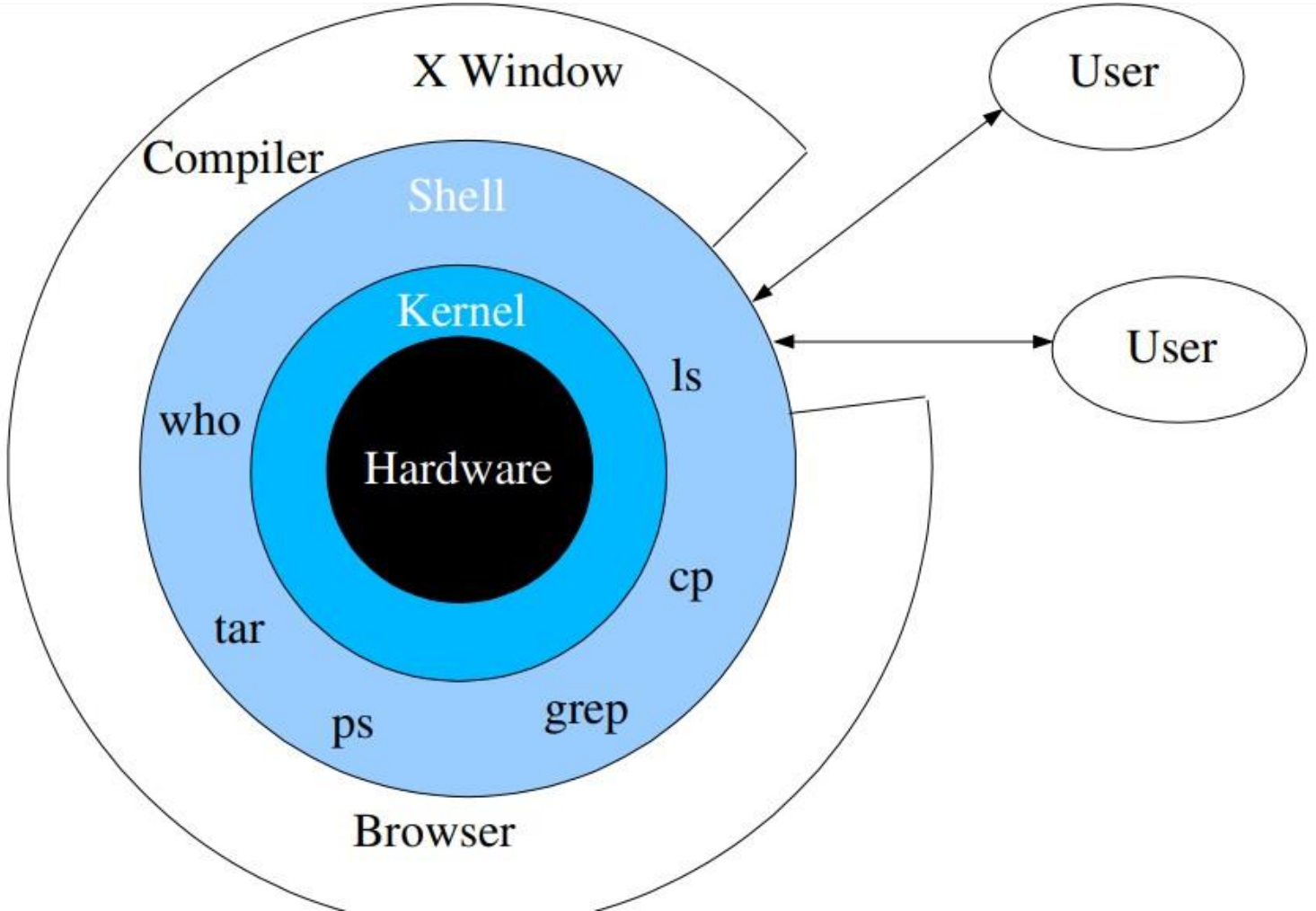
What is a shell?

