Chapter 1 Exercises

1. Given the following directory listing:

```
$ ls
feb96
jan12.02
jan19.02
jan26.02
jan5.02
jan95
jan96
jan97
jan98
mar98
memo1
memo10
memo2
memo2.sv
$
```

   What is the expected output from the following commands?

```
echo *
echo *[!0-9]
echo m[a-df-z]*
echo [A-Z]*
echo jan*
echo *.*
echo ?????
echo *02
echo jan?? feb?? mar??`
echo [fjm][ae][bnr]*
```

2. What is the result of the following command sequences?

```
ls | wc -l
rm ???
who | wc -l
mv progs/* /users/steve/backup
ls *.c | wc -l
rm *.o
who | sort
cd; pwd
cp memo1 ..
plotdata 2>errors &
```

3. You have a filename with spaces in it. How do you copy it? How do you delete it?

4. Describe the difference between > and >>, and then describe <. Explain what the sequence << means to the shell.

5. The `ls` command is important for shell programmers, but it has some quirks. Offer solutions for the following:
   a. `ls` gives you single-column output, but you want it to be shown in multiple columns.
   b. `ls` shows output sorted alphabetically. Sort it by most recently modified to least instead.
   c. Type two different ways to have `ls` include file size information.
   d. The opposite of `a: Force ls` to output a one-column file listing.

Chapter 2 Exercises

1. Gareth has this `/etc/login` entry:

   ```
   gareth:*:117:100:Gareth T:/users/gareth:/bin/bash
   ```

   What are his home directory, login shell, and user ID?

2. Use the command **`echo /bin/*sh`** to see how many shells you have on your system. Are they all login shells?

3. When you type in a command to your shell, it uses a particular variable to ascertain where to look for that command. What is it, and what happens if it's set to `NULL`?

4. Suppose you were in a directory with the following files:

   ```
   $ ls
   arsenal
   chelsea
   manchester city
   tottenham hotspurs
   ```

   How many arguments would be sent to the shell in each of the following cases?

   ```
   echo *
   echo "*"
   echo [ac]*
   echo *\ *
   ```

5. What do you think would happen if you invoked the following command on the shell? Why?

   ```
   cat infile | sort | wc -l > infile
   ```

Chapter 3 Exercises

1. What will be matched by the following `ed` regular expressions? (Tip: If you're using them in other contexts, you should omit the backslashes)

```
x*
[0-9]\{3\}
xx*
[0-9]\{3,5\}
x\{5,\}
^\...
x\{10\}
[A-Za-z_][A-Za-z_0-9]*
[0-9]*
^Begin$
[0-9][0-9][0-9]
^\(.\).*\1$
```

2. What will be the effect of the following commands?

```
who | grep 'mary'
who | grep '^mary'
grep '[Uu]nix' ch?/*
ls -l | sort -k4n
sed '/^$/d' text > text.out
sed 's/\([Uu]nix\)/\1(TM)/g' text > text.out
date | cut -c12-16
```

3. Write the command to find all logged-in users with usernames of at least four characters.

4. Find all users on your system whose user IDs are greater than 99, and the number of users on your system whose user IDs are greater than 99.

5. List all the files in your directory in decreasing order of file size.

Chapter 4 Exercises

1. Which of the following are valid variable names?

   ```
   XxXxXx
   HOMEDIR
   file_name
   x09
   file1
   Slimit
   ```

2. Suppose that your `HOME` directory is `/users/steve`. Assuming that you just logged in to the system and executed the following commands:

   ```
   $ docs=/users/steve/documents
   $ let=$docs/letters
   $ prop=$docs/proposals
   ```

   Write the commands in terms of these variables to:

   a. List the contents of the `documents` directory.

   b. Copy all files from the `letters` directory to the `proposals` directory.

   c. Move all files whose names contain a capital letter from the `letters` directory to the current directory.

   d. Count the number of files in the `memos` directory.

3. Suppose that your `HOME` directory is `/users/steve`. Assuming that you just logged in to the system and executed the following commands:

   ```
   $ docs=/users/steve/documents
   $ let=$docs/letters
   $ prop=$docs/proposals
   ```

   What would be the effect of the following commands?

   a. **`ls $let/..`**

   b. **`cat $prop/sys.A >> $let/no.JSK`**

   c. **`echo $let/*`**

   d. **`cp $let/no.JSK $progs`**

4. Write a program called `nf` to display the number of files in your current directory. Type in the program and test it out.

5. Write a program called `whos` to display a sorted list of the logged-in users. Just display the usernames and no other information. Type in the program and test it out.

