Introduction to UNIX Operating System Ubuntu–based Lab manual.

Book · May 2013

CITATIONS
0

READS
7,038

1 author:

Sulieman Bani-Ahmad
Al-Balqa' Applied University
58 PUBLICATIONS **931** CITATIONS

SEE PROFILE

Introduction to
UNIX Operating System

Ubuntu-based Lab manual.

Last updated on March 23, 2015

Dr. Sulieman Bani-Ahmad
Department of Computer Information Systems
Prince Abdullah ben Ghazi Faculty of Science and Information Technology
Al-Balqa Applied University, Salt, Jordan
URL: Sulieman.net
Email: Sulieman@bau.edu.jo, Sulieman@case.edu

1. Table of contents

2. Familiarization

Reference: Page 2-1 to 2-20

Student Name:_____**Univ ID:**_____ **Section:**___

Goals

- To differentiate between UNIX and UNIX-like environments
- To learn how to obtain Linux distributions
- Comparison of different Linux distribution
- To familiarize student with the Ubuntu Linux environment
- To learn how to start a terminal session
- Learning the basics of UNIX command line syntax and comparing it to the DOS commands.
- Learning the "ls", the "pwd" and the "cd" commands
- Using the "ls", "pwd" and "cd" commands to surf the directory structure and to find the default directory of the current user.

Lab Tasks

[Task 1 – Do it at home] Obtain the Ubuntu Linux distribution and install it on your laptop or desktop.

[Task 2: Entering Ubuntu Linux] Start the computer in-front of you and login to Linux.

[Task 3:] Get yourself familiarized with the Ubuntu GUI interface.

3. Constructing and executing commands from a command line

Student Name:_____ **Univ ID:**_____ **Section:**___

Goals

- To learn how to use the command line interface of UNIX to run a selected set of commands
- To learn the UNIX command line syntax

Tasks

[Task 1 – Simple commands] Start a new terminal session and issue the following commands. Then write next to each command your observations:

$uname

$date

$Date

$cal

$clear

[Task 2 – Commands with options] Start a new terminal session and issue the following commands. Then write next to each command your observations:

$uname –i

$uname –n

$uname –s –r

$uname –rs

$uname –a

[Task 3 – Commands with arguments] Start a new terminal session and issue the following commands. Then write next to each command your observations:

$cal 12 2005

[Task 4 – Commands with options and arguments and multiple commands on a single line] Start a new terminal session and issue the following commands. Then write next to each command your observations:

$gedit ./Desktop/Myname.txt

In the opened window type your name and save the file then close the editor.

$ls

$ls –l

$ls Desktop

$ls –l Desktop

$pwd;ls

[Task 5 – Using online manual pages] Start a new terminal session and issue the following commands. Then write next to each command your observations:

$man uname

Note: Learn scrolling in the manual pages using the following keyboard keys. Write the effect of each key while viewing the manual page of the "uname" command

[The space bar]:

[The Return key]:

[The key 'b']:

[The key '/' with some pattern after it]:

[The key 'n' after the previous test]:

[The key 'h']:

[The key q]:

4. Viewing Directories

Student Name:_____ **Univ ID:**_____ **Section:**___

Goals

- To learn how to work with directories
- To learn how to work with files
- To learn the basic file and directory properties

Tasks

[Task 1 – Determining the current directory and listing directory contents] At the command prompt (i) issue the following commands and (ii) write-down your observations:

$pwd

$ls

$ls /

$ls /home

$ls; ls –a

[Task 2 –Displaying long lists of directory contents] At the command prompt (i) issue the following commands and (ii) write-down your observations:

$ls –l

[Task 3 –Understanding long lists of directory contents] At the command prompt (i) issue the following commands and (ii) write-down your observations:

```
$gedit  Lab3_example.txt
```

In the opened window type your name and save the file then close the editor.

```
$ls
```

```
$ls -l
```

*IMPORTANT [Task 3 –Understanding long lists of directory contents]. Explain the information about the file that you have just created in the previous task.

[Task 4 –Displaying individual directory and Displaying recursive list] At the command prompt (i) issue the following commands and (ii) write-down your observations:

```
$ls -R /home
```

```
$ls -ld .
```

[Task 5 –Displaying file types] At the command prompt (i) issue the following commands and (ii) write-down your observations:

$ls –F

What are the file types that you observed after running the above command.

[Task 6 –File types- checking contents of files] At the command prompt (i) issue the following commands and (ii) write-down your observations:

Note: the *file* command is used to check the type of files. This is useful if the filetype is not clear from its name or extension.

$ file Lab3_example.txt

[Task 7 –Changing directories] At the command prompt (i) issue the following commands and (ii) write-down your observations:

$pwd

$cd ..

$cd /

[Task 8 –returning to your home directory] At the command prompt (i) issue the following commands and (ii) write-down your observations:

$cd ~

$cd ~/Desktop

Question: What do the following path name abbreviations represent:

.	
..	
~	
/	

[Task 9 –The **cat**, **more**, **head**, **tail**, **wc**, and **lp** commands]

[Preparation steps] Create a text document using the gedit GUI editor. Save the file as lab3.txt. The content of the file is "Line 1" in the first line, "Line 2" in the second line and so on until "Line 20".

At the command prompt (i) issue the following commands and (ii) write-down your observations:

```
$ cat lab3.txt

$ more lab3.txt

$ head -5 lab3.txt

$ tail -6 lab3.txt

$ wc lab3.txt

$ wc -l   lab3.txt

$ lp lab3.txt

$ lpstat
```

5. Working with Files and Directories

Reference: Page 4-1 to Page 4-18

Student Name:_____ **Univ ID:**_____ **Section:**___

Goals

- To learn how to manage the directory structure of the UNIX files system.
- To learn how to use wildcards in UNIX environment.

Background

Here's a list of the most commonly used wildcards in bash:

Wildcard	Matches
*	zero or more characters
?	exactly one character
[abcde]	exactly one character listed
[a-e]	exactly one character in the given range
[!abcde]	any character that is not listed
[!a-e]	any character that is not in the given range
{debian,linux}	exactly one entire word in the options given

You can use wildcards with any command that accepts file names as arguments.

Tasks

[Task 1 –Copying files]

[Pre-steps] Use the gedit editor to create a file with the name "lab4.txt" and save it in the following location ~/Desktop

At the command prompt (i) issue the following commands and (ii) write-down your observations:

```
$ cd ~/Desktop


$ pwd


$ ls


$ cp lab4.txt lab4_copy.txt
```

```
$ cp lab4_copy.txt  ~
```

```
$ ls ~
```

```
$ cp  -i  lab4_copy.txt  ~
```

[Task 2 –Moving files]

[A Note] We will be using the files "lab4.txt" and "lab4_copy.txt" created in the previous task.

At the command prompt (i) issue the following commands and (ii) write-down your observations:

```
$ cd ~
```

```
$ mv   ~/Deskto/lab4.txt   .
```

```
$ ls lab*.*
```

```
$ ls ~/Desktop
```

[Task 3 – Renaming files]

[A Note] We will be using the files "lab4.txt" and "lab4_copy.txt" created in the previous task.