- The user interface to the operating system
- Functionality:
  - Execute other programs
  - Manage files
  - Manage processes
- Full programming language
- A program like any other
  - This is why there are so many shells



Most Commonly Used Shells

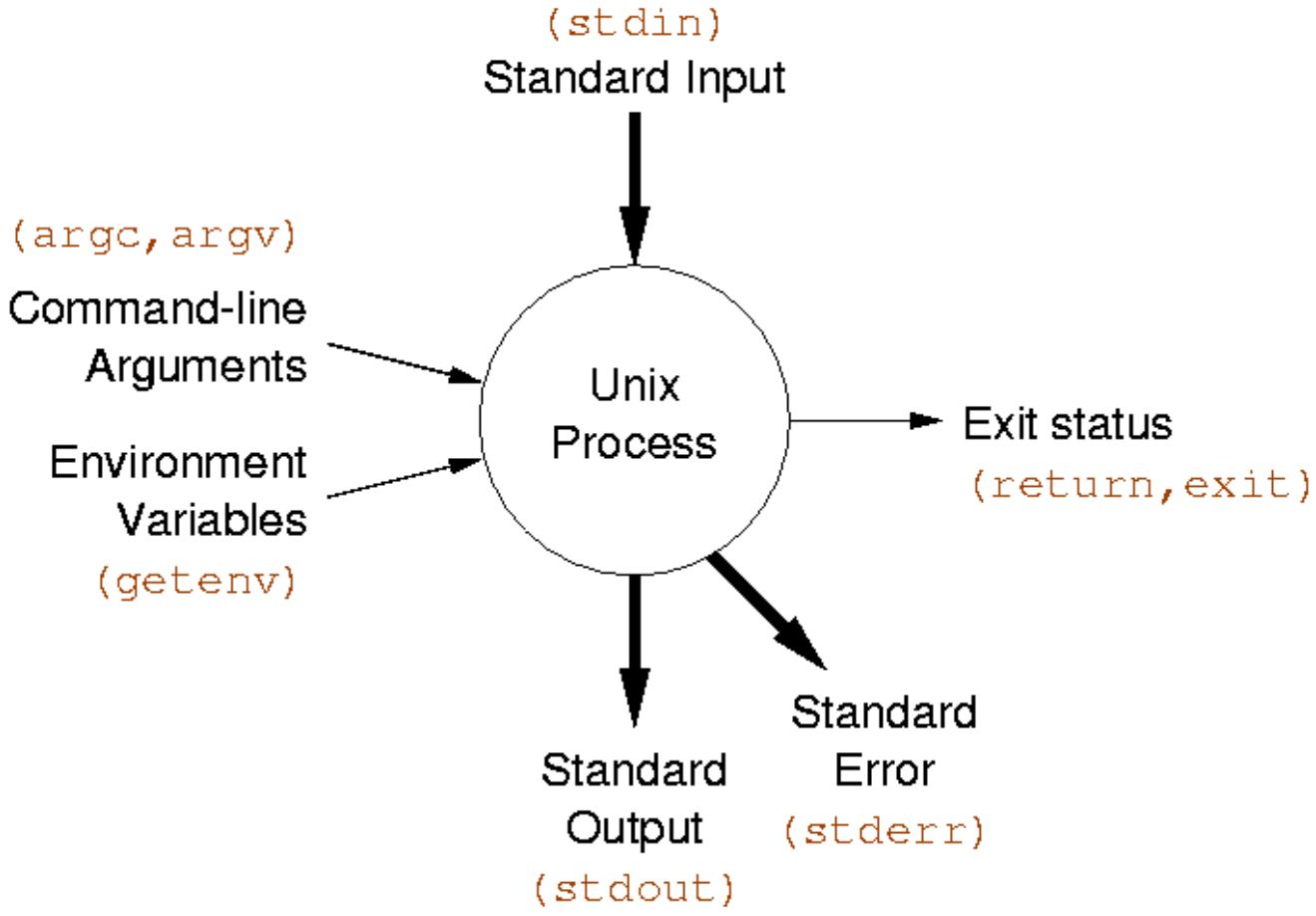
`/bin/csh`	C shell
`/bin/tcsh`	Enhanced C Shell
`/bin/sh`	**The Bourne Shell / POSIX shell**
`/bin/ksh`	Korn shell
`/bin/bash`	Korn shell clone, from GNU