Chapter 5 Exercises

1. Given the following assignments:

```
$ x=*
$ y=?
$ z='one
> two
> three'
$ now=$(date)
$ symbol='>'
```

   and these files in your current directory:

```
$ echo *
names test1 u vv zebra
```

   What will the output be from the following commands?

```
echo *** error ***
echo 'Is 5 * 4 > 18 ?'
echo $x
echo What is your name?
echo $y
echo Would you like to play a game?
echo "$y"
echo \*\*\*
```

2. Given the following assignments:

```
$ x=*
$ y=?
$ z='one
> two
> three'
$ now=$(date)
$ symbol='>'
```

   What would the output of the following commands?

```
echo $z | wc -l
echo \$$symbol
echo "$z" | wc -l
echo $\$symbol
echo '$z' I wc -l
echo "\"
echo _$now_
echo "\\"
echo hello $symbol out
echo \\
echo "\""
echo I don't understand
```

3. Write the commands to remove all the space characters stored in the shell variable `text`. Be sure to assign the result back to `text`. First use `tr` to do it and then do the same thing with `sed`.

4. Write the commands to count the number of characters stored in the shell variable `text`. Then write the commands to count all the alphabetic characters. (Hint: Use `sed` and `wc`.) What happens to special character sequences such as `\n` if they're stored inside `text`?

5. Write the commands to assign the unique lines in the file `names` to the shell variable `namelist`.

Chapter 6 Exercises

1. Modify `lu` so that it ignores case when doing the lookup.

2. What happens if you forget to supply an argument to the `lu` program? What happens if the argument is null (as in, `lu ""`)?

3. The program `ison` from this chapter has a shortcoming as shown in the following example:

   ```
   $ ison ed
   fred     tty03    Sep  4 14:53
   ```

   The output indicates that `fred` is logged on, while we were checking to see whether `ed` was logged on. Modify `ison` to correct this problem.

4. Write a program called `twice` that takes a single integer argument and doubles its value:

   ```
   $ twice 15
   30
   $ twice 0
   0
   $
   ```

   What happens if a non-integer value is typed? What if the argument is omitted?

5. Write a program called `home` that takes the name of a user as its single argument and prints that user's home directory. Specifically:

   ```
   home steve
   ```

   should print

   ```
   /users/steve
   ```

   if `/users/steve` is `steve`'s home directory. (Hint: Recall that the home directory is the sixth field stored in the file `/etc/passwd`.)

Chapter 7 Exercises

1. Write a program called `valid` that prints `yes` if its argument is a legal shell variable name and `no` otherwise:

```
$ valid foo_bar
yes
$ valid 123
no
```

2. Write a program called `thetime` that displays the time of day in am or pm notation rather than in 24-hour clock time.

```
$ date
Wed Aug 28 19:34:01 EDT 2002
$ thetime
7:21 pm
```

   Suggestion: Use the shell's built-in integer arithmetic to convert from 24-hour clock time. Then rewrite the program to use a `case` command instead. Rewrite it again to perform arithmetic with the `expr` command.

3. Write a program called `mysed` that applies the `sed` script given as the first argument against the file given as the second. If the `sed` succeeds (that is, exit status of zero), replace the original file with the modified one. Thus:

```
mysed '1,10d' text
```

   will use `sed` to delete the first 10 lines from `text`, and, if successful, will replace `text` with the modified file.

4. Write a program called `isyes` that returns an exit status of `0` if its argument is `yes`, and `1` otherwise. For purposes of this exercise, consider `y`, `yes`, `Yes`, `YES`, and `Y` all to be valid `yes` arguments:

```
$ isyes yes
$ echo $?
0
$ isyes no
$ echo $?
1
```

   Write the program using an `if` command and then rewrite it using a `case` command. This program can be useful when reading yes/no responses from the terminal.

5. Use the `date` and `who` commands to write a program called `conntime` that prints the number of hours and minutes that a user has been logged on to the system (assume that this is less than 24 hours).

Chapter 8 Exercises

1. Modify the `prargs` program to precede each argument by its number. Thus, typing

   ```
   prargs a 'b c' d
   ```

   should give the following output:

   ```
   1: a
   2: b c
   3: d
   ```

2. Modify the `waitfor` program to also print the `tty` number that the user logs on to. That is, the output should look like:

   ```
   sandy logged onto tty13
   ```

   if `sandy` logs on to `tty13`.

3. Add a `-f` option to `waitfor` to have it periodically check for the existence of a file (ordinary file or directory) instead of for a user logging on. So typing:

   ```
   waitfor -f /usr/spool/steve/newmemo &
   ```

   should cause `waitfor` to periodically check for the existence of the indicated file and inform you when it does (by displaying a message or by mail if the `-m` option is also selected).

4. Add a `-n` option to `waitfor` that inverts the monitoring function. So

   ```
   waitfor -n sandy
   ```

   checks for `sandy` logging off the system, and

   ```
   waitfor -n -f /tmp/dataout &
   ```

   periodically checks for the removal of the specified file.