At the command prompt (i) issue the following commands and (ii) write-down your observations:

```
$ cd ~
```

```
$ mv   lab4.txt  lab4.txt.new_name
```

[Task 4 – Creating empty files] At the command prompt (i) issue the following commands and (ii) write-down your observations:

```
$ touch   moon   sun   earth
$ ls
```

[Task 5 – Removing files] At the command prompt (i) issue the following commands and (ii) write-down your observations:

```
$ rm  moon
```

```
$ rm sun earth
```

```
$ rm –i  lab*.*
```

[Task 6 – Creating directories] At the command prompt (i) issue the following commands and (ii) write-down your observations:

```
$ cd  ~
$ mkdir lab4_folder_level_1
$ cd lab4_folder_level_1
$ mkdir   lab4_folder_level_2
$ mkdir  ./ lab4_folder_level_2/ lab4_folder_level_3
```

What will be the output after computing the following command:

```
$ pwd
```

Question: Draw a tree that shows the path of the folder **"lab4_folder_level_3"** that you have just created starting by the root **"/"**

[Task 6 – Removing directories with their contents] At the command prompt (i) issue the following commands and (ii) write-down your observations:

```
$ rm –r lab4_folder_level_1
```

What did the last command do?

[Task 7 – Using wildcards] At the command prompt (i) issue the following commands and (ii) write-down your observations:

```
$ rm *
```

```
$ mv *linux*.html dir1
```

```
$ less d*.txt
```

```
$ rm junk.???
```

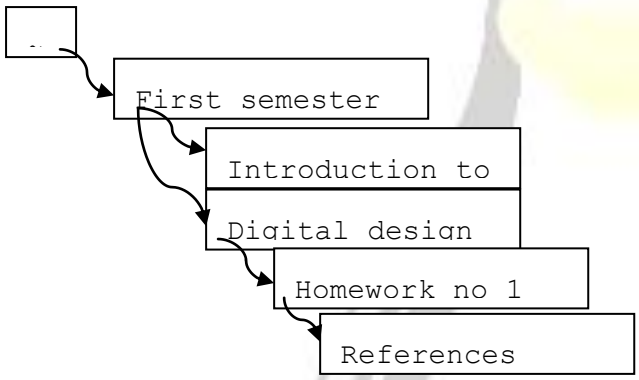
```
$ ls hda[0-9]
```

```
$ ls hda[0-9][0-9]
```

```
$ ls {hd,sd}[a-c]
```

```
$ rm *[!cehg]
```

[Exercise- Creating directories] Create the following folder hierarchy from the command prompt. Write down the set of commands required to do this exercise:



6. BASH Shell Scripting – Part I

Reference: BASH scripting – Simple stuff

Student Name:_____ **Univ ID:**_____ **Section:**___

Goals

- To learn how write and run shell scripts.

Information

String comparison operators

s1 = s2	s1 matches s2
s1 != s2	s1 does not match s2
s1 < s2	
s1 > s2	
-n s1	s1 is not null (contains one or more characters)
-z s1	s1 is null

Arithmetic operators

+ Addition
- Subtraction
* Multiplication
/ Division
% remainder

Arithmetic relational operators

-lt	(<)
-gt	(>)
-le	(<=)
-ge	(>=)
-eq	(==)
-ne	(!=)

Special symbols used in BASH scripting

- $* - This denotes all the parameters passed to the script at the time of its execution. Which includes $1, $2 and so on.
- $0 - Name of the shell script being executed.
- $# - Number of arguments specified in the command line.

- $? - Exit status of the last command.

If statement syntax

```
if condition_is_true
then
   execute commands
else
   execute commands
fi
```

another form is

```
if condition_is_true
then
   execute commands
else
   execute commands
fi
```

a third form is

```
if condition_is_true
then
   execute commands
elif another_condition_is_true
then
   execute commands
else
   execute commands
fi
```

Possible tests

Relational Operators

- -eq - Equal to
- -lt - Less than
- -gt - Greater than
- -ge - Greater than or Equal to
- -le - Less than or Equal to

File related tests

- -f file - True if file exists and is a regular file.
- -r file - True if file exists and is readable.
- -w file - True if file exists and is writable.

- -x file - True if file exists and is executable.
- -d file - True if file exists and is a directory.
- -s file - True if file exists and has a size greater than zero.

String tests

- -n str - True if string str is not a null string.
- -z str - True if string str is a null string.
- str1 == str2 - True if both strings are equal.
- str - True if string str is assigned a value and is not null.
- str1 != str2 - True if both strings are unequal.
- -s file - True if file exists and has a size greater than zero.

A few Example snippets of using test

```
test $d -eq 25 && echo $d
```

... which means, if the value in the variable d is equal to 25, print the value. Otherwise don't print anything.

```
if [ $d -eq 25 ]
then
echo $d
fi
```

In the above example, I have used square brackets instead of the keyword test - which is another way of doing the same thing.

```
if [ $str1 == $str2 ]
then
do something
fi
if [ -n "$str1" -a -n "$str2" ]
then
echo 'Both $str1 and $str2 are not null'
fi
```

... above, I have checked if both strings are not null then execute the echo command.

Things to remember while using test

1. If you are using square brackets [] instead of test, then care should be taken to insert a space after the [and before the].
2. test is confined to integer values only. Decimal values are simply truncated.
3. Do not use wildcards for testing string equality - they are expanded by the shell to match the files in your directory rather than the string.

Tasks

[Task 1 –A Piece of cake] Write Script to see current date, time, username, and current directory

```
#!/bin/bash
# Linux Shell Scripting
#
echo "Hello, $LOGNAME"
echo "Current date is”;
date
echo "User is”;
whoami
echo "Current directory"
pwd
```

[Task 2 –Adding two numbers] Write a shell script that will add two numbers, which are supplied as command line argument, and if these two numbers are not given show error.

```
#!/bin/bash
# Linux Shell Scripting
#Script to sum to numbers
if [ $# -ne 2 ]
then
    echo "Usage - $0   x    y"
    echo "        Where x and y are two nos for which I will print sum"
    exit 1
fi
    echo "Sum of $1 and $2 is”;
    expr $1 + $2
```

[Task 3 –Find the biggest number of the three] Write Script to find out the biggest number from given three numbers. Numbers are supplies as command line argument. Print error if sufficient arguments are not supplied.

```
#!/bin/bash
# Linux Shell Scripting

    if [ $# -ne 3 ]
    then
   echo "$0: number1 number2 number3 are not given"
```

```
        exit 1
    fi
    n1=$1
    n2=$2
    n3=$3
    if [ $n1 -gt $n2 ] && [ $n1 -gt $n3 ]
    then
    echo "$n1 is Bigest number"
    elif [ $n2 -gt $n1 ] && [ $n2 -gt $n3 ]
    then
    echo "$n2 is Bigest number"
    elif [ $n3 -gt $n1 ] && [ $n3 -gt $n2 ]
    then
        echo "$n3 is Bigest number"
    elif [ $1 -eq $2 ] && [ $1 -eq $3 ] && [ $2 -eq $3 ]
    then
    echo "All the three numbers are equal"
    else
        echo "I cannot figure out which number is bigger"
    fi
```

[Task 4 - String comparison examples]Try the following code that compares two strings.

```
#!/bin/bash
S1='string'
S2='String'
if [ $S1=$S2 ];
then
        echo "S1('$S1') is not equal to S2('$S2')"
fi
if [ $S1=$S1 ];
then
        echo "S1('$S1') is equal to S1('$S1')"
fi
```

7. BASH Shell Scripting – Part II

Reference: BASH scripting

Student Name:_____**Univ ID:**_____ **Section:**___

Goals

- To learn how write and run shell scripts.

