Unix Lab Study Material

1a. Write a shell script that takes a valid directory name as a argument recursively descend all the sub-directors, find the maximum length of any file in that hierarchy and write the maximum value to the standard output.

```
if [ $# -eq 1 ]
then
    if [ -e $1 ]
    then
        largeFile=$(find $1 -printf '%k %p\n' | sort -nr | head -n 1 | cut -d "
        echo "Larget File in the directory: $largeFile"

    else
        echo "Path does not exist! Please check the path."
        exit 0
    fi
else
    echo "This script takes only one valid directory name as an arguement!"
fi
```

COMMAND DETAILS:

> *find* : find command searches for a file in directory hierarchy
>
> > %k: amount of disk space used by the file in 1KB Blocks
> > %p: file's name
>
> *sort* : sorts the lines of text files
>
> > -n : numeric sort
> > -r : reverse sort
>
> *head* : output from part of the file
>
> > -n : number of lines to be printed starting from 1
>
> *cut* : removes sections from each line of the file
>
> > -d : delimiting character at which the line has to be split
> > -f : print n or nth field(s) in the result after cut

1b. Write a shell script that accepts a path name and creates all the components in that path name as directories. For example, if the script is named as mpc,

then the command mpc a/b/c/d should create sub-directories a, a/b, a/b/c, a/b/c/d

```
if [ $# -eq 1 ]
then
    mkdir -pv $1
else
    echo "This Script is only programmed to take one arguement as an input!"
fi
```

COMMAND DETAILS:

> *mkdir* : make directories
>
> > -p : make parent directories in the given path, if not existing
> > -v : print a message for each directory created (verbose output)

2a. Write a shell script that accepts two filenames as arguments, checks if the permissions for these files are identical and if the permissions are identical, output common permissions otherwise output each filename followed by its permissions.

```
if [ $# -eq 2 ]
then

    PERM1=$(stat --printf=”%a”  $1)
    PERM2=$(stat --printf=”%a”  $2)

    if [ "$PERM1" = "$PERM2" ]
    then
        echo Both the files have same permissions: $(stat --printf="%A" $1)
    else
        echo The given files have different permissions
    fi
else
    echo "This script is programmed to use two files in order to compare their p

fi
```

COMMAND DETAILS:

> *stat* : display file or filesystem status
>
> > %a : access rights in octal
> > %A : access rights in Human Readable Format

2b. Write a shell script which accepts valid log-in names as arguments and prints their corresponding home directories, if no arguments are specified, print a suitable error message.

```
if [ $# -eq 0 ]
then
    echo "Run this script with one or more  username(s) as arguement!"
else
    for i in $*
    do
        if [ $(grep $i /etc/passwd) ]
        then
            echo "Home directory for $i: "
            eval "echo ~$i"
        else
            echo "user does not exsist"
        fi
    done
fi
```

COMMAND DETAILS:

> *grep* : print lines that match a given pattern
>
> *eval* : command used construct commands by concatenating arguments
>
> **~** : Tilde Operator, used to expand a given user's home directory

3a. Create a script file called file properties that reads a filename entered and outputs it properties.

```
if [ $# -ne 1 ]
then
    echo "Run this cript with only one filename as arguement!"
else
    if [ -f "$1" ]
    then
        echo "Name : $1"
        echo "Permissions : $(stat --format="%A" $1)"
        echo "Type: $(stat --format="%F" $1)"
        echo "Owner: $(stat --format="%U" $1)"
        echo "Group: $(stat --format="%G" $1)"
        echo "Size: $(stat --format="%s" $1)"
    else
        echo "File does not exsist"
    fi
fi
```

COMMAND DETAILS:

> *stat* : display file or filesystem status
>
> > %A : access rights in Human readable format
> > %F : File type
> > %U : Owner of the File
> > %G : Group owner of the File
> > %s : total size of the file in bytes

