Answers to Even-numbered Exercises 28

2. The special parameter **$@** is referenced twice in the **out** script (page 857). Explain what would be different if the parameter **$*** were used in its place.

   If you replace **$@** with **$*** in the out script, cat or less would be given a single argument: a list of all the files you specified on the command line enclosed within single quotation marks. This list will work fine when you specify a single filename. When you specify more than one file, the shell reports No such file or directory because there is not a file named the string you specified on the command line (the SPACEs are not special characters when they are enclosed within single quotation marks).

4. Write a function that takes a single filename as an argument and adds execute permission to the file for the user.

   ```
   $ function perms {
   > chmod u+x $1
   > }
   ```

   a. When might such a function be useful?

   When you are writing many shell scripts, it can get tedious to give many chmod commands. This function speeds up the process.

   b. Revise the script so that it takes one or more filenames as arguments and adds execute permission for the user for each file argument.

   ```
   $ function perms {
   > chmod u+x $*
   > }
   ```

c. What can you do to make the function available every time you log in?

Put the function in your **.bash_profile**, **.bash_login**, or **.profile** file to make it available each time you run a log in (using bash).

d. What if, in addition to having the function available on subsequent login sessions, you want to make the function available now in your current shell?

Use source to execute the file you put the function in, for example,

```
$ source .bash_profile
```

6. Write a shell script that will display the names of all directory files, but no other types of files, in the working directory.

There are many ways to solve this problem. Following, the **listdirs** script uses file to identify directory files and grep to pull them out of the list. Finally, sed removes everything from file's output,including and following the colon.

```
$ cat listdirs
file "$@"      |
grep directory |
sed 's/:.*//'.
```

8. Enter the following script named **savefiles**, and give yourself execute permission to the file:

```
$ cat $HOME/bin/savefiles
#! /bin/bash
echo "Saving files in current directory in file savethem."
exec > savethem
for i in *
do
echo "==========================================================="
echo "File: $i"
echo "==========================================================="
cat "$i"
done
```

a. What error message do you get when you execute this script? Rewrite the script so that the error does not occur, making sure the output still goes to **savethem**.

You get the following error message:

```
cat: savethem: input file is output file
```

Add the following lines after the line with **do** on it:

```
if [ $i == savethem ]
        then
        continue
fi
```

b. What might be a problem with running this script twice in the same directory? Discuss a solution to this problem.

Each time you run **savefiles**, it overwrites the **savethem** file with the current contents of the working directory. When you remove a file and run **savefiles** again, that file will no longer be in **savethem**. If you want to keep an archive of files in the working directory, you need to save the files to a new file each time you run **savefiles**. When you prefix the filename **savethem** with **$$**, you will have a unique filename each time you run **savefiles**.

10. Using the find utility, perform the following steps:

a. List all files in the working directory that have been modified within the last day.

```
$ find . -mtime -1
```

b. List all files on the system that are bigger than 1 megabyte.

```
$ find / -size +1024k
```

c. Remove all files named **core** from the directory structure rooted at your home directory.

```
$ find ~ -name core -exec rm {} \;
```

d. List the inode numbers of all files in the working directory whose filenames end in **.c**.

```
$ find . -name "*.c" -ls
```

e. List all files on the root filesystem that have been modified in the last month.

```
$ find / -xdev -mtime -30
```

12. Write a script that takes the name of a directory as an argument and searches the file hierarchy rooted at that directory for zero-length files. Write the names of all zero-length files to standard output. If there is no option on the command line, have the script delete the file after displaying its name, asking the user for confirmation, and receiving positive confirmation. A **–f** option on the command line indicates that the script should display the filename but not ask for confirmation before deleting the file.

The following script segment deletes only ordinary files, not directories. As always, you must specify a shell and check arguments.

```
$ cat zerdel
if [ $1 == -f ]
        then
                find $2 -empty -print -exec rm -f {} \;
        else
                find $1 -empty -ok rm -f {} \;
fi
```

14. Generalize the function written in exercise 13 so that the character separating the list items is given as an argument to the function. If this argument is absent, the separator should default to a colon.

    This script segment takes an optional option in the form **–d*x*** to specify the delimiter ***x***.

    ```
    $ cat nodel
    if [[ $1 == -d? ]]
            then
                    del=$(echo $1 | cut -b3)
                    shift
            else
                    del=:
    fi
    IFS=$del
    set $*
    for i
            do
            echo $i
    done
    ```

16. If your Linux system runs X Windows, write a script that turns the root window a different color when the amount of free disk space in any filesystem reaches a certain threshold. (*Hint:* See the df man page.) Both the threshold and the color should be specified as arguments. Check disk usage every 30 minutes. Start the script executing when your X Windows session starts.

    As it is more difficult to change the color of the root window when running GNOME or KDE, for this example, place the following **.xinitrc** file in your home directory and run **startx** to bring up the X Window System. The **dfcolor** script is run to make the root window green if any partition reaches the 80 percent full level.

```
$ cat .xinitrc
xterm&
bash dfcolor 80 green &
metacity
```

```
$ cat dfcolor
#!/bin/bash
threshold=$1
color=$2

while true
    do
    # put argument checking here
    val=$(/bin/df      | # set val to list % free on each partition
    tail +2            | # cut off header
    sed 's/  */        /g' | # convert multiple spaces to single tabs
    cut --fields=5     | # display the percent free field
    sed 's/%//')         # get rid of the percent sign

    # loop through percent full values and see if one is > threshold
    for val2 in $val
        do
            if [ $val2 -ge $threshold ]
                then xsetroot -solid $color
            fi
        done
    sleep 30m
    done
```

18. Rewrite **bundle** (page 883) so that the script it creates takes an optional list of filenames as arguments. If one or more filenames are given on the command line, only those files should be recreated; otherwise, all files in the shell archive should be recreated. For example, suppose that all files with the filename extension **.c** are bundled into an archive named **srcshell**, and you want to unbundle just the files **test1.c** and **test2.c**. The following command will unbundle just these two files:

```
$ bash srcshell test1.c test2.c
```

```
$ cat bundle2
#!/bin/bash
# bundle: group files into distribution package

echo "# To unbundle, bash this file"
for i
do
        echo 'if echo $* | grep -q' $i '|| [ $# = 0 ]'
                echo then
                echo "echo $i 1>&2"
                echo "cat >$i <<'End of $i'"
                cat $i
                echo "End of $i"
        echo fi
done
```

20. What kind of links will the **lnks** script (page 860) not find? Why?

The **lnks** script searches for links by the link count that **ls –l** displays and by matching inode numbers. Both of these characteristics identify hard links. Soft, or symbolic, links (page 180) cannot be identified in this manner.