** to enter the insert mode and type over the characters on the screen.

**Replacing Words (cw):** To replace one word with another, move to the start of the word to be replaced and type **cw**. The last letter of the word to be replaced will turn into a '\$'. You are now in **insert mode** and may type the replacement. The new text does not need to be the same length as the original. Press **ESC** to get back to command mode. **3cw** allows you to replace 3 words.

**Inserting Text (i):** To insert text, position the cursor where the new text should go, and type **i**. The mode will be changed to insert mode. Enter the new text. The text is inserted before the cursor. Press **ESC** to get back to command mode.

**Appending Text (a,A):** To add text to the end of a line, type **A**. You will see that cursor will move to the end of the line and mode will be changed to insert. After typing the new text press **ESC** to go back to command mode. Similar process with letter **a** (lowercase) appends text after the cursor.

**Opening a Blank Line (o,O):** Type **o** (lowercase) to insert a blank line below the current line. Type **O** (uppercase) to insert a blank line above the current line.

**Joining Lines (J):** The command **J** (uppercase) lets you join two lines together. Put the cursor on the first line to be joined and type **J**. **3J** lets you join 3 lines.

**Replacing Lines (C):** To change text from the cursor position to the end of the line, type **C** (uppercase) and enter the replacement text .

**Undoing (u,U):** You may undo your most recent edit by typing **u** (lowercase). **U** (uppercase) undoes all editing on a single line.

**Moving Around In a File:** There are shortcuts to move more quickly through a file. All these work in command mode.

**Moving by searching (/):** In command mode, type a slash (/) followed by the text to search for. Question mark (?) is used for moving backward by searching. To repeat the search in the same direction type **n** and for the opposite direction type **N**.

**Closing and saving a file:** To save the file and quit 'vi' type **ZZ** .

**Some Useful Commands:**

Keys	Movement
W	forward word by word
B	backward word by word
\$	to end of line
O (zero)	to beginning of line
H	to top line of screen
M	to middle line of screen
L	to last line of screen
1G	to first line of file
G	to last line of file
Ctrl+F	scroll forward one screen
Ctrl+B	scroll backward one screen
Ctrl+D	scroll down one-half screen
Ctrl+U	scroll up one-half screen

In **Command Mode**, to save the file and quit 'vi' type :wq. To save the file, type :w and to quit vi and discard any changes you have made since last saving, type :q!.

**Some Useful Commands:**

Command	Effect
.	repeat last issued command
^	go to beginning of line
~	switch between upper/lower case
Ctrl+L	refresh screen
Ctrl+G	report the cursor position
Ctrl+V	insert a control character into text
:set all	see the setup of 'vi'
:r! <i>command</i>	write the output of the command in the editor
:%s/word_1/word_2/g	replaces word_1 with word_2 in the whole document
:shell	go to the shell
:! <i>command</i>	execute specified command
:w! <i>filename</i>	save as specified file
:r <i>filename</i>	read from specified file

## 9.2. PICO TEXT EDITOR

pico is an easy to use text editor. The picture below shows the empty screen of pico.

```

UW PICO(tm) 2.3                               New Buffer
                                                                
                                                                
                                                                
                                                                
                                                                
                                                                
                                                                
                                                                
                                                                
                                                                
                                                                
^G Get Help  ^O Write Out  ^R Read File  ^Y Prev Pg  ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify    ^W Where is  ^V Next Pg  ^U UnCut Text ^T To Spell
  
```

**First line at the top:** pico's version, current file edited and whether or not there are outstanding modifications that have not been saved.

**Bottom two lines:** Lists the available editing commands.

Each typed characters are automatically inserted into the buffer at the position.