Information

While statement syntax

```
while condition_is_true
do
execute commands
done
```

example

```
while [ $num -gt 100 ]
do
sleep 5
done
```

Until statement syntax

```
until false
do
execute commands
done
```

Tasks

[Task 1 –Numbers in reverse order] Write script to print numbers as 5,4,3,2,1 using ***while*** loop.

```
#!/bin/bash
# Linux Shell Scripting Tutorial #
i=5
while test $i != 0
do
    echo "$i"
    let i=i-1
done
```

[Task 2 –Numbers in order] Write script to print numbers as 0 … 9 using ***while*** loop.

```
#!/bin/bash
COUNTER=0
while [  $COUNTER -lt 10 ]; do
echo The counter is $COUNTER
let COUNTER=COUNTER+1
done
```

[Task 3 –Numbers in reverse order] Write script to print numbers as 20 … 10 using ***until*** loop.

```
#!/bin/bash
COUNTER=20
until [  $COUNTER -lt 10 ]; do
echo COUNTER $COUNTER
let COUNTER-=1
done
```

[Task 4 –Check is a file exists or not] Write script to determine whether given file exist or not, file name is supplied as command line argument, also check for sufficient number of command line argument.

```
#!/bin/bash
# Linux Shell Scripting
if [ $# -ne 1 ]
then
    echo "Usage - $0  file-name"
    exit 1
fi

if [ -f $1 ]
then
    echo "$1 file exists"
else
    echo "Sorry, $1 file does not exist"
fi
```

[Task 5 –Reading user input with read] Write a shell script to read the name of the user from keyboard and print it back to screen. Use ***read*** command.

```
#!/bin/bash
echo Please, enter your name
read NAME
echo "Hi $NAME!"
```

[Task 6 –Reading user input with read] Write a shell script to read the name (first and last names) of the user from keyboard and print it back to screen. Use ***read*** command.

```
#!/bin/bash
echo Please, enter your firstname and lastname
read FN LN
echo "Hi! $LN, $FN !"
```

8. BASH Shell Scripting – Part III

Reference: BASH scripting – Little more complicated stuff

Student Name:_____ **Univ ID:**_____ **Section:**___

Goals

- To learn how write and run more advanced shell scripts.

Information

Case statement syntax

```
case expression in
pattern1) execute commands ;;
pattern2) execute commands ;;
...
esac
```

Case example

```
...
echo "Enter your option : "
read i;

case $i in
1) ls -l ;;
2) date ;;
3) who ;;
4) exit
esac
```

The for loop syntax

```
for variable in list
do
execute commands
done
```

Example

```
...
for x in 1 2 3 4 5
do
echo "The value of x is $x";
```

```
done
```

Another example

```
...
for file in *.txt
do
echo $file
done
```

Tasks

[Task 1 –A Simple integer-numbers calculator] Write Script, using case statement to perform basic math operation as follows

+ addition
- subtraction
x multiplication
/ division

The name of script must be 'q4' which works as follows

$ ./calc 20 / 3, Also check for sufficient command line arguments

```
#!/bin/bash
# Linux Shell Scripting

if test $# = 3
then
    case $2 in
     +) let z=$1+$3;;
     -) let z=$1-$3;;
     /) let z=$1/$3;;
     x|X) let z=$1*$3;;
     *) echo Warning - $2 invalid operator, only +,-,x,/ operator allowed
        exit;;
    esac
    echo Answer is $z
else
    echo "Usage - $0   value1  operator value2"
    echo "        Where, value1 and value2 are numeric values"
    echo "               operator can be +,-,/,x (For Multiplication)"
fi
```

[Task 2 –A Simple real-numbers calculator] How to perform real number calculation in shell script and store result to third variable, let's say a=5.66, b=8.67, c=a+b?

```
#!/bin/bash
# Linux Shell Scripting
a=5.66
b=8.67
c=`echo $a + $b | bc`
echo "$a + $b = $c"
```

9. BASH Shell Scripting – Part IV

Reference: BASH scripting – Little more complicated stuff

Student Name:_____ **Univ ID:**_____ **Section:**___

Goals

- To learn how write and run more advanced shell scripts.

Tasks

[Task 1: Functions sample] Try out this code.

```
#!/bin/bash
function quit {
    exit
}
function hello {
    echo Hello!
}
hello
quit
echo "This will not be printed"
```

[Task 2: Functions with parameters sample] Try out this code.

```
#!/bin/bash
function quit {
    exit
}
function e {
    echo $1
}
e Hello
e World
quit
echo "This will not be printed"
```

[Task 3 –A Simple real-numbers calculator] Write Script, using case statement to perform basic math operation as follows

+ addition

- subtraction

```
      x multiplication
      / division
The name of script must be 'calc' which works as follows
$ ./calc 20.4 / 3.3, Also check for sufficient command line arguments
```

10. Programming underUNIX/Linux

Reference: Handouts are to be given to students.

Student Name:_____**Univ ID:**_____ **Section:**___

Goals

- To learn how to write, compile, and run C program under Linux.
- To learn how to write, compile, and run C++ program under Linux.
- To learn how to install new packages under Ubuntu Linux.

Introduction

Writing the source

The source of our first program is below

```
#include <stdio.h>
main()
{
  printf("I love Linux\n");
}
```

Save the program above and call it simplelq.c

Compiling the Source

The compile is done from the command line.

```
$  gcc -o simplelq simplelq.c
```

gcc is the GNU C compiler. The -o option tells it what to name the output file and the simplelq.c is the source file. The output from the compiler will be a binary file called simplelq

Running the executable

In order to run our sample executable we will need to apply the execute permission to the file (probably not required). Then we will execute it after.

```
$  chmod 744 simplelq
$  ./simplelq
```

Installing new packages under Ubuntu

The general syntax is

```
sudo apt-get install <selected package>
```

For example

```
$ sudo apt-get install g++
```

```
$ sudo apt-get install g++
    [sudo] password for sxb139:
    Reading package lists... Done
    Building dependency tree
    Reading state information... Done
```

What is sudo ?

In Linux (and UNIX in general), there is a SuperUser named Root. The Windows equivalent of Root is Administrators group. The SuperUser can do anything and everything, and thus doing daily work as the SuperUser can be dangerous. You could type a command incorrectly and destroy the system. Ideally, you run as a user that has only the privileges needed for the task at hand. In some cases, this is necessarily Root, but most of the time it is a regular user.

By default, the Root account password is locked in Ubuntu. This means that you cannot login as Root directly or use the **su** command to become the Root user. However, since the Root account physically exists it is still possible to run programs with root-level privileges. This is where **sudo** comes in - it allows authorized users (normally "Administrative" users to run certain programs as Root without having to know the root password.

Tasks

[Task 1 – Computing the powers of 2 using C – under UNIX] Use the gedit editor to write and save the following C code. Compile this code and run it to make sure it works.

```
#include <stdio.h>
#define N 16

int main(void) {
  int n;           /* The current exponent */
  int val = 1;     /* The current power of 2  */

  printf("\t  n  \t    2^n\n");
  printf("\t================\n");
  for (n=0; n<=N; n++) {
    printf("\t%3d \t %6d\n", n, val);
    val = 2*val;
  }
```

```
  return 0;
}
```

Note: The expected output is as follows:

```
      n       2^n
   ================
      0         1
      1         2
      2         4
      3         8
      4        16
      5        32
      6        64
      7       128
      8       256
      9       512
     10      1024
     11      2048
     12      4096
     13      8192
     14     16384
     15     32768
     16     65536
```

[Task 2 – Compiling C++ codes under UNIX] Use the gedit editor to write and save the following C++ code. Compile this code and run it to make sure it works. (Hint, search for g++ command).

```
// Purpose:   Read 10 integers into an array and then print
//        them out

#include <iostream>
using namespace std;
```

```
int main()
{
    int sample[10]; // this reserves 10 integer elements
    int t;

    // load the array
    for(t = 0; t < 10; t++)
          sample[t] = t;

    // display the array
    for(t = 0; t < 10; ++t)
          cout << sample[t] << ' ' ;

    return 0;
}
```

11. Compressing / Uncompressing files

Reference: Handouts are to be given to students.

