Input & Output in Command Scripts

1 Redirecting Output from a Script

```
set FC = /tmp/$$CC
cat start > $FC
cat $1 >> $FC
cat end >> $FC
lpr $FC
rm $FC
```

- The redirection `>>` means **append** to the file that follows.
- `$$` is used to create a unique name for this process.
- If the script above is called printenvelope, then

```
% printenvelope ~cappello/text/ch1 >& log &
```

  - the file log contains any output generated during the execution of command printenvelope.
  - Since standard output is being redirected, the output that is not being redirected is diagnostic.
  - The `&` at the end says "run this command in the background."

2 Making Unwanted Output Disappear

- In a script, you redirect **unwanted** output to `/dev/null`.
  - `pushd` and `popd` both output the contents of the directory stack.

```
pushd newdirectory >& /dev/null
...
popd >& /dev/null
```

- A useful idiom is:
  ```
  cp /dev/null emptyfile
  ```
  which creates an empty file called `emptyfile`, if it does not exist, and empties it, if it does exist.

3 Redirecting Input from the Terminal

- To request input from the terminal, use the `$<` operator.

  In the script below, the `break` gets out of the `while`, not just the `switch`.

  The `-n` option of the echo below means **suppress** the newline.

```
while (1)
   echo -n "Do you want to proceed? (y/n):"
   switch ("$<")
   case "n":
      set replace = "n"
      break
   case "y":
      set replace = "y"
      break
   endsw
end
if ($replace == "y") rm filex
```