Flow Control - Part 3

Now that we have learned about positional parameters, it's time to cover the remaining flow control statement, **`for`**. Like **`while`** and **`until`**, **`for`** is used to construct loops. **`for`** works like this:

```
for variable in words; do
    commands
done
```

In essence, **`for`** assigns a word from the list of words to the specified variable, executes the commands, and repeats this over and over until all the words have been used up. Here is an example:

```
#!/bin/bash

for i in word1 word2 word3; do
    echo "$i"
done
```

In this example, the variable `i` is assigned the string "`word1`", then the statement `echo "$i"` is executed, then the variable `i` is assigned the string "`word2`", and the statement `echo "$i"` is executed, and so on, until all the words in the list of words have been assigned.

The interesting thing about **`for`** is the many ways we can construct the list of words. All kinds of expansions can be used. In the next example, we will construct the list of words using command substitution:

```
#!/bin/bash

count=0
for i in $(cat ~/.bash_profile); do
    count=$((count + 1))
    echo "Word $count ($i) contains $(echo -n $i | wc -c) characters"
done
```

Here we take the file `.bash_profile` and count the number of words in the file and the number of characters in each word.

So what's this got to do with positional parameters? Well, one of the features of **`for`** is that it can use the positional parameters as the list of words:

```
#!/bin/bash
```

```
for i in "$@"; do
    echo $i
done
```

The shell variable "$@" contains the list of command line arguments. This technique is often used to process a list of files on the command line. Here is a another example:

```
#!/bin/bash

for filename in "$@"; do
    result=
    if [ -f "$filename" ]; then
        result="$filename is a regular file"
    else
        if [ -d "$filename" ]; then
            result="$filename is a directory"
        fi
    fi
    if [ -w "$filename" ]; then
        result="$result and it is writable"
    else
        result="$result and it is not writable"
    fi
    echo "$result"
done
```

Try this script. Give it a list of files or a wildcard like "*" to see it work.

The use of `in "$@"` is so common that it is assumed if the `in words` clause is ommited.

Here is another example script. This one compares the files in two directories and lists which files in the first directory are missing from the second.

```
#!/bin/bash

# cmp_dir - program to compare two directories

# Check for required arguments
if [ $# -ne 2 ]; then
    echo "usage: $0 directory_1 directory_2" 1>&2
    exit 1
fi

# Make sure both arguments are directories
if [ ! -d "$1" ]; then
    echo "$1 is not a directory!" 1>&2
    exit 1
fi

if [ ! -d "$2" ]; then
    echo "$2 is not a directory!" 1>&2
```

```
        exit 1
    fi

    # Process each file in directory_1, comparing it to directory_2
    missing=0

    for filename in "$1"/*; do
        fn=$(basename "$filename")
        if [ -f "$filename" ]; then
            if [ ! -f "$2/$fn" ]; then
                echo "$fn is missing from $2"
                missing=$((missing + 1))
            fi
        fi
    done
    echo "$missing files missing"
```

Now on to the real work. We are going to improve the `home_space` function to output more information. Recall that our previous version looked like this:

```
home_space()
{
    # Only the superuser can get this information

    if [ "$(id -u)" = "0" ]; then
        echo "<h2>Home directory space by user</h2>"
        echo "<pre>"
        echo "Bytes Directory"
        du -s /home/* | sort -nr
        echo "</pre>"
    fi

}   # end of home_space
```

Here is the new version:

```
home_space() {
    echo "<h2>Home directory space by user</h2>"
    echo "<pre>"
    format="%8s%10s%10s   %-s\n"
    printf "$format" "Dirs" "Files" "Blocks" "Directory"
    printf "$format" "----" "-----" "------" "---------"
    if [ $(id -u) = "0" ]; then
        dir_list="/home/*"
    else
        dir_list=$HOME
    fi
    for home_dir in $dir_list; do
        total_dirs=$(find $home_dir -type d | wc -l)
        total_files=$(find $home_dir -type f | wc -l)
        total_blocks=$(du -s $home_dir)
        printf "$format" "$total_dirs" "$total_files" "$total_blocks"
    done
    echo "</pre>"
}   # end of home_space
```

This improved version introduces a new command **printf**, which is used to produce formatted output according to the contents of a *format string*. **printf** comes from the C programming language and has been implemented in many other programming languages including C++, Perl, awk, java, PHP, and of course, bash. More information about **printf** format strings can be found at:

- GNU Awk User's Guide - Control Letters
- GNU Awk User's Guide - Format Modifiers

We also introduce the **find** command. **find** is used to search for files or directories that meet specific criteria. In the `home_space` function, we use **find** to list the directories and regular files in each home directory. Using the **wc** command, we count the number of files and directories found.

The really interesting thing about `home_space` is how we deal with the problem of superuser access. Notice that we test for the superuser with **id** and, according to the outcome of the test, we assign different strings to the variable `dir_list`, which becomes the list of words for the **for** loop that follows. This way, if an ordinary user runs the script, only his/her home directory will be listed.

Another function that can use a **for** loop is our unfinished `system_info` function. We can build it like this:

```
system_info() {
    # Find any release files in /etc

    if ls /etc/*release 1>/dev/null 2>&1; then
        echo "<h2>System release info</h2>"
        echo "<pre>"
        for i in /etc/*release; do

            # Since we can't be sure of the
            # length of the file, only
            # display the first line.

            head -n 1 "$i"
        done
        uname -orp
        echo "</pre>"
    fi

}   # end of system_info
```

In this function, we first determine if there are any release files to process. The release files contain the name of the vendor and the version of the distribution. They are located in the `/etc` directory. To detect them, we perform an **ls** command and throw away all of its output. We are only interested in the exit status. It will be true if any files are found.

Next, we output the HTML for this section of the page, since we now know that there are release files to process. To process the files, we start a **for** loop to act on each one. Inside the loop, we use the **head** command to return the first line of each file.

Finally, we use the **uname** command with the "o", "r", and "p" options to obtain some additional information from the system.
