

## Instructions

Complete the problems assigned below and turn in your answers by the due time above. To receive full credit, you must show both the relevant command(s) and output. If a question asks about what a command does, or what the output would be, give both the command and the output. Demonstrate that you actually performed the command, and didn't just guess.

**Requirements** - All of your shell programs henceforth must:

- a. use the shell template
- b. check for valid arguments
- c. output sensible *usage* or syntax messages
- d. be appropriately commented
- e. exit with an appropriate exit status

### Homework Problems

1. Write a program that identifies some filesystem properties about its arguments. The program must meet the following requirements:
  - a. accepts any number of arguments
  - b. identifies each argument as one of a **file**, a **directory**, a **symbolic link**, **other**, or **non-existent**
  - c. for files
    - i. output which of readable, writable, and/or executable applies for the current user (don't use `ls -l` for this)
    - ii. indicate if the file is empty
  - d. for directories
    - i. indicates the number of directory entries (include dot files, but not `.` or `..`)
  - e. for symbolic links
    - i. indicate if the symbolic link is broken (i.e. the original file is missing)
  - f. outputs a single line per argument - for example: `letter.txt: file readable writable empty`
  - g. Suggestion: test your program on the root directory to see how well it works

This particular solution builds information about the file, directory, etc. into a variable as it is discovered. Because `test` normally follows symbolic links, the `-h` test must be performed first. Notice the use of `/usr/bin/test` and the `-e` primitive to test for existence. Finally, the use of the `printf` command makes output formatting easier.

```
#!/bin/sh

# Outputs some filesystem information about each argument
# Error codes: 0 = success; non-zero otherwise

prog=`basename $0`
usage="usage: $prog path [...]"

# syntax check: arg count
if [ $# -eq 0 ] ; then
    echo $prog: incorrect number of arguments
    echo $usage
    exit 1
fi

for file in "$@" ; do
    if [ -h "$file" ] ; then
        output=symlink
        /usr/bin/test ! -e "$file" && output="$output, broken"
    fi
done
```

```

elif [ -f "$file" ] ; then
    output=file
    [ ! -s "$file" ] && output="$output, empty"
    [ -r "$file" ] && output="$output, readable"
    [ -w "$file" ] && output="$output, writable"
    [ -x "$file" ] && output="$output, executable"
elif [ -d "$file" ] ; then
    output=directory
    nentries=unknown
    if [ -r "$file" ] ; then
        nentries=`ls -lA "$file" | wc -l`
        nentries=`expr $nentries - 1`
    fi
    output="$output, number of entries: $nentries"
elif /usr/bin/test -e "$file" ; then
    output=non-existent
else
    output=other
fi

printf "%-20s: %s\n" "$file" "$output"
done

```

2. Rewrite your Fibonacci program to meet the following new requirements:

- a. use a loop to remove all of the duplication. You should find this much easier to complete and read.
- b. accept a single command line argument that is the number of Fibonacci numbers to output
- c. be sure it works for n=0 and n=1

```

#!/bin/sh

# outputs the first n numbers from the Fibonacci series
#
# Usage: fibonacci n

prog=`basename $0`
usage="usage: $prog n"

n=$1

# syntax check: arg count
if [ $# -ne 1 ] ; then
    echo $prog: incorrect number of arguments
    echo $usage
    exit 1
fi

# syntax check: non-numeric arguments. let grep find bad values.
echo $n | grep -v '^[^0-9]' > /dev/null
if [ $? -ne 0 ] ; then
    echo $prog: invalid argument
    echo $usage
    exit 1
fi

n1=1
n2=2

# Each time through the loop, output a single value and compute the next one
while [ $n -ne 0 ] ; do
    echo $n1
    n3=`expr $n1 + $n2`
    n1=$n2
    n2=$n3

    n=`expr $n - 1`
done

```

3. One of the problems with our earlier **trm**, **trecov**, and **tempty** programs is that you find yourself duplicating commands and logic in each of the programs (for example, you had to define a variable **trash** and keep it the same in each program). This leads to copy/paste errors and is more difficult to make changes, since you have to remember to make changes in each of the three programs. Fortunately, there is a very good way to solve this problem. Put **all** the code into *one* of the files, and create hard links (with the **ln** command) named as the other two programs (eg. all code in **trm**, and links to **trm** named **trecov** and **tempty**). Since **\$0** tells you the name of your program, you can perform the appropriate commands based upon what name you were called by, using simple **if** statements. Rewrite the code to meet these new requirements:
- merge the code from all three programs into one program, and perform the appropriate actions based upon the name by which the program was called
  - output the correct usage messages
  - make sure the program works correctly for all cases
  - fix any problems with your previous code

```
#!/bin/sh

# Implements trash can functionality, that places removed items
# in a local trash for possible future recovery.

# Three functions are implemented:
#   trm
#   removes items specified on the command line, by placing them
#   into the trash in the local directory
#   trecov
#   recovers the command line specified items from the local trash
#   empty
#   empties and removes the local trash can
#
# Error codes: 0 = success; 1 = Usage, 2 = error

trash=.trash                # name of local trash can

prog=`basename $0`

if [ $prog = "empty" ] ; then
    if [ $# -ne 0 ] ; then
        echo $prog: too many arguments
        echo usage: $prog
        exit 1
    fi
else
    if [ $# -eq 0 ] ; then
        echo $prog: too few arguments
        echo usage: $prog file [...]
        exit 1
    fi
fi

# assume good exit status, until proven otherwise
status=0

case $prog in
    # Remove
    trm)
        # don't care if there are errors with mv.
        # return mv's error code (occurs automatically,
        # since mv is the last command)
        mkdir -p $trash && mv "$@" $trash
        ;;

    # Recover
    trec*)
        if [ ! -d $trash ] ; then
            echo "$prog: trash is empty; cannot recover files"
            exit 2
        fi
    ;;
esac
```

```

fi

cd $trash
mv "$@" ..
status=$? # it is mv's status we want...
cd ..

# trying to remove a non-empty trash will fail, but this is desired!
rmdir $trash 2> /dev/null
;;

# Empty
te*)

# no need to complain if there is no trash -
# the job is done either way
rm -rf $trash
;;

*)
echo Unknown program!
exit 2
;;
esac

exit $status

