Bourne Shell Programming

Andrew Arensburger

A Simple Script

```
#!/bin/sh
# Rotate procmail log files
cd /homes/arensb/Mail
rm procmail.log.6       # This is redundant
mv procmail.log.5 procmail.log.6
mv procmail.log.4 procmail.log.5
mv procmail.log.3 procmail.log.4
mv procmail.log.2 procmail.log.3
mv procmail.log.1 procmail.log.2
mv procmail.log.0 procmail.log.1
mv procmail.log procmail.log.0
```

Variables

- Set with VAR=*value*
- Refer to value with $VAR or ${VAR}
- Export a variable with `export` VAR
- Unset a variable with `unset` VAR

```
#!/bin/sh
COLOR=yellow

echo The color is $COLOR
echo This seems ${COLOR}ish
```

Local and Environment Variables

- Local variables are local.
- Environment variables are passed on to subprocesses.
- Turn a local variable into an environment variable with

    `export` VAR

```
#!/bin/sh
NETSCAPE_HOME=/usr/imports/libdata

CLASSPATH=$NETSCAPE_HOME/classes
export CLASSPATH

$NETSCAPE_HOME/bin/netscape.bin
```

Special variables

`$1, $2...$9`	Command-line arguments.
`$0`	Script name.
`$*, $@`	All command-line arguments.
`$#`	Number of command-line arguments.
`$?`	Exit status of last command.
`$-`	`sh` options.
`$$`	PID of the current process.
`$!`	PID of the last background command.
`$IFS`	Input Field Separator.

Variable constructs

${VAR:-*expression*}	Use default value.
${VAR:=*expression*}	Set default value.
${VAR:?[*expression*]}	Die if VAR is not set.
${VAR:+*expression*}	*expression* if VAR is set.

Patterns and Globbing

*	Matches zero or more characters.
?	Matches exactly one character.
[*range*]	Matches any character in *range*.

Quoting Strings

- Backslash escapes the next character.
- Single quotes just quote.
- Double quotes expand variables.
- Backquotes expand to the output of a command.

```
echo \$foo == $foo
echo '* MAKE   $$$   FAST *'
echo "The file is  $filename"
echo `whoami`

echo "The time is `date`"
```

Built-in Commands

{ *commands* `;` },(*commands*)	Executes *commands* in a subshell.
`:` (colon)	Does nothing.
`.` *filename*	Reads in the specified filename.
`bg` [*job*], `fg` [*job*]	Background, foreground.
`cd` [*dir*]	Sets the current directory.
`pwd`	Prints the current directory.
`echo` [*args*]	Prints *args* to standard output.
`eval` *args*	Evaluates *args* as a `sh` expression.

Built-in Commands (cont.)

`exec` *command*	Replaces the shell with *command*.
`exit` [*n*]	Exits the current shell with exit code *n*.
`kill` [-*sig*] %*job*	Sends signal *sig* to the specified job.
`read` *name*...	Reads one line from standard input.
`set` [+/-*flag*] [*arg*]	Sets `sh` options or command-line arguments.
`test` *expression*	Evaluates a boolean expression.
`trap` [*command sig*]...	Traps one or more signals.

Built-in Commands (cont.)

`ulimit` Prints or sets system limits on resource usage.

`umask` [*nnn*] Sets the umask.

`wait` [*n*] Waits for a background process to exit.

if

`if` *condition* `; then`
  commands
[`elif` *condition* `; then`
  commands]...
[`else`
  commands]
`fi`

```
#!/bin/sh
myname=`whoami`

if [ $myname = root ]; then
        echo "Welcome to FooSoft 3.0"
else
        echo "You must be root to run this script"
        exit 1
fi
```

while

```
while condition; do
      commands
done
```

`for`

```
for var in list; do
    commands
done
```

```
#!/bin/sh
for i in foo bar baz "do be do"; do
        echo $i
done
```

`case`

<pre>
case expression in
     pattern)
               commands
               ;;
     ...
esac
</pre>

```
case $filename in
        *.jpg) echo "It's a JPEG file"
               ;;
        *.gif) echo "It's a GIF file"
               ;;
        *)     echo "Unknown file type"
               ;;
esac
```

I/O Redirection

< *filename*	Read stdin from *filename*.
> *filename*	Write stdout to *filename*.
>> *filename*	Appends stdout to *filename*.
<<*word*	Reads stdin up to *word*.
<&*digit*	Use file descriptor *digit* as standard input.
>&*digit*	Use file descriptor *digit* as standard output.
<&-	Close standard input.

I/O Redirection (cont.)

`>&-` Close standard output.

command1 `|` *command2* Connect *command1*'s stdout to *command2*'s stdin.

command1 `&&` *command2* Execute *command2* iff *command2* succeeds.

command1 `||` *command2* Execute *command2* iff *command1* fails.

I/O Redirection Examples

```
echo "Danger! Danger, Will Robinson!" 1>&2

line=13
cat <<EOT
An error occurred on line $line.
See page 98 of the manual for details.
EOT
```

Functions

name () {
        commands
}

```
fancyls() {
    echo "***** Files in $1:"
    ls "$1"
    echo "***** There are ‘ls -1 ${dir} | wc -l‘ files in $1"
}

fancyls /tmp
```

Useful utilities

- basename
- dirname
- [
- echo
- awk

  `awk -F : '{print $1, $3 }' /etc/passwd`
- sed

  `sed -e 's/foo/bar/g'`
- tee

test

`-e` *filename* True if *filename* exists.

`-d` *filename* True if *filename* exists and is a directory.

`-f` *filename* True if *filename* exists and is a plain file.

`-h` *filename* True if *filename* exists and is a symbolic link.

`-r` *filename* True if *filename* exists and is readable.

`-w` *filename* True if *filename* exists and is writable.

`-n` *string* True if the length of *string* is non-zero.

`-z` *string* True if the length of *string* is zero.

`test` (cont.)

string	True if *string* is not the empty string.
s1 `=` *s2*	True if the strings *s1* and *s2* are identical.
s1 `!=` *s2*	True if the strings *s1* and *s2* are not identical.
n1 `-eq` *n2*	True if the numbers *n1* and *n2* are equal.
n1 `-ne` *n2*	True if the numbers *n1* and *n2* are not equal.
n1 `-gt` *n2*	True if the number *n1* is greater than *n2*.
n1 `-ge` *n2*	True if the number *n1* is greater than or equal to *n2*.
n1 `-lt` *n2*	True if the number *n1* is less than *n2*.

`test` (cont.)

n1 `-le` *n2*	True if the number *n1* is less than or equal to *n2*.
`!` *expression*	Negates *expression*.
expr1 `-a` *expr2*	True if both expressions, *expr1* and *expr2* are true.
expr1 `-o` *expr2*	True if either expression, *expr1* or *expr2* is true.
`(` *expression* `)`	True if *expression* is true. This allows expressions to nest.

Debugging

- Trace statements
- `sh -n`
- `sh -x`

And that's about it, unfortunately.

Style

- Prefer simple, linear application.
- Put customization variables at the top.
- Use functions sparingly.
- Multi-line strings.
- Use : ${*VAR*:=*value*} to set defaults.

Paranoia

- No setuid scripts, EVER!
- `IFS=`
- Set `$PATH`
- Quote all variables.
- Watch out for empty or bogus values.

```
if [ x"$answer" = xyes ]; then
```