ISS Lab 2

Shell scripting

Basic everyday commands (will do demos)

- ls
- cat
- echo
- cd
- chmod
- mkdir
- cp, mv, rm

Bash usages

- Automate tasks
- Server usage and submitting jobs
  - ssh
  - scp
- Lets login into the iiit students web server
- Helps to customize and interact with your system

Know your shell

- Bashrc
- Aliases
- Paths
- System variables

Quote marks in bash

- Quotes are needed whenever we assign character data containing spaces or special characters
- Double quotes can resolve variables
  - Let lol = 5
  - echo “$lol”
  - 5
- whereas single quotes can not
  - Let lol = 5
  - echo ‘$lol’
  - $lol
- Back quote (`) is used for command substitution

Variables

- 2 types of variables
  - Environment variables
  - Local variables
- Environment variables are set by system and are used globally across commands and scripts
- Example
  - HOME
  - PATH
- Local variables are normal variables specific to a particular script

The bc program

bc = interactive calculator

Printf “`sin(%g)=%12.5e\n" $r $s`

```
-h, --help
       Print the usage and exit.

-i, --interactive
       Force interactive mode.

-l, --mathlib
       Define the standard math library.

-w, --warn
       Give warnings for extensions to POSIX bc.

-s, --standard
       Process exactly the POSIX bc language.

-q, --quiet
       Do not print the normal GNU bc welcome.

-v, --version
       Print the version number and copyright and quit.
```

Parsing command line arguments

```
# read variables from the command line, one by one:
while [ $# -gt 0 ]  # $# = no of command-line args.
do
    option = $1; # load command-line arg into option
    shift;       # eat currently first command-line arg
    case "$option" in
        -m)
            m=$1; shift; ;;  # load next command-line arg
        -b)
            b=$1; shift; ;;
        ...
        *)
            echo "$0: invalid option \"$option\""; exit ;;
    esac
done
```

Using if-else

```
if [ "$option" == "-m" ]; then
    m=$1; shift;  # load next command-line arg
elif [ "$option" == "-b" ]; then
    b=$1; shift;
else
    echo "$0: invalid option \"$option\""; exit
fi
```

For loop

The for element in list construction:

```
files=‘/bin/ls *.tmp‘
# we use /bin/ls in case ls is aliased

for file in $files
do
  echo removing $file
  rm -f $file
done
```

Traverse command-line arguments:

```
for arg; do
  # do something with $arg
done

# or full syntax; command-line args are stored in $@
for arg in $@; do
  # do something with $arg
done
```

C style for loops

- `arithmetic expressions must appear inside (( ))`
-

```
declare -i i
for ((i=0; i<$n; i++)); do
  echo $c
done
```

Functions

```
function calc() {
    echo "
    if ( $1 >= 0.0 ) {
        ($1)^5*e(-($1))
    } else {
        0.0
    } " | bc -l
}
```

Grep and regex

- grep = global regular expression print
- Used to match regular expression in the given input
- Rest in demo

```
NAME
       grep, egrep, fgrep, rgrep - print lines matching a pattern

SYNOPSIS
       grep [OPTIONS] PATTERN [FILE...]
       grep [OPTIONS] -e PATTERN ... [FILE...]
       grep [OPTIONS] -f FILE ... [FILE...]

DESCRIPTION
       grep  searches  for PATTERN in each FILE.  A FILE of “-” stands for standard
       input.  If  no  FILE  is  given,  recursive  searches  examine  the  working
       directory,  and nonrecursive searches read standard input.  By default, grep
       prints the matching lines.

       In addition, the variant programs egrep, fgrep and rgrep  are  the  same  as
       grep -E, grep -F, and grep -r, respectively.  These variants are deprecated,
       but are provided for backward compatibility.
```

Sed (Stream editor)

- sed can be used at the command-line, or within a shell script, to edit a file non-interactively. most useful feature is to do a 'search-and-replace' for one string to another.
- Examples v
  - `sed -e 's/input/output/' my_file`
  - `sed -e 's/input/output/g' my_file`
- Rest in demo

```
NAME
       sed - stream editor for filtering and transforming text

SYNOPSIS
       sed [OPTION]... {script-only-if-no-other-script} [input-file]...

DESCRIPTION
       Sed  is  a  stream  editor.   A  stream editor is used to perform basic text
       transformations on an input stream (a file or input from a pipeline).  While
       in some ways similar to an editor which permits scripted edits (such as ed),
       sed works by making only one pass over the  input(s),  and  is  consequently
       more  efficient.  But it is sed's ability to filter text in a pipeline which
       particularly distinguishes it from other types of editors.

       -n, --quiet, --silent

              suppress automatic printing of pattern space

       -e script, --expression=script

              add the script to the commands to be executed

       -f script-file, --file=script-file

              add the contents of script-file to the commands to be executed

       --follow-symlinks

              follow symlinks when processing in place

       -i[SUFFIX], --in-place[=SUFFIX]

              edit files in place (makes backup if SUFFIX supplied)
```

awk

- Its a language in itself used for pattern recognition
- [see here]
- Rest in demo

```
Usage: awk [POSIX or GNU style options] -f progfile [--] file ...
Usage: awk [POSIX or GNU style options] [--] 'program' file ...
POSIX options:          GNU long options: (standard)
        -f progfile             --file=progfile
        -F fs                   --field-separator=fs
        -v var=val              --assign=var=val
Short options:          GNU long options: (extensions)
        -b                      --characters-as-bytes
        -c                      --traditional
        -C                      --copyright
        -d[file]                --dump-variables[=file]
        -D[file]                --debug[=file]
        -e 'program-text'       --source='program-text'
        -E file                 --exec=file
        -g                      --gen-pot
        -h                      --help
        -i includefile          --include=includefile
        -l library              --load=library
        -L[fatal|invalid]       --lint[=fatal|invalid]
        -M                      --bignum
        -N                      --use-lc-numeric
        -n                      --non-decimal-data
        -o[file]                --pretty-print[=file]
        -O                      --optimize
        -p[file]                --profile[=file]
        -P                      --posix
        -r                      --re-interval
        -S                      --sandbox
        -t                      --lint-old
        -V                      --version

To report bugs, see node `Bugs' in `gawk.info', which is
section `Reporting Problems and Bugs' in the printed version.

gawk is a pattern scanning and processing language.
By default it reads standard input and writes standard output.

Examples:
        gawk '{ sum += $1 }; END { print sum }' file
        gawk -F: '{ print $1 }' /etc/passwd
```

Thank You !