Student Name:_____**Univ ID:**_____ **Section:**___

Goals

- To learn how compress and uncompress files (and folders) under Linux using the commands: zip, tar and gzip.

Compressing files under Linux or UNIX - Background

Both Linux and UNIX include various commands for Compressing and decompresses. To compress files you can use gzip, bzip2 and zip commands. To expand compressed file (decompresses) you can use and gzip -d, bunzip2 (bzip2 -d), unzip commands.

Compressing files

Syntax	Description	Example(s)
gzip {filename}	Gzip compress the size of the given files using Lempel-Ziv coding (LZ77). Whenever possible, each file is replaced by one with the extension .gz.	gzip mydata.doc gzip *.jpg ls -l
bzip2 {filename}	bzip2 compresses files using the Burrows-Wheeler block sorting text compression algorithm, and Huffman coding. Compression is generally considerably better than that achieved by bzip command (LZ77/LZ78-based compressors). Whenever possible, each file is replaced by one with the extension .bz2.	bzip2 mydata.doc bzip2 *.jpg ls -l
zip {.zip-filename} {filename-to-compress}	zip is a compression and file packaging utility for Unix/Linux. Each file is stored in single .zip {.zip-filename} file with the extension .zip.	zip mydata.zip mydata.doc zip data.zip *.doc ls -l
tar -zcvf {.tgz-file} {files} tar -jcvf {.tbz2-file} {files}	The GNU tar is archiving utility but it can be use to compressing large file(s). GNU tar supports	tar -zcvf data.tgz *.doc tar -zcvf pics.tar.gz *.jpg *.png tar -jcvf data.tbz2 *.doc

	both archive compressing through gzip and bzip2. If you have more than 2 files then it is recommended to use tar instead of gzip or bzip2. -z: use gzip compress -j: use bzip2 compress	ls -l

Decompressing files

Syntax	Description	Example(s)
gzip -d {.gz file} gunzip {.gz file}	Decompressed a file that is created using gzip command. File is restored to their original form using this command.	gzip -d mydata.doc.gz gunzip mydata.doc.gz
bzip2 -d {.bz2-file} bunzip2 {.bz2-file}	Decompressed a file that is created using bzip2 command. File is restored to their original form using this command.	bzip2 -d mydata.doc.bz2 gunzip mydata.doc.bz2
unzip {.zip file}	Extract compressed files in a ZIP archive.	unzip file.zip unzip data.zip resume.doc
tar -zxvf {.tgz-file} tar -jxvf {.tbz2-file}	Untar or decompressed a file(s) that is created using tar compressing through gzip and bzip2 filter	tar -zxvf data.tgz tar -zxvf pics.tar.gz *.jpg tar -jxvf data.tbz2

List the contents of an archive/compressed file

Some time you just wanted to look at files inside an archive or compressed file. Then all of the above command supports file list option.

Syntax	Description	Example(s)
gzip -l {.gz file}	List files from a GZIP archive	gzip -l mydata.doc.gz
unzip -l {.zip file}	List files from a ZIP archive	unzip -l mydata.zip
tar -ztvf {.tar.gz} tar -jtvf {.tbz2}	List files from a TAR archive	tar -ztvf pics.tar.gz tar -jtvf data.tbz2

Tasks

[Task 1 – Compression with the zip command]

```
$ zip alldocs.zip *.doc
```

```
$ zip -r   MyDocs.zip  MyDocs/
```

```
[Task 2 – Decompression with the unzip command]
$ unzip alldocs.zip
```

```
$ unzip  MyDocs.zip
```

```
[Task 3 – Compression with the gzip command]
$ gzip alldocs.gz *.doc
```

```
$ gzip -r   MyDocs.gz  MyDocs/
 [Task 4 – Decompression with the gunzip command]
$ gunzip alldocs.gz
```

```
$ gunzip  MyDocs.gz
```

```
[Task 5 – Archiving with the tar command]
$ tar   cvf    alldocs.tar    *.doc
```

```
$ tar      cvf   MyDocs.tar       MyDocs/
```

[Task 6 – Unarchiving with the tar command]

```
$ tar   xvf    alldocs.tar
```

```
$ tar      xvf   MyDocs.tar
```

[Task 7 – Archiving then compression with the tar then the gzip commands]

```
$ tar   cvf    alldocs.tar    *.doc
```

```
$ gzip   alldocs.tar.gz    alldocs.tar
```

[Task 8 – Decompressing and then unarchiving with the gunzip then the tar commands]

```
$ guzip   alldocs.tar.gz
```

```
$ tar   xvf    alldocs.tar
```

[Task 9 – Archiving then compression with the tar then the gzip commands]

```
$ tar   cvf    alldocs.tar    *.doc   |   gzip   alldocs.tar.gz
```

[Task 10 – Decompressing and then unarchiving with the gunzip then the tar commands]

12. Regular expression with the *grep* command – Part I

Reference: included in this handout.

Student Name: _____ **Univ ID:** _____ **Section:** _____

Goals

- To learn how to use the grep command
- To learn the basics of regular expressions
- To learn how to perform search commands using via grep and employing regular expressions

Preliminaries – Basics of regular expressions

```
^                =    match expression at the start of a line, as in ^A.
$                =    match expression at the end of a line, as in A$.
\ (Back Slash)   =    turn off the special meaning of the next character, as in \^.
[ ] (Brackets)   =    match any one of the enclosed characters, as in [aeiou].
                        Use Hyphen "-" for a range, as in [0-9].
[^ ]             =    match any one character except those enclosed in [ ], as in
[^0-9].
. (Period)       =    match a single character of any value, except end of line.
* (Asterisk)     =    match zero or more of the preceding character or expression.
\{x,y\}          =    match x to y occurrences of the preceding.
\{x\}            =    match exactly x occurrences of the preceding.
\{x,\}           =    match x or more occurrences of the preceding.
```

Examples of grep commands

```
    grep smug files          {search files for lines with 'smug'}
    grep '^smug' files       {'smug' at the start of a line}
    grep 'smug$' files       {'smug' at the end of a line}
    grep '^smug$' files      {lines containing only 'smug'}
    grep '\^s' files         {lines starting with '^s', "\" escapes the ^}
    grep '[Ss]mug' files     {search for 'Smug' or 'smug'}
    grep 'B[oO][bB]' files   {search for BOB, Bob, BOb or BoB }
    grep '^$' files          {search for blank lines}
    grep '[0-9][0-9]' file   {search for pairs of numeric digits}
```

Next are yet more examples

```
    grep '[a-zA-Z]'          {any line with at least one letter}
```

```
grep '[^a-zA-Z0-9]        {anything not a letter or number}
grep '[0-9]\{3\}-[0-9]\{4\}' {999-9999, like phone numbers}
grep '^.$'                {lines with exactly one character}
grep '"smug"'                  {'smug' within double quotes}
grep '"*smug"*'             {'smug', with or without quotes}
grep '^\.'          {any line that starts with a Period "."}
grep '^\.[a-z][a-z]' {line start with "." and 2 lc letters}
```

Tasks

Go to the download section of the instructor https://sites.google.com/site/suliemancourses and download the file named **USdic.txt** (posted in compressed format) into your home folder. After that, answer the following few questions.

