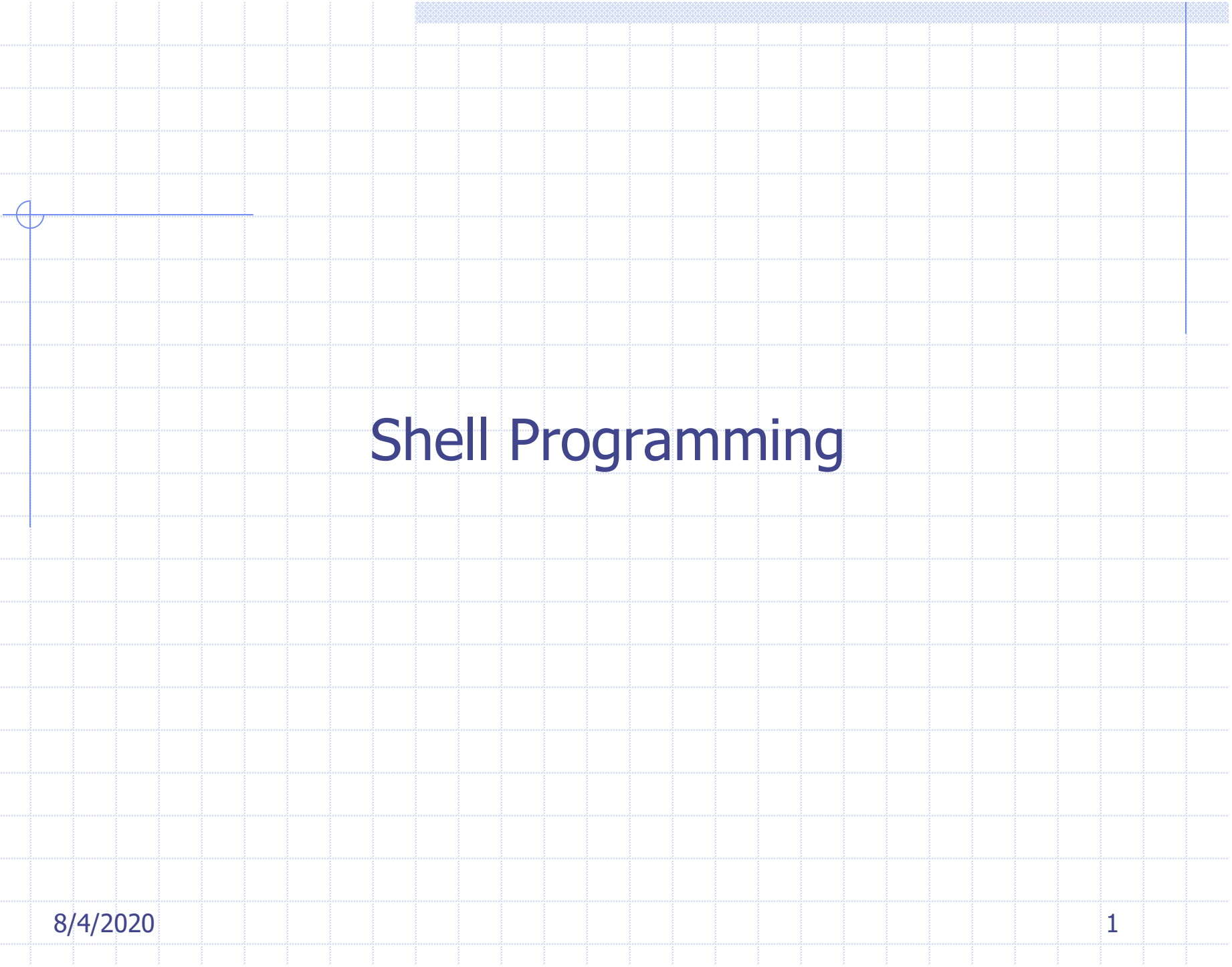
Shell Programming

Shell Commands

- General Purpose Utilities
  - cal
  - date
  - echo
  - bc
  - who
  - uname

- File System Commands
  - pwd
  - cd
  - mkdir
  - rmdir
  - ls
- Handling Ordinary Files
  - cat
  - cp
  - rm
  - mv
  - wc

- Basic File Attribute
  - chmod
  - Find
- Shell
  - tee
  - touch

- Simple Filters
  - head
  - tail
  - cut
  - sort
  - uniq
- Filters using Regular Expression
  - grep
  - sed

grep: Searching for a Pattern

grep options pattern filename(s)

Ex:

grep “sales” emp.lst

Options	Significance
-i	Ignores case for matching
-v	Doesn't display lines matching expression
-n	Displays line nos along with lines
-c	Displays count of no of occurences
-l	Displays list of filenames only
-e	Specifies expression with this option. Can be used multiple times.
-f	Takes pattern from file, one per line.
-E	Treats pattern as Extented Regular Expression (ERE)
-x	Matches pattern with entire line.

