

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 invalid arguments and command line syntax
- c. output sensible *usage* or syntax messages
- d. be appropriately and sufficiently commented
- e. define and exit with appropriate exit status codes
- f. use variables for strings used more than once
- g. not hard-code the program name

Homework Problems

1. Modify your latest version of your **rename** program (your may start with the one provided in the homework solutions). The program takes a list of paths, and renames each according to the options specified. The following specify the supported options and the additional requirements:
 - a. Options supported (can be given on the command line in any order; option-arguments must follow their corresponding option – see **intro(1)** and **getopts(1)** if desired)
 - i. `-f` force overwrite [mutually exclusive with `-i`]
 - ii. `-i` confirm overwrite [mutually exclusive with `-f`]
 - iii. `-v` verbose output
 - iv. `-c` capitalizes filenames (i.e. File1, Test, Hw2) [`-c`, `-l` and `-u` are mutually exclusive]
 - v. `-l` makes filenames all lowercase [`-c`, `-l` and `-u` are mutually exclusive]
 - vi. `-u` makes filenames all uppercase [`-c`, `-l` and `-u` are mutually exclusive]
 - vii. `-w` removes whitespace from each filename
 - viii. `-n num` numbers files; appends the integers `num`, `num+1`, `num+2`, ... to each consecutive filename
 - ix. `-p prefix` adds the prefix `prefix` to each filename [`-p` and `-P` are mutually exclusive]
 - x. `-P prefix` removes the prefix `prefix`, which may be a regular expression, from each filename [`-p` and `-P` are mutually exclusive]
 - xi. `-s suffix` adds the suffix `suffix` to each filename [`-s` and `-S` are mutually exclusive]
 - xii. `-S suffix` removes the suffix `suffix`, which may be a regular expression, from each filename [`-s` and `-S` are mutually exclusive]
 - xiii. `-r` **Extra Credit:** recursively applies changes to files under any directories given as arguments. Normally, directories specified on the command line are themselves renamed and not traversed. With the `-r` option, renames all non-directory items under the specified directory tree (this includes traversing into its sub-directories). Dot files are ignored.
 - b. Output an error and usage message for conflicting mutually exclusive arguments (i.e. `-s` and `-S`, `-i` and `-f`, etc.)
 - c. Do *not* call your program recursively – handle these requirements with only one call to your program.

The code below for `rename` is just one possible implementation. The code is more complex than the problem requires since a) it implements translations based on the order the options were presented on the command line, b) includes the recursive directory support (`-r`), and c) has some debugging techniques that may be useful to you in your future work. Additionally, it is heavily commented to be more instructive.

```

#!/bin/sh

# rename
#
# Renames files according to the command line options specified.  The order
# of the options on the command line affects the resulting file name.  Actions
# for left-most options are performed first.
#
# Supported Options: See Usage function below for options
#
# Error codes: 0 = success; non-zero = one or more failures

prog=`basename $0`

# Syntax: Usage [ args ... ]
# Standard usage message and exit
sopts="cdfilruvw"          # helps usage msg match parsed opts in getopt
Usage() {
    Error $*
    cat << EndOfUsage
usage: $prog [-$sopts] [-[p|P] prefix] [-[s|S] suffix] [-n num] file [...]
  Options:
  -c          capitalize first (title case) (not w/-l, -u)
  -d          debug - no changes will be made
  -f          force overwrite of existing file (not w/-i)
  -i          confirm overwrite of existing file (not w/-f)
  -l          lowercase (not w/-c,-u)
  -n num      append increasing integer starting at num
  -p prefix   insert prefix (not w/-P)
  -P prefix   remove prefix (not w/-p)
  -r          recursively descend into directories
  -s suffix   append suffix (not w/-S)
  -S suffix   remove suffix (not w/-s)
  -u          uppercase (not w/-c,-l)
  -v          verbose output
  -w          remove whitespace
EndOfUsage

    exit 1
}

# unset variables mean option, action, etc. is not being done
#
errorcode=          # error code for exit status
suffix=             # suffix to be added/removed
prefix=             # prefix to be added/removed
number=             # number being currently added
number_orig=       # original starting number
recurse=           # are we recursive?
verbose=           # are we being verbose?
ovrwr=             # what is our overwrite mode (none, i or f)?
DEBUG=             # is debug enabled?  will be set with -d

# Syntax: Debug [ args ... ]
# Prints out debug message if DEBUG is set
Debug() {
    [ -n "$DEBUG" ] && echo "\tDebug: $"
}

# Error [ args ... ]
# Prints out its arguments as an error message
Error() {
    [ $# -ne 0 ] && echo $prog: $*
}