5. Write a shell program called `wgrep` that searches a file for a given pattern, just as `grep` does. For each line in the file that matches, print a "window" around the matching line. That is, print the line preceding the match, the matching line, and the line following the match. Be sure to properly handle the special cases where the pattern matches the first line of the file and where the pattern matches the last line of the file.

Chapter 9 Exercises

1. Write a program called `mymv` that does with the `mv` command what `mycp` does with the `cp` command. How many changes did you have to make to `mycp` to produce this new program?

2. Modify `mycp` to prompt for arguments if none are supplied. A typical execution of the modified version should look like this:

   ```
   $ mycp
   Source file name? voucher
   Destination file name? voucher.sv
   $
   ```

   Make sure that the program allows one or both of the files to be specified with filename substitution characters.

3. Add a `-n` option to `mycp` that suppresses the normal check for the existence of the destination files.

4. Modify the `addi` command to loop until the user enters quit and to allow users to specify what function they want to apply to the pair of numbers given so that the following session would work.

   ```
   $ addi
   100 200
   300
   100 - 50
   50
   5 * 5
   25
   5/5
   error: not enough arguments given
   quit
   $
   ```

5. Modify `lu` from Chapter 6 to use `printf` to print the name and phone number so that they line up in columns for names up to 40 characters in length (Hint: use `cut -f` and the fact that the fields in the `phonebook` are separated by tabs).

Chapter 10 Exercises

1. Write a program called `myrm` that takes as arguments the names of files to be removed. If the global variable `MAXFILES` is set, take it as the maximum number of files to remove without question. If the variable is not set, use 10 as the maximum. If the number of files to be removed exceeds this count, ask the user for confirmation before removing the files.

```
$ ls | we -l
25
$ myrm *                        Remove them all
Remove 25 files (y/n)? n
files not removed
$ MAXFILES=100 myrm *
$ ls
$                                       All files removed
```

   If `MAXFILES` is set to zero, the check should be suppressed.

2. Here are two programs called `prog1` and `prog2`:

```
$ cat prog1
e1=100
export e1
e2=200
e3=300 prog2
$ cat prog2
echo e1=$e1 e2=$e2 e3=$e3 e4=$e4
$
```

What output would you expect after typing the following:

```
$ prog1
$ e2=20; export e2
$ e4=40 prog1
```

3. Modify `rolo` from this chapter so that a person running the program can keep their phone book file in whatever directory they prefer. This can be done by requiring that the user have set a variable called `PHONEBOOK` to be the name of the phone book file. Check to make sure that it's a valid file and default to the phone book file being in their `$HOME` directory if the variable's not set.

   Here is an example:

```
$ PHONEBOOK=/users/steve/personal lu Gregory
Gregory        973-747-0370
$ PHONEBOOK=/users/pat/phonebook lu Toritos
El Toritos     973-945-2236
$
```

In the preceding example, we assume that the user `steve` has been granted read access to `pat`'s phone book file.

4. Write a shell script `checkpath` that extracts each directory from the user's `PATH` and ensures that the directory exists and is readable. Report either error condition or confirm that the directory is correctly configured. Hint: directories in the `PATH` are separated by a colon (`:`).

5. Write a script `cmprpaths` that compares the directories specified in `CDPATH` and `PATH`, reporting on any that don't appear in both. For example, if

```
PATH="/bin:/usr/bin/:$HOME/bin"
CDPATH="$HOME/bin:$HOME/projects"
```

Then the program would report:

```
$ cmprpaths
/bin appears only in PATH
/usr/bin/appears only in PATH
$HOME/projects appears only in CDPATH
$
```

Chapter 11 Exercises

1. Rewrite the `home` program from Exercise 5 in Chapter 6 to use `IFS` to extract the home directory from `/etc/passwd`. What happens to the program if one of the fields in the file is null, as in

   ```
   steve:*:203:100::/users/steve:/bin/ksh
   ```

   Here the fifth field is null (`::`).

2. Using the fact that the shell construct `${#var}` gives the number of characters stored in `var`, rewrite `wc` in the shell. Be sure to use integer arithmetic! (Notes: Change your `IFS` variable to just a newline character so that leading whitespace characters on input are preserved, and also use the `-r` option to the shell's `read` command so that terminating backslash characters on the input are ignored.)

3. Write a function called `rightmatch` that takes two arguments as shown:

   ```
   rightmatch value pattern
   ```

   where *`value`* is a sequence of one or more characters, and *`pattern`* is a shell pattern that is to be removed from the right side of *`value`*. The *shortest* matching pattern should be removed from *`value`* and the result written to standard output. Here is some sample output:

   ```
   $ rightmatch test.c .c
   test
   $ rightmatch /usr/spool/uucppublic '/*'
   /usr/spool
   $ rightmatch /usr/spool/uucppublic o
   /usr/spool/uucppublic
   $
   ```

   The last example shows that the `rightmatch` function should simply echo its first argument if it does not end with the specified pattern.

