shell scripting

COMS10012 Software Tools

scripting

shell scripting

Put shell commands in a file `script`, one per line – then you can call

```
$ sh script
```

Or, add the line `#!/bin/sh` at the start and set the x bit, then you can just do

```
$ ./script
```

.profile and .bashrc

Commands in **`~/.profile`** get run when you log in and start the first shell.

Commands in **`~/.bashrc`** get run whenever you start a bash shell.

System-wide versions of these files are **`/etc/profile`** and **`/etc/bash.bashrc`** .

PATH

If you want to make your own shell commands:

- Make a folder **`bin`** in your home directory.
- Add this line to your **`.profile`**:
  `export PATH="$PATH:~/bin"`

You can now put a file `script` with +x in this folder and run it like any other command (this works for compiled programs too).

return values

return values

Programs can return a 1-byte value (this is why `main()` in C returns int). In a script, `exit N` is the equivalent of a "return".

Convention: return 0 = success, >0 = error.

In the shell, `$?` holds the output of the last command.

return values

```
$ ls
...

$ echo $?
0

$ ls badfile
ls: badfile: no such file or directory

$ echo $?
1
```

Boolean operators

```
$ CMD1 && CMD2
```

Run CMD1 and, if successful (returns 0), also run CMD2. $? is the return value of CMD2.

```
$ CMD1 || CMD2
```

Run CMD1 and, if not successful (returns > 0), also run CMD2. $? is the return value of CMD2.

example

```
$ gcc -Wall program.c -o program && ./program
```

[

```
$ [ $? -eq 2 ] && echo "It returned 2"
$ [ -e FILE ] && echo "File exists"
$ [ -d "$D" ] && echo "$D is a directory"
$ [ -n "$D" ] && echo "String is not empty"
```

[CONDITION(S)]

Returns 0 if (and only if) the conditions are true.

You can combine with –a/-o (and/or), ! (not).

[

Warning: variables can be unset.

If $D is unset:

```
$ [ -e $D ]; echo $?
0

$ [-e "$D" ]; echo $?
1
```

executes: `[ -e ]`

executes: `[ -e "" ]`

scripting techniques

variables

```
$ VAR=VALUE
```

Sets a variable for the current shell.

```
$ export VAR=VALUE
```

Sets a variable for the current shell and any programs launched from it.

arguments

`"$1", "$2" ...`	arguments passed to script always double-quote these!
`$#`	the number of arguments
`"$@"`	all arguments (for example, to pass to another script)

if

```
if COMMAND                  # e.g.: if [ CONDITION ]
then
  COMMANDS
fi
```

one-line version:

```
$ [ CONDITION ] && (COMMAND; COMMAND...)
```

basename

```
$ basename /bin/bash
bash
$ basename image001.jpg .jpg
image001
```

case

```
case $(basename "$SHELL") in
    bash) echo "You are using bash" ;;
    sh) echo "You are using a Bourne shell" ;;
    csh) echo "You are using a C shell"
         echo "Good luck :)"
         ;;
    *) echo "You are using something else" ;;
esac
```

aside …

From the Bourne shell source code in C

(never, *EVER* do this!)

```
#define IF   if(
#define THEN ){
#define ELSE } else {
#define ELIF } else if (
#define FI   ;}
```

aside …

```
IF argc>1 ANDF *argp[1]=='-'
THEN    cp=argp[1];
        flags &= ~(execpr|readpr);
        WHILE *++cp
        DO      flagc=flagchar;

                WHILE *flagc ANDF *flagc != *cp DO flagc++ OD
                IF *cp == *flagc
                THEN    flags |= flagval[flagc-flagchar];
                ELIF *cp=='c' ANDF argc>2 ANDF comdiv==0
                THEN    comdiv=argp[2];
                        argp[1]=argp[0]; argp++; argc--;
                ELSE    failed(argv[1],badopt);
                FI
        OD
        argp[1]=argp[0]; argc--;
FI
```

source: https://www.tuhs.org/cgi-bin/utree.pl?file=V7/usr/src/cmd/sh/args.c

loops

for

```
$ for num in 1 2 3
> do
> echo $num
> done
1
2
3
$ ↑
$ for num in 1 2 3; do echo $num; done
```

for

```
for f in *.jpg
do
    n=$(basename $f .jpg)
    convert -scale 50% $f ${n}_small.jpg
done
```

seq

```
$ seq 3
1
2
3
```

```
$ seq 1 2 4
1
3
```

```
$ seq -s, 5
1,2,3,4,5
```

seq

```
$ for i in $(seq 3); do echo "Line $i"; done
Line 1
Line 2
Line 3

$ seq -s, -w 10
01,02,03,04,05,06,07,08,09,10
```