**Available pico functions:**

Key	Definition
Ctrl+G (F1)	display help
Ctrl+F	move forward a character
Ctrl+B	move backward a character
Ctrl+P	move to the previous line
Ctrl+N	move to the next line
Ctrl+A	move to the beginning of the current line
Ctrl+E	move to the end of the current line
Ctrl+V (F8)	move forward a page of text
Ctrl+Y (F7)	move backward a page of text
Ctrl+W (F6)	search for (where is) text, neglecting case
Ctrl+L	refresh the display
Ctrl+D	delete the character at the cursor position
Ctrl+K (F9)	delete (kill) the entire line at the cursor position
Ctrl+U (F10)	undelete last deleted line(s) at cursor position
Ctrl+l	insert a tab at the current cursor position

Key	Definition
Ctrl+J (F4)	format (justify) the current paragraph
Ctrl+T (F12)	to invoke the spelling checker
Ctrl+C (F11)	report the cursor position
Ctrl+R (F5)	insert an external file at the current cursor position
Ctrl+O (F3)	output the current buffer to a file, saving it
Ctrl+X (F2)	exit pico, saving buffer

Function keys may be undefined for UNIX system you are using. But, you can also use cursor keys instead of function keys.

## 10. READING E-MAILS

### 10.1. PINE

Pine program can be used for reading incoming e-mails and for sending e-mails to other people. Enter the command;

```
pine
```

to run the program. After issuing this command, the screen of Pine's main menu will appear.

```

PINE 4.58      MAIN MENU                               Folder: INBOX 60 Messages

?  HELP                -  Get help using Pine
C  COMPOSE MESSAGE    -  Compose and send a message
I  MESSAGE INDEX      -  View messages in current folder
L  FOLDER LIST        -  Select a folder to view
A  ADDRESS BOOK       -  Update address book
S  SETUP              -  Configure Pine Options
Q  QUIT               -  Leave the Pine program

Copyright 1989-2003.  PINE is a trademark of the University of Washington.

? Help                P PrevCmd                R RelNotes
O OTHER CMDS         > [ListFldrs]          N NextCmd                K KBlock
  
```

In order to select one of the choices, you can highlight the row by using up and down arrow and then press the 'Enter' key or the key which is shown as a character in the left of the row.

In this menu, the following operations can be made:

Key	Definition
?	gives information about Pine
C	for sending a message
I	for reading the incoming mails
L	for selecting folders
A	for recording e-mail addresses of people
S	for configuring Pine program
Q	quits from Pine

At the bottom of the screen, you can see shortcuts of commands that you can use.

You can also use some additional commands by pressing 'O'. Those commands can be listed as:

Key	Definition
>	lists folders
P - N	moving between folders shown in this menu for previous (P) and next (N)
R	for getting release notes for the Pine system
K	locks keyboard setting a password

By the help of the 'Address Book' option in the 'Main Menu', you can record the e-mail addresses of the people. To record a data, nickname field, full name field and e-mail address field should be filled. After filling in these fields, you can compose a message and write the 'nickname' which is supplied before instead of the e-mail address to the 'To' field of the message.

Pine can also be configured by selecting 'Setup' from 'Main Menu'. For example; you can setup printer and a new password, configure the files which pine uses, etc.

In the following screen, as an ordinary e-mail program, next to the 'To' field, the e-mail address of the person that the e-mail will be sent should be written. If a copy of the mail is going to be sent to someone else, e-mail address of that person is also written to 'Cc' field. After writing the subject of the mail to 'Subject' field, press 'Enter' or the down arrow is used to reach the 'Message Text' area. Write your message below the 'Message Text' sign.

```

PINE 4.58      COMPOSE MESSAGE      Folder: INBOX  60 Messages
To :
Cc :
Atchmnt:
Subject :
----- Message Text -----

^G Get Help  ^X Send      ^R Rich Hdr  ^Y PrvPg/Top ^K Cut Line  ^O Postpone
^C Cancel    ^D Del Char  ^J Attach    ^V NxtPg/End ^U UnDel Line ^T To AddrBk
  
```

Again, there are some useful commands below the screen that helps to remember what to do while sending an e-mail:

Key	Definition
Ctrl+C	for giving up writing (then confirming with 'y' for cancelling)
Ctrl+R	for inserting a file
Ctrl+T	for inserting a file which the name or the place of it can not be remembered
Ctrl+X	sends the e-mail
Ctrl+J	for attaching a file
Ctrl+O	for composing a message

The incoming mails are stored in the 'INBOX' folder and the mails which are sent are placed in the 'sent-mail folder'. If you wish to save an e-mail, press 'S' to send that e-mail to another folder.

