

A Shell-Hackers Guide to the Unixverse or Enough to get by on in Bourne shell scripts

N.J.Gunton 11/98 ; 07/07

Grouping	Individual
Prerequisites	A unix account
Courses	Computer Networks, Sys. Admin, any, none
Requirements	Dogged persistence and rugged determination
Summary	An introduction to the key features of shell scripts
Objectives	To enable a student to tackle the writing of simple scripts and to have an understanding of more complex ones.

1. Background

Unix shells, of which there are many, provide powerful scripting facilities. They are akin to programming languages with many of the same features. The `c` shell or `csh` is named for its similarities with the C programming language. The shell features are usually well documented, if somewhat cryptically, in the man pages. A number of books are available dedicated to this arcane craft and several can be found in the library. Two good general books are Steve Bourne's† Unix book and Brian Kernigan et.al.s' book 'The Unix Programming Environment'

This worksheet is only intended to give an introduction to shell scripts written in the original shell `sh`. The reasoning behind this is that this shell can be guaranteed (pretty much) to be found on any Unix system. This is true even if it is not the default login shell. The other shells, `csh` `tcsh` `bash` `ash` `zsh` `ksh`, to name a few, may not be on the system even though they could be more powerful.

Why learn about an old scripting language such as this?

A Unix system is full of shell-scripts controlling any number of tasks from booting the system to controlling the network. There may be nice GUI front ends to these scripts but on occasion you may have to edit the scripts in the raw. When you do, and if you are ever involved in Unix systems/networks administration then you will, then it won't be a complete shock.

- For a taster have a look at the files in `/etc/rc.d/` . They are the scripts that are run at boot time and at system shutdown. Unix versions derived from System V such as Solaris keep the scripts in `/etc/rc.d/init.d/`.

As can be seen from the examples in the above directories, shell scripts are a mixture of unix commands and programming constructs. They contain variables, environment variables, shell directives and comments.

Shell scripts can also be a simple sequence of commands, executed consecutively as the following example shows. It was a script used to standardize the linux machines in 3P11. A number of changes needed to be made to all the machines. It took about half an hour to develop the script and about three minutes to run it on each machine. The script was placed on a floppy along with the required files and the new kernel. It would have otherwise taken about 20 -30 minutes per machine to make these changes. The script is executed from the command line and expects one parameter, the ip address of the current machine.

† He of the bourne shell.

- It uses the `csch` to execute as shown in the first line.
- A simple test is then made to see if the ip address was on the command line. If not then the script exits at this point with a usage message.
- Alter environment variables to allow overwriting of files without warning!
- Move the old kernel, copy in the new kernel, alter the boot configuration file and run `lilo` to update the boot manager.
- install the new hosts file so that the machines can recognize each other.
- Install the ethernet configuration file, needed at boot time and use `sed` to edit it to reflect this machines ip address. The address is held in the variable `$1`.
- ensure that the network is brought up at the correct time during booting.
- Tidy up a few other details and then we're done.

```
#!/bin/csh
# install kernel and set up networking, add hosts
# copyright Craig and Nigel 6/11/98

if ( "$#argv" != 1 ) then
echo "Usage $argv[0] : IPno"
exit
endif

unset noclobber
set verbose

cp -f /boot/vmlinuz-2.0.34-0.6 /boot/old_kernel
cp -f /mnt/flop/vmlinuz /boot/vmlinuz
rm -f /boot/test
cp -f /etc/lilo.conf /etc/lilo.old
cp -f /mnt/flop/lilo.conf /etc/lilo.conf
chmod 600 /etc/lilo.*
/sbin/lilo
cp -f /etc/hosts /etc/hosts.old
cp -f /mnt/flop/hosts /etc/hosts
cat /mnt/flop/ifcfg-eth0 | sed s/103/$1/ > /etc/sysconfig/network-scripts/ifcfg-eth0
/sbin/chkconfig --level 2345 network on
echo "# name changed to notgpm to stop gpm screwing up x by Craig & Nigel \
    6/11/98" >/etc/rc.d/init.d/notgpm
cat /etc/rc.d/init.d/gpm >> /etc/rc.d/init.d/notgpm
rm -f /etc/rc.d/init.d/gpm
cp -f /mnt/flop/rc.local /etc/rc.d/rc.local
```

