

Exercises

1. Read in two numbers from the keyboard and print their sum.
2. Write a shell script that given a person's uid, tells you how many times that person is logged on.
(who, grep, wc)
3. Write a shell script called lsdirs which lists *just* the directories in the current directory (test).
4. In many versions of unix there is a **-i** argument for **cp** so that you will be prompted for confirmation if you are about to overwrite a file. Write a script called cpi which will prompt if necessary without using the **-i** argument. (test, ksh)
5. Suppose you wanted to change the suffix of all your *.tex files to .latex. Doing mv *.tex *.latex won't work (why? think about what exactly the line expands to) but a short script using basename will.
- 6) List the symbolic links in a directory.
- 7) Copy files from current directory to another using a script.
- 8) Convert the contents of a file to uppercase.
- 9) Write a script to convert a decimal to a hexadecimal.
- 10) Calculate number of days between two dates.
- 11) Write a script to display first 15 Fibonacci numbers using functions.

Answers to examples

1) Adding Numbers

```
#!/bin/sh
echo "input a number"
read number1
echo "now input another number"
read number2
let answer=$number1+$number2
echo "$number1 + $number2 = $answer"
```

2) Login Counting Script

```
#!/bin/sh
times=$(who | grep $1 | wc -l)
echo "$1 is logged on $times times."
```

3) List Directories

```
#!/bin/sh
for file in $*
do
if [ -d $file ]
then
ls $file
fi
done
```

4) Safe Copying

```
#!/bin/sh
if [ -f $2 ]
then
echo "$2 exists. Do you want to overwrite it? (y/n)"
```

```
read yn
if [ $yn = "N" -o $yn = "n" ]
then
exit 0
fi
fi
cp $1 $2
```

5) Changing the suffix

```
#!/bin/sh
for f in *.tex
do
#Strip off the suffix and add the new one
newname=$(basename $f tex)latex
# Do the name change
mv $f $newname
done
```

6) Symbolic links in a directory

```
#!/bin/bash
# symlinks.sh: Lists symbolic links in a directory.

# Defaults to current working directory,
#+ if not otherwise specified.
# Equivalent to code block below.
# -----
# ARGS=1 # Expect one command-line argument.
#
# if [ $# -ne "$ARGS" ] # If not 1 arg...
# then
```

```

# directory=`pwd` # current working directory
# else
# directory=$1
# fi
# -----

echo "symbolic links in directory \"${directory}\""

for file in "$( find $directory -type l )" # -type l = symbolic links
do
echo "$file"
done | sort # Otherwise file list is unsorted.

exit 0

```

7) Copy files from current directory to another

```

#!/bin/bash
# copydir.sh

# Copy (verbose) all files in current directory ($PWD)
#+ to directory specified on command line.

E_NOARGS=65

if [ -z "$1" ] # Exit if no argument given.
then
echo "Usage: `basename $0` directory-to-copy-to"
exit $E_NOARGS
fi

ls . | xargs -i -t cp ./{} $1

```

```
# -t is "verbose" (output command line to stderr) option.
# -i is "replace strings" option.
# {} is a placeholder for output text.
# This is similar to the use of a curly bracket pair in "find."
#
# List the files in current directory (ls .),
#+ pass the output of "ls" as arguments to "xargs" (-i -t options),
#+ then copy (cp) these arguments ({} ) to new directory ($1).
#
# The net result is the exact equivalent of
#+ cp * $1
#+ unless any of the filenames has embedded "whitespace" characters.
    exit 0
```

8) Convert the contents of a file to uppercase

```
#!/bin/bash
# Changes a file to all uppercase.

E_BADARGS=65

if [ -z "$1" ] # Standard check for command line arg.
then
echo "Usage: `basename $0` filename"
exit $E_BADARGS
fi

tr a-z A-Z <"$1"

# tr '[:lower:]' '[:upper:]' <"$1"

exit 0
```

9) convert a decimal to a hexadecimal

```
#!/bin/bash
# hexconvert.sh: Convert a decimal number to hexadecimal.

E_NOARGS=65 # Command-line arg missing.
BASE=16 # Hexadecimal.

if [ -z "$1" ]
then
echo "Usage: $0 number"
exit $E_NOARGS
# Need a command line argument.
fi
# Exercise: add argument validity checking.

hexcvt ()
{
if [ -z "$1" ]
then
echo 0
return # "Return" 0 if no arg passed to function.
fi

echo ""$1" "$BASE" o p" | dc
# "o" sets radix (numerical base) of output.
# "p" prints the top of stack.
# See 'man dc' for other options.
return
}
```

```
hexcvt "$1"
```

```
exit 0
```