Q1: Find the list of files that has the permission "-rw-rw-r—"

Q2: Find the list of files that owned by the user with the name "ali"

Q3: Find all English words that start with the letter "f".

Q4: Find all English language words that ends with the letter "f"

Q5: Find all English language words that *do not* end with the letter "f"

Q6: Find all English words that contain the "sh" **or** "ch" parts.

Q7: What do each of the following commands do?

```
%   grep  '^.$'     USdic.txt

%   grep  '.$'      USdic.txt

%   grep  '^.'      USdic.txt

%  grep   '^\.[a-z][a-z]'     USdic.txt

%   grep   '[a-m]$'      USdic.txt

%   grep   '[a-cA-D]'      USdic.txt

%   grep 'M[oO] [oO] [sS][e]'     USdic.txt
```

13. Regular expression with the *grep* command –Part II

Reference: included in this handout.

Student Name: _____ **Univ ID:** _____ **Section:** _____

Goals

- To learn how to use the grep command
- To learn the basics of regular expressions
- To learn how to perform search commands using via grep and employing regular expressions

Tasks

[Task 1]: Write a shell scrip code to look for *your name* inside *all files* in the *present working directory* from where the script is called.

14. Viewing and Changing File and Directory Permissions

Reference: Module 7 of the textbook. Pages 7-1 to 7-27.

Student Name:_____**Univ ID:**_____ **Section:**___

Goals

- To learn how the basics of Unix Access Control Lists (ACL) and the basic of viewing/changing access permission of files and directories for the three main UNIX user types.

Tasks

[Task 1 –Review questions] Answer each one of the following questions:

Q1: What are the three main categories of permissions?

Q2: What are the three main types of UNIX OS users?

Q3: Given the following file permission set "-rwx---r—".

What can/cannot the owner of the file do?

What can/cannot a user from the same group of the owner of the file do?

What can/cannot an anonymous user to do to this file?

Q4: Draw a decision tree for determining file/directory permissions.

Q5: What are the two permission modes used in the *chmod* UNIX command?

[Task 2 –Review questions] What does each of the following UNIX commands do? Be as specific as possible.

```
$ chmod o-r homework1.doc
```

```
$ chmod o+w homework1.doc
```

```
$ chmod u-x myfile
```

```
$ chmod g+x myfile
```

```
$ chmod a+x myfile
```

```
$ chmod go+x myfile
```

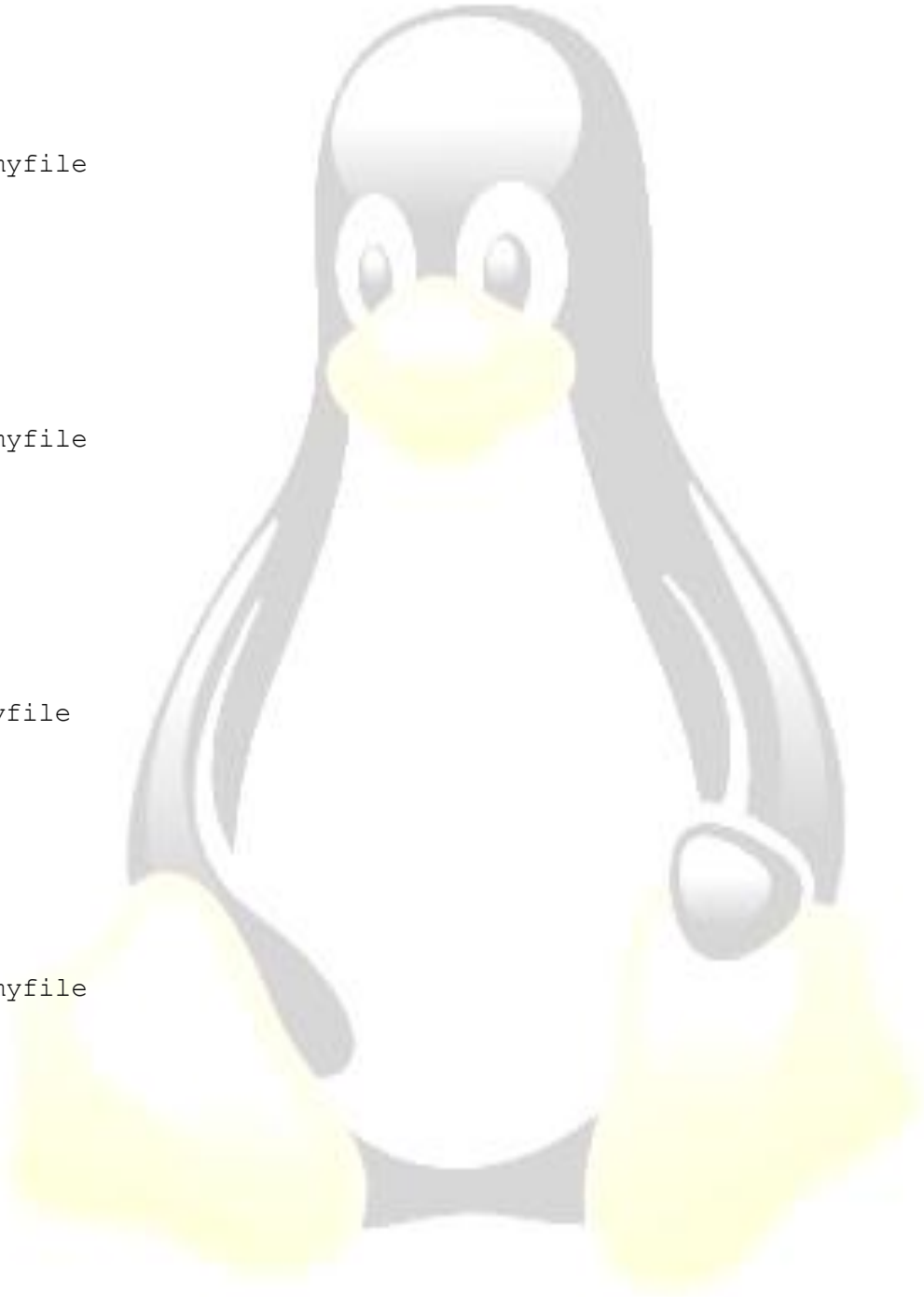
```
$ chmod 555 myfile
```

```
$ chmod 111 myfile
```

```
$ chmod 762 myfile
```

```
$ chmod 76 myfile
```

```
$ chmod 755 myfile
```



15. Handling packages for Debian-based Linux

Reference: Included in this handout.

Student Name:_____Univ **ID:**_____ **Section:**___

Goals

- To learn how the basics of Unix Access Control Lists (ACL) and the basic of viewing/changing access permission of files and directories for the three main UNIX user types.

Background

apt-get command cheat sheet for Debian Linux

apt-get is the command-line tool for handling packages for Debian Linux which is use to:

- Install/manage individual packages
- Upgrade packages
- Apply security patch(s)
- Keep Debian system up to date
- Download source .deb files
- Front-end to many GUI and other utilities

Here is quick cheat sheet you will find handy while using apt-get at shell prompt:

Syntax	Description	Example(s)
apt-get install {package}	Install the new package. If package is installed then try to upgrade to latest version	apt-get install zip apt-get install lsof samba mysql-client
apt-get remove {package}	Remove/Delete an installed package except configuration files	apt-get remove zip
apt-get --purge remove {package}	Remove/Delete everything including configuration files	apt-get --purge remove mysql-server
apt-get update apt-get upgrade	Resynchronize the package index files and Upgrade the Debian Linux system including security update (Internet access required)	apt-get update apt-get upgrade
apt-get update apt-get dist-upgrade	Usually use to upgrade to Debian distribution. For example Woody to Sarge upgrade. 'dist-upgrade' in addition to performing the function of upgrade, also	apt-get update apt-get dist-upgrade

	intelligently handles changing dependencies with new versions of packages; apt-get has a "smart" conflict resolution system, and it will attempt to upgrade the most important packages at the expense of less important ones if necessary.	