While using the Pine program, always help keys which are placed at the bottom of the screen must be referred. When you finish your work with Pine, press 'Q' to quit. By this way you return to UNIX prompt.

## 10.2. MUTT

Mutt is a small but very powerful text based program for reading e-mails under UNIX operating systems. Enter the following command;

```
mutt
```

to run the program. After issuing this command, if you are using Mutt for the first time, program asks:

```
/home501/user_name/Mail does not exist. Create it? ([yes]/no):
```

You may answer 'yes' by pressing the 'Enter'. Then the following screen will appear:

```

q:Quit    d:Del    u:Undel   s:Save    m:Mail    r:Reply    g:Group    ?:Help
1   Jul 06  Uzman Celal Dik ( 37) [HOTLINE:21745] Öğrenci işleri Programı
2   Jul 18  fatihkockesen K ( 40) [HOTLINE:22099] Yardım edebilirmisiniz?
3   Jul 19  h kaya          (  6) ACCESS ile ilgili soruya devam
4   Jul 21  Safak Kaya      ( 19) ftp.cc.metu.edu.tr
5   Jul 21  Servet Guney    ( 34) [HOTLINE:22202] sensoy Yedek Talebi Istem
6   Jul 21  Servet Guney    ( 26) [HOTLINE:22210] e119601 Yedek Talebi Iste
7   Jul 21  Feyza Taskazan  ( 23) [HOTLINE:22214] e129669 Yedek Talebi Iste
8   Jul 21  Mail System Int (  4) DON'T DELETE THIS MESSAGE -- FOLDER INTER

---Mutt: /usr/spool/mail/hot-line [Msgs:8 22K]---(date/date)----- (all) ---
  
```

Some commands for using Mutt are given above and below the screen.

Key	Definition
Q	quit
D	delete the current message
U	Undelete the message
S	save the message
M	compose amessage
R	reply to sender
G	reply to all recipients
?	help

In addition to 'From' and 'Subject' fields, a short summary of the disposition of each message is printed beside the message number. These are called as 'flags' which some of them mean:

Key	Definition
D	message is marked for deletion
D	message have attachments marked for deletion
N	message is new
O	message is old
R	message is replied before
+	message is to you only
T	message is to you but also to or cc'ed to others
C	message is cc'ed to you
F	message is from you
L	message is sent to subscribed mailing list

When a message is selected by pressing 'Enter', the content of the message can be seen. The following commands can be used at that screen for making the described operations below.

Key	Definition
i	exit the message
-	go back to the previous page
<space>	display the next page
v	view attachments
d	delete the current message
r	reply to sender
j	go to the next message
?	help

For composing a message, press 'm'. Mutt will then enter the compose menu and prompt you for the recipients to place on the 'To:' header field. Next, you should write the 'Subject:' of the message. Mutt will then automatically start the 'vi' editor on the message body. If a signature is set, it will be at the top of the message in the editor. When you finish editing the body of your mail message, Mutt returns to the compose menu. The following options are available:

Key	Definition	Key	Definition
y	send the message	b	edit the 'Bcc' field
q	quit (abort) sending the message	f	edit the 'Fcc' field
t	edit the 'To' field	p	postpone this message
C	edit the 'Cc' field	a	attach a file
s	edit the 'Subject' field	d	edit the description on the attachment
r	edit the 'Reply-To' field	?	help

## 11. COMPRESSING AND ARCHIVING FILES

Files, especially which take up large disk space, may be compressed to save disk space. Compression means reducing the size of files using some kind of algorithm. To use a file in the future, you can set aside it in a compressed form. Also, copying the compressed form of the file into another disk media may enable you to bring it back if the original file is damaged or lost. For this purpose there are some commonly used commands like **compress**, **gzip**, **gunzip** and **tar**. Below, it is explained only the general common usage of these commands. To get more information about these commands enter:

```
man command_name
```

### 11.1. COMPRESS AND UNCOMPRESS COMMANDS

One way to compress a file is to use **compress** command. The format of the command is:

```
compress options file_name
```

Each original file specified by the *file\_name* variable is removed and replaced by a compressed file with a *.z* appended to its name. The newly created file is the compressed version of the original one. You can see the saved disk space by looking the difference of the sizes between two files.

