

Scripting, Part Two: Looping for Fun and Profit

Crafty System Administrators who want to conserve energy need to learn the fine art of looping.

By [Ken Hess](#)

Thursday, June 30th, 2011

16Share

You energy-conserving* system administrators will enjoy learning to use loops in your scripts. Looping is a technique that allows you to repeat a process or set of commands indefinitely or until the loop exhausts a particular list of items. For example, you want to copy a particular file to everyone's home directory. How do you do it? Don't say that you have a junior-level administrator do it. The correct answer is that you'd create a looping script to handle the job.

Don't worry if you aren't a scripting master, I'm going to take it slow through this series so that you can absorb what's going on. Looping is not a particularly advanced concept. Its purpose is to do some task quickly that would take hours or days to do it by hand. Looping leverages the computer's power to do what it's best at: repetitive processing.

The Basics

You need access to a Linux system and last week's post, "[Scripting, Part One](#)". You'll also need access to an open mind about scripting. It isn't difficult at all, so approach this with no fear and in no time, you'll create your own scripts and possibly impress your leadership with your innovation, business concern, and cost-saving automation.**

The Lively Loop

There's nothing particularly special about a loop. It takes a bit of thought to make one work but it's well worth the effort. How do you know that you need to use a loop instead of simply running a script multiple times with different parameters? The answer depends on the amount of effort required to edit the script, execute the script, enter information interactively, and so on. It's really a decision you'll have to make as you gain experience with scripting. There's no easy answer or number of iterations cutoff.

My original example is a good one. You need to copy a file to everyone's home directory and you have dozens of users. This task, if done manually, would take hours. The solution is to write a script to handle the task for you.

First, look at what's needed to make this happen: a list of users, the file in question, and, depending on the file's purpose, an optional permissions change. Pretty simple task really. To make the reading (and writing) less tedious, I'll use the handful of users on my system to make this happen.

Next, put your needs into Linux terms. You need to iterate through a list of users from the `/etc/passwd` file, copy a file to each user's `/home/username` directory, and then change its permissions so that the user has access to it.

You have to grab a list of real user's names from the password file. To do that, you have to write part of the script. Create a new file and enter that file in edit mode.

```
#!/bin/bash

# Grab a list of users from the /etc/passwd file

cat /etc/passwd |grep bash |grep home | awk -F: '{print $1}'
```

This part of the script reads the */etc/passwd* file, selects only those entries with the term *bash*, further selects those that also contain *home*, and then pipes that output to *awk*. The *awk* piece divides each line into fields separated at a colon (:).

An */etc/passwd* file looks like: khess:x:1000:1000:Ken Hess,,,:/home/khess:/bin/bash. The *awk* command would break this line into seven fields. Using a field variable (\$1, \$2, \$3,...\$7), you could print any of them in any order.

Try it at the command line to convince yourself of how *awk* works.

```
cat /etc/passwd |grep khess | awk -F: '{print $1, $3, $5}'

khess 1000 Ken Hess,,,
```

Try it with other field variables in other files. If the only field separator is a space, then use *awk -F* " " with a quoted space as the field separator.

The output from the original script(*cat /etc/passwd |grep bash |grep home | awk -F: '{print \$1}'*) is:

```
khess
nimbus
bob
matthew
mark
luke
john
```

The most difficult part of creating a list of users is done. The rest of the script is a simple while loop and some additions to make the whole thing a little prettier to read. Enter the following, as shown, into your editor and save the file to *copy_file.sh*. I've included comments in the file.

```
#!/bin/bash

# Grab a list of users from the /etc/passwd file and direct to a file.
cat /etc/passwd |grep bash |grep home | awk -F: '{print $1}' > users.txt

# Begin the while loop. The variable NAME will be read from users.txt
while read NAME

# Setup the file copy in the "do" part of the loop.
do cp /tmp/test.txt /home/$NAME

echo "Copied file to $NAME"
echo " "

# Change the permissions on the copied file to the user's.
chown $NAME:$NAME /home/$NAME/test.txt
```

```
echo "And, changed ownership to $NAME"
echo " "

# Tell the loop when to stop, when there's no more names in the list.
done < names.txt
```

Execute the file and watch the output from the echo commands.

```
root@aspen:~# ./copy_file.sh

Copied file to khess
And, changed ownership to khess

Copied file to nimbus
And, changed ownership to nimbus

Copied file to bob
And, changed ownership to bob

Copied file to matthew
And, changed ownership to matthew

Copied file to mark
And, changed ownership to mark

Copied file to luke
And, changed ownership to luke

Copied file to john
And, changed ownership to john
```

To test that the script ran correctly, check one of the user's home directories.

```
ls -la /home/matthew/
total 24
drwxr-xr-x  2 matthew matthew 4096 2011-06-12 14:54 .
drwxr-xr-x 10 root     root    4096 2011-06-12 14:53 ..
-rw-r--r--  1 matthew matthew  220 2011-06-12 14:20 .bash_logout
-rw-r--r--  1 matthew matthew 3353 2011-06-12 14:20 .bashrc
-rw-r--r--  1 matthew matthew  179 2011-06-12 14:20 examples.desktop
-rw-r--r--  1 matthew matthew  675 2011-06-12 14:20 .profile
-rw-r--r--  1 matthew matthew    0 2011-06-12 14:54 test.txt
```

Now you can put any commands that you want into the loop. Generate a list of whatever you want, act upon each member of the list, and you're done.