4. Write a function called `leftmatch` that works similarly to the `rightmatch` function developed in Exercise 3. Its two arguments should be as follows:

   ```
   leftmatch pattern value
   ```

   Here are some example uses:

   ```
   $ leftmatch /usr/spool/ /usr/spool/uucppublic
   uucppublic
   $ leftmatch s. s.main.c
   main.c
   $
   ```

5. Write a function called `substring` that uses the `leftmatch` and `rightmatch` functions developed in Exercises 3 and 4 to remove a pattern from the left and right side of a value. It should take three arguments as shown:

   ```
   $ substring /usr/ /usr/spool/uucppublic /uucppublic
   spool
   $ substring s. s.main.c .c
   main
   $ substring s. s.main.c .o        Only left match
   ```

```
main.c
$ substring x. s.main.c .o        No matches
s.main.c
$
```

Chapter 12 Exercises

1. Using `eval`, write a program called `recho` that prints its arguments in reverse order. So

   ```
   recho one two three
   ```

   should produce

   ```
   three two one
   ```

   Assume that more than nine arguments can be passed to the program.

2. Write a script that counts how many times a user tries to quit the program with an interrupt (typically Control-C), showing the count and finally letting them quit on the fifth attempt. Tip: You'll need to use `trap` for this and the interrupt signal is `SIGINT`.

3. Write a script that lets users specify `echo` and `wc` command sequences, allowing them to use the word "`pipe`" to represent a pipe symbol. So input like:

   ```
   echo this is a test pipe wc -w
   ```

   should result in the script displaying 4 since there are four words in the invocation.

4. Modify the `shar` program presented in this chapter to handle directories: If the user specifies a directory instead of a file, it will archive every file within that directory and create the directory as needed when the archive is unpacked.

5. Modify `shar` to include in the archive the character count for each file and commands to compare the count of each extracted file against the count of the original file. If a discrepancy occurs, output an appropriate error message.

Chapter 13 Exercises

1. Add a `-m` command-line option to `rolo` to make it easy to look up someone and have the result emailed to the specified recipient. So

   ```
   rolo -m john Susan
   ```

   will result in `john` getting an email containing the result of a lookup for `Susan` in the phonebook.

2. After adding the `-m` option, add a `-s` option to specify that the mail message is only to be sent if there's one and only one match to the search query.

3. Can you think of other ways to use `rolo`? For example, could it be used as a small general-purpose database program (for example, for storing recipes or employee data)?

4. Modify `rolo` to look for a file `.rolo` in each `rolo` user's home directory that contains the full pathname to that user's phone book file. For example:

   ```
   $ cat $HOME/.rolo
   /users/steve/misc/phonebook
   $
   ```

5. Add the `-u` flag to `rolo` to allow you to look up someone in another user's phone book (provided that you have read access to it). For example:

   ```
   $ rolo -u pat Pizza
   ```

   would look up `Pizza` in `pat`'s phone book, no matter who is running `rolo`. The program can find `pat`'s phone book by looking at `.rolo` in `pat`'s home directory.

Chapter 14 Exercises

1. Write a function that prints all filenames in a specified directory hierarchy without using either `ls` or `find`. Its output should be similar to the output of the `find` command:

```
$ myfind /users/pat
/users/pat
/users/pat/bin
/users/pat/bin/ksh
/users/pat/bin/lf
/users/pat/bin/pic
/users/pat/chapt1
/users/pat/chapt1/intro
/users/pat/rje
/users/pat/rje/filel
```

   (Hint: Bash and Korn shell functions can be recursive.)

2. Write a shell function called `octal` that converts octal numbers given as command-line arguments to decimal numbers and prints them out, one per line:

```
$ octal 10 11 12
8
9
10
$
```

   (Korn shell users, remember that if you assign a decimal number to a variable when it's declared – for example, `typeset -i d=10#0` – assignments to this variable from other bases are automatically converted to decimal.)

3. Modify the `cdh` function to filter out multiple occurrences of the same directory. The new function should work as shown:

```
$ cdh -l
  0 /users/pat
$ cdh
$ cdh
$ cdh -l
  0 /users/pat
$
```

4. Modify the `cdh` function to set the prompt (`PS1`) to show the current directory; for example:

```
/users/pat: cdh /tmp
/tmp: cdh
/users/pat:
```

5. Modify the `cdh` function to allow the user to specify a partial name of a directory in the history file preceded by a dash:

```
/etc: cdh -l
  0 /users/pat
```

```
  1 /tmp
  2 /users/steve
  3 /usr/spool/uucppublic
  4 /usr/local/bin
  5 /etc
/etc: cdh -pub
/usr/spool/uucppublic: cdh -bin
/usr/local/bin:
```