**uncompress** command restores original files that were compressed by the **compress** command. Format of this command is:

```
uncompress options file_name
```

The expanded file has the same name as the compressed version but without *.z* extension. Some of the options used with **compress** and **uncompress** are listed below.

Option	Meaning
<b>-d</b>	makes the <b>compress</b> command function exactly like the <b>uncompress</b> command (can be specified only with the <b>compress</b> command)
<b>-f</b>	forces compression/expansion of the file specified by the <i>file_name</i> variable (user is not prompted that an existing file will be overwritten)
<b>-c</b>	makes the <b>compress</b> or <b>uncompress</b> command write to the standard output; no files are changed
<b>-v</b>	writes each input file's percentage of compression to standard error

## 11.2. GZIP AND GUNZIP COMMANDS

Another way of compressing files is to use **gzip** command. The syntax of this command is:

```
gzip file_name
```

The default extension of the file specified with the *file\_name* is *.gz* for UNIX, and *.z* for MSDOS and OS/2. You can restore the original form of a file having extension *.gz*, by using **-d** option with the **gzip** command. The command syntax is shown below:

```
gzip -d file_name.gz
```

You can expand more than one file which are compressed with **gzip** at a time by using the command below:

```
gunzip *.gz
```

Also, you can compress a directory with all files inside it, by the command:

```
gzip -r directory_name
```

To see the contents of a file, compressed with **gzip** without expanding it, use the command:

```
gunzip -c file_name.gz
```

## 11.3. BZIP2 AND BUNZIP2 COMMANDS

Another way of compressing files is to use **bzip2** command. The syntax of this command is:

```
bzip2 file_name
```



The default extension of the file specified with the *file\_name* is **.bz2** for UNIX. You can restore the original form of a file having extension **.bz2**, by using **-d** option with the **bzip2** command. The command syntax is shown below:

```
bzip2 -d file_name.bz2
```

Also, you can expand more than one file which are compressed with **bzip2** at a time by using the command below:

```
bunzip2 *.bz2
```

Additionally, there exist **zip** and **unzip** commands for compressing files also. Detailed information can be reached by the **man** command.

## 11.4. TAR COMMAND

**tar** is short for 'tape archive' and is used to make backup of UNIX files. Because it can also be used to bundle many smaller files into one large file, it is frequently used to transfer programs from one UNIX machine to another. You can recognize these files because they have **.tar** extension at the end. Command syntax is:

```
tar options file_name(s)
```

Options may be:

Option	Meaning
<b>-c</b>	Creates a new archive and writes the file parameter at the beginning of the archive
<b>-f <i>Archive</i></b>	Uses the <i>Archive</i> parameter as the archive to be read or written
<b>-x</b>	Extracts the file or files specified by the File parameter from the archive
<b>-v</b>	Lists the name of each file as it is processed. With the <b>-t</b> flag, <b>-v</b> gives more information
<b>-t</b>	Lists the files in the order in which they appear in the archive. i.e. shows what's in an archive file like a table of contents.
<b>-w</b>	Displays the action to be taken, followed by the <i>file_name</i> , and then waits for user confirmation

### Examples:

Suppose we have files named as **file1** and **file2**.

1. To create an archive file named 'files.tar' which will include 'file1' and 'file2':

```
tar -cvf files.tar file1 file2
```

2. To compress this archive file, files.tar:

```
gzip files.tar
```

Now let's delete **file1** and **file2** with **rm file1 file2**



3. To expand the compressed archive file named files.tar.gz :

```
gzip -d files.tar.gz
```

4. To extract all files from the archive file 'files.tar':

```
tar -xvf files.tar
```

Now if you use the command `ls`, you will see file1 and file2 (previously deleted ones) still exist in your directory, since you archived, compressed before deleting and then extracted them, in a way you backed up them.