Ways to use the shell

- **Interactively**
  - When you log in, you interactively use the shell
- **Scripting**
  - A set of shell commands that constitute an executable *program*

Review: UNIX Program

- **Means of input**:
  - Program arguments [control information]
  - Environment variables [state information]
  - Standard input [data]
- **Means of output**:
  - Return status code [control information]
  - Standard out [data]
  - Standard error [error messages]



Basic Shell Programming

- A script is a file that contains shell commands
  - data structure: variables
  - control structure: sequence, decision, loop
- Shebang line for bash shell script:

```
#! /bin/sh
```

- to run:
  - make executable: `% chmod +x script`
  - invoke via: `% ./script`

The *read* Statement

- Use to get input (data from user) from keyboard and store (data) to variable.
  Syntax:

  ```
  read variable1, variable2,...variableN
  ```

- Example: Write a shell script to
  - first ask user, name
  - then waits to enter name from the user via keyboard.
  - Then user enters name from keyboard (after giving name you have to press ENTER key)
  - entered name through keyboard is stored (assigned) to variable fname.

  Solution is in the next slide

Example (***read*** **Statement**)

$ vi sayH

```
#
#Script to read your name from key-board #
echo "Your first name please:"
read fname
echo "Hello $fname, Lets be friend!"
```

- Run it as follows:
  $ ***chmod*** **+x sayH**
  $ **./sayH**
  Your first name please: ***vivek***
  Hello vivek, Lets be friend!

Wild Cards

Wild card /Shorthand	Meaning	Examples	
*	Matches any string or group of characters.	**$ ls ***	will show all files
		$ ls a*	will show all files whose first name is starting with letter 'a'
		$ ls *.c	will show all files having extension .c
		$ ls ut*.c	will show all files having extension .c but file name must begin with 'ut'.
?	Matches any single character.	**$ ls ?**	will show all files whose names are 1 character long
		$ ls fo?	will show all files whose names are 3 character long and file name begin with fo
[...]	Matches any one of the enclosed characters	**$ ls [abc]***	will show all files beginning with letters a,b,c

More commands on one command line

- *Syntax:*
  `command1 ; command2`
  To run two command in one command line.
- *Examples:*

- **$ *`date ; who`***
  Will print today's date followed by users who are currently login.

- Note that You can't use
  $ *`date who`*
  for same purpose, you must put semicolon
  - in between the ***`date`*** and ***`who`*** command.

Command Line Arguments

1. Telling the command/utility
   - which option to use.
2. Informing the utility/command
   - which file or group of files to process

- Let's take ***rm*** command,
  - is used to remove file,
  - which of the file?
  - how to tail this to ***rm*** command
    - ***rm*** command does not ask the name of the file
  - So what we do is to write command as follows:

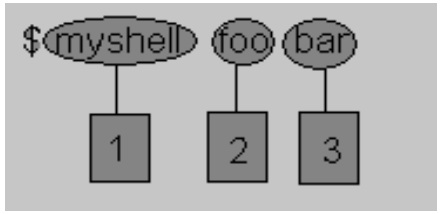
- **$ *rm* {file-name}**

rm : is the command

file-name :file to remove

Arguments - Specification

$ myshell foo bar



Shell Script name i.e. **myshell**
First command line argument passed to **myshell** i.e. ***foo***
Second command line argument passed to **myshell** i.e. **bar**

In shell if we wish to refer this command line argument we refer above as follows

```
myshell it is $0
foo it is $1
bar it is $2
```

- Here **$#** (built in shell variable) will be 2 (Since ***foo*** and **bar** only two Arguments),
- Please note at a time such 9 arguments can be used from $1..$9,
- You can also refer all of them by using $* (which expand to `$1,$2...$9`).
- Note that $1..$9 i.e command line arguments to shell script is know as "*positional parameters*".