```

```

# Translation functions
# All set global variable new to translated filename

# Synopsis:
# doConversion type string
# Performs a basic conversion selected by $1 on a string in $2
# all lowercase, all uppercase, capitalize first only, remove whitespace
doConversion() {
    case $1 in
        lower)
            new=`echo $2 | tr '[A-Z]' '[a-z]` ;;
        upper)
            new=`echo $2 | tr '[a-z]' '[A-Z]` ;;
        capitalize)
            first=`echo $2 | cut -c1 | tr '[a-z]' '[A-Z]`
            others=`echo $2 | cut -c2- | tr '[A-Z]' '[a-z]`
            new="$first{others}" ;;
        remwhite)
            new=`echo $2 | sed 's/[ \t]//g` ;;
    esac
    Debug "Conversion($1): '$2' --> '$new'"
}

# Synopsis:
# doStrcat string1 string2
# Concatenates two arguments $1 and $2
doStrcat() {
    new="$1$2"
    Debug "doStrcat: '$1' + '$2' --> '$new'"
}

# Synopsis:
# doRemSuffix string suffix
# Removes suffix $2 from the end of string in $1
doRemSuffix() {
    new=`echo $1 | sed "s/$2$//"`
    Debug "doRemSuffix: '$1' - '$2\$' --> '$new'"
}

# Synopsis:
# doRemPrefix filename prefix
# Removes prefix $2 from the beginning of string in $1
doRemPrefix() {
    new=`echo $1 | sed "s/^$2//"`
    Debug "doRemPrefix: '$1' - '^$2' --> '$new'"
}

# Synopsis:
# doTranslateAndMove filename
# Translates a filename in $1 based on options previously set in the global
# variable xlate. Name translations are done in order presented in $xlate,
# so that user can rename in creative ways. Loops calling itself recursively,
# if $recurse is set and $1 is a directory, on each file under the directory.
#
doTranslateAndMove() {
    if [ -n "$recurse" -a -d "$1" ] ; then
        (
            # Call ourself once for each file in the sub-directory
            # This is done in a subshell - note the surrounding parenthesis -
            # so we don't need to remember our current working directory

            cd "$1" || exit 2

            [ -n "$DEBUG" ] && Debug New Directory: `pwd`

            # It makes more sense to restart numbering within each dir, rather
            # than increasing the number no matter what. This allows numbering
            # all files within each directory from some starting number.

```

```

# again, since this is being done in a sub-shell, the parent
# shell's number variable is unaffected
number=$number_orig

# Must count the number of files in the directory first to be sure
# that * will expand to something, since shell will leave it as
# '*' if it fails to match any files during filename expansion
if [ `ls | wc -l` -ne 0 ] ; then
    for filename in * ; do
        doTranslateAndMove "$filename"
    done
fi
exit $errorcode
)
return $?
fi

new="$1"
# At this point, $old can be renamed as is

if /usr/bin/test ! -e "$1" ; then
    Error "$1: no such file or directory"
    return 1
fi

for translation in $xlate ; do
    case "$translation" in
        u) doConversion upper "$new" ;;
        l) doConversion lower "$new" ;;
        c) doConversion capitalize "$new" ;;
        w) doConversion remwhite "$new" ;;
        S) doRemSuffix "$new" "$suffix" ;;
        P) doRemPrefix "$new" "$prefix" ;;
        s) doStrcat "$new" "$suffix" ;;
        p) doStrcat "$prefix" "$new" ;;
        n) doStrcat "$new" "$number"
            number=`expr $number + 1` ;;
    esac
done

# no need to rename if there was no change between new and orig
[ "$new" = "$1" ] && return

[ "$new" = "" ] && Error Unable to rename \"$1\" to \"\" && return 1

# It is an error if overwriting is not enabled and a file
# with the new name already exists
if /usr/bin/test -e "$new" ; then
    if [ -d "$new" ] ; then
        Error "Renaming \"$1\" would move it into existing directory \"$new\"
        return 1
    fi
    if [ -z "$ovrwr" ] ; then
        Error "$1: Cannot rename: \"$new\" exists. Use -f or -i to overwrite"
        return 1
    fi
fi

[ "$verbose" -eq 1 ] && echo renaming \"$1\" to \"$new\"

# $DEBUG will be set to "Debug" in debug mode (with -d). This causes
# the mv command to become an argument to the Debug function, and not a
# command itself. This avoids actually doing anything. This is safer
# and it saves time by not having to constantly rename test files during
# testing. When -d is not supplied as an option, then we are not
# in debug mode, DEBUG is null, and the mv command runs. Very cool.

$DEBUG /usr/bin/mv $ovrwr "$1" "$new"