Installing new packages under Ubuntu

The general syntax is

```
sudo apt-get install <selected package>
```

For example

```
$ sudo apt-get install g++
```

```
$ sudo apt-get install g++
    [sudo] password for sxb139:
    Reading package lists... Done
    Building dependency tree
    Reading state information... Done
```

What is sudo ?

In Linux (and UNIX in general), there is a SuperUser named Root. The Windows equivalent of Root is Administrators group. The SuperUser can do anything and everything, and thus doing daily work as the SuperUser can be dangerous. You could type a command incorrectly and destroy the system. Ideally, you run as a user that has only the privileges needed for the task at hand. In some cases, this is necessarily Root, but most of the time it is a regular user.

By default, the Root account password is locked in Ubuntu. This means that you cannot login as Root directly or use the **su** command to become the Root user. However, since the Root account physically exists it is still possible to run programs with root-level privileges. This is where **sudo** comes in - it allows authorized users (normally "Administrative" users to run certain programs as Root without having to know the root password.

16. Appendices

The Linux filesystem explained

This article

- explains the differences between the Windows and Linux filesystems,
- takes you through the layout of the Linux filesystem.

Differences between the Linux and Windows files system:

In Linux, there is only a single hierarchal directory structure.

- Every path in Linux starts from the root directory, represented by '/', and then expands into sub-directories.
- Where DOS/Windows had various partitions and then directories under those partitions,
- Linux places all the partitions under the root directory by 'mounting' them under specific directories. Closest to root under Windows would be drive c: in Windows.

Under Windows, the various partitions are detected at boot and assigned a drive letter.

Under Linux, unless you mount a partition or a device, the system does not know of the existence of that partition or device.

This kind of layout, known as the *unified filesystem*, does offer several advantages over the approach that Windows uses.

Let's take the example of the /usr directory. This directory off the root directory contains most of the system executables. With the Linux filesystem, you can choose to mount it off another partition or even off another machine over the network. The underlying system will not know the difference because /usr appears to be a local directory that is part of the local directory structure! How many times have you wished to move around executables and data under Windows, only to run into registry and system errors? Try moving c:windowssystem to another partition or drive.

Linux is *case sensitive*. What this means that the case, whether in capitals or not, of the characters becomes very important. So this is not the same as THIS or ThIs for that matter.

Linux filesystems layout

We now move on to the layout or the directory structure of the Linux filesystem. Given below is the result of a 'ls -p' in the root directory.

```
bin/ dev/ home/ lost+found/ proc/ sbin/ usr/ boot/ etc/ lib/ mnt/ root/ tmp/
var/
```

/sbin - This directory contains all the binaries that are essential to the working of the system. These include system administration as well as maintenance and hardware configuration programs. *These are the essential programs that are required by all the users.*

Another directory that contains system binaries is **/usr/sbin**. This directory contains other binaries of use to the system administrator. This is where you will find the binaries that only the system administrator has access to.

/bin - In contrast to /sbin, the bin directory contains several useful commands that are used by both the system administrator as well as non-privileged users. This directory usually contains the shells like ***bash***, csh etc. as well as much used commands like cp, mv, rm, cat, ls. There also is /usr/bin, which contains other user binaries. These binaries on the other hand are not essential for the user. The binaries in /bin however, a user cannot do without.

/boot - This directory contains the *system.map file* as well as the Linux kernel. Lilo places the boot sector backups in this directory.

A symbol table a symbol table used by the kernel. A symbol table is a look-up between symbol names and their addresses in memory. A symbol name may be the name of a *variable* or the name of a function.

/dev - This is a very interesting directory that highlights one important characteristic of the Linux filesystem - everything is a file or a directory. Look through this directory and you should see hda1, hda2 etc, which represent the various partitions on the first master drive of the system. /dev/cdrom and /dev/fd0 represent your CDROM drive and your floppy drive. This may seem strange but it will make sense if you compare the characteristics of files to that of your hardware. Both can be read from and written to.

Take /dev/dsp, for instance. This file represents your speaker device. So any data written to this file will be re-directed to your speaker.

Try 'cat /etc/lilo.conf > /dev/dsp' and you should hear some sound on the speaker. That's the sound of your lilo.conf file! Similarly, sending data to and reading from /dev/ttyS0 (COM 1) will allow you to communicate with a device attached there - your modem.

/etc - This directory contains all the configuration files for your system.

/home - Linux is a multi-user environment so each user is also assigned a specific directory which is accessible only to them and the system administrator. These are the user home directories, which can be found under /home/username.

/lib - This contains all the shared libraries that are required by system programs. Windows equivalent to a shared library would be a DLL file.

/lost+found - Linux should always go through a proper shutdown. Sometimes your system might crash or a power failure might take the

machine down. Either way, at the next boot, a lengthy filesystem check using ***fsck*** will be done. ***Fsck*** will go through the system and try to recover any corrupt files that it finds. The result of this recovery operation will be placed in this directory.

/mnt - This is a generic mount point under which you mount your filesystems or devices. Mounting is the process by which you make a filesystem available to the system. After mounting your files will be accessible under the mount-point. This directory usually contains mount points or sub-directories where you mount your floppy and your CD. You can also create additional mount-points here if you want. There is no limitation to creating a mount-point anywhere on your system but convention says that you do not litter your file system with mount-points.

/opt - This directory contains all the software and add-on packages that are not part of the default installation. Generally you will find KDE and StarOffice here. Again, this directory is not used very often as it's mostly a standard in Unix installations.

/proc - This is a special directory on your system.

/root - We talked about user home directories earlier and well this one is the home directory of the user root. This is not to be confused with the system root, which is directory at the highest level in the filesystem.

/tmp - This directory contains mostly files that are required temporarily. Many programs use this to create lock files and for temporary storage of data.

/usr - This is one of the most important directories in the system as it contains all the user binaries. X and its supporting libraries can be found here. User programs like telnet, ftp etc are also placed here. **/usr/doc** contains useful system documentation. **/usr/src/linux** contains the source code for the Linux kernel.

/var - This directory contains spooling data like mail and also the output from the printer daemon. The system logs are also kept here in **/var/log/messages**.

Basic UNIX Commands

Copying Files

cp (copy)

cp *file1 file2* is the command which makes a copy of **file1** in the current working directory and calls it **file2**

What we are going to do now, is to take a file stored in an open access area of the file system, and use the cp command to copy it to your unixstuff directory.

First, cd to your **unixstuff** directory.

```
% cd ~/unixstuff
```

Then at the UNIX prompt, type,

```
% cp /vol/examples/tutorial/science.txt .
```

Note: Don't forget the dot . at the end. Remember, in UNIX, the dot means the current directory.

The above command means copy the file **science.txt** to the current directory, keeping the name the same.

(Note: The directory **/vol/examples/tutorial/** is an area to which everyone in the school has read and copy access. If you are from outside the University, you can grab a copy of the file here. Use 'File/Save As..' from the menu bar to save it into your **unixstuff** directory.)