Wild-Cards

Wild-Cards	Matches
*	Any number of characters including none
?	A single character
[ijk]	A single character – either i, j, or k.
[x-z]	A single character within the ASCII range of x and z
[!ijk]	A single character that is not an i, j, or k
[!x-z]	A single character that is not within the ASCII range of x and z

Symbols or Expression	Matches
*	Zero or More occurrences of the previous character
g*	Nothing or g, gg, ggg etc
.	A Single character
.*	Nothing or any number of Characters
[pqr]	A single character p, q or r
[c1-c2]	A character within ASCII range represented by c1 & c2
[1-3]	A digit between 1-3
[^pqr]	A single character which is not p, q or r
[^a-zA-Z]	Non-alphabetic character
^pat	Pattern pat at beginning of line
pat$	Pattern pat at end of line
^bash$	Bash as the only word in the line
^$	Lines containing nothing

Shell Programming

- A Shell Program runs in Interpretive Mode. It is not compiled to a separate executable file as a C program is. Each statement is loaded into memory when it is to be executed.

- When a group of command have to be executed regularly the should be stored in a file, and the file itself is executed as a shell script or a shell program.

Example

```
#! /bin/sh
# Sample Shell Script
echo “Today’s Date: date”
echo “This Month’s Calendar:”
cal `date “+%m 20%y”`
echo “My Shell: $SHELL”
```

 # - Comment Character:

The Shell ignores all characters placed on its right.

Contd...

- Shell scripts are executed in a separate child shell process. This is done by providing special interpreter line at the beginning (starting with #!).
- To run the script we make it executable and then invoke the script name.

  $ chmod +x script.sh or $ chmod 755 script.sh
  $ script.sh

- User can explicitly spawn a child shell of his choice with the script name as argument. In this case it is not mandatory to include the interpreter line.

read: Making Scripts Interactive

- The read statement is the shell's internal tool for taking inputs from the user, i.e., making scripts interactive.
- It is used with one or more variable. When we use a statement like

  read name

  the script pauses at that point to take i/p from the keyboard. Since this is the form of assignment, no $ is used before name.

Example

```
#!/bin/sh
echo “Enter the pattern to be searched: \c”
read pname
echo “Enter the file to be used: \c”
read flname
echo “Searching for $pname from file $flname”
grep “$pname” $flname
echo “Selected records shown above”
```

Using Command-Line Arguments

- The shell script accepts arguments from the command line. The first argument is read by shell into parameter $1, second into $2, and so on.
- Special Parameter used by Shell:

$* - It stores complete set of positional parameters as a single string.

$# - It is set to number of arguments specified. Used to check whether right number of argument have been entered.

$0 - Holds the command name itself.

"$@" - Each Quoted string treated as a separate argument.

$? - Exit Status of Last Command.

Example

```
#!/bin/sh
echo “Program: $0”
The number of arguments specified is
   $#
The arguments are $*”
grep “$1” $2
echo “\n Job Over”
```

exit and EXIT STATUS OF COMMAND

exit - Command to terminate a program.

This command is generally run with numeric arguments.

exit0	-	When everything went fine
exit1	-	When something went wrong
exit2	-	Failure in opening a file.

Example:

```
$ grep "director" emp.lst >/home/vishal; echo$?
```

"All command return an exit Status"

Logical Operators && and || - Conditional Execution

- *Cmd1 && Cmd2*: The Cmd2 will execute only when Cmd1 is succeeds.
- *Cmd1 // Cmd2*: The Cmd2 will execute only when Cmd1 is fails.

Example:

$ grep “director” emp.lst >/home/vishal && echo “Pattern found in file”

$ grep “manager” emp.lst >/home/vishal || echo “Pattern not found in file”

THE if CONDITIONAL

Form 1	Form 2	Form 3
if command is successful then execute command else execute command fi	if command is successful then execute command fi	if command is successful then execute command elif command is successful then …. else ….. fi

if command is successful ; then

Using **test** and **[]** to evaluate expressions

- When we use if to evaluate expressions, we require the **test** statement because the true or false values returned by expressions cant be directly handled by **if**.
- **test** use certain operator to evaluate the condition on its right and returns either true or false exit status, which is then used by if for making decisions.

test works in 3 ways:

- Compare two numbers
- Compare two strings or a single string for a null value
- Check a file attribute

Numeric Comparison

- The numeric comparison operators always begin with a – (hyphen), followed by two character word, and enclosed either side by a whitespace.

Example:

```
$ x=5; y=7; z=7.2
$test $x –eq $y; echo $?
1
$test $z –eq $y; echo $?
0
```

- Numeric comparison is restricted to integers only.
- Operators: -eq, -ne, -gt, -ge, -lt, -le.