2. In at the deep end

Table 1 : key sh features		
<i>character</i>	<i>explanation</i>	<i>example</i>
#	comment marker	# get input from keyboard
` `	execute command and substitute result	`ls`
' '	hard quotes, will not expand variables	'this variable \$HOME prints as is '
" "	soft quotes, will print variable contents	"your home directory is \$HOME"
#	set to the number of command line parameters	if [\$#=2]

The following simple script demonstrates many of these features.

```
#!/bin/sh
#
# variable assignment in a shell script
#
TODAY=`date|cut -c1-3`
echo $TODAY
if [ "$TODAY" = "Wed" ]; then
    echo 'Its not a Crunchy day'
fi
if [ $TODAY = "Fri" ]; then
    echo 'Yippee it really is poets day'
else echo 'doing anything today ?'
fi
```

In shell scripts variables can be declared as needed and are effectively typeless. This is shown in the script with the variable `TODAY`. It is assigned the value returned by executing the command sequence ``date|cut -c1-3``. This executes the `date` command and pipes its output to the `cut` utility. This slices out the first three characters (`-c1-3`) from the stream and returns these to the command line. These three characters are then assigned to `TODAY`.

- Read the man page for `cut`. Execute the command string `date|cut -c1-3` on the command line. Play with the parameters to `cut` and try to slice out different parts of the date.

To print the contents of a variable to the screen or to a file you need to put the dollar sign `$` in front of the variable name. Traditionally all variables are given uppercase names to make them easier to spot.

☞ *an aside: not all the variables in the following scripts use uppercase names. Is this a good idea? Discuss.*

A simple boolean test of the value of `TODAY` is then made. **Note that the spaces inside of [and] are crucial.** The syntax of the `if` statement is shown, as is the `if-else`. Both are terminated with `fi`, (if backwards). The command `echo` simply prints the string or variable value to the standard output.

- Fire up emacs and enter this script. Make it executable by altering the file permissions.

```
chmod 700 scriptname
```

will make the file readable, writeable and executable for the owner. RTFM if you want more info on the `chmod` command. Once you have done this you can test the script. If it is in your current directory you can execute it as

```
./myscriptname
```

The next script makes use of the shells ability to automatically assign parameters to variables. There is a limit to the number of variables that can be set in this way. See the man pages for details. Note that if you have a script that takes command line parameters then these may be overwritten if you then use the `set` command.

```
#!/bin/sh
#
# example of automatically setting the positional parameters
#
set 'date'
echo "It is $4 on $1, $2, $3, $5 "
if [ "$1" = "Wed" ]; then
    echo 'Its not a crunchy day'
fi
if [ $1 = "Fri" ]; then
    echo 'Yippee it really is poets day'
else echo 'doing anything today?'
fi
```

- Again, enter this script, make it executable, play with the parameters etc.

The following scripts are all similar and steadily increase in complexity. They show the use of default settings if there are no positional parameters.

```
dir=${1:-.}
```

This sets the value of `dir` to the first positional parameter if there is one, otherwise it sets it to `.` the current directory as a default. The scripts also show the reading of strings from the keyboard. ie interactive scripts. They also show the use of loops.

script 1

```
#!/bin/sh
#
# A more complex example that uses the ls command to list
# all the files in the current directory. Each file is tested
# to see if it is an ordinary file. For each ordinary file the
# user is then prompted as to whether to back up the file.

for fn in `ls`
do
    if [ -f $fn ] ; then
        echo " backup $fn " ;
        filebak="$fn";
        read response
        fi
        if [ "$response" = "y" ]; then
            mv $filebak $filebak.bak;
            echo "moving $filebak to $filebak.bak"
        fi
    done
```