## 12. COMPILING AND RUNNING C

This part of the document introduces you to the IBM AIX XL C compiler running on RS/6000 UNIX machines. The steps to create, compile, and run a C program are discussed in the following paragraphs.

### 12.1. CREATING SOURCE FILE

A source file is the file that contains your C program. Use an editor, e.g. pico or vi, to type your program into the source file or to edit it. The filename normally has file extension .c or .i.

### 12.2. COMPILING YOUR PROGRAM

Commands listed below invoke versions of XL C compiler, which then translates C source code statements into a single executable file called 'a.out' by default. -o flag may be used to rename the resulting object file.

Using the generic version of the C compiler, which references ANSI C, you may compile a source program and/or subprograms by typing the following command;

```
xlc cmd_line_opts input_file ...
```

where:

<i>cmd_line_opts</i>	refers to compiler options,
<i>input_file</i>	source file (.c), object files (.o), or assembler files (.s).

Instead of `xlc` command, `c++` or `gcc` commands may also be used.

For example, to compile a C program whose source is in source file 'prog.c' you would enter the following command:

```
xlc prog.c
```

After `xlc` command completed, you will see a new executable file named 'a.out' in your directory.

The command `xlc` without any options and arguments gives you detailed information about the syntax of the command. The following examples illustrate the syntax required for some of more commonly used options:



1. To compile with optimization:

```
xlc -O prog.c
```

2. To produce a listing file 'prog.lst,' and 'a.out':

```
xlc -qsource prog.c
```

3. To produce an executable file named 'runprog' instead of 'a.out':

```
xlc -o runprog prog.c
```

4. To compile the file named 'myprg.c' including C programming language source code into an executable file named 'a.out':

```
xlc myprg.c
```

5. To compile multiple C programs together by a single command:

```
xlc prog1.c prog2.c prog3.c -o allprog
```

6. To state any additional libraries for the C code to be compiled, the -l parameter is used and to add mathematics libraries to the executable code -lm option is used:

```
xlc mymathprg.c -o mymathprg -lm
```

7. To state the places of any additional library files the -L parameter is used:

```
xlc mylibprg.c -o mylibprg -lm -L/usr/lib
```

8. To state the places of any additional included files (files that have extension .h) -I parameter is used:

```
xlc myprog.c -o myprog -I/usr/include
```

### 12.3. LINKING YOUR PROGRAM

If you specify `-c` as a compiler option, XL C only compiles source code, producing an object file whose default name is that of the code with a `.o` extension. To run the code, in other words, to create an executable file you must invoke the linkage editor phase. Either invoke the linker using the `ld` command or issue the `xlc` command a second time without the `-c` option, using the desired object (`.o`) filenames. For example, you may compile a subprogram 'second.c' and obtain an object file 'second.o' by typing:

```
xlc -c second.c
```

To link 'second.o' and the main 'prog.c' together and have an executable file 'a.out' type the following command:

```
xlc prog.c second.o
```

### 12.4. RUNNING YOUR PROGRAM

After your program is compiled and linked without any error, it is ready to run. You can run your program with the executable 'data.out' file as follows:

```
data.out in_file out_file
```

where:

*in\_file* is the input filename,  
*out\_file* is the output filename.

This process can also be used for an executable 'a.out' file.

## 12.5. C COMPILERS

Several commands have been aliased on AIX to invoke the compiler in various ways, to accommodate various flavors of C in which programs are written. Commands are as follows:

- xlc ANSI C compiler
- cc Kernighan and Ritchie C compiler
- c89ANSI C compiler
- gcc GNU C compiler
- c++GNU C++ compiler

Additional options, as outlined above, may be used with these commands. You are advised not to use the optimization option when debugging, but instead to use the basic xlc command until the program compiles and runs.

## 12.6. DEBUGGING YOUR PROGRAM

To debug your program, you may use **dbx** command (one can also use **xde** command which is a graphical debugger based on dbx). This command provides you an environment to debug and run programs under AIX system, allowing you to carry out operations such as;

- Examine object and core files.
- Provide a controlled environment for running a program.
- Set breakpoints at selected statements or run the program one line at a time.
- Debug using symbolic variables and display them in their correct format.