Example

```
#!/bin/sh
if test $# –ne 2; then
        echo “You did not enter 2 arguments”
else
        grep “$1” $2 || echo “$1 not found in $2”
fi
```

Shorthand for test

test $x –eq $y is same as [$x –eq $y]

String Comparison

- Another set of operator is used for string comparison.

String tests used by test

Test	True if
S1 = S2	String S1 is equal to String S2
S1 != S2	String S1 is not equal to String S2
-n stg	String stg is not a null String
-z stg	String stg is a null String
stg	String stg is assigned and not a null String

Example

```
#!/bin/sh
if [ $# –eq 0 ] ; then
echo “Enter the string to be searched: \c” ; read pname
if  [ -z “$pname” ] ; then
echo “You have not entered the string” ; exit 1
fi
echo “Enter the file to be used: \c” ; read flname
if  [ ! -n “$flname” ] ; then
echo “You have not entered the filename” ; exit 2
fi
demo.sh “$pname” “$flname”
else
demo.sh $*
fi
```

Contd…

test also permits the checking of more than one condition in the same line using the –a (AND) and –o(OR) operators.

Example:

```
if  [ -n “$pname” –a -n “$flname” ] ; then
demo.sh “$pname” “$flname”
else
echo “ At least one input was null string ” ; exit 1
fi
```

File Tests

test can be used to test various file attributes like its type or permissions.

Example:

```
#!/bin/sh
if [ ! -e $1 ] ; then
        echo " File does not exists "
elif [ ! -r $1 ] ; then
        echo " File is not readable "
elif [ ! -w $1 ] ; then
        echo " File is not writable "
else
        echo " File is both readable and writable "
fi
```

Contd…

File – Related Tests with test

Test	true if
-f *file*	*file* exists and is a regular file
-r *file*	*file* exists and is a readable
-w *file*	*file* exists and is a writable
-x *file*	*file* exists and is a executable
-d *file*	*file* exists and is a directory
-s *file*	*file* exists and has a size greater than zero

THE case CONDITIONAL

case statement matches an expression or string for more than one alternative, in more efficient manner than if.

Form:

```
case expr in
        pattern 1) cmd1 ;;
        pattern 2) cmd2 ;;
        pattern 3) cmd3 ;;
esac
```

Example

```
#!/bin/sh
echo”                MENU\n
1.List of files\n2. Processes of user\n3. Today’s Date\n4. Users of System\n5. Quit to UNIX\n Enter your option: \c”
read choice
case “$choice” in
1) ls –l ;;
2) ps –f ;;
3) date ;;
4) who ;;
5) exit ;;
*) echo “ Invalid Option”
esac
```

The last option (*) matches any option not matched by the previous options.

Contd…

It is very effective when string is fetched by command substitution.

Example:

```
case `date | cut –d " " –f1` in
Mon) Command1;;
        Tue) Command2 ;;
Wed) Command3 ;;
        *) ;;
esac
```

Matching Multiple Patterns:

```
echo " do you wish to continue? (y/n) \c "
read answer
case "$answer" in
y|Y) ;;
n|N) exit;;
esac
```

Contd…

Matching the DOT

The * does not match all file beginning with the a **.** (dot) or the **/** of a pathname.

Example: $ ls .???*

The Character Class

The character class comprises of set of characters enclosed of rectangular brackets [and], but it matches a single character in a class.

Example:

```
case "$answer" in
[yY][eE]*) ;;
[nN][oO]) exit;;
        *) echo " Invalid Response"
esac
```

expr : Computation and String Handling

The shell relies on external **expr** command for computing features.

Functions of expr:

- Perform arithmetic operations on integers
- Manipulate strings

Computation

expr can perform four basic arithmetic operation as well as modulus function

Examples

```
$x=3; y=5
$expr 3 + 5
$expr $x - $y
$expr 3 \* 5
$expr 3 / 5
$expr $y % $x
```

Command substitution:

```
z =`expr $x + $y`
```

Incrementing value of a variable

```
x =`expr $x + 1`
```

String Handling

expr can perform 3 important string functions

- Determine the length of the string.
- Extract a sub-string.
- Locate the position of a character in a string

Contd…

Length of the string

$ expr "abcdefgh" : '.*'

The regular expression ".* " signifies to expr that it has to print the number of characters matching the pattern

Extracting a sub-string

Expr can extract a string enclosed by escaped character \(and \)

$ stg = 2003

$ expr "$stg" : '..\(..\)'

Locating the position of a character

$ stg = abcdefgh

$ expr "$stg" : '[^d]*d'

while loop

Format:

```
while cmd is successful
do
cmd1
cmd2
done
```

Assignments

1. Use a script to take two numbers as arguments and output their sum using

   i) bc ii) expr.

   Include error checking to test if two arguments were entered.

2. Write a shell script that uses find to look for a file and echo a suitable msg if the file is not found. You must not store the find output in a file.