

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.

Homework Problems

1. Implement the phonebook programs **add**, **rem**, and **lu** from Chapter 7, with the following modifications:
 - a. Use a TAB separator between the name and phone number fields, not spaces.
 - b. Do not use the TAB character itself - rather, use the appropriate **echo** escapes.
 - c. Modify **lu** to ignore case during lookups.
 - d. Do not use the **/tmp/phonebook** file for the **rem** program - instead, use **./phonebook.tmp**.
 - e. Use variables for all filenames in each program.
 - f. Do not allow duplicate entries to be added. [hint: no extra code or commands is required]

```
$ cat add
#!/bin/sh

# adds a name and phone number into the phonebook
# sort removes duplicates via -u

phonebookfile=./phonebook

echo "$1\t$2" >> $phonebookfile
sort -u -o $phonebookfile $phonebookfile

$ cat lu
#!/bin/sh

# looks up the specified first argument from the phonebook
# case is ignored

phonebookfile=./phonebook

grep -i "$1" $phonebookfile

$ cat rem
#!/bin/sh

# removes the specified entry from the phonebook

phonebookfile=./phonebook
tmpfile=./phonebook.tmp

grep -v "$1" $phonebookfile > $tmpfile
mv $tmpfile $phonebookfile
```

2. Answer the questions in Exercise 2, page 147. Also, what happens when you pass a space as an argument?

If you do not supply an argument to **lu**, then you should receive the error message:

```
grep: RE error 41: No remembered search string.
```

If you did not double quote the **\$1** argument, you would find that **lu** waits for input from you - this is because **\$1** would be null, and **grep** would have the wrong number of arguments. It would be waiting for input from STDIN. If you properly

double quoted \$1 (to preserve any filenames with spaces, for example), then both no argument, or a null argument to **lu** would produce the same error message. If you supply space as an argument (i.e. " "), then **lu** will output every entry that contains a space somewhere in the line, which is likely to be all entries from the phonebook if each entry has a first name and last name separated by a space.

3. What happens if you only give 1 argument to the **add** program?

The **add** program will happily add the single argument as a new name in the phonebook, without a corresponding phone number.

4. For problem 1, above, modification (d), why did I have you not use the file **/tmp/phonebook**?

The file **/tmp/phonebook** is not used if everyone in the class used the same filename, there would likely be corruption or an error when you and your classmates were running the **rem** program. Since your **rem** program would be writing the file named **/tmp/phonebook**, if a classmate was running their **rem** program at the same time, their program would attempt to overwrite your **/tmp/phonebook**. When writing any form of program, you have to make sure that files created in your program will not collide with those created by another user of the same program. A more robust way (but not yet perfect) is to create temporary files in the user's current directory. Later I will show you how to make this much more safe.

5. Modify the program from problem 6, homework 3 to use two numbers passed on the command line, instead of from two files. If wrote your program appropriately, this will only require two simple changes at the top of your file.

```
#!/bin/sh

# performs various mathematical operations on two numbers
# arguments 1 and 2 are the operands

x=$1
y=$2

echo x = $x
echo y = $y
echo "Addition:      $x + $y = `expr $x + $y`"
echo "Subtraction:  $x - $y = `expr $x - $y`"
echo "Multiplication: $x * $y = `expr $x \* $y`"
echo "Division:      $x / $y = `expr $x / $y`"
echo "Remainder:     $x % $y = `expr $x % $y`"
echo "Equals:        $x = $y ? `expr $x = $y` (1 = Yes, 0 = No)"
echo "Less Than:     $x < $y ? `expr $x \< $y` (1 = Yes, 0 = No)"
echo "Greater Than:  $x > $y ? `expr $x \> $y` (1 = Yes, 0 = No)"
```

6. Do Exercise 6, page 148.

```
#!/bin/sh

file="$1"
suffix="$2"

mv "$file" "$file$suffix"
```

7. Do Exercise 7, page 148.

```
#!/bin/sh

file="$1"
suffix="$2"

mv "$file" `echo "$file" | sed "s/$suffix\$//"`
```

8. Write a program that does the following:

- Line 1 of the output states how many arguments were passed.
- Line 2 of the output states how many actual words existed on the command line (ignoring the command itself).

- c. Outputs the first nine command line arguments, one per line. Each argument should be output such that it is surrounded by single quotes, so that you can exactly see what the argument was. Also precede each single quoted argument with an appropriate label.

```
#!/bin/sh

echo Args Passed: $#
echo Words on Command Line : `echo $* | wc -w`
echo Arg1: "'$1'"
echo Arg2: "'$2'"
echo Arg3: "'$3'"
echo Arg4: "'$4'"
echo Arg5: "'$5'"
echo Arg6: "'$6'"
echo Arg7: "'$7'"
echo Arg8: "'$8'"
echo Arg9: "'$9'"
```

9. Write two additional programs that output a single line. The output line announces the program's name and the arguments it received (single quote each for clarity). The first program is to run the second program; the second program runs the program you created from problem 8. Be sure all arguments pass correctly.

```
$ cat p9.1
#!/bin/sh

# outputs its arguments in single quotes and calls
# the next program with the same arguments
#
# p9.1 calls ...
#   p9.2 which calls ...
#     p8

# show our first 9 arguments in single quotes
echo p9.1: passing "'$1'" "'$2'" "'$3'" "'$4'" "'$5'" "'$6'" "'$7'" "'$8'" "'$9'"

# now call the next program with the same arguments passed to us
p9.2 "$1" "$2" "$3" "$4" "$5" "$6" "$7" "$8" "$9"

$ cat p9.2
#!/bin/sh

# outputs its arguments in single quotes and calls the program from
# problem 8 with the same arguments

# show our first 9 arguments in single quotes
echo p9.2: passing "'$1'" "'$2'" "'$3'" "'$4'" "'$5'" "'$6'" "'$7'" "'$8'" "'$9'"

# now call the p8 program with the same arguments passed to us
p8 "$1" "$2" "$3" "$4" "$5" "$6" "$7" "$8" "$9"
```

10. Write three programs called **trm**, **treco**, and **tempty**. The **trm** program is like **rm**, but instead of removing files, it places them in a directory named **Trash** (similar to the Recycle Bin in Windows or the Trash in Mac OS). The **treco** program recovers the files named on the command line from the **Trash** and places them in the current directory. Finally, the **tempty** program permanently deletes items in the **Trash**. You do not have to worry about duplicate files names or errors messages - these programs should be simple and straight forward, using what has been explored thus far in the course. We will add to these programs and make them more robust in future assignments.

```
$ cat trm
#!/bin/sh

# part of a trash can series of programs, that places removed items
# in the trash for future recovery.
#
# limitation - does not handle file names with whitespace

# moves the files specified on the command line into the trash can
```

```
mkdir -p Trash
mv $* Trash

$ cat trecov
#!/bin/sh

# part of a trash can series of programs, that places removed items
# in the trash for future recovery.
#
# limitation - does not handle file names with whitespace

# recovers the requested items from the local trash can

cd Trash
mv $* ..
cd ..

$ cat tempty
#!/bin/sh

# part of a trash can series of programs, that places removed items
# in the trash for future recovery.

# empties the trash can and removes the empty trash
rm -rf Trash
```

Optional Problems - I would recommend that you try the Exercises on pages 147-148 that were not required above. You do not need to turn these in.