Arguments - Example

```
$ vi demo  #!/bin/sh   #
  # Script that demos, command line args
  #
  echo "Total number of command line argument are $#"
  echo "$0 is script name"  echo "$1 is first
  argument"  echo "$2 is second argument"
  echo "All of them are :- $* or $@"
```

- Run it as follows
- Set execute permission as follows:

```
$ chmod 755 demo
```

- Run it & test it as follows:

```
$ ./demo Hello World
Total number of command line argument are 2
Demo is script name  Hello first argument
World is second argument
```

Redirection of Input/Output

- In Linux (and in other OSs also)
  - it's possible to send output to file
  - or to read input from a file
- For e.g.
  $ ***ls*** command gives output to screen;
- to send output to file of ls command give command
  ls > filename
  It means put output of ***ls*** command to filename.

redirection symbols: ‘>’

- There are three main redirection symbols: **>,>>,<**

(1) **>** Redirector Symbol
Syntax:
`Linux-command > filename`

- To output Linux-commands result to file.
  - Note that if the file already exist,
    - it will be overwritten
  - else a new file will be created.

- For e.g.
- To send output of ***`ls`*** command give
  $ *`ls`* > `myfiles`

- if '**`myfiles`**' exist in your current directory
  - it will be overwritten without any warning.

redirection symbols: '>>'

(2) **>>** Redirector Symbol

Syntax:

```
Linux-command >> filename
```

- To output Linux-commands result
  - to the END of the file.

- if file exist:
  - it will be opened
  - new information/data will be written to the **END** of the file,
  - without losing previous information/data,
- if file does not exist, a new file is created.
- For e.g.
- To send output of date command
  - to already exist file give command

```
$ date >> myfiles
```

redirection symbols: ‘<’

(3) **<** Redirector Symbol

Syntax:

```
Linux-command < filename
```

- To provide input to Linux-command
  - from the file instead of standard input (key-board).

- For e.g. To take input for cat command give

```
$ cat < myfilesstandard
```

Pipes

- A pipe is a way
  - to connect the output of one program
  - to the input of another program
    - without any temporary file.
- Definition
  - "*A pipe is nothing but a temporary storage place*
    - *where the output of one command*
    - *is stored and then passed*
      - *as the input for second command.*
    - *Pipes are used*
      - *to run more than two commands*
        - *Multiple commands*
      - *from same command line.*"
- *Syntax:*
  `command1 | command2`

Pipe - Examples

Command using Pipes	Meaning or Use of Pipes
$ ***ls*** **|** ***more***	Output of ***ls*** command is given as input to the command ***more*** So output is printed one screen full page at a time.
$ ***who*** **|** ***sort***	Output of ***who*** command is given as input to sort command So it will print sorted list of users
$ ***who*** **|** ***sort*** **> user_list**	Same as above except output of sort is send to (redirected) the file named user_list
$ ***who*** **|** ***wc*** **-l**	***who*** command provides the input of ***wc*** command So it will count the users logged in.
$ ***ls*** **-l |** ***wc*** **-l**	***ls*** command provides the input of ***wc*** command So it will count files in current directory.
$ ***who*** **|** ***grep*** **raju**	Output of ***who*** command is given as input to ***grep*** command So it will print if particular user is logged in. Otherwise nothing is printed

Filter

- Accepting the input
  - from the standard input
- and producing the result
  - on the standard output
- A filter
  - performs some kind of process on the input
  - and provides output.

grep

- The grep command searches through one or more files for lines containing a target and then prints all of the matching lines it finds. For example, the following command prints all lines in the file mtg_note that contain the word “room”:

$ grep room mtg_note

will be held at 2:00 in room 1J303. We will discuss

Note that you specify the target as the first argument and follow it with the names of the files to search. Think of the command as “search for target in file.” The target can be a phrase-that is, two or more words separated by spaces.

$ grep "boxing wizards" pangrams

*$ grep vacation * mbox*

Filter: Examples

- Consider one more following example
  $ *sort* < sname | *uniq* > u_sname
- Here ***uniq*** is filter
  - takes its input from ***sort*** command
  - and redirects to "u_sname" file.

Processing in Background

- **Use ampersand (&)**
  - **at the end of command**
- **To start the execution in background**
  - **and enable the user to continue his/her processing**
  - **during the execution of the command**
    - **without interrupting**
- **`$ ls / -R | wc -l`**
- This command will take lot of time
  - to search all files on your system.