3b. Write a shell script to implement terminal locking (Similar to the lock command). It should prompt for the user for a password. After accepting the password entered by the user, it must prompt again for the matching password as confirmation and if match occurs, it must lock the keyword until a matching password is entered again by the user. Note the Script must be written to disregard BREAK, control-D. No time limit need be implemented for the lock duration.

```
while true
do
    clear
    echo "**Password entered will not visible for security reasons**"
    echo "Enter Password: "
    read -s passFirst
    echo "Re-enter Password: "
    read -s passConfirm

    if [ "$passFirst" = "$passConfirm" ]
    then
        clear
        echo "Terminal Locked !"
        stty intr ''
        stty eof ''
        stty kill ''
        stty stop ''
        stty susp ''
        echo "To unlock, Enter Password: "
        passFirst=""
        until [ "$passFirst" = "$passConfirm" ]
        do
            read -s passFirst
        done
        stty intr '^C'
        stty eof '^D'
        stty kill '^U'
        stty stop '^S'
        stty susp '^Z'
        echo "Terminal Unlocked !"
        exit
    else
        echo "Password Mismatch !"
        sleep 3
```

```
    fi
done
```

COMMAND DETAILS:

> *read* : read from a file descriptor
>
> > -s : does not echo input coming from a terminal
>
> *stty* : change and print terminal line settings
>
> > intr : interupt,Terminates the current job (Default : "^C")
> > eof : end of file, Forced Exit (Default : "^D")
> > kill : erases the text before the cursor (Defulat : "^U")
> > stop : stops the output (Default : "^S")
> > susp : sends the current job to backgroud (Default : "^Z")

4a. Write a shell script that accept one or more file names as argument and convert all of them to uppercase, provided they exists in current directory.

```
if [ $# -eq 0 ]
then
    echo "This script requires atleast one filename as arguement"
else
    for i in $*
    do
        if [ -f $i ]
        then
            tr '[a-z]' '[A-Z]' < $i>tempFile
            mv tempFile $i
            echo "File $i has been translated."
        else
            echo "$i does not exist in the current directory"
            exit 1
        fi
    done
fi
```

COMMAND DETAILS:

> *tr* : translate or delete characters
>
> *mv* : move or rename files

4b. Write a shell script that displays all the links to a file specified as the first argument to the script. The second argument, which is optional, can be used to specify in which the search is to begin. If this second argument is not present, the search is to begin in the current working directory. In either case, the

starting directory as well as its subdirectories at all levels must be searched. The script need not include error checking.

```
if [ $# -eq 0 ]
then
    printf "Invalid arguments"
else
    if [ $# -eq 1 ]
    then
        dir=`pwd`
    elif [ $# -eq 2 ]
    then
        dir=$2
    fi

if [ -f $1 ]
then
    inode=`ls -i $1 | cut -d " " -f 1`
    printf "hard link of $1 are:\n"
    find $dir -inum $inode
    find $dir -type l -ls |tr -s " " |grep $1 |cut -d " " -f 11 > soft
    s=`wc -l < soft`
    if [ $S -eq 0 ]
    then
        echo “There is no soft links”
    else
        echo “soft links of $1 are”
        cat soft
    fi
else
    printf "file doesn't exist"
fi
fi
```

COMMAND DETAILS:

> *pwd* : prints working directory
>
> *ls* : list directory contents
>
> > -i : print the index number of each file
>
> *cut* : removes sections from each line of the file
>
> > -d : delimiting character at which the line has to be split
> > -f : print n or nth field(s) in the result after cut
>
> *find* : find command searches for a file in directory hierarchy
>
> > -inum : looks for file(s) with index number passwd as argument
> > -type : look for file(s) of specific type ("l" for Symbolic link)

> -ls : list the files found in output similar to `ls -dils`
>
> *tr* : translate or delete characters
>
> > -s : replace each sequence of the repeated character specified with songle occurrence of that character
>
> *grep* : print lines that match a given pattern
>
> *wc* : print newline, word, and byte counts for each file
>
> > -l : print newline counts

5a. Write a shell script that accepts filename as argument and display its creation time if file exist and if does not send output error message.

```
if [[ ! -f "$1"  ||  $# -ne 1 ]]
then
    echo "This script only accepts one valid filename as arguement!"
else
    statLAT=$(stat --printf "%x")
    lat=$(date --date="$statLAT" +"%d/%m/%Y  %I:%M %p")
    echo "FileName: $1"
    echo "Last Access Time: $lat"
fi
```