```

4. Write a program that renames files, directories, or symbolic links according to the requirements below:
- do not use the **getopts** program
  - option **-u** renames filenames in all uppercase (mutually exclusive with **-l**)
  - option **-l** renames filenames in all lowercase (mutually exclusive with **-u**)
  - option **-w** renames filenames, removing whitespace from within a name
  - option **-v** enables verbose output while program is operating, indicating what rename changes (not using **set -x** or **-v**)
  - does not overwrite existing files; gives an error, but continues processing other files
  - accepts any number of filename arguments
  - usage message must show acceptable options and appropriate syntax (like Synopsis in man pages)

The key to solving this problem is to break the problem into manageable parts. The first part handles argument processing – it's job is to remove all the options from the command line leaving only the file list, and to setup the program to perform the actions specified by those options. The second part does translation - it performs the translations specified by the options that were found on the command line. The third part does the actual rename. Variables are used to indicate which translations should be performed on the filename(s).

```

#!/bin/sh

# Renames files
#
# Supported Options
#   -u translates filenames to uppercase
#   -l translates filenames to lowercase
#   -w removes whitespace from filenames
#   -v gives verbose output
#
# Error codes: 0 = success; 1 = syntax error, 2 = other error

prog=`basename $0`
usage="usage: $prog [-u | -l] [-w] [-v] file [...]"

if [ $# -eq 0 ] ; then
    echo $prog: too few arguments

```

```

    echo $usage
    exit 1
fi

status=0          # assume good exit status, until proven otherwise
removewhite=0     # if set, remove whitespace from name
filecase=""       # will set to "upper" or "lower" depending on -u or -l
verbose=0         # if set, give verbose output

# First, process the options.  Just remember what was found by use of
# variables - this will indicate what translations to perform later.
while [ $# -ne 0 ] ; do
    case $1 in
        -u) filecase=upper ;;
        -l) filecase=lower ;;
        -w) removewhite=1 ;;
        -v) verbose=1 ;;
        -*)
            # It is common practice that anything starting with a dash that
            # isn't one of the allowable options is an error.
            echo Unknown option \"$1\"
            echo $usage
            exit 1
            ;;
        *)
            break          # presume filelist has started
            ;;
    esac
    shift                # option processed, now shift it out
done

# Now that the options are out of the way, we can process the remaining
# arguments, which are assumed to be the list of files.
# The strategy here is to first calculate the new name of the file, and
# after that, use mv to rename it
while [ $# -ne 0 ] ; do
    newname=$1          # start with the original name

    if /usr/bin/test ! -e "$1" ; then
        echo "$1: no such file or directory"
        status=2        # indicate for later that there was an error
        shift; continue; # toss this one, and go onto the next one
    fi

    # these are mutually exclusive, so if/elif makes the most sense here
    if [ "$filecase" = upper ] ; then
        newname=`echo $newname | tr '[a-z]' '[A-Z]'`
    elif [ "$filecase" = lower ] ; then
        newname=`echo $newname | tr '[A-Z]' '[a-z]'`
    fi

    [ "$removewhite" -eq 1 ] && newname=`echo $newname | sed 's/[ \t]//g'`

    if [ "$newname" != "$1" ] ; then
        [ "$verbose" -eq 1 ] && echo renaming \"$1\" to \"$newname\"

        # better make sure there isn't something already there!
        if /usr/bin/test -e "$newname" ; then
            echo "$1": Cannot rename - $newname exists!
            status=2        # indicate there was some error
        else
            mv "$1" "$newname"
        fi
    fi

    shift
done

```

`exit $status`

Extra Credit Problems - Do one or more of the following exercises for extra credit. Extra credit will only be received if you have completed the required problems above.

5. Write a program that acts just like **grep**, but adds highlighting capabilities by reversing the video of the *matched* text, and *always* shows the name of the file like **grep** does when there are multiple files. Hint: try the following command:  

```
$ echo Please `tput smso`reverse`tput rmso` me!
```
6. Rewrite program 4 above to use **getopts**, which will allow joining of arguments (e.g. -vuw)