The command syntax is:

```
dbx options object_file core_file
```

Options may be:

- g Unless -g is specified, symbolic capabilities of the dbx command are limited.
- r Runs the object file immediately. If it terminates successfully, **dbx** debug program is exited. Otherwise, the debug program is entered and the reason for termination is reported.

Also, there are some subcommands used after **dbx** command is issued. Some of these are:

COMMAND	DEFINITION
<b>Run</b>	begin execution of the program
<b>print exp</b>	print the value of the expression
<b>Where</b>	print currently active procedures
<b>stop at line</b>	suspend execution at the line



COMMAND	DEFINITION
<code>stop in proc</code>	suspend execution when <proc> is called
<code>Cont</code>	continue execution
<code>Step</code>	single step one line
<code>Next</code>	step to next line (skip over calls)
<code>trace line#</code>	trace execution of the line
<code>trace proc</code>	trace calls to the procedure
<code>trace var</code>	trace changes to the variable
<code>Status</code>	print trace/stop's in effect
<code>delete number</code>	remove trace or stop of given number
<code>Screen</code>	switch dbx to another virtual terminal
<code>Call proc</code>	call a procedure in program
<code>Whatis name</code>	print the declaration of the name
<code>List line,line</code>	list source lines
<code>Quit</code>	exit dbx (program terminated)
<code>Detach</code>	exit dbx without terminating program

For example; to run the dbx environment, enter the command:

```
dbx
```

Then the system will ask you the name of the program which you want to run. If you do not specify a name while compiling, you have to enter 'a.out'. Be careful that you have to compile your program with '-g' option, e.g. `xlf -g myprog.f`. Then the system will prompt by:

```
(dbx)
```

Here you can write any command you want to perform. To do this, refer to the table given above. For example, to run your program, enter the command:

```
run
```

Tracing your program will help to see errors easily.

### 13. COMPILING AND RUNNING FORTRAN AND PASCAL

Similar to C, several commands have been aliased for FORTRAN to invoke the compiler in various ways, but for Pascal there exist one command. Compiler names for FORTRAN and Pascal are given in the table below.

Compiler Names	Language
xlf f77 xlf90	Fortran
Xlp	Pascal



All we said about C in the previous paragraphs are applicable to both of FORTRAN and Pascal. Refer to section 12.2 and section 12.4 to see how to compile and run a FORTRAN and Pascal program.

In FORTRAN, to create an executable file named 'myexe' other than 'a.out', following command line should be used:

```
xlf myprg.f -o myexe
```

Multiple FORTRAN programs including main and subprograms can be compiled and linked together by a single command entry such as:

```
xlf prog1.f prog2.f prog3.f -o allprog
```

Here in the above example, 'allprog' is the name of the executable program.

In Pascal, to create an executable file named 'myexe' other than 'a.out', following command line should be used:

```
xlp myprg.pas -o myexe
```

Multiple Pascal programs including main and subprograms can be compiled and linked together by a single command entry such as:

```
xlp prog1.pas prog2.pas prog3.pas -o allprog
```

### 13.1. LIBRARIES AT METU-CC CENTRAL SERVICES

There exist some FORTRAN libraries such as **NAG**, **ESSL** and **LAPACK** on our systems. NAG and LAPACK are under the directory '/academic'. ESSL is under the directory 'usr/lib'.

The NAG FORTRAN Library is a comprehensive collection of FORTRAN 77 routines for the solution of numerical and statistical problems. It contains all the Basic Linear Algebra Subprograms, BLAS, needed to solve linear equations, eigenvalue problems, to perform matrix operations etc., and it also contains subprograms to minimize or maximize a function, to perform complex arithmetic, and simple calculations on statistical data etc.