- So you can run such commands
  - in Background or simultaneously
  - by adding the ampersand (&):
- **`$ ls / -R | wc -l&`**

Commands Related With Processes

For this purpose	Use this Command	Examples
To see currently running process	`ps`	`$ ps`
To stop any process by PID i.e. to kill process	`kill {PID}`	`$ kill 1012`
To stop processes by name i.e. to kill process	`killall {Proc-name}`	`$ killall httpd`
To get information about all running process	`ps -ag`	`$ ps -ag`
To stop all process except your shell	`kill 0`	`$ kill 0`
For background processing (With &, use to put particular command and program in background)	`linux-command &`	`$ ls / -R | wc -l &`
To display the owner of the processes along with the processes	`ps aux`	`$ ps aux`
To see if a particular process is running or not. For this purpose you have to use ps command in combination with the grep command	`ps ax | grep {Proc-name}`	For e.g. you want to see whether Apache web server process is running or not then give command **`$ ps ax | grep httpd`**
To see currently running processes and other information like memory and CPU usage with real time updates.	`top`	`$ top` Note that to exit from top command press q.
To display a tree of processes	`pstree`	`$ pstree`

if condition

- if condition
  - used for making decisions in shell script,
  - If the condition is true
  - then command1 is executed.

- *Syntax:*

```
if condition then command1 if condition is true or if exit status of
  condition is 0 (zero) ... ... fi
```

- condition
- is defined as:
  "*Condition is nothing but comparison between two values.*"

    - For compression
      - you can use test
  - or [expr] statements
  - or even exist status

if condition - Examples

- $ *cat* > showfile
  #!/bin/sh
  #
  #Script to print file #
  if cat $1
  then
  echo -e "\n\nFile $1, found and successfully echoed"
  fi

- Run above script as:
  $ *chmod* 755 showfile
  $./showfile foo

- Shell script name is: **showfile** (**$0**)
- The argument is **foo** (**$1**).
- Then shell compare it as follows:
  if cat **$1** :is expanded to ***if cat*** **foo**.

Example: Detailed explanation

- if ***cat*** command finds foo file

  and if its successfully shown on screen,

  - it means our ***cat*** command
    - is successful and
    - its exist status is 0 (indicates success),
  - So our if condition is also true
    - the statement ***echo*** **-e "\n\nFile $1, found and successfully echoed"**
    - is proceed by shell.

- if cat command is not successful
  - then it returns non-zero value
    - indicates some sort of failure
  - the statement ***echo*** **-e** "\n\nFile $1, found and successfully echoed"
    - is skipped by our shell.

test **command** `or` **`[ expr ]`**

- ***test*** command or **`[ expr ]`**
  - is used to see if an expression is true,
  - and if it is true it return zero(0),
  - otherwise returns nonzero for false.

  - *Syntax:*
    `test` `expression` or **`[`** `expression` **`]`**

test command - Example

- determine whether given argument number is positive.
- $ ***cat*** **> ispostive**

```
#!/bin/sh
#
# Script to see whether argument is positive
#
if test $1 -gt 0
then
echo "$1 number is positive" fi
```

- Run it as follows

  $ *chmod* 755 ispostive

- **$ispostive 5**

  5 number is positive

- **$ispostive -45**

  Nothing is printed

Mathematical Operators

Mathematical Operator in Shell Script	Meaning	Normal Arithmetical/ Mathematical Statements	But in Shell	
			For test statement with if command	For [expr] statement with if command
-eq	is equal to	5 == 6	if test 5 -eq 6	if [5 -eq 6]
-ne	is not equal to	5 != 6	if test 5 -ne 6	if [5 -ne 6]
-lt	is less than	5 < 6	if test 5 -lt 6	if [5 -lt 6]
-le	is less than or equal to	5 <= 6	if test 5 -le 6	if [5 -le 6]
-gt	is greater than	5 > 6	if test 5 -gt 6	if [5 -gt 6]
-ge	is greater than or equal to	5 >= 6	if test 5 -ge 6	if [5 -ge 6]