script 2

```
#!/bin/sh
#
# The command accepts an optional pathname for the backup
# directory. If no pathname is given then the backup directory
# defaults to the current directory.
#
#
dir=${1:-.}
echo "back up directory is $dir"
for fn in `ls`
do
    if [ -f $fn ] ; then
        echo " backup $fn " ;
        filebak="$fn";
        read response
        fi
        if [ "$response" = "y" ]; then
            mv $filebak $dir/$filebak.bak;
            echo "moving $filebak to $dir/$filebak.bak"
        fi
    done
```

script 3

```
#!/bin/sh
#
# The command accepts an optional pathname for the backup
# directory. If no pathname is given then the backup directory
# defaults to the current directory. If a directory name is
# provided then a test is performed to see if it is a
# directory
dir=${1:-.}
if [ -d $dir ] ; then
    echo "back up directory is $dir"
else
    echo "$dir does not exist or is not a directory";
    exit 0
fi
for fn in `ls`
do
    if [ -f $fn ] ; then
        echo " backup $fn " ;
        filebak="$fn";
        read response
        fi
        if [ "$response" = "y" ]; then
            mv $filebak $dir/$filebak.bak;
            echo "moving $filebak to $dir/$filebak.bak"
        fi
    fi
done
```

script 4

```
#!/bin/sh
#
# user is prompted as to whether to remove the file.
# The command accepts an optional pathname for the remove
# directory. If no pathname is given then the remove directory
# defaults to the current directory.
dir=${1:-.}
if [ -d $dir ] ; then
    echo "chosen directory is $dir"
else
    echo "$dir does not exist or is not a directory";
    exit 0
fi
for fn in `ls $dir*`
do
    if [ -f $fn ] ; then
        echo " remove $fn " ;
        filebak="$fn";
        read response
        fi
        if [ "$response" = "y" ]; then
            rm "$filebak";
            echo "removing $filebak"
        echo
        fi
    fi
done
```

Finally a couple of other scripts that I have knocked together at various times to solve problems. One of them makes great use of the stream editor sed. This edits a text stream on the fly!. cut, join and sort are also old favourites for shell scripts as are find and grep for which there is a separate worksheet.

```
#!/bin/sh
#
# rough draft of file to filter practical lists out of classlist.text
#
# usage is extract modulenummer [PT] pathname of classlist
if [ $# -ne 3 ]
then
    echo "Usage: extract modulenummer group filetosearch";
    echo "eg. extract UQC606S2 P/2 ./classlist.txt";
else
    GROUP=`echo "$2" |sed 's///g'`
    `echo "$1 $2 " > $1.$GROUP.list`
    `grep $1 $3 > /tmp/list1.$$`
    `cut -b 10-80 /tmp/list1.$$ | grep $2 > /tmp/list2.$$`
    `cut -f 3 /tmp/list2.$$ | sed 'y/ABCDEFGHIJKLMNOPQRSTUVWXYZ/abcdefghi\
        jklmnopqrstuvwxyz/'|sed 's/[+][123]/ /g'>/tmp/list3.$$`
    `cat < /tmp/list3.$$ >> $1.$GROUP.list`
    rm -f /tmp/list*.$$
fi
```

```

#!/bin/sh
#
# Grind will grind its way through all the files in a directory looking for
# files that contain a given regular expression. It will print the
# file name of any files containing the regular expression along with a
# count of the number of occurrences within a file.
#
case $# in
  1)
    for fn in `ls`
    do
      if [ -f $fn ] ; then
        grep -Gc1 $1 $fn
      fi
    done ;;
  2)
    now=`pwd`;
    dir=${1}
    if [ -d $dir ] ; then
      echo "searching directory $dir"
    else
      echo "$dir does not exist or is not a directory";
      exit 0
    fi
    cd $dir;
    for fn in `ls`
    do
      if [ -f $fn ] ; then
        grep -Gc1 $2 $fn
      fi
    done
    cd $now ;;
  *)
    echo "Usage : grind [expression] | [directory expression] " ;
    exit 0 ;;
esac
exit 0;

```

Note that the construct `for i in `ls`` is redundant in the scripts above and could be replaced with `for i in *`