To state any additional libraries, e.g. from NAG package, for FORTRAN code to be compiled, -l parameter is used. e.g.

```
xlf -L/academic/nag my90prg.f -o my90prg -lnag
```

ESSL provides a variety of mathematical functions for many different types of scientific and engineering applications. ESSL area of computation contains linear algebra subprograms, matrix operations, linear algebraic equations, eigensystem analysis, signal processing computations, sorting and searching, interpolation, numerical quadrature, random number generation, computations in a parallel processing environment, and general purpose functions.

LAPACK is also a library of FORTRAN 77 subroutines for solving the most commonly occurring problems in numerical linear algebra.

## 13.2. SOFTWARE AT METU-CC CENTRAL SERVICES

The Computer Center acquires licensed software for campus-wide use based on user needs and demands. Depending on needs and demands, our users use the latest versions of every software licensed.

Information about available licensed software and their usage can be reached from the link <http://software.cc.metu.edu.tr> . As an example of running software on central servers SPSS is explained below where SPSS is a statistical analysis program that produces descriptive statistics, complex statistical analysis, tabulated reports, and plots of distribution and trends.

To start an SPSS session in a **text based environment**, one of the following commands on the command line of server systems (such as beluga, narwhal etc.) should be entered.

```
spss +m      (for text-based environment for running the character based manager.)
spss -m      (for text-based environment for prompted or batch execution.)
```

For running SPSS in an **X-Windows environment**, you must first run an X-session with a software like X-WinPro or Cygwin/Xfree86, that will turn the PC into an X-Window client.

Then, the following command should be entered to the command line;

```
export DISPLAY=IP_NO:0.0
```

(You should enter the IP number of your PC on the IP\_NO section of this command.)

Having performed these steps, you can now be able to run the software. For not writing the same command each time logging on to the central servers, you must create a file named “.profile” on the server system and enter that command into that file.

After this process you must enter the following command for running SPSS:

```
spss
```

## 14. JOB CONTROL

### 14.1. BACKGROUND JOBS

To run any UNIX command in background so that you can continue other work at your terminal, append an ampersand (&) to the command. You will immediately be returned to system prompt so that you can issue other commands. You may see the status of your background jobs, by entering the command: `jobs`.

If you log out before the process has finished, it will be killed by the system. To avoid this, precede the command with `nohup` (no hang-up). The syntax for the complete command is:

```
nohup command arguments&
```

If your command has a pipe ( | ) in it, issuing the above command will not work properly. In order to avoid the problem, put the command into a file, make the file executable using `chmod`, then run it by typing the name of the file.

To lower the priority of your background jobs, use the `nice` command as described in the table in section 14.2. You can also specify a time to run your job using `at` command. See `man` pages for `nice` and `at` for more information and usage examples.

On AIX operating system, to suspend a job running in the background, enter `CTRL+Z`. To resume the suspended job, enter `bg` command. You will then get the system prompt so that you can do other work on the computer. Similarly, if one of your background jobs seems to be running excessively and you suspect it has gone away, issue the `jobs` command to get the job number, then regain control of it by putting it into the foreground with `fg %job_number`. Then you can terminate the job with `CTRL+C` if you want.

## 14.2. JOB CONTROL COMMANDS

Table below shows useful commands for job control. Most of them can be used on any UNIX systems.

Job Control Commands	Result
<code>at time [command_file]</code>	run <i>command_file</i> at specified time
<code>atq</code>	list all <code>at</code> jobs belonging to user
<code>atrm username</code>	remove all <code>at</code> jobs belonging to your username
<code>bg [%job_number]</code>	put <i>job_number</i> in the background; default is job that has been suspended with <code>CTRL+Z</code>
<code>CTRL+Z</code>	Suspend current job
<code>fg %job_number</code>	put <i>job_number</i> in the foreground
<code>jobs</code>	list background jobs
<code>kill -9 processID</code>	terminates process number <i>processID</i> , which is determined by issuing <code>ps</code> command
<code>nice command</code>	run command at lower priority; use for CPU intensive programs
<code>nohup command</code>	no hang-up; use to keep a background job from terminating at logout
<code>ps</code>	lists running processes
<code>ps -e</code>	show all processes
<code>ps -f</code>	full listing
<code>ps -l</code>	long listing
<code>ps -x</code>	show processes with no controlling terminal