10) Calculate number of days between two dates.

```
#!/bin/bash
```

```
# days-between.sh: Number of days between two dates.
```

```
# Usage: ./days-between.sh [M]M/[D]D/YYYY [M]M/[D]D/YYYY
```

```
#
```

```
ARGS=2 # Two command line parameters expected.
```

```
E_PARAM_ERR=65 # Param error.
```

```
REFYR=1600 # Reference year.
```

```
CENTURY=100
```

```
DIY=365
```

```
ADJ_DIY=367 # Adjusted for leap year + fraction.
```

```
MIY=12
```

```
DIM=31
```

```
LEAPCYCLE=4
```

```
MAXRETVL=255 # Largest permissible
```

```
#+ positive return value from a function.
```

```
diff= # Declare global variable for date difference.
```

```
value= # Declare global variable for absolute value.
```

```
day= # Declare globals for day, month, year.
```

```
month=
```

```
year=
```

```

Param_Error () # Command line parameters wrong.
{
echo "Usage: `basename $0` [M]M/[D]D/YYYY [M]M/[D]D/YYYY"
echo " (date must be after 1/3/1600)"
exit $_PARAM_ERR
}

```

```

Parse_Date () # Parse date from command line params.
{
month=${1%%/**}
dm=${1%/**} # Day and month.
day=${dm#*/}
let "year = `basename $1`" # Not a filename, but works just the same.
}

```

```

check_date () # Checks for invalid date(s) passed.
{
[ "$day" -gt "$DIM" ] || [ "$month" -gt "$MIY" ] ||
[ "$year" -lt "$REFYR" ] && Param_Error
# Exit script on bad value(s).
#
}

```

```

strip_leading_zero () # Better to strip possible leading zero(s)
{ #+ from day and/or month
return ${1#0}          #+ since otherwise Bash will interpret them
}                      #+ as octal values (POSIX.2, sect 2.9.2.1).

```

```

day_index ()
{ # Days from March 1, 1600 to date passed as param.

day=$1
month=$2
year=$3

let "month = $month - 2"
if [ "$month" -le 0 ]
then
let "month += 12"
let "year -= 1"
fi

let "year -= $REFYR"
let "indexyr = $year / $CENTURY"

let "Days = $DIY*$year + $year/$LEAPCYCLE - $indexyr \
+ $indexyr/$LEAPCYCLE + $ADJ_DIY*$month/$MIY + $day - $DIM"

echo $Days

}

calculate_difference () # Difference between two day indices.
{
let "diff = $1 - $2" # Global variable.
}

```

```
abs () # Absolute value
{ # Uses global "value" variable.
if [ "$1" -lt 0 ] # If negative
then #+ then
let "value = 0 - $1" #+ change sign,
else #+ else
let "value = $1" #+ leave it alone.
fi
}
```

```
if [ $# -ne "$ARGS" ] # Require two command line params.
then
Param_Error
fi
```

```
Parse_Date $1
check_date $day $month $year # See if valid date.
```

```
strip_leading_zero $day # Remove any leading zeroes
day=$? #+ on day and/or month.
strip_leading_zero $month
month=$?
```

```
let "date1 = `day_index $day $month $year`"
```

```
Parse_Date $2
check_date $day $month $year
```

```
strip_leading_zero $day
```

```

day=$?
strip_leading_zero $month
month=$?

date2=$(day_index $day $month $year) # Command substitution.

calculate_difference $date1 $date2

abs $diff # Make sure it's positive.
diff=$value

echo $diff

exit 0

```

11)The Fibonacci Sequence

```

#!/bin/bash
# -----
# Fibo(0) = 0
# Fibo(1) = 1
# else
# Fibo(j) = Fibo(j-1) + Fibo(j-2)
# -----

MAXTERM=15    # Number of terms (+1) to generate.
MINIDX=2      # If idx is less than 2, then Fibo(idx) = idx.

Fibonacci ()
{

```

```

idx=$1          # Doesn't need to be local. Why not?
if [ "$idx" -lt "$MINIDX" ]
then
echo "$idx"     # First two terms are 0 1 ... see above.
else
(( --idx ))    # j-1
term1=$( Fibonacci $idx ) # Fibo(j-1)

(( --idx ))    # j-2
term2=$( Fibonacci $idx ) # Fibo(j-2)

echo $(( term1 + term2 ))
fi

}

for i in $(seq 0 $MAXTERM)
do # Calculate $MAXTERM+1 terms.
FIBO=$(Fibonacci $i)
echo -n "$FIBO "
done
echo
exit 0

```

Expected output

0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610