COMMAND DETAILS:

> *stat* : display file or filesystem status
>
> > %x : last access time in human readable format

5b. Write a shell script to display the calendar for the current month with current date replaced by * or depending whether the date is one digit or two digit.**

```
echo -e "Current Date: $(date +"%d/%m/%Y")\n"
currDate=$(date +"%d")
#currDate=21
if [ $currDate -le 9 ]
then
    currDate=$(echo $currDate | cut -f 2)
    echo "$(ncal | sed 's/\b'"$currDate"'\b/'*'/')"
else
    echo "$(ncal | sed 's/\b'"$currDate"'\b/'**'/')"
fi
```

COMMAND DETAILS:

> *ncal* : displays a calendar of the current month highlighting the current date

> *date* : print or set the system date and time
>
> > %d : day of the month
>
> *cut* : removes sections from each line of the file
>
> > -f : print n or nth field(s) in the result after cut
>
> *sed* : stream editor for filtering and transforming text
>
> > s/regexp/replacement/ : search and attempt to match the given regular expression against the pattern space and If search is successful, replace the portion matched with the replacement.
> > \b : matches for complete word

6a. Write s a shell script to find a file/s that matches a pattern given as command line argument in the home directory, display the contents of the file and copy the file into the directory ~/mydir.

```
if [ $# -eq 0 ]
then
    echo “No arguments”
    exit
fi
for i in $*
do
    echo grep –riew $* ~/
    ls $*
    cat $*
           cp $* ~/mydir
done
```

COMMAND DETAILS:

> *grep* : print lines that match a given pattern
>
> > -r : Read all files under each directory, recursively, following symbolic links only if they are on the command line
> > -i : Ignore case distinctions in patterns and input data -e : Use the argument as regexp pattern -w : Select only those lines containing matches that form whole words.

6b. Write a shell script to list all the files in a directory whose filename is at least 10 characters. (use expr command to check the length).

```
currentDir=$(pwd)
listOfFiles=$(ls -l "$currentDir" | awk '{print $9}')
```

```
echo "Current Directory: $currentDir"
echo "All files whose filename is at least 10 characters: "
for f in $listOfFiles
do
    if [ $(expr length "$f") -gt 10 ]
    then
        echo $f
    fi
done
```

COMMMAND DETAILS:

> *ls* : list directory contents
>
> > -l : use a long listing format
>
> *awk* : pattern scanning and processing language
>
> *expr* : evaluate expressions
>
> > length : computes the length of the argument

7a.Write a shell script that gets executed and displays the message either "Good Morning" or "Good Afternoon" or "Good Evening" depending upon time at which the user logs in.

```
currentTime=$(date +"%H")
if [[ $currentTime -ge 00 && $currentTime -le 11 ]]
then
    echo "Good Morning!"
elif [[ $currentTime -ge 12 && $currentTime -le 14 ]]
then
    echo "Good Afternoon!"
elif [[ $currentTime -ge 15 && $currentTime -le 18 ]]
then
    echo "Good Evening!"
else
    echo "Good Night!"
fi
```

COMMAND DETAILS:

> *date* : print or set the system date and time
>
> > %H : Time in 24H format

7b.Write a shell script that accepts a list of filenames as its argument, count and report occurrence of each word that is present in the first argument file on other argument files.

```
if [ $# -ne 2 ]
then
    echo "This script takes in two filenames as arguements"
    exit
else
    string1=`cat $1 | tr '\n' ' '`
    for ((i=2; i<=$#; i++))
    do
        echo "Filename: ${!i}"
        for a in $string1
        do
            echo "$a: `grep -c "$a" "${!i}"`"
        done
    done
fi
```

COMMAND DETAILS:

> *cat* : concatenate files and print on the standard output
>
> *tr* : translate or delete characters
>
> *grep* : print lines that match a given pattern
>
> > -c : Suppress normal output; instead print a count of matching lines for each input file.