Moving files

mv (move)

mv *file1 file2* moves (or renames) **file1** to **file2**

To move a file from one place to another, use the mv command. This has the effect of moving rather than copying the file, so you end up with only one file rather than two.

It can also be used to rename a file, by moving the file to the same directory, but giving it a different name.

We are now going to move the file science.bak to your backup directory.

First, change directories to your unixstuff directory (can you remember how?). Then, inside the **unixstuff** directory, type

```
% mv science.bak backups/.
```

Type ls and ls backups to see if it has worked.

Removing files and directories

rm (remove), rmdir (remove directory)

To delete (remove) a file, use the rm command. As an example, we are going to create a copy of the **science.txt** file then delete it.

Inside your **unixstuff** directory, type

```
% cp science.txt tempfile.txt
% ls
% rm tempfile.txt
% ls
```

You can use the rmdir command to remove a directory (make sure it is empty first). Try to remove the **backups** directory. You will not be able to since UNIX will not let you remove a non-empty directory.

Displaying the contents of a file on the screen

clear (clear screen)

Before you start the next section, you may like to clear the terminal window of the previous commands so the output of the following commands can be clearly understood.

At the prompt, type

```
% clear
```

This will clear all text and leave you with the % prompt at the top of the window.

cat (concatenate)

The command cat can be used to display the contents of a file on the screen. Type:

```
% cat science.txt
```

As you can see, the file is longer than than the size of the window, so it scrolls past making it unreadable.

less

The command less writes the contents of a file onto the screen a page at a time. Type

```
% less science.txt
```

Press the [**space-bar**] if you want to see another page, and type [**q**] if you want to quit reading. As you can see, less is used in preference to cat for long files.

head

The head command writes the first ten lines of a file to the screen.

First clear the screen then type

```
% head science.txt
```

Then type

```
% head -5 science.txt
```

What difference did the -5 do to the head command?

tail

The tail command writes the last ten lines of a file to the screen.

Clear the screen and type

```
% tail science.txt
```

Q. How can you view the last 15 lines of the file?

Searching the contents of a file

Simple searching using less

Using less, you can search though a text file for a keyword (pattern). For example, to search through **science.txt** for the word '**science**', type

```
% less science.txt
```

then, still in less, type a forward slash [**/**] followed by the word to search

```
/science
```

As you can see, less finds and highlights the keyword. Type [**n**] to search for the next occurrence of the word.

grep (don't ask why it is called grep)

grep is one of many standard UNIX utilities. It searches files for specified words or patterns. First clear the screen, then type

```
% grep science science.txt
```

As you can see, grep has printed out each line containing the word **science.**

Try typing

```
% grep Science science.txt
```

The grep command is case sensitive; it distinguishes between Science and science.

To ignore upper/lower case distinctions, use the -i option, i.e. type

```
% grep -i science science.txt
```

To search for a phrase or pattern, you must enclose it in single quotes (the apostrophe symbol). For example to search for spinning top, type

```
% grep -i 'spinning top' science.txt
```

Some of the other options of grep are:

-v display those lines that do NOT match

-n precede each matching line with the line number

-c print only the total count of matched lines

Try some of them and see the different results. Don't forget, you can use more than one option at a time. For example, the number of lines without the words science or Science is

```
% grep -ivc science science.txt
```

wc (word count)

A handy little utility is the wc command, short for word count. To do a word count on **science.txt**, type

```
% wc -w science.txt
```

To find out how many lines the file has, type

```
% wc -l science.txt
```

UNIX Quick Reference Sheet

1 Log In Session

1.1 Log In

Enter username at login: prompt. Be carefull - Unix is case sensitive.

Enter password at password: prompt.

1.2 Change Password

```
$ passwd
```

1.3 Log Out

```
$ logout
```

or

```
$ exit
```

2 File System

2.1 Create a File

```
cat >  file
```

Enter text and end with ctrl-D

```
vi  file
```

Edit file using the vi editor

2.2 Make a Directory

```
mkdir   directory-name
```

2.3 Display File Contents

cat	file	display contents of file
more	file	display contents of file one screenfull at a time.
view	file	a read only version of vi.
less	file	similar to, but more powerfull than more.
		See the man page for more infomation on less.

2.4 Comparing Files

diff file1 file2	line by comparison
cmp file1 file2	byte by byte comparison

2.5 Changing Access Modes

chmod mode file1 file2 ...

chmod -R mode dir (changes all files in dir)

Mode Settings

- u user (owner)
- g group
- other

(+) add permission (-) remove permission

- r read
- w write
- x execute

Example:

```
$ chmod go+rwx public.html
```

adds read, write, and execute permissions for group and other on public.html.

2.6 List Files and Directories

```
$ ls        list contents of directory
$ ls -a     include files with "." (dot files)
$ ls -l     list contents in long format (show modes)
```

2.7 Move (or Rename) Files and Directories

```
$ mv src-file dest-file    rename src-file to  dest-file
$ mv src-file dest-dir     move a file into a directory
$ mv src-dir dest-dir      rename src-dir, or move to dest-dir
$ mv -i src dest           copy & prompt before overwriting
```

2.8 Copy Files

```
$ cp src-file dest-file    copy src-file to  dest-file
$ cp src-file dest-dir     copy a file into a directory
$ cp -R src-dir dest-dir   copy one directory into another
$ cp -i src dest           copy & prompt before overwriting
```

2.9 Remove File

```
$ rm file      remove (delete) a file
$ rmdir dir    remove an empty directory
$ rm -r dir    remove a directory and its contents
$ rm -i file   remove file, but prompt before deleting
```

2.10 Compressing files

```
$ compress file       encode file, replacing it with file.Z
$ zcat file.Z         display compressed file
$ uncompress file.Z   decode file.Z, replacing it with file
```

2.11 Find Name of Current Directory

```
pwd    display absolute path of working directory
```

2.12 Pathnames

simple:

One filename or directory name for accessing local file or directory.

Example: foo.c

absolute:

List of directory names from root directory to desired file or directory name, each separated by /.

Example: /src/shared

relative:

List of directory names from working directory to desired file or directory name, each separated by /.

Example: Mail/inbox/23

2.13 Directory Abbreviations

~	Your home (login) directory
~username	Another user's home directory
.	Working (current) directory
..	Parent of working directory
../..	Parent of parent directory

2.14 Change Working Directory

cd /	go to the root directory
cd	go to your login (home) directory
cd ~username	go to username's login (home) directory
cd ~username/directory	go to username's indicated directory
cd ..	go up one directory level from here
cd ../..	go up two directory levels from here
cd /full/path/name/from/root	change directory to absolute path named. note the leading slash
cd path/from/current/directory	change directory to path relative to

	here. note there is no leading slash

3.0 Commands

3.1 Date

```
$ date    display date and time
```

3.2 Wild Cards

```
?    single character wild card
*    Arbitrary number of characters
```

3.3 Printing

```
$ lpr file            print file on default printer
$ lpr -Pprinter file  print file on printer
$ lpr -c# file        print # copies of file
$ lpr -d file         interpret file as a dvi file
$ lpq                 show print queue (-Pprinter also valid)
$ lprm -#             remove print request # (listed with lpq)
```

3.4 Redirection

```
$ command > file
```

direct output of command to file instead of to standard output (screen), replacing current contents of file.

```
$ command >> file
```

as above, except output is appended to the current contents of file.

```
$ command < file
```

command receives input from file instead of from standard input (keyboard)

```
$ cmd1 | cmd2
```