String Operators

Operator	Meaning
string1 = string2	string1 is equal to string2
string1 != string2	string1 is NOT equal to string2
string1	string1 is NOT NULL or not defined
-n string1	string1 is NOT NULL and does exist
-z string1	string1 is NULL and does exist

File and Directory Operators

Test	Meaning
-s file	Non empty file
-f file	File exists or is a normal file and not a directory
-d dir	Directory exists and not a file
-w file	file is a writeable file
-r file	file is a read-only file
-x file	file is executable

Logical Operators

Operator	Meaning
! expression	Logical NOT
expression1 -a expression2	Logical AND
expression1 -o expression2	Logical OR

if…else…fi

- If given condition is true
- then command1 is executed
- otherwise command2 is executed.

- *Syntax:*

if condition

then

condition is zero (true - 0)

execute all commands up to else statement

else

if condition is not true then

execute all commands up to fi

fi

if...else...fi -Example

```
$ vi isnump_n
#!/bin/sh
#
# Script to see whether argument is
positive or negative
#
if [ $# -eq 0 ]
then
  echo "$0 : You must give/supply one
  integers"
  exit 1
fi
if test $1 -gt 0
then
  echo "$1 number is positive"
  else echo "$1 number is negative"
fi
```

Try it as follows:
$ chmod 755 isnump_n

$ isnump_n 5
5 number is positive

$ isnump_n -45
-45 number is negative

$ isnump_n
./ispos_n : You must give/supply one integers

$ isnump_n 0
0 number is negative

Loops in Shell Scripts

- Bash supports:
  - for loop
  - while loop
- **Note** that in each and every loop,

(a) First, the variable used in loop condition

- must be initialized,
- then execution of the loop begins.

(b) A test (condition) is made

at the beginning of each iteration.

(c) The body of loop ends

with a statement modifies

the value of the test (condition) variable.

for Loop

- *Syntax:*

```
for { variable name } in { list }
do
  execute one for each item in
  the list until the list is not
  finished
  (And repeat all statements between do and
  done)
done
```

for Loop: Example

Example:

```
$ cat > testfor
for i in 1 2 3 4 5
do

        echo "Welcome $i times"

done
```

Run it above script as follows:

```
$ chmod +x testfor
$ ./testfor
```

- The for loop first creates i variable
  - and assigned a number to i from the list of numbers 1 to 5,
- The shell executes echo statement for each assignment of i.
- This process will continue until all the items in the list were not finished,
- because of this it will repeat 5 echo statements.

while Loop

- *Syntax:*

```
while [ condition ]
do
  command1
  command2
  ..
  ....
done
```

Example:

```
while [ $i -le 10 ]
do
  echo "$n * $i = `expr $i \* $n`"
  i=`expr $i + 1`
done
```

- #!/bin/bash.
- echo -n "Enter the first **number** : "
- read num1.
- echo -n "Enter the second **number** : "
- read num2.
- **sum**=`expr $num1 + $num2`
- echo "**sum** of **two** value is $**sum**"

Exercise problems

1. Write Script to see current date, time, username, and current directory
2. How to write shell script that will add two numbers, which are supplied as command line argument, and if this two numbers are not given show error and its usage.
3. Write Script to find out biggest number from given three nos. Numbers are supplied as command line argument. Print error if sufficient arguments are not supplied.
4. Write script to print the following numbers as 5,4,3,2,1 using while loop.
5. Write Script, using case statement to perform basic math operation as follows: + addition, - subtraction, x multiplication, / division
6. Write script to print given number in reverse order, for eg. If no is 123 it must print as 321.
7. Write script to print given numbers sum of all digit, For eg. If no is 123 it's sum of all digit will be 1+2+3 = 6.
8. 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

Exercise problems

9. Write a shell script takes the name a path (eg: /afs/andrew/course/15/123/handin), and counts all the sub directories (recursively).

10.Write a shell script that takes a name of a folder as a command line argument, and produce a file that contains the names of all sub folders with size 0 (that is empty sub folders)

11. Write a shell script that takes a name of a folder, and delete all sub folders of size 0

12.write a shell script that will take an input file and remove identical lines (or duplicate lines from the file)