## 14.3. TERMINATING RUNAWAY PROCESSES

Occasionally you may lose keyboard control of your terminal. If this happens, go to another terminal and log on to the same server you were logged on to when you lost control. Enter the command:

```
ps -fu username
```



The screen might look similar to the figure shown below.

```
USER      PID      PPID     C  STIME      TTY      TIME  CMD
metuuser  20488    23462    1  15:33:39   pts/2    0:00  -ksh
metuuser  21567    22222    1  15:53:11   pts/2    0:00  -ksh
metuuser  18877    25708    3  16:07:52   pts/2    0:00  ps -fu metuuser
```

Each login session can be identified by a line ending in your shell name (**-ksh/-bash/-sh/-csh**). The current session will be the one with the higher process number (PID 21567 in the listing above); therefore, the runaway session above has PID 20488. Terminate the session by entering the command:

```
kill -9 20488
```



## APPENDIX: SOME UTILITY COMMANDS

Utility Commands	Result
<code>cal month year</code>	returns a calendar for <i>month</i> (digit 1-12) and <i>year</i> (4 digits)
<code>date</code>	returns date and time of day
<code>df</code>	disk free; returns available disk space in kilobytes
<code>du directory</code>	disk usage; returns number of blocks used by <i>directory</i>
<code>file filenames</code>	shows the type of a file (e.g., ascii, directory, data, C program, etc.)
<code>more filename</code>	shows file contents
<code>paste file1 file2</code>	concatenates corresponding lines of files by joining them
<code>rlogin machine</code>	logs in to remote UNIX machine
<code>sort file</code>	sorts <i>file</i> in ascending order; many options available
<code>spell filename</code>	returns misspelled words
<code>stty</code>	returns terminal status; can be used to modify terminal options
<code>w</code>	returns who is doing what on the system
<code>who</code>	lists who is logged on to the system
<code>pg filename</code>	shows <i>file</i> page by page
<code>echo text</code>	displays <i>text</i> on the screen
<code>cat filename</code>	displays the contents of a file on the screen
<code>grep text</code>	searches <i>text</i> in files
<code>uname -mrs</code>	returns info about the system
<code>uname -a</code>	Returns all details about the system
<code>dig domainname</code>	retrieves information about a domain, such as name servers
<code>netstat</code>	shows network status and statistics
<code>id</code>	displays user's ID, name, group and group name
<code>finger username</code>	returns information about a user or a node
<code>ping ip</code>	tests and time a route to other computer
<code>traceroute ip/adress</code>	tests routes to the given ip (address)
<code>nslookup ip/adress</code>	returns address information

## REFERENCES

1. Hewlett Packard Publications, Beginner's Guide to HP-UX 1991, B1862-90000.
2. Wijay Mukh's, Working With UNIX 1993, BPB Publications
3. S. R. Bourne, The UNIX System 1983, International Computer Science Series, 0-201-13791-7.
4. Graham Glass, UNIX For Programmers and Users 1993, Prentice-Hall International Editions, 0-13-061771-7.
5. Brian W. Kernighan, Rob Pike, The UNIX Programming Environment 1984, Prentice-Hall Software Series, 0-13-937699-2.
6. J. Michael Lowe, Info World, A DOS User's Guide To UNIX 1990, Prentice-Hall International Editions, ISBN 0-13-219098-2.
7. IBM, AIX, XL Fortran Compiler/6000 1993, User's Guide, SC09-1610-00.
8. IBM, AIX XL Pascal Compiler/6000 1993, User's Guide, SC09-1756-00.
9. IBM, C Set ++ For AIX / 6000 1993, User's Guide, SC09-1605-00.
10. How to, Simon Fraser University 1993, Computing Handouts.
11. Learning The Bash Shell, O'Reilly & Associates, Inc. ISBN 1-56592-147-X
12. Selçuk Han Aydın, Linux İşletim Sistemi, ODTÜ-BİDB Yayınları, Ankara.
13. <http://goforit.unk.edu/UNIX/awk000.htm>

