Input/Output Redirection

One of the principal strengths of shell environments, is the way that several small commands can be linked together to provide greater functionality. Input/Output Redirection is the process by which a user may direct the input or output of a command to a file, or even another command.

Straight Pipes

Straight pipes take the output of one command and pass it to another. The second command, rather than looking for input from the keyboard, takes input from the first command. An example of a simple pipe would be:

```
hello | notify
```

The "|" character is used to specify a pipe. In this example the hello command produces the string "Hello World", which is then sent to the notify command. The notify command pops a dialog containing the "Hello World" string.

A longer series of commands could be piped together:

```
hello | wc | notify
```

In this case the "Hello World" string is sent to wc, a word count command, and the result of wc is sent to notify.

Redirecting To And From Files

The inputs and outputs of commands can be directed to files. For example, the command:

```
hello >hello.txt
```

directs the output of the hello command to a file called hello.txt.

In a similar way, inputs to commands can be directed to files. For example, the command:

```
wc <hello.txt
```

directs the wc command to take its input from the file called hello.txt.

Input and output may be redirected at the same time, the command:

```
wc <hello.txt >wc.txt
```

directs the wc command to take its input from hello.txt and to write its output to wc.txt.

Adding To Existing Files

The '>' redirector creates a new file to hold the command's output. If a file exists, it will be overwritten. The '>>' redirector tells the shell to "add to" the existing file:

```
hello >>hello.txt
```

Standard Error

To prevent errors from being hidden, the nShell splits output into two streams "Standard Output" and "Standard Error". Consider the example:

```
% man >man.txt

Usage: man <command or keyword>.
```

In this case an error message was generated and written to "Standard Error". If we wanted to direct the error message to a file we would use the "&" character:

```
man >&man.out
```

The "&" character can be used with the ">>" redirector also, as in:

```
man >>&man.out
```

Special Devices

Two special output devices are available within the nShell:

dev:null	Discard output
dev:tty	Redirect to the nShell window

The first device, dev:null, is useful when you wish to run a command, but are not interested in its output. Consider the "chattr" command used to modify a file's creator or type:

```
% chattr script.txt -c john

Crea    Type   Name

------ ------ ----

'john' 'TEXT' script.txt
```

By redirecting the result of this command to dev:null, the output can be suppressed:

```
% chattr script.txt -c john >dev:null
```

The second device, dev:tty, is useful when overriding a previous output redirection. Consider the script:

```
echo "hello one"

echo "hello two" >dev:tty

echo "hello three"
```

If no redirection is performed, the output of this script would be:

```
% script

hello one

hello two

hello three
```

If we were to try redirecting the output to a file, the second line would still appear on the console:

```
% script >script.txt

hello two
```

And the file "script.txt" would contain:

```
hello one
hello three
```

Use dev:tty when you want to make sure that a message goes to the console.

Protecting Characters

As you can see above, the shell assumes that ">" and "<" characters are requesting redirection.   This could result in some strange effects.   The command

```
notify <oops!>
```

would mean take input from a file called "oops!>".   (The input filename is terminated by the first space.).

Protecting redirection characters with quotes:

```
notify "<oops!>"
```

would have the desired results.