20.2. Redirecting Code Blocks

Blocks of code, such as while, until, and for loops, even if/then test blocks can also incorporate redirection of stdin. Even a function may use this form of redirection (see Example 24-11). The < operator at the end of the code block accomplishes this.

Example 20-5. Redirected *while* loop

```
#!/bin/bash
# redir2.sh

if [ -z "$1" ]
then
  Filename=names.data       # Default, if no filename specified.
else
  Filename=$1
fi
#+ Filename=${1:-names.data}
#  can replace the above test (parameter substitution).

count=0

echo

while [ "$name" != Smith ]  # Why is variable $name in quotes?
do
  read name                 # Reads from $Filename, rather than stdin.
  echo $name
  let "count += 1"
done <"$Filename"           # Redirects stdin to file $Filename.
#    ^^^^^^^^^^^^

echo; echo "$count names read"; echo

exit 0

#  Note that in some older shell scripting languages,
#+ the redirected loop would run as a subshell.
#  Therefore, $count would return 0, the initialized value outside the loop.
#  Bash and ksh avoid starting a subshell *whenever possible*,
#+ so that this script, for example, runs correctly.
#  (Thanks to Heiner Steven for pointing this out.)

#  However . . .
#  Bash *can* sometimes start a subshell in a PIPED "while-read" loop,
#+ as distinct from a REDIRECTED "while" loop.

abc=hi
echo -e "1\n2\n3" | while read l
     do abc="$l"
        echo $abc
     done
echo $abc

#  Thanks, Bruno de Oliveira Schneider, for demonstrating this
#+ with the above snippet of code.
#  And, thanks, Brian Onn, for correcting an annotation error.
```

Example 20-6. Alternate form of redirected *while* loop

```
#!/bin/bash

# This is an alternate form of the preceding script.

#  Suggested by Heiner Steven
#+ as a workaround in those situations when a redirect loop
#+ runs as a subshell, and therefore variables inside the loop
# +do not keep their values upon loop termination.


if [ -z "$1" ]
then
  Filename=names.data     # Default, if no filename specified.
else
  Filename=$1
fi


exec 3<&0                 # Save stdin to file descriptor 3.
exec 0<"$Filename"        # Redirect standard input.

count=0
echo


while [ "$name" != Smith ]
do
  read name               # Reads from redirected stdin ($Filename).
  echo $name
  let "count += 1"
done                      #  Loop reads from file $Filename
                          #+ because of line 20.

#  The original version of this script terminated the "while" loop with
#+      done <"$Filename"
#  Exercise:
#  Why is this unnecessary?


exec 0<&3                 # Restore old stdin.
exec 3<&-                 # Close temporary fd 3.

echo; echo "$count names read"; echo

exit 0
```

Example 20-7. Redirected *until* loop

```
#!/bin/bash
# Same as previous example, but with "until" loop.

if [ -z "$1" ]
then
  Filename=names.data         # Default, if no filename specified.
else
  Filename=$1
fi

# while [ "$name" != Smith ]
until [ "$name" = Smith ]     # Change  !=  to =.
do
  read name                   # Reads from $Filename, rather than stdin.
  echo $name
done <"$Filename"             # Redirects stdin to file $Filename.
#    ^^^^^^^^^^^^

# Same results as with "while" loop in previous example.

exit 0
```

Example 20-8. Redirected *for* loop

```
#!/bin/bash

if [ -z "$1" ]
then
  Filename=names.data          # Default, if no filename specified.
else
  Filename=$1
fi

line_count=`wc $Filename | awk '{ print $1 }'`
#           Number of lines in target file.
#
#  Very contrived and kludgy, nevertheless shows that
#+ it's possible to redirect stdin within a "for" loop...
#+ if you're clever enough.
#
# More concise is     line_count=$(wc -l < "$Filename")


for name in `seq $line_count`  # Recall that "seq" prints sequence of numbers.
# while [ "$name" != Smith ]   --   more complicated than a "while" loop   --
do
  read name                    # Reads from $Filename, rather than stdin.
  echo $name
  if [ "$name" = Smith ]       # Need all this extra baggage here.
  then
    break
  fi
done <"$Filename"              # Redirects stdin to file $Filename.
#    ^^^^^^^^^^^^

exit 0
```

We can modify the previous example to also redirect the output of the loop.

Example 20-9. Redirected *for* loop (both `stdin` and `stdout` redirected)

```
#!/bin/bash

if [ -z "$1" ]
then
  Filename=names.data          # Default, if no filename specified.
else
  Filename=$1
fi

Savefile=$Filename.new         # Filename to save results in.
FinalName=Jonah                # Name to terminate "read" on.

line_count=`wc $Filename | awk '{ print $1 }'`  # Number of lines in target file.


for name in `seq $line_count`
do
  read name
  echo "$name"
  if [ "$name" = "$FinalName" ]
  then
    break
  fi
done < "$Filename" > "$Savefile"     # Redirects stdin to file $Filename,
#    ^^^^^^^^^^^^^^^^^^^^^^^^^^^       and saves it to backup file.

exit 0
```

Example 20-10. Redirected *if/then* test

```
#!/bin/bash

if [ -z "$1" ]
then
  Filename=names.data   # Default, if no filename specified.
else
  Filename=$1
fi

TRUE=1

if [ "$TRUE" ]          # if true    and   if :   also work.
then
 read name
 echo $name
fi <"$Filename"
#  ^^^^^^^^^^^^

# Reads only first line of file.
# An "if/then" test has no way of iterating unless embedded in a loop.

exit 0
```

Example 20-11. Data file *names.data* for above examples

```
Aristotle
Arrhenius
Belisarius
Capablanca
Dickens
Euler
Goethe
Hegel
Jonah
Laplace
Maroczy
Purcell
Schmidt
Schopenhauer
Semmelweiss
Smith
Steinmetz
Tukhashevsky
Turing
Venn
Warshawski
Znosko-Borowski

#  This is a data file for
#+ "redir2.sh", "redir3.sh", "redir4.sh", "redir4a.sh", "redir5.sh".
```

Redirecting the `stdout` of a code block has the effect of saving its output to a file. See Example 3-2.

Here documents are a special case of redirected code blocks. That being the case, it should be possible to feed the output of a *here document* into the `stdin` for a *while loop*.

```
# This example by Albert Siersema
# Used with permission (thanks!).

function doesOutput()
 # Could be an external command too, of course.
 # Here we show you can use a function as well.
{
  ls -al *.jpg | awk '{print $5,$9}'
}
```

```
nr=0          #  We want the while loop to be able to manipulate these and
totalSize=0   #+ to be able to see the changes after the 'while' finished.

while read fileSize fileName ; do
  echo "$fileName is $fileSize bytes"
  let nr++
  totalSize=$((totalSize+fileSize))   # Or: "let totalSize+=fileSize"
done<<EOF
$(doesOutput)
EOF

echo "$nr files totaling $totalSize bytes"
```

Prev Home Next
Using *exec* Up Applications