8a. Write a shell script that determine the period for which as specified user is working on a system and display appropriate message.

```
if [ $# -ne 1 ]
then
    echo "This script takes one username as arguement!"
    exit
else
    if [ $(grep -c $1 /etc/passwd) -ne 0 ]
    then
        if [ $(last -Fw|grep -c $1) -ne 0 ]
        then
            loginDate=$(last -Fw |grep $1 | head -1 | tr -s " " | awk '{printf('
            loginDate=$(date -d "$loginDate" "+%s")
            currDate=$(date "+%s")
            sessionTime=$((currDate-loginDate))
            sday=$(( sessionTime/86400 ))
            shour=$(( (sessionTime-(sday*86400))/3600 ))
            smin=$(( (sessionTime-(sday*86400)-(shour*3600))/60 ))
            ssec=$(( (sessionTime-(sday*86400)-(shour*3600)-(smin*60)) ))
            printf "Active Session Time: %02d days %02d hours %02d mins %02d sec

        else
            echo "The user has no recent logins"
```

```
            exit
        fi
    else
        echo "Invalid Username!"
        exit
    fi
fi
```

COMMAND DETAILS:

> *grep* : print lines that match a given pattern
>
> > -c : Suppress normal output; instead print a count of matching lines for each input file.
>
> *last* : show a listing of last logged in users
>
> > -F : Print full login and logout times and dates
> > -w : Display full user names and domain names in the output
>
> *grep* : print lines that match a given pattern
>
> *head* : output from part of the file
>
> > -n : number of lines to be printed starting from 1
>
> *tr* : translate or delete characters
>
> *awk* : pattern scanning and processing language
>
> *date* : print or set the system date and time
>
> > %s : seconds since 1970-01-01 -d : display time described by STRING argument

8b. Write a shell script that reports the logging on of as specified user within one minute after he/she login. The script automatically terminates if specified user does not login during specified in period of time.

```
if [ $# -ne 1 ]
then
    echo "This script requires only one username as arguement"
    exit
else
    startTime=$(date -d "now" "+%s")
    until who|grep -sw "$1"
    do
        curTime=$(date -d "now" "+%s")
        if [ $(( $curTime-$startTime )) -ge 90 ]
```

```
        then
            echo "Timed Out!"
            exit
        fi
    done
    echo "User $1 logged in !"
    exit
fi
```

COMMAND DETAILS:

> *date* : print or set the system date and time
>
> > %s : seconds since 1970-01-01 -d : display time described by STRING argument
>
> *who* : show who is logged on
>
> *grep* : print lines that match a given pattern
>
> > -s : Suppress error messages about nonexistent or unreadable files -w : Select only those lines containing matches that form whole words

9a. Write a shell script that accepts the filename, starting and ending line number as an argument and display all the lines between the given line number.

```
if [ $# -ne 3 ]
then
    echo "This script requires three arguemnts! FileName,Starting Line Number ar
    exit
else
    if [ -f $1 ]
    then
        eval "sed -n $2,$3\p $1"
        exit
    else
        echo "No Such File!"
        exit
    fi
fi
```

COMMAND DETAILS:

> *sed* : stream editor for filtering and transforming text
>
> > -n : suppress automatic printing of pattern space p : print only specific lines based on the line number or pattern matches

9b. Write a shell script that folds long lines into 40 columns. Thus any line that exceeds 40 characters must be broken after 40th, a "/" is to be appended as the indication of folding and processing is to be continued with the residue. The input is to be supplied through a text file created by the user.

```
echo "Enter the filename:"
read fn
if [ ! -f "$fn" ]
then
    echo "Invalid Filename!"
    exit
fi

for line in `cat $fn`
do
    length=$(echo $line|wc -c)
    length=$(($length-1))
    s=1;e=40
    if [ $length -gt 40 ]
    then
        while [ $length -gt 40 ]
        do
            echo -e "$(echo $line| cut -c $s-$e) /"
            s=$(($e+1))
            e=$(($e+40))
            length=$(($length-40))
        done
        echo "$(echo $line | cut -c $s- )"
    else
        echo $line
    fi
done
echo "File Folded!"
```

COMMAND DETAILS:

> *cat* : concatenate files and print on the standard output
>
> *wc* : print newline, word, and byte counts for each file
>
> > -c : print the byte counts
>
> *cut* : removes sections from each line of the file
>
> > -c : select only these characters