```

```

}

# Beginning of main portion of the code

[ $# -eq 0 ] && Usage "too few arguments"

exclusive="are mutually exclusive options."
exclusive_ulc="-u -l and -c $exclusive"
exclusive_pP="-p and -P $exclusive"
exclusive_sS="-s and -S $exclusive"
exclusive_iF="-i and -f $exclusive"

# First, process the options. Remember what was found by use of
# variables - this will indicate what translations to perform later, or
# how the program should behave.
while getopts ${sopts}s:S:p:P:n: option ; do
    case "$option" in
        d) DEBUG=Debug; continue ;;
        r) recurse=1 ; continue ;;
        v) verbose=1 ; continue ;;

        u | l | c)
            [ -n "$casechanged" ] && Usage $exclusive_ulc ; casechanged=1 ;;

        n) number="$OPTARG" number_orig="$OPTARG"
            # syntax check: non-numeric arguments. let grep find bad values.
            echo $number | grep -v '^[^0-9]' > /dev/null
            [ $? -ne 0 ] && Usage -n requires a number: $number is non-numeric.
            ;;

        i | f) [ -n "$ovrwrtr" ] && Usage $exclusive_iF ; ovrwrtr=$option ;;
        s | S) [ -n "$suffix" ] && Usage $exclusive_sS ; suffix=$OPTARG ;;
        p | P) [ -n "$prefix" ] && Usage $exclusive_pP ; prefix=$OPTARG ;;
        w) : ;; # action captured by setting xlate below
        \?) Usage
    esac
    xlate="$xlate $option"
done

Debug Translations: "$xlate"

[ -z "$xlate" ] && Usage "No translation option specified"

# shift away the options just processed
shift `expr $OPTIND - 1`

[ $# -eq 0 ] && Usage "missing file argument"

# Now that the options are out of the way, we can process the remaining
# arguments, which are assumed to be a list of files.
# The function doTranslateAndMove will translate and move each filename
# and set errorcode if there is an error.
# Since doTranslateAndMove only works on a filename, and does not handle
# any initial path components, they are separated here into their file
# and directory parts; cd'ing into the directory where the file lives
# ensure that doTranslateAndMove only deals with filenames.
while [ $# -ne 0 ] ; do
    file=`basename "$1"`
    dir=`dirname $1` # dirname returns . if no dir part

    Debug file: \"$file\", dir: \"$dir\"
    if [ \"$dir\" != "." ] ; then # path contains dir part
        ocwd=`pwd`
        cd \"$dir\"
        [ $? -ne 0 ] && errorcode=1 && continue

        [ -n \"$DEBUG\" ] && Debug New Directory: \"$dir\"
    fi
done

```

```
doTranslateAndMove "$file" || errorcode=1

# go back to the original directory - we'd better exit if this fails
if [ $dir != "." ] ; then
    cd "$ocwd" || exit
fi

shift
done

exit ${errorcode:-0}
```

2. Create a program named **dom2dir** which takes reads a list of domain names (i.e. www.yahoo.com) and creates a sub-directory tree based on the (dot separated) components of the reversed domain name.
 - a. The rightmost components of the domain name are created highest in the directory sub-directory tree, and the leftmost components are created deepest in the sub-directory tree (e.g. for www.yahoo.com, the directory tree com/yahoo/www would be the created). Don't assume only three levels deep – any number should be handled.
 - b. Use the directory named **./TLD** as the base directory for the directory tree.
 - c. This problem *must* be solved entirely with Bourne built-in shell commands, with the exception of the external command **mkdir**.
 - d. The list of domain names contains one domain name per line (newline separated)
 - e. The program must be able to read the list as an argument, from a pipe, or from redirection. Examples:
 - i. dom2dir urlfile
 - ii. cat urlfile | dom2dir
 - iii. dom2dir < urlfile

Hints

- Think about how you will parse each domain name and how then to reverse the components
- The bulk of this program can be implemented using about 13 lines of code

```
#!/bin/sh

# urltree
#
# Creates a directory tree of domains and sub-domains, and hosts with
# TLDs being the top-most directories, and hosts being deepest.
#
# Does not use any non-built-in commands except mkdir. This helps
# to demonstrate how proper implementation can substantially decrease
# code size, decreases chance for errors (fewer lines means fewer lines that
# can have errors), and increase
# code performance substantially.
#
prog=`basename $0`
basedir=TLD
DEBUG=                # is debug enabled? will be set with -d

Usage() {
    [ $# -ne 0 ] && echo "$prog: $"
    echo usage: $prog [-d] [file]
    exit 1
}

# Syntax: Debug [ args ... ]
# Prints out debug message if DEBUG is set
Debug() {
    [ -n "$DEBUG" ] && echo "\tDebug: $"
}

[ $# -ne 0 -a "$1" = "-d" ] && DEBUG=Debug && shift;
[ $# -gt 1 ] && Usage "too many arguments"

IFS_ORIG=$IFS
# Set STDIN to come from $1 - if $1 is null, STDIN is the terminal, or
# a pipeline if one was used on the command line. This is the standard
# technique to setup a program to act as a filter.
exec < "$1"

while read line ; do
    Debug Read $line
    IFS=.
done
```

```
set -- $line
IFS=$IFS_ORIG
reverse=
# spin through each component, adding each to the beginning of the
# variable reverses the string.
while [ $# -gt 0 ] ; do
    reverse="$1/$reverse"
    shift
done
$DEBUG mkdir -p $basedir/$reverse
[ $? -ne 0 ] && exit 2
done
```