Creating a User Space Spy Script

I call this script a user space "spy" because it creates a report in html that tells you how much space, in megabytes (MB), each user is using in his home directory. Here is the entire script and the explanation follows.

```
#!/bin/bash

# Create a list of users and direct it to a file.
cat /etc/passwd |grep bash |grep home | awk -F: '{print $1}' > names.txt

# Create an HTML file. This part must be outside the loop.
echo "

" > user_space.html
Space" >> user_space.html # Start the WHILE loop while read NAME do # This sets output
echo " User                of the command to the variable name, SPACE. SPACE=`du -sm /home/$NAME |awk -F" "
                                '{print $1}'` echo "
                                $SPACE" >> user_space.html done < names.txt # End the table entry outside the loop. echo
                                "

" >> user_space.html
```

Read the documentation inside the file and note that items that only appear once, need to be written outside the loop. Also remember to use double redirects (>>), when writing to a file that exists. If you use the single redirect, (>), you'll overwrite the file with each new line. Run the script and view the resulting HTML file in a web browser. Figure 1 shows the file in Firefox.

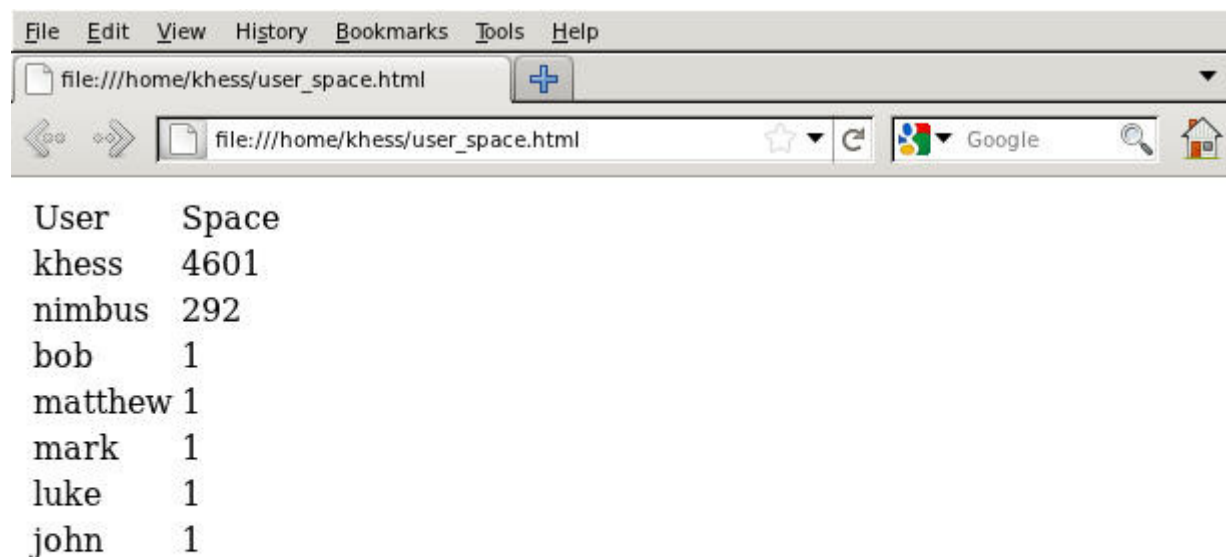


Figure 1: The User Space Spy HTML File in Firefox.

Note that you can set an entire command's output to a variable name. Use tick marks (`) to surround the shell command. Use no spaces between the variable name, the equal sign, and the first tick mark.

For an independent assignment, write this script to make the HTML file fully HTML compliant and dress it up a bit with labels, bold headings, and more.

This is a simple sample of what loops can do for you. You should also investigate *For* loops and *Until* loops. I've never used *For* or *Until* loops because the *While* loop is extremely versatile. The type of loop you use is entirely up to you and your needs. The BASH *For* loop is more versatile than its equivalent in other languages.

Stay tuned next week for conditionals, which are better known as IF-THEN-ELSE statements. These types of programming structures allow your scripts to make decisions and give a bit of intelligence to your scripts. If you don't get caught in any infinite loops, I'll see you there.

* A politically correct way of saying lazy.

** Make it known that you're using your advanced System Administrator skills to automate certain processes as a labor-saving technique. That kind of attitude and business thinking may net you a cash bonus, promotion, or at the very least some positive recognition.