"pipe" output of cmd1 to input of cmd2

4 Search Files

```
$ grep string filelist
```

show lines containing string in any file in filelist

```
$ grep -v string filelist
```

show lines not containing string

```
$ grep -i string filelist
```

show lines containing string, ignore case

5 Information on Users

```
$ finger user or
$ finger user@machine
```

get information on a user

```
$ finger @machine
```

list users on machine

```
$ who
```

list current users

6 Timesavers

6.1 Aliases

```
$ alias string command
```

abbreviate command to string

6.2 History: Command Repetition

Commands may be recalled

```
history     show command history
!num        repeat command with history number num
!str        repeat last command beginning with string str
!!          repeat entire last command line
!$          repeat last word of last command line
```

7.0 Process and Job Control

7.1 Important Terms

```
pid         Process IDentification number.
job-id      Job identification number.
```

7.2 Display Process and/or Job IDs

```
ps        report processes and pid numbers
ps gx     as above, but include "hidden" processes
jobs      report current jobs and job id numbers
```

7.3 Stop (Suspend) a Job

```
ctrl-Z    NOTE: process still exists!
```

7.4 Run a Job in the Background

To start a job in background add & to the end of the command.

Example: xv foo.gif &

To force a running job into the background:

ctrl-Z stop the job

bg "push" the job into the background

7.5 Bring a Job to the Foreground

fg bring a job to foreground

fg %job-id foreground by job-id (see 7.2)

7.6 Kill a Process or Job

ctrl-C kill foreground process

kill -KILL pid# see 7.2 for

kill -KILL %job-id# displaying pids & job-ids

8.0 Mail Handler (MH)

MH commands are issued directly to the terminal.

inc incorporate new mail

scan show list of mail messages

show show current message

next show next message

prev show previous message

repl reply to current message

forw forward current message

comp compose a mail message

rmm remove current mail message

cmd -help print help on mh commmand cmd

The file .mh_profile is used to enable/disable MH features. man mh-profile for more information.

9.0 On-line Assistance

On-line Documentation

```
$ man command-name
```

display on-line manual pages

```
$ man -k  string
```

list one-line summaries of manual pages containing string

Wildcards in UNIX

What are shell wildcards?

Wildcards are a shell feature that makes the command line much more powerful than any GUI file managers. You see, if you want to select a big

group of files in a graphical file manager, you usually have to select them with your mouse. This may seem simple, but in some cases it can be very frustrating. For example, suppose you have a directory with a huge amount of all kinds of files and subdirectories, and you decide to move all the HTML files, that have the word "linux" somewhere in the middle of their names, from that big directory into another directory. What's a simple way to do this? If the directory contains a huge amount of differently named HTML files, your task is everything but simple!

In the Linux CLI that task is just as simple to perform as moving only one HTML file, and it's so easy because of the shell wildcards. Wildcards are special characters that allow you to select filenames that match certain patterns of characters. This helps you to select even a big group of files with typing just a few characters, and in most cases it's easier than selecting the files with a mouse.

Here's a list of the most commonly used wildcards in bash:

Wildcard	Matches
*	zero or more characters
?	exactly one character
[abcde]	exactly one character listed
[a-e]	exactly one character in the given range
[!abcde]	any character that is not listed
[!a-e]	any character that is not in the given range
{debian,linux}	exactly one entire word in the options given

You can use wildcards with any command that accepts file names as arguments.

Wildcard examples

Let's have a few examples. Probably the * character is already familiar to you, because it's widely used in many other places, too, not just in Linux. For example, the following removes every file from the current directory:

```
$ rm *
```

The following command moves all the HTML files, that have the word "linux" in their names, from the working directory into a directory named dir1:

```
$ mv *linux*.html dir1
```

The following displays all files that begin with d and end with .txt:

```
$ less d*.txt
```

The following command removes all files whose names begin with junk., followed by exactly three characters:

```
$ rm junk.???
```

With this command you list all files or directories whose names begin with hda, followed by exactly one numeral:

```
$ ls hda[0-9]
```

This lists all files or directories beginning with hda, followed by exactly two numerals:

```
$ ls hda[0-9][0-9]
```

The following lists all files or directories whose name starts with either hd or sd, followed by any single character between a and c:

```
$ ls {hd,sd}[a-c]
```

This command copies all files, that begin with an uppercase letter, to directory dir2:

```
$ cp [A-Z]* dir2
```

This deletes all files that don't end with c, e, h or g:

```
$ rm *[!cehg]
```

Redirection

Most processes initiated by UNIX commands write to the standard output (that is, they write to the terminal screen), and many take their input from the standard input (that is, they read it from the keyboard). There is also the standard error, where processes write their error messages, by default, to the terminal screen.

The cat command is used to write the contents of a file to the screen. Now type cat without specifying a file to read

```
% cat
```

Then type a few words on the keyboard and press the [Return] key.

Finally hold the [Ctrl] key down and press [d] (written as ^D for short) to end the input.

What has happened?

If you run the cat command without specifying a file to read, it reads the standard input (the keyboard), and on receiving the'end of file' (^D), copies it to the standard output (the screen).

In UNIX, we can redirect both the input and the output of commands.

Redirecting the Output

We use the > symbol to redirect the output of a command. For example, to create a file called list1 containing a list of fruit, type

```
% cat > list1
```

Then type in the names of some fruit. Press [Return] after each one.

```
% cat > list1
pear
banana
apple
^D (Control D to stop)
```

What happens is the cat command reads the standard input (the keyboard) and the > redirects the output, which normally goes to the screen, into a file called list1

To read the contents of the file, type

```
% cat list1
```

Exercise 3a

Using the above method, create another file called list2 containing the following fruit: orange, plum, mango, grapefruit. Read the contents of list2.

The form >> appends standard output to a file. So to add more items to the file list1, type

```
% cat >> list1
```

Then type in the names of more fruit

```
peach
grape
orange
^D (Control D to stop)
```

To read the contents of the file, type

```
% cat list1
```

You should now have two files. One contains six fruit, the other contains four fruit. We will now use the cat command to join (concatenate) list1 and list2 into a new file called biglist. Type

```
% cat list1 list2 > biglist
```

What this is doing is reading the contents of list1 and list2 in turn, then outputting the text to the file biglist

To read the contents of the new file, type

```
% cat biglist
```

Redirecting the Input

We use the < symbol to redirect the input of a command.

The command sort alphabetically or numerically sorts a list. Type

```
% sort
```

Then type in the names of some vegetables. Press [Return] after each one.

```
carrot
beetroot
artichoke
^D (control d to stop)
```

The output will be

```
artichoke
beetroot
carrot
```

Using < you can redirect the input to come from a file rather than the keyboard. For example, to sort the list of fruit, type

```
% sort < biglist
```

and the sorted list will be output to the screen.

To output the sorted list to a file, type,

```
% sort < biglist > slist
```

Use cat to read the contents of the file slist

Pipes

To see who is on the system with you, type

```
% who
```

One method to get a sorted list of names is to type,

```
% who > names.txt
% sort < names.txt
```

This is a bit slow and you have to remember to remove the temporary file called names when you have finished. What you really want to do is connect the output of the who command directly to the input of the sort command. This is exactly what pipes do. The symbol for a pipe is the vertical bar |

For example, typing

```
% who | sort
```

will give the same result as above, but quicker and cleaner.

To find out how many users are logged on, type

```
% who | wc -l
```

Summary

command > file	redirect standard output to a file
command >> file	append standard output to a file
command < file	redirect standard input from a file
command1 | command2	pipe the output of command1 to the

	input of command2